

# Introduction to R Programming

## Data Visualitazion with ggplot2

Pedro Fonseca

19 Abril 2020

# ggplot2:

# Build a data MASTERPIECE



# Introduction

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

# Preliminars

```
library(ggplot2)
```

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

# The mpg dataset

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

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

```
View(mpg)
```

For additional information see `?mpg`.

## My first ggplot()

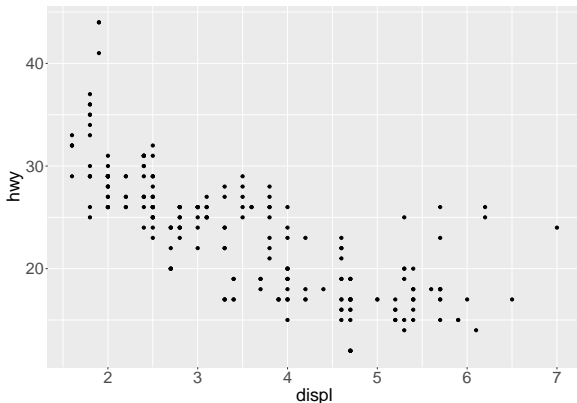
Let's build a graph to answer the following questions:

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

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

## My first ggplot()

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



## My first ggplot()

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



## A simple ggplot() template

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

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

## A simple `ggplot()` template

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

# Geometrical objects

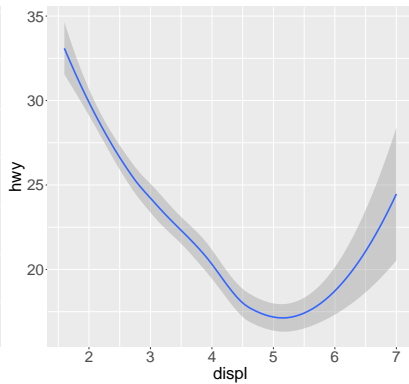
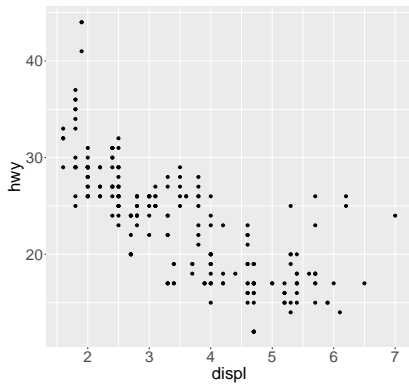
- ▶ A **geom** is the geometrical object that a plot uses to represent data (e.g. points, bars, lines. . . ).
- ▶ People often describe plots by the type of **geom** that it uses:
  - ▶ Bar charts use **bar** geoms
  - ▶ Line charts use **line** geoms
  - ▶ Boxplots use **boxplot** geom
  - ▶ ...
- ▶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

## Geometrical objects

Compare the next two plots. How are they similar?

```
left <- ggplot(data = mpg) +  
  geom_point(mapping = aes(x = displ, y = hwy))  
  
right <- ggplot(data = mpg) +  
  geom_smooth(mapping = aes(x = displ, y = hwy))  
  
ggarrange(left, right, nrow = 1)
```

# Geometrical objects



# Geometrical objects

Both plots:

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

But they are not identical:

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

# The ggplot2 cheatsheet

- ▶ ggplot2 provides over 40 geoms
- ▶ The best way to get a comprehensive overview is the [ggplot2 cheatsheet](#).

# Aesthetics

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



## My first ggplot() revisited

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

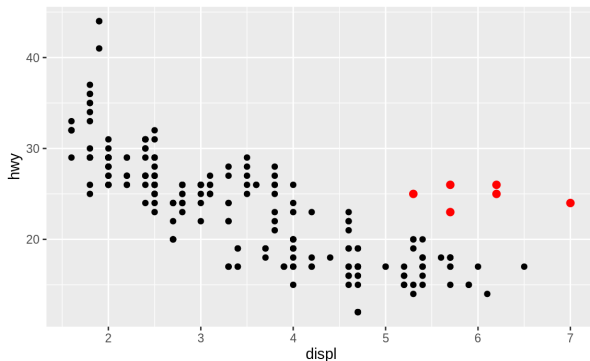


Figure 1: Some cars have a higher mileage than we might expect

## My first ggplot() revisited

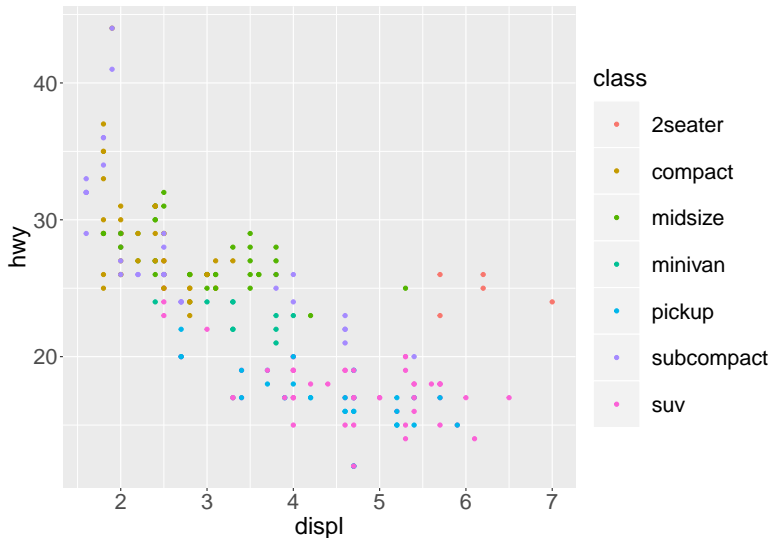
- ▶ The highlighted cars may have some common characteristic with respect to some other variable (e.g. they might all be hybrids).
- ▶ Lets try `class`, a variable that classifies cars into groups such as compact, midsize, and SUV.
- ▶ We can add a third variable to a two dimensional scatterplot by mapping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

# The color aesthetic

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

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

## The color aesthetic



## The color aesthetic

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

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

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

# The color aesthetic

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



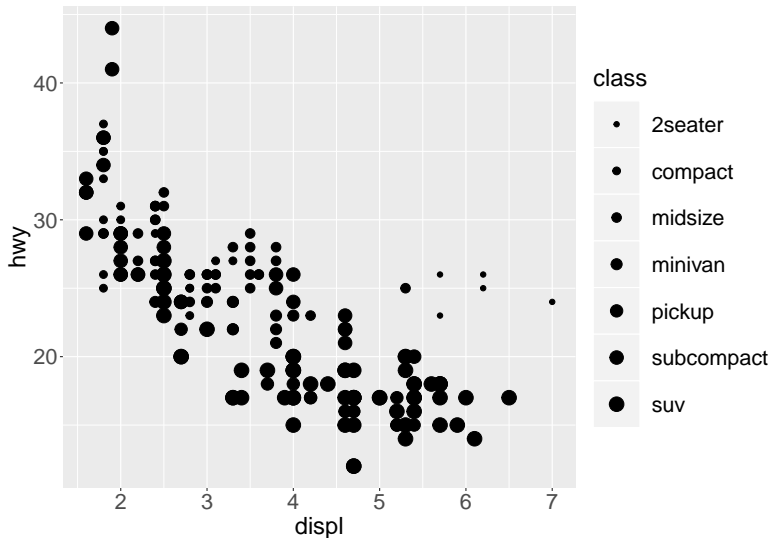
Figure 2: 2008 Chevrolet Corvette

## The size aesthetic

Now let's map class to the size aesthetic:

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

## The size aesthetic





## The size aesthetic

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

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

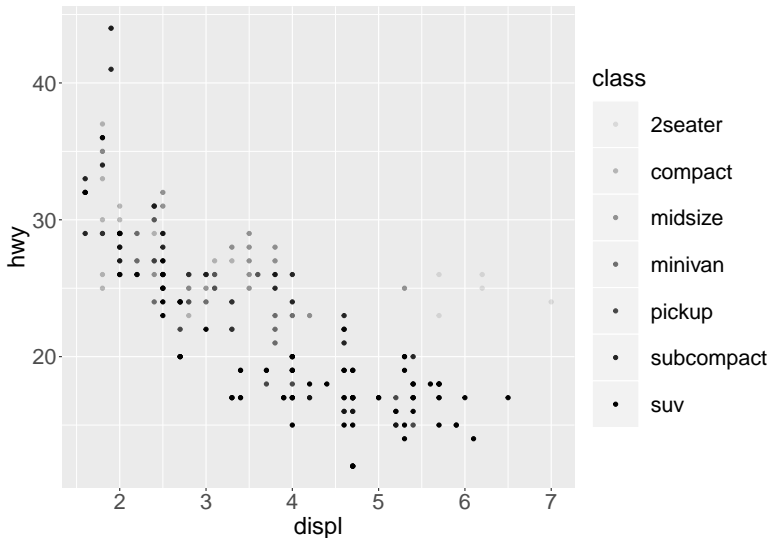
- ▶ This is because mapping an unordered variable (class) to ordered aesthetic (size) is not a good idea.
- ▶ Ordered aesthetics, like size and alpha, are more appropriate for continuous variables.

# The alpha aesthetic

The alpha aesthetic changes the transparency of the points:

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

## The alpha aesthetic

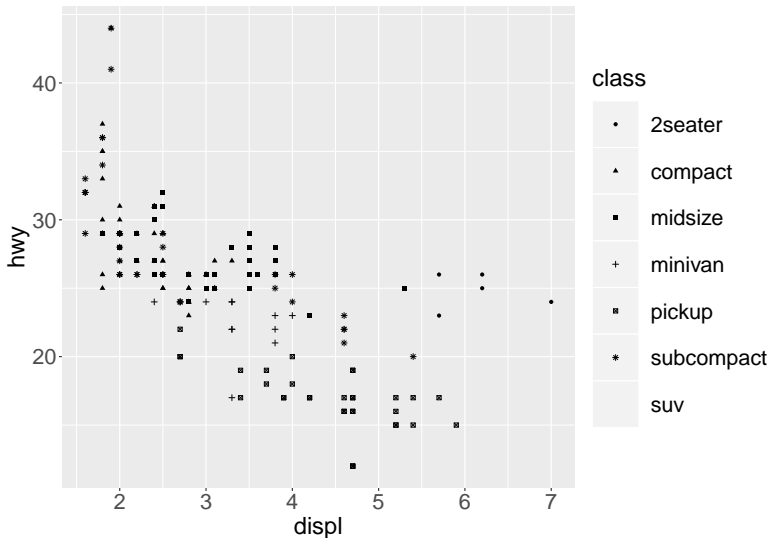


# The shape aesthetic

The shape aesthetic changes the shape of the points:

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

# The shape aesthetic



# The shape aesthetic

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

# The shape aesthetic

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


























|   |   |   |   |   |    |   |    |   |    |
|---|---|---|---|---|----|---|----|---|----|
|  | 0 |  | 4 |  | 10 |  | 15 |  | 22 |
|  | 1 |  | 6 |  | 11 |  | 16 |  | 21 |
|  | 2 |  | 7 |  | 12 |  | 17 |  | 24 |
|  | 5 |  | 8 |  | 13 |  | 18 |  | 23 |
|  | 3 |  | 9 |  | 14 |  | 19 |  | 20 |

Figure 3: R Built in shapes

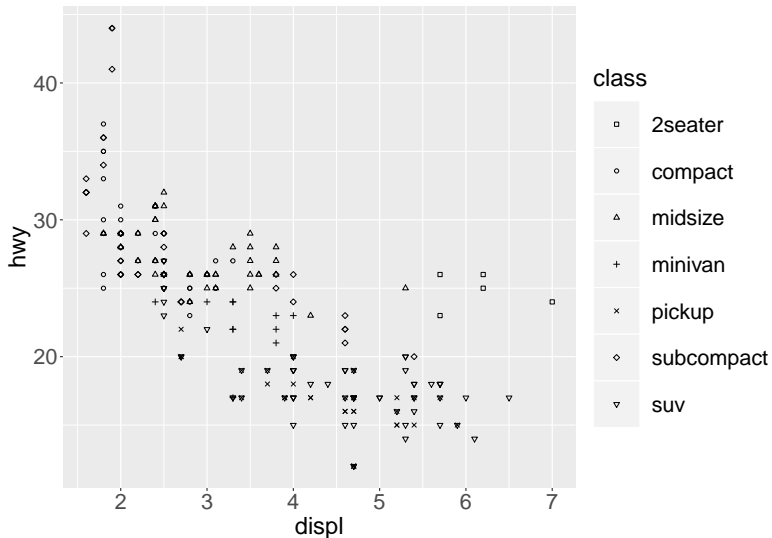
# The shape aesthetic

Let's choose the first 7 shapes:

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



## The shape aesthetic

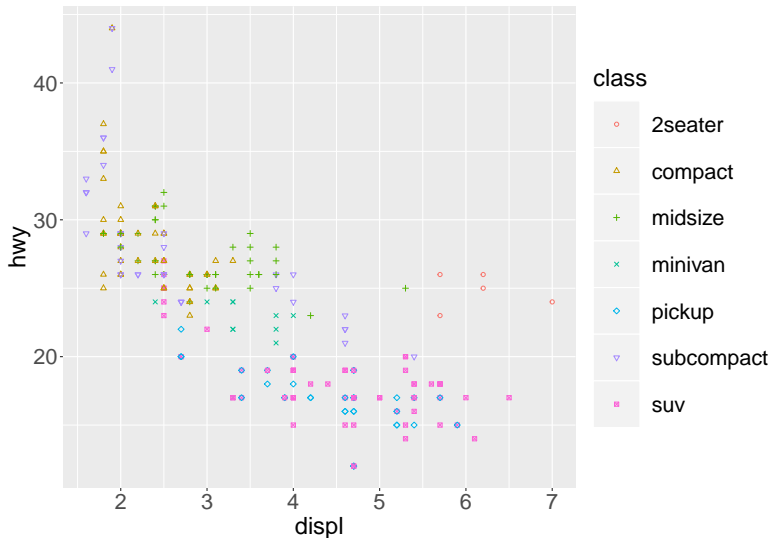


## Mapping one variable to two aesthetics

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

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

## Mapping one variable to two aesthetics



# Mapping continuous variables to aesthetics

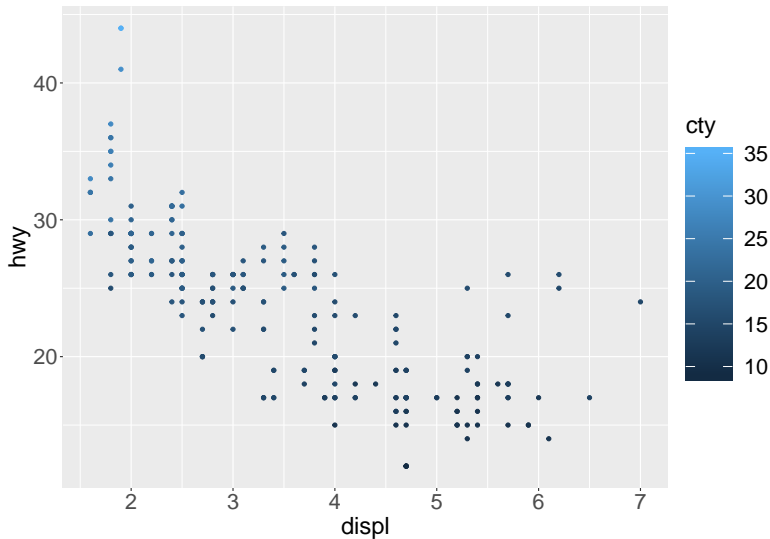
- ▶ So far we mapped `class`, a categorical variable, to the `color`, `size`, `shape` and `alpha` aesthetics of `geom_point()`.
- ▶ Now let's map continuous variables to these aesthetics and see how they behave differently for categorical versus continuous variables.

## Mapping continuous variables to aesthetics

The variable `cty` (city miles per gallon) is a continuous variable. Let's map it to the color aesthetic:

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

## Mapping continuous variables to aesthetics

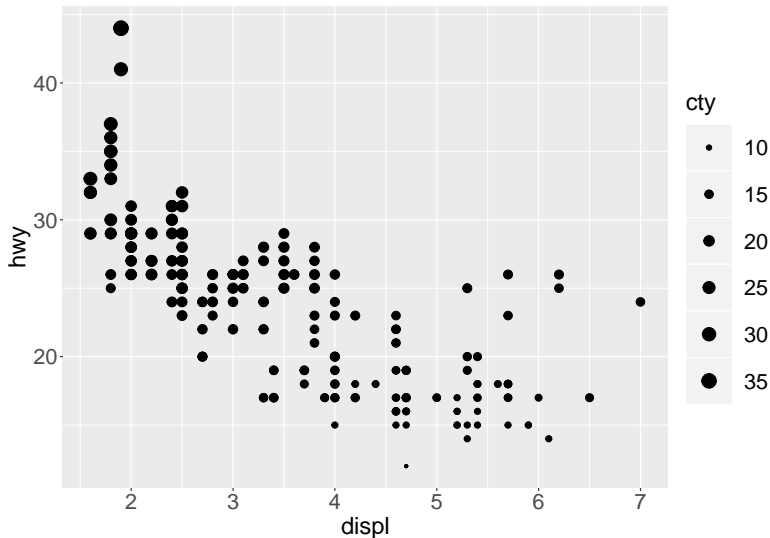


# Mapping continuous variables to aesthetics

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

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

## Mapping continuous variables to aesthetics



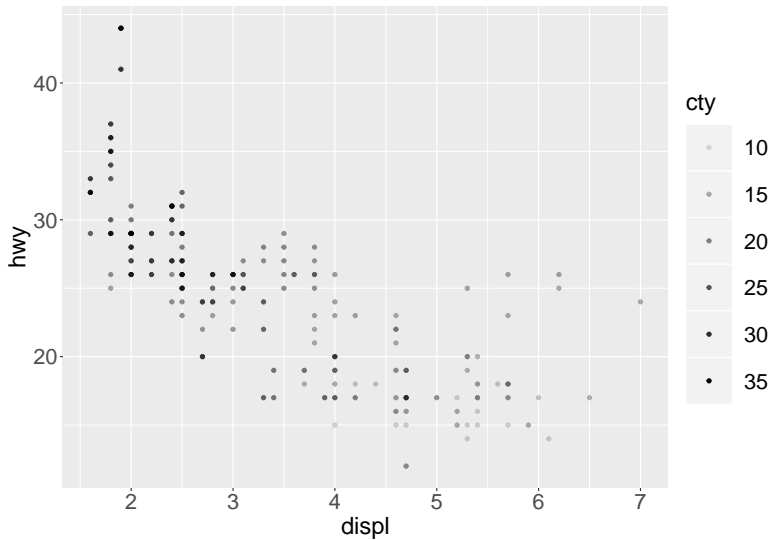


# Mapping continuous variables to aesthetics

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

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

## Mapping continuous variables to aesthetics



## Mapping continuous variables to aesthetics

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

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

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

# Aesthetic mappings: categorical vs continuous variables

## **Color** and **alpha** aesthetics:

- ▶ Each level of a categorical variable is assigned to a different (discrete) color.
- ▶ The values of continuous variables are mapped to a continuous color scale, varying from light to dark.

## **Size** aesthetic:

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

# Aesthetic mappings: categorical vs continuous variables

## **Shape** aesthetic:

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

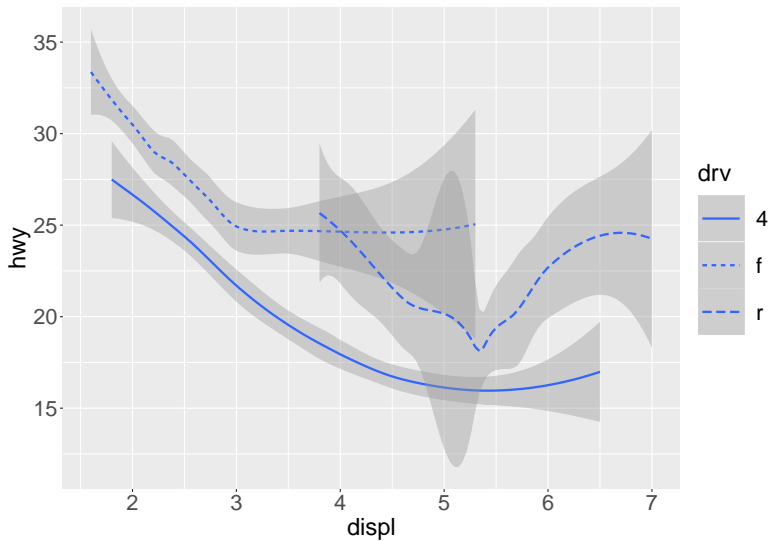
.

## The linetype aesthetic

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

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

## The linetype aesthetic



# The group aesthetic

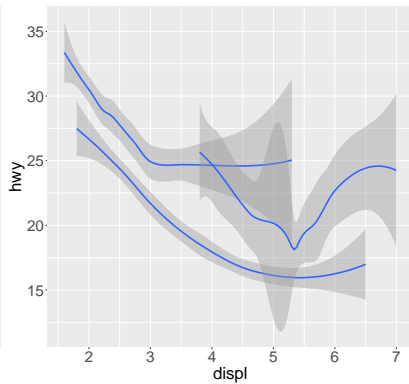
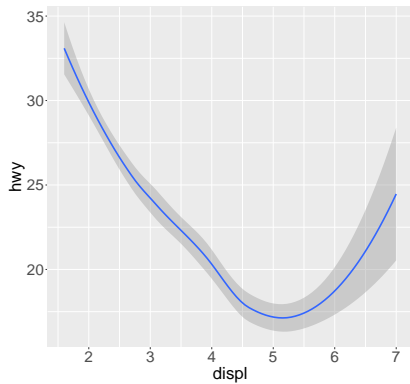
- ▶ Many geoms, like `geom_smooth()`, use a single geometric object to display multiple rows of data.
- ▶ For these geoms, you can map the group aesthetic to a categorical variable to draw multiple objects. `ggplot2` will draw a separate object for each level of the grouping variable.



## The group aesthetic

```
p1 <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy))  
  
p2 <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, group = drv))  
  
ggarrange(p1, p2, nrow = 1)
```

# The group aesthetic

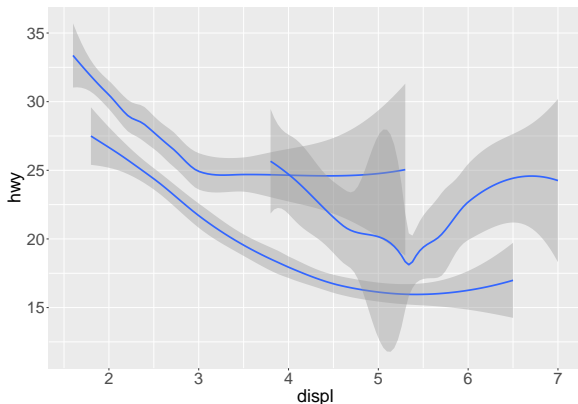


## The group aesthetic

- ▶ In practice, ggplot2 will automatically group the data for these geoms whenever you map an aesthetic to a discrete variable.
- ▶ 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.
- ▶ Compare the three examples below:

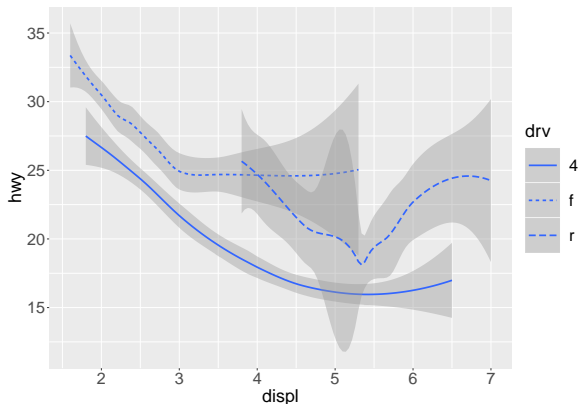
## geom\_smooth() with the group aesthetic

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



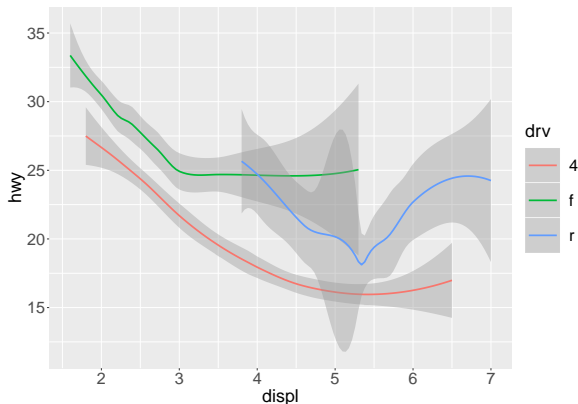
## geom\_smooth() with the linetype aesthetic

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



## geom\_smooth() with the color aesthetic

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



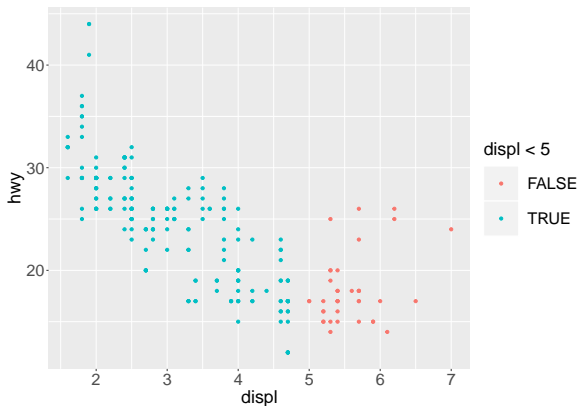
# Aesthetics and logical conditions

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

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

# Aesthetics and logical conditions

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





# Aesthetics and logical conditions

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



## Changing default aesthetic properties of geoms

- ▶ We can change the default aesthetic properties of the geom functions.
- ▶ To change a default aesthetic property, set the aesthetic by name as an argument of the geom function.
- ▶ Names of colors should be indicated as character strings.
- ▶ Size of points should be indicated in mm.

# Changing default aesthetic properties of geoms

Shapes are identified by the numbers in figure 3.

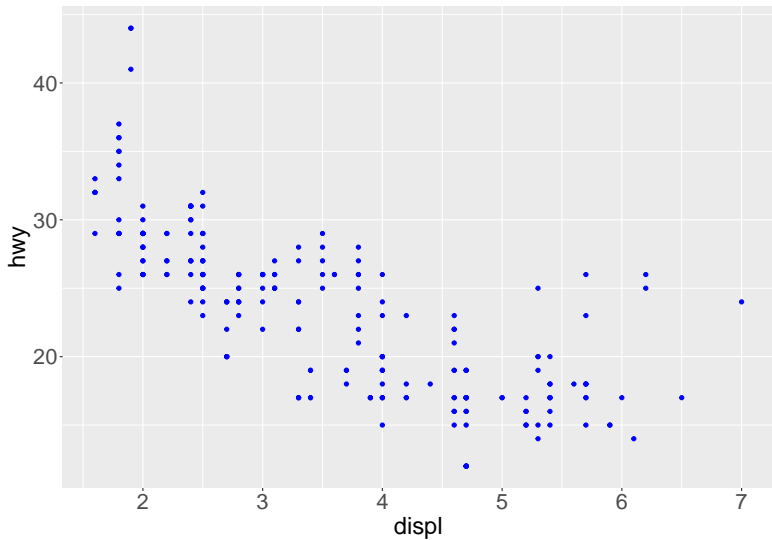
- ▶ There are some seeming duplicates (e.g. 0, 15, and 22 are all squares).
- ▶ The difference comes from the interaction of the `colour` and `fill` aesthetics:
  - ▶ The hollow shapes (0–14) have a border determined by the `colour` aesthetic
  - ▶ The solid shapes (15–18) are filled with the `colour`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

## Changing default aesthetic properties of geoms

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

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

## Changing default aesthetic properties of geoms

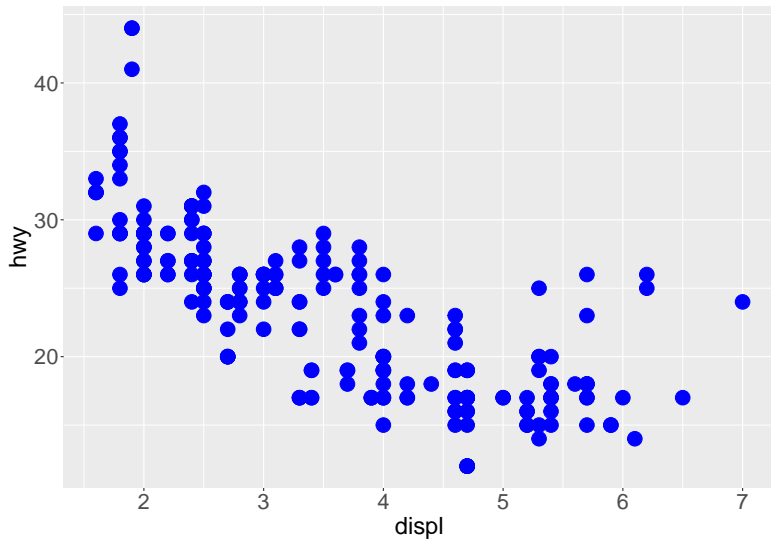


## Changing default aesthetic properties of geoms

Now let's change the size of the points:

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

## Changing default aesthetic properties of geoms



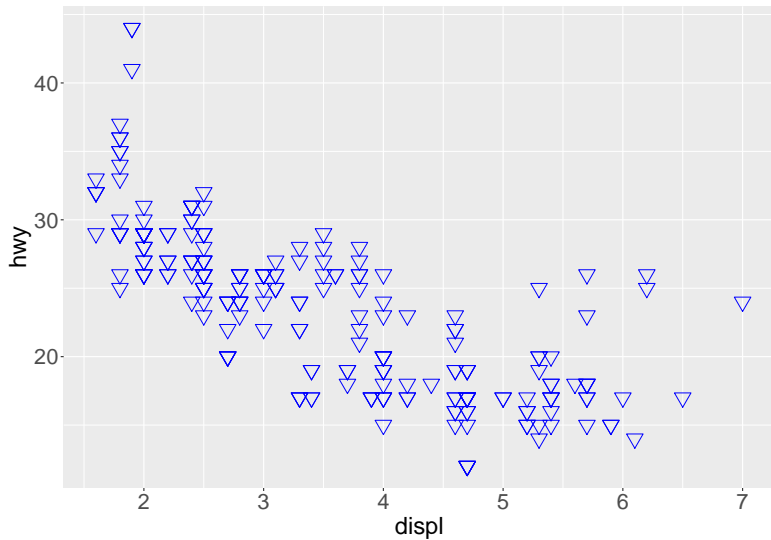
## Changing default aesthetic properties of geoms

We can also change the default shape of the points:

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



## Changing default aesthetic properties of geoms

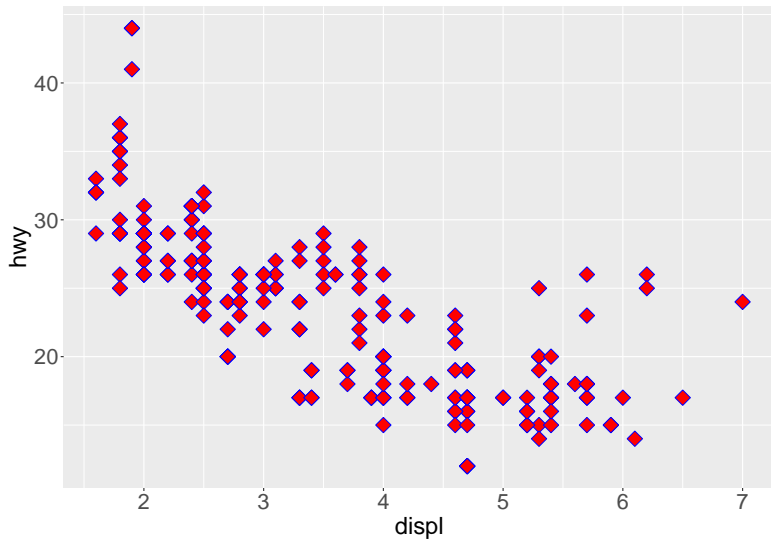


## Changing default aesthetic properties of geoms

- ▶ Shape 6 is a hollow shape, it interacts only with the color aesthetic.
- ▶ If we want triangles filled with color, we must use shape 23, which interacts both with the color and fill aesthetics:

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

## Changing default aesthetic properties of geoms

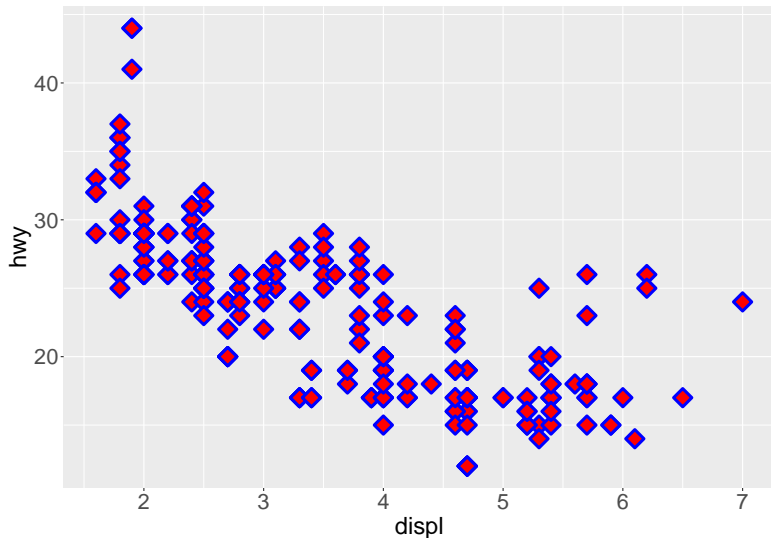


## Changing default aesthetic properties of geoms

The stroke aesthetic changes the thickness of the border of the shapes:

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

## Changing default aesthetic properties of geoms

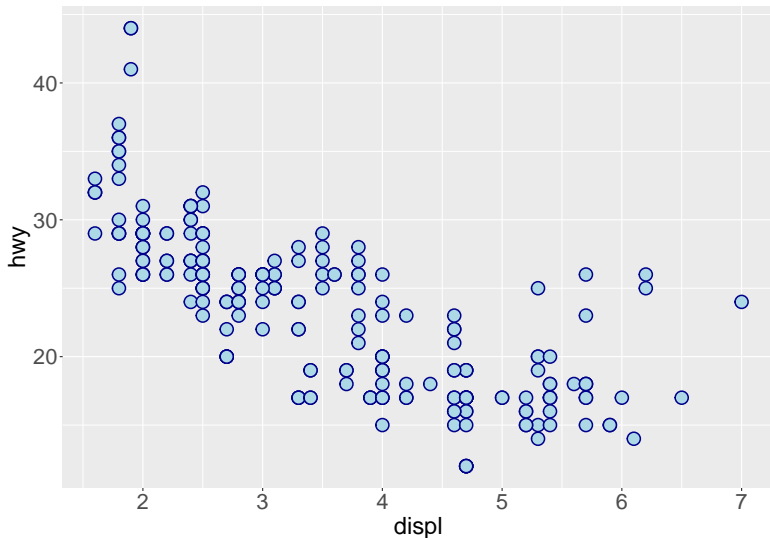


## Changing default aesthetic properties of geoms

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

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

## Changing default aesthetic properties of geoms

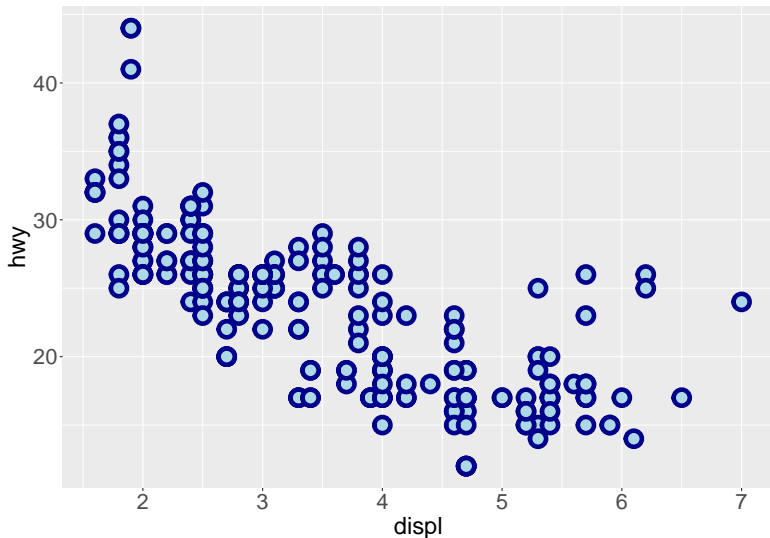


## Changing default aesthetic properties of geoms

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



## Changing default aesthetic properties of geoms



# Changing default aesthetic properties of geoms

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

# Changing default aesthetic properties of geoms

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

See:

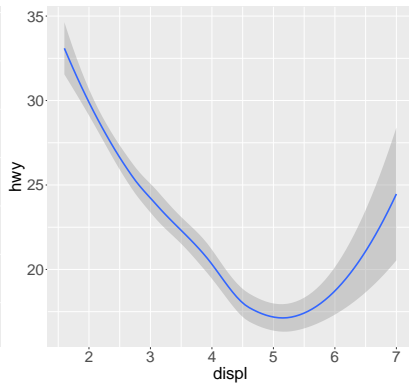
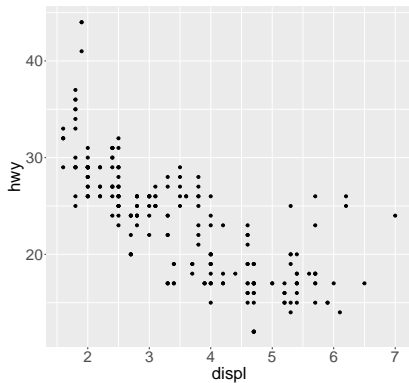
- ▶ [https://www.rapidtables.com/web/color/RGB\\_Color.html](https://www.rapidtables.com/web/color/RGB_Color.html)
- ▶ <https://htmlcolorcodes.com>

## Changing default aesthetic properties of geoms

We already saw that `geom_smooth()` adds a smoothed line:

```
left <- ggplot(data = mpg) +  
  geom_point(aes(x = displ, y = hwy))  
  
right <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy))  
  
ggarrange(left, right, nrow = 1)
```

## Changing default aesthetic properties of geoms



## Changing default aesthetic properties of geoms

We can change the default line type of geom functions like `geom_smooth()`. These are the available line types:



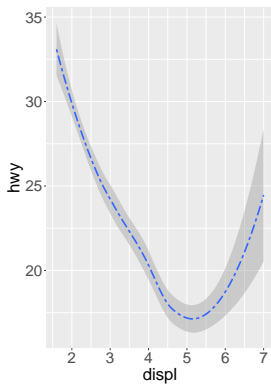
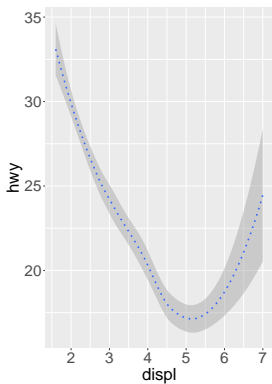
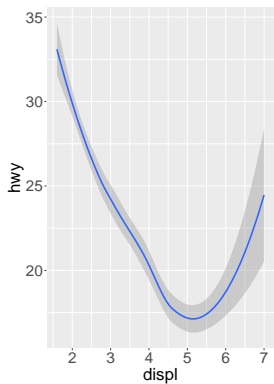
Figure 4: Built in line types

## Changing default aesthetic properties of geoms

To change the default linetype, set it as an argument of the geom function:

```
left <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy))  
  
center <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy),  
    linetype = "dotted")  
  
right <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy),  
    linetype = "twodash")  
  
ggarrange(left, center, right, nrow = 1)
```

## Changing default aesthetic properties of geoms



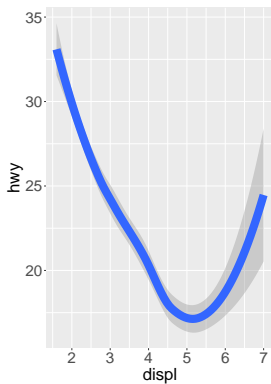
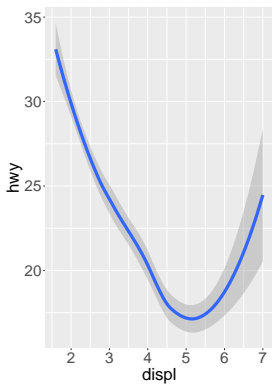
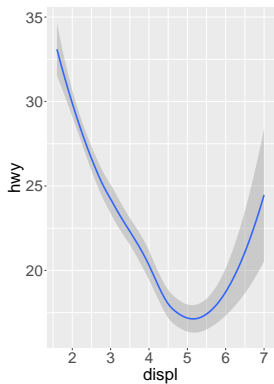


## Changing default aesthetic properties of geoms

We can also change the thickness of the lines:

```
left <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy))  
  
center <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy), size = 2)  
  
right <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy), size = 5)  
  
ggarrange(left, center, right, nrow = 1)
```

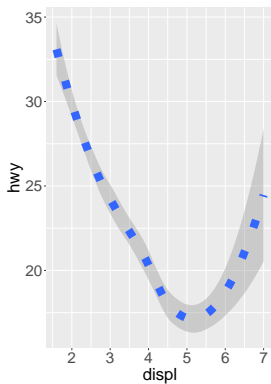
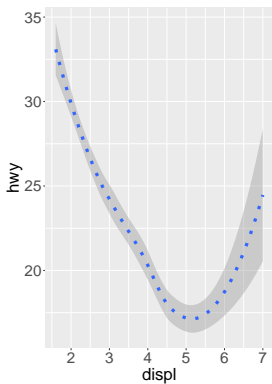
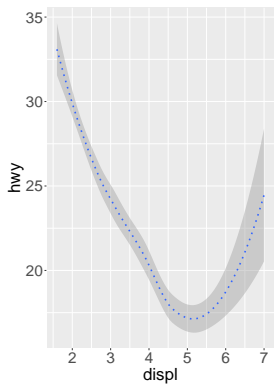
## Changing default aesthetic properties of geoms



## Changing default aesthetic properties of geoms

```
left <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy),  
              linetype = "dotted")  
  
center <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy),  
              linetype = "dotted", size = 2)  
  
right <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy),  
              linetype = "dotted", size = 5)  
  
ggarrange(left, center, right, nrow = 1)
```

## Changing default aesthetic properties of geoms



## Changing default aesthetic properties of geoms

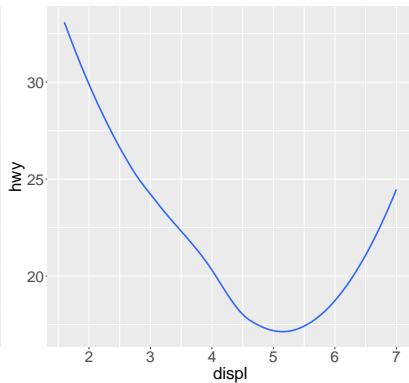
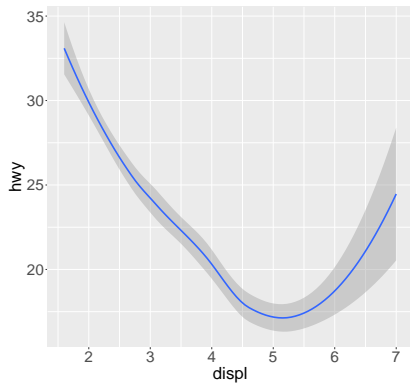
- ▶ By default, `geom_smooth()` displays confidence intervals around the smoothed line.
- ▶ Confidence intervals may be disabled by setting the `se` (standard error) aesthetic to `FALSE`:

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

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

```
ggarrange(left, right, nrow = 1)
```

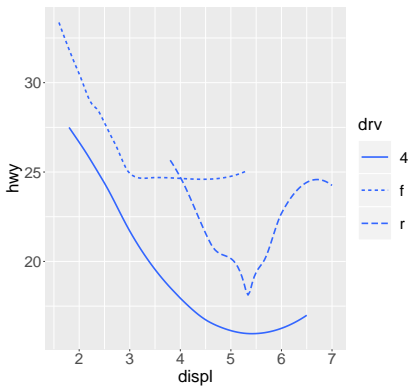
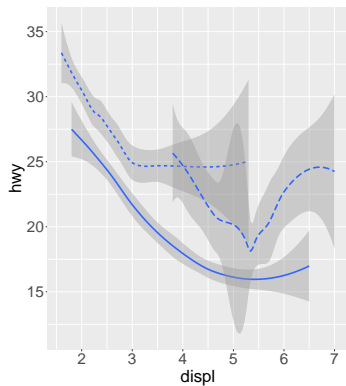
## Changing default aesthetic properties of geoms



## Changing default aesthetic properties of geoms

```
left <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, linetype = drv))  
  
right <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, linetype = drv),  
    se = FALSE)  
  
ggarrange(left, right, nrow = 1)
```

# Changing default aesthetic properties of geoms

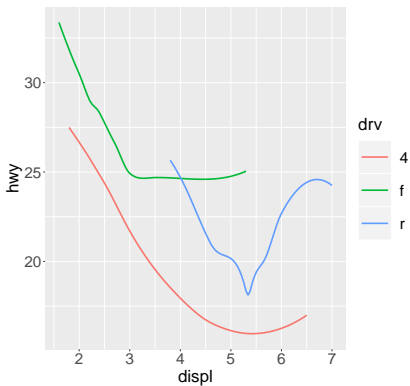
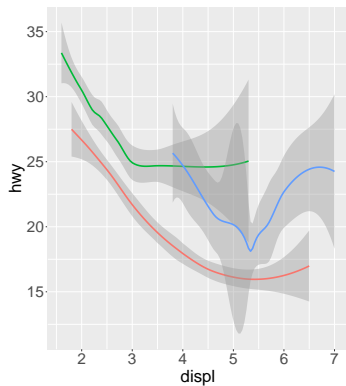




## Changing default aesthetic properties of geoms

```
l <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, color = drv))  
  
r <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, color = drv),  
    se = FALSE)  
  
ggarrange(l, r, nrow = 1)
```

# Changing default aesthetic properties of geoms

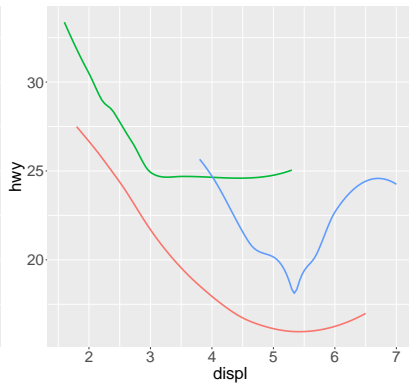
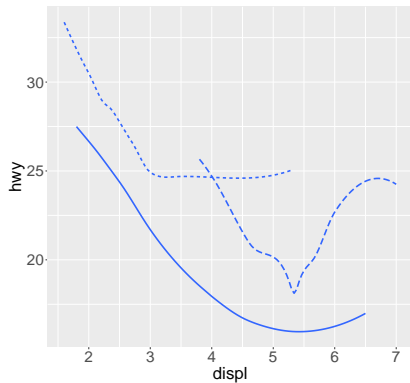


## Changing default aesthetic properties of geoms

Legends can also be disabled:

```
l <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, linetype = drv),  
    se = FALSE)  
  
r <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, color = drv),  
    se = FALSE)  
  
ggarrange(l, r, nrow = 1)
```

# Changing default aesthetic properties of geoms



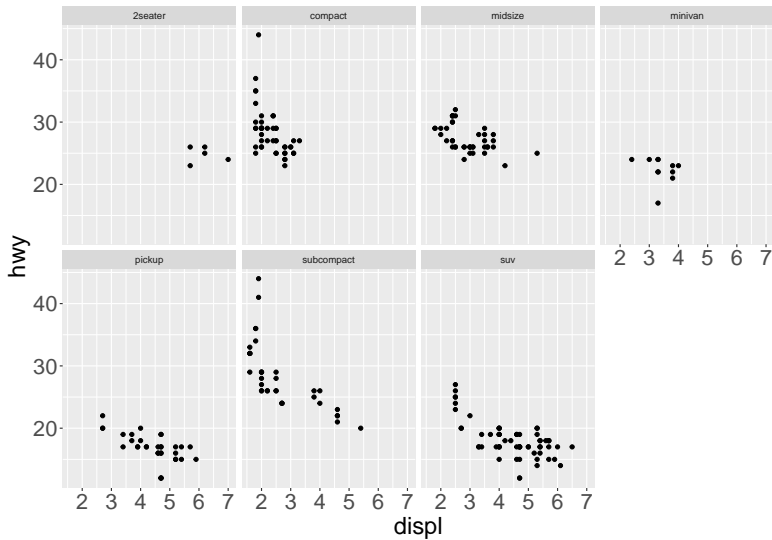
# Facets

- ▶ One way to add additional variables to a plot is with **aesthetics**.
- ▶ Another way, particularly useful for categorical variables, is to split 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()` should be "~" followed the name of a variable.

# Facets

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

# Facets

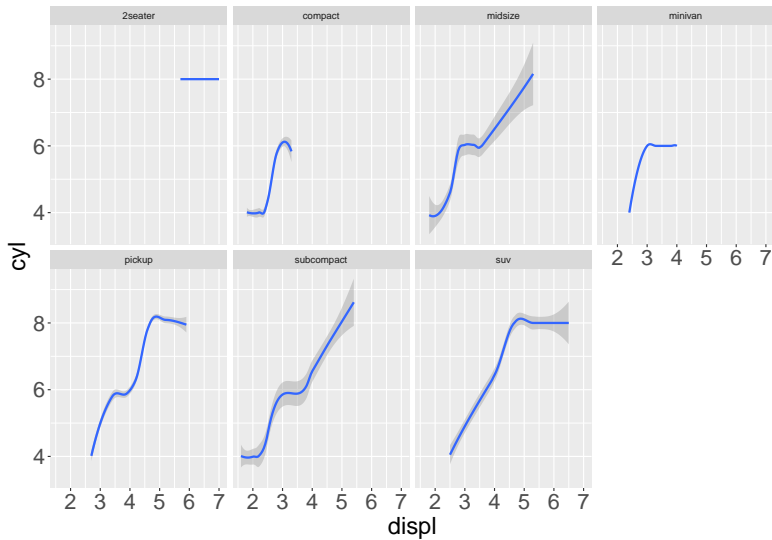


# Facets

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



# Facets

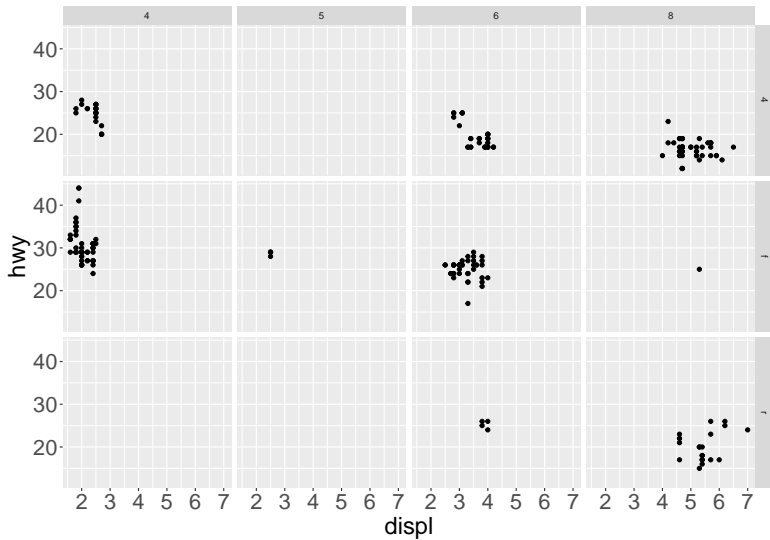


# Facets

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

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

# Facets

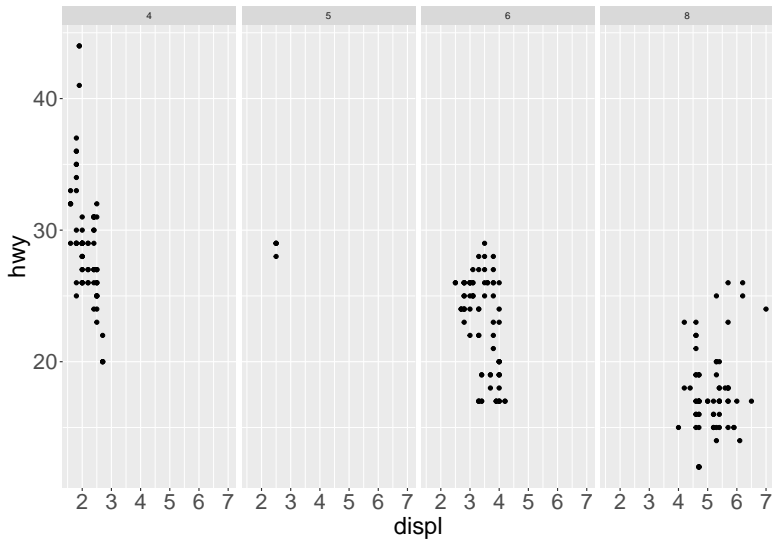


# Facets

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

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

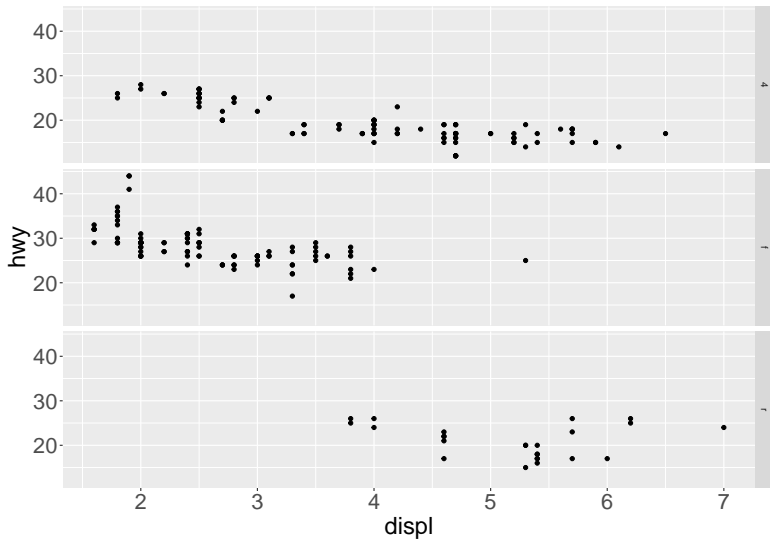
# Facets



# Facets

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

# Facets



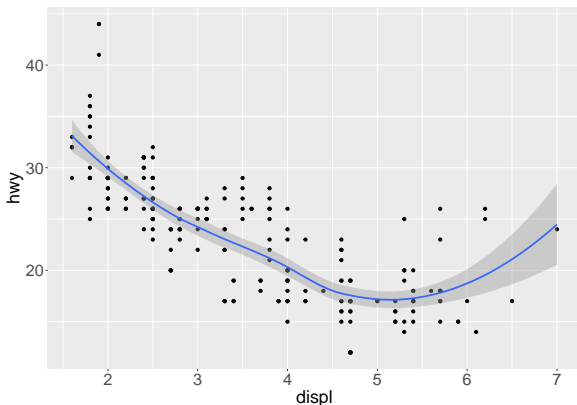
## Multiple geoms

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



## Multiple geoms

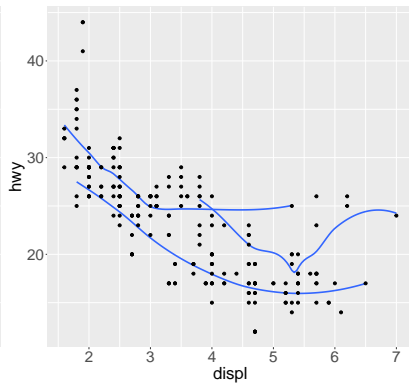
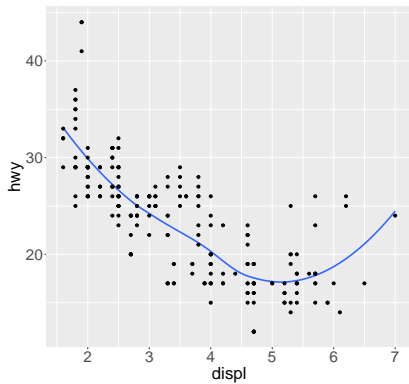
```
ggplot(data = mpg) +  
  geom_point(aes(x = displ, y = hwy)) +  
  geom_smooth(aes(x = displ, y = hwy))
```



## Multiple geoms

```
p1 <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy), se = FALSE) +  
  geom_point(aes(x = displ, y = hwy))  
  
p2 <- ggplot(data = mpg) +  
  geom_smooth(aes(x = displ, y = hwy, group = drv),  
              se = FALSE) +  
  geom_point(aes(x = displ, y = hwy))  
  
ggarrange(p1, p2, nrow = 1)
```

# Multiple geoms



## Multiple geoms

- ▶ This, however, induces duplication in our code.
- ▶ We built the same aesthetic mapping twice in each graph.
- ▶ We can avoid this type of repetition by passing a set of mappings to `ggplot()`.
- ▶ `ggplot2` will treat these mappings as global mappings that apply to each geom in the graph.

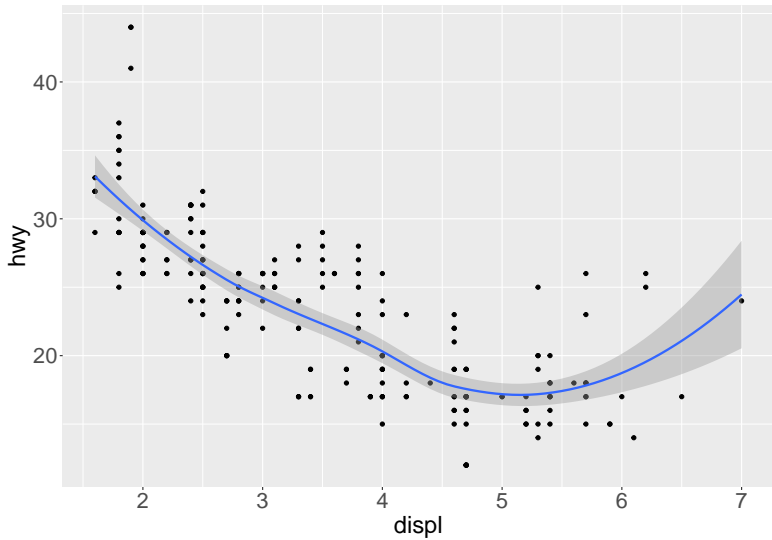
## Multiple geoms

These two chunks of code result in the same plot:

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

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

## Multiple geoms



## Multiple geoms

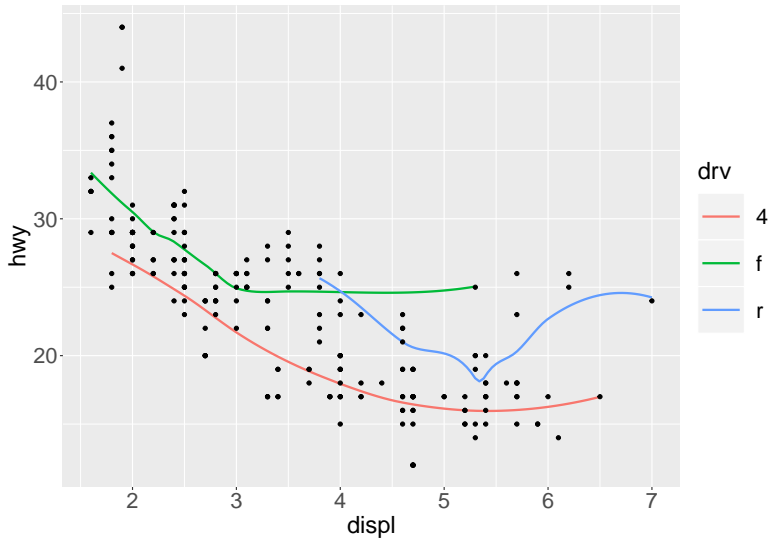
- ▶ If you place mappings in a geom function, ggplot2 will treat them as local mappings for that layer.
- ▶ It will use these mappings to extend or overwrite the global mappings for that layer only.
- ▶ This makes it possible to display different aesthetics in different layers.

## Multiple geoms

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



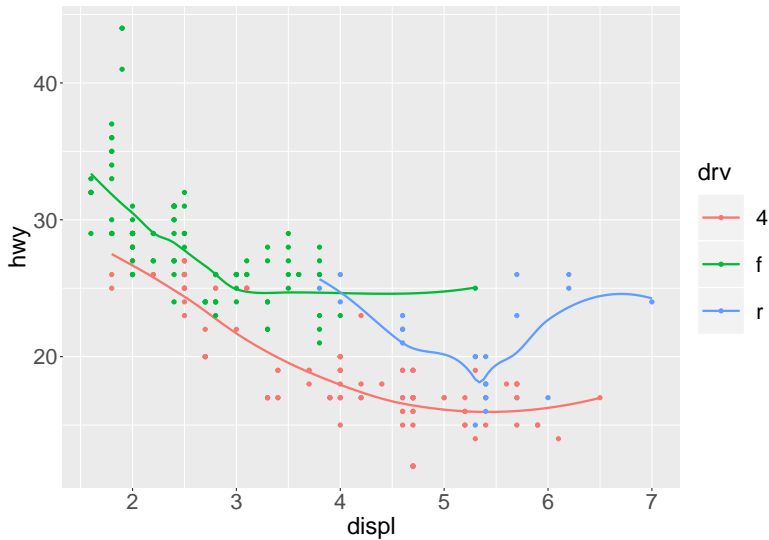
## Multiple geoms



## Multiple geoms

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

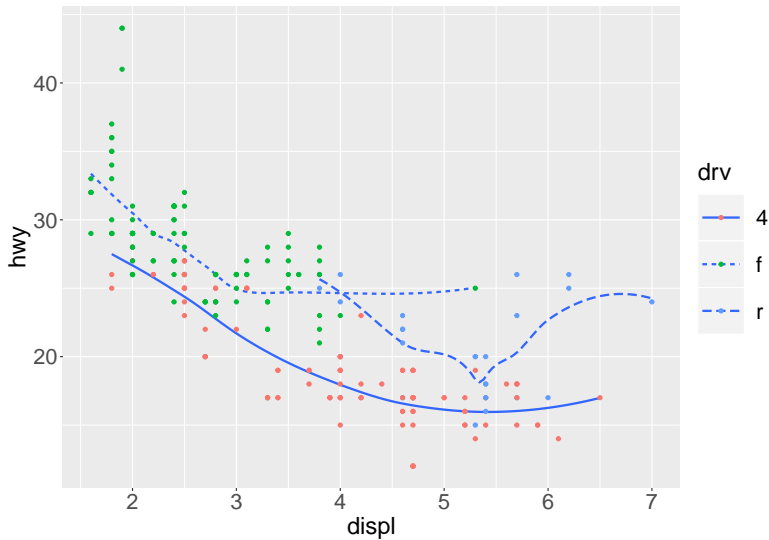
## Multiple geoms



## Multiple geoms

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

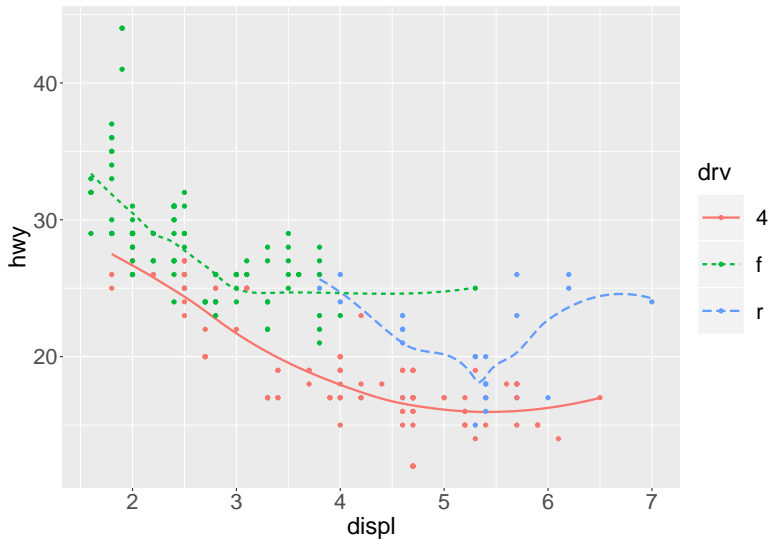
## Multiple geoms



## Multiple geoms

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

## Multiple geoms

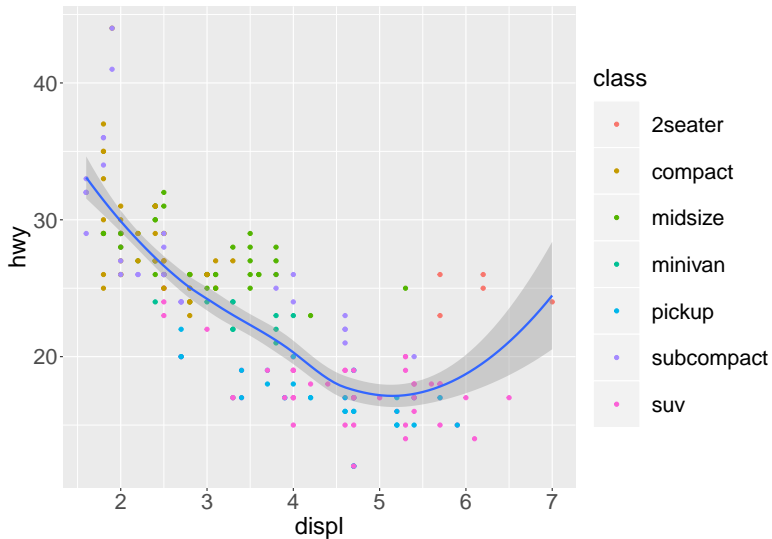


## Multiple geoms

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



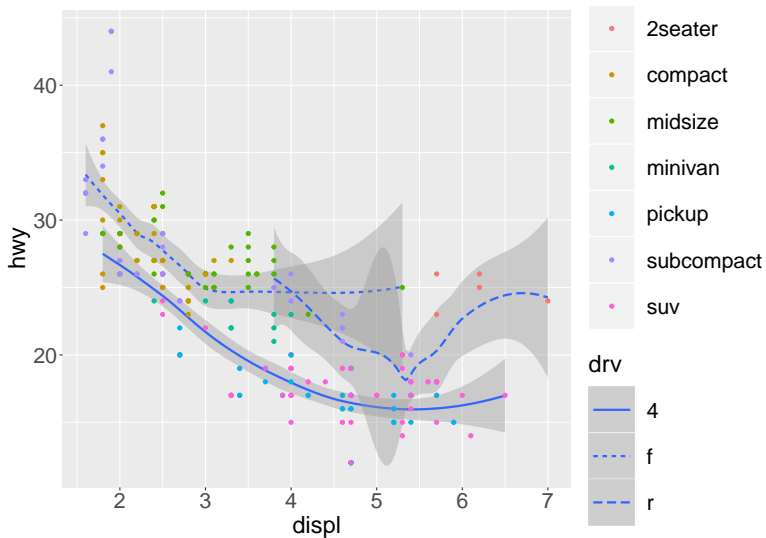
## Multiple geoms



## Multiple geoms

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

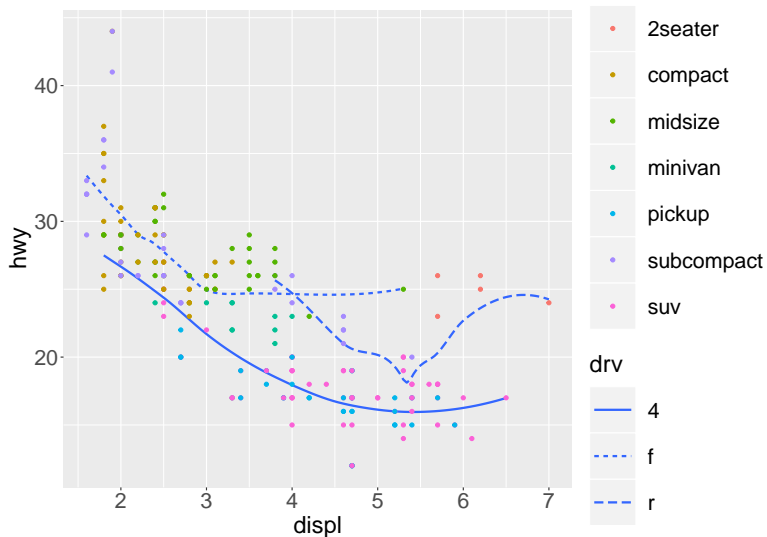
## Multiple geoms



## Multiple geoms

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

# Multiple geoms



## Multiple geoms

- ▶ You can use the same logic to specify different data for each layer.

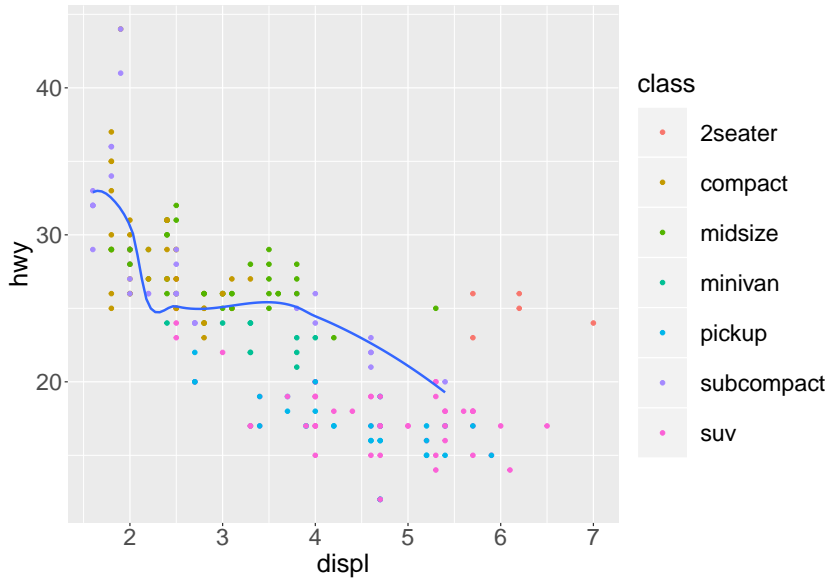
## Multiple geoms

In this example:

- ▶ The smooth line displays just a subset of the mpg dataset, the subcompact cars.
- ▶ The local data argument in `geom_smooth()` overrides the global data argument in `ggplot()` for that layer only.

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

## Multiple geoms

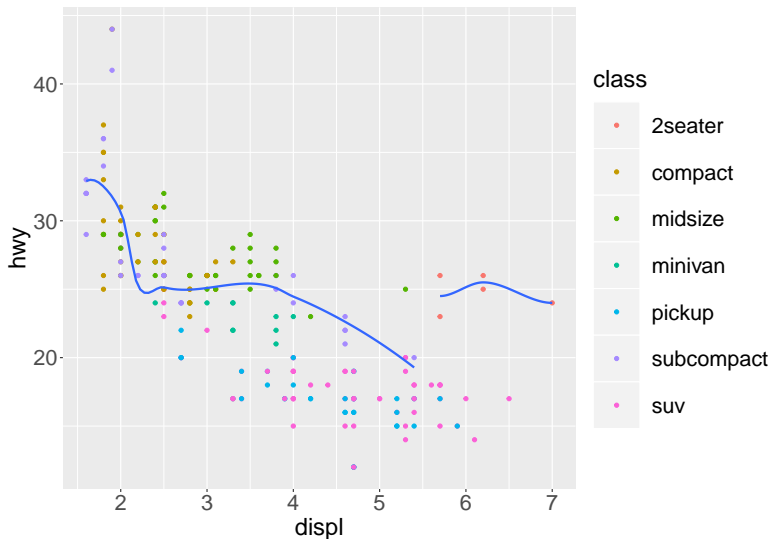




## Multiple geoms

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

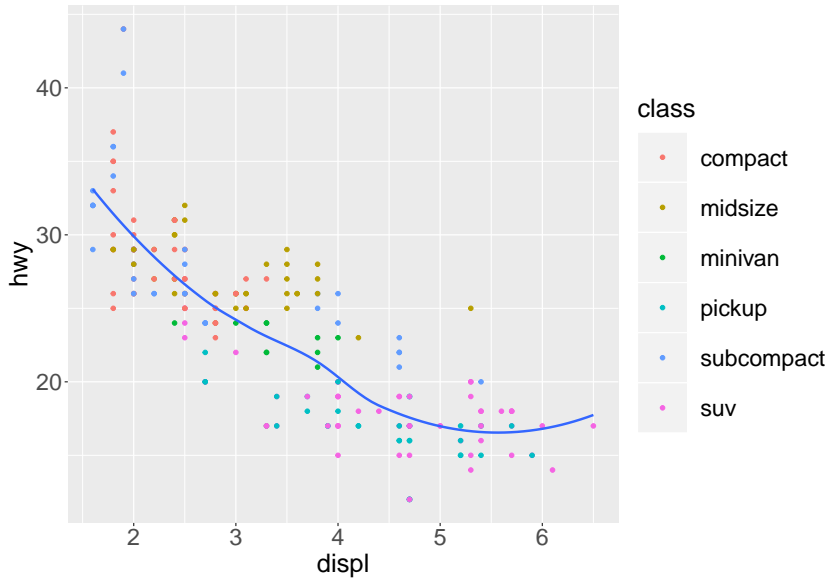
## Multiple geoms



## Multiple geoms

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

## Multiple geoms



## Multiple geoms

```
ggplot(data = subset(mpg, class != "2seater"),
       aes(x = displ, y = hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(
    data = subset(mpg,
                  class %in% c("suv", "subcompact")),
    aes(linetype = class), se = FALSE)
```

## Multiple geoms



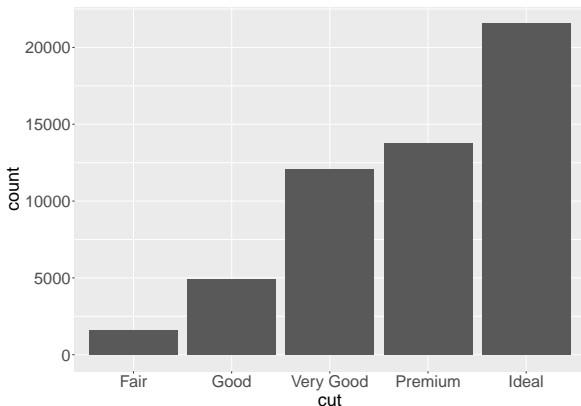
# The diamonds dataset

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

## Bar charts

Let's use a bar chart to display the number of diamonds for each quality of cut:

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





# Statistical transformations

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

# Statistical transformations

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

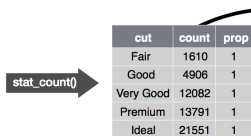
## Statistical transformations

1. **geom\_bar()** begins with the **diamonds** data set

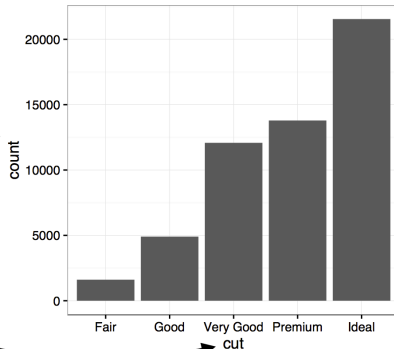
[illegible]

# Statistical transformations

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



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



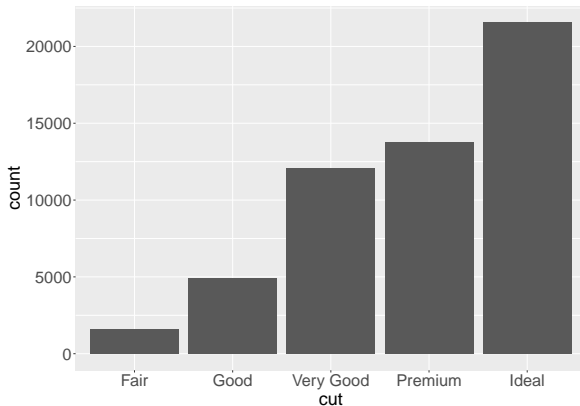
# Statistical transformations

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

# Statistical transformations

For example, you can recreate the previous plot using `stat_count()` instead of `geom_bar()`:

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



# Statistical transformations

This works because:

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

# Overwriting the default stat

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

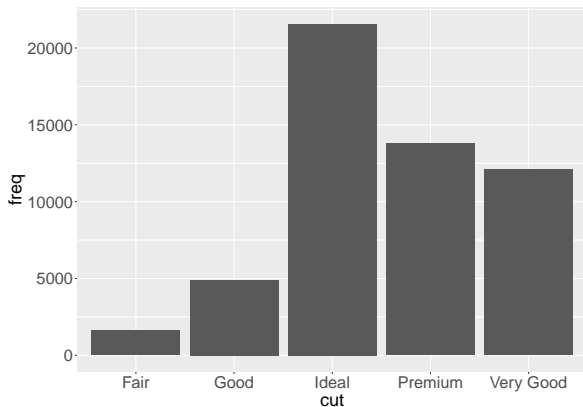
- ▶ We may have a table with column heights.
- ▶ In that case, we need to change the default statistical transformation

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



## Overriding the default stat

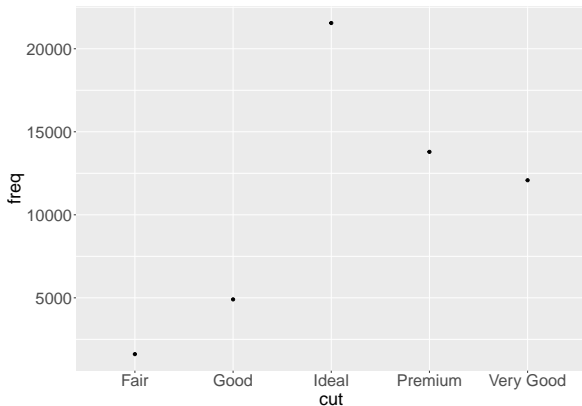
```
ggplot(data = tib) +  
  geom_bar(aes(x = cut, y = freq),  
    stat = "identity")
```



# Overriding the default stat

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

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

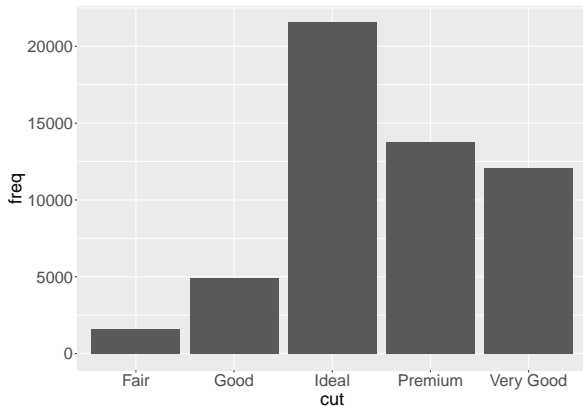


## geom\_col()

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

```
geom_col()
```

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



# Statistical transformations

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

# Statistical transformations

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

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

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

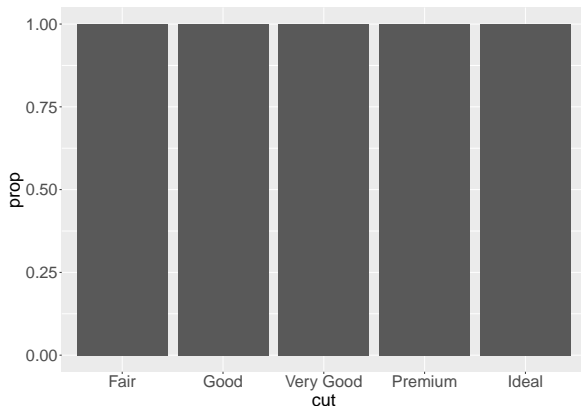
## Mapping from transformed variables to aesthetics

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

# Mapping from transformed variables to aesthetics

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

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



What went wrong?



## Mapping from transformed variables to aesthetics

Lets see the computed values:

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

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

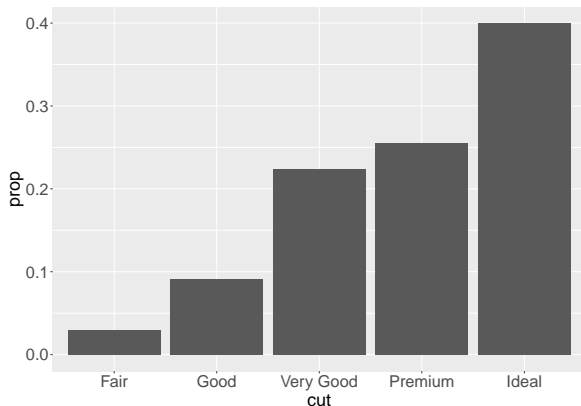
| ##   | y | count | prop | x | PANEL | group | ymin | ymax |
|------|---|-------|------|---|-------|-------|------|------|
| ## 1 | 1 | 1610  | 1    | 1 | 1     | 1     | 0    | 1    |
| ## 2 | 1 | 4906  | 1    | 2 | 1     | 2     | 0    | 1    |
| ## 3 | 1 | 12082 | 1    | 3 | 1     | 3     | 0   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

## Mapping from transformed variables to aesthetics

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

## Mapping from transformed variables to aesthetics

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



## Mapping from transformed variables to aesthetics

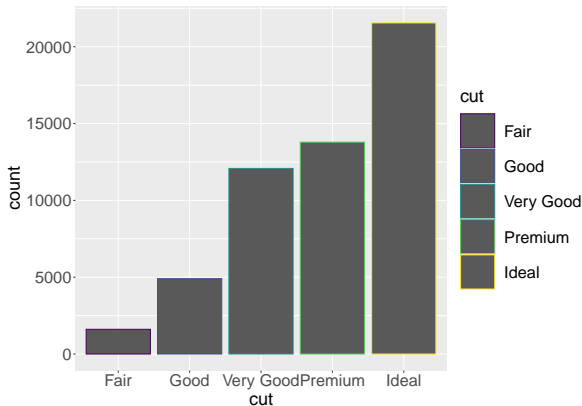
```
plt_3 <- ggplot(data = diamonds) +  
  geom_bar(aes(x = cut, y = stat(prop), group = 1))  
plt_3 <- ggplot_build(plt_3)  
  
plt_3$data[[1]][, 1:5]
```

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

# Aesthetic of bar charts

You can map the color aesthetic to the grouping variable:

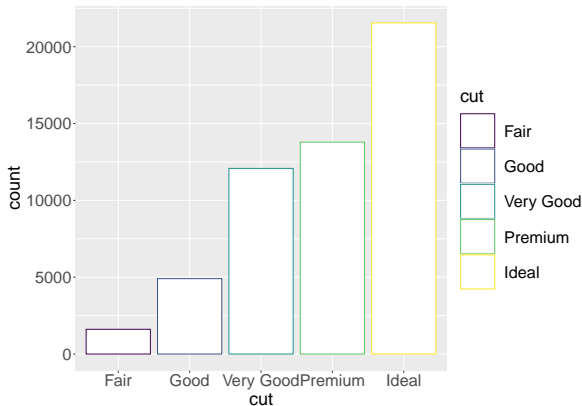
```
ggplot(data = diamonds) +  
  geom_bar(aes(x = cut, color = cut))
```



# Aesthetic of bar charts

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

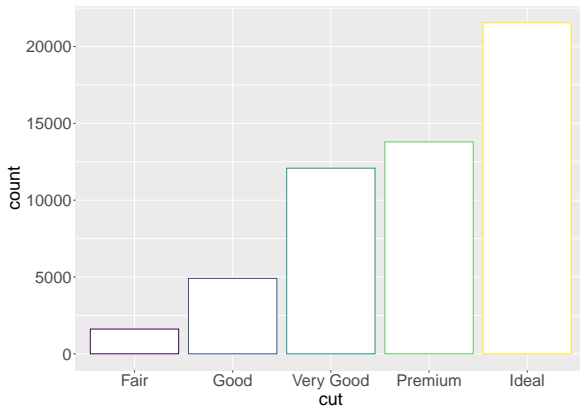
```
ggplot(data = diamonds) +  
  geom_bar(aes(x = cut, color = cut), fill = "white")
```



# Aesthetic of bar charts

And disable the legend:

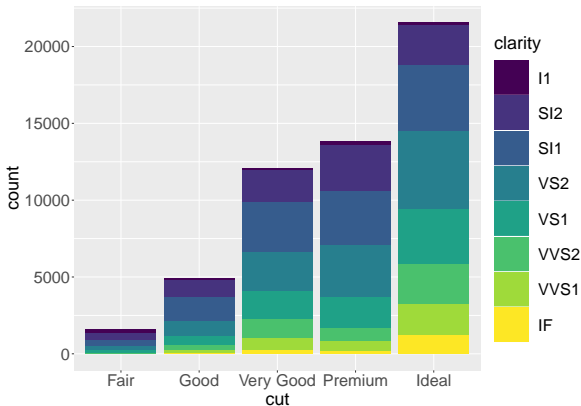
```
ggplot(data = diamonds) +  
  geom_bar(aes(x = cut, color = cut),  
           fill = "white", show.legend = FALSE)
```



## Stacked bar charts

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

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





## Position adjustments

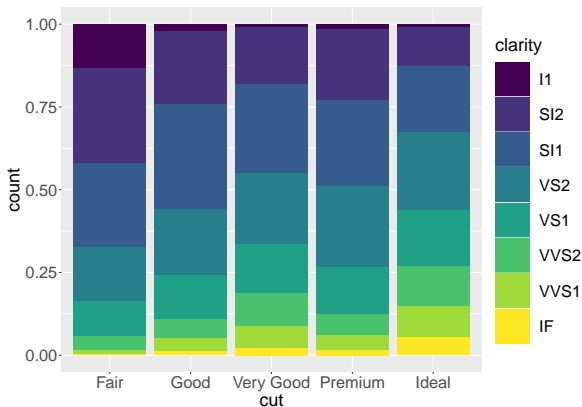
- ▶ Each colored rectangle represents a combination of cut and clarity.
- ▶ The stacking is performed automatically by the position adjustment specified in the `position` argument of the `geom` function.
- ▶ The default value is `stack`.

## Position adjustments

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

# Position adjustments

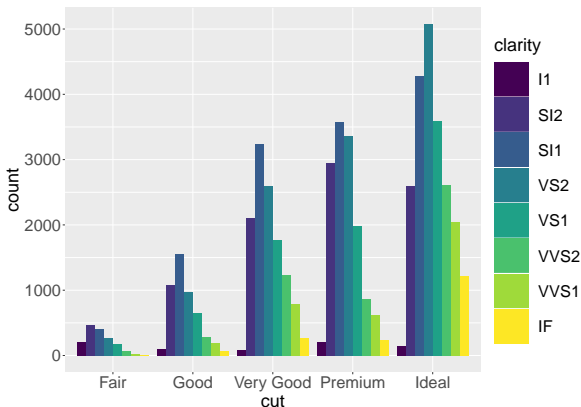
```
ggplot(data = diamonds) +  
  geom_bar(aes(x = cut, fill = clarity),  
    position = "fill")
```



## Position adjustments

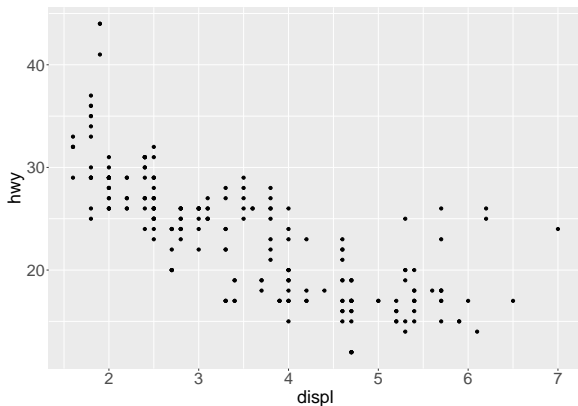
Setting `position = "dodge"` places overlapping objects directly beside one another:

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



## Position adjustments

Recall our first scatterplot:



Did you notice that the plot displays only 126 points, even though there are 234 observations in the dataset?

## Position adjustments

- ▶ The values of `hwy` and `displ` are rounded, and many points overlap each other.
- ▶ This problem is known as overplotting.
- ▶ This makes it hard to see where the mass of the data is.
- ▶ Are the data points spread equally throughout the graph, or is there one special combination of `hwy` and `displ` that contains 109 values?

## Position adjustments

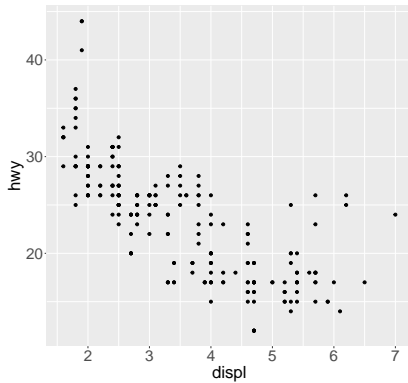
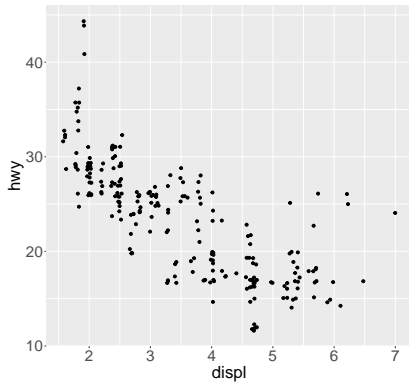
- ▶ Setting `position = "jitter"` adds a small amount of random noise to each point, spreading the points.
- ▶ While it makes your graph less accurate at small scales, it makes your graph more revealing at large scales.
- ▶ `ggplot2` comes with a shorthand for `geom_point(position = "jitter")`:
  - ▶ `geom_jitter()`.

## Position adjustments

```
p1 <- ggplot(data = mpg) +  
  geom_point(aes(x = displ, y = hwy),  
             position = "jitter")  
  
p2 <- ggplot(data = mpg) +  
  geom_point(aes(x = displ, y = hwy))  
  
ggarrange(p1, p2, nrow = 1)
```



# Position adjustments



## Position adjustments

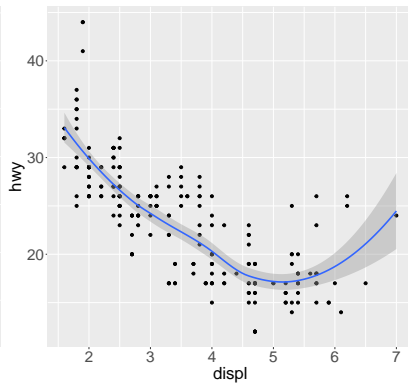
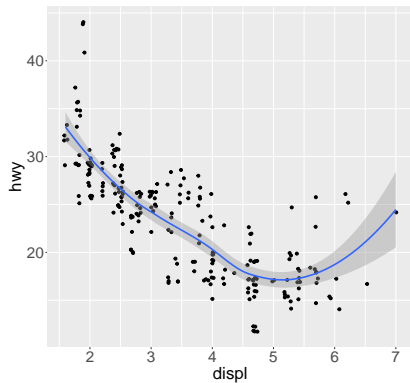
```
p1 <- ggplot(data = mpg, aes(x = displ, y = hwy)) +  
  geom_point(position = "jitter") +  
  geom_smooth()
```

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

```
ggarrange(p1, p2, nrow = 1)
```

# Position adjustments

The confidence interval of the smoothed lines can also help:



## Position adjustments

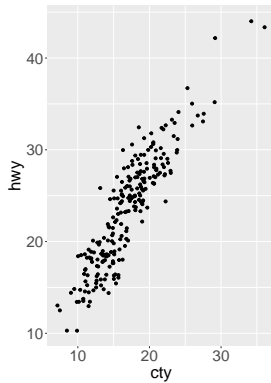
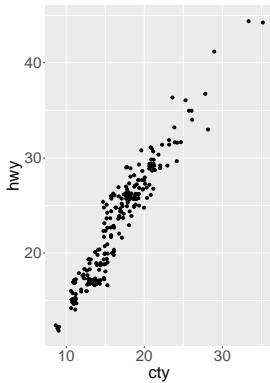
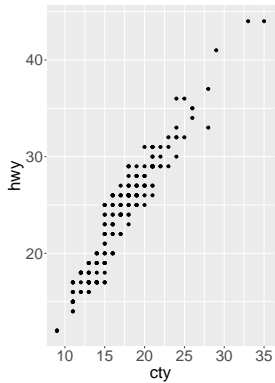
There are two optional arguments to jitter:

- ▶ Width controls the amount of vertical displacement.
- ▶ Height controls the amount of horizontal displacement.

## Position adjustments

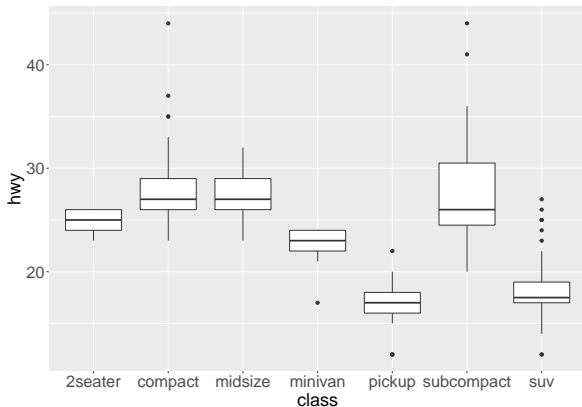
```
p1 <- ggplot(data = mpg, aes(x = cty, y = hwy)) +  
  geom_point()  
  
p2 <- ggplot(data = mpg, aes(x = cty, y = hwy)) +  
  geom_jitter()  
  
p3 <- ggplot(data = mpg, aes(x = cty, y = hwy)) +  
  geom_jitter(height = 2, width = 2)  
  
ggarrange(p1, p2, p3, nrow = 1)
```

# Position adjustments



# Boxplots

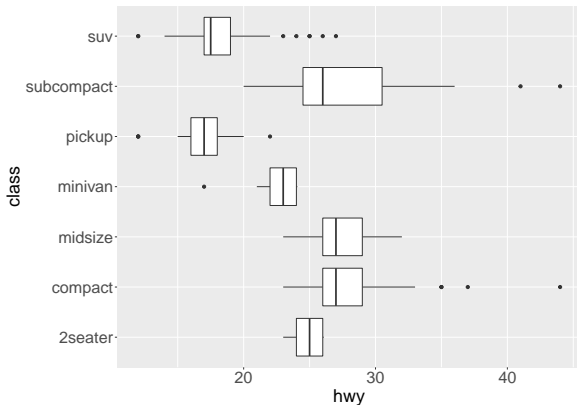
```
ggplot(data = mpg, aes(x = class, y = hwy)) +  
  geom_boxplot()
```



# Coordinate systems

`coord_flip()` flips the coordinate system:

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

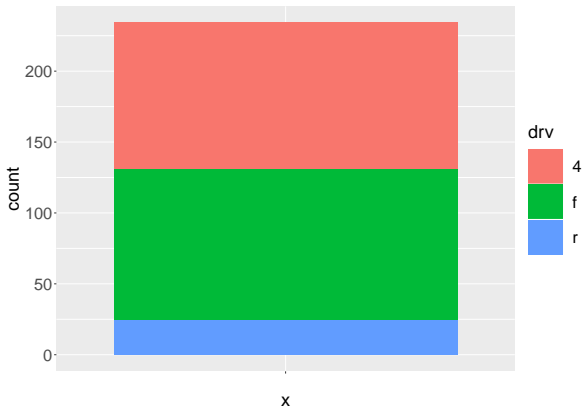




# Coordinate systems

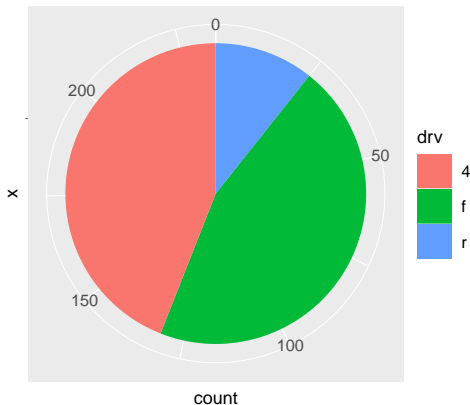
A pie chart is a stacked bar chart with polar coordinates:

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



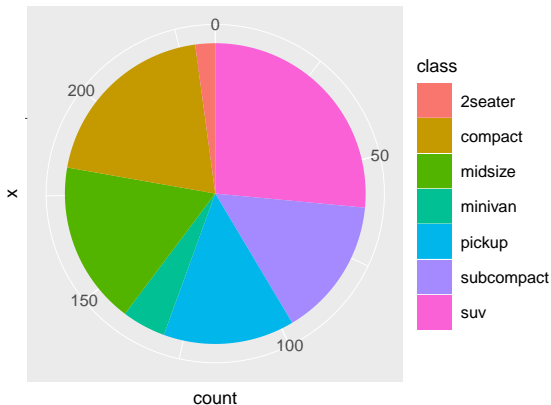
# Coordinate systems

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



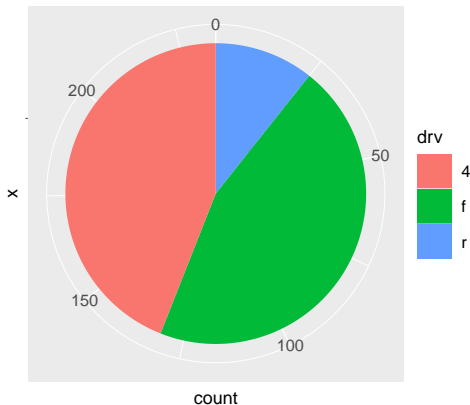
# Coordinate systems

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



# Coordinate systems

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



# The ggplot template

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

# Bibliography

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

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