# Data-based Statistical Decision Model

## *Lecture 4 (Part III) - Data Visualization: ggplot2*

*Sungkyu Jung*

# Prerequisite: Loading packages

```
library(tidyverse)
#library(mosaic)
```

The `tidyverse` (https://www.tidyverse.org/ (https://www.tidyverse.org/)) is an opinionated collection of R packages designed for data science, managed by a group of people including Hadley Wickham, statistician and chief scientist at RStudio, Inc. See the excerpt of the Tidyverse slides at Tidyverse Slide (Tidyverse-intro.pdf). We use packages `dplyr` and `ggplot2` as part of `tidyverse`.

# ggplot2

The `ggplot2` package is the primary tool of data visualization, and implements *the grammar of graphics* in the book "The Grammar of Graphics" by Leland Wilkinson, now chief scientist at h2o, Inc (https://www.h2o.ai/). The four elements of graphics identified by Yau (Visual Cues, Coordinate System, Scale and Context) are also found in the grammar of graphics, albeit by different terms. Thus, it is essential to understand the taxonomy of graphics in order to use `ggplot2`.

# `ggplot2::mpg` data example

We will follow the examples in *R for data science* (http://r4ds.had.co.nz/data-visualisation.html). Let's first look at the data set. `mpg` contains observations collected by the US Environment Protection Agency on 38 models of car. `mpg` is a `tibble`, which is a simplified `data.frame`, modified for better handling large data. For now, it is okay to think a `tibble` as a `data.frame`.

```
class(mpg)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
mpg
```

| manufacturer<br><chr> | model<br><chr> | displ<br><dbl> | year<br><int> | cyl<br><int> | trans<br><chr> | …<br><chr> | …<br><int> | …<br><int> | fl<br><chr> |
|---|---|---|---|---|---|---|---|---|---|
| audi | a4 | 1.8 | 1999 | 4 | auto(l5) | f | 18 | 29 | p |
| audi | a4 | 1.8 | 1999 | 4 | manual(m5) | f | 21 | 29 | p |
| audi | a4 | 2.0 | 2008 | 4 | manual(m6) | f | 20 | 31 | p |
| audi | a4 | 2.0 | 2008 | 4 | auto(av) | f | 21 | 30 | p |
| audi | a4 | 2.8 | 1999 | 6 | auto(l5) | f | 16 | 26 | p |

| manufacturer<br><chr> | model<br><chr> | displ<br><dbl> | year<br><int> | cyl<br><int> | trans<br><chr> | … | … | … | fl<br><chr> |
|---|---|---|---|---|---|---|---|---|---|
| audi | a4 | 2.8 | 1999 | 6 | manual(m5) | f | 18 | 26 | p |
| audi | a4 | 3.1 | 2008 | 6 | auto(av) | f | 18 | 27 | p |
| audi | a4 quattro | 1.8 | 1999 | 4 | manual(m5) | 4 | 18 | 26 | p |
| audi | a4 quattro | 1.8 | 1999 | 4 | auto(l5) | 4 | 16 | 25 | p |
| audi | a4 quattro | 2.0 | 2008 | 4 | manual(m6) | 4 | 20 | 28 | p |

1-10 of 234 rows | 1-10 of 11 columns  Previous **1** 2 3 4 5 6 … 24 Next

Among the variables in `mpg` are:

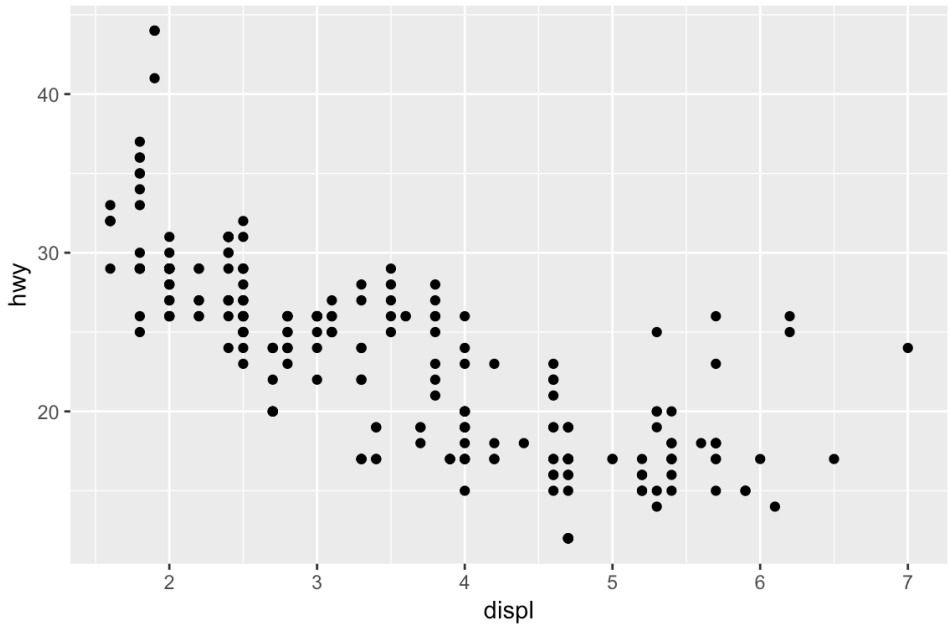
1. `displ`, a car's engine size, in litres.
2. `hwy`, a car's fuel efficiency on the highway, in miles per gallon (mpg). A car with a low fuel efficiency consumes more fuel than a car with a high fuel efficiency when they travel the same distance.

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

# Creating a ggplot

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))
```




With ggplot2, you begin a plot with the function `ggplot()`. `ggplot()` creates a coordinate system that you can add layers to. The first argument of `ggplot()` is the dataset to use in the graph. So `ggplot(data = mpg)` creates an empty graph.

You complete your graph by adding one or more layers to `ggplot()`. The function `geom_point()` adds a layer of points to your plot, which creates a scatterplot. ggplot2 comes with many geom functions that each add a different type of layer to a plot.

Each geom function in ggplot2 takes a `mapping` argument. This defines how variables in your dataset are mapped to visual properties. The `mapping` argument is always paired with `aes()`, and the `x` and `y` arguments of `aes()` specify which variables to map to the x and y axes. ggplot2 looks for the mapped variable in the data argument, in this case, `mpg`.

In connection to the four elements of data graphics,

1. `ggplot()` (by default) sets the coordinate system as the *Cartesian coordinate system*;
2. Visual cue used is the *position*, set by `mapping = aes(x = ..., y = ...)`, paired with the use of `geom_point()`;
3. *scale* is automatically chosen as appropriate as possible;
4. *context* is (minimally) given by the axis labels.

# A graphing template

To make a graph, replace the bracketed sections in the code below with a dataset, a geom function, or a collection of mappings.

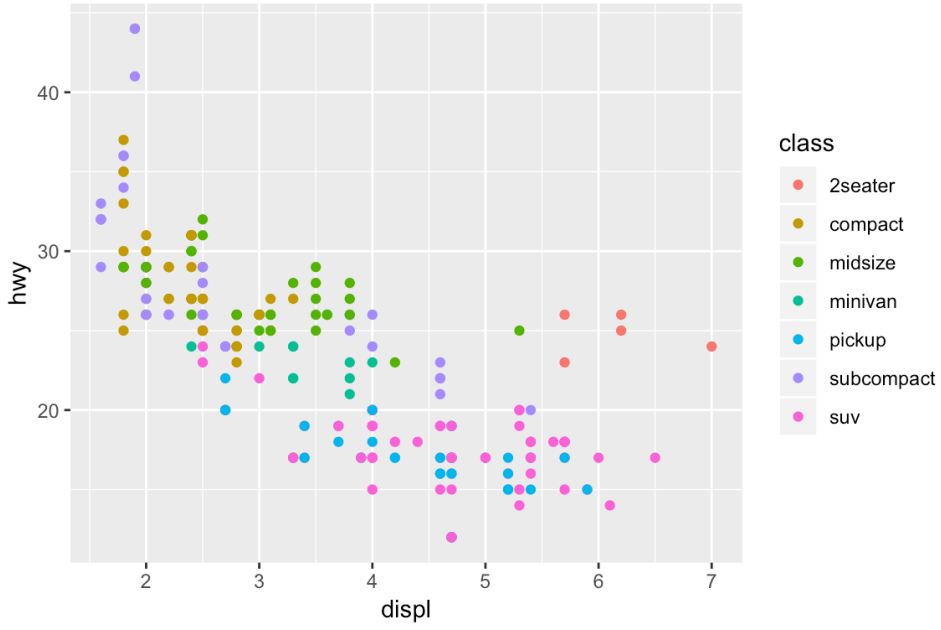
```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

# Adding more visual cues

You can add a third variable, like `class`, 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.

You can convey information about your data by mapping the aesthetics in your plot to the variables in your dataset. For example, you can map the colors of your 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))
```

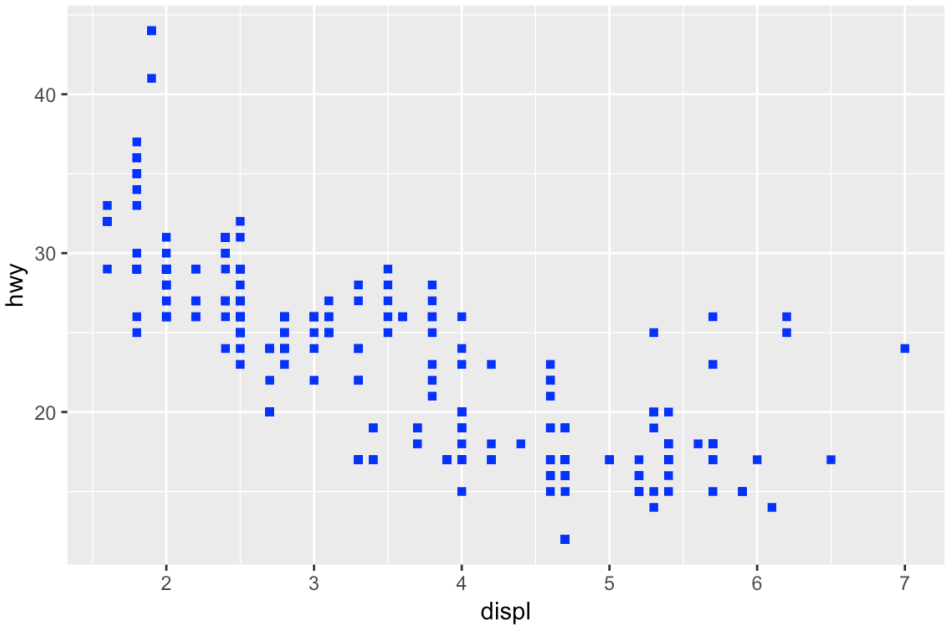


Try mapping the `class` variable using the visual cues `size`, `shape`, or `alpha` (transparency), `fill` (with set `shape = 22`).

```
g <- ggplot(data = mpg, mapping = aes(x = displ, y = hwy))
g + geom_point(mapping = aes(size = class))
g + geom_point(mapping = aes(shape = class))
g + geom_point(mapping = aes(alpha = class))
g + geom_point(mapping = aes(fill = class), shape = 22)
```

Your visual cue is the aesthetic, and must be mapped to graphics by `aes()`. You can also *set* the aesthetic properties of your geom manually. For example, we can make all of the points in our plot blue with square shape:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), color = "blue", shape = 15)
```




Here, the color doesn't convey information about a variable, but only changes the appearance of the plot. To set an aesthetic manually, set the aesthetic by name as an argument of your geom function; i.e. it goes *outside* of `aes()`.

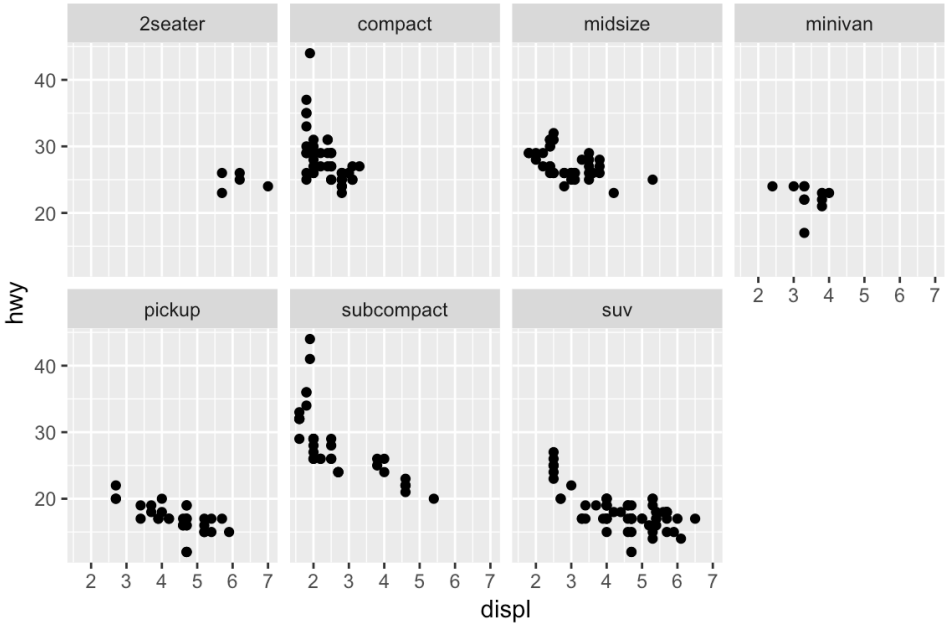
Listing all available colors, shapes, linetypes, etc, is out of scope of this course. Web references include Cookbook for R (http://www.cookbook-r.com/Graphs/Shapes_and_line_types/) and Tian Zheng's "Colors in R" (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf). The ColorBrewer scales are also useful and documented online at http://colorbrewer2.org/ (http://colorbrewer2.org/) and made available in R via the `RColorBrewer` package, by Erich Neuwirth. It is very possible things may change rapidly, I generally recommend googling "R shape codes", "R color codes", etc, for reference.

# 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.

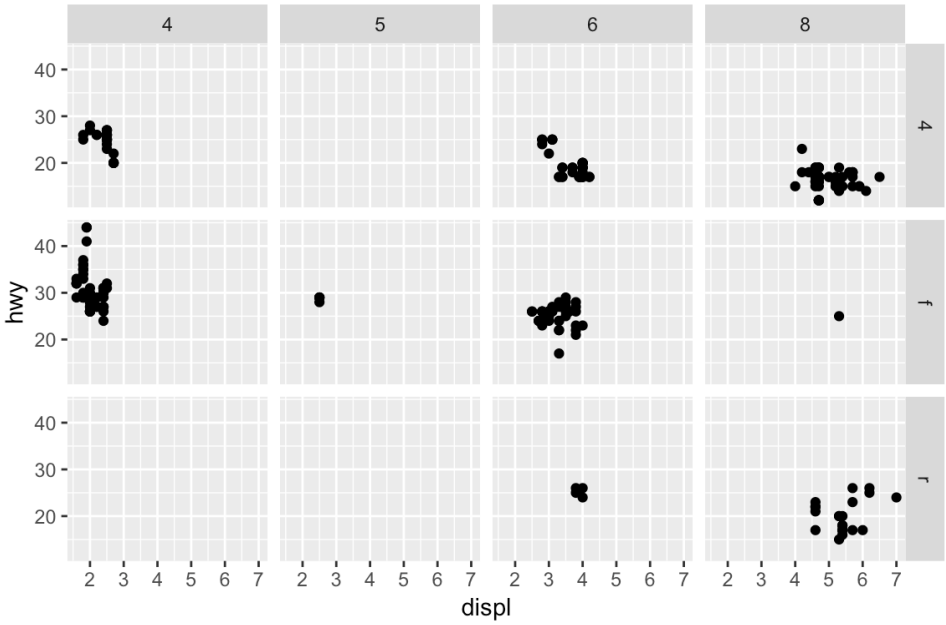
To facet your plot by a single variable, use `facet_wrap()`. The first argument of `facet_wrap()` should be a formula, which you create with `~` followed by a variable name (here "formula" is the name of a data structure in R, not a synonym for "equation"). The variable that you pass to `facet_wrap()` should be discrete.

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




To facet your plot on the combination of two variables, add `facet_grid()` to your plot call. The first argument of `facet_grid()` is also a formula. This time the formula should contain two variable names separated by a `~`.

```
g + facet_grid(drv ~ cyl)
```

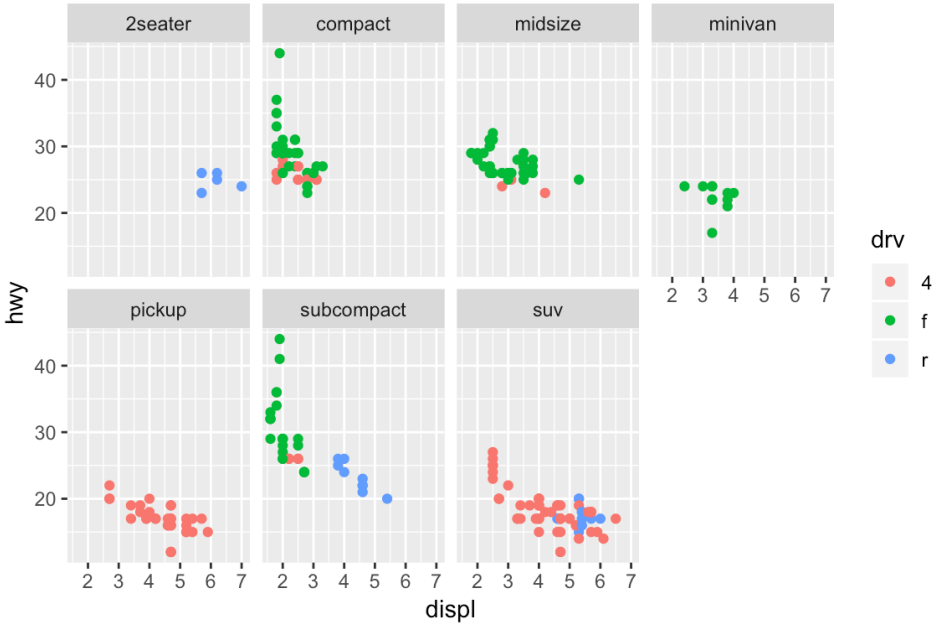



See that I am not retyping `ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy))`. The result was already stored in the R object `g` and you can simply reuse it.

Finally, use facet_grid() to facet into columns (or rows) based on `drv`

```
g + facet_grid(. ~ drv)
g + facet_grid(drv ~ . )
```

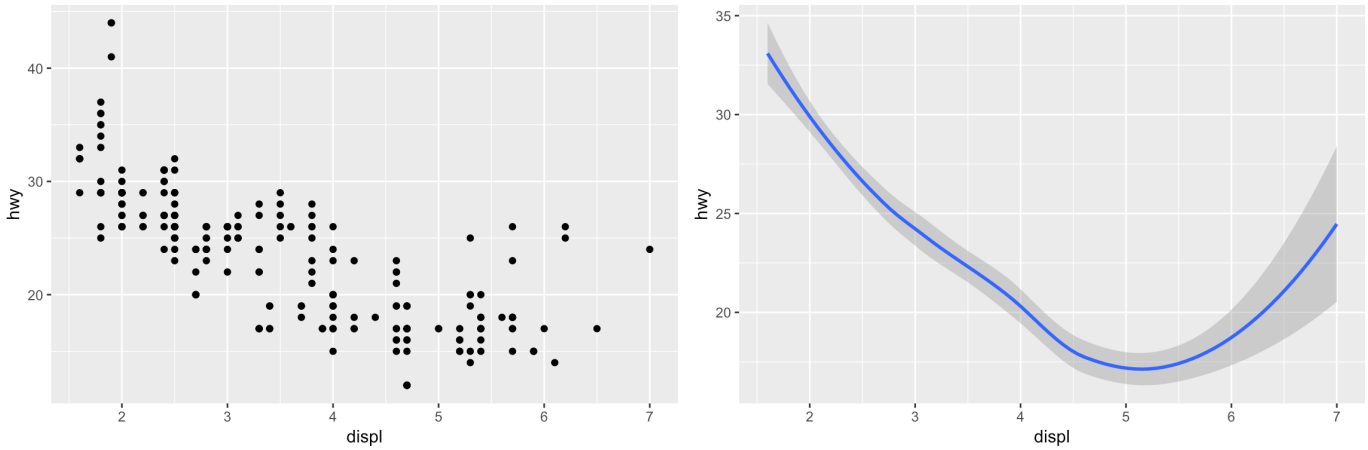
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = drv)) +
  facet_wrap(~ class, nrow = 2)
```




# Geometric objects

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

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

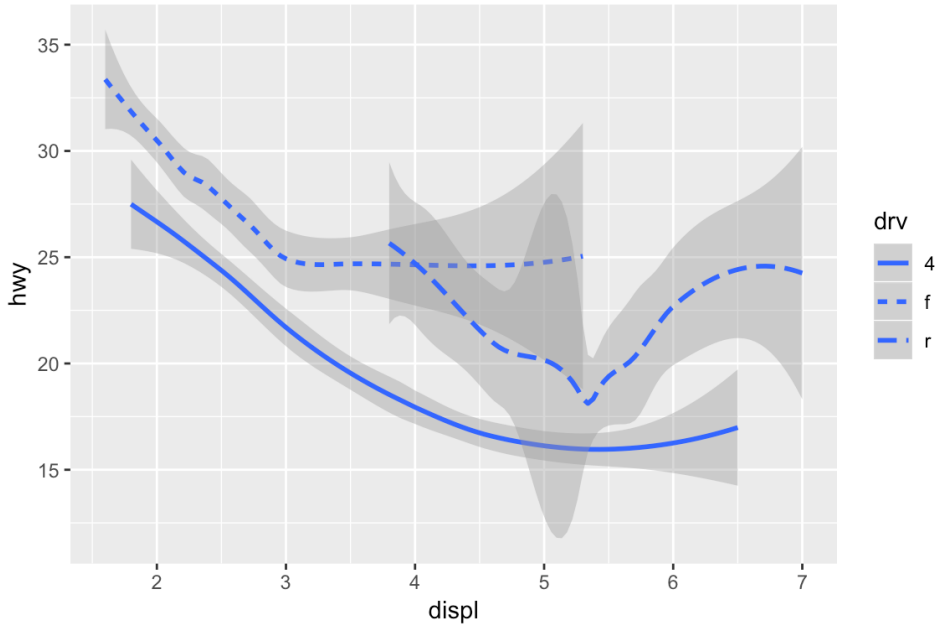
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




A **geom** is the geometrical object that a plot uses to represent data. People often describe plots by the type of geom that the plot uses. For example, bar charts use bar geoms, line charts use line geoms, boxplots use boxplot geoms, and so on. Scatterplots break the trend; they use the point geom. As we see above, you can use different geoms to plot the same 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.

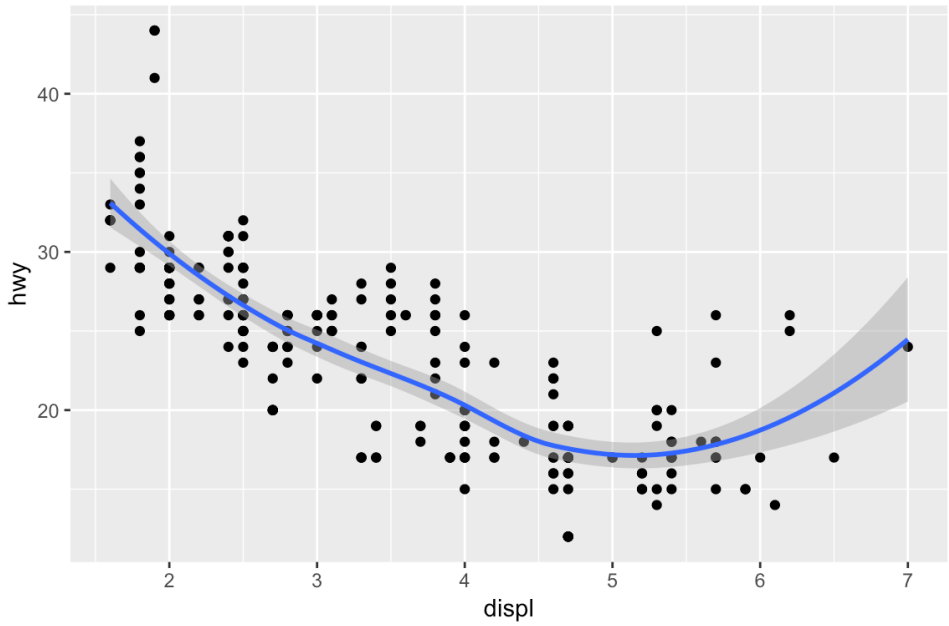
Every geom function in ggplot2 takes a `mapping` argument. However, not every aesthetic works with every geom. You could set the shape of a point, but you couldn't set the "shape" of a line. On the other hand, you could set the linetype of a line. `geom_smooth()` will draw a different line, with a different `linetype`, for each unique value of the variable that you map to linetype.

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



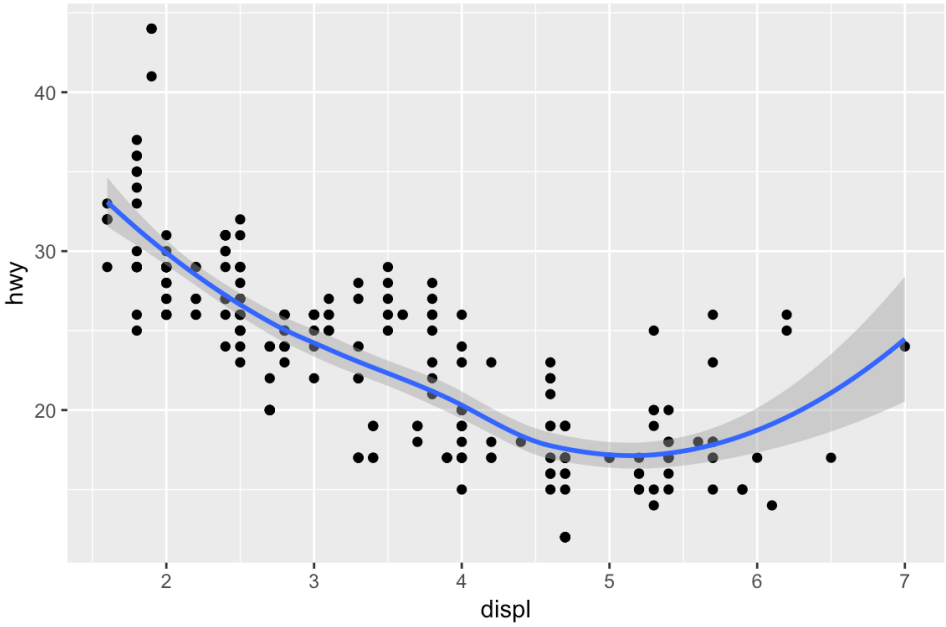

To display multiple geoms in the same plot, add multiple geom functions to `ggplot()`:

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



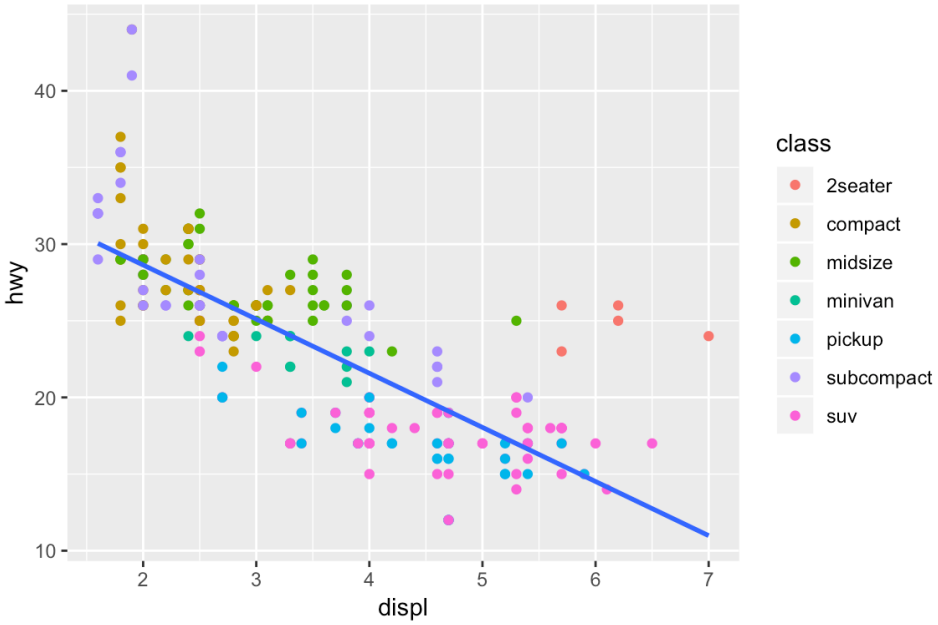

This, however, introduces some duplication in our code. Imagine if you wanted to change the y-axis to display `cty` instead of `hwy`. You'd need to change the variable in two places, and you might forget to update one. You can avoid this type of repetition by passing a set of mappings to `ggplot()`. ggplot2 will treat these mappings as global mappings that apply to each geom in the graph. In other words, this code will produce the same plot as the previous code:

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




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(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point(mapping = aes(color = class)) +
  geom_smooth(method = "lm", se = FALSE) # We've been using method = "loess"
```




Think about this:

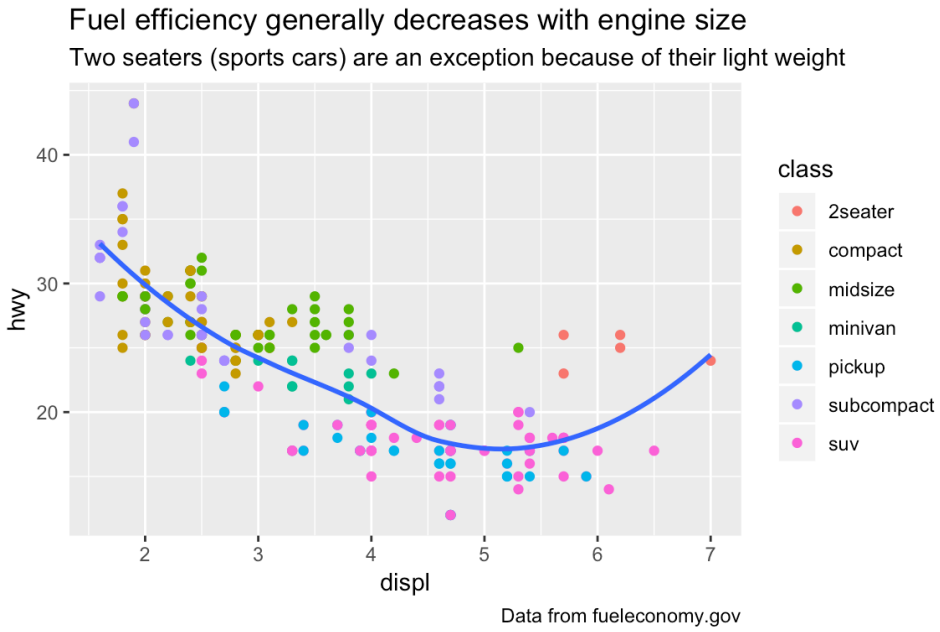
1. What geom would you use to draw a line chart? A boxplot? A histogram? An area chart?
2. What aesthetics can you use to each geom?

To get answers, `Help > Cheatsheets > Data Visualization with ggplot2`

# Adding context by labels

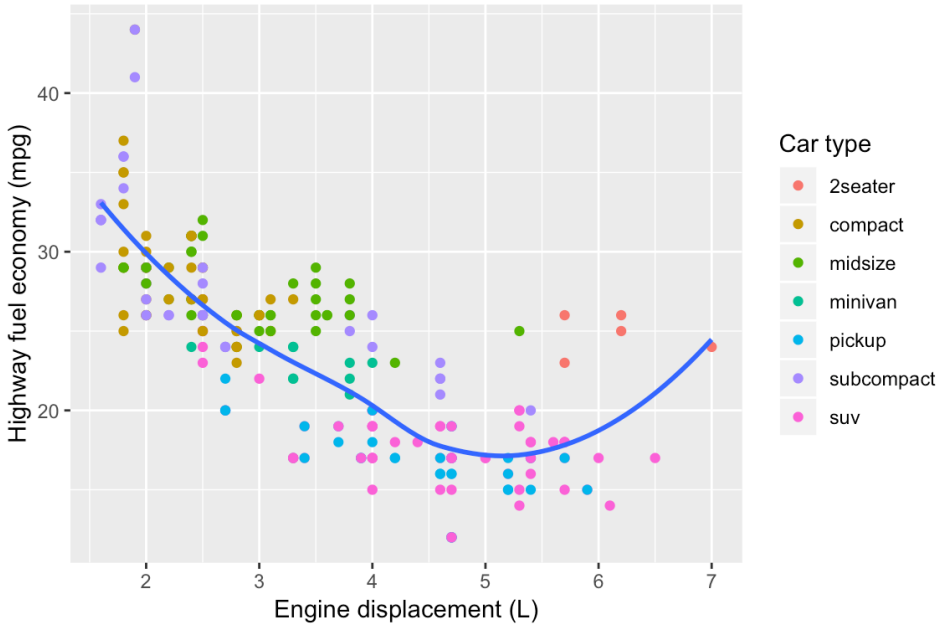
The easiest place to start when turning an exploratory graphic into an expository graphic is with good labels. You add labels with the labs() function. This example adds a plot title:

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    title = "Fuel efficiency generally decreases with engine size",
    subtitle = "Two seaters (sports cars) are an exception because of their light weight",
    caption = "Data from fueleconomy.gov"
  )
```




You can also use `labs()` to replace the axis and legend titles. It's usually a good idea to replace short variable names with more detailed descriptions, and to include the units.

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = class)) +
  geom_smooth(se = FALSE) +
  labs(
    x = "Engine displacement (L)",
    y = "Highway fuel economy (mpg)",
    colour = "Car type"
  )
```

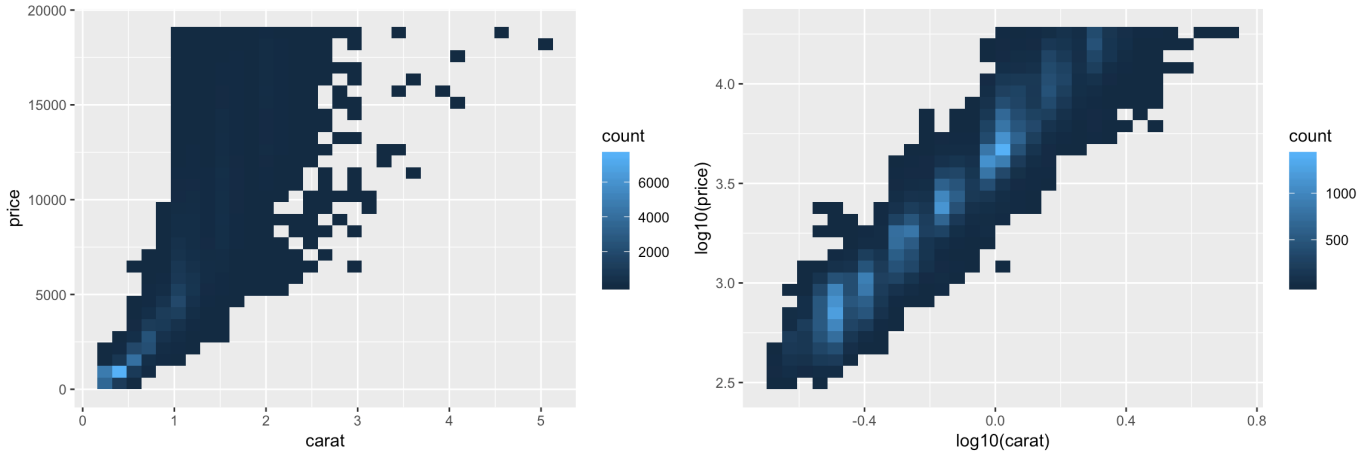



Context is also provided by guides (more commonly called legends). By mapping a discrete variable to one of the visual cues of shape, color or linetype, ggplot2 by default creates a legend. The `geom_text()` and `geom_annotate()` functions can also be used to provide specific textual annotations on the plot. We will see this in the lab activity.

# Scales

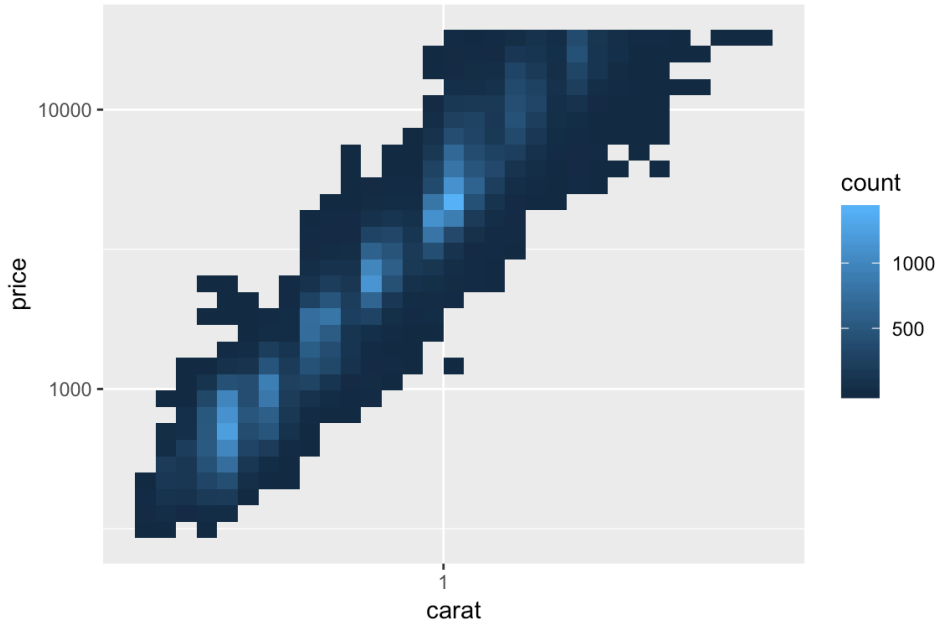
It's very useful to plot transformations of your variable. To elucidate this idea, let's use `diamond` dataset which comes in ggplot2 and contains information about ~54,000 diamonds, including the `price`, `carat`, `color`, `clarity`, and `cut` of each diamond. It's easier to see the precise relationship between carat and price if we log transform them:

```
ggplot(diamonds, aes(carat, price)) +
  geom_bin2d()
ggplot(diamonds, aes(log10(carat), log10(price))) +
  geom_bin2d()
```




However, the disadvantage of this transformation is that the axes are now labelled with the transformed values, making it hard to interpret the plot. Instead of doing the transformation in the aesthetic mapping, we can instead do it with the scale. This is visually identical, except the axes are labelled on the original data scale.

```
ggplot(diamonds, aes(carat, price)) +
  geom_bin2d() +
  scale_x_log10() +
  scale_y_log10()
```

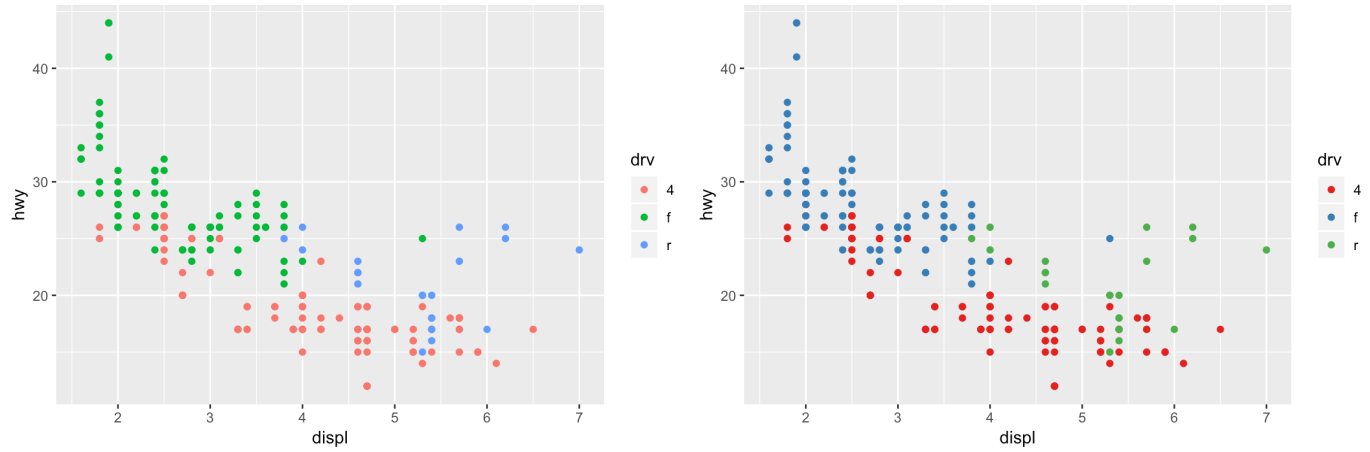



Here's an identical graph using `scale_y_continuous()` function:

```
ggplot(diamonds, aes(carat, price)) +
  geom_bin2d() +
  scale_x_continuous(trans = "log10") +
  scale_y_continuous(trans = "log10")
```

Another scale that is frequently customised is colour. Below, `"Set1"` is defined in `RColorBrewer` package; see Figure 2.11 in MDSR (textbook).

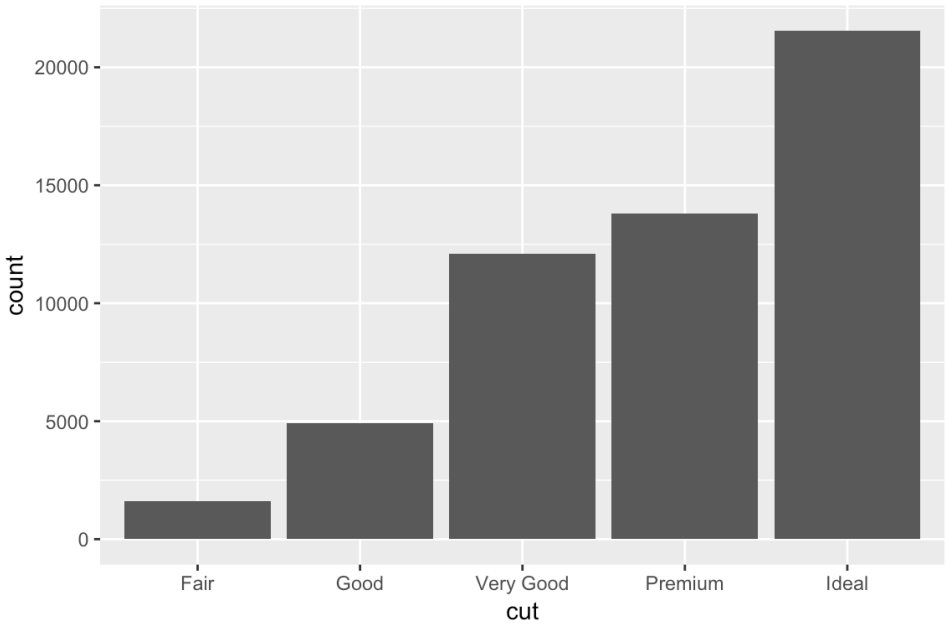
```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv))

ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv)) +
  scale_colour_brewer(palette = "Set1")
```




# Statistical transformations

Next, let's take a look at a bar chart. Bar charts seem simple, but they are interesting because they reveal something subtle about plots. Consider a basic bar chart, as drawn with `geom_bar()`

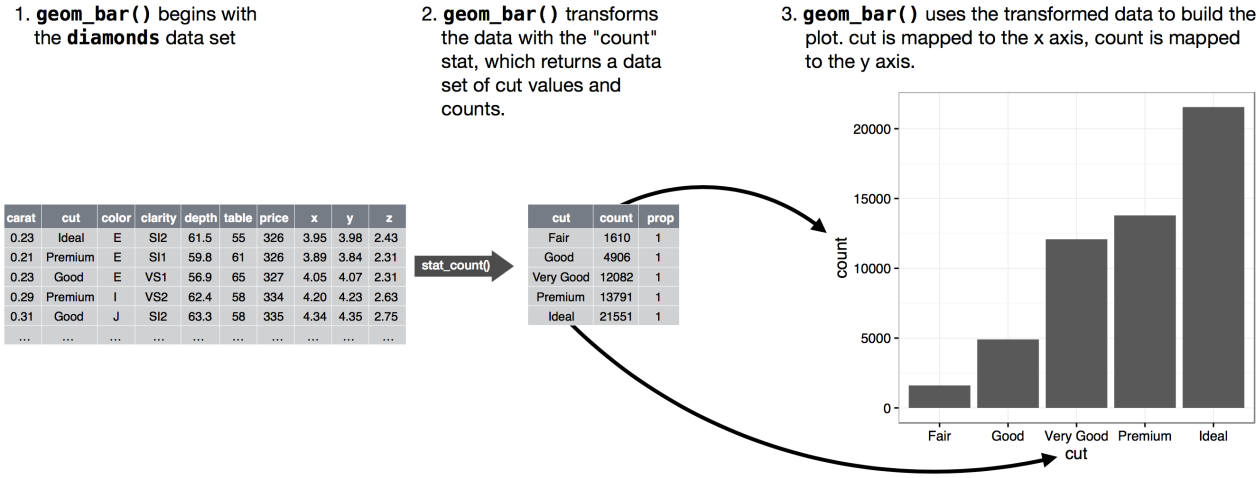
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut))
```




On the x-axis, the chart displays `cut`, a variable from `diamonds`. On the y-axis, it displays count, but count is not a variable in `diamonds`! Where does count come from? Many graphs, like scatterplots, plot the raw values of your dataset. Other graphs, like bar charts, calculate new values to plot:

- bar charts, histograms, and frequency polygons bin your data and then plot bin counts (`geom_bar()` and `geom_bin2d()`)
- smoothers fit a model to your data and then plot predictions from the model (`geom_smooth()`)

The algorithm used to calculate new values for a graph is called a `stat`, short for statistical transformation. The figure below describes how this process works with `geom_bar()`.



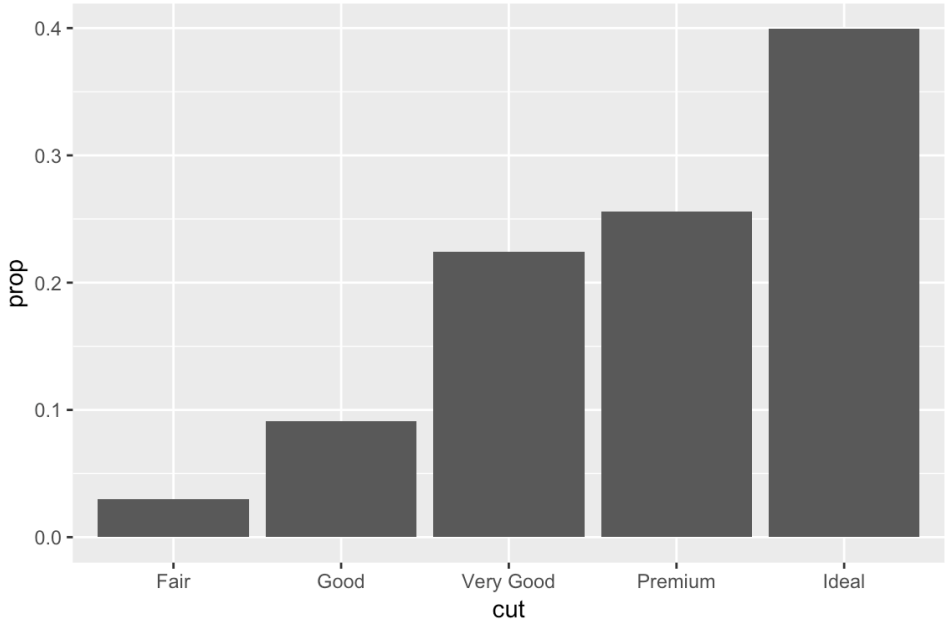

You can learn which stat a geom uses by inspecting the default value for the stat argument. For example, `?geom_bar` shows that the default value for stat is `count`, which means that `geom_bar()` uses `stat_count()`. `stat_count()` is documented on the same page as `geom_bar()`, and if you scroll down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

You can generally use geoms and stats interchangeably. For example, you can recreate the previous plot using `stat_count()` instead of `geom_bar()`:

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

This works because every geom has a default stat; and every stat has a default geom. You might want to override the default mapping from transformed variables to aesthetics.

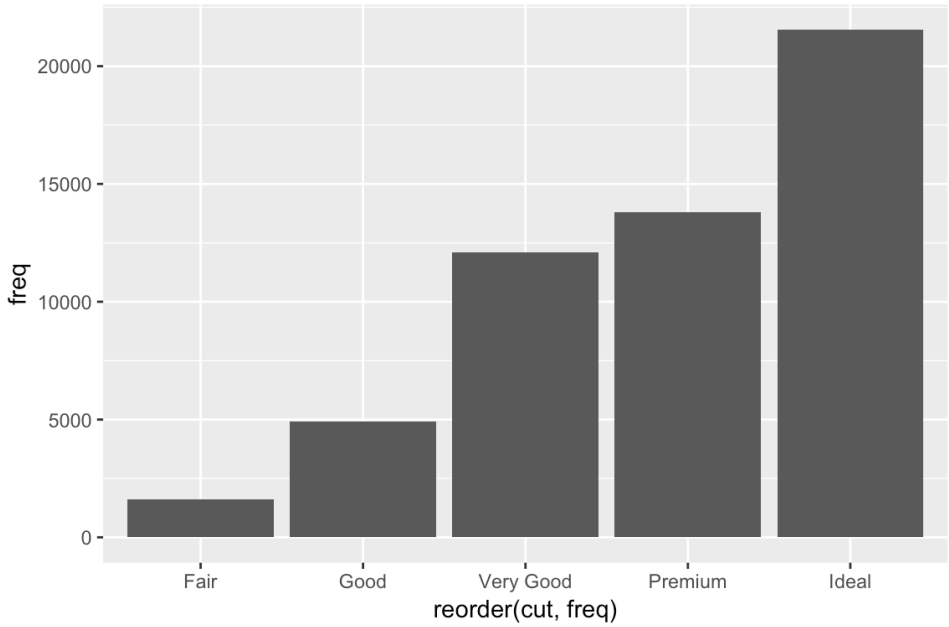
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y = ..prop.., group = 1))
```




`group="whatever"` is a “dummy” grouping to override the default behavior, which is to group by the x variable `cut` (in this example). The default for geom_bar is to group by the x variable in order to separately count the number of rows in each level of the x variable. To compute the proportion of each level of `cut` among all, we do not want to group by `cut`. Sepecifying a dummy group `group = 1`, i.e. all are in `group 1`, achieves this.

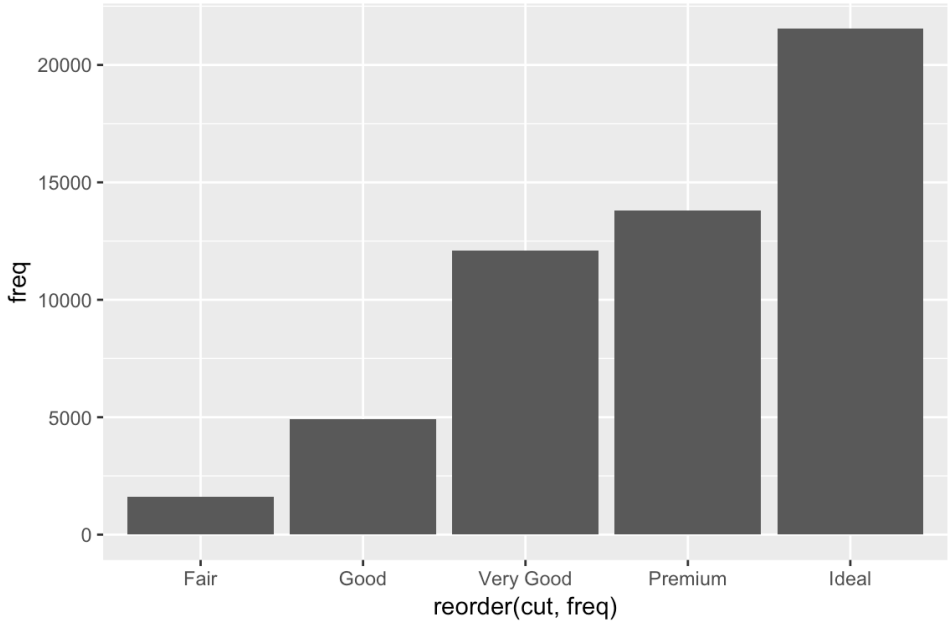
When there is no need for any statistical transformation, I can change the `stat` of `geom_bar()` from count (the default) to identity, as shown in the example below.

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

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



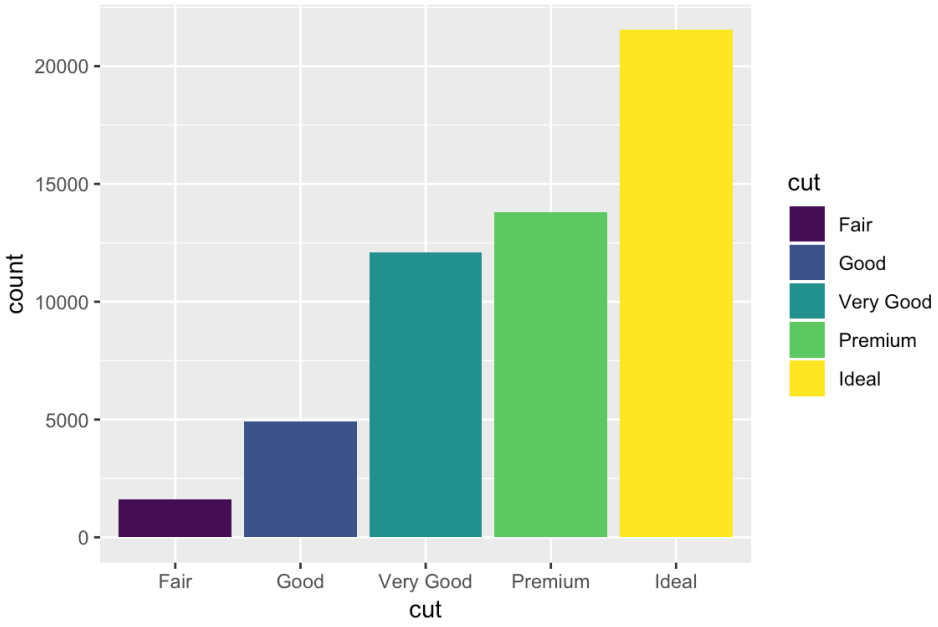

```
# see what happens if `reorder(cut,freq)` is replaced by `cut`. Type `head(diamonds$cut)`
```

```
ggplot(data = demo) +
  geom_col(mapping = aes(x = reorder(cut,freq), y = freq))
```



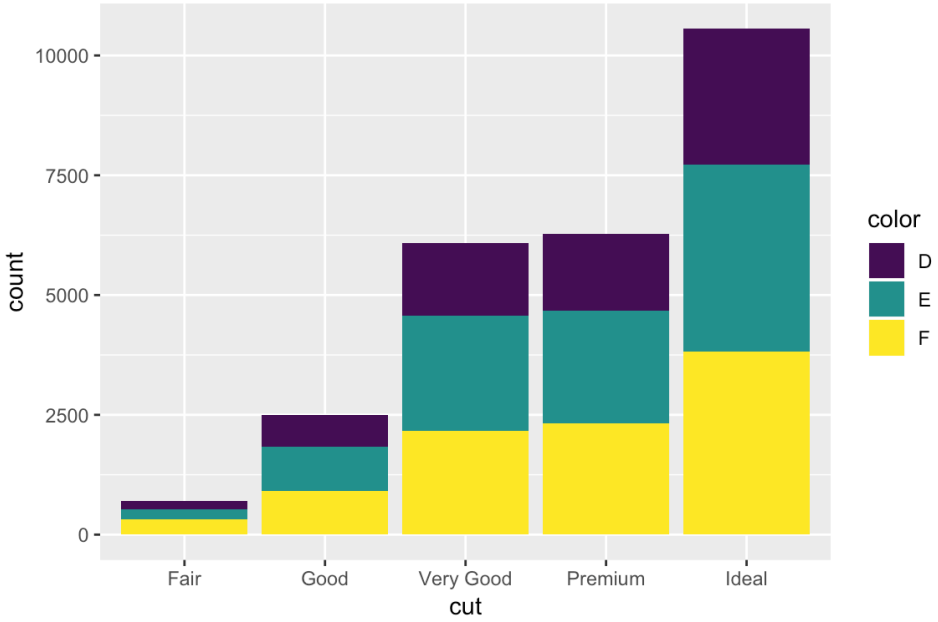

Let us browse some other aesthetic options in `geom_bar()`. You can colour a bar chart using either the `color` aesthetic, or, more usefully, `fill`:

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



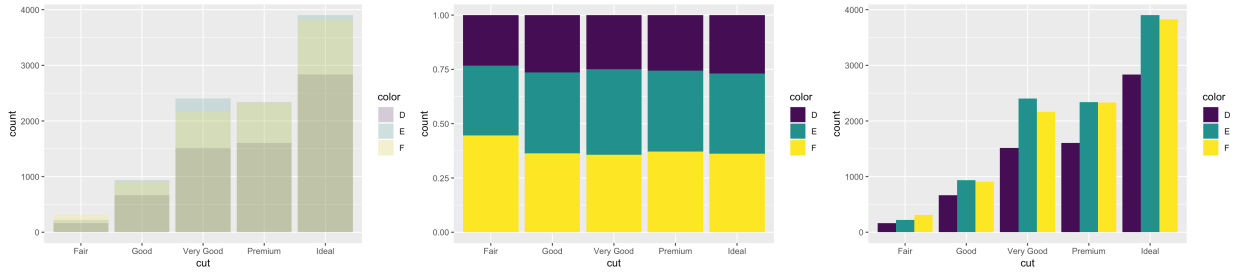

It is of course more useful when `fill` aesthetic is mapped to another categorical variable, like `clarity`.

```
library(dplyr)
diamondss <- diamonds %>% filter(color %in% c('D','E','F')) # don't worry about this for now
g <- ggplot(data = diamondss, mapping = aes(x = cut, fill = color))
g + geom_bar()
```




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"`.

```
g + geom_bar(alpha = 1/5, position = "identity")
g + geom_bar(position = "fill")
g + geom_bar(position = "dodge")
```




# Saving your plots

```
ggsave("my-plot.pdf")
```