ggplot2

#swRk – October 2020

GGplot2 – Data Visualization

"R has several systems for making graphs, but ggplot2 is one of the most elegant and most versatile. ggplot2 implements the **grammar of graphics**, a coherent system for describing and building graphs. With ggplot2, you can do more faster by learning one system and applying it in many places.

If you'd like to learn more about the theoretical underpinnings of ggplot2 before you start, I'd recommend reading "The Layered Grammar of Graphics", http://vita.had.co.nz/papers/layered-grammar.pdf."

GGplot2 – Data Visualization

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

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

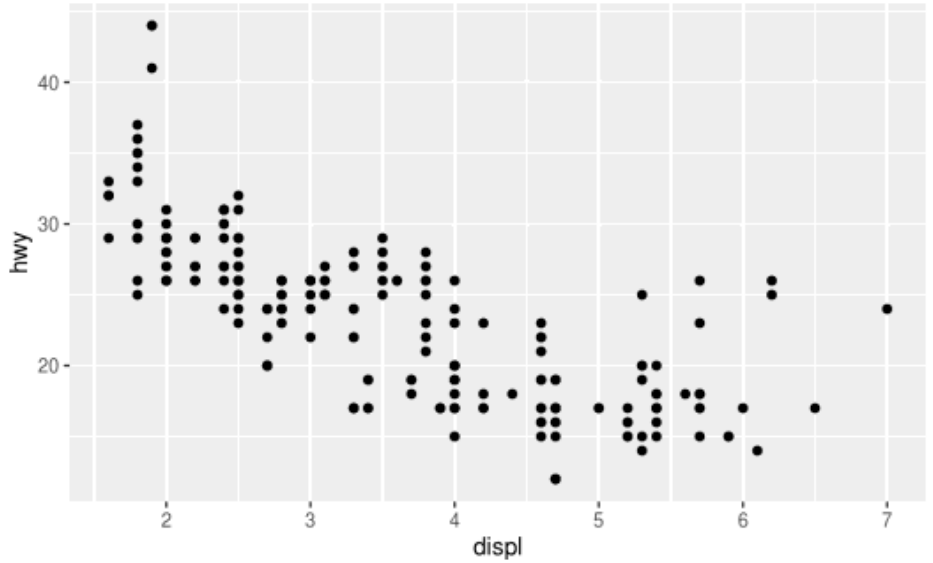
GGplot2 – Data Visualization

Scatter plot of **mpg**:

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

OR:

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



GGplot2 – Data Visualization

Every ggplot2 plot has three components:

1. Data

2. A set of aesthetic mappings between variables in the data and visual properties

3. At least one layer describing how to render each observation. Layers are usually created with a **geom** function.

GGplot2 – Data Visualization

- You complete your graph by adding one or more layers to ggplot()
- The function 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. You'll learn a whole bunch of them throughout this chapter.

- 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 variables in the **data** argument, in this case **mpg**.

GGplot2 – Data Visualization

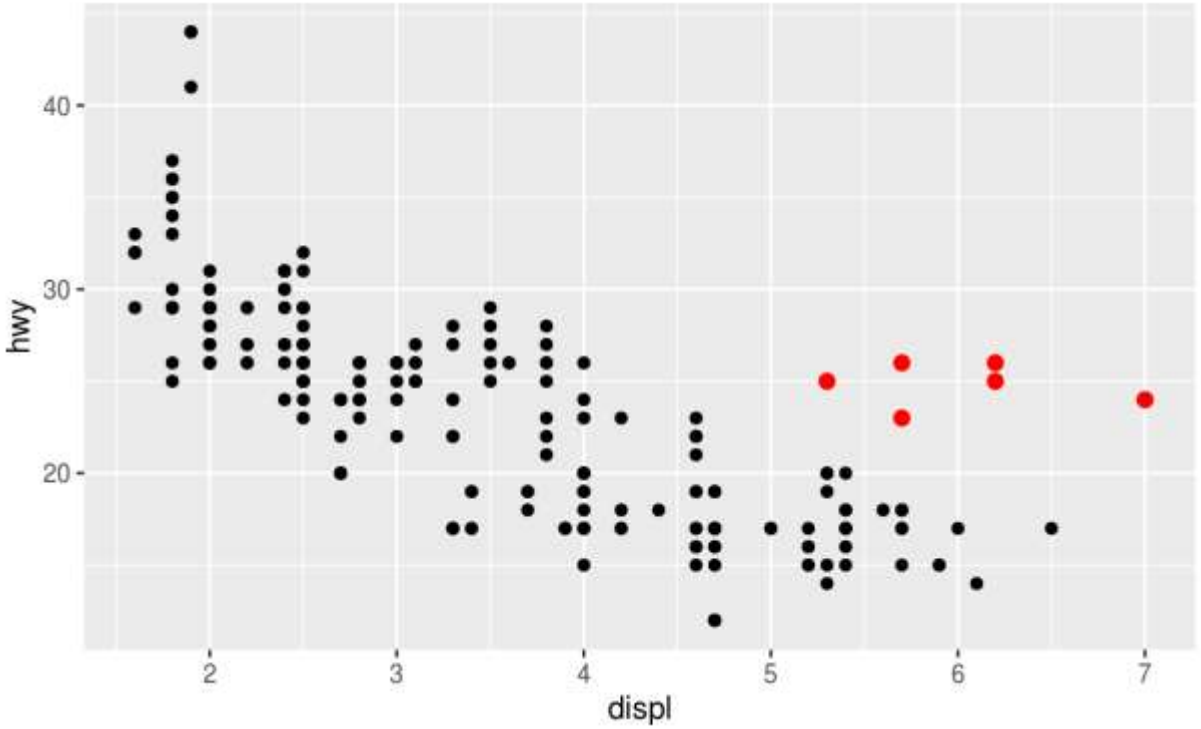
1. Run ggplot(data = mpg). What do you see?
2. How many rows are in mpg? How many columns?
3. What does the **drv** variable describe? Read the ?mpg to find out.
4. Make a scatterplot of **hwy** vs. **cyl**.
5. What happens if you make a scatterplot of **class** vs. **drv**? Why is the plot not useful?

GGplot2 – Data Visualization

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 the **shape**, or the **color** of your points.

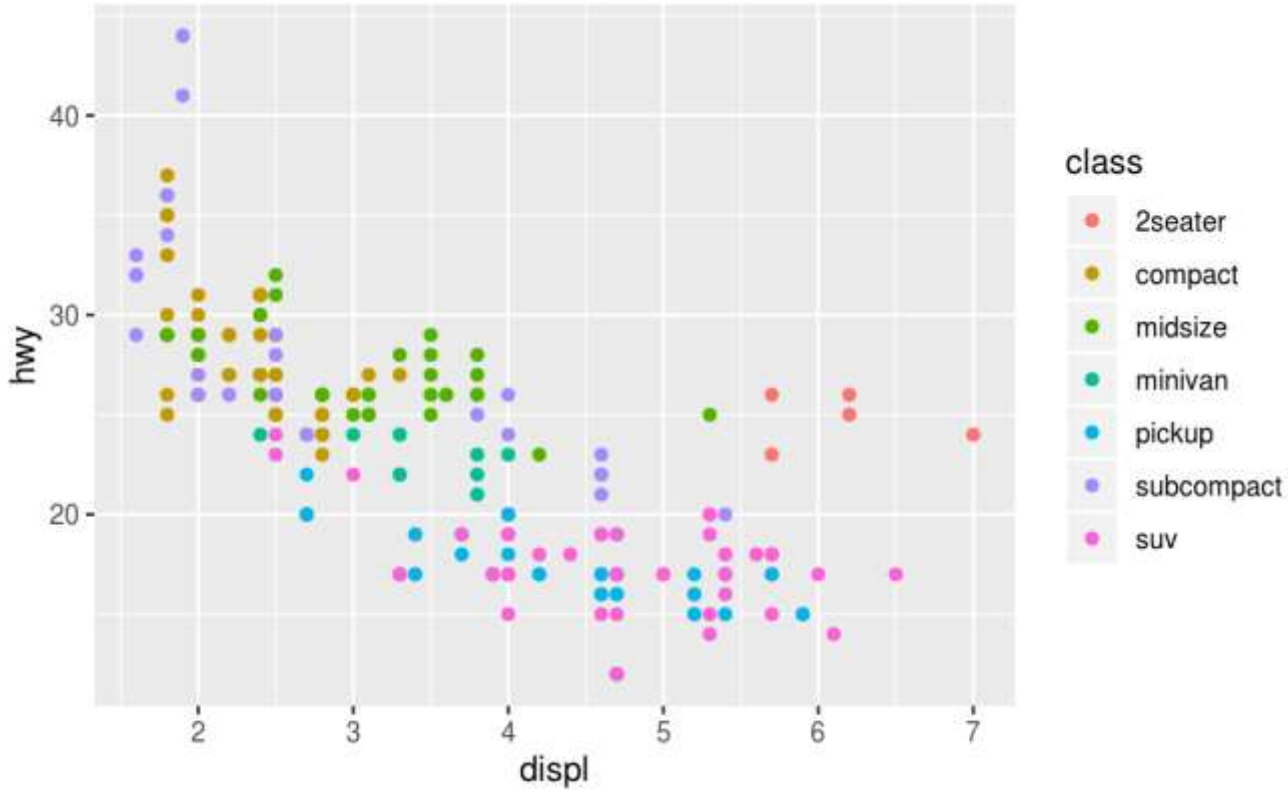
You can display a point (like the one below) in different ways by changing the values of its aesthetic properties. Since we already use the word “value” to describe data, let’s use the word “**level**” to describe aesthetic properties.



GGplot2 – Data Visualization

This is a plot with the colors of the points mapped to the **class** variable.

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

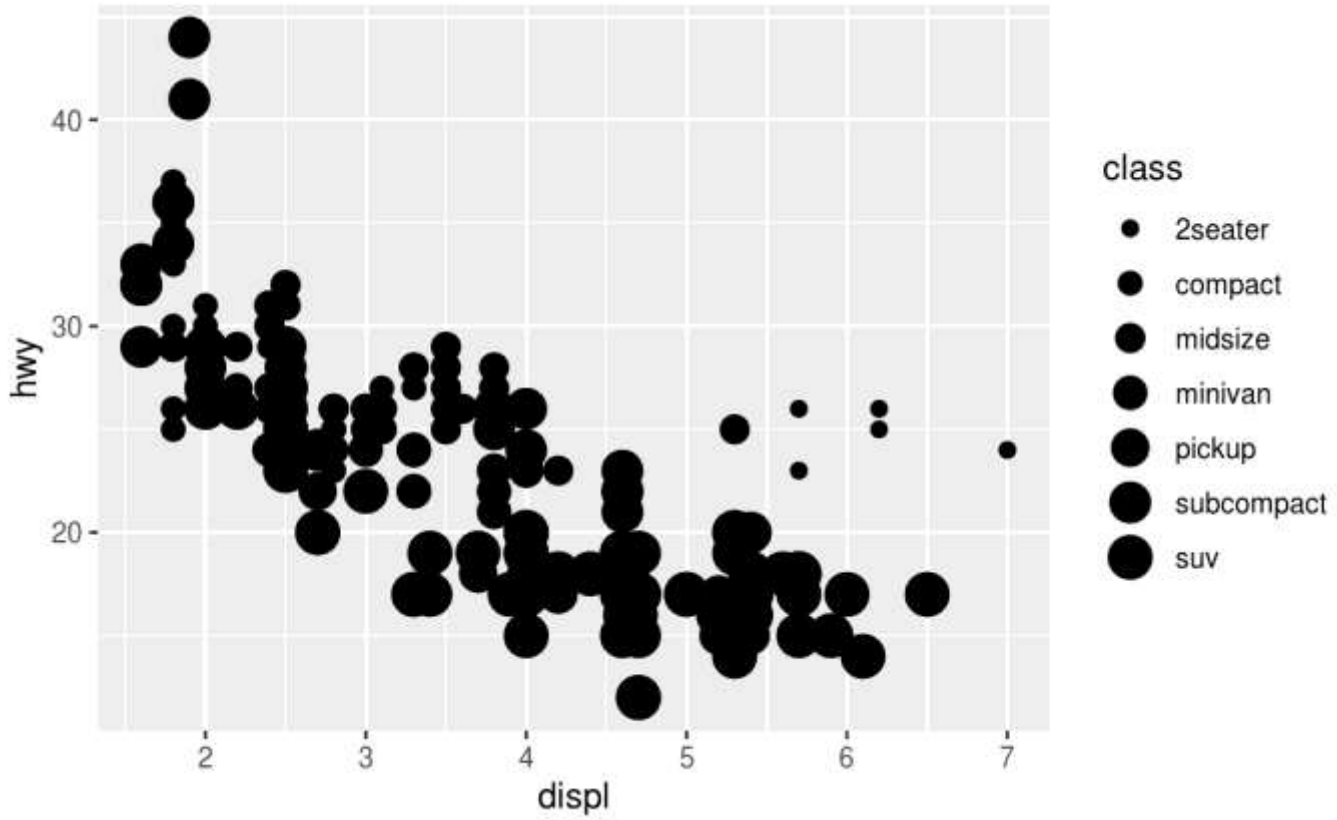


GGplot2 – Data Visualization

ggplot will assign a level to each aesthetic and variable, and plot a legend for you.

This is an example mapping **class** to **size**:

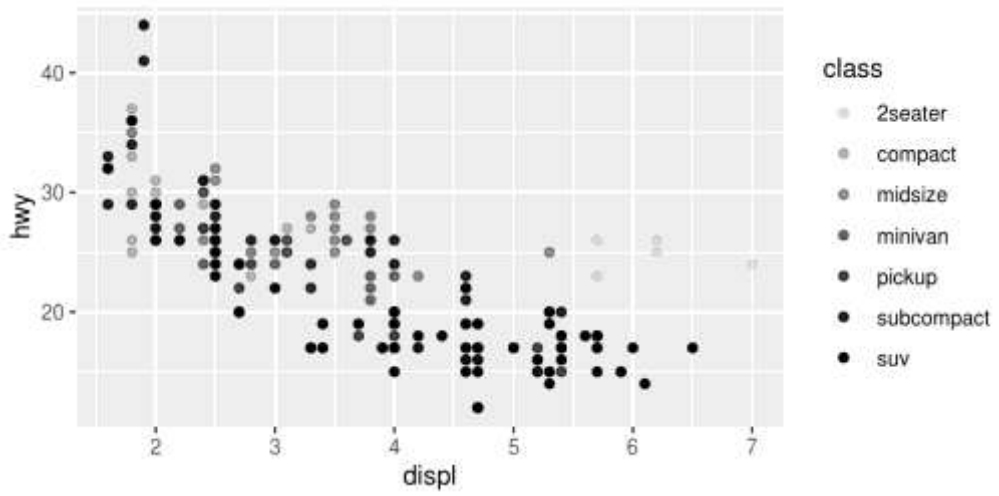
```
ggplot(data = mpg) +
    geom_point(mapping =
    aes(x = displ, y = hwy,
    size = class)
```



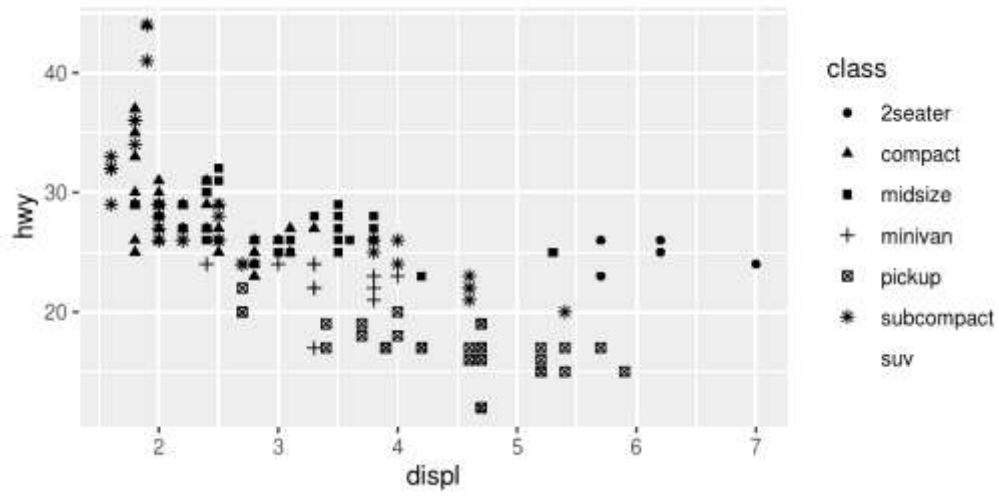
GGplot2 – Data Visualization

This is an example mapping to the alpha aesthetic, which controls transparency; and to shape, which controls the shape of the points.

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




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



GGplot2 – Data Visualization

- Once you map an aesthetic, ggplot2 takes care of the rest. It selects a reasonable scale to use with the aesthetic, and it constructs a legend that explains the mapping between levels and values. For x and y aesthetics, ggplot2 does not create a legend, but it creates an axis line with tick marks and a label. The axis line acts as a legend; it explains the mapping between locations and values.

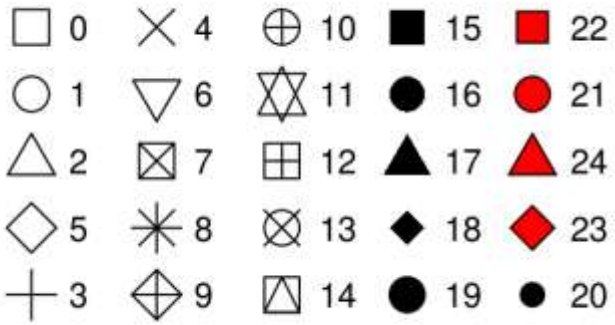
GGplot2 – Data Visualization

We can set the aesthetic properties manually. Let's make all of the points blue:

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

GGplot2 – Data Visualization

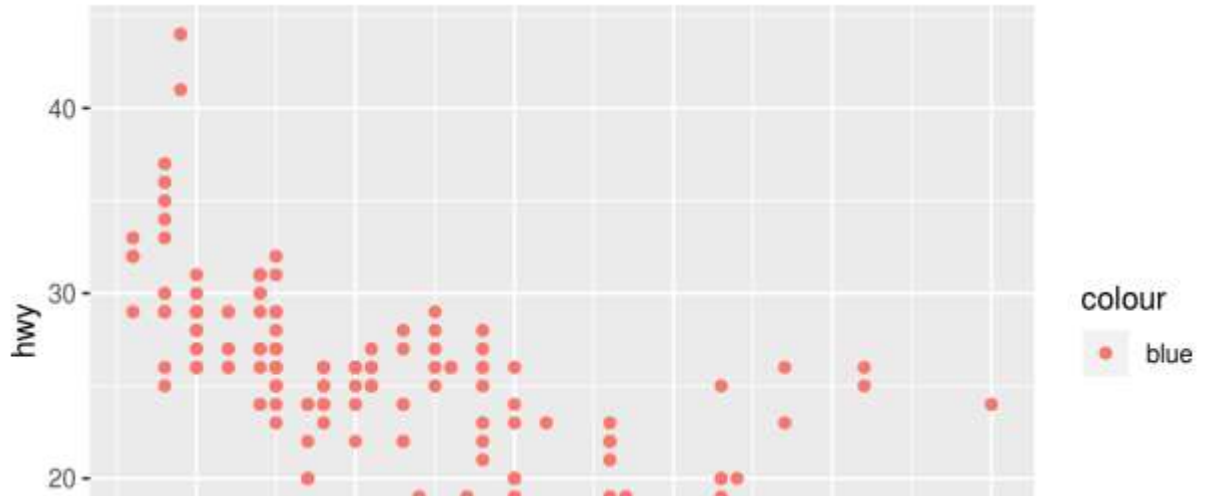
- Here, the color doesn't convey information about a variable, but only changes the appearance of the plot. To set an aesthetic manually, set the aesthetic by name as an argument of your geom function; i.e. it goes *outside* of aes().
- You'll need to pick a level that makes sense for that aesthetic:
- The name of a color as a character string
- The size of a point in mm.
- The shape of a point as a number



GGplot2 – Data Visualization

1. What's wrong with this code? Why are the points not blue?

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



GGplot2 – Data Visualization

2. Which variables are categorical, and which are continuous? (Hint: Type **?mpg** to read the documentation for the dataset?) How can you see this information when you run **mpg**?

3. Map a continuous variable to color, size, and shape? How can you see this information when you aesthetics behave differently for categorical vs. continuous variables?

4. What happens if you map the same variable to multiple aesthetics?

5. What does the stroke aesthetic do? What shapes does it work with? (Hint: use ?geom_point)

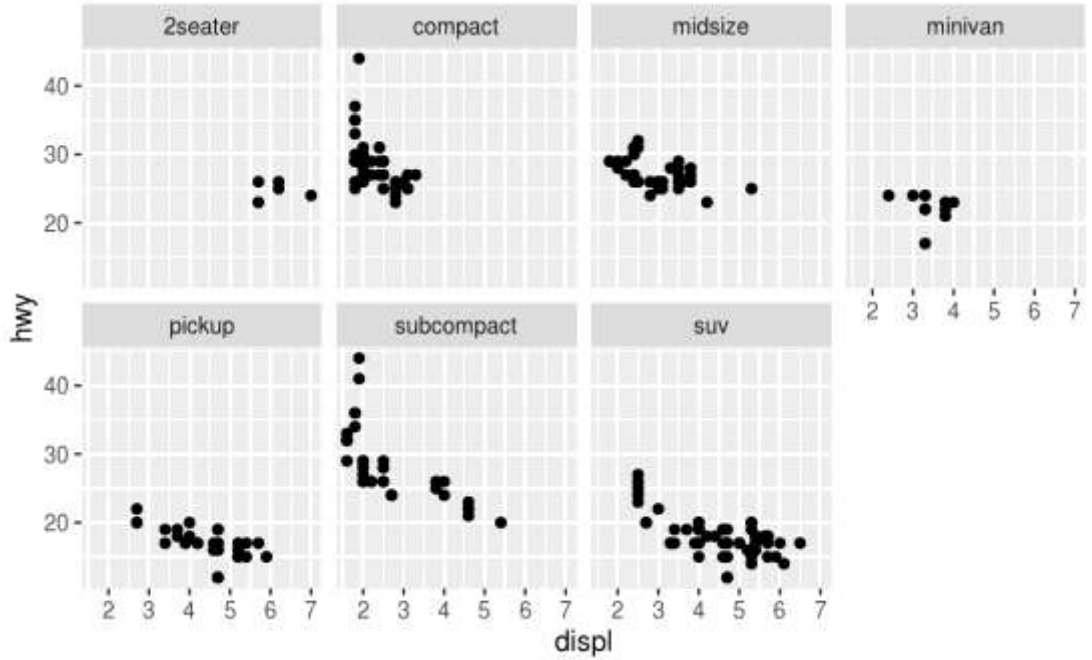
6. What happens if you map an aesthetic to something other than a variable name, like aes(color = displ <5)? Note, you'll also need to specify x and y.

GGplot2 – Data Visualization

Another way to add additional variables is with facets. It's particularly good for categorical variables.

Facets splits your plot into subplots, each of which displays one subset of data.

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



GGplot2 – Data Visualization

Another way to add additional variables is with facets. It's particularly good for categorical variables.

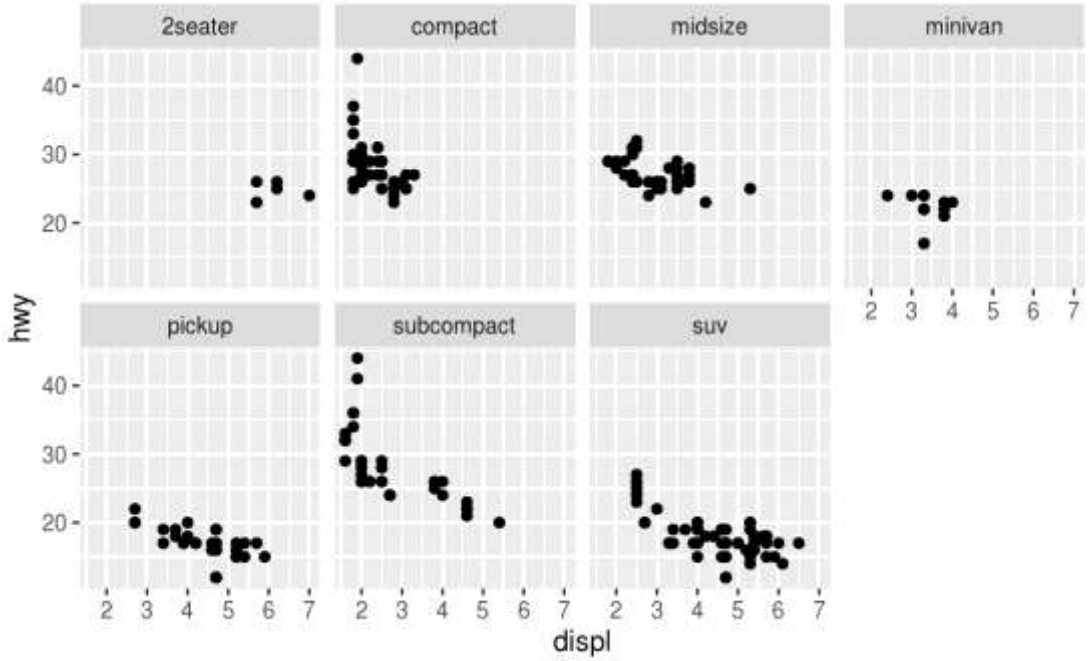
Facets splits your plot into subplots, each of which displays one subset of data.

Two types of faceting: Grid and Wrapped.

Wrapped is more useful if you have a single variable with many levels, and want to arrange the plots in a more space-efficient manner.

You can control how the ribbon is wrapped into a grid with ncol, nrow, as.table, and dir.

nrow and ncol control how many columns and rows. (you only need to set one.)



GGplot2 – Data Visualization

1. What happens if you facet on a continuous variable?

2. What do the empty cells in plot with facet_grid(drv ~ cyl) mean? How do they relate to this plot?

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

3. What plots does the following code make? What does . do?

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

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

GGplot2 – Data Visualization

4. Take the first faceted plot in this section:

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

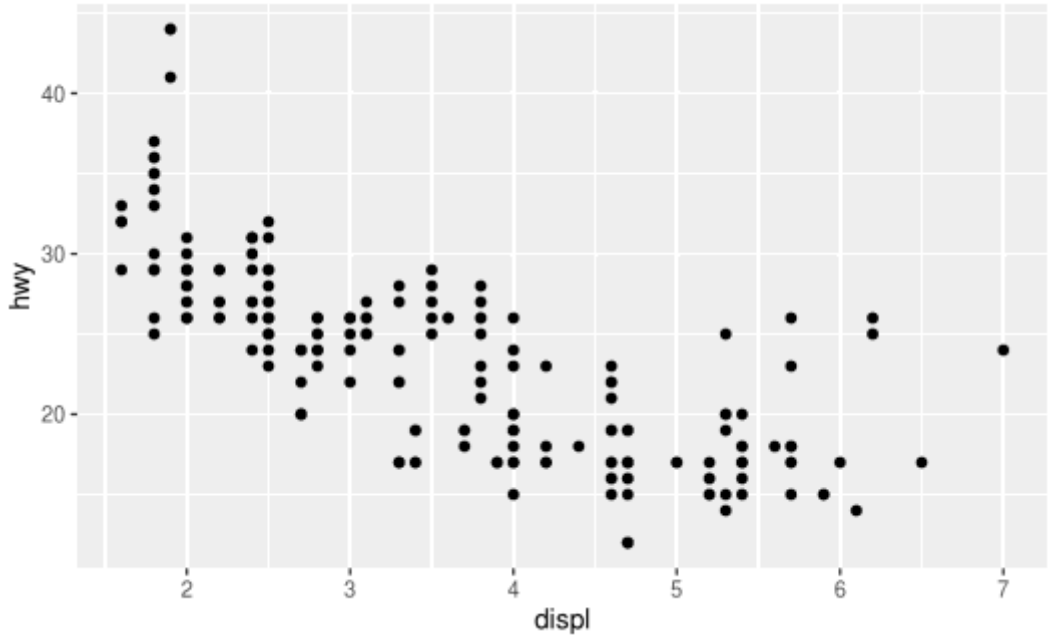
What are the advantages to using faceting instead of the color aesthetic? What are the disadvantages? How might the balance change if you had a larger dataset?

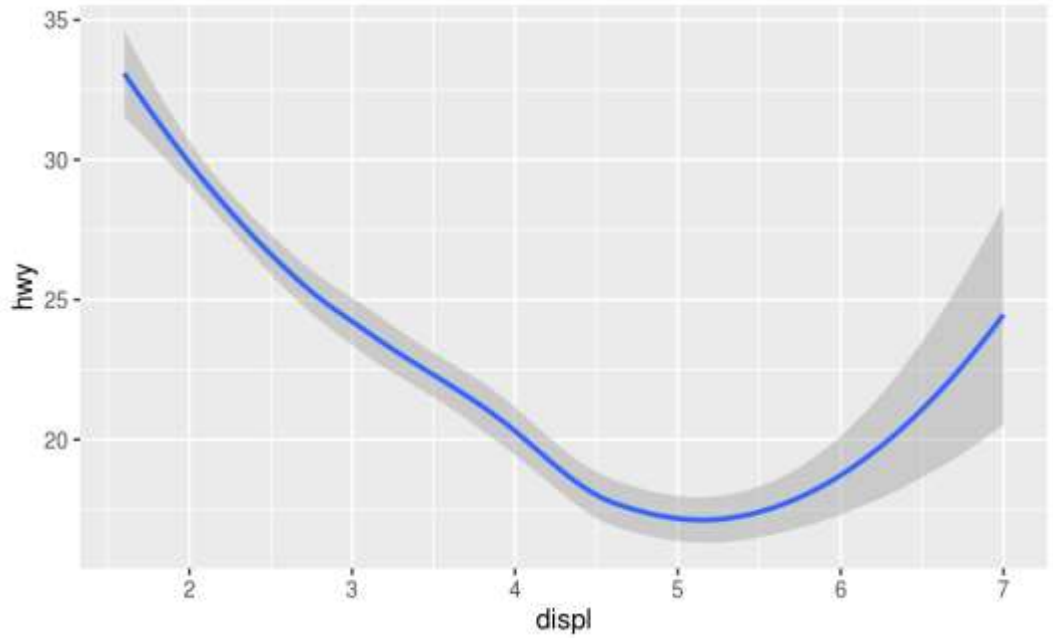
5. Read ?facet_wrap. What does nrow do? What does ncol do? What other options control the layout of the individual panels? Why doesn't facet_grid have nrow and ncol arguments?

6. When using facet_grid() you should usually put the variable with more unique levels in the columns. Why?

GGplot2 – Data Visualization

Geometric Objects:





GGplot2 – Data Visualization

- A **geom** is the geometrical object that a plot uses to represent data.
- We usually 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.

GGplot2 – Data Visualization

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

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

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

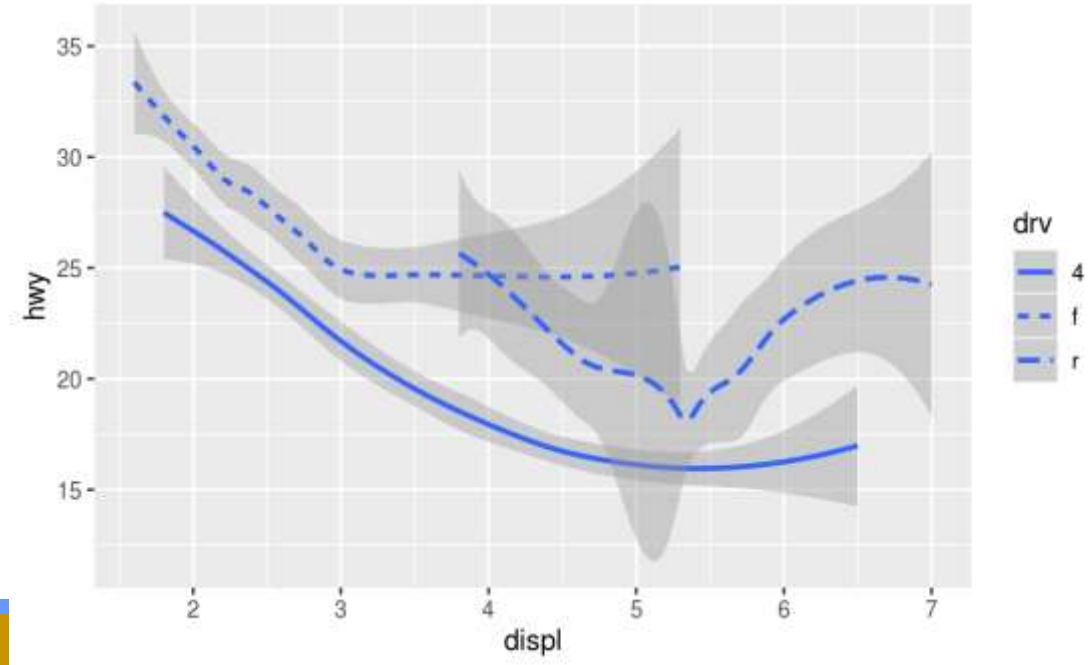
GGplot2 – Data Visualization

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_point(mapping = aes(
    x = displ, y = hwy,
    linetype = drv))
```



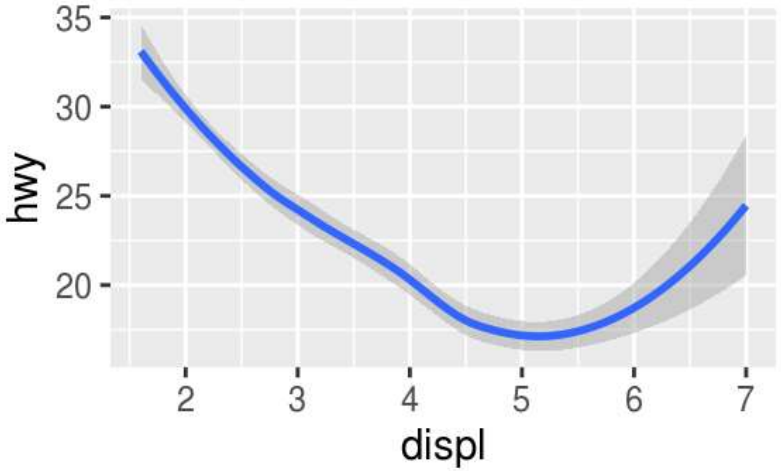
GGplot2 – Data Visualization

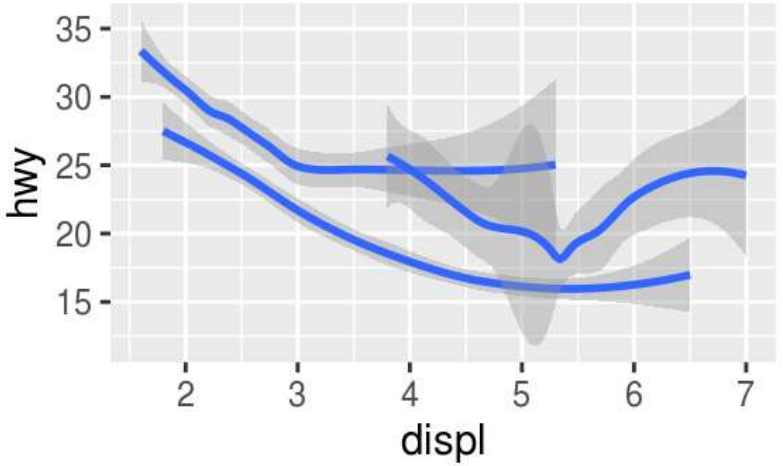
You can use the **group** aesthetic to draw multiple objects.

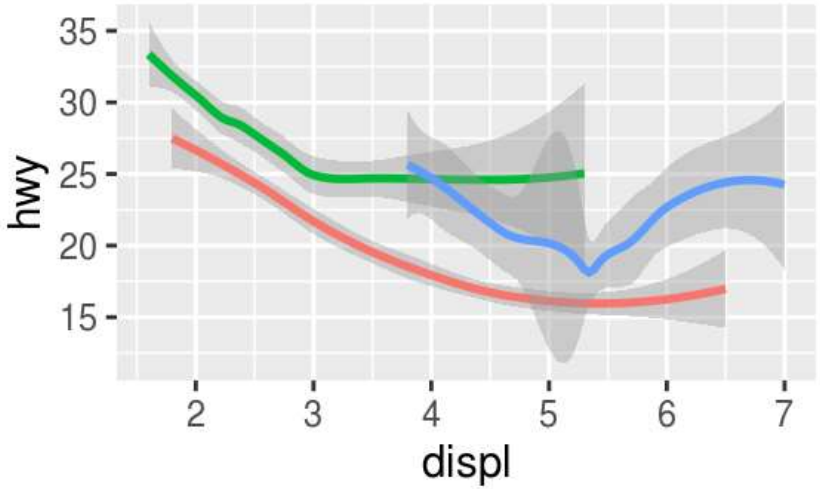
ggplot2 will draw a separate object for each unique value of the grouping variable. In practice, ggplot2 will automatically group the data for these geoms whenever you map an aesthetic to a discrete variable.

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

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

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

GGplot2 – Data Visualization



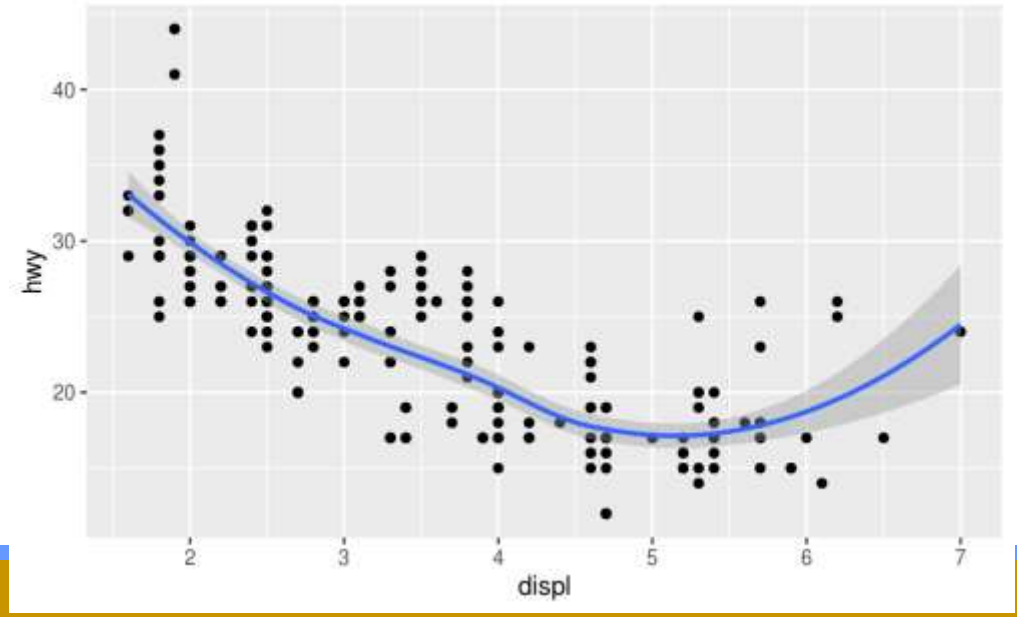




GGplot2 – Data Visualization

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))
```

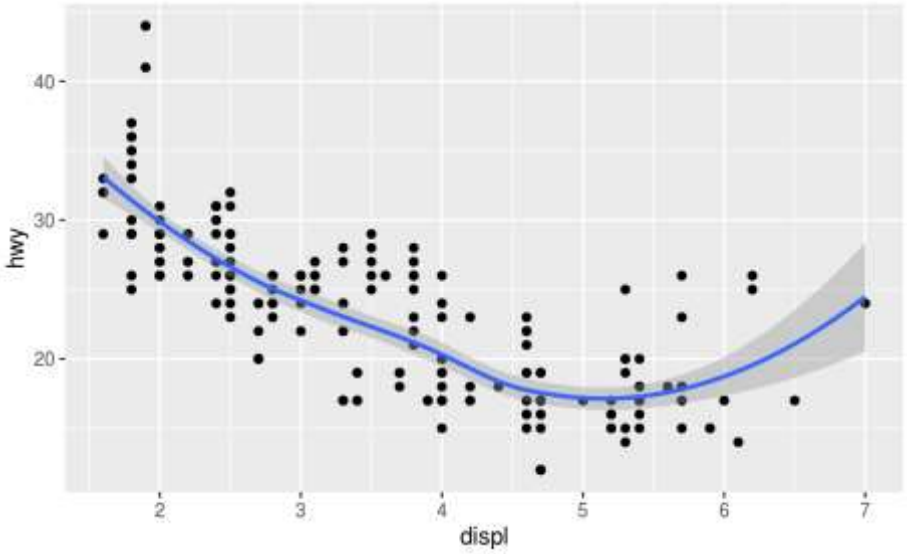


GGplot2 – Data Visualization

You can avoid errors by using a set of global mappings in ggplot()

This will create the same plot:

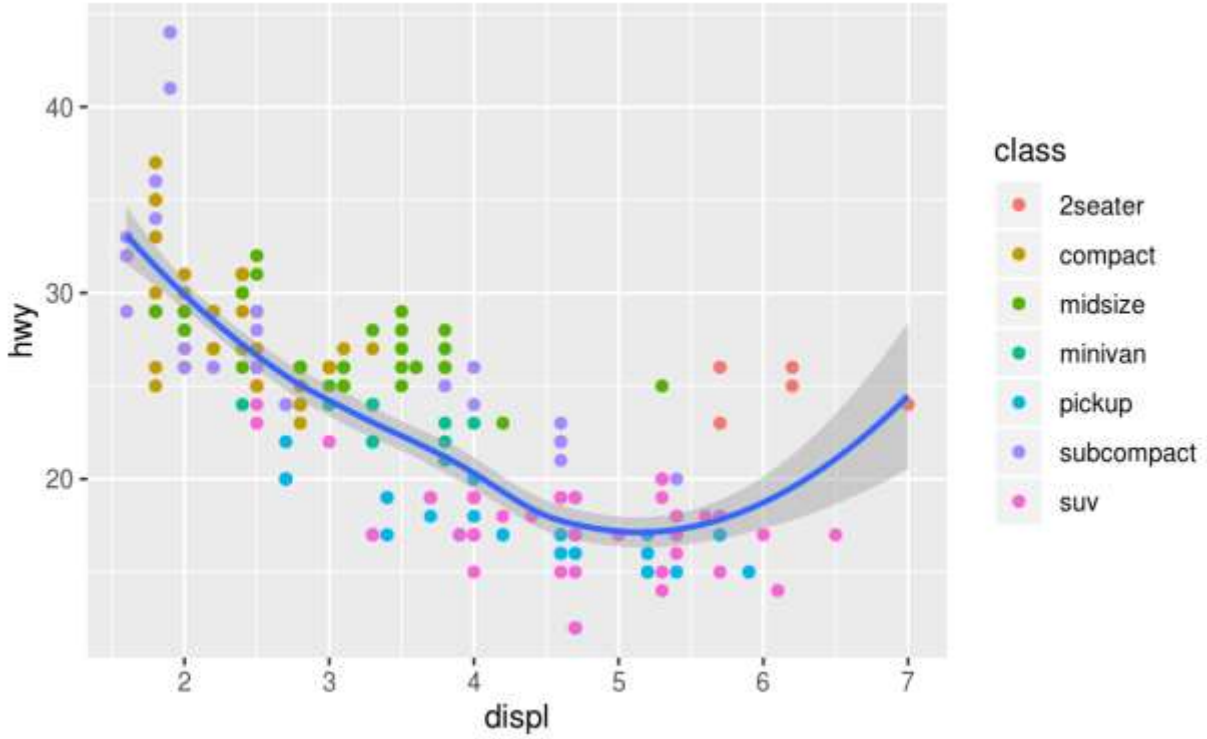
```
ggplot(data = mpg, mapping = aes(x = displ,
    y = hwy)) +
    geom_point() +
    geom_smooth()
```



GGplot2 – Data Visualization

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()
```



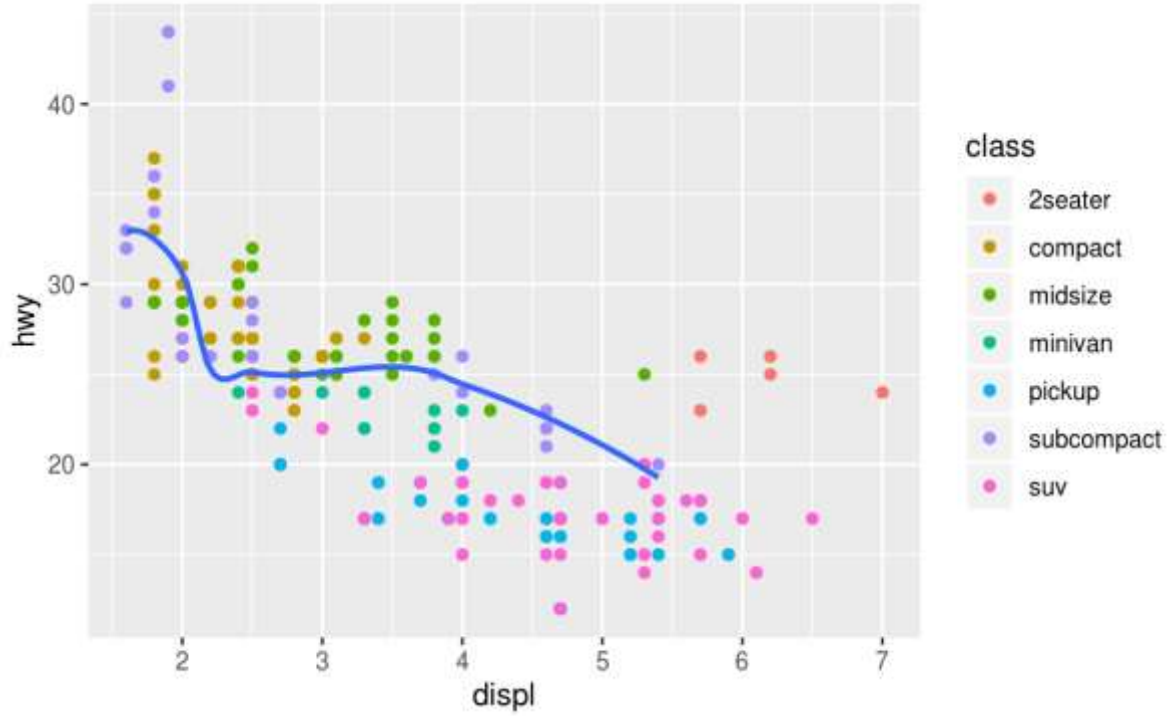
GGplot2 – Data Visualization

You can use the same idea to specify different **data** for each layer.

Here, our smooth line displays just a subset of the mpg dataset, subcompact cars.

he local data argument in geom_smooth() overrides the global data argument in ggplot() for that layer only.

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



GGplot2 – Data Visualization

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

2. Run this code in your head and predict what the output will look like. Then, run the code in R and check your predictions.

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

3. What does show.legend = FALSE do? What happens if you remove it?
Why do you think I used it earlier in the chapter?

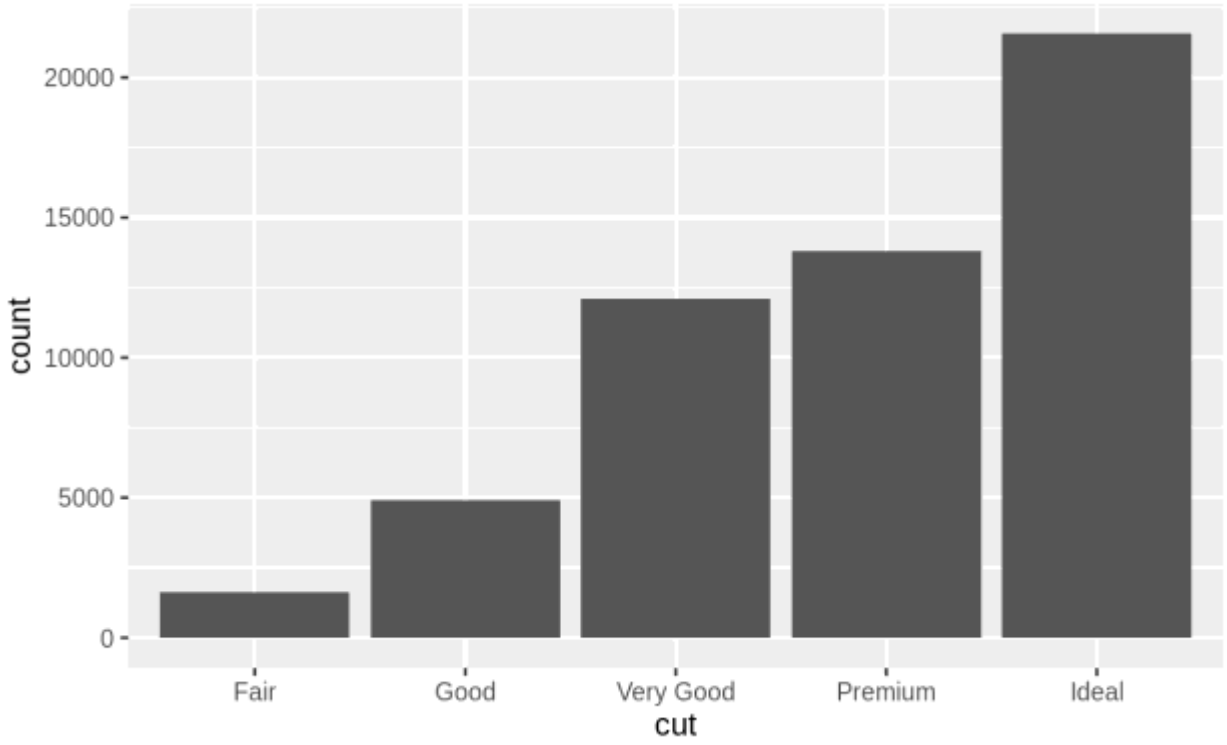
4. What does the se argument to the geom_smooth do?

GGplot2 – Data Visualization

Statistical Transformations (3.7):

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

The chart shows that more diamonds are available with high quality cuts than with low quality cuts.



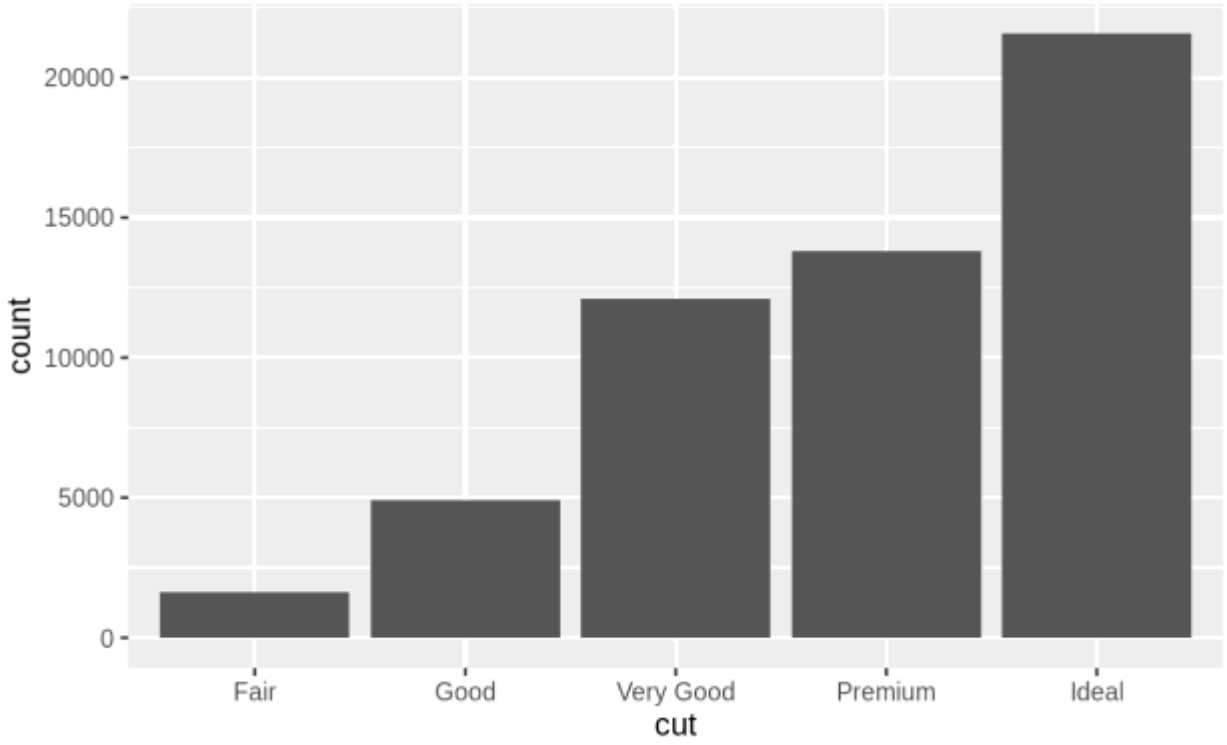
GGplot2 – Data Visualization

Statistical Transformations (3.7):

The dataset comes in ggplot2, and contains

information about ~54,000 diamonds, including the price, carat, color, clarity, and cut of diamond.

The x-axis displays the “cut” variable. The y-axis displays “count” which is not a variable in diamonds. So where does it comes from.

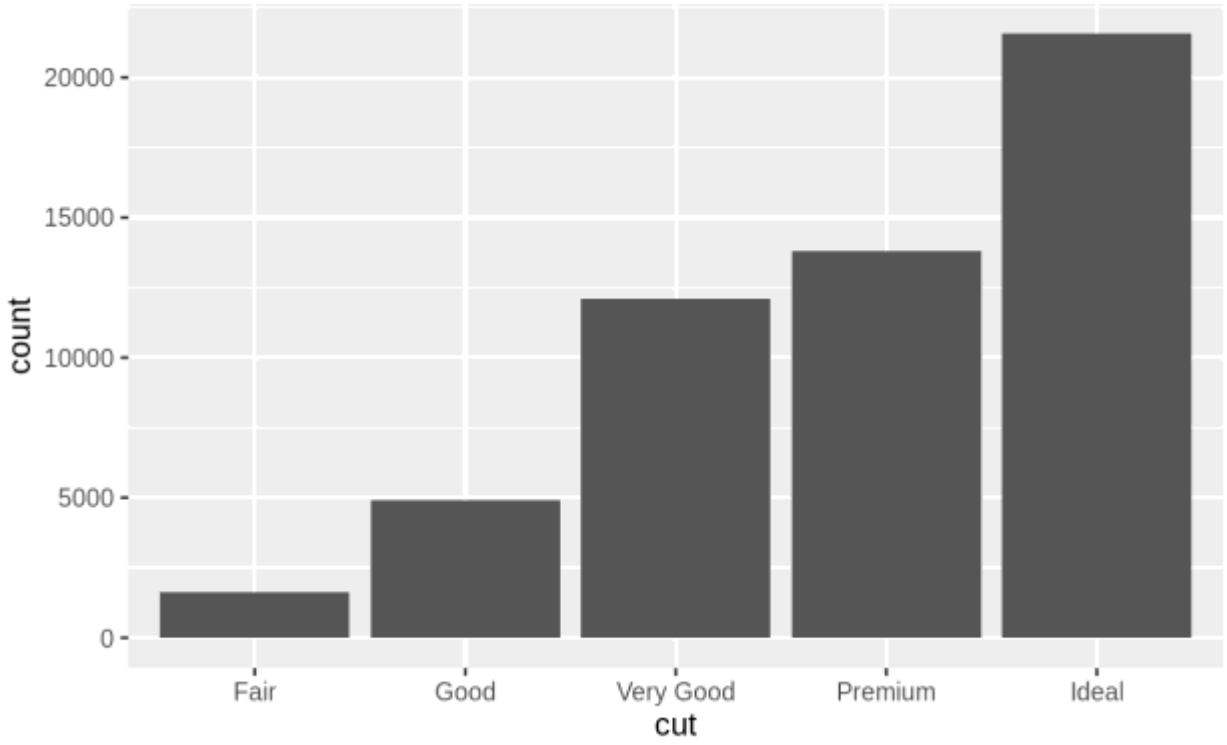


GGplot2 – Data Visualization

Statistical Transformations (3.7):

Many graphs, like scatterplot, plot the raw values of your dataset. Other plots, like bar charts, calculate new values to plot.

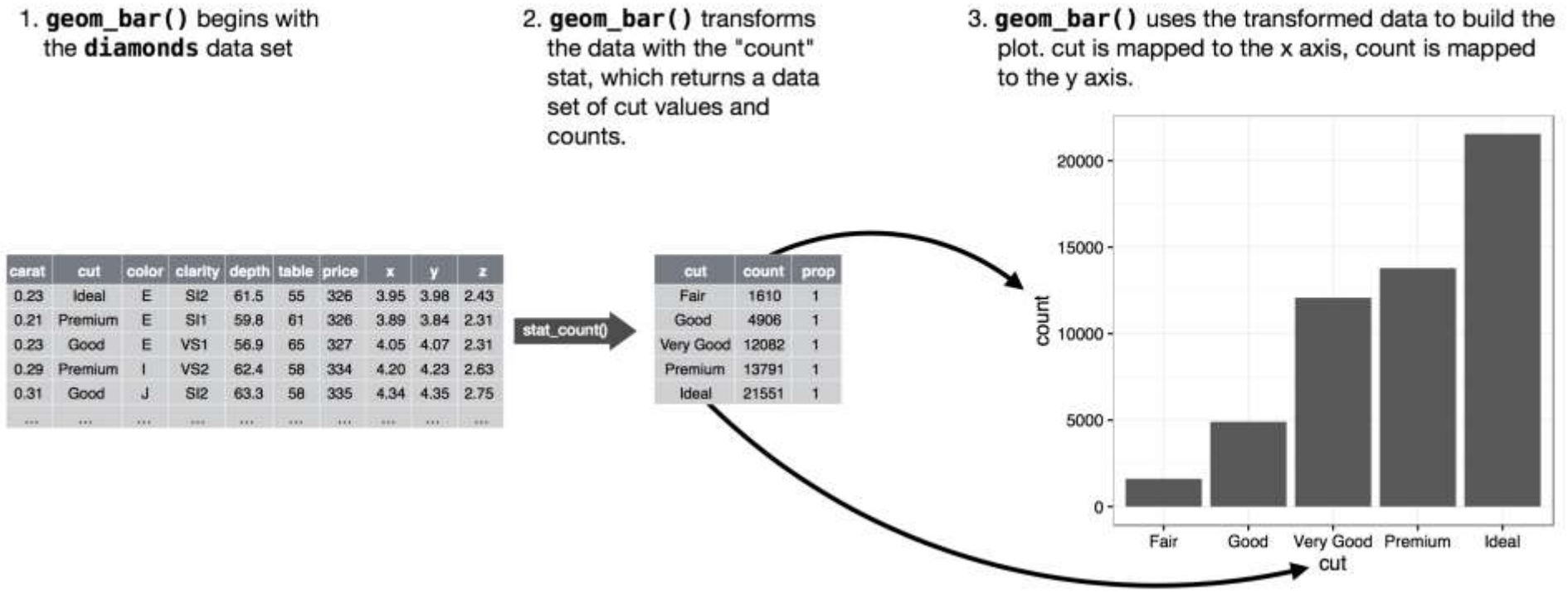
- bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- smoothers fit a model to your data and the
- boxplots compute a robust summary of th box.



GGplot2 – Data Visualization

Statistical Transformations (3.7):

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()**.



GGplot2 – Data Visualization

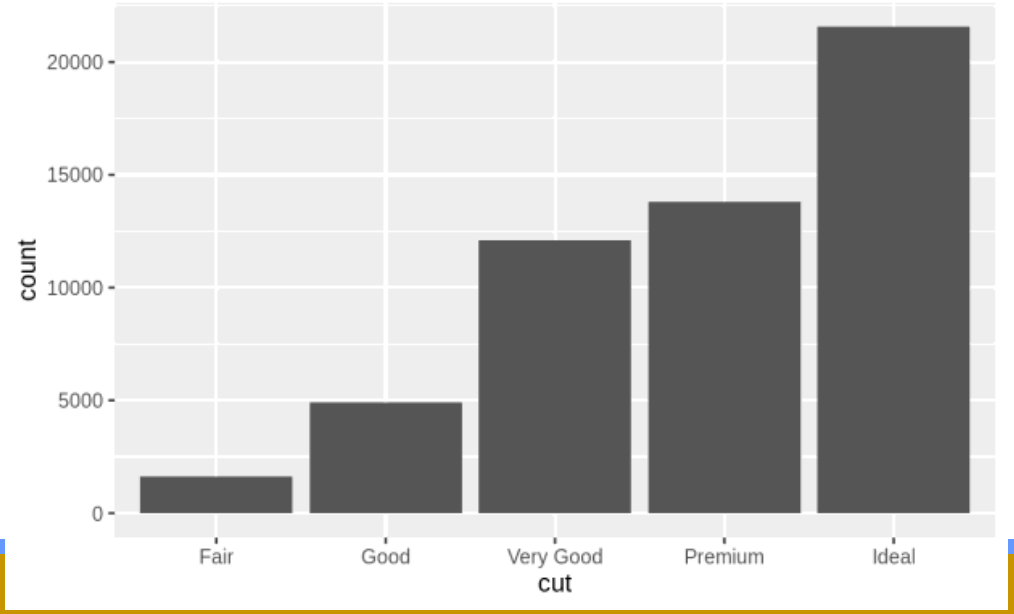
Statistical Transformations (3.7):

You can learn which stat a geom uses by inspecting the default value for the **stat** argument.

?geom_bar shows that the default value for “stat” is “count.” **geom_bar** uses **stat_count**.

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))
```



GGplot2 – Data Visualization

Statistical Transformations (3.7):

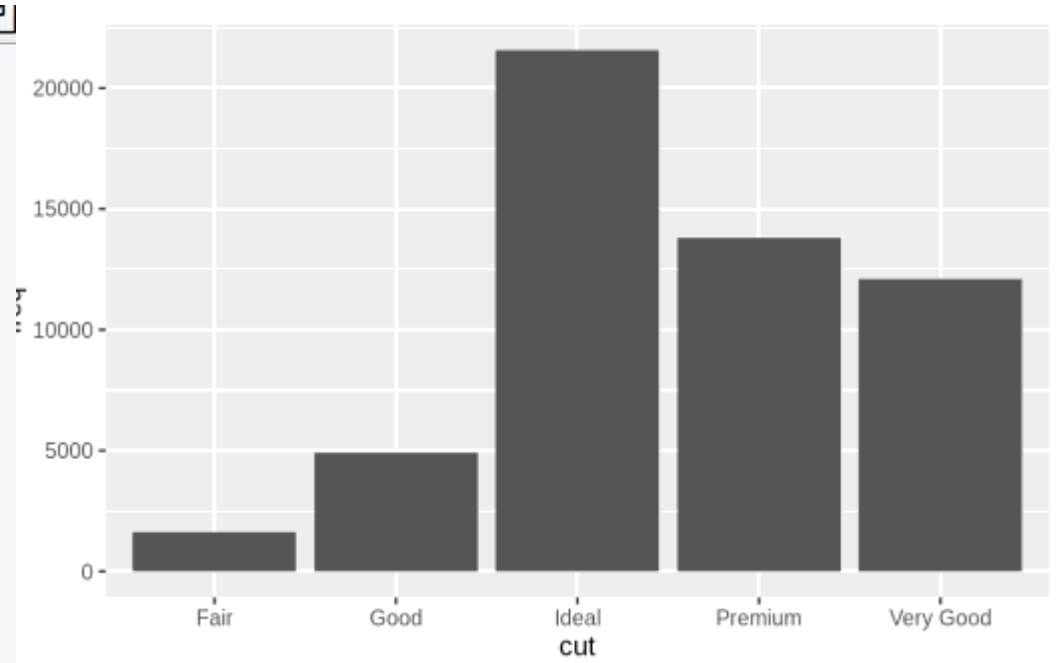
This works because every geom has a default stat; and every stat has a default geom. This means that you can typically use geoms without worrying about the underlying statistical transformation. There are three reasons you might need to use a stat explicitly:

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

GGplot2 – Data Visualization

Statistical Transformations (3.7):

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

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

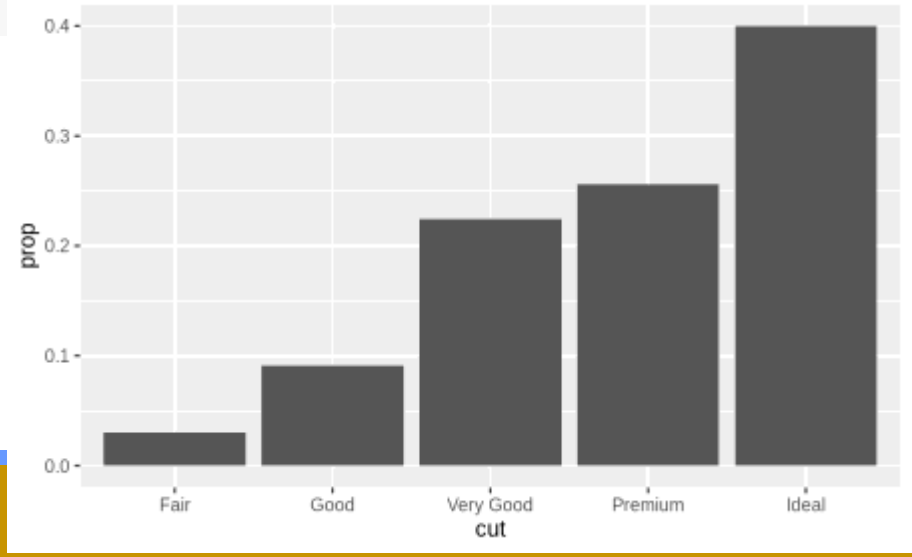


GGplot2 – Data Visualization

Statistical Transformations (3.7):

2. You might want to override the default mapping from transformed variables to aesthetics. For example, you might 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))
```

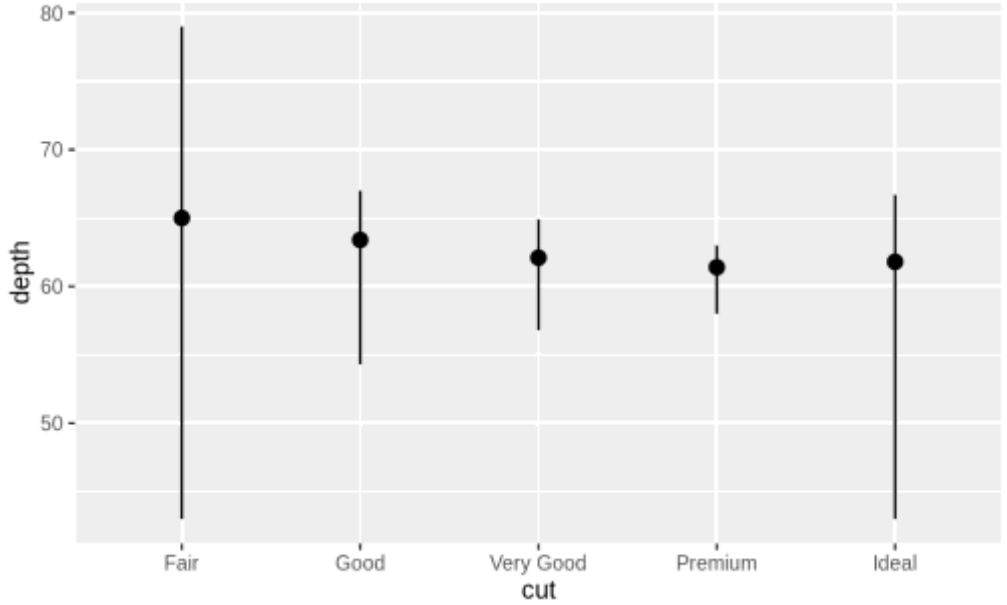


GGplot2 – Data Visualization

Statistical Transformations (3.7):

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

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



GGplot2 – Data Visualization

Statistical Transformations (3.7):

ggplot2 provides over 20 stats for you to use. Each stat is a function, so you can get help in the usual way, e.g. **?stat_bin.** To see a complete list of stats, try the ggplot2 cheatsheet.

Exercises:

1. What is the default geom associated with **stat_summary()**? How could you rewrite the previous plot to use that geom function instead of the stat function?

2. What does **geom_col()** do? How is it different from **geom_bar()**?

3. Most geoms and stats come in pairs that are almost always used in concert. Read through the documentation and make a list of all the pairs. What do they have in common?

4. What variables does stat_smooth() compute? What parameters control its behavior?

GGplot2 – Data Visualization

Exercises (contd):

5. In our proportion bar chart, we need to set **group = 1**. In other words what is the problem with these two graphs?

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

GGplot2 – Data Visualization

Position Adjustments (3.8):

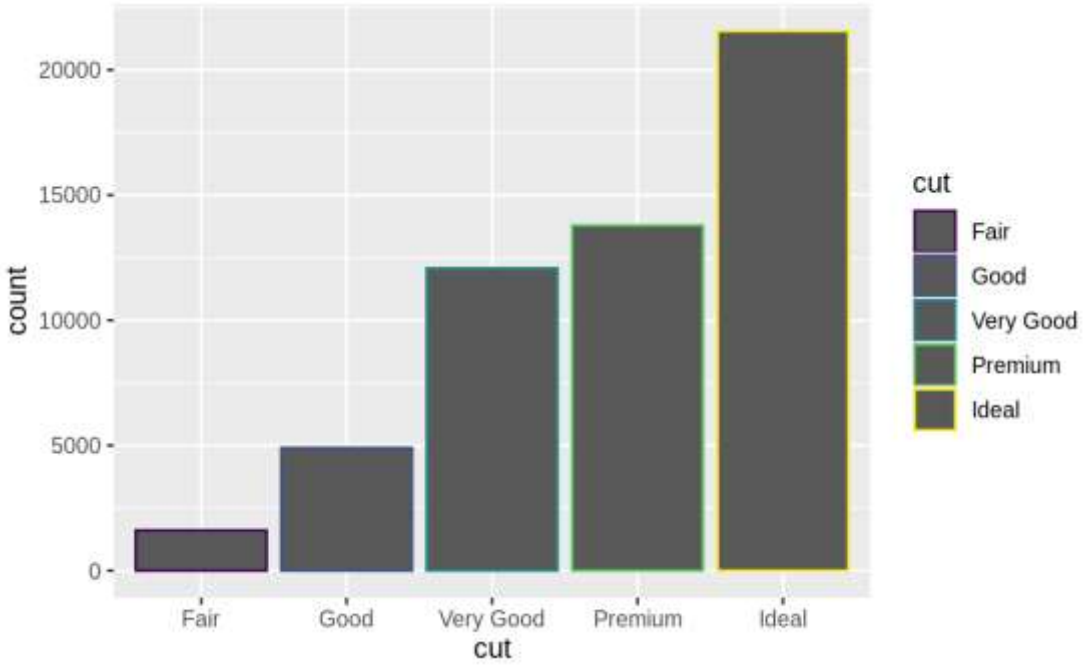
There's one more piece of magic associated with bar charts. You can colour a bar chart using either the **color** aesthetic, or more usefully, **fill**:

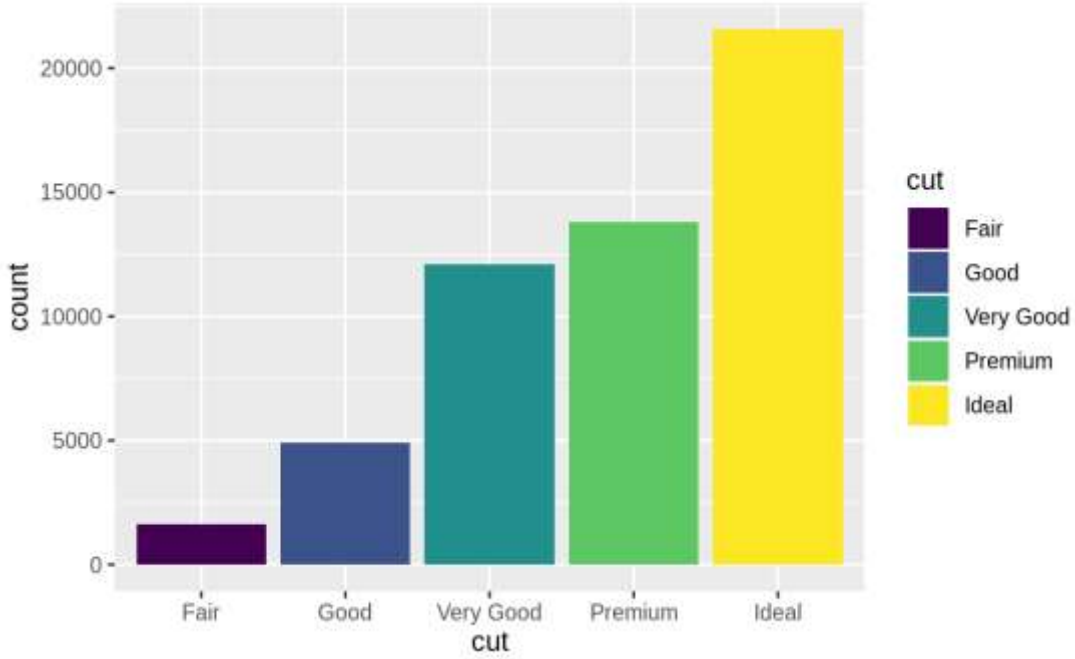
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, colour = cut))
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
```

GGplot2 – Data Visualization

Position Adjustments (3.8):

There's one more piece of magic associated with bar charts. You can colour a bar chart using either the **color** aesthetic, or more usefully, **fill**:





GGplot2 – Data Visualization

Position Adjustments (3.8):

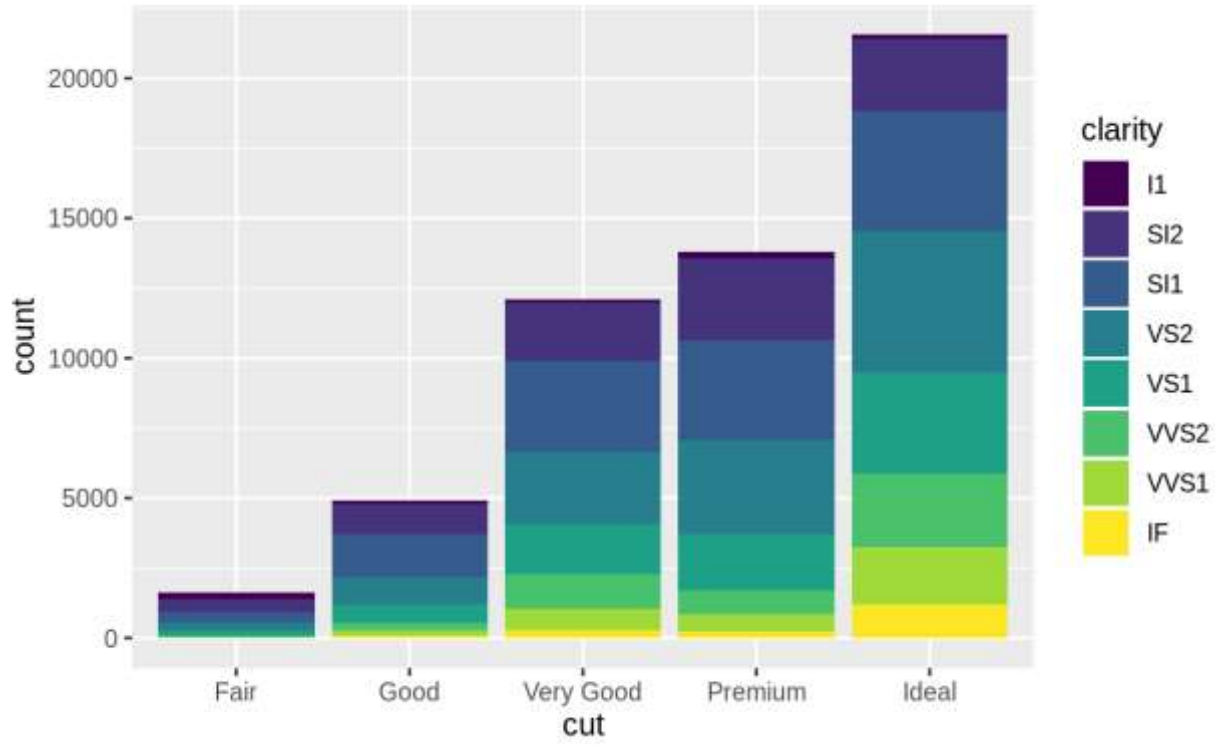
Note what happens if you map the fill aesthetic to another variable, like clarity: the bars are automatically stacked. Each colored rectangle represents a combination of **cut** and **clarity**.

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

GGplot2 – Data Visualization

Position Adjustments (3.8):

Note what happens if you map the fill aesthetic to another variable, like clarity: the bars are automatically stacked. Each colored rectangle represents a combination of **cut** and **clarity**.



GGplot2 – Data Visualization

Position Adjustments (3.8):

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 = "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**.

GGplot2 – Data Visualization

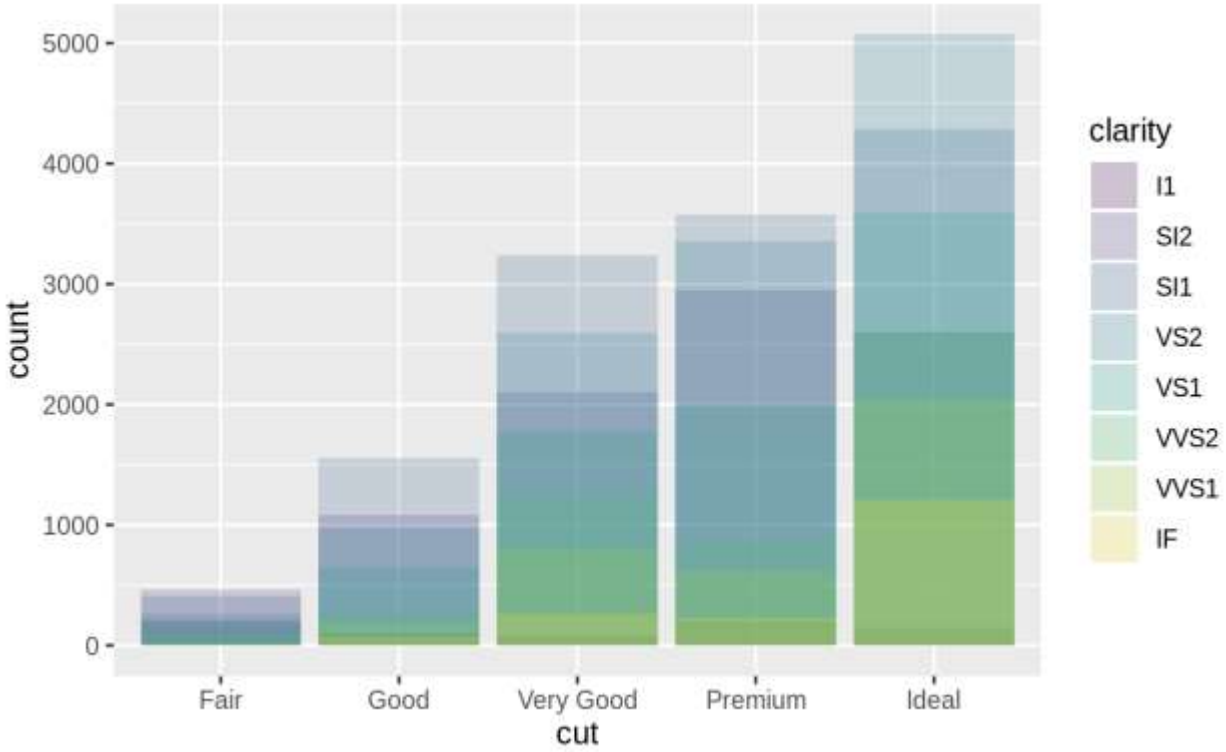
Position Adjustments (3.8):

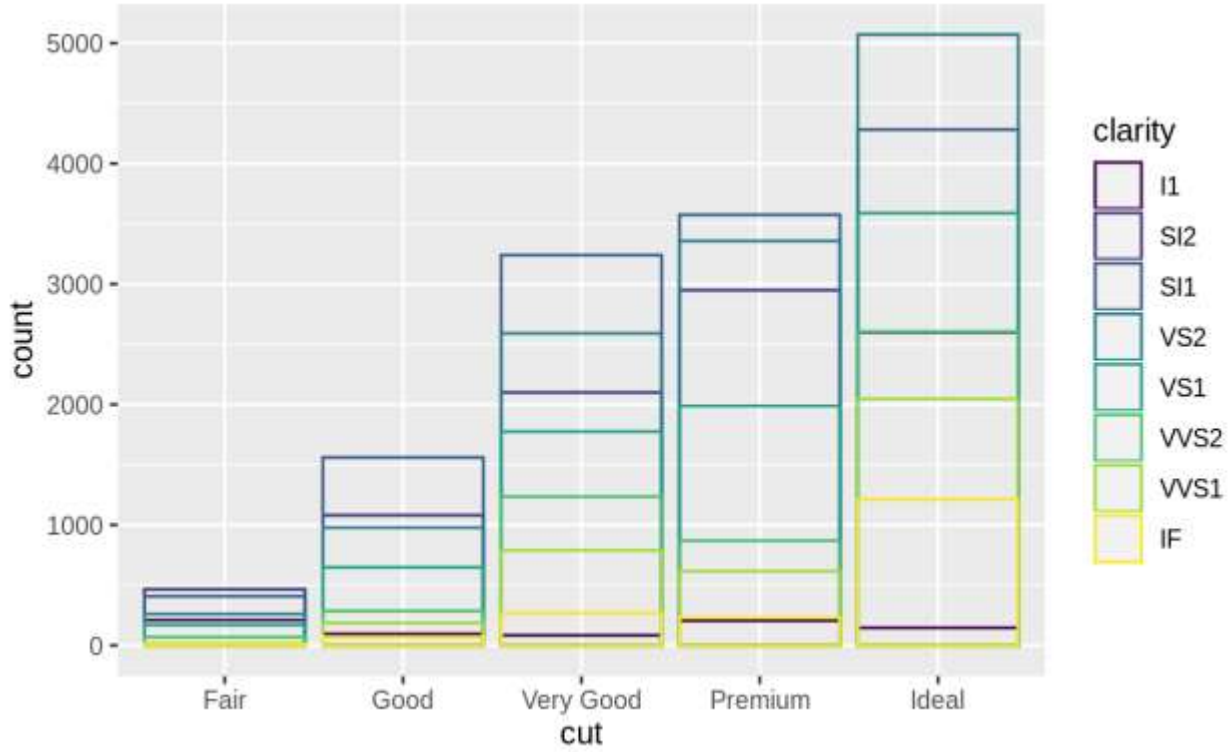
```
ggplot(data = diamonds, mapping = aes(x = cut, fill = clarity)) +
  geom_bar(alpha = 1/5, position = "identity")
ggplot(data = diamonds, mapping = aes(x = cut, colour = clarity)) +
  geom_bar(fill = NA, position = "identity")
```

GGplot2 – Data Visualization

Position Adjustments (3.8):

The identity position adjustment is more useful for 2d geoms, like points, where it is the default.





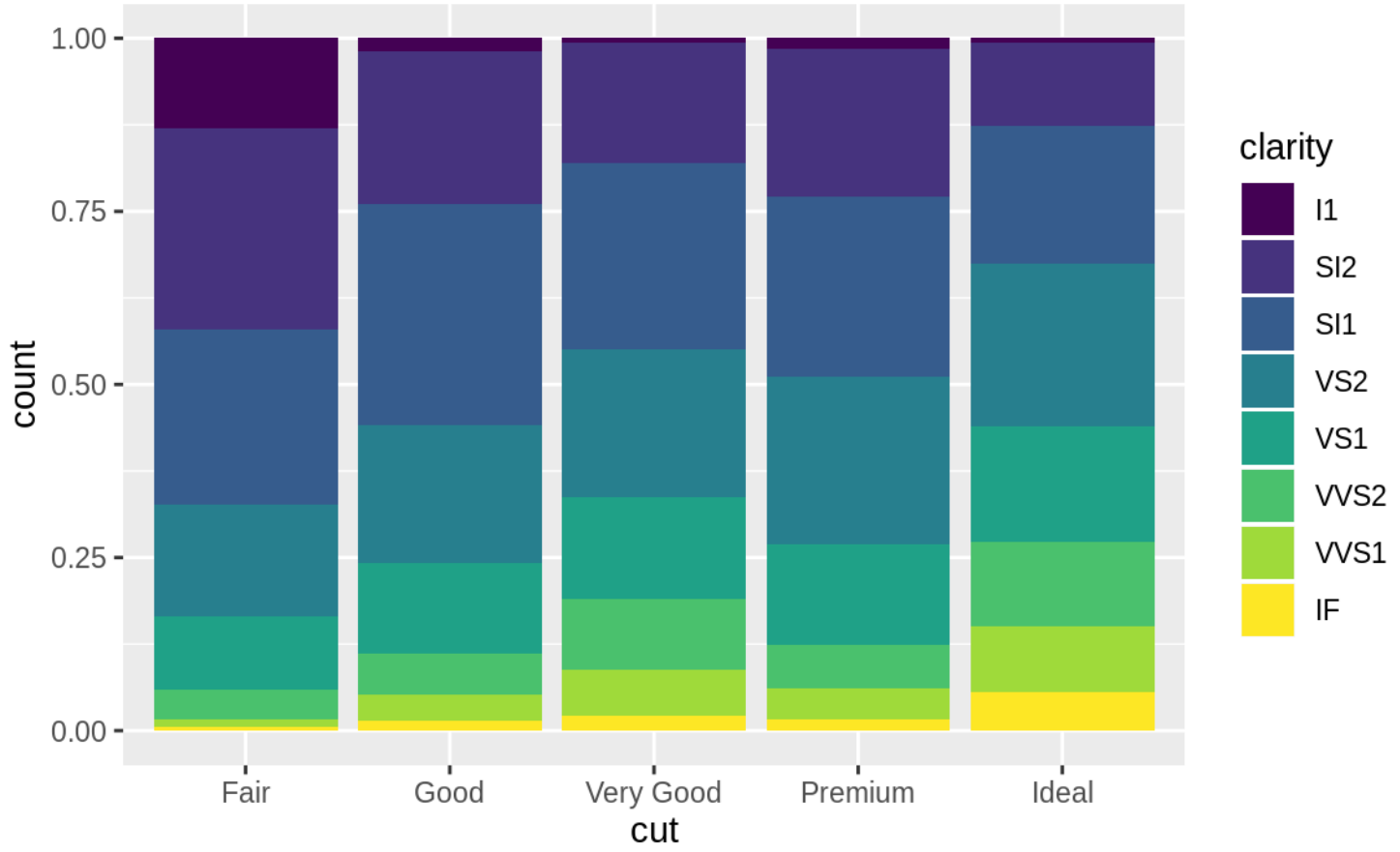
GGplot2 – Data Visualization

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

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

GGplot2 – Data Visualization

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



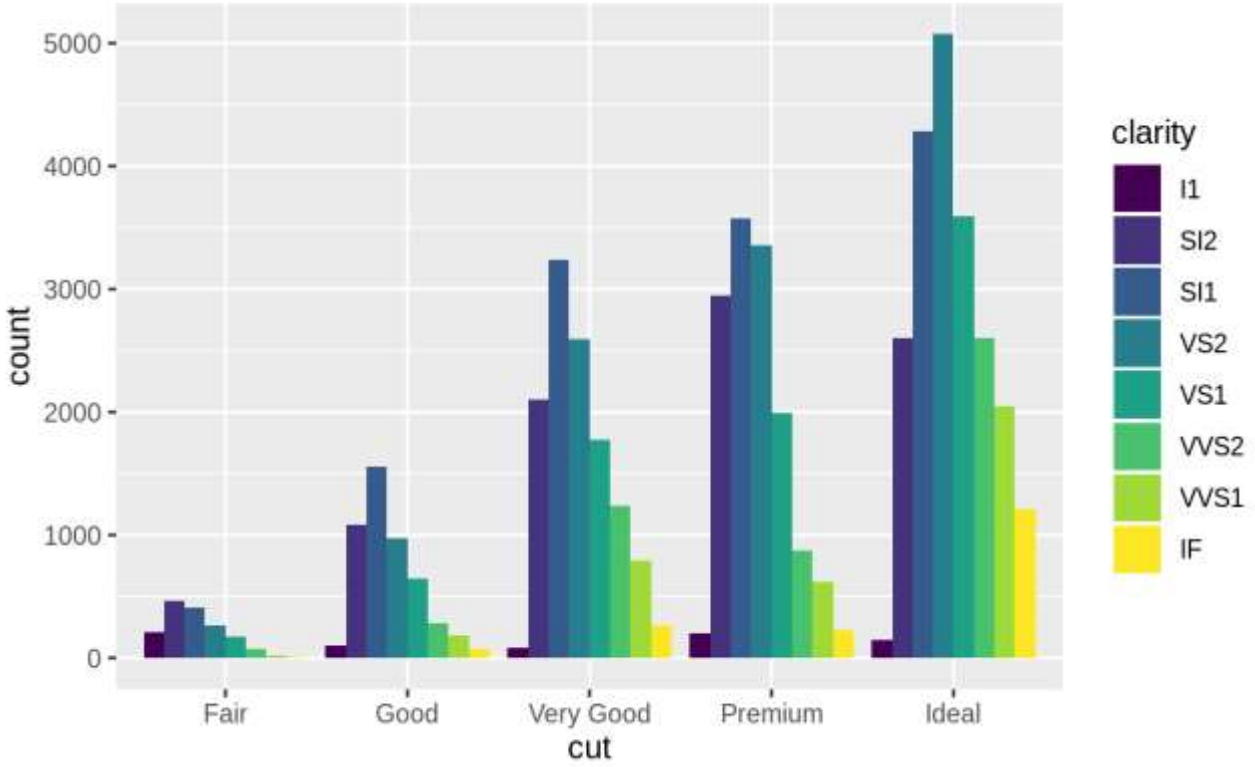
GGplot2 – Data Visualization

position = “dodge” places overlapping objects directly beside one another. This makes it easier to compare individual variables.

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

GGplot2 – Data Visualization

position = “dodge” places overlapping objects directly beside one another. This makes it easier to compare individual variables.



GGplot2 – Data Visualization

Resources:

R for Data Science: https://r4ds.had.co.nz/data-visualisation.html

Layer Stats: https://ggplot2.tidyverse.org/reference/

Layer Geoms: https://ggplot2.tidyverse.org/reference/#section-layer-geoms

Ggplot2: Elegant Graphics for Data Analysis: https://ggplot2-book.org/

Smooth Conditionals: https://ggplot2.tidyverse.org/reference/geom_smooth.html