# Data Visualization with ggplot2

**Step1: Creating ggplot:**

- You begin a plot with the function **ggplot()**.It 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 = iris)** creates an empty graph.
- Now 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, iris.

**Step2: Graphing Template**

- For making graphs with **ggplot2** the following code with a dataset, a geom function, or a collection of mappings and extend this template to make different types of graphs:

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

## Aesthetic Mappings

- Graph plotting in R is of two types:

1. **One-dimensional Plotting:**

➢ In one-dimensional plotting, we plot one variable at a time.

➢ For example, we may plot a variable with the number of times each of its values occurred in the entire dataset (frequency).

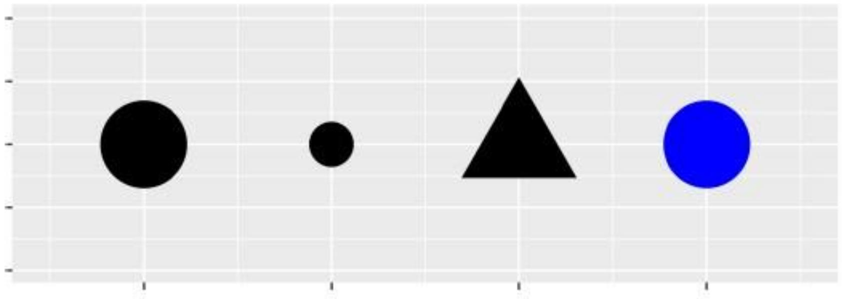
➢ So, it is not compared to any other variable of the dataset. These are the 4 major types of graphs that are used for One-dimensional analysis –

  - Five Point Summary
  - Box Plotting
  - Histograms
  - Bar Plotting

## 2. Two-dimensional Plotting:

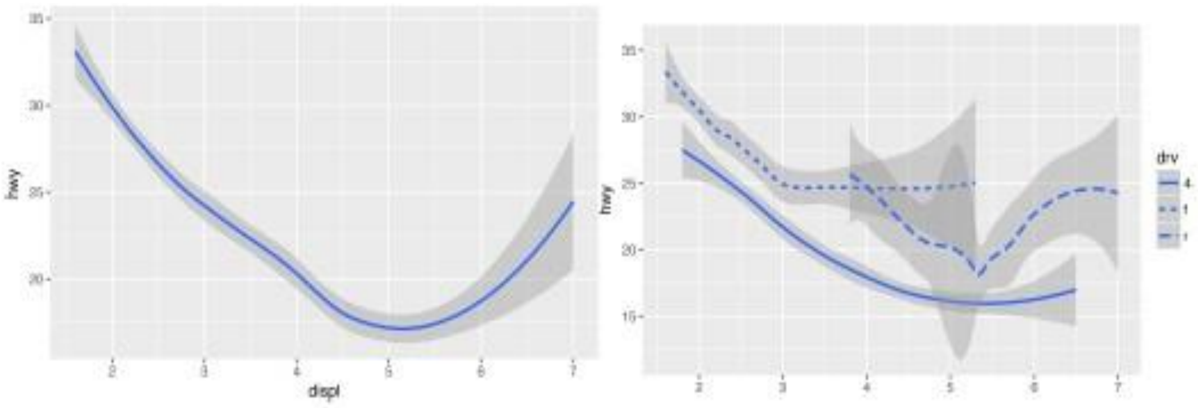
- In two-dimensional plotting, we visualize and compare one variable with respect to the other.
- For example, in Air Quality dataset, we would like to compare how the AQI varies with the temperature at a particular place.
- Temperature and AQI are two different variables and we wish to see how one changes with respect to the other.
- These are the 3 major kinds of graphs used for such kinds of analysis
  - Box Plotting
  - Histograms
  - Scatter plots

- 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 display a point in different ways by changing the values of its aesthetic properties.
- By using these aesthetic properties we change the levels of a point’s size, shape, and color to make the point small, triangular, or blue:

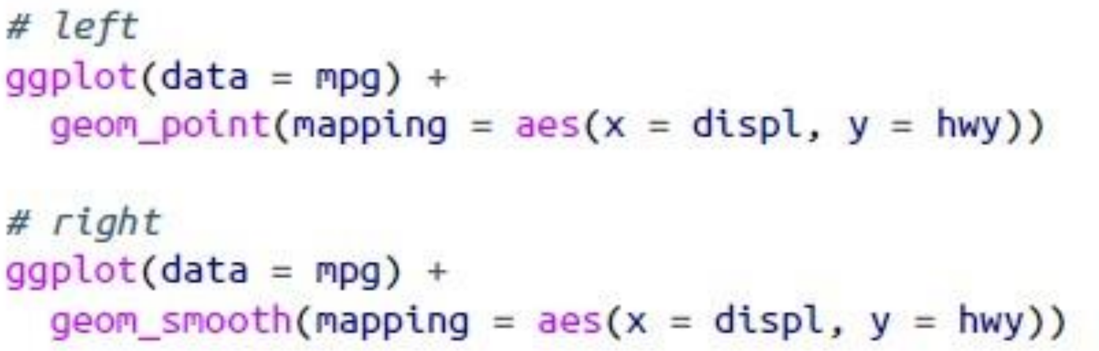


# Geometric Objects

- A *geom* is the geometrical object that a plot uses to represent data.
- 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**.
- You can use different geoms to plot the same data. To change the geom in your plot, change the geom function that you add to ggplot().
- ggplot() is providing two such geom's- **geom_point()** and **geom_smooth().**



- Both plots contain the same x variable and the same y variable, and both describe the same data. But the plots are not identical.
- Each plot uses a different visual object to represent the data. **ggplot2** use different *geoms*.

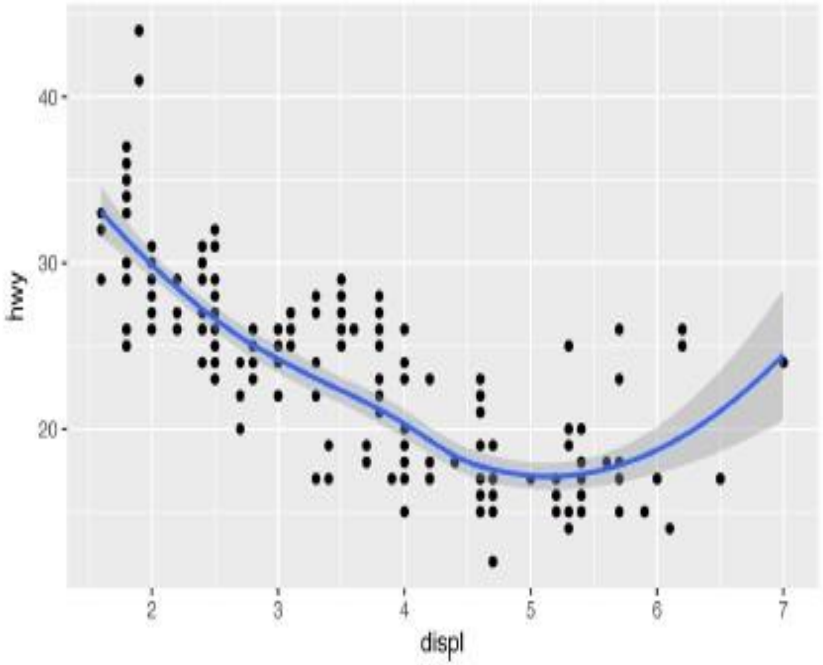


```
# left
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))

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

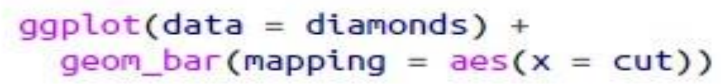
- geom_smooth() will draw a different line, with a different linetype, for each unique value of the variable that you map to **linetype =.**
- To display multiple geoms in the same plot, add multiple geom functions to ggplot():

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

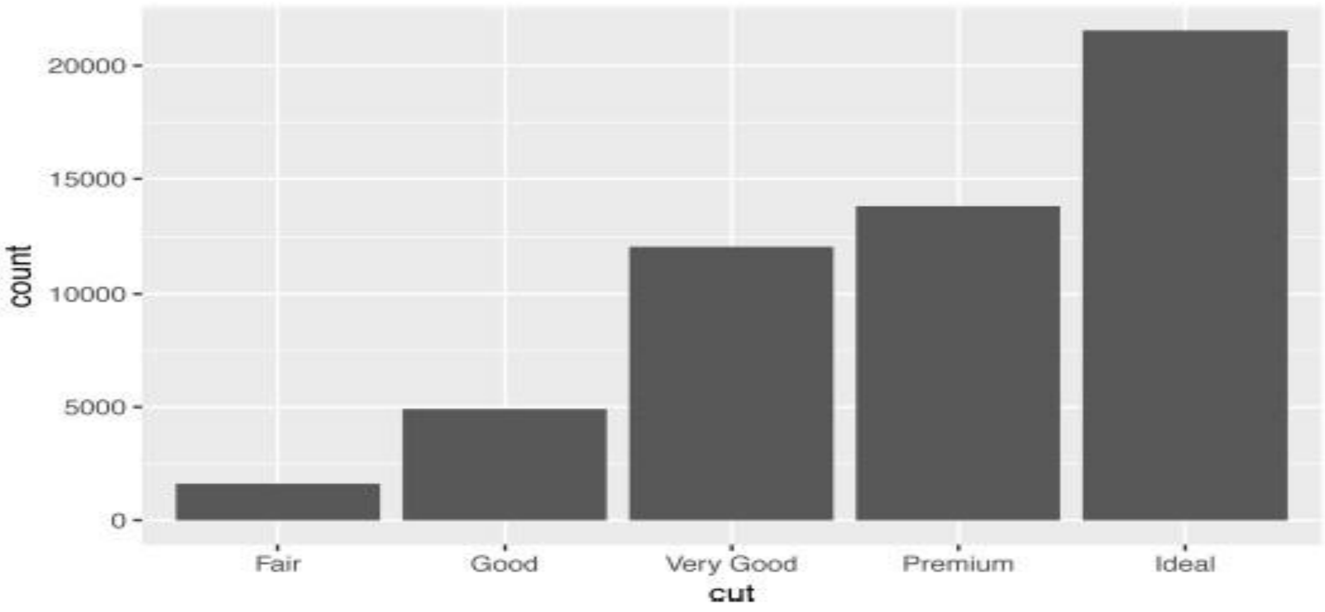


# Statistical Transformations

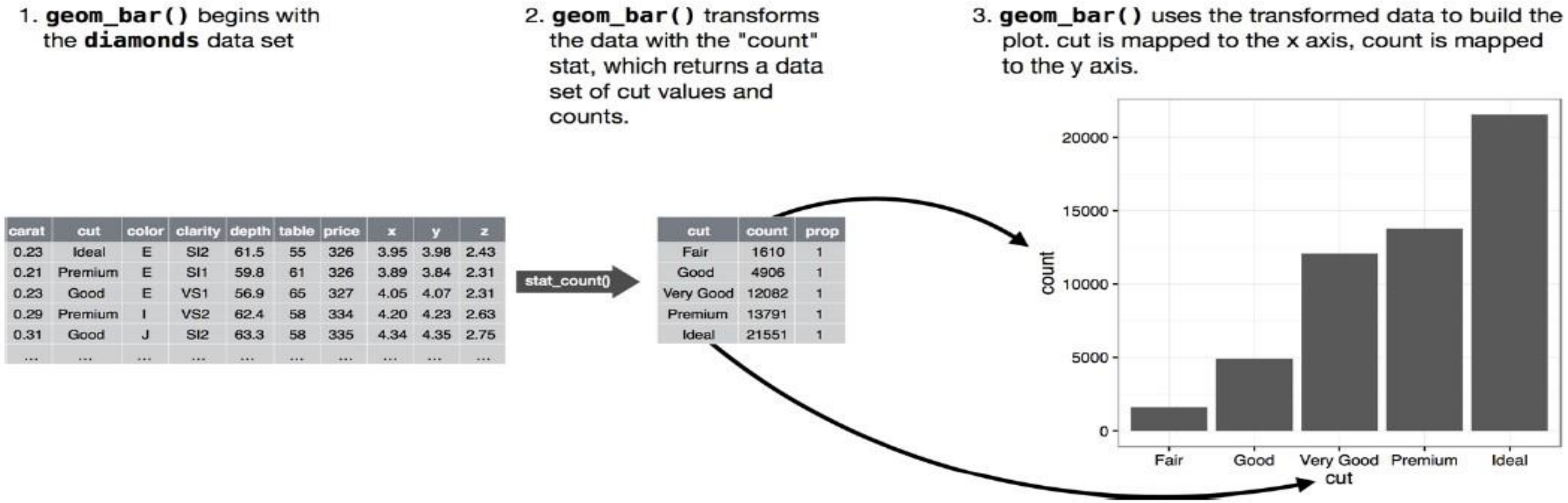
- Bar charts seem simple, but they are interesting because they reveal something subtle about plots.
- A basic bar chart, as drawn with **geom_bar().**



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



- Bar charts, histograms, and frequency polygons bin your data and then plot bin counts, the number of points that fall in each bin.
- Smoothers fit a model to your data and then plot prediction from the model.
- Boxplots compute a robust summary of the distribution and display a specially formatted box.
- The default value for stat is “count,” which means that **geom_bar()** uses **stat_count().**



1. **geom_bar()** begins with the **diamonds** data set

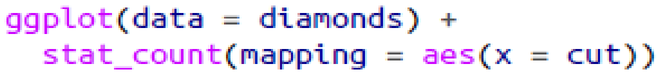
2. **geom_bar()** transforms the data with the "count" stat, which returns a data set of cut values and counts.

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.

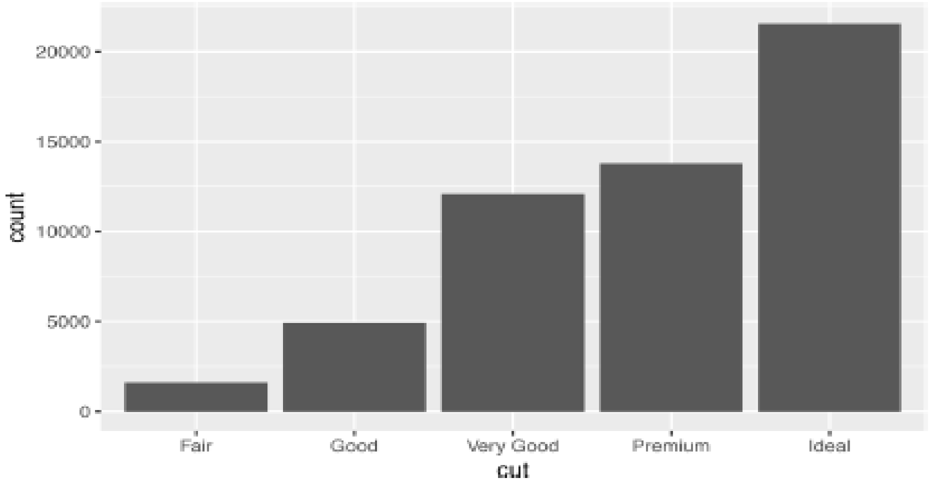
| carat | cut | color | clarity | depth | table | price | x | y | z |
|---|---|---|---|---|---|---|---|---|---|
| 0.23 | Ideal | E | SI2 | 61.5 | 55 | 326 | 3.95 | 3.98 | 2.43 |
| 0.21 | Premium | E | SI1 | 59.8 | 61 | 326 | 3.89 | 3.84 | 2.31 |
| 0.23 | Good | E | VS1 | 56.9 | 65 | 327 | 4.05 | 4.07 | 2.31 |
| 0.29 | Premium | I | VS2 | 62.4 | 58 | 334 | 4.20 | 4.23 | 2.63 |
| 0.31 | Good | J | SI2 | 63.3 | 58 | 335 | 4.34 | 4.35 | 2.75 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

| cut | count | prop |
|---|---|---|
| Fair | 1610 | 1 |
| Good | 4906 | 1 |
| Very Good | 12082 | 1 |
| Premium | 13791 | 1 |
| Ideal | 21551 | 1 |

- You can re-create the previous plot using **stat_count()** instead of geom_bar():

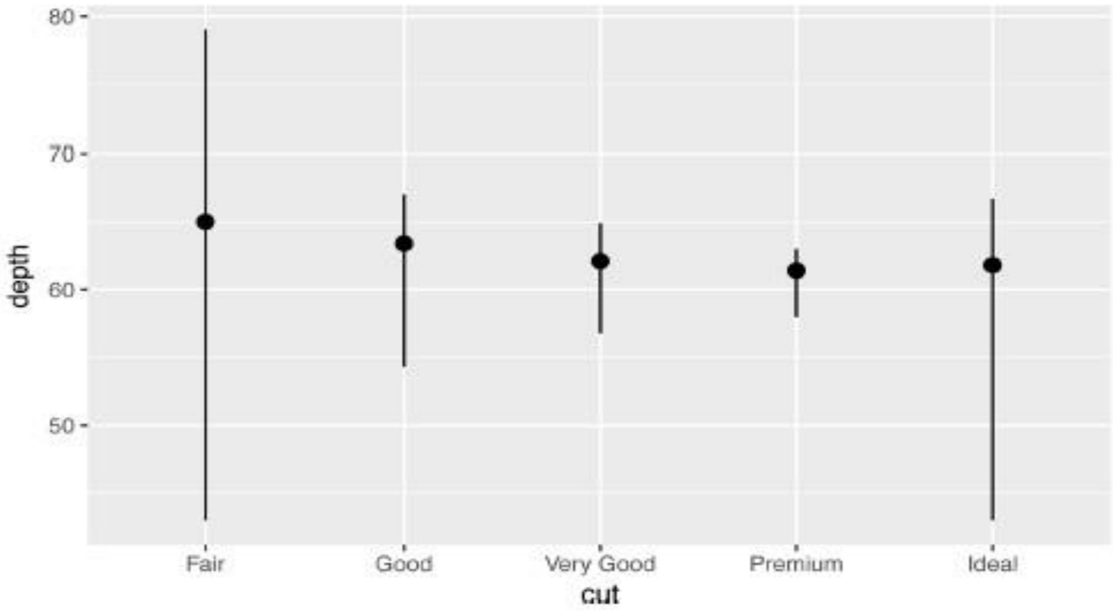


```
ggplot(data = diamonds) +
  stat_count(mapping = aes(x = cut))
```



- You might use **stat_summary(),** which summarizes the y values for each unique x value, to draw attention.

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



# Position Adjustments

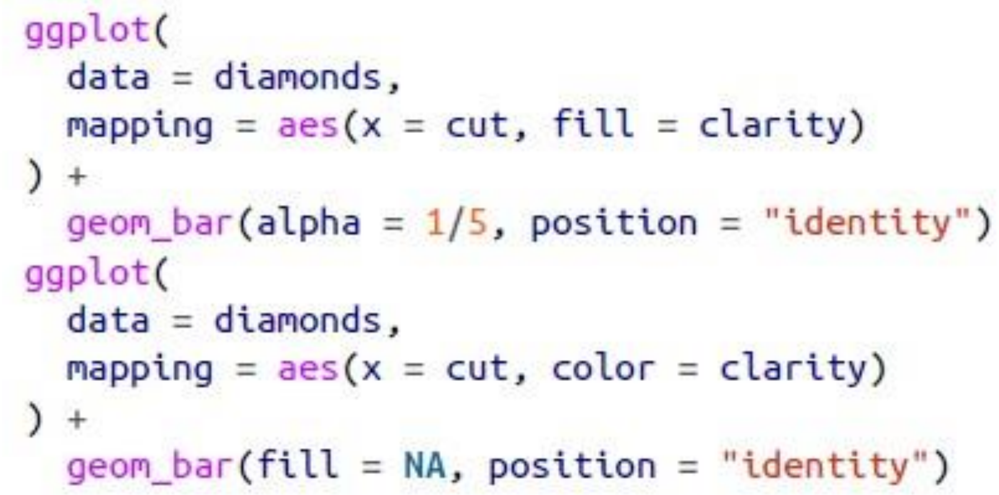
- You can color a bar chart using either the **color** aesthetic, or
  - ✓ by **fill = x** axes variable.
  - ✓ **clarity:** the bars are automatically stacked.
- The stacking is performed automatically by the *position adjustment* specified by the position argument. So ggplot is providing 3 such position options. They are: "identity", "dodge" or "fill".

**position = "identity“:**

➢ It 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**

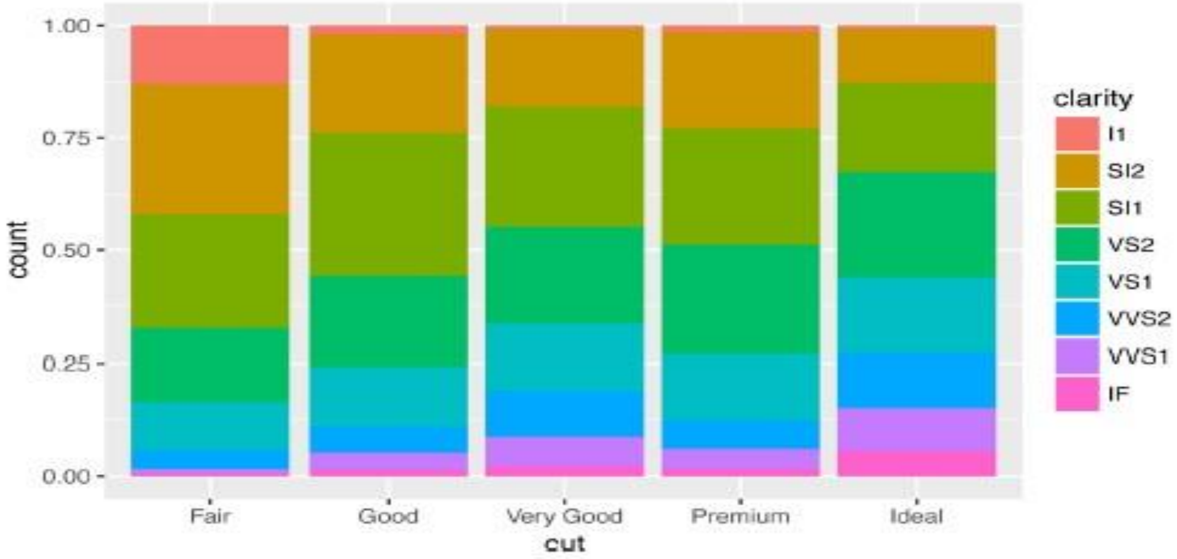


```
ggplot(
  data = diamonds,
  mapping = aes(x = cut, fill = clarity)
) +
  geom_bar(alpha = 1/5, position = "identity")
ggplot(
  data = diamonds,
  mapping = aes(x = cut, color = clarity)
) +
  geom_bar(fill = NA, position = "identity")
```

**position = "fill":**

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

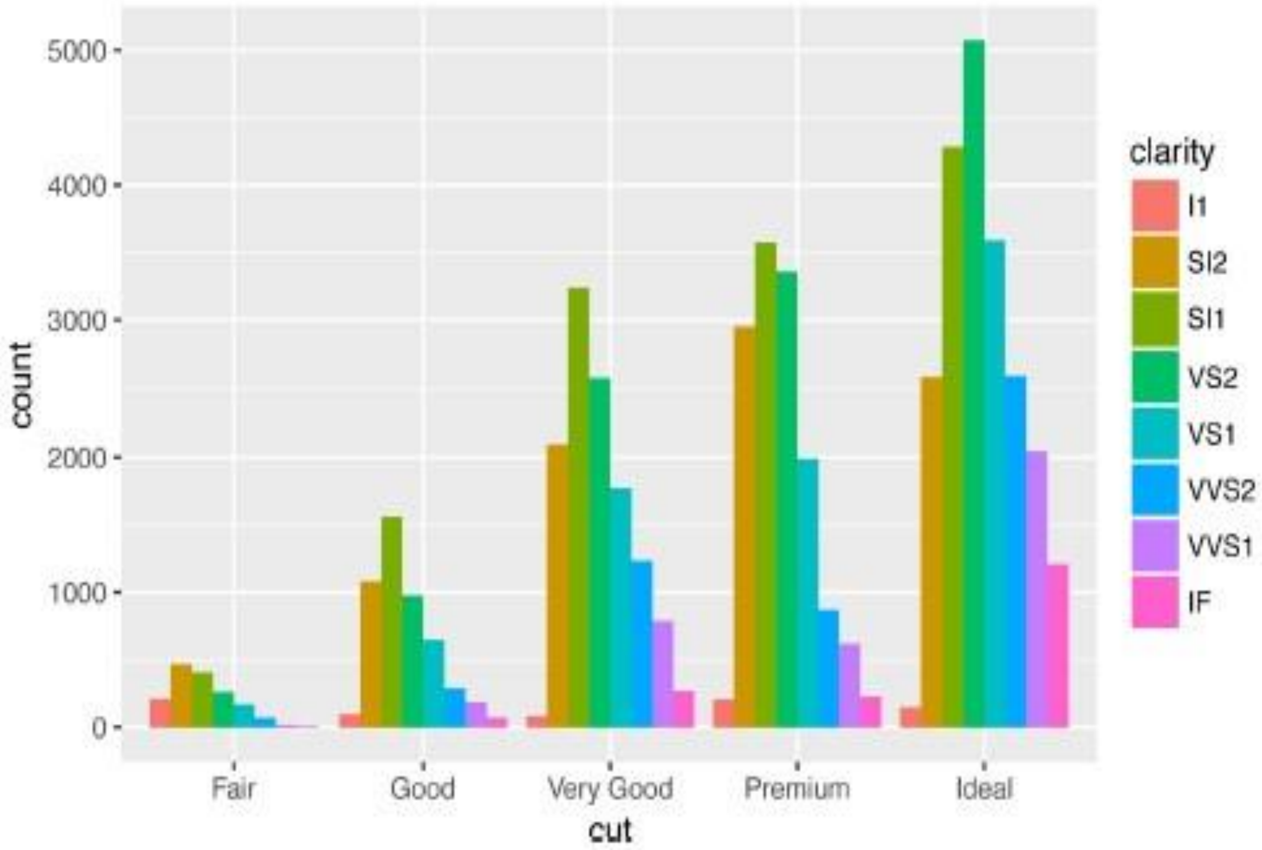
```
ggplot(data = diamonds) +
   geom_bar(mapping = aes(x = cut, fill = clarity), position = "fill")
```



**position = "dodge":**

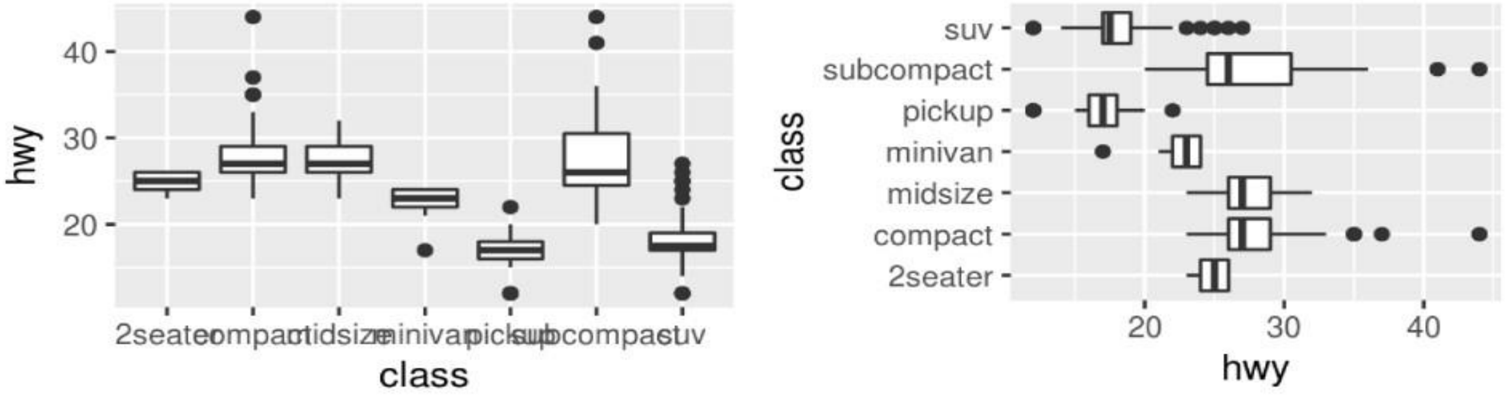
- It places overlapping objects directly *beside* one another. This makes it easier to compare individual values:

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

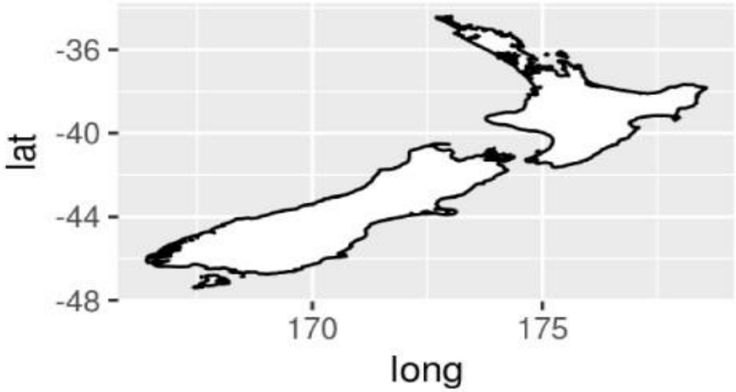

count
5000
4000
3000
2000
1000
0
Fair
Good
Very Good
Premium
Ideal
cut
clarity
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF

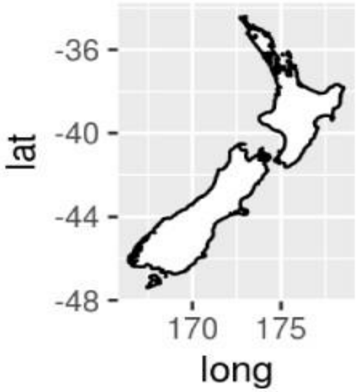
# Coordinate Systems

- ➢Coordinate systems are probably the most complicated part of **ggplot2**.
- ➢The default coordinate system is the Cartesian coordinate system where the x and y position act independently to find the location of each point.
- ➢There are a number of other coordinate systems that are occasionally helpful:
  - ✓**coord_flip():** It switches the x- and y-axes, if you want horizontal boxplots. It’s also useful for long labels.

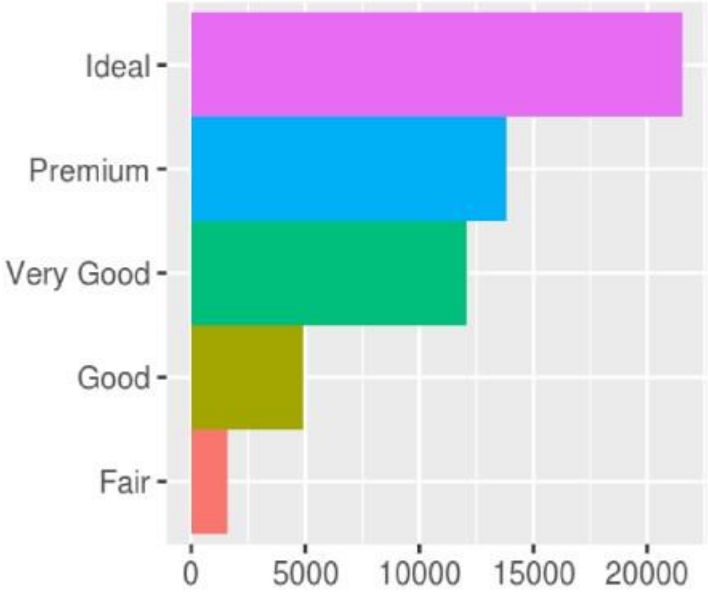


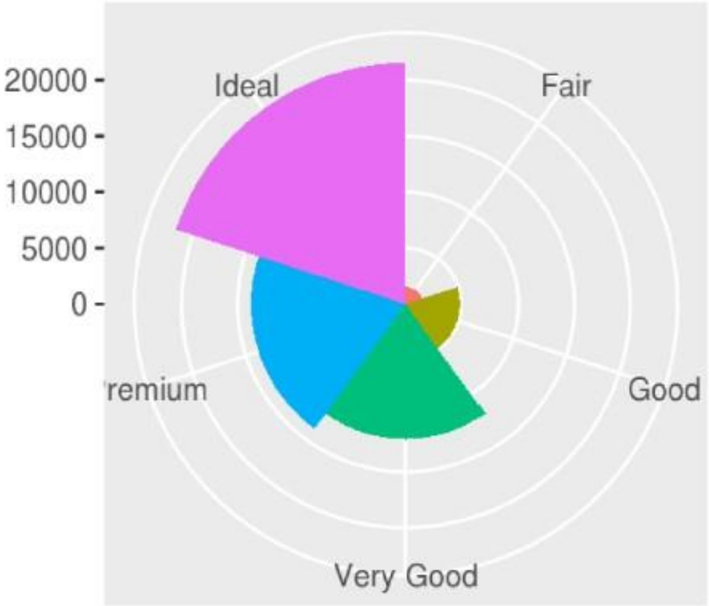
- **coord_quickmap()** sets the aspect ratio correctly for maps. This is very important if you’re plotting spatial data with **ggplot2.** Example: maps





- **coord_polar()** uses polar coordinates. Polar coordinates reveal an interesting connection between a bar chart and a Coxcomb chart.


Ideal
Premium
Very Good
Good
Fair
0
5000
10000
15000
20000


20000
15000
10000
5000
0
Ideal
Fair
remium
Good
Very Good