Data Visualization in R

Open Source Tools for Intelligent Systems
summer semester 2017

Rami Yaari (ramiyaari@gmail.com)
Naama Kopelman (naama.kopelman@gmail.com)

Plotting Using Base R Functions

The package graphics, which is part of the base R installation, has various functions for creating plots, including:

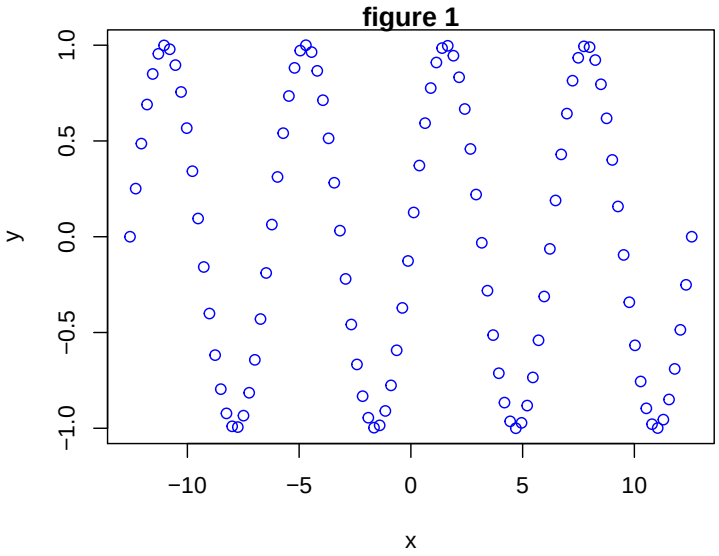
- plot - a generic function for plotting R objects
- hist - producing a histogram
- boxplot - producing a box-and-whisker plot
- barplot - producing a bar plot
- pairs - producing a matrix of scatterplots
- lines - adding connected line segments to a plot
- points - adding points to a plot
- par - set the graphical parameters for a plot

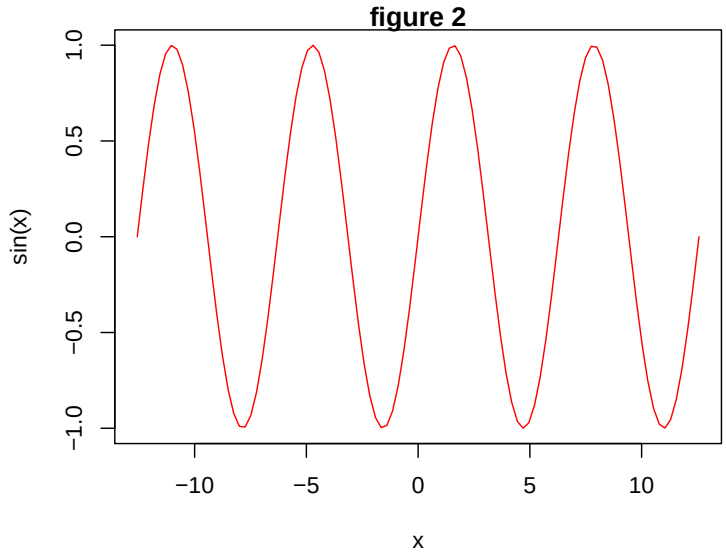
For the full list of functions in the graphics package type **library(help = "graphics")** in the RStudio console.

Base R plot() Function

Example using plot:

```
x <- seq(-4*pi,4*pi,length=100)
y <- sin(x)
plot(x,y,type='p',col='blue')                    # plot as points in blue
title(main='figure 1')
plot(x,y,type='l',col='red',ylab='sin(x)')   # plot as a red line
title(main='figure 2')
```





The plots Window

When creating a plot in RStudio the plot is shown in the **Plots** window.
Using this window it is possible to:

- Navigate backward and forward between all the open plots
- Zoom in by openning a pop-up window with a plot
- Export a plot to an image file of various possible types
- Remove the current or all open plots

The ggplot2 Package

- We will not go into more details with the graphics package. Instead we will focus on the **ggplot2** package, which had become the popular method for creating plots in R.

The ggplot2 Package

- We will not go into more details with the graphics package. Instead we will focus on the **ggplot2** package, which had become the popular method for creating plots in R.

- The package was created by Hadley Wickham in 2005, and it is based on principles he devloped and termed as the "Grammar of Graphics" (hence the 'gg' in ggplot).

The ggplot2 Package

- We will not go into more details with the graphics package. Instead we will focus on the **ggplot2** package, which had become the popular method for creating plots in R.

- The package was created by Hadley Wickham in 2005, and it is based on principles he devloped and termed as the "Grammar of Graphics" (hence the 'gg' in ggplot).

- The syntax of ggplot is a little less straight forward at first than that of the graphics package, but once you grasp the concept, it becomes easier to produce more complex graphs, and the resulting figures looks nicer.

The ggplot2 Package

- We will not go into more details with the graphics package. Instead we will focus on the **ggplot2** package, which had become the popular method for creating plots in R.

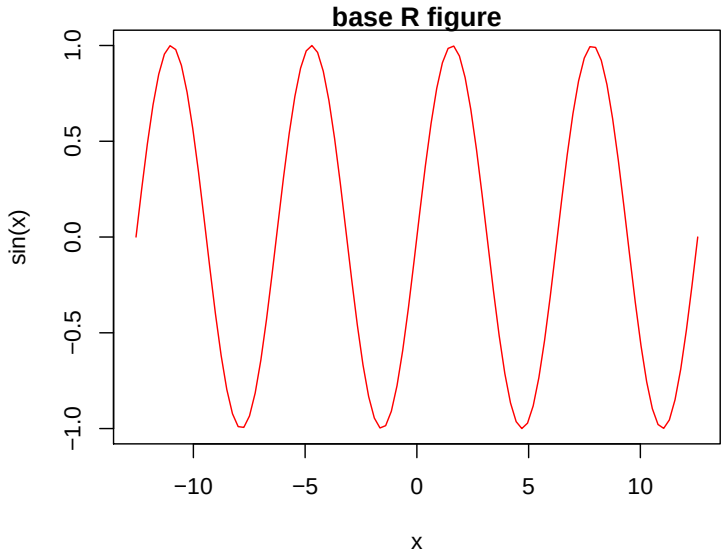
- The package was created by Hadley Wickham in 2005, and it is based on principles he devloped and termed as the "Grammar of Graphics" (hence the 'gg' in ggplot).

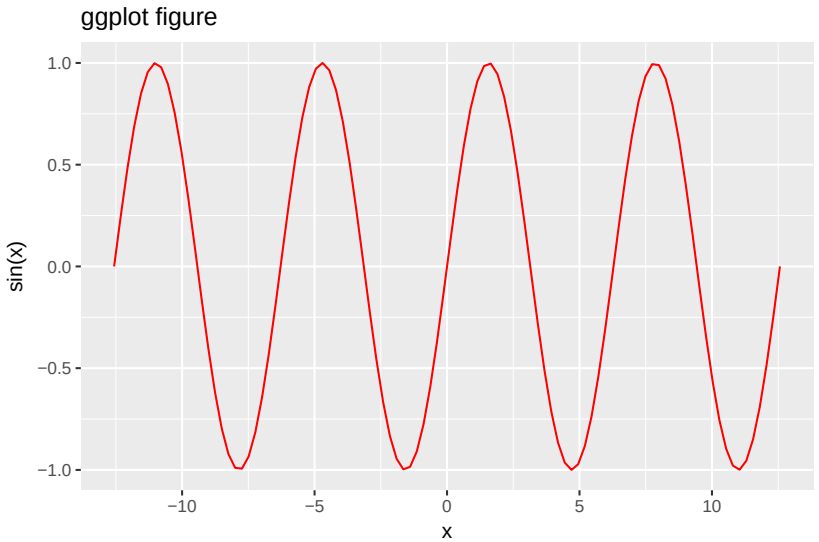
- The syntax of ggplot is a little less straight forward at first than that of the graphics package, but once you grasp the concept, it becomes easier to produce more complex graphs, and the resulting figures looks nicer.

- To use ggplot, it is necessary to install and load the **ggplot2** package. Another option is to install and load the **tidyverse** package which includes the **ggplot2** package as well as other packages and some data sets. We will install **tidyverse** as it includes the data sets will be using for examples below.

Base graphics Vs. ggplot

```r
x <- seq(-4*pi,4*pi,length=100)
y <- sin(x)
# plot x vs. sin(x) using base R graphics
plot(x,y,type='l',col='red',ylab='sin(x)')
title(main='base R figure')
# plot x vs. sin(x) using ggplot
ggplot() +
  geom_line(mapping=aes(x=x,y=y),col='red') +
  labs(y='sin(x)',title='ggplot figure')
```





Basic ggplot Template

The basic template for making graphs using ggplot looks like this:

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

ggplot graphs are composed in layers. The '+' character is used to add another layer. In this template there are just two layers but more layers can be added. The components of this template are:

- ggplot() - creates a coordinate system the we can began to add layers to
- <DATA> - the default data set to be used in creating the plot (optional).
- <GEOM_FUNCTION> - the geometrical object that will be representing the data in the plot. There are numerous different geom objects in the package (type **library(help=ggplot2)** to see them all). Some notable ones are:
  - geom_point - used for scatterplots
  - geom_line - used for line charts
  - geom_bar - used for bar charts
  - geom_histogram - used for histograms
  - geom_boxplot - used for box and whiskers plots
- aes(<MAPPINGS>) - aesthetic mappings describing how variables in the data are mapped to visual properties of geoms (i.e., x, y, shape, color, size . . .)

The mpg Data Set

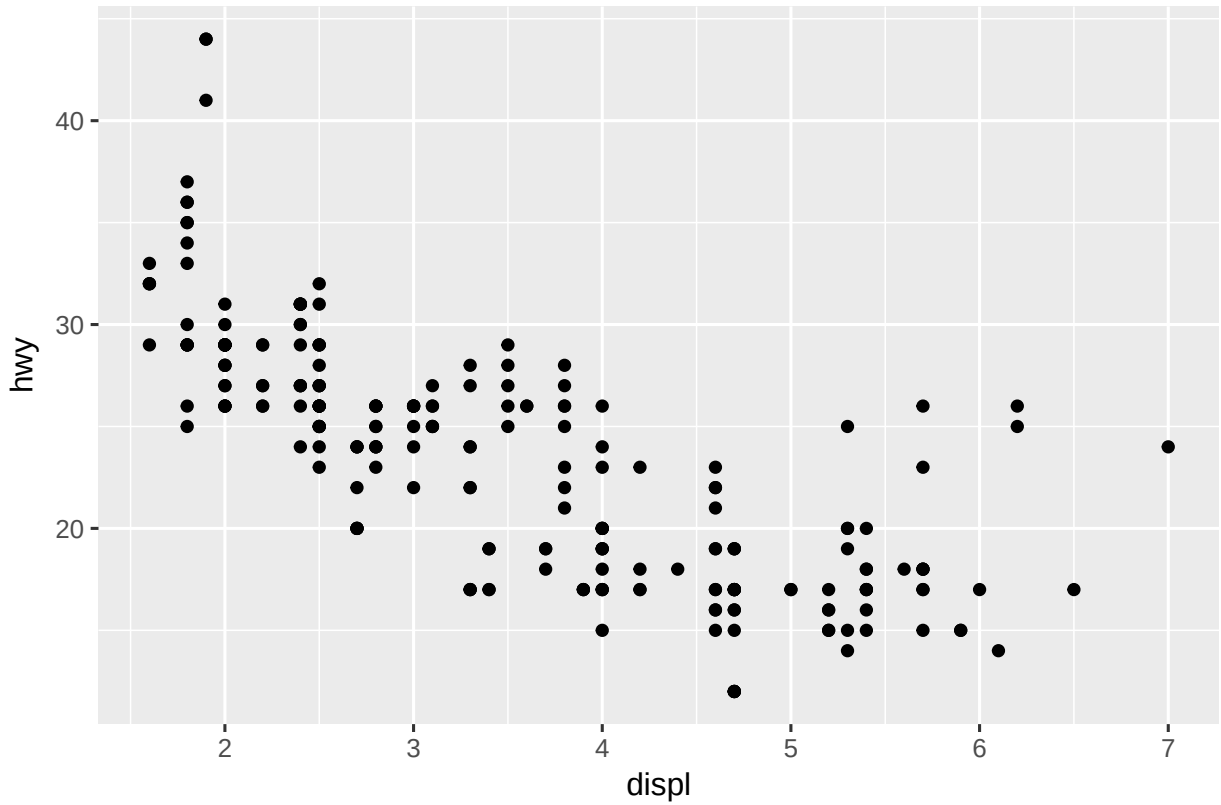
To start exploring how ggplot works we will be using the **mpg** data set included in the **tidyverse** package. Type **?mpg** to see information about this dataset. It contains fuel economy data for cars from the years 1999 and 2008. Here is a summary of some selected variables (after converting some variables to factors):

```
##  manufacturer          model               displ          year      cyl
##  Length:234         Length:234         Min.   :1.600   1999:117   4:81
##  Class :character   Class :character   1st Qu.:2.400   2008:117   5: 4
##  Mode  :character   Mode  :character   Median :3.300              6:79
##                                        Mean   :3.472              8:70
##                                        3rd Qu.:4.600
##                                        Max.   :7.000
##
##  drv           cty              hwy          fl              class
##  4:103   Min.   : 9.00   Min.   :12.00   c:  1   2seater   : 5
##  f:106   1st Qu.:14.00   1st Qu.:18.00   d:  5   compact   :47
##  r: 25   Median :17.00   Median :24.00   e:  8   midsize   :41
##          Mean   :16.86   Mean   :23.44   p: 52   minivan   :11
##          3rd Qu.:19.00   3rd Qu.:27.00   r:168   pickup    :33
##          Max.   :35.00   Max.   :44.00           subcompact:35
##                                                  suv       :62
```

Scatterplots

Ploting engine displacement (engine size) versus miles per gallon on the highway (fuel efficiency). We see there is a negative relationship between engine size and fuel efficiency, which almost seems to follow a linear trend:

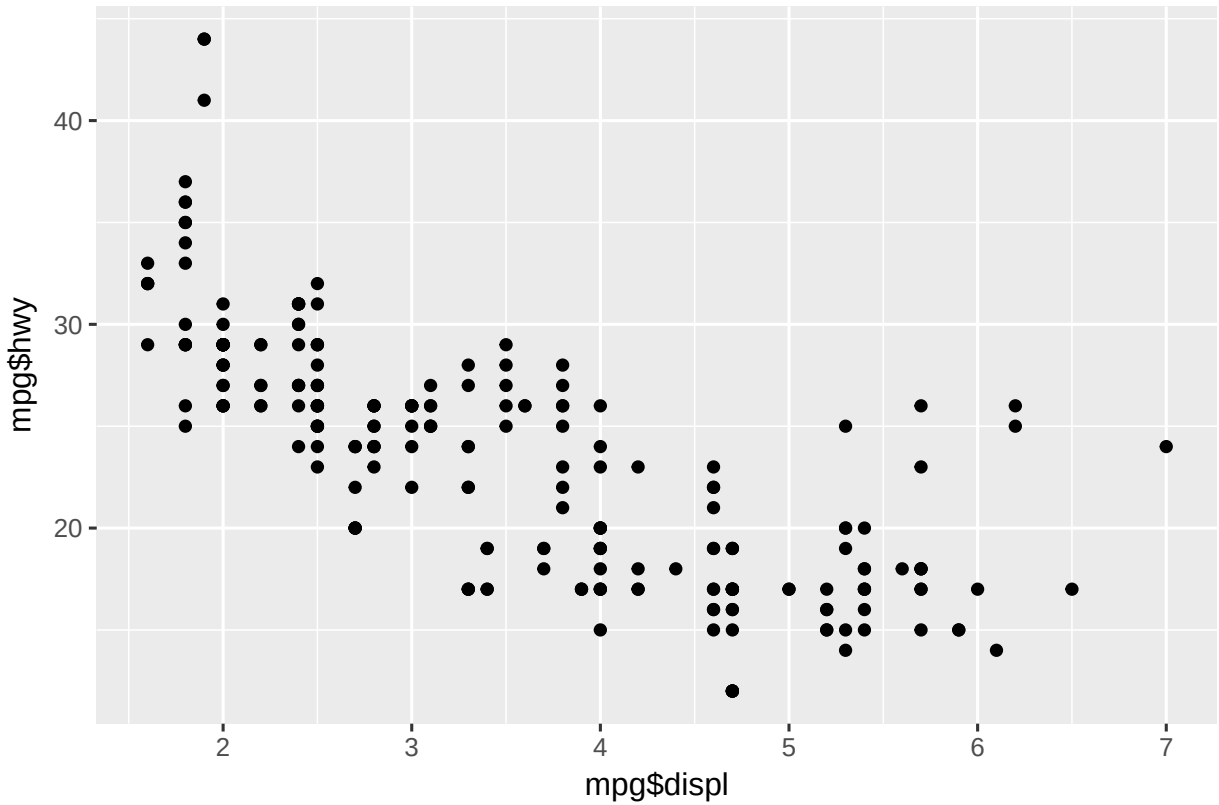
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy))
```



The Default Data Set

Specifying a data set inside the ggplot() call, allows refering to the variables of the data set using only their names in all subsequent layers related to this ggplot. If the default data set is not specified, we need to refer to the variables using the data set name:

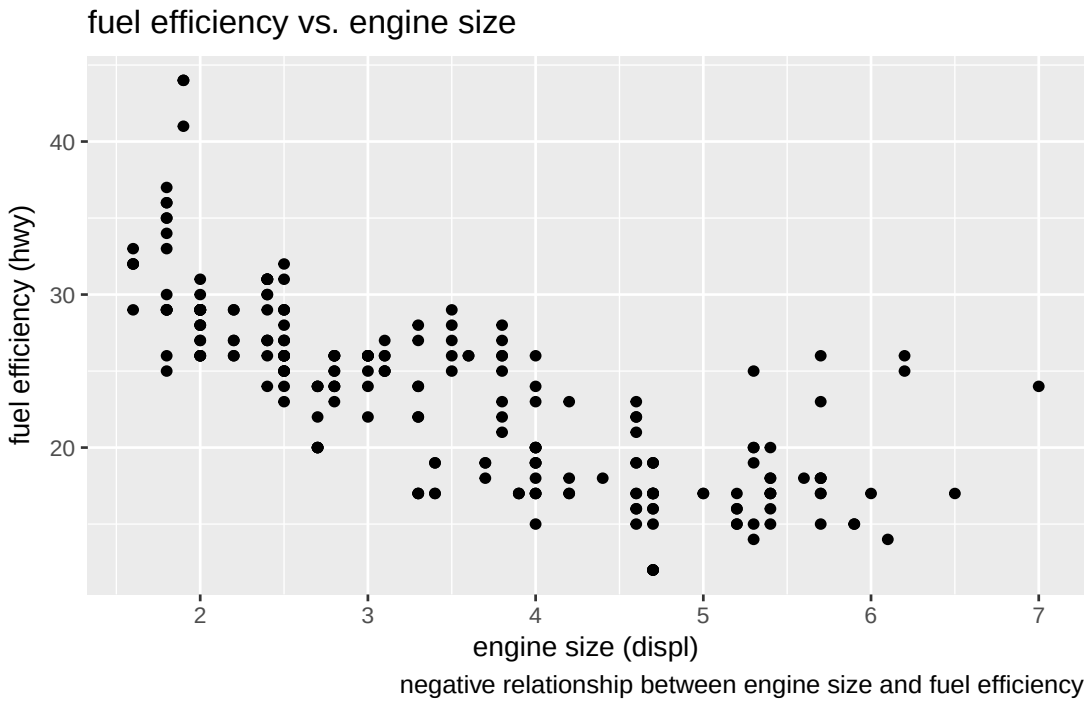
```
ggplot() +
  geom_point(mapping = aes(x = mpg$displ, y = mpg$hwy))
```



Setting Labels

By default, the axis labels are set using the variable names that were mapped to x and y. To set different values to the axis labels, as well as to set a title and a caption to the figure, we add another layer to the ggplot called **labs**:

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  labs(x='engine size (displ)', y='fuel efficiency (hwy)',
       title='fuel efficiency vs. engine size',
       caption='negative relationship between engine size and fuel efficiency')
```



Notes on Syntax

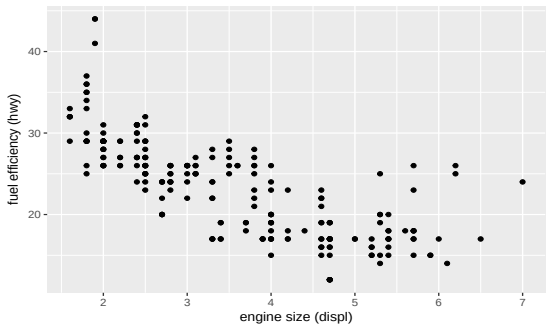
If breaking the code with the layers of a ggplot into several rows, the '+' character that connects the layers must appear at the end of the row before the break and not at the beginning of the row after the break, or an error occur:

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

```
## Error in +geom_point(mapping = aes(x = displ, y = hwy)): invalid argument to
```

It is possible to build a ggplot in several steps by assigning the plot to a variable and adding layers to this variable. The plot wont be presented until the variable is called:

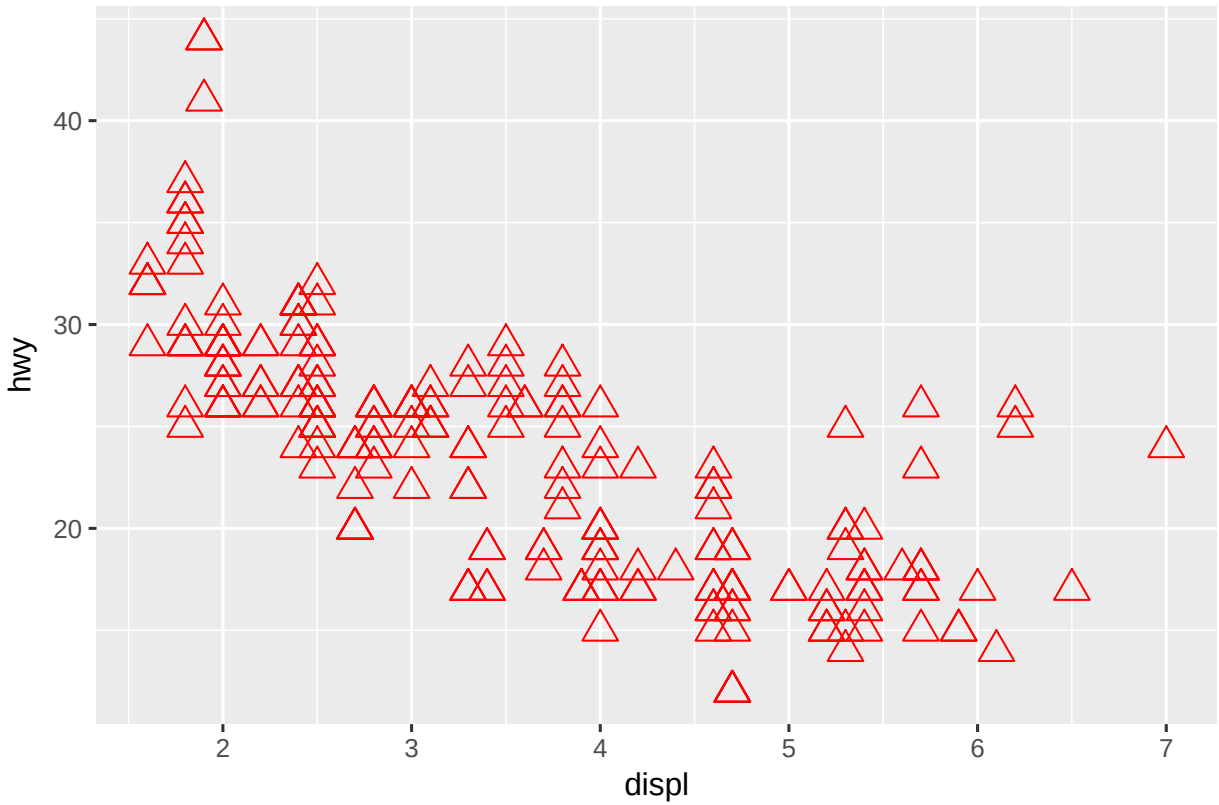
```
fig <- ggplot(data = mpg) + geom_point(mapping = aes(x = displ, y = hwy))
fig <- fig + labs(x='engine size (displ)', y='fuel efficiency (hwy)')
fig
```



Aesthetics

It is possible to set the visual properties (aesthetics) of a geom object such as color, shape and size. Each type of geom object has a different list of aesthetics that can be manipulated (e.g., a shape property is relevant for geom_point but not for geom_line).

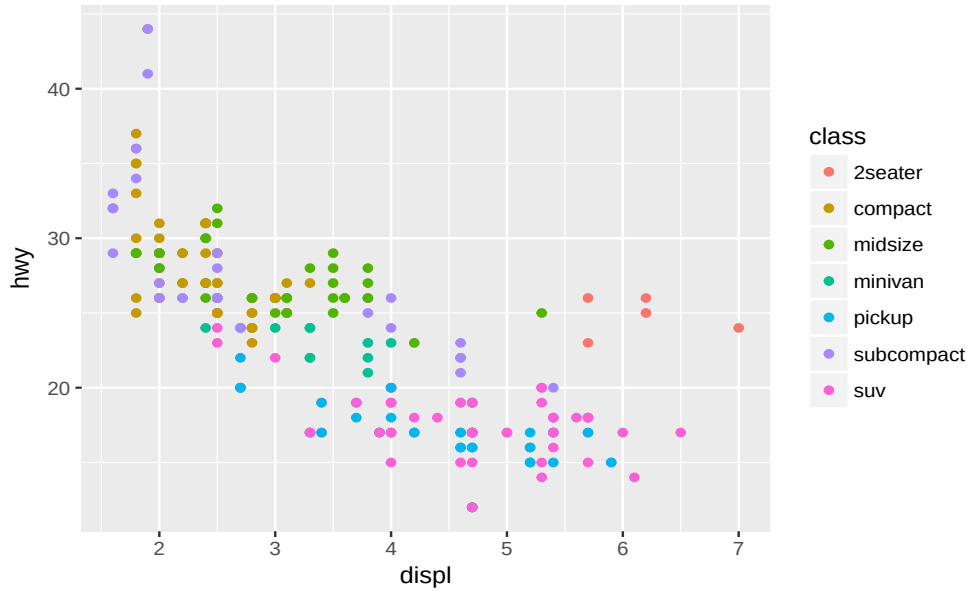
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), col='red', shape=2, size=4)
```



Aesthetic Mapping

Instead of setting a fixed value to aesthetics as in the previous example, we can map the aesthetics to variables from the data, thus adding another layer of information to the graph. Here, we add a mapping between the class variable (type of car) and the color of the points. Such mappings must be added within the aes() part. ggplot automatically assigns a color to each possible class value and adds a legend to the figure:

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



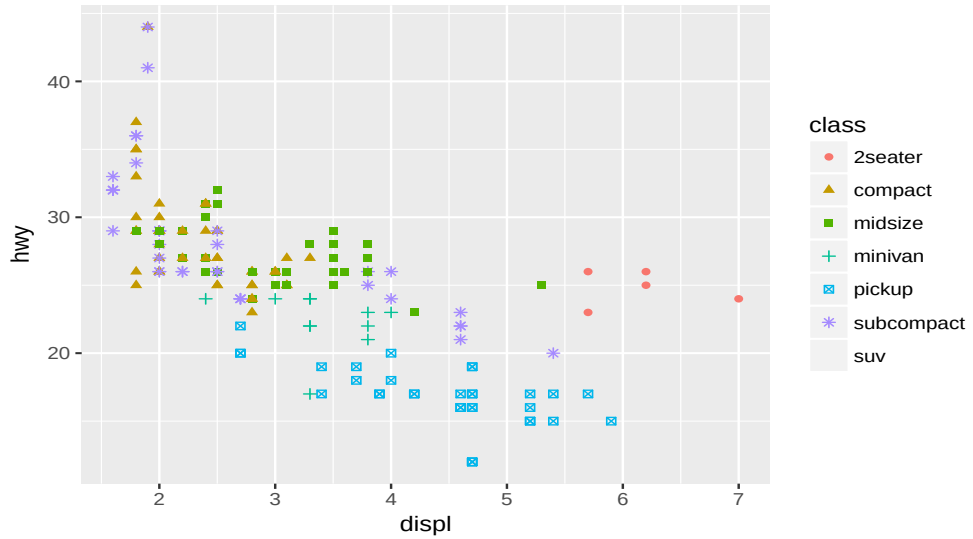
With this mapping we see that the points at bottom right which are less consistent with the linear trend, are 2-seaters. These are sport cars, with big engines but small frame, which improves their fuel efficiency compared to other cars with similar engine size.

Aesthetic Mapping

It is possible to map more than one aesthetics to the same variable. Here we attempt to add a mapping between the class variable and the shape property. However, we get a warning since, by default, there are only 6 available shapes while there are 7 class types. Points with the 7th class type (SUV) are omitted from the graph:

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

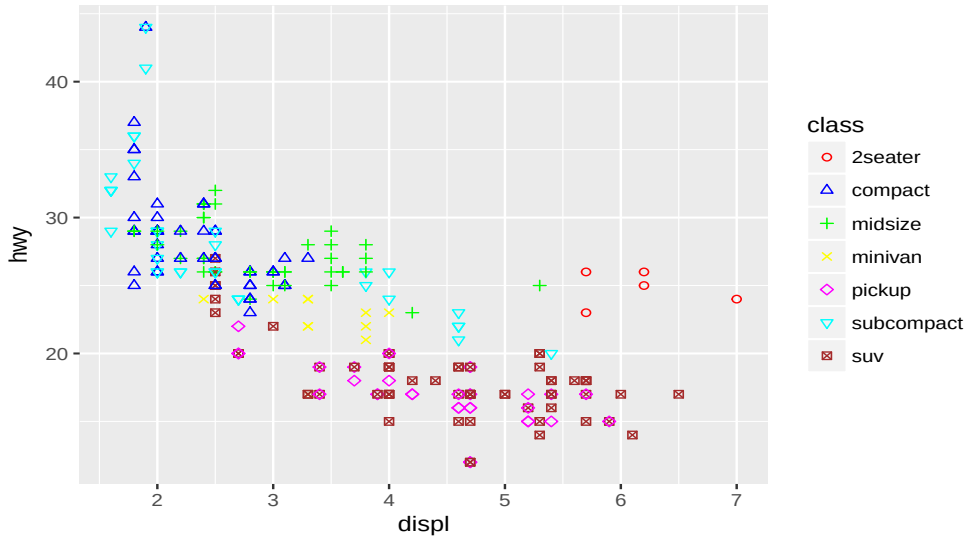
```
## Warning: The shape palette can deal with a maximum of 6 discrete values
## because more than 6 becomes difficult to discriminate; you have 7.
## Consider specifying shapes manually if you must have them.

## Warning: Removed 62 rows containing missing values (geom_point).
```



Aesthetic Mapping - Manually Setting the Scale

It is possible to solve the problem from the previous example by manually setting the shapes to use according to the number of class types. In the same manner we can set manually the colors to be used:

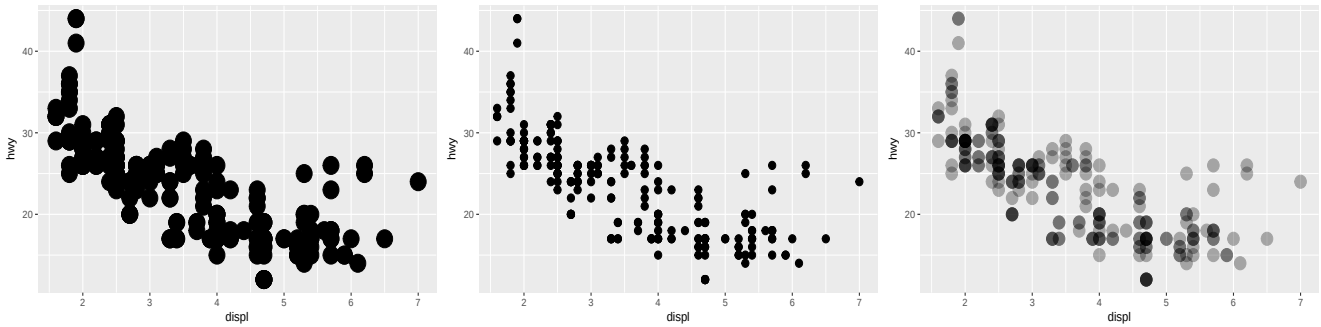
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color= class, shape = class)) +
  scale_shape_manual(values=1:nlevels(mpg$class)) +
  scale_color_manual(
    values=c("red","blue","green","yellow","magenta","cyan","brown"))
```



Avoiding Overplotting Using the Size and Alpha Aesthetics

Overplotting is a situation where several points are overlapping, so we cannot tell how many points are plotted at each position. This could happen when the **size** aesthetic value is set too high. In this case, the solution would be to reduce the **size** and/or change the value of the **alpha** aesthetic which controls the level of transparency (1 being not transparent and 0 completely transparent).

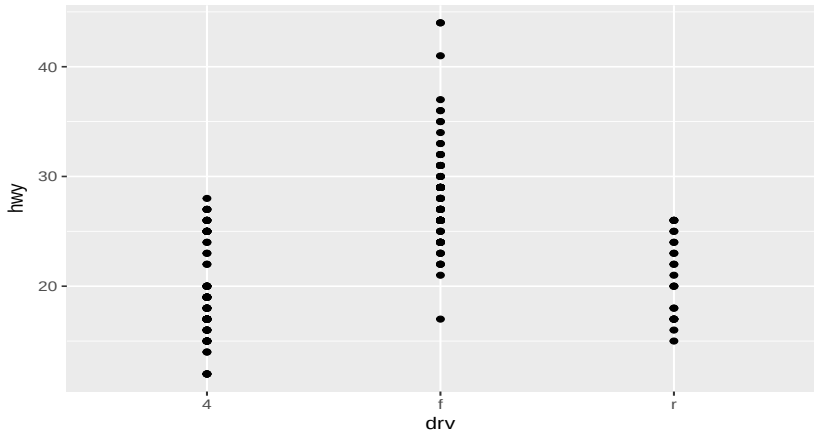
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), size = 7)
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), size = 3)
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy), size = 5, alpha=0.3)
```

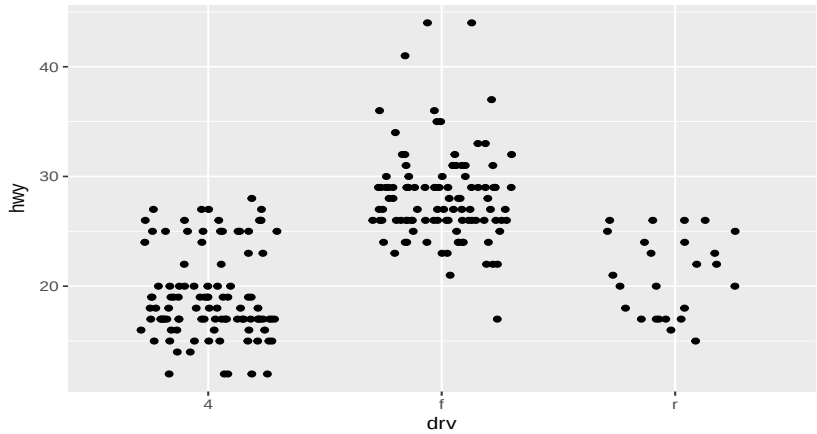


Avoiding Overplotting Using the Jitter Position Adjustment

In a scatterplot where one or both of the variables are categorical, there could be many points right on top of each other. In this case, reducing **size** and **alpha** won't help to avoid overplotting. An alternative solution is to use a jitter position adjustment. Each geom object has a **position** argument that controls the position adjustments for the geom. The default **position** value for **geom_point** is **position_identity** which means each point is plotted exactly at its <x,y> values. Setting the **position** to **position_jitter** adds small random noise to the positioning of each point so instead of having all points with the same <x,y> values plotted on top of each other, they are plotted next to each other around their <x,y> value. The resulting plot is called a stripplot.

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = drv, y = hwy))
ggplot(data = mpg) +
  geom_point(mapping = aes(x = drv, y = hwy),
             position = position_jitter(width=0.3,height=0))
```

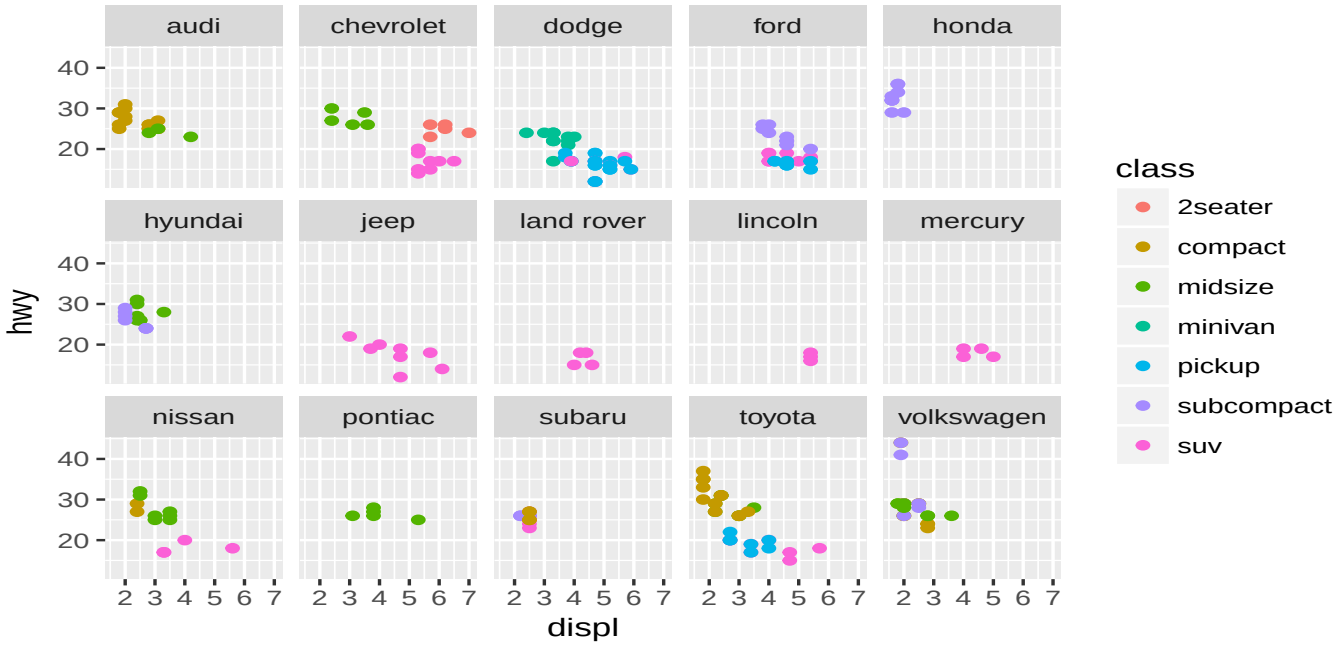




Facets - Subplotting According to a Single Variable

Facets are a way to split the figure into sub-plots, with each sub-plot displaying a subset of the data. To facet a plot according to a single variable - add another layer to the ggplot using **facet_wrap()**. Faceting the previous scatterplot according to the variable **manufacturer** creates a sub-plot for each manufacturer:

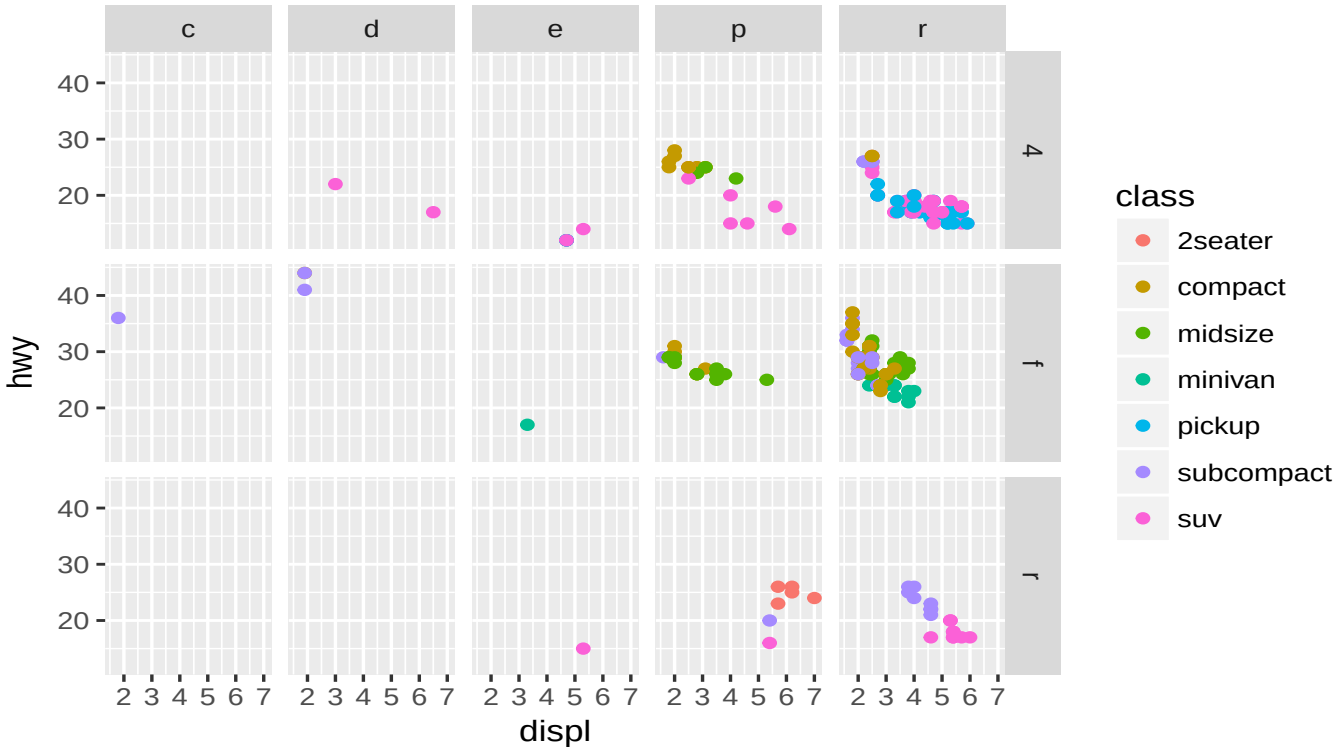
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color=class)) +
  facet_wrap(~ manufacturer, nrow=3)
```



Facets - Subplotting According to Two Variables

To facet a plot according to two variables use **facet_grid()**. Facetting according to the variables **drv** (drive type) and **fl** (fuel type) creates a subplot for each pair of <**drv,fl**> values. If there is no data with a given pair of values, the related sub-plot will be empty:

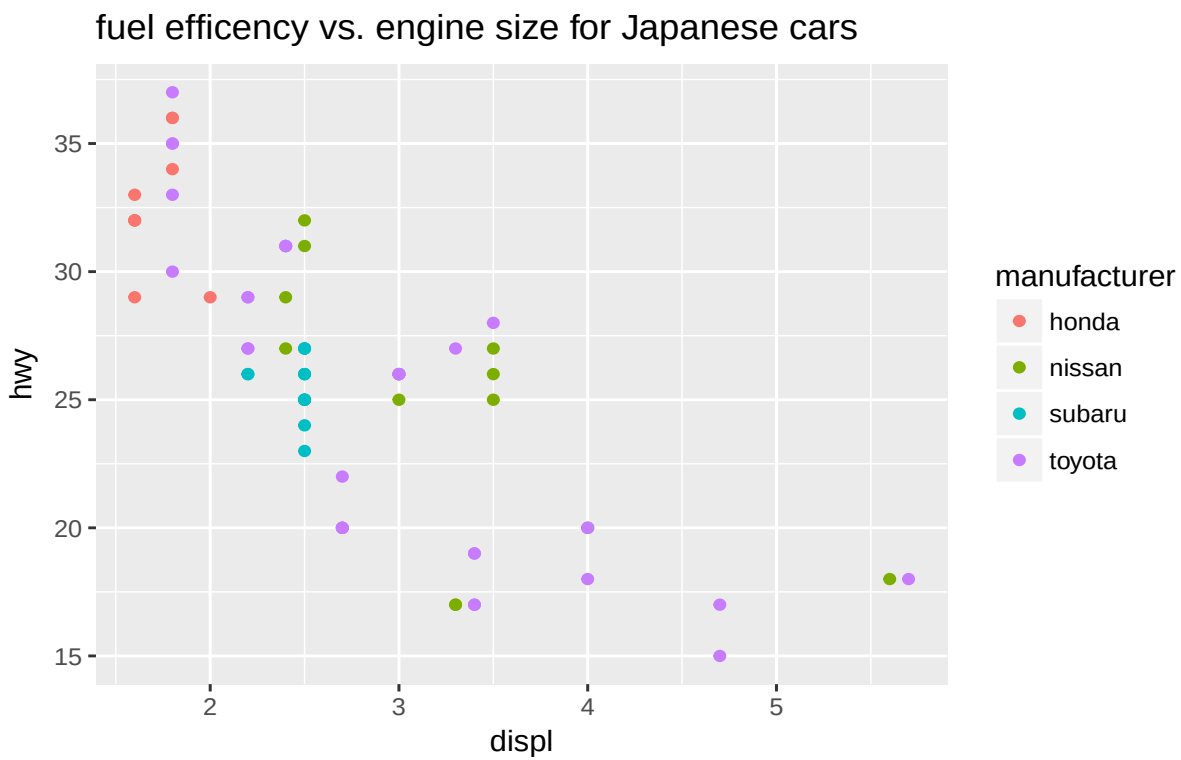
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color=class)) +
  facet_grid(drv ~ fl)
```



Plotting a Subset of the Data

What if we want to plot just a subset of the data by itself? We can use the subset() function (see 'Introduction to R' slides) to select the subset we are interested in, and use this subset as the data given to ggplot:

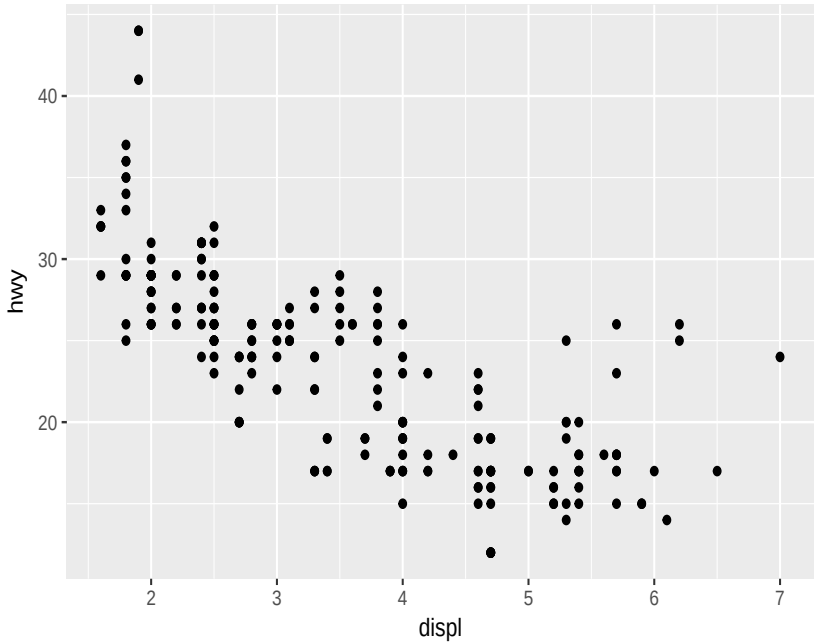
```
japanese_cars <- c('honda','nissan','subaru','toyota')
mpg_japanese <- subset(mpg,subset=mpg$manufacturer %in% japanese_cars)
ggplot(data = mpg_japanese) +
  geom_point(mapping = aes(x = displ, y = hwy, color = manufacturer)) +
  labs(title='fuel efficency vs. engine size for Japanese cars')
```

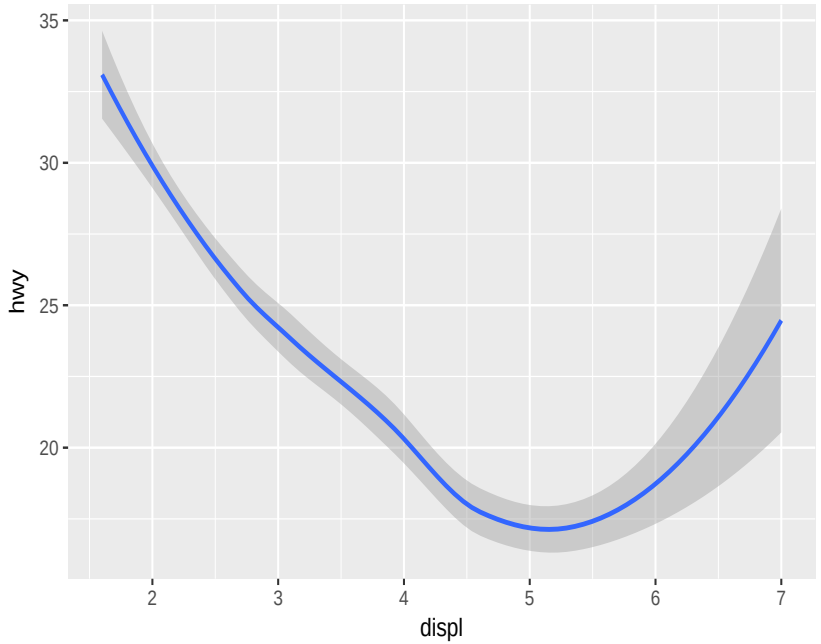


The Smooth Geom

So far we used only one geomertical object available in the ggplot2 library - **geom_point** which creates a scatterplot. **geom_smooth** is another geom that plots a line obtained from a smoothing of the data given to it, with or without a shaded area signifying the confidence intervals around the smoothed line. It is helpful in seeing patterns in the data. Here we see the same data, plotted using the two different geom objects, side by side:

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

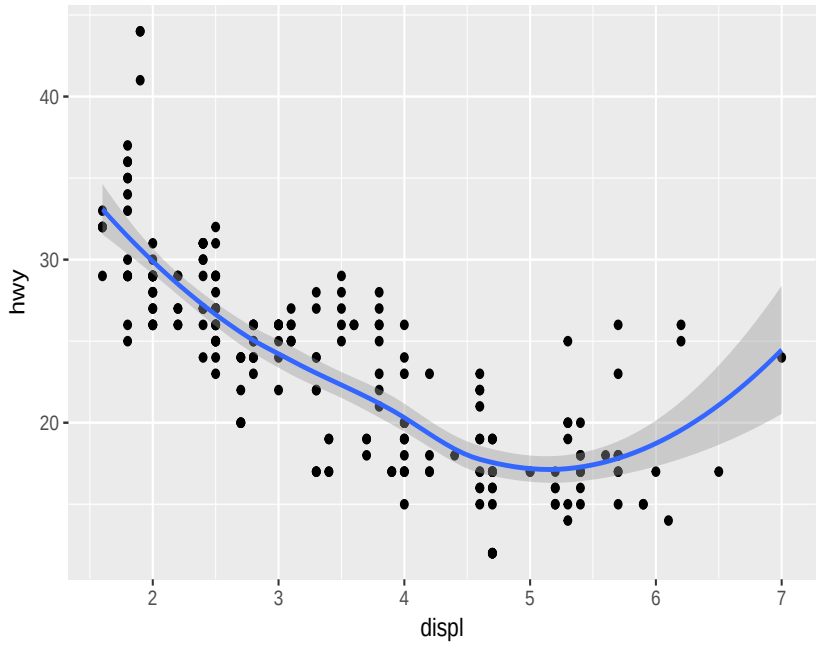




Plotting Geoms on Top of Each Other

It is possible and very common that we would want to plot several geom objects on top of each other on the same plot. To do this we just add the geom we want as multiple layers to the ggplot. The **geom_smooth** object, in particular, is usually something we would like to add on top of a scatterplot:

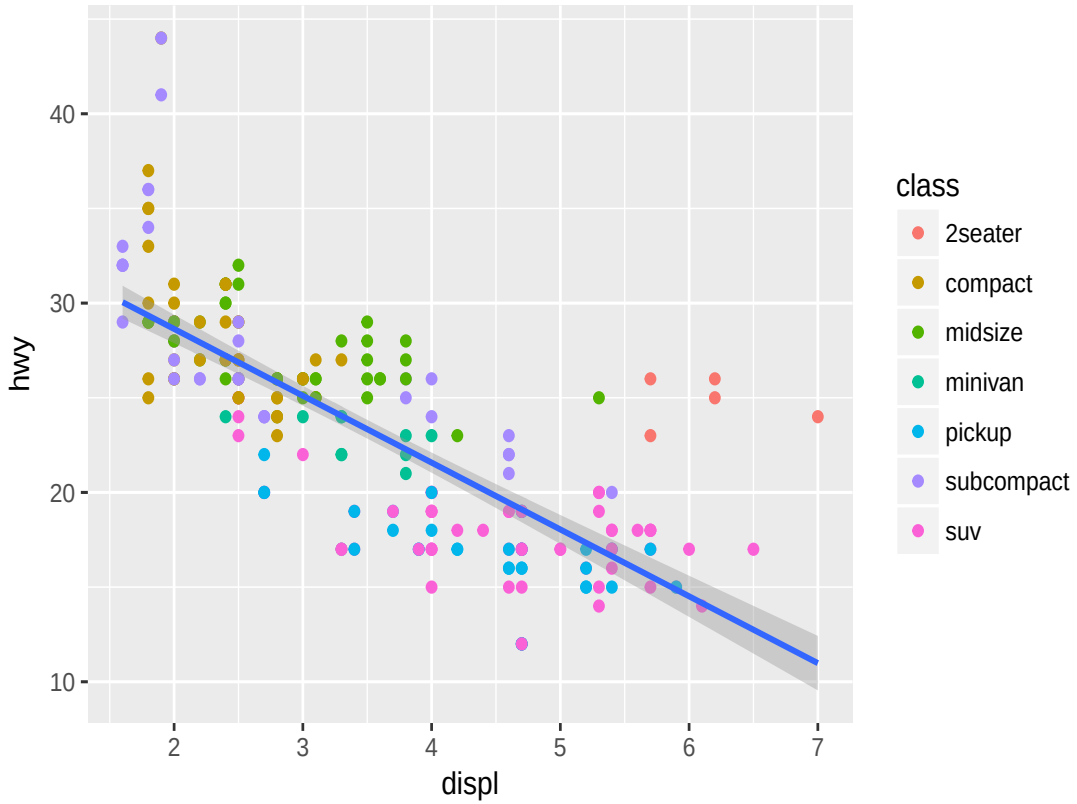
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(x = displ, y = hwy), method='loess')
```



Using the Smooth Geom for Presenting a Linear Regression Line

geom_smooth can also be used to present a linear regression line, using the 'lm' method option. We will get more into the fitting linear models and the **lm()** function later in the course.

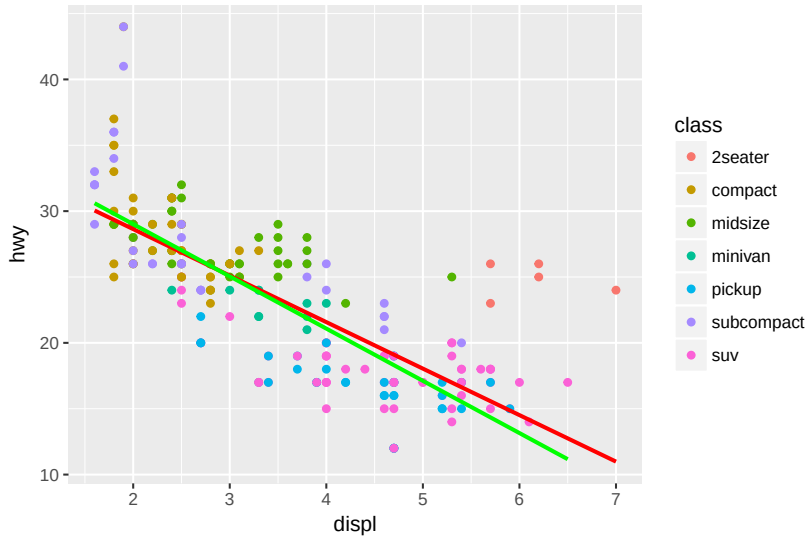
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color=class)) +
  geom_smooth(mapping = aes(x = displ, y = hwy), method='lm')
```



Plotting Geom Objects using Different Data on the Same Plot

In the previous slide, we can see again that the 2-seater cars (sport cars) are outliers from the linear trend. What if we want to check the effect of removing these cars on the obtained regression line? We can create a subset of the data that doesn't include these cars and add another **geom_smooth** object that operates on this subset by specifying the subset as the data to use inside the geom function. This will override the data declaration given inside ggplot() for this specific object:

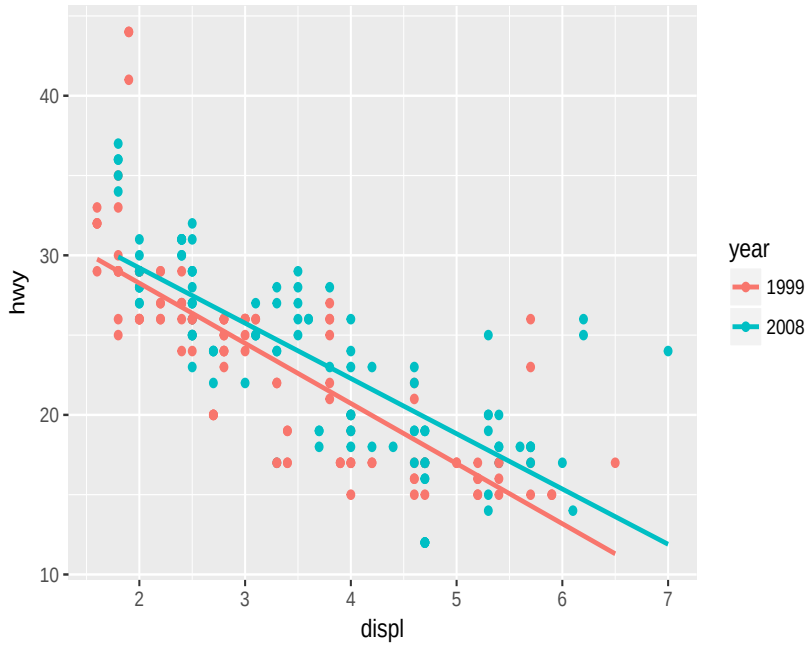
```
mpg_no2s <- subset(mpg,subset=mpg$class!='2seater')
ggplot(data = mpg) +
  geom_point(mapping = aes(x =displ, y=hwy, color=class)) +
  geom_smooth(mapping = aes(x=displ, y=hwy), method='lm', color='red', se=F) +
  geom_smooth(data = mpg_no2s, mapping = aes(x=displ, y=hwy),
              method='lm', color='green', se=F)
```



Aethetic Mapping with Multiple Geom Objects

Each geom object can have its own aethtetic mapping. In this example, we map the color of the two objects to the same variable **year** (either 1999 or 2008). Mapping the color to the variable **year** in **geom_smooth** generates a smoothed line for each value of year. Since the regression line for the year 2008 is always higher than that obtained for 1999, we can deduce that on average, the fuel efficiency of cars from 2008 have improved since 1999, across all engine sizes:

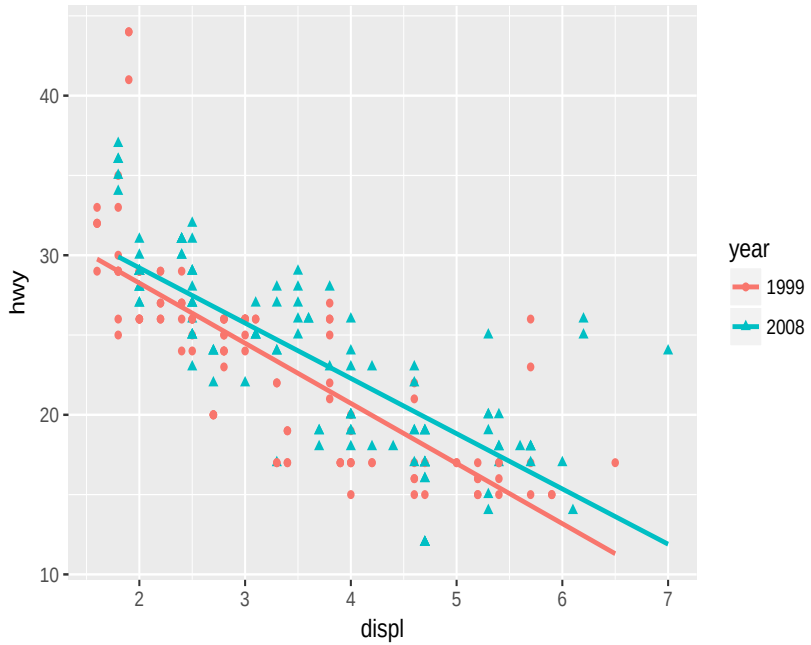
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color=year)) +
  geom_smooth(mapping = aes(x = displ, y = hwy, color=year), method='lm', se=F)
```



Default Aesthetic Mapping for a Plot

In the previous example, we used the same aesthetic mapping for both objects. We can create the same plot with less code by declaring the aesthetic mapping inside the ggplot() call and removing it from both geom object functions. The aesthetic mapping inside the ggplot() will be the default mapping used for all geom objects belonging to this plot. Any additional aesthetic mapping specified inside a geom object will override or enhance the default mapping for that particular object:

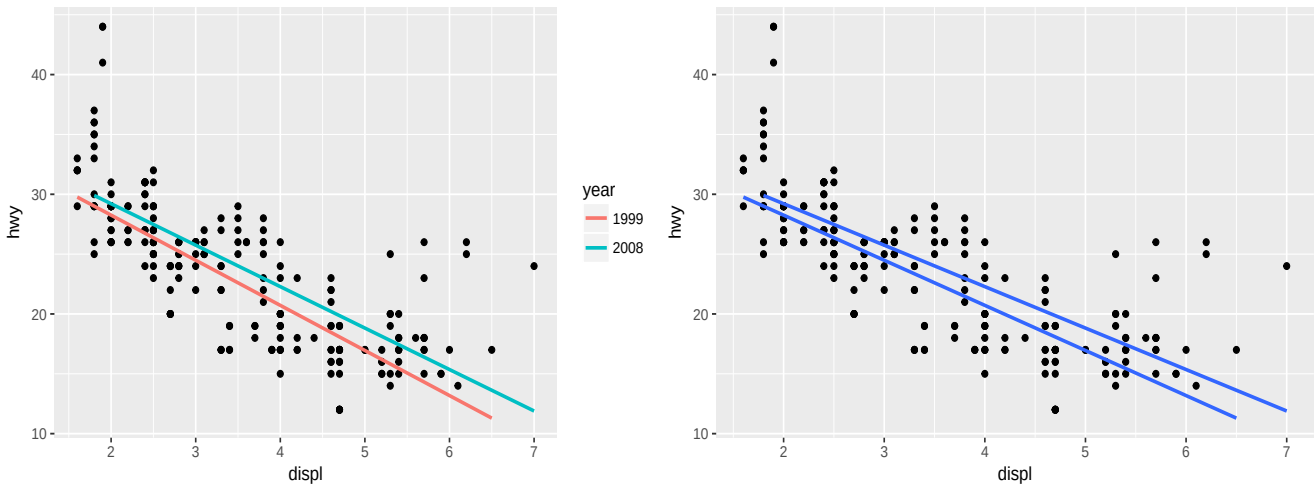
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy, color=year)) +
  geom_point(mapping = aes(shape = year)) +
  geom_smooth(method='lm', se=F)
```



The group Aesthetic

For geom objects like geom_smooth that use a single geometric object to represent multiple rows of data (multiple observations), there is an available aesthetic called **group** that can be mapped to a variable in order to create multiple objects for each possible value of the variable. It will have the same effect as mapping another aesthetic (such as color) to the variable, only that using **group** - all the created objects will have the same properties (and not a different color as when mapping color), and there will be no legend added to the figure for this mapping.

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(aes(color = year), method='lm', se=F)
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(mapping = aes(group = year), method='lm', se=F)
```



The diamonds Data Set

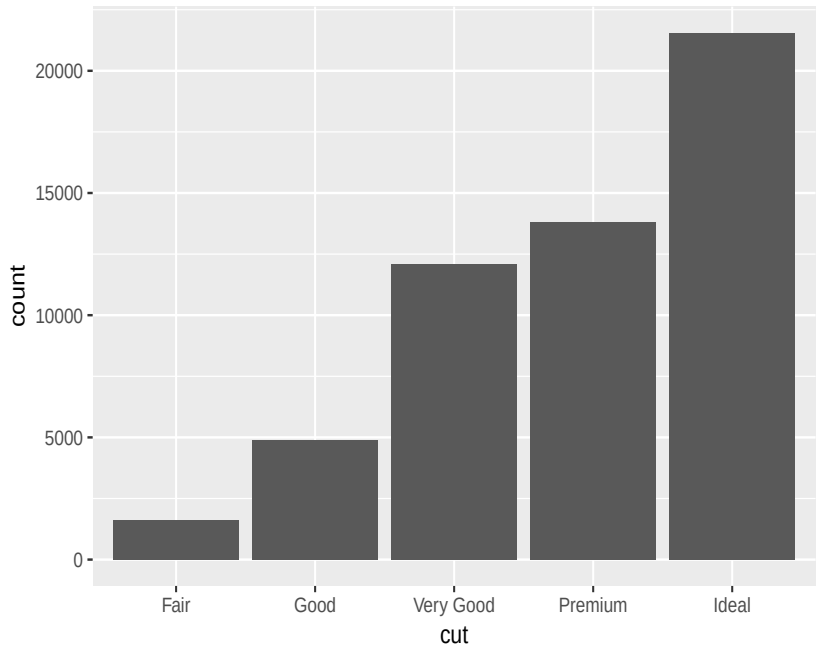
For our next examples we will be using another data set included in the **tidyverse** package called **diamonds**, which contains the prices and other attributes of ~54,000 diamonds. Here is a summary of some selected variables:

```
##      carat               cut        color        clarity
##  Min.   :0.2000   Fair      : 1610   D: 6775   SI1     :13065
##  1st Qu.:0.4000   Good      : 4906   E: 9797   VS2     :12258
##  Median :0.7000   Very Good:12082   F: 9542   SI2     : 9194
##  Mean   :0.7979   Premium   :13791   G:11292   VS1     : 8171
##  3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2    : 5066
##  Max.   :5.0100                     I: 5422   VVS1    : 3655
##                                     J: 2808   (Other): 2531
##      depth             x                y                z
##  Min.   :43.00   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.:61.00   1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
##  Median :61.80   Median : 5.700   Median : 5.710   Median : 3.530
##  Mean   :61.75   Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
##  3rd Qu.:62.50   3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
##  Max.   :79.00   Max.   :10.740   Max.   :58.900   Max.   :31.800
##
```

Bar Charts

To create a bar chart we use a new geom object called **geom_bar**. This bar chart shows the number of diamonds in the data from each quality of cut. It shows there are more high quality cut diamonds in the data than low quality cut diamonds:

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



Statistical Transformations

How did **geom_bar** created the bar chart shown in the previous slide? It calculated the number of diamonds for each **cut**, storing the results in a new variable called **count**, and then ploted the count for each cut using bars.

In ggplot terminology, it used a statistical transformation of type **stat_count**. In general, statistical transformations in ggplot, or stats, are operations performed on a data set, which are used by geom objects in the visualization of the data. Each type of geom object has its own default stat (can be seen in the help page of the geom object).

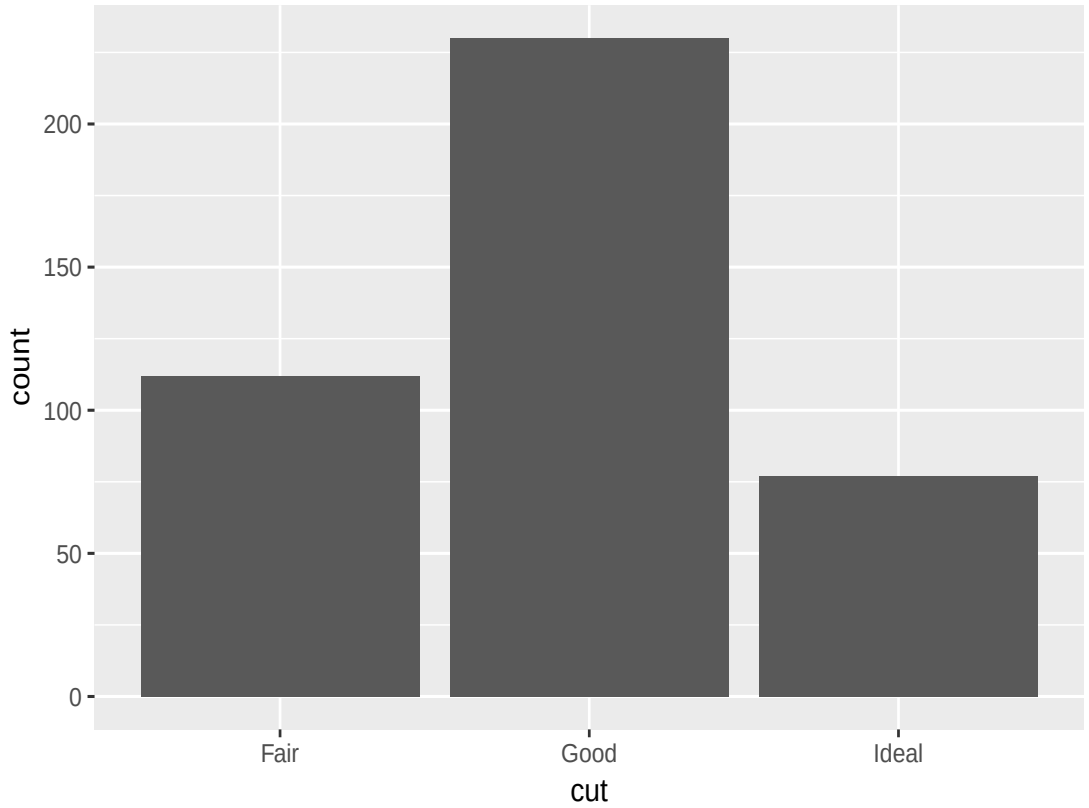
Notable types of stats include:

- stat_identity - leaves the data unchanged (default stat for geom_point/geom_line)
- stat_smooth - smoothes or fits a model to the data (default stat for geom_smooth)
- stat_count - counts the number of cases for a categorical variable (default stat for geom_bar)
- stat_bin - bins a continuous variable and counts the number of cases within each bin (default stat for geom_histogram)
- stat_boxplot - computes the distribution of a continuous variable (default stat for geom_boxplot)

Overriding the Default Stat of a Geom Object

What if we had a data set which is already summarized as count data and we wanted to plot these data as a bar chart? If we try to specify the y mapping in **geom_bar** while using the default **stat_count** we would get an error message. Instead, we need to override the default stat and set it to **stat_identity**.

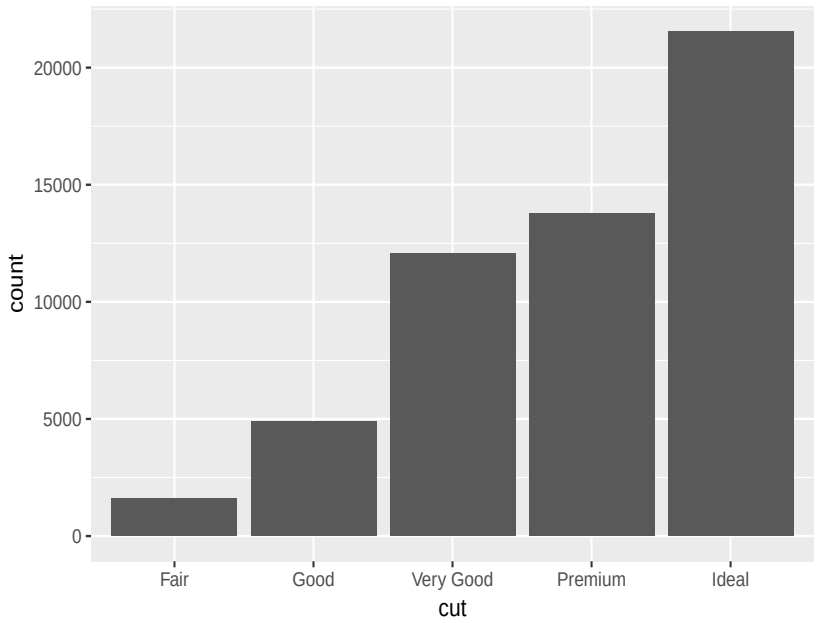
```
cut_count <- data.frame(cut=c('Fair','Good','Ideal'), count=c(112,230,77))
ggplot(data = cut_count) +
  geom_bar(mapping = aes(x = cut, y = count), stat = "identity")
```

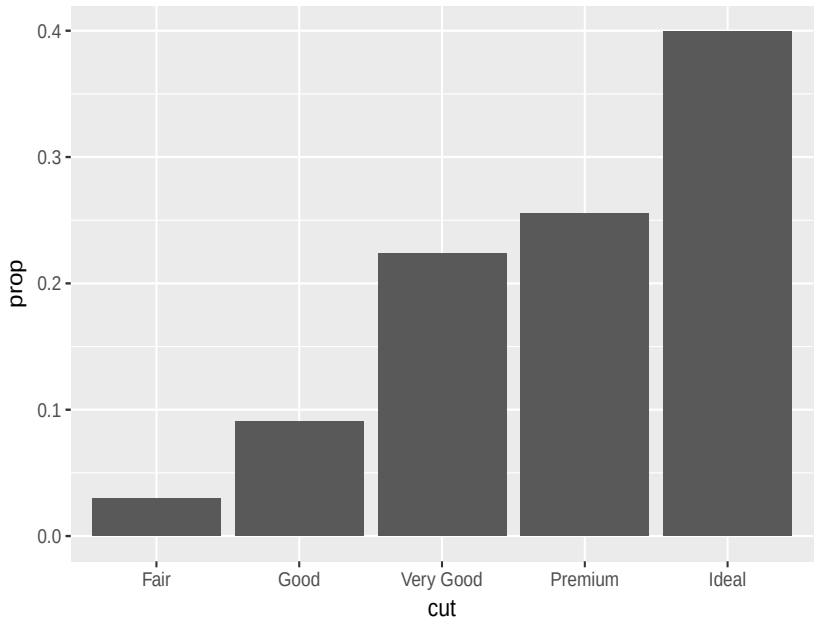


Plotting a Bar Chart of Proportions

We can plot the proportions instead of the counts of the values of a categorical variable using bar chart, by referencing the computed variable of **stat_count** called **count** (which is the default value set to y). We can refer to this value inside geom_bar using **..count..**.

```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y=..count..)) +
  labs(y='count')
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, y=..count../sum(..count..))) +
  labs(y='prop')
```

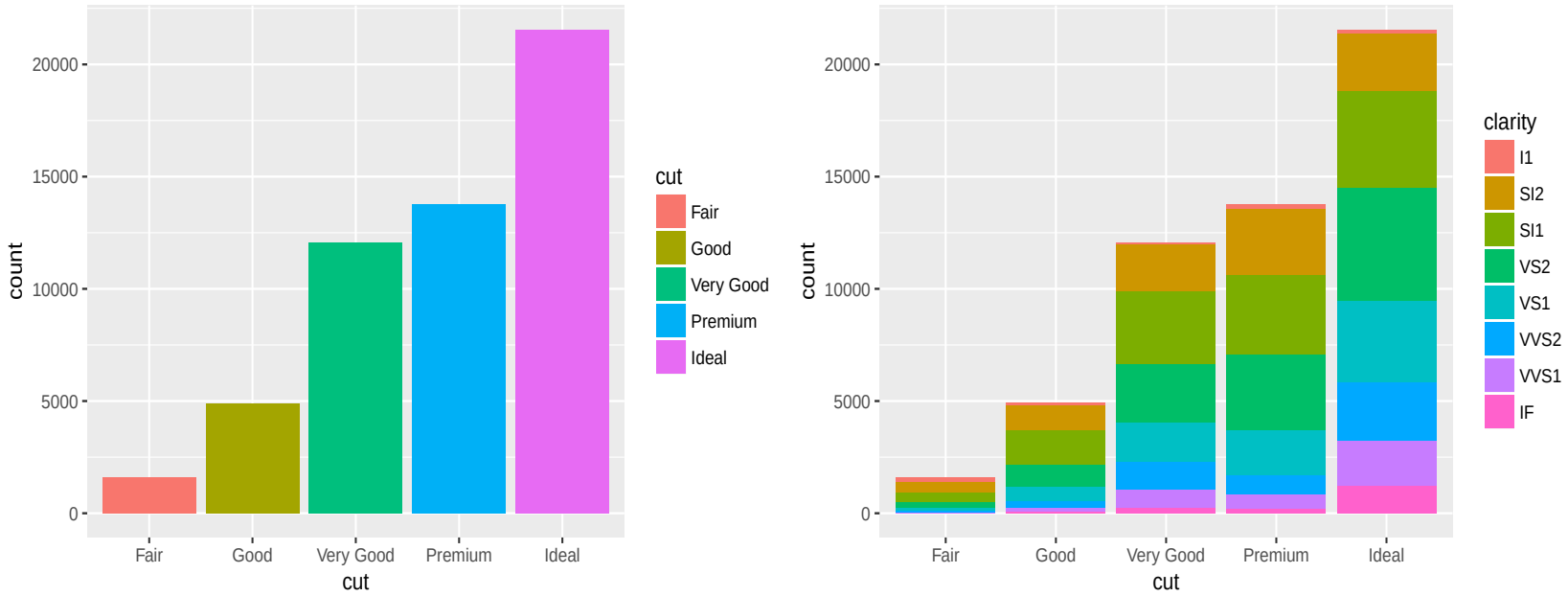




The fill Aesthetic in Bar Chart

The **fill** aesthetic in a bar chart controls the colors used for plotting the bars. If this aesthetic is mapped to the same variable used for mapping x, each bar will have its own color. However, if **fill** is mapped to a second variable, each bar will be divided into a secondary division, according to the count of the second variable values for each value of the primary division variable (variable mapped to x).

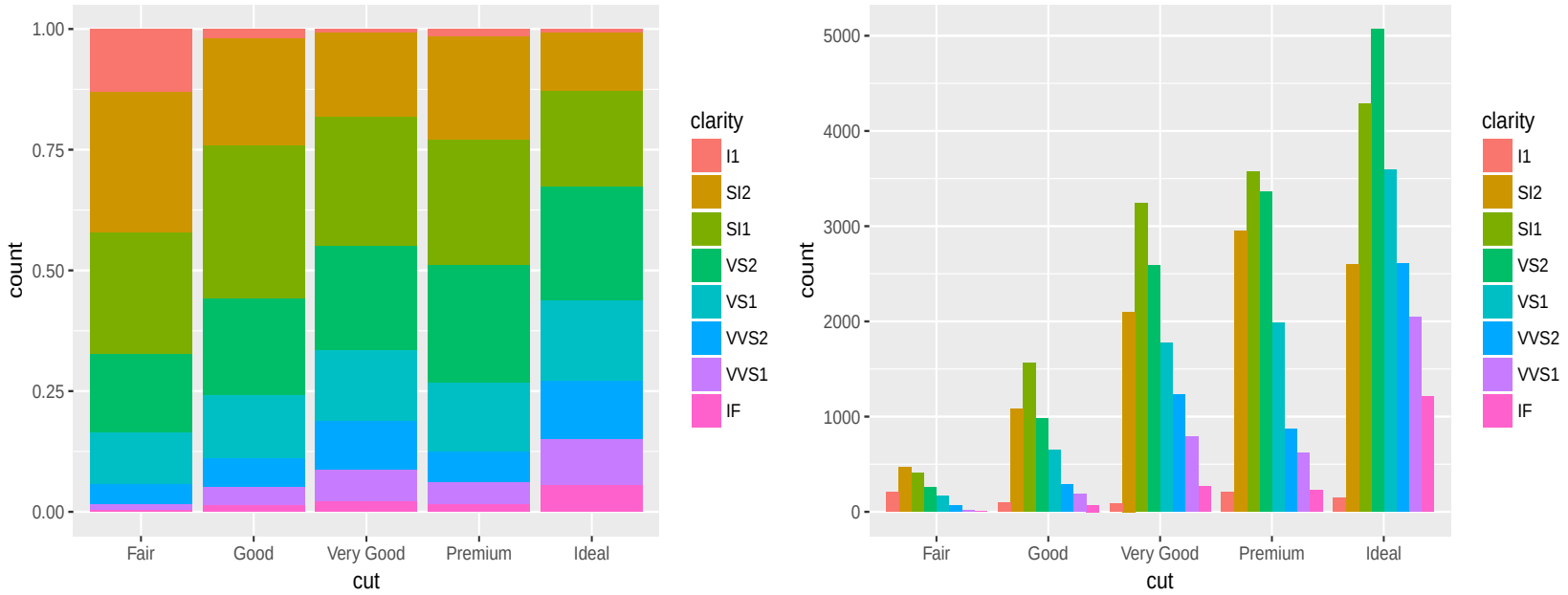
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity))
```



Position Adjustments in Bar Charts

We already encountered the **position** argument in **geom_point** when discussing the use of a "jitter" **position**. The default value for **position** in **geom_bar** is "stacked". This is why, when creating a sub-division in the bar chart, the sub-division bars were stacked on top of each other, so the bars have the same height as the bars without the sub-division. It is possible to change the appearance of the bar chart by setting a different value to the **position** argument.

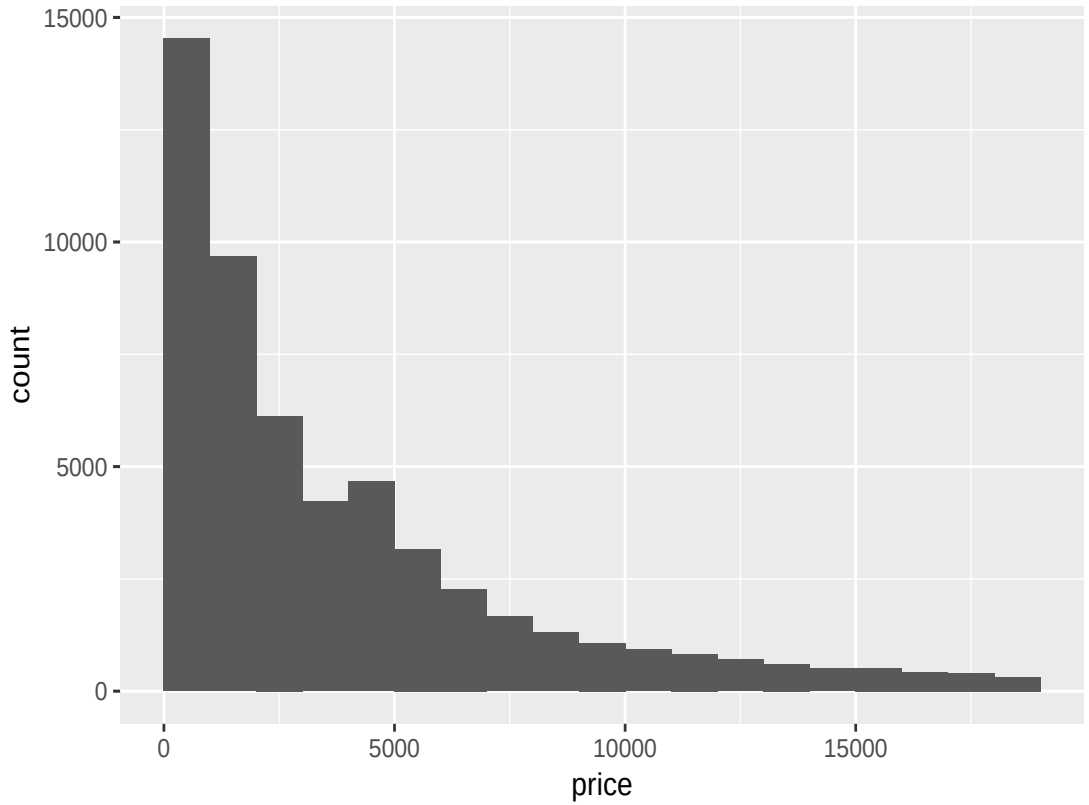
```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position="fill")
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = clarity), position="dodge")
```



Histograms

Histograms are similar to bar charts only they are used with continuous variables and not categorical variables. The variable mapped to x is binned into several bins and the histogram presents the number of appearences in the data for each bin (the distribution of the variable in the data). Here we see a histogram for the price of diamonds, showing a decline in the number of diamonds in the data, as the price gets higher.

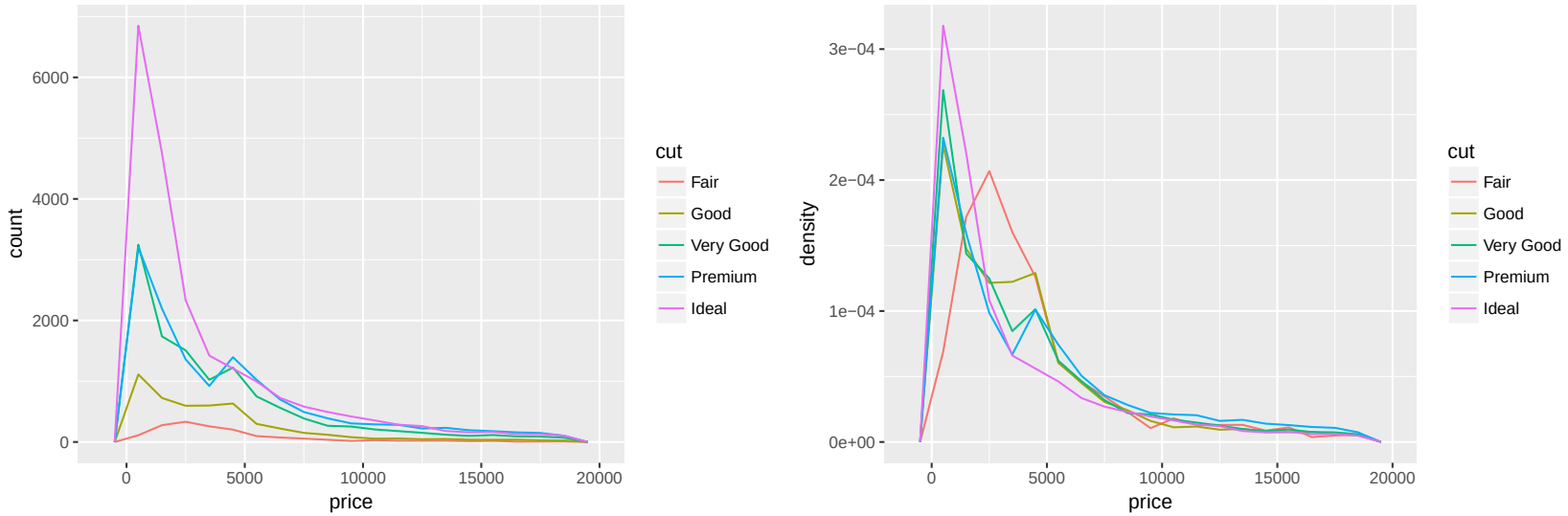
```
ggplot(data=diamonds) +
  geom_histogram(mapping=aes(x=price), binwidth=1000, center=500)
```



Multiple Histograms on the Same Plot

geom_freqpoly does the same thing as **geom_histogram** only it plots the histogram as a line instead of bars. It is helpful when wanting to display multiple histograms on top of each other. Here we see the distribution of the **price** of diamonds in the data, broken down according to the **cut** of the diamonds. Displaying the density instead of the count of each bin (using the computed variable **..density..** of **stat_bin** - the stat used by **geom_histogram** and **geom_freqpoly**), makes it easier to compare the distributions:

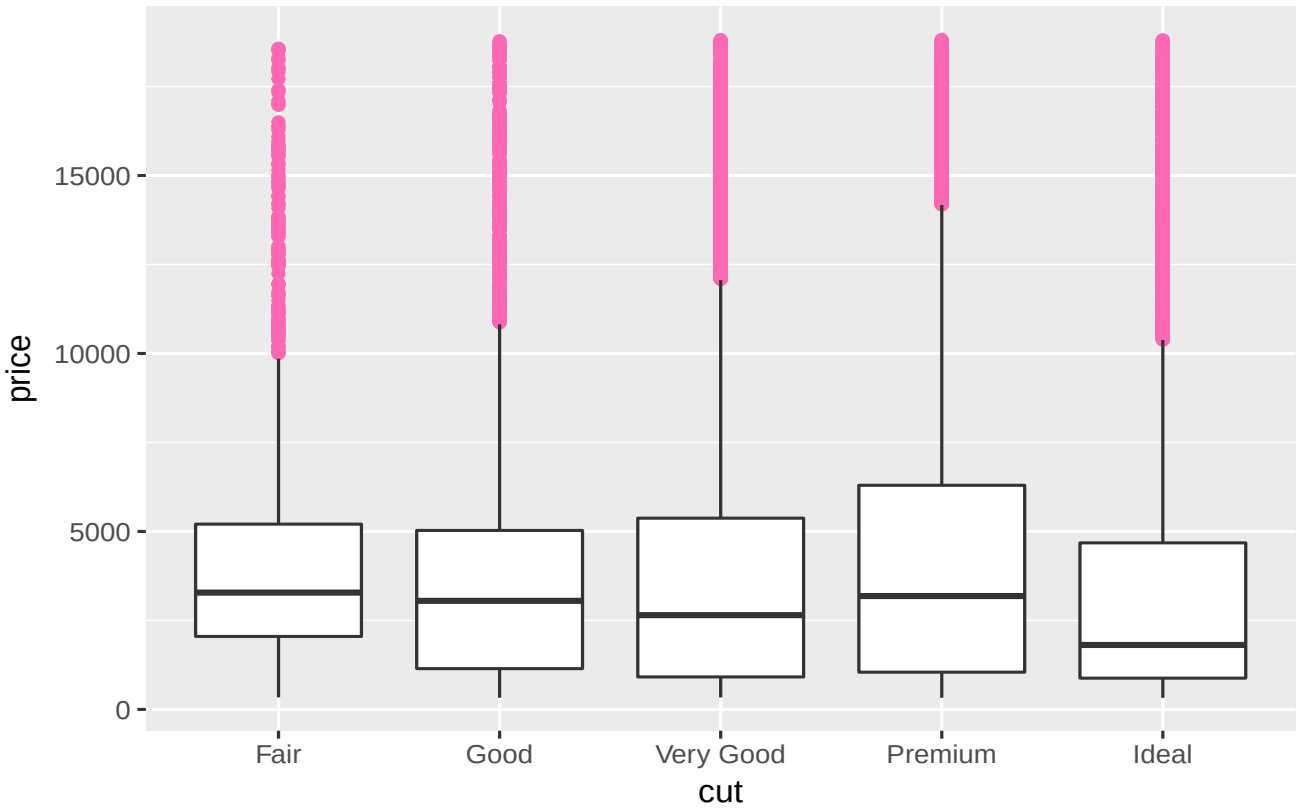
```
ggplot(data=diamonds) +
  geom_freqpoly(mapping=aes(x=price, color=cut), binwidth=1000, center=500)
ggplot(data=diamonds) +
  geom_freqpoly(
    mapping=aes(x=price, y=..density.., color=cut), binwidth=1000, center=500)
```



Boxplots

A boxplot is another way to display the distribution of a continuous variable broken down by a categorical variable (i.e., x should be mapped to a categorical variable and y to a continuous variable). Here is the data from the previous slide presented as a boxplot:

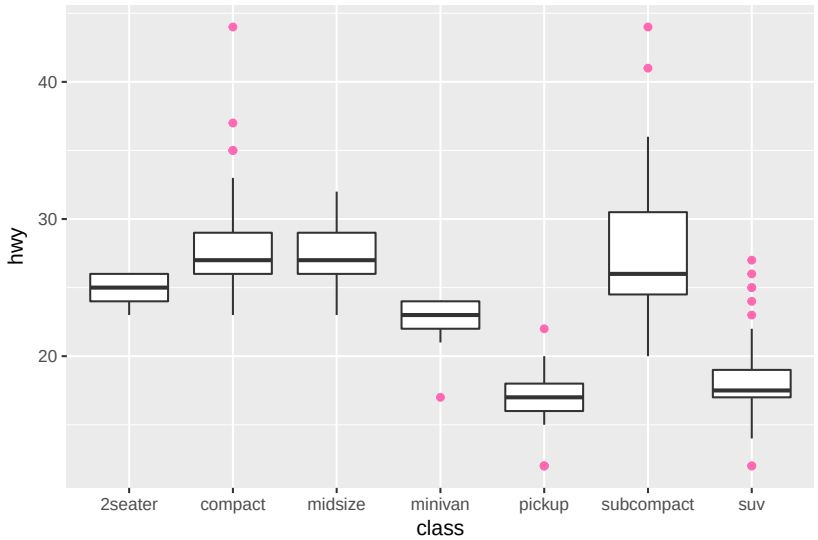
```
ggplot(data=diamonds) +
  geom_boxplot(mapping=aes(x=cut, y=price), outlier.color='hotpink')
```

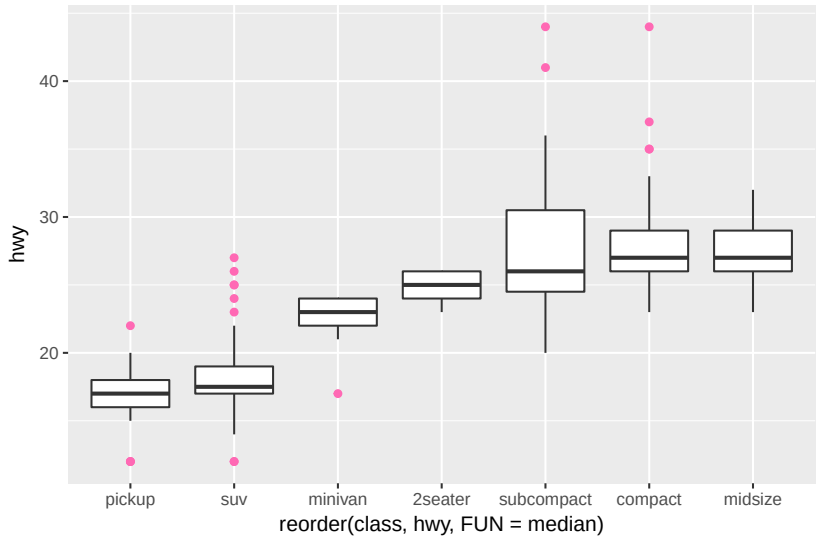


The reorder Function

Back to the mpg data set, we plot the distribution of fuel efficiency values (**hwy**) for each car type (**class**) using a boxplot. We can reorder the boxplots to see more clearly the trend in the data. To do so, we use the **reorder()** function, which reorders the levels of a categorical variable according to a function of the values of a second variable. Here we reorder the levels of the **class** variable according to the median of **hwy** for each class, so that boxplots will appear in an order of increasing median.

```
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(x = class, y = hwy), outlier.color='hotpink')
ggplot(data = mpg) +
  geom_boxplot(mapping = aes(x = reorder(class,hwy,FUN=median), y = hwy),
               outlier.color='hotpink')
```

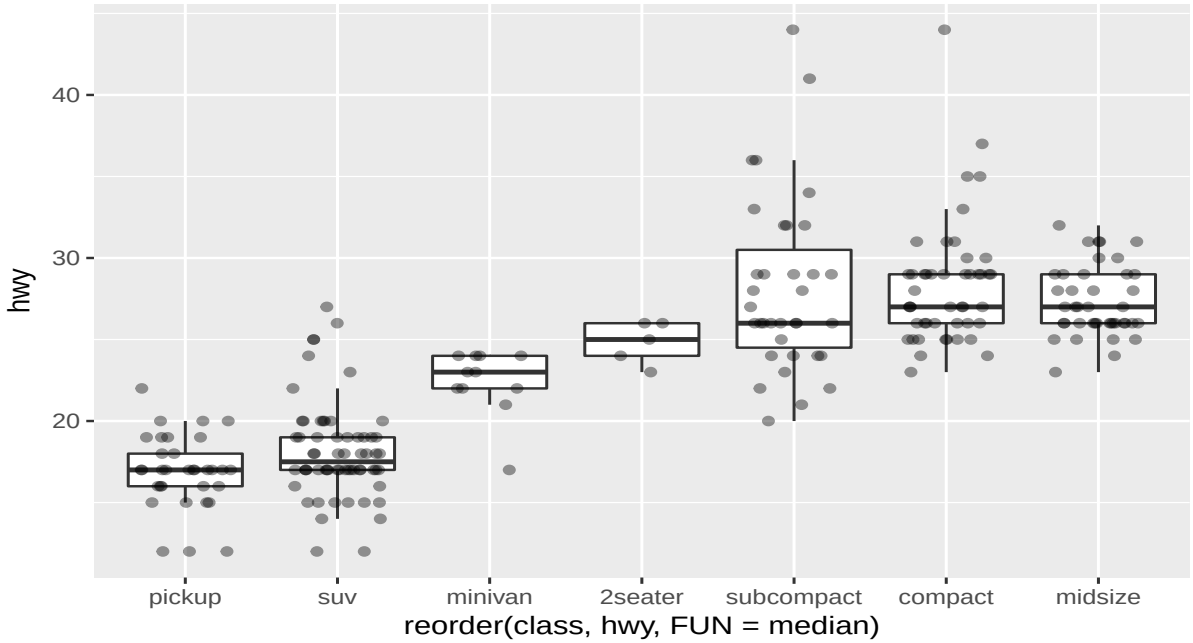




Boxplots with Stripplots

We can combine boxplot and stripplots together to present simultaniously both summary information regarding the distribution and all the data points that make up the distribution. We saw previously how to create stripplots by using **position_jitter** with **geom_point** (an alternative is to use **geom_jitter** which is similar to **geom_point** only with **position_jitter** being the default **poistion**). Here we add the stripplots to the reordered boxplots from the previous slide. Note that the aesthetic mapping was moved to the ggplot section so that it applies to both geoms:

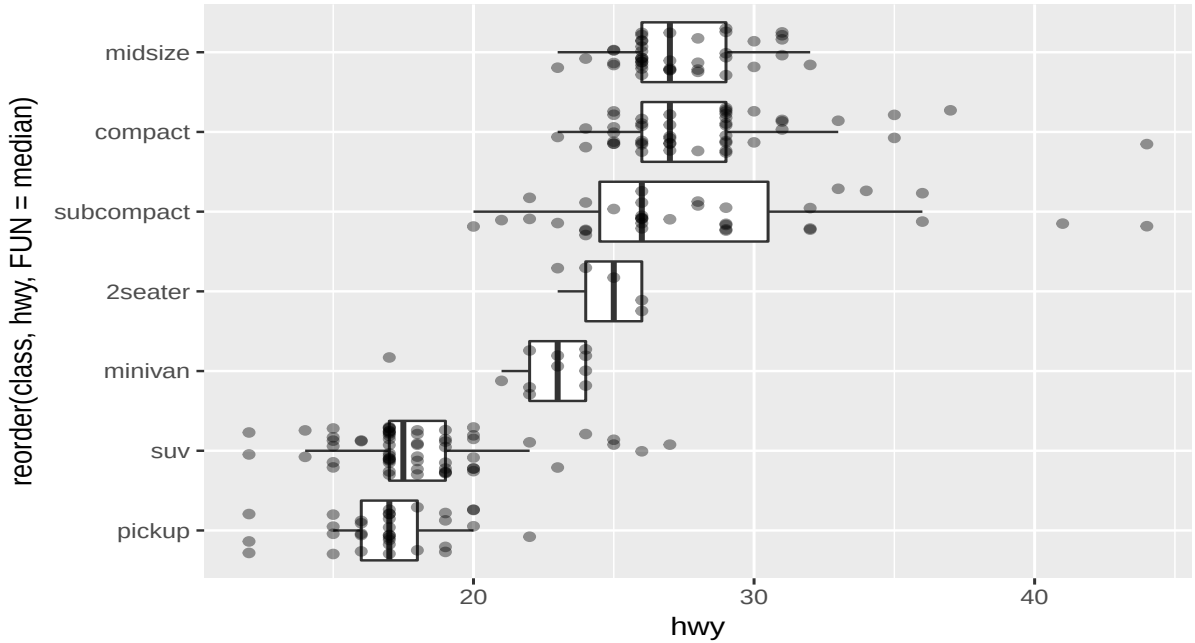
```
ggplot(data = mpg, mapping = aes(x = reorder(class,hwy,FUN=median), y = hwy)) +
  geom_boxplot(outlier.alpha=0) +
  geom_point(position = position_jitter(width=0.3,height=0), alpha=0.4)
```



Coordinate Systems

The default coordinate system in ggplot is the Cartesian coordinate system with x as the horizontal axis and y as the vertical axis. There are several functions in ggplot that can be used to manipulate the coordinate system used to present the data. The most simple manipulation is to switch between the x and y axis using the **coord_flip()** function. To do so we just add it as an another layer to the ggplot. Here is the plot from the previous slide with the coordinates flipped:

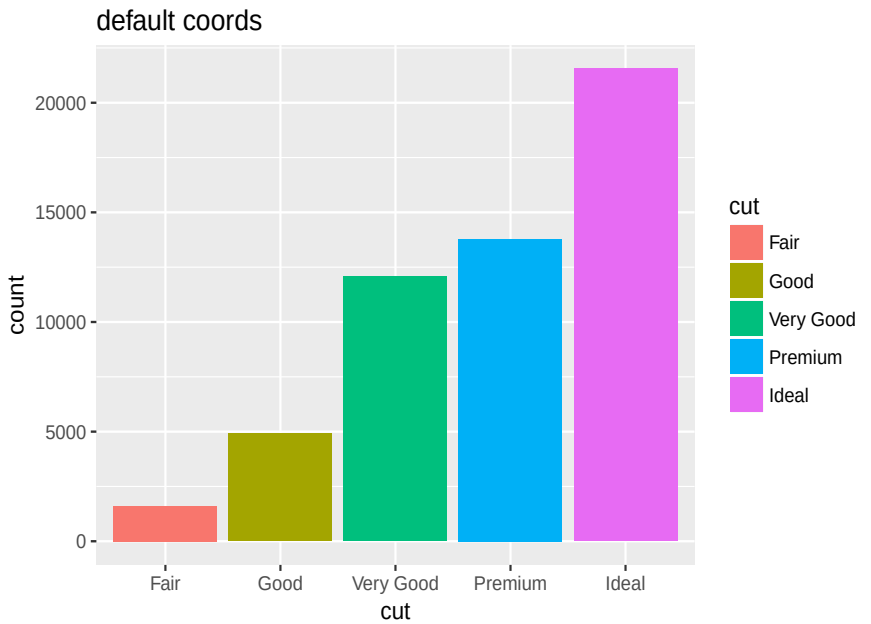
```
ggplot(data = mpg, mapping = aes(x = reorder(class,hwy,FUN=median), y = hwy)) +
  geom_boxplot(outlier.alpha=0) +
  geom_point(position = position_jitter(width=0.3,height=0), alpha=0.4) +
  coord_flip()
```

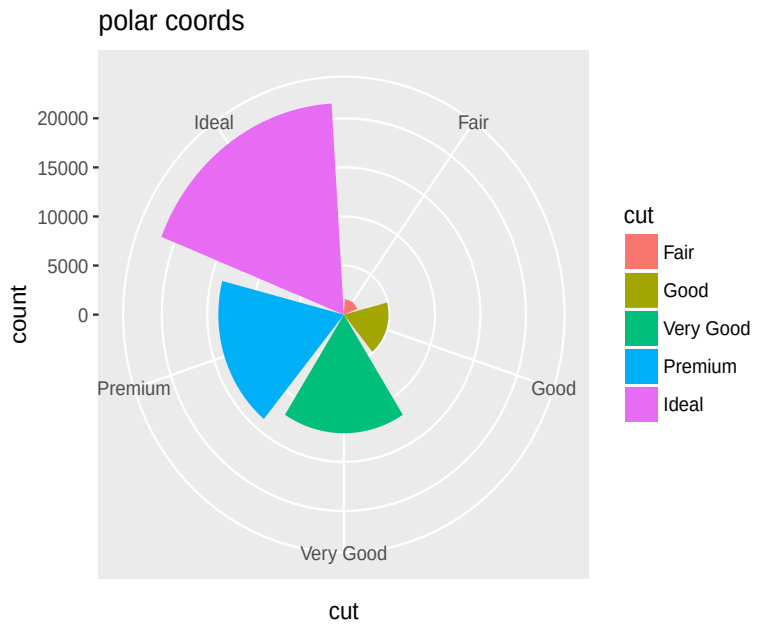


Using Polar Coordinates in a Bar Chart

Another example of manipulation of the coordinates system is switching to polar coordinates system using **coord_polar()**. Doing that on a bar chart has an interesting effect. Here is how the same bar chart looks before and after switching to the polar coordinate system:

```
fig <- ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = cut))
fig + labs(title='default coords')
fig + coord_polar() + labs(title='polar coords')
```

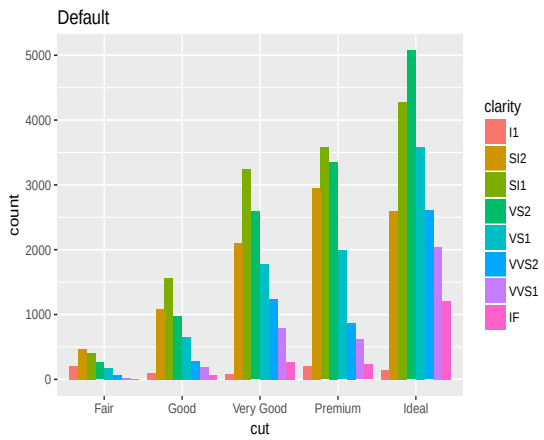


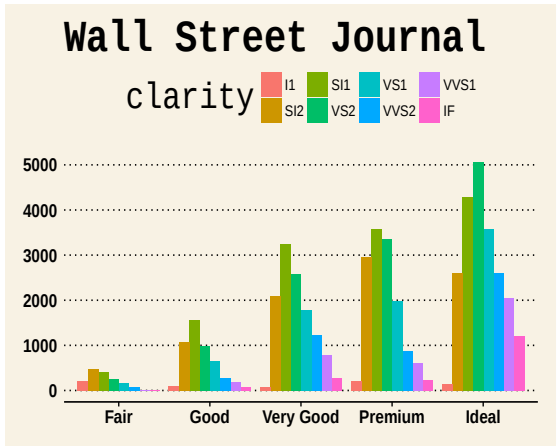


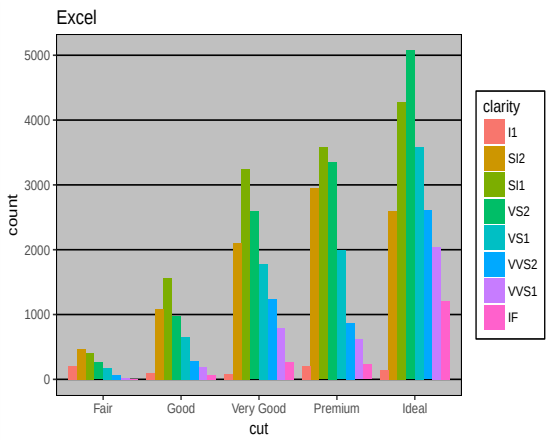
Themes

All the figures so far used the same default visual style or theme. In the library **ggthemes** there are several other themes for ggplot that can used to change the visual style of the produced figures. Some of these themes try to imitate the visual style of notable magazines or applications. To use these themes, add the selected theme as another layer to the ggplot.

```
library(ggthemes)
fig <- ggplot(data = diamonds) +
       geom_bar(mapping = aes(x = cut, fill = clarity), position = 'dodge')
fig + labs(title='Default')
fig + theme_wsj() + labs(title='Wall Street Journal')
fig + theme_excel() + labs(title='Excel')
```







Recap - ggplot Template

To recap, here is the ggplot template that summarizes all the components we have seen:

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