ST 537: More ggplot2 (3.6 and 3.7): Facets, Geoms, Stats, Positioning and Coords

Amy Ly, Abraham Mendoza, Brandon Booth, Miles Moran

Section Assigments:

3.5: Amy Ly

3.6: Brandon Booth

3.7: Brandon Booth

3.8: Abraham Mendoza

3.9: Amy Ly

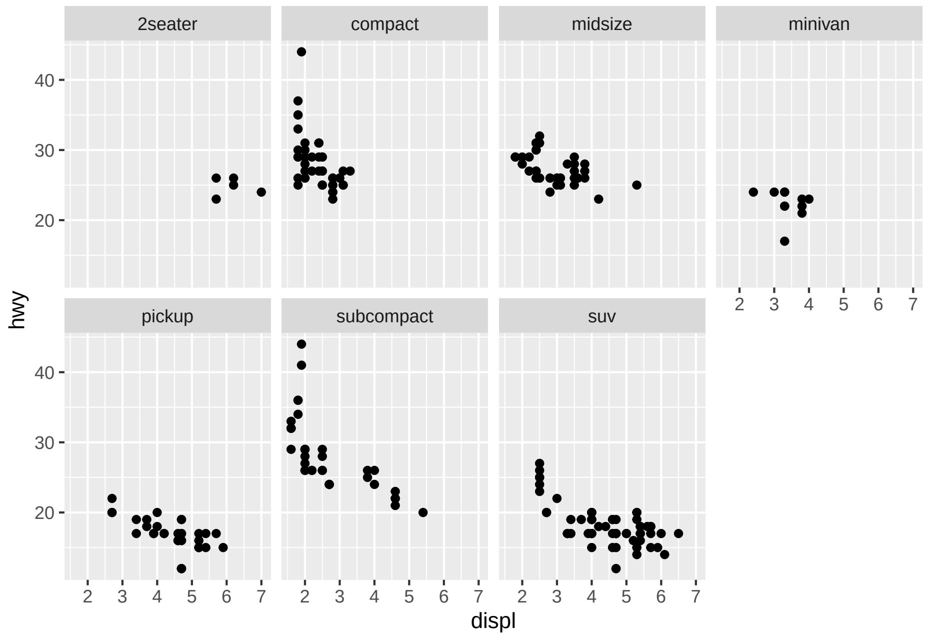
3.10: Miles Moran

Section 3.5: Facets

What are facets? Just think of them as subplots that each display one subset of the data.

Another way to think of them is as trellised visualizations. The benefit is so that you can recognize patterns between different categories in the data very quickly.

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

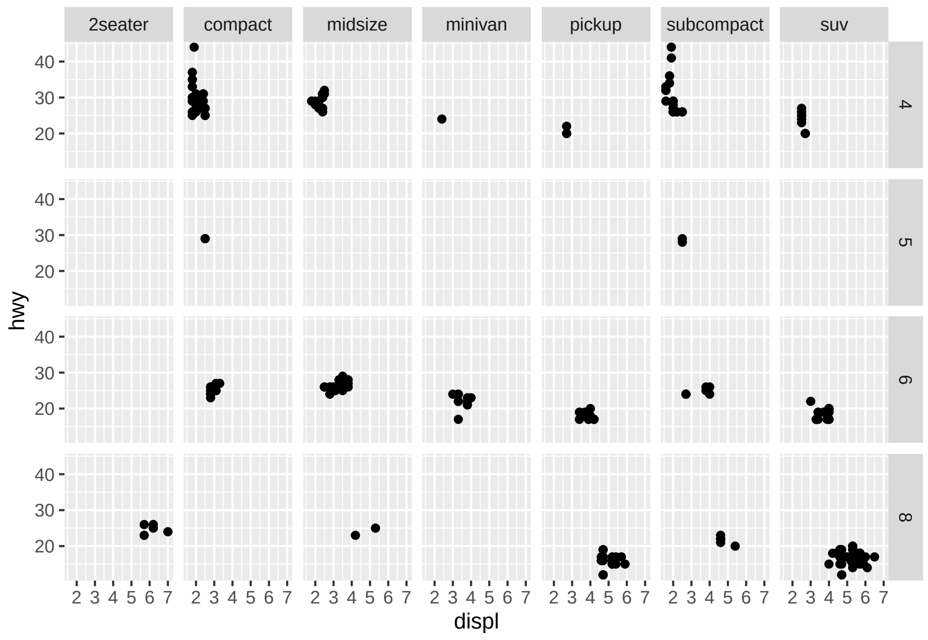


```
# nrows specifies how many rows the graphs are divided into
#subset data based on class of cars
```

The general format of for the code chunk above is facet_wrap(~ 'variable name').

This creates a long ribbon of panels and wraps it into 2d.

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



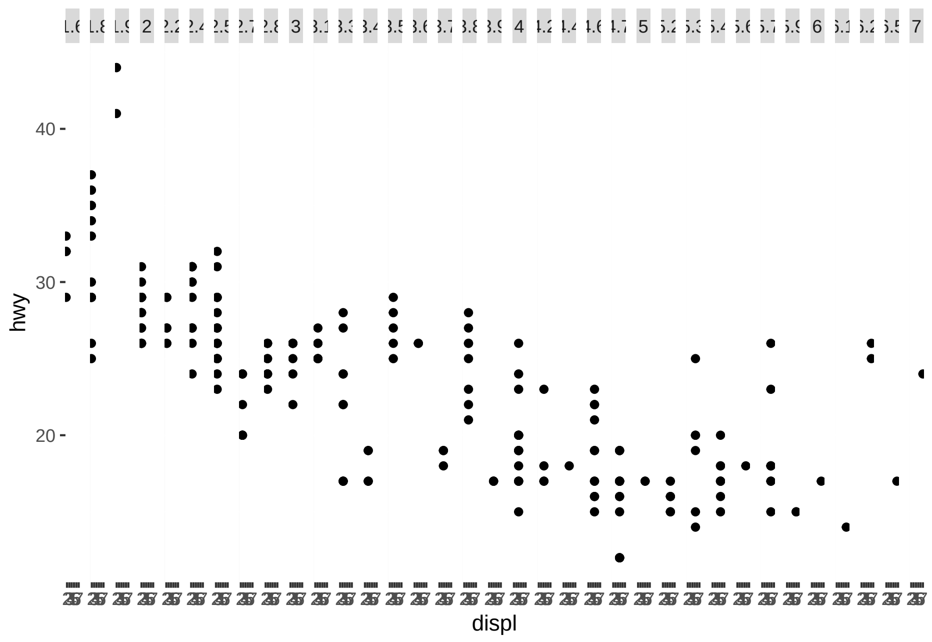
The code chunk above is an example of how to facet the plot based on a combination of two variables. Unlike facet_wrap, facet_grid will wrap the ribbon into a grid.

3.5.1 Exercises

Problem 1:

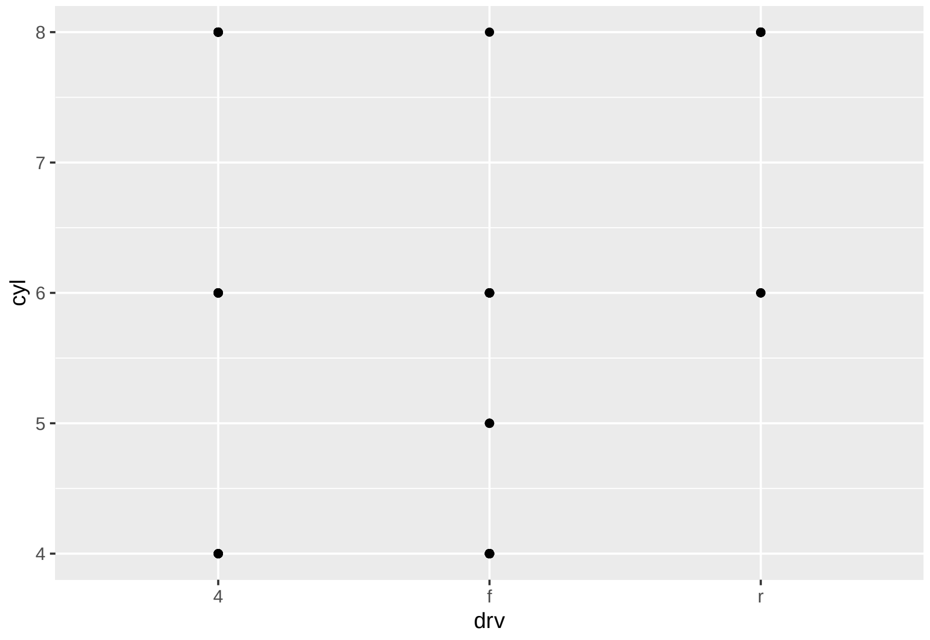
If you facet on a continuous variable, the continuous variable converts to a categorical variable. Then for each distinct value, the plot will contain a facet.

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



Problem 2:

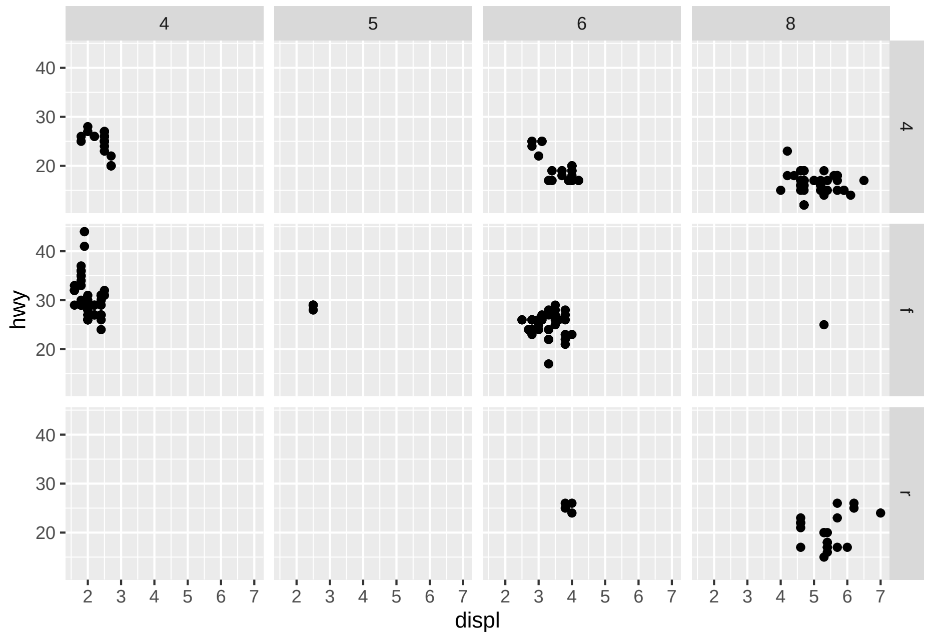
```r
ggplot(data = mpg) +
  geom_point(mapping = aes(x = drv, y = cyl))
```



When we plot cyl versus drv, there are some gaps where no observations exist between drv cyl and drv.

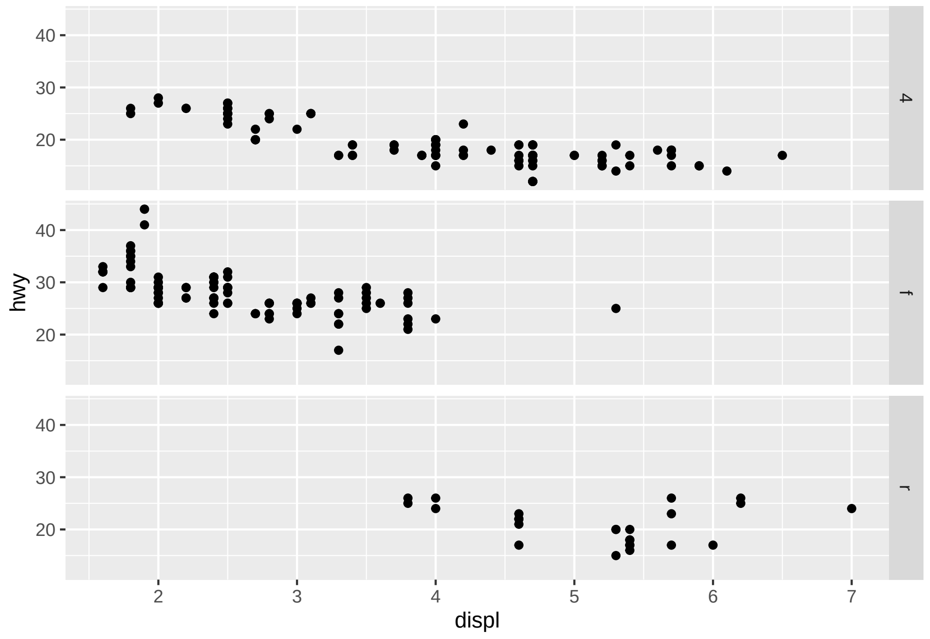
There are no observations for cyl = 7 and we should not expect to see a facet for (drv = 4, cyl = 5) and (drv = r, cyl = 4 or 5)

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


4
5
6
8
4
f
r
hwy
40
30
20
2 3 4 5 6 7
displ

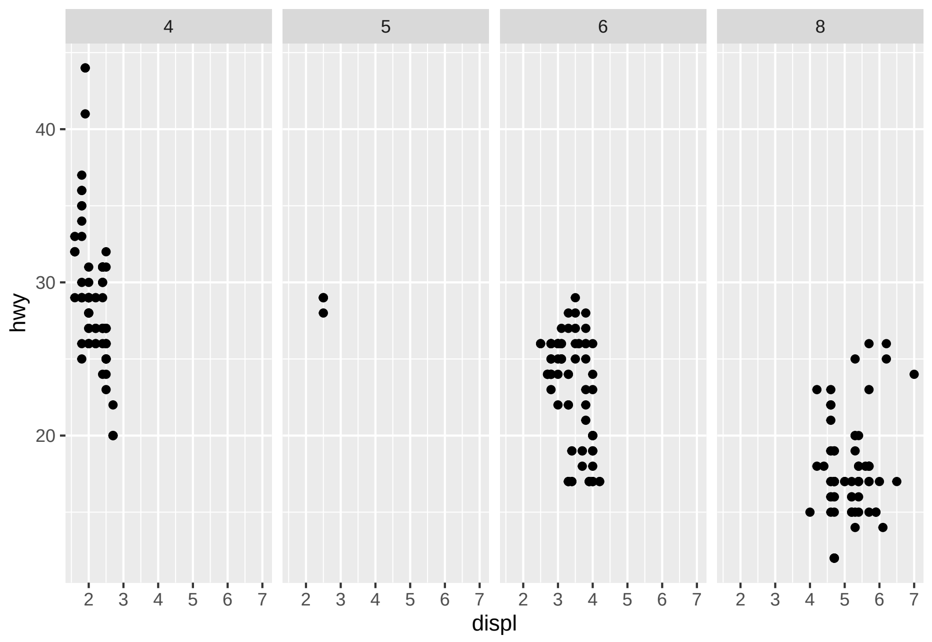
Problem 3:

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



When you specify the formula as facet_grid('variable name' ~ .), you will not have the option to facet by values of drv on the y-axis.

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

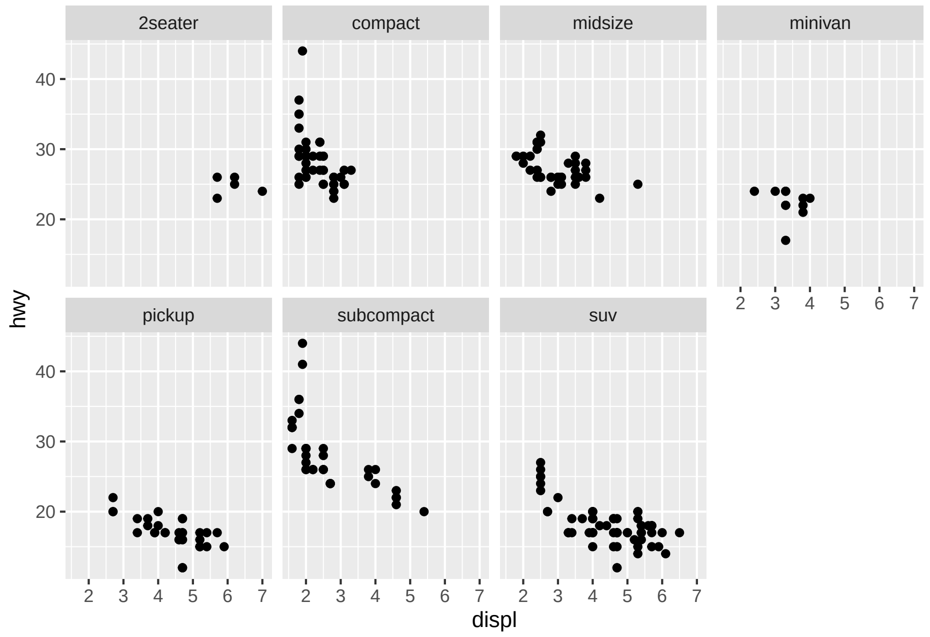


When you specify the formula as facet_grid(. ~ 'variable name'), you will not have the option to to facet by values of cyl on the x-axis.

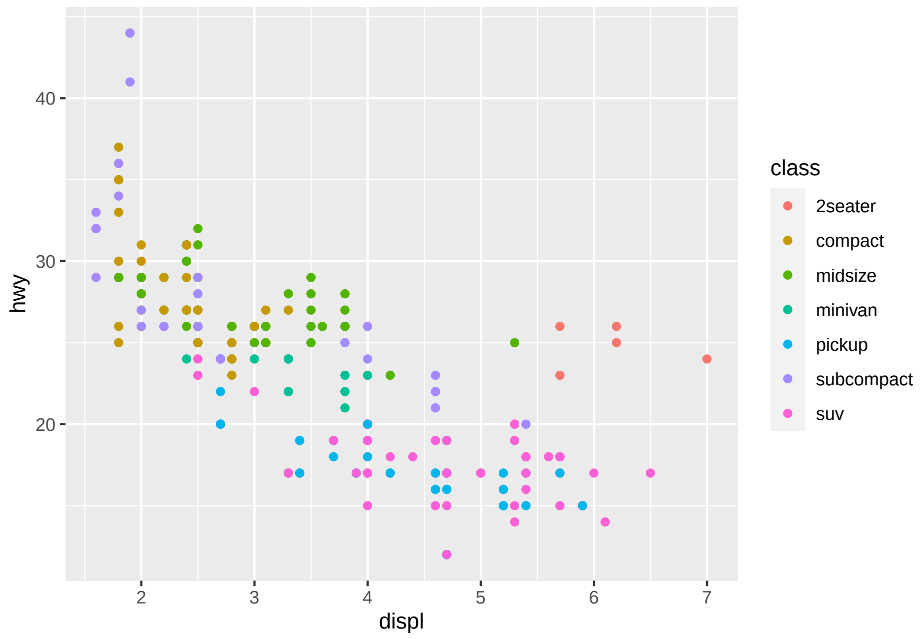
In general, the '.' ignores a particular dimension when faceting.

Problem 4:

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



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



If I facet instead of using the color aesthetic, it is easy to visualize the general trends between the highway miles per gallon and engine displacement in liters for each class of car. If I had used the color aesthetic, then the points would overlap each other and I would have to pick out the colored points to identify the trend.

The disadvantage of faceting is tht there is no overlap between the groups. This makes it harder to visualize the overall relationship. Scales may also be different between groups too.

When using aesthetics like color to differentiate groups and if groups do overlap, then small differences are easier to see.

If you have a larger dataset, it is better to utilize faceting because there will be many data points that overlap. If you used aesthetics like color or size, it would be difficult to see relationships. If the number of categories increase, then it's difficult to see the differences between them since differences between color or size would decrease.

Problem 5:

The arguments ncol and nrow specifies the number of rows and columns that the ribbon will wrap facets into.

Unlike facet_grid(), facet_wrap() only facets on one variable. This is why the nrow and ncol arguments aren't necessary. The number of rows and columns will depend on the unique values of the variables that are specified in facet_grid('variable name1' ~ 'variable name2').

Problem 6:

When using facet_grid(), the variable with more unique levels should be in columns because it will be easier to look at and there is more room horizontally.

Section 3.6: Geometric objects

This section discusses the different visual objects that can be used to to represent data. These are called geoms in ggplot2 syntax. Geom is defined in this section as the geometrical object that a plot uses to represent data. Some examples are a point geom, a smooth geom, bar geoms, line geoms, and boxplot geoms. Geoms can be changed for a plot by changing the geom function of ggplot(). The geom functions take a mapping argument but not all mapping arguments work for all geoms. Multiple geoms can be used in a plot and a legend can be added.

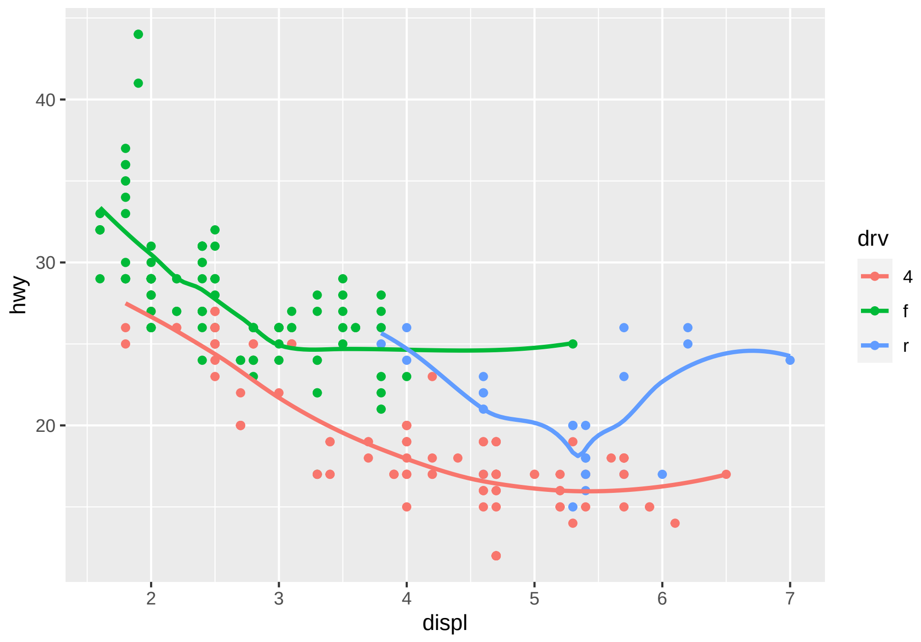
3.6.1 Exercises

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

line chart: geom_line() boxplot: geom_boxplot() histogram: geom_histogram() area chart: geom_area()

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)
```



I predict the output contain two geoms, geom_point and geom_smooth, and will not have error standard bands since se = FALSE. The x-axis will be displ and the y-axis will be hwy with 3 lines for types of drv.

The plot matches with what I predicted. I did not know what the lines would look like but it appears each of the lines have different shapes. Where drv = r, the line goes down close to linear and then archs back up where displ = 5.25. The lines also differ in length.

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

When show.legend = FALSE is added to function the lengend on the right of the chart is removed. The legend will appear if this statement is removed. In some cases the legend may added confusing. It was likely removed earlier since it was unnecessary to see the differences between the plots.

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

The se argument adds a standard error band to the lines in a plot when set to TRUE. By default, se = TRUE.

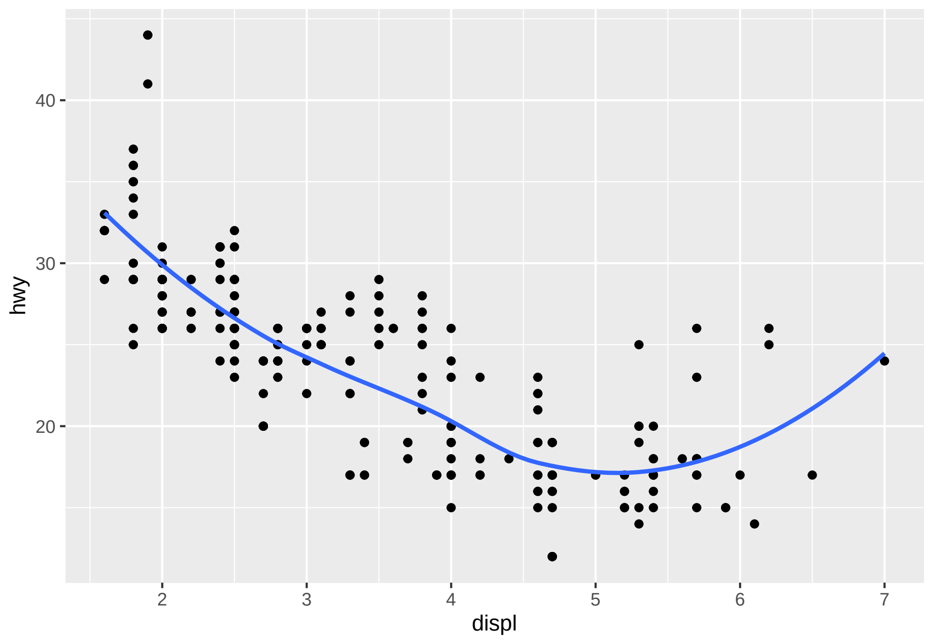
5. Will these two graphs look different? Why/why not?

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

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

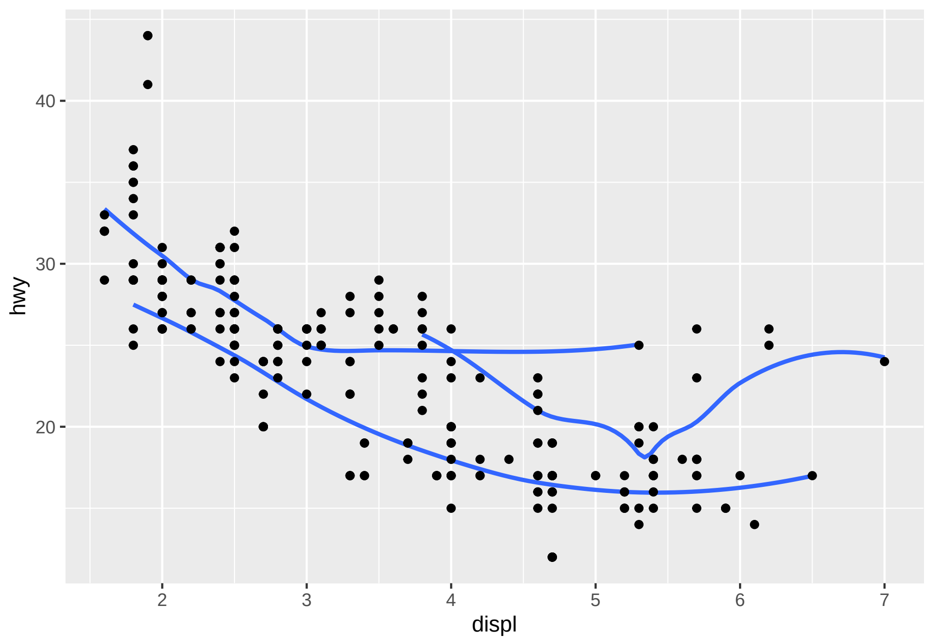
These plots will not look different. The same variables and geoms are included and the only difference being that the first function uses a more condensed method to generate the plot.

6. Recreate the R code necessary to generate the following graphs.

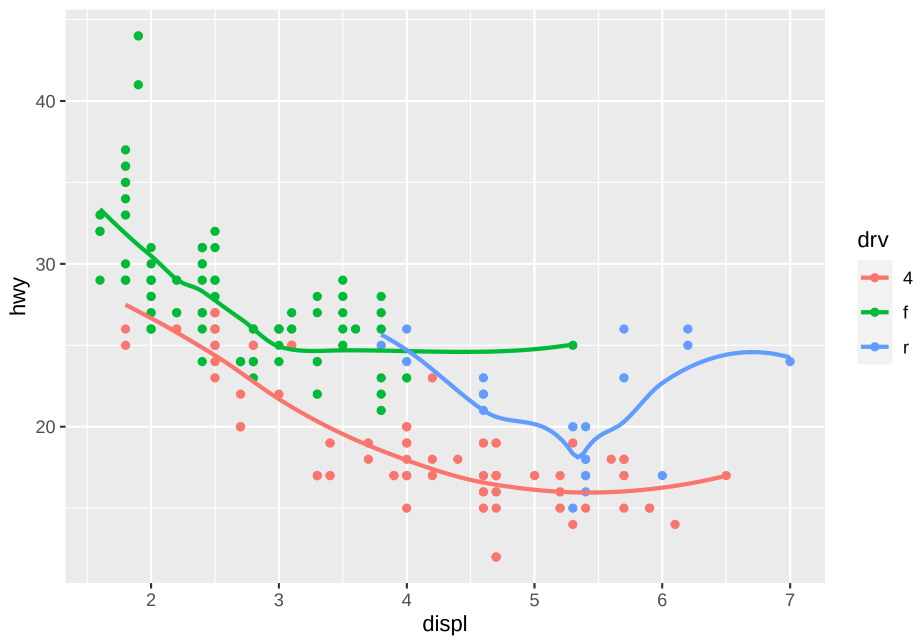
```r
# plot 1
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  geom_smooth(se = FALSE)
```



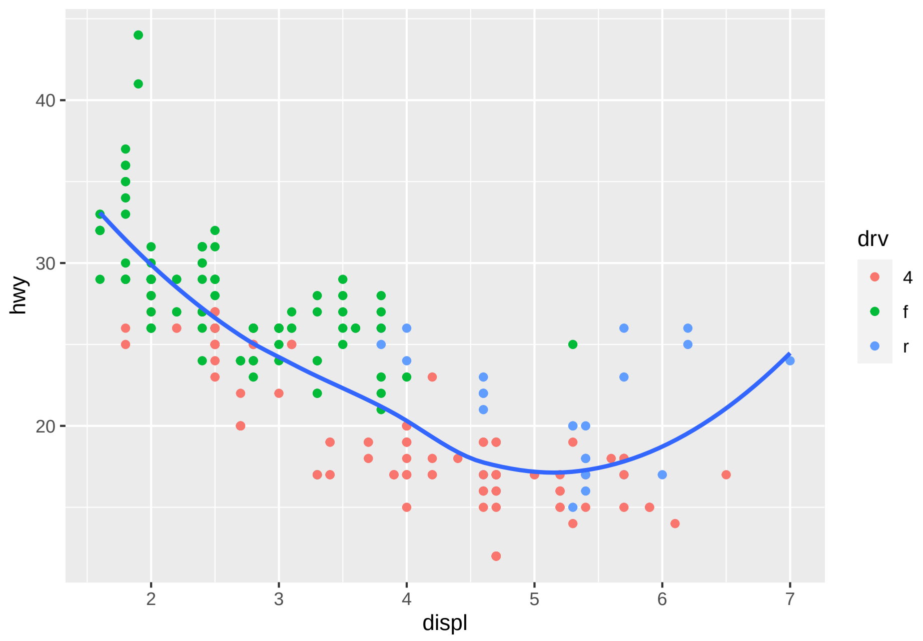
```r
# plot 2
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_smooth(mapping = aes(group = drv), se = FALSE) +
  geom_point()
```



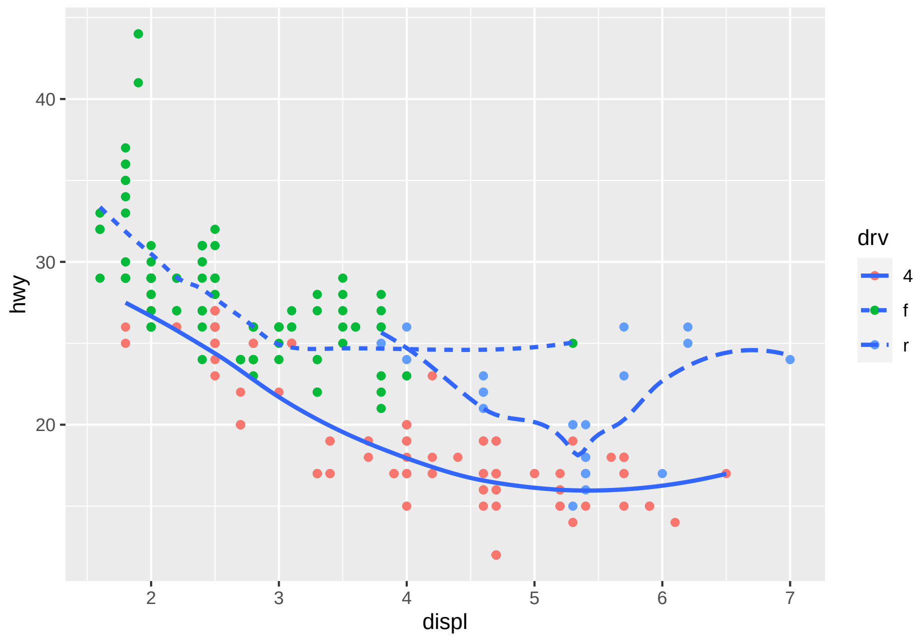
```r
# plot 3
ggplot(mpg, aes(x = displ, y = hwy, colour = drv)) +
  geom_point() +
  geom_smooth(se = FALSE)
```



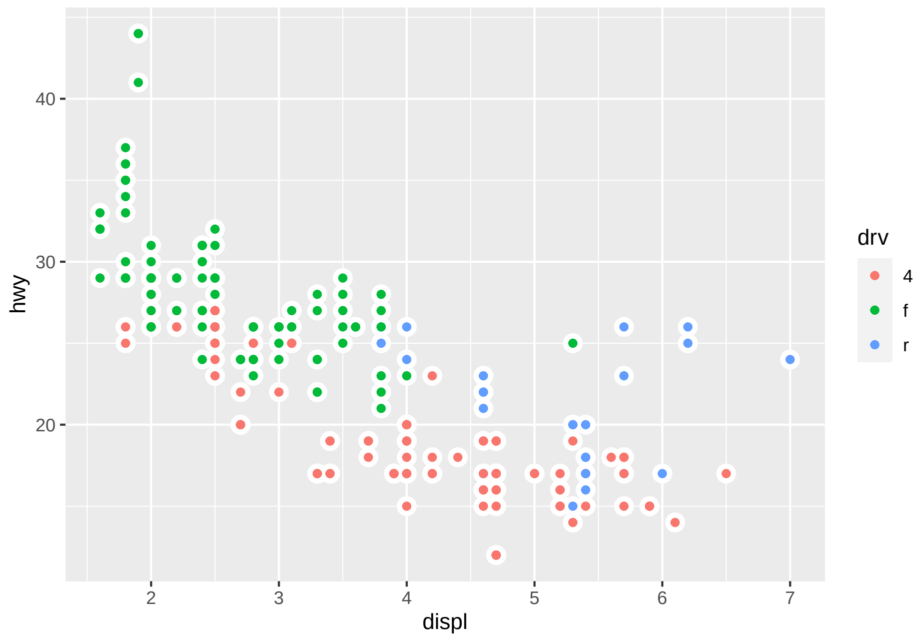
```r
# plot 4
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(se = FALSE)
```



```r
# plot 5
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(aes(colour = drv)) +
  geom_smooth(aes(linetype = drv), se = FALSE)
```



```r
# plot 6
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point(size = 4, color = "white") +
  geom_point(aes(colour = drv))
```



Section 3.7: Statistical transformations

This section describes the process off adding statistics to plot. Many plots will plot raw values of the data set used by default, while others will calculate new values to plot. Bar charts, histograms, and frequency polygons will bin your data and then plot bin counts, the number of points that fall in each bin. Smoothers will fit a model to your data and then plot predictions from the model. Boxplots will compute a robust summary of the distribution and then display a specially formatted box. A stat, or statistical transformation, is the algorithm used to calculate new values for a graph. You can see the default value of a stat for a geom by inspection the geom (ex: ?geom_bar). Geoms and stats can be used interchangeably since geoms have a default stat and stats have a default geom. Tyipically we use geoms but there are three reasons why we may want to use a stat explicitly: 1. To override the default stat 2. To override the default mapping from transformed variables to aesthetics 3. To draw greater attention to the statistical transformation in your code

3.7.1 Exercises

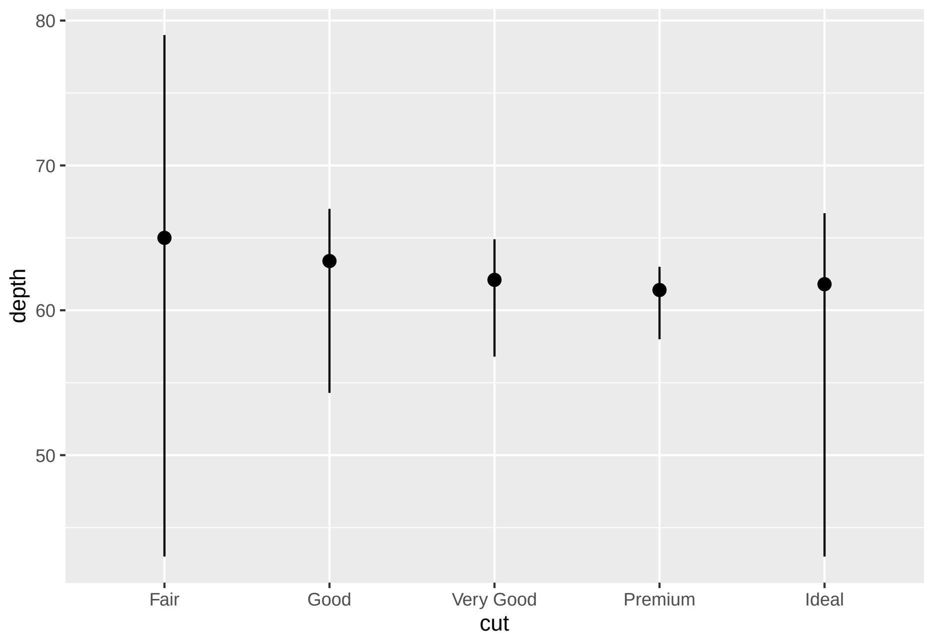
1. What is the default geom associated with stat_summary()? How could you rewrite the previous plot to use that geom function instead of the stat function?

```
#?stat_summary
```

The default geom associated with stat_summary() is "pointrange".

The plot is rewritten with function stat_summary() replaced by geom_pointrange(). Stat argument is added and equal to "summary".

```
#?geom_pointrange
ggplot(data = diamonds) +
  geom_pointrange(
    mapping = aes(x = cut, y = depth),
    stat = "summary",
    fun.min = min,
    fun.max = max,
    fun = median
  )
```



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

```
#?geom_col()
#or
#?geom_bar()
```

The main difference is that the default stat for geom_bar() is "count" and geom_col() uses stat_identity(), leaving the data as is.

Per the R documentation, there are two types of bar charts: geom_bar() and geom_col(). geom_bar() makes the height of the bar proportional to the number of cases in each group (or if the weight aesthetic is supplied, the sum of the weights). If you want the heights of the bars to represent values in the data, use geom_col() instead. geom_bar() uses stat_count() by default: it counts the number of cases at each x position. geom_col() uses stat_identity(): it leaves the data as is.

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?

```
#?geom_bar()
#?geom_boxplot()
#?geom_point()
#?geom_histogram()
#?geom_density()
#?geom_count()
#?geom_curve()
```

Geom Stat geom_bar() stat_count() geom_boxplot() stat_boxplot() geom_point() stat_identity() geom_histogram() stat_bin() geom_density() stat_density() geom_count() stat_sum() geom_curve() stat_identity()

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

```
#?stat_smooth()
```

Computed variables stat_smooth() provides the following variables, some of which depend on the orientation:

y or x: predicted value

ymin or xmin: lower pointwise confidence interval around the mean

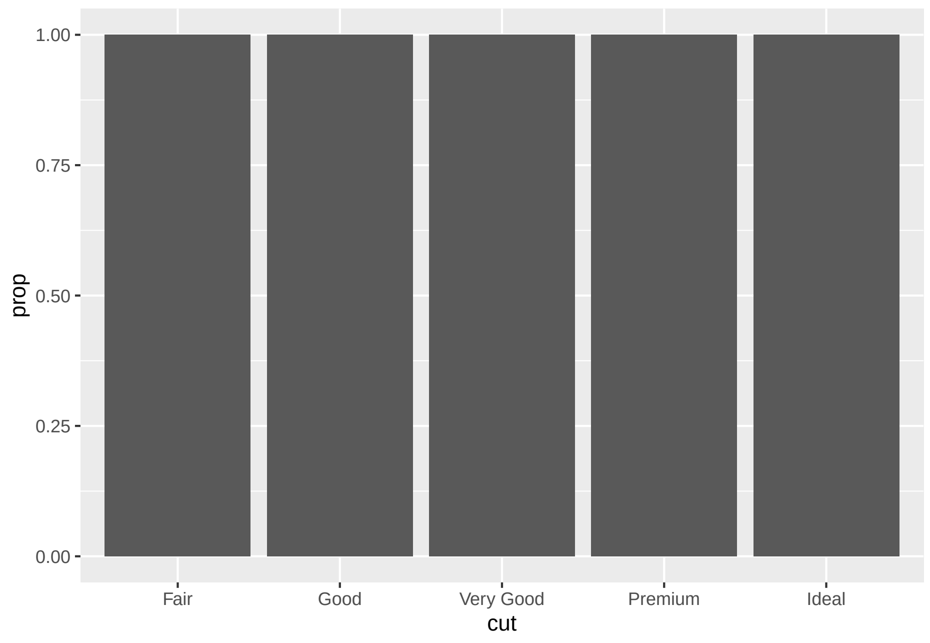
ymax or xmax: upper pointwise confidence interval around the mean

se: standard error

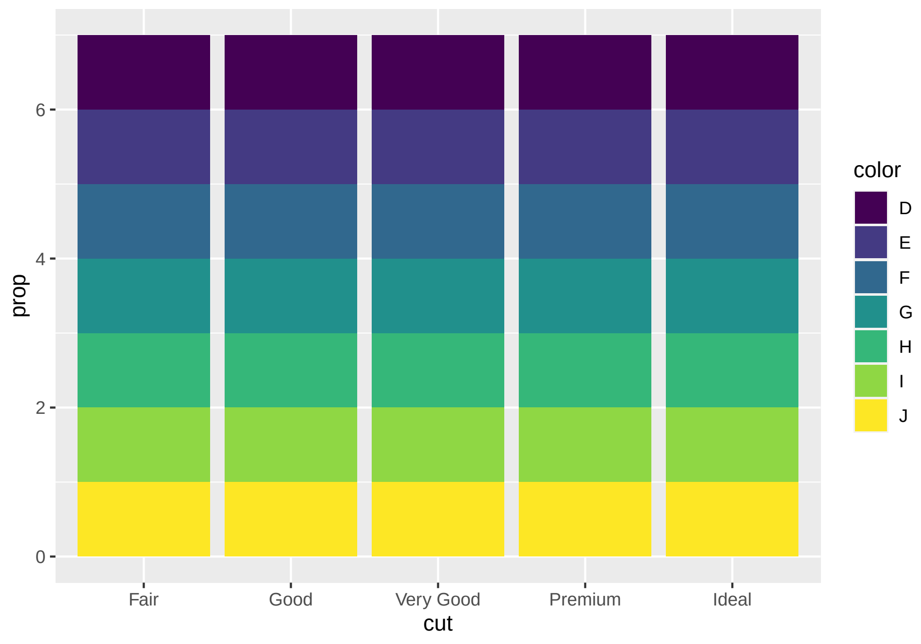
The parameters that control its behavior are method, formula, se, na.rm, n, span, fullrange, level, method.args, orientation, show.legend, inherit.aes.

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

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

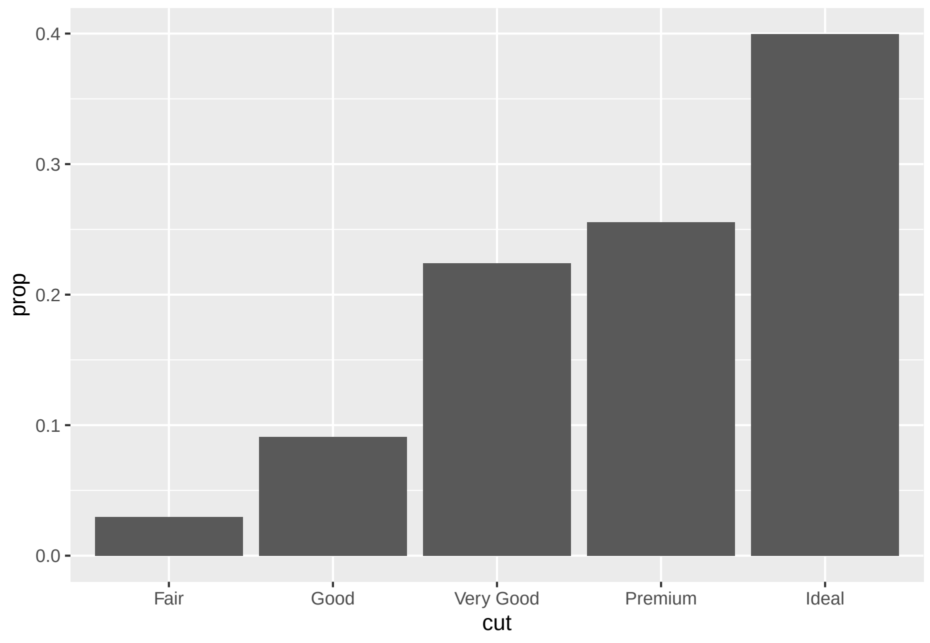


```
ggplot(data = diamonds) +
  geom_bar(mapping = aes(x = cut, fill = color, y = after_stat(prop)))
```

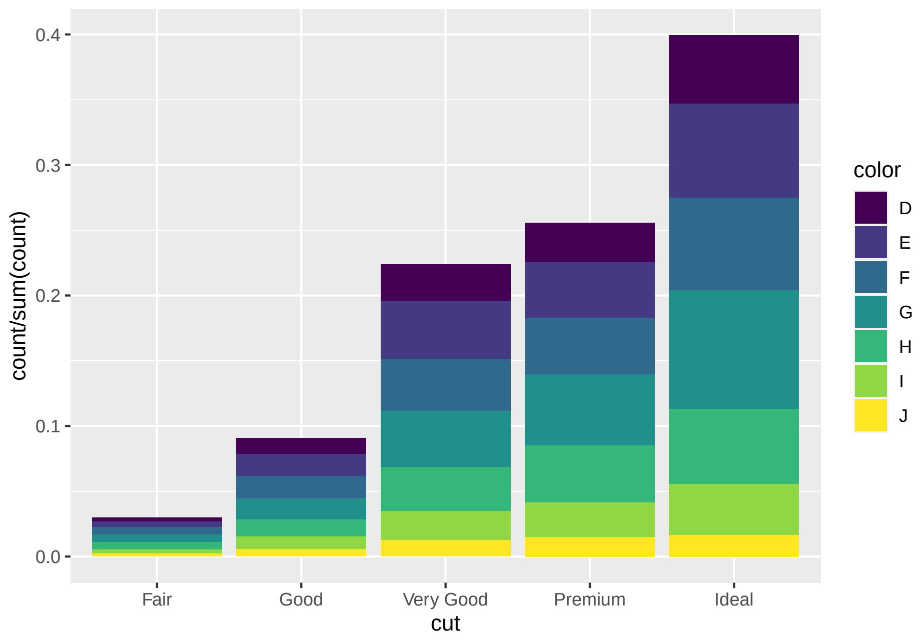


The problem is that the proportion is within each group. Without group = 1, all bars will be the same height. We need to set group = 1 and ensure the counts are relative to the total counts.

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



```
ggplot(data = diamonds) +
  geom_bar(aes(x = cut, y = ..count.. / sum(..count..), fill = color))
```

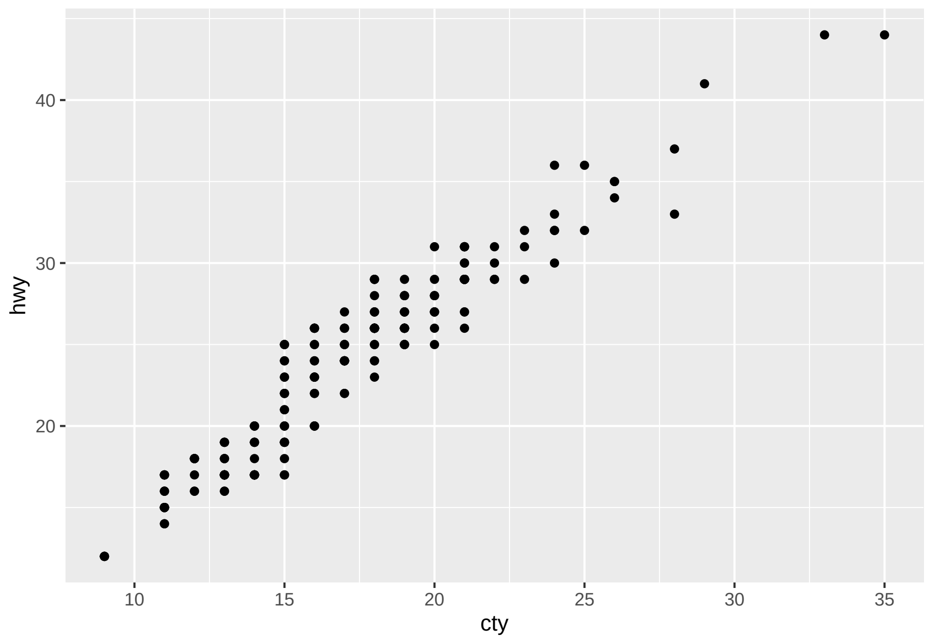


0.4
0.3
0.2
0.1
0.0
count/sum(count)
Fair
Good
Very Good
Premium
Ideal
cut
color
D
E
F
G
H
I
J

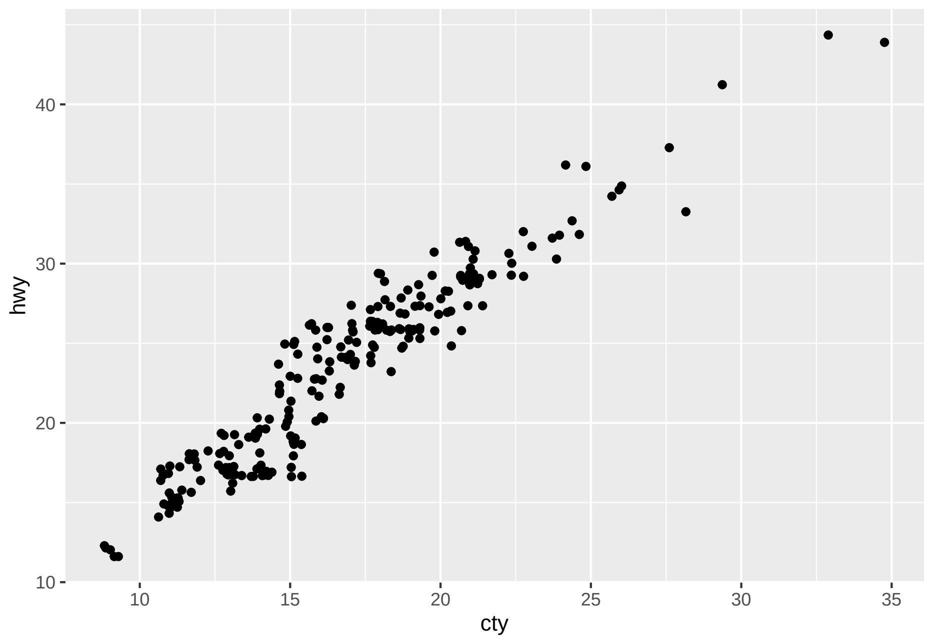
Section 3.8: Position adjustments

1: The problem with the plot is that a lot of the points in the scatterplot are overlapping. One way to improve it would be to add "jitter" so that we are able to see more of the data points and get a better understanding of the graph in a large scale.

```r
#Original Scatterplot
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point()
```



```r
#Updated Scatterplot
ggplot(data=mpg) + geom_point(mapping=aes(x = cty, y=hwy), position='jitter')
```

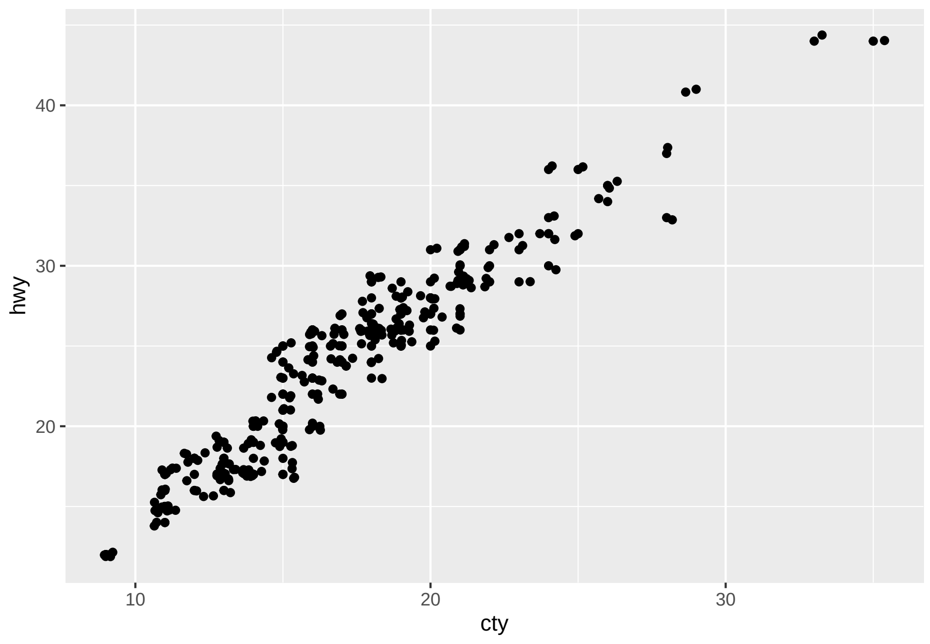


2: The parameters that control the amount of jittering in 'geom_jitter()' are "width" and "height". Both will control the amount of vertical and horizontal jitter.

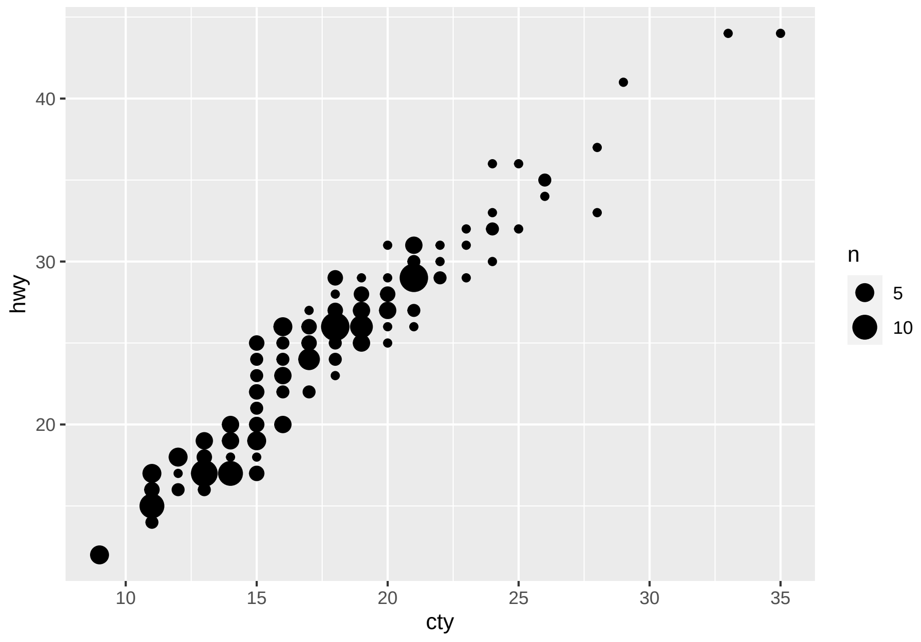
```
#?geom_jitter
```

3: 'geom_jitter" will add small amounts of variation "jitter" to the location of each point, this will reduce overlapping points. 'geom_count' counts the number of observations at each location in a plot and will then return a map of the counts to the points in the plots. Thus, if there is overlapping and you use 'geom_count' the legend will provide further insight on the amount of points that might be overlapping at a particular location on the plot. Where 'geom_jitter' will individuall add 'jitter' to each point. Both assist with decrease overlapping points in plots which can assist with individuals being able to have a better general overview of the data as a whole.

```
#?geom_jitter
#?geom_count

#Scatterplot using 'geom_jitter'
ggplot(data=mpg) + geom_point(mapping=aes(x = cty, y=hwy)) + geom_jitter(mapping=aes(x=cty, y=hwy),posi
```

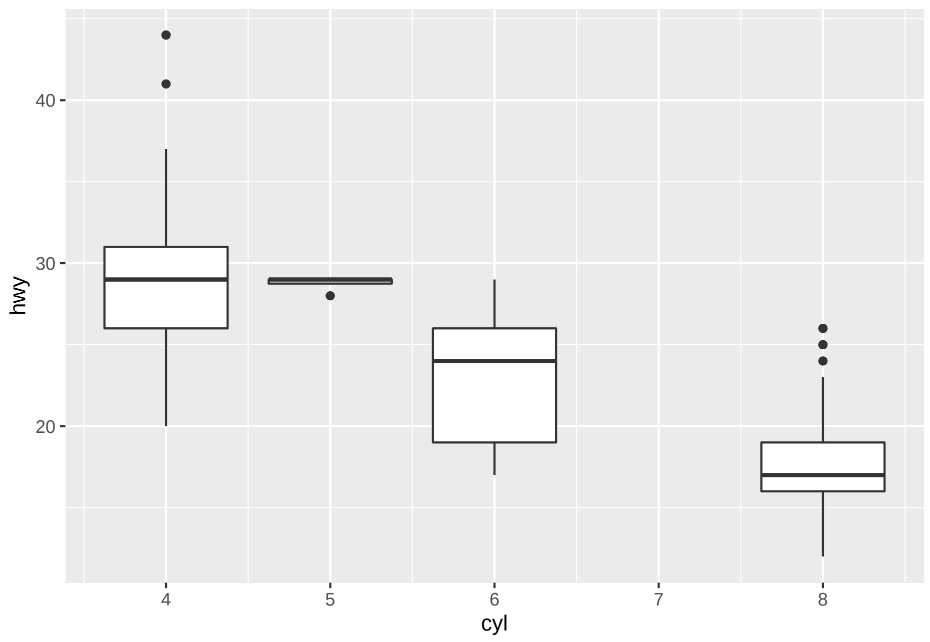


```
#Scatterplot using 'geom_count'
ggplot(data=mpg) + geom_point(mapping=aes(x = cty, y=hwy)) + geom_count(mapping=aes(x=cty, y=hwy))
```



4: The default position adjustment for 'geom_boxplot()' is "dodge2."

```
#?mpg
#?geom_boxplot

ggplot(data=mpg, mapping=aes(x=cyl, y=hwy, group=cyl)) + geom_boxplot()
```


40
30
20
hwy
4
5
6
7
8
cyl

Section 3.9: Coordinate systems

The default coordinate system for ggplot is the Cartesian coordinate system (x-axis is the horizontal axis and y-axis is the vertical axis).

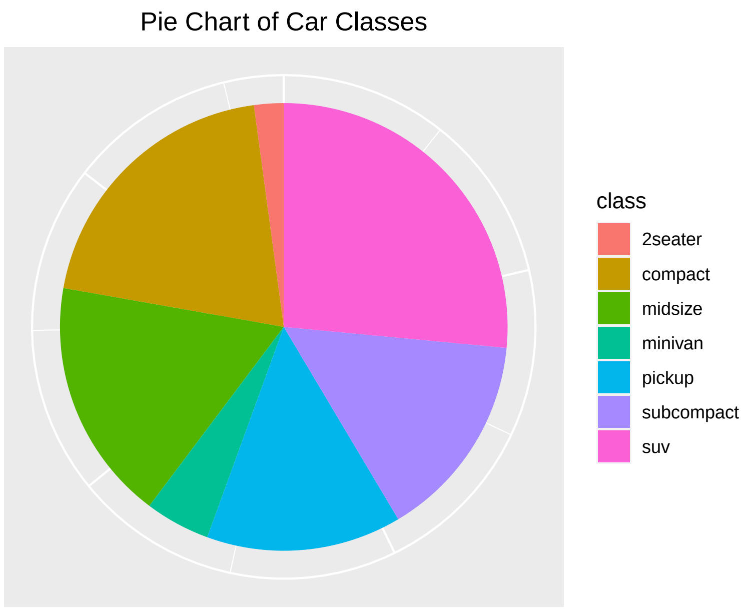
Some useful functions are summarized below:

- coord_flip() switches the x and y axes, which is useful for creating horizontal box plots or having long labels on the y-axis.
- coord_quickmap() projects a portion of the earth by approximating a spherical space onto a flat 2D plane so that straight lines are preserved.
- coord_polar() uses polar coordinates which you can use to turn a bar chart into a Coxcomb chart.

3.9.1 Exercises

Problem 1:

```
ggplot(mpg, aes(x = factor(1), fill = class)) +
  geom_bar() +
  coord_polar(theta = "y") +
  labs(title = "Pie Chart of Car Classes", x = "", y = "") + theme(axis.text.y=element_blank(),
        axis.ticks.y=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```



```
# remove y and x axis labels
```

The above code chunk will turn a stacked bar chart into a pie chart using coord_polar().

Problem 2:

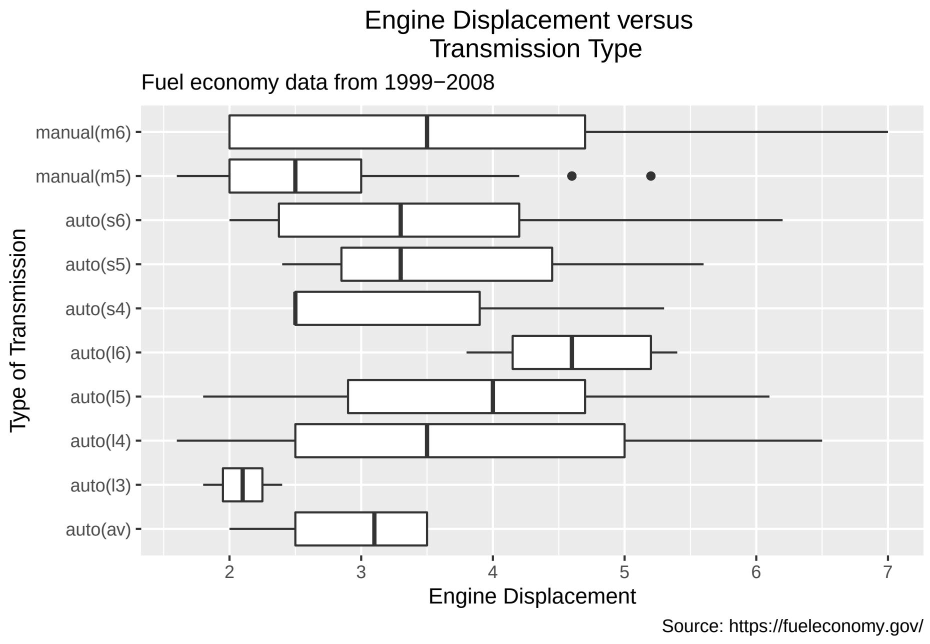
The labs() function lets us change the axis labels and legend titles to help make plots more informative. You can add (+) them using:

- xlab
- ylab *ggtitle

Alternatively, you can also specify the following as arguments within labs(): * title * subtitle * caption

An example is shown below:

```r
ggplot(data = mpg, mapping = aes(x = trans, y = displ)) +
  geom_boxplot() +
  coord_flip() +
  labs(
    y = "Engine Displacement",
    x = "Type of Transmission",
    title = "Engine Displacement versus \n Transmission Type",
    subtitle = "Fuel economy data from 1999-2008",
    caption = "Source: https://fueleconomy.gov/"
  )
```



Problem 3:

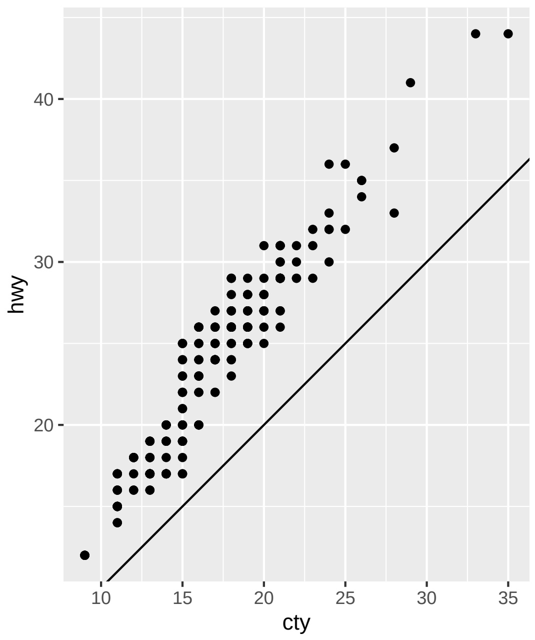
According to the help page for the functions,

coord_map() projects a portion of the earth, which is in 3D, onto a flat 2D plane. The default projection used is "mercator". Map projections do not, in general, preserve straight lines, so this requires considerable computation.

coord_quickmap() is quicker computationally and can preserve straight lines. It works best for smaller areas closer to the equator, for regions that span only a few degrees and are not too close to the poles. By setting the aspect ratio of the plot to the appropriate lat/lon ratio approximates the usual mercator projection at the expense of correctness.

Problem 4:

```r
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline() +
  coord_fixed()
```

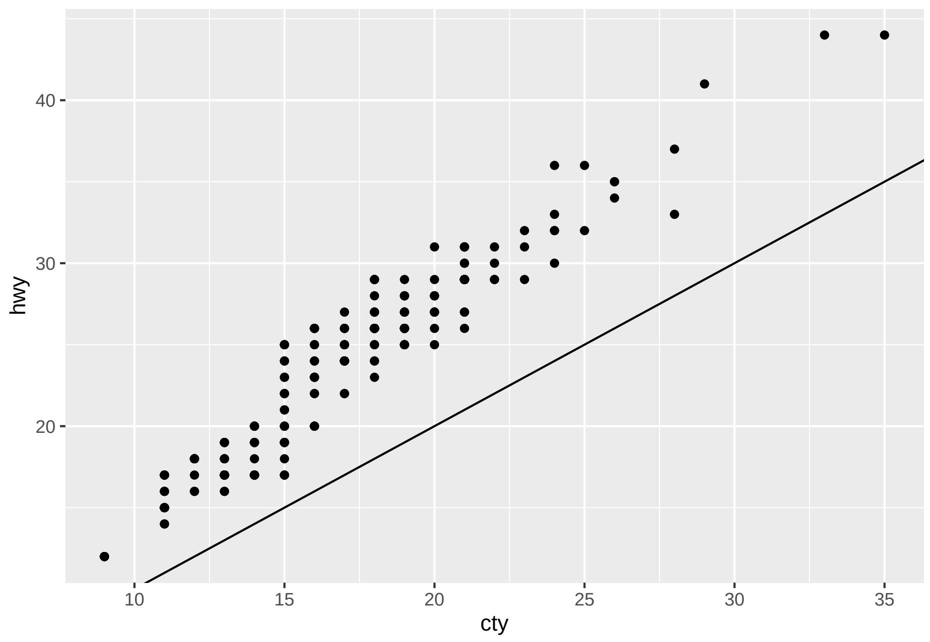


The plot above tells me that city and highway mpg is linearly related.

The help page states that coord_fixed helps to adjust the plot aspect ratio. By default, the ratio = 1 and ensures that one unit on the x-axis is the same length as one unit on the y-axis.

As shown below, the plot looks skewed due to the angle of the line (and our perspective) when it is plotted without coord_fixed():

```r
ggplot(data = mpg, mapping = aes(x = cty, y = hwy)) +
  geom_point() +
  geom_abline()
```



The function geom_abline() adds a reference line to the plot. By default, the reference line will have an intercept = 0 and a slope = 1

Section 3.10: The layered grammar of graphics

In short, the creators of `ggplot2` believe that the elements composing *any* plot can be categorized as one of those discussed in sections 3.1-3.9:

1. Data
2. Geoms
3. Mappings (Aesthetics)
4. Statistics (Transformations)
5. Position Adjustments
6. Coordinate Systems
7. Faceting Schemes

with a general-template looking something like

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

It's important to note that these elements all *share* the many different responsibilities that must be fulfilled to make a legible plot. While Faceting Schemes and Position Adjustments are particularly useful for making comparisons, this is also feasible by using unique aesthetics (e.g. grouping using plotting characters or colors). Similarly, while stats are useful for changing the shape/scale of the plot (by transforming the data itself), this is also feasible by changing the axis scales or the coordinate system. There will never be a clear-cut path to the plot you envision.

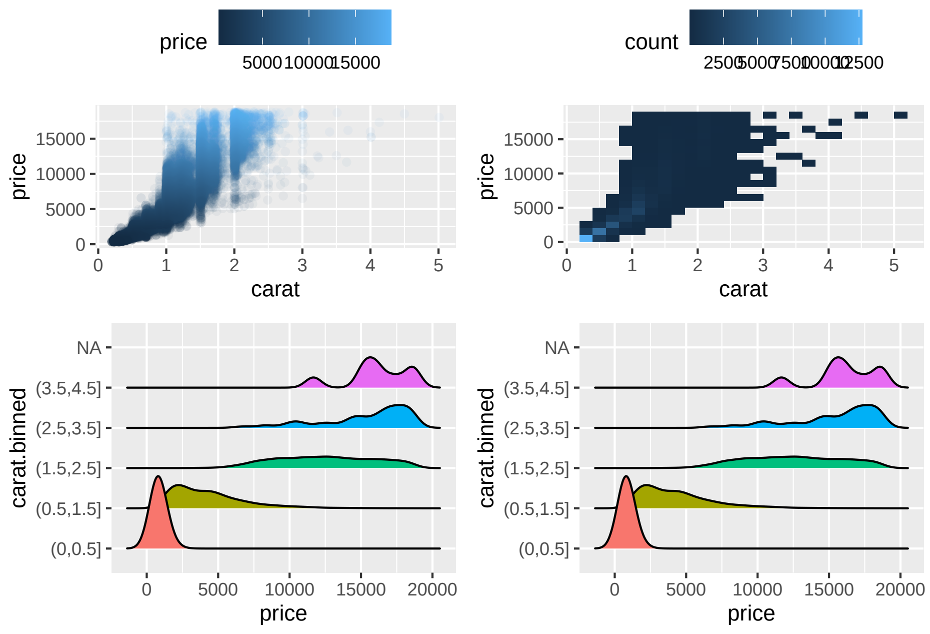
With these elements in mind, we can come up with a (very incomplete) procedure for determining what geoms, mappings, stats, etc. we might want to use when plotting our data:

1. Obtain data
2. Choose a set of response variables (or statistics) of interest
    - Do we want to know the count / number of occurrences of a particular value?
    - Do we want to know the proportion of occurrences of a particular value?
    - Do I want to summarize the distribution of a particular variable?
3. Consider the properties of the response…
    - Is the response continuous or discrete?
    - Is the response univariate or multivariate?
    - If we bin the data (e.g. for a histogram), how should we define the bins?
4. Choose a set of comparisons of interest
    - Should we group the response using the response or a separate variable/statistic?
5. Consider the properties of the grouping variable…
    - Is the grouping variable continuous or discrete?
6. Other things to consider
    - If the grouping variable is discrete/categorical…
        - Should we display the groups side-by-side on the same plot?
        - Or, should we display the groups overlapping in the same plot?
        - Or, should we display the groups in separate plots?
    - If the grouping variable is continuous…
        - should we discretize/bin it?

For example, suppose we want to know the relationship between the price of a diamond and its quality. Using the dataset `ggplot2::diamonds`, we might want to…

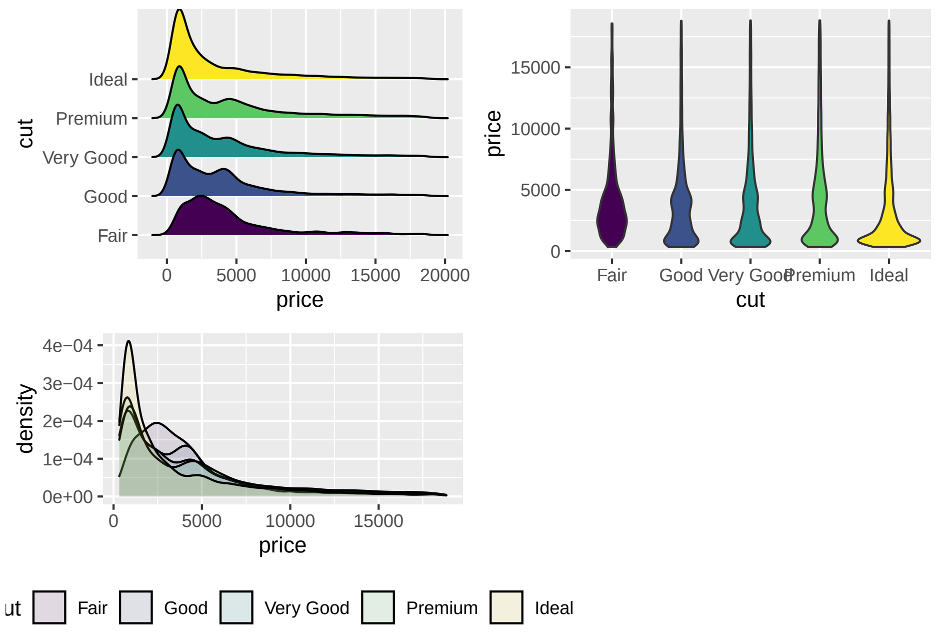
1. Plot the distribution of diamond `price` (continuous), grouping by `carat` (continuous)

```
library(ggridges)
library(gridExtra)

p1 <- ggplot(diamonds) +
  geom_point(aes(carat, price, color=price), alpha=0.05) +
  theme(legend.position = "top")

p2 <- ggplot(diamonds) +
  geom_bin2d(aes(carat, price, color=price), binwidth=c(0.2, 1000)) +
  theme(legend.position = "top")

# discretize our continous grouping variable
carat.binned <- cut(diamonds$carat, breaks=seq(0,5,1))
p3 <- ggplot(diamonds) +
  geom_density_ridges(aes(price, carat.binned, fill=carat.binned)) +
  theme(legend.position = "none")

# notice how the distributions change depending on how we pick the bins!!!
carat.binned <- cut(diamonds$carat, breaks=c(0,0.5,1.5,2.5,3.5,4.5))
p4 <- ggplot(diamonds) +
  geom_density_ridges(aes(price, carat.binned, fill=carat.binned)) +
  theme(legend.position = "none")

grid.arrange(p1, p2, p3, p4, ncol=2)
```



2. Plot the distribution of diamond `price` (continuous), grouping by `cut` (categorical)

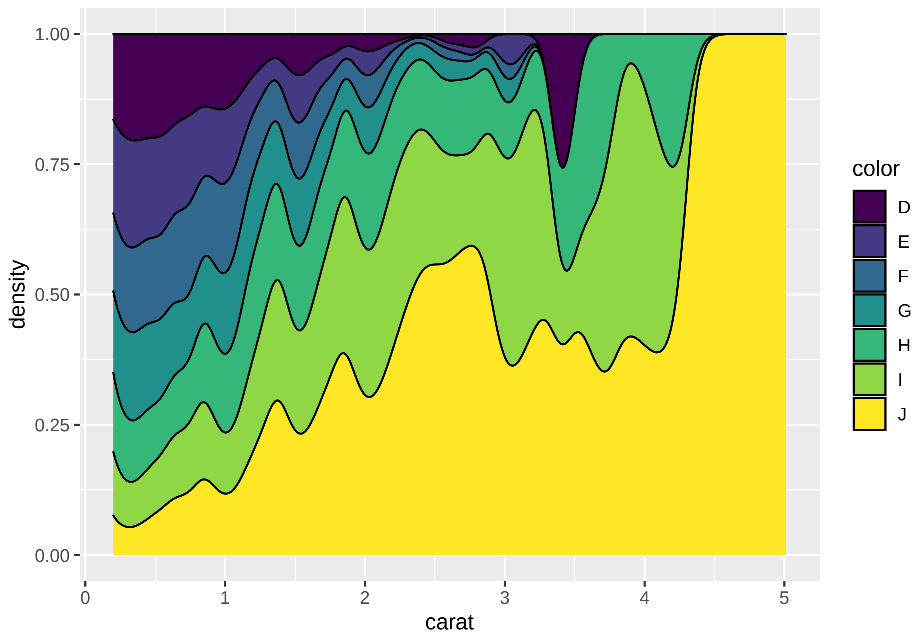
```r
p1 <- ggplot(diamonds) +
  geom_density_ridges(aes(price, cut, fill=cut)) +
  theme(legend.position = "none")

p2 <- ggplot(diamonds) +
  geom_violin(aes(cut, price, fill=cut)) +
  theme(legend.position = "none")

p3 <- ggplot(diamonds) +
  geom_density(aes(price, fill=cut), alpha=.1) +
  theme(legend.position = "bottom")

grid.arrange(p1, p2, p3, ncol=2)
```



Alternatively, suppose we want to know the relationship between the quality of a diamond (via its `color`) and its physical properties. We might want to. . .

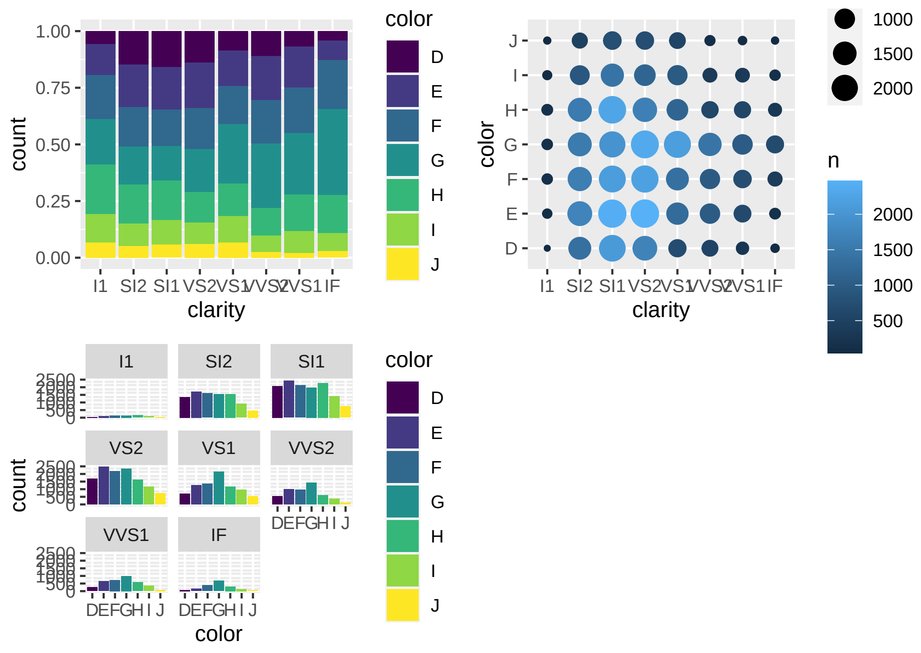
3. Plot the distribution of diamond `carat` (categorical), grouping by `color` (continuous)

```
ggplot(diamonds) +
  geom_density(aes(carat, group=color, fill=color), adjust=1.5, position="fill")
```



4. Plot the distribution of diamond `clarity` (categorical), grouping by `color` (categorical)

```
p1 <- ggplot(diamonds) +
  geom_bar(aes(clarity, group=color, fill=color), position="fill")

p2 <- ggplot(diamonds) +
  geom_count(aes(clarity, color, color = ..n..))

p3 <- ggplot(diamonds) +
  geom_bar(aes(color, fill=color)) +
  facet_wrap(~clarity)

grid.arrange(p1, p2, p3, ncol=2)
```


count
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF
clarity
color
D
E
F
G
H
I
J
color
J
I
H
G
F
E
D
n
I1
SI2
SI1
VS2
VS1
VVS2
VVS1
IF
count
DEFGHIJ
color