Introduction to ggplot2

Rolf Bänziger, rbanziger@westminster.ac.uk

30/01/2023

Introduction

Last week, we saw how to produce charts with the base R functions such as plot, pie, bar plot, hist, etc. While these functions are easy to use, we need to learn many different functions; and each of these functions has slightly different parameters. You probably also noticed that the produced graphics have a rather plain look. While there are ways to improve these charts' look, we do need to learn even more functions to do so.

ggplot2 aims to solve these issues. ggplot2 is a system for declaratively creating graphics, based on the ideas presented in The Grammar of Graphics. The Grammar of Graphics provides a framework to construct graphics by concisely described its components.

All plots are composed of data, i.e. the information you want to visualise, and a mapping, the description of how the data's variables are mapped to aestethic attributes. There are five mapping components:

- A *layer* is a collection of geometric elements and statistical transformations. Geometric elements, geoms for short, represent what you actually see in the plot: points, lines, polygons, etc. Statistical transformations, stats for short, summarise the data: for example, binning and counting observations to create a histogram, or fitting a linear model.
- *Scales* map values in the data space to values in the aesthetic space. This includes the use of colour, shape or size. Scales also draw the legend and axes, which make it possible to read the original data values from the plot (an inverse mapping).
- A *coord*, or coordinate system, describes how data coordinates are mapped to the plane of the graphic. It also provides axes and gridlines to help read the graph. We normally use the Cartesian coordinate system, but a number of others are available, including polar coordinates and map projections.
- A *facet* specifies how to break up and display subsets of data as small multiples. This is sometimes also known as conditioning or latticing/trellising.
- A *theme* controls the finer points of display, like the font size and background colour. While the defaults in ggplot2 have been chosen with care, you may need to consult other references to create an attractive plot.

Hadley Wickham, the author of the *ggplot2* package wrote an excellent book about *ggplot2*, which is freely available here: ggplot2.

RStudio also provides a cheat sheet for data visualisation, which is highly recommended: Data visualization with ggplot2.

You will soon see that ggplot2 is much clearer structured than the base R plotting functions, and - once you grasp the underlying concepts - is easier to use. It also produces neater graphics with the default parameters. All these properties make ggplot2 one of the most used graphics packages for R.

Preliminaries

ggplot2 is not part of the base R installation. You need to install it first if you haven't already done so. You can use `install.packages()` to install the package or use the Packages window to the left.

```
install.packages("ggplot2")
```

As usual, once you've installed the package, you will need to load it.

```
library(ggplot2)
```

Getting started

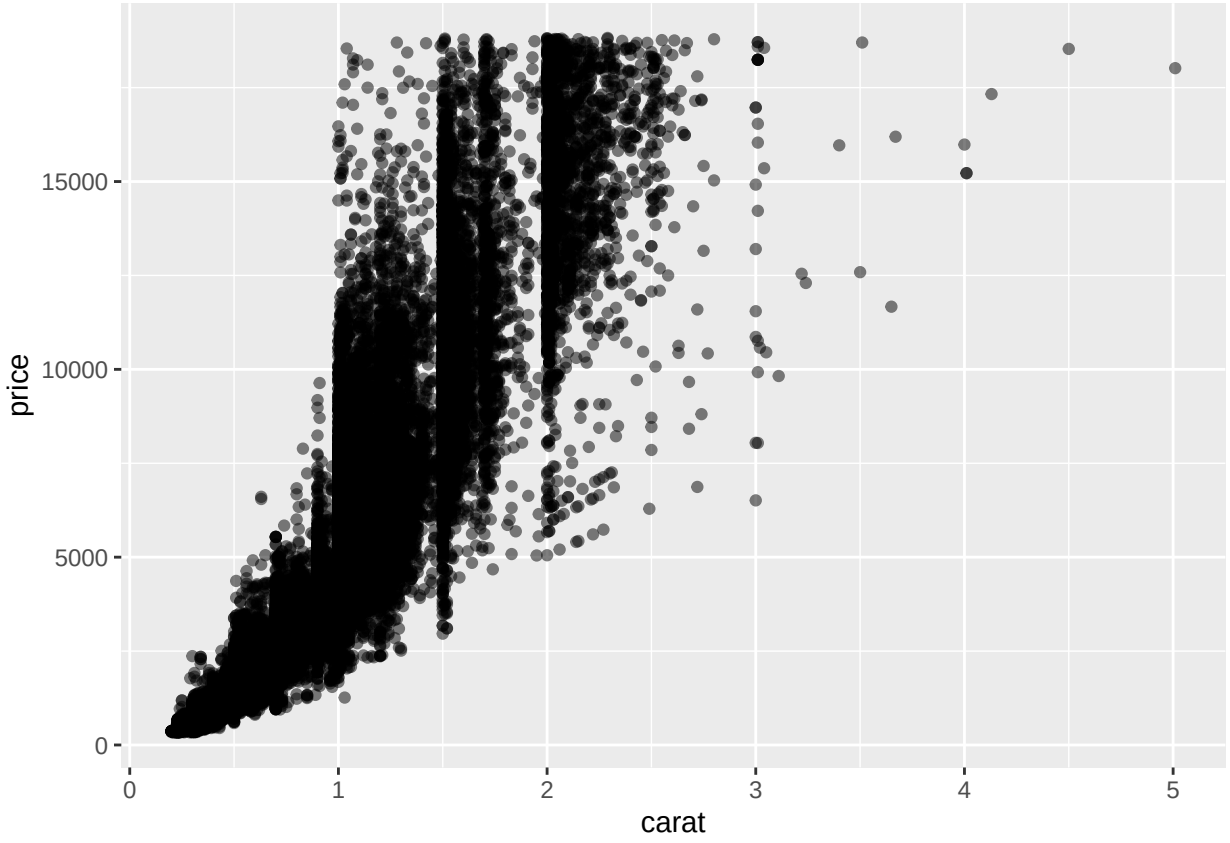
We will use the built-in diamonds data set.

```
? diamonds
```

A basic plot

Let's a basic scatter plot showing the weight (i.e. carat) by its price.

```
ggplot(diamonds) + aes(carat, price) + geom_point(alpha = 0.5)
```



To create a graphic with ggplot2, you begin by creating the basic coordinate system with `ggplot()` and then add layers to it. We made the data `diamonds` available to use in all layers by passing it to `ggplot()`. You need at least one aesthetic mapping layer and one geometry layer. We use the `aes()` function to construct the aesthetic mapping. In the example above, we mapped the dimensions carat and price to the x and y axis (other aesthetic are available to map to dimensions, we'll come to that shortly). Finally, we create the geometry layer with `geom_point()`, which creates a scatter plot (because we specified the parameter alpha the dots are translucent).

A note about the data: The diamonds data set contains over 50000 records:

```
nrow(diamonds)
```

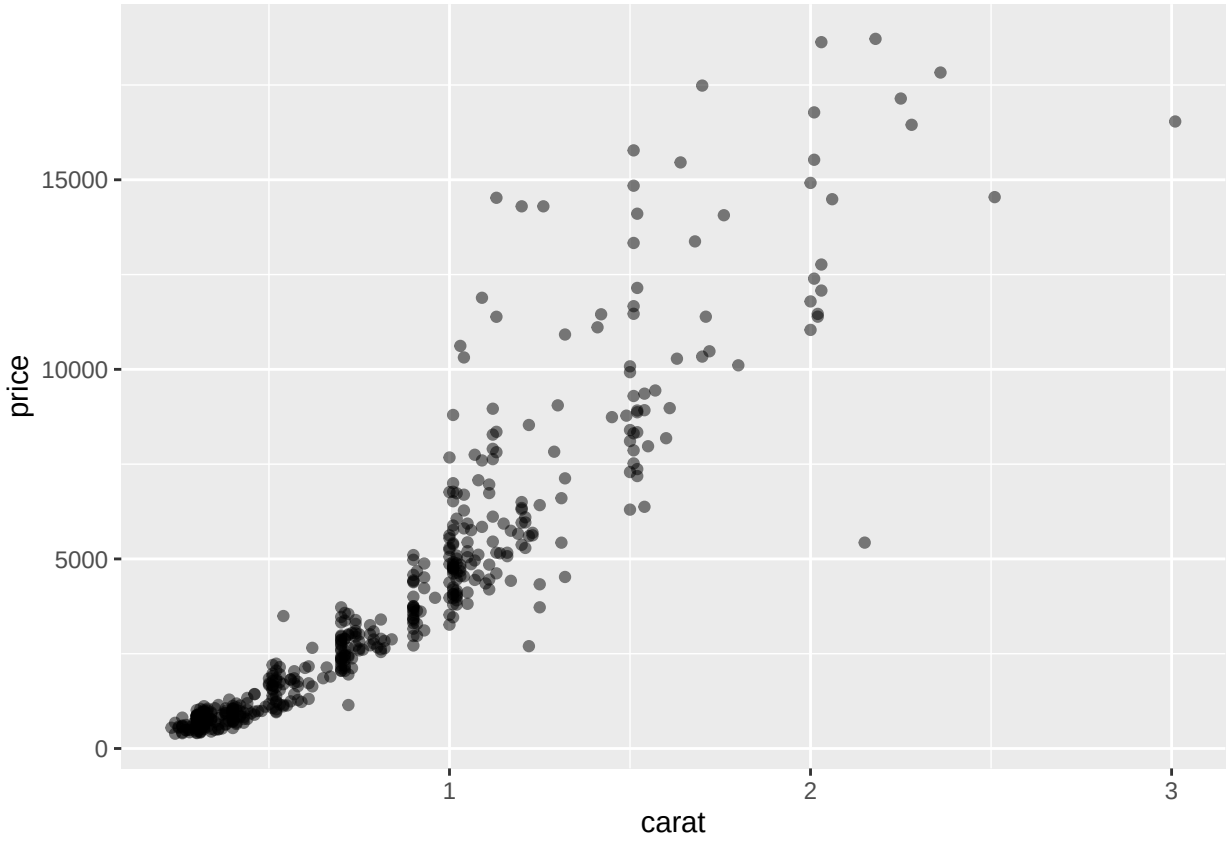
```
## [1] 53940
```

It's often not effective to create plots, especially scatter plots, with so many data points. Let's use the `sample()` function to extract a random sample of 500 cases. `sample()` produces 500 random indices between 0 and 53940, which we use to retrieve random records from the diamond data set. Using `set.seed()` makes the code reproducible, by making sure you always pick always the same sample.

```
set.seed(2)
sampleDiamonds <- diamonds[sample(nrow(diamonds), 500), ]
```

Using the sampleDiamonds data set shows a much cleaner chart:

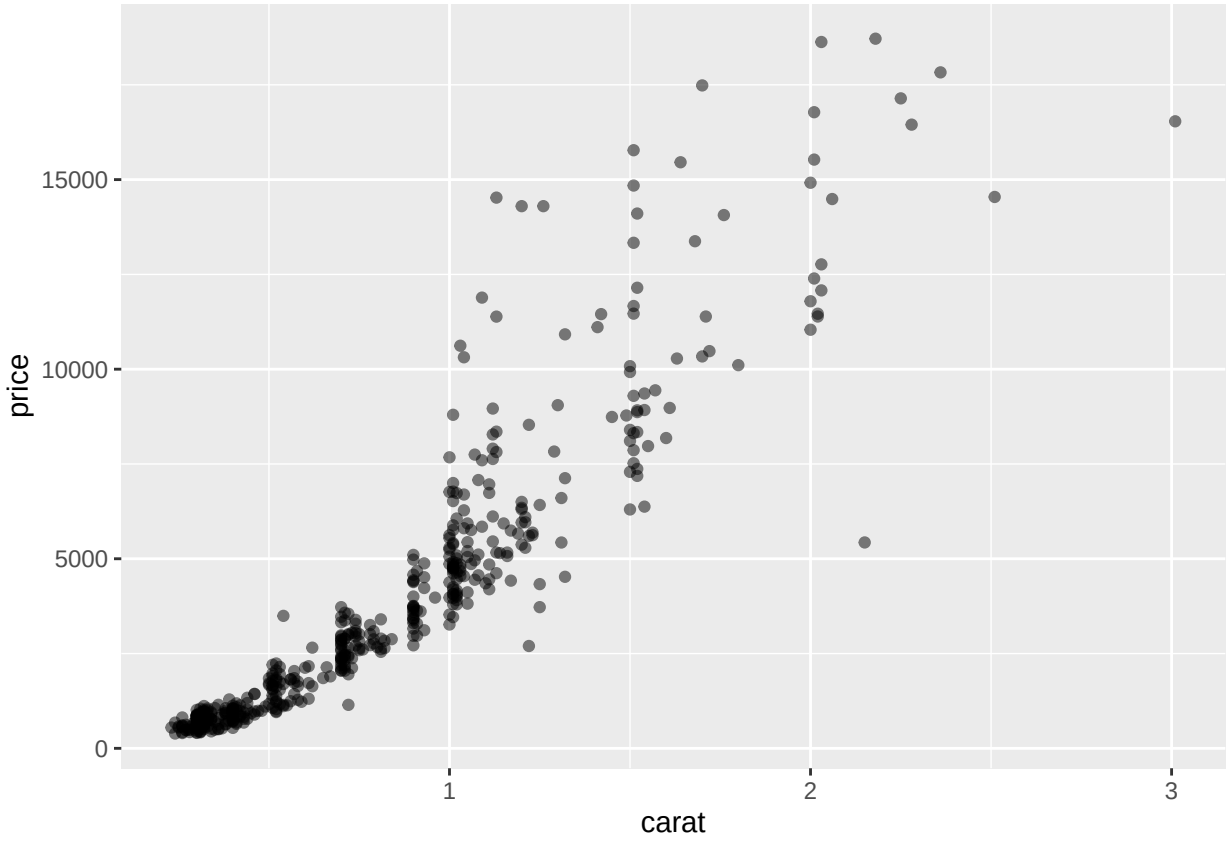
```
ggplot(sampleDiamonds) + aes(carat, price) + geom_point(alpha = 0.5)
```



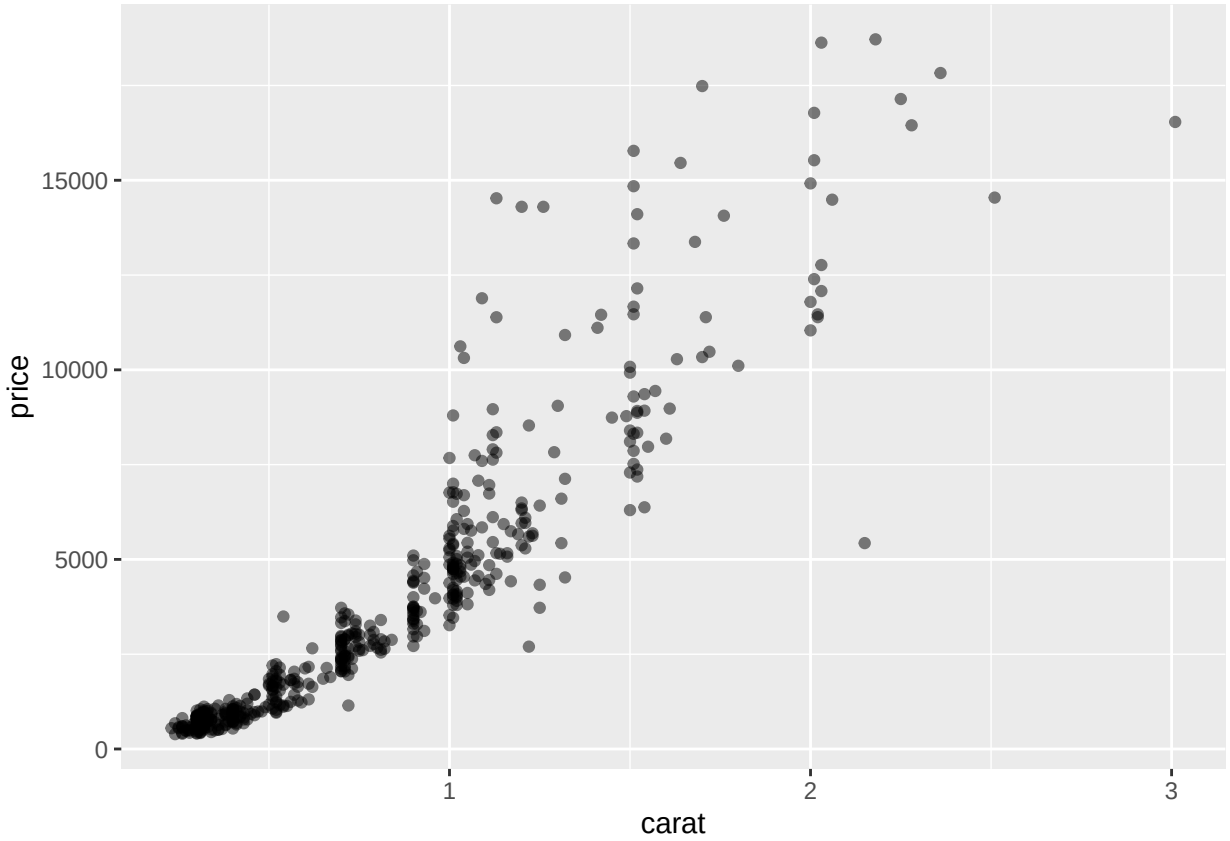
Same same but different

ggplot2 is very flexibly (and somewhat forgiving), so you will sometimes see plots created with slightly different syntax. The following lines all produce the same chart as above. The difference in these examples is merely theoretical.

```
ggplot(sampleDiamonds) + geom_point(aes(carat, price), alpha = 0.5)
```




```
ggplot() + geom_point(aes(carat, price), alpha = 0.5, data = sampleDiamonds)
```

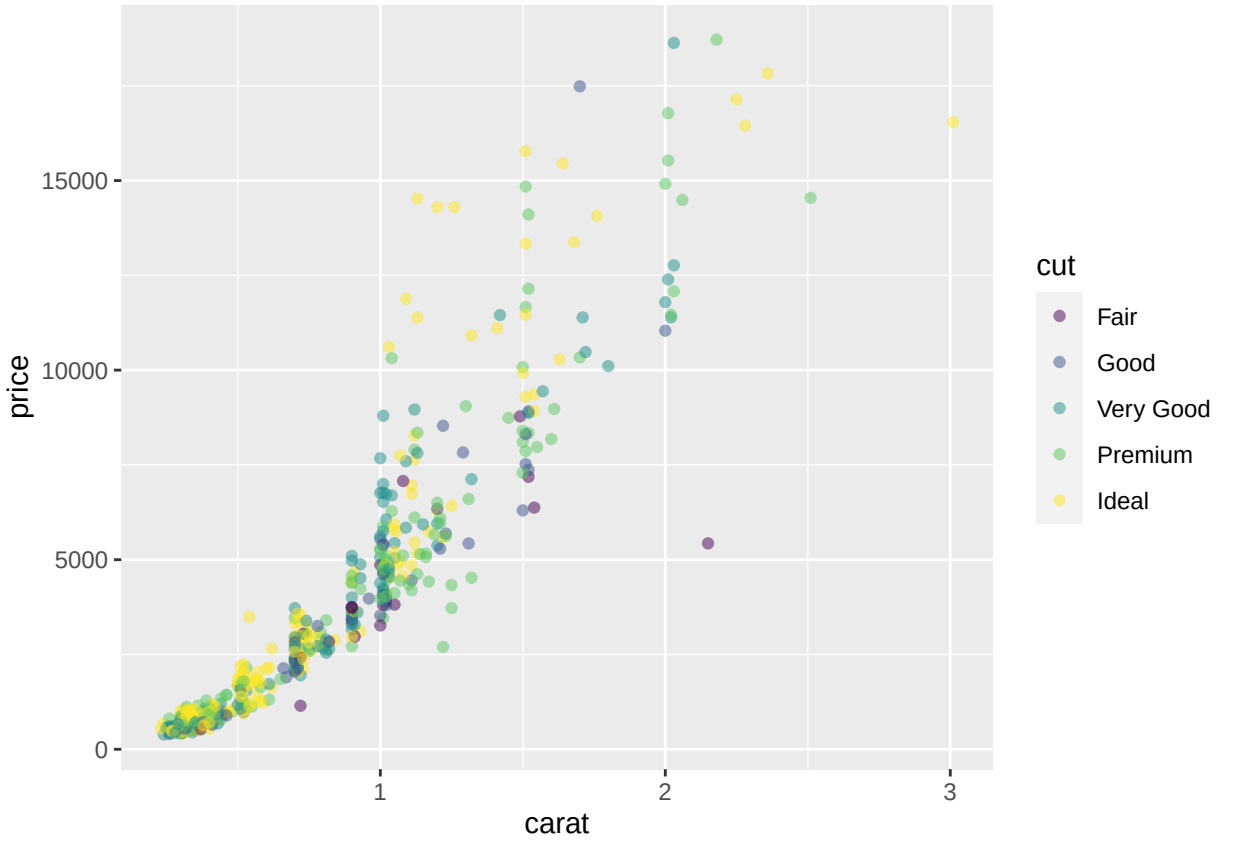

price
15000
10000
5000
0
1
2
3
carat

Additional mappings

So far, we've plotted two dimensions, weight in carat and price. `ggplot2` makes it very easy to add more dimensions. We only need to add another aesthetic mapping.

Here we plot carat by price and cut.

```
ggplot(sampleDiamonds) + aes(carat, price, colour = cut) + geom_point(alpha = 0.5)
```



Not surprisingly, better cuts are more expensive.

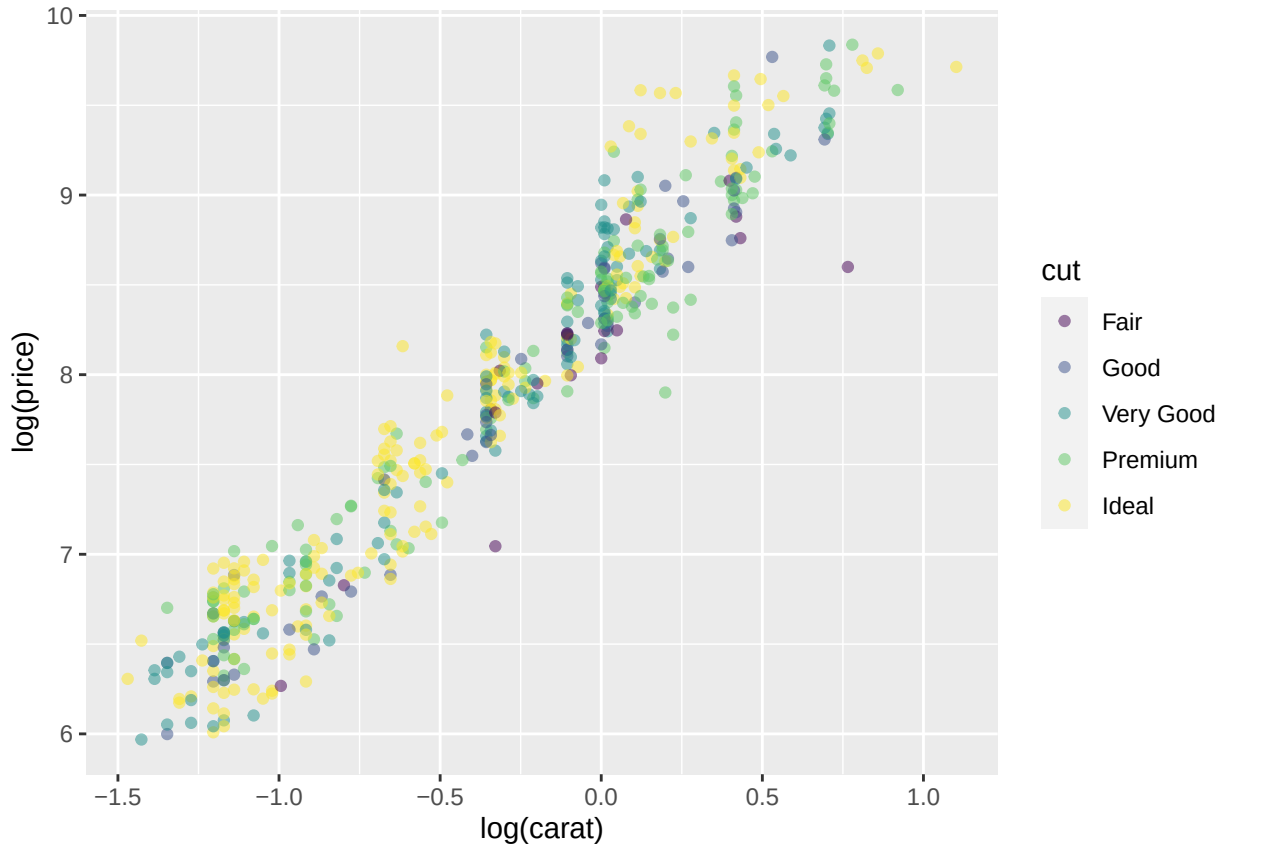
The exact aesthetics available are dependent on the geometry layer - a scatter plot has different aesthetic mappings than a line plot or a bar plot. The Data visualization cheat sheet provides a good overview of the available aesthetic mappings. The help for the respective geom_* function will also show what aesthetic mappings are available.

```
? geom_point
```

Derived dimensions

On the chart above, we see that prices are rising exponentially, indicating that a log-log plot might be useful. A log-log plot uses a logarithmic scale for both the x and y-axis. Creating a log-log plot in R is easy; we simply need to wrap the log function around each dimension in the aesthetic mapping:

```
ggplot(sampleDiamonds) + aes(log(carat), log(price), colour = cut) + geom_point(alpha = 0.5)
```

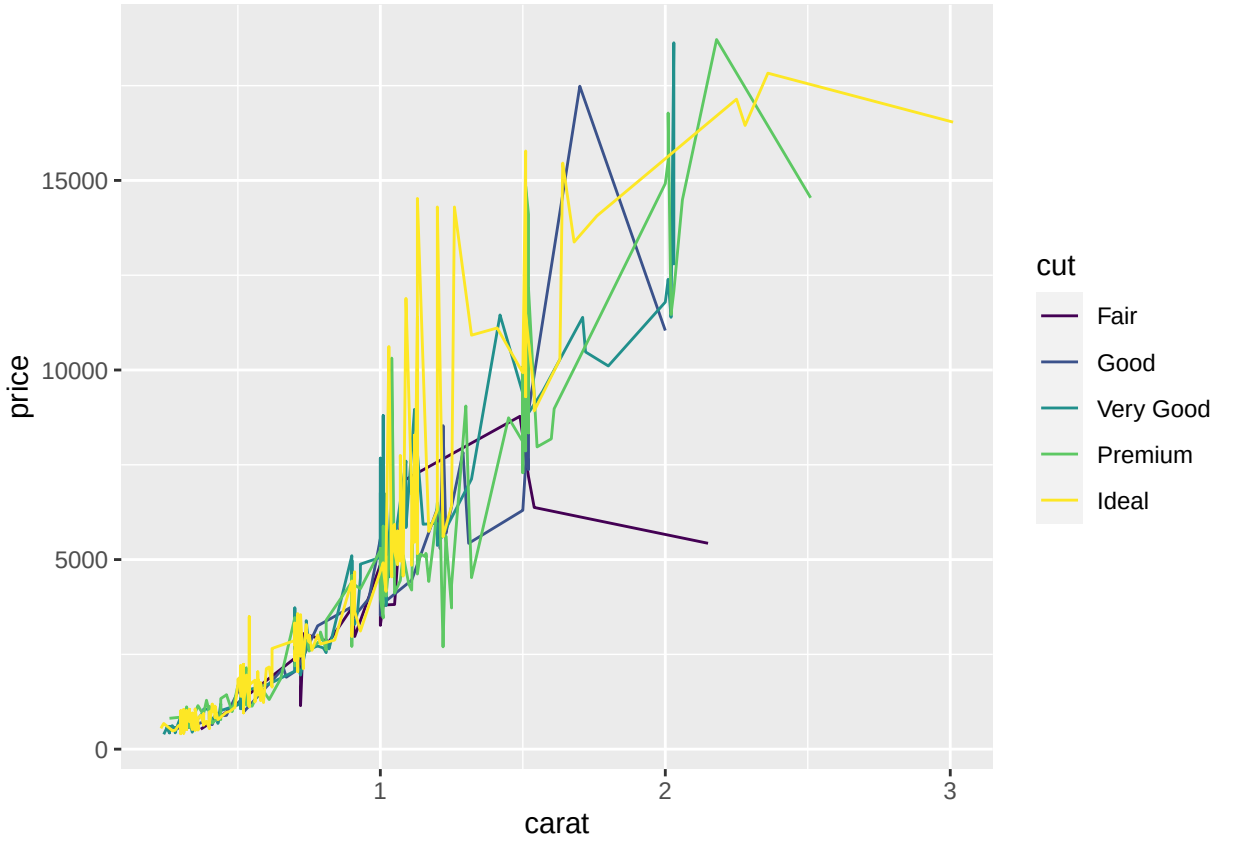


Other plot types

We create other plot types, such as a line chart or a bar plot, by adding different geometry layers.

Let's construct a line plot with `geom_line` (note, we store the coordinate system and aesthetic mappings in the variable `diamondPlot`, so we can re-use it).

```
diamondPlot <- ggplot(sampleDiamonds) + aes(carat, price, color = cut)
diamondPlot + geom_line()
```



Note that this code is equivalent to the above:

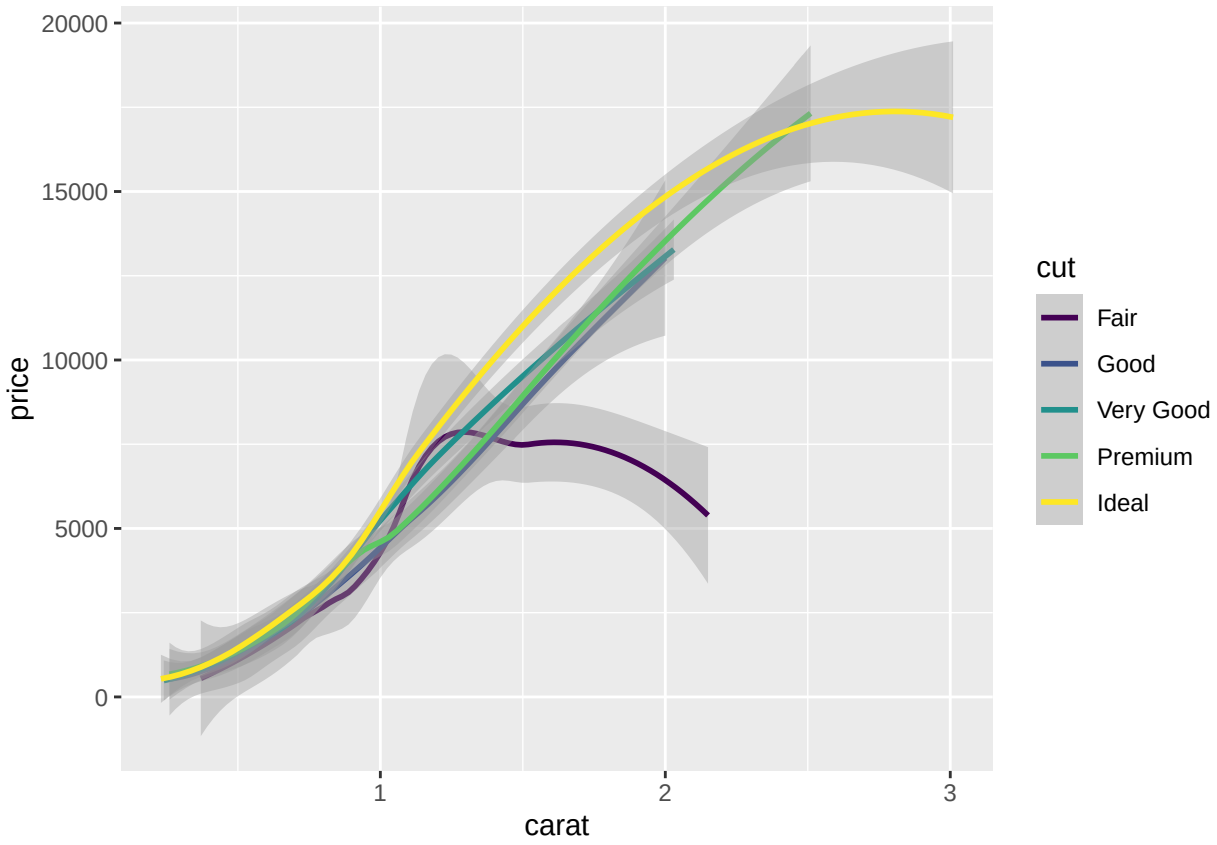
```
ggplot(sampleDiamonds) + aes(carat, price, color = cut) + geom_line()
```

Trend lines

Plotting smoothed averaged values using `geom_smooth()`.

```
diamondPlot + geom_smooth()
```

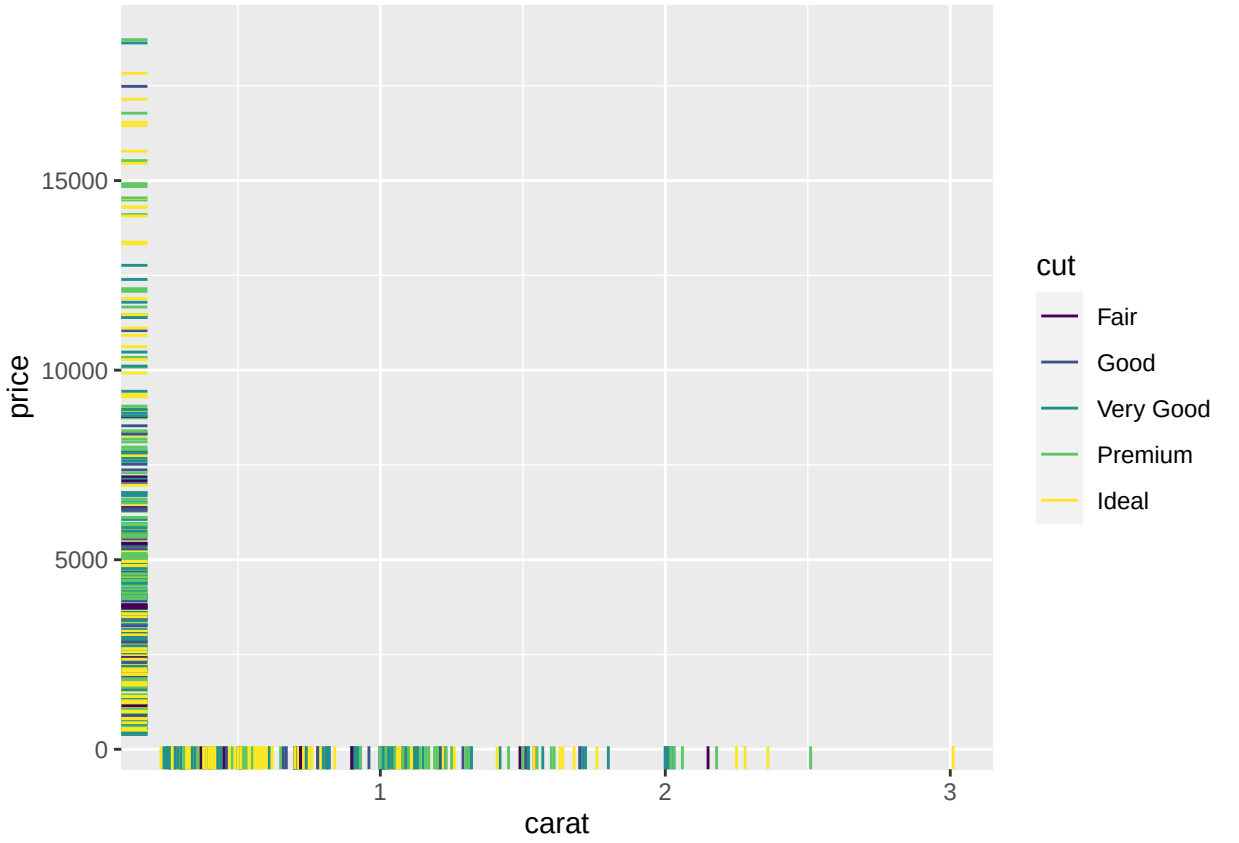
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```


20000
15000
10000
5000
0
price
1
2
3
carat
cut
Fair
Good
Very Good
Premium
Ideal

Rug plot

Or let's create a (somewhat obscure) rug plot with `geom_rug` (the rug plot is useful to combine with other geometries).

```
diamondPlot + geom_rug()
```



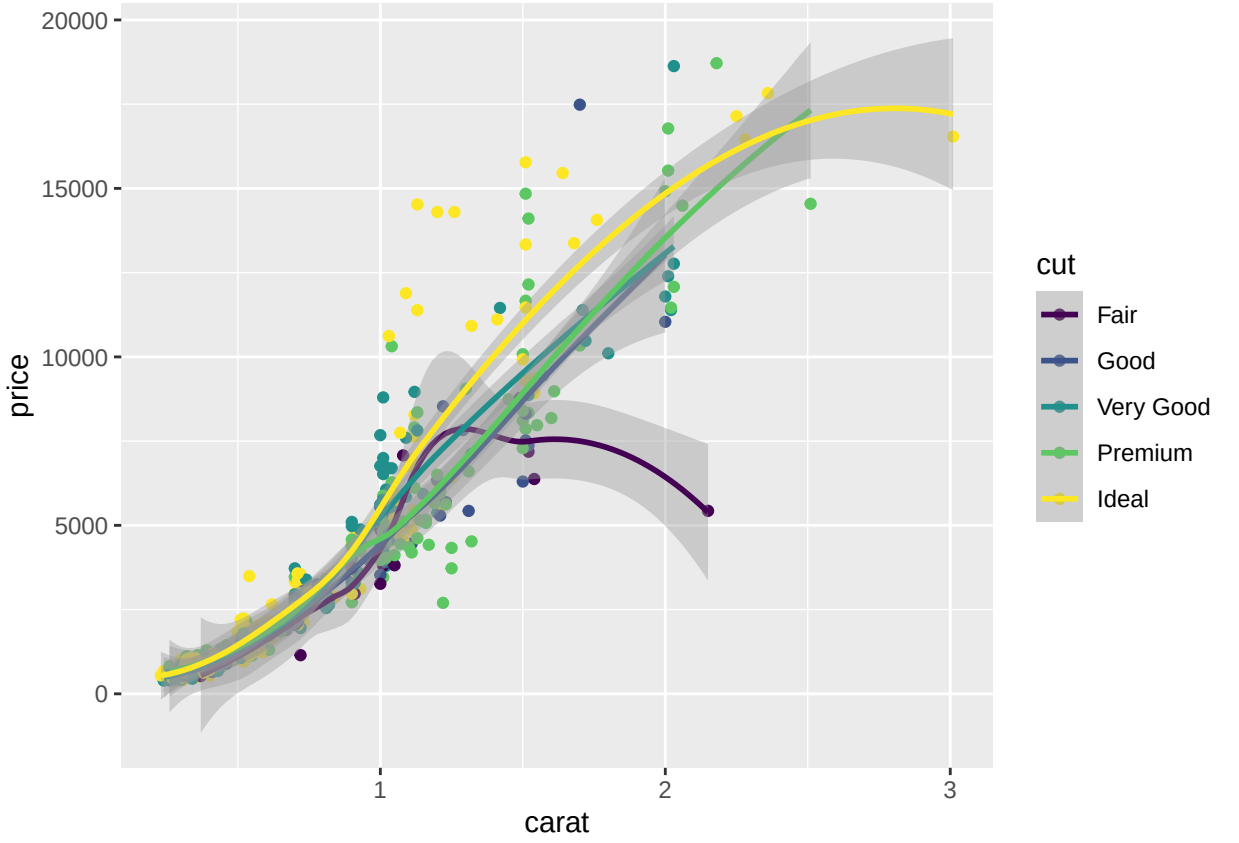
Adding multiple geometries

One of the many advantages of `ggplot2` is how easy it is to add multiple geometries in the same chart. We just need to add two or more geometries.

Let's overlay the smoothed means on the scatter plot:

```
ggplot(sampleDiamonds) + aes(carat, price, colour = cut) +
    geom_point() + geom_smooth()
```

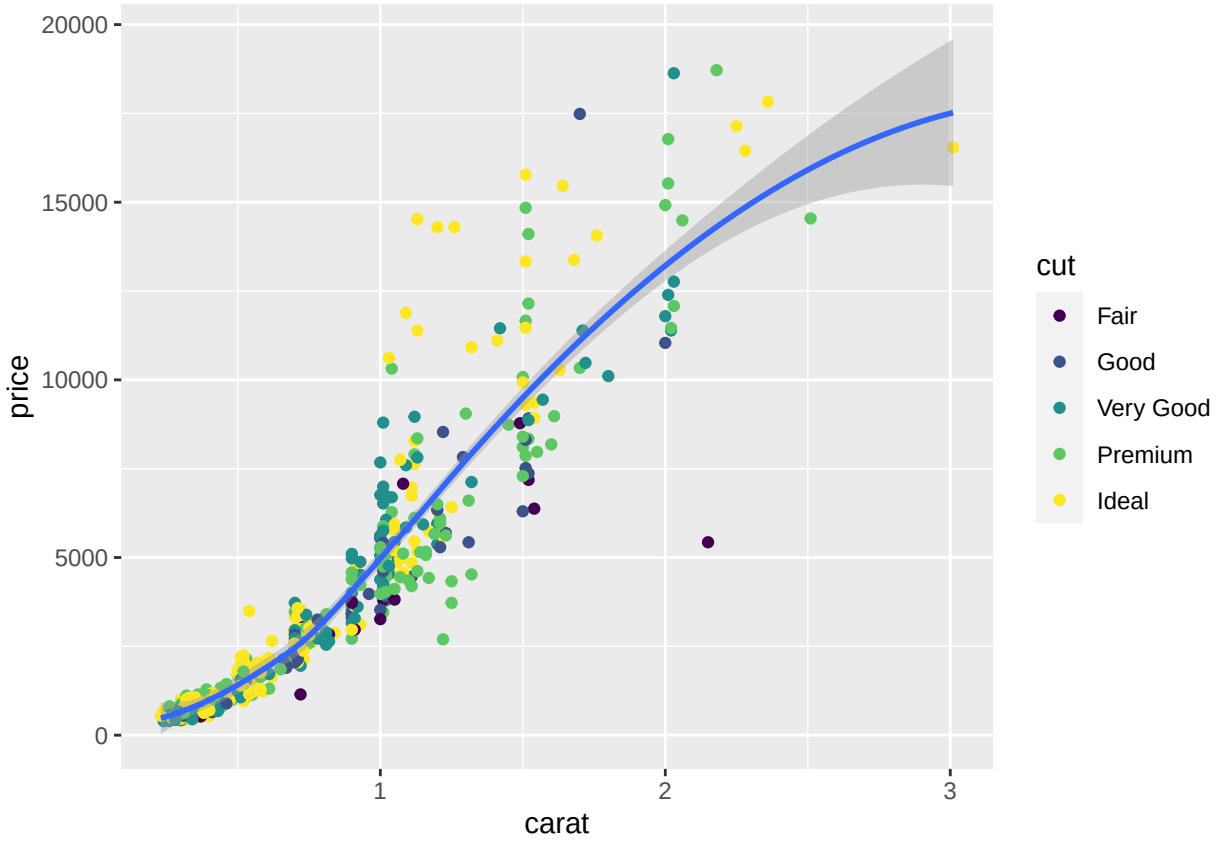
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



If we only want to fit one average line, we need to pass different aesthetic mappings to each geometry. Here, we tell geom_smooth to treat all cut grades as one data series.

```
ggplot(sampleDiamonds) +
    geom_point(aes(carat, price, colour = cut)) + geom_smooth(aes(carat, price))
```

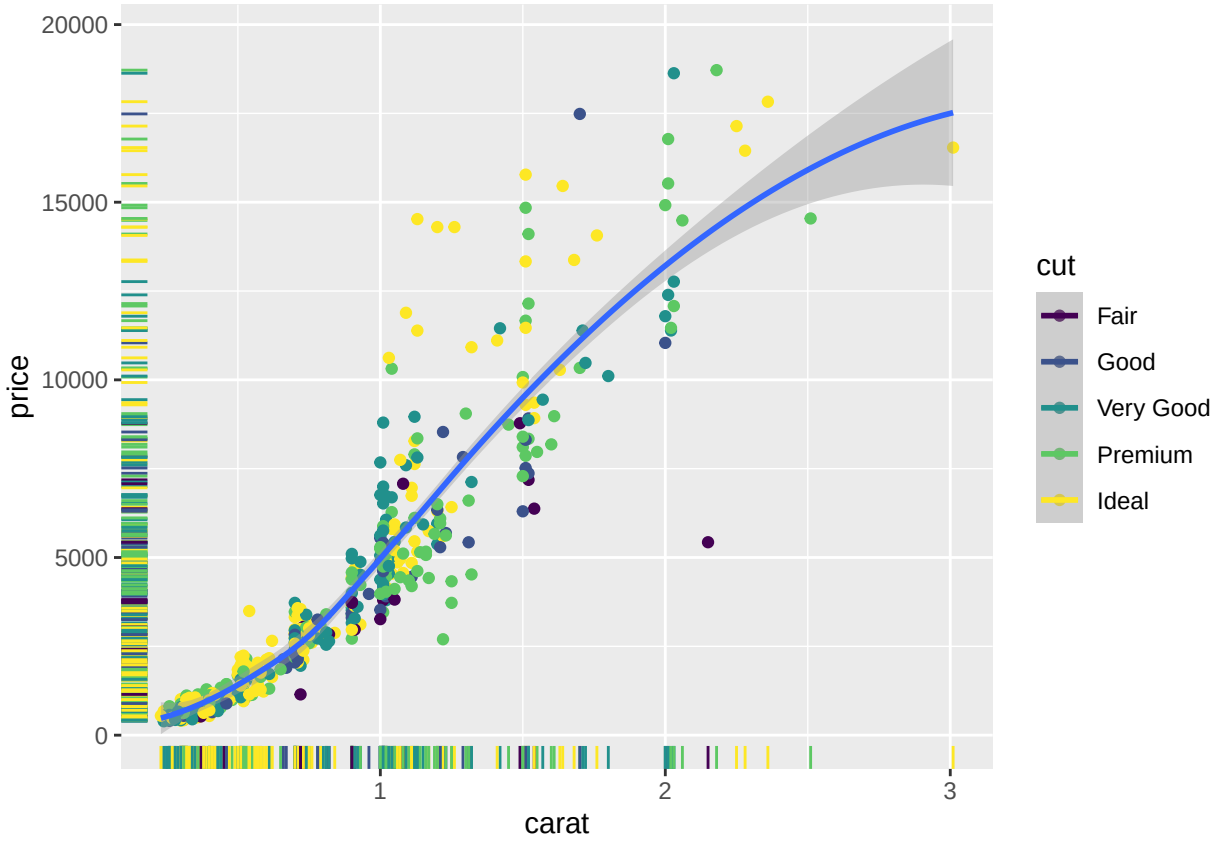
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



Of course, we can even add the rug geometry. Note how we specify one default aesthetic mapping for `geom_point()` and `geom_rug()`, and one specific mapping for `geom_smooth()`

```
ggplot(sampleDiamonds) + aes(carat, price, colour = cut) +
    geom_point() + geom_rug() + geom_smooth(aes(carat, price, colour = NULL))
```

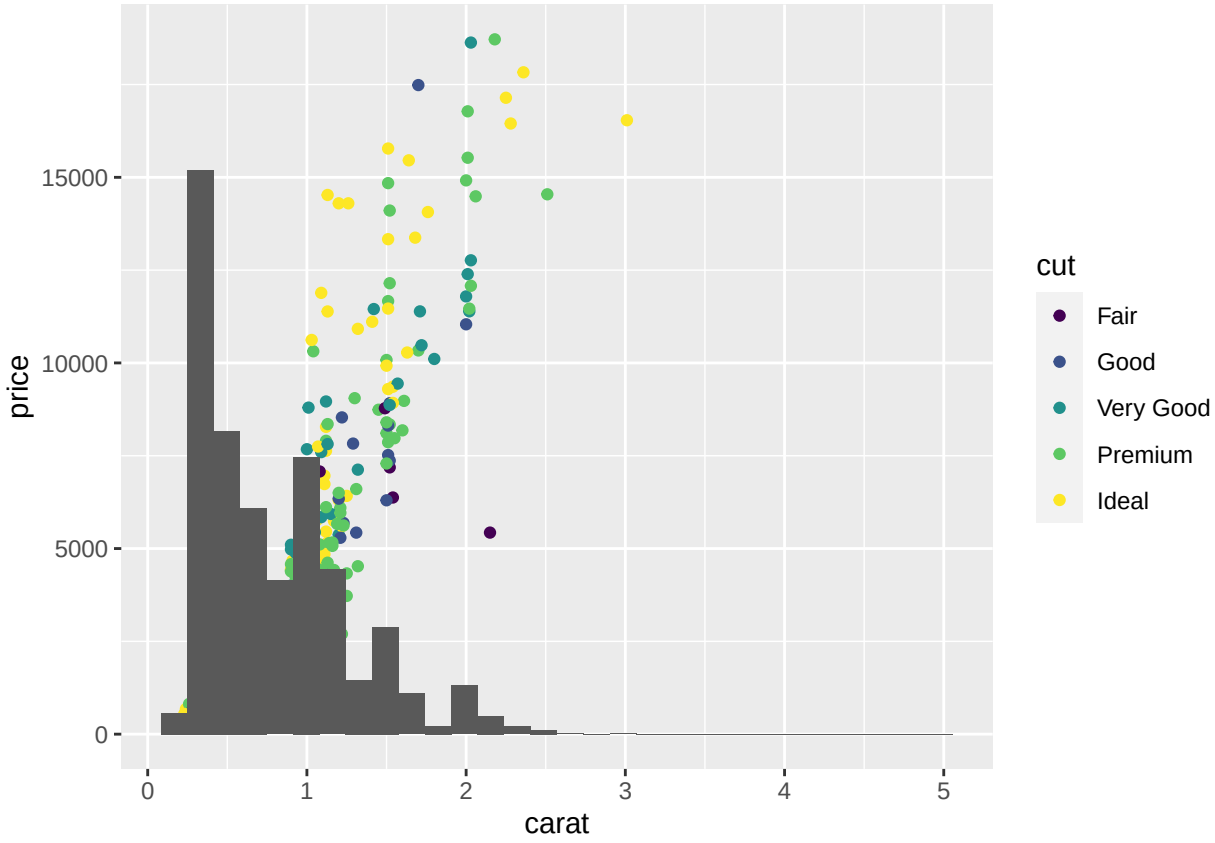
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



We can even combine geometries that may not make much sense, e.g. a scatter plot and a histogram.

```
ggplot(sampleDiamonds) + geom_point(aes(carat, price, colour = cut)) +
  geom_histogram(aes(carat), data = diamonds)
```

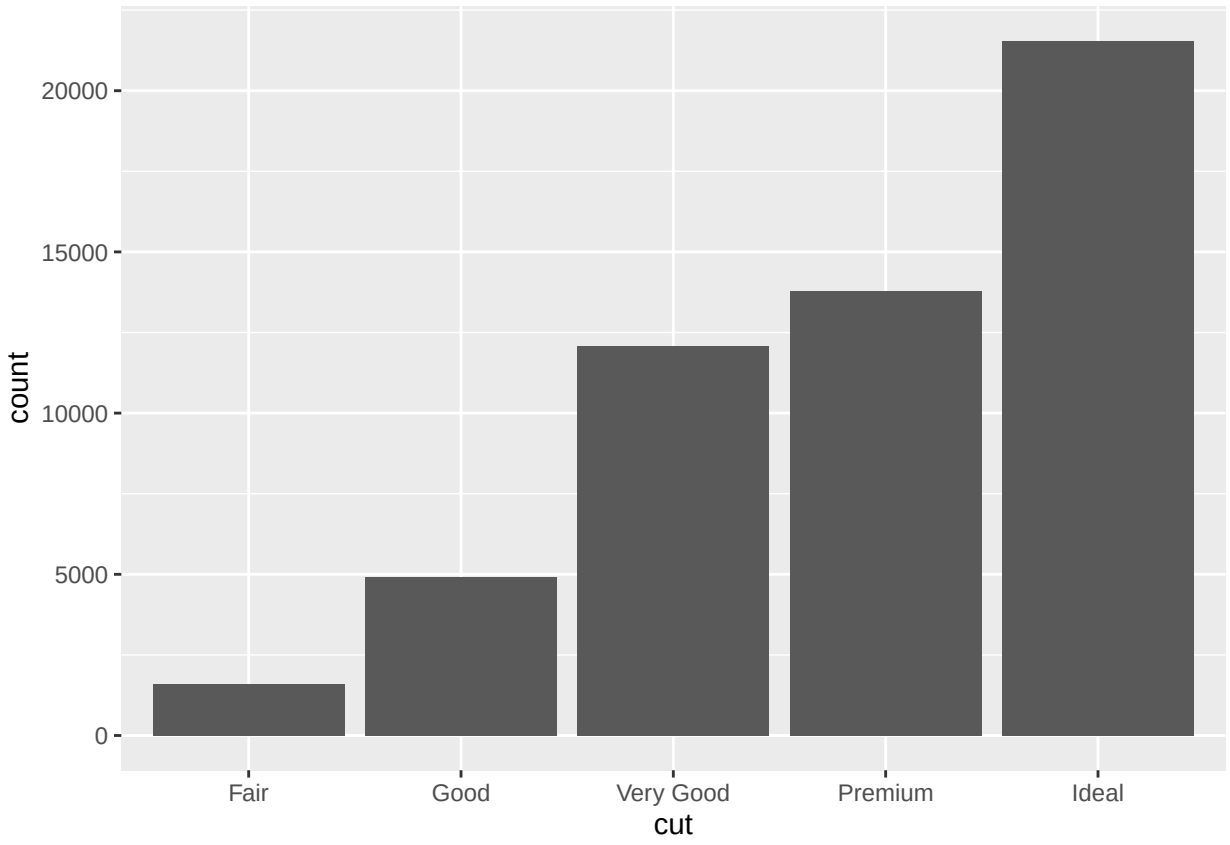
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


price
15000
10000
5000
0
0
1
2
3
4
5
carat
cut
Fair
Good
Very Good
Premium
Ideal

Bar chart

Let's create a bar chart showing the number of diamonds per cut grade. `ggplot2` assumes that we want to count records in a bar chart, therefore it does the counting for us, and we only need to pass the category in the aesthetic mapping.

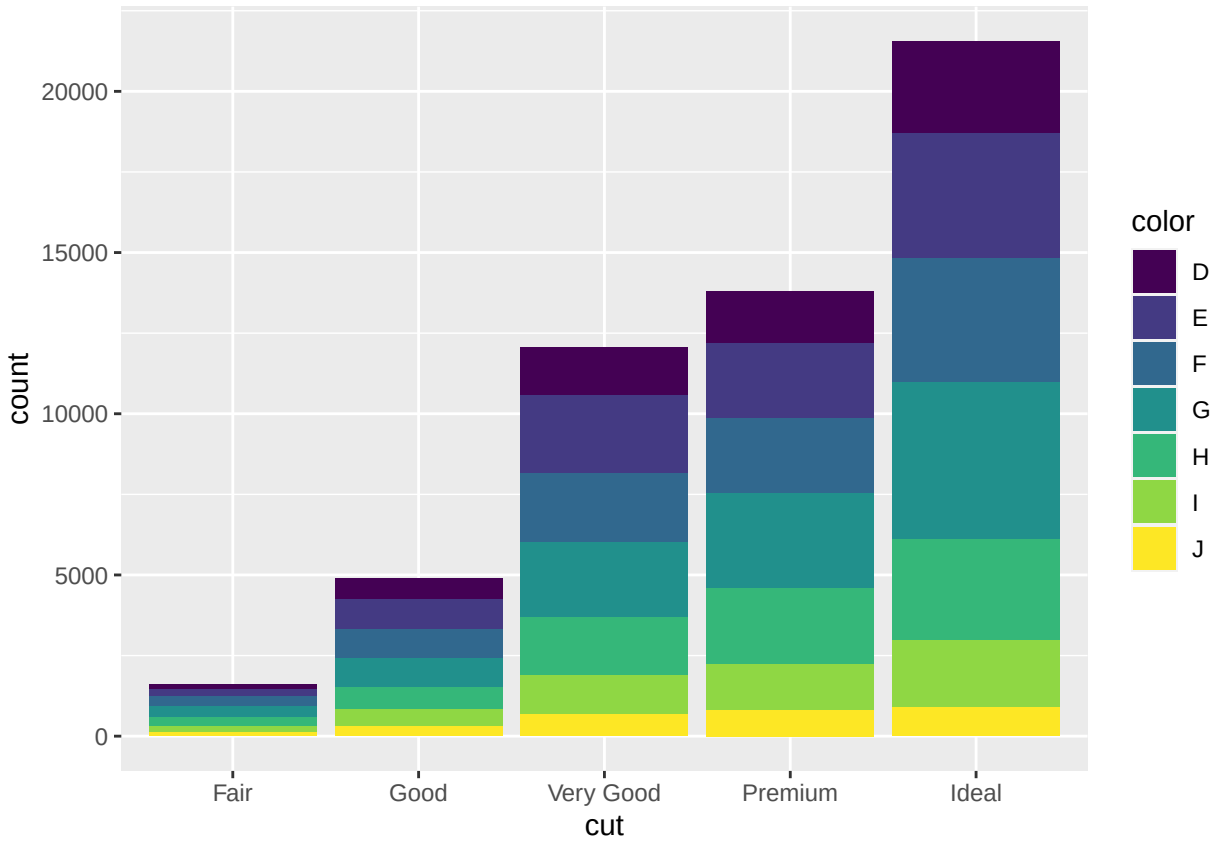
```
ggplot(diamonds) + aes(cut) + geom_bar()
```



Stacked bar chart

To create a stacked bar chart, we need to define an additional aesthetic mapping. Let's plot the number of diamonds per cut grade and clarity grade.

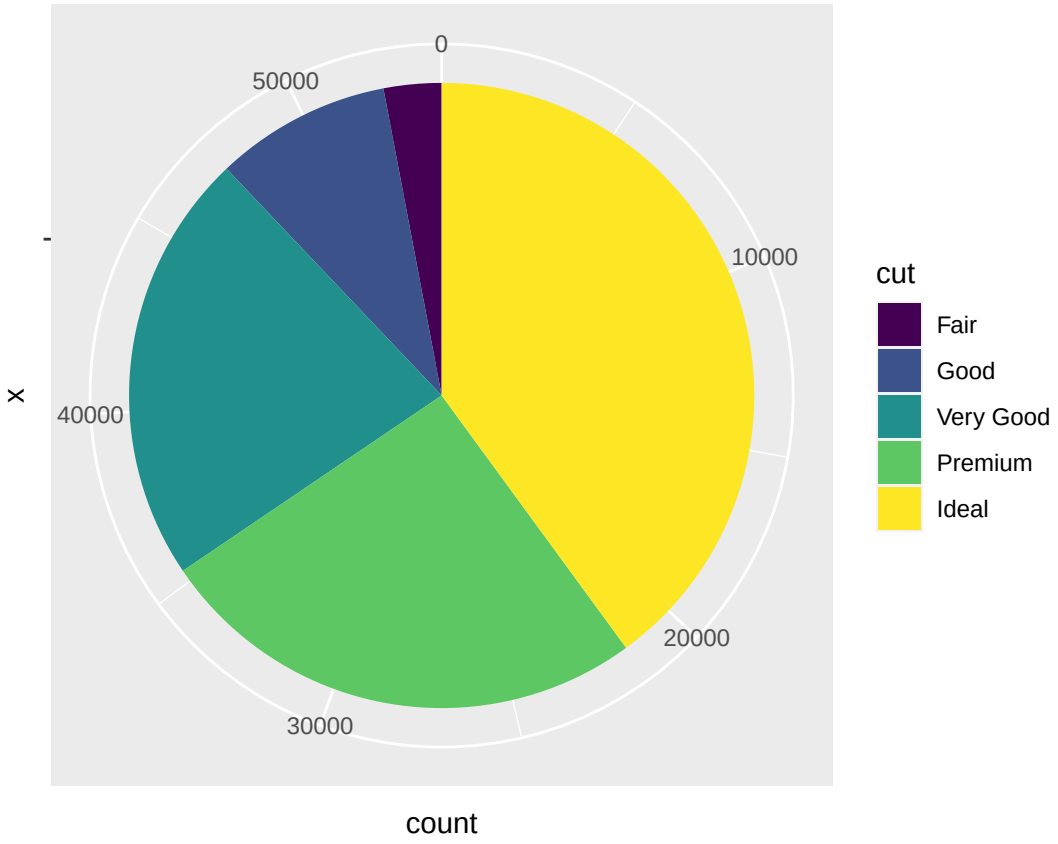
```
ggplot(diamonds) + aes(cut, fill = color) + geom_bar()
```


20000
15000
10000
5000
0
count
Fair
Good
Very Good
Premium
Ideal
cut
color
D
E
F
G
H
I
J

Pie chart

Alas, there is no native support for pie charts in `ggplot2`. However, it is still possible to create pie (and donut) charts, just not as straightforward as other charts. We can imagine a pie chart as a stacked bar chart with one bar wrapped around one point - or a round y-axis. To create a bar chart with one column, we map the `x` aesthetic to an empty element, and use the `fill` aesthetic to colour in the segments. `geom_bar` draws the bar chart and `coord_polar("y")` wraps the y axis.

```
ggplot(diamonds) + aes(x = "", fill = cut) + geom_bar() + coord_polar("y")
```

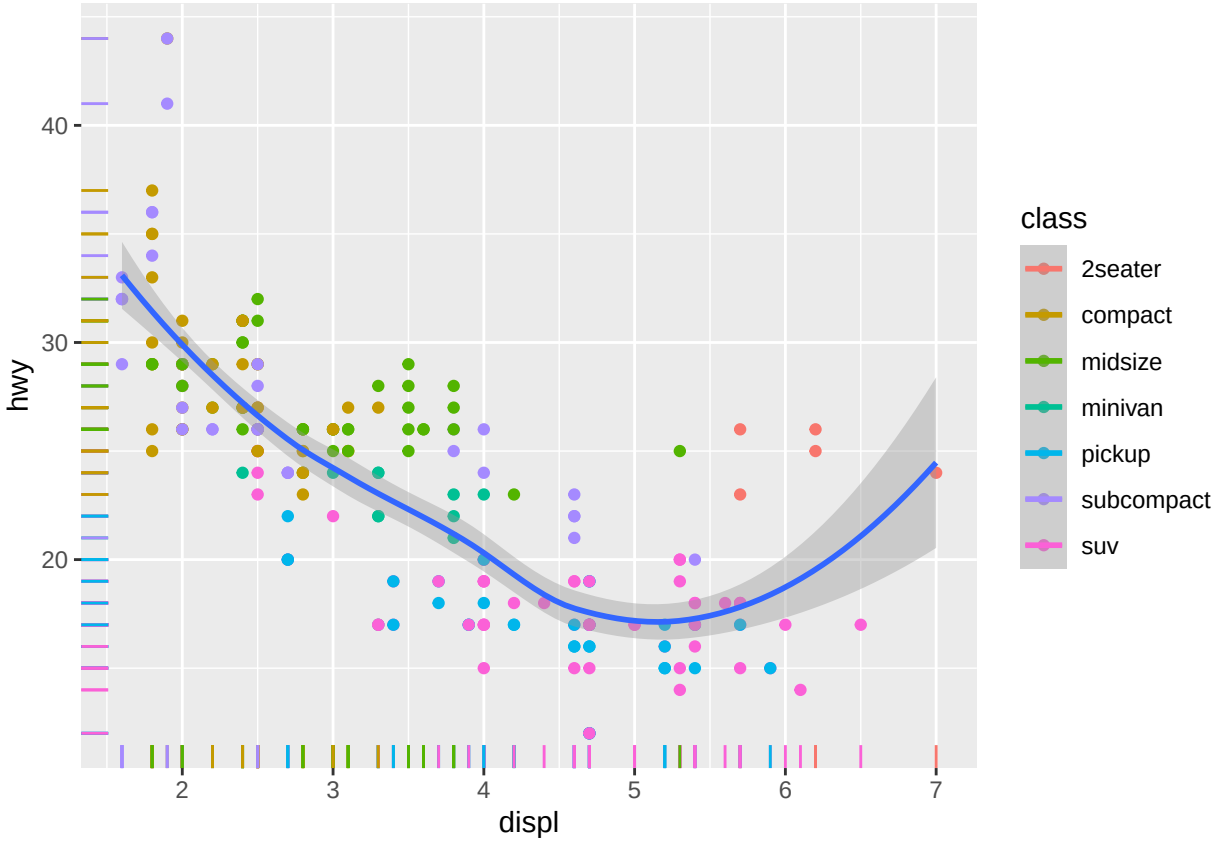


Exercise

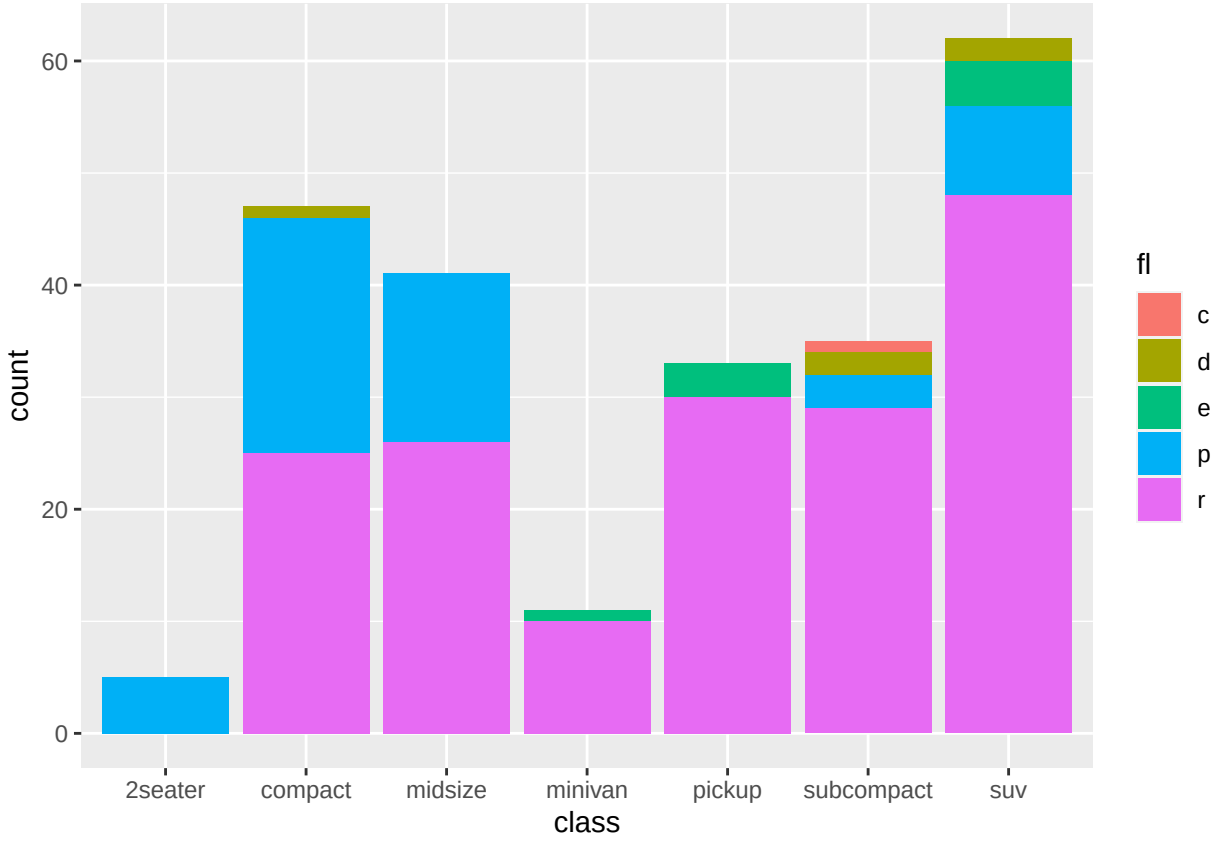
Use `ggplot2` and the `mpg` data set to create the following graphics.

1. Familiarise yourself with the built-in `mpg` data set. Note that the data set is only available after you load the `ggplot2` library.
2. Create a scatter plot of the `mpg` data set showing displacement by highway miles.
3. In the same chart, map the city miles per gallon to the `size` aesthetic. What does the chart show? Did you expect this result?
4. Colour the dots with the type of car.
5. Remove the size mapping and overlay a rug chart.
6. On the same chart add one smooth line showing displacement by highway miles trend. The chart shall look like this:

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

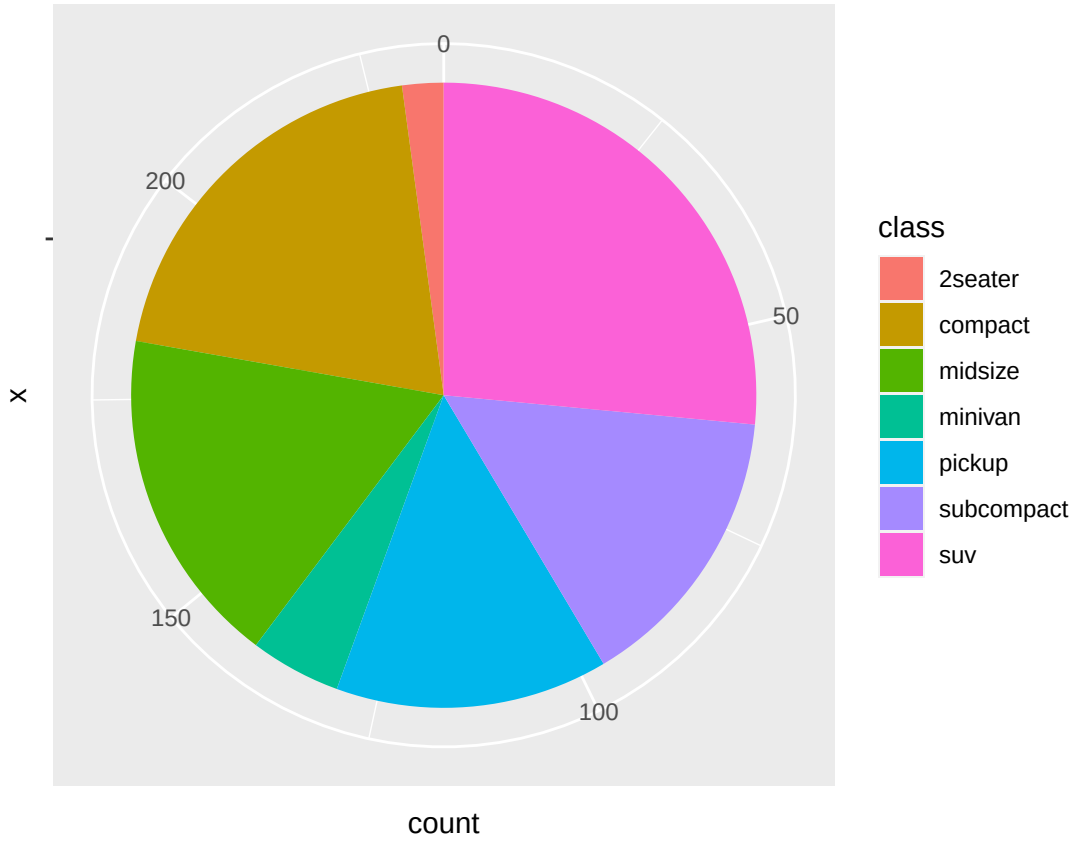


7. Create a bar chart showing the number of cars by type of car.
8. Create the same bar chart, showing also the fuel type as stacked bars.



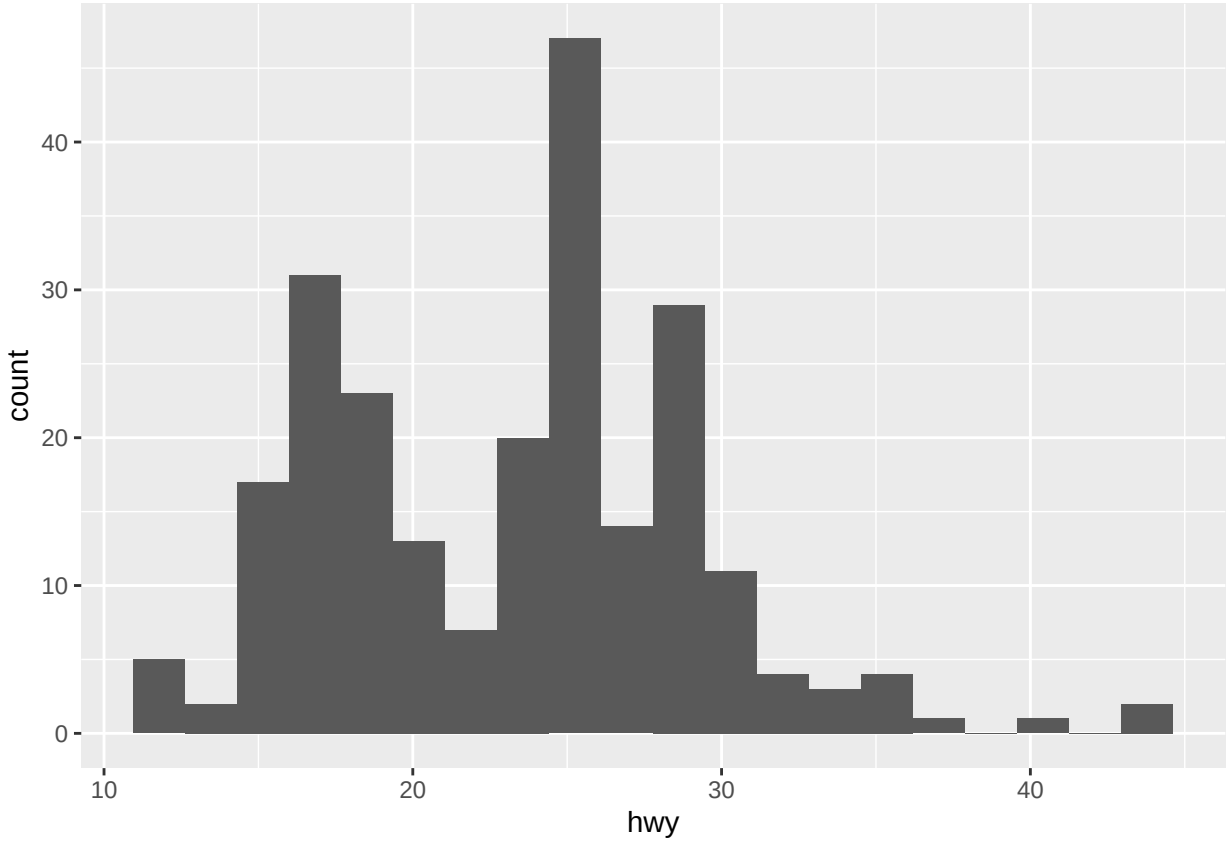
9

. Create a pie chart showing the number of cars by car type.



10

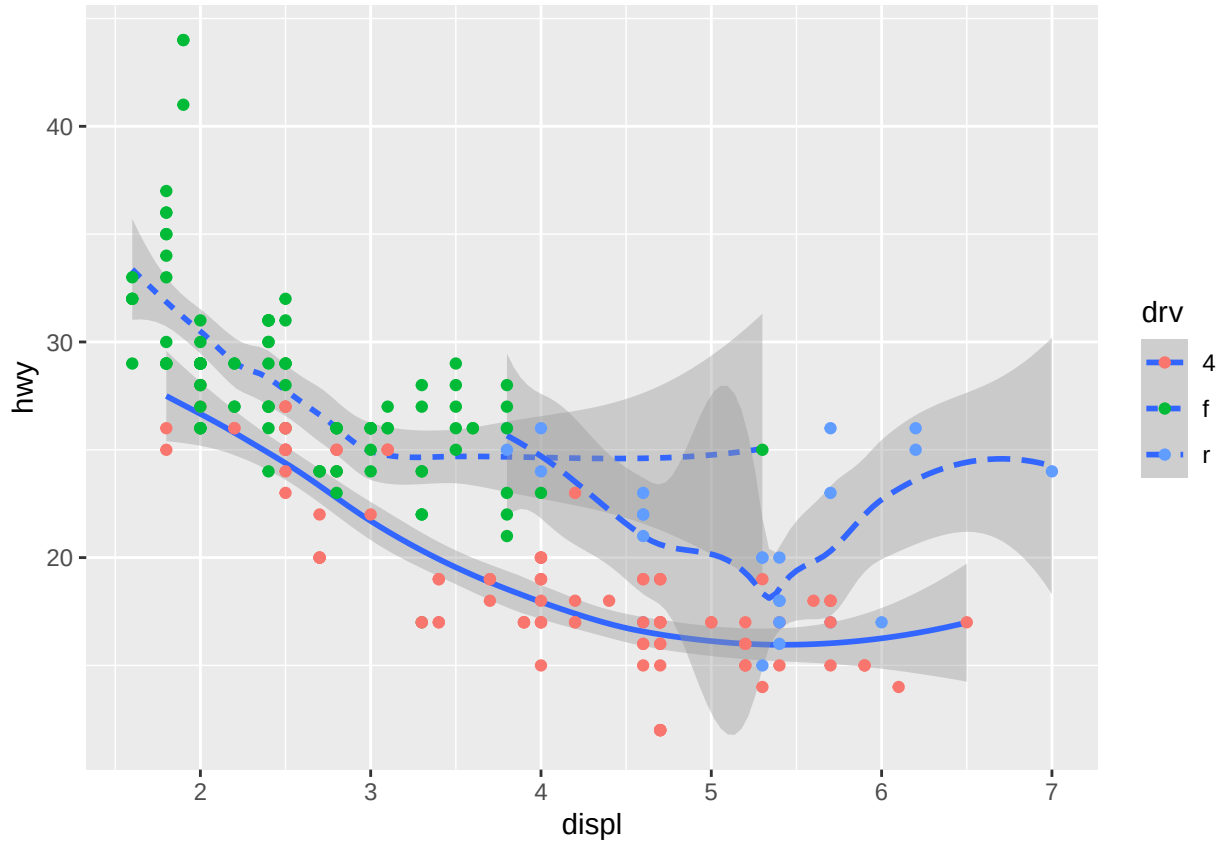
. Create a histogram showing the distribution of the highway miles per gallon in 20 bins.



Bonus 1: Explore additional chart types in `ggplot2`. Use the Data visualization cheat sheet.

Bonus 2: Create the following chart.

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



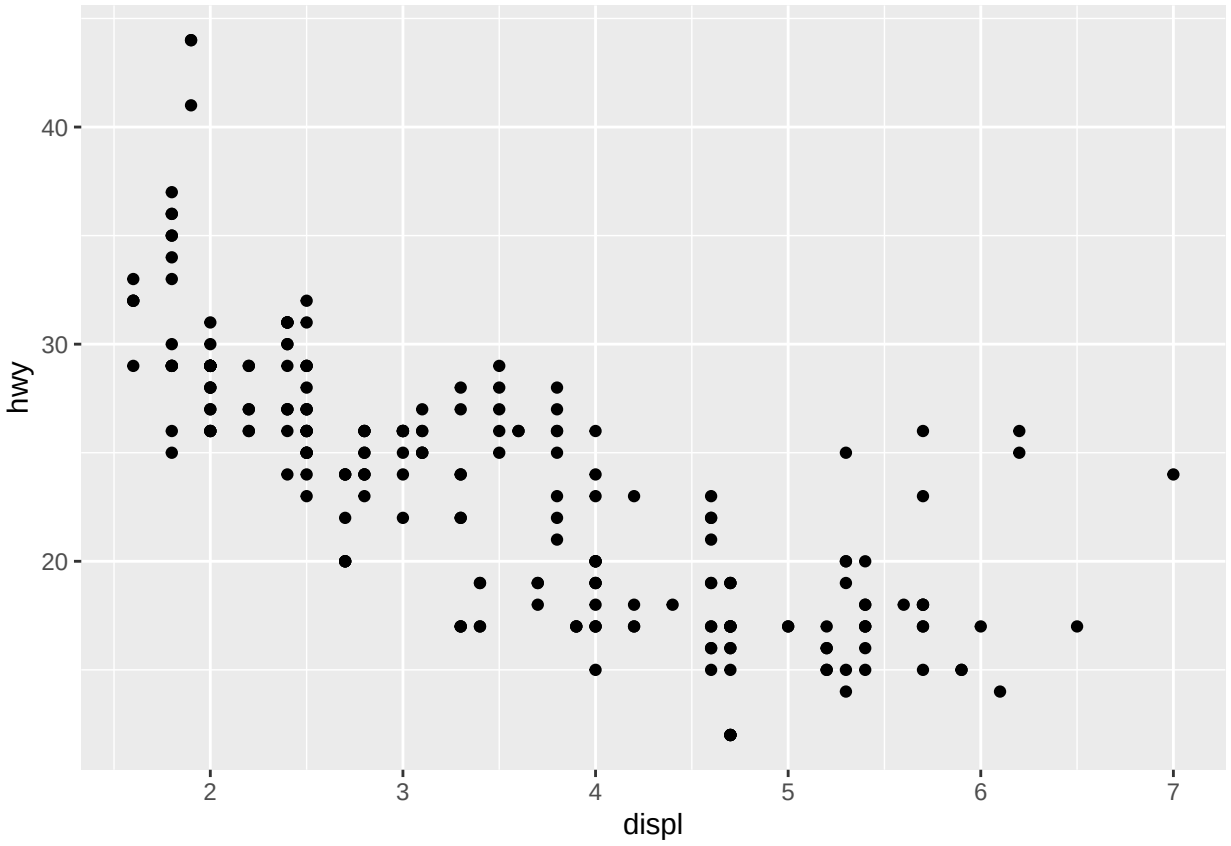
Solutions

1. Familiarise yourself with the built-in `mpg` data set. Note that the data set is only available after you load the `ggplot2` library.

```
? mpg
```

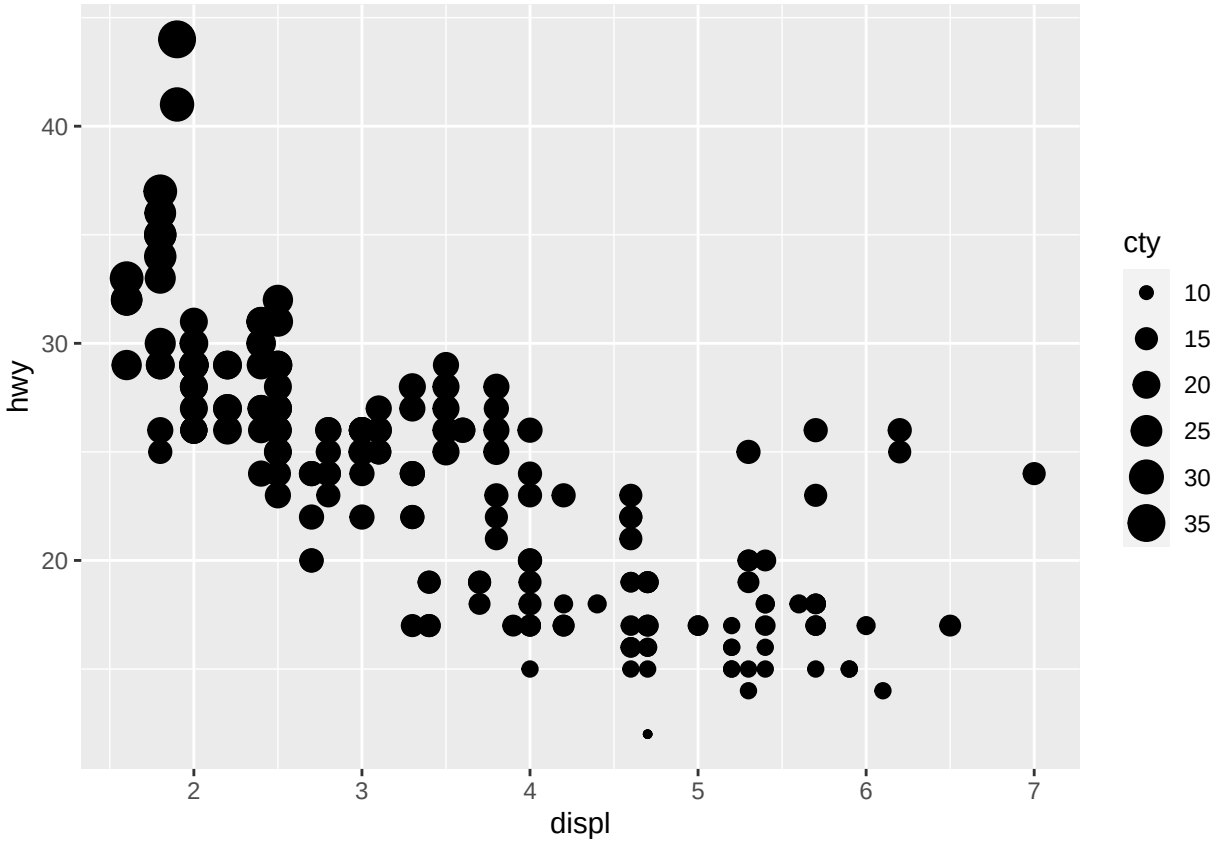
2. Create a scatter plot of the `mpg` data set showing displacement by highway miles.

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



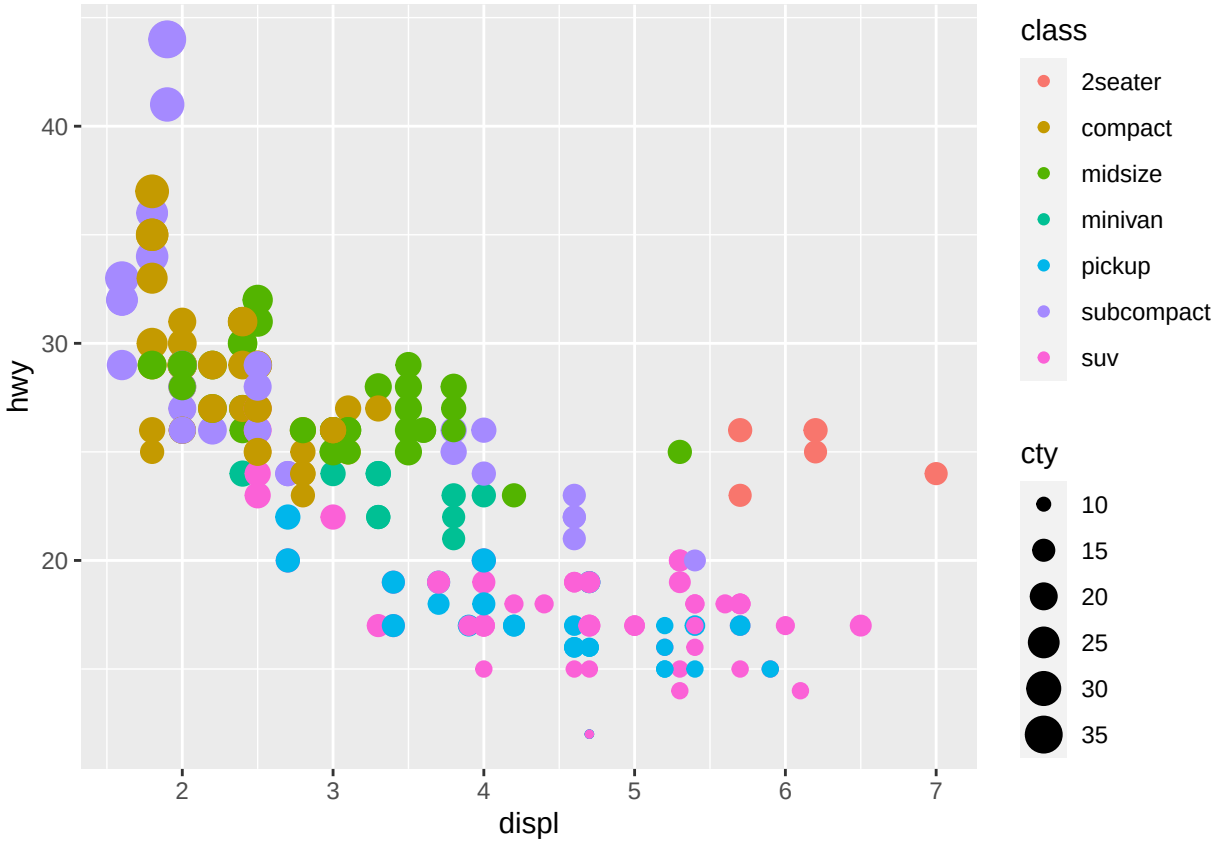
3. In the same chart, map the city miles per gallon to the `size` aesthetic. What does the chart show? Did you expect this result?

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



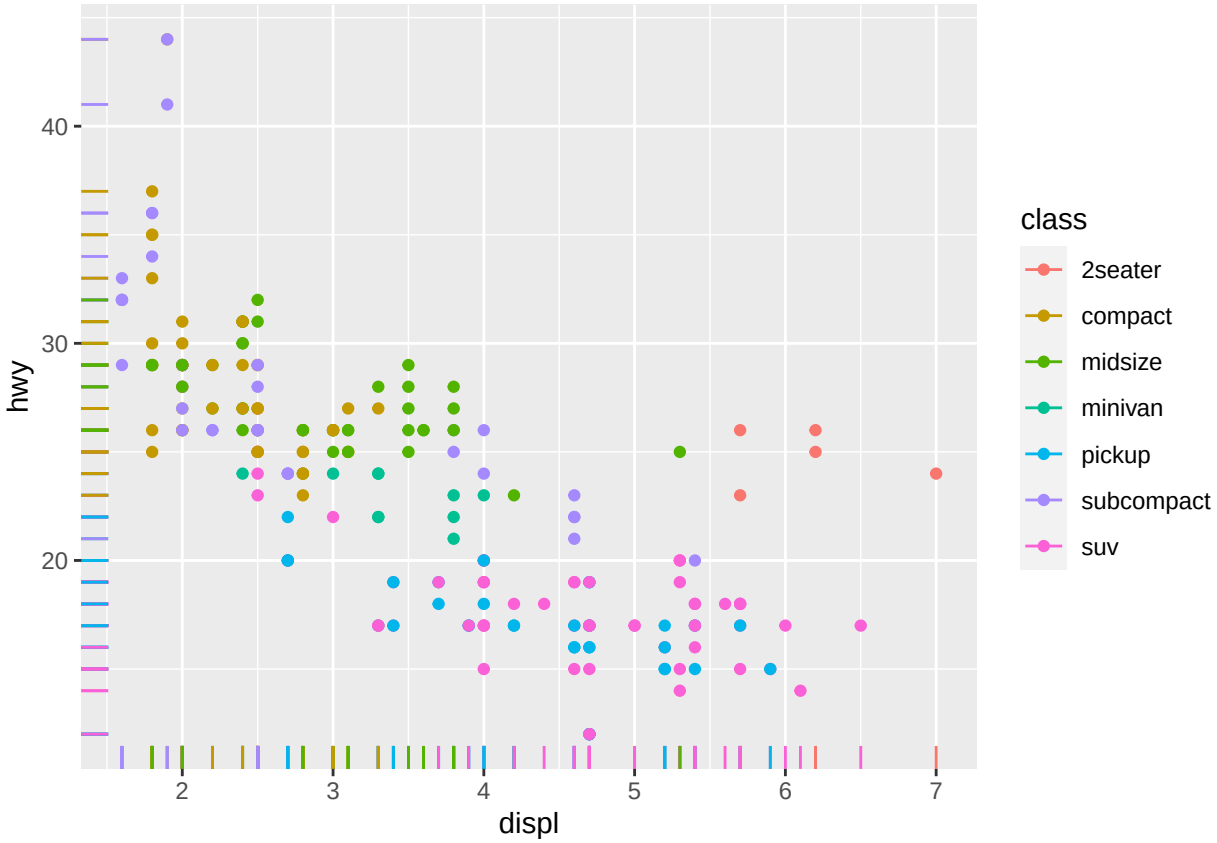
4. Colour the dots with the type of car.

```
ggplot(mpg) + aes(displ, hwy, size = cty, colour = class) + geom_point()
```



5. Remove the size mapping and overlay a rug chart.

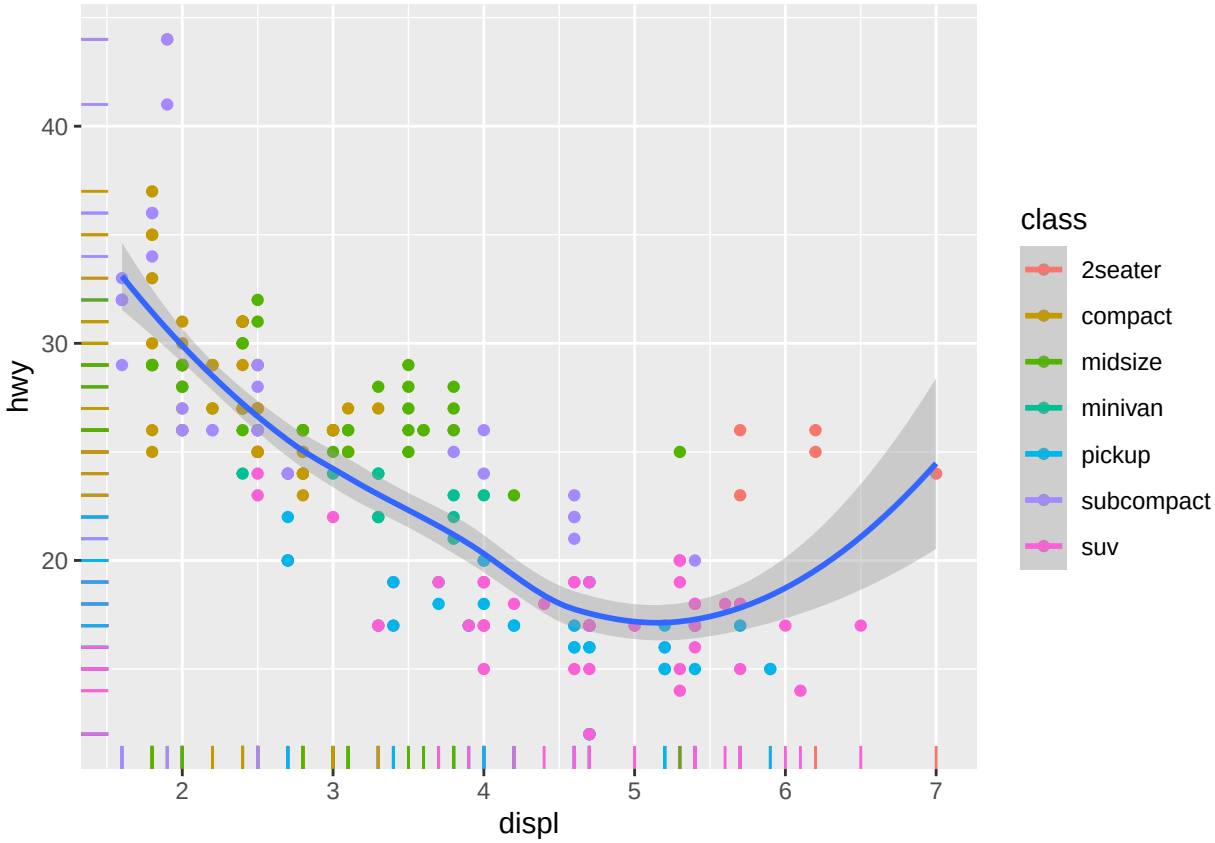
```
ggplot(mpg) + aes(displ, hwy, colour = class) + geom_point() +
  geom_rug()
```



6. On the same chart add one smooth line showing displacement by highway miles trend. The chart shall look like this:

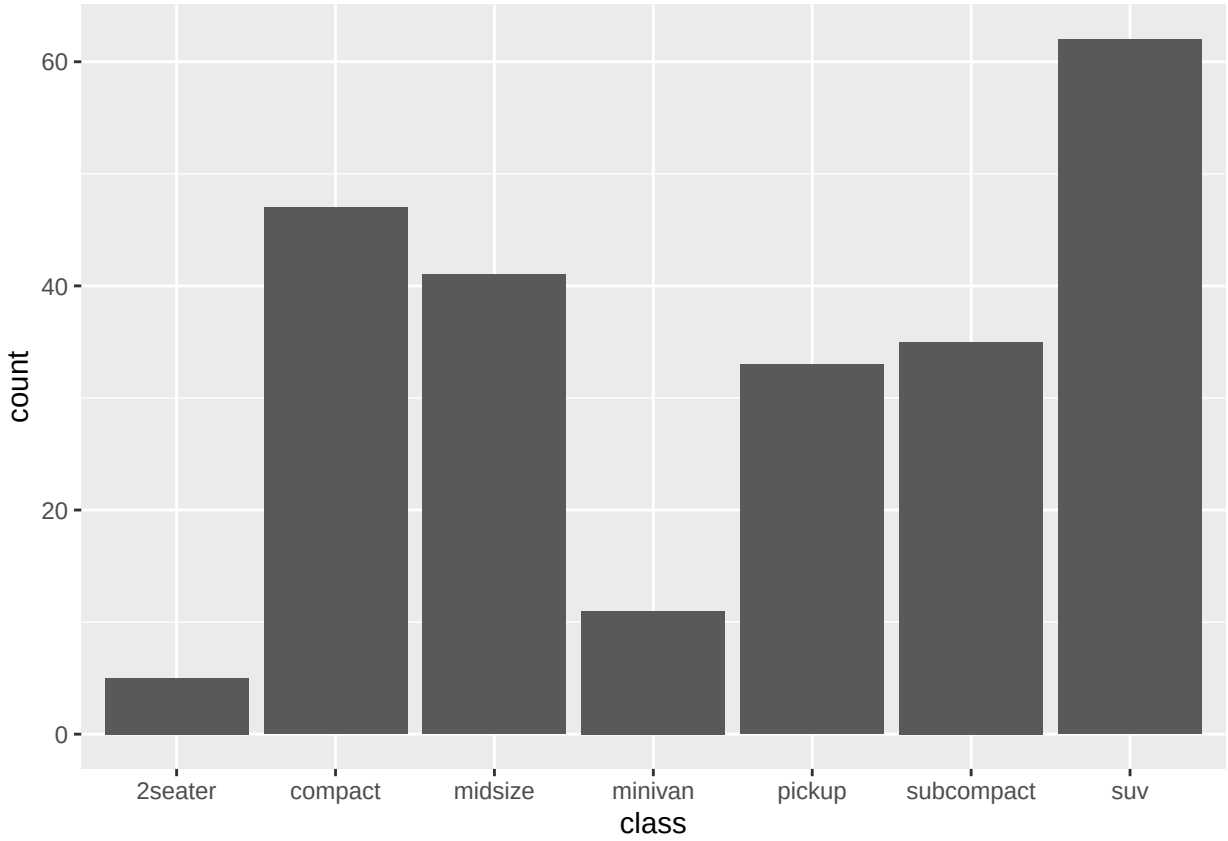
```
ggplot(mpg) + aes(displ, hwy, colour = class) + geom_point() +
  geom_rug() + geom_smooth(aes(colour = NULL))
```

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



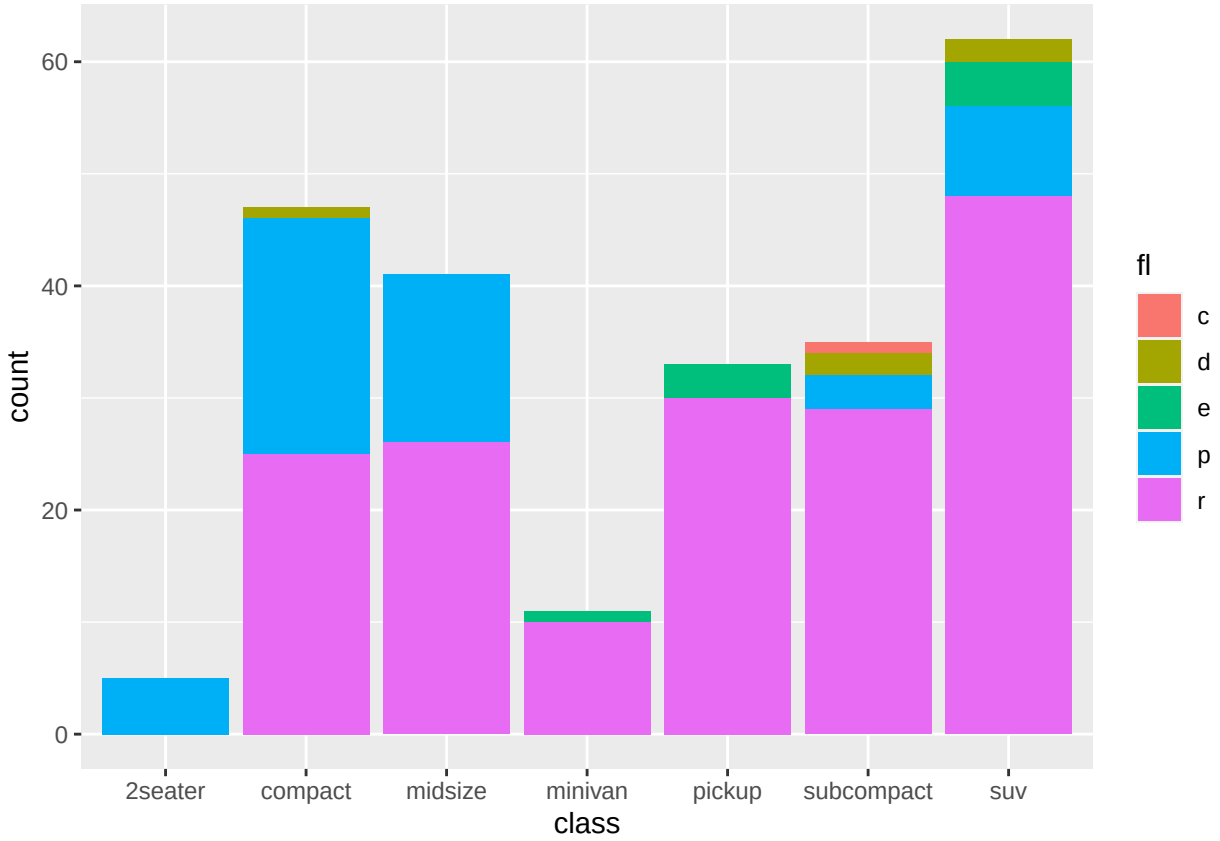
7. Create a bar chart showing the number of cars by type of car.

```
ggplot(mpg) + aes(class) + geom_bar()
```



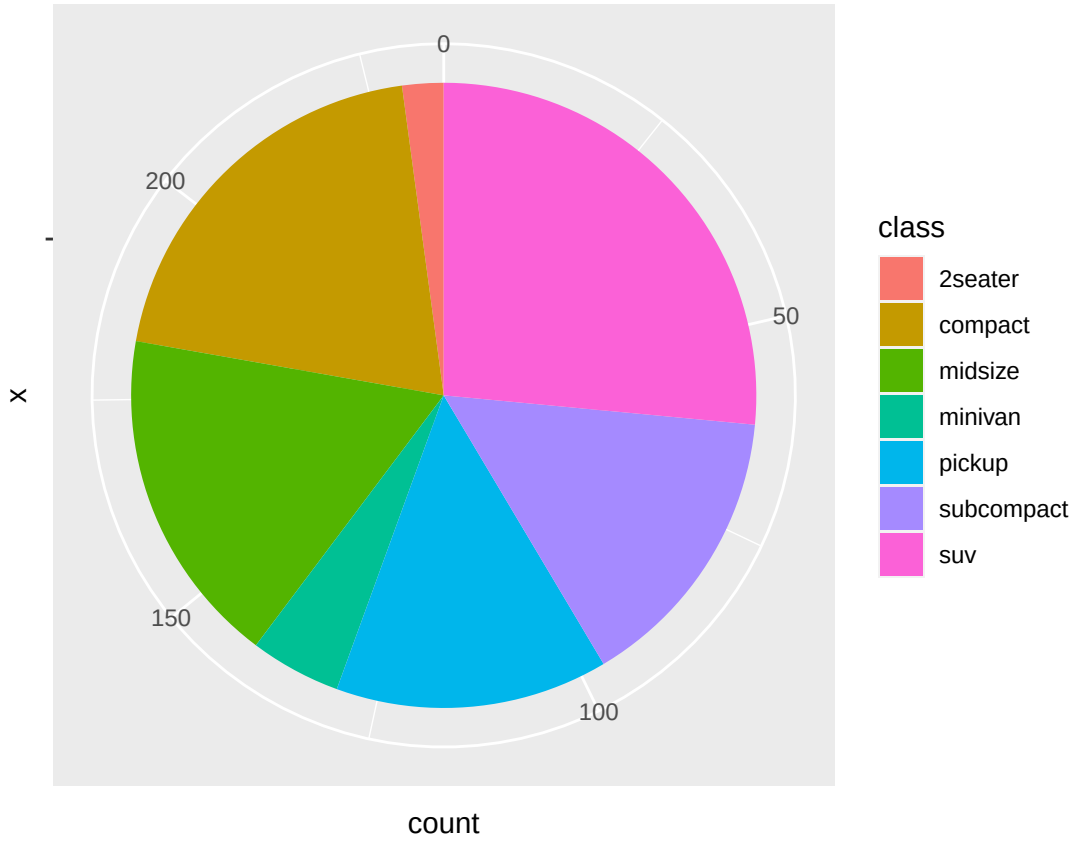
8. Create the same bar chart, showing also the fuel type as stacked bars.

```
ggplot(mpg) + aes(class, fill = fl) + geom_bar()
```



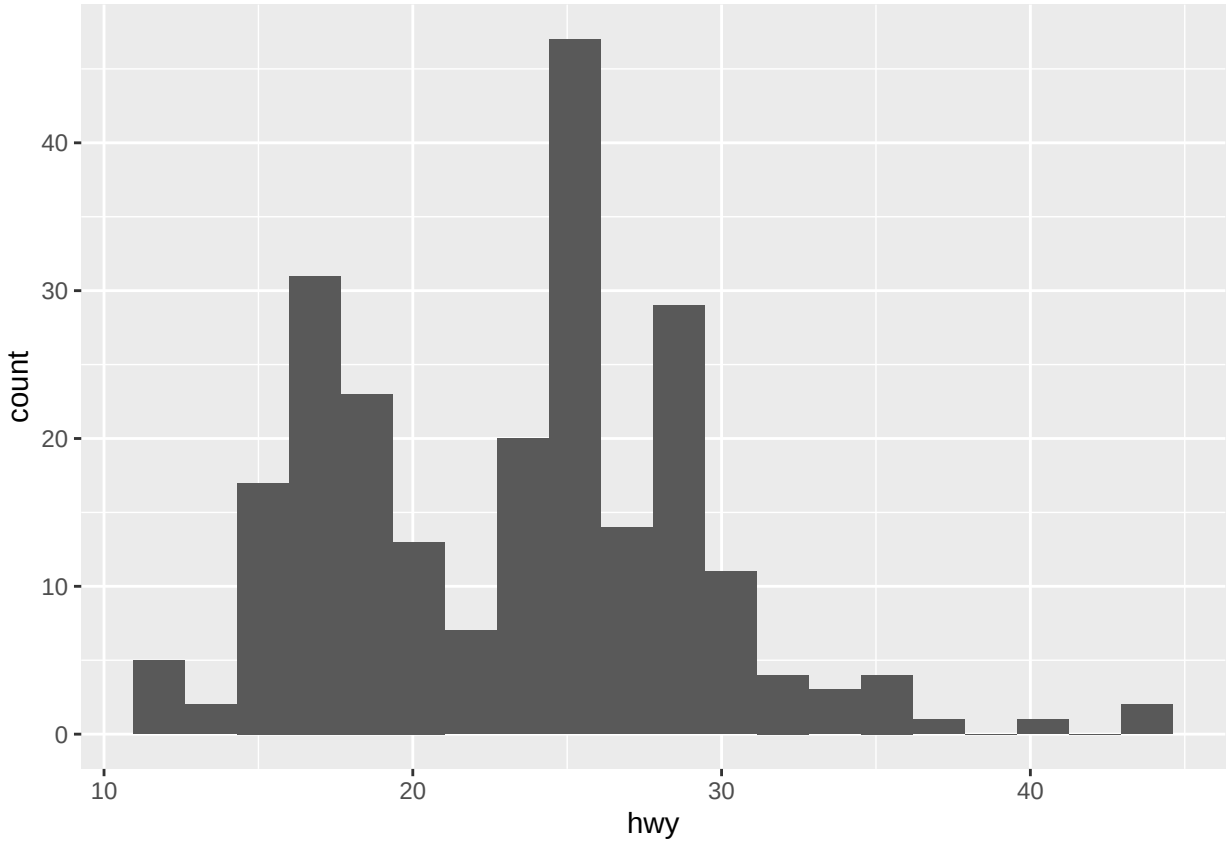
9. Create a pie chart showing the number of cars by car type.

```
ggplot(mpg) + aes(x = "", fill = class) + geom_bar() + coord_polar("y")
```



10. Create a histogram showing the distribution of the highway miles per gallon in 20 bins.

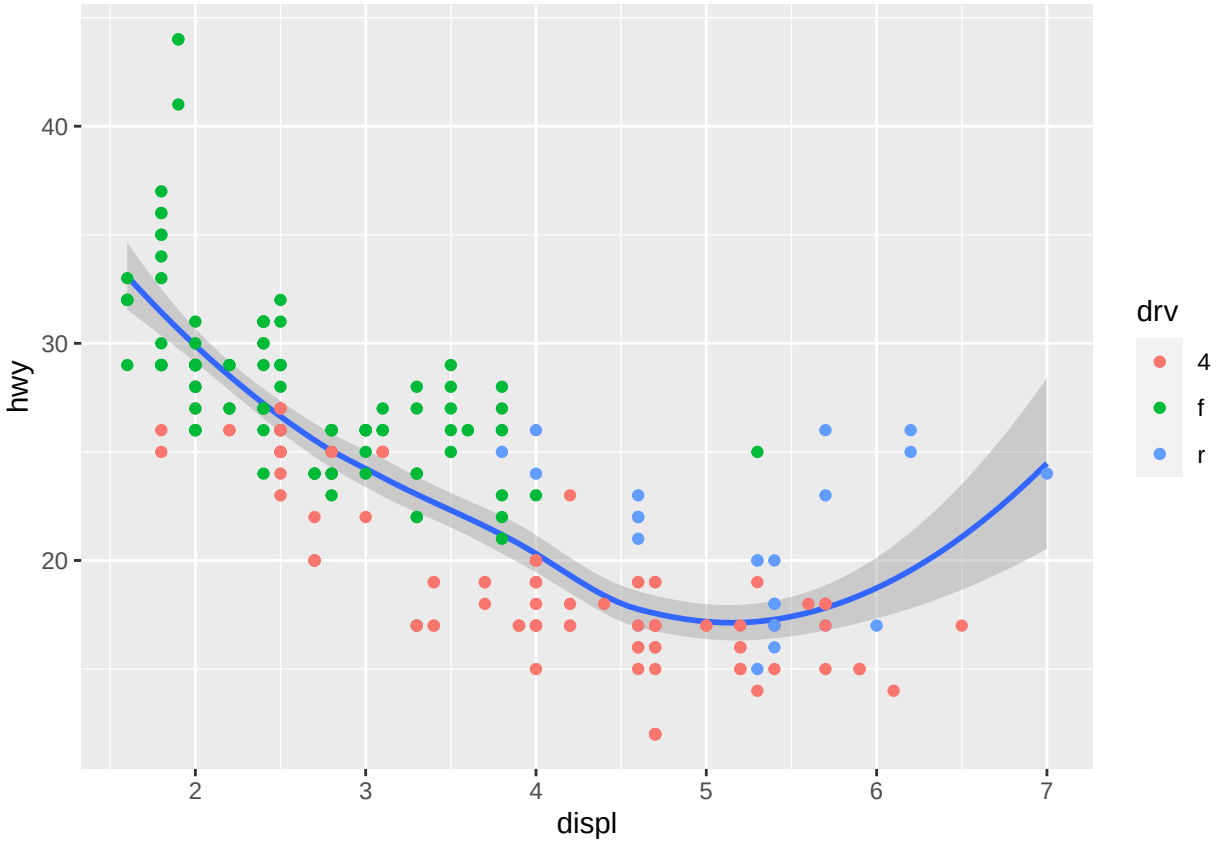
```
ggplot(mpg) + aes(hwy) + geom_histogram(bins = 20)
```



Bonus 2: Create the following chart.

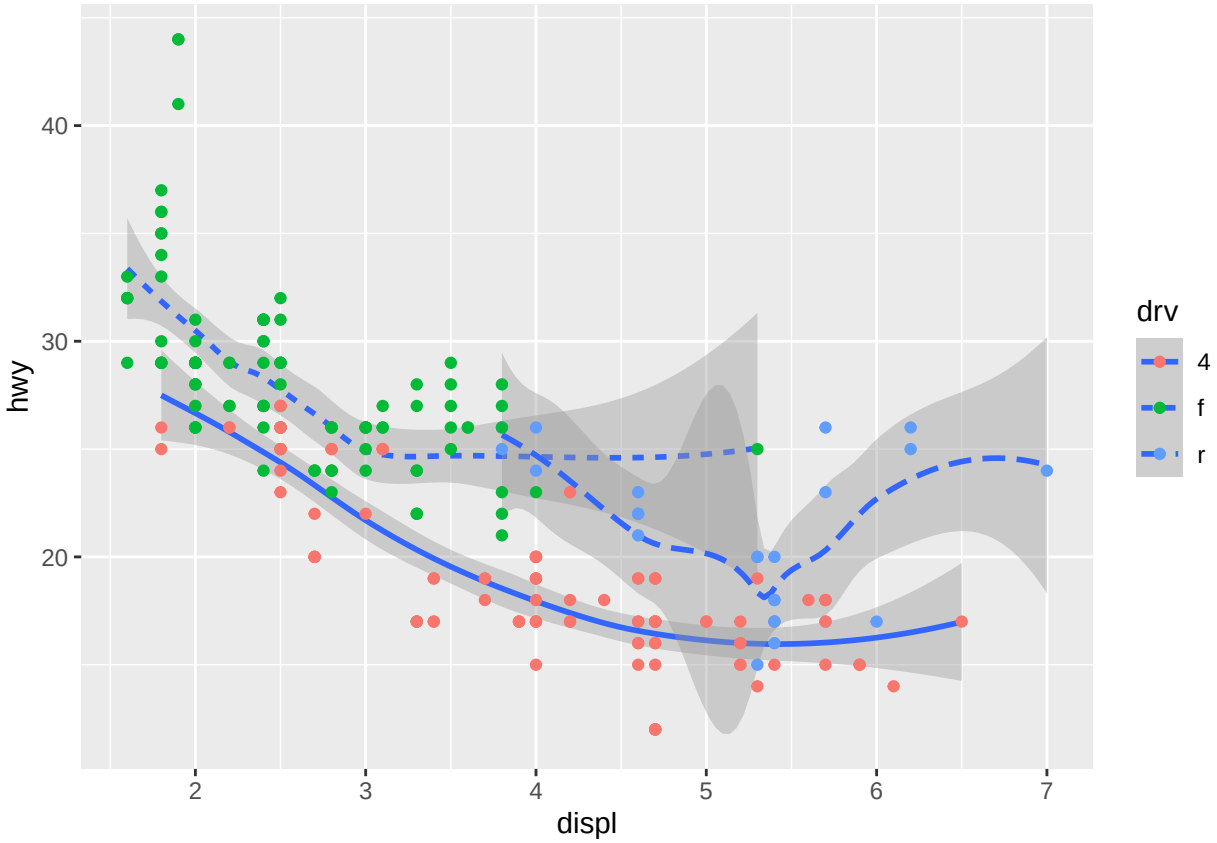
```
ggplot(mpg) + aes(displ, hwy) +
    geom_smooth() + geom_point(aes(colour = drv))
```

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




```
ggplot(mpg) + aes(displ, hwy) +
    geom_smooth(aes(linetype = drv)) + geom_point(aes(colour = drv))
```

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


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