R Graphics with Ggplot2: Day 1&2

October,13-20, 2017

Saving your plot

You can save you plot with ggsave.

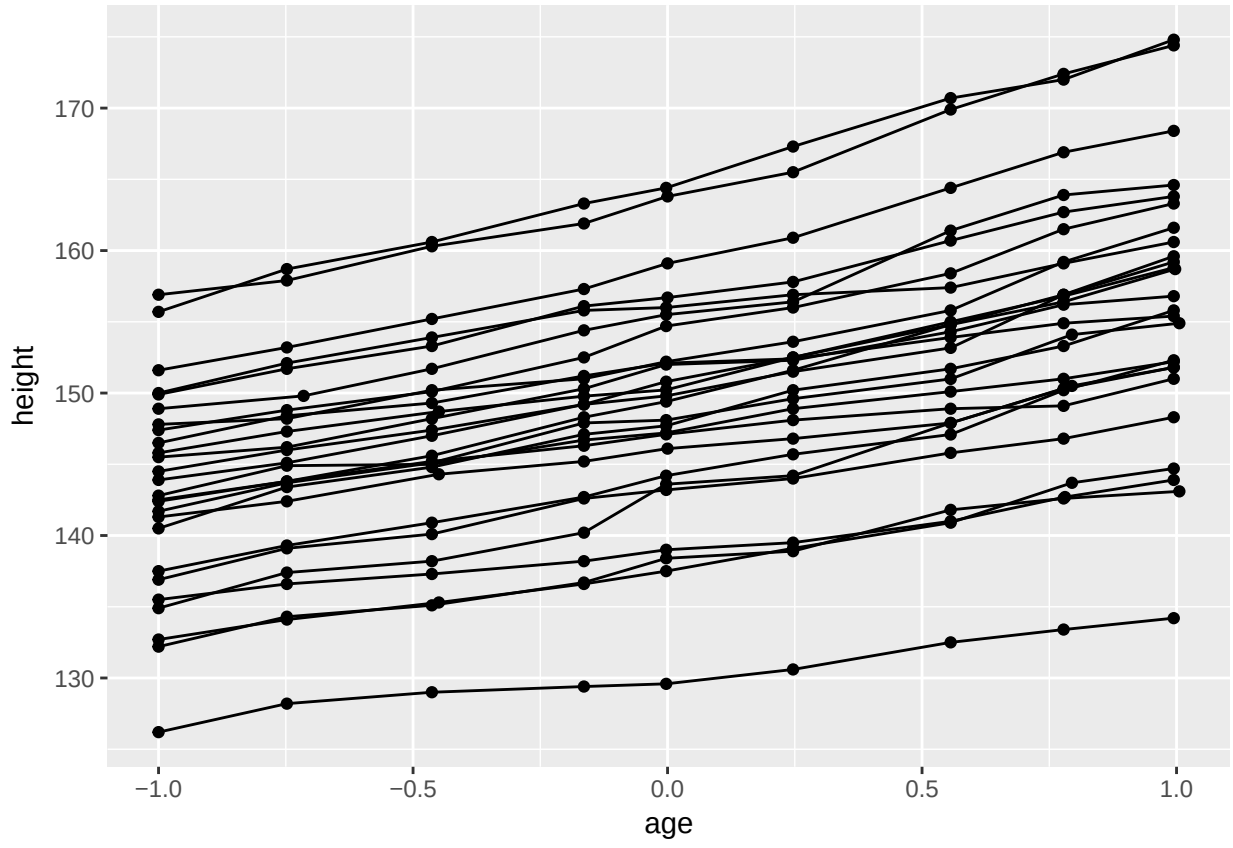
```
library(ggplot2)
ggsave("cars.pdf")
ggsave("cars.tiff")
```

The default is to save the last plot created in the format determined by the file extension at size of the current graphics device. You can adjust all these in the ggsave command. Look at the help file.

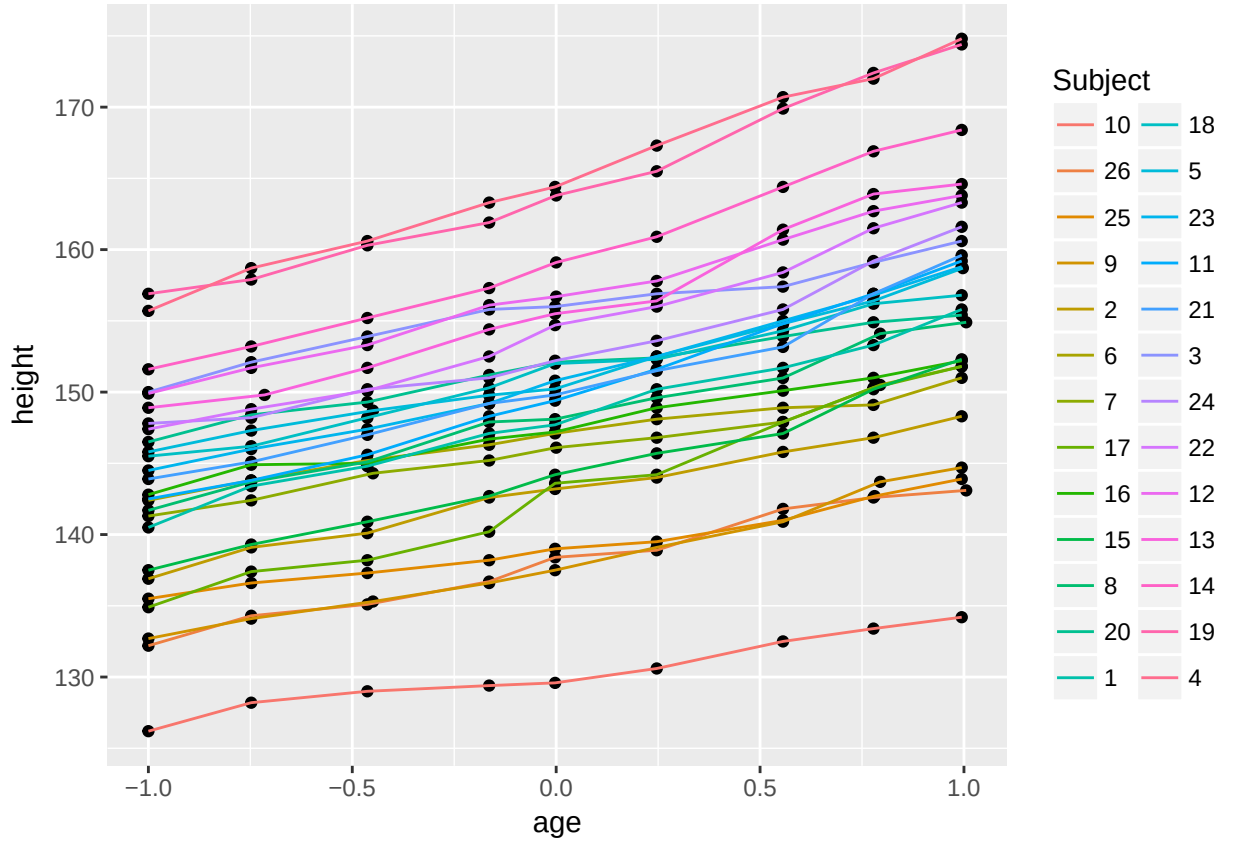
Grouping

In ggplot, `group` is another aesthetic. However any aesthetic that creates distinction between items will cause groups to occur:

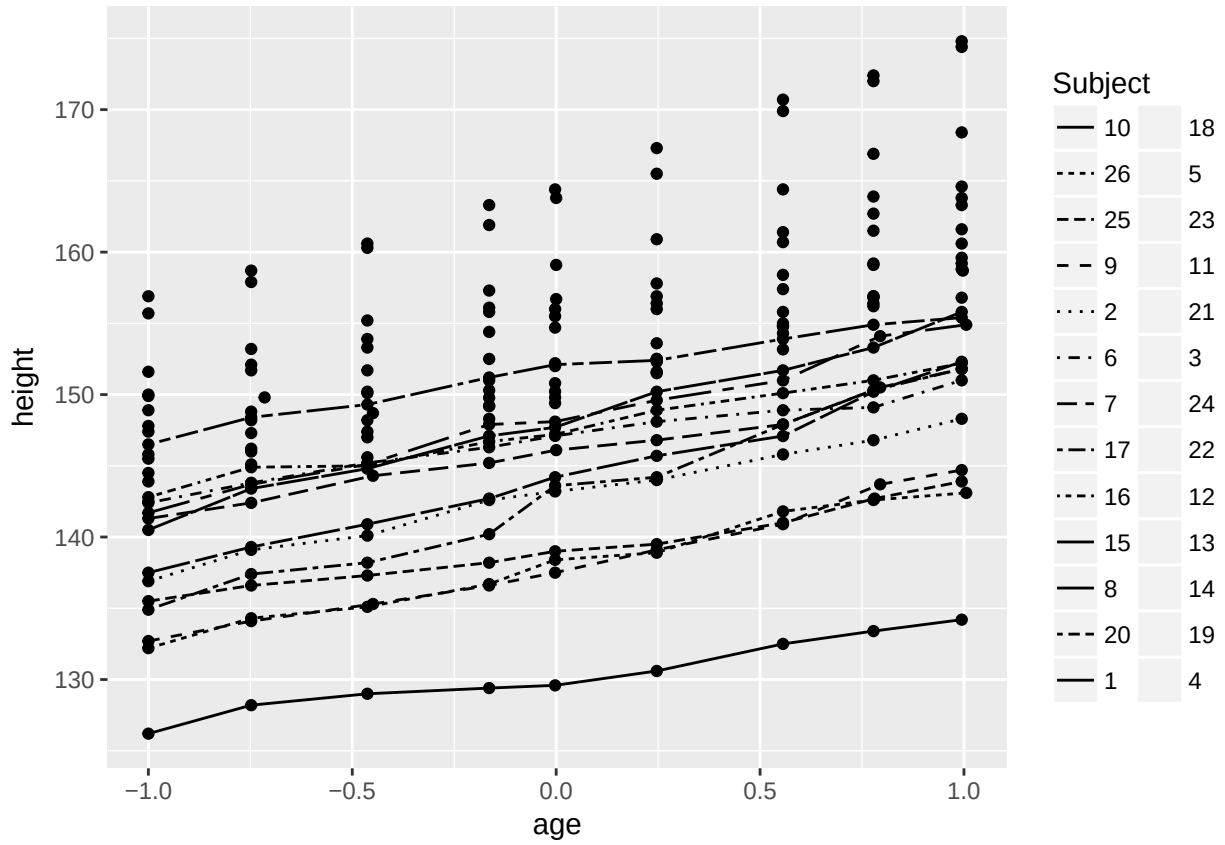
```
library(ggplot2)
library(nlme)
library(dplyr)
ggplot(Oxboys,aes(age,height)) + geom_point() + geom_line(aes(group=Subject))
```



```
ggplot(Oxboys,aes(age,height)) + geom_point() + geom_line(aes(colour=Subject))
```



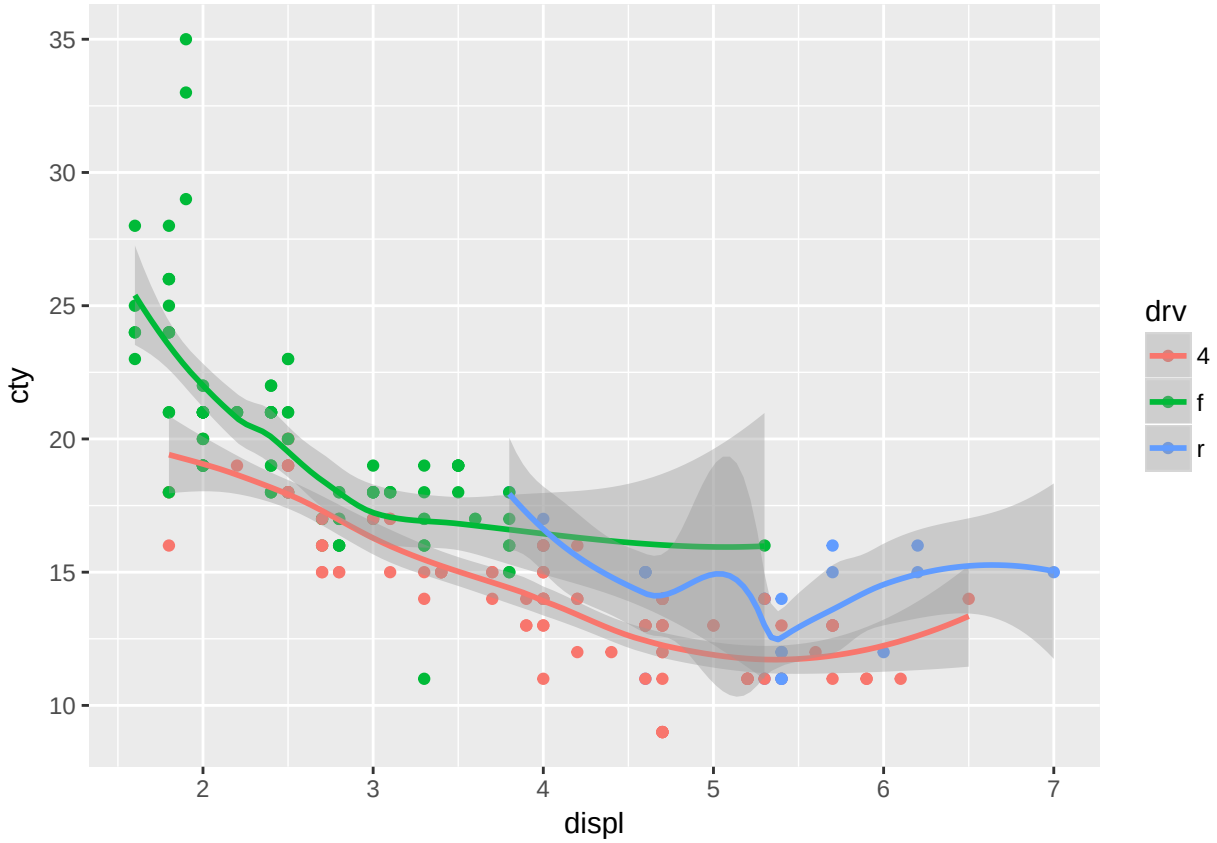

```
ggplot(Oxboys,aes(age,height)) + geom_point() + geom_line(aes(linetype=Subject))
```



Once the grouping is defined, it is in effect for all subsequent layers:

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

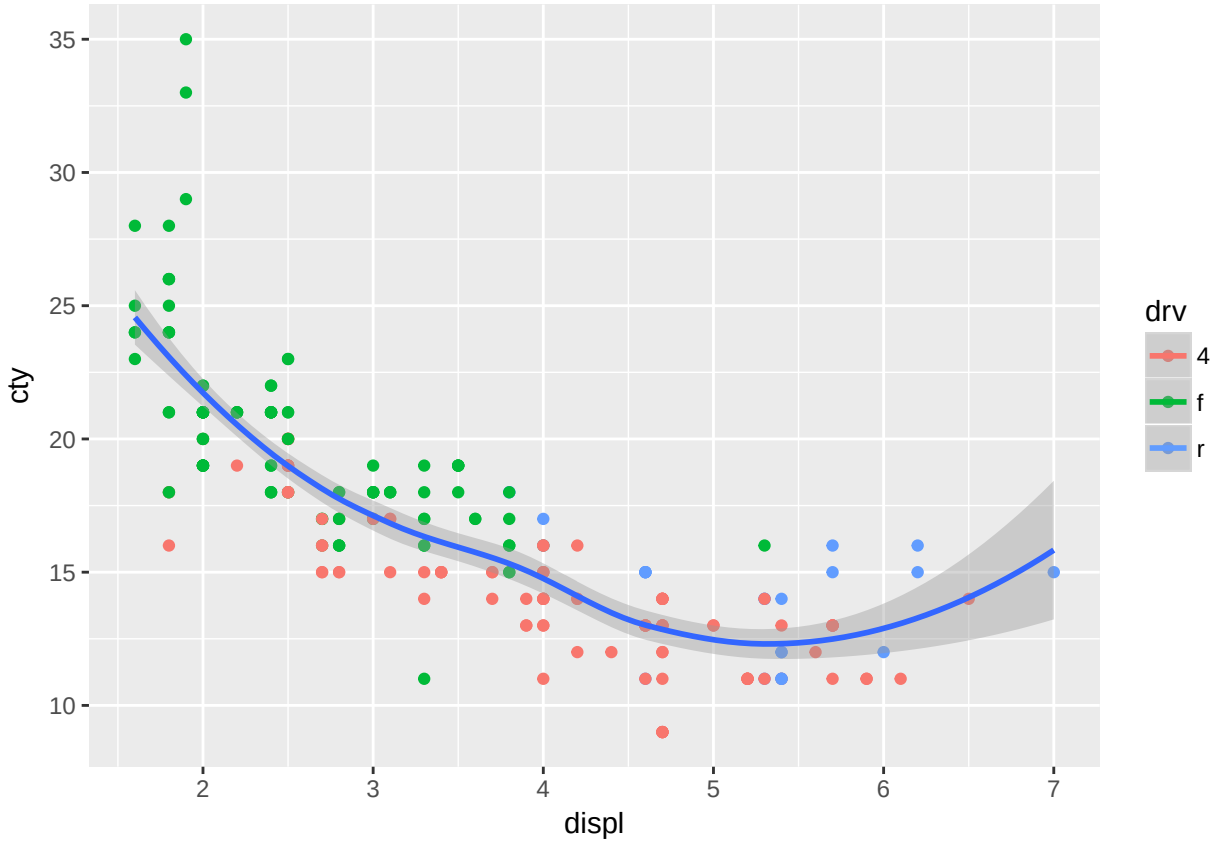
```
## `geom_smooth()` using method = 'loess'
```



We can add an overall best-fit line by redefining the group in the aesthetic *for* `geom_smooth`:

```
ggplot(mpg, aes(displ, cty, colour=drv)) +
  geom_point() +
  geom_smooth(aes(group=1))
```

```
## `geom_smooth()` using method = 'loess'
```



Exercise: Using the gapminder dataset (`library(gapminder)`), make a line graph of life expectancy of each country over time. Colour countries by the continent.

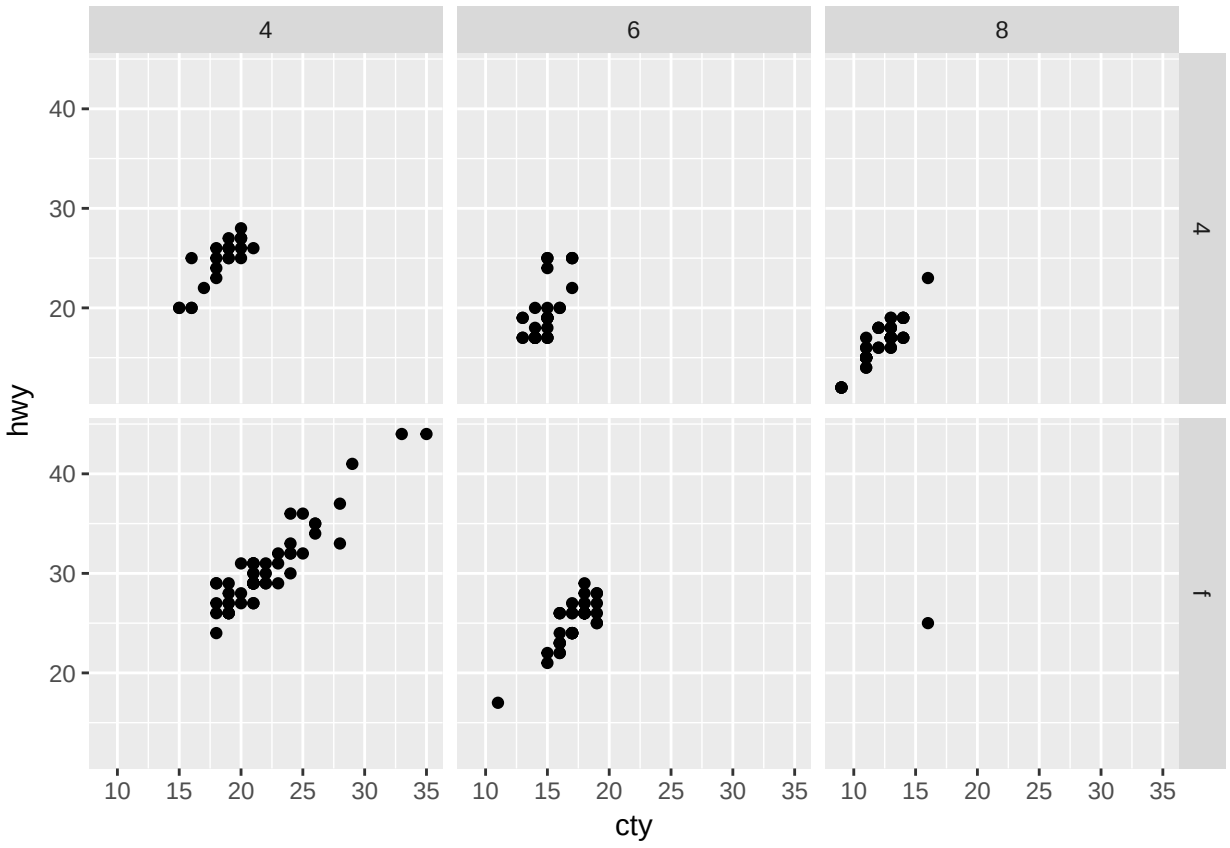
Faceting

Rather than putting a lot of information in a single graphic, we can split the graphic by certain features and plot a "matrix" of graphics to see the effect of the feature on the data. This is called faceting.

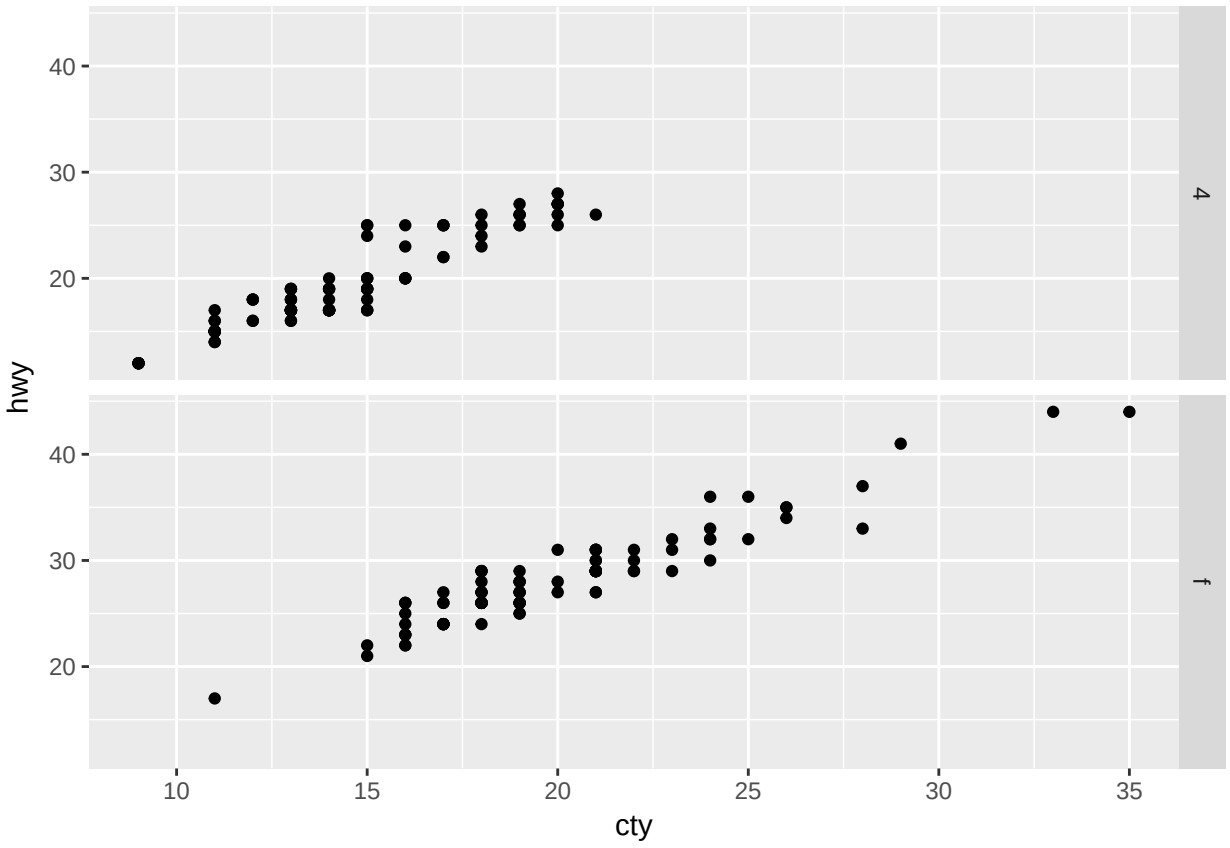
There are two different facet commands, "`facet_grid`" and "`facet_wrap`".

"`facet_grid`" is typically used when you have at least 2 factors and you want the layout to be a matrix with one factor defining the rows and the other defining the columns.

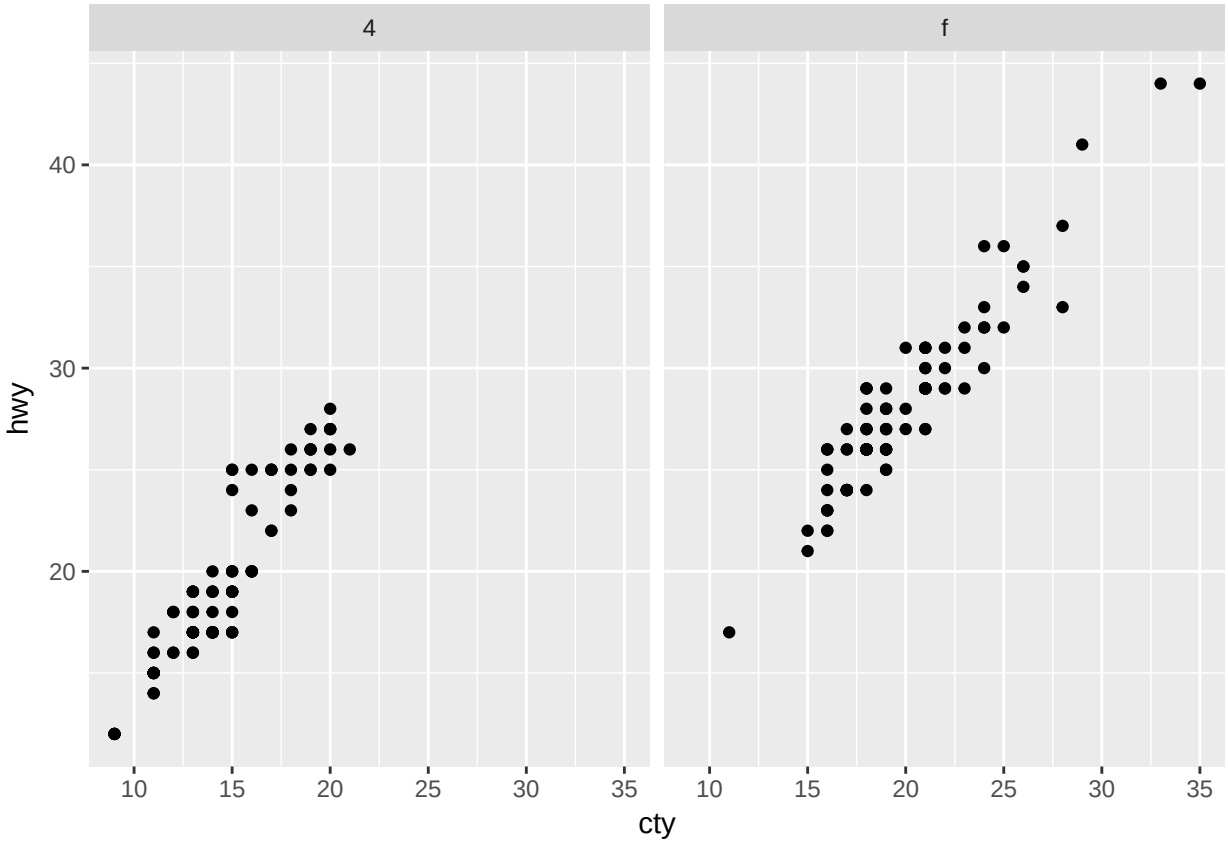
```
p0 <- filter(mpg, cyl != 5, drv %in% c("4", "f")) %>%
  ggplot(aes(cty,hwy))
p1 <- p0 + geom_point()
p1 + facet_grid(drv ~ cyl)
```




```
p1 + facet_grid(drv ~ .)
```

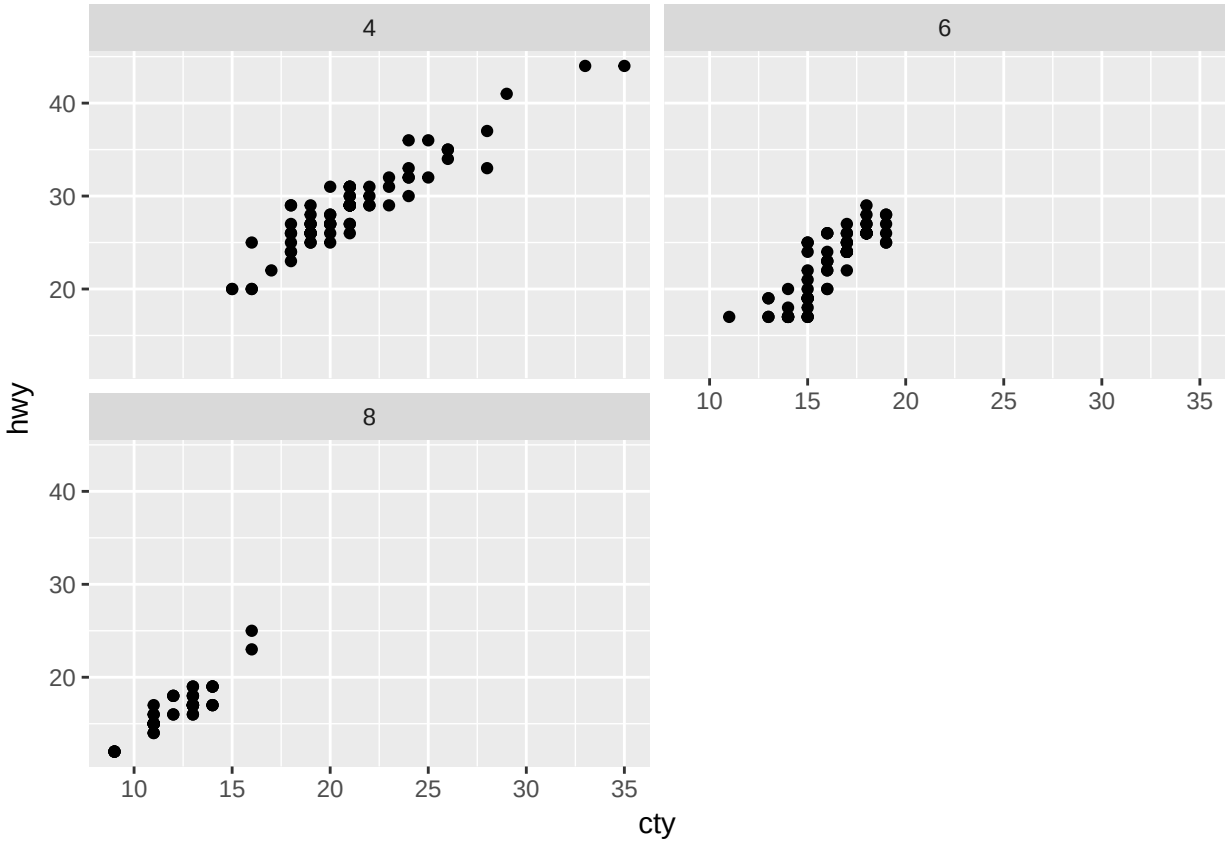



```
p1 + facet_grid(. ~ drv)
```

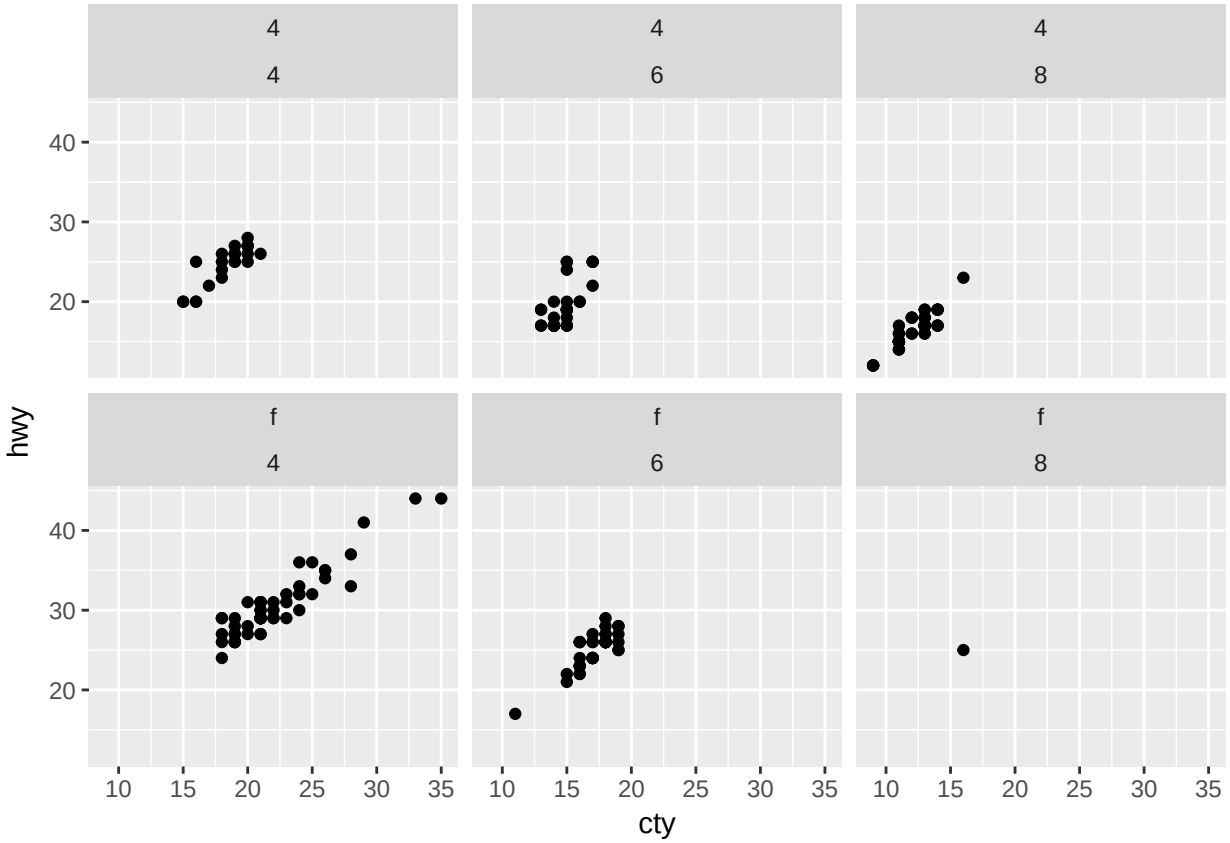


"`facet_wrap`" is typically used when you have a single factor with too many levels to fit in a single row or column

```
p1 + facet_wrap( ~ cyl, ncol=2)
```



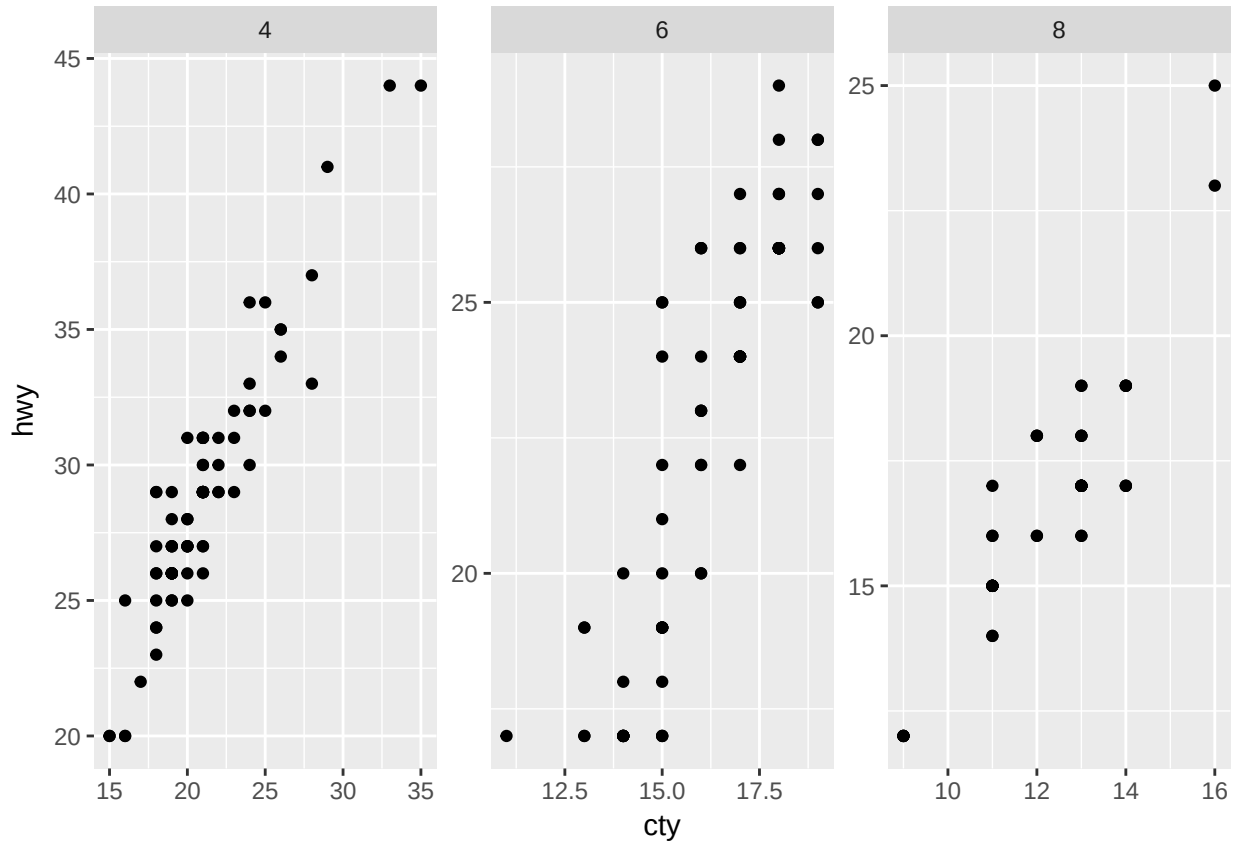

```
p1 + facet_wrap( ~ drv + cyl)
```



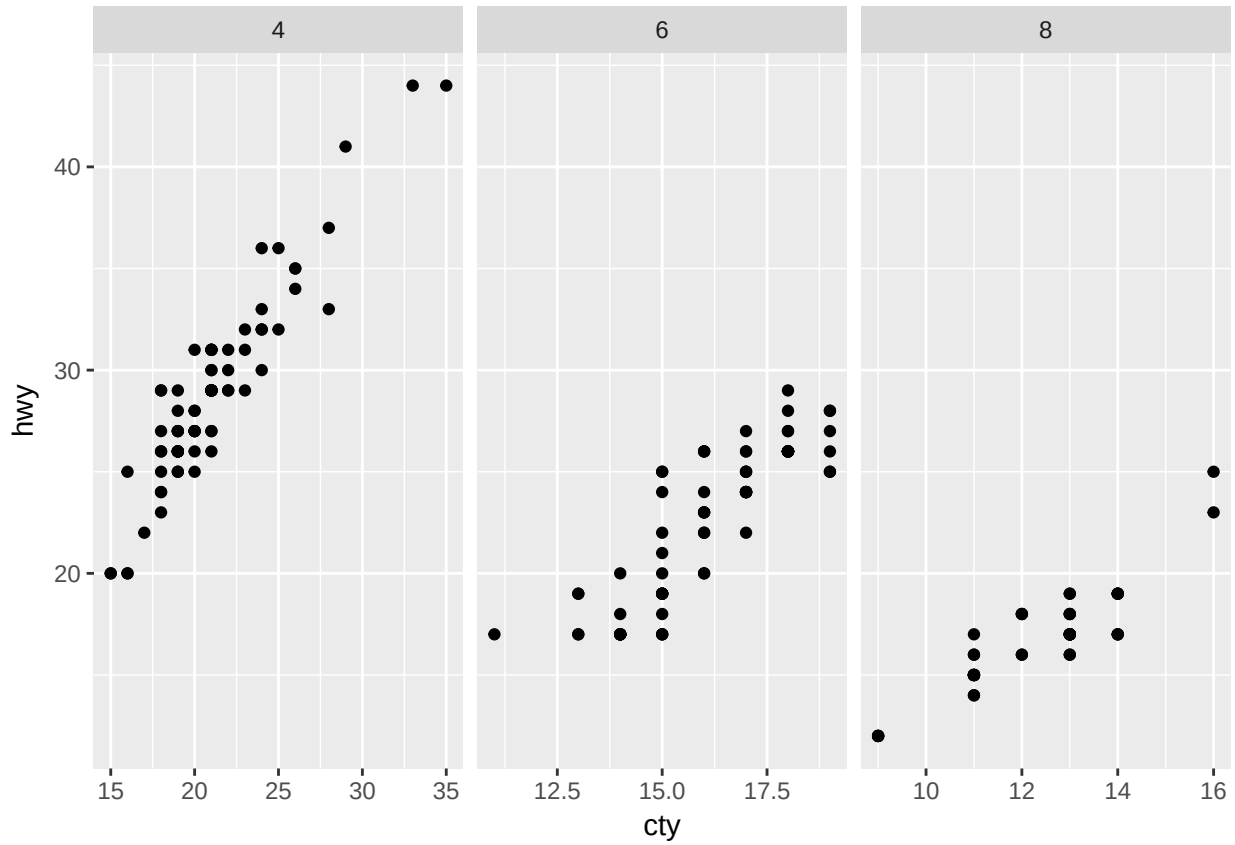
Exercise: Use a histogram to explore the distribution of engine size for different vehicle classes.

The default for faceting is to use the same scale for each plot. This is usually a wise choice but you can allow each plot to have its own scale.

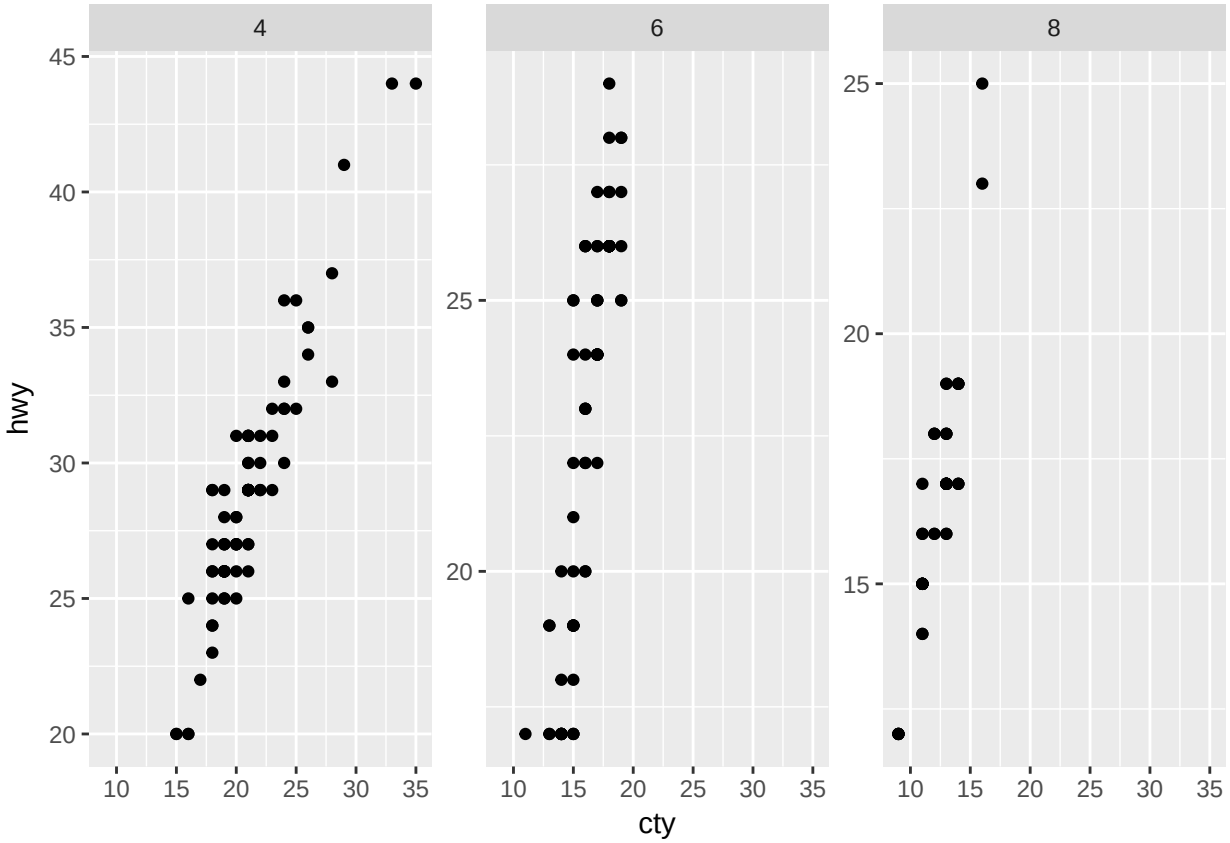
```
#both x and y axes have their own scales
p1 + facet_wrap(~cyl,scales="free")
```




```
#x has its own scale, while y has the same scale for each plot
p1 + facet_wrap(~cyl,scales="free_x")
```

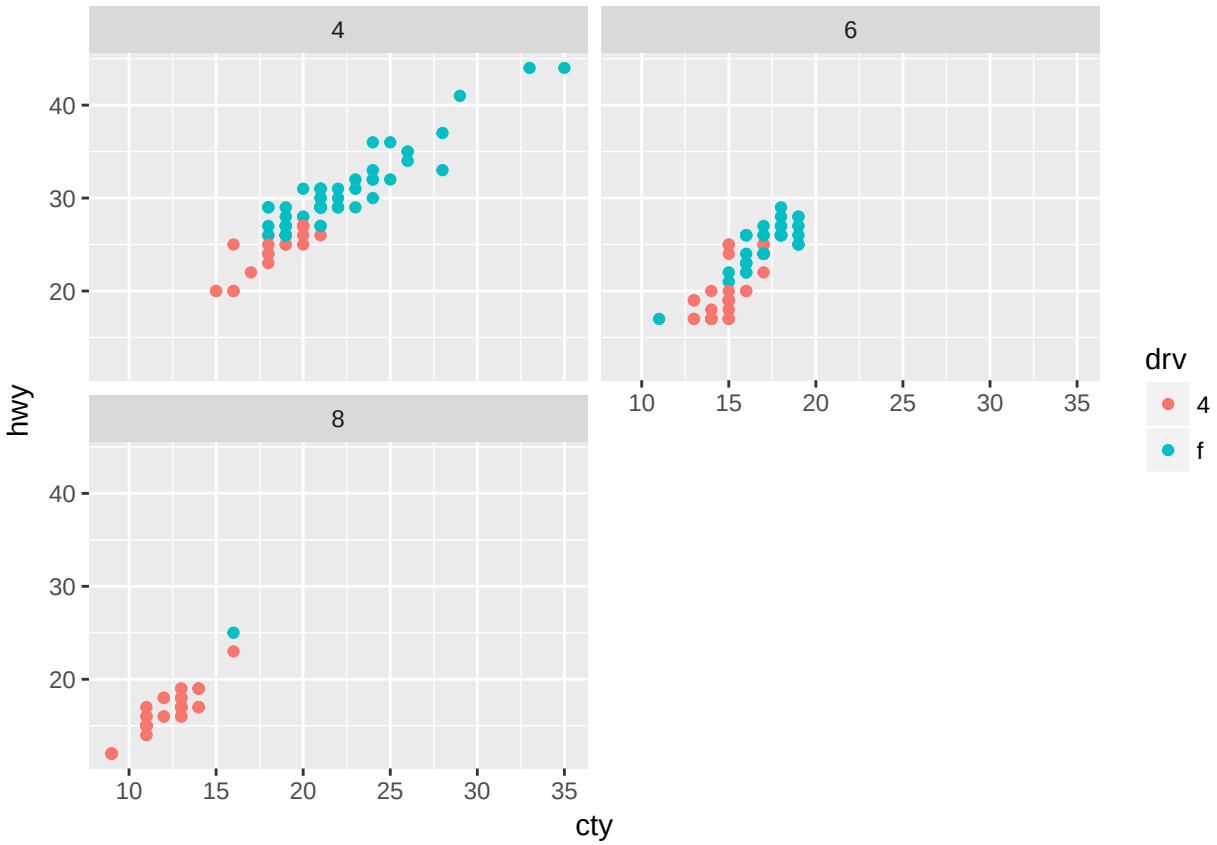



```
#y has its own scale, while x has the same scale for each plot
p1 + facet_wrap(~cyl,scales="free_y")
```



Facets also work well with groups:

```
p0 + geom_point(aes(color=drv)) + facet_wrap(~cyl, ncol=2)
```

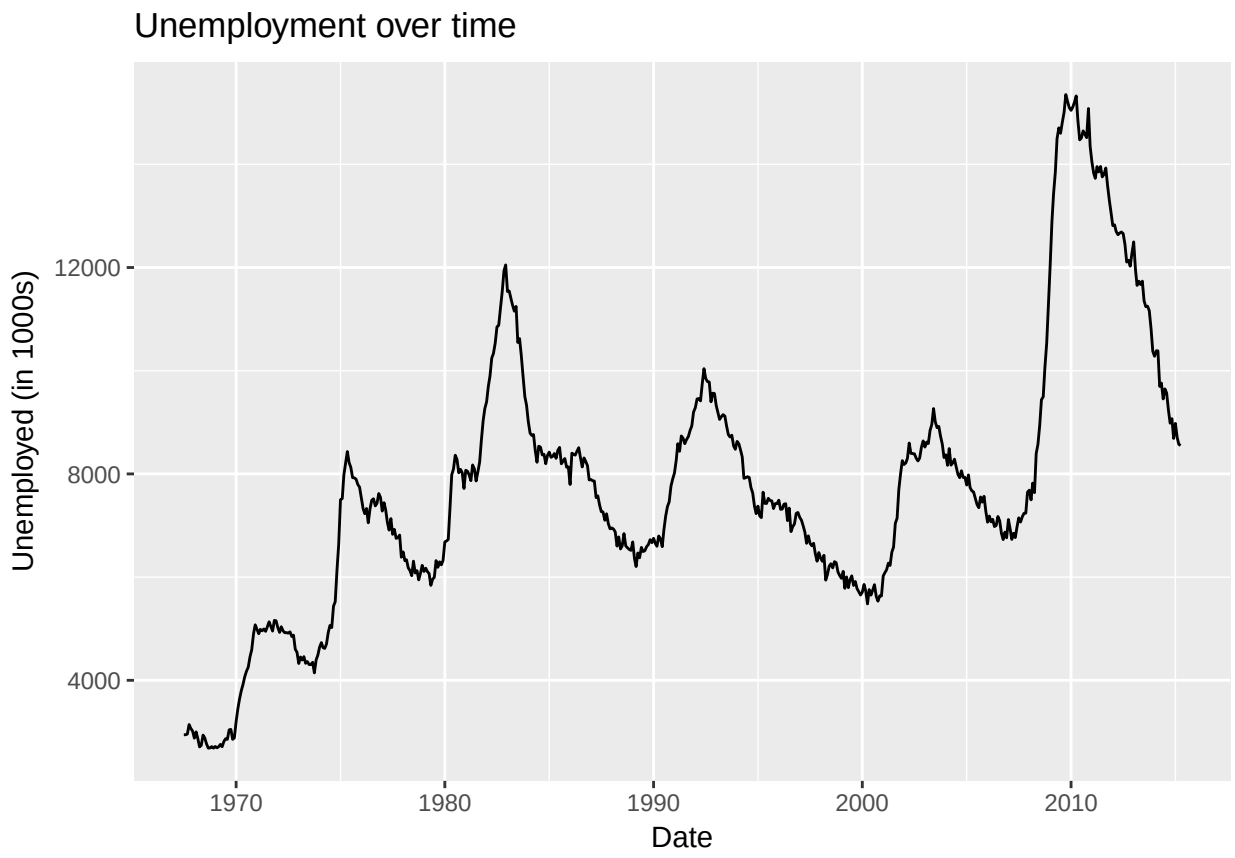


Note: You cannot use a continuous variable to facet. You must first convert it to a discrete variable.

Changing title and axis labels

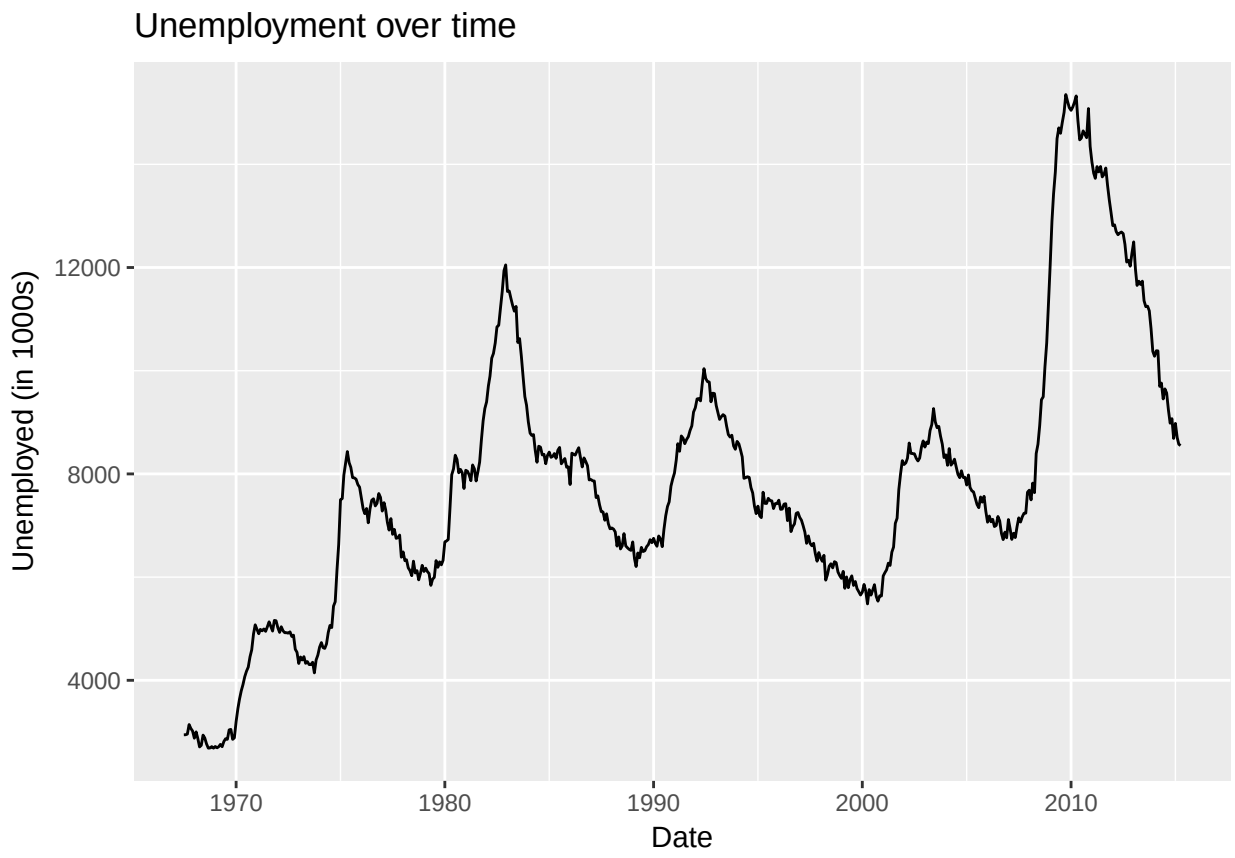
You can add a title or change the axis labels

```
ggplot(economics, aes(date, unemploy)) +
  geom_line() +
  xlab('Date') +
  ylab('Unemployed (in 1000s)') +
  ggtitle('Unemployment over time')
```



You can add them all with `labs()`

```
ggplot(economics, aes(date, unemploy)) +
  geom_line() +
  labs(x = 'Date',
       y = 'Unemployed (in 1000s)',
       title = 'Unemployment over time')
```



We can modify the axis limits in a similar fashion.

```
p1 <- ggplot(mpg, aes(displ, cty)) + geom_point()
p1
p1 + xlim(-1,4) + ylim(-5,25)
```

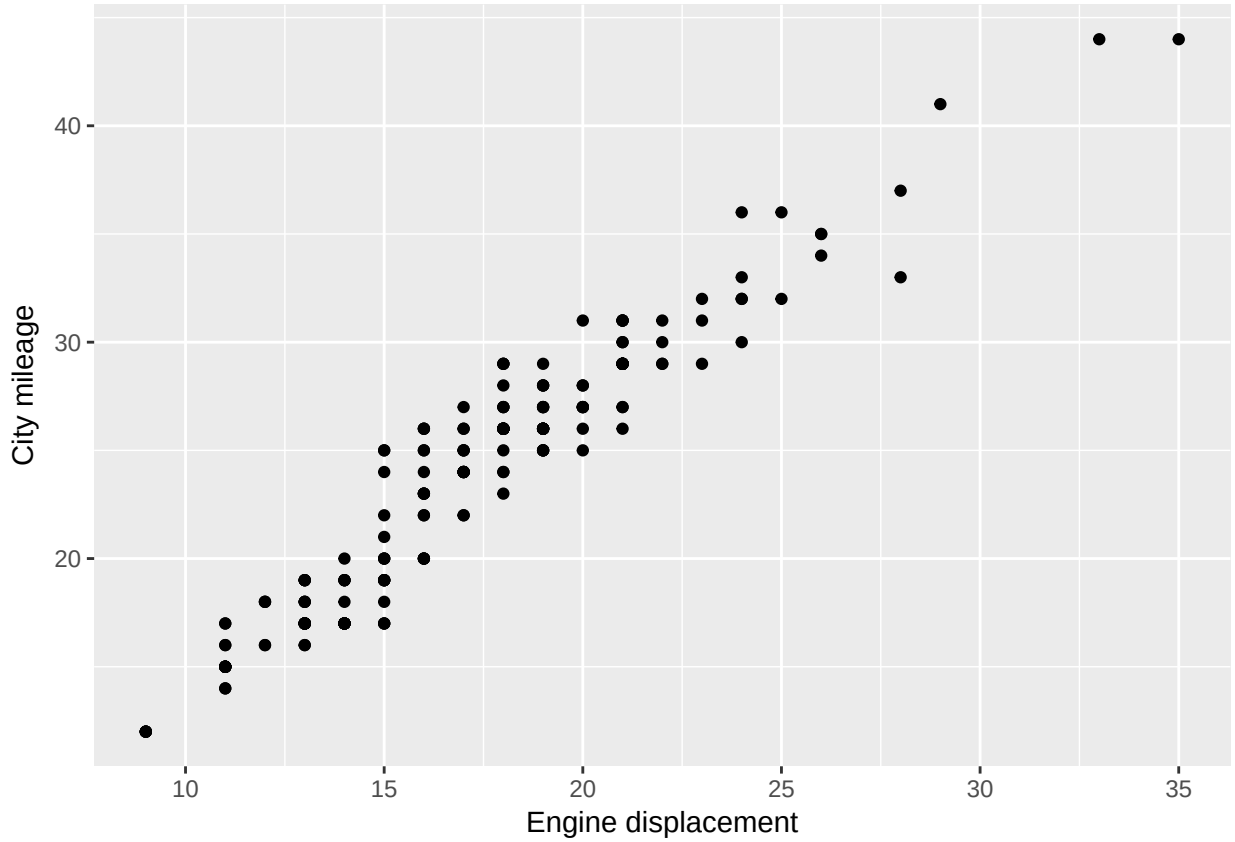
Scales in general

Labelling and axis limits are really part of scales but quite often they are all you want to change so ggplot2 contains special functions to control just those elements.

Scales control the mapping from data to aesthetics. They take your data and turn it into something that you can see, like size, colour, position or shape. Scales also provide the *guides* like axes marks and legends to allow you to read observations from the plot and map them back to their original values.

Every plot created by ggplot has a scale for every aesthetic it uses, even if only implicitly. The default scales will contain reasonable defaults, but you can take as much control as you want by providing your own scale. This is done by simply adding another layer to the plot, like this:

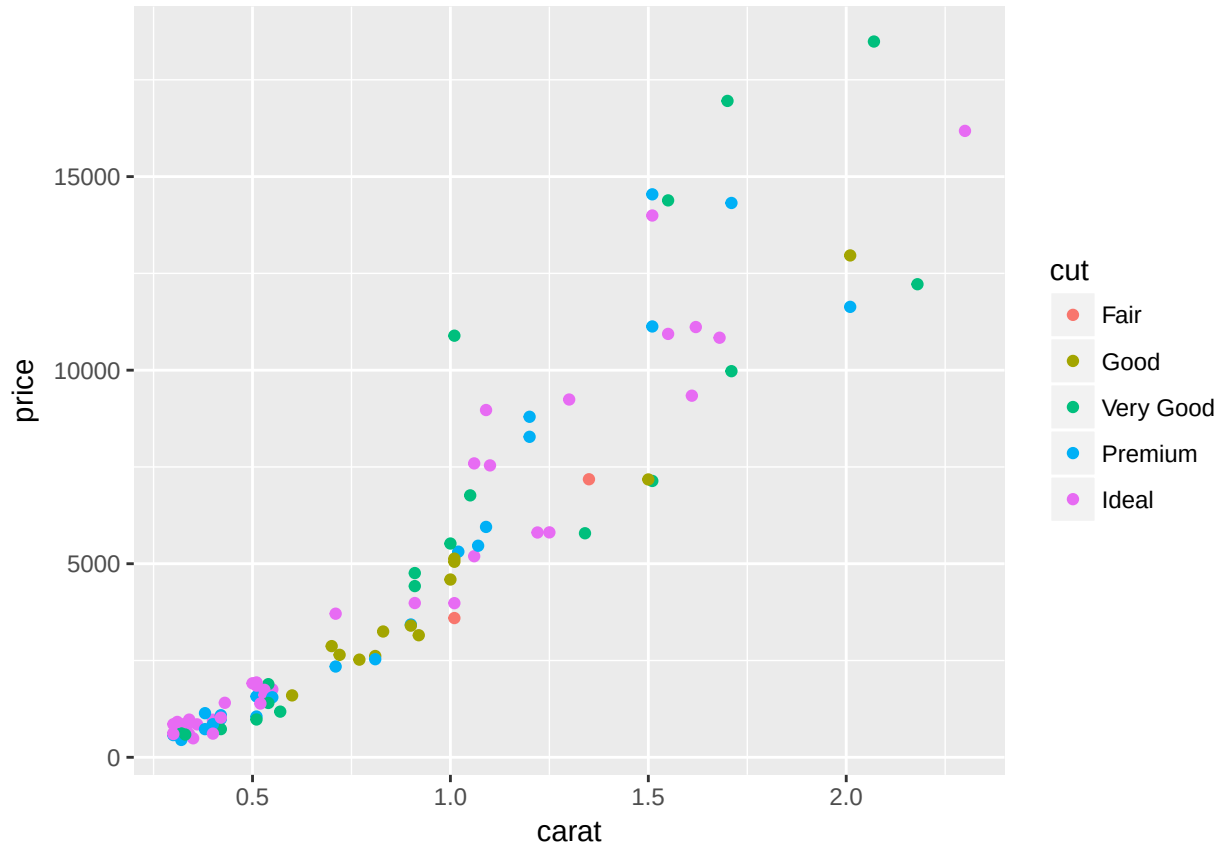
```
p1 + scale_x_continuous("Engine displacement") +
  scale_y_continuous("City mileage")
```



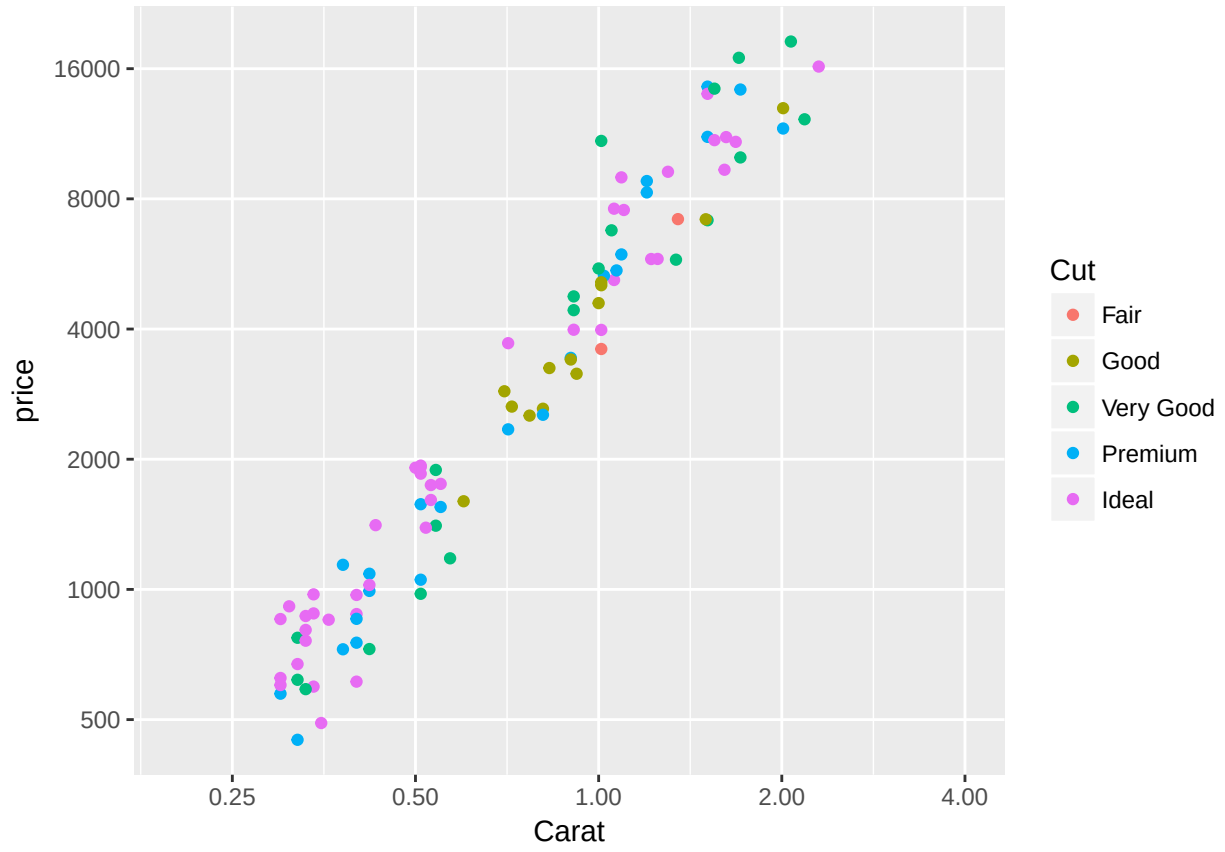
All scaling functions are of the form `scale_` + *aesthetic* + `_` + *name of scale*, such as "`scale_x_continous`" above. Their first argument is always "`name`", which produces the label for the axes and title for the legend. (Setting just this in the last example was equivalent to using "`xlab`" and "`ylab`". But you can also control which values appear in the axes as the tick marks and keys in the legend, using the "`breaks`" argument, as well as their labelling using the "'labels" argument. (Notice the parallels between axes and legends.)

Here's an example of changing the scale for the axes:

```
p1 <- ggplot(sample_n(diamonds, 100),
   aes(carat,price, colour=cut)) + geom_point()
p1
```



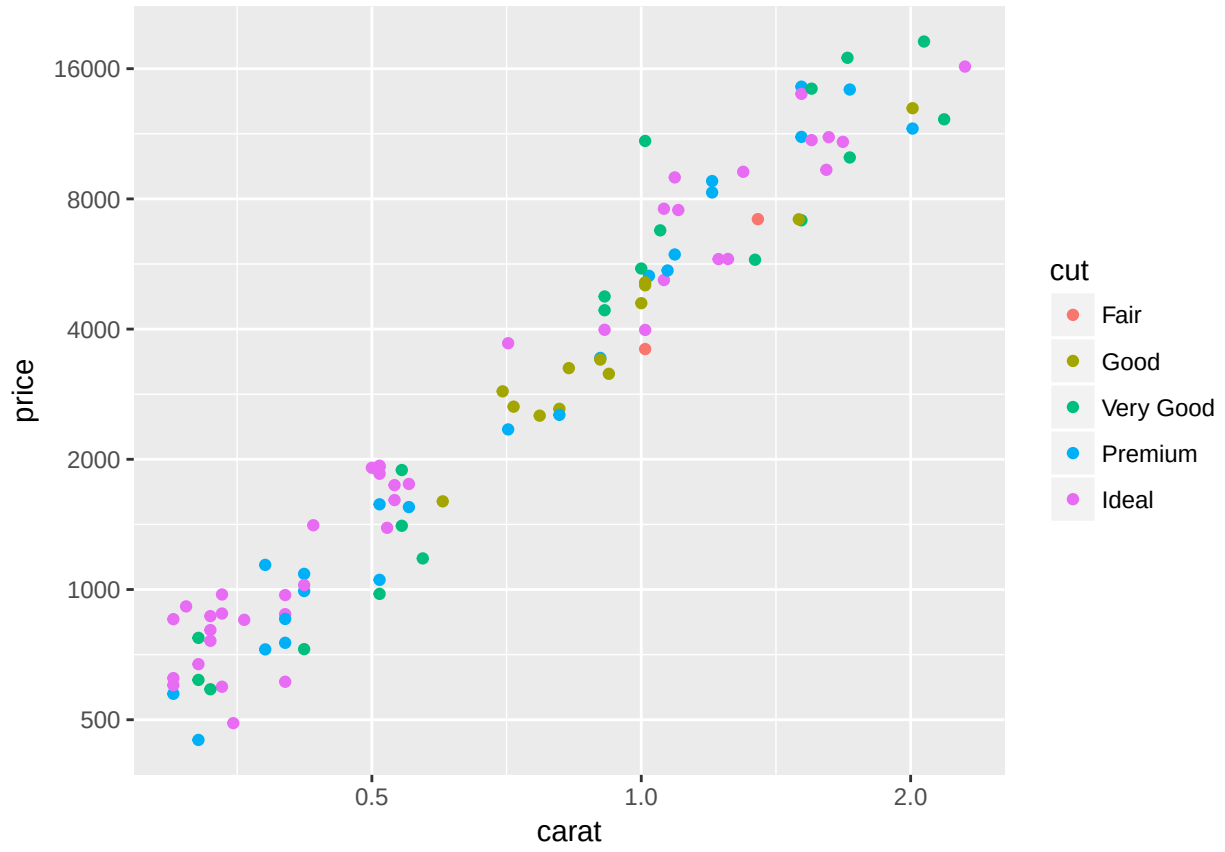

```
p1 + scale_x_continuous(name="Carat", limits=c(.2,4), trans="log",
    breaks=c(.25,.5,1,2,4)) +
  scale_y_continuous(trans="log10",
    breaks=1000*c(.5,1,2,4,8,16),
    minor_breaks=NULL) +
  scale_colour_discrete("Cut")
```



(Use "`minor_breaks`" to control where to place the fainter grid lines that go between the ticks, or NULL to disable them entirely.)

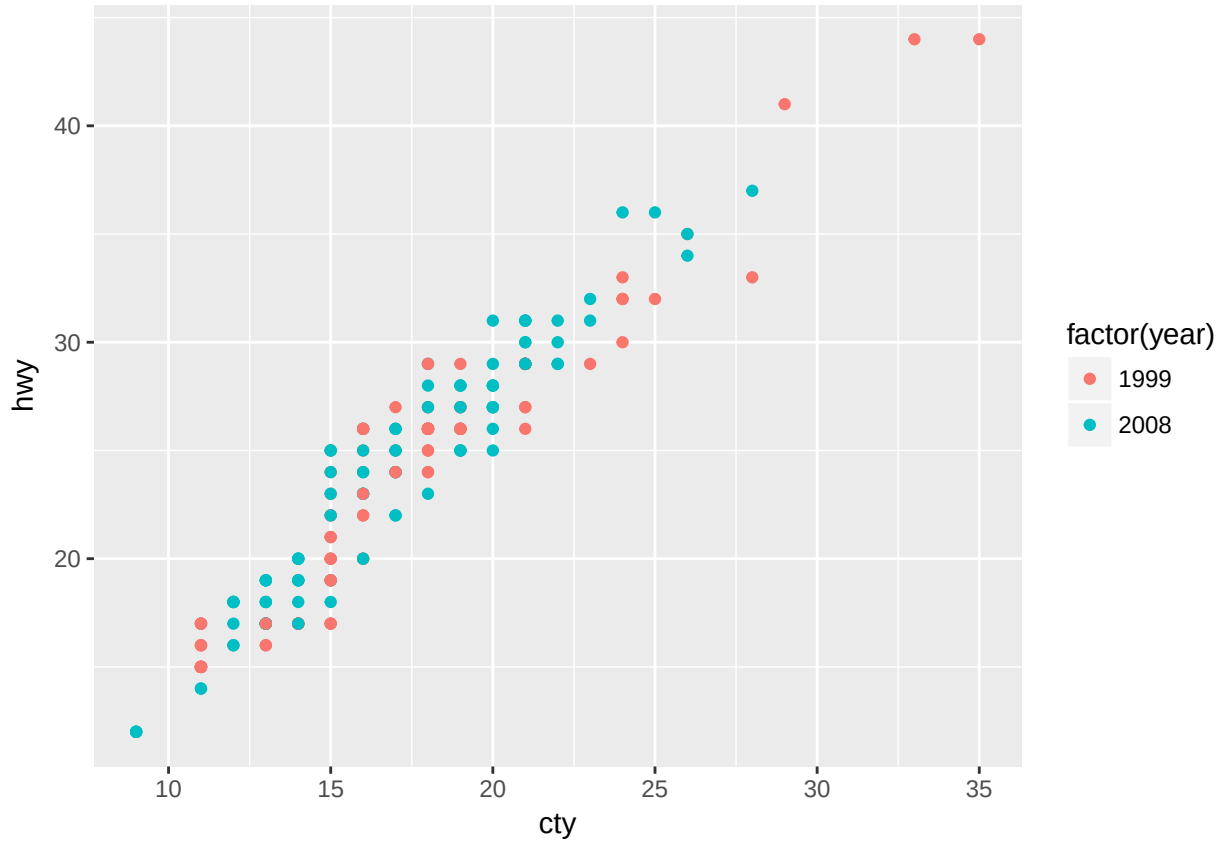
Since log and sqrt transformations are so common, ggplot has defined convenience scale functions for these transformations.

```
p1 + scale_x_log10(breaks=c(.25,.5,1,2,4)) +
  scale_y_log10(breaks=1000*c(.5,1,2,4,8,16))
```

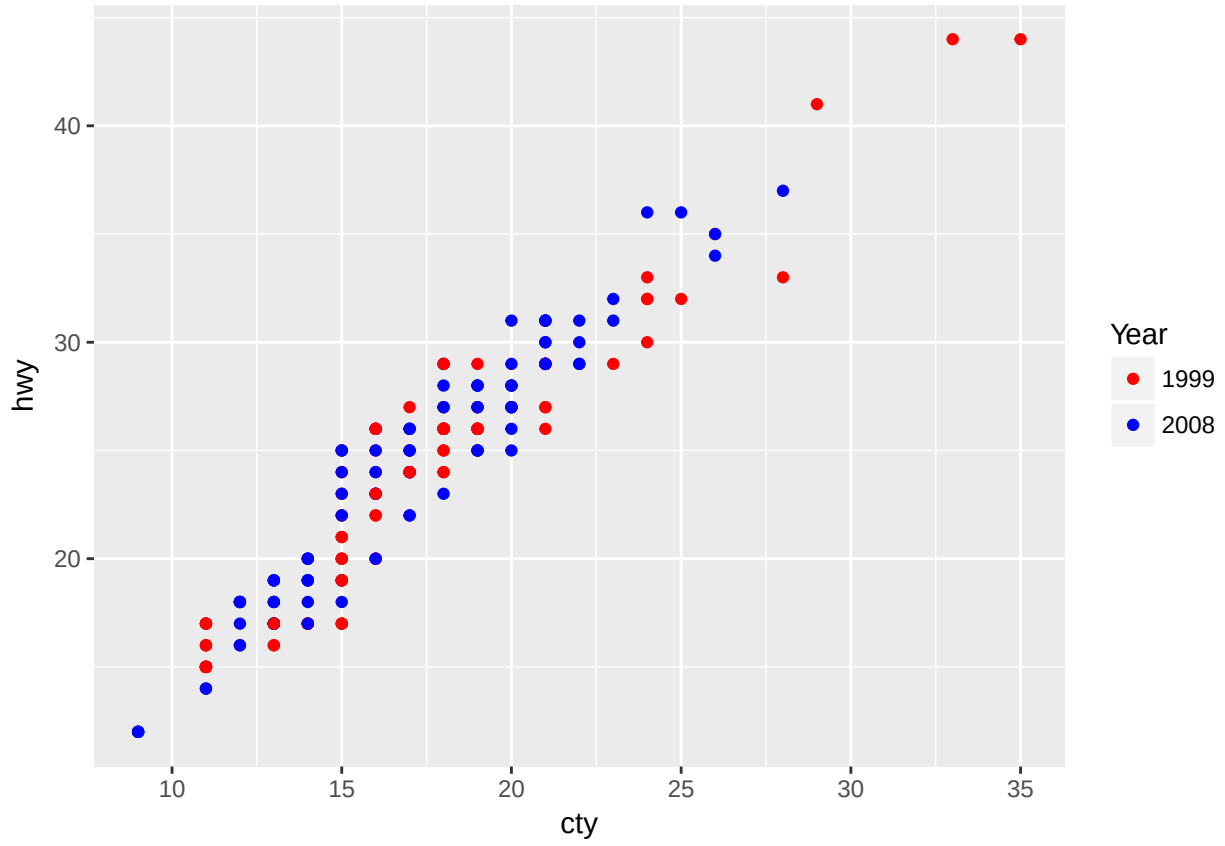


Lets change the scale for colours.

```
p1 <- ggplot(mpg,aes(cty,hwy, colour=factor(year))) +
  geom_point()
p1
```

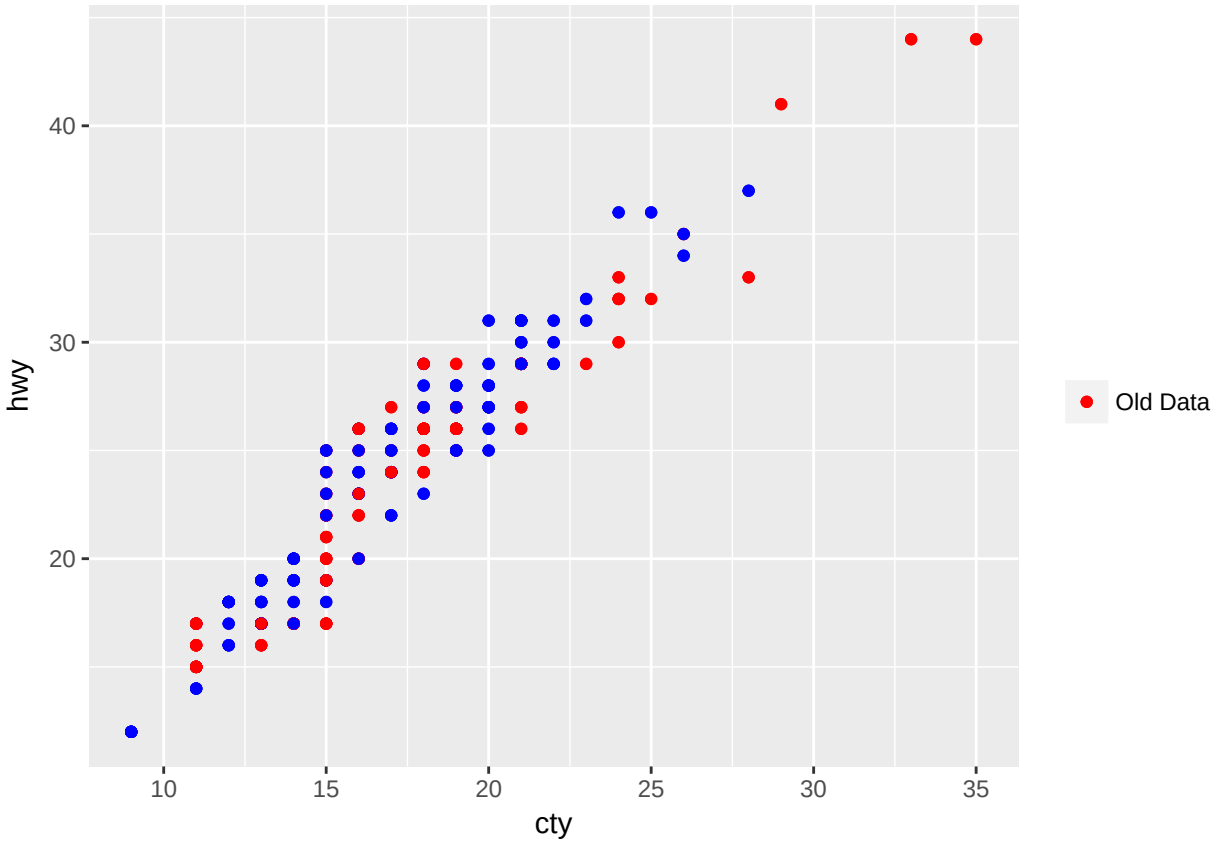



```
p1 + scale_colour_manual(name="Year",values=c("red","blue"))
```



You can even choose to display in the legend only some of the keys:

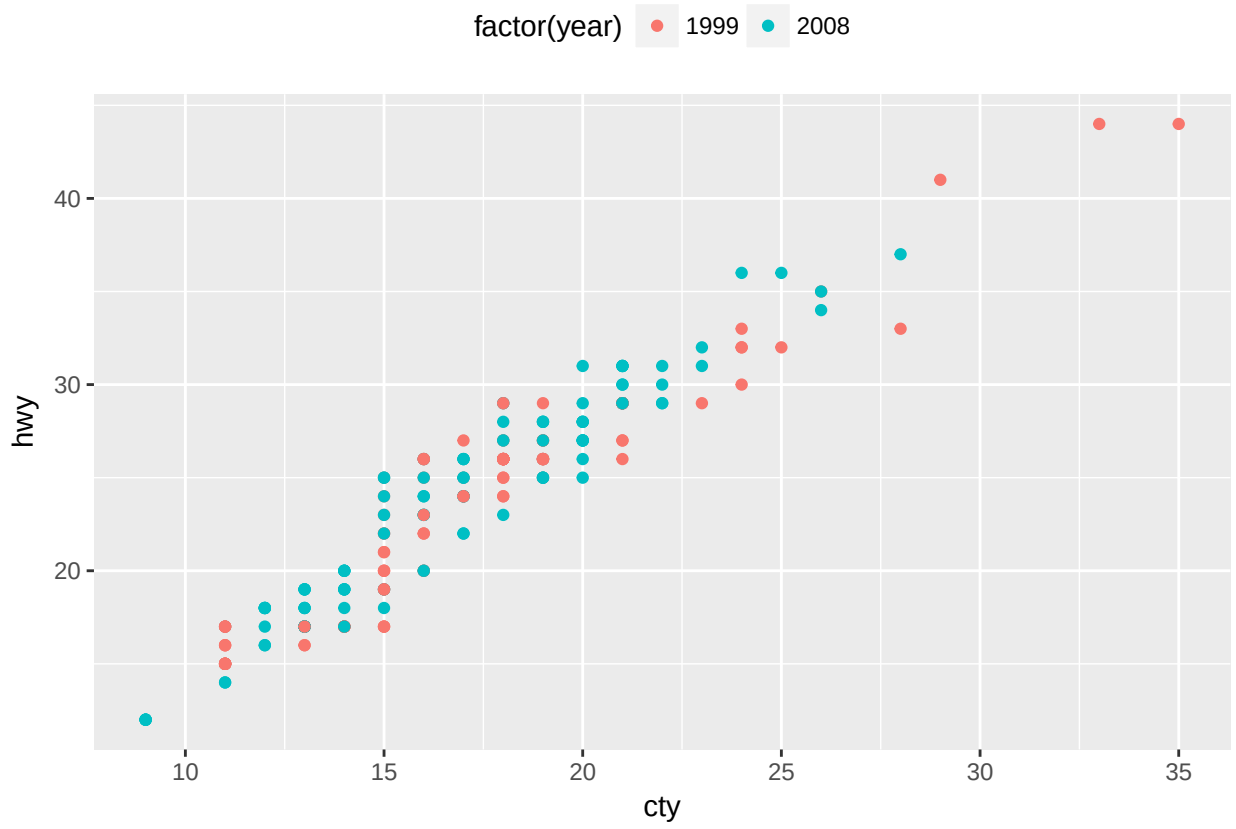
```
p1 + geom_point(aes(colour=factor(year))) +
  scale_colour_manual(name="",values=c("red","blue"),breaks=1999,labels="Old Data")
```



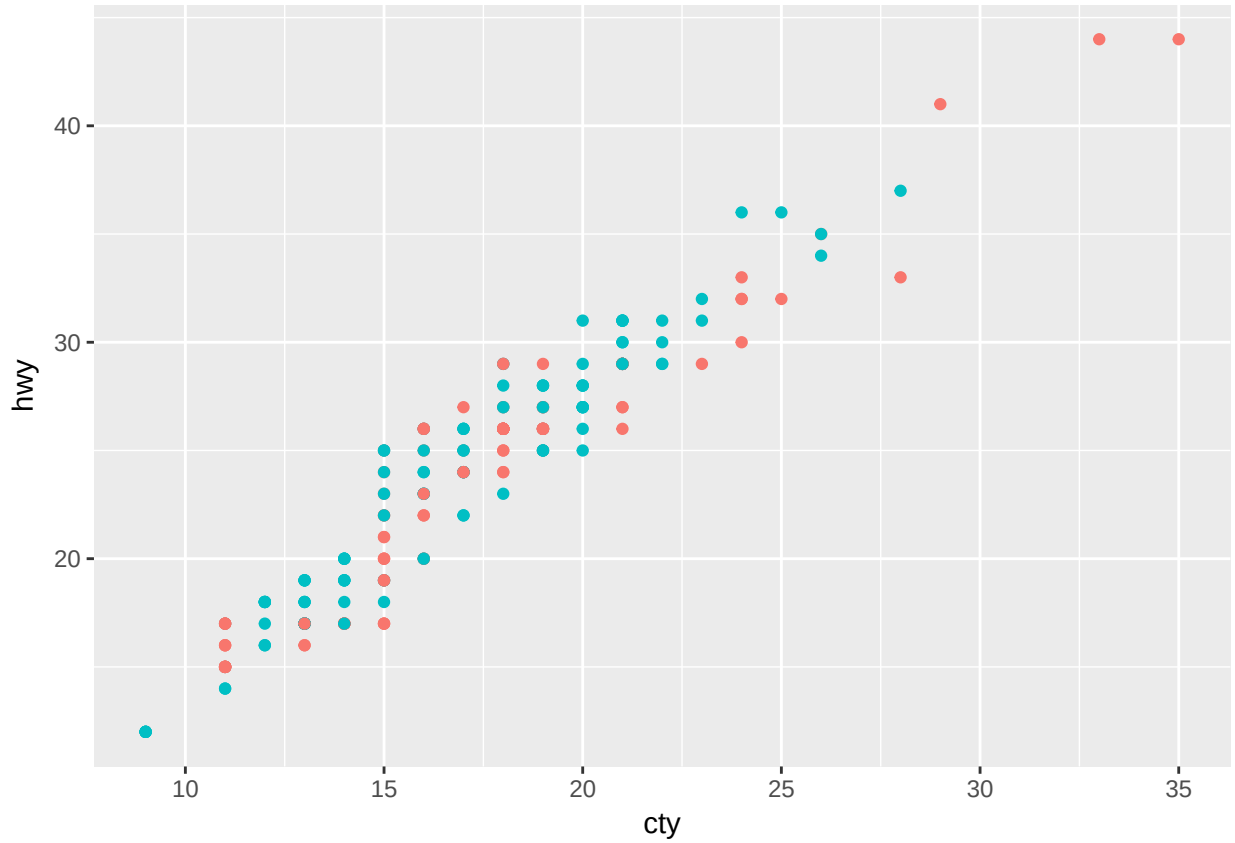
Exercise: Using the msleep data frame, plot the length of the sleep cycle versus the total amount of sleep. Indicate the animal's diet by colour. Add appropriate labels to the axes and colour legend, and make the colour key labels more reader-friendly.

We can further control the placement of the legend (or even turn it off) with the "`legend.position`" argument of the "`theme`" function:

```
p1 + theme(legend.position='top')
```

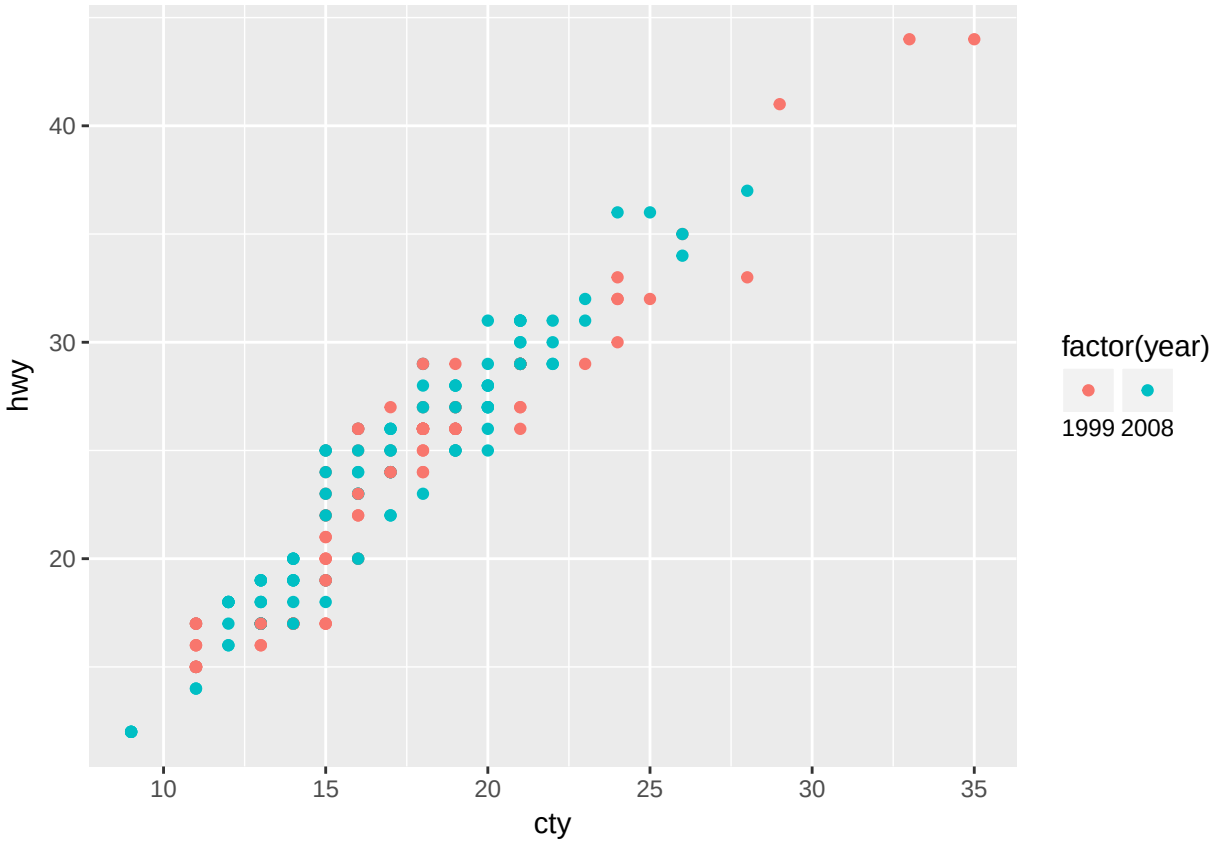



```
p1 + theme(legend.position="none")
```



Even finer control of the legend is available through the "guides"

```
p1 + guides(colour=guide_legend(label.position="bottom",ncol=2))
```

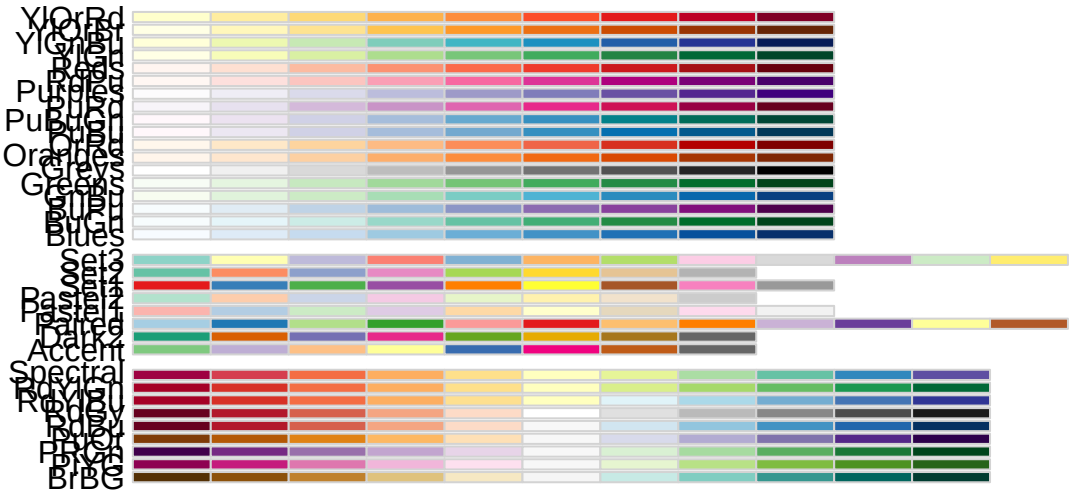


The default colour scale for continuous variables is `scale_colour_hue()`. You can use this scale to adjust the gradient used to colour from low to high values of the variable.

If you do not like the default colours but don't want to choose your own with "`scale_colour_manual`" (probably a good idea), you can use "`scale_colour_brewer`" which uses a preselected set of colours that work well together. (Note: this requires that the package RColorBrewer be installed.)

These are the options available

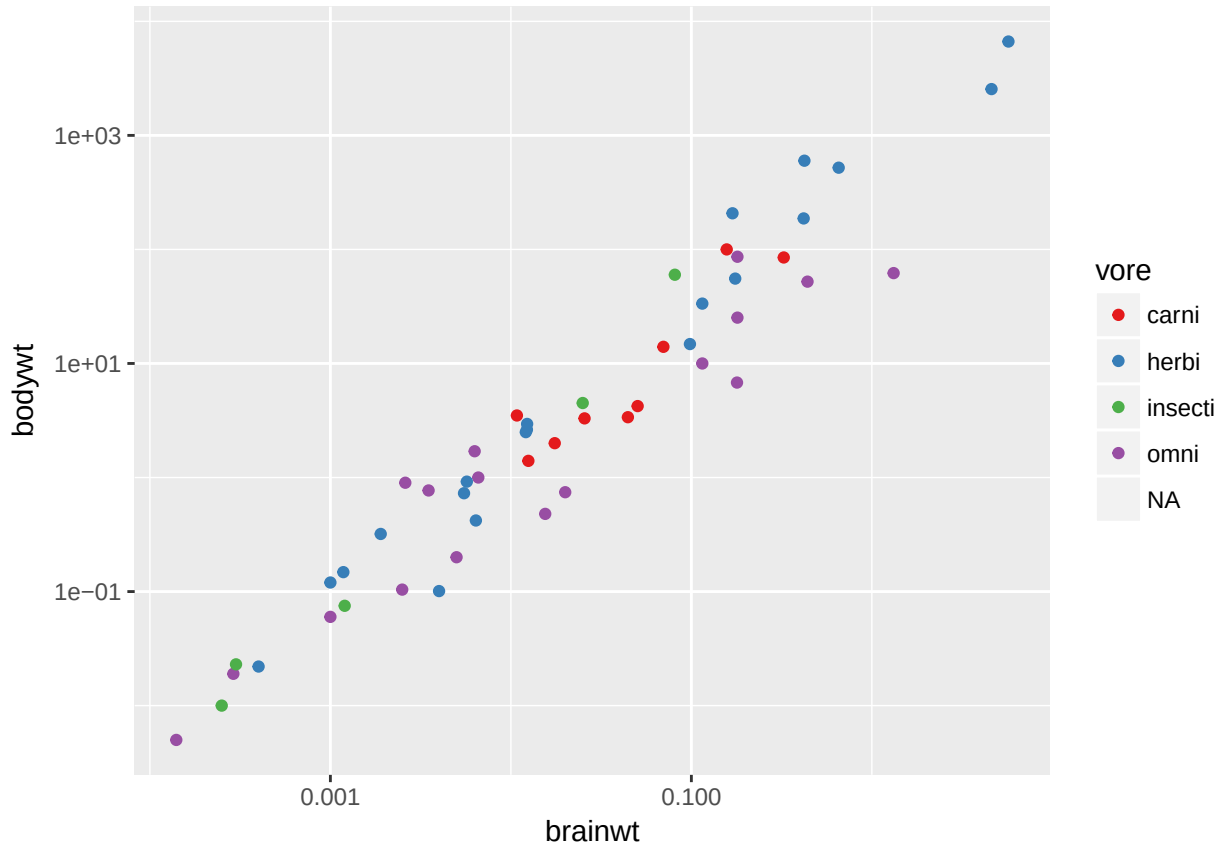
```
library(RColorBrewer)
display.brewer.all()
```



Examples for points:

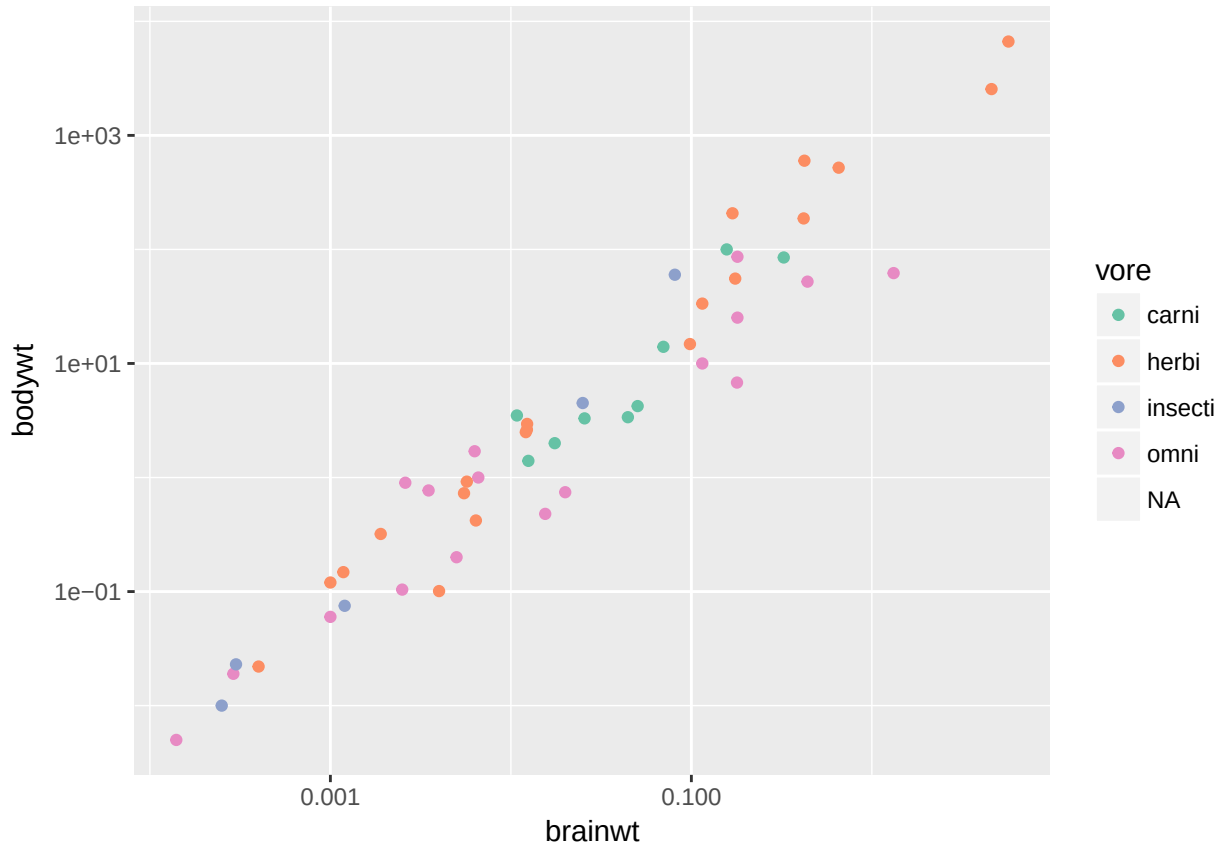
```
p1 <- ggplot(msleep,aes(brainwt,bodywt,colour=vore)) + geom_point() +
  scale_y_log10() + scale_x_log10()
p1 + scale_colour_brewer(palette="Set1")
```

```
## Warning: Removed 32 rows containing missing values (geom_point).
```



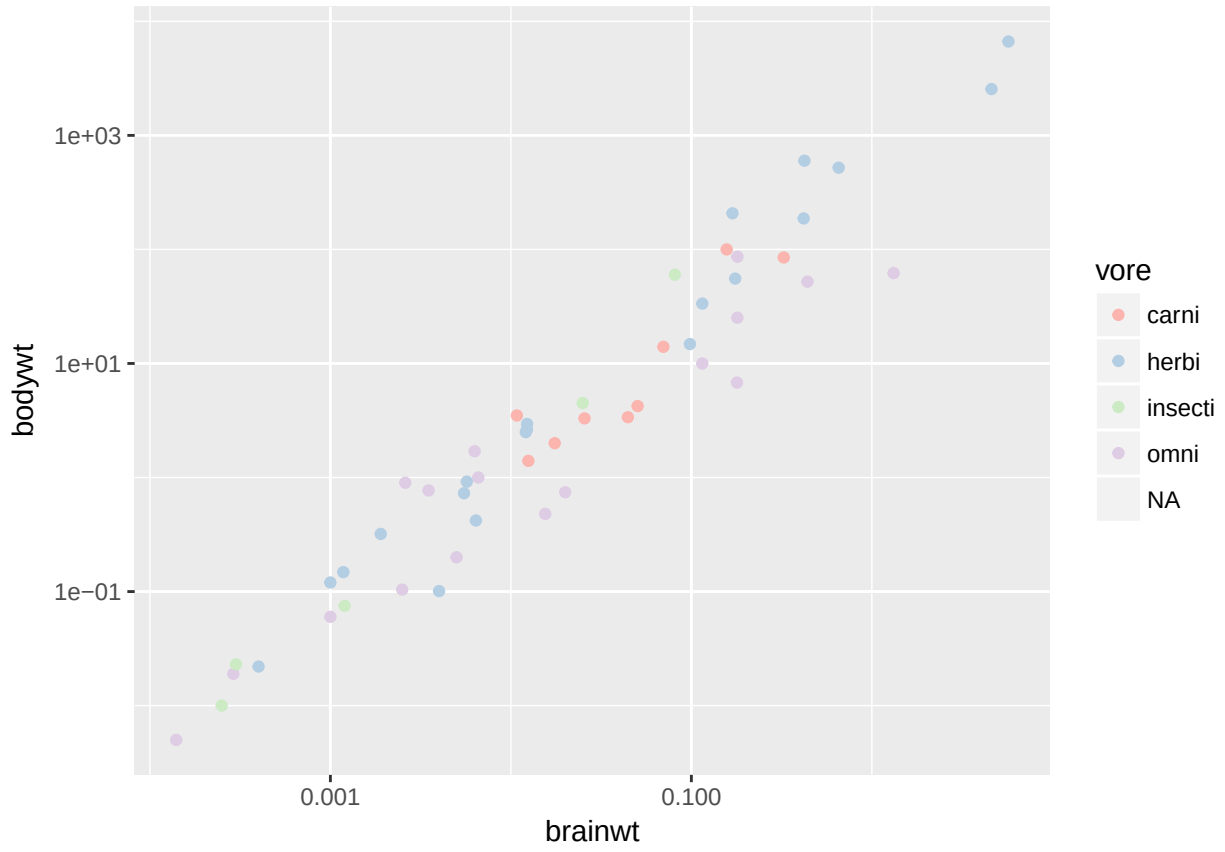

```
p1 + scale_colour_brewer(palette="Set2")
```

```
## Warning: Removed 32 rows containing missing values (geom_point).
```




```
p1 + scale_colour_brewer(palette="Pastel1")
```

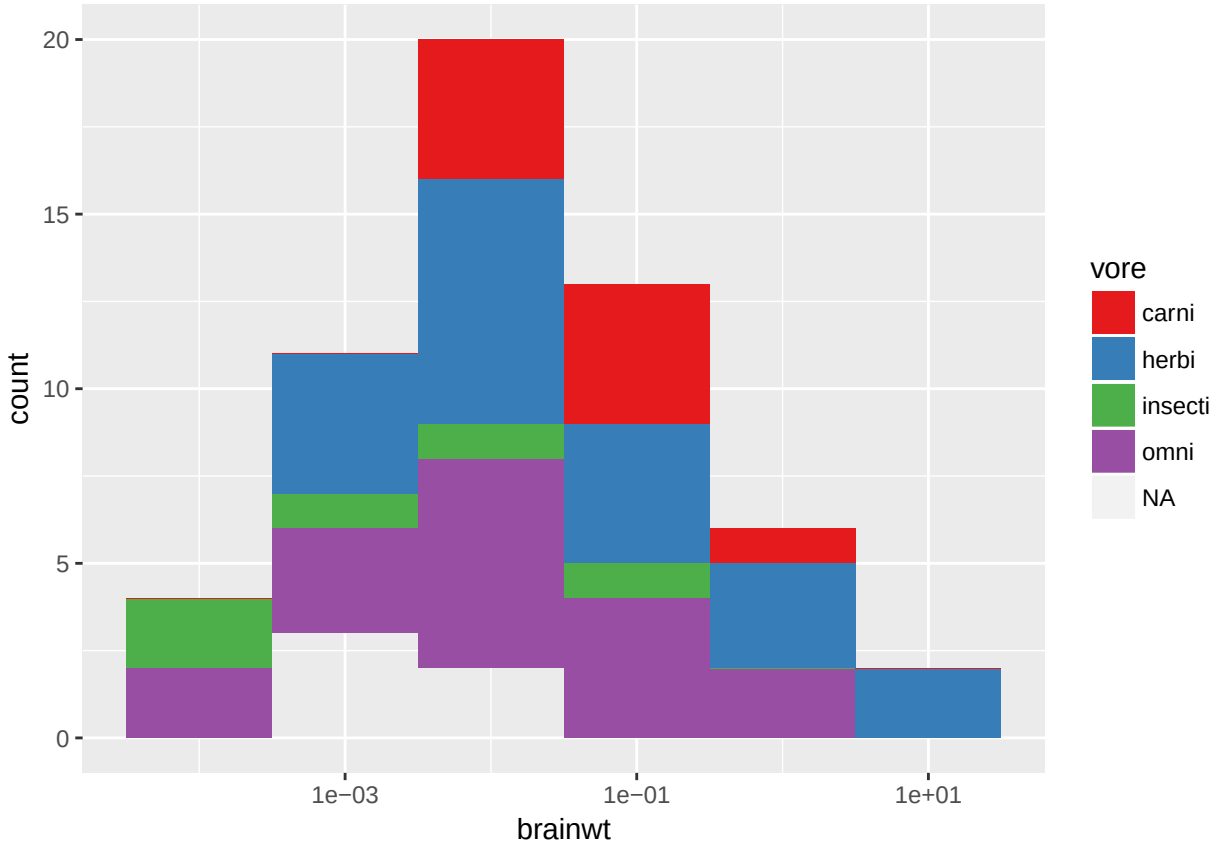
```
## Warning: Removed 32 rows containing missing values (geom_point).
```



Examples for areas

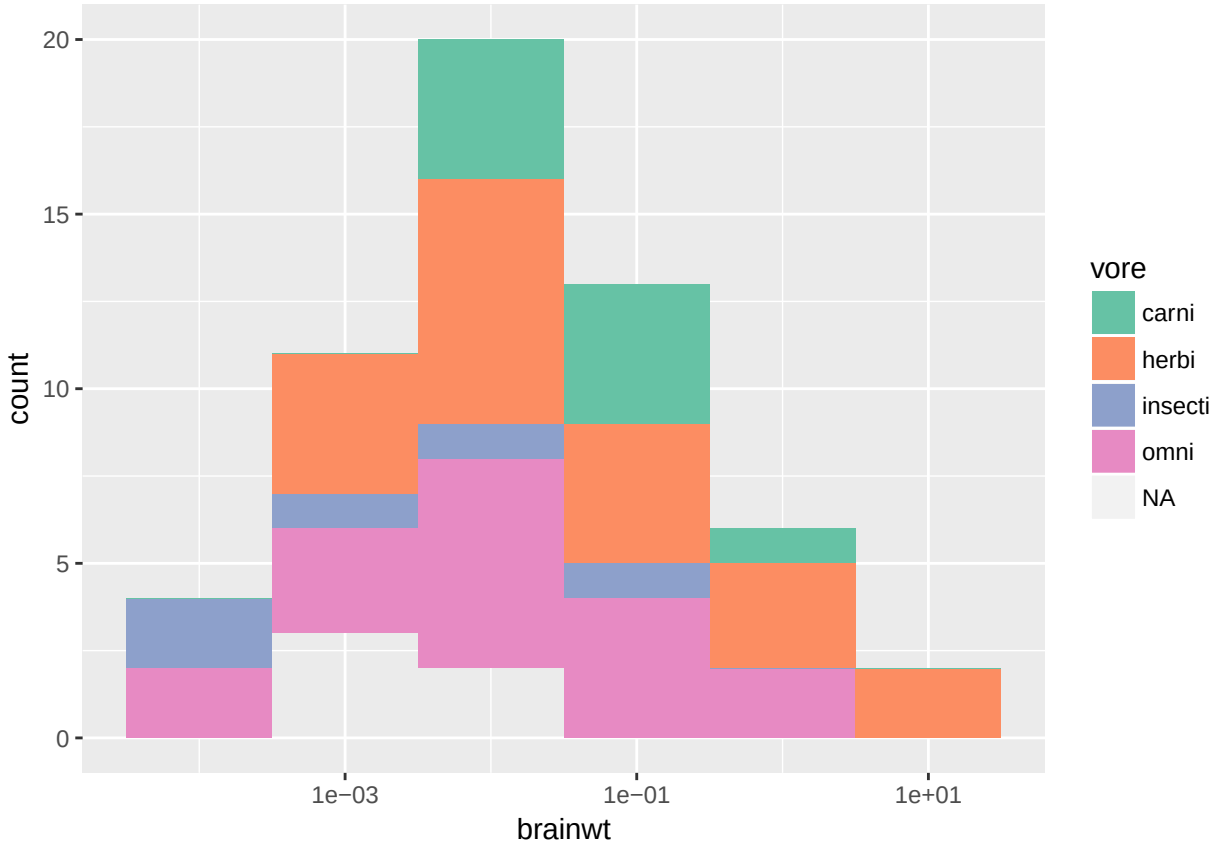
```
p2 <- ggplot(msleep,aes(brainwt,fill=vore)) + geom_histogram(binwidth=1) + scale_x_log10()
p2 + scale_fill_brewer(palette="Set1")
```

```
## Warning: Removed 27 rows containing non-finite values (stat_bin).
```



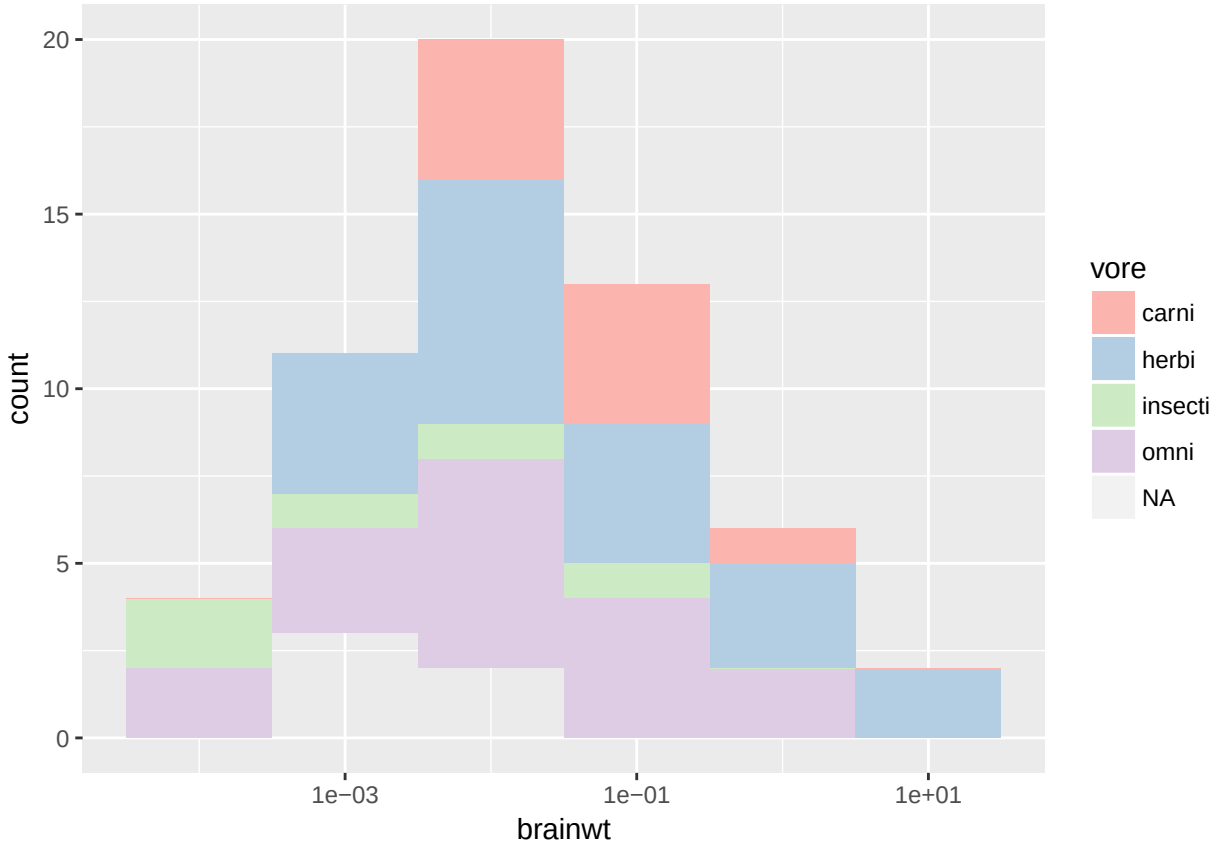

```
p2 + scale_fill_brewer(palette="Set2")
```

```
## Warning: Removed 27 rows containing non-finite values (stat_bin).
```




```
p2 + scale_fill_brewer(palette="Pastel1")
```

```
## Warning: Removed 27 rows containing non-finite values (stat_bin).
```

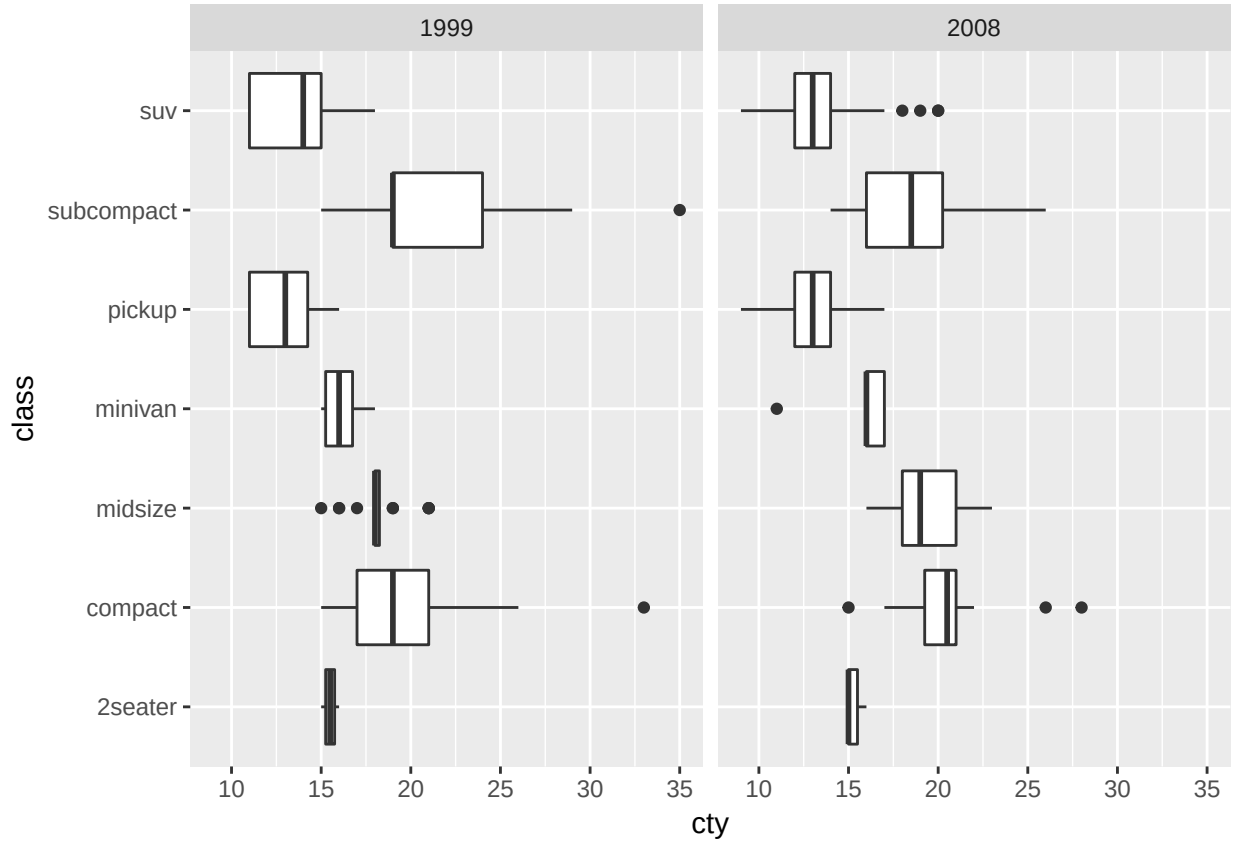


Coordinate systems

Sometimes you want to force certain features on the coordinate system in a graphic. This can be accomplished by the `coord_XXX` set of functions.

We can flip the x and y axis. In fact this is the only way to get horizontal boxplots.

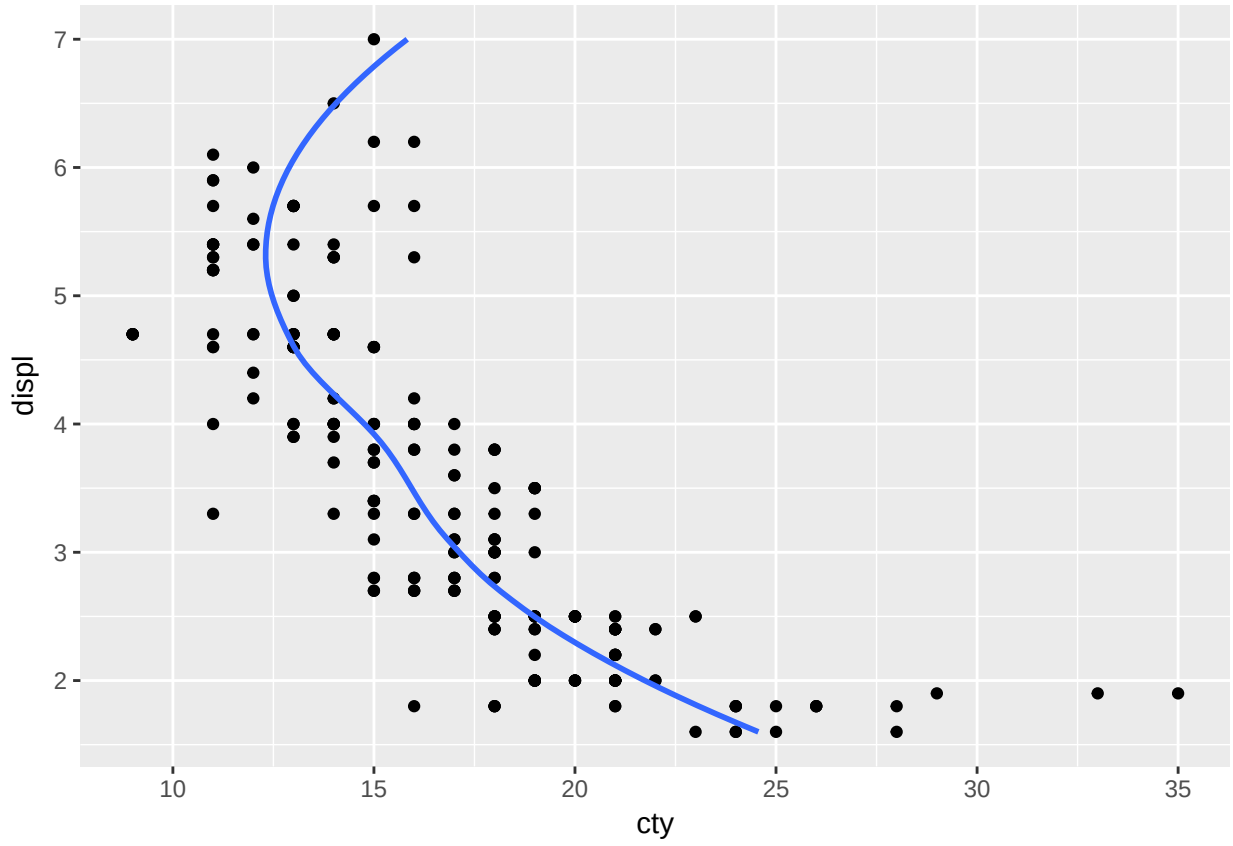
```
library(gapminder)
p1 <- ggplot(mpg, aes(class, cty)) + geom_boxplot() + facet_wrap("year")
p1 + coord_flip()
```



Notice that flipping the co-ordinates is not always the same as redoing the graph with the variable roles reversed, espe

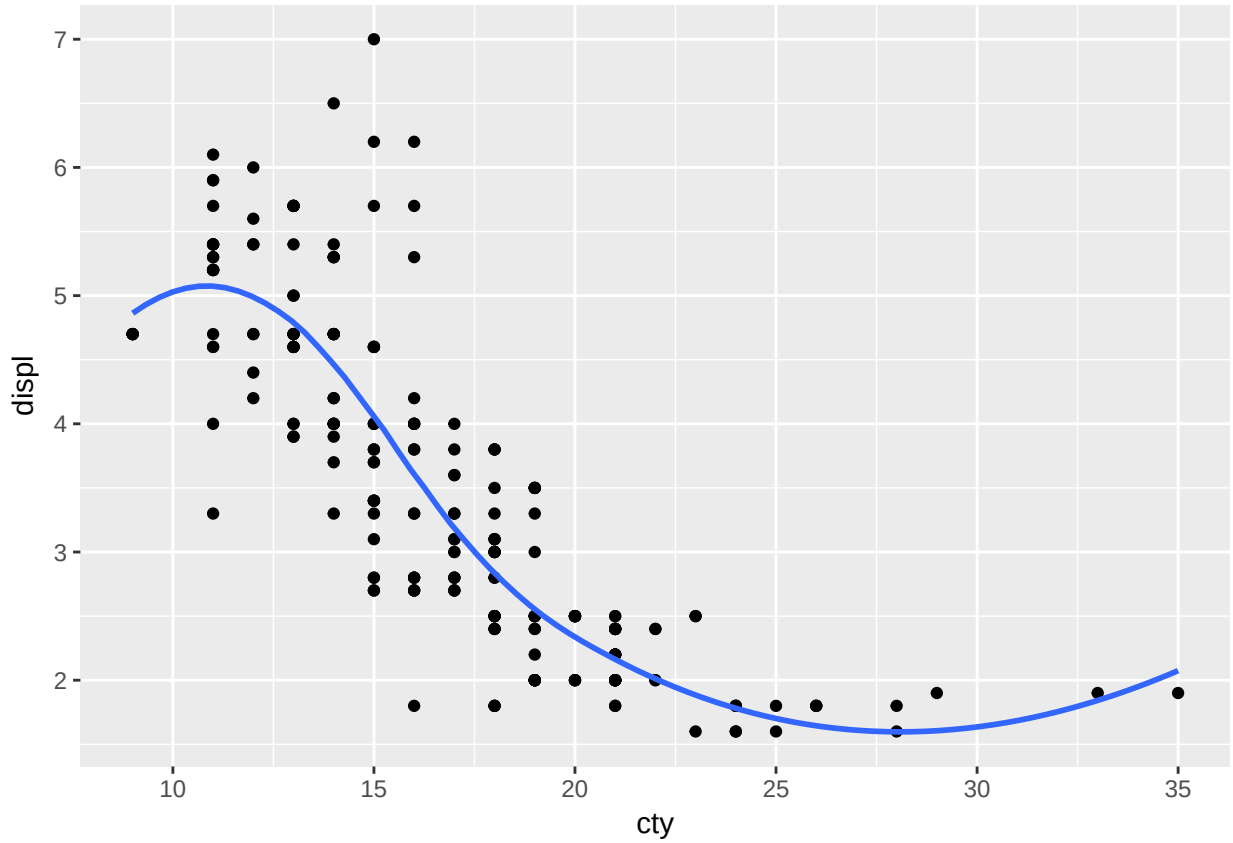
```
ggplot(mpg, aes(displ, cty)) +
  geom_point() +
  geom_smooth(se=FALSE) +
  coord_flip()
```

```
## `geom_smooth()` using method = 'loess'
```



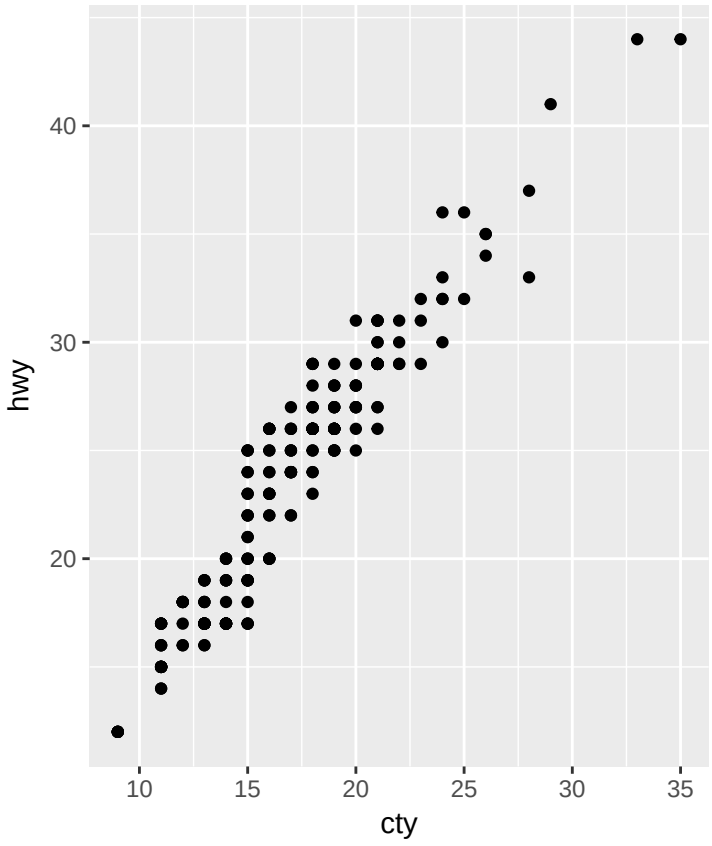

```
ggplot(mpg, aes(cty, displ)) +
  geom_point() +
  geom_smooth(se=FALSE)
```

```
## `geom_smooth()` using method = 'loess'
```



We can force a specific aspect ratio between the x and y axis

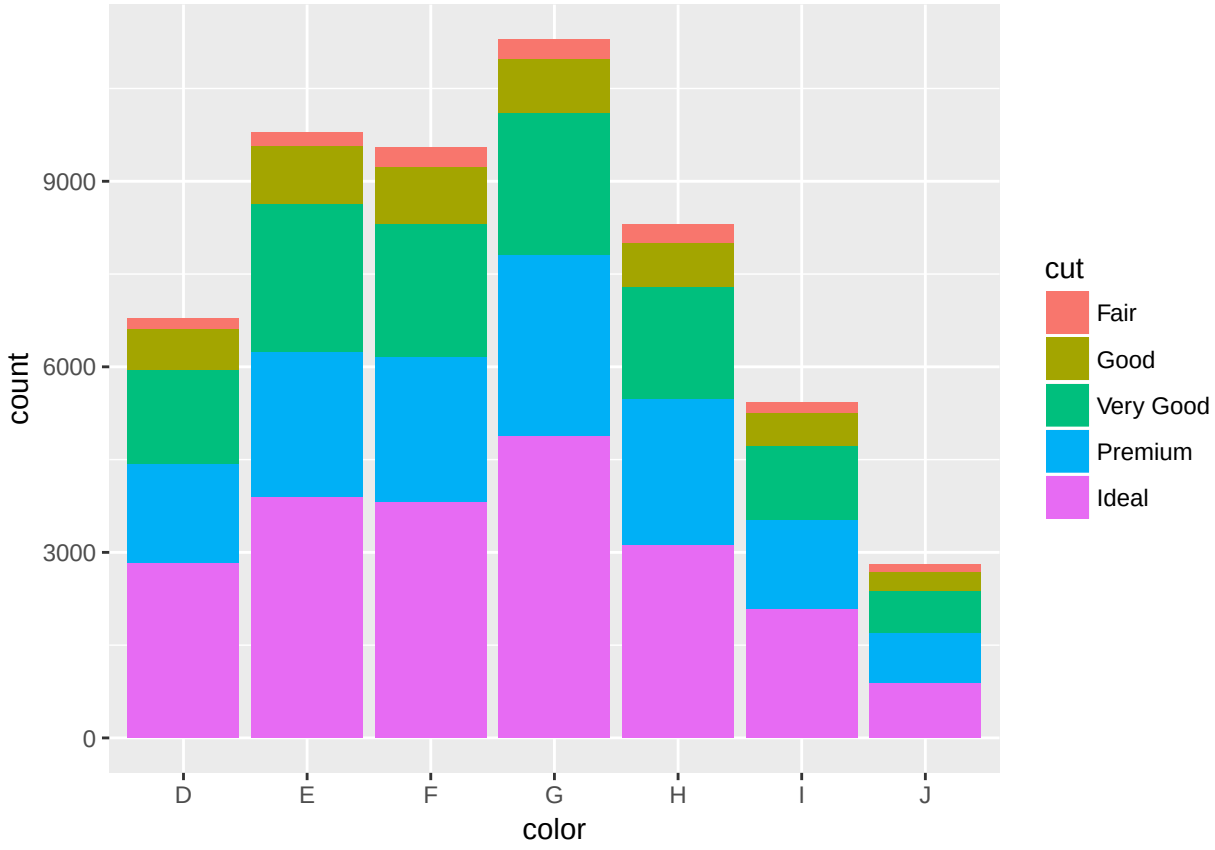
```
ggplot(mpg,aes(cty,hwy)) +
  geom_point() +
  coord_fixed()
```



Positioning

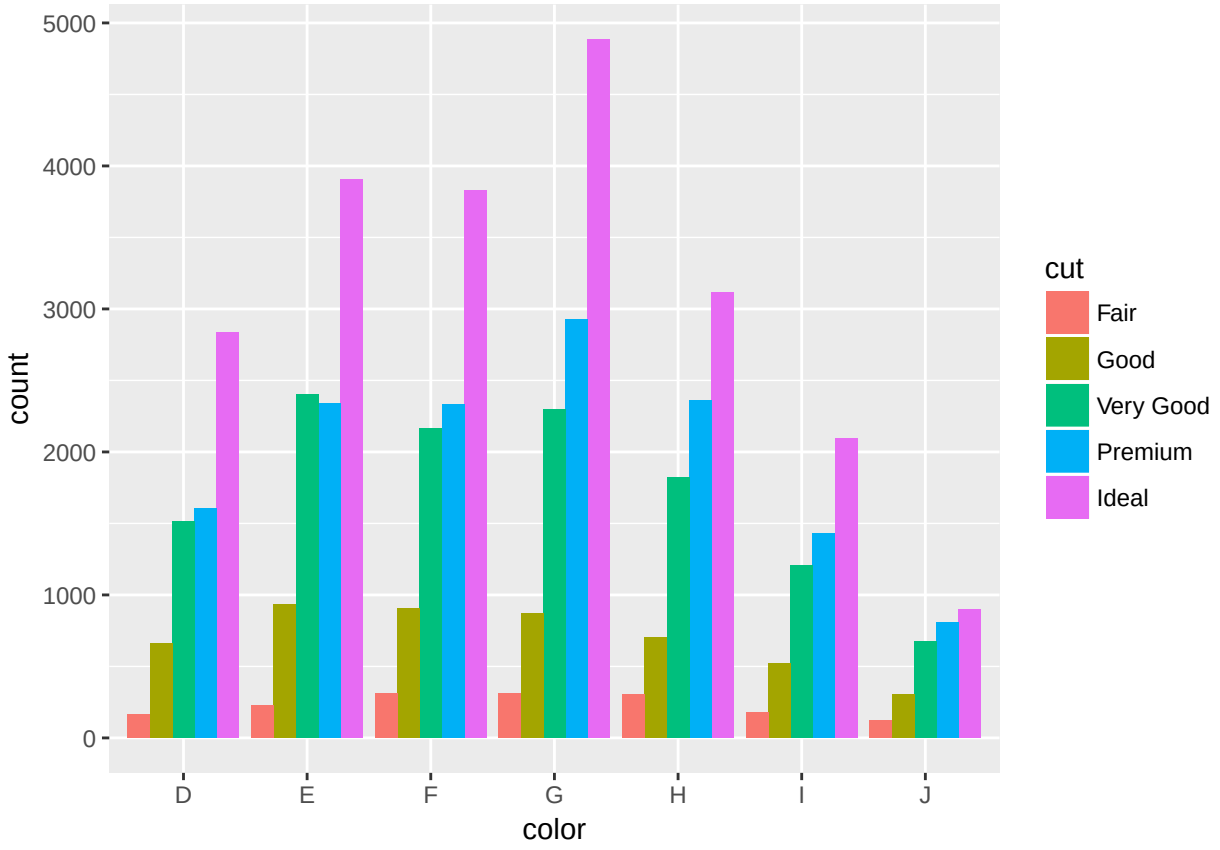
Position adjustments apply minor tweaks to the position of elements within a layer. They are particularly useful for bar-type plots, which by default stack the different groups:

```
p1 <- ggplot(diamonds,aes(color,fill=cut))
p1 + geom_bar()
```



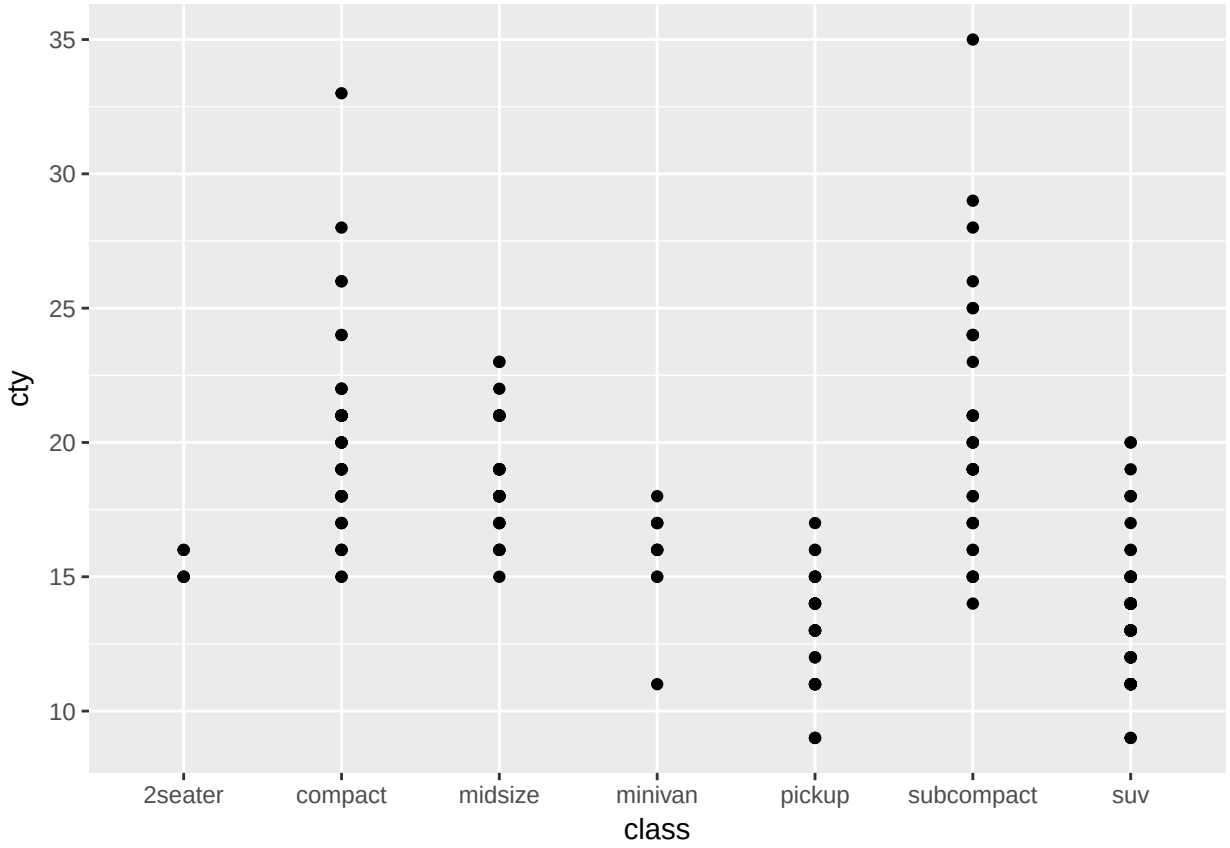
With argument '`position = "dodge"`', you can have ggplot place the grouped bars side-by-side:

```
p1 + geom_bar(position="dodge")
```

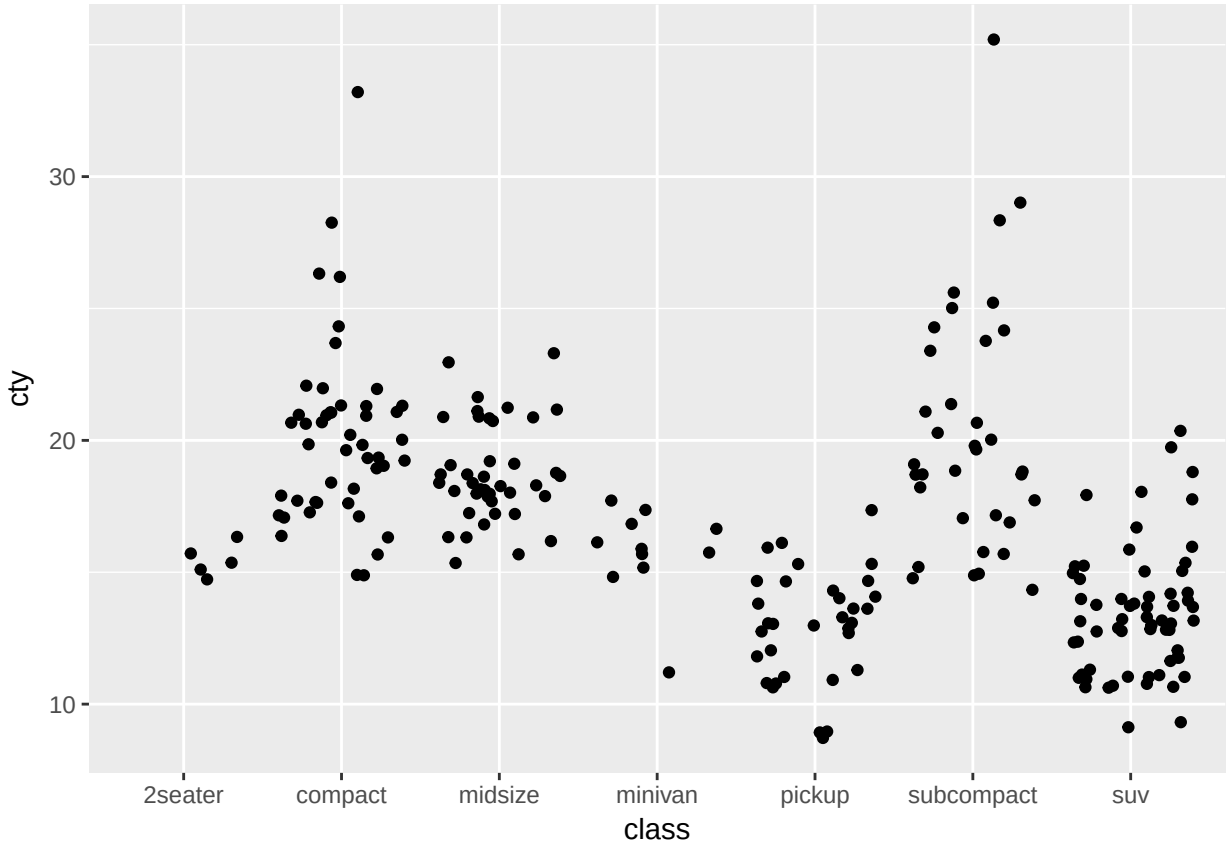


Point geoms use different position adjustments, primarily to reduce overplotting by slightly moving the marks. "Jitter" moves the points randomly:

```
ggplot(mpg, aes(class, cty)) + geom_point()
```




```
ggplot(mpg, aes(class, cty)) + geom_point(position = 'jitter')
```

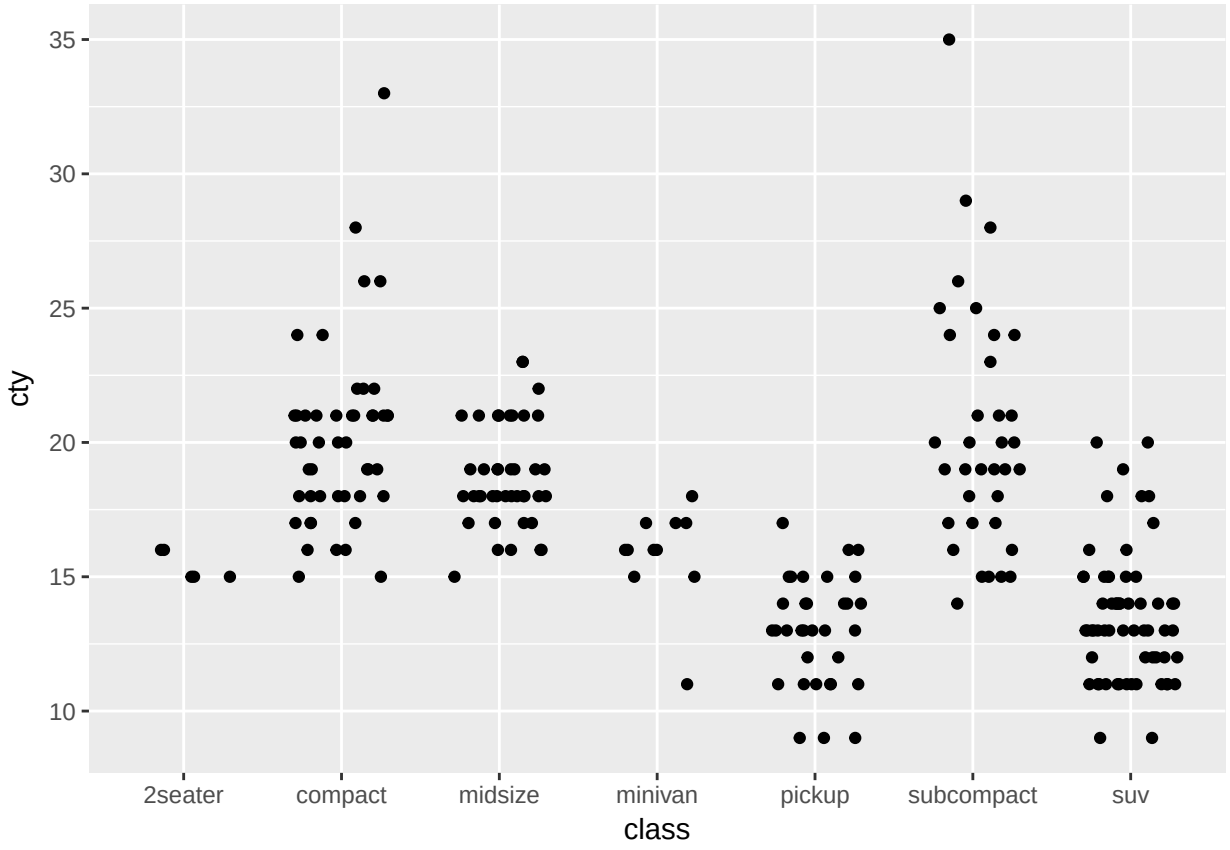


The default amount of jitter is quite large, but you can control it with "`width`" and "`height`" arguments to jitter.

"{r} ggplot(mpg, aes(class, cty)) + geom_point(position=position_jitter(width=.3, height=0)) "'

This is getting kind of verbose, so there is a convenience "`geom_jitter`":

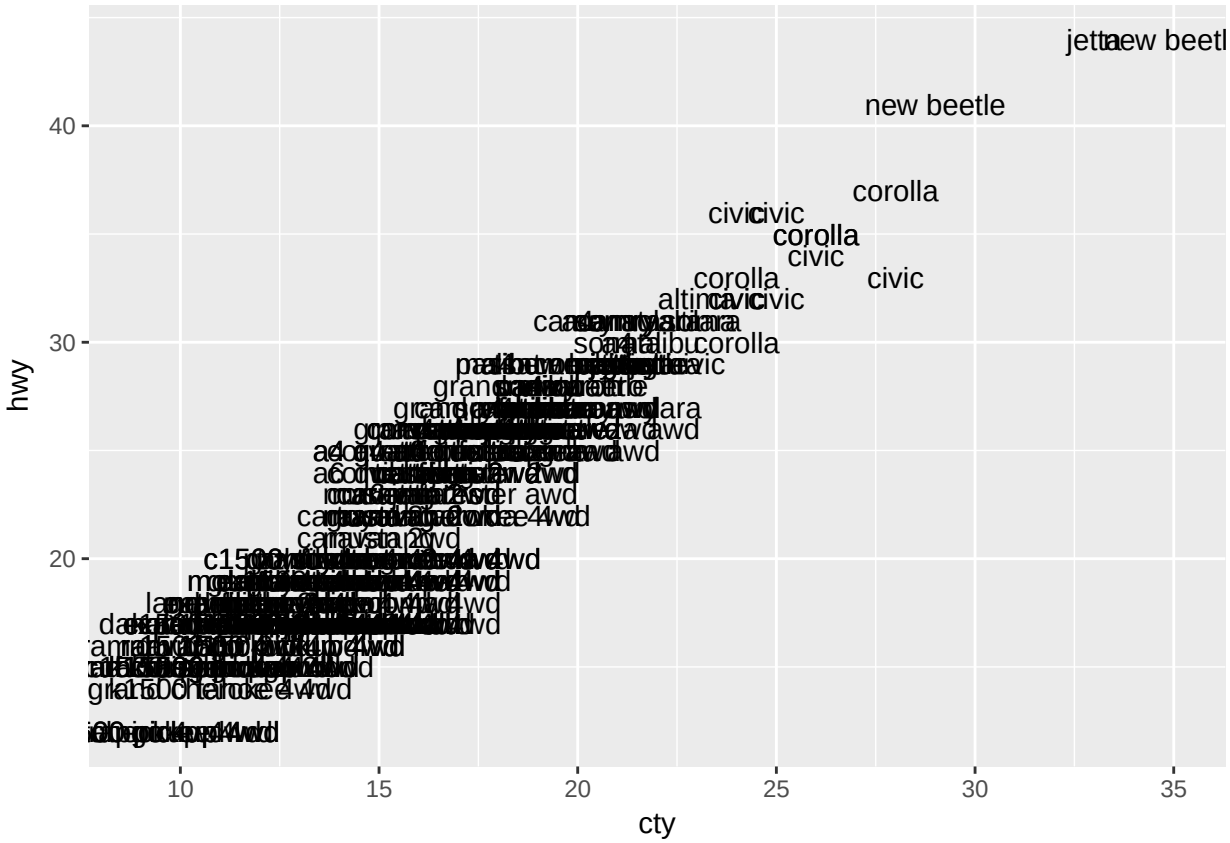
```
ggplot(mpg, aes(class, cty)) +
  geom_jitter(width=.3, height=0)
```



Adding text to a plot

You can add text to any plot using "`geom_text`", with the text to display in the "`label`" aesthetic:

```
ggplot(mpg, aes(cty, hwy, label = model)) + geom_text()
```

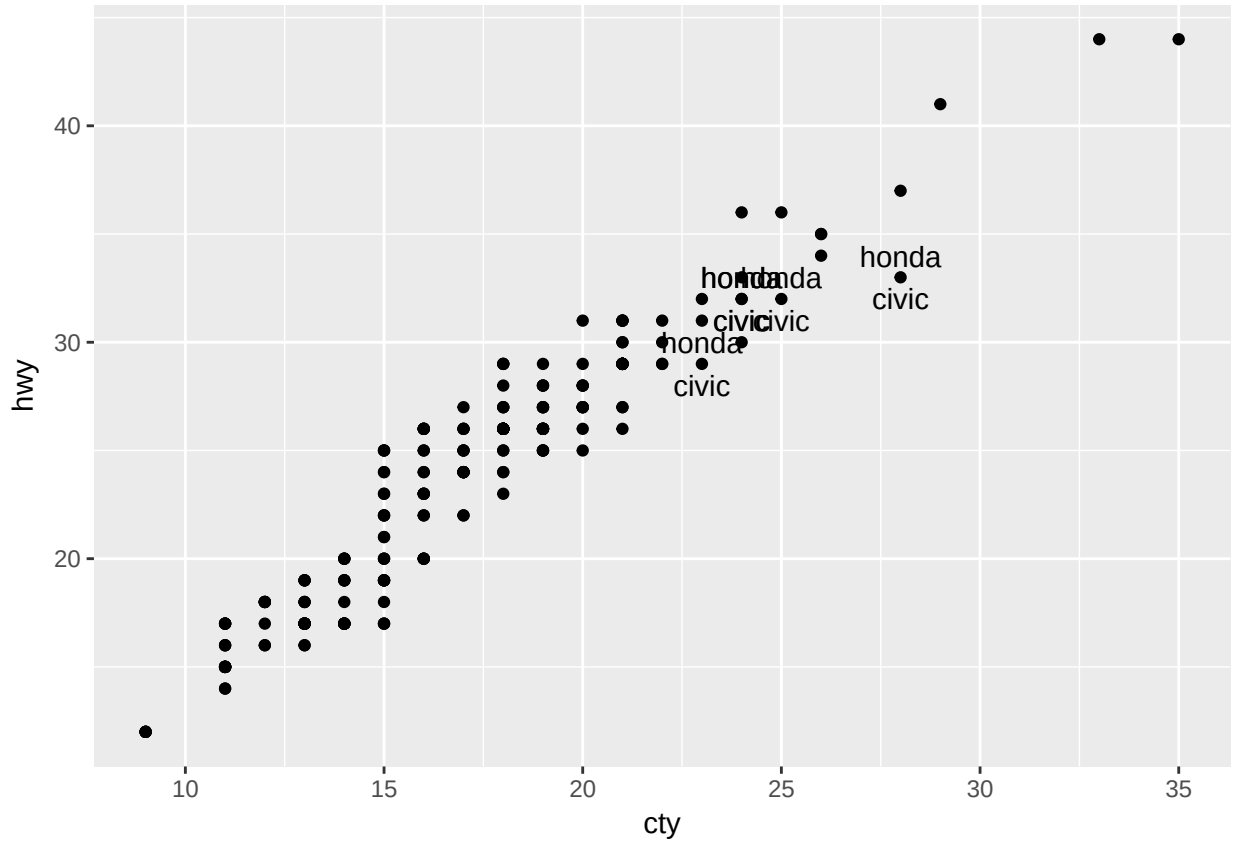


This geom accepts many additional aesthetics that affect the way the text is printed. Look at the help documentation.

Clearly, the text geom should be used judiciously, because it quickly becomes unreadable because of overlap. You can use the `nudge_x` and `nudge_y` arguments to "`geom_text`" to manually adjust the position (as well as the "`hjust`" and "`vjust`" aesthetics for alignment relative to the x-y position), but when there are too many points there won't be any good way to label them all.

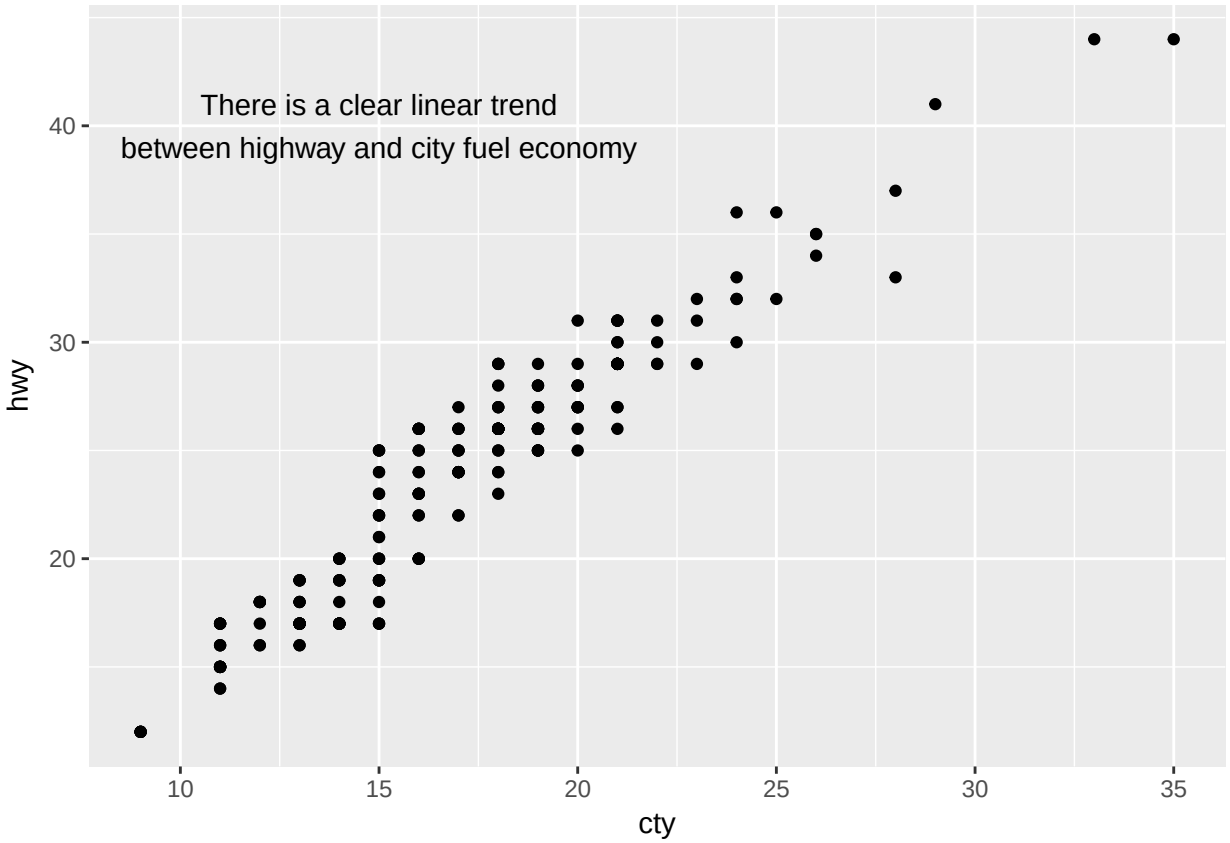
To label specific points, just give a custom "`data`" argument to `geom_text`. For instance, to label just the smallest engines:

```
p1 <- ggplot(mpg, aes(cty, hwy)) + geom_point()
p1 + geom_text(aes(label=paste(manufacturer,model,sep="\n")),
               data = filter(mpg, displ == min(displ)))
```



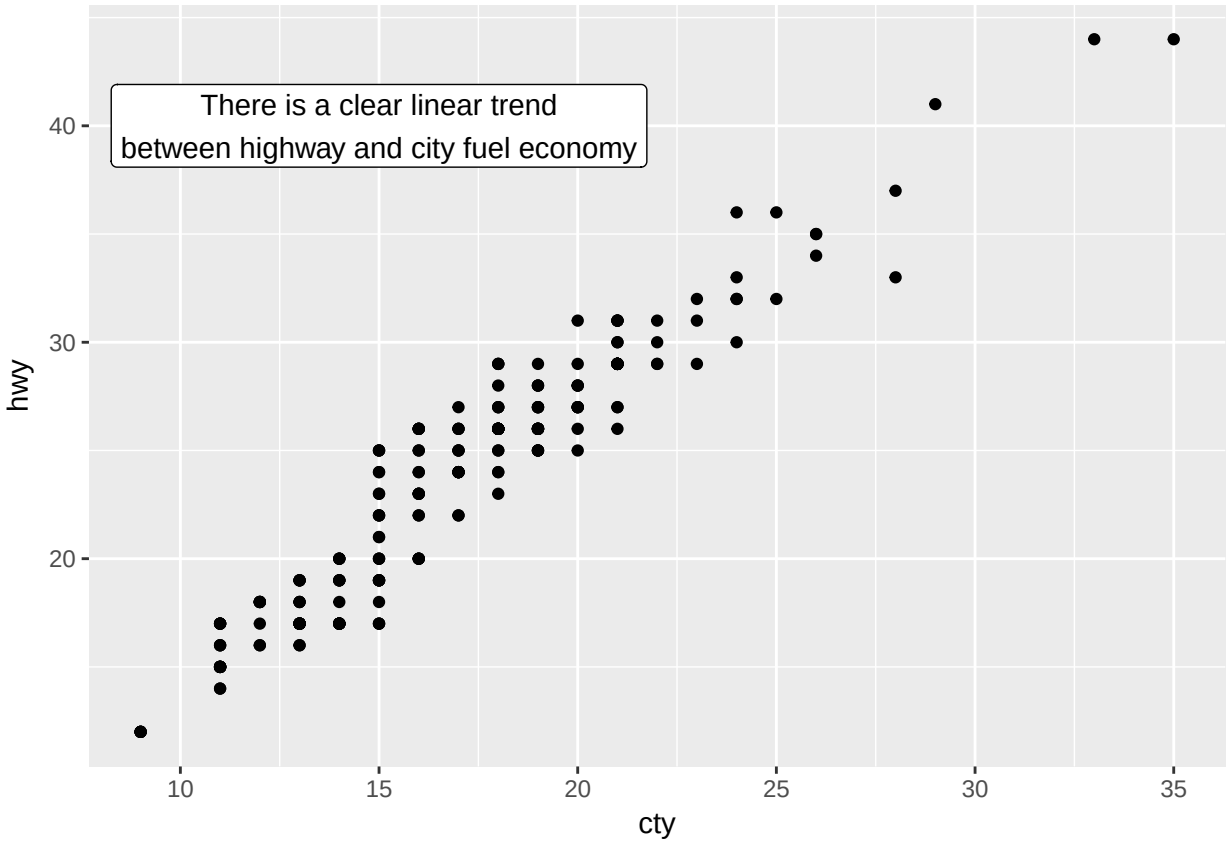
To add just a one-off text annotation, you can also use the convenience "`annotate`" function.

```
p1 + annotate('text', 15, 40, label="There is a clear linear trend\nbetween highway and city fuel econo
```



The `label` geom puts the text inside a rectangle for a better visibility, which is particularly handy for this purpose, if also (currently) considerably slower than the `text` geom:

```
p1 + annotate('label', 15, 40, label="There is a clear linear trend\nbetween highway and city fuel econ
```



** Exercise: Using `msleep` data, plot total sleep against body weight using the species name instead of a point. (Use a log axis if warranted.) **