Visualisation in R

Sam Keat

PhD Student - UK Dementia Research Institute, Cardiff

KeatS@cardiff.ac.uk

UK Dementia Research Institute

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

Alzheimer's Society
United Against Dementia


Visualisation
Making your data come to life!

Objectives

- How data can be visualised in R
- Learn how to adjust graphics
- Get familiar with `ggplot2` and it's *grammar of graphics* to create amazing figures

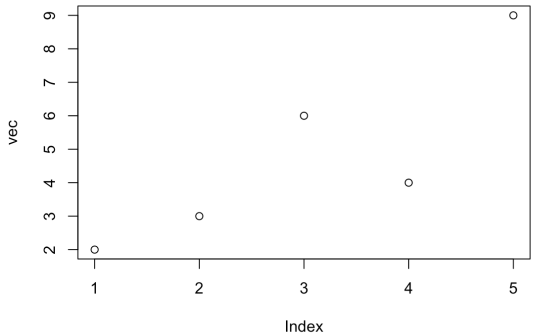
Data Visualisation - An Overview

- The overall aim of visualising data:
  - Make all your plots as self explanatory as possible!
- For these lectures, we will focus on `ggplot2`, a tidyverse package and one of the most useful visualisation tools for a bioinformatician
- Versatile, powerful and easy to use (once you learn the *graphics grammar*) package
- Has multiple add-ons, such as `ggrepel` (for text labels) and `ggpubr` (for publication-ready plots) or `patchwork` (for combining multiple plots)

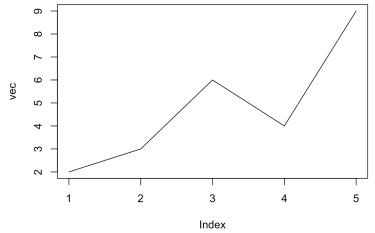
Data Visualisation - An Overview

- Graphs can be easily generated with the base R syntax

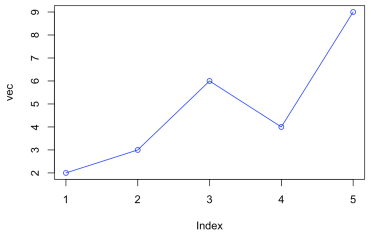
```r
## some simple base R plots
vec <- c(2,3,6,4,9)
plot(vec)
```



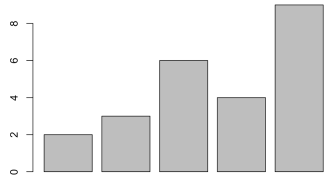

```r
plot(vec,
     type = "l")
```




```r
plot(vec,
     type = "o",
     col = "blue")
```



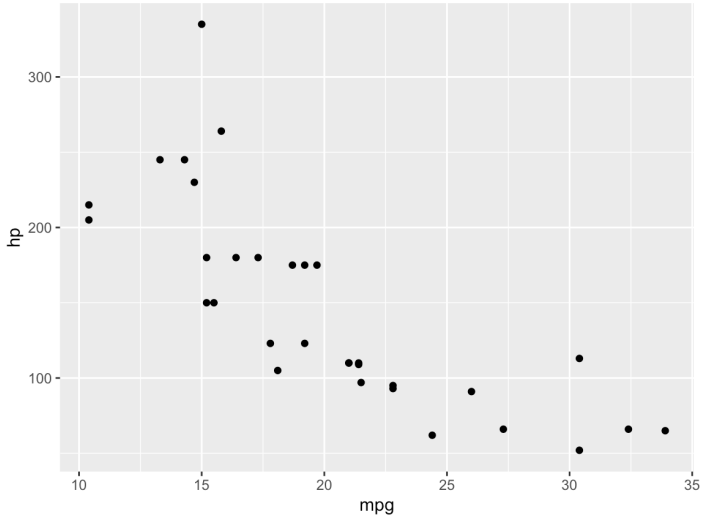

```r
barplot(vec)
```



ggplot2

- `ggplot2` is a lot more useful and user friendly than base R, making plots look a lot nicer and with more options for building and displaying graphics
- We'll start with an old favourite, the mtcars dataset!

```
library(tidyverse) # This includes
ggplot2! To just load that,
library(ggplot2)

ggplot(data = mtcars) +
    geom_point(mapping =
        aes(x = mpg, y = hp))
```

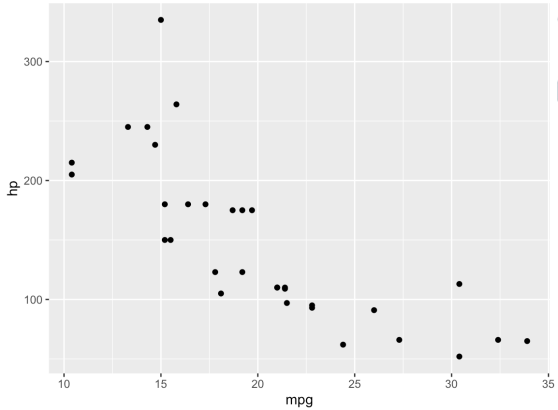


ggplot2

- `ggplot()` is the main function, and this creates the initial ggplot object where we then add multiple **layers**
- A layer is a collection of geometric elements (`geoms`) and statistical transformations
- An easy example of a "geom" element layer is `geom_point()`, which adds a scatter plot
- Aesthetic mappings (`aes`) are specified with `aes()`. This is how variables (columns) in the input data are mapped to visual, or "aesthetic", properties.
- You can give **global** "aesthetics" to a plot (will appear in every layer) by specifying this in the `ggplot()` function, or **local** aesthetics in individual layers (such as in `geom_point()`).

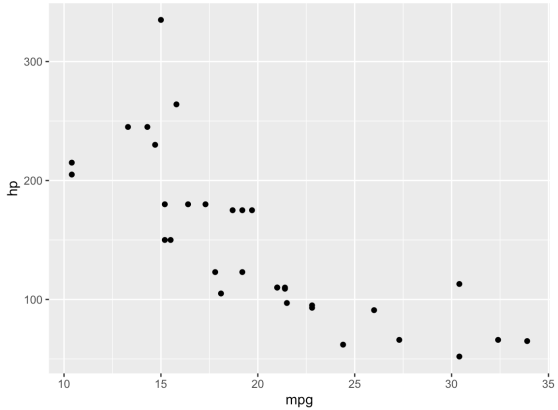
ggplot2

```
ggplot(data = mtcars,
    mapping = aes(x = mpg, y = hp)) +
    geom_point()
```



- We can also map the aesthetics via `aes_string()`:

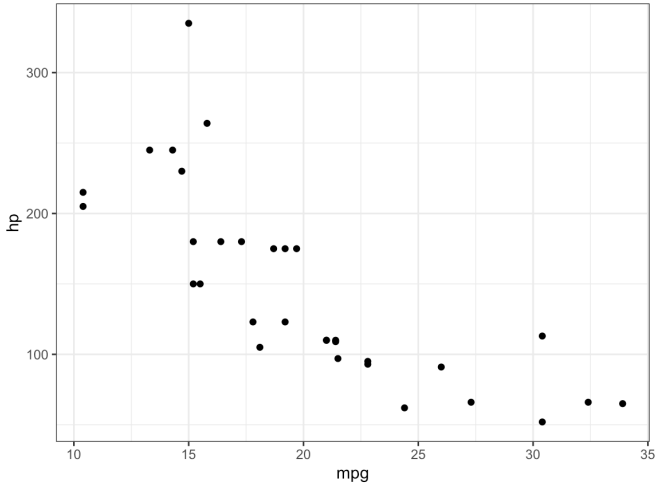
```
x_var <- "mpg"
y_var <- "hp"

ggplot(data = mtcars,
    mapping = aes_string(x = x_var, y = y_var)) +
    geom_point()
```



Themes

- A “**theme**” controls the finer points of the plot, like the font size and background colour
- This is essentially **customising the non-data elements**
- For example, change the default grey background to white background

```
ggplot(data = mtcars,
    mapping = aes_string(x = x_var, y = y_var)) +
    geom_point() +
    theme_bw()
```



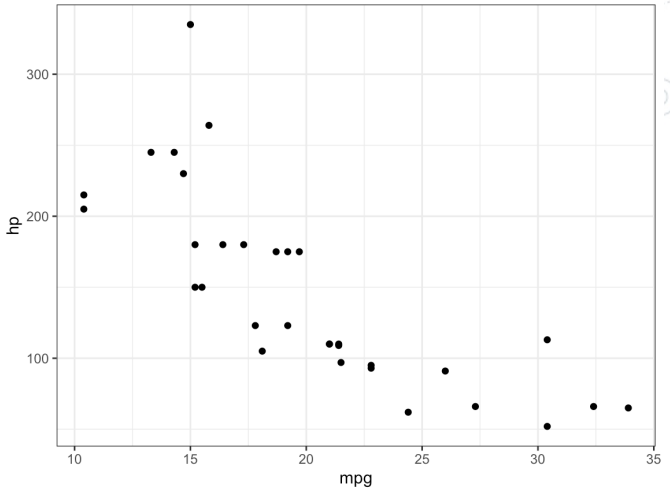
- I personally prefer a white background

(the default is `theme_grey()`)

Themes

- These themes can be set **globally**, so for **all plots**, using the `theme_set()` function

```
theme_set(theme_bw())

ggplot(data = mtcars,
    mapping = aes_string(x = x_var, y = y_var)) +
    geom_point()
```



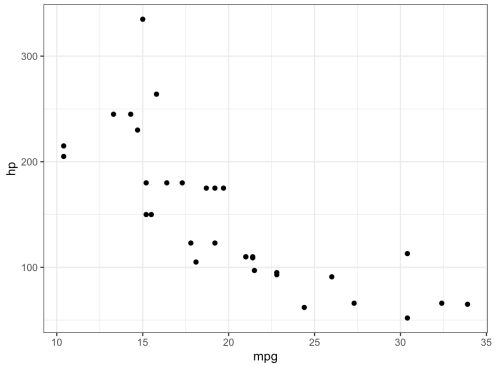
- You can see some more themes provided by ggplot2 here:

https://r4ds.had.co.nz/graphics-for-communication.html#themes

qplot()

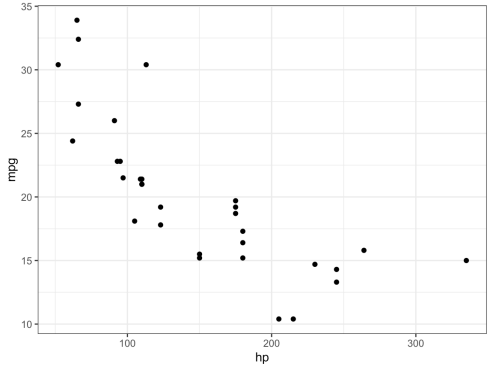
- A quicker version of ggplot! Good for very basic figures:

```
qplot(mpg, hp, data = mtcars)
```



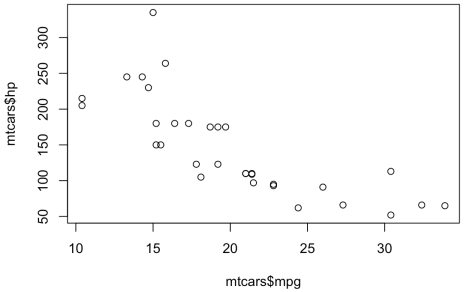
- Which is exactly the same as:

```
ggplot(mtcars, aes(hp ,mpg)) +
    geom_point()
```



Same syntax as base plot() !

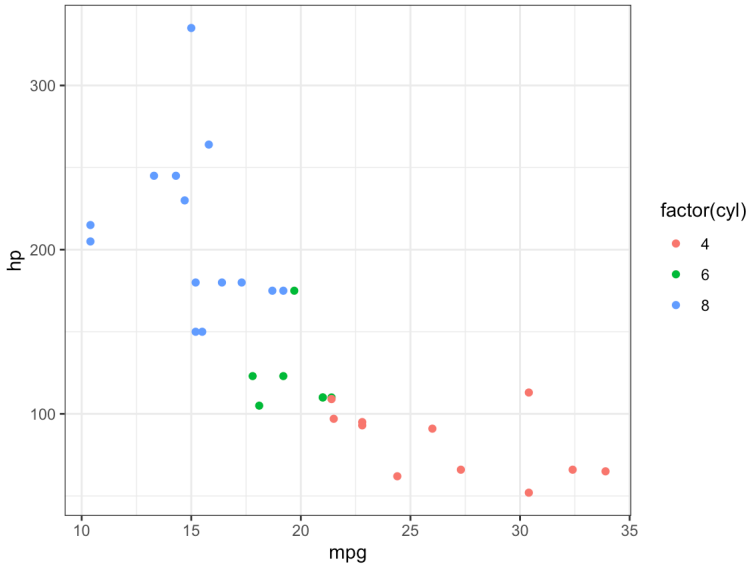
```
plot(mtcars$mpg, mtcars$hp)
```



Aesthetics: Colour, Size and Shape

- You can map a colour to data points by simply giving it the name of a colour (e.g. “brickred”), but mapping a variable to the data points can allow you to distinguish between different data points using different colour
- For example, in mtcars we can map the number of cylinders to the **colour** aesthetic (or color if you want to spell it wrong…)

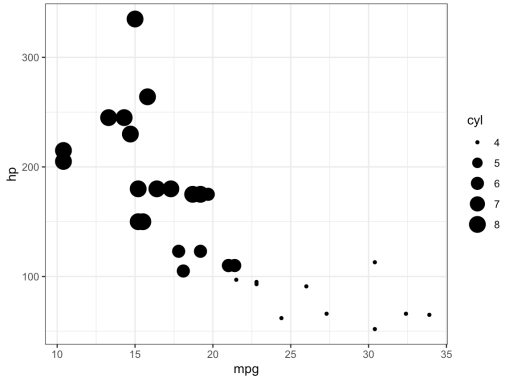
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_point()
```



Aesthetics: Colour, Size and Shape

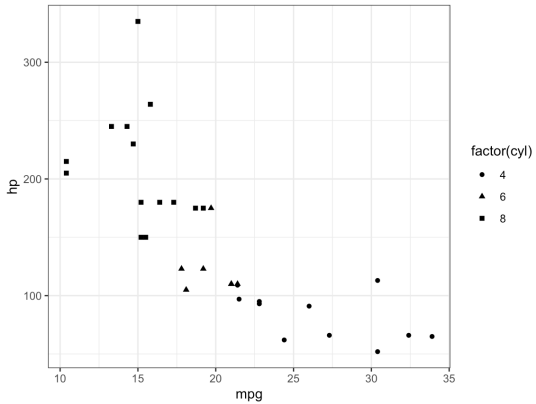
- We can also map the number of cylinders to the **size** aesthetic:

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  size = cyl)) +
    geom_point()
```



- Or even to the **shape** aesthetic:

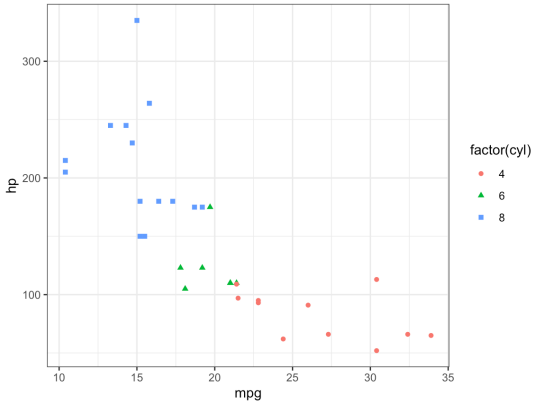
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  shape = factor(cyl))) +
    geom_point()
```



Aesthetics: Colour, Size and Shape

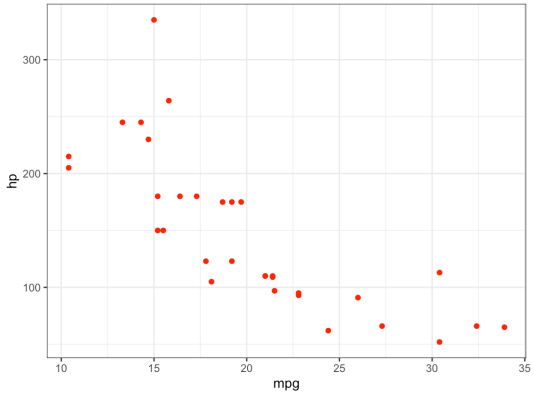
- Or we can combine both **colour** and **shape!**

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  shape = factor(cyl),
                  colour = factor(cyl))) +
    geom_point()
```



- We can set an aesthetic **locally** outside of aes()

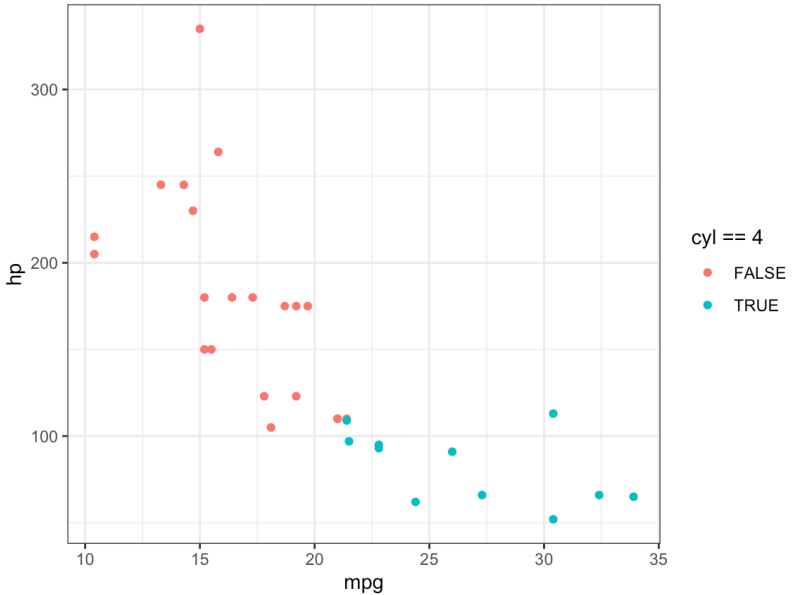
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp)) +
    geom_point(colour = "red")
```



Aesthetics: Colour, Size and Shape

- We can even map an aesthetic to a datapoint based on a **condition**, i.e. only change colour when a certain condition is met.
- For example here, the colour varies depending on whether the car has 4 cylinders or not (`cyl == 4` being TRUE or FALSE)

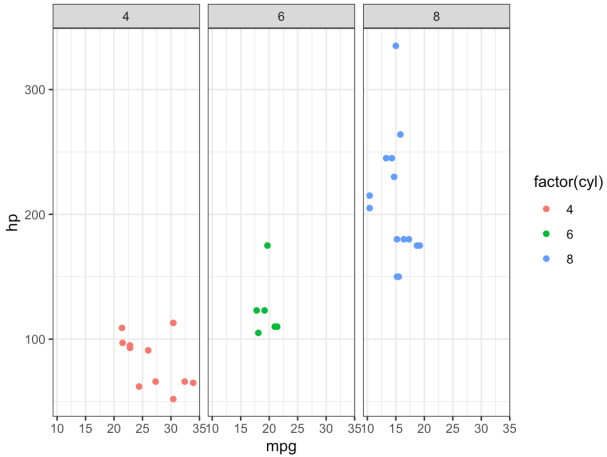
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = cyl == 4)) +
    geom_point()
```



Facets

- A ”facet” is one section of something that has many sections.
- A ”dacet” in ggplot allows you to **break up the data** into different subsets and plot **individual panels** based on it
- Creates “subplots” or **panel-like figures**
- Really useful when you’ve got categorical variables (such as gender)

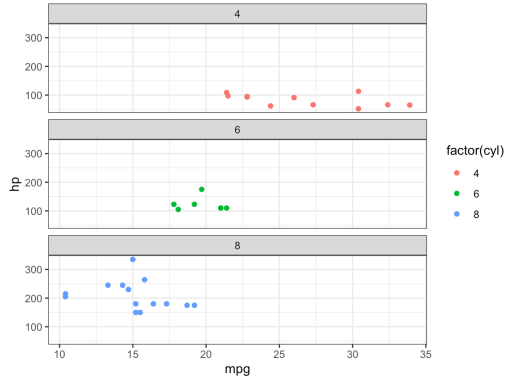
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_point() +
    facet_wrap(~ cyl)
```



Facets

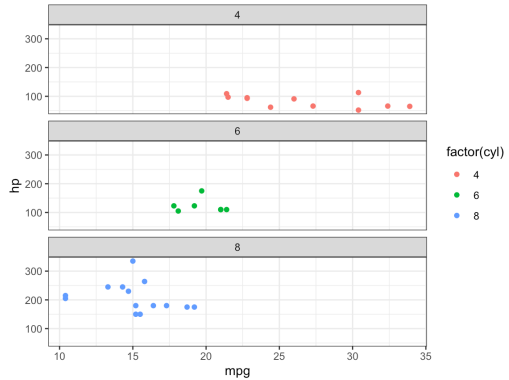
- We can change the layout of the “facets” **locally:**

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_point() +
    facet_wrap(~ cyl, ncol = 1)
```



- Which in this instance does the same thing as this:

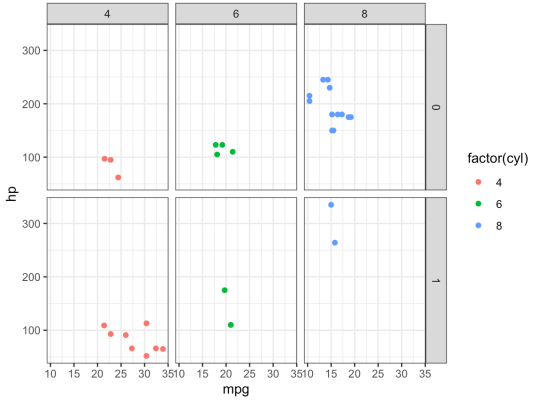
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_point() +
    facet_wrap(~ cyl, dir = “v”) #vertical
```



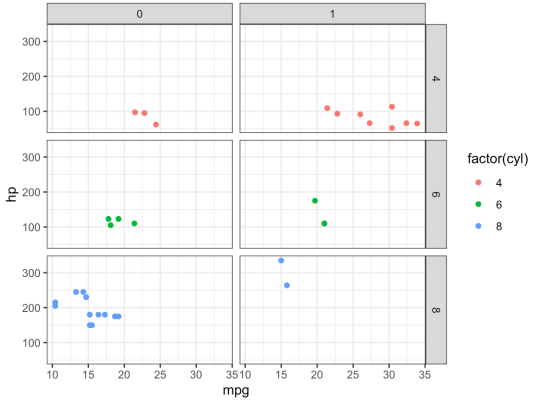
Facets

- We can even combine 2 variables with `facet_grid()`

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_point() +
    facet_grid(am ~ cyl)
```

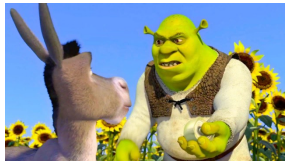



```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_point() +
    facet_grid(cyl ~ am)
```

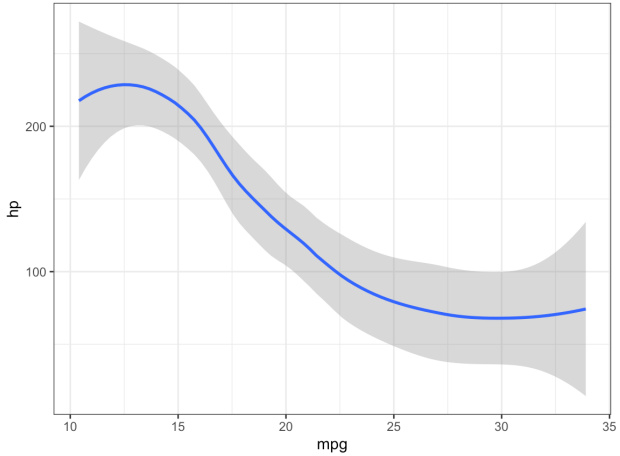


Other Geom Objects

- So far we've only used the basic `geom_point()`, but there are a lot more than that!
- `geom_smooth()` draws a smoothed line based on the trend of the provided data
- You can use multiple geoms on the same plot (ggplots have **layers**!)



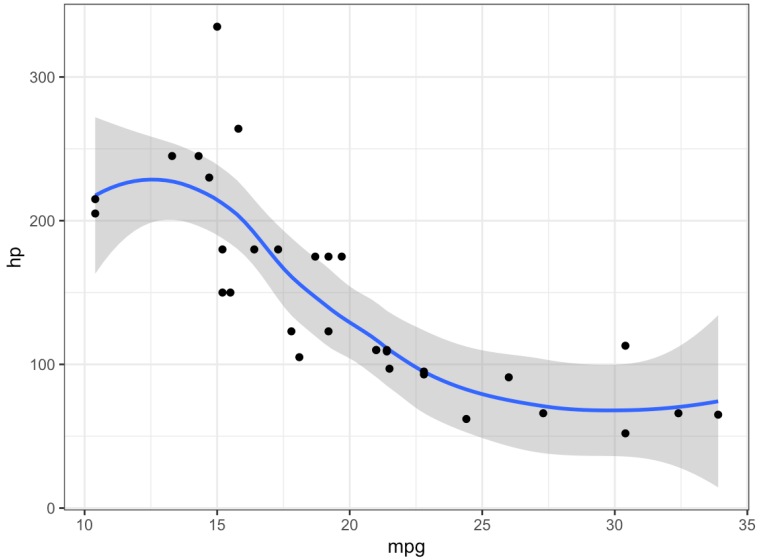
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp)) +
    geom_smooth()
```



Other Geom Objects

- Combining two **LAYERS**

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp)) +
    geom_smooth() +
    geom_point()
```

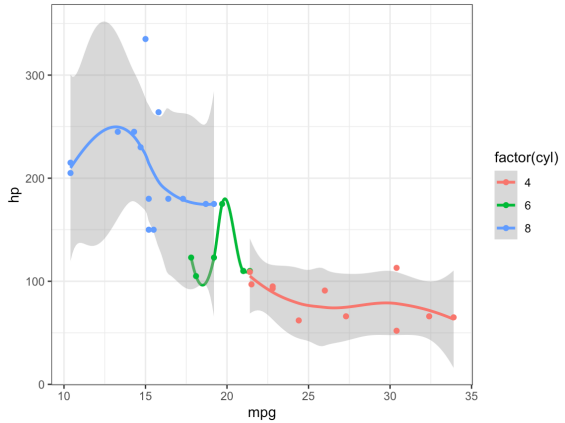


- Often the order of the layers matters, fortunately not in this case (if you swapped the geoms around it'll be the same)

Other Geom Objects

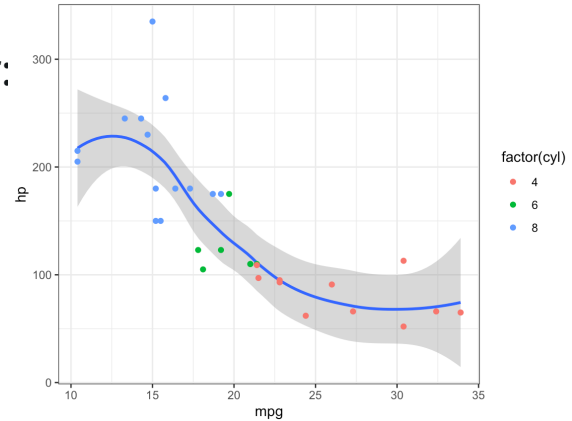
- Using the **full** data:

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_smooth() +
    geom_point()
```



- **Locally** mapped makes it look a lot better:

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp,
                  colour = factor(cyl))) +
    geom_smooth() +
    geom_point()
```



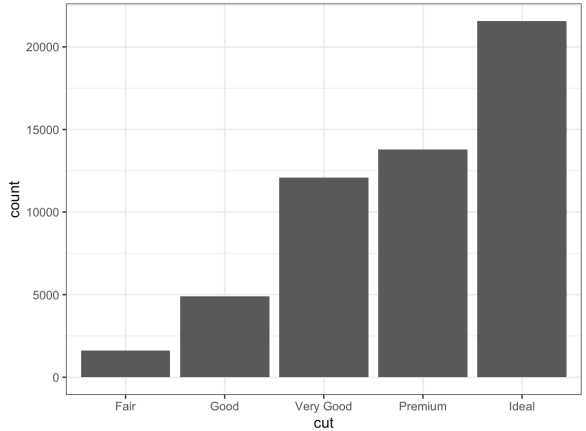
Statistical Transformations

- Some plots transform your data internally and plot those new values instead of raw values
- The `stat` argument of different plot types (geom functions) specifies the statistical transformation
- For example, `geom_bar()` uses `stat = "count"` as it's default to create counts of the mapped variable (as in what a bar chart does):

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

- Worth noting, geom and stats are often interchangeable

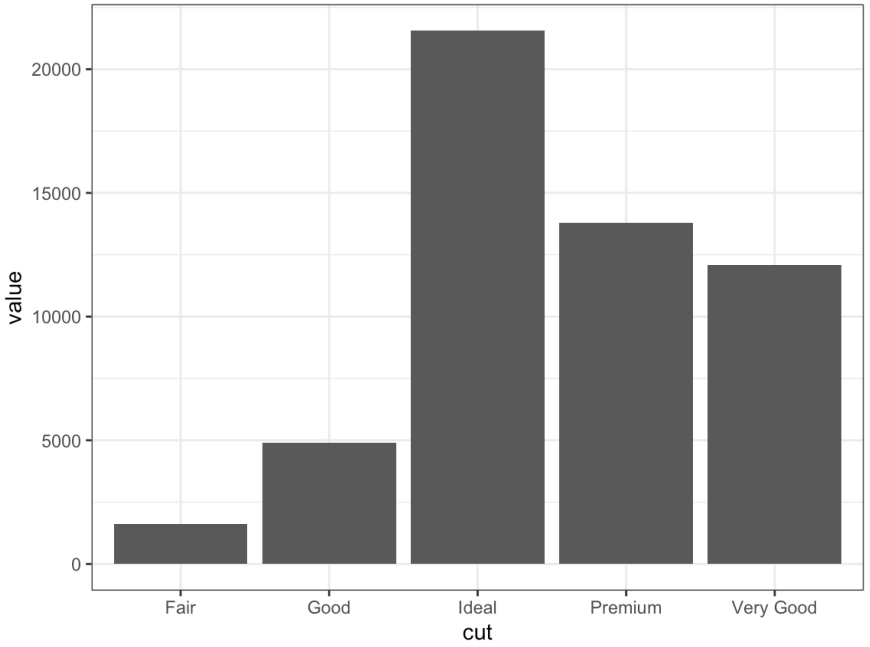
```
ggplot(data = diamonds) +
    stat_count(mapping = aes(x = cut))
```



Statistical Transformations

- Another commonly used **stat** is "**identity**" when plotting bars with heights based on raw values

```
## example tibble
demo <- tribble(
~cut,              ~value,
"Fair",            1610,
"Good",            4906,
"Very Good",       12082,
"Premium",         13791,
 "Ideal",          21551 )

ggplot(data = demo) +
    geom_bar(mapping = aes(x = cut,
                           y = value),
             stat = "identity")
```



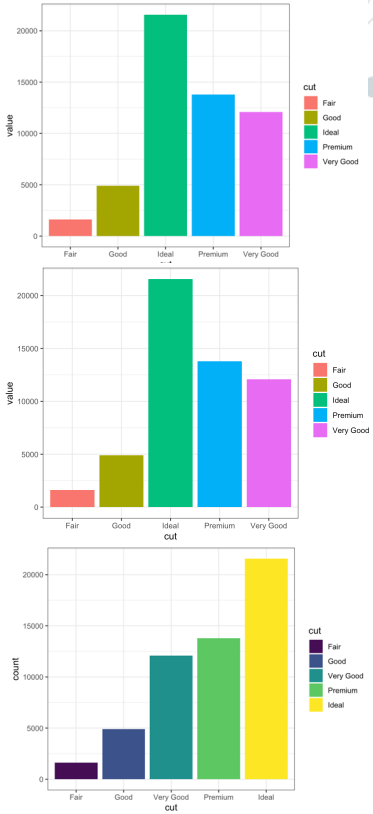
Statistical Transformations

- Another common aesthetic is the "**fill**" argument:

```
ggplot(data = demo) +
    geom_bar(mapping = aes(x = cut,
                           y = value,
                           fill = cut),
             stat = "identity")
```

```
ggplot(data = demo) +
    geom_bar(mapping = aes(x = cut,
                           y = value,
                           fill = cut),
             stat = "identity")
```

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

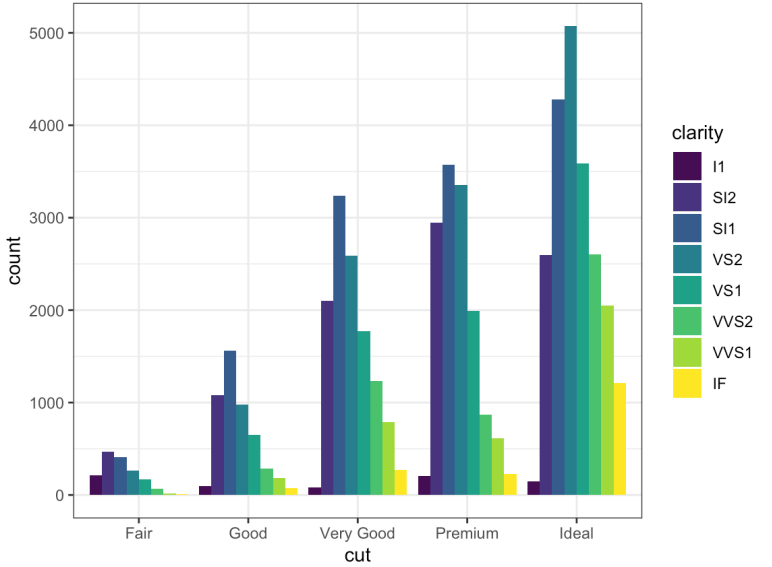



BREAK!

Positional Adjustments

- This is how we can arrange geoms that would otherwise occupy the same space
- You can do this by setting a value for the `position` argument of geom

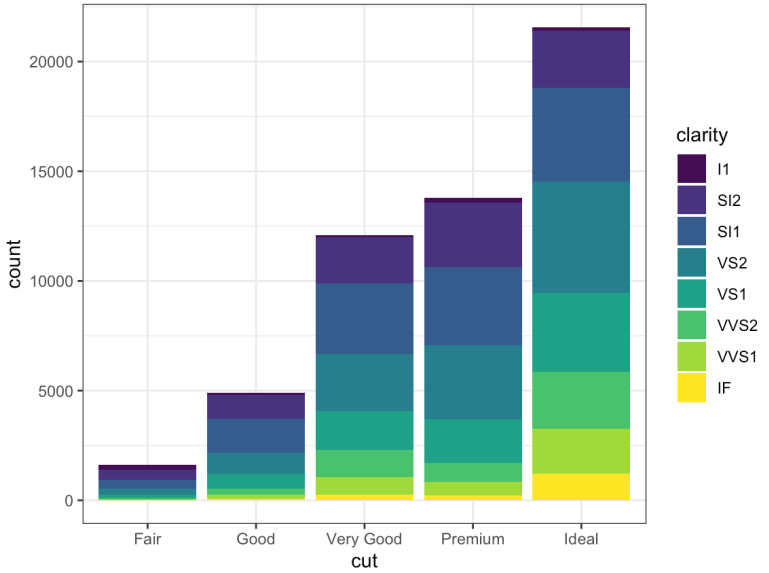
```
# side by side
ggplot(data = diamonds) +
    geom_bar(mapping = aes(x = cut,
                           fill = clarity),
        position = "dodge")
```



Positional Adjustments

- The `position` argument "`stack`" stacks them on top of each other

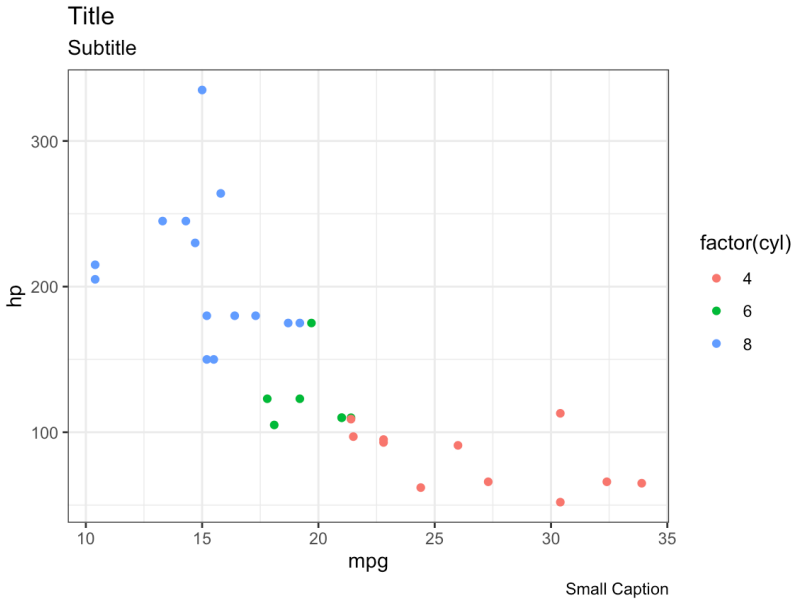
```
# stack elements on top of each other
ggplot(data = diamonds) +
    geom_bar(mapping = aes(x = cut,
                           fill = clarity),
             position = "stack")
```



Labels

- Another editable part of a plot are the text labels, which we can add or modify using `labs()`

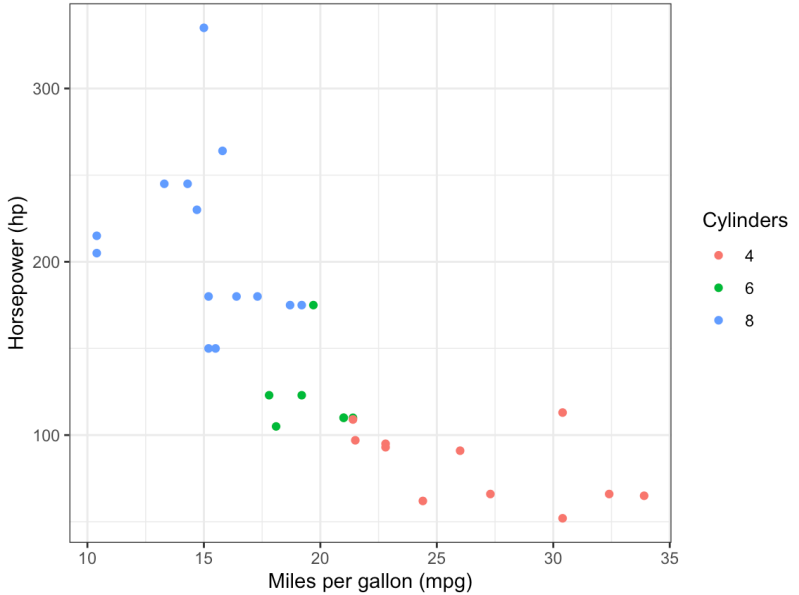
```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    labs(
         title = "Title",
         subtitle = "Subtitle",
         caption = "Small Caption”
        )
```



Labels

- `labs()` also lets you modify axis and legend labels

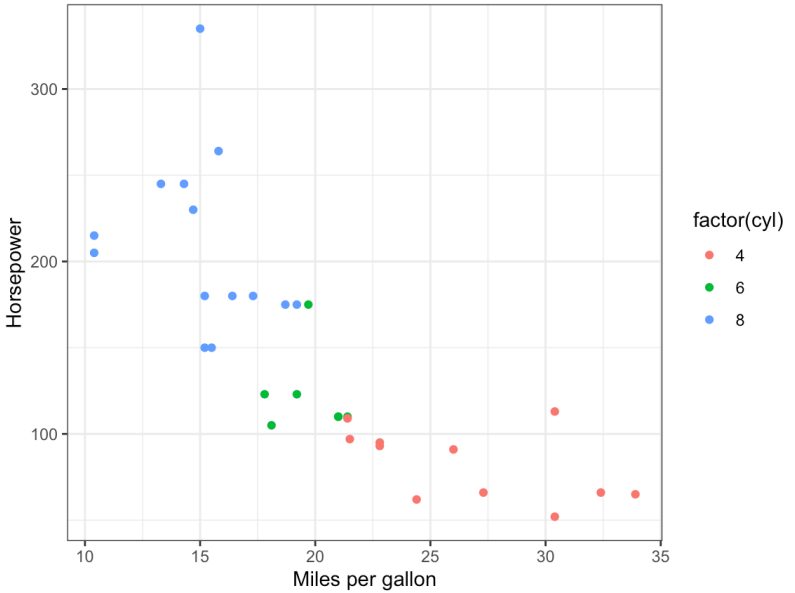
```
ggplot(data = mtcars,
       mapping = aes(x = mpg,
                     y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    labs(
         x = "Miles per gallon (mpg)",
         y = "Horsepower (hp)",
         colour = "Cylinders”
         )
```



Labels

- Sometimes a useful short-cut can be to use `xlabs()` or `ylabs()` to specifically define each of the axes

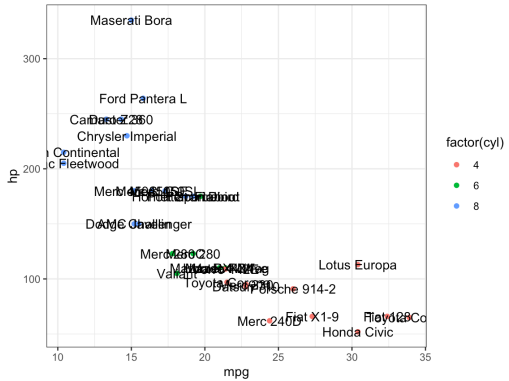
```
ggplot(data = mtcars,
       mapping = aes(x = mpg,
                     y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    xlab("Miles per gallon") +
    ylab("Horsepower")
```



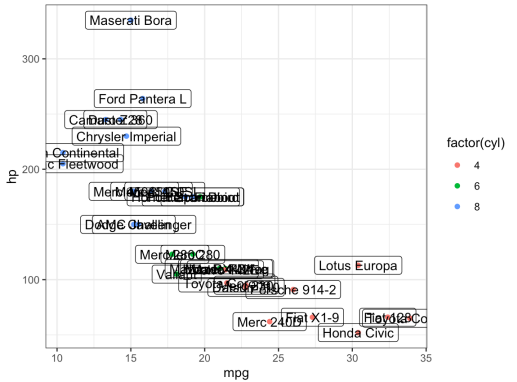
Annotations

- To add annotation to data points, we can use `geom_text()` and `geom_label()`

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    geom_text(aes(label = rownames(mtcars)))
```



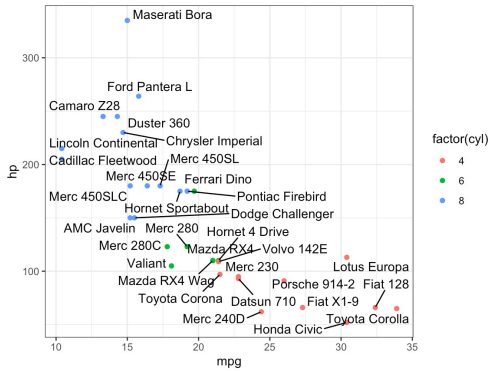

```
ggplot(data = mtcars,
    mapping = aes(x = mpg,
                  y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    geom_label(aes(label = rownames(mtcars)),
                   alpha = 0)
```



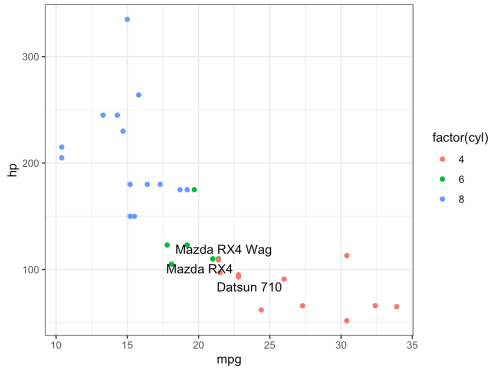
Annotations

- These look a bit messy though, the package **ggrepel** helps to avoid overlapping within plots

```
# install.packages("ggrepel")
ggplot(data = mtcars,mapping = aes(x = mpg,
                                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    ggrepel::geom_text_repel(aes(
        label = rownames(mtcars)),
        max.overlaps = 100)
```




```
ggplot(data = mtcars,mapping = aes(x = mpg,
                                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    ggrepel::geom_text_repel(aes(label =
                  rownames(mtcars[1:3, ])),
                    data = mtcars[1:3, ])
```

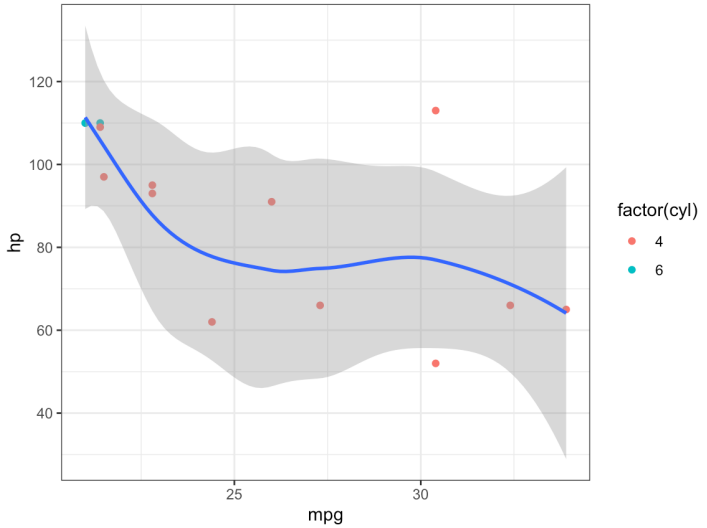


Zooming

- To control the plot limits, you have 3 methods:
  - Adjusting the data that's plotted
  - Setting `xlim` and `ylim` in `coord_cartesian()` (do this!!)
  - Setting the limits in each scale

```
mtcars %>%
    filter(mpg >= 20, hp <= 150) %>%
    ggplot(mapping = aes(x = mpg, y = hp)) +
        geom_point(aes(color = factor(cyl))) +
        geom_smooth()
```

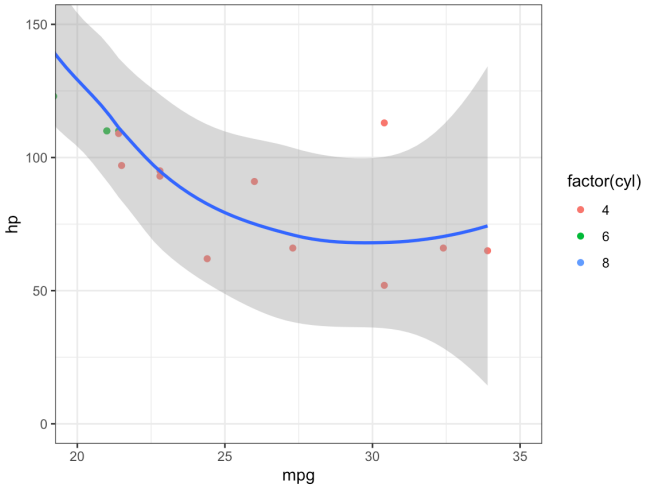
This is adjusting the data



Zooming

```
ggplot(data = mtcars,
       mapping = aes(x = mpg,
                     y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    geom_smooth() +
    coord_cartesian(xlim = c(20, 35),
                    ylim = c(0, 150))
```

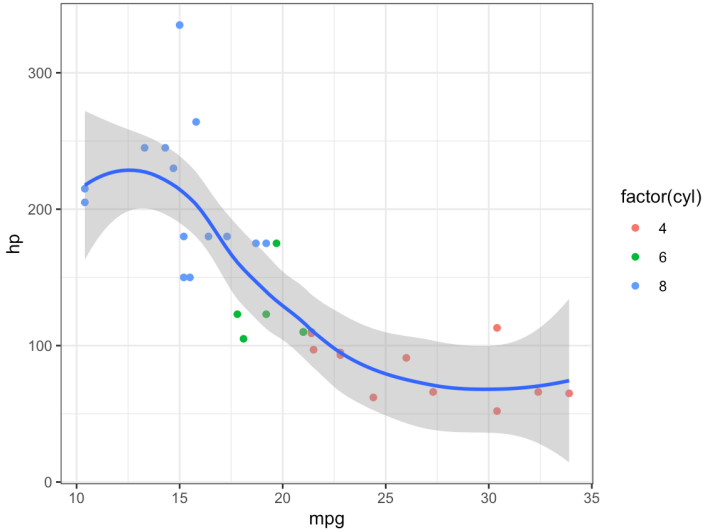
This is the **RIGHT WAY** ☺, using `coord_cartesian()`



Zooming

```
ggplot(data = mtcars,
       mapping = aes(x = mpg,
                     y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    geom_smooth()
```

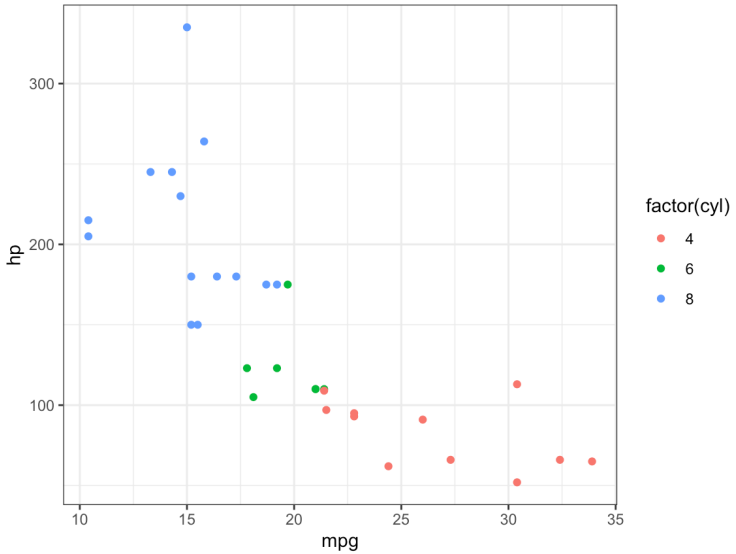
And this is setting the limits in each scale



Scales

- "`scale`" allows you control mapping things like colour, size and shape to data values
- "`scale`" draws a legend or axes
- ggplot2 automatically adds default scales behind the scenes

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl)))
```

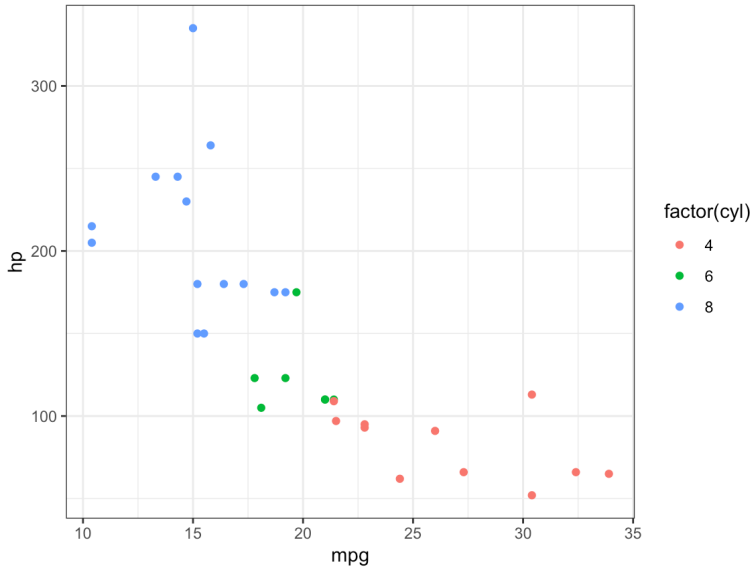


Scales

- Is the same as:

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_colour_discrete()
```

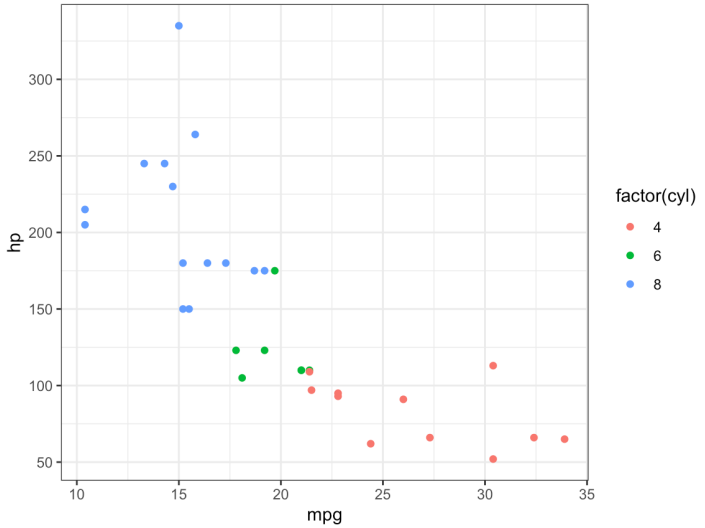
- Because we haven't set them
  - ggplot has given them default values



Scales

- Have a look at the naming scheme:
  1. ‘`scale_`’
  2. The name of the aesthetic (`x_`, `y_`, `colour_`, `fill_`…)
  3. The name of the scale (`continuous`, `discrete`)
- Changes to axes with arguments breaks and labels

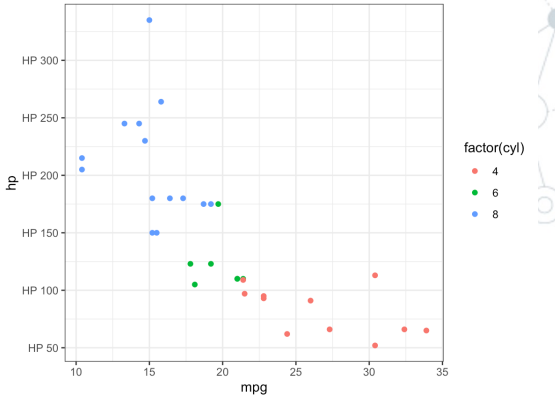
```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_y_continuous(
        breaks = seq(0, 350, by = 50))
```



- Breaks

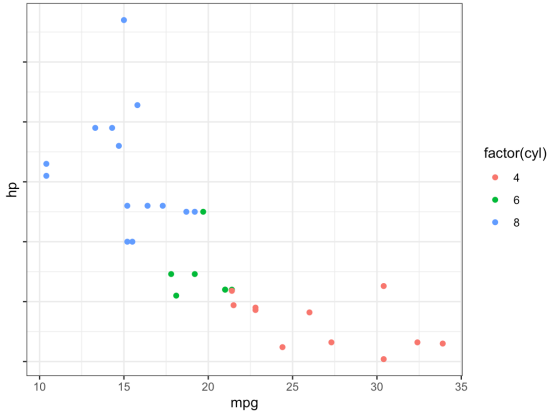
Scales

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_y_continuous(
        breaks = seq(0, 350, by = 50),
        labels = paste0("HP ", seq(0, 350, by = 50)))
```



- Labels!

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_y_continuous(
        breaks = seq(0, 350, by = 50),
        labels = NULL)
```

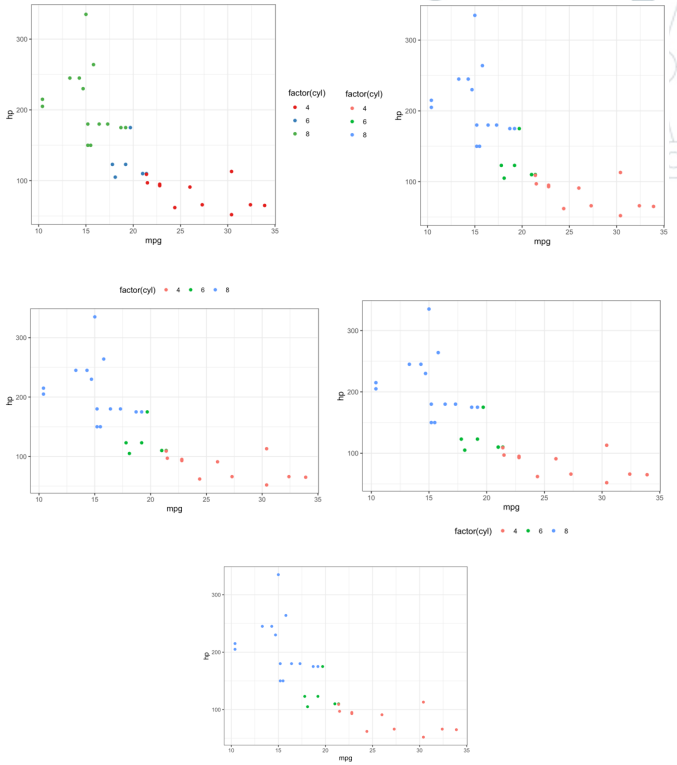


- Or no labels X

Legend Layout

- We can change a lot about the layout of the figure legend, such as the `legend.positon` argument

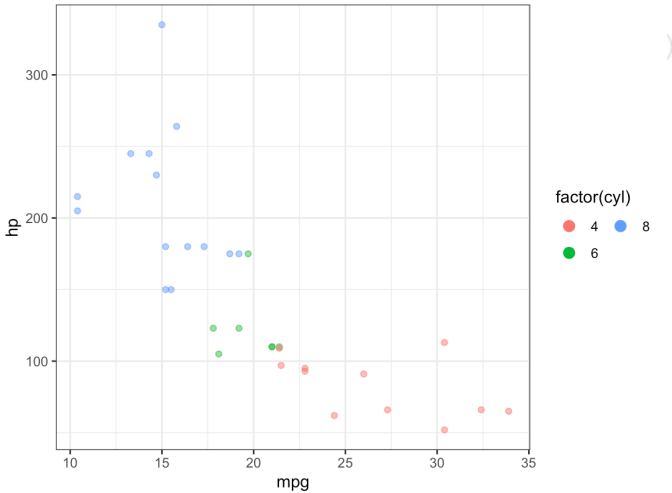
```
p <- ggplot(mtcars, aes(x = mpg,
                        y = hp)) +
    geom_point(aes(color = factor(cyl)))

p + theme(legend.position = "right") # the default

p + theme(legend.position = "left")

p + theme(legend.position = "top")

p + theme(legend.position = "bottom")

p + theme(legend.position = "none") # no legend
```



Legend Layout

- We can use the `guides()` function to control the legend display

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(colour = factor(cyl)),
                   alpha = 0.5) +
    guides(colour = guide_legend(ncol = 2,
                                 override.aes =
                       list(size = 3, alpha = 1)))
```

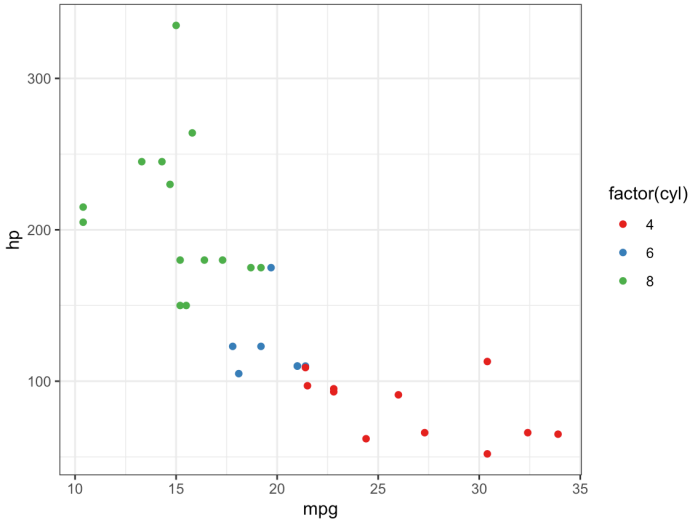


Replacing the Colour Scale

- The default ggplot2 colours we get are a bit rubbish
- Many pre-defined colour palettes are available to change that
- Such as from the RColorBrewer package:

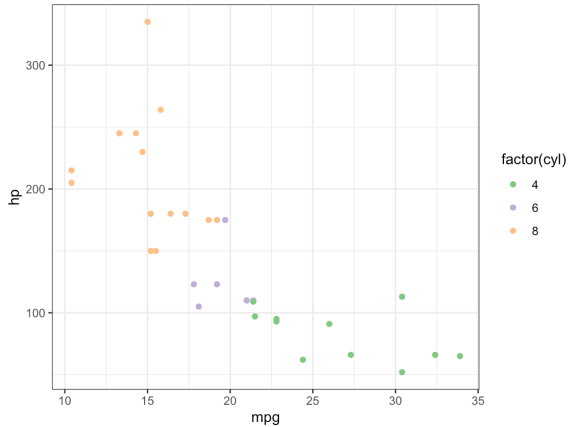
https://www.r-graph-gallery.com/38-rcolorbrewers-palettes.html

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
   geom_point(aes(color = factor(cyl))) +
   scale_colour_brewer(palette = "Set1")
```



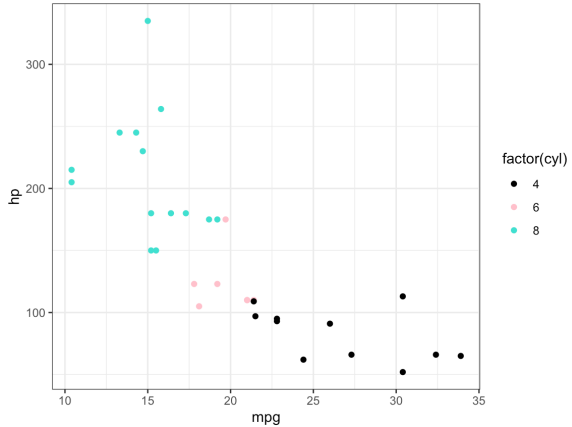
Replacing the Colour Scale

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_colour_brewer(palette = "Accent")
```



- Setting the colours manually

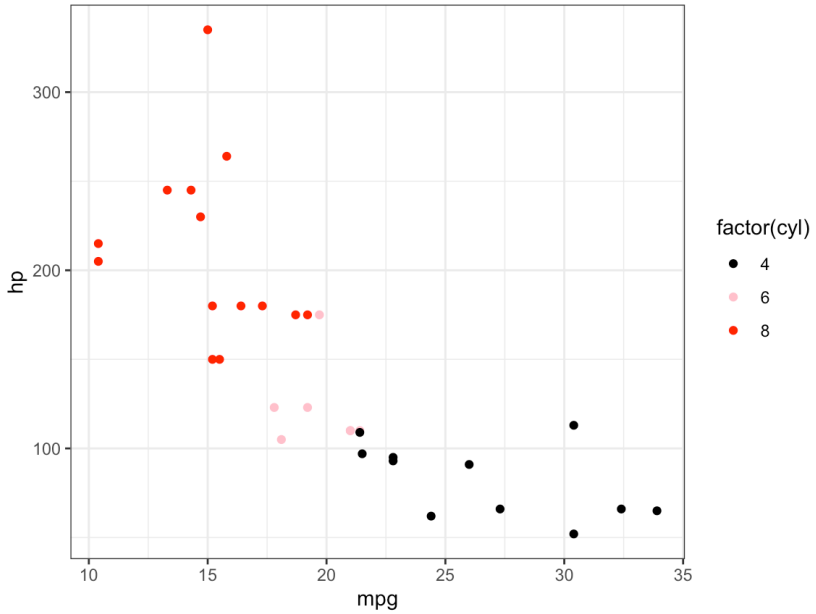
```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_colour_manual(values =
        c("black", "pink", "turquoise"))
```



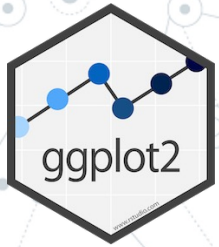
Replacing the Colour Scale

- Setting the colours manually but for specific data groups

```
ggplot(mtcars, aes(x = mpg,
                   y = hp)) +
    geom_point(aes(color = factor(cyl))) +
    scale_colour_manual(values =
        c(`4` = "black",
          `6` = "pink",
          `8` = "red"))
```



Saving a Plot



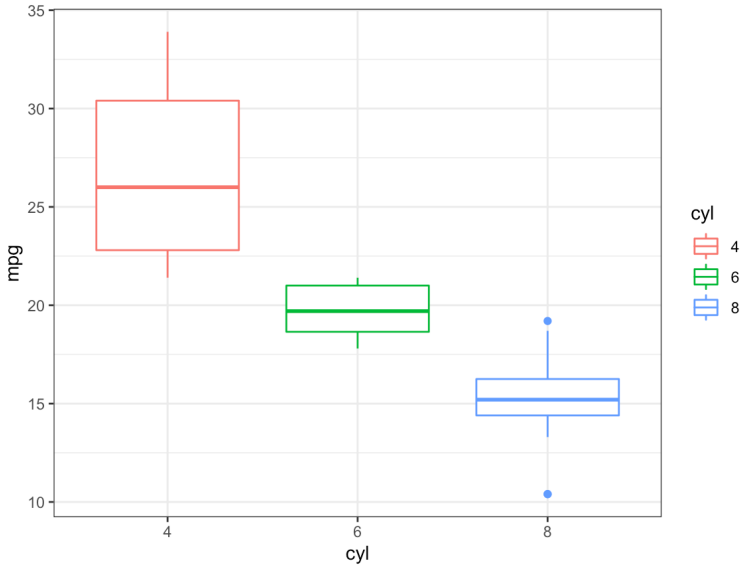
- We of course want to be able to save the beautiful plots we make! We can do this using `ggsave()`
- If not specified, it will save the most recent plot we create to our disk
- The format of the plot is defined in the filename extension (.pdf or .png for example)
- A good habit will be to work with R markdown files and save them to HTML outputs (like me and Frank!)

```
ggsave(filename = "my_plot_1.pdf")

ggsave(filename = "my_plot_2.png")
```

Boxplots and Violin Plots

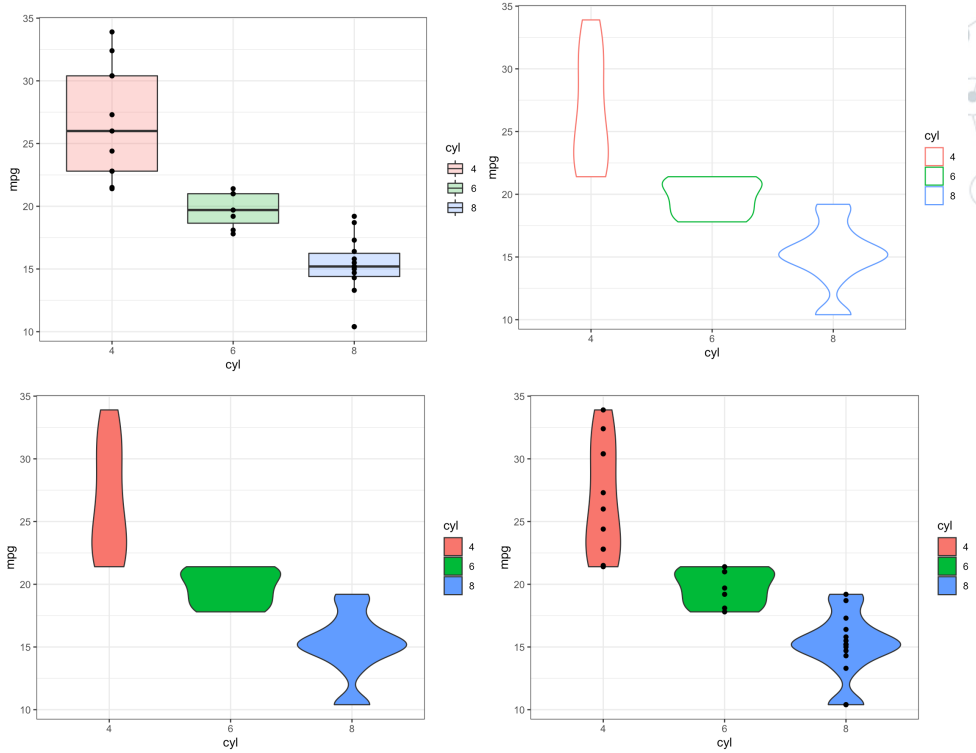
- Boxplots and Violin Plots are very common within biolosciences (protein levels, patient data, SNP frequency etc.)
- Have a go at creating your own boxplots and violin plots using the mtcars/diamonds/other datasets!

```
mtcars$cyl <- as.factor(mtcars$cyl)

p <- ggplot(mtcars, aes(cyl, mpg))
p + geom_boxplot(aes(colour = cyl))
```



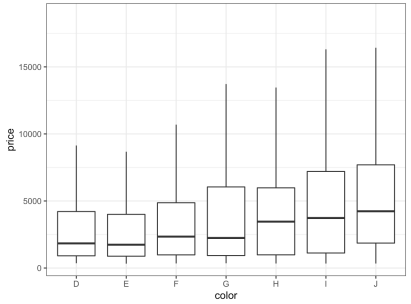
Boxplots and Violin Plots

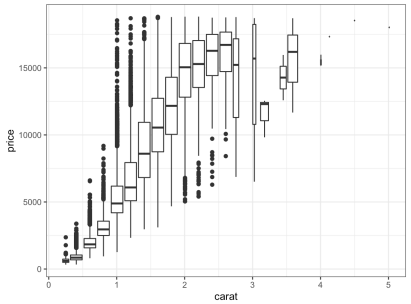
```
p + geom_boxplot(aes(fill = cyl), alpha = 0.3) +
    geom_point()

p + geom_violin(aes(colour = cyl))

p + geom_violin(aes(fill = cyl))

p + geom_violin(aes(fill = cyl)) + geom_point()
```

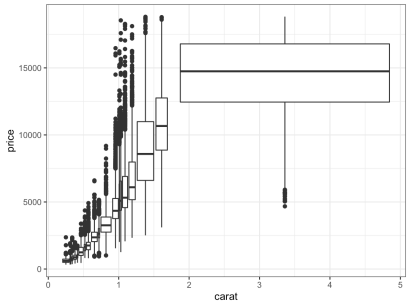


Boxplots and Violin Plots

```
ggplot(data = diamonds,
    mapping = aes(x = color,
                  y = price)) +
    geom_boxplot(outlier.size = -1)

# discretise numeric data into categorical
ggplot(diamonds, aes(x = carat,
                     y = price)) +
    geom_boxplot(aes(group = cut_width(carat, 0.2)))

ggplot(diamonds, aes(x = carat,
                     y = price)) +
    geom_boxplot(aes(group = cut_number(carat, 20)))
```



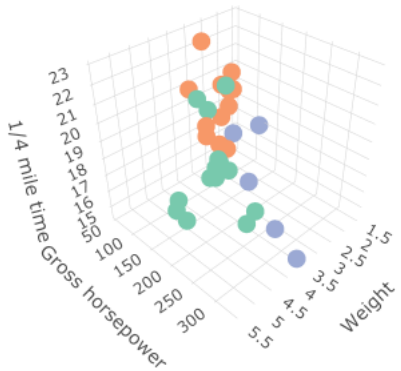




3D Plots

- Plotly is a great package for creating interactive 3D plots for web graphics

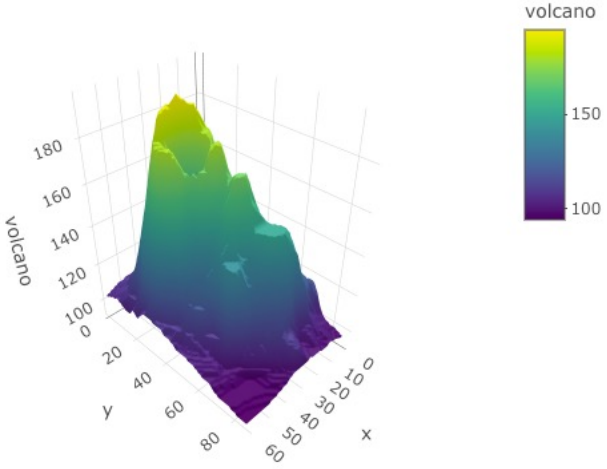
```
# install.packages("plotly")
library(plotly)

mtcars$gear <- as.factor(mtcars$gear)

plot_ly(mtcars, x = ~wt,
                y = ~hp,
                z = ~qsec,
                color = ~gear) %>%
    add_markers() %>%
    layout(scene = list(
                    xaxis = list(
                            title = 'Weight'),
                    yaxis = list(
                            title = 'Gross horsepower'),
                    zaxis = list(
            title = '1/4 mile time')))
```



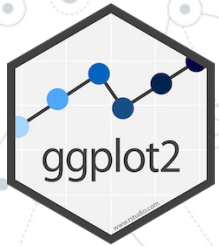
3D Plots

- You can even create a volcano!

```
# volcano is a numeric matrix that ships with R
plot_ly(z = ~volcano) %>%
    add_surface()
```



Some Useful Links and Resources



- Will you remember all of this? Almost certainly not!
- We're bioinformaticians, use the cheat sheets!

ggplot2 cheat sheet available from: http://rstudio.com/resources/cheatsheets

https://ggplot2-book.org

https://exts.ggplot2.tidyverse.org/gallery/

https://www.r-graph-gallery.com

https://github.com/jrnold/ggthemes

Visualisation in R

Sam Keat

PhD Student - UK Dementia Research Institute, Cardiff

KeatS@cardiff.ac.uk

UK Dementia Research Institute

CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD

Alzheimer's Society
United Against Dementia