Class 3: Customisation of ggplots

Andrew Parnell
`andrew.parnell@mu.ie`

PRESS RECORD
https://andrewcparnell.github.io/dataviz_course

Learning outcomes

- See most of the remaining customisation routines
- Learn how to fine-tune your ggplots
- Use a different data set

New data set: Horseshoe crabs

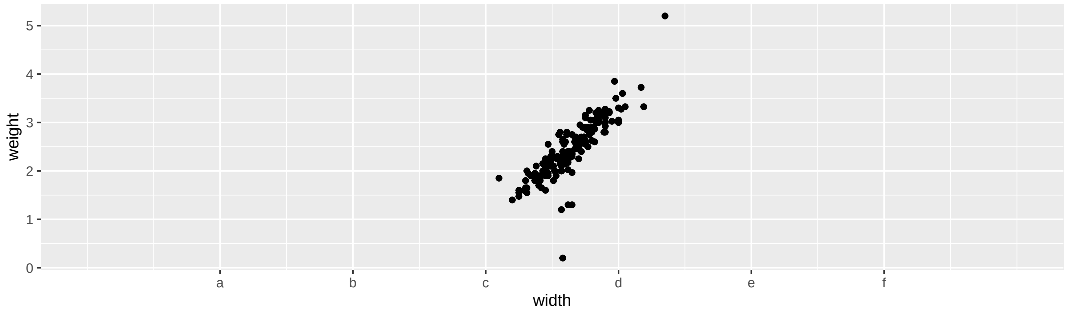
- Response is the number of satellites (other males residing nearby).
- Covariates are: female crab's colour (color; 1 = light medium, 2 = medium, 3 = dark medium, 4 = dark),
- spine condition (1 = both good, 2 = one worn or broken, 3 = both worn or broken),
- width (cm),
- weight (kg)

```
horseshoe <- readRDS("../data/horseshoe.rds")
horseshoe %>% glimpse
```

```
## Rows: 173
## Columns: 5
## $ color  <fct> medium, dark medium, light medium, dark medium, dark medi
## $ spine  <fct> both worn or broken, both worn or broken, both good, both
## $ width  <dbl> 28.3, 22.5, 26.0, 24.8, 26.0, 23.8, 26.5, 24.7, 23.7, 25
## $ satell <int> 8, 0, 9, 0, 4, 0, 0, 0, 0, 0, 0, 0, 11, 0, 14, 8, 1, 1, 
## $ weight <dbl> 3.05, 1.55, 2.30, 2.10, 2.60, 2.10, 2.35, 1.90, 1.95, 2.
```

Changing axis formats: breaks, limits and labels

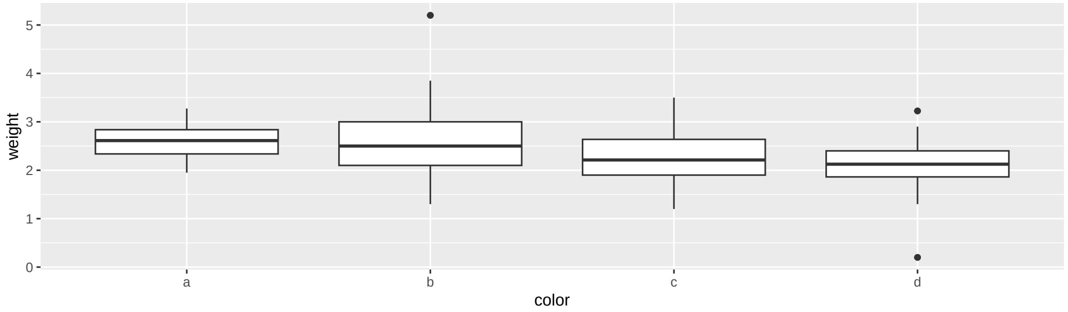
```r
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point() +
  scale_x_continuous(limits = c(-10, 60),
                     breaks = seq(0, 50, by = 10),
                     labels = letters[1:6])
```



Also works for y axis

Changing a discrete axis

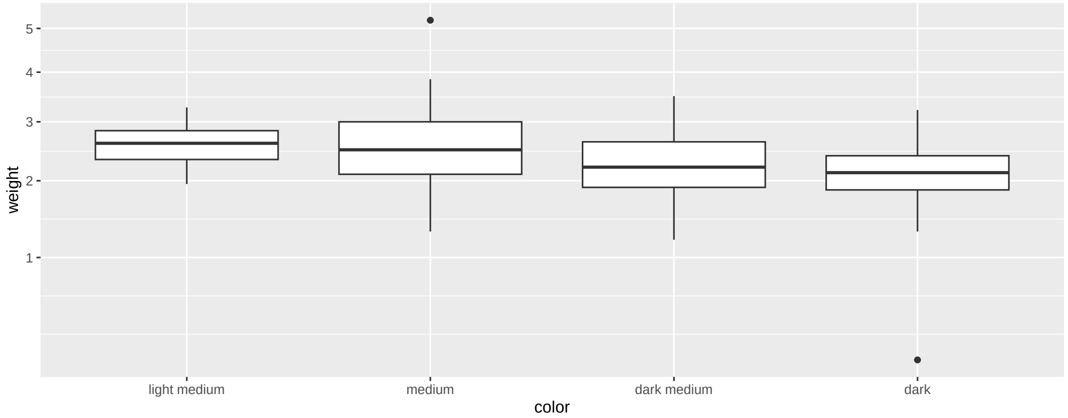
```
ggplot(horseshoe, aes(x = color, y = weight)) +
    geom_boxplot() +
  scale_x_discrete(labels = letters[1:4])
```



Can also set `limits` to remove some categories

Changing axis formats: transformations

```
ggplot(horseshoe, aes(x = color, y = weight)) +
    geom_boxplot() +
  scale_y_sqrt()
```



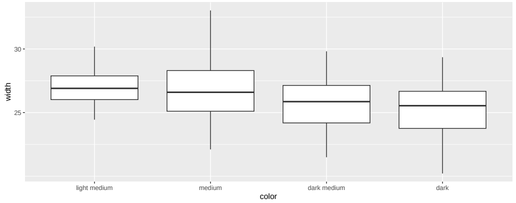
Also useful: `scale_x_reverse`, and also useful date/time transformations

Changing axis formats 2: advanced transformations

The scales package (in tidyverse) has more clever ways of labelling axes within scale_*_continuous

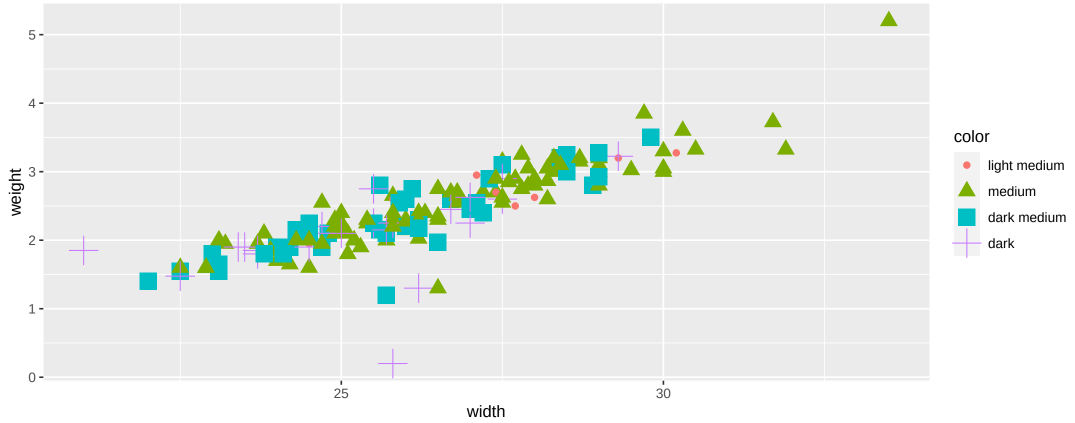
```
ggplot(horseshoe, aes(x = color,
                      y = width)) +
    geom_boxplot() +
  scale_y_continuous(
    trans = scales::log_trans(),
    breaks = scales::log_breaks()
  )
```

The scales package can also label axes better in e.g. financial or scientific measurement formats



Changing lines and points

```
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point(aes(colour = color, size = color, shape = color))
```

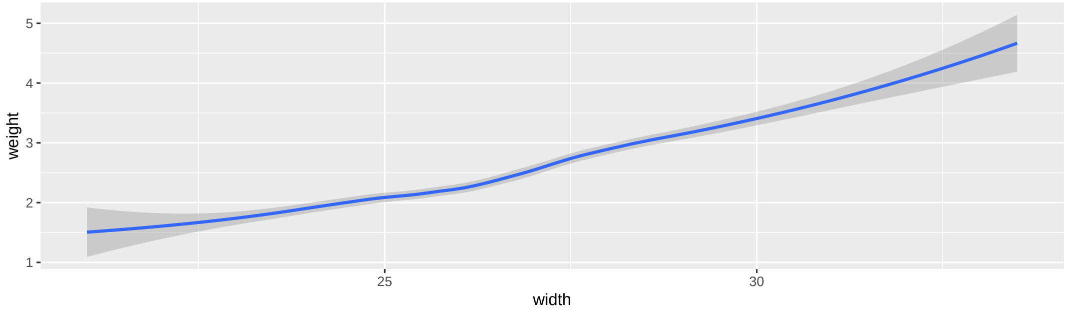


Also `linewidth` and `linetype` in `geom_line`

Adding smoothers and other geoms

There are some nice geoms to automatically add smoothes to data sets

```
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_smooth()
```



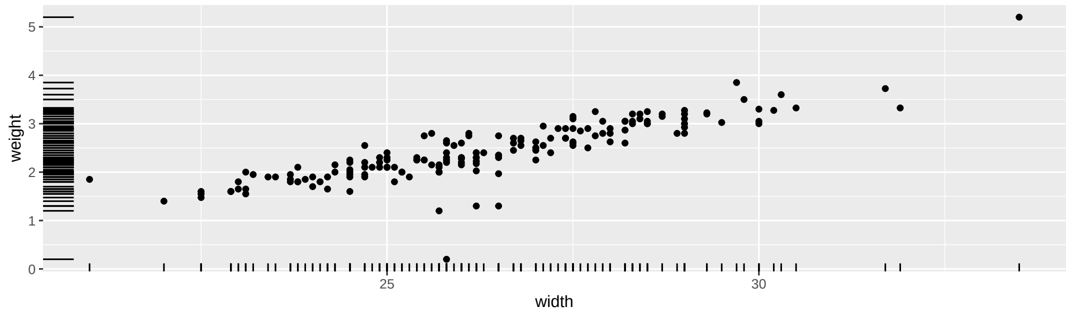
- Note that this does not add the original data points as there is no `geom_point()`

Notes on `geom_smooth`

- The default is to use a loess smooth if there are less than 1000 observation, otherwise a spline is used
- You can control the wiggliness of the loess with `span`
- You can use the `method` argument to specify another method such as `lm` or `glm`
- The default also includes a 95% confidence interval which you can remove with `se = FALSE` or change the level with `level = 0.75` for a 75% CI

Rug plots

```
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point() +
  geom_rug()
```

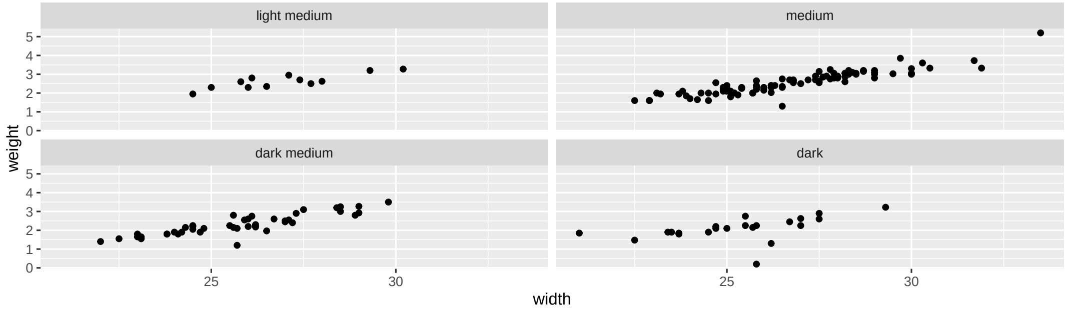


- Change which axes are used with the `sides = "trbl"` argument
- Use other arguments (`alpha`, `colour`, `linewidth`) to change appearance

Facets

When too many points start to appear on a plot, think about creating facets:

```
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point() +
  facet_wrap(vars(color))
```



```
  # (vars is just like aes, looks inside the data set)
```

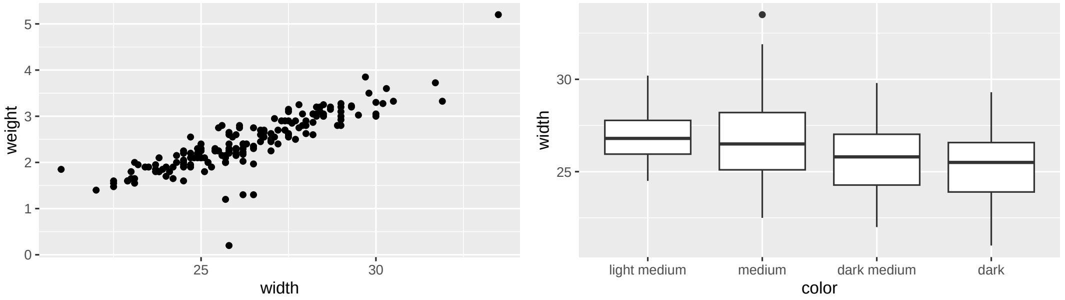
Notes on facets

- There are two common versions: `facet_wrap` which takes variables and wraps them round the screen in an efficient manner,
- ...and `facet_grid` which enables you to specify how the (potentially multiple) variables are used in the rows and columns of the facets
- Use the extra arguments `scales = "free_x"`, `scales = "free_y"`, and `scales = "free"` to unrestrict the axis limits

Multiple plots

Lots of ways to add disparate plots together; simplest is possibly `patchwork`

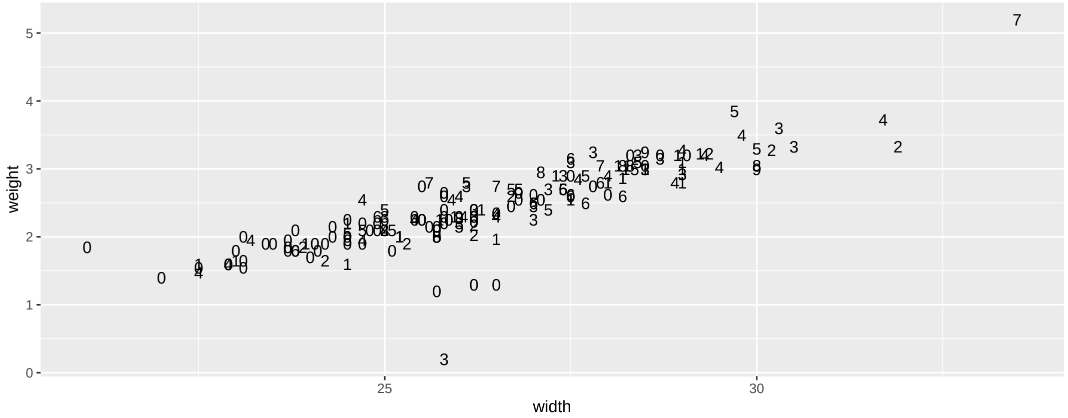
```
library(patchwork)
p1 <- ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point()
p2 <- ggplot(horseshoe, aes(x = color, y = width)) +
    geom_boxplot()
p1 + p2
```



Text labels and annotations

- `geom_text` labels points with their given `label`:

```
ggplot(horseshoe, aes(x = width, y = weight,
                      label = satell)) +
  geom_text()
```



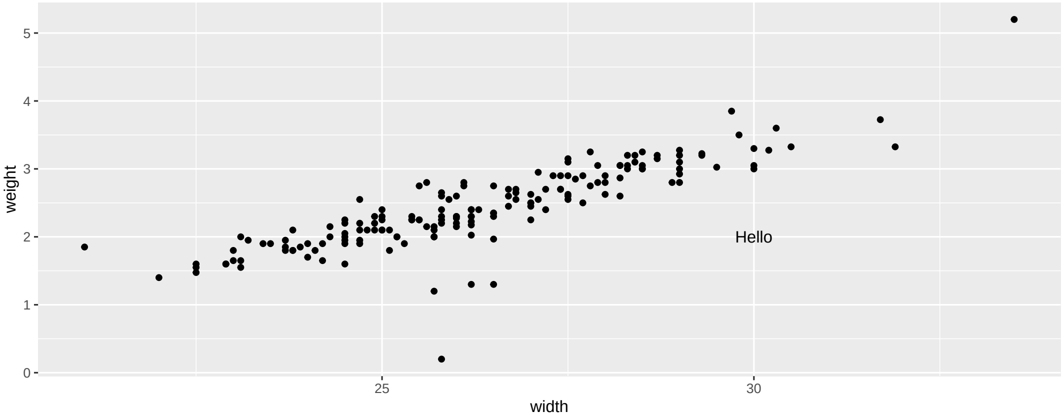
Notes on text labels

- Here the `label` is the count variable but it could be a text string
- You can add both points and labels, using the `vjust` and `hjust` to shift the labels out of the way of the points
- The `ggrepel` package rearranges the labels to that they don't overlap - very useful
- A related function is `geom_label` which draws a rectangle around the string

Annotation

To add a specific text label to a plot use `annotate`:

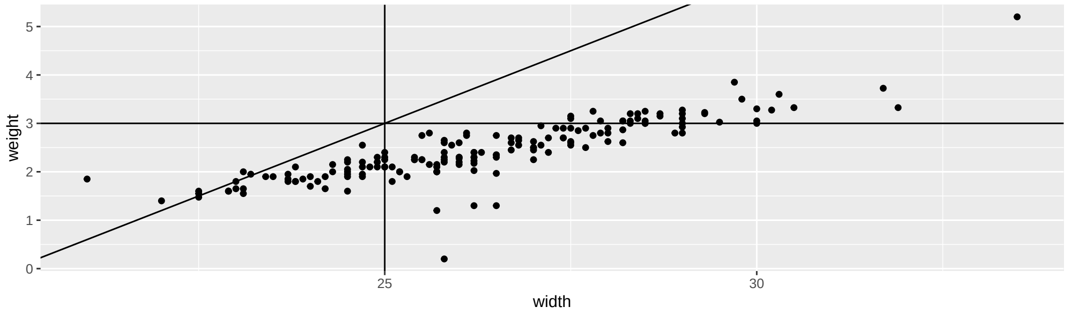
```
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point() +
  annotate("text", x = 30, y = 2, label = "Hello")
```



Adding straight lines

You can do this using `annotate`, or you can use:

```
ggplot(horseshoe, aes(x = width, y = weight)) +
  geom_point() +
  geom_hline(yintercept = 3) +
  geom_vline(xintercept = 25) +
  geom_abline(slope = 0.6, intercept = -12)
```

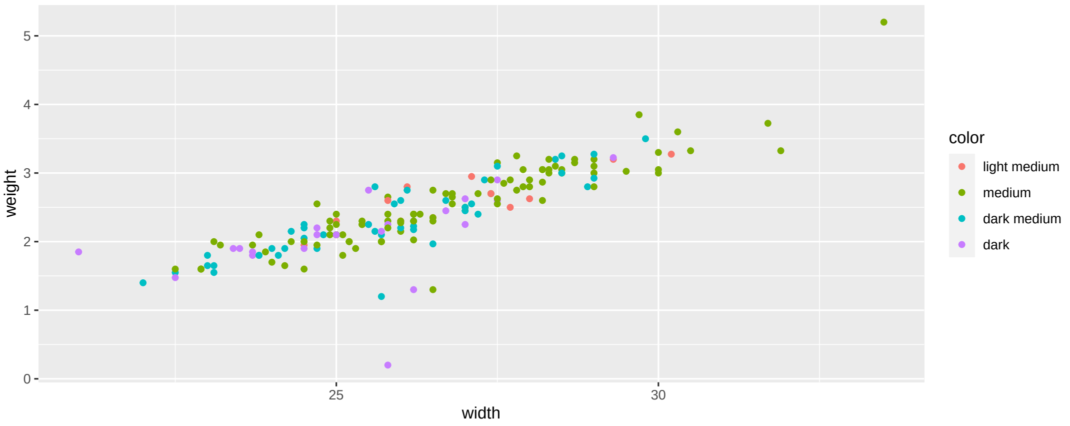


`geom_abline()` useful if you just want a line of identity

Specifying colour palettes

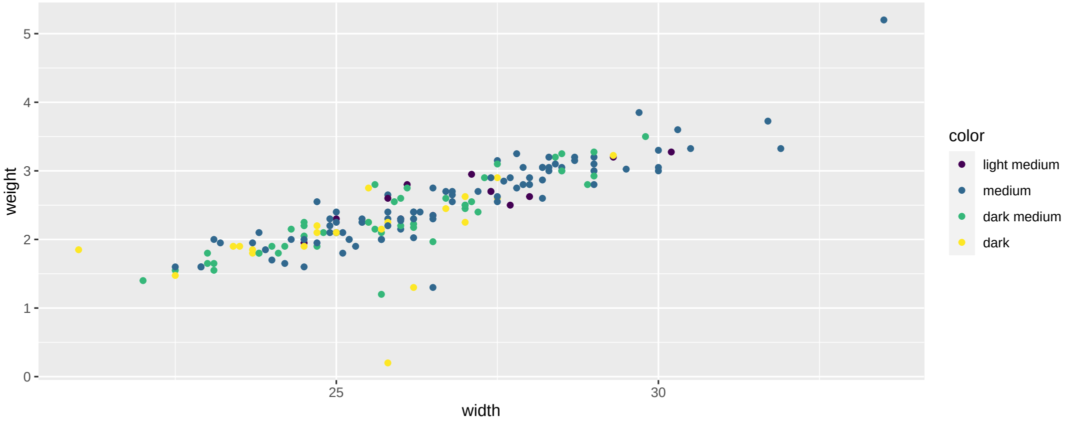
The default (discrete) palette:

```
ggplot(horseshoe, aes(x = width, y = weight,
                      colour = color)) +
  geom_point()
```

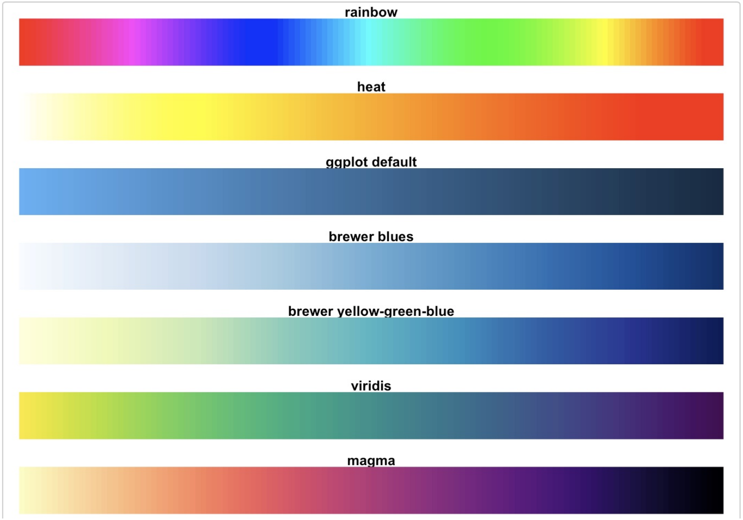


Changing the colour palette: viridis

```
ggplot(horseshoe, aes(x = width, y = weight,
                      colour = color)) +
  geom_point() +
  scale_colour_viridis_d()
```



Some example colour scales



From the viridis package documentation

Choosing a colour palette

In general use colour to separate groups, represent data, or highlight certain observations

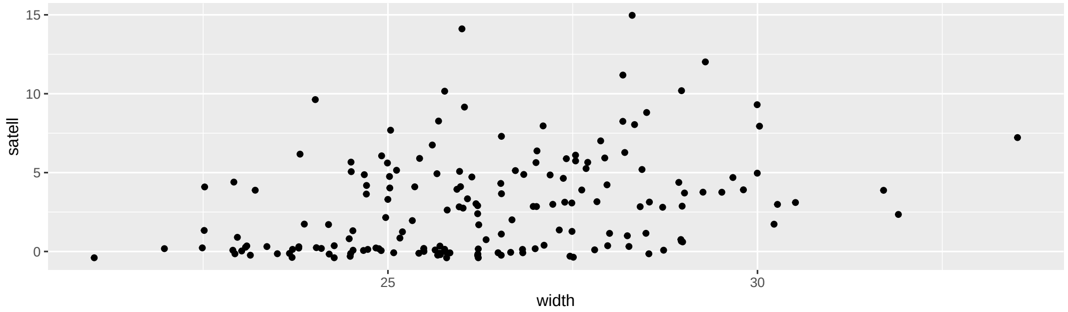
Choosing the colour palette is a bit of a minefield, but...

- The `viridis` options are popular because they look nice, print in black and white, and are easier for those with colour blindness
- You can specify invidual colours by name but make sure to put them inside the `aes` call
- Use `scale_fill_manual`, `scale_colour_manual` or `scale_fill gradient` to put in your own colour palettes (lots of examples of t)

Jittering

When points overlap, use jittering to separate them out (but make sure to tell people)

```
ggplot(horseshoe, aes(x = width, y = satell)) +
  geom_jitter()
```

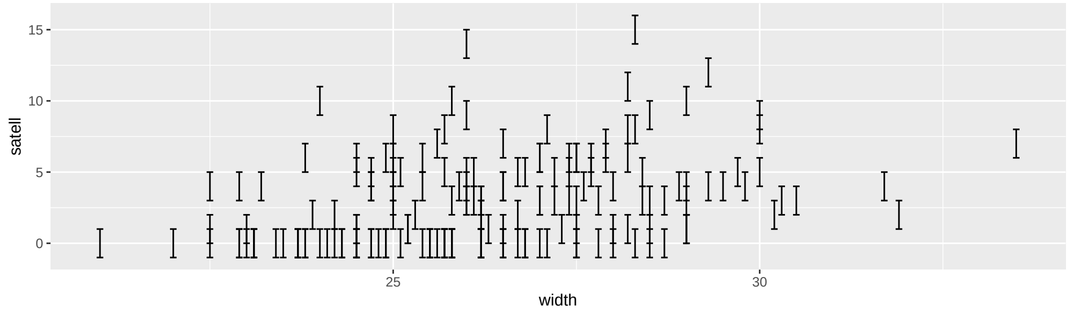


- Use `width` and `height` arguments to vary the amount of jittering

Plotting data with uncertainty

When the data has uncertainty use `geom_pointrange` (points with errors), `geom_linerange` (candlesticks), and `geom_errorbar` (line ranges with upper/lower horizontal)

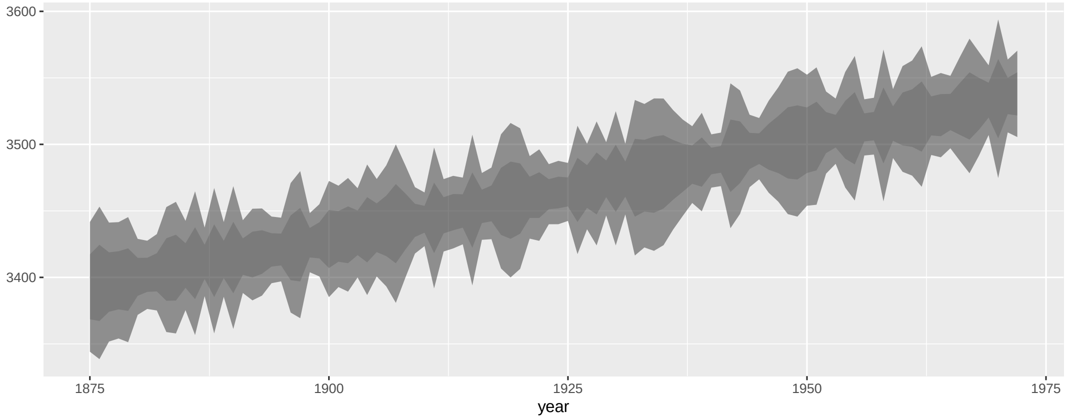
```r
ggplot(horseshoe, aes(x = width, y = satell)) +
  geom_errorbar(aes(ymin = satell - 1, ymax = satell + 1))
```



Plotting data with uncertainty part 2: ribbons

If a line plot has uncertainty use `geom_ribbon`

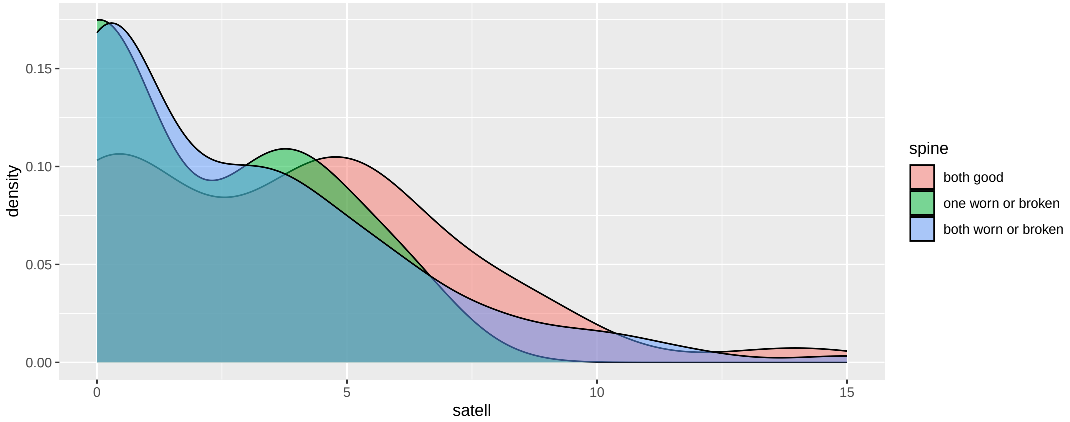
```
ggplot(huron, aes(year)) +
  geom_ribbon(aes(ymin=level - 2*se, ymax=level + 2*se), alpha = 0.5) +
  geom_ribbon(aes(ymin=level - 1*se, ymax=level + 1*se), alpha = 0.2)
```



Density plots

A nice alternative to histograms or boxplots (see also violin plots)

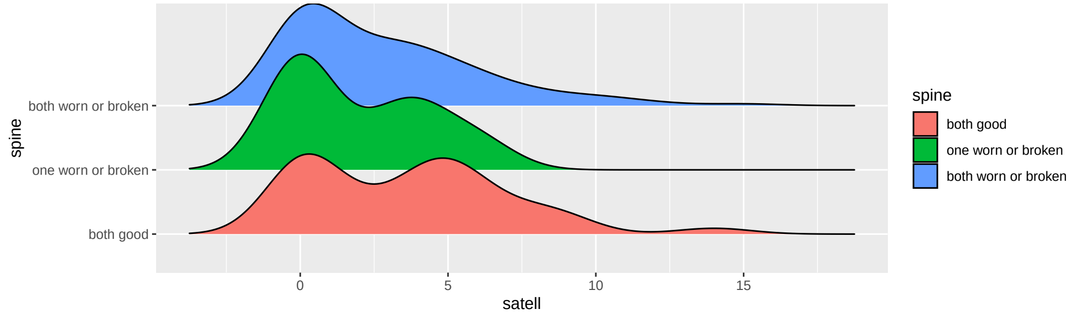
```
ggplot(horseshoe, aes(x = satell, group = spine,
                      fill = spine)) +
  geom_density(alpha = 0.5)
```



Ridge line plots

Use ggridges

```
library(ggridges)
ggplot(horseshoe, aes(x = satell, y = spine,
                      fill = spine)) +
  geom_density_ridges()
```



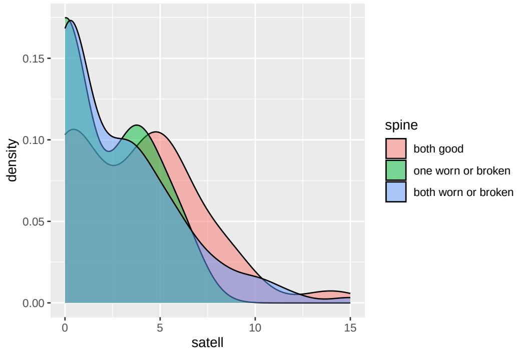
- See later slide for removing the unnecessary legend

Useful tools for changing coordinates

Some very useful options for changing coordinate systems:

- `coord_flip` flips the x and y axes
- `coord_polar` turns to polar coordinates
- `coord_equal` to make the axis differences equal
- Use instead `theme(aspect.ratio = 1)` to make the plot square

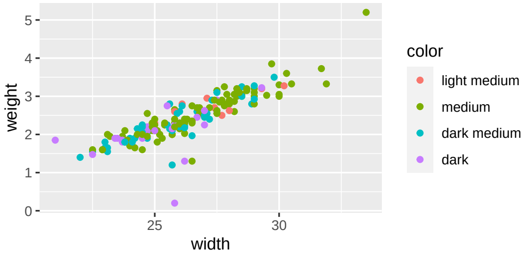
```r
ggplot(horseshoe,
       aes(x = satell,
           group = spine,
           fill = spine)) +
  geom_density(alpha = 0.5) +
  theme(aspect.ratio = 1)
```

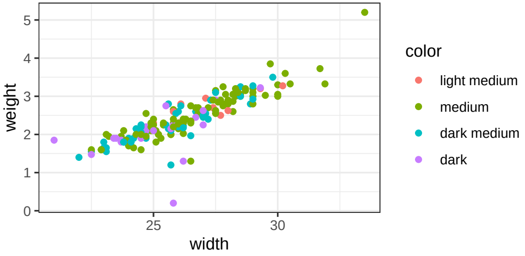


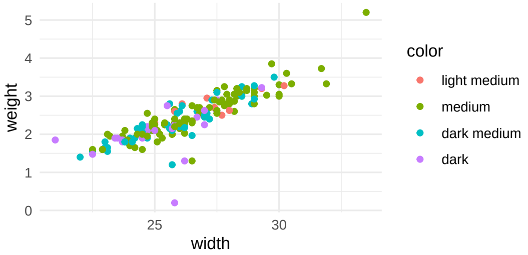
Themes

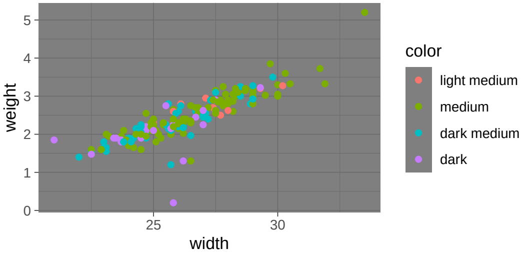
```r
p1 <- ggplot(horseshoe, aes(x = width, y = weight, colour = color)) +
  geom_point()
p2 <- ggplot(horseshoe, aes(x = width, y = weight, colour = color)) +
  geom_point() + theme_bw()
p3 <- ggplot(horseshoe, aes(x = width, y = weight, colour = color)) +
  geom_point() + theme_minimal()
p4 <- ggplot(horseshoe, aes(x = width, y = weight, colour = color)) +
  geom_point() + theme_dark()
p1 + p2 + p3 + p4 + plot_layout(ncol = 2)
```

Themes 2









See also `ggdark` for proper dark theme

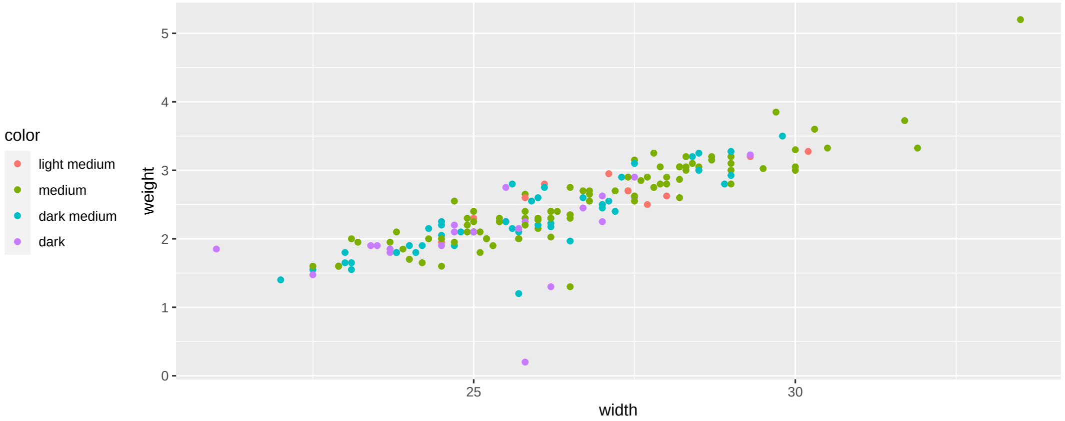
More on themes

- `theme` is much more than just changing the basic structure of the plot; you can define you own themes
- It also gives you control over all the elements including the title font, size, etc, and similarly for axis controls, background colours, etc
- Lots of these use a weird syntax starting `element_` which allow for arguments specifying the elements of the text (size, font, etc)

Adjusting legends

Often don't need the legend or want to remove it entirely

```
ggplot(horseshoe, aes(x = width, y = weight, colour = color)) +
  geom_point() +
  theme(legend.position = "left") # Also right, etc and none
```



Other bespoke plotting features

Most of the geoms we have met have their own individual features, for example:

- `geom_bar` has special functions for how to fill the bars, whether to stack them or not, widths of the bars
- `geom_boxplot` has many options to change how the boxes and whiskers are arranged, and how outliers are dealt with
- `geom_density` has controls on the bandwidth, which kernel is used, and

For all of these (and the others we have met) see the help files to fully understand

Exercise

Use the horseshoe data to draw the ugliest plot you can using the commands we have covered above! Submit it to Slack. Like other people's graphs to find the winner

Summary

- Nobody can remember all of these different options. Use Stack Overflow
- Nobody ever builds a detailed plot from scratch. Start from someone else's plot and update it
- Almost all of the functions shown above have multiple clever options that you might find useful - make sure to look at the help files of the functions you are using