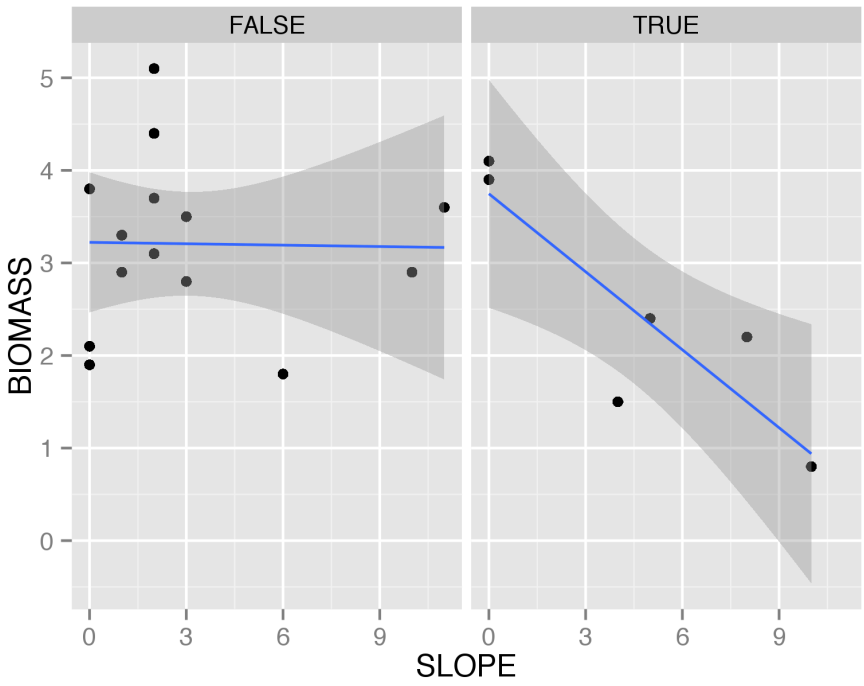
Advancing in R



Module 2: Data Visualisation

Introduction

In this practical, we will cover the basics of visualising data using the package {ggplot2} in R.

We will assume that you are familiar with the general syntax of coding and plotting using base graphics in R.

The ggplot2 lecture notes contain information on the grammar of graphics and on getting started that may be helpful as you work through and review the material below.

Learning outcomes

Know how to install and load R packages

Know how to cite R software and R packages

Know the basics of “the grammar of graphics”

Know how to build graphical outputs by layer using {ggplot2}

Some useful commands

Below is a list of some (but not all) of the useful commands you will probably need to run at some stage of today’s practical.

ggplot()
aes()
geom_point()
geom_boxplot()
geom_histogram()
stat_smooth()
stat_summary()
facet_grid()
facet_wrap()

Building a plot

Begin by loading the RStudio package. Start a new project and a new Rscript, and save each of these using meaningful and descriptive filenames. Recall the preamble to a well-annotated script: what information and commands should you include at the beginning of your script?

For this exercise, we shall use the 'dplyr' and 'ggplot2' packages. You should have installed dplyr during the previous module, so now install ggplot2:

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

Now that you have installed the required packages, load these into your project with the following commands:

```
library(dplyr)
library(ggplot2)
```

For this exercise, we will be using a data set on unicorn horn size that I have painstakingly collected. The data is saved as 'RData' format, so you can load it directly into RStudio and then query the structure:

```
load("Unicorns.RData")
str(UNICORNS)
```

Investigating geoms

1. Histograms

One of the first steps you should take is to investigate the distribution of your response variable (in this case, **HornLength**). Remember that a histogram is actually a transformation of your data: X is your response variable, and Y is created automatically as counts of 'bins'.

A histogram 'bins' the full range of values into a series of small intervals, then counts how many values fall into each interval. Are you happy with the size of the bin that has been chosen for you? Try modifying it with the 'binwidth' argument in your geom specification.

```
ggplot(UNICORNS, aes(x = HornLength))+
      geom_histogram(binwidth = 4)
```

Try a range of values until you find one that you are happy with. Changing some other arguments of your geom specification can help you to separate adjacent bins; for example, try changing the fill to white, and the colour to black.

Some people don't like the default theme, giving a grey background with grid lines. You can change the overall 'theme' of the plot, e.g. by appending '+ theme_bw()' or '+ theme_classic()' to your figure call. It is also possible to build your own custom themes for figures (but this is beyond the scope of this module).

2. Scatterplots

Scatterplots enable us to visualise the relationship between two continuous variables. In this case, try making a scatterplot of horn length (response variable) plotted against height (continuous predictor).

HINT: check the documentation for the commands 'ggplot' and 'geom_point'.

The points seem to be clustered along the x-axis, meaning that there is likely **overplotting** (data points plotted on top of others). We can reduce this by using the 'jitter' geometric object style instead, which adds small amounts of random noise to the data. Try the following:

```
ggplot(UNICORNS, aes(x = Height, y = HornLength))+
      geom_jitter(position = position_jitter(width = 0.02))
```

Try playing around with the jitter width, and see how this affects your plots.

At the risk of complicating our figure, let's try adding another predictor: this time, a categorical variable. The 'Magic' variable gives 7 different grouping levels of the data (i.e., each unicorn belongs to 1 of 7 distinct magic classes). Try adding this as a colour **aesthetic**. You can do that either within the default data command ('ggplot'), or within the geom specification ('geom_jitter'). This doesn't

make a difference here, but it will later on as we add statistical transformations – do you remember why?

(HINT: one changes the aesthetics of how data are mapped, and the other simply changes the aesthetics of a geometric object)

3. Boxplots

After the last scatterplot, it's likely that you want to see whether there is (visual) evidence of a relationship between horn length and magic class, irrespective of height. Boxplots are a way to graphically depict groups of numerical data. Create a boxplot to investigate the relationship between horn length and magic class. Is there a general trend here? If so, is it what you expected after your previous scatterplot?

There are quite a few outliers here, and the range for each magic class is quite large – you can use the 'jitter' geom to overlay the points for each boxplot to see how they are distributed:

```
ggplot(UNICORNS, aes(x = Magic, y = HornLength))+
      geom_boxplot() +
      geom_jitter(position = position_jitter(width = 0.1))
```

This becomes quite convoluted, and it's still a little hard to see how the density of points changes. We can change the 'alpha' aesthetic, which affects the transparency of points, enabling us to see variation in density of data points:

```
ggplot(UNICORNS, aes(x = Magic, y = HornLength))+
      geom_boxplot() +
      geom_jitter(position = position_jitter(width = 0.1),
                  alpha = 0.4)
```

Statistical transformations

1. Summarising data

The distribution of your histogram and the number of outliers in your boxplots might be concerning. Are the mean and median of your data likely to be similar? Add a mean point for each group to your boxplot using the 'stat_summary' function, changing the size and shape to differentiate from data points:

```
ggplot(UNICORNS, aes(x = Magic, y = HornLength))+
      geom_boxplot() +
      stat_summary(fun.y = "mean",
```

```
        geom = "point",
        shape = 5,
        size = 4)
```

'Stat_summary' gives you flexibility to summarise the data in different ways; you can also have several on a single plot. Try adding a minimum and maximum point for each group, with different shapes for each.

(HINT: try 'min' and 'max' instead of 'mean'…)

2. 'Smoothers'

Smoothers help you to see relationships in your data (but are no replacement for real statistical analysis!). You can use 'stat_smooth' to add a smoother to your scatterplot:

```
ggplot(UNICORNS, aes(x = Height, y = HornLength)) +
      geom_jitter(position = position_jitter(width = 0.02)) +
      stat_smooth(method = "lm")
```

The smoothing method here is a linear regression model (we'll cover these properly later on).

Add the 'Magic' variable in again as a colour aesthetic. Remember this could be specified within the aesthetics for the main 'ggplot' call or within 'geom_jitter' (this also needs to be in an 'aes' argument). Try **both** – how do they differ in terms of what the final figure looks like?

We probably do want to show a regression slope for each 'color' category, but this does make the figure quite busy to look at. How could we represent this more clearly?

Faceting ('small multiples')

Faceting in ggplot2 is achieved by adding a 'facet_grid' or 'facet_wrap' command, along with the variable(s) on which we wish to slice our data. You should select one based on how you want to lay out your panels:

- **facet_grid** lays out panels in a grid (remember the maps of USA laid out by ethnic/religious groups and income)
- **facet_wrap** allows the panels to 'wrap' across multiple lines (remember the maps of USA laid out by year).

1. Grid

Add a 'facet_grid' command to your scatterplot with 'lm' smoother, slicing your data by Magic.

Look at your figure: is there 'chartjunk'? What might you want to get rid of?

```
ggplot(UNICORNS, aes(x = Height, y = HornLength, colour = Magic)) +
      geom_jitter(position = position_jitter(width = 0.02)) +
      stat_smooth(method = "lm") +
      facet_grid(. ~ Magic)
```

An easy way to check for chartjunk is even to look at your specification: is each part of the figure giving you something new?

Try adding a vertical component to your facet grid by replacing the '.' with 'ManeStyle' (because unicorns also vary in terms of mane style, obviously). What do these **small multiples** tell you about the relationship between horn length, height, magic and mane style?

2. Wrap

Instead of a grid, try using a wrap when slicing by 'Magic. You will notice that your data points occupy certain regions of each panel. When faceting by **grids**, we always want each panel to have the same scales, but with **wraps** we are sometimes more interested just in seeing the general relationships. Try adding 'scales = "free"' to your facet_wrap command…

Further reading

The ggplot2 package allows you complete control over your figures. This has been a brief introduction to how to plot data effectively in R, but it is only the merest tip of the iceberg! Here are some resources on what makes an effective data visualisation, and others on how to create these using ggplot2:

Rougier NP, Droettboom M, Bourne PE (2014) Ten Simple Rules for Better Figures. PLoS Comput Biol 10(9)

Kelleher C, Wagener T (2011) Ten Guidelines for Effective Data Visualisation in Scientific Publications. Environmental Modelling and Software 26(6)

'The Visual Display of Quantitative Information'. Edward Tufte.

Practical rules for using colour.

The R Graphics cookbook.

ggplot2 documentation.

RStudio's ggplot2 cheatsheet.