



Data Visualisation in Rstudio using GGPlot2

Anders Jensen

Contents

Introduction.....	2
Installing and loading GGPlot2.....	3
The fundamentals of a ggplot2.....	3
The data	3
The type of graph	4
GGPlot2 Basic code.....	5
Making a scatter plot.....	6
Making a bar plot.....	8
Making a histogram	9
Making a Boxplot.....	10
Making a line plot	11
Editing the ggplots.....	12
Changing colours	12
Adding data from another dataset onto a ggplot.....	13
Changing axis limits and breaks	13
Labelling points.....	13
Faceting plots	14
Saving ggplot images to your computer	14
Additional ggplot features.....	14
Plotly for animated plots	14
Ggimage for images on graphs	15
Ggpubr and rstatix for statistical tests on graphs.....	15
Dplyr cheat sheet.....	16
How to implement the ggplot2 package to recreate plots you see online	17
Code for plots in table 1	17
Further information	18
References	18

Introduction

One of the most important stages on all data analysis pipelines is understanding how to best visualise your data so that they can be easily understood and represented. The challenge of presenting our research and findings is made much simpler with aesthetically pleasing diagrams and figures that keep audiences informed and engaged.

The biggest problems we may face with data visualisation include, determining which type of graph to use and how to then produce the figure. Understanding which type of graph to use comes with experience and considering two main factors; What is the important part of your data? What type of variables are you trying to show? In this workbook we will go through some examples of how to decide on the type of graph. This workbook will also provide the framework for how to make these plots using Rstudio and specifically a package called GGPlot2.

GGPlot2 is a highly popular R package invented by **Hadley Wickham** in 2005 and is used to design graphics for people utilising R programming language. There is a built in system for making plots in R, however GGPlot2 is much more effective, due to simplicity and ability to control more plot features.

GGPlot2 aims to implement **Leland Wilkinsons Grammer of Graphics** which is a book that was released in 2005 and is a fascinating insight into how all graphics can be split into certain components. The following [youtube video](#) explains these principles well for those who may be interested. For anyone really interested here is the [link to original book](#).

Before attempting this guide, it is recommended that you look through the [Introduction to R programming and using Rstudio. A beginner's guide to Rstudio](#). This can be found on the [Knowhow Statistics & Coding page](#) along with other handbooks.

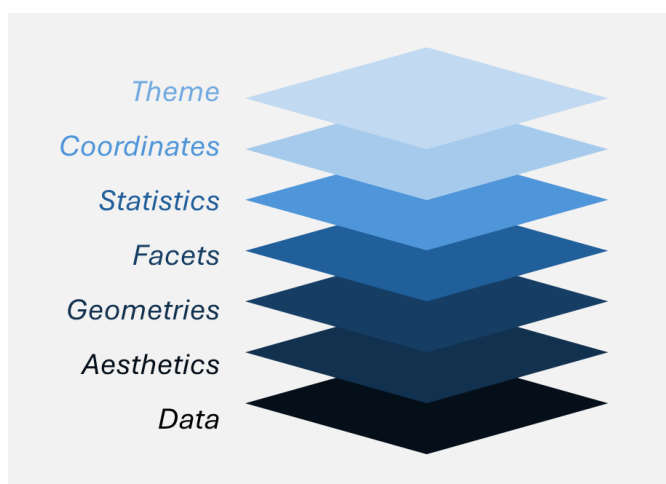


Fig 1 – Grammer of graphics redrawn from Wilkinson and Wills (2005)

Installing and loading GGPlot2

This should be very simple if you have worked through the previous handbook but as a reminder:

To install a package in Rstudio you need to run `install.packages("name_of_package")` and then run `library(name_of_package)`, see below.

```
install.packages('ggplot2')  
library(ggplot2)
```

If you have issues with this then please visit the [GGplot website](#) for more information.

The fundamentals of a ggplot2

The data

To begin producing a graphic you first need a dataset to work with. For the purposes of this workbook we will be using datasets that are already built into R. To look at the available datasets you can run the following code:

```
# Look at available datasets  
data()
```

This should show up a list of datasets that you can use, for example: `AirPassengers` or `ChickWeight`

To then save one of these datasets into your environment you need to run the following code

```
# Save data into environment  
data('dataset_name')
```

Make sure to change `dataset_name` to the name of the data: 'iris', 'ChickWeight' etc

Have a go at loading in the iris dataset into your environment. This dataset is Edgar Andersons famous data on the variable's sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.

The type of graph

There are too many types of graphs to list, however some of the most common include scatter plot, box plot, violin plot, line plot, bar plot, histogram etc.

The **following table** contains information about when these plots may be appropriate and the type of data that may suit them.

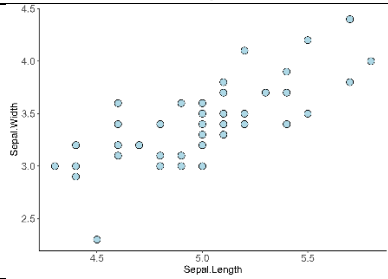
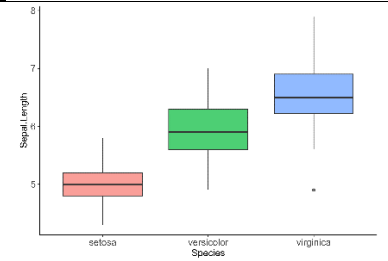
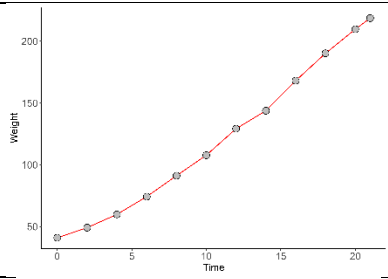
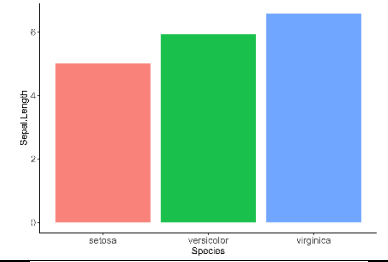
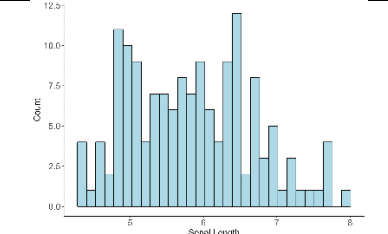
Plot	Variables	Information	Sample
Scatter plot	Two numeric continuous variables	Shows two values on an x and y axis.	
Box plot	One numeric variable and one continuous variable.	This allows you to show a continuous variable amongst different groups. You can also visualise distribution	
Line plot	One numeric variable on the Y-axis then either a ordinal or continuous variable	Useful for tracking a value over another constant (time, concentration etc)	
Bar plot	One numeric variable and one continuous variable.	Similar to boxplot but shows less information. Can be easier to interpret.	
Histogram	One continuous or ordinal variable	Shows the distribution of data	

Table 1: Types of ggplot2 visuals

The code used to make the graphs in the table is included at the back of the book.

GGPlot2 Basic code

This is the most important part. How to begin writing ggplot2 code.

You can think of ggplot2 as a cake where you can keep on adding ingredients to make it more complex. No matter how good the cherry on top looks, you still need flour, eggs and milk.

The following line of code is the backbone of every ggplot and should always be your starting point whenever making a plot. You will see this a lot more moving forward in this document so take the time to familiarise yourself.

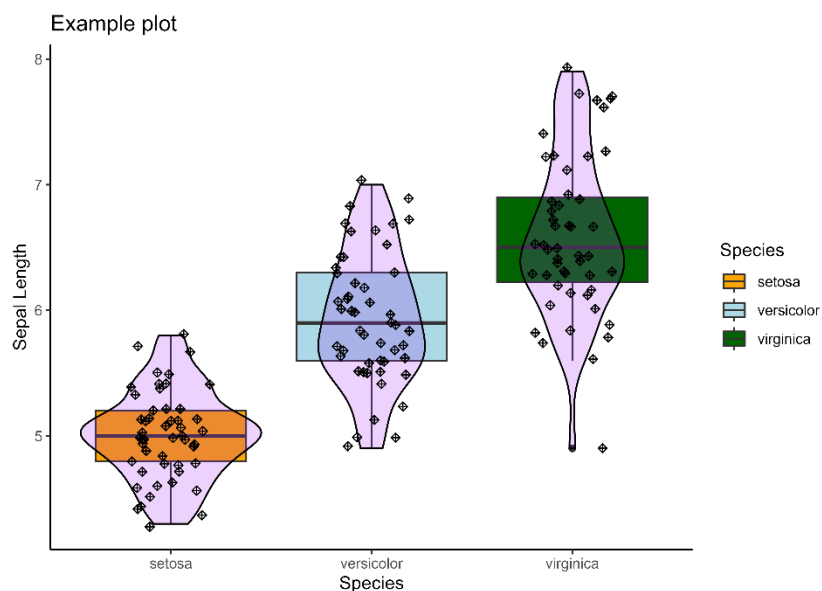
ggplot(data, aes(x = x_axis, y = y_axis))

So lets break this code down:

- **ggplot** – This is the ggplot2 function to say you are wanting to make a graphic
- **data** – this is the name of the data frame saved in your environment
- **aes** – this is a function to state that you are looking inside the data frame
- **x** – this is the column name in your data frame you want on the x-axis of the plot
- **y** - this is the column name in your data frame you want on the x-axis of the plot

Any of the text highlighted in green will change depending on the names you use in your data

Then if you wanted to add extra information to the ggplot2 you would use a **+** sign and write a new line of code. This is where you can add information about the type of plot.



```
ggplot(iris, aes(x = Species, y = Sepal.Length))+  
  geom_boxplot(aes(fill = Species))+  
  geom_violin(alpha = 0.2, fill = "purple", col = "black")+  
  geom_jitter(width = 0.2, shape = 9)+  
  theme_classic()+  
  scale_fill_manual(values = c("orange", "lightblue", "darkgreen"))+  
  labs(x = "Species", y = "Sepal Length", title = "Example plot")
```

Making a scatter plot

We will begin with making a simple scatter plot. A scatter plot is appropriate for when you are trying to graph 2 continuous variables against each other, such as height and weight.

For this example, we will use the 'iris' dataset. For more information on this data, you can run the following code `?iris`

Firstly, we will load in the data

```
# Load in dataset
data("iris")
```

Then we will use the code from above to build a blank ggplot

```
# Make a blank ggplot
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))
```

Then we will use the + sign to add some points to the graph

```
# Add some points to the plot
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point()
```

Hopefully if you followed this code, you should see a simple ggplot in the plots window with Sepal width on the y-axis and Sepal length on the x-axis.

Make sure column names are spelt correctly, and brackets are closed correctly.

We will now change the colour of the points depending on the species of the plant

```
# Change colour of points
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(col = Species))+
  scale_color_manual(values = c("purple", "green", "blue"))
```

Notice how we used `col = Species` inside of the aesthetics. This is because Species is within the dataset.

If we just wanted all the dots to be red then we wouldn't need `aes()` and the code would be:

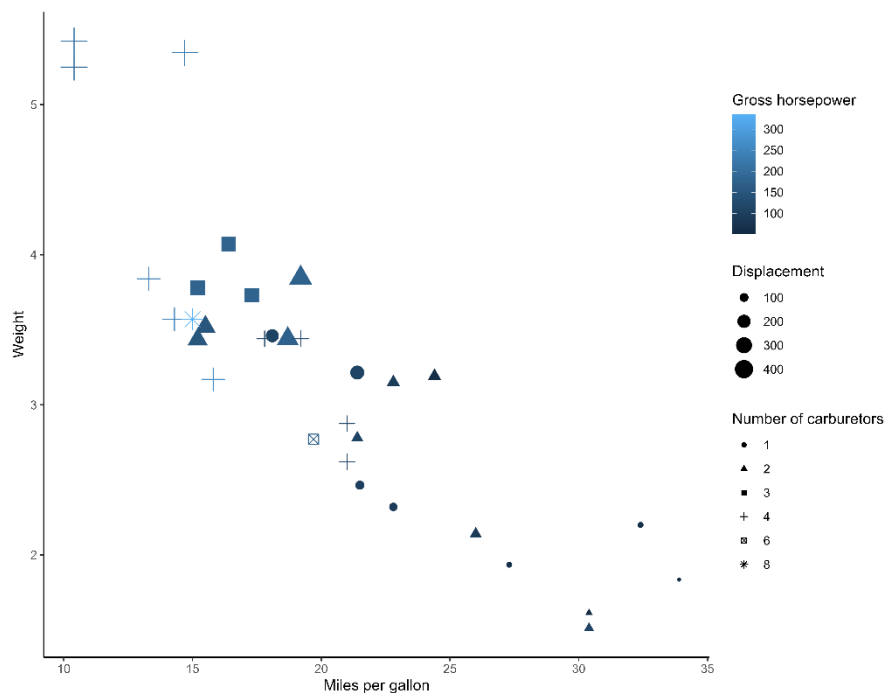
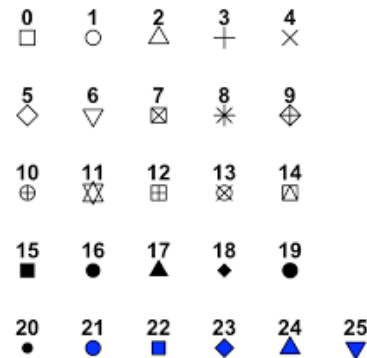
```
geom_point(col = 'red')+
```

Finally, we can try changing both the size and the shape of the points

```
# Change size and shape of ggplot points
ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(aes(col = Species),
            size = 10,
            shape = 21,
            fill = "grey")+
  scale_color_manual(values = c("purple", "green", "blue"))
```

This plot is not the most visually appealing plot so have a go changing some parameters to improve the appearance.

The different shape options can be seen here.



```
ggplot(mtcars, aes(x = mpg, y = wt))+
  geom_point(aes(size = disp,
                shape = as.factor(carb),
                col = hp))+
  theme_classic()+
  labs(x = "Miles per gallon", y = "Weight",
       shape = "Number of carburetors",
       col = "Gross horsepower", size = "Displacement")
```

Making a bar plot

A bar plot or bar chart is another kind of plot that is suitable for when you have a continuous variable and a categorical variable.

For this example, we will use the "InsectSprays" dataset. For more information on this data, you can run the following code **?InsectSprays**

Firstly, we will load in the data

```
# Load in data
data("InsectSprays")
```

Then we will build the blank plot

```
# Make a blank ggplot
ggplot(InsectSprays, aes(x = spray, y = count))
```

Then we will add the columns

```
# Add bar to plot
ggplot(InsectSprays, aes(x = spray, y = count))+
  geom_bar(stat = 'summary', fun = mean)
```

Then we can change colours

```
# Change colour of bars based on count
ggplot(InsectSprays, aes(x = spray, y = count))+
  geom_bar(stat = 'summary',
          fun = mean,
          col = "black",
          aes(fill = spray))
```

Then we can change labels and size of text

```
# Change labels and size of text
ggplot(InsectSprays, aes(x = spray, y = count))+
  geom_bar(stat = 'summary',
          fun = mean,
          col = "black",
          aes(fill = spray))+
  labs(x = "X axis label",
       y = "y axis label",
       fill = "colour label")+
  theme(axis.text = element_text(size = 15),
        axis.title = element_text(size = 18),
        legend.text = element_text(size = 14),
        legend.title = element_text(size = 16))
```


The **labs()** functions allows you to specify different labels on the graph and the **theme()** function allows you to control various aspects of your graphs.

The **stat = summary, fun = mean** arguments allow you to take long data and only plot the mean value, if you want to plot the sum you can change the code to **fun = sum**

The **theme()** function allows you to change certain aspects of the plot. In this example we have changed the size of the text of the axis titles as well as the text on the axis themselves. You can also change the colour and font this way.

Making a histogram

A histogram is another kind of plot that is suitable for presenting the distribution of data.

For this example, we will use the “ToothGrowth” dataset. For more information on this data, you can run the following code **?ToothGrowth**

The sequence of building this ggplot is the same as the ones before it.

- Load data
- Build blank ggplot
- Specify the type of ggplot
- Tidy up the plot

```
# Load in data
data("ToothGrowth")
?ToothGrowth
# Build plot
ggplot(ToothGrowth, aes(x = len))+
  geom_histogram(col = 'black', fill = "lightblue")+
  theme_classic()+
  theme(axis.text = element_text(size = 13),
        axis.title = element_text(size = 13))+
  labs(y = "Count", x = "Length of tooth")
```

When plotting a histogram you don't need to specify the y-axis as this is assumed as the count when used the function **geom_histogram**.

While histograms are good at visualising data distribution, the Shapiro-wilk's test or the Kolmogorov-Smirnoff test may be preferred choices to quantitatively test the normality of data.

Instead of a histogram you can also represent data as a density plot using the code below.

```
ggplot(ToothGrowth, aes(x = len))+
  geom_density(col = 'black', fill = "lightblue")+
  theme_classic()+
  theme(axis.text = element_text(size = 13),
        axis.title = element_text(size = 13))+
  labs(x = "Length of tooth", y = "Density")
```

Making a Boxplot

A boxplot is another plot, which is similar to a barplot and can be used when you have one continuous variable and a categorical variable.

For this example, we will use the “starwars” dataset. For more information on this data, you can run the following code **?starwars**

Again the basic structure of the plot is the same as the others

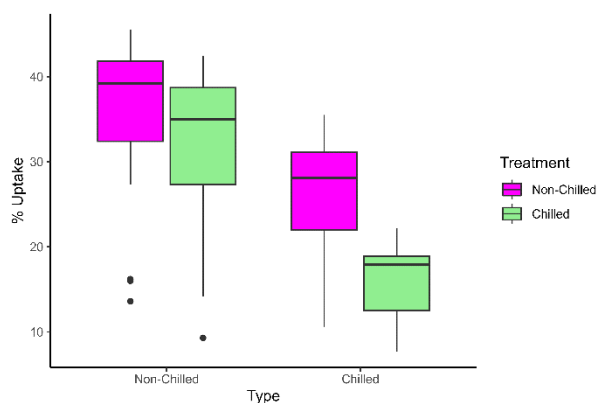
- Load data
- Build blank ggplot
- Specify the type of ggplot
- Tidy up the plot

```
# load in data
data("starwars")
?starwars
# Build plot
ggplot(starwars, aes(x = gender, y = height))+
  geom_boxplot(aes(fill = gender),
               show.legend = F,
               outlier.shape = NA)+
  scale_fill_manual(values = c("red", "purple"))
```

The **scale_fill_manual()** function allows you to specify the colours for each of the boxes.

As a challenge have a go at changing the labels and the size of the labels in the plot.

Hint: **labs(), theme()**



```
ggplot(CO2, aes(x = Type, y = uptake))+
  geom_boxplot(aes(fill = Treatment))+
  theme_classic()+
  labs(x = "Type", y = "% Uptake")+
  scale_x_discrete(labels = c('Non-Chilled', 'Chilled'))+
  scale_fill_manual(values = c("magenta", "lightgreen"),
                    labels = c('Non-Chilled', 'Chilled'))
```

Making a line plot

A line plot is useful for representing a change in a continuous variable over a certain time, concentration or amount.

For this example, we will use the “ChickWeight” dataset. For more information on this data, you can run the following code **?ChickWeight**

```
# Load in data
data("ChickWeight")
?ChickWeight

# transform data
ChickWeight <- ChickWeight %>%
  group_by(Time) %>%
  summarise(avg_weight = mean(weight))

# build plot
ggplot(ChickWeight, aes(x = Time, y = avg_weight))+
  geom_line(col = 'red')+
  geom_point(fill = "blue",
             col = "black",
             shape = 21,
             size = 4)+
  theme_classic()+
  theme(axis.text = element_text(size = 13),
        axis.title = element_text(size = 13))+
  labs(x = "Time", y = "Weight")
```

This is the first example of a plot where we had to do some data cleaning before building the plot. This was completed using the dplyr package which you will need to install and load first.

There is some basic information about dplyr near the end of this book. For further information the following [youtube channel](#) has many useful guides.

Editing the ggplots

Once you have built the basic ggplot there will be lots of subtle changes you may want to make to increase the readability and design of your plot.

Nearly every change you want to make will be achievable but may get slightly more complicated.

Some possible problems are listed below but the best sources to find answers to problems are YouTube, stack overflow or statistics tutors at the University of Liverpool

The following code is used to make an example plot which will be used to answer the questions below.

```
# Make example plot
plot <- ggplot(iris, aes(x = Sepal.Length, y = Sepal.Width))
plot2 <- ggplot(iris, aes(x = Species, y = Sepal.Length))
```

Be sure to run this code before proceeding so `plot` and `plot2` is saved into your environment

Changing colours

To change colours of things in your ggplot you need to use the `col =` or the `fill =` arguments

```
plot +
  geom_point(shape = 21, col = "black", fill = "red")+
  theme(axis.title = element_text(colour = "red"))
```

You can see here we have changed the outline and colour of the points as well as the colour of the axis titles.

You can use hex colour codes instead of the name of the colour for a more specific look to your ggplot.

The [rapidtables website](#) if useful for this.

If you want your colours to be based on another column in the data set you can do this by using the **aes()** function. To change these colours you can use a couple of different functions

```
library(RColorBrewer)
plot+
  geom_point(aes(fill = Species, col = Species),
             shape = 21,
             size = 4)+
  scale_fill_manual(values = c("#33FF33", "blue", "#9932CC"))+
  scale_color_brewer(palette = "Set2")
```

The code above shows how to use `scale_fill_manual` and `scale_col_manual` as well as how to use the **rcolorbrewer** package to use pre-made colour palettes. [See the options for this package.](#)

Make sure to install and load “RColorBrewer” before trying this.

Adding data from another dataset onto a ggplot

Let's say you made a ggplot using one dataset and then wanted to also include some information from another dataset then this is how you would do it.

```
data("iris3")
plot +
  geom_point(shape = 21, col = "black", fill = "red") +
  geom_point(inherit.aes = F,
             data = as.data.frame(iris3),
             size = 6,
             aes(x = `Sepal L..Setosa`, y = `Sepal W..Setosa`))
```

This is useful for when you are combining data from separate datasets.

Changing axis limits and breaks

This is when you want to specify the limits of the axis and also change the breaks on the plot so that more or less numbers are shown.

```
plot +
  scale_x_continuous(breaks = c(5, 5.3, 5.5, 6, 6.5, 7),
                    limits = c(5, 7)) +
  scale_y_continuous(breaks = c(2.5, 3.0, 3.2, 3.5, 4.0, 4.5),
                    limits = c(2.5, 4))
```

This example may seem a bit silly, however sometimes it can be useful to specify the breaks on a plot.

Labelling points

If you wanted to have text on a ggplot to label some information from your dataframe you can do that.

```
plot +
  geom_point() +
  geom_label(aes(label = Species,
                fill = Species,
                col = Sepal.Width))
```

The R package **ggrepel** has a really useful function called `geom_label_repel` which can be used to avoid overcrowding when labelling points

```
library(ggrepel)
plot +
  geom_point() +
  geom_label_repel(aes(label = Species,
                    fill = Species,
                    col = Sepal.Width))
```

Faceting plots

Faceting in ggplot refers to the process of creating multiple plots based on subsets of the data, all laid out together in a grid. This is particularly useful when you want to compare the same plot type (e.g., scatter plot, line plot) across different categories or levels of a variable.

```
plot+  
  geom_point(aes(col = Species))+  
  facet_wrap(~Species)
```

Saving ggplot images to your computer

Saving your ggplot is one of the most important steps so that you can produce high quality images.

The **ggsave()** function is perfect for this and is really useful for making nice looking plots

```
plot_save <- plot+  
  geom_point(aes(col = Species))  
  
ggsave(plot = plot_save, # name of plot to save  
       device = 'png', # type of file  
       height = 5, # height  
       width = 5, # width  
       dpi = 600, # image quality (600) is good  
       path = "", # need to specify this to save it somewhere  
       filename = "saved_plot.png") # name of file
```

Additional ggplot features

The following features are slightly more advanced and don't worry if they seem a bit confusing as this is just meant to introduce you to some of the ways in which ggplot can be used to do more complex things.

Also, this will give you an idea of where to look for more information

Plotly for animated plots

```
library(plotly)  
plot_ly(z = ~volcano) %>%  
  add_surface()
```

This will make an animated map of a volcano, specifically Auckland's Maunga Whau Volcano

Ggimage for images on graphs

For this you will need to save an image to your computer that you want on your graph

```
library(ggimage)
# Edit data
new_data <- iris %>%
  mutate(image = case_when(Species == 'setosa' ~ "path_to_image.png"))
# make plot
plot_image <- new_data %>%
  filter(Species == "setosa") %>%
  ggplot(aes(x = Sepal.Length, y = Sepal.Width)) +
  geom_image(aes(image = image)) +
# save plot
ggsave(plot = plot_image,
        width = 8,
        height = 8,
        filename = "image.png",
        device = "png",
        dpi = 800,
        path = "../")
```

Just change the “path_to_image.png” to wherever your image is saved

Ggpubr and rstatix for statistical tests on graphs

The ggpubr package is used for improving the readability of graphs and including statistical tests on plots. When used with the rstatix package you can make figures with the statistical test directly on the plot.

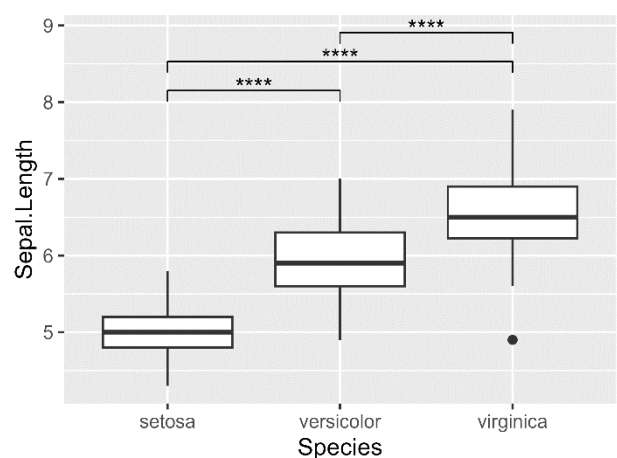
This takes a little bit of data cleaning and organising.

Remember to install and load required packages

```
# packages
library(rstatix)
library(ggpubr)
library(reshape2)

# statistical test
stats_test <- iris %>%
  select(Sepal.Length, Species) %>%
  melt() %>%
  tukey_hsd(value~Species) %>%
  add_xy_position() %>%
  add_significance()

# put statistics on plot
plot2+
  geom_boxplot() +
  stat_pvalue_manual(data = stats_test)
```



Dplyr cheat sheet

Dplyr is a really useful package for data cleaning and the following functions are a good start to learn more

`filter()` – Extract rows based on a specific condition

`rename()` – Rename columns in a dataframe

`mutate()` – Make new columns in a dataframe

`summarise()` – Transform data into a single row

There are many more but this is a good starting point

The final bit of information regarding dplyr is to understand how the `%>%` (pipe operator) works.

The pipe allows you to pass many functions at the same time. So instead of this:

```
# Without the %>%
iris1 <- rename(iris, sepal.length = Sepal.Length)
iris2 <- mutate(iris1, length = sepal.length+Petal.Length)
iris3 <- filter(iris2, Species == "setosa")
```

You can code it like this:

```
# With the %>%
iris_all <- iris %>%
  rename(sepal.length = Sepal.Length) %>%
  mutate(length = sepal.length+Petal.Length) %>%
  filter(Species == "setosa")
```

Both methods work however the second one is easier to write and saves less objects into the environment.



How to implement the ggplot2 package to recreate plots you see online

Instructions for recreating plots found online can be found on Canvas. Don't forget to reference any figures or plots you recreate if you use someone else's data.

Access Canvas using one of the following options (Undergraduates and Taught Postgraduates will automatically be enrolled in their faculty/PGT course. Staff and PGRs will need to self-enrol).

- [Health and Life Sciences – Undergraduate](#)
- [Humanities and Social Sciences – Undergraduate](#)
- [Science and Engineering – Undergraduate](#)
- [Taught Postgraduate](#)
- [Self-Enrol](#)

From the home page go to 'Statistics' 'Making Graphs' and scroll to the end of the page.

Code for plots in table 1

Here is the code for all the plots used to make table 1

```
iris %>%
  filter(Species == 'setosa') %>%
  ggplot(aes(x = Sepal.Length, y = Sepal.Width))+
  geom_point(size = 4,
             col = 'black',
             shape = 21,
             fill = "lightblue")+
  theme_classic()+
  theme(axis.text = element_text(size = 13),
        axis.title = element_text(size = 13))

iris %>%
  ggplot(aes(x = Species, y = Sepal.Length))+
  geom_boxplot(aes(fill = Species),
              alpha = 0.7, show.legend = F) +
  theme_classic()+
  theme(axis.text = element_text(size = 13),
        axis.title = element_text(size = 13))

ChickWeight %>%
  group_by(Time) %>%
  summarise(weight = mean(weight)) %>%
  ggplot(aes(x = Time, y = weight))+
  geom_line(col = "red")+
  geom_point(size = 4, shape = 21, col = "black", fill = "grey")+
  theme_classic()+
  theme(axis.text = element_text(size = 13),
        axis.title = element_text(size = 13))+
  labs(x = "Time", y = "Weight")
```

```
iris %>%
  ggplot(aes(x = Species, y = Sepal.Length))+
  geom_bar(aes(fill = Species),
    alpha = 0.9,
    show.legend = F,
    stat = "summary", fun = mean) +
  theme_classic()+
  theme(axis.text = element_text(size = 13),
    axis.title = element_text(size = 13))

iris %>%
  ggplot(aes(x = Sepal.Length))+
  geom_histogram(col = "black", fill = "lightblue")+
  theme_classic()+
  theme(axis.text = element_text(size = 13),
    axis.title = element_text(size = 13))+
  labs(y = "Count")
```

Further information

If you require any further information or assistance please visit the [knowhow statistics & coding page](#).

The aim of this book was to give you an introduction to plotting in ggplot2 and to hopefully improve your confidence using Rstudio.

When applying these principles to your own data make sure the data is in the same structure as the examples used in this booklet.

If you have any advice on how to improve this booklet or anything that should be included, please do not hesitate to get in touch (knowhow@liverpool.ac.uk). Likewise, if this book has been useful then please let us know.

Good luck with your R coding and ggplotting.

References

Wilkinson, L. and Wills, G. (2005) *The Grammar of Graphics by Leland Wilkinson*. 2nd ed. 2005. New York, NY: Springer New York. Available at: <https://doi.org/10.1007/0-387-28695-0>.