Data visualisation tutorial

# Practical

Please note: the code you need to type is in the grey boxes. Sometimes comments are added following a #, to further explain some of the code.

## ggplot2

To start using ggplot2, first open RStudio, then install and load the package using library(ggplot2). As we are working in the cloud today, you won’t need to do this (but you will once you use it on your own computer).

There is a lot of information about ggplot2 online, so if you get stuck you can try to google what you are trying to do. There is also a useful data visualisation cheat sheet - once you have used ggplot2 a bit, it will become more and more useful: <https://www.rstudio.com/resources/cheatsheets/>

## Useful links/resources

There is lots of useful info online on making plots with ggplot2, but this book is a great starting point:

Wickham, H. and Grolemund, G. (2016) R for data science. Free at <http://r4ds.had.co.nz/> It was written by the guy who wrote the package ggplot2.

Another useful book is this one: <http://www.cookbook-r.com/Graphs/>

The evolution of a ggplot:

<https://cedricscherer.netlify.com/2019/05/17/the-evolution-of-a-ggplot-ep.-1/>

## A very basic plot in ggplot2

We will use data that are in-built in R for this session, using the iris and gapminder datasets.

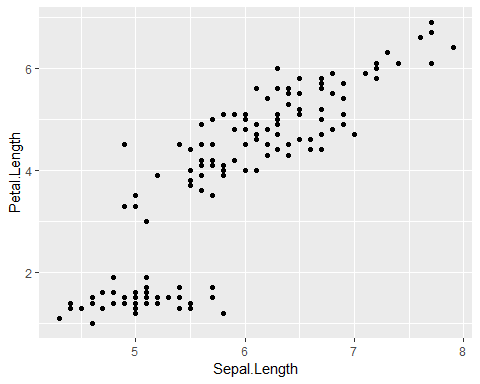
str(iris)

## 'data.frame': 150 obs. of 5 variables:  
## $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
## $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
## $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
## $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
## $ Species : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...

# Have a look at the iris dataset, so you know which variables there are and what types of variables they are

We will start with some short code that will produce a plot in ggplot2. The basic pattern of the ggplot2 function first names the dataset, then the x and y variables. With geom\_, you add the type of plot you want:

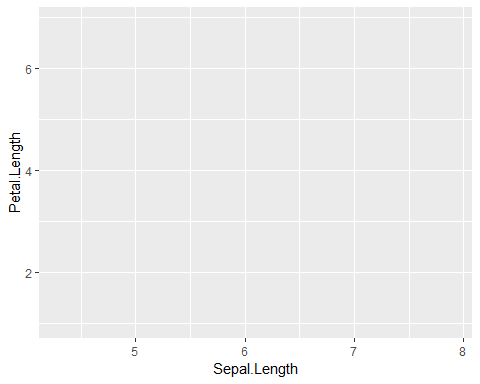
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point()



# this will now produce a basic scatterplot of sepal lengths against petal lengths

The plot will be empty if you don’t specify the type of plot you want:

ggplot(data = iris, aes(x = Sepal.Length, Petal.Length))

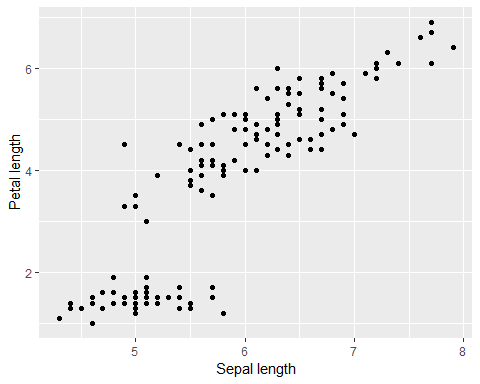


## Customising a plot

### Axis labels

We can now start to customise the plot, so it looks exactly how we want it to look. We can change the axis labels by adding xlab or ylab:

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +   
 geom\_point() +  
 xlab("Sepal length") +  
 ylab("Petal length")

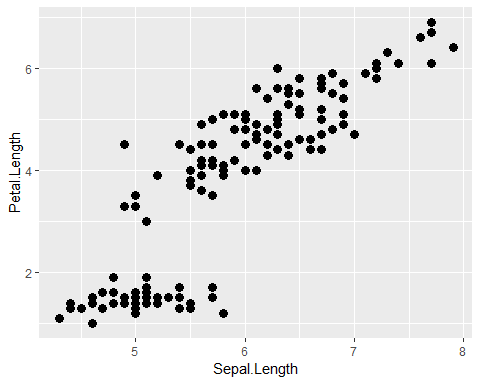


**Exercise**: Change the variable on the x-axis from Sepal.Length to Petal.Width. Make sure to change the axis labels too.

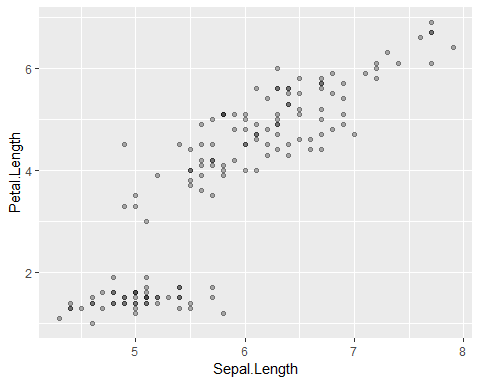
### Size and transparency

If you have a lot of data points, it can be difficult to see all of the points as they might overlap. You can change the size or transparency of the points so they are easier to see:

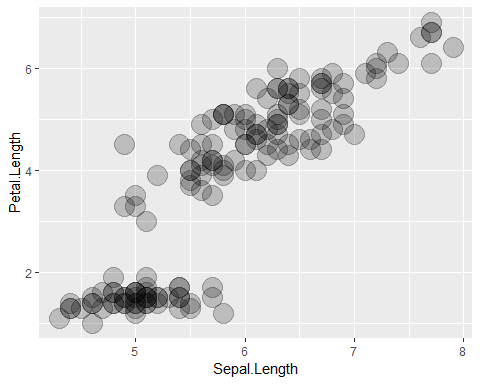
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(size = 3)



# changing the size to 3 will make the dots bigger  
  
  
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(alpha = 0.3)



# Changing alpha to 0.3 will make the dots more transparent. Some of the dots appear darker because there is actually more than one data point in the same place  
  
  
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(size = 7, alpha = 0.2)



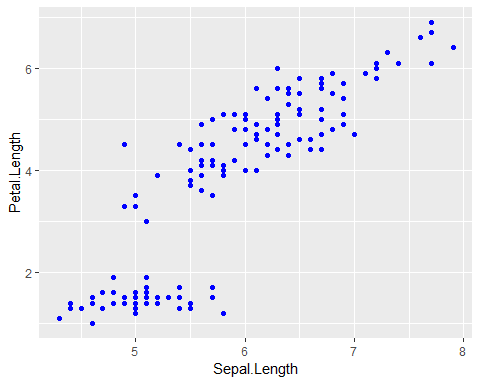
# You can change the look of a plot by changing both size and transparency

**Exercise**: Change size, transparency, or both so that the dots can be seen better.

### Colours, sizes, scales

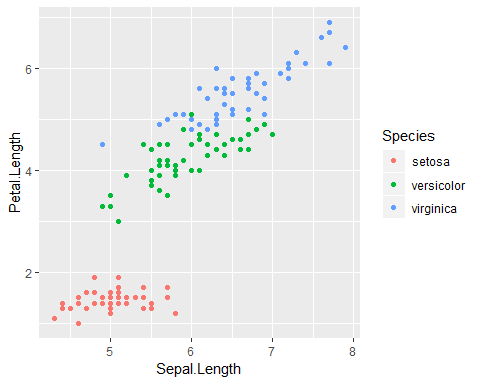
In ggplot2, we can change things like the colour, size or shape of our data points, either by changing all of them, or by making them look different depending on another variable. We can change the colour of all dots by defining the colour within geom\_point():

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(color = "blue")



We could also change the colour of each dot, depending on the type of species it is. To do so we need to add aes within geom\_point():

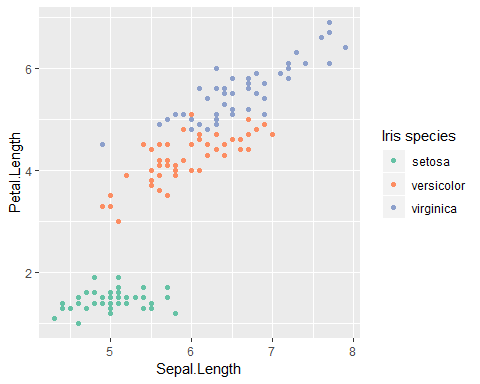
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(aes(color = Species))



# Note that there is a legend in your plot now!

The plot will be made with default colours, different for each species. To customise the colours, we need to adjust the scales:

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(aes(colour = Species)) +  
 scale\_colour\_brewer(palette = "Set2", name = "Iris species")



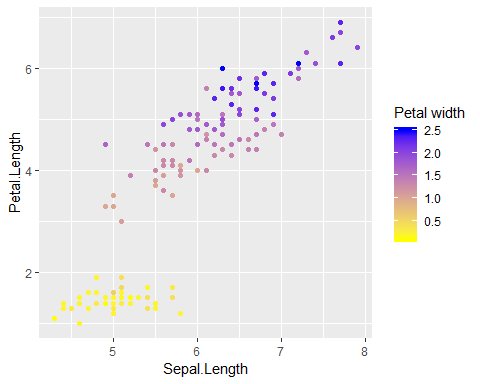
The scale function is used to customise how our data look, in this case we want to customise the colours in the plot. Whatever we specified in geom\_point as an aesthetic, in this case colour, we need to put into the scale function: scale\_colour. Here we will use colours from the RColorBrewer package, so we add brewer at the end. Inside the bracket we specify which colours we want to use, and name specifies the title of the legend. If you want to see all of the colour options in RColorBrewer, run the following code:

library(RColorBrewer)  
  
display.brewer.all()

**Exercise**: Change the colours of the plot, by picking a colour option from RColorBrewer.

In the previous example we used a categorical variable (Species) to set the colours. We can also use continuous variables for colours - the colours will change along a gradient:

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(aes(colour = Petal.Width)) +  
 scale\_colour\_gradient(low = "yellow", high = "blue", name = "Petal width")



In this example, we did not use RColorBrewer, but in-built R colour names. You can find a (very long) list of colour names in R here: <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

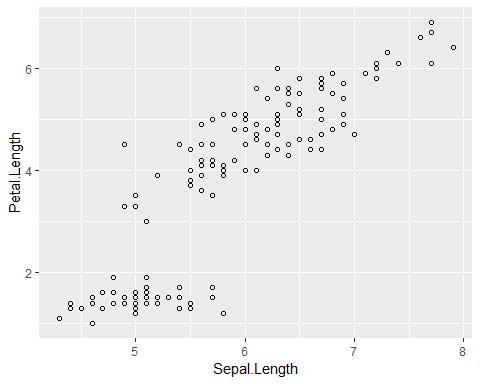
**Exercise**: Change the colours in the plot, using the in-built R colour names.

**Exercise**: We can also change the size of the dot according to another variable. Instead of changing the colour, can you change the size of the dot according to the Petal.Width variable? What is the difference in the code between making all dots bigger, and making them bigger depending on another variable?

### Shapes

We can change the dots to different symbols too, for example open circles or triangles. Each symbol corresponds to a number, which we add in geom\_point():

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point(shape = 1)

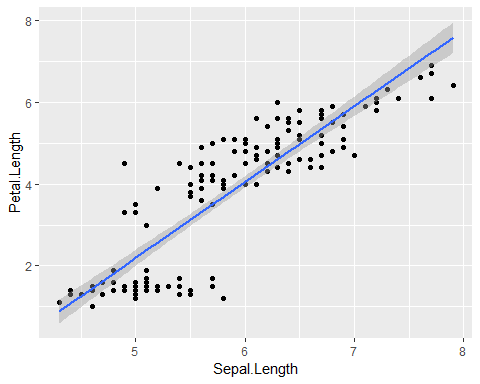


**Exercise**: Can you change the symbol to a square, and then to a cross? Use the ggplot2 cheat sheet (link at the start of this document) to find which number corresponds to which.

### Adding a line from your model

If you are presenting data to which you fitted a model, you can add the line by using geom\_smooth():

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point() +  
 geom\_smooth(method = glm)

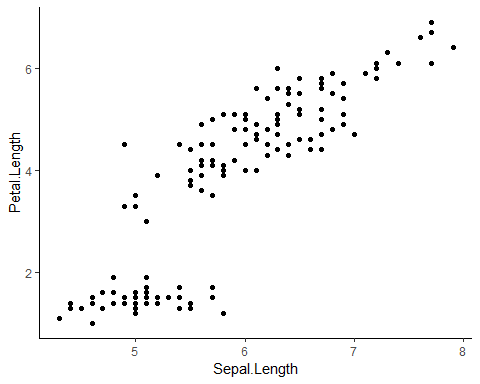


# Here we fit a generalised linear model to the data. 95% confidence intervals are displayed automatically

## Customising the overall look of the plot

We can change the overall look of the plot by changing the theme. The theme will not affect how your plotted data look, just the setup of the plot:

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) +  
 geom\_point() +  
 theme\_classic()



For a list of themes see <http://ggplot2.tidyverse.org/reference/ggtheme.html>.

**Exercise**: Try out three of the themes and see how the plots change.

## Using ggplot2 for different types of plots

Your plots will look different depending on the variables you are displaying - whether there are one, two or more to display, and whether they are categorical, ordinal or continuous. We will now use the dataset gapminder which is part of the gapminder package:

str(gapminder)

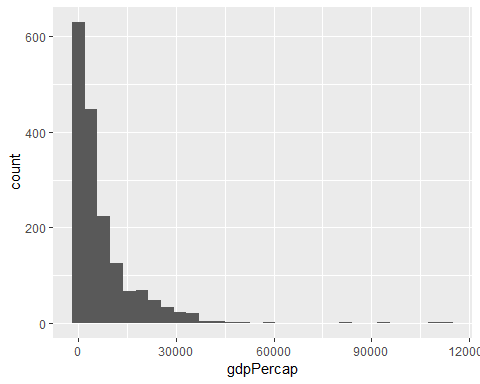
## Classes 'tbl\_df', 'tbl' and 'data.frame': 1704 obs. of 6 variables:  
## $ country : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num 28.8 30.3 32 34 36.1 ...  
## $ pop : int 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num 779 821 853 836 740 ...

### One continuous variable

If you have one continuous variable, you will likely use a histogram:

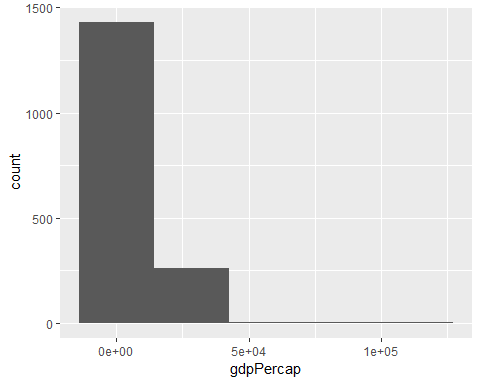
ggplot(data = gapminder, aes(x = gdpPercap)) +  
 geom\_histogram()

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.



You can adjust the number of bars by setting the binwidth:

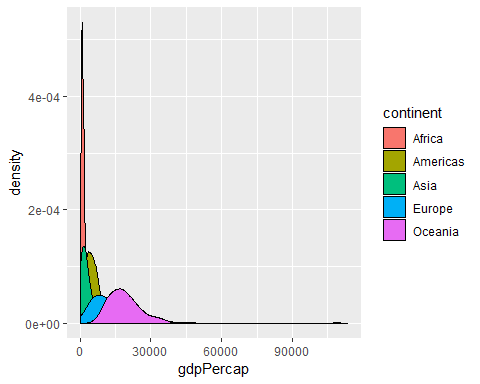
ggplot(data = gapminder, aes(x = gdpPercap)) +  
 geom\_histogram(bins = 5)



**Exercise**: Change the number of bins until they show the pattern in the data clearly.

You could also use a density plot to display one continuous variable. Note what happens to the y-axis:

ggplot(data = gapminder, aes(x = gdpPercap)) +  
 geom\_density(aes(fill = continent))



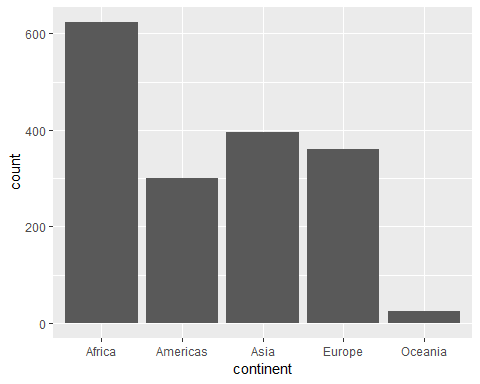
The area under the curve in a density plot sums to one, so the y-axis does not show a count, but a value between 0 and 1.

**Exercise**: In our last plot, the different densities were overlapping. Can you change the plot so we can see all densities at once?

### One categorical variable

If you have one categorical variable, you will need a bar graph. This will show a count of each of the categories:

ggplot(data = gapminder, aes(x = continent)) +  
 geom\_bar()



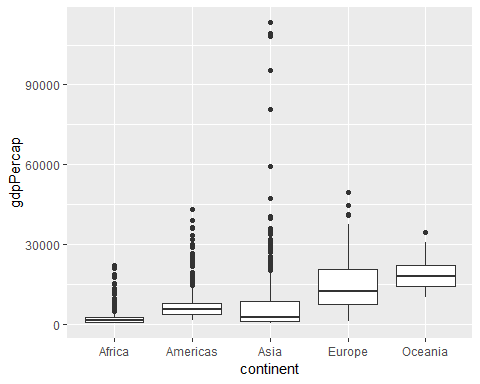
### Two continuous variables

If you have two continuous variables, you normally represent them with dots - see the examples from the iris dataset we used at the start.

### One continuous and one categorical variable

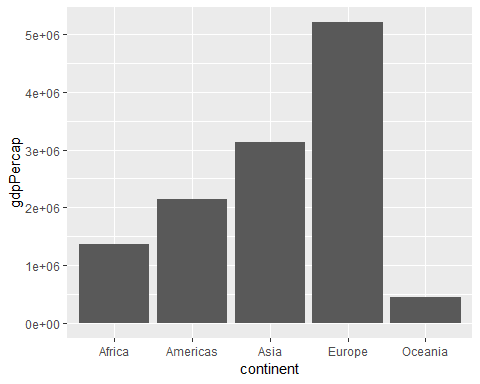
If you have a categorical and a continuous variable, you would normally put the categorical variable on your x-axis and the continuous variable on your y-axis. These are often displayed as box plots (they show the four quartiles in the data, and outliers as dots):

ggplot(data = gapminder, aes(x = continent, y = gdpPercap)) +  
 geom\_boxplot()



Or you could present your data as a bar plot. The default geom\_bar() option just uses a categorical variable for x. To show a categorical x and a continuous y variable, we need to add stat="identity" here:

ggplot(data = gapminder, aes(x = continent, y = gdpPercap)) +  
 geom\_bar(stat = "identity")



## Saving a plot

Your plots will be saved in the folder that you set as your working directory. This line of code will save the last ggplot you made to a pdf:

ggsave("plot.pdf")

Or you can save it as a jpg:

ggsave("plot.jpg")

You can specify the size by adding dpi. You can also specify the exact width and height (in inches) which is useful for publications:

ggsave("plot.pdf", dpi=300, width=5, height=5)