Data visualisation handout

## 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/>

On the grid package: <http://ww2.amstat.org/publications/jse/v18n3/zhou.pdf>

## Loading ggplot2 into R

To start using ggplot2, first open RStudio, then install and load the package with the following code:

install.packages("ggplot2")  
library(ggplot2)

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/>

## Some basics of ggplot2

In your code, you need to specify a few things:

* the type of plot you want (called geom)
* how you display your data - for example the colour or size of dots, bars etc (with scale functions)
* how the overall plot looks (the theme)
* whether you want subplots of your data (called facets)

## A very basic plot in ggplot2

We will use data that are in-built in R, using the iris, diamonds and mtcars datasets.

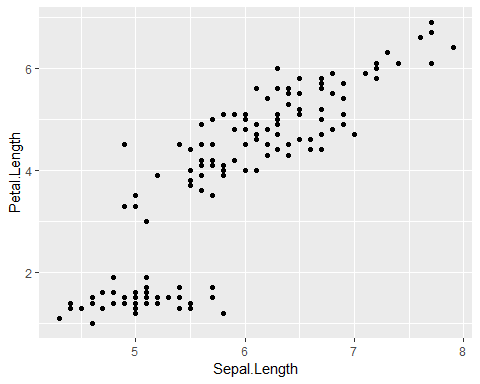
str(iris)

## '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:

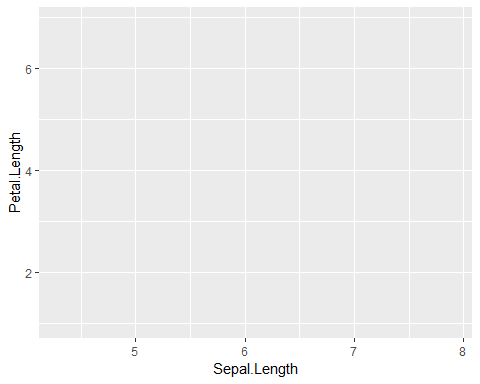
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point()



# 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))

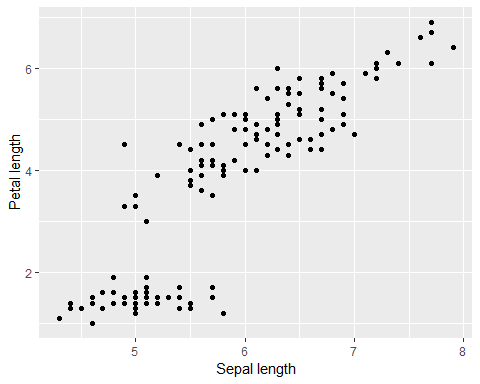


## Customising a plot

### 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:

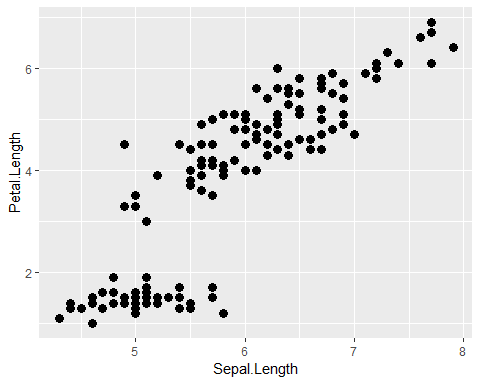
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point() + xlab("Sepal length") + ylab("Petal length")



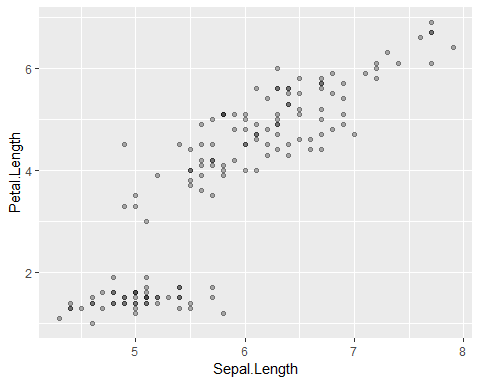
### Size and transparency

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:

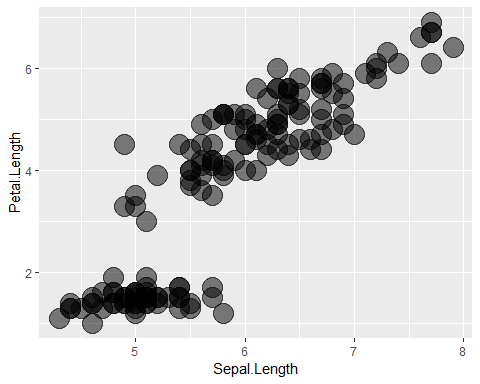
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point(size = 3)



# 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 = 0.5)

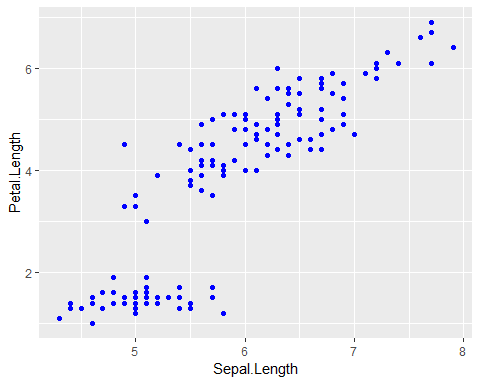


# You can change the look of a plot by changing both size and transparency

### Colour

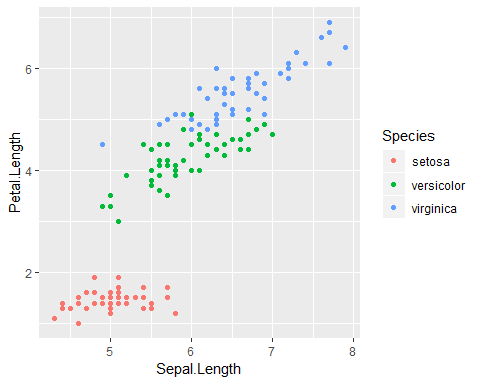
In ggplot2, we can change things like the colour, size or shape of our data points, either by changing all of them, or by making them look different depending on another variable. We can change the colour of all dots by defining the colour within geom\_point():

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():

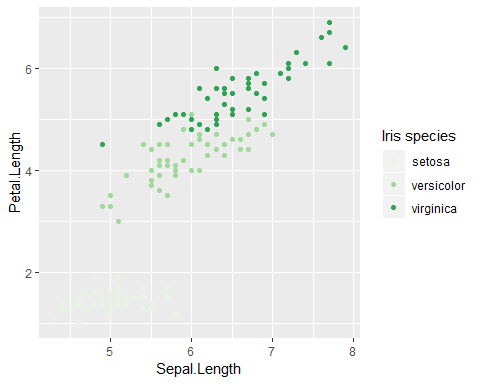
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point(aes(color = Species))



# 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 = "Greens", name = "Iris species")



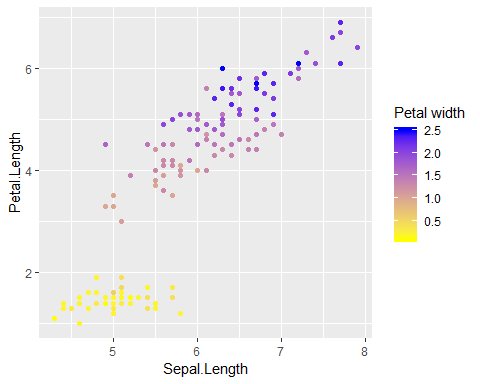
The scale function is used to customise how our data look, in this case we want to customise the colours in the plot. Whatever we specified in geom\_point as an aesthetic, in this case colour, we need to put into the scale function: scale\_colour. Here we will use colours from the RColorBrewer package, 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()

There is also a handy website which shows all the colour options with ColorBrewer, and you can filter them as well, for example for colour-blind friendly palettes: <http://colorbrewer2.org/>

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")

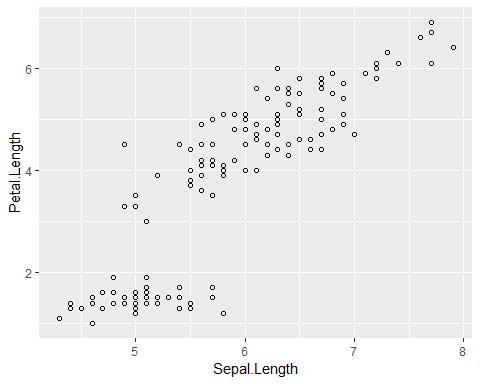


In this example, we did not use RColorBrewer, but in-built R colour names. You can find a (very long) list of colour names in R here: <http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf>

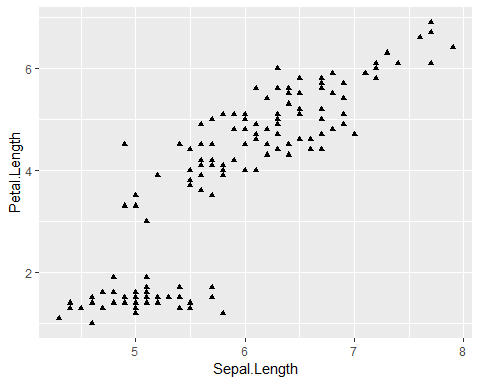
### Shapes

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)

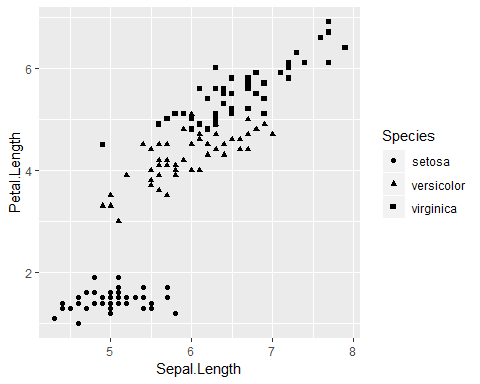


ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point(shape = 17)



We could also change the dots to symbols depending on another variable - similar to changing the colour depending on the species, as in the colour example. This can be useful if we need to use black-and-white plots:

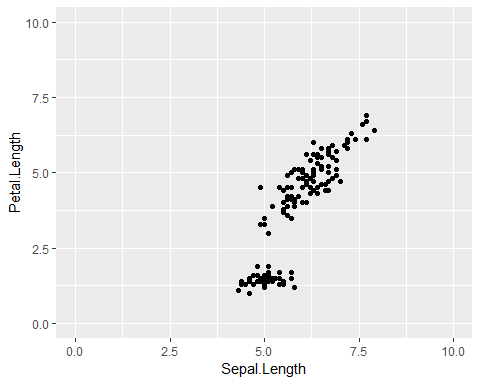
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point(aes(shape = Species))



### Zooming in on a plot

Sometimes we might not want to display the whole range of the x- and y-axes, or we want to start them from 0. We can specify the exact axes ranges we want to display using coord\_cartesian():

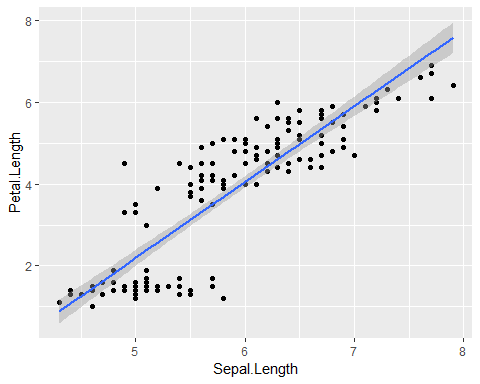
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point() + coord\_cartesian(xlim = c(0,10), ylim = c(0,10))



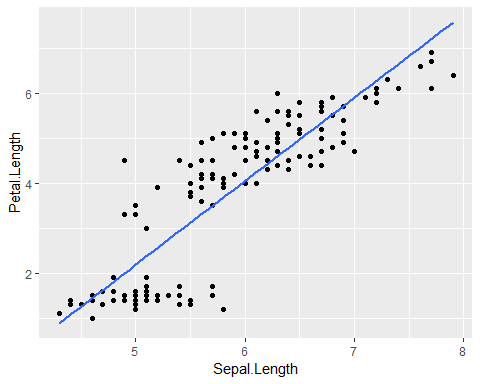
### Adding a line from your model

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

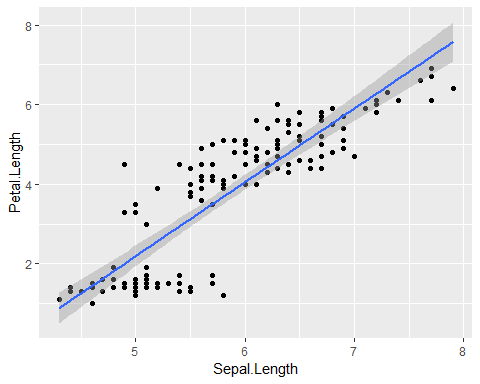
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point() + geom\_smooth(method = glm)



# Here we fit a generalised linear model to the data. 95% confidence intervals are displayed automatically  
  
ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point() + geom\_smooth(method = glm, se = FALSE)



ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point() + geom\_smooth(method = glm, level = 0.99)

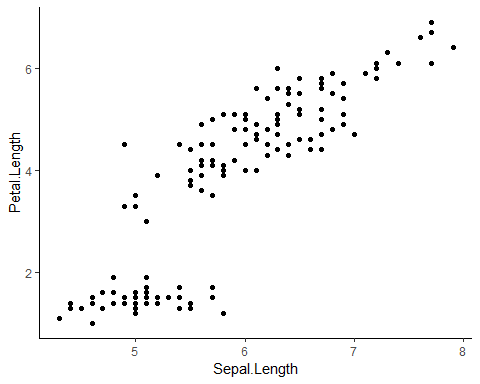


# We can also remove the confidence intervals, or change them to 99% confidence intervals

## 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>. We will look at how we can change these settings manually later.

## 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 diamonds which is part of the ggplot2 package:

str(diamonds)

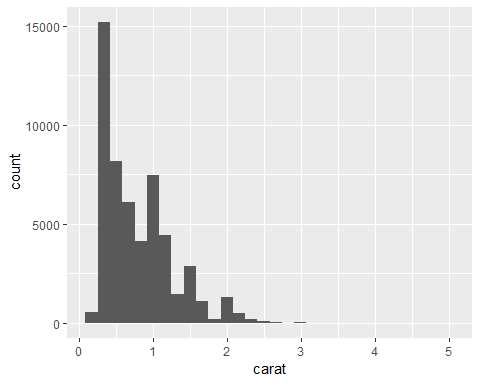
## Classes 'tbl\_df', 'tbl' and 'data.frame': 53940 obs. of 10 variables:  
## $ carat : num 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...  
## $ cut : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...  
## $ color : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...  
## $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...  
## $ depth : num 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...  
## $ table : num 55 61 65 58 58 57 57 55 61 61 ...  
## $ price : int 326 326 327 334 335 336 336 337 337 338 ...  
## $ x : num 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...  
## $ y : num 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...  
## $ z : num 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

### One continuous variable

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

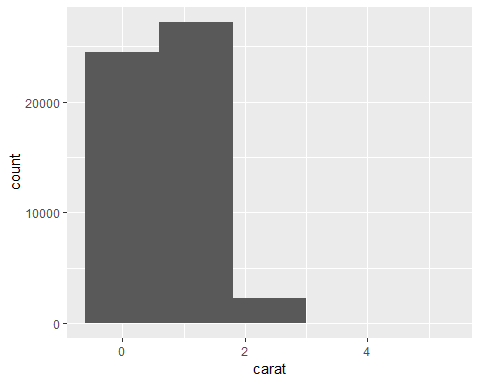
ggplot(data = diamonds, aes(x = carat)) + geom\_histogram()

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

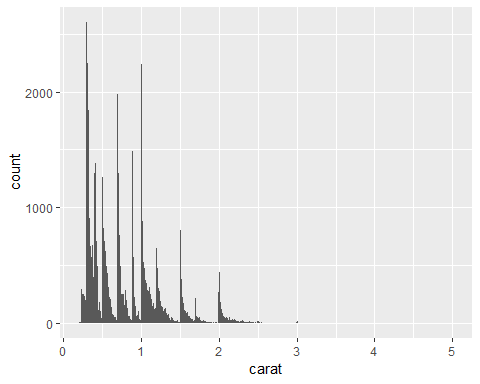


You can adjust the number of bars by setting the binwidth:

ggplot(data = diamonds, aes(x = carat)) + geom\_histogram(bins = 5)

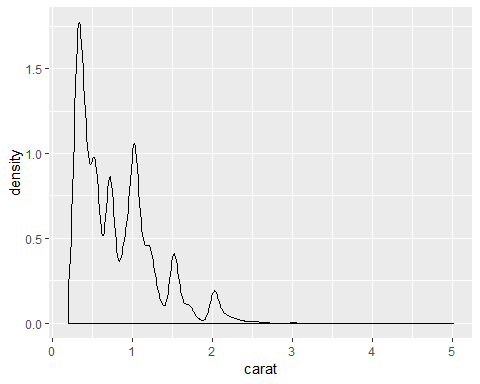


ggplot(data = diamonds, aes(x = carat)) + geom\_histogram(bins = 500)

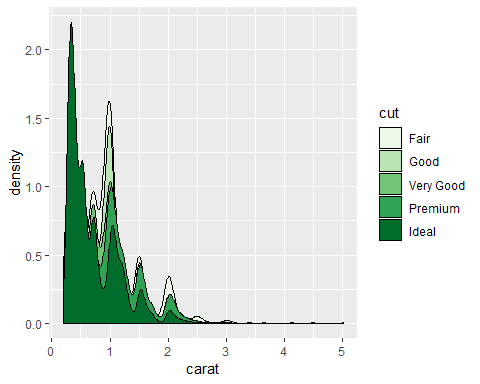


You could also use a density plot to display one continuous variable. Note what happens to the y-axis:

ggplot(data = diamonds, aes(x = carat)) + geom\_density()



ggplot(data = diamonds, aes(x = carat)) + geom\_density(aes(fill = cut)) +  
 scale\_fill\_brewer(palette = "Greens")

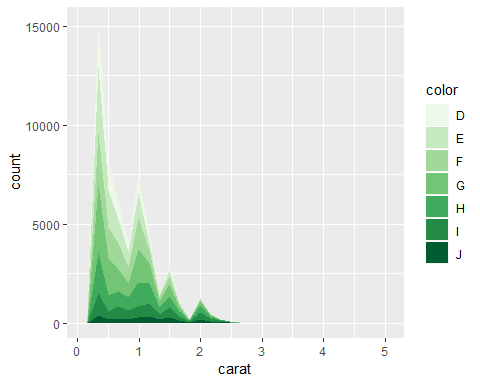


# note that for density plots, we need to use fill not colour - colour would just change the colour of the outline

Another option would be an area plot:

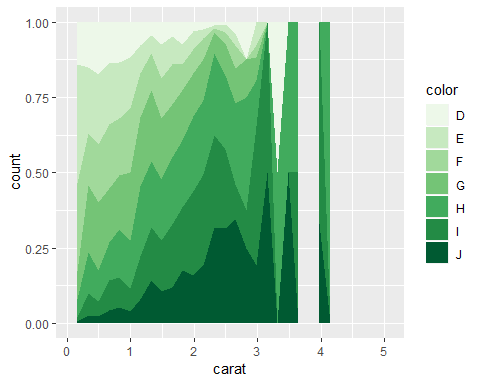
ggplot(data = diamonds, aes(x = carat)) + geom\_area(stat = "bin", aes(fill = color)) +  
 scale\_fill\_brewer(palette = "Greens")

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



ggplot(data = diamonds, aes(x = carat)) + geom\_area(stat = "bin", aes(fill = color), position = "fill") + scale\_fill\_brewer(palette = "Greens")

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

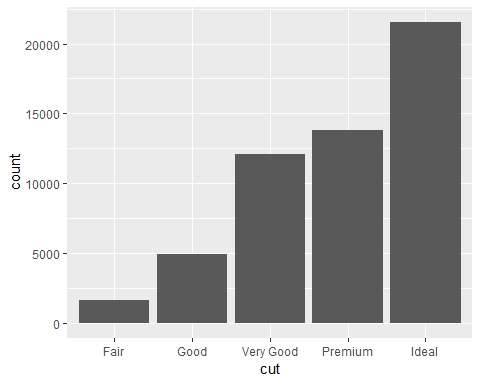


If you change position = "stack" (which is the default for geom\_area()) to position = "fill", the plot effectively shows a percentage on the y-axis.

### 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 = diamonds, aes(x = cut)) + geom\_bar()



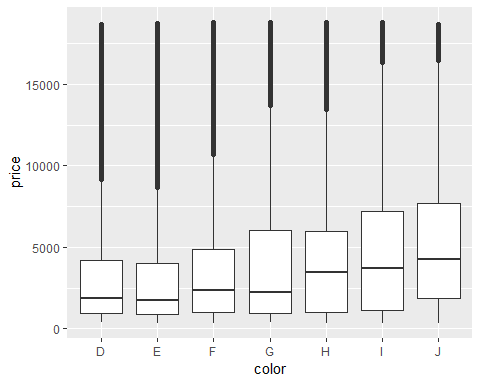
### Two continuous variables

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

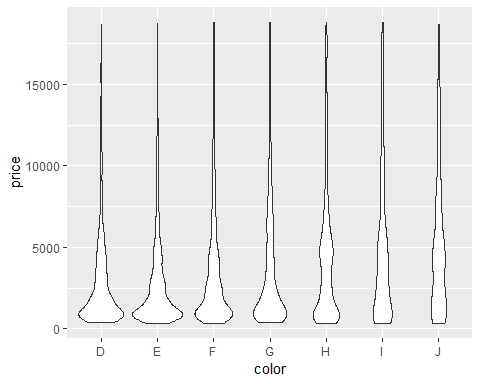
If you have a categorical and a continuous variable, you would normally put the categorical variable on your x-axis and the continuous variable on your y-axis. These are often displayed as box plots (they show the four quartiles in the data, and outliers as dots):

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_boxplot()



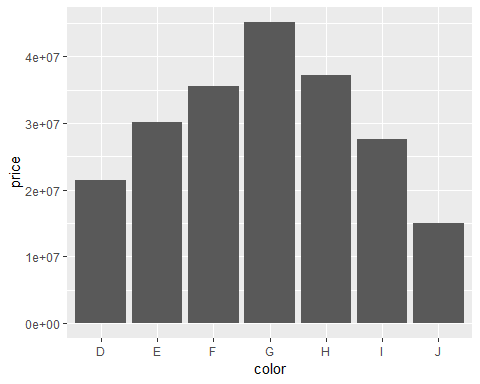
There are also other options in ggplot2 such as violin plots:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_violin(scale = "area")



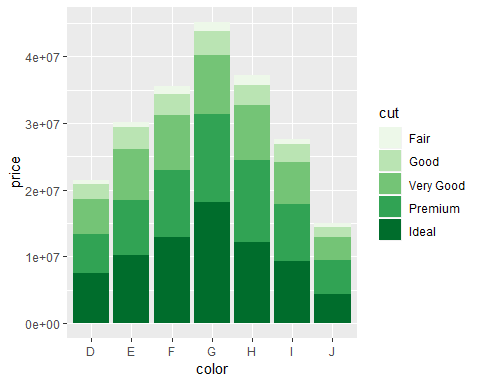
Or you could present your data as a bar plot. The default geom\_bar() option just uses a categorical variable for x. To show a categorical x and a continuous y variable, we need to add stat = "identity" here:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_bar(stat = "identity")



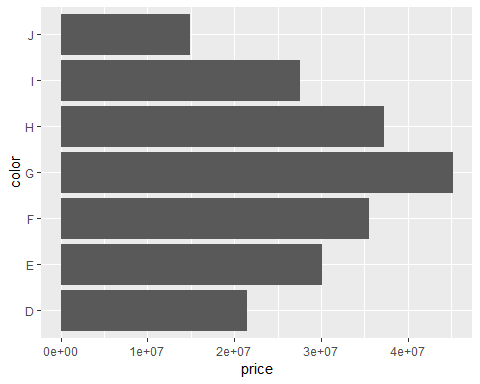
You can change the colour of the bar by another categorical variable:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_bar(aes(fill = cut), stat = "identity") +  
 scale\_fill\_brewer(palette = "Greens")



You can also flip the x- and y-axis:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_bar(stat = "identity") + coord\_flip()



## Adding error bars

To add error bars to a ggplot2 plot, we need to calculate the means and standard errors first. We will create a new dataset with the aggregate function. If you are unsure what your new data look like, use the str() or summary() functions:

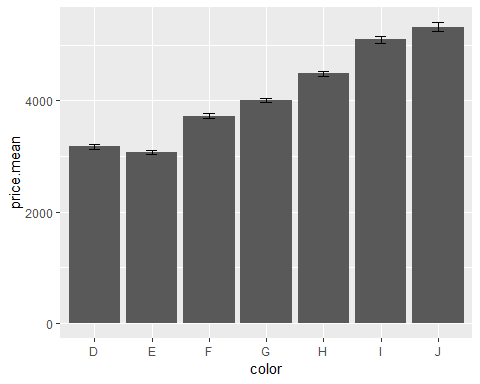
summaryd <- aggregate(price ~ color, diamonds, FUN = function(x) c(mean = mean(x), sd = sd(x), n = length(x)))

We are creating a new dataframe called summaryd. In it, we are summarising the continuous variable price by the color groups. We will get a mean with mean, standard deviation with sd and number of observations with length of price within each color group.

summaryd <- do.call(data.frame, summaryd)   
# this turns it into a dataframe  
  
summaryd$se <- summaryd$price.sd / sqrt(summaryd$price.n)

We add a new variable called se which is the standard deviation divided by the sample size - the standard error. Now we can plot the error bars:

ggplot(data = summaryd, aes(x = color, y = price.mean)) + geom\_bar(stat = "identity", position = "dodge") + geom\_errorbar(aes(ymin = price.mean-se, ymax = price.mean+se), width=0.2, position="dodge")



## Time series

To plot time series, we might have to change the format of our dataset. Have a look at the inbuilt dataset uspop:

str(uspop)

## Time-Series [1:19] from 1790 to 1970: 3.93 5.31 7.24 9.64 12.9 17.1 23.2 31.4 39.8 50.2 ...

uspop

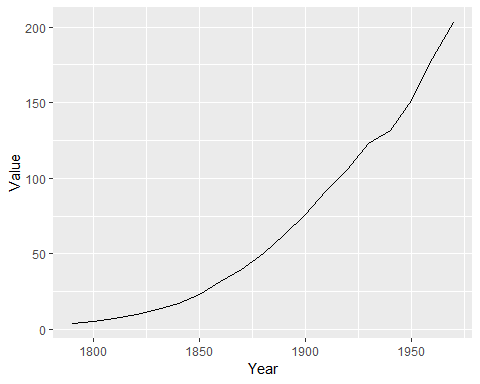
## Time Series:  
## Start = 1790   
## End = 1970   
## Frequency = 0.1   
## [1] 3.93 5.31 7.24 9.64 12.90 17.10 23.20 31.40 39.80 50.20  
## [11] 62.90 76.00 92.00 105.70 122.80 131.70 151.30 179.30 203.20

It is in a different format to the data we used so far. First we need to ensure that year and the values are both variables:

uspop2 <- data.frame(Year = c(time(uspop)), Value = c(uspop))  
  
# this turns the time values of uspop into the variable Year, and the other values into variable Value, in the new dataset uspop2

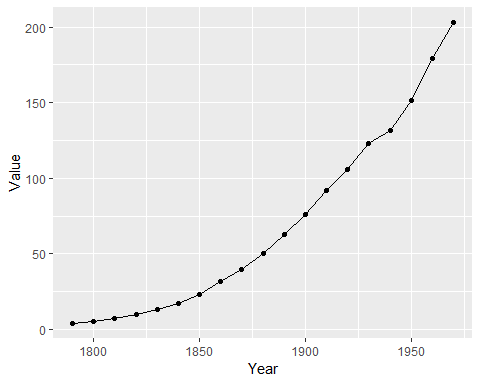
Now we can use the data to plot the population over time:

ggplot(data = uspop2, aes(x = Year, y = Value)) + geom\_line()



We can also add points for each year in addition to the line:

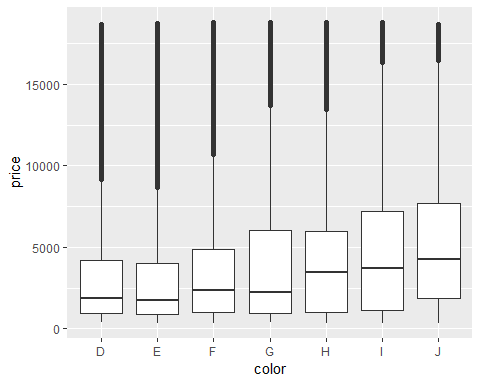
ggplot(data = uspop2, aes(x = Year, y = Value)) + geom\_line() + geom\_point()



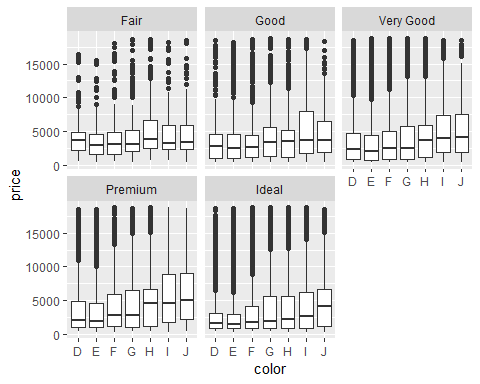
## Facets

One very useful feature of ggplot2 is the use of facets, where we split the data into subgroups, depending on a categorical variable, and make individual plots for each group. We can do this very easily by just adding a little bit of code. We will use our boxplot from earlier again to see how the facetting changes the plot:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_boxplot()

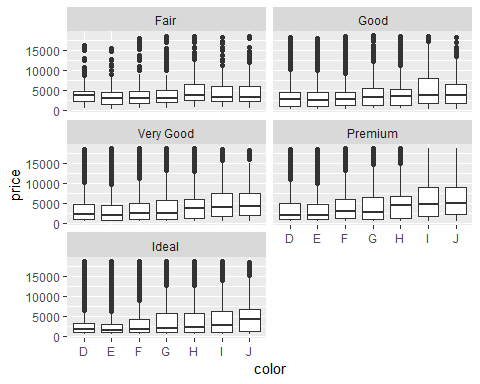


ggplot(data = diamonds, aes(x = color, y = price)) + geom\_boxplot() + facet\_wrap(~cut)



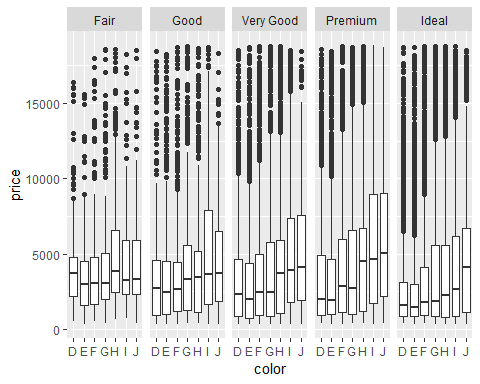
facet\_wrap() can be arranged by specifying the number of rows or columns with nrow or ncol:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_boxplot() + facet\_wrap(~cut, nrow = 3)

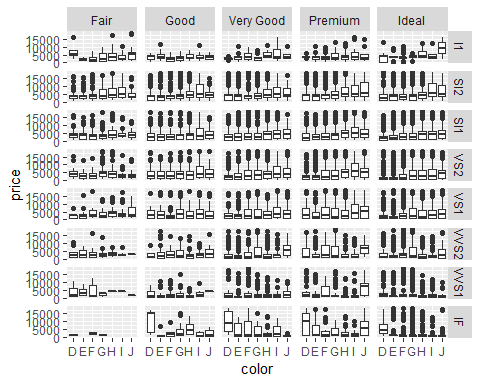


facet\_grid() on the other hand always arranges the plots in a grid:

ggplot(data = diamonds, aes(x = color, y = price)) + geom\_boxplot() + facet\_grid(~cut)



ggplot(data = diamonds, aes(x = color, y = price)) + geom\_boxplot() + facet\_grid(clarity~cut)

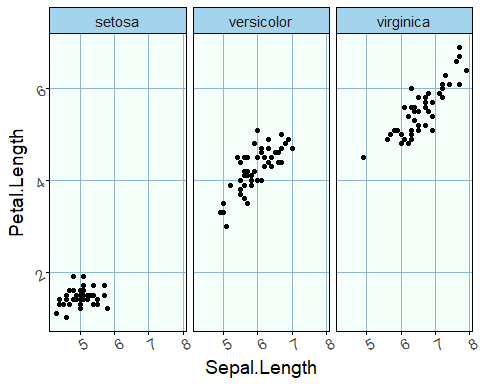


## A completely custom ggplot2 theme

Now you know how to make plots in ggplot2 with the default settings, you can start to make customised plots by changing the theme. This will not change how the data are displayed, but how the plot itself looks. It is useful if you want all plots in your thesis to have the same look.

Here is an example of a customised plot, with annotation for the code:

ggplot(data = iris, aes(x = Sepal.Length, y = Petal.Length)) + geom\_point() + facet\_wrap(~Species)+  
scale\_colour\_gradient(low = "green", high = "blue", name = "Petal width")+   
# this is similar to a plot we made earlier  
  
theme(text = element\_text(size = 14),   
# this changes the font size to 14  
  
axis.text = element\_text(angle=30, size = rel(0.8)),   
# this changes the angle of the axis labels (often useful for bar plots), and makes the font slightly smaller than the rest of the plot  
  
axis.ticks = element\_line(colour = "black"),  
  
plot.background = element\_rect(fill = "white"),   
#this is the colour behind the plot  
  
panel.background = element\_rect(fill = "mintcream"),   
# this is the colour inside the plots  
  
panel.border = element\_rect(fill = NA, colour = "black"),  
  
panel.grid.major = element\_line(colour = "lightskyblue3", size = .1),  
  
panel.grid.minor = element\_line(colour = "mintcream"),   
  
# the minor grid lines - same colour as each panel  
  
strip.background = element\_rect(fill = "lightskyblue2", colour = "black", size = 0.5))#this refers to the box with the heading above each facet



## 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)

## Additional packages

There are a lot of other packages in R now that add more functionality to ggplot2. Have a look at the ggplot2 extensions website (<http://www.ggplot2-exts.org/gallery/>) and see whether any of the packages might be useful for your own work. There are packages with more themes (for example ggthemes), a package for making animated graphs (gganimate), packages for plotting graph structures (for example ggraph) and so on.