Using R as a Research Tool.

Dr Susan Johnston: Susan.Johnston@ed.ac.uk

Demonstrators: Gergana Dalaskova, John Godlee.
Hat-Tips to Kyle Dexter, The Coding Club and R4all.

November 6, 2017

1 Introduction

1.1 What is R?

R began its life in New Zealand in 1993 as a language and environment for statistical computing and graphics. It is an interpreted programming language, meaning that rather than pointing and clicking, the user types in commands. It is **free** and works across all platforms.

1.2 Why use R?

```
"This is R. There is no if.  Only how."
-- Simon 'Yoda' Blomberg, R-help (April 2005)
```

Almost anything is possible in R. It is fast becoming the *lingua franca* of academic research and data science. It is used for:

- Processing and tidying data
- Statistical analyses
- Data visualistion (`ggplot`)
- Creating interactive web applications (`shiny`)
- Generating reports and presentations (`knitr`, `slidify`)
- Creating portable projects (RStudio Projects)

The analytical power of R lies in its many packages (11,172 at the time of writing). At least 300 of these are written for ecologists and evolutionary biologists. A list of packages are hosted on the Comprehensive R Archive Network (known as **CRAN**): `https://cran.r-project.org/`.

2 Getting Started: R and the RStudio Environment.

2.1 Installing R and RStudio

R can be downloaded from the CRAN website. Whilst the CRAN download version provides a simple user interface, we recommend that R is run through the software RStudio This is open-source, free, and available at `http://www.rstudio.com/`.

2.2 Creating an R Project.

Using R Projects (`.RProj`) allows easier file imports, improved reproducibility and collaboration. This is primarily because it tells R where to look for data files and scripts, meaning that a script can be run different machines and environments without any problems. We have provided you with the project `Intro_to_R.RProj`. Opening this file will open RStudio. If you would rather not use projects, you can set the working directory by using the command `setwd()` or by selecting `Session > Set working directory > Choose directory`.

On the Files tab in the **lower right** corner, you will see the files in the current working directory. This will be useful later when we tell R to load files. You can check the working directory by typing `getwd()`.

2.3 Using RStudio

Open `Intro_to_R.RProj` and open the example R Script (`1_Example_Script`). RStudio should look like Figure 1.

On the **lower left** is the **Console** pane - this is the engine of R. You can give instructions to R by directly typing at the prompt (`>`).

On the **upper left** is your R Script - here, you can write commands and send them to the console by clicking "Run" or by typing `Ctrl-Enter`. `#` is the comment character in R.

On the **lower right**, you can browse the packages installed on your machine, open files and search R Help. This pane will also show plots when we run them later in the practical.

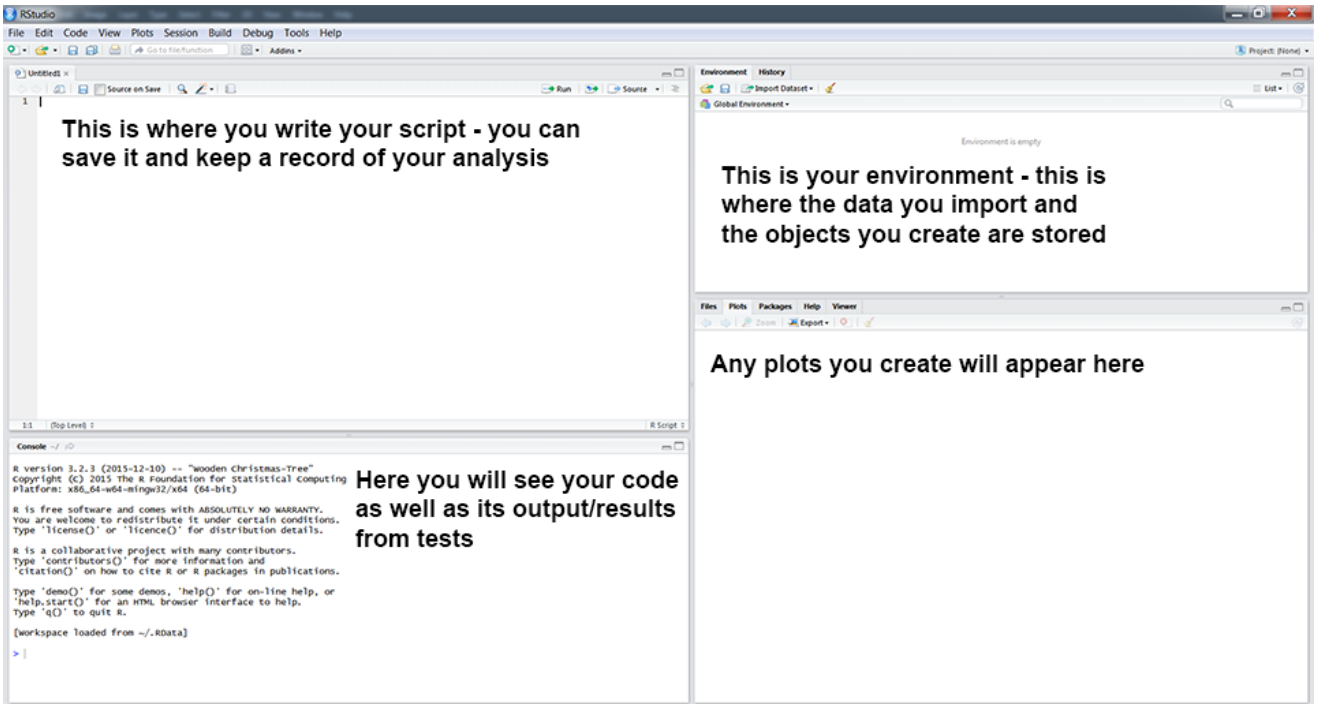


Figure 1: The RStudio Environment. Taken from OurCodingClub.io

Exercise 1.

Try running some basic commands directly in the console and from the R Script:

```
> 2+3
> 1:10
> seq(1, 20, 4)
> mean(c(3, 6, 9, 3, 6, 7))
```

Let's assign a sequence of numbers to an object, x:

```
> x <- 1:10
> x
> y <- seq(0, 4.5, 0.5)
> y
```

You can see that in the upper right pane, we can see this new objects x and y in the environment.

2.4 Finding Help within R.

The fastest way to find help in R is to search using `?`. For example:

```
> ?mean
```

should bring up a help page for the function `mean()` in the lower right corner. Typing two question marks will search all help files and return a list of those that match.

```
> ??mean
> ??"standard error"
```

Exercise 2.

1. Using only `?` and/or `??`, find a function for calculating the standard deviation. What is the standard deviation of `x`?
2. Using `?`, find the help file for the `sort()` function. Sort `x` and `y` in reverse order.

2.5 Troubleshooting and finding help outside of R.

- Coding Club Tutorials & Useful Links `https://ourcodingclub.github.io/`
- Stack Overflow `https://stackoverflow.com/`: Try searching with the tag [R]
- RStudio Cheatsheets `https://www.rstudio.com/resources/cheatsheets/`

3 Loading data into R.

Now that we are familiar with the RStudio environment, it's time to start working with real data.

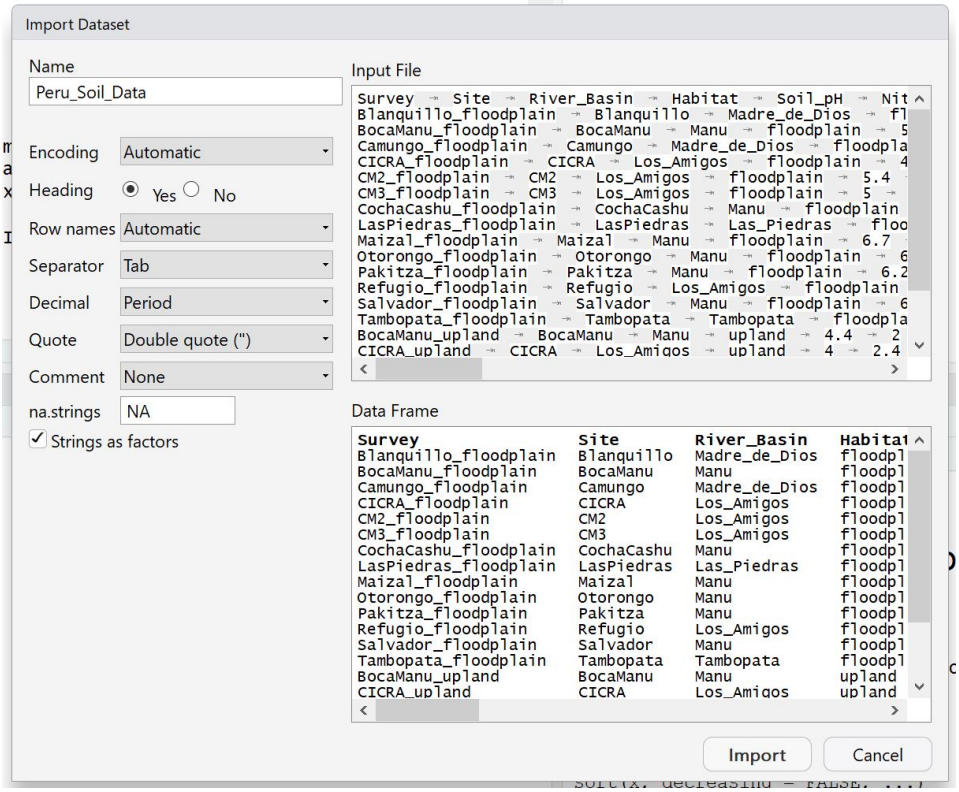


Figure 2: Importing data manually into **R**

In the folder `data`, you have been provided with a single dataset on Peruvian Soil in two common formats - `.txt` (tab-delimited) and `.csv` (comma-delimited).

The easiest way to read the data into **R** is to use the `Import dataset` button in the Environment tab, selecting `From Text (base)...` and choosing either data file (`.txt` or `.csv`). A box will appear (Figure 2). Check each of the options that applies to the data (i.e. change nothing) and click "`Import`". You will notice that the object `Peru_Soil_Data` is now in the **R** environment, but you may also have noticed a command appearing in the Console:

```
> Peru_Soil_Data <- read.delim("C:/Users/sjohns10/Google Drive/
+ Teaching and Seminars/201711 NERC E3 DTP/Intro_to_R/data/
+ Peru_Soil_Data.txt")
```

This command can be copied and pasted into your script, which would save you from clicking the next time. However, providing such a long path if problematic - if you were to

rename one of the directories, or move the folder somewhere else, then the code would no longer work.

R Projects provide the solution to this. Try typing the following into your script, and guiding the command to the data file using the Tab key:

```
Peru_Soil_Data <- read.delim("
```

You should now have the following code in your script:

```
> Peru_Soil_Data <- read.delim("data/Peru_Soil_Data.txt")
```

We recommend that instead of `read.delim()`, that you use either `read.table()` or `read.csv()`, as they offer more flexibility on defining various features about the input files.

Exercise 3.

Using `?`, find the help page for `read.table`. Read the file `Peru_Soil_Data.txt` into R. Check the loaded object using the `head()` function. Do you need additional arguments to read in the file properly?

The object `Peru_Soil_Data` is a type of object known as a **data frame**. You can explore the data visually by clicking on its entry in the Environment tab. Alternatively, there are functions in base R for exploring data e.g. `head()` and `str()`. Try these out.

4 Data management in R.

Exploring and manipulating data is fundamental to data analysis. In this section, we will briefly cover how to sort and filter the soil dataset. There are several approaches to doing this in the base code of R, but here we will use on the functions `select()`, `filter()` and `arrange()` from the package **dplyr**.

First, we need to load the `dplyr` package. It is good practice to do this at the beginning of the script when you are setting up the working environment.

```
> library(dplyr)
```

NB. Some of you may get an error message:
`Error in library(dplyr) : there is no package called 'dplyr'`

In this case, the library has to be installed from CRAN. This can be done by typing `install.packages("dplyr")`. It may ask you to select a mirror - select the RStudio Global mirror, or one that which is geographically closest to you.

4.1 Summary Statistics.

Functions such as `head()` and `str()` are useful to telling you what your data look like, but don't give much information on what the data say.

A versatile function for exploring objects is `summary()`. This function summarises each numeric column in terms of the median, mean, inter-quartile range, minimum and maximum. It also provides information on levels and sample sizes for categorical variables.

```
> summary(Peru_Soil_Data)

         Site           River_Basin       Habitat      Soil_pH
 BocaManu  : 2   Las_Piedras  : 2   floodplain:14   Min.   :4.000
 CICRA     : 2   Los_Amigos   : 8   upland    :11   1st Qu.:4.500
 CM2       : 2   Madre_de_Dios: 2                   Median :5.200
 CM3       : 2   Manu         :11                   Mean   :5.228
 CochaCashu: 2   Tambopata    : 2                   3rd Qu.:5.900
 LasPiedras: 2                                      Max.   :6.700
 (Other)   :13
 Nitrate_Nitrogen   Phosphorus       Calcium         Magnesium
 Min.   :0.00     Min.   : 1.80   Min.   :  39.0   Min.   : 10.5
 1st Qu.:0.50     1st Qu.: 3.50   1st Qu.: 104.5   1st Qu.: 22.0
 Median :1.40     Median : 4.30   Median : 543.0   Median :125.5
 Mean   :1.26     Mean   : 9.44   Mean   : 875.6   Mean   :127.7
 3rd Qu.:2.00     3rd Qu.: 6.50   3rd Qu.:1853.0   3rd Qu.:226.0
 Max.   :2.40     Max.   :57.00   Max.   :2292.0   Max.   :323.0
```

```
   Potassium         Sodium          Manganese
 Min.   :14.5   Min.   :  3.50   Min.   :  5.0
 1st Qu.:27.0   1st Qu.:  5.00   1st Qu.: 15.0
 Median :35.5   Median :  7.50   Median : 24.5
 Mean   :42.7   Mean   : 13.05   Mean   : 33.9
 3rd Qu.:63.1   3rd Qu.: 11.00   3rd Qu.: 44.1
 Max.   :88.5   Max.   :136.00   Max.   :103.5
```

4.2 Sorting data with `arrange()`

There are occasions where it is useful to have sorted data, either because we would like to examine it, or for some types of statistical analyses i.e. with time-series data. The `arrange()` function sorts data frames as so:

```
>    # sort by Soil pH value:
> arrange(Peru_Soil_Data, Soil_pH)
>    # sort by decreasing Soil pH value:
> arrange(Peru_Soil_Data, -Soil_pH)
>    # sort by habitat and then soil pH within habitat:
> arrange(Peru_Soil_Data, Habitat, Soil_pH)
```

4.3 Sub-setting columns with `select()`

At its simplest, columns can be selected using their numeric references in square brackets (**after** the comma):

```
> Peru_Soil_Data[,1]
> Peru_Soil_Data[,3:5]
> Peru_Soil_Data[,c(1, 4, 5)]
```

It is not always recommended to use numerical references, as addition or removal of columns can change the numbers, leading to mistakes. The best solution is to use the column names themselves. A convenient way to do this is to used the `dplyr` `select()` function, will select or remove columns of the data. Try running:

```
> select(Peru_Soil_Data, River_Basin)
> select(Peru_Soil_Data, -River_Basin)
```

More than one column can be selected or removed by adding more column names:

```
> select(Peru_Soil_Data, River_Basin, Magnesium, Sodium)
> select(Peru_Soil_Data, -River_Basin, -Magnesium, -Sodium)
```

4.4 Adding columns.

New columns can be added to the data containing information or calculations that you are interested in. We can do this in standard base **R** using the code `$`, which can be used to call existing variables from a data frame. For example, river basin can be called as so:

```
> Peru_Soil_Data$River_Basin

 [1] Madre_de_Dios Manu          Madre_de_Dios Los_Amigos    Los_Amigos
 [6] Los_Amigos    Manu          Las_Piedras   Manu          Manu
[11] Manu          Los_Amigos    Manu          Tambopata     Manu
[16] Los_Amigos    Los_Amigos    Los_Amigos    Manu          Las_Piedras
[21] Manu          Manu          Los_Amigos    Manu          Tambopata
Levels: Las_Piedras Los_Amigos Madre_de_Dios Manu Tambopata
```

Columns can be added by creating new variables within the data frame. Here, we can create a new column called log_Calcium which takes the log_{10} of the calcium column:

```
> Peru_Soil_Data$log_Calcium <- log10(Peru_Soil_Data$Calcium)
> head(select(Peru_Soil_Data, Site, Calcium, log_Calcium))

        Site Calcium log_Calcium
1 Blanquillo   864.0    2.936514
2   BocaManu  2260.5    3.354205
3    Camungo  1853.0    3.267875
4      CICRA   861.3    2.935154
5        CM2  1473.0    3.168203
6        CM3   524.5    2.719745
```

4.5 Sub-setting rows with `filter()`

Sub-setting data by rows is one of the most common tasks carried out in data manipulation steps. Again, at it's simplest, rows can be selected using their numeric references in square brackets (**before** the comma):

```
> Peru_Soil_Data[1,]
> Peru_Soil_Data[1:5,]
```

However, this is clearly not useful if we wish to select rows based on a particular criteria. For this, we can used the `filter()` function, specifying an argument with the following logical operators:

Operator	Function
$<$	less than
$>$	greater than
$=<$	less than or equal to
$=>$	greater than or equal to
$==$	equals
$!=$	does not equal
$\%in\%$	matches

For examples, in the soil data, we may wish to select only rows for the floodplain habitat and the Los Amigos River Basin:

```
> filter(Peru_Soil_Data, Habitat == "floodplain", River_Basin == "Manu")
```

Exercise 4.

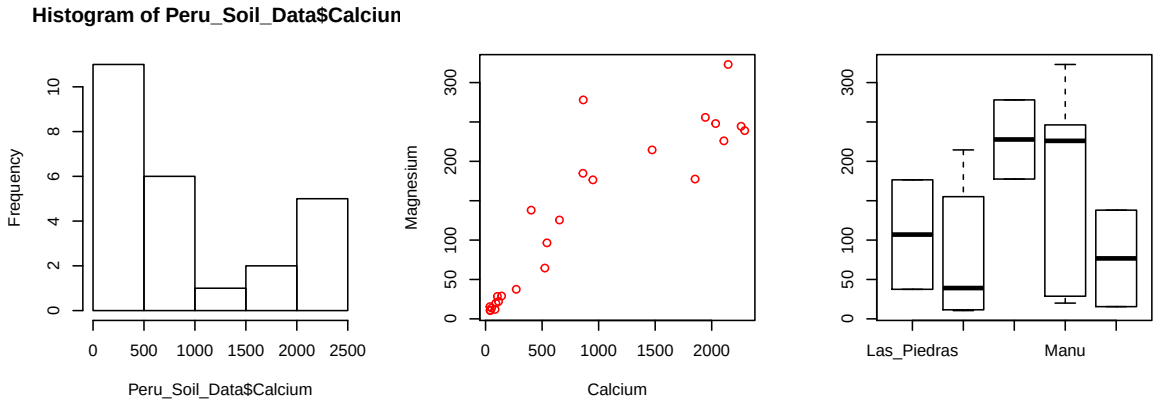
1. Create a new data frame, Peru_Upland_Soil, which includes row only from upland habitats.
2. Edit this data frame so that it only includes data from the Manu and Los Amigos river basins (Hint: use $\%in\%$)
3. Edit this data frame again so that it is sorted by increasing Calcium levels.
4. Create a new column called Sum_Ca_Mg that is the sum of the calcium and magnesium columns.

5 Data visualisation with `ggplot2`

One of **R**'s most powerful tools is its ability to produce publication quality graphics in an automated and reproducible way. Base **R** contains graphical tools for simple, exploratory graphics. These are easily called with functions such as `hist()` and `plot()`:

```
>    # histogram
> hist(Peru_Soil_Data$Calcium)
>    # plot
> plot(Magnesium ~ Calcium, data = Peru_Soil_Data, col = "red")
>    # boxplot
> boxplot(Magnesium ~ River_Basin, data = Peru_Soil_Data)
```

It is feasible to do anything you require in base graphics, but common actions are not straightforward, including legends, faceting, error bars and customisations of the plot area. Base graphics for quick data exploration, but for everything else we recommend the graphics library `ggplot2`.



5.1 `ggplot2` - the grammar of graphics.

`ggplot2` is a plotting system in **R** which aims to take the best parts of base graphics and allow for easy plot customisation. It has extensive documentation and examples at `docs.ggplot2.org/current/`.

`ggplot2` uses **three** components to construct a graph:

1. **Layers:** `ggplot()` Data with aesthetic properties (`aes()`).
2. **Geoms:** `geom_...()` The type of plot (line, scatter, box-plot, etc.)
3. **Stats:** `stat_...()` Statistical transformations (smoothing lines, etc) - NB. every geom has a default statistic, so this component is not always specified.

All `ggplot2` graphs begin with the `ggplot()` function, specifying the default data and aesthetic properties. This is just a fancy way of saying that we specify the data frame, and then the x and y variables we want to plot.

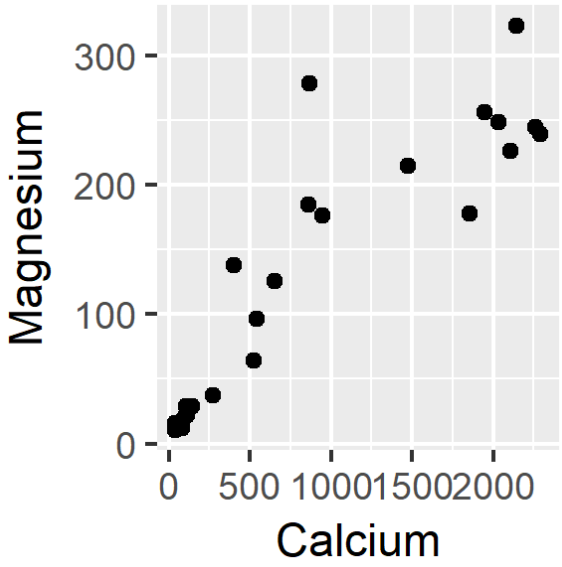
5.2 `ggplot2` basics - a scatter-plot.

We can start with a simple scatter-plot of Calcium and Magnesium in the Peru soil dataset. We start by defining the **layer**.

```
> library(ggplot2)
> ggplot(Peru_Soil_Data, aes(x = Calcium, y = Magnesium))
```

You can see by running the layer alone, we have already run into a problem - there is no data plotted on the graph. This is because we need to specify the **geom** i.e. the type of plot. Let's use a scatter-plot (`geom_point()`):

```
> ggplot(Peru_Soil_Data, aes(x = Calcium, y = Magnesium)) +
    geom_point()
```



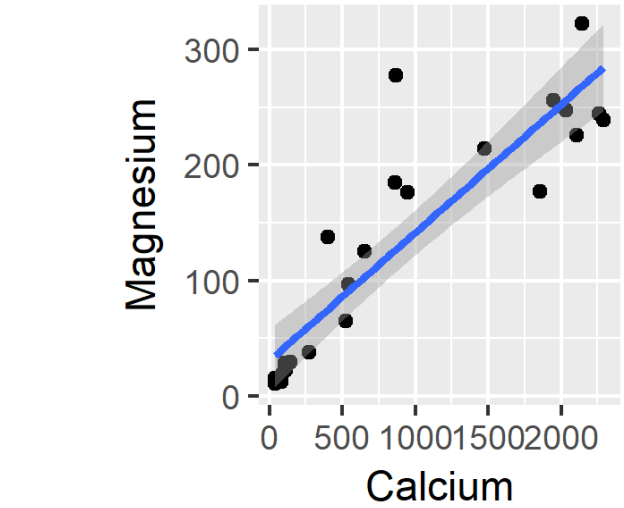
If you wanted to add a regression line, we can specify this with a **Stat**. In this case, we can use `stat_smooth(method = "lm")` to add a linear regression line:

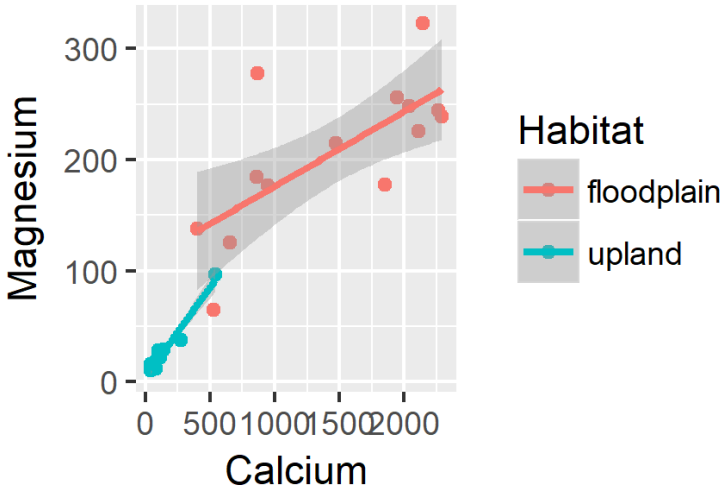
```
> ggplot(Peru_Soil_Data, aes(x = Calcium, y = Magnesium)) +
    geom_point() +
    stat_smooth(method = "lm")
```

Adding customisations to the plot is straightforward. For example, colouring the points by habitat is done by adding `col = Habitat` to the `aes()` properties:

```
> ggplot(Peru_Soil_Data, aes(x = Calcium, y = Magnesium,
    col = Habitat)) +
    geom_point() +
    stat_smooth(method = "lm")
```

As you can see, this automatically adds a legend to the plot for the different habitat types.



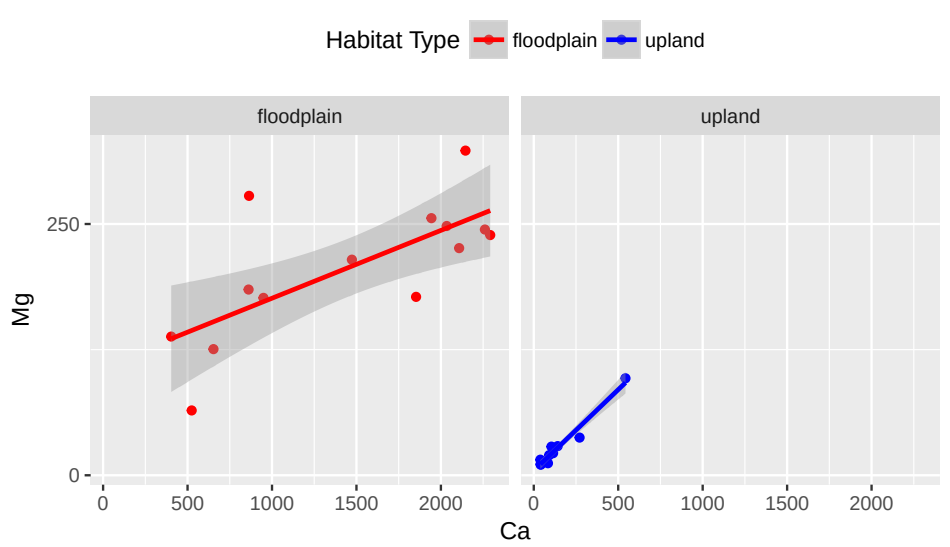


5.3 Customising plots.

`ggplot2` has huge capabilities for plot customisations. The sheer amount of things you can do to the graph can be overwhelming (see the help pages), but most commonly, we will want to change labels, scales, colour palettes and facets. Here are a variety of customisations you can play with, and an example graph is given below.

Function	What it does
`labs()`	Customise the labels associated with aes() e.g. `labs(x = "Ca", y = "Mg", col = "Habitat Type")`
`scale_x_continuous()` `scale_y_continuous()`	Change the scale of the axes e.g. `scale_x_continuous(breaks = seq(0, 2500, 250))`
`facet_wrap()`	Facet the plot by a discrete variable e.g. `facet_wrap( Habitat)`
`scale_colour_manual`	Example of one of many ways to define colours e.g. `scale_colour_manual(values = c("red", "blue"))`
`theme()`	Formatting options for axes, plot area, etc. e.g. `theme(legend.position = "top")` (see documentation for **many, many** options)
`Default themes`	`theme_bw()`, `theme_light()`, `theme_dark()`, `theme_minimal()`, etc.

```
> ggplot(Peru_Soil_Data, aes(x = Calcium, y = Magnesium, col = Habitat)) +
+   geom_point() +
+   stat_smooth(method = "lm") +
+   labs(x = "Ca", y = "Mg", col = "Habitat Type") +
+   scale_y_continuous(breaks = seq(0, 2500, 250)) +
+   facet_wrap(~Habitat) +
+   scale_colour_manual(values = c("red", "blue")) +
+   theme(legend.position = "top")
```



If you have questions when it comes to customisation, consult the ggplot2 documentation for the types of graph you wish to make, or search Stack Overflow with the tags [r] and [ggplot2].

5.4 Saving plots

Plots can be saved to file using `ggsave()`. It defaults to saving the last plot that you displayed, using the current size of the graphics window. The type of graphic that is saved depends on the extension you define (e.g. .png, .pdf, etc.). At it's simplest, it can be run as so:

```
> ggsave("Habitat_Ca_MG.png")
```

Other specifications can be made, such as plot height, width, resolution and so on:

```
> ggsave("Habitat_Ca_Mg.png", width = 12, height = 8, units = "cm", dpi = 300)
```

Exercise 5.

1. Create a scatter-plot of Calcium against Potassium. Colour the points based on the Soil pH.
2. Using the ggplot2 help pages, make a box-plot of River Basin (x-axis) and Soil pH (y-axis). Facet the plot by habitat.
3. Using the ggplot2 help pages, make a bar-plot of Site (x-axis) and Sodium (y-axis). (NB - Getting an error? Read the help pages carefully!)
4. Colour the bars in the previous plot by River Basin (Hint: swap `col` for another command)