INTRODUCTION TO R

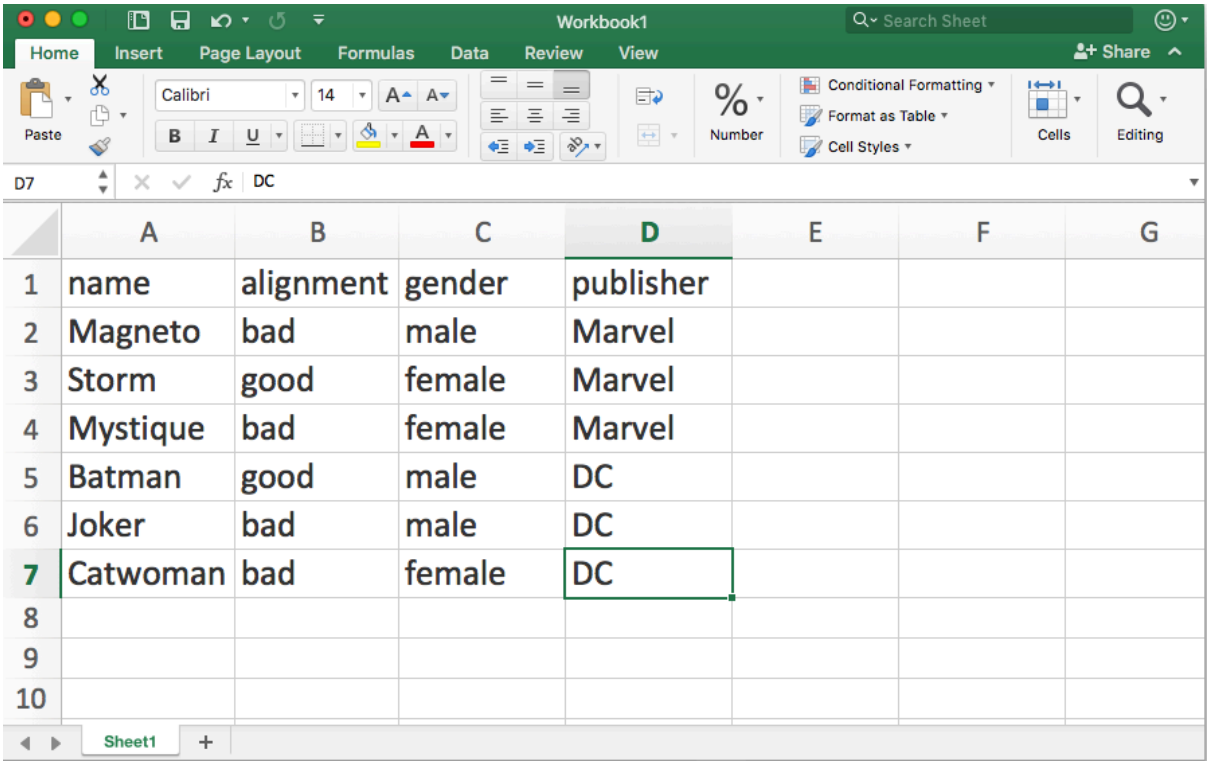
Maria Christodoulou

Part I - Importing data

Often you will record your data using more user-friendly software (such as Excel) for data collection. Importing data into R is relatively straightforward following some basic steps. **The following instructions are suitable for windows users. If you are using Linux you will find any additional instructions in brackets and italics.**

First, open excel and create a spreadsheet *(Linux users: open LibreOffice Calc, instead of excel)*. Below is an example using the superheroes dataset from Jenny Bryan -follow @JennyBryan on twitter for excellent advice and tutorials on all things R.

Give a header for each column (row 1 of the spreadsheet) and populate the following cells with appropriate values.



	A	B	C	D	E	F	G
1	name	alignment	gender	publisher			
2	Magneto	bad	male	Marvel			
3	Storm	good	female	Marvel			
4	Mystique	bad	female	Marvel			
5	Batman	good	male	DC			
6	Joker	bad	male	DC			
7	Catwoman	bad	female	DC			
8							
9							
10							

Sheet1

Note a few things that will help you when importing:

- There are no spaces in the entries, if you need to have more than one words per cell, just merge them. For example, if we were to add a row for "Hellboy" we would record publisher as "DarkHorseComics".

- There are no special characters. If you need to use some, the ones that are OK to use are underscore and full stop.

- In the example spreadsheet we have populated 7 rows and 4 columns. Within this 7 by 4 box we cannot have any empty cells as this will cause problems when importing. If you have missing values for any cell, instead of leaving it blank use NA.

Save the spreadsheet normally as an .xls file for your records *(Linux users: save the file as ODF for your records)*, this is not the copy we will be using to import but it's always good to backup. Next use the "Save as" option to save the file as a tab delimited .txt file in an appropriate location on your hard drive *(Linux users: save the file as a text csv, then in the export text file window select {Tab} under the "field delimit" option)*. This is the version we will be importing so you would normally store this in the same folder as you would store all your R scripts for a particular analysis.

Next go to RStudio and make sure your directory is set to the folder that contains the .txt *(Linux users: .csv file)* version of the spreadsheet. The reason you are doing this is to avoid typing the exact location for the files you want to load. It is good practice to create a folder for each experiment you are conducting where all the scripts and analyses are in one place for you to find.

To set your directory in RStudio: go to "Session" > "Set working directory" > "Choose Directory" and select the right folder.

Once your directory has been set you can use `read.table()` and the file name to import it. *(Linux users: replace "superheroes.txt" with "superheroes.csv" in the line below)*

```
> superheroes<-read.table("superheroes.txt", header=TRUE)
>
> superheroes
      name alignment gender publisher
1  Magneto       bad   male    Marvel
2    Storm      good female    Marvel
3 Mystique       bad female    Marvel
4   Batman      good   male        DC
5    Joker       bad   male        DC
6 Catwoman       bad female        DC
>
```

In the above example, the imported table was assigned the name "superheroes" and then I used the `read.table()` to import the .txt file (in quotes, with file extension included). The parameter `header=TRUE` tells R that the first row of the new table is the header.

Part II – Adding rows and columns to a data frame

To add extra observations to your data frame you can use the `rbind()` function. In the example below we create a data frame for the new row for Hellboy (alignment: good, gender: male, publisher: Dark Horse Comics) and then bind it with the superheroes dataset.

```
> super2<-data.frame(name="Hellboy",alignment="good", gender="male", publisher="DarkHorseComics")
> super2
     name alignment gender       publisher
1 Hellboy      good   male DarkHorseComics
> superheroes<-rbind(superheroes,super2)
> superheroes
      name alignment gender       publisher
1  Magneto       bad   male          Marvel
2    Storm      good female          Marvel
3 Mystique       bad female          Marvel
4   Batman      good   male              DC
5    Joker       bad   male              DC
6 Catwoman       bad female              DC
7  Hellboy      good   male DarkHorseComics
>
```

To add a column with an extra variable, use `cbind()` in the same way. First create the variable as a data frame, with a variable name (e.g. in this case "height") and add the values in order (so Magneto's height first, then Storm, then Mystique and so on)

```
> super.tall<-data.frame(height=c(1.83,1.80,NA,1.88,1.95,1.70,2.10))
>
> super.tall
  height
1   1.83
2   1.80
3     NA
4   1.88
5   1.95
6   1.70
7   2.10
> superheroes<-cbind(superheroes,super.tall)
> superheroes
      name alignment gender       publisher height
1  Magneto       bad   male          Marvel   1.83
2    Storm      good female          Marvel   1.80
3 Mystique       bad female          Marvel     NA
4   Batman      good   male              DC   1.88
5    Joker       bad   male              DC   1.95
6 Catwoman       bad female              DC   1.70
7  Hellboy      good   male DarkHorseComics   2.10
```

Part III - Indexing

Selecting objects can be done through indexing, for example if you wanted to select the 4th row (Batman):

```
> superheroes[4,]
    name alignment gender publisher height
4 Batman      good   male        DC   1.88
>
```

or the 3rd column (gender):

```
> superheroes[,3]
[1] male   female female male   male   female male
Levels: female male
>
```

You could select the 4th row and 3rd column simultaneously if you wanted to simply select Batman's gender.

```
> superheroes[4,3]
[1] male
Levels: female male
>
```

You could use vectors to select all the columns and rows you wanted. For example, you can select the first 3 columns for rows 1,3,5 and 7.

```
> superheroes[seq(1,7,by=2),1:3]
      name alignment gender
1  Magneto       bad   male
3 Mystique       bad female
5    Joker       bad   male
7  Hellboy      good   male
>
```

More advanced: you can use column names and logical tests. For example, to select all the male superheroes:

```
> superheroes[superheroes$"gender"=="male",]
     name alignment gender       publisher height
1 Magneto       bad   male          Marvel   1.83
4  Batman      good   male              DC   1.88
5   Joker       bad   male              DC   1.95
7 Hellboy      good   male DarkHorseComics   2.10
>
```

Notice the syntax:

1. first you declare the dataset you want (superheroes),
2. then you use the square brackets of indexing
3. within that you declare the condition for which rows to keep (remember, in data frame and matrix indexing, rows go before the comma and columns after),
4. followed by a comma and the closing square bracket since you want to select all columns.

You can practice these skills further by using some of the built-in datasets, such as `iris`, or `mtcars`. Try different ways to select parts of the data frame, see what works and what doesn't, and importantly – when things don't work try to understand where you went wrong.

Part IV - Functions and packages

Use the help (and online resources if you think the help page is not helpful) to see how some basic functions work such as: `mean()`, `median()`, `max()`, `min()`, and `range()`.

In the examples below I have used the above functions on the sepal lengths from the `iris` dataset.

```
> mean(iris$Sepal.Length)
[1] 5.843333
> median(iris$Sepal.Length)
[1] 5.8
> min(iris$Sepal.Length)
[1] 4.3
> max(iris$Sepal.Length)
[1] 7.9
> range(iris$Sepal.Length)
[1] 4.3 7.9
>
```

Try them on another variable, or combine indexing and these functions for calculations on specific groups. For example, what is the mean of sepal lengths just for the *I. setosa* samples?

Start thinking of your own projects. What are you likely to be using R for? Look through CRAN Task views for inspiration about possible packages and techniques that may be useful to you in the future.

Part V – Randomness and sampling

Use the `sample()` to select 10 numbers from a sequence of your choice (it can be numbers from one to one thousand, even numbers between 20 and 60 – `seq(20, 60, 2)`, years from 1917 until 2017 – `seq(1917, 2017, 1)`, Summer Olympic years, World Cup years, get as creative as you wish!)

Repeat the process to select more or less numbers, as you wish. Repeat the process with or without replacement.

Use `rnorm()` and `rbinom()` to simulate from the normal and binomial distributions using parameters of your choice. If uncertain how to do it check the help pages for hints.

Part VI – Graphics

Use the `plot()` function to plot the sepal length vs the sepal width for the iris dataset. Add labels to your axes, a title and change the colour and shape of your points. For an added challenge, use a different colour for each species. Try to add a legend using the `legend()` function, for each species. Look through the parameter options for graphics (type `?par` in the console) for extra options.

If you have gone through these and would like to go a bit further, look through last year's practical in the following pages and give it a go.

R PRACTICAL

DAVID STEINSALTZ

(First two based in large part on problems in the MSc course by Robin Evans.)

(1) Create the following vectors in R using `seq()` and `rep()`.
   (a) $1, 1.5, 2, \ldots, 12$.
   (b) $1, 8, 27, \ldots, 1000$.
   (c) $1, -\frac{1}{2}, \frac{1}{3}, -\frac{1}{4}, \ldots, -\frac{1}{100}$.
   (d) $1, 0, 3, 0, 5, 0, \ldots, 49$.
   (e) $1, 9, 36, 100, \ldots, \sum_{i=1}^{k} i^3, \ldots, 44100$. [look up `?cumsum`]
   (f) $1, 3, 3, 5, 5, 5, \ldots, \underbrace{19, \ldots, 19}_{10 \text{ times}}$.

(2) The ith term in the Taylor series expansion of $\log(1+x)$ is $(-1)^{i+1}x^i/i$.
   (a) Create a vector containing the first 100 terms for $x = 0.5$.
   (b) Let
$$r_n(x) = \log(1+x) - \sum_{i=1}^{n} \frac{(-1)^{i+1}x^i}{i}\,.$$
   Evaluate $r_n(1)$ for $n = 10, 100, 1000, \ldots, 10^6$.

(3) Make a plot of the graphs of the functions $y = x^p$ for $p = \frac{1}{4}, \frac{1}{2}, 1, 2, 3$, and $x \in [0, 2]$, all on the same set of axes, with different colours and line types. For an extra challenge use the function `legend` to add a legend that explains the plot.

(4) Load the package `MASS` with the command `require(MASS)`. The data frame `hills` gives record times in 1984 for 35 Scottish hill races.
   (a) Look at the help file for this data set.
   (b) Try applying commands like `head`, `summary`, `dim`, `attributes`.
   (c) What happens when you plot `hills`?
   (d) Make a histogram of the `time` variable.
   (e) Compute the mean and SD of the times for those races where the climb was above the median, and those where it was below the median.

(5) Type `load(url('http://steinsaltz.me.uk/DTC/abdata.RData'))` to load two vectors, named `a` and `b`. Think of `a[i]` and `b[i]` as being measurements of two different quantities for a given patient i.
   (a) Use the functions `length`, `summary`, and others you know to figure out what you can about these vectors.
   (b) Look up the function `unique`. Use it to determine how many individuals have identical values of `a`. And of `b`.
   (c) Make new vectors `x` and `y`, containing in place i the larger of $b[i]$ and $e^{a[i]}$, and the smaller of $b[i]$ and $e^{a[i]}$, respectively
   (d) `summary` gives you the quartiles of the vector; that is, the values that split up the range of the vector into equal quarters. Make new vectors `a1` through `a4`, consisting of the values of `a` corresponding to the patients whose `b` value is in the first, second, third, and fourth quartiles respectively.
   (e) Investigate how spread out these four vectors are, and their means and medians. Does it seem that there is a connection between an individual's `a` and `b` values? Why or why not?

(f) Look up the `sort` and `order` functions. Use them to create a new vector `aSb` which contains the values of `b`, ordered by `a`. That is, the first element is the value `b[i]`, where i is the index of the patient with the smallest value of `a`. Next the `b[i]` corresponding to the second smallest value of `a`. And so on.