# MATH3091 Computer Lab 1

31 Jan 2022

## `R` and `RStudio`

As in MATH2010, we will use `R` to analyse data in MATH3091. In this lab, we will review some basic `R` commands and best practices, and download all the datasets needed for later worksheets.

`RStudio` is one *interface* to `R`: it provides a nice editor for `R` scripts, and also has panels for the console (where your `R` code is run, and where you view the output), the environment (to see which variables you have already defined), and for viewing files, plots and help. You can download both `R` and `Rstudio` for free onto your own computer. We recommend using `Rstudio` as a nice interface to `R`, but all of the code we will write could be run through any interface to `R`.

## Preparing for the MATH3091 computer labs

Download the file `MATH3091_labs.zip` from Blackboard, and unzip the contents somewhere sensible on your computer. We will use the folder `MATH3091_labs` for the computer labs throughout the course.

You will need to tell `R` to use your `MATH3091_labs` folder as the working directory. There are various ways to do this. In `Rstudio`, perhaps the easiest is to type `Ctrl+Shift+H`, and navigate to `MATH3091_labs` from there. Alternatively, you could use

```r
setwd("<path>/MATH3091_labs")
```

where you should replace `<path>` with the path to the `MATH3091_labs` folder on your computer.

Create a new `R` script and save it as `lab1.R` in the `MATH3091_labs` folder. You should record the work you do in each practical session in a new `R` script. Write code in the script file, rather than directly into the console, whenever you can. In `RStudio`, you can press the `Run` button to run the line with the cursor in it. You can run multiple lines at once by selecting them all before pressing `Run`.

## Reading data into `R`

There are various ways that a dataset might be stored on your computer.

In MATH3091, most of our datasets will be stored as *comma separated value* (or `csv`) files. An example is the file `nitric.csv`. Try opening `nitric.csv` outside of `RStudio`. On most computers it will open as a spreadsheet, for example in Excel. This also gives you a way to import spreadsheet data into `R`, as you can save simple spreadsheets in `csv` format.

To read a `csv` file into `R`, we can use `read.csv`. For example, add the following to your `R` script, and run the line:

```r
nitric <- read.csv("nitric.csv")
```

We have read in the file `nitric.csv`, and stored the result in a variable called `nitric`.

Always try to give your variables meaningful names (avoid using names like `a`, `b`, `x`, unless those names actually mean something in the context of your example). This makes things much easier for someone else to read your `R` code, and makes it much easier for you too, especially if you come back to your own code again after a few weeks!

We can run

```r
nitric
```

to view the data set we have just imported, or

```r
head(nitric)
```

just to view the first few rows. The `head` command is useful for larger datasets, to check what we have imported looks reasonable, without trying to print out all of the data.

Another way to make your code readable is to use comments in your `R` script, by putting a `#` character at the start of a line. `R` won't try to run anything you write on these lines, and they allow you to explain (to someone else reading your code, or to your future self) what you are doing. Add some comments to your `R` script so far.

Your `R` script might now look something like:

```r
# read in a dataset from a csv file:
nitric <- read.csv("nitric.csv")

# view the dataset:
nitric

# view just the first few rows of the dataset:
head(nitric)
```

## Getting help

`R` has a large range of built-in functions, and each of these has a help file. Some functions have a large number of arguments you could potentially specify, but most have default values. The `Usage` section tells you the arguments to the function, and what the default values are. The `Examples` section right at the end of a help file is often very useful.

If we wanted to know more about the `head` function we just used, we could check the help

```
?head
```

**Question:** How many rows of `nitric` were printed by default by `head(nitric)`? How could you change that?

It is important for you to be able to work out for yourself how to use a function: there are huge numbers of functions available in `R`, and even very experienced `R` users won't know how to use all of them. In the worksheets, you won't always be given all the code: you should be prepared to check the help files to work out what you need to do. If you don't know which function to use, try searching for what you are trying to achieve online: there is a large community of `R` users worldwide, and it is likely that someone else has already solved a similar problem.

## Working with data frames

Our data `nitric` is stored as a data frame in `R`. We can acess each of the four variables by name using `$`, for example:

```
nitric$yield
```

We can also access the data by row or column.

**Question:** What does each of the following commands do?

```
nitric[, 1]
```

```
nitric[2, ]
```

```
nitric[1:5, ]
```

```
nitric[, c(2, 4)]
```

```
nitric[-1, ]
```

We sometimes want to consider a subset of the data which meets a particular condition. Use the `subset` function to find a new data frame containing all the records with `temp` $> 20$.

## Plotting data

Suppose we want to plot `yield` against `temp`. There are various ways to do this. The most basic is

```
plot(nitric$temp, nitric$yield)
```

**Question:** Give this plot sensible axis labels. How would you change the colour of the plotting points? How would you change the shape of the plotting points?

Equivalently, we can use R's formula interface:

```r
plot(yield ~ temp, data = nitric)
```

This way of plotting is quite clever, and automatically gives a plot which is appropriate to the type of variables being plotted. We will see an example of plotting using the formula interface with categorical variables in lab 3.

A matrix of scatterplots for all possible pairs of variables in the data can be produced with

```r
pairs(nitric)
```

Remove the pairs plot from your screen by typing:

```r
dev.off()
```

**Question:** What do you think `plot(nitric)` will do? Run the command and see what happens. Are you surprised?

## Posh plot with ggplot 2

`ggplot2` is an add-on R package. It provides a powerful graphics tool for creating nice and complex plots. Its popularity in the R community has exploded in recent days. The package allows you to create graphs that present both univariate and multivariate numerical and categorical data in a straightforward manner. Grouping can be represented by colour, symbol, size, and transparency.

Mastering the ggplot2 language can be challenging, and it is not required for assessment in this module. In the computer lab series we will gradually scratch the surface of ggplot2.

To get started, you will need to first install and load the package by:

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

We can plot `yield` against `temp` by ggplot2. The most basic way is

```r
g1 <- ggplot(data=nitric, aes(x=temp, y=yield)) + geom_point()
print(g1)
```

Check how the scatterplot looks. Here `aes` mentions the legend and attributes which need to be plotted in the data, and `geom_point` controls the fact that we are producing a scattered plot.

**Question:** How would you add sensible axis labels to this plot? How would you change the colour of the plotting points? How would you change the shape of the plotting points?

Similarly, we can produce a matrix of scatterplots for all possible pairs of variables, with the help of the function `ggparis` from another R package called `GGally`. Try

```r
install.packages("GGally")
library(GGally)

g2 <- ggpairs(data=nitric)
print(g2)
```

You can find that in the resulting plot, the diagonal consists of the densities of each variable and the upper panels consist of the correlation coefficients between the variables.

There is a helper function called `qplot` (stands for quick plot). This can hide much of the complexity of `ggplot2` when creating standard graphs. Please have a read of the help file of qplot.

```r
help(qplot)
```

I attach a chart here for the graphic parameters in `qplot`.

| **option** | **description** |
|---|---|
| alpha | Alpha transparency for overlapping elements expressed as a fraction between 0 (complete transparency) and 1 (complete opacity) |
| color, shape, size, fill | Associates the levels of variable with symbol color, shape, or size. For line plots, color associates levels of a variable with line color. For density and box plots, fill associates fill colors with a variable. Legends are drawn automatically. |
| data | Specifies a data frame |
| facets | Creates a trellis graph by specifying conditioning variables. Its value is expressed as *rowvar ~ colvar*. To create trellis graphs based on a single conditioning variable, use *rowvar~*. or *.~colvar*) |
| geom | Specifies the geometric objects that define the graph type. The geom option is expressed as a character vector with one or more entries. geom values include "point", "smooth", "boxplot", "line", "histogram", "density", "bar", and "jitter". |
| main, sub | Character vectors specifying the title and subtitle |
| method, formula | If geom="smooth", a loess fit line and confidence limits are added by default. When the number of observations is greater than 1,000, a more efficient smoothing algorithm is employed. Methods include "lm" for regression, "gam" for generalized additive models, and "rlm" for robust regression. The formula parameter gives the form of the fit.<br><br>For example, to add simple linear regression lines, you'd specify geom="smooth", method="lm", formula=y~x. Changing the formula to y~poly(x,2) would produce a quadratic fit. Note that the formula uses the letters x and y, not the names of the variables.<br><br>For method="gam", be sure to load the mgcv package. For method="rml", load the MASS package. |
| *x*, *y* | Specifies the variables placed on the horizontal and vertical axis. For univariate plots (for example, histograms), omit *y* |
| xlab, ylab | Character vectors specifying horizontal and vertical axis labels |
| xlim,ylim | Two-element numeric vectors giving the minimum and maximum values for the horizontal and vertical axes, respectively |

Next, we give some toy examples of using `qplot` for produce graphs.

Let divide the temperature variable `temp` into two groups: `high` and `low`, depending on if it is greater than 20 °F , i.e.,.

```
# we need to transfer temp into a factor(categorical variable)
nitric$temp <- factor(nitric$temp>20 )
levels(nitric$temp)=c("Low", "High")
```

For example, we first plot the density of `yield` response, divided into 2 groups by the temperature as follow:

```
# Kernel density plots for flow
# grouped by temperature (indicated by color)
qplot(yield, data=nitric, geom="density", fill=temp, alpha=I(.5),xlim=c(5,50),
   main="Density of yield", xlab="yield",
   ylab="Density")
```

Next, we produce a scatter plot of `flow` versus `yield` for different `temp`:

```
# scatter plot for flow and yield, under different temperature
qplot(flow, yield, data=nitric,
    facets=temp~1, size=I(3),
   xlab="flow", ylab="yield")
```

Then we plot the regressions of `yield` on `flow` for two groups of `temp`

```
# linear regressions for flow on yield,
# under different temperature
qplot(flow, yield, data=nitric, geom=c("point", "smooth"),
   method="lm", formula=y ~ x,
   color= temp,
   main="Regression of yield on flow",
   xlab="flow", ylab="yield")
```

Last, we produce a boxplots of `conc` by `temp`.

```
# boxplots for conc, grouped by different temp
qplot(temp, conc, data=nitric, geom=c("boxplot", "jitter"),
   fill=temp, main="Contentration by temperature",
   xlab="Temperature", ylab="Contentration")
```

To learn more about `ggplot2`, see the the ggplot reference site, ggplot2 Tutorial and a cheatsheet (click for the web links).