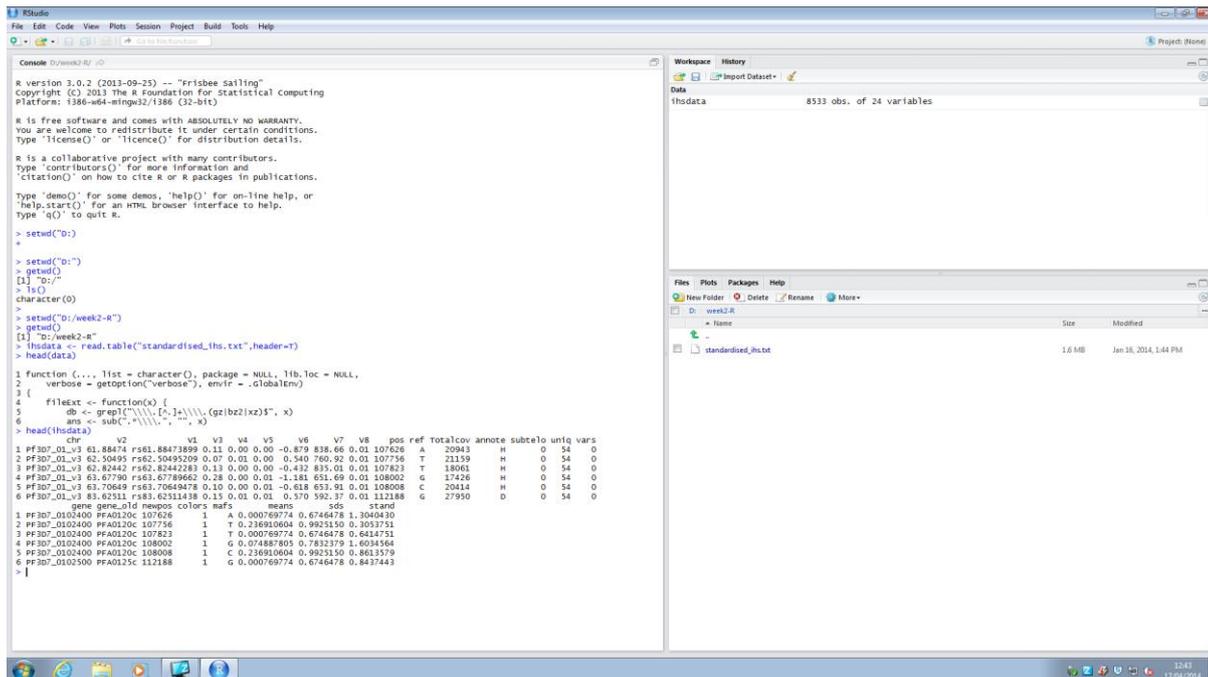


A brief introduction to R – 2017

Overview of R

R is a powerful and free scripting language and environment designed for statistical analysis and visualisation of data. It is suited to the use of large datasets, particularly those formatted along the lines of a spreadsheet. Due to its widespread use guidance and tutorials are easy to find online so if you run into a problem it is likely that a quick web search will yield a solution. For this introduction we shall be using R Studio, which can be downloaded for free from <https://www.rstudio.com/>.

Layout of R studio



When you open R Studio you will be presented with three windows, as shown above. The large window on the left hand side is the console window, where you input command and can view any data that has been created. The top right window is the workspace window, which lists both the recent commands used and all the data that has been loaded into R. The bottom right window contains tabs for files, plots, packages and help. Today we will mostly be using the plots tab. Throughout this introduction you will see text highlighted in the following ways:

Things for you to attempt in R

Commands to type into the console window

Variable, folder and file names

Getting help

If you continue to use R outside of this introduction you will find yourself running into problems on a regular basis. I say this with certainty because even experienced programmers run into problems and often spend more time debugging their code than they did writing it in the first place. When this happens there are a few important steps to solving the issue:

1. Check your code for mistakes such as spelling or putting the wrong number in. If R can run the command you typed it will, even if the result isn't what you wanted.
2. Keep a copy of your code in a separate file. This will help when it comes to finding errors but also makes it much easier to repeat an analysis. In the long run you can transform this into a pipeline, a set of instructions that can describes all the analysis in a project from start to finish.
3. Look at the output from each step and check it did what you intended to. Many mistakes are missed because R returned an output that looked correct at first glance even though it isn't.
4. Search online. If you run into a problem it's likely that somebody else has run into it beforehand, especially when using the most common functions. Learning how to search effectively will save you a lot of time in the long run.

Working directory & loading files

The working directory is the folder on your computer from which you are currently working and contains all the files needed for the current piece of work. For example if you are working on multiple projects you may have folders named *plasmodium*, *trypanosoma* and *influenza*. All of these folders may contain files with similar names, for example *snps.txt*, *genomesequence.txt* or *currentcode.r*.

In order for R to know which folder we wish to work from we need to define the current working directory. After doing this R will automatically look in this folder when loading files and save data / plots to this folder unless we tell it otherwise. To check the current working directory we use the command:

```
getwd()
```

which will provide us with a line that looks like this:

```
[1] "D:/folder/asecondfolder"
```

To set the working directory we use the command:

```
setwd("D:/folder/asecondfolder")
```

This will return an error as those folders don't exist. It's important to note that the location is surrounded by "quote marks" which tells R that the content is plain text and not a variable.

Try setting the working directory to *D:/R-intro*

Using R as a calculator

R can be used as a powerful calculator due to its ability to store and manipulate data and results. For example inputting the following:

```
((15 * 2.59^3)-5)/13
```

quickly returns 19.66228

Basic variables

A variable is a piece of data that has been assigned an name, which we can use to tell R which data to use when running scripts. The advantage here is that once data has been stored in a variable it can be recalled, copied and manipulated using its name rather than having to type it all in again. The simplest variables contain just 1 piece of data but more complex variables could contain an entire spreadsheet.

To define a variable we take the data and assign it a name in one of two ways (they do exactly the same thing):

```
a = 15
```

```
a <- 15
```

Here the variable is called 'a' and the data inside it is the number 15.

Any time you want to see what is in a variable you can just type its name into R. Variables can be used in place of the data they contain, for example if we replace the 15 in our earlier calculation with the variable:

```
((a*2.59^3)-5)/13
```

it will still return 19.66228. We can also use variables to store text, such as:

```
fish = "salmon"
```

Note the quote marks, which tells R we wish to store the text "salmon" in the variable fish.

```
fish = salmon
```

If you type the above what error do you get? What do you think this means?

Basic functions

Functions are predefined instructions that tell R to perform a specific operation. An individual function can perform multiple actions using a single command, saving a lot of time when it comes to writing scripts. It is possible to define your own functions in addition to the hundreds that are installed as part of R. A typical function takes the form of

```
functionname(variable, variable, etc)
```

You can often find out what a function does by typing

```
?functionname
```

While typing

```
functionname()
```

without any variables will show the individual actions that are performed when the function is run.

What does the class function do? Try using it with your *a* and *fish* variables

Expanding variables further

Most variables are more complex than the *a* and *fish* variables we defined above. Those were examples of scalar variables, which means they contains a single number or block of text.

Vectors are like a row of numbers or words which are collected together into a single place. One of the ways you can create them is by concatenating data together using the *c* function:

```
b = c(15,6,7)
```

```
c = c(fish,"crab","boat")
```

which tells R to create a variables that groups the numbers or words together.

Does *c* contain what you expected it to?

What happens if you use *b* in our equation from earlier? What happens if you try and use *c* in the equation?

What is the difference between *c* and *c()*?

Variables can be expanded further by adding extra dimensions, generating tables such as those found in spreadsheets. We can do this using the `matrix()` command:

```
matrixone = matrix(data = c(b,4,5,6,15,2,"fish",7,44,2), ncol=3)
```

What does `matrixone` look like? What do you think the `ncol=3` is doing?

If you look closely at the above you can see that inside the matrix function we have also used the concatenate `c()` function to group our data together, including the variable `b` which we defined earlier. Learning to combine functions and variables within one another is a core skill in scripting. Don't be afraid to try different combinations but always check the results to see if it did what you wanted it to do.

Dataframes are a special type of matrix where the columns are all assigned names in addition to the column number. This allows you to call or use a column without knowing the column number or order of all the columns. To create a dataframe use the following:

```
sampledf = data.frame("numbers" = b, "ocean" = c, "morenumbers" = c(11,33,65))
```

In order to call one individual column you use the `$` sign as so:

```
sampledf$ocean
```

which should return salmon, crab and boat. This only works with dataframes as the `$` symbol is pulling up columns based off of their name.

Try creating a new dataframe called `Continents` that contains the columns `Europe`, `Asia`, `Africa` and `NorthAmerica` each containing 3 countries that can be found in these continents.

Why did we call the 4th column `NorthAmerica` and not `North America`?

Accessing data

Once we have created a variable we will often then need to view, use or alter it. Using the \$ symbol in our dataframe is one way of doing this. For scalars (such as *a*) this means just using the ID we gave it but what if we wanted to change the “boat” in *c* into “ship”? Here we use square brackets [] to tell R the position of the item we wish to change. “boat” is the third entry of *c*, so to get just that entry we would write:

```
c[3]
```

while to change it to “ship” we would do:

```
c[3] = "ship"  
or  
c[3] <- "ship"
```

where we have multiple dimensions, such as in *matrixone* we need to specify the position of each dimension.

When we created *matrixone* we used the variable *c* to fill it with data. Has changing the contents of *c* changed *matrixone*?

For a table we do this in the order of row number then column number. For example:

```
matrixone[1,2]
```

would return the number 5. If you wished to get all the data in column 2 you would remove the row number as so but leave the , in place so that R knows the 2 is the column number:

```
matrixone[,2]
```

Try changing the “fish” in *matrixone* into the number 42.

Searching data

When dealing with large datasets it is often necessary to search through them to find only the small number of entries you are interested in. While it is possible to use the above approach of listing specific rows or columns this isn't feasible when you have thousands of possible entries to check.

To search through variables we need to tell R to match an entry in the data to the number or piece of text we are looking for. The simplest way of doing this is to use a double equals sign == which means we want R to check if what is on both sides is the same, in other words do they match. For example:

```
2 == 5
```

```
"fish" == "fish"
```

```
fish == "salmon"
```

would return FALSE, TRUE and TRUE respectively.

The last two examples above both return TRUE, what is the difference between them?

Using this we can search data to find cases that return TRUE. If we go back to *sampledf* we can try to search for the entries that contain the word crab using:

```
sampledf == "crab"
```

What does this return? What do these results correspond to?

What if we wanted to know all the data in the row associated with "crab"? (It helps here to imagine data being collected together as a spreadsheet with each row being a single entry). To answer this we need to specify that we're looking for the word "crab" in the column called "ocean" which we do using one of these two ways:

```
sampledf$ocean == "crab"  
which(sampledf$ocean == "crab")
```

This will either return a FALSE TRUE FALSE or just the number 2, both of which tell us that "crab" is the second entry (which in this case is a row) in the "ocean" column. To get the entire row we combine this with the square brackets [] we used earlier:

```
sampledf[sampledf$ocean == "crab",]  
sampledf[which(sampledf$ocean == "crab"),]
```

These should both give the same result, though for large datasets using the which() function is often quicker and more efficient.

Copying data

Often we will want to copy data from one variable to another. The most common reason for this is that we are interested in only a small part of the data and want to copy it to another variable where we can change it without changing the original. We do this in the same way as creating a new variable, by using the = sign. For example:

```
matrixtwo = matrixone
```

which will create an identical copy of matrixone called matrixtwo. This is particularly useful for creating a backup of your data in case you make a mistake later on. Alternatively we may want to copy just a portion of the data, which we can do as below:

```
oceandata = sampledf$ocean
```

More Functions – Try and put each of these to use

Functions are a way of automating a process in order to speed up what you are doing, we've already seen examples of this with `c()` and `matrix()`. Many additional functions exist and the best way to learn how they work is to try them and see what they do. Most functions have multiple options that you can include, you can find out what these are by putting a `?` in front of the function name, eg `?plot`

The following functions are some of the most commonly used and will be of use in the exercise at the end. It is also possible to write your own functions, allowing you to automate complex tasks that you perform on a regular basis.

`read.table()` – Reads a file that is organised as a table. Normally we will want to then store this in a variable using `variableID <- read.table("filename.txt")`

`head()` – Will return the first 6 row of a matrix or dataframe, useful for when there are hundreds or thousands of rows present. If you want more or less rows then use `head(matrix, n=number)`

`tail()` – Will return the last 6 row of a matrix or dataframe, useful for when there are hundreds or thousands of rows present.

`dim()` – Returns the dimensions of a matrix or dataframe, which will normally be the number of rows and columns.

`nrow()` – Returns the number of rows in a matrix or dataframe

`ncol()` – Returns the number of columns in a matrix or dataframe

`colnames()` – Returns the column names for a dataframe

`length()` – Returns the length of a vector or number of entries in a matrix / dataframe

`max()` – Returns the largest value from a vector of numbers

`min()` – Returns the smallest value from a vector of numbers

`sum()` – Adds all the numbers in a vector together

`mean()` – Calculates the mean of the numbers in a vector

`median()` – Calculates the median value of the numbers in a vector

`table()` – Summarises a vector as a table listing all the unique entries and the frequency with which they appear

`plot()` – Attempts to plot the data in a manner that R thinks is correct. This is a powerful function with many inbuilt options

`barplot()` – Attempts to plot a barplot of the data

`hist()` – Attempts to plot a histogram of the data

`abline(h=X)` or `abline(v=X)` – Add a horizontal or vertical line to a plot at position X

Exercise

The following exercise is designed to give you an opportunity to learn how to use R with a dataset from a study into genome wide selection in malaria (<http://mbe.oxfordjournals.org/content/early/2014/04/08/molbev.msu106>). The exercise will require you to put the prior examples to use, changing the input as required. For the most part we have avoided providing the actual commands you will need to type as working out how to structure the input is the best approach to learning how to use R. All of the tasks in this exercise can be completed using the functions and information you have already been provided with so try experimenting with them to work out what they do to the data.

Set the working directory to *D:/R-intro/*

Load the provided data in this folder into a variable called *ihsdata* using

```
ihsdata <- read.table("standardised_ihs.txt", header=T)
```

here the header=T tells R to treat the first line as column names and load the data as a dataframe.

Have a look at the dataframe to get an idea of how it is organised.

Each row in *ihsdata* represents a single SNP in the genome of the human malaria parasite *P. falciparum* while the columns contain information about that SNP. The columns are:

1. chr – which chromosome the SNP is on
2. pos – the position of the SNP on the chromosome
3. ref – the base present in the genome reference strain at this position
4. Totalcov – the total sequencing coverage at this position from the 100 isolates this data was generated from
5. gene – the ID of the gene that this SNP is positioned within (if within a gene)
6. gene_old – the previous version of the gene ID (useful if you are looking up older papers)
7. genpos – the position of the SNP in the genome
8. colors – a column we'll use to colour the chromosome later
9. ihs – the ihs score for this SNP, which is a measure of directional selection at this locus

How many SNPs are there in the dataset?

How many chromosomes are there? How many SNPs are on chromosome 5?

What is the mean, max and min of the coverage? Why do you think none have a coverage of 0?

Using the hist() function try and plot a histogram of the total coverage for the dataset

This dataset was used to identify regions of the genome under directional selection by calculating their integrated haplotype score (ihs) for each SNP and plotting these scores genome wide. To plot the scores across the genome, with the chromosomes coloured in an alternating red / black pattern use the following:

```
plot(ihsdata$genpos, ihsdata$ihs, pch='.', cex=2, col=c("black","red")[ihsdata$colors])
```

which can be broken down as follows:

plot – The plot function which is used to plot basic figures in R

ihsdata\$genpos – the position of each point of the x-axis, defined by the genpos column of *ihsdata*

ihsdata\$ihs – the position of each point on the y-axis, defined by the ihs column of *ihsdata*

pch='.' – tells R to use . for drawing the points

cex=2 – tells R how big to make each point

col=c("black","red")[ihsdata\$colors] – tells R to use the colours black and red determined by the values in the ihsdata\$colors

Now that you have the plot you should be able to see that there are clusters of SNPs with high scores.

How many SNPs in *ihsdata* have ihs scores > 3?

Only 4 SNPs have ihs scores > 7. Which genes are they in?

The chloroquine resistance transporter (gene ID PF3D7_0709000) is the main gene responsible for chloroquine resistance in malaria and is located on chromosome 7.

How many SNPs are present within this gene?

Can you add a line to the plot to mark the position of this gene?

Finally can you plot only the SNPs that are on chromosome 7?