**BIOL2202 – R Tutorial**

R is a language for statistics and scientific programming, designed by statisticians themselves to be easy to learn and intuitive. R is free of cost with no limits of use. It is open source, so anyone can redistribute it and reproduce it. You can survey other people’s code, modify it, and make your own and make it public for others to survey and modify.

R is an easy to learn, natural and expressive. Because R is a programming language, you can save your script with every step of your work, promoting reproducibility and transparency. Programming also allows you to create highly customized analyses; for instance, you may wish to automate a repetitive task or integrate several kinds of tasks into one.

Graphical depictions are essential for data analysis. R is equipped with outstanding tools for visualization of data, while many tools can be installed into it. This makes the graphical power of R close to limitless.

R is object-oriented, so data are treated as objects. This means that you can easily access any part of your data, transform it, and aggregate it. The R community can also produce programs called “packages” that allow for powerful cutting-edge analyses. There are >2000 of these packages that allow you to handle highly specialized data.

One more reason to use R is that the support of online tutorials, forums, blogs, and books is exceptional. If you are willing to get a task done, you will not be stuck for long. A simple Google search beginning with the letter “r” followed by your question will probably solve your issue!

The official introduction to the R language is available at

<http://cran.r-project.org/doc/manuals/R-intro.pdf>

The R homepage is http://www.r-project.org/

If R is not installed in your machine, download the program from

<http://cran.csiro.au/>.

It is much easier to use R from an Integrated Development Environment (IDE). An IDE is a program that provides a user-friendly interface for executing R commands and running R scripts. In class, we will use RStudio. This is a freely available IDE that can be downloaded at:

<http://www.rstudio.com>

Be sure to download the free Desktop version.

**1. The Basics**

When you first open R, you are presented with a screen which is referred to as the “console”. This is where you type in commands to manipulate data, carry out mathematical calculations and statistical tests, create plots etc. You can revisit your past calculations by using the up and down keys.

In RStudio, this is the window in the bottom left.

If you don’t understand what a command or function does, or how to use it, use a question mark before the function’s name to get the help file. For example:

?sqrt

**2. R as a calculator**

R can be used as a calculator. R will ignore anything written after a hash symbol (#), which allows you to add notes to your script. Try these simple calculations, hitting RETURN to see your answer.

5 + 2

2 \* 25 # Multiplication.

67 / 13 # Division.

**R is object-oriented.** This means that data in R are treated as independent pieces of information stored in your computer’s Random Access Memory (RAM). The arrow <- is an “operator” used to temporarily save data in your RAM. When you press RETURN the data you assigned becomes an “object”. The following command assigns the result of 5 + 6 (=11) to the variable x.

x <- 5 + 6

To see the value of a variable, type its name.

x

Assign the following values to the variables y and z.

y <- 15 / 3

z <- (x + y) / x # This will save a 2.

Check that you have been successful by typing in the names of each of the variables.

Notice that R saves the result of an assignment, not the assignment itself. The name of your object can only begin with a letter and can only include letters, numbers and dots. Note that R is case sensitive. Avoid c, T, or F as names because they stand for the function “concatenate”, and the logical TRUE, and FALSE respectively.

**TIP! Name objects well**

Keep names short but meaningful, and remember that R is case sensitive. For example, use a name like david.yeast.dat not d.y.d nor biol2202.davids.bench.yeast.prac.data

The objects ‘x’, ‘y’, and ‘z’ should now be stored in the current R session. You can access them at any time until you close R.

**3. Functions and types of objects in R**

**Definition of function:** In R, a function is a written command to perform an action on data, given to the function in brackets and separated by commas. Each comma-separated piece of information is called an “argument”. From now on, most of your interaction with R will be using functions. For example, the function rm removes or deletes objects.

rm(x)

x # This will return an error because x has been deleted.

You can keep track of the objects you have currently saved by typing ls(). Multiple objects can be deleted by listing them in brackets separated by commas inside the rm function. The inside of the brackets is called the “function call”.

rm(y, z)

In RStudio, you will see your variables listed in the top right window under the “Environment” tab

**Definition of class:** Every object in R has a type or “class”. A few examples of classes are “numeric”, “character”, “list”, “matrix” and “data.frame”. The function class tells us the class of an object.

x <- 5

class(x)# This command will return the word "numeric".

Perhaps the most basic function in R is the concatenation function c() which is used to create “vectors”. A “vector” is simply a one-dimensional group of values of data. The data in the parentheses should be of the same class, otherwise R will convert all the characters to be in the same class, and this can give you strange results.

c(1, 2, -3, x)

a <- c(1, 2, 3, x)

b <- c(2, 2, 2, x)

d <- c("non.UV", "UVplusdark", "UVpluslight")

class(a) # A numeric vector

class(d) # A character vector

If possible, R will make objects interact in an intuitive way; for example, it allows arithmetic equations between two vectors of equal length as if they were matrices. For example, try:

a + b

a \* 7

**TIP! Check results**

Always check whether the result returned is what you expect, even if it’s a calculation and you can only tell roughly.

As examples of functions, the following functions transform your objects by taking the square root, the logarithm, the absolute values, and elevating to the power of two every “element” in your object.

sqrt(a)

log(a)

abs(a)

a^2

The following functions take a vector and calculate the sum, the mean, the standard deviation, and a summary (quantiles, mean, and median) respectively.

sum(a)

mean(a)

sd(a)

summary(a)

**4. Importing/Exporting data**

It is often more simple to enter raw data in software other than R (such as Microsoft Excel) and then import it. There are many functions to import data; here we will focus on read.csv. Functions described in this tutorial from here onwards can be complex, so remember to use the author’s documentation.

All the files and data in your computer have a file path. Data are imported into R by typing the file path into an import function. R has a **working directory**, which is the folder where data will be saved to and retrieved from by default. This can be changed with the function setwd . If your data are in the working directory, you only need the file name in order to import it. In this tutorial we assume that your data are inside your working directory, and that your working directory is the desktop.

getwd() # Shows your current working directory.

setwd("~/Desktop")

You may need help here: different computer systems use different ways of referencing their directories. For instance, in Windows, you may need to specify:

setwd("C:/Users/YourName/Desktop")

Check the documentation for importing datasets in the format “comma-separated values (.csv)” using ?read.csv to make sure that the arguments are what you want (e.g. if you have a header you might want to set header = TRUE ). We will import the dataset from the 2015 yeast practical.

yeast.data <- read.csv(file = "yeast\_data.csv")

It is important to check whether the object was created and whether its content is what you expected. The following gives you a description of the object, with its class, its number of observations, and variables contained.

str(yeast.data)

To export your data in its initial format you can use write.csv. Remember that using the same file name may rewrite your previous file.

write.csv(yeast.data, file = "new\_yeast.csv")

**Tip! Import/export R objects in RStudio**

In the “Environment” tab on the top right window in RStudio, there is an option to import a dataset. RStudio actually performs this import by generating the command “read.csv()” for the appropriate file that you select.

**6. Plotting**

Plotting is perhaps the most important way to explore and visualize data. The most basic function to make graphics in R is plot, which receives a variable for the x-axis, followed by a variable of the same length for the y-axis. Additionally, it can receive a large range of arguments to change the plot’s aesthetics.

plot(a, b)

Histograms are important exploratory plots of the frequency distribution of a variable. They are made with the function hist.

hist(yeast.data$high.concentration) # Plots a frequency histogram of the data produced under high concentration.

In RStudio, you will see the plots appear in the bottom left window.

**7. Running an R script**

You can write a set of R commands in any text file (not a word processor!) and save it for future use. This is called a script. For example:

yeast.data <- read.csv(file = "yeast\_data.csv")

str(yeast.data)

hist(yeast.data$high.concentration)

hist(yeast.data$low.concentration)

Once you’ve saved this to a file in the appropriate working directory, you can run this script in the console window with the command:

source(“filename”)

For instance, if you named your file myRprogram.R (its always a good idea to label the file with a .R suffix), then you will type:

source(“myRprogram.R”)

This will allow you to save your work and reuse your commands.

**GLOSSARY**

**Argument**

The information given to a function, separated by commas. Arguments are often given as objects. Functions often require specific classes of objects as arguments. Arguments are also often optional, or have default settings, so always check the required arguments in the function documentation.

**Character string**

Words or letters in an object, usually enclosed by speech marks and used to label other objects or as categorical variables (often known as factors). These alone are not considered objects, they are attributes of objects, they do not have a class, and unless saved to an object they cannot be called from the R prompt.

**Class**

All objects in R have a class, which defines the type of data that they contain. Objects of different classes will behave differently to actions in R (e.g. a matrix behaves as a mathematical entity while a data frame does not). Functions are often specific about the class of objects they receive.

**Console**

The R screen showing the prompt and commands executed in the current session.

**Element**

An object inside an object. Elements alone often have no class unless they are assigned as elements. Elements in an object have an index, which can be used to extract and perform computations on them.

**Function**

A code to perform an action in R. Most functions have arguments to specify the action and the data to be performed upon.

**Indexing**

The location of every element in an object is called its index. Indexing is the action of locating elements to extract them from a parent object.

**Object**

R is object-oriented. This means every piece of data in R is an object. All objects have a specific class and an address in the Random Access Memory (RAM) of the machine.

**Package**

All R users can create their own code and upload it to a public R database in the form of a package of functions. Packages are usually specialised on a kind of functions or a class of object. Exploring specialised packages, programming in R, and publishing code is not described in this tutorial, but many tutorials are available online.

**Prompt**

The location in the R console where commands can be executed, at the lower end of the console.

**Vector**

One-dimensional objects in R that may contain character strings or numeric values.

**Working directory**

The default folder to read and write data. Objects are not saved into this folder until an export function has been used. Data from other folders can be accessed and saved by typing the full path of the target folders in R.