CM 2062 - Statistical Computing with R
Lab Sheet 1

Introduction

R is a programming language use for statistical analysis and graphics.

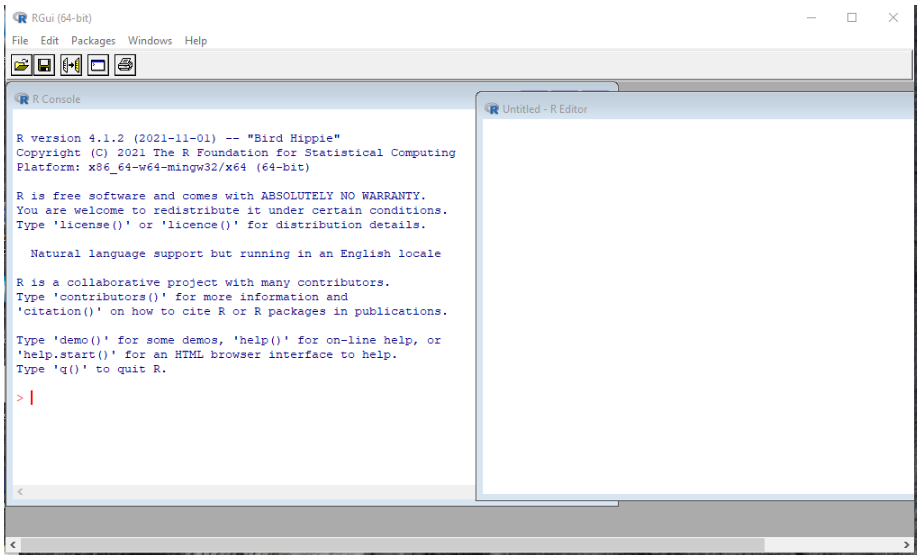
R is offered as open source (i.e. free)

Getting and installing R

The main sources for R are CRAN (http://www.cran.r-project.org) and its mirrors. You can get the source code, but most users will prefer a precompiled version. To get one from CRAN, click on the link for your OS, continue to the folder corresponding to your OS version, and find the appropriate download file for your computer. For Windows or OS X, R is installed by launching the downloaded file and following the on-screen instructions. At the end you'll have an R icon on your desktop that can be used to launch the program.

Working with R

starting R under windows opens a simple graphical interface shown in below Figure 1. One can now start to enter R commands in the R console or in a new script file.



```
R version 4.1.2 (2021-11-01) -- "Bird Hippie"
Copyright (C) 2021 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 
```

Example

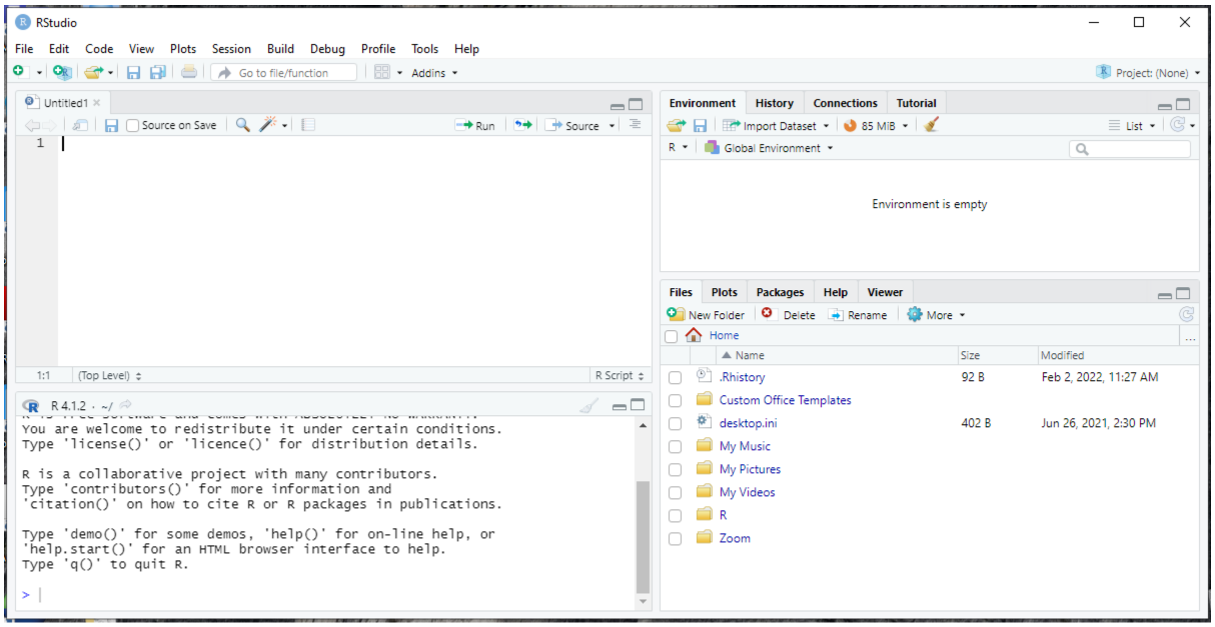
Let's consider a simple example. Let a=2 and b=5-a. Get the value of b from R. (Use R console only and a new script as well)

Getting and installing Rstudio

Rstudio can be downloaded for free from https://www.rstudio.com/, and should be installed after R. R studio is more user friendly compared to R.

Working with R Studio

starting R Studio under windows opens a more user friendly graphical interface shown in below Figure 2.



Environment is empty

Name	Size	Modified
.Rhistory	92 B	Feb 2, 2022, 11:27 AM
Custom Office Templates		
desktop.ini	402 B	Jun 26, 2021, 2:30 PM
My Music		
My Pictures		
My Videos		
R		
Zoom		

```
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Do the same previous example using R studio.

Saving R Script

Save your first R document as "R1.R" in any folder which you like in your computer. File extension will be .R for any R file.

Set working directory in R

The working directory is the default location on your computer where R assumes all of your work is being conducted at a given time. For example, if you were to import data from an external file, R would automatically look for it in your working directory. Furthermore, all file path arguments in functions are evaluated relative to the working directory. Therefore, it is important to set your working directory each time you use R.

Session → Set Working Directory → Choose Directory

Then set your working directory to any folder that you like.

Save Work Space in R

Saving your workspace creates an image of your current variables and functions, and saves them to a file called ". RData". When you re-open R from that working directory, the workspace will be loaded, and all these things will be available to you again.

Session → Save Workspace As

Using the R online help system

For help at any time, simply type ? or help("subject") where "subject" is the area you´re interested in; for example, if you want to know how to calculate mean, simply type

?mean
help("mean")

help.search() will search all sources of documentation and return those that match the search string. ?? is a shortcut for help.search()

??Regression
help.search("Regression")

R as a calculator

R can be used as a powerful calculator by entering equations directly at the prompt in the command console. Simply type your arithmetic expression and press ENTER. R will evaluate the expressions and respond with the result. While this is a simple interaction interface, there could be problems if you are not careful. R will normally execute your arithmetic expression by evaluating each item from left to right, but some operators have precedence in the order of evaluation. Let's start with some simple expressions as examples.

Simple arithmetic expressions

The operators R uses for basic arithmetic are: Addition, Subtraction, Multiplication, Division, Exponentiation.

Example 1
Do these calculations using R.

1) 4 + 8
2) 5 * 14
3) 7 / 4
4) 4 + 5 + 3
5) $4 \wedge 3$
6) 4 + 5 * 3
7) (4 + 5) * 3
8) $4 + 3 \wedge 2$
9) $(4 + 3) \wedge 2$
10) $4 \wedge 0.5$
11) $16 \wedge 0.25$
12) $16 \wedge (1/4)$
13) $8 \wedge (1/3)$

Example 2
Calculate

$$\sqrt[3]{\frac{435.4 * 3.56}{(34+3)^2}}$$

using R.

You will enter this in the RStudio command console:

$((435.4 * 3.56)/(34 + 3) \wedge 2) \wedge (1/3)$

R in-Built Functions:

```
pi
exp(3) ## provides the cube of e
log(1.4) ## provides the natural logarithm of the number 1.4
log10(1.4) ## provides the log to the base of 10
sqrt(16) ## provides the square root of 16
```

Assignment Statements
Just like in algebra, we often want to store a computation under some variable name. The result is assigned to a variable with the symbols $< -$ which is formed by the "less than" symbol followed immediately by a hyphen.

```
 x <- 2.5
```

When you want to know what is in a variable simply ask by typing the variable name.

```
 x
```

We can store a computation under a new variable name or change the current value in an old variable.

```
y <- 3*exp(x)
x <- 3*exp(x)
```

Data Types in R

R has a number of basic data types.

- Numeric – Examples: 1, 1.0, 42.5
- Integer – Examples: 1, 2, -1
- Complex – Example: 4 + 2i
- Logical – Two possible values: TRUE and FALSE
- Character – Examples: "a", "Statistics", "Hello"

Variable (Object) Names

You should create variable names that show the meaning of what is computed. Do not begin a variable name with a number. Variable names are case (upper/lower) sensitive. Abc and abc are not the same variable. Do not use blank space as a separator in variable names (R will probably give you an error message if you do). Use '_' or '.' as a separator.

Examples:

```
Sample Average <- 35
SampleAverage <- 35
Sample.Average <- 35
Sample_Average <- 35
```

Data Structures

R also has a number of basic data structures. A data structure is either homogeneous (all elements are of the same data type) or heterogeneous (elements can be of more than one data type).

Dimension	**Homogeneous**	**Heterogeneous**
1	Vector	List
2	Matrix	Data Frame
3+	Array	

Vectors

Vector represents a set of elements of the same mode whether they are logical, numeric (integer or double), complex or character.

Many operations in R make heavy use of vectors. Vectors in R are indexed starting at 1. That is what the [1] in the output is indicating, that the first element of the row being displayed is the first element of the vector. Larger vectors will start additional rows with [*] where * is the index of the first element of the row.

We can create vectors at the command prompt using the concatenation function c(...) which is short for "combine."" As the name suggests, it combines a list of elements separated by commas.

Examples

```
 > c(1,2,3)
[1] 1 2 3

> c("Amal","Nimal","Kasun")
[1] "Amal" "Nimal" "Kasun"
```

Because vectors must contain elements that are all the same type, R will automatically coerce to a single type when attempting to create a vector that combines multiple types.

```
> c(42, "Statistics", TRUE)
  [1] "42" "Statistics" "TRUE"

> c(42, TRUE)
  [1] 42 1
```

In order to make use of vectors, we need identifiers for them.

```
 > n1 <- c(1,2,3)
> people <- c("Amal","Nimal","Kasun")

> n1
[1] 1 2 3
> people
[1] "Amal" "Nimal" "Kasun"

# Typing an object's identifier causes R to print the contents of the object
```

Notes:
1. To reuse commands, press the up arrow key.
2. The length of a vector can be obtained with the length() function.

```
> length(n1)
[1] 3
```

Simple arithmetic operations can be performed with vectors.

```
 > c(1,2,3)+c(4,5,6)
[1] 5 7 9

> n1 + n1
[1] 2 4 6

> n2 <- c(5,2,-3)
> n2
[1] 5 2 -3

> c(1,2,3)*c(1,3,3)
[1] 1 6 9

> c(12,12,12)/n1
[1] 12 6 4

> n1 <- n1+n2
> n1
[1] 6 4 0

> n2
[1] 5 2 -3

# Note: in the final step we have updated
the value of n1 by adding n2 to the old
value; n1 changes but n2 is unchanged.

> n3 <- n1 + n2
> n3
[1] 11 6-3

> n4 <- n1*n2
> n4
[1] 30 8 0
```

The concatenation function can be used to concatenate vectors.

```
> a <- c(1,1)
> b <- c(0,0)
> c(a,b,a)
[1] 1 1 0 0 1 1
```

You can view the objects created in your R program by calling the function **objects()**.

```
> objects()
```

```
[1] "a" "b" "num1" "num2" "num3" "num4"
```

We have now created a number of objects. To ensure clarity in the following examples we need to remove all of the objects we have created.

```
> rm(a)
> objects()

[1] "b" "num1" "num2" "num3" "num4"

# notice that object    a    is not now in the objects list
```

By the following command, all the objects can be removed.

```
> rm(list=objects())
> objects()
character(0)
```

Regular Sequences

A regular sequence is a sequence of numbers or characters that follows a fixed pattern. These are useful for selecting portions of a vector and in generating values for categorical variables.

We can use a number of different methods for generating sequences. First we investigate the sequence generator.

```
> 1:10
[1] 1 2 3 4 5 6 7 8 9 10

> 10:1
[1] 10 9 8 7 6 5 4 3 2 1

> 2*1:10
[1] 2 4 6 8 10 12 14 16 18 20

> 1:10 + 1:20
[1] 2 4 6 8 10 12 14 16 18 20 12 14 16 18 20 22 24 26 28 30

> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
# Notice that takes precedence over arithmetic operations.
```

The **seq** function allows a greater degree of sophistication in generating sequences. The function definition is,

```
seq(from, to, by, length, along)
```

The arguments "from" and "to" are self explanatory. By giving the increment for the sequence and length, the number of entries we can get the sequence that we need. Notice that if we include all of these arguments there will be some redundancy and also an error message will be given. Notice how the named arguments are used below. By playing around with the command, see whether you can work out what the default values for the arguments are.

```
> seq(1,10)
[1] 1 2 3 4 5 6 7 8 9 10

> seq(to=10, from=1)
[1] 1 2 3 4 5 6 7 8 9 10

> seq(1,10,by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5
[15] 8.0 8.5 9.0 9.5 10.0

> seq(1,10,by=0.25)
 [1]  1.00  1.25  1.50  1.75  2.00  2.25  2.50  2.75  3.00  3.25  3.50
3.75  4.00
[14]  4.25  4.50  4.75  5.00  5.25  5.50  5.75  6.00  6.25  6.50  6.75
7.00  7.25
[27]  7.50  7.75  8.00  8.25  8.50  8.75  9.00  9.25  9.50  9.75 10.00

> seq(1,10,length=19)
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
8.0  8.5
[17]  9.0  9.5 10.0

> seq(1,10,length=19,by=0.25)
Error in seq.default(1, 10, length = 19, by = 0.25) :
Too many arguments
> seq(1,by=2,length=6)
[1] 1 3 5 7 9 11
```

Another common operation to create a vector is **rep**, which can repeat a single value a number of times.

```
> rep(1, times = 3)
[1] 1 1 1
> rep((1:3), each =5)
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

Note: We have now seen four different ways to create vectors:

- c()
- :
- seq()

- rep()

So far we have mostly used them in isolation, but they are often used together.

```
> c(rep(seq(1, 9, 2), 3), c(1, 2, 3), 42, 2:4)
 [1]  1  3  5  7  9  1  3  5  7  9  1  3  5  7  9  1  2  3 42  2  3  4
```

Indexing Vectors and Subset Selection

In data analysis or in data manipulation, you may want to view/use a selection of some data points as per some requirements. R contains several constructs which allow access to individual elements or subsets through indexing operations. The method for referring to a part of a vector is the use of square brackets [ ] indicating inside the bracket the subset of the vector you want out.

Indexing can be used to extract a part of an object or to replace parts of an object (or to add parts).

```
> marks<-c(23,45,67,89,43,67,21,55)
> marks[2]
[1] 45

> marks[2,4]
Error in marks[2, 4] : incorrect number of dimensions

> marks[dim=c(2,4)]
[1] 45 89

> marks[1:3]
[1] 23 45 67

# We can also exclude certain indexes, in this case the second element.
> marks[-2]
[1] 23 67 89 43 67 21 55

> marks[1]<-50
> marks
[1] 50 45 67 89 43 67 21 55
```

To index the portion of vectors which satisfies a particular criterion, you have to specify the criterion inside the [] brackets as a logical vector. The result is the elements for which the value of the logical vector is true.

```
> marks[marks>50]
[1] 67 89 67 55
```