Introduction to R and RStudio

Dr. Isuru Dassanayake

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows, and MacOS. R can be downloaded from CRAN (Comprehensive R Archive Network) at http://cran.r-project.org/. One should prefer the sixty-four bit version, which can handle arbitrarily large amounts of data…..

General Remarks:

• Commands in R are written following the " > " prompt. Once we are done writing the command, we hit Enter to move to the new command line. The space in which we put the commands is called the console.

• R is interactive – results can be seen one command at a time. If there is an incorrect entry in a line of code, R will immediately let us know of this fact – we do not have to wait until all the commands are entered.

• R is case-sensitive. So, uv, Uv, uV, and UV are all different variables.

• We can always add comments to an R program by starting the statement with the # symbol.
• To repeat a line of code, we just press the up arrow key.

• To interrupt a command, we have to press Esc.

• To get help on any function we type ? and the name of the function. For example, to get help on the function mean, we type:

```
>?mean
```

For operations we use ‘‘. For example, to get help on +, we type:

```
>? '+'
```

• There may be occasions when we have only a sense of the function we want to use. In that case, we can look up the function by using a part of its name along with the function "apropos". For example, say we do not know the exact name of the mean function, but we think it starts with m, e , and a:

```
>apropos ("mea")
```

• To copy the session commands and their results to a Word document, simply select and highlight these lines, and Copy and Paste. To copy a graph, after creating the graph, click on Export and then choose the option Copy to Clipboard. Click on Copy Plot. Then go to the Word document and paste.

RStudio

RStudio is an integrated development environment (IDE) for R, that is, it is a software application that consolidates the basic tools needed to write and test R programs.
To download Rstudio, after downloading R, go to http://www.rstudio.com/download

Basic Operations and Functions:

As usual, we use + for addition, - for subtraction, * for multiplication, / for division, ^ for powers, and sqrt for the square root. For the rth root of a number x, we simply write x^(1/r). To get the whole number part in a division we write x%/%y and to get the remainder we write x. So for example,

```
17%/%3
```

```
[1] 5
```

and

```
17%%3
```

```
[1] 2
```

Thus,

```
x = y*(x%/%y)+(x%%y)
17 = 3*(17%/%3) + (17%%3)
```

R follows the usual order of operations: parenthesis, exponents, multiplications, and divisions from left to right, and finally additions and subtractions from left to right.

We can write some common functions very easily in R. We use abs(x) for the absolute value, exp(x) for e^x, and log(x) for the natural logarithm of x. To write log_a (x), we write loga(x). So, the common logarithms, log_10 (x), is written as log10(x). To write e, the base of the natural logarithmic function, we write exp(1).

We write pi for π. The trigonometric functions are written as $sin(x)$, $cos(x)$, tan($x), etc. Here, the argument x given in radians. The inverse trigonometric functions are written as acos(x), asin(x), atan(x), etc. The function atan2(y,x) gives the angle (in radians) from the positive x-axis to the vector (x,y).

Recall that the floor function is the function that takes as input a real number x and gives as output the greatest integer less than or equal to x. It is denoted as x . Similarly, the ceiling function gives the least integer greater than or equal to x, and is denoted as $\lceil x \rceil$. Thus, for example, 3.2 =3 3.2 =4 Of course, for any integer n, n = n =n

To get these functions in R, we write:

```
floor(x)

#and

ceiling(x)
```

Basic Arithmetic Statements:

To perform any arithmetic operation, at the > prompt, we write down the operational statement and hit enter.

```
1 + 2*3 - 15/3
```

```
[1] 2
```

```
log(3*5-7)
```

```
[1] 2.079442
```

Variables

A variable name can be any combination of alphanumeric characters along with period "." and underscore "_". It must start with a letter. To assign a value to a variable we use <- symbol (or =). Thus, to assign the value 3 to variable x, we write:

```
x <- 3

#or

x = 3
```

if then we write x we get its value:

```
x
```

```
[1] 3
```

Assignments can be done simultaneously:

```
a <- b <- 7

a
```

```
[1] 7
```

```
b
```

```
[1] 7
```

Another method is to use the assign function:

```
assign ('h', 4)

h
```

```
[1] 4
```

To remove a variable we use the remove or rm function:

```
j = 4

j
```

```
[1] 4
```

```
rm(j)

j
```

```
Error in eval(expr, envir, enclos): object 'j' not found
```

Data Types

We will discuss numeric, character (string), and logical (true/false) data.

Numeric Data

Any real number stored in a variable is automatically assumed to be numeric. To test whether a variable is numeric we use the "is.numeric" function. So, if for example, x is a numeric variable,

```
is.numeric(x)
```

```
[1] TRUE
```

A sub-category of numeric is integer. To set an integer value to a variable, it is necessary to append the value with an "L". To check whether a variable is integer, we use the is.integer function.

```
g = 5L
g
```

```
[1] 5
```

```r
#or
l = as.integer(5)
l
```

```
[1] 5
```

```r
is.integer(g)
```

```
[1] TRUE
```

Character Data

Character data is always entered in quotes:

```r
x = "table"

x
```

```
[1] "table"
```

Notice in this case the answer is also given in quotes.

Logicals

These are ways of representing data that can be either true or false. Numerically TRUE is same as 1 and FALSE is same as 0.

```r
TRUE*5
```

```
[1] 5
```

```r
FALSE*7
```

```
[1] 0
```

We can test whether a data is logical by using the "is.logical" function

```
k = TRUE

class(k)
```

```
[1] "logical"
```

```
is.logical(k)
```

```
[1] TRUE
```

Binary Comparisons

a is equal to b: a = = b

a is less than b: a < b

a is greater than b: a > b

a is less than or equal to b: a < = b

a is greater than or equal to b: a>=b

a is not equal to b: a != b

```
# Let's check if 2 is equal to 3

2==3
```

```
[1] FALSE
```

```
# Let's check if 4 is not equal to 5

4!=5
```

```
[1] TRUE
```

Vectors

In R, a vector is defined as a collection of elements, all of which are of the same type. There is no distinction between a column vector and a row vector.

The most common way to create a vector is with c (c stands for combine) function: to create a vector $ x_1,x_2,…,x_k$ we write: $c(x_1, x_2, ..., x_k)$ So, for example:

```
x = c(1,2,3,4,5)
x
```

```
[1] 1 2 3 4 5
```

```
y = c ("alpha","beta","gamma","delta")
y
```

```
[1] "alpha" "beta"  "gamma" "delta"
```

Suppose $x = c(x_1, x_2, ..., x_k)$.

If we write: $y = c(y_1, ..., y_j, x)$

this will give us the vector $(y_1, ..., y_j, x_1, x_2, ..., x_k)$.

So if,

```
x = c(7,8,9,10)

y= c(1,2,3,4,5,6,x)

z = c("a","b","c")

m = c(x,z)

n =as.integer(x)
n
```

```
[1]  7  8  9 10
```

```
class(n)
```

```
[1] "integer"
```

The : Operator

If a vector is formed by consecutive numbers from a to b, a<b, we can simply write: a:b,

So for example:

```
1:6
```

```
[1] 1 2 3 4 5 6
```

```
-2:2
```

```
[1] -2 -1  0  1  2
```

```
# also in decreasing order from b to a ==>  b:a

10:2
```

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

```
# Also,

x = 1:9
x
```

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

```
y = 10*x # every element on x will be multiplied by 10

y
```

```
[1] 10 20 30 40 50 60 70 80 90
```

Vector Operations

Operations are applied to each element of the vector automatically, without the need to loop through the vector.

To multiply each element of a vector by a number n, enter the vector in x and then x*n.

Similarly we can perform operations x+n, x/n, x^n, or sqrt (x).

```
x = c(3,7,11,25)
x
```

```
[1]  3  7 11 25
```

```
7*x
```

```
[1]  21  49  77 175
```

```
sqrt(x)
```

```
[1] 1.732051 2.645751 3.316625 5.000000
```

If we have vectors of equal length saved in x and y, we can perform the operations x+y, x-y, xy, x/y and x^y.

```
x = c(1,3,5)
y = c(2,4,6,8,6,3)

x+y
```

```
[1]  3  7 11  9  9  8
```

```
x^y
```

```
[1]     1    81 15625     1   729   125
```

Also,

```r
x = c(1,2,3,4)
y = c(2,3,0,5)

x/y
```

```
[1] 0.5000000 0.6666667       Inf 0.8000000
```

Note that R denotes division by zero as Inf.

Data.frame

A data.frame is a rectangular collection with variables in columns and observations in rows. In a data.frame, columns are actually vectors, each of which must have the same length.

To create a data.frame, we use the data.frame function:

```r
x = 12:4
y = 8:0
z = c("a","b","c","d","e","f","g","h","i")

dat = data.frame(x,y,z) # all the vectors should be the same length.
dat
```

```
   x y z
1 12 8 a
2 11 7 b
3 10 6 c
4  9 5 d
5  8 4 e
6  7 3 f
7  6 2 g
8  5 1 h
9  4 0 i
```

```r
class(dat)
```

```
[1] "data.frame"
```

```
str(dat)
```

```
'data.frame':   9 obs. of  3 variables:
 $ x: int  12 11 10 9 8 7 6 5 4
 $ y: int  8 7 6 5 4 3 2 1 0
 $ z: chr  "a" "b" "c" "d" ...
```

Therefore, we created a 9×3 data.frame.

Alternatively, we could assign names to the variables:

```
dat  = data.frame("Firs tNum" = x,"SecondNum" = y, "Letter" = z)


dat
```

```
  Firs.tNum SecondNum Letter
1        12         8      a
2        11         7      b
3        10         6      c
4         9         5      d
5         8         4      e
6         7         3      f
7         6         2      g
8         5         1      h
9         4         0      i
```

The nrow and ncol functions give us the number of rows and columns, respectively:

```
nrow(dat)
```

```
[1] 9
```

```
ncol(dat)
```

```
[1] 3
```

The dim function gives both the number of rows and the number of columns:

```
dim(dat)
```

```
[1] 9 3
```

Here the first letter is the number of rows and the second number is the number of columns.

Usually, a data .frame has far too many rows to fit in one screen. In this case, we may want to view just the first few or last few rows. To this end, we use the head and the tail functions:

```
head(dat) # to see the top of the data set
```

```
  Firs.tNum SecondNum Letter
1        12         8      a
2        11         7      b
3        10         6      c
4         9         5      d
5         8         4      e
6         7         3      f
```

```
tail(dat) # to see the bottom of the data set
```

```
  Firs.tNum SecondNum Letter
4         9         5      d
5         8         4      e
6         7         3      f
7         6         2      g
8         5         1      h
9         4         0      i
```

```
head(dat, n=3) # to see just the top three rows
```

```
  Firs.tNum SecondNum Letter
1        12         8      a
2        11         7      b
3        10         6      c
```

```r
tail(dat, n=4) # to see just the bottom 4 rows
```

```
  Firs.tNum SecondNum Letter
6         7         3      f
7         6         2      g
8         5         1      h
9         4         0      i
```

Examples:

1. Combine the following vectors and identify the data type of each vector

```r
n = c(1,2,3)
s = c("aa","bb","cc","dd","ff")
c(n,s)
```

```
[1] "1"  "2"  "3"  "aa" "bb" "cc" "dd" "ff"
```

2. How many elements are there in the following vector:

```r
 a = c(65,45,2,54,87,4,32,54,56,87,8,65,3,56,67,7,8765,34,3,43,65,76,
       45,34,3,65,75,65,54,43,54,76,87,9,64,35,46,68,79,68,64,3,57,88,9,79,5)

length(a)
```

```
[1] 47
```

Recycling Rule

If two vectors are of unequal length, the shorter one will be recycled in order to match the longer vector. For example, the following vectors u and v have different lengths, and their sum is computed by recycling values of the shorter vector u.

```r
u = c(10,20,30)
v = c(1,2,3,4,5,6,7,8,9)

u+v
```

```
[1] 11 22 33 14 25 36 17 28 39
```

- Important : Longer object length should be a multiple of the shorter objects length.

Vector Index

```r
s = c("aa","bb","cc","dd","ff")
```

- What is the 3rd element of s?
- What is the 15th element of s?
- Delete the 4th element of s

```r
#Answer:

s[3] # Extracting 3rd element from S
```

```
[1] "cc"
```

```r
s[15]
```

```
[1] NA
```

```r
e =s[-4]
```

Similarly,

```r
s[c(2, 3)]
```

```
[1] "bb" "cc"
```

```
s[c(2, 3, 3)]
```

```
[1] "bb" "cc" "cc"
```

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

```
[1] "bb" "aa" "cc"
```

```
s[2:4]
```

```
[1] "bb" "cc" "dd"
```

Matrices

Matrices are similar to data.frames except that all elements must be of the same type, most commonly, numeric. To construct this data structure we use the matrix function. So for example, to write the matrix;

we write:

```
A = matrix(c(1,2,3,6,9,15),
           nrow = 2,    # number of rows
           ncol = 3,     # number of columns
           byrow = TRUE # fill the matrix by rows
           )
A
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    6    9   15
```

you can include either nrow or ncol or both.

An element at the mth row, nth column of A can be accessed by the expression A[m, n].

```r
A[2,3]  # extracting the element at the 2nd row, 3rd column
```

```
[1] 15
```

```r
A[,3]  # Extracting the 3rd column
```

```
[1]  3 15
```

```r
A[2,]  # Extracting the 2nd column
```

```
[1]  6  9 15
```

```r
A[,c(1,3)]  # Extracting the 1st and 3rd Column
```

```
     [,1] [,2]
[1,]    1    3
[2,]    6   15
```

To get the transpose of a matrix we use:

```r
B = t(A)
B
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    9
[3,]    3   15
```

Given any two matrices A and B, we can perform the operations A+B and A-B as long as the number of rows and columns agree.

- Combine the matrices B and C:

```r
C = matrix (c(4,6,8), nrow = 3)

cbind(B,C)
```

```
     [,1] [,2] [,3]
[1,]    1    6    4
[2,]    2    9    6
[3,]    3   15    8
```

- Combine matrices B and D:

```
D = matrix(c(12,53,65,77,35,98,89,85), ncol = 2, byrow = T)

rbind(B,D)
```

```
     [,1] [,2]
[1,]    1    6
[2,]    2    9
[3,]    3   15
[4,]   12   53
[5,]   65   77
[6,]   35   98
[7,]   89   85
```

Two multiply two matrices A,B (as long as the number of columns of A is equal to the number of rows of B) we use,

```
A%*%B
```

```
     [,1] [,2]
[1,]   14   69
[2,]   69  342
```

We can use the colnames, rownames, nrow, ncol, and dim functions with matrices as well.

The function "det (A)" gives us the determinant of a matrix A, and the function "solve(A)" gives us A^(-1), the inverse of A.

Change Working Directories

```
getwd()  #get current working directory

setwd("new path") # Set working directory
```

Note that the forward slash should be used as the path separator even on Windows platform.

Data Import

```
#From Text
 # can use the in-built function
```

```
#From Excel
 # can use the in-built function
```

```
#From CSV (comma separated values)

#dat_1 = read.csv("/Users/idassana/Desktop/for 515/Week 1/electronic-card-transactions-july-2
```