Introduction to R software and basic statistical data analysis

Dr. V. Geethalakshmi, Principal Scientist,
EIS Division, ICAR-CIFT , Cochin

geethasankar@gmail.com

Basics of R

R is a open source software that provides a programming environment for doing statistical data analysis. This software was written by Robert Gentleman and Ross Ihaka and the name of the software bear the name of the creators. It is a free implementation of S, another popular statistical software. R can be effectively used for data storage, data analysis and a variety of graphing functions. When you use the R program it issues a prompt when it expects input commands. The default prompt is '>'. R is object oriented and "<-" denotes the assignment operator. For example X<- 5 assigns the value 5 to X.

R works on the principle of 'functions' and objects. Functions are statements that accomplish a task (say you have a function for calculating the mean of x so that it will state mean(x) = sum(x)/length(x) where mean() is a function, sum is a function, and length is a function. Functions are denoted by (). Whatever goes in between the parentheses are arguments, so in the above examples, x is an argument to the function mean, or sum, or length. The entities that R creates and manipulates are known as *objects*. These may be variables, arrays of numbers, character strings, functions, or more general structures built from such components. Every object has its own properties, that can be explored by other functions. Documentation or help in R can be got by typing "?" in the R prompt and giving the topic on which user wants information.

There are about 25 packages supplied with R (called "standard" and "recommended" packages) and many more are available through the CRAN family of Internet sites (via https://CRAN.R-project.org) and elsewhere.

Data Structure in R

R supports all types of data. The simplest type of data is vector, which is one dimensional representation of a string of similar entities. An example of vector is this: a = ("rohu","catla","mrigal", "tilapia", "barbus"). This represents a series of strings (names of fishes). If another vector of numbers represent their mean length, say, b = (110,120,130,125,100) then it is a series of strings. Another term is a "list" which is also one dimensional, but it consists of a mixture of different types of data. For example, c = (120,"tilapia","YES") is an example of a list, one that contains a number, a string, and a logical YES in it. Extending this concept further, a matrix is two dimensional (or at least n dimensional) that contains only one type of data in it. A data frame is a two dimensional (or n dimensional) entity that contains mix of different data types in it. In terms of statistical data analysis, a data frame is the most commonly encountered rectangular dataset that most people work with.

	Linear	Rectangular
All same type	Vectors	Matrix
Mixed	List	Data Frame

Data manipulation

In general, when we work in an R session for doing statistical analysis, we first read the data, then use one or more packages, libraries, or sets of functions to work on the dataset. In order to know which directories and data are loaded, type search() and to see which objects are stored type ls() at R console.

c() command is a quick way of getting a series of values stored in a data object. Let us create a data set. Below command is used for the same :

```
>data1<-c(3,5,7,6,8,9)

>data1

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

All possible functions addition, multiplying by scalar, etc. can be applied to data1. It is possible to create another dataset from data1.

```
>data2<-c(data1,19,26,7,8)

>data2

[1] 3 5 7 6 8 9 19 26 7 8
```

Combining datasets of numeric and string types using c() command will result in creation of dataset of string type of dataset. Another command to create dataset is the scan() command.

```
> x<-scan()
1: 3
2: 5
3: 7
4: 6
5: 9
6:
Read 5 items
> x
[1] 3 5 7 6 9
```

For reading string values using scan() use
> x<-scan(what="character"))
Another way of importing data interactively into R is to use the Clipboard to copy and paste data.

The scan() command can be used with programs, such as a spreadsheet for entering data into R.

The steps to import data are:

a. If the spreadsheet data is in the form of numbers, simply type the command in R as usual before switching to the spreadsheet containing the data.
b. Highlight the necessary cells in the spreadsheet and copy them to the clipboard.
c. Then return to R and paste the data from the clipboard into R. As usual, R waits until a blank line

is entered before ending the data entry so you can continue to copy and paste more data as required.

d. Enter a blank line to complete data entry.

Reading of data from .csv file is possible with 'file=' option in the scan() function. Specify the complete path where the file is stored. For example if the .csv file is having details on fish catch from 24 zones, we give

```
> x<-scan(file="C:/Users/ My Documents/test.csv",sep=",")
Read 20 items
> x
 [1] 5821   103  6996   139  5737   139 12646   602 26500   243
[15] 8762    60  3756    34 24554   745 42258   442 19530   235

>
```

To read from file from a specific location "read.table" command can be used. For example if test.csv file contains a matrix of numbers, it can be read with the command as follows:

```
> x<-read.table(file="C:/Users/My Documents/test.csv",sep=",")
> x
   V1  V2   V3  V4   V5  V6
1  5821 103  6996 139  5737 139
2 12646 602 26500 243 10037  62
3  6087  40  8762  60  3756  34
4 24554 745 42258 442 19530 235
```

Another easy way to get data from a file is to use the file.choose() option as follows :
x<-read.table(file.choose(),sep=",",header=FALSE)
# if your data has headings in the first line header=TRUE can be used
# When no heading row is there then by default the names are v1, v2,...
# The program will prompt you to select the file from the disk.
Access to individual variables in the dataset x will be x$v1, x$v2, etc. Typing ?read.table at R console will explain more options for this command.

To get the current working directory we use

```
>getwd()
```

The working directory can be altered in R. In case you want to load files by just typing their names fro any directory, the task becomes easier if the working directory is permanently set as different directo The directory can be altered using the setwd() command as below:

```
>setwd('Desktop')
```

To read data into R, read.table() of scan() function are used. Try to convert data into a text file and th input into R. You can use the "foreign" library to import data from other statistical systems (eg SPSS, or E Info). If you are working with datasets, use the function "data.frame" or "as.data.frame" for converting yc dataset into a dataframe. For example, a typical command could run like:

```
library(foreign)
```

```
mydata <- data.frame(read.spss("myfile.sav",use.value.labels=TRUE))
```

Here, you call the library "foreign" so that you can read data that is stored in a foreign format. Then store that data in the object called "mydata". You have to remember that you need to give full path of the file that is read in. In this example, the data file was in the same folder as the working directory of R.

Once you have obtained a data frame, you can export it to different file formats. We prefer exporting it to a comma separated or tab delimited file for ease of use. Use the function "write.table" to accomplish the task. See "?write.table" in an R session for more information.

Functions in R

(1) Random number generation

runif() generates random number between 0 and 1

```
>x=runif(10)
```

generates 10 random numbers within the range 0 to 1

```
>x=100*runif(10)
```

10 random numbers between 10 and 99 can be generated

(2) **Creating new variable from data**

The subset() function forms a subset of numbers

```
>x<-c(57,26,34,23,11,7)
>subset(y,y<20)
>y
[1] 11 7
```

(3) **Matrix**
The matrix() function to define an existing object into a matrix

```
>A<-matrix(
+ c(2,4,3,1,5,7)
+nrow=2
+ncol=3
+byrow=TRUE)
```

Above code gives the following output

```
>A
     [1] [2] [3]
[1,]  2   4   3
[2,]   1  5   7
```

how to assess individual elements of a matrix

```
>A[2,3]
[1]  7
```

```
>A[2,]
[1] 1 5 7
```

(4) Numeric computations using data

```
# Numeric measures in R
> x<-c(57,26,34,23,11,7)
> mean(x)
 [1] 26.33333
>median(x)
[1] 24.5
# to get quantiles
>quantile(x)
0%  25%  50%  75% 100%
 7.0 14.0 24.5 32.0 57.0

#to specify the particular quantiles like 32% or 52%
> quantile(x,c(.32,.52,.75))
32%  52%  75%
18.2 24.8 32.0

>max(x)
[1] 57
>min(x)
[1] 7
```

Likewise the var(), sd(), cov() functions can be used to find out the variance, standard devia covariance of variables. R stores certain functions in libraries/packages which have to be installed invoked to apply them. For example to obtain skewness and kurtosis, install the package "moments' library("moments").

```
> install.packages("moments")
Installing package into 'C:/Users/GEETHALAKSHMI/Documents/R/win-library/3.4'
(as 'lib' is unspecified)
Warning in install.packages :
  cannot open URL 'http://www.stats.ox.ac.uk/pub/RWin/bin/windows/contrib
/3.4/PACKAGES.rds': HTTP status was '404 Not Found'
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/moments_0.14.zip'
Content type 'application/zip' length 40993 bytes (40 KB)
downloaded 40 KB

package 'moments' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
C:\Users\GEETHALAKSHMI\AppData\Local\Temp\RtmpIv0sce\downloaded_packages
> library("moments")

>skewness(x)
>kurtosis(x)
```

(5) Descriptive statistics

By using of "pastecs" package in R, the mean, median and other descriptive statistcs can be derived from data.

```
>install.packages("pastecs")
>library("pastecs")
>summary(x)
```

(6) Hypothesis testing

Suppose the manufacturer claims that the mean life time of a marine engine is >10000 days. In a sample of 30 engines, it was found that average life time was 9900 days. Assuming $\sigma = 120$, at 5% level of significance can we reject the manufacturers claim?

Using simple commands in R testing of single mean using t-test can be done.

```
# First declare the information in hand as variables and parameters
>xbar=9900
>mu0=10000
>sigma=120
>n=30
# Then compute the z value
>z=(xbar-mu0)/(sigma/sqrt(n))
[1] -4.56    #output
>alpha=0.05
# Now we should know the critical value at 5%
>z.alpha=qnorm(1-alpha)
>-z.alpha   #critical value
[1] -1.6449
#Since computed z is less than the critical value, the
# hypothesis is rejected
# When you have a value more than declared mean to test use z.alpha without sign for comparing.
```

In R package, a built-in dataset 'mtcars' has gas mileage data of various cars which are grouped into auto and manual. To invoke the data type 'mtcars' at R console. 'mpg' denotes the mileage. It can be accessed as 'mtcars$mpg' and used in computations.

```
> mtcars$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7
[18] 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
```

Comparison of sample means of 'auto' and manual' cars can be done using t.test() function.

```
> t.test(mpg~am,data=mtcars)

        Welch Two Sample t-test

data:  mpg by am
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.280194  -3.209684
sample estimates:
```

```
mean in group 0 mean in group 1
      17.14737        24.39231
```

Since R is a highly functional language, everything in R can be accomplished by using or writing functions. Rules for writing functions are very simple. You can declare a function like this:

```
>myfunction <- function(){statements}
```

where myfunction is the object that will be outcome of writing the function. {} indicate opening and closing of statements of the function. These are not mandatory, for instance if you plan to write a simple one-liner function, you need not put curly brackets. However, we recommend putting curly brackets to write functions for they are good practices. Semicolons are not necessary unless you want to separate two pieces of codes written in the same line.

In terms of programming in R, R itself comes with a rich set of pre-built functions that you can use. All statistical functions known to date are supported in R. Some we need almost everytime, some we will perhaps never need in a lifetime. You can easily customize which functions you need. You can and should take help of numerous in built and contributed packages that are part of R. You can learn about them in the R website and in the documentation and various other tutorials. The R base and stats package are good and powerful enough for doing most routine statistical work and drawing graphs. Type packages() at the R session to learn about the packages installed in your R.