UNIT-I

Introduction, How to run R, R Sessions and Functions, Basic Math, Variables, Data Types, Vectors, Conclusion, Advanced Data Structures, Data Frames, Lists, Matrices, Arrays, Classes.

R was created by Ross Ihaka and Robert Gentleman[at the University of Auckland, New Zealand, and is currently developed by the *R Development Core Team*, of which Chambers is a member. R is named partly after the first names of the first two R authors and partly as a play on the name of S. The project was conceived in 1992, with an initial version released in 1995 and a stable beta version in 2000.

Why R?

What Is R?

R is a scripting language for statistical data manipulation and analysis. It was inspired by, and is mostly compatible with, the statistical language S developed by AT&T. The name S, obviously standing for statistics, was an allusion to another programming language developed at AT&T with a one-letter name, C. S later was sold to a small firm, which added a GUI interface and named the result S-Plus. R has become more popular than S/S-Plus, both because it's free and because more people are contributing to it. R is sometimes called "GNU S."

Why Use R for Your Statistical Work?

Why use anything else? As the Cantonese say, yauh peng, yauh leng—"both inexpensive and beautiful."

Its virtues:

-a public-domain implementation of the widely-regarded S statistical language; R/S is the de facto standard among professional statisticians comparable, and often superior, in power to commercial products in most senses

-available for Windows, Macs, Linux

- in addition to enabling statistical operations, it's a general programming language, so that you can automate your analyses and create new functions

-object-oriented and functional programming structure

-your data sets are saved between sessions, so you don't have to reload each time

- open-software nature means it's easy to get help from the user community, and lots of new functions get contributed by users, many of which are prominent statisticians

Commands to R via text, rather than mouse clicks in a Graphical User Interface (GUI). If you can't live without GUIs, you should consider using one of the free GUIs that have been developed for R.

How to Run R

R has two modes, interactive and batch.

The former is the typical one used.

Interactive Mode

You start R by typing "R" on the command line in Linux or on a Mac, or in a Windows Run window. You'll get a greeting, and then the R prompt, >.

You can then execute R commands, as you'll see in the quick sample session discussed in Section 2.2. Or, you may have your own R code which you want to execute, say in a file z.r. You could issue the command

```
> source("z.r")
```

which would execute the contents of that file. Note by the way that the contents of that file may well just be a function you've written, say f(). In that case, "executing" the file would mean simply that the R interpreter reads in the function and stores the function's definition in memory. You could then execute the function itself by calling it from the R command line, e.g.

```
> f(12)
```

Running R in Batch Mode

Sometimes it's preferable to automate the process of running R. For example, we may wish to run an R script that generates a graph output file, and not have to bother with manually running R. Here's how it could be done. Consider the file z.r, which produces a histogram and saves it to a PDF file:

```
pdf("xh.pdf") # set graphical output file
hist(rnorm(100)) # generate 100 N(0,1) variates and plot their histogram
dev.off() # close the file
```

We could run it automatically by simply typing

```
R CMD BATCH --vanilla < z.r
```

The –vanilla option tells R not to load any startup file information, and not to save any.

Example:

```
> x <- c(1,2,4)
```

The standard assignment operator in R is <-. However, there are also ->, = and even the assign() function.

The "c" stands for "concatenate." Here we are concatenating the numbers 1, 2 and 4. Or more precisely, we are concatenating three one-element vectors consisting of those numbers. This is because any object is considered a one-element vector.

Thus we can also do, for instance,

```
> q <- c(x,x,8)
```

The Result:

q to (1,2,4,1,2,4,8).

```
> x
[1] 1 2 4
```

Working with R session

Once we are inside the R session, we can directly execute R language commands by typing them line by line. Pressing the *enter key* terminates typing of command and brings the > prompt again. In the example session below, we declare 2 variables 'a' and 'b' to have values 5 and 6 respectively, and assign their sum to another variable called 'c':

```
> a = 5
> b = 6
> c = a + b
> c
```

The value of the variable 'c' is printed as,

```
[1] 11
```

Function

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions. In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Function Definition

An R function is created by using the keyword **function. The basic syntax of an R function** definition is as follows:

```
function_name <- function(arg_1, arg_2, ...) {
Function body
}
```

Function Components

The different parts of a function are:

Function Name: This is the actual name of the function. It is stored in R environment as an object with this name.

Arguments: An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

Function Body: The function body contains a collection of statements that defines what the function does.

Return Value: The return value of a function is the last expression in the function body to be evaluated.

R has many **in-built functions which can be directly called in the program without defining**

them first. We can also create and use our own functions referred as **user defined functions.**

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
for(i in 1:a) {
b <- i^2
print(b)
}
}
```

Calling a Function

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
for(i in 1:a) {
b <- i^2
print(b)
}
}
# Call the function new.function supplying 6 as an argument.
```

Calling a Function without an Argument

```
# Create a function without an argument.
new.function <- function() {
for(i in 1:5) {
print(i^2)
}
}
# Call the function without supplying an argument.
new.function()
```

When we execute the above code, it produces the following result:

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

```
> f1 <- function(a,b) return(a+b)
> f2 <- function(a,b) return(a-b)
```

```
> f <- f1
> f(3,2)
[1] 5
> f <- f2
> f(3,2)
[1] 1
> g <- function(h,a,b) h(a,b)
> g(f1,3,2)
[1] 5
> g(f2,3,2)
[1] 1
```

The return value of function() is a function, even if you don't assign it to a variable

R can perform the usual mathematical operations

t – transpose
eigen – eigenvalues and eigenvectors
solve – inverse of matrix
ginv – generalized inverse, requires MASS package
rbind – combines vectors of observations horizontally into matrix class
cbind – combines vectors of observations vertically into matrix class

R Variable

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many R objects.

A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name , var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
_var_name	invalid	Starts with _ which is not valid

Data Types

The other R-Objects are built upon the atomic vectors.

Logical, Numeric, Integer, Complex, Character, Raw

Data Type	Example	Verify
Logical	TRUE, FALSE	v <- TRUE print(class(v)) it produces the following result − [1] "logical"
Numeric	12.3, 5, 999	v <- 23.5 print(class(v)) it produces the following result −
Integer	2L, 34L, 0L	[1] "numeric" v <- 2L print(class(v)) it produces the following result −

		[1] "integer" v <- 2+5i print(class(v))
Complex	3 + 2i	it produces the following result − [1] "complex"
Character	'a' , '"good", "TRUE", '23.4'	v <- "TRUE" print(class(v)) it produces the following result − [1] "character"
Raw	"Hello" is stored as 48 65 6c 6c 6f	v <- charToRaw("Hello") print(class(v)) it produces the following result − [1] "raw"

In R programming, the very basic data types are the R-objects called **vectors**

Vectors

To create vector with more than one element, should use **c() function**
which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

It produces the following result:
[1] "red" "green" "yellow"
[1] "character"

Data Frame

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.
BMI <-   data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

When we execute the above code, it produces the following result −

```
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
```

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

The column names should be non-empty.

The row names should be unique.

The data stored in a data frame can be of numeric, factor or character type.

Each column should contain same number of data items.

Create Data Frame

```
# Create the
data frame. emp.data <- data.frame( emp_id = c (1:5),
emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
salary = c(623.3,515.2,611.0,729.0,843.25),
start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11", "2015-03-
27")),
stringsAsFactors = FALSE )
# Print the data frame. print(emp.data)
```

Lists

Lists are the R objects which contain elements of different types like − numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values

```
# Create a list containing strings, numbers, vectors and a logical values.
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

Syntax

The basic syntax for creating a matrix in R is −

matrix(data, nrow, ncol, byrow, dimnames) Following is the description of the parameters used −

data is the input vector which becomes the data elements of the matrix.

nrow is the number of rows to be created.

ncol is the number of columns to be created.

byrow is a logical clue. If TRUE then the input vector elements are arranged by row.

dimname is the names assigned to the rows and columns.

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

When we execute the above code, it produces the following result −

```
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
```

Example

Create a matrix taking a vector of numbers as input

```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)
# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)
```

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4") colnames = c("col1", "col2", "col3")
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames)) print(P)
```

Arrays

Arrays are the R data objects which can store data in more than two dimensions. For example − If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the **array()** function. It takes vectors as input and uses the values in the **dim** parameter to create an array.

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('green','yellow'),dim = c(3,3,2))
print(a)
```

When we execute the above code, it produces the following result −

```
, , 1

     [,1]     [,2]     [,3]
[1,] "green"  "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green"

, , 2

     [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green"
[3,] "yellow" "green"  "yellow"
```

Example

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)
# Take these vectors as input to the
array. result <- array(c(vector1,vector2),dim = c(3,3,2))
```

```
print(result)
```

Classes

R is an object-oriented language. *Objects* are instances of *classes*. Classes are a bit more abstract than the data types you've met so far. Here, we'll look briefly at the concept using R's S3 classes. (The name stems from their use in the old S language, version 3, which was the inspiration for R.) Most of R is based on these classes, and they are exceedingly simple. Their instances are simply R lists but with an extra attribute: the class name.

For example, we noted earlier that the (nongraphical) output of the hist() histogram function is a list with various components, such as break and count components. There was also an *attribute*, which specified the class of the list, namely histogram.

```
> print(hn)
$breaks
[1] 400 500 600 700 800 900 1000 1100 1200 1300 1400
$counts
[1] 1 0 5 20 25 19 12 11 6 1
...
...
attr(,"class")
[1] "histogram"
```