R Programming - Introduction

R is a programming language and software environment for statistical computing and graphics supported by the R Foundation for Statistical Computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis. Polls, surveys of data miners, and studies of scholarly literature databases show that R's popularity has increased substantially in recent years.

R is a GNU package.The source code for the R software environment is written primarily in C, Fortran, and R. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems. While R has a command line interface, there are several graphical front-ends available

Evolution of R

R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

A large group of individuals has contributed to R by sending code and bug reports. Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

Features of R

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R:

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

As a conclusion, R is world's most widely used statistics programming language. It's the # 1 choice of data scientists and supported by a vibrant and talented community of contributors. R is taught in universities and deployed in mission critical business applications.

R – Basic Syntax

As a convention, we will start learning R programming by writing a "Hello, World!"
program. Depending on the needs, you can program either at R command prompt or
you
can use an R script file to write your program. Let's check both one by one.

```
> myString <- "Hello, World!"
> print ( myString)
 "Hello, World!"
```

Comments
Comments are like helping text in your R program and they are ignored by the
interpreter while executing your actual program. Single comment is written using # in
the beginning of the statement as follows:

```
# My first program in R Programming
```

R does not support multi-line comments but you can perform a trick which is
something as follows:

```
if(FALSE){
"This is a demo for multi-line comments and it should be put
inside either a single of double quote"
}
myString <- "Hello, World!"
print ( myString)
```

R – Basic Syntax

Generally, while doing programming in any programming language, you need to use
various variables to store various information. Variables are nothing but reserved
memory locations to store values. This means that, when you create a variable you
reserve some space in memory.
You may like to store information of various data types like character, wide
character,
integer, floating point, double floating point, Boolean etc. Based on the datatype of a

variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the datatype of the variable. There are many types of R-objects.

The frequently used ones are:
1. Vectors
2. Lists
3. Matrices
4. Arrays
5. Factors
6. Data Frames

The simplest of these objects is the vector object and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

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: [1] "numeric"
Integer	2L, 34L, 0L	``` v <- 2L print(class(v)) ``` it produces the following

		result: [1] "integer"
Complex	3 + 2i	v <- 2+5i print(class(v)) 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 which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types. For example, we can use many atomic vectors and create an array whose class will become array.

Vectors

When you want to create vector with more than one element, you 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))
```

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

```
[1] "red" "green" "yellow"
[1] "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.
list1 <- list(c(2,5,3),21.3,sin)
# Print the list.
print(list1)
When we execute the above code, it produces the following result:
```

```
[[1]]
[1] 2 5 3
[[2]]
[1] 21.3
[[3]]
function (x) .Primitive("sin")
```

Matrices

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

Arrays

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"
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

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

```
# Create a factor object.
factor_apple <- factor(apple_colors)
# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

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

```
[1] green green yellow red red red yellow green
Levels: green red yellow
# applying the nlevels function we can know the number of distinct values
[1] 3
```

Data Frames

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
```

R – Variables

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 Robjects. 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.

.2var_name	invalid	The starting dot is followed by a number making it invalid
_var_name	invalid	Starts with _ which is not valid

Variable Assignment

The variables can be assigned values using leftward, rightward and equal to operator. The values of the variables can be printed using print() or cat()function. The cat() function combines multiple items into a continuous print output.

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)
# Assignment using leftward operator.
var.2 <- c("learn","R")
# Assignment using rightward operator.
c(TRUE,1) -> var.3
print(var.1)
```

```
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

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

```
[1] 0 1 2 3
var.1 is 0 1 2 3
var.2 is learn R
var.3 is 1 1
```

Note: The vector c(TRUE,1) has a mix of logical and numeric class. So logical class is coerced to numeric class making TRUE as 1.

Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R -object assigned to it. So R is called a dynamically typed language, which means that we can change a variable's data type of the same variable again and again when using it in a program.

```
var_x <- "Hello"
cat("The class of var_x is ",class(var_x),"\n")
var_x <- 34.5
cat(" Now the class of var_x is ",class(var_x),"\n")
var_x <- 27L
cat(" Next the class of var_x becomes ",class(var_x),"\n")
```

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

```
The class of var_x is character
 Now the class of var_x is numeric
  Next the class of var_x becomes integer
```

Finding Variables

To know all the variables currently available in the workspace we use the ls() function. Also the **ls()** function can use patterns to match the variable names.

```
print(ls())
```

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

```
[1] "my var" "my_new_var" "my_var" "var.1"
[5] "var.2" "var.3" "var.name" "var_name2."
[9] "var_x" "varname"
```

Note: It is a sample output depending on what variables are declared in your environment.
The ls() function can use patterns to match the variable names.

```
# List the variables starting with the pattern "var".
print(ls(pattern="var"))
```

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

```
[1] "my var" "my_new_var" "my_var" "var.1"
[5] "var.2" "var.3" "var.name" "var_name2."
[9] "var_x" "varname"
```

The variables starting with dot(.) are hidden, they can be listed using "all.names=TRUE" argument to ls() function.

```
print(ls(all.name=TRUE))
```

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

```
[1] ".cars" ".Random.seed" ".var_name" ".varname" ".varname2"
[6] "my var" "my_new_var" "my_var" "var.1" "var.2"
[11]"var.3" "var.name" "var_name2." "var_x"
```

Deleting Variables

Variables can be deleted by using the rm() function. Below we delete the variable var.3. On printing the value of the variable error is thrown.

```
rm(var.3)
print(var.3)
```

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

```
[1] "var.3"
Error in print(var.3) : object 'var.3' not found
```

All the variables can be deleted by using the rm() and ls() function together.

```
rm(list=ls())
print(ls())
```

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

```
character(0)
```

R – Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

We have the following types of operators in R programming:
- **Arithmetic Operators**
- **Relational Operators**
- **Logical Operators**
- **Assignment Operators**
- **Miscellaneous Operators**

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example

| + | Adds two vectors | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v+t)
``` |
| | | it produces the following result − |
| | | `[1] 10.0  8.5  10.0` |
| − | Subtracts second vector from the first | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v-t)
``` |
| | | it produces the following result − |
| | | `[1] -6.0  2.5  2.0` |
| * | Multiplies both vectors | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v*t)
``` |
| | | it produces the following result − |
| | | `[1] 16.0 16.5 24.0` |
| / | Divide the first vector with the second | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v/t)
``` |
| | | When we execute the above code, it produces the following result − |

		`[1] 0.250000 1.833333 1.500000`
| %% | Give the remainder of the first vector with the second | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v%%t)
```  it produces the following result −  `[1] 2.0 2.5 2.0` |
| %/% | The result of division of first vector with second (quotient) | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v%/%t)
```  it produces the following result −  `[1] 0 1 1` |
| ^ | The first vector raised to the exponent of second vector | ```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v^t)
```  it produces the following result −  `[1]  256.000  166.375 1296.000` |

Relational Operators

Following table shows the relational operators supported by R language. Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example
| > | Checks if each element of the first vector is greater than the corresponding element of the second vector. | ```r
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v>t)
```<br><br>it produces the following result _<br><br>`[1] FALSE  TRUE FALSE FALSE` |
| < | Checks if each element of the first vector is less than the corresponding element of the second vector. | ```r
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v < t)
```<br><br>it produces the following result _<br><br>`[1]  TRUE FALSE  TRUE FALSE` |
| == | Checks if each element of the first vector is equal to the corresponding element of the second vector. | ```r
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v == t)
```<br><br>it produces the following result _<br><br>`[1] FALSE FALSE FALSE  TRUE` |

| <= | Checks if each element of the first vector is less than or equal to the corresponding element of the second vector. | ```r
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v<=t)
```<br><br>it produces the following result −<br><br>```
[1]  TRUE FALSE  TRUE  TRUE
``` |
| >= | Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector. | ```r
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v>=t)
```<br><br>it produces the following result −<br><br>```
[1] FALSE  TRUE FALSE  TRUE
``` |
| != | Checks if each element of the first vector is unequal to the corresponding element of the second vector. | ```r
v <- c(2,5.5,6,9)
t <- c(8,2.5,14,9)
print(v!=t)
```<br><br>it produces the following result −<br><br>```
[1]  TRUE  TRUE  TRUE FALSE
``` |

Logical Operators

Following table shows the logical operators supported by R language. It is applicable only to vectors of type logical, numeric or complex. All numbers greater than 1 are considered as logical value TRUE.

Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.	```v <- c(3,1,TRUE,2+3i)```
		```t <- c(4,1,FALSE,2+3i)```
		```print(v&t)```
		
		it produces the following result –
		
		```[1] TRUE TRUE FALSE TRUE```

|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	```v <- c(3,0,TRUE,2+2i)```
		```t <- c(4,0,FALSE,2+3i)```
		```print(v|t)```
		
		it produces the following result –
		
		```[1] TRUE FALSE TRUE TRUE```

!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	```v <- c(3,0,TRUE,2+2i)``` ```print(!v)```

		it produces the following result − `[1] FALSE   TRUE FALSE FALSE`

The logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

Operator	Description	Example
&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	```r v <- c(3,0,TRUE,2+2i) t <- c(1,3,TRUE,2+3i) print(v&&t) ``` it produces the following result − `[1] TRUE`
||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	```r v <- c(0,0,TRUE,2+2i) t <- c(0,3,TRUE,2+3i) print(v||t) ``` it produces the following result − `[1] FALSE`

Assignment Operators

These operators are used to assign values to vectors.

Operator	Description	Example
| <−<br><br>or<br><br>=<br><br>or<br><br><<− | Called Left Assignment | ```r
v1 <- c(3,1,TRUE,2+3i)
v2 <<- c(3,1,TRUE,2+3i)
v3 = c(3,1,TRUE,2+3i)
print(v1)
print(v2)
print(v3)
```<br><br>it produces the following result −<br><br>```
[1] 3+0i 1+0i 1+0i 2+3i
[1] 3+0i 1+0i 1+0i 2+3i
[1] 3+0i 1+0i 1+0i 2+3i
``` |
| -><br><br>or<br><br>->> | Called Right Assignment | ```r
c(3,1,TRUE,2+3i) -> v1
c(3,1,TRUE,2+3i) ->> v2
print(v1)
print(v2)
```<br><br>it produces the following result −<br><br>```
[1] 3+0i 1+0i 1+0i 2+3i
[1] 3+0i 1+0i 1+0i 2+3i
``` |

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logical computation.

Operator	Description	Example

:	Colon operator. It creates the series of numbers in sequence for a vector.	```r\nv <- 2:8\nprint(v)\n```
		
		it produces the following result —
		
		```\n[1] 2 3 4 5 6 7 8\n```

%in%	This operator is used to identify if an element belongs to a vector.	```r\nv1 <- 8\nv2 <- 12\nt <- 1:10\nprint(v1 %in% t)\nprint(v2 %in% t)\n```
		
		it produces the following result —
		
		```\n[1] TRUE\n[1] FALSE\n```

%*%	This operator is used to multiply a matrix with its transpose.	```r\nM = matrix(c(2,6,5,1,10,4),\nnrow = 2,ncol = 3,byrow =\nTRUE)\nt = M %*% t(M)\nprint(t)\n``` it produces the following result —

			[,1] [,2]
		[1,] 65 82
		[2,] 82 117

Transpose of a Matrix

A matrix which is formed by turning all the rows of a given matrix into columns and vice-versa. The transpose of matrix A is written A^T.

$$\text{Example:} \quad \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

How to Multiply Matrices

A Matrix is an array of numbers:

$$\begin{bmatrix} 6 & 4 & 24 \\ 1 & -9 & 8 \end{bmatrix}$$

A Matrix

(This one has 2 Rows and 3 Columns)

To multiply a matrix by a single number is easy:

$$2 \times 4 = 8$$

$$2 \times \begin{bmatrix} 4 & 0 \\ 1 & -9 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 2 & -18 \end{bmatrix}$$

These are the calculations:

$2 \times 4 = 8$ $2 \times 0 = 0$

$2 \times 1 = 2$ $2 \times -9 = -18$

We call the number ("2" in this case) a **scalar**, so this is called "scalar multiplication".

Multiplying a Matrix by Another Matrix

But to multiply a matrix **by another matrix** we need to do the "dot product" of rows and columns ... what does that mean? Let us see with an example:

To work out the answer for the **1st row** and **1st column**:

"Dot Product"

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & \\ & \end{bmatrix}$$

The "Dot Product" is where we **multiply matching members**,

then sum up:

$$(1, 2, 3) \bullet (7, 9, 11) = 1{\times}7 + 2{\times}9 + 3{\times}11 = 58$$

We match the 1st members (1 and 7), multiply them, likewise for the 2nd members (2 and 9) and the 3rd members (3 and 11), and finally sum them up.

Want to see another example? Here it is for the 1st row and **2nd column**:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ & \end{bmatrix}$$

$$(1, 2, 3) \bullet (8, 10, 12) = 1{\times}8 + 2{\times}10 + 3{\times}12 = 64$$

We can do the same thing for the **2nd row** and **1st column**:

$$(4, 5, 6) \bullet (7, 9, 11) = 4{\times}7 + 5{\times}9 + 6{\times}11 = 139$$

And for the **2nd row** and **2nd column**:

$$(4, 5, 6) \bullet (8, 10, 12) = 4{\times}8 + 5{\times}10 + 6{\times}12 = 154$$

And we get:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix} \checkmark$$

DONE!