R over view, basic syntax data types, variable operators, decision making loops, functions

---

R overview:

- R is a programming language and environment commonly used in statistical computing, data analytics and scientific research.
- It is one of the most popular languages used by statisticians, data analysts, researchers and marketers to retrieve, clean, analyze, visualize and present data.
- 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.
- This programming language was named **R**, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka)
- The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions.
- R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.
- Due to its expressive syntax and easy-to-use interface, it has grown in popularity in recent years.

Why use R for statistical computing and graphics?

1. **R is open source and free!**
   R is free to download as it is licensed under the terms of GNU General Public license.
   There's more, most R packages are available under the same license so you can use them, even in commercial applications without having to call your lawyer.
2. **R is popular - and increasing in popularity**
   IEEE publishes a list of the most popular programming languages each year. R was ranked 5th in 2016, up from 6th in 2015. It is a big deal for a domain-specific language like R to be more popular than a general purpose language like C#.

This not only shows the increasing interest in R as a programming language, but also of the fields like Data Science and Machine Learning where R is commonly used.

3. **R runs on all platforms**
   You can find distributions of R for all popular platforms - Windows, Linux and Mac.

   R code that you write on one platform can easily be ported to another without any issues. Cross-platform interoperability is an important feature to have in today's computing world.

4. **Learning R will increase your chances of getting a job**
   According to the Data Science Salary Survey conducted by O'Reilly Media in 2014, data scientists are paid a median of $98,000 worldwide. The figure is higher in the US - around $144,000.

5. R **is being used by the biggest tech giants**
   Adoption by tech giants is always a sign of a programming language's potential. Today's companies don't make their decisions on a whim. Every major decision has to be backed by concrete analysis of data. Companies using R language are Google,Microsoft,Ford,Twitter etc.

Features of R :

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

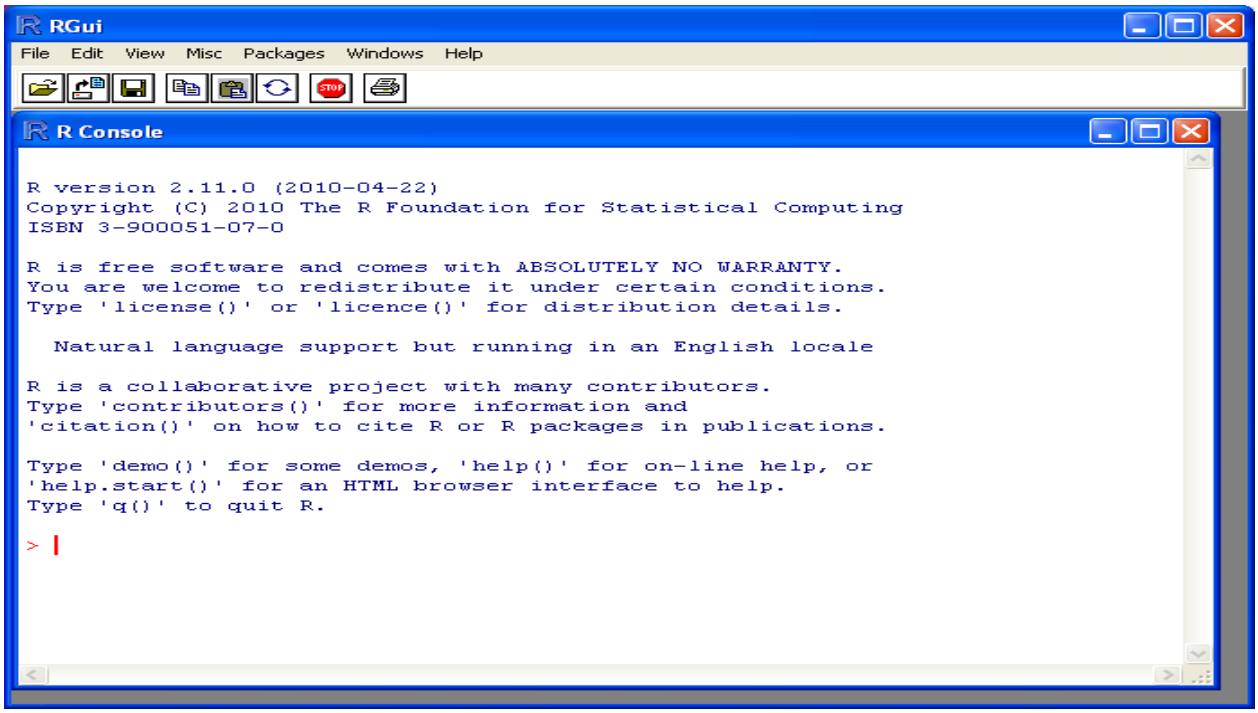
- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors 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.
- R is free, open source, powerful and highly extensible.

- The CRAN (The Comprehensive R Archive Network) package repository features has more than 8270 available packages.
- R is platform-independent, so you can use it on any operating system

Installing R on a Windows PC:

To install R on your Windows computer, follow these steps:

1. Go to http://ftp.heanet.ie/mirrors/cran.r-project.org.
2. Under "Download and Install R", click on the "Windows" link.
3. Under "Subdirectories", click on the "base" link.
4. On the next page, you should see a link saying something like "Download R 3.4.3 for Windows" (or R X.X.X, where X.X.X gives the version of R, eg. R 3.4.3). Click on this link.
5. You may be asked if you want to save or run a file "R-3.4.3-win32.exe". Choose "Save" and save the file on the Desktop. Then double-click on the icon for the file to run it.
6. You will be asked what language to install it in - choose English.
7. The R Setup Wizard will appear in a window. Click "Next" at the bottom of the R Setup wizard window.
8. The next page says "Information" at the top. Click "Next" again.
9. The next page says "Information" at the top. Click "Next" again.
10. The next page says "Select Destination Location" at the top. By default, it will suggest to install R in "C:\Program Files" on your computer.
11. Click "Next" at the bottom of the R Setup wizard window.
12. The next page says "Select components" at the top. Click "Next" again.
13. The next page says "Startup options" at the top. Click "Next" again.
14. The next page says "Select start menu folder" at the top. Click "Next" again.
15. The next page says "Select additional tasks" at the top. Click "Next" again.
16. R should now be installed. This will take about a minute. When R has finished, you will see "Completing the R for Windows Setup Wizard" appear. Click "Finish".
17. To start R, you can either follow step 18, or 19:
18. Check if there is an "R" icon on the desktop of the computer that you are using. If so, double-click on the "R" icon to start R. If you cannot find an "R" icon, try step 19 instead.
19. Click on the "Start" button at the bottom left of your computer screen, and then choose "All programs", and start R by selecting "R" (or R X.X.X, where X.X.X gives the version of R, eg. R 3.4.3) from the menu of programs.
20. The R console (a rectangle) should pop up:

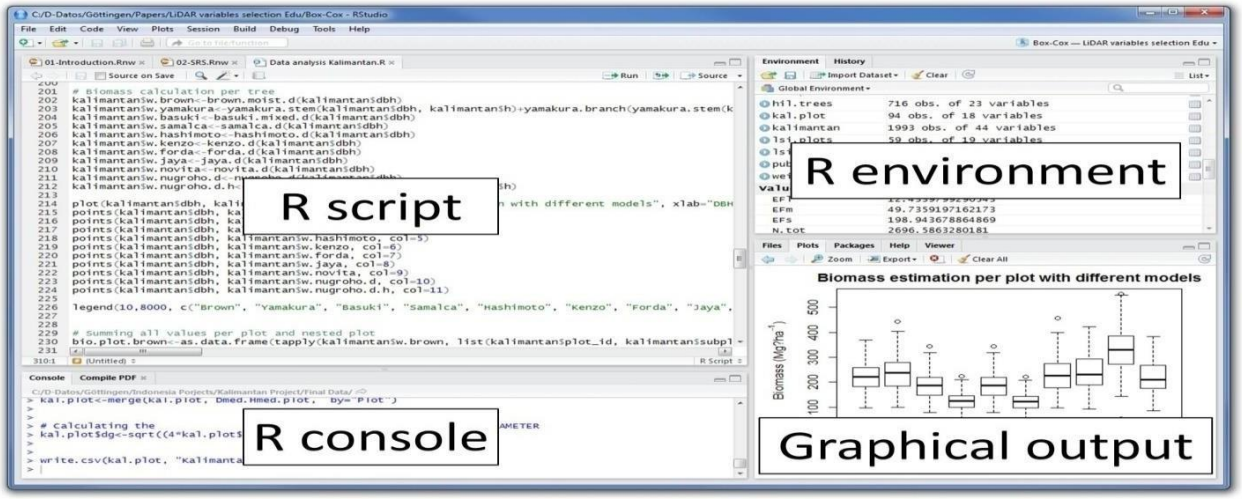


How to install R / R Studio:

For Windows users, R Studio is available for Windows Vista and above versions.

Follow the steps below for installing R Studio:

1. Go to https://www.rstudio.com/products/rstudio/download/
2. In 'Installers for Supported Platforms' section, choose and click the R Studio installer based on your operating system. The download should begin as soon as you click.
3. Click Next..Next..Finish.
4. Download Complete.
5. To Start R Studio, click on its desktop icon or use 'search windows' to access the program. It looks like this:



Let's quickly understand the interface of R Studio:

1. **R Console:** This area shows the output of code you run. Also, you can directly write codes in console. Code entered directly in R console cannot be traced later. This is where R script comes to use.
2. **R Script:** As the name suggest, here you get space to write codes. To run those codes, simply select the line(s) of code and press Ctrl + Enter. Alternatively, you can click on little 'Run' button location at top right corner of R Script.
3. **R environment:** This space displays the set of external elements added. This includes data set, variables, vectors, functions etc. To check if data has been loaded properly in R, always look at this area.
4. **Graphical Output:** This space display the graphs created during exploratory data analysis. Not just graphs, you could select packages, seek help with embedded R's official documentation.

How to install R Packages:

Most data handling tasks can be performed in 2 ways: Using R packages and R base functions. To install a package, simply type:

```
install.packages("package name")
```

R - Basic Syntax:

You will type R commands into the R console in order to carry out analyses in R.

In the R console you will see:

```
>
```

This is the R prompt. We type the commands needed for a particular task after this prompt..

Once you have started R, you can start typing in commands, and the results will be calculated immediately, for example:

Ex: > 2*3
[1] 6
> 10-3
[1] 7

Variables in R

Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.

Rules for writing Identifiers in R

1. Identifiers can be a combination of letters, digits, period (.) and underscore (_).
2. It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.
3. Reserved words in R cannot be used as identifiers.

Valid identifiers in R

total, Sum, .fine.with.dot, this_is_acceptable, Number5

Invalid identifiers in R

tot@l, 5um, _fine, TRUE, .0ne

All variables (scalars, vectors, matrices, etc.) created by R are called *objects*. In R, we assign values to variables using an arrow and equals to operators.

For example, we can assign the value 2*3 to the variable *x* using the command:

EX: > x <- 2*3
OR
> x=2*3
OR
> 2*3->x

To view the contents of any R object, just type its name, and the contents of that R object will be displayed:

Ex: > x

[1] 6

OR

>print(x)

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
```

Constants in R

Constants, as the name suggests, are entities whose value cannot be altered. Basic types of constant are numeric constants and character constants.

Numeric Constants

All numbers fall under this category. They can be of type integer, double or complex.
It can be checked with the **typeof()** function.
Numeric constants followed by L are regarded as integer and those followed by i are regarded as complex.

```
> typeof(5)
[1] "double"


> typeof(5L)
[1] "integer"
> typeof(5i)
[1] "complex"
```

Numeric constants preceded by 0x or 0X are interpreted as hexadecimal numbers.

```
> 0xff
[1] 255
> 0XF + 1
```

```
[1] 16
```

Character Constants

Character constants can be represented using either single quotes (') or double quotes (") as delimiters.

```
> 'example'
[1] "example"
> typeof("5")
[1] "character"
```

Built-in Constants

Some of the built-in constants defined in R along with their values is shown below.

```
> LETTERS
 [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"

> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

> pi
 [1] 3.141593

> month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"

> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

R "Hello World" Program:

A simple program to display "Hello World!" on the screen using print() function.

Example: Hello World Program

```
> # We can use the print() function
```

```
> print("Hello World!")
[1] "Hello World!"
> # Quotes can be suppressed in the output
> print("Hello World!", quote = FALSE)
[1] Hello World!
> # If there are more than 1 item, we can concatenate using paste()
> print(paste("How","are","you?"))
[1] "How are you?"
```

In this program, we have used the built-in function `print()` to print the string `Hello World!`

The quotes are printed by default. To avoid this we can pass the argument `quote = FALSE`.

If there are more than one item, we can use the `paste()` or `cat()` function to concatenate the strings together.

Example: Take input from user

```
my.name <- readline(prompt="Enter name: ")
my.age <- readline(prompt="Enter age: ")


# convert character into integer
my.age <- as.integer(my.age)


print(paste("Hi,", my.name, "next year you will be", my.age+1, "years old."))
```

Output

```
Enter name: Mary
Enter age: 17
[1] "Hi, Mary next year you will be 18 years old."
```

R Reserved Words

Reserved words in R programming are a set of words that have special meaning and cannot be used as an identifier (variable name, function name etc.).

Here is a list of reserved words in the R's parser.

Reserved words in R

if	else	repeat	while	function
for	in	next	break	TRUE
FALSE	NULL	Inf	NaN	NA
NA_integer_	NA_real_	NA_complex_	NA_character_	...

This list can be viewed by typing help(reserved) or ?reserved at the R command prompt as follows.

```
> ?reserved
```

R - Data Types:

There are several basic data types in R which are of frequent occurrence in coding R calculations and programs. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects.

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- 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.

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] [,2] [,3]
[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.

Factors are created using the **factor()** function.The **nlevels** functions gives the count of levels.

```
# 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  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-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	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9)

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

		[1] FALSE TRUE FALSE TRUE
!=	Checks if each element of the first vector is unequal to the corresponding element of the second vector.	v <- c(2,5.5,6,9) t <- c(8,2.5,14,9) print(v!=t) it produces the following result – [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	v <- c(3,0,TRUE,2+2i) t <- c(4,0,FALSE,2+3i) print(v|t) it produces the following result –

	elements is TRUE.	[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.	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.	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
<– or = or <<–	Called Left Assignment	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) it produces the following result – [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
–> or –>>	Called Right Assignment	c(3,1,TRUE,2+3i) -> v1 c(3,1,TRUE,2+3i) ->> v2 print(v1) print(v2) it produces the following result – [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.

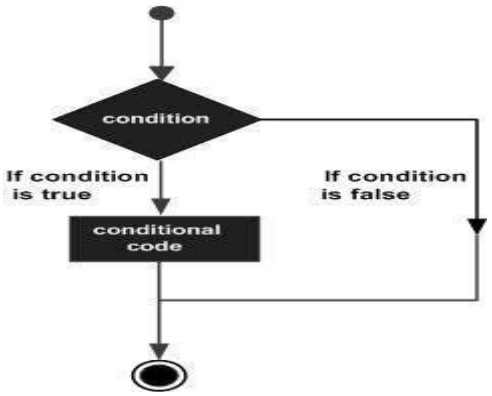
Operator	Description	Example
:	Colon operator. It creates the	v <- 2:8

	series of numbers in sequence for a vector.	print(v) it produces the following result – [1] 2 3 4 5 6 7 8
%in%	This operator is used to identify if an element belongs to a vector.	v1 <- 8 v2 <- 12 t <- 1:10 print(v1 %in% t) print(v2 %in% t) it produces the following result – [1] TRUE [1] FALSE
%*%	This operator is used to multiply a matrix with its transpose.	M = matrix(c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE) t = M %*% t(M) print(t) it produces the following result – [,1] [,2] [1,] 65 82 [2,] 82 117

R - Decision making:

Decision making structures require the programmer to specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages –



R provides the following types of decision making statements.

1. if Statement

2. If -else Statement

3. if...else if...else Statement

4. Switch Statement

If Statement:

An if statement consists of a Boolean expression followed by one or more statements.

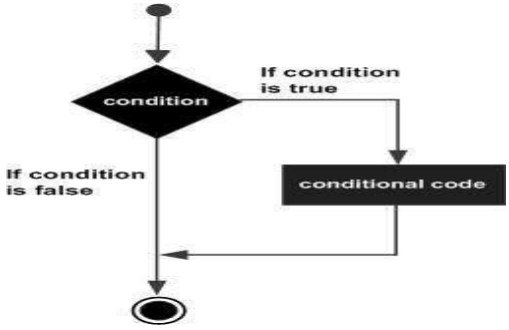
Syntax

The basic syntax for creating an if statement in R is –

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
}
```

If the Boolean expression evaluates to be true, then the block of code inside the if statement will be executed. If Boolean expression evaluates to be false, then the first set of code after the end of the if statement (after the closing curly brace) will be executed.

Flow Diagram



Example

```
x <- 30L
if(is.integer(x)) {
  print("X is an Integer")
}
```

When the above code is compiled and executed, it produces the following result –

```
[1] "X is an Integer"
```

If...Else Statement

An **if** statement can be followed by an optional **else** statement which executes when the boolean expression is false.

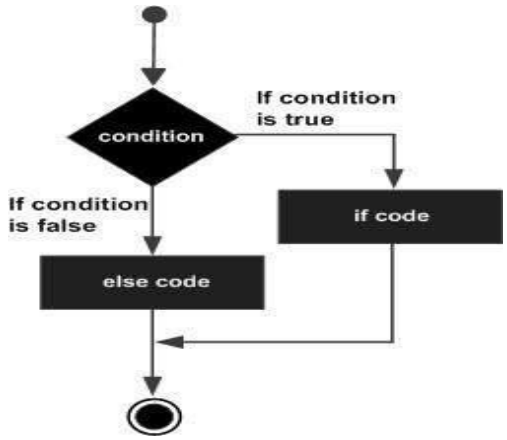
Syntax

The basic syntax for creating an **if...else** statement in R is –

```
if(boolean_expression) {
   // statement(s) will execute if the boolean expression is true.
} else {
   // statement(s) will execute if the boolean expression is false.
}
```

If the Boolean expression evaluates to be **true**, then the **if block** of code will be executed, otherwise **else block** of code will be executed.

Flow Diagram



Example

```
x <- c("what","is","truth")
if("Truth" %in% x) {
   print("Truth is found")
} else {
   print("Truth is not found")
}
```

When the above code is compiled and executed, it produces the following result –

```
[1] "Truth is not found"
```

Here "Truth" and "truth" are two different strings.

The if...else if...else Statement

An **if** statement can be followed by an optional **else if...else** statement, which is very useful to test various conditions using single if...else if statement.

When using **if**, **else if**, **else** statements there are few points to keep in mind.

- An **if** can have zero or one **else** and it must come after any **else if**s.
- An **if** can have zero to many **else if's** and they must come before the else.
- Once an **else if** succeeds, none of the remaining **else if**'s or **else**'s will be tested.

Syntax

The basic syntax for creating an **if...else if...else** statement in R is –

```
if(boolean_expression 1) {
   // Executes when the boolean expression 1 is true.
} else if( boolean_expression 2) {
   // Executes when the boolean expression 2 is true.
} else if( boolean_expression 3) {
   // Executes when the boolean expression 3 is true.
} else {
   // executes when none of the above condition is true.
}
```

Example

```
x <- c("what","is","truth")
if("Truth" %in% x) {
print("Truth is found the first time")
} else if ("truth" %in% x) {
print("truth is found the second time")
} else {
print("No truth found")
}
```

When the above code is compiled and executed, it produces the following result –

```
[1] "truth is found the second time
```

Switch Statement:

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax :The basic syntax for creating a switch statement in R is –

```
switch(expression, case1, case2, case3....)
```

The following rules apply to a switch statement –

- If the value of expression is not a character string it is coerced to integer.
- You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
- If the value of the integer is between 1 and nargs()–1 (The max number of arguments)then the corresponding element of case condition is evaluated and the result returned.
- If expression evaluates to a character string then that string is matched (exactly) to the names of the elements.
- If there is more than one match, the first matching element is returned.

Example

```
x <- switch( 3,
   "first",
```

```
 "second",
  "third",
  "fourth"
 )
print(x)
```