# Basic R

1. Basic Syntax:

   1) Example ❶:

| (R) Code | Output Program |
| --- | --- |
| myString <- "Hello, World!"<br><br>print (myString) | [1] "Hello, World!" |

---

Here first statement in code defines a string variable myString, where we assign a string "Hello, World!" and then next statement print() is being used to print the value stored in variable myString.

---

2. Data Types:

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

1) Example ❶: vectors

(R) Code

```
apple <- c("red","green","yellow")
print(apple)
print(class(apple))
```

Output Program

```
[1] "character"
```

---

1-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.

---

2-To get class of vector use class(Name_of_vector).

---

2) Example ❷:

(C++) Code

```
list1 <- list(c(2,5,3),21.3,sin)
print(list1)
```

Output Program

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

[[2]]
[1] 21.3


[[3]]
function (x)  .Primitive("sin")
```

---

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.

---

3) Example ❸: Matrix

(R) Code

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

print(M)
```

Output Program

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

---

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

---

4) Example 4:Array

(R) Code

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

Output Program

```
, , 1

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

, , 2

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

---

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 above example we create an array with two elements which are 3x3 matrices each

---

5) Example 5: Factor

(R) Code

```
apple_colors <-
c("green","green","yellow","red"
,"red","red","green")

factor_apple <- factor(apple_colors)

print(factor_apple)

print(nlevels(factor_apple))
```

Output Program

```
[1] green green yellow red red red green
 levels:green yellow red
 3
```

3) DataFrames:

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.

(R) Code

```
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)
```

Output Program

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

4) 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.

a. 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.

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.

b. 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.

(R) Code

```
var.1 = c(0,1,2,3)

var.2 <- c("learn","R")

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")

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")
```

## Output Program

```
[1] 0 1 2 3
var.1 is  0 1 2 3
var.2 is  learn R
var.3 is  1 1
The class of var_x is  character
 Now the class of var_x is  numeric
 Next the class of var_x becomes  integer
```

5) 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 in R programming:

a. Arithmetic Operators
b. Relational Operators
c. Logical Operators
d. Assignment Operators
e. Miscellaneous Operators

- Arithmetic Operators:
  (+/-) Adds and Subtracts two vectors
  (*) Multiplies both vectors
  (/) Divide the first vector with the second
  (%%) Give the remainder of the first vector with the second
  (%/%) The result of division of first vector with second (quotient)
  (^) The first vector raised to the exponent of second vector
- Relational Operators:
  >, <, == , >= , <= , !=
- Logical Operators:
  (&) 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.
  (|) 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.
  (!) It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.
  (&&) Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.
  (||) Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE
- Assignment Operators
  (<–) or (=) or («–) Called Left Assignment
  (–>) or (–») Called Right Assignment)

## (R) Code

```
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v+t)
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v%/%t)
v <- c( 2,5.5,6)
t <- c(8, 3, 4)
print(v^t)

var=2<3
print(var)

print(v||t)
print(!t)

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)

c(3,1,TRUE,2+3i) -> v1
c(3,1,TRUE,2+3i) ->> v2
print(v1)
print(v2)
```

## Output Program

```
[1] 10.0  8.5 10.0
[1] 0 1 1
[1]  256.000  166.375 1296.000
[1] TRUE
[1] TRUE
[1] FALSE FALSE FALSE
[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
[1] 3+0i 1+0i 1+0i 2+3i
```

## Decision Making:

### (R) Code

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

#If else
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")
}


#Switch
x <- switch(
3,
"first",
"second",
"third",
"fourth"
)
print(x)

#Loops

#While
v <- c("Hello","while loop")
cnt <- 2
while (cnt < 7) {
print(v)
cnt = cnt + 1
}

#For loop
v <- LETTERS[1:4]
for ( i in v) {
print(i)
}

#Loop Control Statements
#Break
v <- c("Hello","loop")
cnt <- 2
repeat {
print(v)
cnt <- cnt + 1
if(cnt > 5) {
break
}
}

#Next:like continue
v <- LETTERS[1:6]
for ( i in v) {
if (i == "D") {
next
}
print(i)
}
```

## Output Program

```
[1] "X is an Integer"
[1] "truth is found the second time"
[1] "third"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "Hello"      "while loop"
[1] "A"
[1] "B"
[1] "C"
[1] "D"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "Hello" "loop"
[1] "A"
[1] "B"
[1] "C"
[1] "E"
[1] "F"
```

7. R – Functions:

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
}

There are two types of functions:

1-Built-in Function:

Simple examples of in-built functions are seq(), mean(), max(), sum(x)and paste(...) etc. They are directly called by user written programs. You can refer most widely used R functions.

2-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.

## (R) Code

```
# Create a sequence of numbers from 20 to 30.
print(seq(20,30))

# Find mean of numbers from 25 to 30.
print(mean(25:30))

# Find sum of numbers frm 1 to 5.
print(sum(1:5))#5*(6)/2:algebric series

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

## Output Program

```
[1] 20 21 22 23 24 25 26 27 28 29 30
[1] 27.5
[1] 15
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 3
```

# Data Structures

1. String:
   Any value written within a pair of single quote or double quotes in R is treated as a string. Internally R stores every string within double quotes, even when you create them with single quote. Rules Applied in String Construction
   • The quotes at the beginning and end of a string should be both double quotes or both single quote. They can not be mixed.
   • Double quotes can be inserted into a string starting and ending with single quote.
   • Single quote can be inserted into a string starting and ending with double quotes.
   • Double quotes can not be inserted into a string starting and ending with double quotes.
   • Single quote can not be inserted into a string starting and ending with single quote.

   Operations:

   a-String Manipulation
   Concatenating Strings - paste() function
   Many strings in R are combined using the paste() function. It can take any number of arguments to be combined together.
   Syntax:
   The basic syntax for paste function is - paste(..., sep = " ", collapse = NULL)
   Following is the description of the parameters used :
   • ... represents any number of arguments to be combined.
   • sep represents any separator between the arguments. It is optional.
   • collapse is used to eliminate the space in between two strings. But not the space within two words of one string.

   b-Counting number of characters in a string - nchar() function
   This function counts the number of characters including spaces in a string.
   Syntax:
   The basic syntax for nchar() function is - nchar(x) Following is the description of the parameters used:
   - x is the vector input

   c-Changing the case - toupper() & tolower() functions
   These functions change the case of characters of a string. Subsubsection
   Syntax:
   The basic syntax for toupper() & tolower() function is toupper(x) ,tolower(x)
   Following is the description of the parameters used :
   - x is the vector input.

d-Extracting parts of a string - substring() function

This function extracts parts of a String.

Syntax:

The basic syntax for substring() function is -substring(x,first,last)

Following is the description of the parameters used :

- x is the character vector input.
- first is the position of the first character to be extracted.
- last is the position of the last character to be extracted.

## (R) Code

```
#Examples of Valid Strings
#Following examples clarify the rules about creating a string in R.
a <- 'Start and end with single quote'
print(a)
b <- "Start and end with double quotes"
print(b)
c <- "single quote ' in between double quotes"
print(c)
d <- 'Double quotes " in between single quote'
print(d)

#String Manipulation
#Example
a <- "Hello"
b <- 'How'
c <- "are you? "
print(paste(a,b,c))
print(paste(a,b,c, sep = "-"))
print(paste(a,b,c, sep = "", collapse = ""))
#Example
result <- nchar("Count the number of characters")
print(result)

# Changing to Upper case.
result <- toupper("Changing To Upper")
print(result)
# Changing to lower case.
result <- tolower("Changing To Lower")
print(result)

#Example
# Extract characters from 5th to 7th position.
result <- substring("Extract", 5, 7)
print(result)
```

## Output Program

```
[1] "Start and end with single quote"
[1] "Start and end with double quotes"
[1] "single quote ' in between double quotes"
[1] "Double quotes \" in between single quote"
[1] "Hello How are you? "
[1] "Hello-How-are you? "
[1] "HelloHoware you? "
[1] 30
[1] "CHANGING TO UPPER"
[1] "changing to lower"
[1] "act"
```

## 2. R - Vectors

**Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical, integer, double, complex, character and raw.**

### 1)Example ❶:

(R) Code

```
# Atomic vector of type character.
print("abc");
# Atomic vector of type double.
print(12.5)
# Atomic vector of type integer.
print(63L)
# Atomic vector of type logical.
print(TRUE)
# Atomic vector of type complex.
print(2+3i)
# Atomic vector of type raw.
print(charToRaw('hello'))

#Multiple Elements Vector
#Using colon operator with numeric data
# Creating a sequence from 5 to 13.
v <- 5:13
print(v)

# Creating a sequence from 6.6 to 12.6.
v <- 6.6:12.6
print(v)
v <- 3.8:11.4
print(v)

#Using sequence (Seq.) operator
# Create vector with elements from 5 ato 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```

Output Program

```
[1] "abc"
[1] 12.5
[1] 63
[1] TRUE
[1] 2+3i
[1] 68 65 6c 6c 6f
[1]  5  6  7  8  9 10 11 12 13
[1]  6.6  7.6  8.6  9.6 10.6 11.6 12.6
[1]  3.8  4.8  5.8  6.8  7.8  8.8  9.8 10.8
[1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

---

when you write just one value in R, it becomes a vector of length 1 and belongs to one of the above.

2) Example ❷:

## (R) Code

```
#Accessing Vector Elements

t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")

u <- t[c(2,3,6)]

print(u)

# Accessing vector elements using logical indexing.

v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]

print(v)

# Accessing vector elements using negative indexing.

x <- t[c(-2,-5)]

print(x)

# Accessing vector elements using 0/1 indexing.

y <- t[c(0,0,1,0,0,0,0)]

print(y)



#Vector Manipulation

v1 <- c(3,8,4,5,0,11)

v2 <- c(4,11)

# V2 becomes c(4,11,4,11,4,11)

add.result <- v1+v2

print(add.result)

sub.result <- v1-v2

print(sub.result)
```

## Output Program

```
[1] "Mon" "Tue" "Fri"
[1] "Sun" "Fri"
[1] "Sun" "Tue" "Wed" "Fri" "Sat"
[1] "Sun"
[1]  7 19  8 16  4 22
[1] -1 -3  0 -6 -4  0
```

---

1-Elements of a Vector are accessed using indexing. The [ ] brackets are used for indexing. Indexing starts

with position 1. Giving a negative value in the index drops that element from result.TRUE, FALSE or 0

and 1 can also be used for indexing.

2-Vector element recycling If we apply arithmetic operations to two vectors of unequal length,

then the elements of the shorter vector are recycled to complete the operations.

---

### 3)Example ❸1: Matrix

## (R) Code

```
v <- c(3,8,4,5,0,11, -9, 304)
sort.result <- sort(v)
print(sort.result)
revsort.result <- sort(v,
decreasing = TRUE)

print(revsort.result)
v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
print(sort.result)
 revsort.result <- sort(v, decreasing =
TRUE)
print(revsort.result)
```

## Output Program

```
[1] -9  0  3  4  5  8  11  304
[1] 304  11  8  5  4  3  0 -9
[1] “Blue” “Red” “violet” “yellow”
[1] “yellow” “violet” “Red” “Blue”
```

3. 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.

The list elements can be given names and they can be accessed using these names.
Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list.But we can update any element.
You can merge many lists into one list by placing all the lists inside one list() function.

A list can be converted to a vector so that the elements of the vector can be used for further manipulation.
All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this
conversion, we use the unlist() function. It takes the list as input and produces a vector.

## (R) Code

```
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,2,8), nrow = 2),
list("green",12.3))
print(list_data)

names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
# Access the first element of the list.
print(list_data[1])
# Access the thrid element. As it is also a list, all its elements will be printed.
print(list_data[3])
# Access the list element using the name of the element.
print(list_data$A_Matrix)

# Add element at the end of the list.
list_data[4] <- "New element"
print(list_data[4])

# Remove the last element.
list_data[4] <- NULL
# Print the 4th Element.
print(list_data[4])
# Update the 3rd Element.
list_data[3] <- "updated element"
print(list_data[3])


#Merging Lists
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")
# Merge the two lists.
merged.list <- c(list1,list2)
# Print the merged list.
print(merged.list)

#Converting List to Vector
# Create lists.
list1 <- list(1:5)
print(list1)
list2 <-list(10:14)
print(list2)
# Convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
print(v1)
print(v2)
# Now add the vectors
result <- v1+v2
print(result)
```

## Output Program

```
[[1]]
[1] "Jan" "Feb" "Mar"

[[2]]

     [,1] [,2] [,3]

[1,]    3    5    2

[2,]    9    1    8


[[3]]

[[3]][[1]]

[1] "green"


[[3]][[2]]

[1] 12.3

$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$`A Inner list`

$`A Inner list`[[1]]
[1] "green"


$`A Inner list`[[2]]
[1] 12.3

     [,1] [,2] [,3]

[1,]    3    5    2

[2,]    9    1    8

[[1]]
[1] "New element"


$<NA>
NULL

$`A Inner list`
[1] "updated element"
```

```
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] "Sun"


[[5]]
[1] "Mon"


[[6]]
[1] "Tue"

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

[[1]]
[1] 10 11 12 13 14

[1] 1 2 3 4 5

[1] 10 11 12 13 14

[1] 11 13 15 17 19
```

## 4.R – 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.

Elements of a matrix can be accessed by using the column and row index of the element

Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix. The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

## (R) Code

```
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)

print(M)

N <- matrix(c(3:14), nrow = 4, byrow = FALSE)

print(N)

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)

print(P[1,3])

print(P[,3])


matrix1 <- matrix(c(3, 9, 1, 4, 2, 6), nrow = 2)

print(matrix1)

matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)

print(matrix2)

result <- matrix1 + matrix2

cat("Result of addition","\n")

print(result)

result <- matrix1 - matrix2

cat("Result of subtraction","\n")

print(result)

#Matrix Multiplication & Division

# Create two 2x3 matrices.

matrix1 <- matrix(c(3, 9, 1, 4, 2, 6), nrow = 2)

print(matrix1)
```

```
matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)

print(matrix2)

# Multiply the matrices.

result <- matrix1 * matrix2

cat("Result of multiplication","\n")

print(result)

# Divide the matrices;

result <- matrix1 / matrix2

cat("Result of division","\n")

print(result)
```

## Output Program

```
     [,1] [,2] [,3]

[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14


     [,1] [,2] [,3]

[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14


     col1 col2 col3

row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

```
[1] 5
row1 row2 row3 row4

   5    8   11   14

      [,1] [,2] [,3]

[1,]    3     1    2
[2,]    9     4    6

      [,1] [,2] [,3]

[1,]    5    0    3
[2,]    2    9    4


Result of addition
      [,1] [,2] [,3]

[1,]    8   1    5
[2,]   11   13   10

Result of subtraction
      [,1] [,2] [,3]

[1,]   -2    1   -1
[2,]    7   -5    2

      [,1] [,2] [,3]

[1,]    3    1    2
[2,]    9    4    6

     [,1] [,2] [,3]

[1,]    5    0    3
[2,]    2    9    4

Result of multiplication
     [,1] [,2] [,3]

[1,]   15    0    6
[2,]   18   36   24

Result of division
     [,1] [,2] [,3]
[1,]  0.6   Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

## 5.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.**

### (R) Code

```
vector1 <- c(5,9,3)

vector2 <- c(10,11,12,13,14,15)

result <- array(c(vector1,vector2),dim = c(3,3,2))

print(result)
```

### Output Program

```
, , 1

     [,1] [,2] [,3]

[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15



, , 2

     [,1] [,2] [,3]

[1,]    5   10   13
[2,]    9   11   14
[3,]    3   12   15
```

## 6. Factors

**Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values.**
**Like "Male, "Female" and True, False etc. They are useful in data analysis for statistical modeling. Factors are created using the factor () function by taking a vector as input. Example**

**Factors in Data Frame:**
**On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.**

(R) Code

```
height <- c(132,151,162,139,166,147,122)

weight <- c(48,49,66,53,67,52,40)

gender <- c("male","male","female","female","male","female","male")

input_data <- data.frame(height,weight,gender)

print(input_data)

print(is.factor(input_data$gender))

print(input_data$gender)
```

Output Program

```
  height weight gender

1    132     48   male
2    151     49   male
3    162     66 female
4    139     53 female
5    166     67   male
6    147     52 female
7    122     40   male

[1] TRUE
[1] male   male   female female male   female male
Levels: female male
```

# Data Frames

R Data Frame

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.

(R) Code

```
# 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)
#Get the Structure of the Data Frame
```

```
str(emp.data)

#Summary of Data in Data Frame

print(summary(emp.data))


#Extract Data from Data Frame

# Extract Specific columns.

result <- data.frame(emp.data$emp_name,emp.data$salary)

print(result)


#Extract the first two rows and then all columns

# Extract first two rows.

result <- emp.data[1:2,]

print(result)



#Extract 3rd and 5th row with 2nd and 4th column

result <- emp.data[c(3,5),c(2,4)]

print(result)


#Expand Data Frame

# Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance")

v <- emp.data

print(v)


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")),

dept = c("IT","Operations","IT","HR","Finance"),
```

```
stringsAsFactors = FALSE

)

# Create the second data frame36

emp.newdata <- data.frame(

emp_id = c (6:8),

emp_name = c("Rasmi","Pranab","Tusar"),

salary = c(578.0,722.5,632.8),

start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),

dept = c("IT","Operations","Fianance"),

stringsAsFactors = FALSE

)

# Bind the two data frames.

emp.finaldata <- rbind(emp.data,emp.newdata)

print(emp.finaldata)
```

## Output Program

```
emp_id emp_name salary start_date

1      1     Rick 623.30 2012-01-01

2      2      Dan 515.20 2013-09-23

3      3 Michelle 611.00 2014-11-15

4      4     Ryan 729.00 2014-05-11

5      5     Gary 843.25 2015-03-27

'data.frame':     5 obs. of  4 variables:

 $ emp_id    : int  1 2 3 4 5

 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...

 $ salary    : num  623 515 611 729 843

 $ start_date: Date, format: "2012-01-01" "2013-09-23" ...

     emp_id    emp_name            salary        start_date

 Min.   :1   Length:5           Min.   :515.2   Min.   :2012-01-01
```

```
1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
Median :3   Mode  :character   Median :623.3   Median :2014-05-11
Mean   :3                      Mean   :664.4   Mean   :2014-01-14
3rd Qu.:4                      3rd Qu.:729.0   3rd Qu.:2014-11-15
Max.   :5                      Max.   :843.2   Max.   :2015-03-27
  emp.data.emp_name emp.data.salary
1              Rick          623.30
2               Dan          515.20
3          Michelle          611.00
4              Ryan          729.00
5              Gary          843.25


  emp_id emp_name salary start_date
1      1     Rick  623.3 2012-01-01
2      2      Dan  515.2 2013-09-23
  emp_name start_date
3 Michelle 2014-11-15
5     Gary 2015-03-27


  emp_id emp_name salary start_date       dept
1      1     Rick 623.30 2012-01-01         IT
2      2      Dan 515.20 2013-09-23 Operations
3      3 Michelle 611.00 2014-11-15         IT
4      4     Ryan 729.00 2014-05-11         HR
5      5     Gary 843.25 2015-03-27    Finance
  emp_id emp_name salary start_date       dept
1      1     Rick 623.30 2012-01-01         IT
2      2      Dan 515.20 2013-09-23 Operations
3      3 Michelle 611.00 2014-11-15         IT
4      4     Ryan 729.00 2014-05-11         HR
5      5     Gary 843.25 2015-03-27    Finance
6      6    Rasmi 578.00 2013-05-21         IT
```

```
7      7   Pranab 722.50 2013-07-30 Operations
8      8    Tusar 632.80 2014-06-17   Fianance
```

---

Notes on Data-frame example

The structure of dataframe can be seen by using str() function

The statistical summary and nature of the data can be obtained by applying summary() function.

A data frame can be expanded by adding columns and rows.

1-For adding Column Just add the column vector using a new column name.

2-For adding Row: To add more rows permanently to an existing data frame, we need to bring in the

new rows in the same structure as the existing data frame and use the rbind() function. In the example above we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

# 4-Statstical Analysis

1. **MMM(Stands for Mean, Median and Mode)**

   **Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of**
   **the R base package. These functions take R vector as an input along with the arguments and give the result.**
   **The functions we are discussing in this chapter are mean, median and mode.**

   **Notes**
   **1-Mean:**
   **It is calculated by taking the sum of the values and dividing with the number of values in a data series. The**
   **function mean() is used to calculate this in R.**

   **2-Median:**
   **The middle most value in a data series is called the median. The median() function is used in R to calculate**
   **this value.**

   **3-Mode:**
   **The mode is the value that has highest number of occurrences in a set of data. Unike mean and median,**
   **mode can have both numeric and character data.**
   **R does not have a standard in-built function to calculate mode. So we create a user function to calculate**
   **mode of a data set in R. This function takes the vector as input and gives the mode value as output.**

## (R) Code

```
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.

result.mean <- mean(x)

print(result.mean)


#Median

x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

median.result <- median(x)

print(median.result)

#Mode

getmode <- function(v) {

uniqv <- unique(v)

uniqv[which.max(tabulate(match(v, uniqv)))]

}

v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

result <- getmode(v)

print(result)

charv <- c("o","it","the","it","it")

result <- getmode(charv)

print(result)
```

## Output Program

```
[1] 8.22

[1] 5.6

[1] 2

[1] "it"
```

## 2.Linear Regression

Regression analysis is a very widely used statistical tool to establish a relationship model between two

variables. One of these variable is called predictor variable whose value is gathered through experiments.

The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of

both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a

graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is :

$y = ax + b$

Following is the description of the parameters used :

- y is the response variable.
- x is the predictor variable.
- a and b are constants which are called the coefficients.

Steps to Establish a Regression

A simple example of regression is predicting weight of a person when his height is known. To do this we

need to have the relationship between height and weight of a person.

The steps to create the relationship is :

- Carry out the experiment of gathering a sample of observed values of height and corresponding weight.
- Create a relationship model using the lm() functions in R.
- Find the coefficients from the model created and create the mathematical equation using these
- Get a summary of the relationship model to know the average error in prediction. Also called residuals.
- To predict the weight of new persons, use the predict() function in R.

**Input Data**

**Below is the sample data representing the observations :**

**Values of height**

**151, 174, 138, 186, 128, 136, 179, 163, 152, 131**

**Values of weight.**

**63, 81, 56, 91, 47, 57, 76, 72, 62, 4841**

**lm() Function**

**This function creates the relationship model between the predictor and the response variable.**

**Syntax:**

**The basic syntax for lm() function in linear regression is :**

**lm(formula,data)**

**Following is the description of the parameters used :**

- **formula is a symbol presenting the relation between x and y.**
- **data is the vector on which the formula will be applied.**

**predict() Function**

**Syntax The basic syntax for predict() in linear regression is :**

**predict(object, newdata)**

**Following is the description of the parameters used :**

- **object is the formula which is already created using the lm() function.**
- **newdata is the vector containing the new value for predictor variable.**

**Predict the weight of new persons**

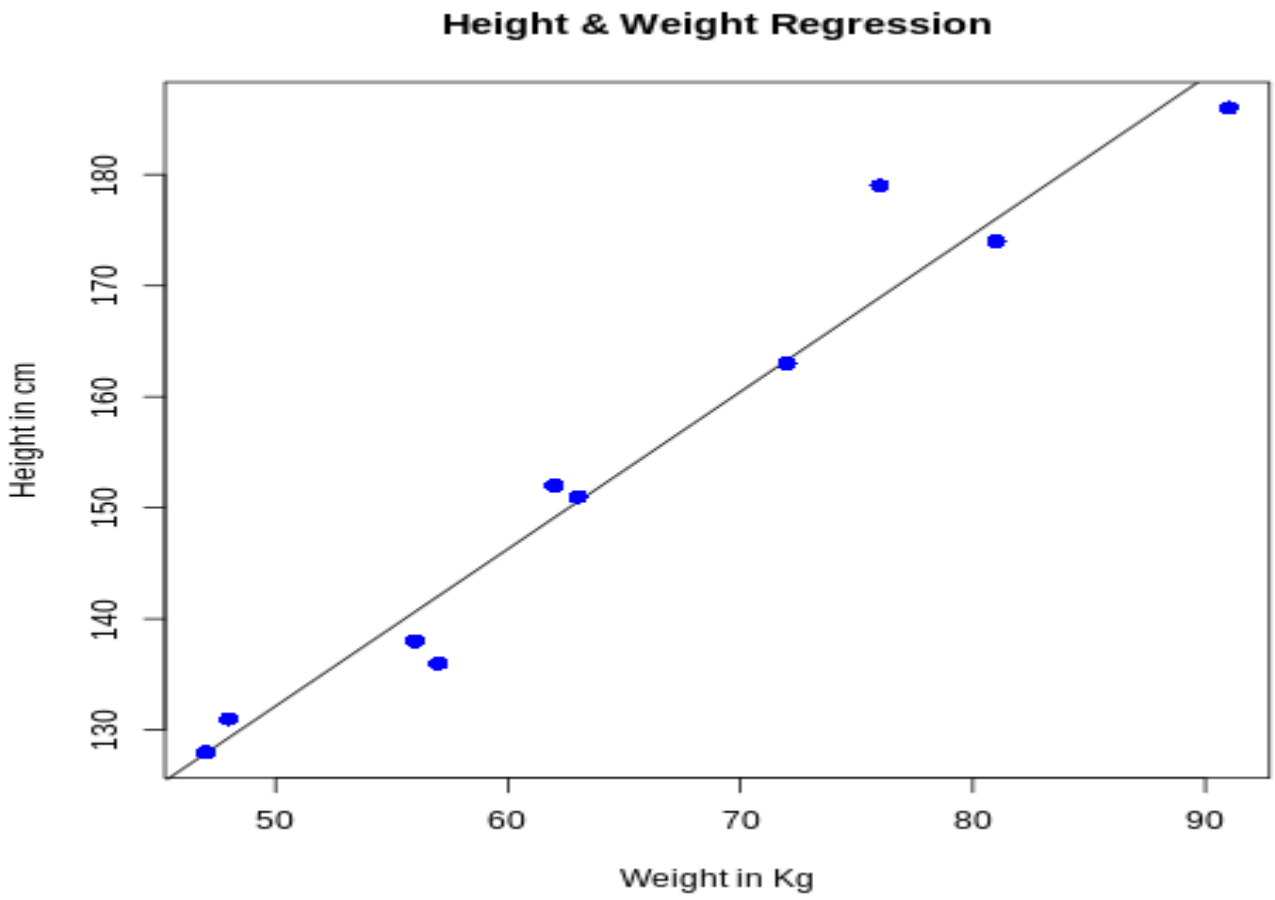
## (R) Code

```r
x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)
y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)
relation <- lm(y~x)
print(relation)
cat("--------------------------------","\n")

print(summary(relation))
cat("--------------------------------","\n")

a <- data.frame(x = 170)
result <- predict(relation,a)
print(result)
cat("--------------------------------","\n")

png(file = "linearregression.png")
plot(y,x,col = "blue",main = "Height & Weight Regression",abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab =
"Height in cm")
# Save the file.
dev.off()
```

## Output 1

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -38.4551       0.6746

--------------------------------

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-6.3002 -1.6629  0.0412  1.8944  3.9775

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -38.45509    8.04901  -4.778  0.00139 **
x             0.67461    0.05191  12.997 1.16e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.253 on 8 degrees of freedom
Multiple R-squared:  0.9548,    Adjusted R-squared:  0.9491
F-statistic: 168.9 on 1 and 8 DF,  p-value: 1.164e-06

--------------------------------
       1
76.22869
--------------------------------
null device
          1
```

## Output 2


Height & Weight Regression
Height in cm
130
140
150
160
170
180
50
60
70
80
90
Weight in Kg

**3. Non Linear Least square**

**When modeling real world data for regression analysis, we observe that it is rarely the case that the equation**

**of the model is a linear equation giving a linear graph. Most of the time, the equation of the model of real**

**world data involves mathematical functions of higher degree like an exponent of 3 or a sin function. In such**

**a scenario, the plot of the model gives a curve rather than a line. The goal of both linear and non-linear**

**regression is to adjust the values of the model's parameters to find the line or curve that comes closest to**

**your data. On finding these values we will be able to estimate the response variable with good accuracy.**

**In Least Square regression, we establish a regression model in which the sum of the squares of the vertical**

**distances of different points from the regression curve is minimized. We generally start with a defined model**

**and assume some values for the coefficients. We then apply the nls() function of R to get the more accurate**

**values along with the confidence intervals.**

**Syntax:**

**The basic syntax for creating a nonlinear least square test in R is -**

**nls(formula, data, start)**

**Following is the description of the parameters used - formula is a nonlinear model formula including**

**variables and parameters. data is a data frame used to evaluate the variables in the formula. start is a**

**named list or named numeric vector of starting estimates.**

**Example**

**We will consider a nonlinear model with assumption of initial values of its coefficients. Next we will see what**

**is the confidence intervals of these assumed values so that we can judge how well these values fir into the**

**model. So let's consider the below equation for this purpose -**

**a = b1*x^2+b2**

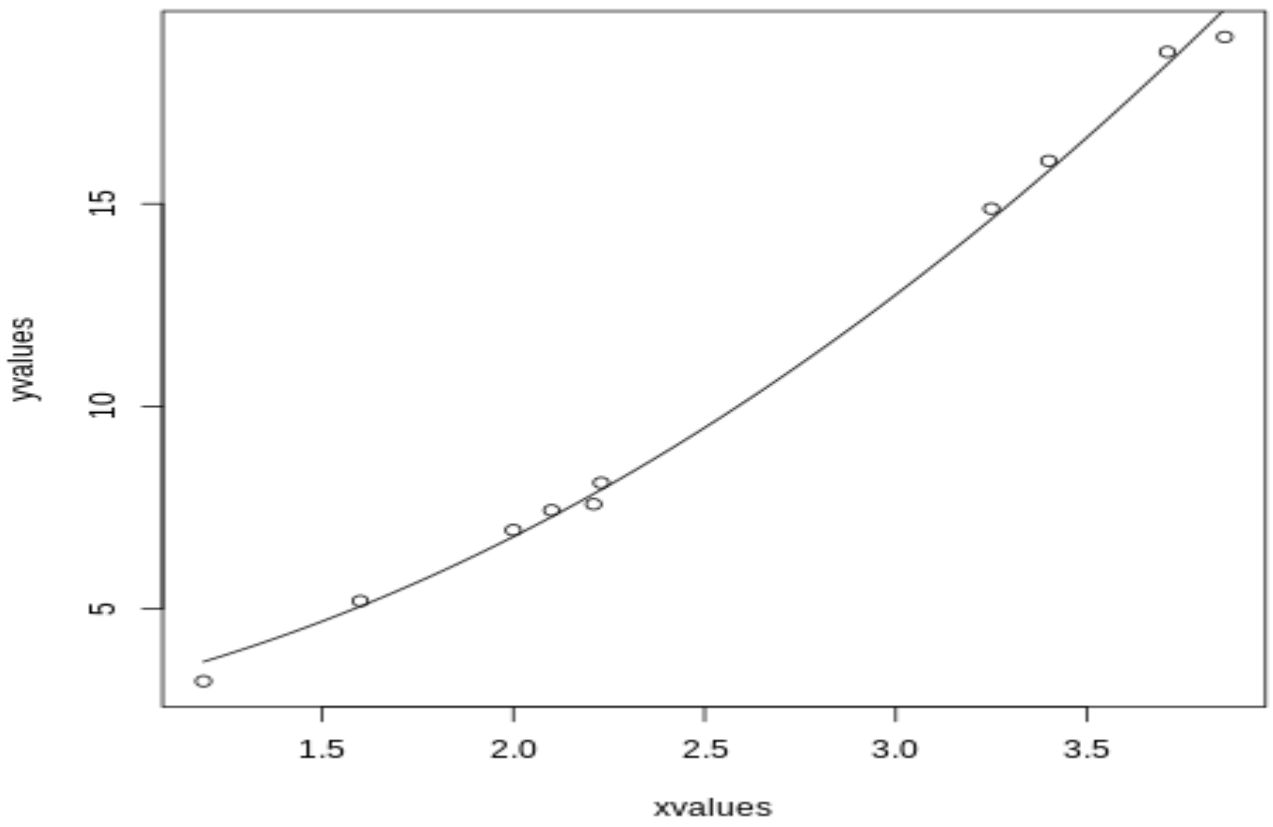
(R) Code

```
#Let's assume the initial coefficients to be 1 and 3 and fit these values into nls() function.
xvalues <- c(1.6,2.1,2,2.23,3.71,3.25,3.4,3.86,1.19,2.21)
yvalues <- c(5.19,7.43,6.94,8.11,18.75,14.88,16.06,19.12,3.21,7.58)
# Give the chart file a name.
png(file = "nls.png")
# Plot these values.
plot(xvalues,yvalues)
# Take the assumed values and fit into the model.
model <- nls(yvalues ~ b1*xvalues^2+b2,start = list(b1 = 1,b2 = 3))
# Plot the chart with new data by fitting it to a prediction from 100 data points.
new.data <- data.frame(xvalues = seq(min(xvalues),max(xvalues),len = 100))
lines(new.data$xvalues,predict(model,newdata = new.data))
# Save the file.
dev.off()
# Get the sum of the squared residuals.
print(sum(resid(model)^2))
# Get the confidence intervals on the chosen values of the coefficients.
print(confint(model))
```

Output 1

```
null device
          1
[1] 1.081935
Waiting for profiling to be done...
       2.5%     97.5%
b1 1.137708 1.253135
b2 1.497364 2.496484
```

## Output 2



**4. Chi-Square-Test**

**Chi-Square test is a statistical method to determine if two categorical variables have a significant correlation**

**between them. Both those variables should be from same population and they should be categorical like -**

**Yes/No, Male/Female, Red/Green etc. For example, we can build a data set with observations on people's**

**ice-cream buying pattern and try to correlate the gender of a person with the flavor of the ice-cream they**

**prefer. If a correlation is found we can plan for appropriate stock of flavors by knowing the number of gender**

**of people visiting.**

**Syntax:**

**The function used for performing chi-Square test is chisq.test(). The basic syntax for creating a chi-square**

**test in R is -**

**chisq.test(data)**

**Following is the description of the parameters used - data is the data in form of a table containing the**

**count value of the variables in the observation.**

**Example**

**We will take the Cars93 data in the "MASS" library which represents the sales of different models of car in**

**the year 1993.**

```
library("MASS")

print(str(Cars93))
```

The above result of code shows the dataset has many Factor variables which can be considered as categorical variables. For our model we will consider the variables "AirBags" and "Type". Here we aim to find out any

significant correlation between the types of car sold and the type of Air bags it has. If correlation is observed

we can estimate which types of cars can sell better with what types of air bags.

(R) Code

```r
library("MASS")

# Create a data frame from the main data set.
car.data <- data.frame(Cars93$AirBags, Cars93$Type)
# Create a table with the needed variables.
car.data = table(Cars93$AirBags, Cars93$Type)
print(car.data)
# Perform the Chi-Square test.
print(chisq.test(car.data))
```

## Output

```
                    Compact Large Midsize Small Sporty Van
  Driver & Passenger       2     4       7     0      3   0
  Driver only              9     7      11     5      8   3
  None                     5     0       4    16      3   6

        Pearson's Chi-squared test

data:  car.data
X-squared = 33.001, df = 10, p-value = 0.0002723
```

The result shows the p-value of less than 0.05 which indicates a significant correlation.