UNIT 13 BASICS OF RPROGRAMMING

Structure **Page Nos.**

13.0 Introduction
13.1 Objectives
13.2 Environment of R
13.3 Data types, Variables, Operators, Factors
13.4 Decision Making, Loops, Functions
13.5 Data Structures in R
  13.5.1 Strings and Vectors
  13.5.2 Lists
  13.5.3 Matrices, Arrays and Frames
13.6 Summary
13.7 Answers

13.0 INTRODUCTION

This unit covers the fundamental concepts of R programming. The unit familiarises with the environment of R and covers the details of the global environment. It further discusses the various types of data that is associated with every variable to reserve some memory space and store values in R. The unit discusses the various types of the data objects known as factors and the types of operators used in R programming. The unit also explains the important elements of decision making, the general form of a typical decision-making structures and the loops and functions. R's basic data structures including vector, strings, lists, frames, matrices and arrays would also be discussed.

13.1 OBJECTIVES

After going through this Unit, you will be able to:

- explain about the environment of R, the global environment and their elements;
- explain and distinguish between the data types and assign them to variables;
- explain about the different types of operators and the factors;
- explain the basics of decision making, the structure and the types of loops;
- explain about the function- their components and the types;
- explain the data structures including vector, strings, lists, frames, matrices, and arrays.

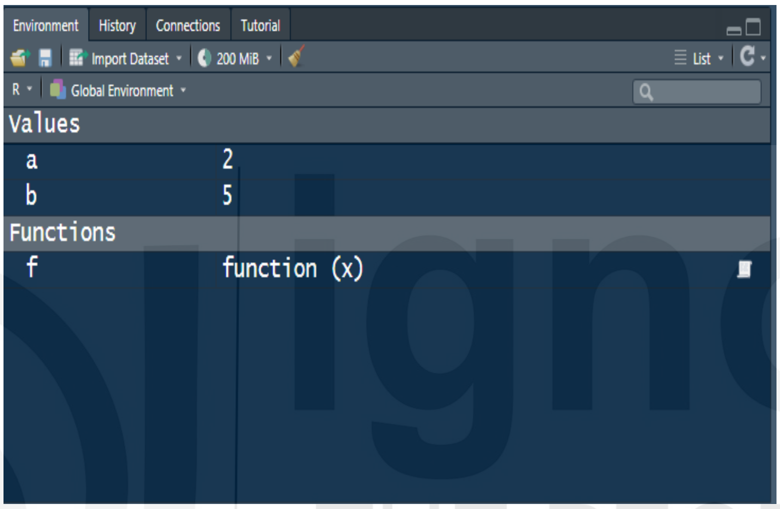
13.2 ENVIRONMENT OF R

R Programming language has been designed for statistical analysis of data. It also has a very good support for graphical representation of data. It has a vast set of commands. In this Block, we will cover some of the essential component of R programming, which would be useful for you for the purpose of data analysis. We will not be covering all aspects of this programming language; therefore, you may refer to the further readings for more details.

The discussion on R programming will be in the context of R-Studio, which is an open-source software. You may try various commands listed in this unit to facilitate your learning. The first important concept of R is its environment, which is discussed next.

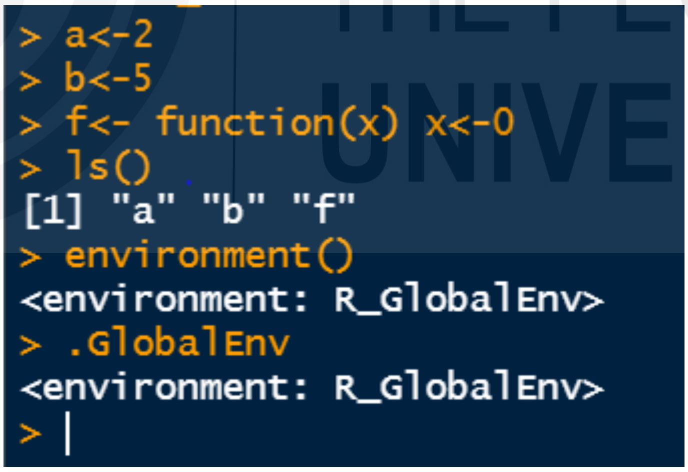
Environment can be thought of as a virtual space having collection of objects (variables, functions etc.) An environment is created when you first hit the R interpreter.

The top level environment present at R command prompt is the global environment known as R_GlobalEnv, it can also be referred as .GlobalEnv. You can use ls() command to know what variables/ functions are defined in the working environment. You can even check it in the Environment section of R Studio.



```
Values
 a                2
 b                5
Functions
 f                function (x)
```

Figure 5.1: Environment with a variable in RStudio



```
> a<-2
> b<-5
> f<- function(x) x<-0
> ls()
[1] "a" "b" "f"
> environment()
<environment: R_GlobalEnv>
> .GlobalEnv
<environment: R_GlobalEnv>
> 
```

Figure 5.2: Variables in Global Environment

In Figure 5.2, variables a, b and f are in R_GlobalEnv. Notice that x (as an argument to the function) is not in the global environment. When you define a function, a new environment is created. In Figure 5.1, a function f created a new environment inside the Global environment.

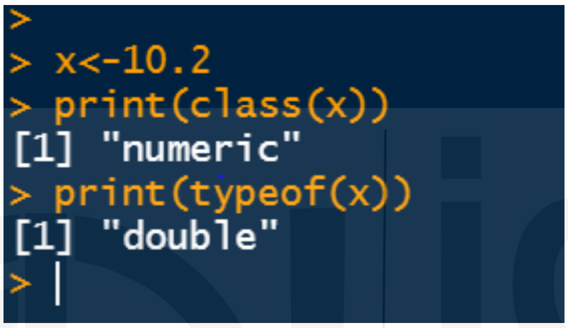
13.3 DATA TYPES, VARIABLES, OPERATORS, FACTORS

Every variable in R has an associated data type, which is known as the reserved memory. This reserved memory is needed for storing the values. Given below is a list of basic data types available in R programming:

DATA TYPE	Allowable Values
Integer	Values from the Set of Integers, Z
Numeric	Values from the Set of Real Numbers, R
Complex	Values from the Set of Complex numbers, C
Logical	Only allowable values are True ; False
Character	Possible values are -“x”, “@”, “1”, etc.

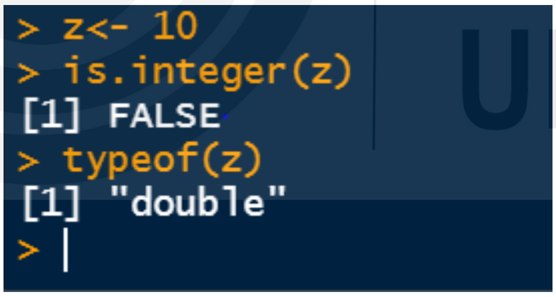
Table 1: Basic Data Types

Numeric Datatype:
Decimal values are known to be numeric in R and is default datatype for any number in R.



```
>
> x<-10.2
> print(class(x))
[1] "numeric"
> print(typeof(x))
[1] "double"
> |
```

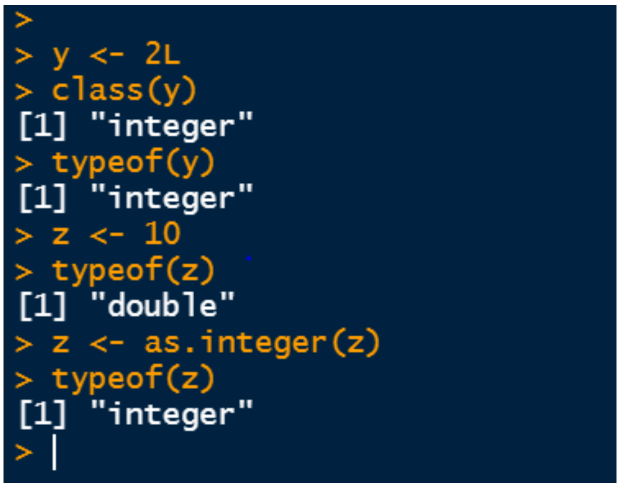
Whenever a number is stored in R, it gets converted into decimal type with at least 2 decimal points or the “double” value. So, if you enter a normal integer value also, for example 10, then R interpreter will convert it into double i.e. 10.00. You can even confirm this by checking the type of the variable, as given below:



```
> z<- 10
> is.integer(z)
[1] FALSE
> typeof(z)
[1] "double"
> |
```

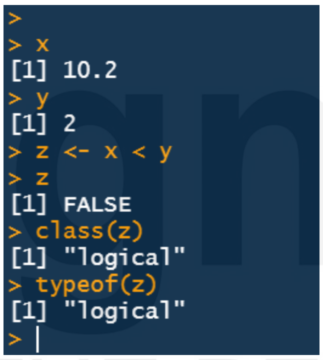
is.integer() function returning FALSE confirms that the variable z is converted into double or the decimal type.

Integer Datatype:
R supports integer data type, you can create an integer by suffixing “L” to denote that particular variable as integer as well as convert a value to an integer by passing the variable to as.integer() function.



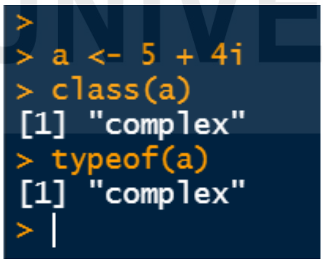
```
>
> y <- 2L
> class(y)
[1] "integer"
> typeof(y)
[1] "integer"
> z <- 10
> typeof(z)
[1] "double"
> z <- as.integer(z)
> typeof(z)
[1] "integer"
> |
```

Logical Datatype:
R has a logical datatype which returns value as either TRUE or FALSE. It is usually used while comparing two variables in a condition.



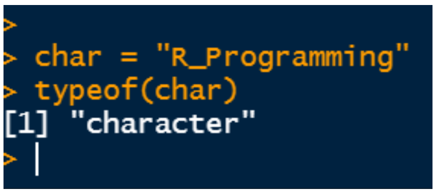
```
>
> x
[1] 10.2
> y
[1] 2
> z <- x < y
> z
[1] FALSE
> class(z)
[1] "logical"
> typeof(z)
[1] "logical"
> |
```

Complex Datatype:
Complex data types are also supported in R. These datatype includes the set of all complex numbers.



```
>
> a <- 5 + 4i
> class(a)
[1] "complex"
> typeof(a)
[1] "complex"
> |
```

Character Datatype:
R supports character datatype which includes alphabets and special characters. We need to include the value of the character type inside single or double inverted commas.



```
>
> char = "R_Programming"
> typeof(char)
[1] "character"
> |
```

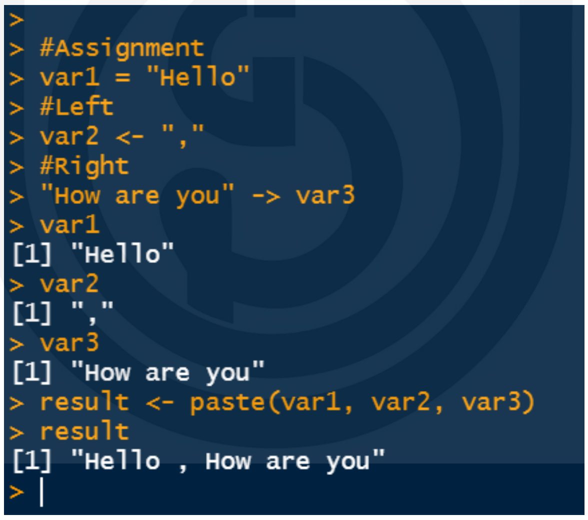
VARIABLES:

A variable as discussed in the previous section, allocates a memory space and stores the values, which can be manipulated. A valid variable name consists of letters, numbers and dot or underline characters

Variable Name	Valid	Reason
var_name1.	Valid	Contains letters, number, dot and underscore
1var_name	Invalid	Starting with a number
Var_name@	Invalid	Has special character (@). Only dot and underscore is allowed.
.var_name, var.name	Valid	Can start with a dot, which is followed by an alphabet.
_var_name	Invalid	Should not start with underscore.
.2var_name	Invalid	Dot is followed by a number and hence invalid.

Variables Assignment: Variables can be assigned in multiple ways –

- Assignment (=): var1 = "Hello"
- Left (←): var2 ← ", "
- Right (→): "How are you" → var3



```
>
> #Assignment
> var1 = "Hello"
> #Left
> var2 <- ","
> #Right
> "How are you" -> var3
> var1
[1] "Hello"
> var2
[1] ","
> var3
[1] "How are you"
> result <- paste(var1, var2, var3)
> result
[1] "Hello , How are you"
> |
```

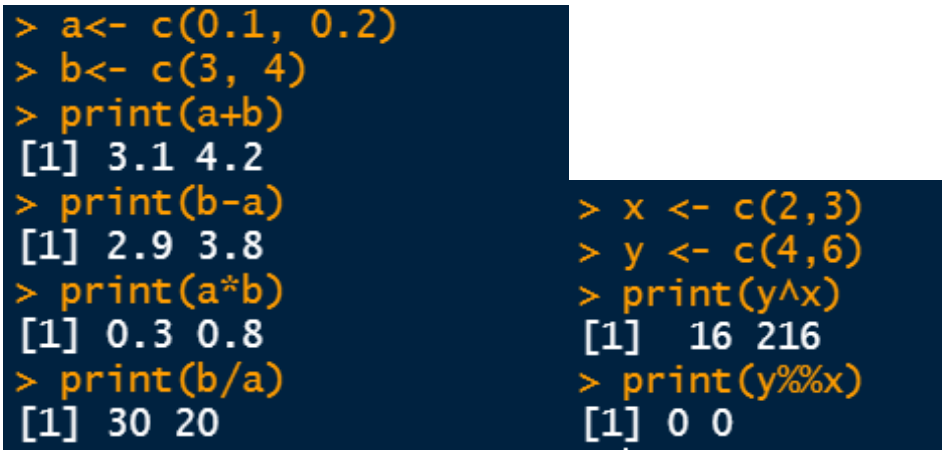
OPERATORS:

As the case with other programming languages, R also supports assignment, arithmetic, relational and logical operators. The logical operators of R include element by element operations. In addition, several other operators are supported by R, as explained in this section.

Arithmetic Operators:

- Addition (+): The value at the corresponding positions in the vectors are added. Please note the difference with C programming, as you are adding a complete vector using a single operator.
- Subtraction (-): The value at the corresponding positions are subtracted. Once again please note that single operator performs the task of subtracting elements of two vectors.
- Multiplication (*): The value at the corresponding positions are multiplied.

- Division (/): The value at the corresponding positions are divided.
- Power (^): The first vector is raised to the exponent (power) of the second.
- Modulo (%%): The remainder after dividing the two will be returned.

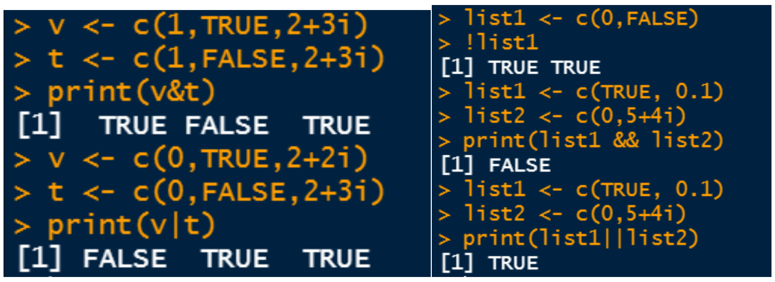


```
> a<- c(0.1, 0.2)
> b<- c(3, 4)
> print(a+b)
[1] 3.1 4.2
> print(b-a)
[1] 2.9 3.8
> print(a*b)
[1] 0.3 0.8
> print(b/a)
[1] 30 20
```

```
> x <- c(2,3)
> y <- c(4,6)
> print(y^x)
[1]  16 216
> print(y%%x)
[1] 0 0
```

Logical Operators:

- Element-wise Logical AND Operator (&): If both the corresponding operands are true, then this operator returns the Boolean value TRUE for that element. Please note the difference with C programming, in which it is a bitwise AND operator, whereas in R it is an element wise AND operator.
- Element-wise Logical OR Operator (|): If either of the corresponding operands are TRUE, then this operator returns the Boolean value TRUE for that element.
- Not Operator (!): This is a unary operator that is used to negate the operand.
- Logical AND Operator (&&): If the first element of both the operand are TRUE, then this operator returns the Boolean value TRUE.
  Logical OR Operator (||): If either of the first elements of the operands are true, then this operator returns Boolean value TRUE.



```
> v <- c(1,TRUE,2+3i)
> t <- c(1,FALSE,2+3i)
> print(v&t)
[1]  TRUE FALSE  TRUE
> v <- c(0,TRUE,2+2i)
> t <- c(0,FALSE,2+3i)
> print(v|t)
[1] FALSE  TRUE  TRUE
```

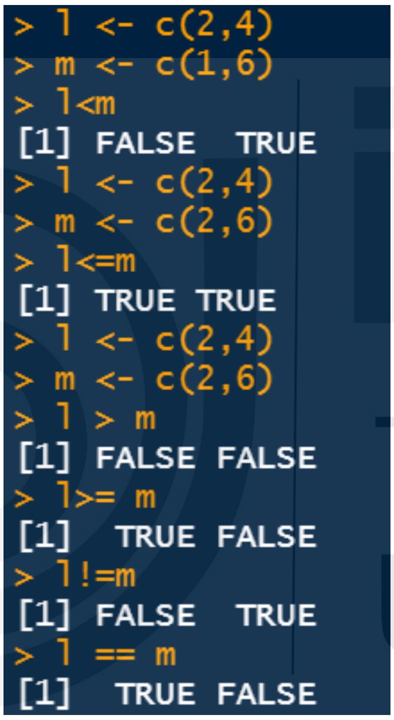
```
> list1 <- c(0,FALSE)
> !list1
[1] TRUE TRUE
> list1 <- c(TRUE, 0.1)
> list2 <- c(0,5+4i)
> print(list1 && list2)
[1] FALSE
> list1 <- c(TRUE, 0.1)
> list2 <- c(0,5+4i)
> print(list1||list2)
[1] TRUE
```

Relational Operators:

The relational operators can take scalar or vector operands. In case of vector operands comparison is done element by element and a vector of TRUE/FALSE values is returned.

- Less than (<): If an element of the first operand (scalar or vector) is less than that the corresponding element of the second operand, then this operator returns Boolean value TRUE.

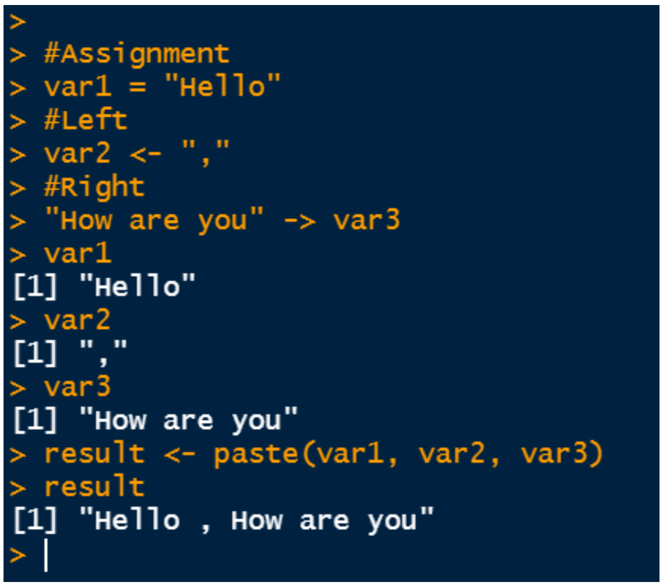
- Less than Equal to (<=): If every element in the first operand or vector is less than or equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Greater than (>): If every element in the first operand or vector is greater than that the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Greater than (>=): If every element in the first operand or vector is greater than or equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Not equal to (!=): If every element in the first operand or vector is not equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.
- Equal to (==): If every element in the first operand or vector is equal to the corresponding element of the second operand, then this operator returns Boolean value TRUE.



```
> l <- c(2,4)
> m <- c(1,6)
> l<m
[1] FALSE  TRUE
> l <- c(2,4)
> m <- c(2,6)
> l<=m
[1] TRUE TRUE
> l <- c(2,4)
> m <- c(2,6)
> l > m
[1] FALSE FALSE
> l>= m
[1]  TRUE FALSE
> l!=m
[1] FALSE  TRUE
> l == m
[1]  TRUE FALSE
```

Assignment Operators:

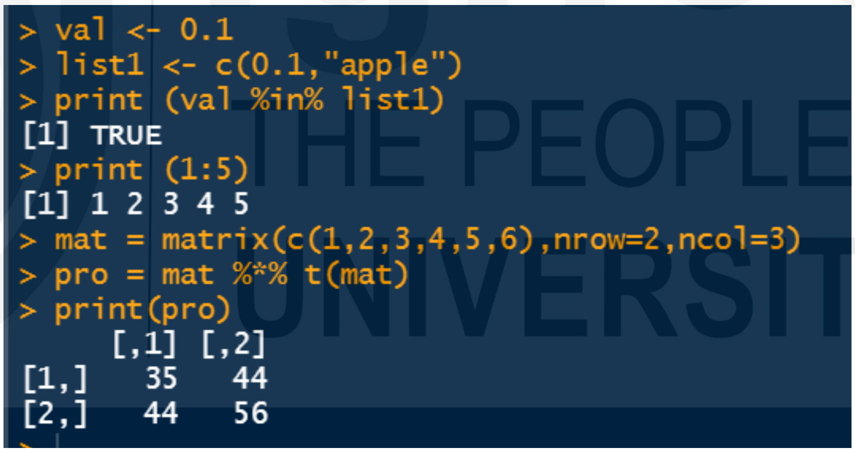
- Left Assignment (← or <<-or =): Used for assigning value to a vector.
- Right Assignment (-> or ->>): Used for assigning value to a vector.



```
>
> #Assignment
> var1 = "Hello"
> #Left
> var2 <- ","
> #Right
> "How are you" -> var3
> var1
[1] "Hello"
> var2
[1] ","
> var3
[1] "How are you"
> result <- paste(var1, var2, var3)
> result
[1] "Hello , How are you"
> |
```

Miscellaneous Operators:

- %in% operator: It determines whether a data element is contained in a list and returns a Boolean value TRUE if the element is found to exist.
- Colon(:) Operator: It prints a list of elements from before the colon to after the colon.
- %*% Operator: It helps in multiplying a matrix with its transpose.



```
> val <- 0.1
> list1 <- c(0.1,"apple")
> print (val %in% list1)
[1] TRUE
> print (1:5)
[1] 1 2 3 4 5
> mat = matrix(c(1,2,3,4,5,6),nrow=2,ncol=3)
> pro = mat %*% t(mat)
> print(pro)
     [,1] [,2]
[1,]   35   44
[2,]   44   56
```

FACTORS:
Factors are the data objects are used for categorizing and further storing the data as levels. They store both, strings and integer values. Factors are useful in the columns that have a limited number of unique values also known to be categorical variable. They are useful in data analysis for statistical modelling. For example, a categorical variable employment types – (Unemployed, Self-Employed, Salaried, Others) can be represented using factors. More details on factors can be obtained from the further readings.

Check Your Progress 1

1. What are various Operators in R?

…………………………………………………………………………………

…………………………………………………………………………………

2. What does %*% operator do?

   ……………………………………………………………………………….

   …………………………………………………………………………………

3. Is .5Var a valid variable name? Give reason in support of your answer.

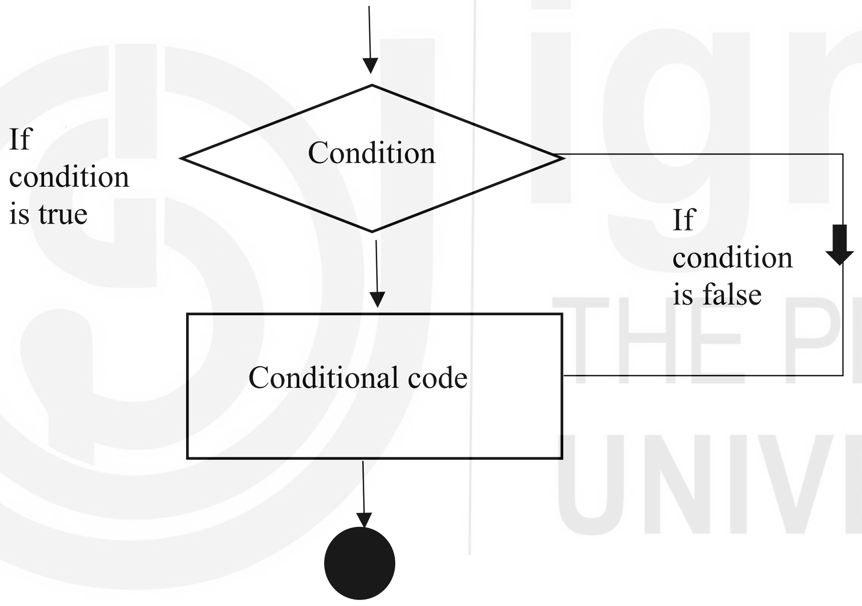
   …………………………………………………………………………………

   …………………………………………………………………………………

13.4 DECISION MAKING, LOOPS, FUNCTIONS

Decision making requires the programmer to specify one or more conditions which will be evaluated or tested by the program, along with the statements to be executed if the condition is determined to be true, and optional statements to be executed if the condition is determined to be false.

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



The format of if statement in R is as follows:

***if* (*conditional statement*, may include relational and logical operator) {**

R statements to be executed, if the *conditional statement* is true

}

***else* {**

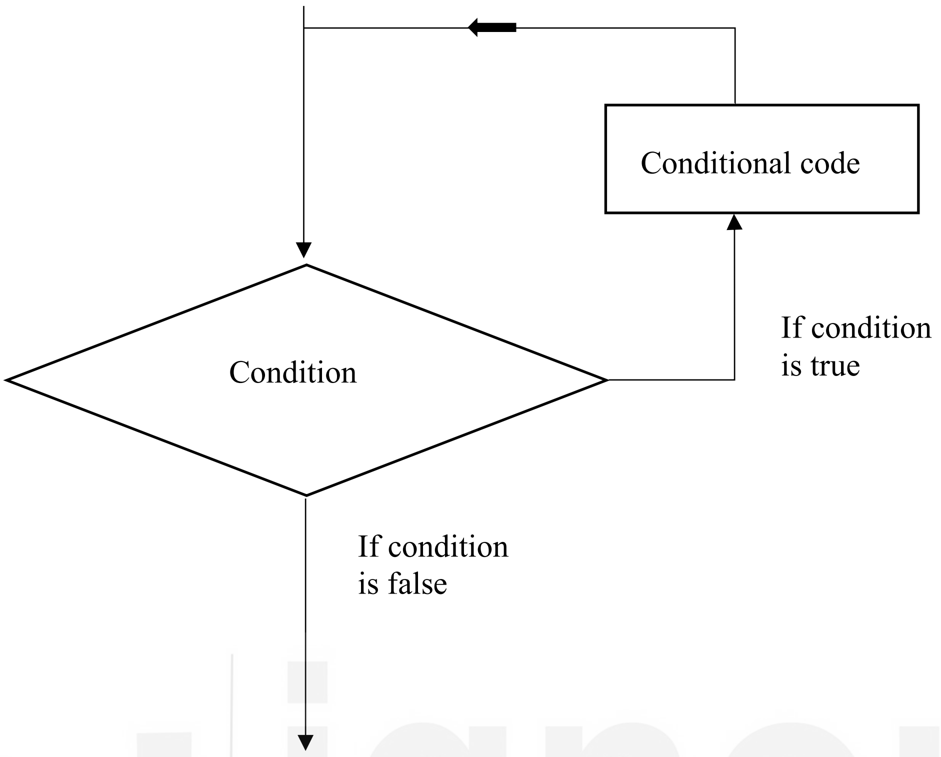
R statements to be executed, if the *conditional statement* is FALSE

}

You may use *else if* instead of *else*

LOOPS:

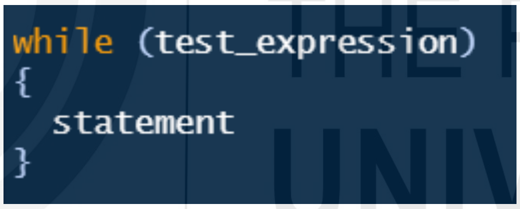
A loop is defined as a situation where we need to execute a block of code several number of times. In the case of loops, the statements are executed sequentially.



Loop Type and Description:

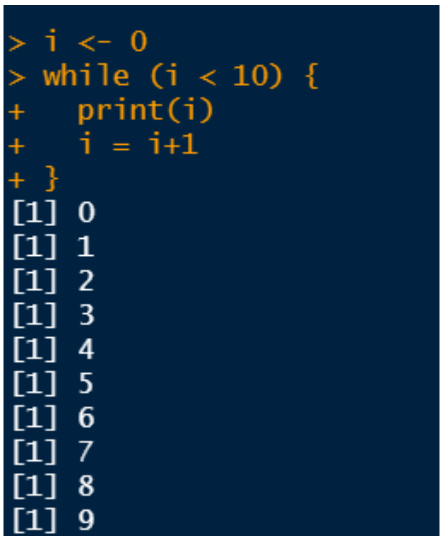
- Repeat loop: Executes sequence of statements multiple times.
- While loop: Repeat a given statement while the given condition is true, executes before executing the loop body.

  Syntax:



```
while (test_expression)
{
  statement
}
```

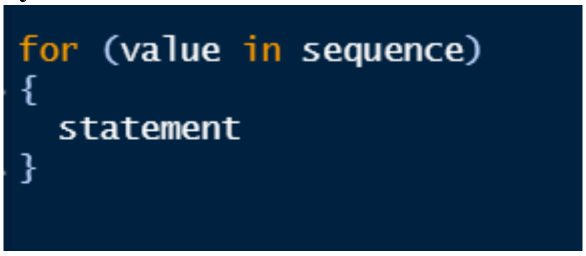
Example:



```
> i <- 0
> while (i < 10) {
+   print(i)
+   i = i+1
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

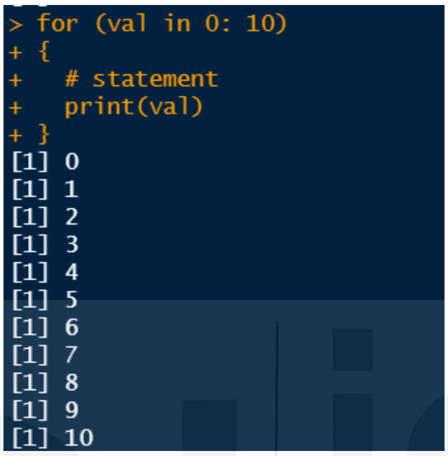
- For loop: Like while statement, executes the test condition at the end of the loop body.

Syntax:



```
for (value in sequence)
{
  statement
}
```

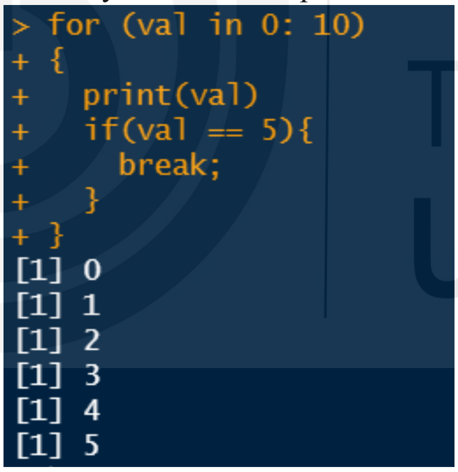
Example:



```
> for (val in 0: 10)
+ {
+   # statement
+   print(val)
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Loop Control Statements:

- Break Statements: Terminates the loop statement and execute the statements immediately below the loop.



```
> for (val in 0: 10)
+ {
+   print(val)
+   if(val == 5){
+     break;
+   }
+ }
[1] 0
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

FUNCTIONS:

A function refers to a set of instructions that is required to execute a command to achieve a task in R. There are several built-in functions available in R. Further, users may create a function basis their requirements.

Definition:

A function can be defined as:

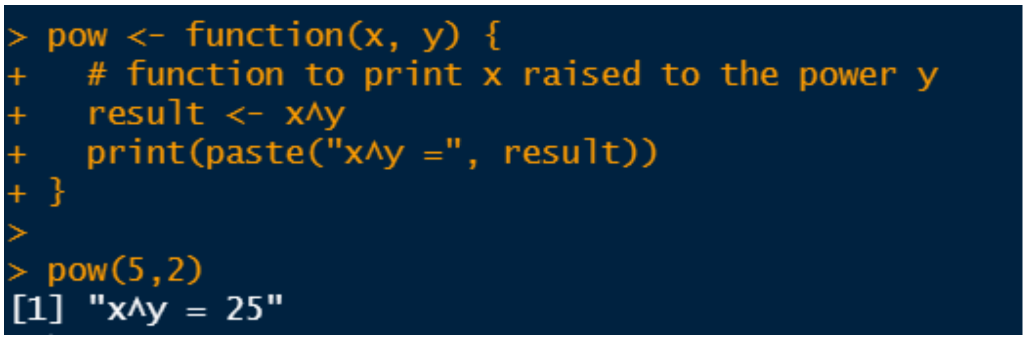
```
function_name<- function(arg_1, arg_2, ...) {
  Function body
}
```

Function Components

- Function Name: Actual name of the function.

- Arguments: Passed when the function is invoked. They are optional.
- Function Body: statements that define the logic of the function.
- Return value: last expression of the function to be executed.

Built-in function: Built in functions are the functions already written and is accessible just by calling the function name. Some examples are seq(), mean(), min(), max(), sqrt(), paste() and many more.



```
> pow <- function(x, y) {
+   # function to print x raised to the power y
+   result <- x^y
+   print(paste("x^y =", result))
+ }
>
> pow(5,2)
[1] "x^y = 25"
```

13.5 Data Structures in R

R's basic data structures include Vector, Strings, Lists, Frames, Matrices and Arrays.

13.5.1 Strings and Vectors

Vectors:

A vector is a one-dimensional array of data elements that have same data type. The most basic data structure are the Vectors, which supports logical, integer, double, complex, character datatypes.

Strings:
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.

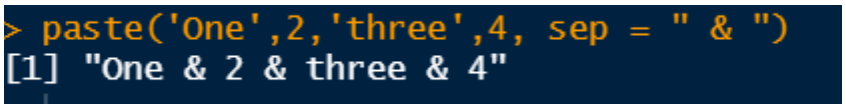
Rules Applied in String Construction

- The quotes at the beginning and end of a string should be either both double quotes or both single quote. They cannot 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 cannot be inserted into a string starting and ending with double quotes.
- Single quote cannot be inserted into a string starting and ending with single quote.

Length of String: The length of strings tells the number of characters in a string. The inbuilt function nchar() or function str_length() of the stringr package can be used to get the length of the string.

String Manipulations:

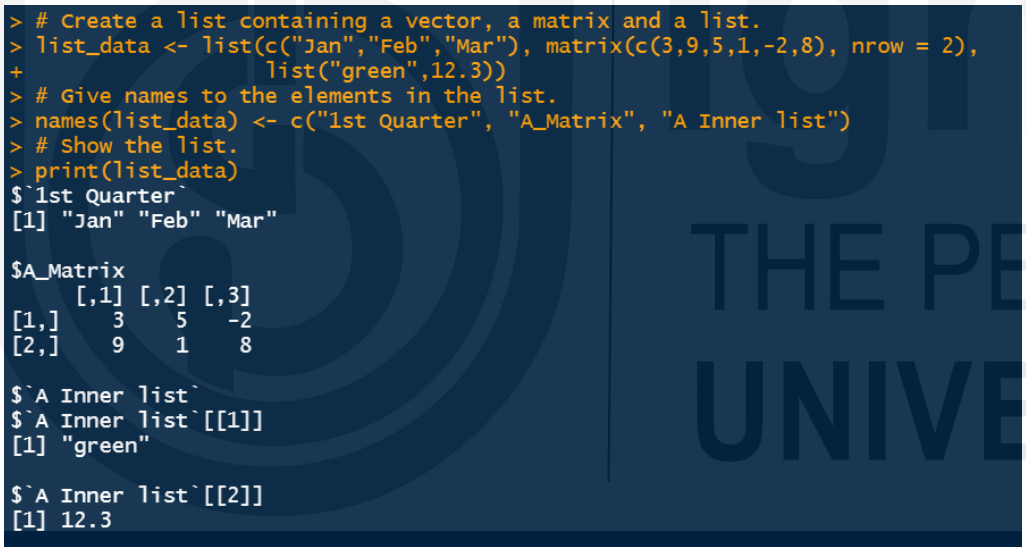
- Substring: Accessing the different portions of the strings. The 2 inbuilt functions present for this is substr() or substring() to extract the sub-strings.
- Case Conversion: The characters of the string can be converted to upper or the lower case by using toupper() or tolower().
- Concatenation: The strings in R can be combined by using the paste() function. It can concatenate any number of strings together. For example, paste(..., sep = " ", collapse = NULL)where x is vector having values, sep: is a separator symbol that is used to separate elements& collapse gives value to collapse.



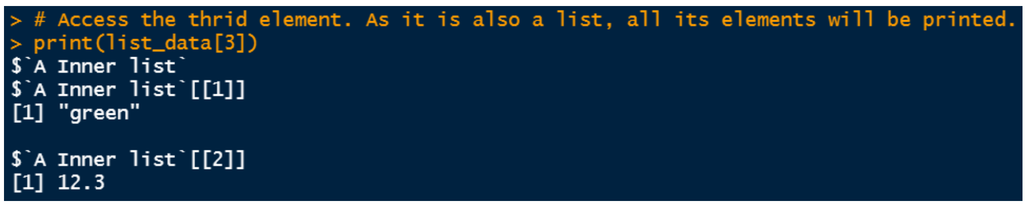
```
> paste('One',2,'three',4, sep = " & ")
[1] "One & 2 & three & 4"
```

13.5.2 Lists

Lists are the objects in R that contains different types of objects within itself like number, string, vectors or even another list, matrix or any function as its element It is created by calling list() function.



```
> # Create a list containing a vector, a matrix and a list.
> list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
+                   list("green",12.3))
> # Give names to the elements in the list.
> names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")
> # Show the list.
> print(list_data)
$`1st Quarter`
[1] "Jan" "Feb" "Mar"

$A_Matrix
     [,1] [,2] [,3]
[1,]    3    5   -2
[2,]    9    1    8

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

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

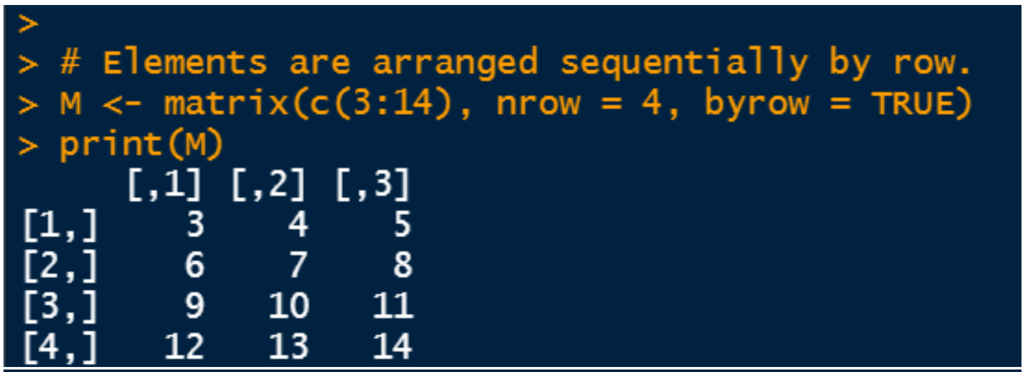


```
> # Access the thrid element. As it is also a list, all its elements will be printed.
> print(list_data[3])
$`A Inner list`
$`A Inner list`[[1]]
[1] "green"

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

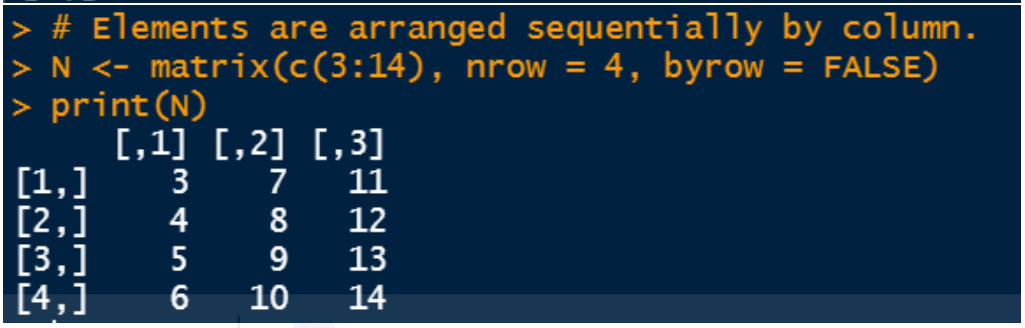
13.5.3 Matrices, Arrays and Frames

Matrices are R objects which are arranged in 2-D layout. They contain element of same type. The basic syntax of creating a matrix in R is:

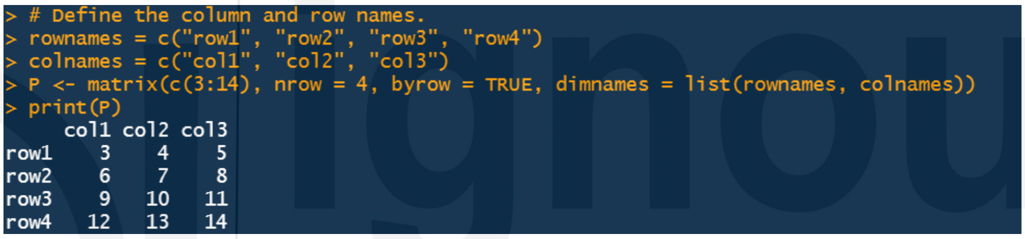
matrix(data, nrow, ncol, byrow, dimnames), where *data* is the name of input vector, *nrow* is no of rows, *ncol* is no of columns, *byrow* is to specify either row matrix or column matrix and *dimname*is the name assigned to rows and columns.



```
>
> # Elements are arranged sequentially by row.
> M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
> print(M)
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
```

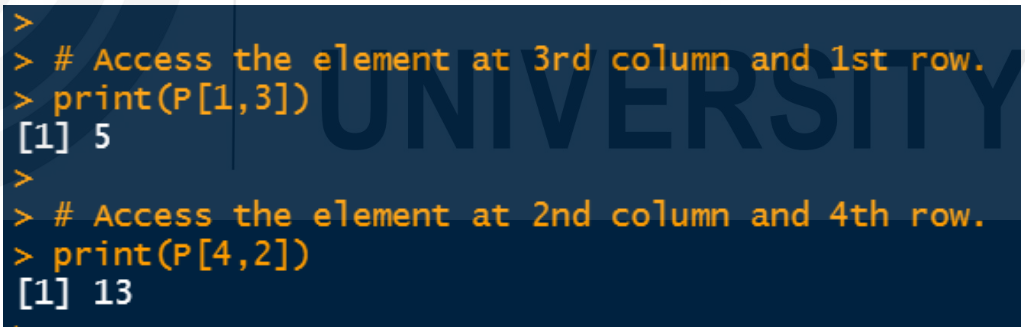


```
> # Elements are arranged sequentially by column.
> N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
> print(N)
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
```



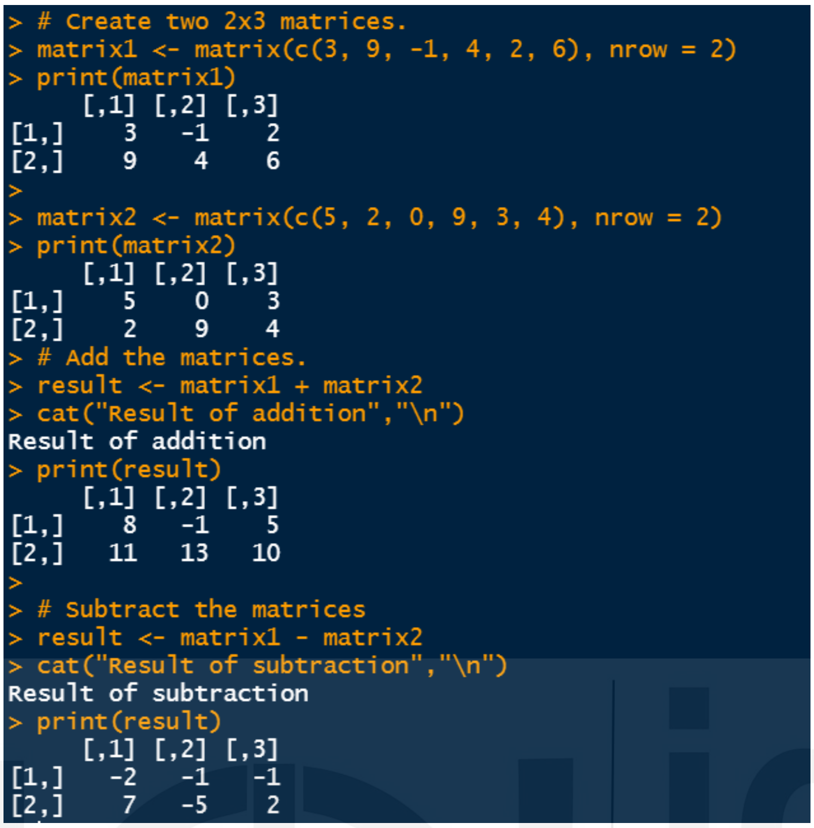
```
> # Define the column and row names.
> rownames = c("row1", "row2", "row3", "row4")
> colnames = c("col1", "col2", "col3")
> P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
> print(P)
     col1 col2 col3
row1    3    4    5
row2    6    7    8
row3    9   10   11
row4   12   13   14
```

Accessing the elements of the matrix: Elements of a matrix can be accessed by specifying the row and column number.

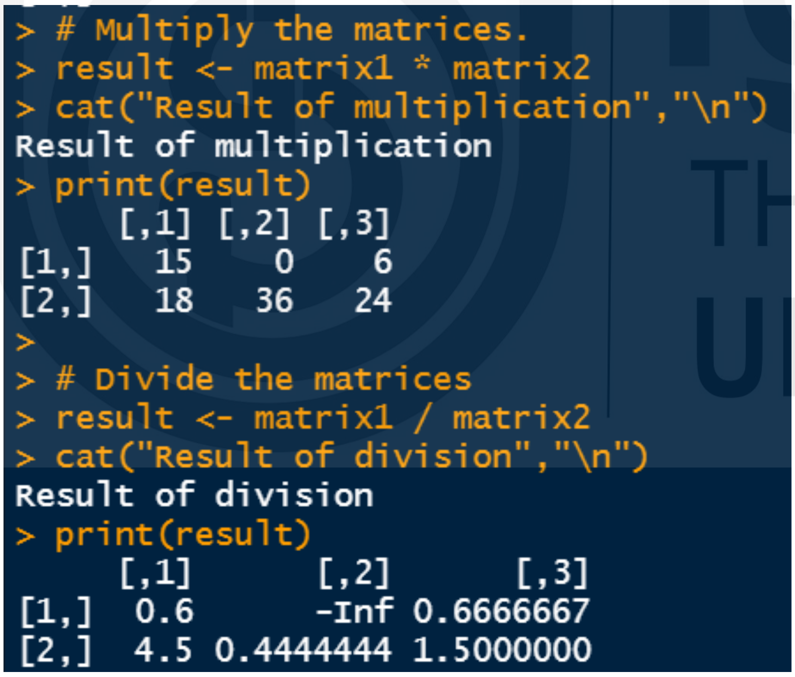


```
>
> # Access the element at 3rd column and 1st row.
> print(P[1,3])
[1] 5
>
> # Access the element at 2nd column and 4th row.
> print(P[4,2])
[1] 13
```

Matrix Manipulations:
Mathematical operations can be performed on the matrix like addition, subtraction, multiplication and division. You may please note that matrix division is not defined mathematically, but in R each element of a matrix is divided by the corresponding element of other matrix.



```
> # Create two 2x3 matrices.
> matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
> print(matrix1)
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
>
> matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
> print(matrix2)
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
> # Add the matrices.
> result <- matrix1 + matrix2
> cat("Result of addition","\n")
Result of addition
> print(result)
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
>
> # Subtract the matrices
> result <- matrix1 - matrix2
> cat("Result of subtraction","\n")
Result of subtraction
> print(result)
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
```

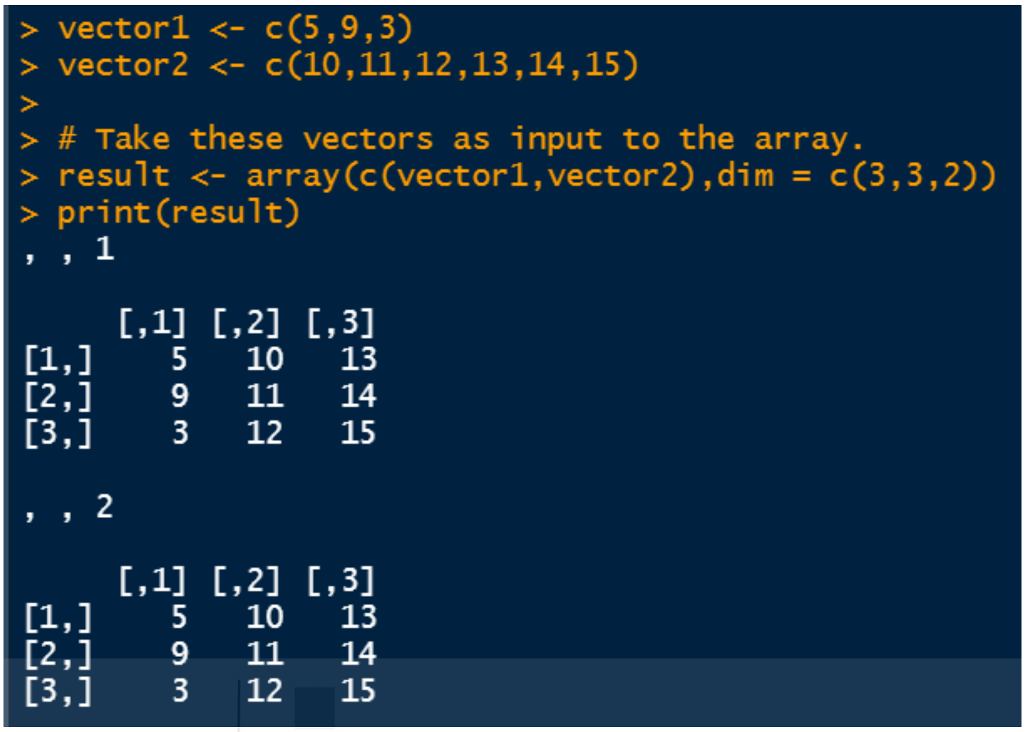


```
> # Multiply the matrices.
> result <- matrix1 * matrix2
> cat("Result of multiplication","\n")
Result of multiplication
> print(result)
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24
>
> # Divide the matrices
> result <- matrix1 / matrix2
> cat("Result of division","\n")
Result of division
> print(result)
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Arrays:

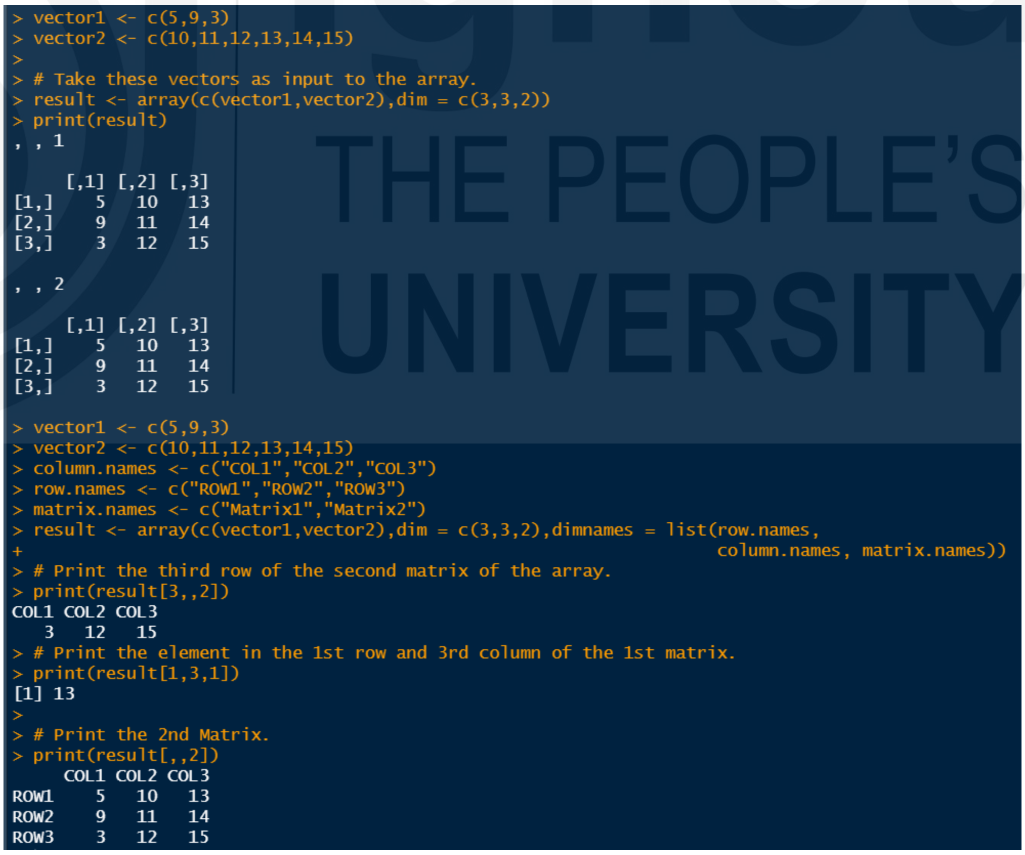
An array is a data object in R that can store multi-dimensional data that have the same data type. It is used using the array() function and can accept vectors as an input. An array is created using the values passed in the *dim* parameter.

For instance, an array is created with dimensions (2,3,5); then R would create 5 rectangular matrices comprising of 2 rows and 3 columns each. However, the data elements in each of the array will be of the same data type.



```
> 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)
, , 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
```

Accessing Array Elements:



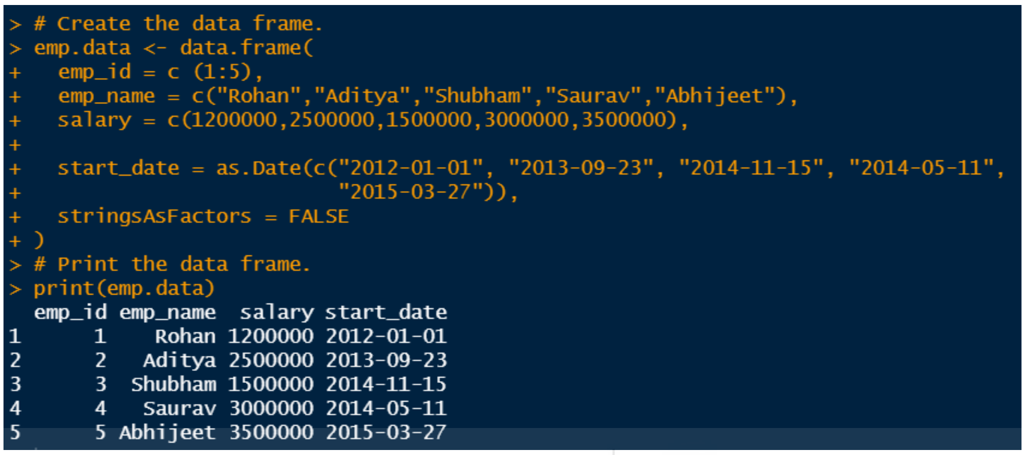
```
> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
>
> # Take these vectors as input to the array.
> result <- array(c(vector1,vector2),dim = c(3,3,2))
> print(result)
, , 1

     [,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

> vector1 <- c(5,9,3)
> vector2 <- c(10,11,12,13,14,15)
> column.names <- c("COL1","COL2","COL3")
> row.names <- c("ROW1","ROW2","ROW3")
> matrix.names <- c("Matrix1","Matrix2")
> result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames = list(row.names,
+                                                         column.names, matrix.names))
> # Print the third row of the second matrix of the array.
> print(result[3,,2])
COL1 COL2 COL3
   3   12   15
> # Print the element in the 1st row and 3rd column of the 1st matrix.
> print(result[1,3,1])
[1] 13
>
> # Print the 2nd Matrix.
> print(result[,,2])
     COL1 COL2 COL3
ROW1    5   10   13
ROW2    9   11   14
ROW3    3   12   15
```

Dataframe:

A data frame represents a table or a structure similar to an array with two dimensions It can be interpreted as matrices where each column of that matrix can be of different data types.

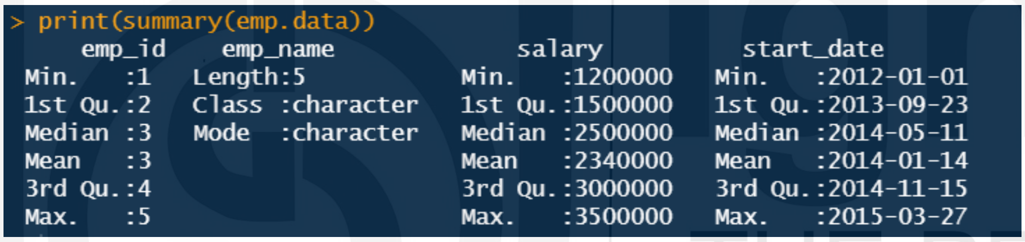
The characteristics of a data frame are given as follow

- The names of the columns should not be left blank
- The row names should be unique.
- The data frame can contain elements with numeric, factor or character data type
- Each column should contain same number of data items.



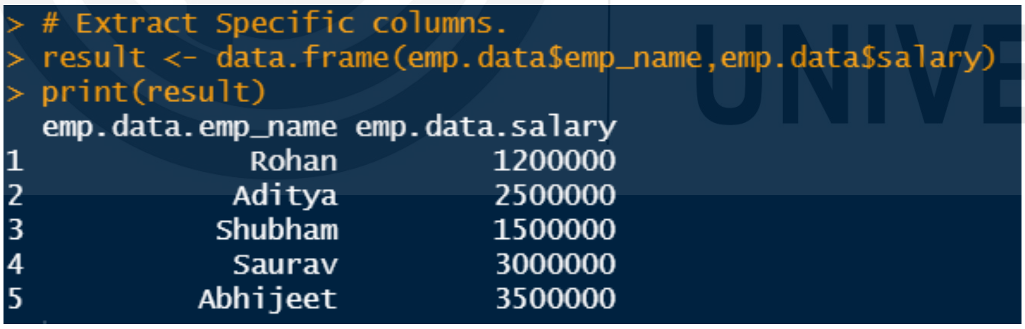
```
> # Create the data frame.
> emp.data <- data.frame(
+   emp_id = c (1:5),
+   emp_name = c("Rohan","Aditya","Shubham","Saurav","Abhijeet"),
+   salary = c(1200000,2500000,1500000,3000000,3500000),
+
+   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
+                        "2015-03-27")),
+   stringsAsFactors = FALSE
+ )
> # Print the data frame.
> print(emp.data)
  emp_id emp_name  salary start_date
1      1    Rohan 1200000 2012-01-01
2      2   Aditya 2500000 2013-09-23
3      3  Shubham 1500000 2014-11-15
4      4   Saurav 3000000 2014-05-11
5      5 Abhijeet 3500000 2015-03-27
```

Statistical summary of the dataframe can be fetched using summary() function.



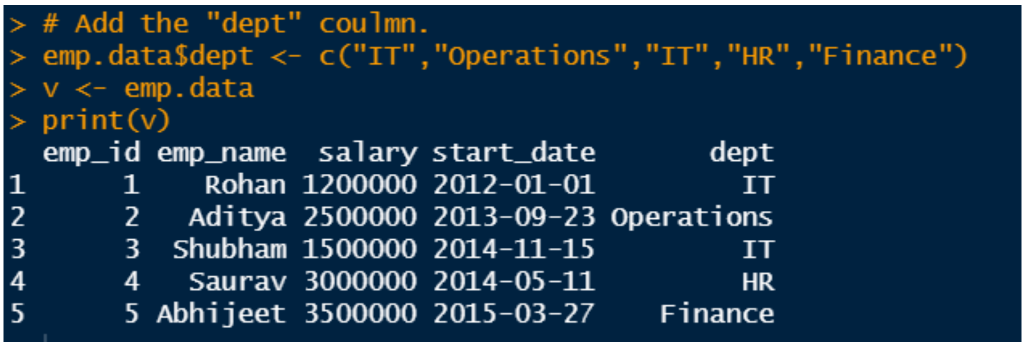
```
> print(summary(emp.data))
     emp_id     emp_name             salary          start_date        
 Min.   :1   Length:5           Min.   :1200000   Min.   :2012-01-01  
 1st Qu.:2   Class :character   1st Qu.:1500000   1st Qu.:2013-09-23  
 Median :3   Mode  :character   Median :2500000   Median :2014-05-11  
 Mean   :3                      Mean   :2340000   Mean   :2014-01-14  
 3rd Qu.:4                      3rd Qu.:3000000   3rd Qu.:2014-11-15  
 Max.   :5                      Max.   :3500000   Max.   :2015-03-27  
```

Extracting specific data from data frame by specifying the column name.



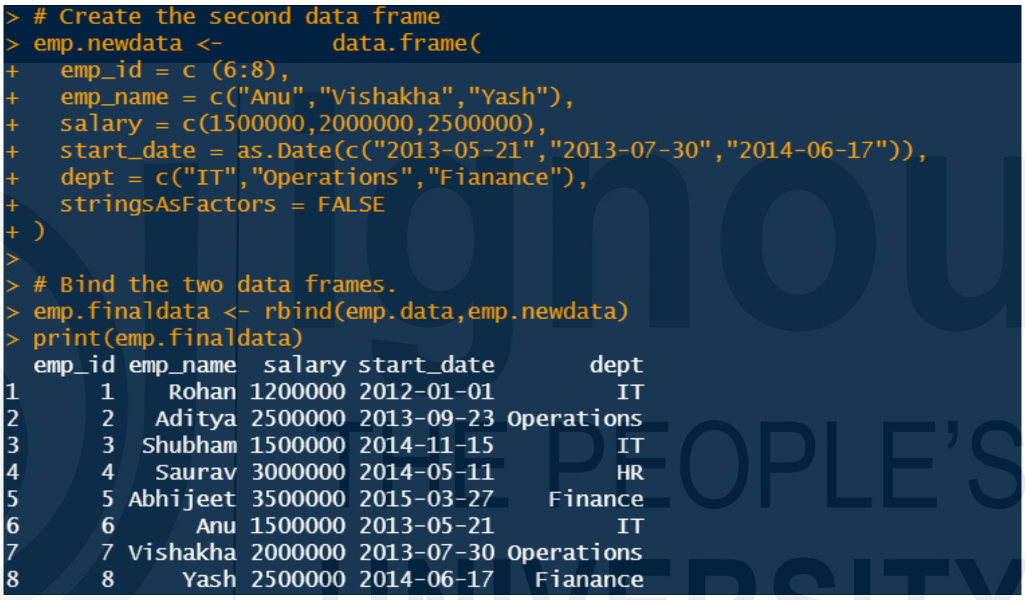
```
> # Extract Specific columns.
> result <- data.frame(emp.data$emp_name,emp.data$salary)
> print(result)
  emp.data.emp_name emp.data.salary
1             Rohan         1200000
2            Aditya         2500000
3           Shubham         1500000
4            Saurav         3000000
5          Abhijeet         3500000
```

Expanding the data frame by Adding additional column.



```
> # Add the "dept" coulmn.
> emp.data$dept <- c("IT","Operations","IT","HR","Finance")
> v <- emp.data
> print(v)
  emp_id emp_name  salary start_date       dept
1      1    Rohan 1200000 2012-01-01         IT
2      2   Aditya 2500000 2013-09-23 Operations
3      3  Shubham 1500000 2014-11-15         IT
4      4   Saurav 3000000 2014-05-11         HR
5      5 Abhijeet 3500000 2015-03-27    Finance
```

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.



```
> # Create the second data frame
> emp.newdata <-        data.frame(
+   emp_id = c (6:8),
+   emp_name = c("Anu","Vishakha","Yash"),
+   salary = c(1500000,2000000,2500000),
+   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)
  emp_id emp_name  salary start_date       dept
1      1    Rohan 1200000 2012-01-01         IT
2      2   Aditya 2500000 2013-09-23 Operations
3      3  Shubham 1500000 2014-11-15         IT
4      4   Saurav 3000000 2014-05-11         HR
5      5 Abhijeet 3500000 2015-03-27    Finance
6      6      Anu 1500000 2013-05-21         IT
7      7 Vishakha 2000000 2013-07-30 Operations
8      8     Yash 2500000 2014-06-17   Fianance
```

Check Your Progress 2

1. Why are Matrices data structure not used that often?

…………………………………………………………………………………

………………………………………………………………………………

2. What are the different data structures in R? Briefly explain about them.

…………………………………………………………………………………

…………………………………………………………………………………

3. What is the function used for adding datasets in R?

…………………………………………………………………………………

…………………………………………………………………………………

13.6 SUMMARY

The unit introduces you to the basics of R programming. It explains about the environment of R, a virtual space having collection of objects and how a new environment can be created within the global environment. The unit also explains about the various types of data associated with the variables that allocates a memory space and stores the values that can be manipulated. It also gives the details of the five types of operators in R programming. It also explains about factors that are the data objects used for organizing and storing the data as levels. The concept of decision making is also been discussed in detail that requires the programmer to specify one or more conditions to be evaluated or tested by the program. The concept of loops and their types has also been defined in this unit. It gives the details of function in R that is a set of instructions that is required to execute a a command to achieve a task in R. There are several built-in functions available in R. Further, users may create a function basis their requirements. The concept of matrices, arrays, dataframes etc have also been discussed in detail.

13.7 ANSWERS

Check Your Progress 1

1. The various operators in R are Arithmetic, Relational, Logical, assignment and Miscellaneous Operators. All of the above briefly explained in section 5.3

2. %*% Operator: It helps in multiplying a matrix with its transpose.

3. .5Var is an Invalid variable name as the dot is followed by a number

Check Your Progress 2

1. Matrices are not used much often as they contains only one data type and that too usually character or logical values.

2. Various Data Structures in R:

Data Structure	Description
Vector	A vector is a one-dimensional array of data elements that have same data type. These data elements in a vector are referred to as components.
List	Lists are the R objects which contain elements of different types like- numbers, strings, vectors or another list inside it.

Matrix	A matrix is a two-dimensional data structure. Matrices are used to bind vectors from the same length. All the elements of a matrix must have the same data type, i.e. (numeric, logical, character, complex).
Dataframe	A dataframe is more generic than a matrix, i.e. different columns can have different data types (numeric, logical etc). It combines features of matrices and lists like a rectangular list.

3. Rbind() is the function used to add datasets in R.

13.8 REFERENCES AND FURTHER READINGS

1. De Vries, A., & Meys, J. (2015). *R for Dummies*. John Wiley & Sons.
2. Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Victoria, BC, Canada: Leanpub.
3. Schmuller, J. (2017). *Statistical Analysis with R For Dummies*. John Wiley & Sons.
4. Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage publications.
5. Lander, J. P. (2014). *R for everyone: Advanced analytics and graphics*. Pearson Education.
6. Lantz, B. (2019). *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd.
7. Heumann, C., & Schomaker, M. (2016). *Introduction to statistics and data analysis*. Springer International Publishing Switzerland.
8. Davies, T. M. (2016). *The book of R: a first course in programming and statistics*. No Starch Press.
9. https://www.tutorialspoint.com/r/index.html

UNIT 14 DATA INTERFACING AND VISUALISATION IN R

Structure **Page Nos.**

14.1 Introduction
14.2 Objectives
14.3 Reading Data From Files
    14.3.1 CSV Files
    14.3.2 Excel Files
    14.3.3 Binary Files
    14.3.4 XML Files
    14.3.5 JSON Files
    14.3.6 Interfacing with Databases
    14.3.7 Web Data
14.4 Data Cleaning and Pre-processing
14.5 Visualizations in R
    14.5.1 Bar Charts
    14.5.2 Box Plots
    14.5.3 Histograms
    14.5.4 Line Graphs
    14.5.5 Scatterplots
14.6 Summary
14.7 Answers
14.8. References and Further Readings

14.1 INTRODUCTION

In the previous unit, you have learnt about basic concepts of R programming. This unit explains how to read and analyse data in R from various file types including- CSV, Excel, binary, XML, JSON, etc. It also discusses how to extract and work on data in R from databases and also web data. The unit also explains in detail about data cleaning and pre-processing in R. In the later sections, the unit explores the concept of visualisations in R. Various types of graphs and charts, including - bar charts, box plots, histograms, line graphs and scatterplots, are discussed.

14.2 OBJECTIVES

After going through this Unit, you will be able to:

- explain the various file types and their interface that can be processed for data analysis in R;
- read, write and analyse data in R from different type of files including- CSV, Excel, binary, XML and JSON;
- extract and use data from databases and web for analysis in R;
- explain the steps involved in data cleaning and pre-processing using R;
- Visualise the data using various types of graphs and charts using R and explain their usage.

14.3 READING DATA FROM FILES

In R, you can read data from files outside of the R environment. One may also write data to files that the operating system can store and further access. There is a wide range of file formats, including CSV, Excel, binary, and XML, etc., R can read and write from.

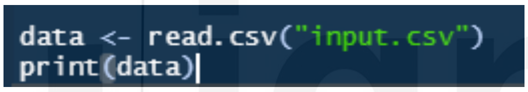
14.3.1 CSV Files

Input as CSV File:

CSV file is a text file in which column values are separated by commas. For example, you can create data with name, programme, phone of students. By copying and pasting this data into Windows Notepad, you can create the CSV file. Using notepad's *Save As* option, save the file as input.csv.

Reading a CSV File:

Function used to read a CSV file: **read.csv()**

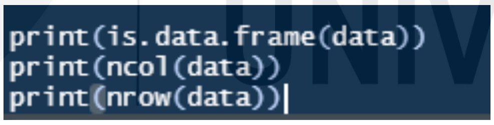


```
data <- read.csv("input.csv")
print(data)
```

Figure 14.1: Reading data from a CSV file

Analysing the CSV File:

The read.csv() function returns a data frame as its default output. You can use the following three print functions to:(1) verify if the read data input from CSV file is in frame format or not; (2) find the number of columns in the data; and (3) find the number of rows in the data.

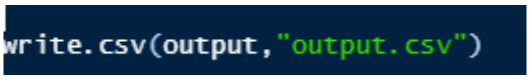


```
print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

Figure 14.2: Checking read data

Writing into a CSV File:

The **write.csv()** function of R can be used to generate a CSV file from a data frame. For example, the following function will generate an output.csv file. Please note that this output.csv will be created in the present directory in which you are working.



```
write.csv(output,"output.csv")
```

14.3.2 Excel Files

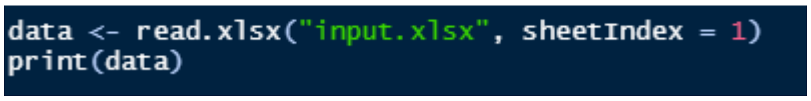
Microsoft Excel is the most extensively used spreadsheet tool and it uses the.xls or.xlsx file extension to store data. Using various Excel-specific packages, R can read directly from these files. XLConnect, xlsx, and gdata are a few examples of such packages. The xlsx package also allows R to write to an Excel file.

Install xlsx Package

- Command to install ‘xlsx’ package: **install.packages("xlsx")**
- To Load the library into R workspace: **library("xlsx")**

Reading the Excel File

The **read.xlsx()** function is used to read the input.xlsx file, as illustrated below. In the R environment, the result is saved as a data frame.

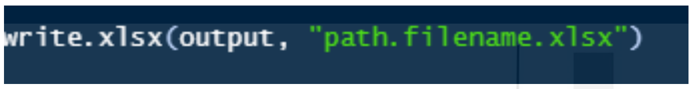


```
data <- read.xlsx("input.xlsx", sheetIndex = 1)
print(data)
```

Figure 14.3: Reading data from an Excel file

Writing the Excel File

For writing to a new Excel file, you use the write function, as shown below:



```
write.xlsx(output, "path.filename.xlsx")
```

14.3.3 Binary Files

A binary file is one that solely includes data in the form of bits and bytes. (0's and 1's). When you try to read a binary file, the sequence of bits is translated as bytes or characters, which include numerous other non-printable characters, that are not human readable. Any text editor that tries to read a binary file will display characters like Ø , ð, printable characters and many other characters including beeps.

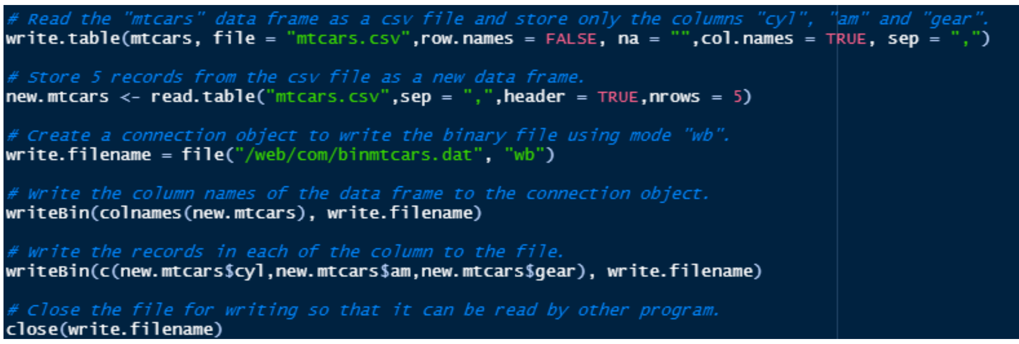
R has two functions **writeBin()** and **readBin()** to create and read binary files.

Syntax:
writeBin(**object, con**)
readBin(**con, what, n**)

where,

- The connection object **con** is used to read or write a binary file.
- The binary file to be written is the **object.**
- The mode that represents the bytes to be read, such as character, integer, etc is **what**.
- The number of bytes to read from the binary file is given by **n**.

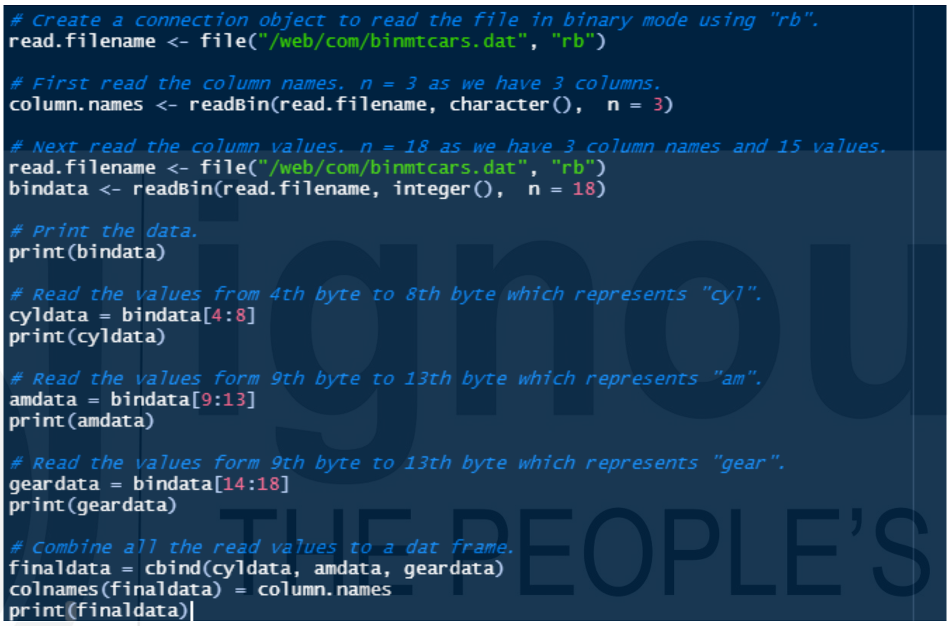
Writing the Binary File(You should read the comments for explanation on each command.)



```
# Read the "mtcars" data frame as a csv file and store only the columns "cyl", "am" and "gear".
write.table(mtcars, file = "mtcars.csv",row.names = FALSE, na = "",col.names = TRUE, sep = ",")

# Store 5 records from the csv file as a new data frame.
new.mtcars <- read.table("mtcars.csv",sep = ",",header = TRUE,nrows = 5)

# Create a connection object to write the binary file using mode "wb".
write.filename = file("/web/com/binmtcars.dat", "wb")

# Write the column names of the data frame to the connection object.
writeBin(colnames(new.mtcars), write.filename)

# Write the records in each of the column to the file.
writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)

# Close the file for writing so that it can be read by other program.
close(write.filename)
```

Figure14.4: An example of Writing data to a Binary file

Reading the Binary File(You should read the comments for explanation on each command.)



```
# Create a connection object to read the file in binary mode using "rb".
read.filename <- file("/web/com/binmtcars.dat", "rb")

# First read the column names. n = 3 as we have 3 columns.
column.names <- readBin(read.filename, character(),  n = 3)

# Next read the column values. n = 18 as we have 3 column names and 15 values.
read.filename <- file("/web/com/binmtcars.dat", "rb")
bindata <- readBin(read.filename, integer(),  n = 18)

# Print the data.
print(bindata)

# Read the values from 4th byte to 8th byte which represents "cyl".
cyldata = bindata[4:8]
print(cyldata)

# Read the values form 9th byte to 13th byte which represents "am".
amdata = bindata[9:13]
print(amdata)

# Read the values form 9th byte to 13th byte which represents "gear".
geardata = bindata[14:18]
print(geardata)

# Combine all the read values to a dat frame.
finaldata = cbind(cyldata, amdata, geardata)
colnames(finaldata) = column.names
print(finaldata)
```

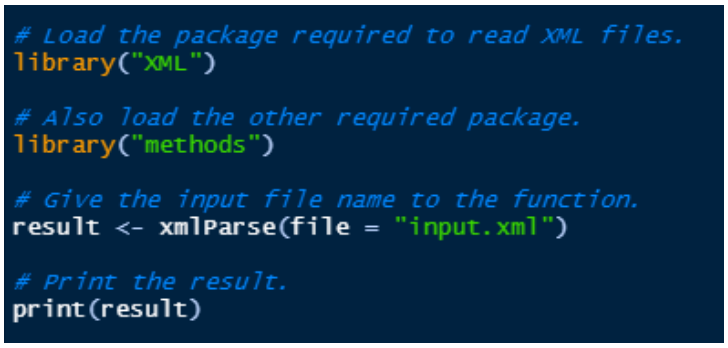
Figure14.5: An example of Reading data to a Binary file

14.3.4 XML Files

XML is an acronym for "extensible markup language". It is a file format that allows users to share the file format as well as the data over the internet, intranet and other places, as standard ASCII text. XML uses markup tags that describe the meaning of the data stored in the file. This is similar to the markup tags used in HTML wherein the markup tag describes the structure of the page instead.
The "XML" package in R can be used to read an xml file. The following command can be used to install this package:

install.packages("XML")

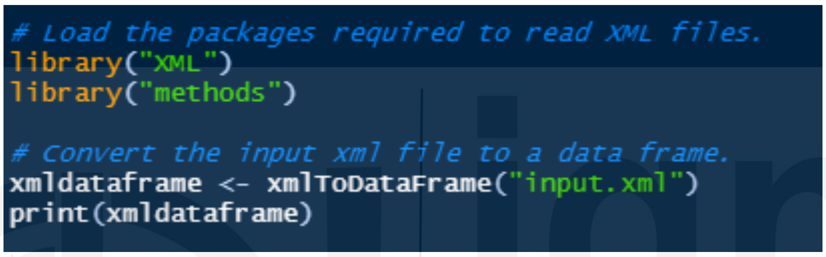
Reading XML File
R reads the xml file using the function **xmlParse()**. In R, it is saved as a list.



```
# Load the package required to read XML files.
library("XML")

# Also load the other required package.
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Print the result.
print(result)
```

Figure14.6: An example of reading data from a Binary file

XML to Data Frame
In order to manage the data appropriately in huge files, the data in the xml file can be read as a data frame. The data frame should then be processed for data analysis.



```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Convert the input xml file to a data frame.
xmldataframe <- xmlToDataFrame("input.xml")
print(xmldataframe)
```

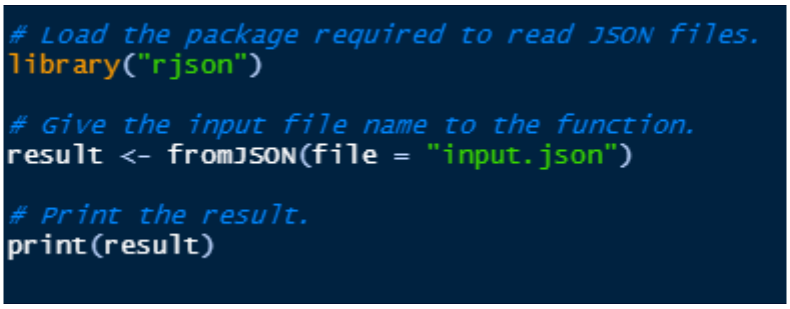
Figure14.7: converting the read data to a data frame

14.3.5 JSON Files

The data in a JSON file is stored as text in a human-readable format. JavaScript Object Notation is abbreviated as JSON. The rjson package in R can read JSON files.

Install rjson Package
To install the rjson package, type the following command in the R console: **install.packages("rjson")**

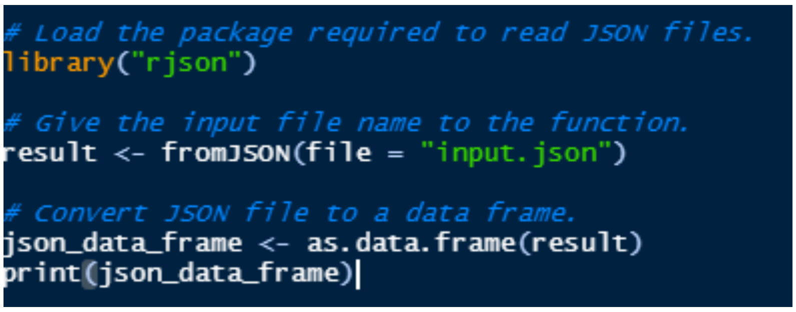
Read the JSON File
R reads the JSON file using the function **fromJSON()**. In R, it is saved as a list.



```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file = "input.json")

# Print the result.
print(result)
```

Figure14.8: An example of reading data from JSON file

Convert JSON to a Data Frame
Using the **as.data.frame()** function, you can turn the retrieved data above into a R data frame for further study.



```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file = "input.json")

# Convert JSON file to a data frame.
json_data_frame <- as.data.frame(result)
print(json_data_frame)
```

Figure14.9: Converting read data to data frame

14.3.6 Databases

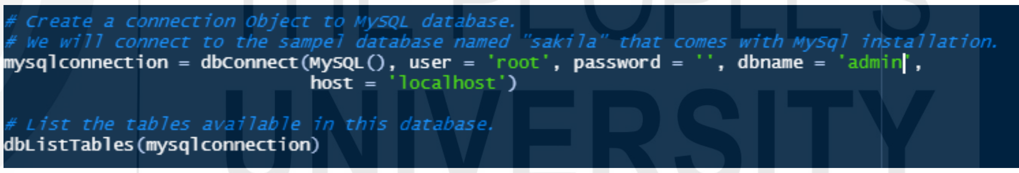
Data is stored in a normalised way in relational database systems. As a result, you will require quite advanced and complex SQL queries to perform statistical computing. However, R can readily connect to various relational databases, such as MySQL, Oracle, and SQL Server, and retrieve records as a data frame. Once the data is in the R environment, it becomes a standard R data set that can be modified and analysed with all of R's sophisticated packages and functions.

RMySQL Package

R contains a built-in package called "RMySQL" that allows you to connect to a MySql database natively. The following command will install this package in the R environment.

install.packages("RMySQL")

Connecting R to MySQL

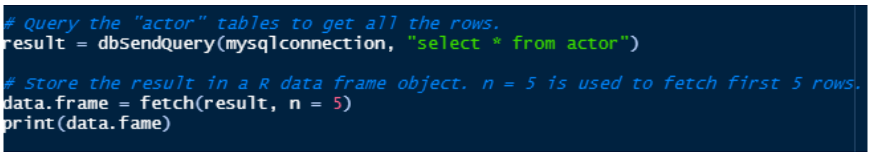


```
# Create a connection Object to MySQL database.
# We will connect to the sampel database named "sakila" that comes with MySql installation.
mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'admin',
                  host = 'localhost')

# List the tables available in this database.
dbListTables(mysqlconnection)
```

Figure14.10: Connecting to MySQL database

Querying the Tables

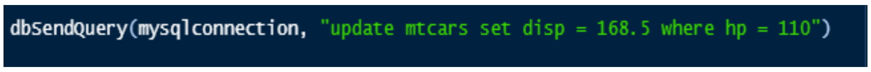
Using the MySQL function **dbSendQuery()**, you can query the database tables . The query is run in MySQL, and the results are returned with the **R fetch()** function. Finally, it is saved in R as a data frame.



```
# Query the "actor" tables to get all the rows.
result = dbSendQuery(mysqlconnection, "select * from actor")

# Store the result in a R data frame object. n = 5 is used to fetch first 5 rows.
data.frame = fetch(result, n = 5)
print(data.fame)
```

Figure 14.11: Querying the MySQL table

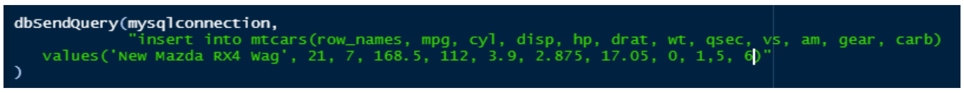
Updating Rows in the Tables



```
dbSendQuery(mysqlconnection, "update mtcars set disp = 168.5 where hp = 110")
```

Figure 14.12: Updating rows in MySQL table

Inserting Data into the Tables

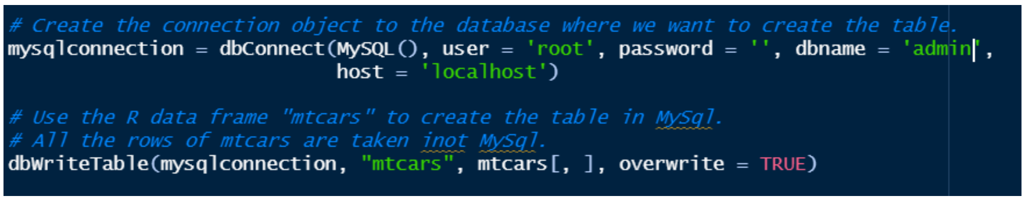


```
dbSendQuery(mysqlconnection,
            "insert into mtcars(row_names, mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb)
   values('New Mazda RX4 Wag', 21, 7, 168.5, 112, 3.9, 2.875, 17.05, 0, 1,5, 6)"
)
```

Figure 14.13: Inserting data in a MySQL table

Creating Tables in MySQL

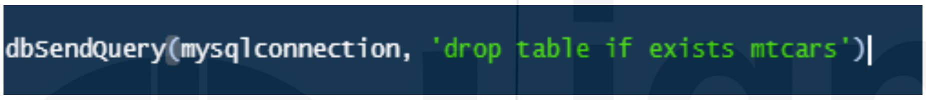
The function **dbWriteTable()** in MySQL can be used to create tables. It takes a data frame as input and overwrites the table if it already exists.



```
# Create the connection object to the database where we want to create the table.
mysqlconnection = dbConnect(MySQL(), user = 'root', password = '', dbname = 'admin',
                            host = 'localhost')

# Use the R data frame "mtcars" to create the table in MySql.
# All the rows of mtcars are taken inot MySql.
dbWriteTable(mysqlconnection, "mtcars", mtcars[, ], overwrite = TRUE)
```

Figure 14.14: Creating a table in MySQL

Dropping Tables in MySQL



```
dbSendQuery(mysqlconnection, 'drop table if exists mtcars')
```

Figure 14.15: Dropping a table in MySQL

14.3.7 Web Data

Many websites make data available for users to consume. The World Health Organization (WHO), for example, provides reports on health and medical information in CSV, txt, and XML formats. You can programmatically extract certain data from such websites using R applications. "RCurl," "XML," and "stringr" are some R packages that are used to scrape data from the web. They are used to connect to URLs, detect required file links, and download the files to the local environment.

Install R Packages

For processing the URLs and links to the files, the following packages are necessary.

install.packages("RCurl")
install.packages("XML")
install.packages("stringr")
install.packages("plyr")

14.4 DATA CLEANING AND PRE-PROCESSING

Data cleaning is the process of identifying, correcting and removing incorrect raw data. The clean data is then fed to the models to build the logical conclusions.
If the data is poorly prepped, unreliable results can destroy the assumptions and insights.
Packages like tidy verse can make complex data manipulations easier.
The following is the checklist of cleaning and preparing data which is mainly

considered among the best practices.

- **Familiarization with the dataset**: to get good domain knowledge so that one is aware which variable represents what.
- **Check for structural errors:** You may check for mislabelled variables, faulty Data types, non-unique (duplicated values) and string inconsistencies or typing errors.
- **Check for data irregularities:** You may check for the invalid values and outliers.
- **Decide on how to deal missing values:** Either delete the observations if they are not providing any meaningful insights to our data or imputing the data with some logical values like mean or median based on the observations.

Check your Progress 1

1. What is the package used to use JSON Files in R?

   ……………………………………………………………………………….

………………………………………………………………..

2. What are wb and rb mode while dealing with binary files?

   ……………………………………………………………………………….

   ………………………………………………………………..

3. Mention any 2 checklist points used for cleaning/ preparing data?

   ……………………………………………………………………………….

   ………………………………………………………………..

14.5 VISUALIZATION IN R

In the previous section, we have discussed about obtaining input from different types of data. This section explains various types of graphs that can be drawn using R. It may please be noted that only selected types of graphs have been presented here.

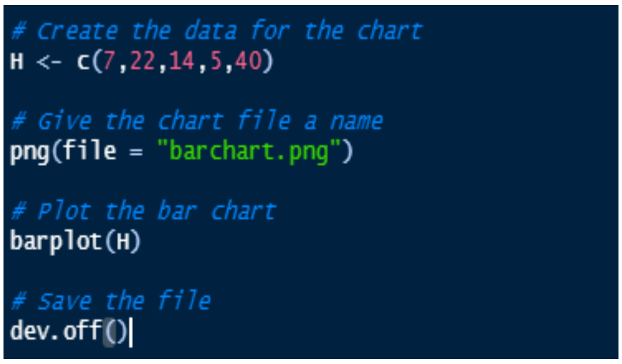
14.5.1 Bar Charts

A bar chart depicts data as rectangular bars whose length is proportionate to the variable's value. The function barplot() in R is used to make bar charts. In a bar chart, R can create both vertical and horizontal bars. Each of the bars in a bar chart can be coloured differently.

Syntax:
barplot(H,xlab,ylab,main, names.arg,col)

where,

- In a bar chart, H is a vector or matrix containing numeric values.
- The x axis label as **xlab**.
- The y axis label is **ylab**.
- The title of the bar chart is **main**.
- **names.arg** is a list of names that appear beneath each bar.
- **col** is used to color the graph's bars.



```
# Create the data for the chart
H <- c(7,22,14,5,40)

# Give the chart file a name
png(file = "barchart.png")

# Plot the bar chart
barplot(H)

# Save the file
dev.off()
```

Figure 14.16: Creating a Bar chart

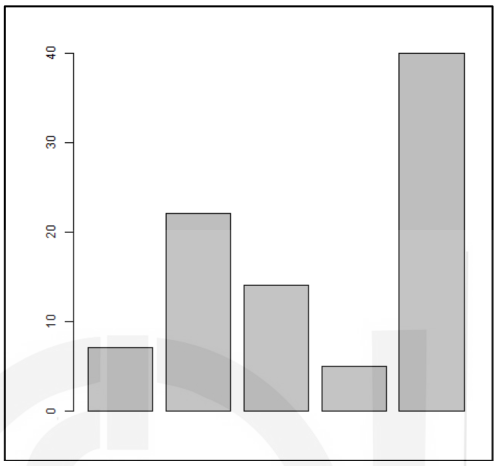
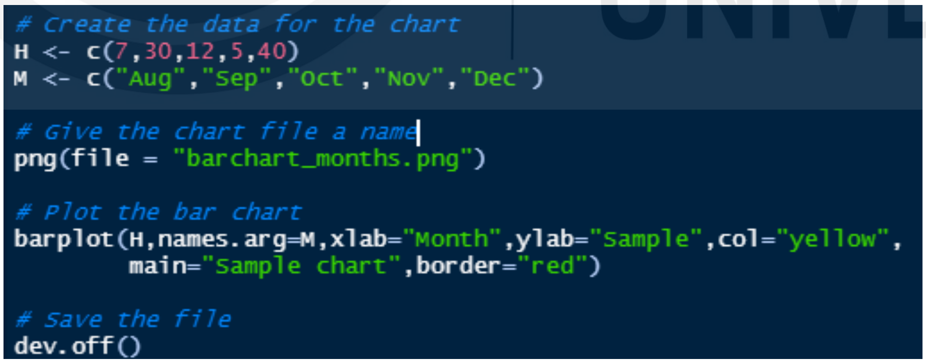


Figure 14.17: A Bar Chart of data of Figure 14.16

Bar Chart Labels, Title and Colors

More parameters can be added to the bar chart to increase its capabilities. The **title** is added using the **main** parameter. Colors are added to the bars using the **col** parameter. To express the meaning of each bar, **args.name** is a vector with the same number of values as the input vector.



```
# Create the data for the chart
H <- c(7,30,12,5,40)
M <- c("Aug","Sep","Oct","Nov","Dec")

# Give the chart file a name
png(file = "barchart_months.png")

# Plot the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Sample",col="yellow",
        main="Sample chart",border="red")

# Save the file
dev.off()
```

Figure 14.18: Function for plotting Bar chart with labels and colours

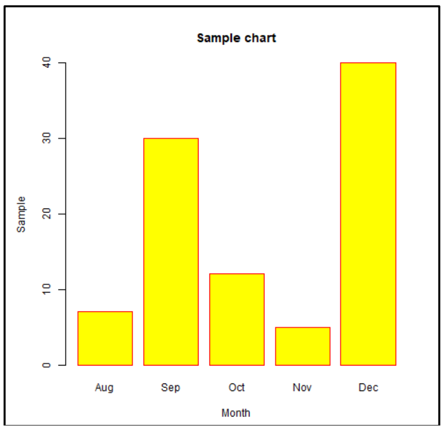


Figure 14.19: A Bar Chart of with labels and colors

14.5.2 Box Plots

In order to determine how evenly the data is distributed in a dataset, you can use boxplot, which is very effective tool. The dataset is split using quartiles. This graph depicts the data set's minimum, first quartile, median, third quartile and maximum. Drawing boxplots for each data set allows you to compare the distribution of data across data sets.

The **boxplot()** function in R is used to make boxplots.

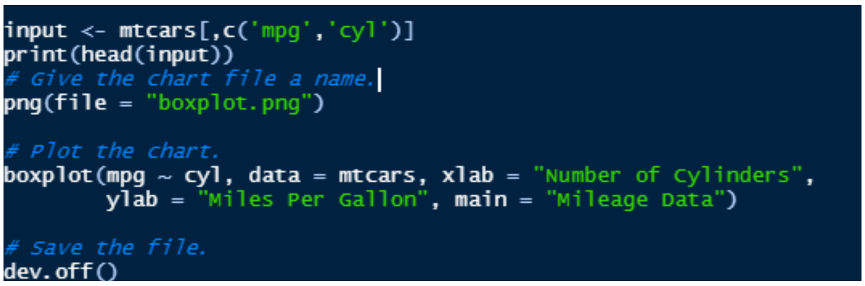
Syntax:

boxplot(x, data, notch, varwidth, names, main)

The parameters of the functions are as follows:

- Parameter **x** either can specify a formula or it can specify a vector.
- Parameter **data** is used to specify the data frame that contains the data required to be plotted.
- Parameter **notch** represents a logical value. In case, you want to draw a notch in the box plot, you may set its value to TRUE.
- Parameter **varwidth** is also logical. It can be set to TRUE, if you want to make the box's width proportional to the sample size.
- Parameter **names** can be used to specify the group labels that will be printed beneath each boxplot.
- main is used to give the graph a **title**.

Creating the Boxplot


```
input <- mtcars[,c('mpg','cyl')]
print(head(input))
# Give the chart file a name.
png(file = "boxplot.png")

# Plot the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Number of Cylinders",
        ylab = "Miles Per Gallon", main = "Mileage Data")

# Save the file.
dev.off()
```
Figure 14.20: Coding for Box plot

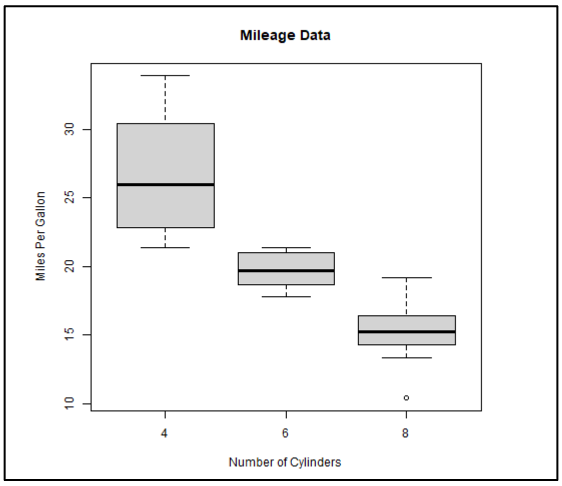


Figure 14.21: A Box plot of Figure 14.20

14.5.3 Histograms

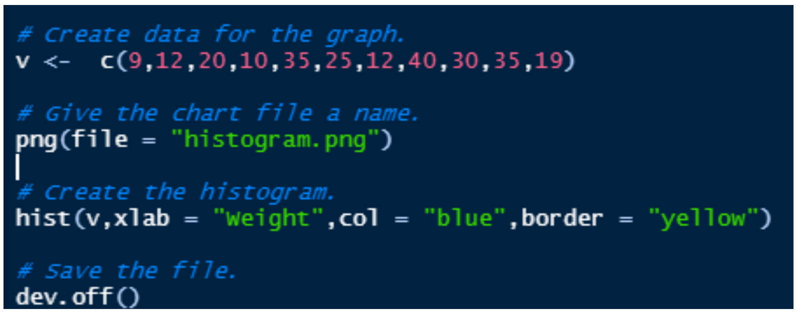
The frequency of values of a variable bucketed into ranges is represented by a histogram. The difference between a histogram and a bar chart is that a histogram groups the numbers into continuous ranges. The height of each bar in a histogram represents the number of items present in that range.

The **hist()** function in R is used to produce a histogram. This function accepts a vector as an input and plots histograms using additional parameters.

Syntax:
hist(v,main,xlab,xlim,ylim,breaks,col,border)

where,

- The parameter **v** is a vector that contains the numeric values for which histogram is to be drawn.
- The title of the chart is shown by the **main**.
- The colour of the bars is controlled by **col**.
- Each bar's border colour is controlled by the **border** parameter.
- The **xlab** command is used to describe the x-axis.
- The x-axis range is specified using the **xlim** parameter.
- The y-axis range is specified with the **ylim** parameter.
- The term "**breaks**" refers to the breadth of each bar.



```
# Create data for the graph.
v <-  c(9,12,20,10,35,25,12,40,30,35,19)

# Give the chart file a name.
png(file = "histogram.png")
|
# Create the histogram.
hist(v,xlab = "weight",col = "blue",border = "yellow")

# Save the file.
dev.off()
```

Figure 14.22: Creating a Histogram

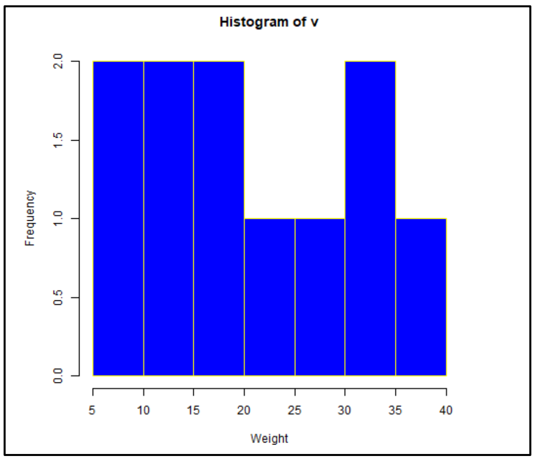


Figure 14.23: Histogram of data used in Figure 14.22

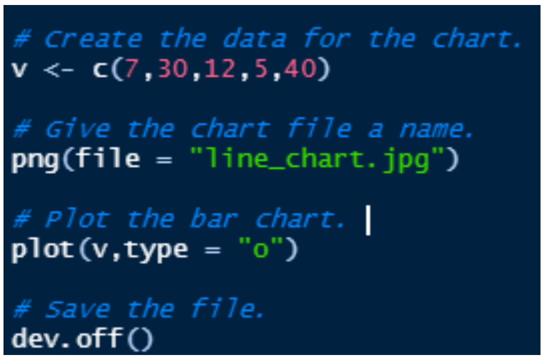
14.5.3 Line Graphs

A graph that uses line segments to connect a set of points is known as the line graph. These points are sorted according to the value of one of their coordinates (typically the x-coordinate). Line charts are commonly used to identify data trends.
The line graph was created using R's **plot()** function.

Syntax:
plot(v,type,col,xlab,ylab)

where,

- The numeric values are stored in **v**, which is a vector.
- **type** takes values, "**p**","**l**","**o**". The value "p" is used to draw only points, "l" is used to draw only lines, and "o" is used to draw both points and lines.
- **xlab** specifies the label for the x axis.
- **ylab** specifies the label for the x axis..
- **Main** is used to specify the title of chart .
- **col** is used to specify the color of the points and/or the lines.



```
# Create the data for the chart.
v <- c(7,30,12,5,40)

# Give the chart file a name.
png(file = "line_chart.jpg")

# Plot the bar chart.
plot(v,type = "o")

# Save the file.
dev.off()
```

Figure 14.24: Function to draw a line chart

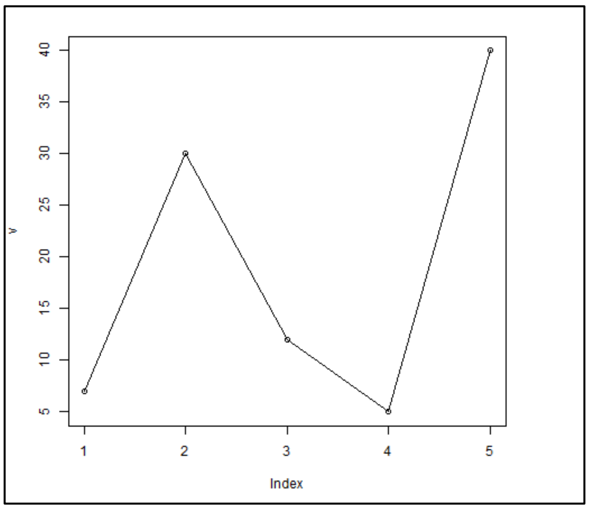
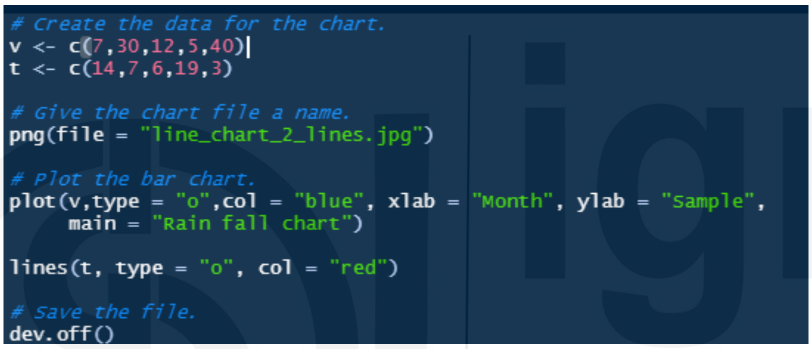


Figure 14.25: A Line Chart for data of Figure 14.24

Multiple Lines in Line Chart & Axis Details



```
# Create the data for the chart.
v <- c(7,30,12,5,40)
t <- c(14,7,6,19,3)

# Give the chart file a name.
png(file = "line_chart_2_lines.jpg")

# Plot the bar chart.
plot(v,type = "o",col = "blue", xlab = "Month", ylab = "Sample",
    main = "Rain fall chart")

lines(t, type = "o", col = "red")

# Save the file.
dev.off()
```

Figure 14.25: Function for Line Chart with multiple lines

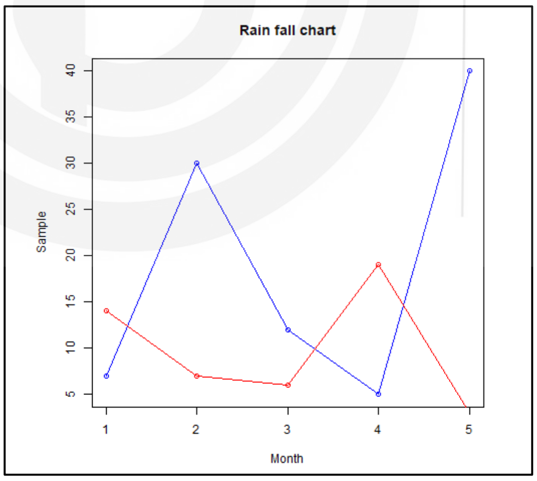


Figure 14.27: A Line Chart with multiple lines for data of Figure 14.26

14.5.4 Scatterplots

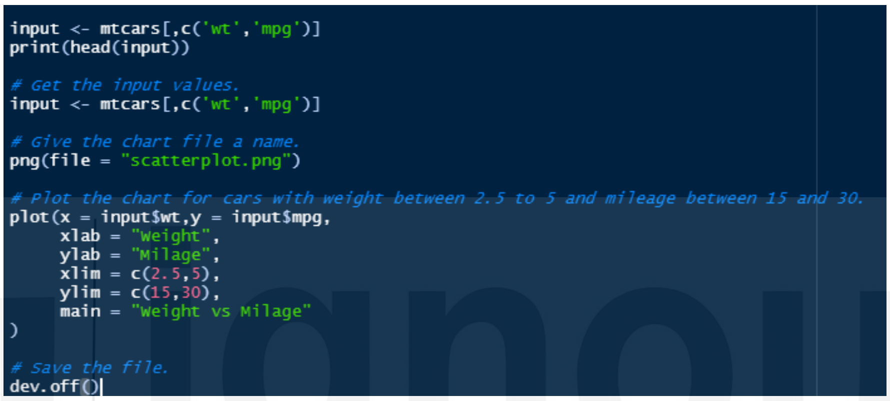
Scatterplots are diagrams that display a large number of points shown in the Cartesian plane. The values of two variables are represented by each point.
One variable is chosen on the horizontal axis, while another is chosen on the vertical axis. To create a simple scatterplot, use the plot() method.

Syntax:

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

The parameters of the plot functions are as follows:

- parameter **x** is the data values for x-axis.
- parameter **y** is the data values for y-axis
- parameter **main** is used for title of the grpah
- parameters **xlab** and **ylab** are used to specify the Labels for x-axis and y-axis respectively.
- Parameters **xlim** and **ylim** used define the limits of values of x and y respectively.
- **axes** specifies if the plot should include both axes.



```
input <- mtcars[,c('wt','mpg')]
print(head(input))

# Get the input values.
input <- mtcars[,c('wt','mpg')]

# Give the chart file a name.
png(file = "scatterplot.png")

# Plot the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
plot(x = input$wt,y = input$mpg,
     xlab = "Weight",
     ylab = "Milage",
     xlim = c(2.5,5),
     ylim = c(15,30),
     main = "Weight vs Milage"
)

# Save the file.
dev.off()
```

Figure 14.28: Plot function to draw Scatter Plot

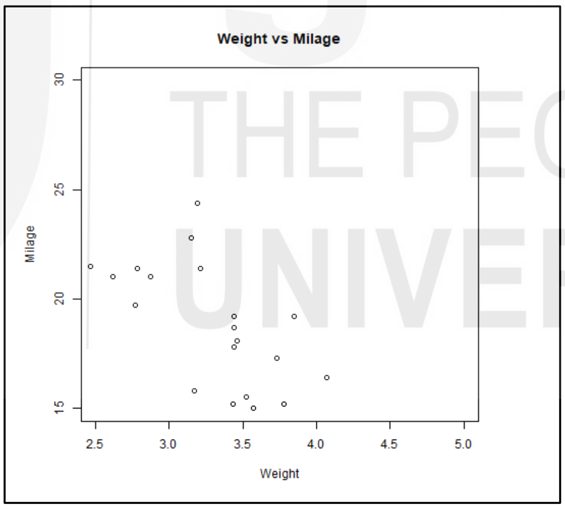


Figure 14.29: Scatter plot for the data of Figure 14.28

Scatterplot Matrices

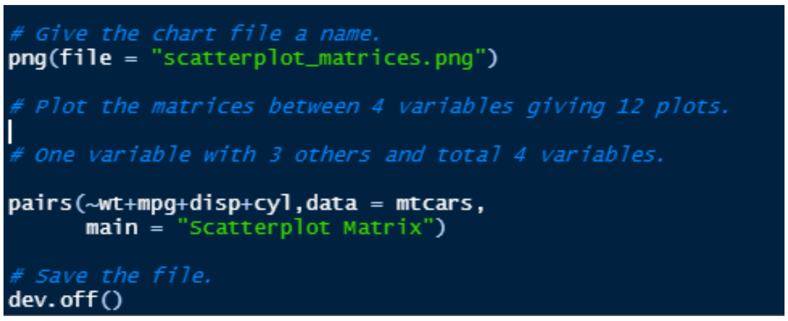
The scatterplot matrix is used when there are more than two variables and you want to identify the correlation between one variable and the others. To make scatterplot matrices, we use **pairs()** function.

Syntax:

pairs(formula, data)

where,

- The **formula** represents a set of variables that are utilised in pairs.
- The data set from which the variables will be derived is referred to as **data**.



```
# Give the chart file a name.
png(file = "scatterplot_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~wt+mpg+disp+cyl,data = mtcars,
      main = "Scatterplot Matrix")

# Save the file.
dev.off()
```

Figure 14.30: Function for Scatterplot matrix

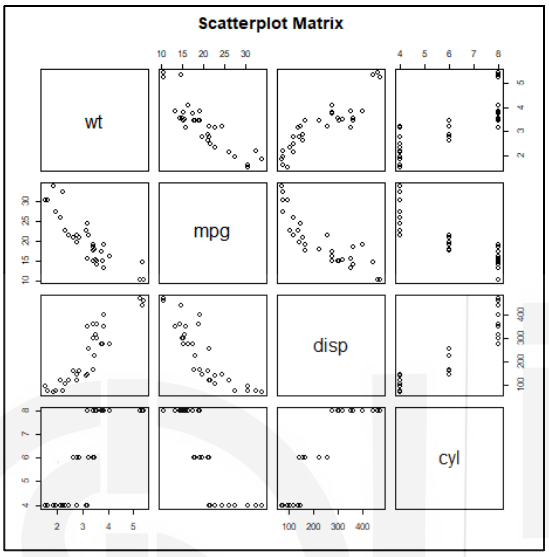


Figure 14.31: A Scatterplot matrix

Check your Progress 2

1. What is scatter plot?
2. When you will use histogram and when you will use bar chart in R?
3. What type of chart you consider when trying to demonstrate "relationship" between variables/parameters?

14.6 Summary

In this unit you have gone though various file types that can be processed for data analysis in R and further discussed their interfaces. R can read and write a variety of file types outside the R environment, including CSV, Excel, binary, XML and JSON. Further, R can readily connect to various relational databases, such as MySQL, Oracle, and SQL Server, and retrieve records as a data frame that can be modified and analysed with all of R's sophisticated packages and functions. The data can also be programmatically extracted from websites using R applications. "RCurl," "XML," and "stringr" are some R packages that are used to scrape data from the web. The unit also explains the concept of data cleaning and pre-processing which is the process of identifying, correcting and removing incorrect raw data, familiarization with the dataset, checking data for structural errors and data irregularities and deciding on how to deal with missing values are the steps involved in cleaning and preparing data which is mainly considered among the best practices. The unit finally explores the concept of visualisations in R. There are various types of graphs and charts including- bar charts, box plots, histograms, line graphs and scatterplots that can be used to visualise the data effectively. The unit explained the usage and syntax for each of the illustration with graphics.

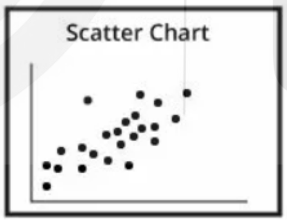
14.7 Answers

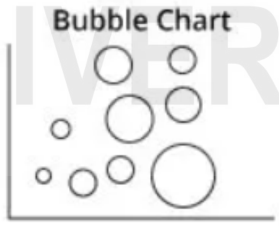
Check your progress 1

1. Install.packages(“**rjson”)**
   library(**rjson)**
2. rb mode opens the file in the binary format for reading and wb mode opens the file in the binary format for writing.
3. The checklist points used for cleaning/ preparing data:
   Check for data irregularities: You may check for the invalid values and outliers.
   Decide on how to deal missing values: Either delete the observations if they are not providing any meaningful insights to our data or imputing the data with some logical values like mean or median based on the observations.

Check your progress 2

1. A scatter plot is a chart used to plot a correlation between two or more variables at the same time
2. We use a histogram to plot the distribution of a continuous variable, while we can use a bar chart to plot the distribution of a categorical variable.
3. When you are trying to show “relationship” between two variables, you will use a scatter plot or chart. When you are trying to show “relationship” between three variables, you will have to use a bubble chart.





14.8 REFERENCES AND FURTHER READINGS

1. De Vries, A., & Meys, J. (2015). *R for Dummies*. John Wiley & Sons.
2. Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Victoria, BC, Canada: Leanpub.
3. Schmuller, J. (2017). *Statistical Analysis with R For Dummies*. John Wiley & Sons.
4. Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage publications.
5. Lander, J. P. (2014). *R for everyone: Advanced analytics and graphics*. Pearson Education.
6. Lantz, B. (2019). *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd.
7. Heumann, C., & Schomaker, M. (2016). *Introduction to statistics and data analysis*. Springer International Publishing Switzerland.
8. Davies, T. M. (2016). *The book of R: a first course in programming and statistics*. No Starch Press.
9. https://www.tutorialspoint.com/r/index.html

UNIT 15 DATA ANALYSIS AND R

Structure **Page Nos.**

15.1 Introduction
15.2 Objectives
15.3 Chi Square Test
15.4 Linear Regression
15.5 Multiple Regression
15.6 Logistic Regression
15.7 Time Series Analysis
15.8 Summary
15.9 Answers
15.10 References and Further Readings

15.1 INTRODUCTION

This unit deals with the concept of data analysis and how to leverage it by using R programming. The unit discusses various tests and techniques to operate on data in R and how to draw insights from it. The unit covers the Chi-Square Test, its significance and the application in R with the help of an example. The unit also familiarises with the concept of Regression Analysis and its types including- Simple Linear and Multiple Linear Regression and afterwards, Logistic Regression. It is further substantiated with examples in R that explain the steps, functions and syntax to use correctly. It also explains how to interpret the output and visualise the data. Subsequently, the unit explains the concept of Time Series Analysis and how to run it on R. It also discusses about the Stationary Time Series, extraction of trend, seasonality, and error and how to create lags of a time series in R.

15.2 OBJECTIVES

After going through this Unit, you will be able to:-

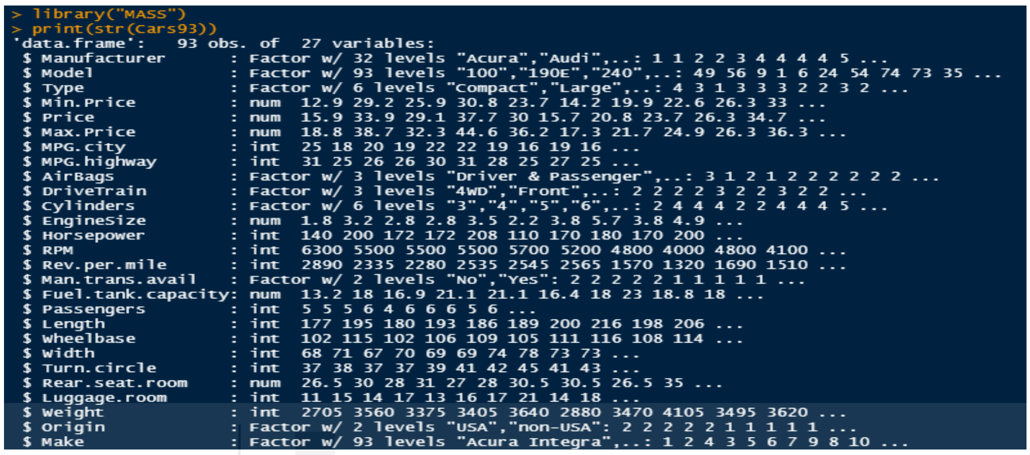
- Run tests and techniques on data and interpret the results using R;
- explain the correlation between two variables in a dataset by running Chi-Square Test in R;
- explain the concept of Regression Analysis and distinguish between their types- simple Linear and Multiple Linear;
- build relationship models in R to plot and interpret the data and further use it to predict the unknown variable values;
- explain the concept of Logistic Regression and its application on R;
- explain about the Time Series Analysis and the special case of Stationary Time Series;
- explain about extraction of trend, seasonality, and error and how to create lags of a time series in R.

15.3 CHI-SQUARE TEST

The Chi-Square test is a statistical tool for determining if two categorical variables are significantly correlated. Both variables should come from the same population and be categorical in nature, such as – top/bottom, True/False, Black/White.Syntax of a chi-square test: chisq.test(data)

EXAMPLE:

Let's consider R's built in "MASS" library that contains Cars93 dataset that represents the sales of different models of car.

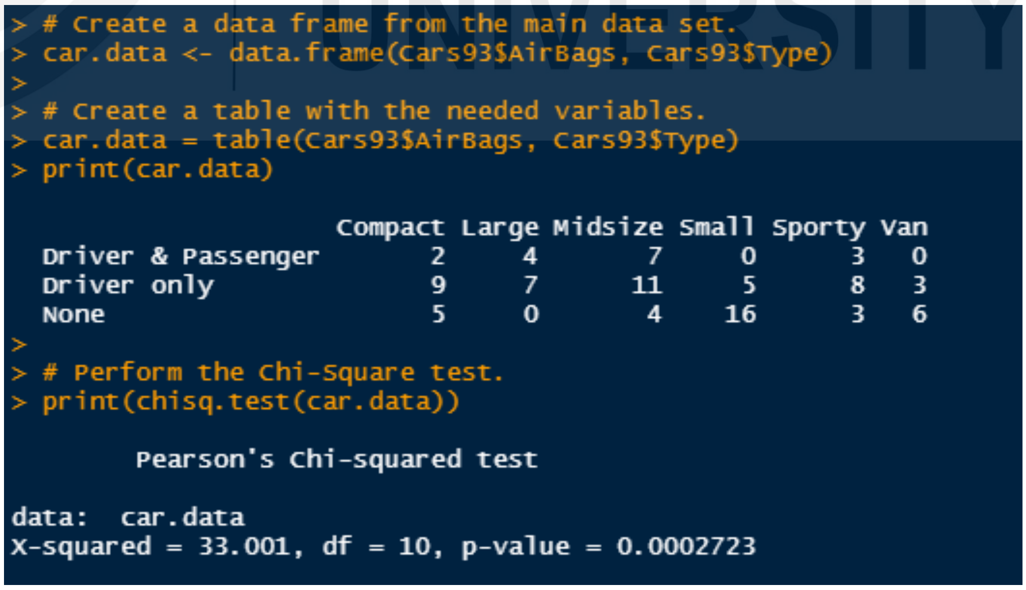


```
> library("MASS")
> print(str(Cars93))
'data.frame':	93 obs. of  27 variables:
 $ Manufacturer      : Factor w/ 32 levels "Acura","Audi",..: 1 1 2 2 3 4 4 4 4 5 ...
 $ Model             : Factor w/ 93 levels "100","190E","240",..: 49 56 9 1 6 24 54 74 73 35 ...
 $ Type              : Factor w/ 6 levels "Compact","Large",..: 4 3 1 3 3 3 2 2 3 2 ...
 $ Min.Price         : num  12.9 29.2 25.9 30.8 23.7 14.2 19.9 22.6 26.3 33 ...
 $ Price             : num  15.9 33.9 29.1 37.7 30 15.7 20.8 23.7 26.3 34.7 ...
 $ Max.Price         : num  18.8 38.7 32.3 44.6 36.2 17.3 21.7 24.9 26.3 36.3 ...
 $ MPG.city          : int  25 18 20 19 22 22 19 16 19 16 ...
 $ MPG.highway       : int  31 25 26 26 30 31 28 25 27 25 ...
 $ AirBags           : Factor w/ 3 levels "Driver & Passenger",..: 3 1 2 1 2 2 2 2 2 2 ...
 $ DriveTrain        : Factor w/ 3 levels "4WD","Front",..: 2 2 2 2 3 2 2 3 2 2 ...
 $ Cylinders         : Factor w/ 6 levels "3","4","5","6",..: 2 4 4 4 2 2 4 4 4 5 ...
 $ EngineSize        : num  1.8 3.2 2.8 2.8 3.5 2.2 3.8 5.7 3.8 4.9 ...
 $ Horsepower        : int  140 200 172 172 208 110 170 180 170 200 ...
 $ RPM               : int  6300 5500 5500 5500 5700 5200 4800 4000 4800 4100 ...
 $ Rev.per.mile      : int  2890 2335 2280 2535 2545 2565 1570 1320 1690 1510 ...
 $ Man.trans.avail   : Factor w/ 2 levels "No","Yes": 2 2 2 2 2 1 1 1 1 1 ...
 $ Fuel.tank.capacity: num  13.2 18 16.9 21.1 21.1 16.4 18 23 18.8 18 ...
 $ Passengers        : int  5 5 5 6 4 6 6 6 5 6 ...
 $ Length            : int  177 195 180 193 186 189 200 216 198 206 ...
 $ Wheelbase         : int  102 115 102 106 109 105 111 116 108 114 ...
 $ Width             : int  68 71 67 70 69 69 74 78 73 73 ...
 $ Turn.circle       : int  37 38 37 37 39 41 42 45 41 43 ...
 $ Rear.seat.room    : num  26.5 30 28 31 27 28 30.5 30.5 26.5 35 ...
 $ Luggage.room      : int  11 15 14 17 13 16 17 21 14 18 ...
 $ Weight            : int  2705 3560 3375 3405 3640 2880 3470 4105 3495 3620 ...
 $ Origin            : Factor w/ 2 levels "USA","non-USA": 2 2 2 2 2 1 1 1 1 1 ...
 $ Make              : Factor w/ 93 levels "Acura Integra",..: 1 2 4 3 5 6 7 9 8 10 ...
```

Figure 15.1: Description of sample data set

As you can see, we have various variables that can be considered as categorical variable. Let's consider "Airbags" and "Type" for our model. You want to check, if there is a correlation in these two categorical variables. Chi-square test is a good indicator for such information. To perform the chi-square test, you may perform the following steps:

- First, you need to extract this data from the dataset (see Figure 15.2).
- Next, create the table of the data(See Figure 15.2) and
- Perform chi square test on the table (See Figure 15.2)



```
> # 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)

                     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
>
> # Perform the Chi-Square test.
> print(chisq.test(car.data))

	Pearson's Chi-squared test

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

Figure 15.2: Chi-square testing

The result shows the p value 0.0002723 which is less than 0.05 which indicates strong correlation. In addition, the value of chi square is also high. Thus, the variable type of car is strongly related to number of air bags.

Chi-square test is one of the most useful test in finding relationships between categorical variables.

How can you find the relationships between two scale or numeric variables using R? One such technique, which helps in establishing a model-based relationship is regression, which is discussed next.

15.4 LINEAR REGRESSION

Regression analysis is a common statistical technique for establishing a relationship model between two variables. One of these variables is known as a predictor variable, and its value is derived via experimentation. The response variable, whose value is generated from the predictor variable, is the other variable.
A regression model that employs a straight line to explain the relationship between variables is known as linear regression. In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is one. It searches for the value of the regression coefficient(s) that minimises the total error of the model to find the line of best fit through your data.

The general equation for a linear regression is –

$$y = a + b \times x$$

In the equation given above:

- y is called response/dependent variable, whereas x is a independent/predictor variable.
- The a and b values are the coefficients used in the equation, which are to be predicted.

The objective of the regression model is to determine the values of these two constants.

There are two main types of linear regression:

- *Simple Linear Regression*: This kind of regression uses only one independent variable, as shown in the equation above.
- *Multiple Linear Regression*: However, if you add more independent variables like: $y = a + b \times x_1 + c \times x_{2+\ldots}$, then it is called multiple regression.

Steps for Establishing a Linear Regression:

A basic example of regression is guessing a person's weight based on his/her height. To do so, you need to know the correlation between a person's height and weight.

The steps to establishing a relationship are as follows:

1. Carry out an experiment in which you collect a sample of observed height and weight values.
2. Create a relationship model using the **lm()** functions in R.
3. Find the coefficients from the model you constructed and use them to create a mathematical equation.

4. To find out the average error in prediction, get a summary of the relationship model. Also known as residuals, as shown in Figure 15.3
5. The **predict()** function in R can be used to predict the weight of new person. A sample regression line and residual are shown in Figure 15.3

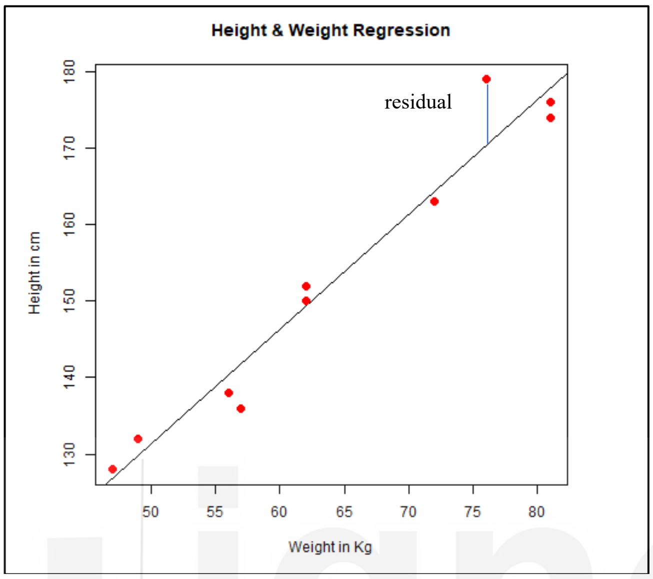
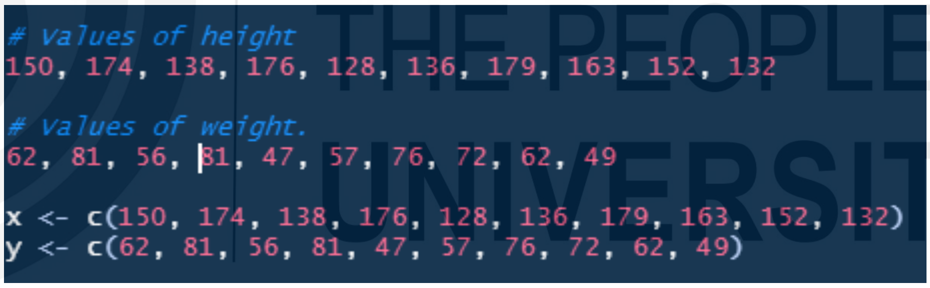


Figure 15.3: An example of regression mode and residual

Input Data

Below is the sample data with the observations between weight and height, which is experimentally collected and is input in the Figure 15.4



```
# Values of height
150, 174, 138, 176, 128, 136, 179, 163, 152, 132

# Values of weight.
62, 81, 56, 81, 47, 57, 76, 72, 62, 49

x <- c(150, 174, 138, 176, 128, 136, 179, 163, 152, 132)
y <- c(62, 81, 56, 81, 47, 57, 76, 72, 62, 49)
```

Figure 15.4: Sample data for linear regression

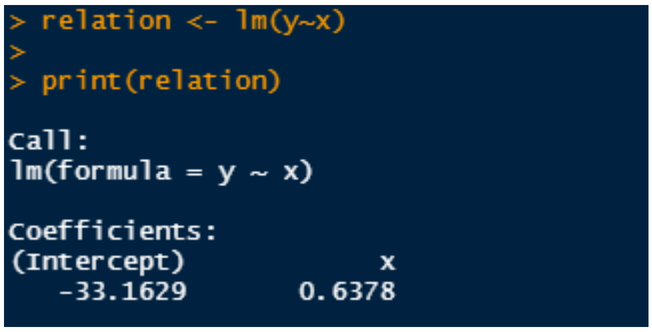
lm() function create the relation model between the variable i.e. predictor and response.

Syntax: lm(formula, data),where

formula: presenting the relation between x and y.

data: data on which the formula needs to be applied.

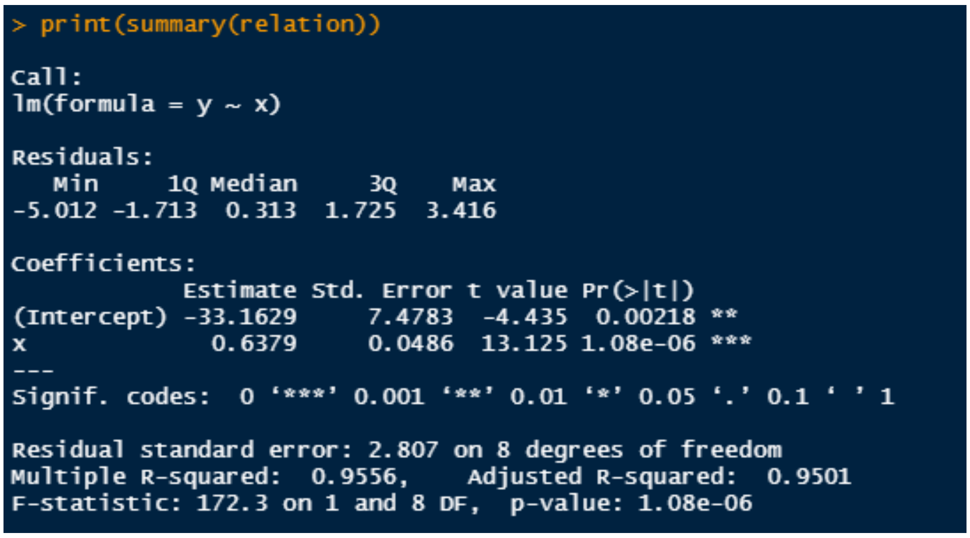
Figure 15.5 shows the use of this function.



```
> relation <- lm(y~x)
>
> print(relation)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   -33.1629       0.6378
```

Figure 15.5: Use of lm function in linear regression

Summary of the relationship:



```
> print(summary(relation))

Call:
lm(formula = y ~ x)

Residuals:
   Min     1Q Median     3Q    Max 
-5.012 -1.713  0.313  1.725  3.416 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -33.1629     7.4783  -4.435  0.00218 ** 
x             0.6379     0.0486  13.125 1.08e-06 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.807 on 8 degrees of freedom
Multiple R-squared:  0.9556,	Adjusted R-squared:  0.9501 
F-statistic: 172.3 on 1 and 8 DF,  p-value: 1.08e-06
```

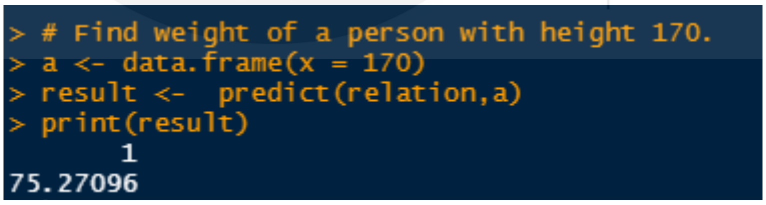
Figure 15.6: Results of regression

The results of regression as presented by R includes the following:

1. Five-point summary (Minimum, First Quartile, Median, Third Quartile, and Maximum). This shows the spread of the residual. You may observe that about 50% of residuals are in the range -1.713 to +1.725, which shows a good model fit.
2. The t value and Pr values for the intercept (that is b in the equation y = ax +b) and x (that is a in the equation y = ax +b).
3. The F-statistics is very high with a low p-value, indicating statistical difference between group means.

Predict function:
Function which will be used to predict the weight of the new person.

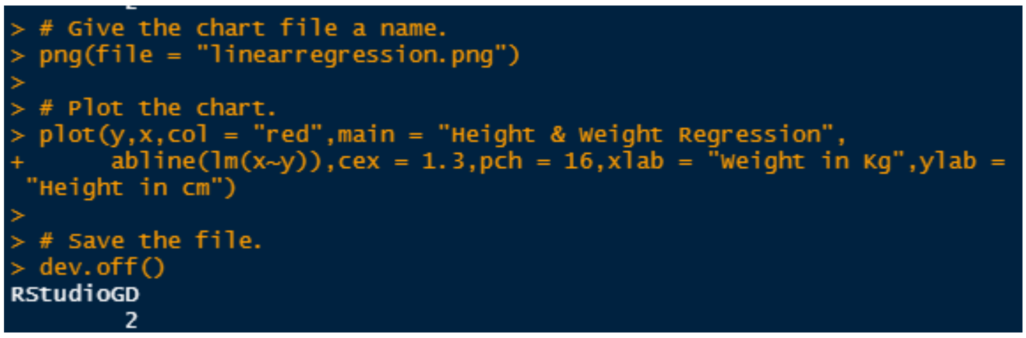
Syntax: Predict(object, newdata),
object is the formula already formulated using lm() function.
newdata is the vector containing new value for predictor variable.



```
> # Find weight of a person with height 170.
> a <- data.frame(x = 170)
> result <-  predict(relation,a)
> print(result)
       1 
75.27096 
```

Figure 15.7: The Predict function

Plot for Visualization: Finally, you may plot these values by setting the plot title and axis titles (see Figure 15.8). The linear regression line is shown in Figure 15.3.



```
> # Give the chart file a name.
> png(file = "linearregression.png")
>
> # Plot the chart.
> plot(y,x,col = "red",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()
RStudioGD
        2
```

Figure 15.8: Making a chart of linear regression

Linear regression has one response variable and one predictor variables, however, in many practical cases there can be more than one predictor variables. This is the case of multiple regression and is discussed next.

15.5 MULTIPLE REGRESSION

The relationship between two or more independent variables and a single dependent variable is estimated using multiple linear regression. When you need to know the following, you can utilize multiple linear regression.

- The degree to which two or more independent variables and one dependent variable are related (e.g. how baking soda, baking temperature, and amount of flour added affect the taste of cake).
- The dependent variable's value at a given value of the independent variables (e.g. the taste of cake for different amount of baking soda, baking temperature, and flour).

The general equation for multiple linear regression is –

y = a + b1X1 + b2X2 +...bnXn

where,

- **y** is response variable.
- **a, b1, b2...bn** are coefficients.
- **X1, X2, ...Xn** are predictor variables.

The **lm()** function in R is used to generate the regression model. Using the input data, the model calculates the coefficient values. Using these coefficients, you can then predict the value of the response variable for a given collection of predictor variables.

lm() Function:

The relationship model between the predictor and the response variable is created using this function.

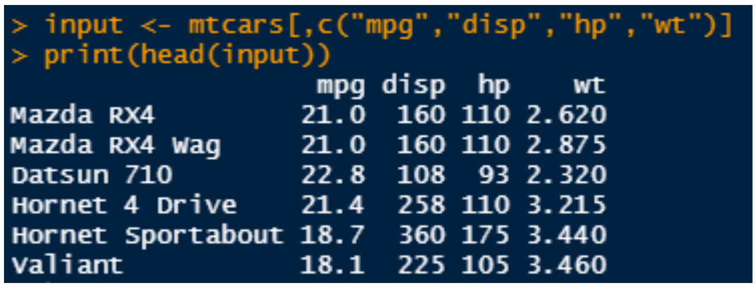
Syntax: The basic syntax for **lm()** function in multiple regression is –

lm(y ~ x1+x2+x3...,data),

The relationship between the response variable and the predictor variables is represented by a **formula**. The vector on which the formula will be applied is called **data**.

INPUT Data

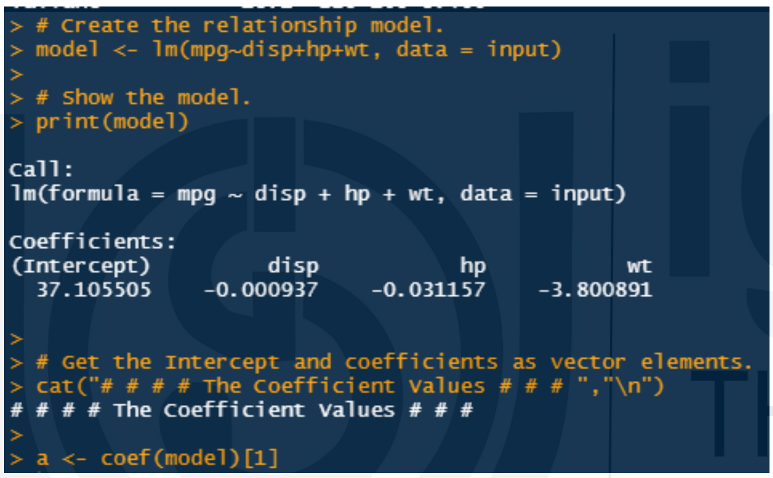
Let's take the R inbuilt data set "mtcars", which gives comparison between various car models based on the mileage per gallon (mpg), cylinder displacement ("disp"), horse power("hp"), weight of the car("wt") & more.
The aim is to establish relationship of mpg (response variable) with predictor variable (disp, hp, wt). The head function, as used in Figure 15.9, shows the first 5 rows of the dataset.



```
> input <- mtcars[,c("mpg","disp","hp","wt")]
> print(head(input))
                   mpg disp  hp    wt
Mazda RX4         21.0  160 110 2.620
Mazda RX4 Wag     21.0  160 110 2.875
Datsun 710        22.8  108  93 2.320
Hornet 4 Drive    21.4  258 110 3.215
Hornet Sportabout 18.7  360 175 3.440
Valiant           18.1  225 105 3.460
```

Figure 15.9: Sample data for Multiple regression

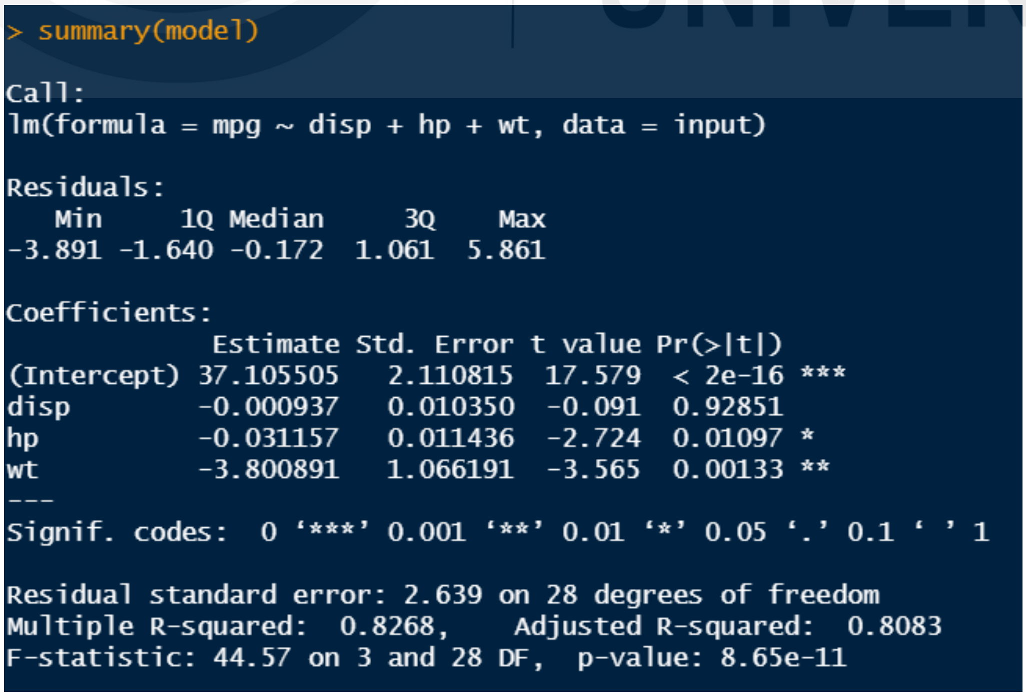
Creating Relationship model & getting the coefficients



```
> # Create the relationship model.
> model <- lm(mpg~disp+hp+wt, data = input)
>
> # Show the model.
> print(model)

Call:
lm(formula = mpg ~ disp + hp + wt, data = input)

Coefficients:
(Intercept)         disp           hp           wt
  37.105505    -0.000937    -0.031157    -3.800891

>
> # Get the Intercept and coefficients as vector elements.
> cat("# # # # The Coefficient Values # # # ","\n")
# # # # The Coefficient Values # # #
>
> a <- coef(model)[1]
```

Figure 15.10: The Regression model

Please note that the *input* is the name of a variable, which was created in Figure 15.9.



```
> summary(model)

Call:
lm(formula = mpg ~ disp + hp + wt, data = input)

Residuals:
   Min     1Q Median     3Q    Max
-3.891 -1.640 -0.172  1.061  5.861

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.105505   2.110815  17.579  < 2e-16 ***
disp        -0.000937   0.010350  -0.091  0.92851
hp          -0.031157   0.011436  -2.724  0.01097 *
wt          -3.800891   1.066191  -3.565  0.00133 **
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.639 on 28 degrees of freedom
Multiple R-squared:  0.8268,    Adjusted R-squared:  0.8083
F-statistic: 44.57 on 3 and 28 DF,  p-value: 8.65e-11
```

Figure 15.11: Display of various output parameters

The results of regression as presented by R includes the following:

1. Five-point summary (Minimum, First Quartile, Median, Third Quartile, and Maximum). This shows the spread of the residual. You may observe that about 50% of residuals are in the range -1.640 to +1.061, which shows a good model fit.
2. Low p values mean the model is statistically significant.

Creating Equation for Regression Model: Based on the intercept & coefficient values one can create the mathematical equation as follows:

$$Y = a + b \times x_{disp} + c \times x_{hp} + d \times x_{wt}$$

or

$$Y = 37.15 - 0.000937 \times x_{disp} - 0.0311 \times x_{hp} - 3.8008 \times x_{wt}$$

The same equation will be applied in predicting new values.

Check your Progress 1

1. What is linear regression?

   ……………………………………………………………………………..

   ……………………………………………………………………………..
2. What does chi-square test answers?

   ……………………………………………………………………………

   ……………………………………………………………………………
3. Difference between linear and multiple regression?

   ……………………………………………………………………………

   ……………………………………………………………………………

15.6 LOGISTIC REGRESSION

In R Programming, logistic regression is a classification algorithm for determining the probability of event success and failure. When the dependent variable is binary (0/1, True/False, Yes/No), logistic regression is utilised. In a binomial distribution, the logit function is utilised as a link function.
Binomial logistic regression is another name for logistic regression. It is based on the sigmoid function, with probability as the output and input ranging from -∞ to +∞.The sigmoid function is given below:

$$g(z) = \frac{1}{1 + e^{-z}} \text{ Where } z = a + b \times x$$

The general equation for logistic regression is –

$$g(z) = \frac{1}{1 + e^{-(a + b_1 \times x_1 + b_2 \times x_2 + b_3 \times x_3 + \ldots)}}$$

where, **y** is called as the response variable, and x_i are predictors.
The a and b_i are coefficients.

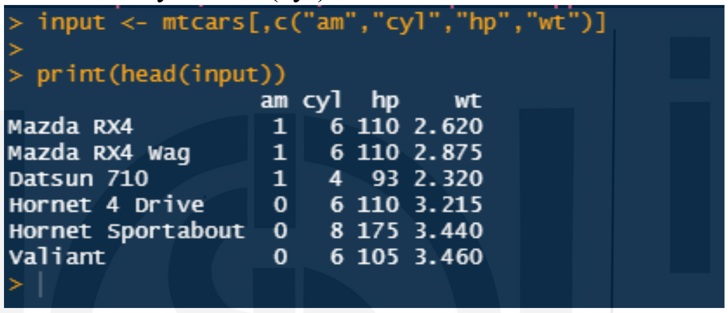
The glm() function is used to construct the regression model.

Syntax:

glm (formula, data,family)

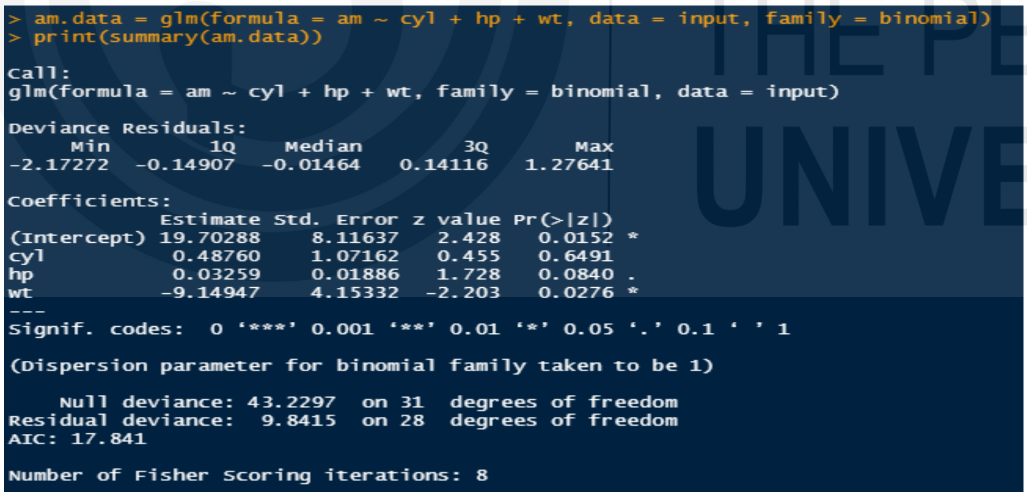
- The symbol expressing the relationship between the variables is a formula.
- The data set containing the values of these variables is known as data.
- family is a R object that specifies the model's details. For logistic regression, it has a binomial value.

Input Data: Let's take the R inbuilt data set "mtcars", which provides details of various car models & engine specifications. The transmission mode of the car i.e. whether the car is manual or automatic is described by the column *am* having a binary value as 0 or 1. You can create the model between columns "am" (Outcome/ dependent/ response variable) and three others – hp, wt and cyl (predictor variables).This model is aimed at determining, if car would have manual or automatic transmission, given the horse power (hp), weight (wt) and number of cylinders (cyl) in the car.



```
> input <- mtcars[,c("am","cyl","hp","wt")]
>
> print(head(input))
                  am cyl  hp    wt
Mazda RX4          1   6 110 2.620
Mazda RX4 Wag      1   6 110 2.875
Datsun 710         1   4  93 2.320
Hornet 4 Drive     0   6 110 3.215
Hornet Sportabout  0   8 175 3.440
Valiant            0   6 105 3.460
>
```

Figure 15.12: The sample data set for logistic regression



```
> am.data = glm(formula = am ~ cyl + hp + wt, data = input, family = binomial)
> print(summary(am.data))

Call:
glm(formula = am ~ cyl + hp + wt, family = binomial, data = input)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-2.17272  -0.14907  -0.01464   0.14116   1.27641

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 19.70288    8.11637   2.428   0.0152 *
cyl          0.48760    1.07162   0.455   0.6491
hp           0.03259    0.01886   1.728   0.0840 .
wt          -9.14947    4.15332  -2.203   0.0276 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 43.2297  on 31  degrees of freedom
Residual deviance:  9.8415  on 28  degrees of freedom
AIC: 17.841

Number of Fisher Scoring iterations: 8
```

Figure 15.13: The logistic regression model

The null deviance demonstrates how well a model with an intercept term can predict the dependent variable, whereas the residual deviance represents how well a model with n predictor variables can predict the dependent variable. Deviance is measure of goodness of fit of a model.

In the summary as the p-value is more than 0.05 for the variables "cyl" (0.0152) and "hp" (0.0276), we will consider them insignificant in contributing to the value of the variable "am". Only weight (wt) impacts the "am" value in this regression model.

15.7 TIME SERIES ANALYSIS

A Time Series is any metric that is measured at regular intervals. It entails deriving hidden insights from time-based data (years, days, hours, minutes) in order to make informed decisions. When you have serially associated data, time series models are particularly beneficial. Weather data, stock prices, industry projections, and so on are just a few examples.

A time series is represented as follows:

A data point, say (Y_t), at a specific time *t* (indicated by subscript *t*) is defined as the either sum or product of the following three components:

Seasonality (S_t), Trend (T_t); and Error (e_t) (also known as, **White Noise**).

Input: Import the data set and then use ts() function.
The steps to use the function are given below. However, it is pertinent to note here that the input values used in this case should ideally be a numeric vector belonging to the "numeric" or "integer" class.
The following functions will generate quarterly data series from 1959:
ts(inputData, frequency =4, start = c(1959,2)) #frequency 4 => QuarterlyData
The following function will generate monthly data series from 1990
ts(1:10, frequency =12, start = 1990) #freq 12 => MonthlyData
The following function will generate yearly data series from 2009 to 2014.
ts(inputData, start=c(2009), end=c(2014), frequency=1) # YearlyData

In case, you want to use Additive Time Series, you use the following:

$$Y_t = S_t + T_t + e_t$$

However, for Multiplicative Time Series, you may use:

$$Y_t = S_t \times T_t \times e_t$$

The additive time series can be converted from multiplicative time series by taking using the log function on the time series as represented below:

$$additiveTS = log(multiplcativeTS)$$

15.7.1 Stationary Time Series

A time series is considered "stationary" if the following criteria are satisfied:

1. When the mean value of a time series remains constant over a period of time and hence, the trend component is removed Over time, the variance does not increase.
2. Seasonality has a minor impact.

This means it has no trend or seasonal characteristics, making it appear to be random white noise regardless of the time span viewed.

Steps to convert a time series as stationary
Each data point in a time series is differentiated by subtracting it from the one before it. It is a frequent technique for making a time series immobile. To make a stationary series out of most time series patterns 1 or 2 differencing is required.

15.7.2 Extraction of trend, seasonality and error

Using decompose() and forecast::stl, the time series is separated into seasonality, trend, and error components (). You may use the following set of commands to do so.

```
timeSeriesData = EuStockMarkets[,1]
resultofDecompose = decompose(timeSeriesData, type=”mult”)
plot(resultofDecompose)
resultsofSt1 = stl(timeSeriesData, s.window = “periodic”)
```

15.7.3 Creating lags of a time-series

A lag of time series is generated when the time basis is shifted by a given number of periods. Moreover, the state of a time series a few periods ago, however, may still have an effect on its current state. Hence, in the time series models, the delays of a time series are typically used as explanatory variables.

```
lagTimeSeries = lag(timeSeriesData, 3) #Shifting to 3 periods earlier
library(DataCombine)
mydf = as.data.frame(timeSeriesData)
mydf = slide(mydf, “x”, NewVar = “xLag1”, slideBy = -1) #create lag1
variable
mydf = slide(mydf, “x”, NewVar = “xLag1”, slideBy = 1)
```

Check your Progress 2

1. What is logistic regression?

   ……………………………………………………………………………..

2. What are the uses of Time-Series analysis?

   …………………………………………………………………………….

3. Differentiate between linear regression and logistic regression?

   …………………………………………………………………………….

15.8 SUMMARY

This unit introduces the concept of data analysis and examine its application using R programming. It explains about the Chi-Square Test that is used to determine if two categorical variables are significantly correlated and further study its application on R. The unit explains the Regression Analysis, which is a common statistical technique for establishing a relationship model between two variables- a predictor variable and the response variable. It further explains the various models in Regression Analysis including Linear and Logistics Regression Analysis. In Linear Regression the two variables are related through an equation of degree is one and employs a straight line to explain the relationship between variables. It is categorised into two types- Simple Linear Regression which uses only one independent variable and Multiple Linear Regression which uses two or more independent variables. Once familiar with the Regression, the unit proceeds to explain about the logistic regression, which is a classification algorithm for determining the probability of event success and failure. It is also known as Binomial logistic regression and is based on the sigmoid function, with probability as the output and input ranging from $-\infty$ to $+\infty$. At the end, the unit introduces the concept of time series analysis and help understand its application and usage on R. It also discusses the special case of Stationary Time Series and how to make a time series stationary. This section further explains how to extract the trend, seasonality and error in a time series in R and the creating lags of a time series.

15.9 ANSWERS

Check your Progress 1

1. A regression model that employs a straight line to explain the relationship between variables is known as linear regression. In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is one. It searches for the value of the regression coefficient(s) that minimises the total error of the model to find the line of best fit through your data.
2. The Chi-square test of independence determines whether there is a statistically significant relationship between categorical variables. It's a hypothesis test that answers the question—do the values of one categorical variable depend on the value of other categorical variables?
3. Linear regression considers 2 variables whereas multiple regression consists of 2 or more variables.

Check your Progress 2

1. Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring.
2. Time series analysis is used to identify the fluctuation in economics and business. It helps in the evaluation of current achievements. Time series is used in pattern recognition, signal processing, weather forecasting and earthquake prediction.
3. The problems pertaining to regression are solved using linear regression; however, the problems pertaining to classification are solved using the logistic regression. The linear regression yields a continuous result, whereas logistic regression yields discrete results.

15.10 REFERENCES AND FURTHER READINGS

1. De Vries, A., & Meys, J. (2015). *R for Dummies*. John Wiley & Sons.
2. Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Victoria, BC, Canada: Leanpub.
3. Schmuller, J. (2017). *Statistical Analysis with R For Dummies*. John Wiley & Sons.
4. Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage publications.
5. Lander, J. P. (2014). *R for everyone: Advanced analytics and graphics*. Pearson Education.
6. Lantz, B. (2019). *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd.
7. Heumann, C., & Schomaker, M. (2016). *Introduction to statistics and data analysis*. Springer International Publishing Switzerland.
8. Davies, T. M. (2016). *The book of R: a first course in programming and statistics*. No Starch Press.
9. https://www.tutorialspoint.com/r/index.html
10. https://data-flair.training/blogs/chi-square-test-in-r/
11. http://r-statistics.co/Time-Series-Analysis-With-R.html
12. http://r-statistics.co/Logistic-Regression-With-R.html

UNIT 16 ADVANCE ANALYSIS USING R

Structure **Page Nos.**

16.0 Introduction
16.1 Objectives
16.2 Decision Trees
16.3 Random Forest
16.4 Classification
16.5 Clustering
16.6 Association rules
16.7 Summary
16.7 Answers
16.8 References and Further Readings

16.0 INTRODUCTION

This unit explores the concepts pertaining to advance level of data analysis and their application in R. The unit explains the theory and the working of the decision tree model and how to run it on R. It further discusses its various types that may fall under the 2 categories based on the target variables. The unit also explores the concept of Random Forest and discusses its application on R. In the subsequent sections, the unit explains the details of classification algorithm and its features and types. It further explains the unsupervised learning technique- Clustering and its application in R programming. It further discusses the 2 types of clustering in R programming including the concept and algorithm of K-Means Clustering. The unit concludes by drawing insights on the theory of the Association Rules and its application in R.

16.1 OBJECTIVES

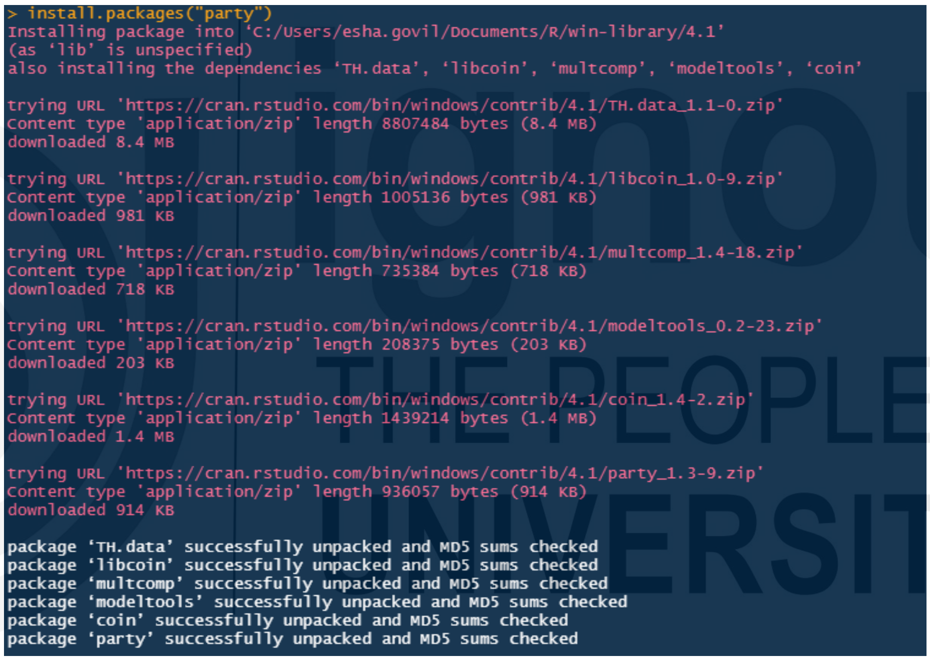
After going through this Unit, you will be able to:

- explain the concept of Decision Tree- including its types and application in R;
- explain the working of Decision Tree and the factors to consider when choosing a tree in R;
- explain the concept of Random Forest and its application on R;
- explain the concept of Classification algorithm and its features including- classifier, characteristics, binary classification, multi-class classification and multi-label classification;
- explain the types of classifications including- linear classifier and its types, support vector machine and decision tree;
- explain the concept of Clustering and its application in R;
- explain the methods of Clustering and their types including the K-Means clustering and its application in R;
- explain the concept and the theory behind the Association Rule Mining and its further application in R Language.

16.2 DECISION TREES

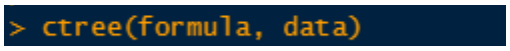
A decision tree is a graph that represents decisions and their results in a tree format. Graph nodes represent events or selections, and graph edges represent decision rules or conditions. It is primarily used in machine learning and data mining applications that use R. Examples of use of decision trees include predicting email as spam or non-spam, predicting cancerous tumours, or predicting credit based on the credit risk of each of these factors. Models are typically built using observational data, also known as training data. Then use a set of validation data to validate and improve the model. R has packages used to build and visualize decision trees. For a new set of predictors, this model is used to reach decisions about the categories of data (yes / no, spam / non-spam).

Installing R Package: Package "party" is used for decision tree. It has a function ctree() which is used to create and analyse decision tree. Figure 16.1 shows the output, when you install the Package using install command.



```
> install.packages("party")
Installing package into 'C:/Users/esha.govil/Documents/R/win-library/4.1'
(as 'lib' is unspecified)
also installing the dependencies 'TH.data', 'libcoin', 'multcomp', 'modeltools', 'coin'

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/TH.data_1.1-0.zip'
Content type 'application/zip' length 8807484 bytes (8.4 MB)
downloaded 8.4 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/libcoin_1.0-9.zip'
Content type 'application/zip' length 1005136 bytes (981 KB)
downloaded 981 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/multcomp_1.4-18.zip'
Content type 'application/zip' length 735384 bytes (718 KB)
downloaded 718 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/modeltools_0.2-23.zip'
Content type 'application/zip' length 208375 bytes (203 KB)
downloaded 203 KB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/coin_1.4-2.zip'
Content type 'application/zip' length 1439214 bytes (1.4 MB)
downloaded 1.4 MB

trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.1/party_1.3-9.zip'
Content type 'application/zip' length 936057 bytes (914 KB)
downloaded 914 KB

package 'TH.data' successfully unpacked and MD5 sums checked
package 'libcoin' successfully unpacked and MD5 sums checked
package 'multcomp' successfully unpacked and MD5 sums checked
package 'modeltools' successfully unpacked and MD5 sums checked
package 'coin' successfully unpacked and MD5 sums checked
package 'party' successfully unpacked and MD5 sums checked
```

Figure 16.1: Installation of package party used for decision trees



```
> ctree(formula, data)
```

Syntax: Defining tree formula describes the predictor and response variables and data is the name of dataset used. The following are the different types of decision trees that can be created using this package.

- **Decision Stump**: It is used to generate decision trees with only one split and is therefore also known as one-level decision tree. In most cases, known for its low predictive performance due to its simplicity.

- **M5**: It is known for its exact classification accuracy, and the ability to work well with small noisy datasets.

- **ID3 (Iterative Dichroatiser 3):** One of the core and a wide range of decision structures is the best attribute for classifying the specified record with the top-down, greedy search approach via the specified dataset.

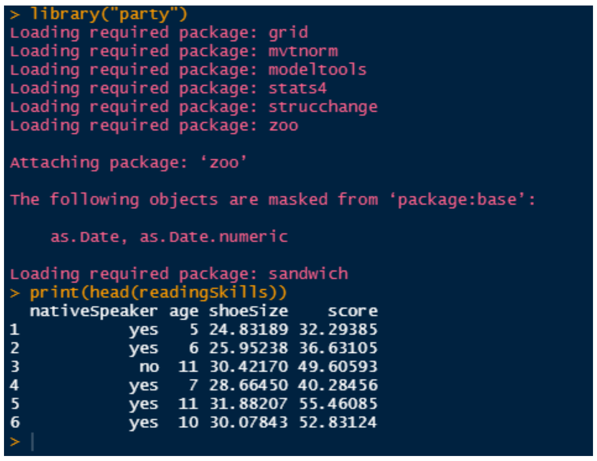
- **C4.5**: This type of decision tree, known as a statistical classifier, is derived from its parent ID3. This creates a decision based on the predictor's bundle.
- **C5.0**: As a successor to C4.5, there are two models, the base tree and the rule-based model, whose nodes can only predict category targets.
- **CHAID**: This algorithm is extended as a chi-square automatic interaction detector and basically examines the merged variables and justifies the result of the dependent variable by building a predictive model.
- **MARS**: Extended as a multivariate adaptive regression spline, this algorithm builds a set of piecewise linear models used to model anomalies and interactions between variables. They are known for their ability to process numerical data more efficiently.
- **Conditional inference tree**: This is a type of decision tree that recursively separates response variables using the conditional inference framework. It is known for its flexibility and strong fundamentals.
- **CART**: Expanded as a classification and regression tree, the value of the target variable is predicted if it is continuous. Otherwise, if it is a category, the required class is identified.

There are many types of decision tree but all of them fall under two main categories based on the target variable.

- **Categorical Variable:** It refers to the variables whose target variables has definite set of values and belong to a group.
- **Continuous Variable**: It refers to the variables whose target variables can choose value from the wide range of data types.

Input Data:

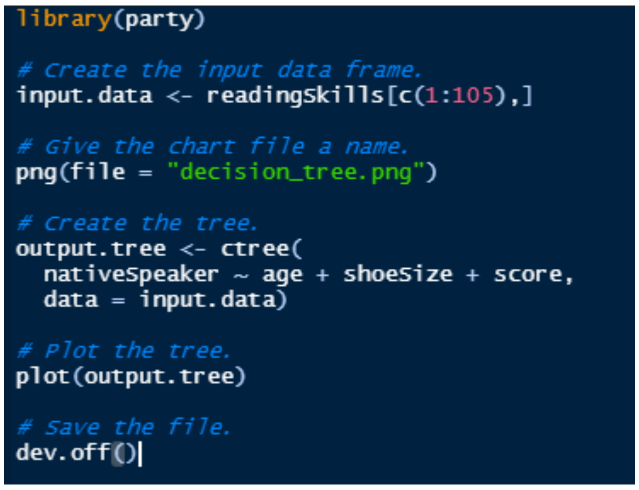
Use R's built-in dataset “readingSkills” to build a decision tree. If you know the age, shoe size, score (raw score on reading test), it represents the person's reading literacy score and also if the person is native speaker or not. Figure 16.2 shows this data.



```
> library("party")
Loading required package: grid
Loading required package: mvtnorm
Loading required package: modeltools
Loading required package: stats4
Loading required package: strucchange
Loading required package: zoo

Attaching package: ‘zoo’

The following objects are masked from ‘package:base’:

    as.Date, as.Date.numeric

Loading required package: sandwich
> print(head(readingSkills))
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
> |
```

Figure 16.2: Sample data for decision tree

Let's use **ctree()** function on the above data set to create decision tree and its graph.



```
library(party)

# Create the input data frame.
input.data <- readingSkills[c(1:105),]

# Give the chart file a name.
png(file = "decision_tree.png")

# Create the tree.
output.tree <- ctree(
  nativeSpeaker ~ age + shoeSize + score,
  data = input.data)

# Plot the tree.
plot(output.tree)

# Save the file.
dev.off()
```

Figure 16.3: Making decision tree

Output: The ellipse in the diagram represents a node of decision tree. It shows the name of the variable and the calculated p-value. The links are marked with the cut-off values on which the decision is taken. From the decision tree of Figure 16.4, we can conclude that people whose reading skills is less than 38.306 and age more than 6 is not a native speaker. The black rectangles states that they are native speakers and the grey one shows they aren't the native speakers. The reading score greater than 38.306 determines that the probability of determining 0.6+ is "yes" (native speaker) and the remaining probability is "no" (not a native speaker). People whose age is less than 6 and reading skills greater than 30.766 are native speakers and reading skills less than equal to 30.766 are not native speakers.

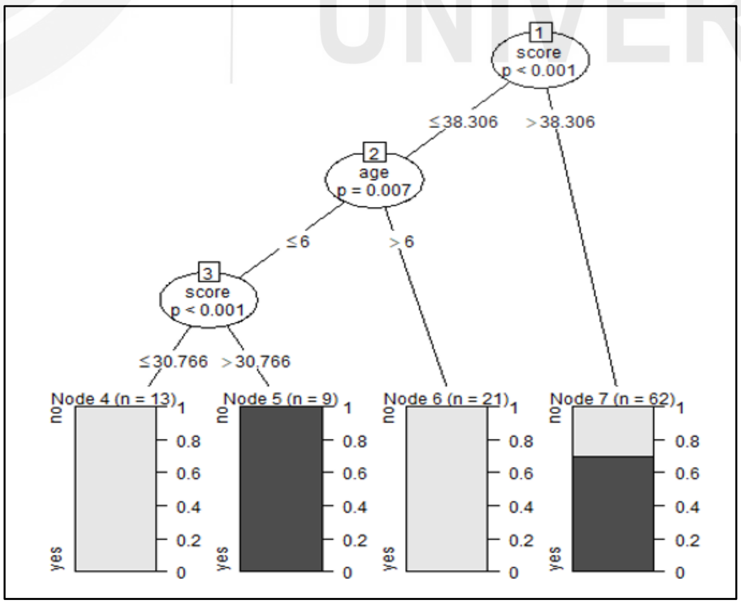


Figure 16.4: The Decision Tree for the example

Working of Decision Tree:

- **Partitioning:** Refers to the process of partitioning a dataset into subsets. The decision to make a strategic split has a significant impact on the accuracy of the tree. Many algorithms are used in the tree to divide a node into sub-nodes. As a result, the overall clarity of the node with respect to the target variable is improved. To do this, various algorithms such as chi-square and Gini coefficient are used and the most efficient algorithm is selected.

- **Pruning:** This refers to the process of turning a branch node into a leaf node and shortening the branches of a tree. The essence behind this idea is that most complex classification trees fit well into training data, but do not do the compelling task of classifying new values, thus avoiding overfitting with simpler trees. That is.

- **Tree selection:** The main goal of this process is to select the smallest tree that fits your data for the reasons described in the pruning section.

Important factors to consider when choosing a tree in R

- **Entropy:** Mainly used to determine the uniformity of a particular sample. If the sample is perfectly uniform, the entropy is 0, and if it is evenly distributed, the entropy is 1. The higher the entropy, the harder it is to draw conclusions from this information.

- **Information Gain:** Statistical property that measures how well the training samples are separated based on the target classification. The main idea behind building a decision tree is to find the attributes that provide the minimum entropy and maximum information gain. It is basically a measure of the decrease in total entropy and is calculated by taking the difference between the undivided entropy of the dataset and the average entropy after it is divided, based on the specified attribute value.

16.3 RANDOM FOREST

Random forests are a set of decision trees that are used in supervised learning algorithms for classification and regression analysis, but primarily for classification. This classification algorithm is non-linear. To achieve more accurate predictions and forecasts, Random Forest creates and combines numerous decision trees together. However, when utilised alone, each decision tree model is used. In the cases where the tree is not built, error estimation is performed. This method is termed as the out-of-bag percent error estimation.

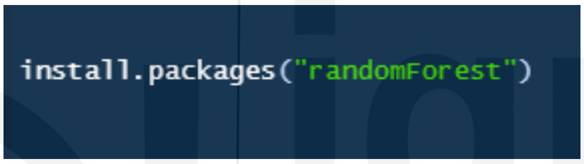
The "Random Forest" is named 'random' since the predictors are chosen at random during training. It is termed as 'forest' because a Random Forest makes decisions based on the findings of several trees. Since multiple uncorrelated trees (models) that operate as committees are always better than individual composition models, therefore the random forests are considered to be better than the decision trees.

Random forest attempts to develop a model using samples from observations and random beginning variables (columns).

The random forest algorithm is:

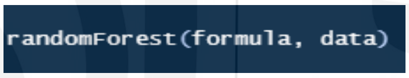
- Draw size n bootstrap random samples (randomly select n samples from the training data).
- Build a decision tree from the bootstrap sample. Randomly select features on each tree node.
- Split the node using a feature (variable) that provides the best split according to the objective function. One such example is to maximise the information gain.
- Repeat the first two steps “k” number of times, where k represents the number of trees that you will create from subset of the sample.
- Aggregate the predictions from each tree of new data points and assign a class label by majority vote. Select the selected group in the most trees and assign new data points to that group.

Install R package



```
install.packages("randomForest")
```

Syntax:

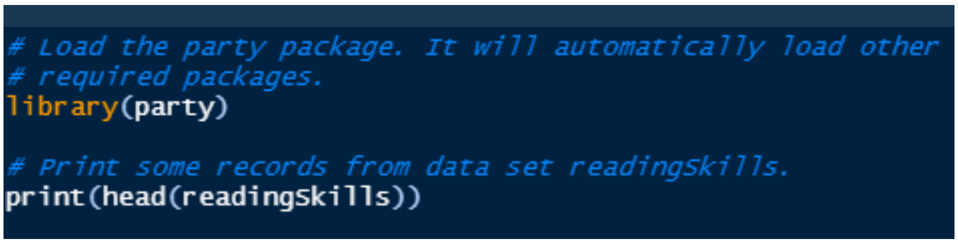


```
randomForest(formula, data)
```

Formula is the formula which describes the variables i.e. predictor and response.

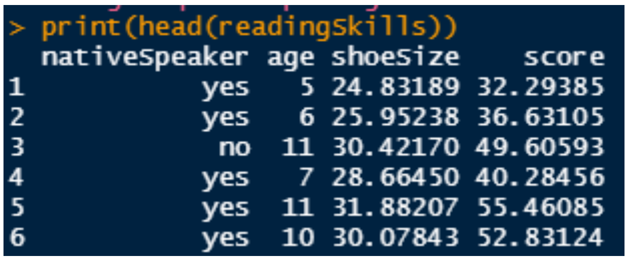
Data is the name of the dataset used.

Input Data: Use R's built-in dataset “readingSkills” to build a decision tree. If you know the age, shoe size, score (raw score on reading test), it represents the person's reading literacy score and also if the person is native speaker or not.



```
# Load the party package. It will automatically load other
# required packages.
library(party)

# Print some records from data set readingSkills.
print(head(readingSkills))
```

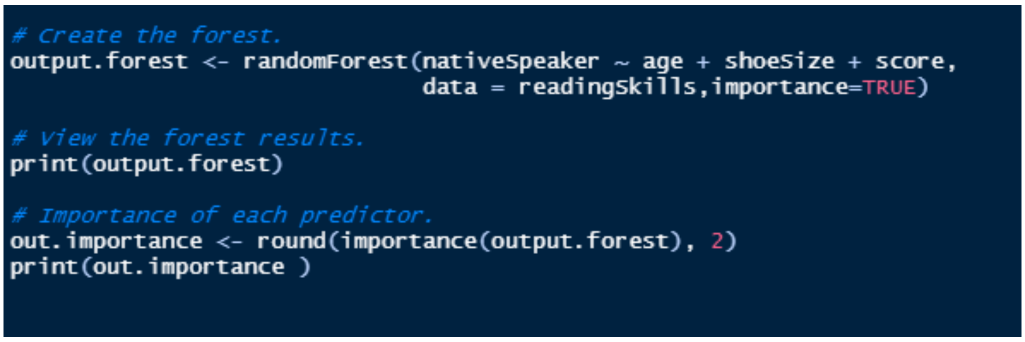
Output:



```
> print(head(readingSkills))
  nativeSpeaker age shoeSize    score
1           yes   5 24.83189 32.29385
2           yes   6 25.95238 36.63105
3            no  11 30.42170 49.60593
4           yes   7 28.66450 40.28456
5           yes  11 31.88207 55.46085
6           yes  10 30.07843 52.83124
```

Figure 16.5: Sample data for random forest

You can now create the random forest by applying the syntax given above and print the results



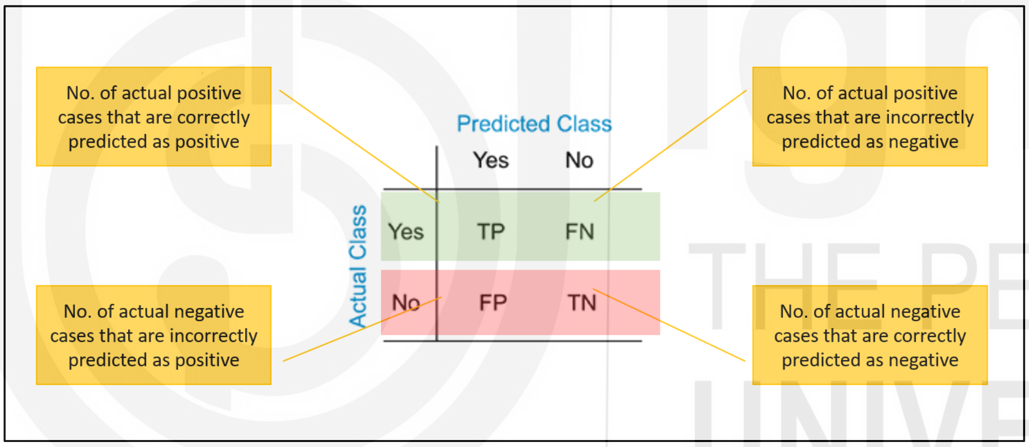
```
# Create the forest.
output.forest <- randomForest(nativeSpeaker ~ age + shoeSize + score,
                              data = readingSkills,importance=TRUE)

# View the forest results.
print(output.forest)

# Importance of each predictor.
out.importance <- round(importance(output.forest), 2)
print(out.importance )
```

Figure 16.6: Sequence of commands using R for random forest

The output of random forest is in the form of confusion matrix. Therefore, before showing the output, let us discuss about confusion matrix.

Confusion matrix:

A confusion matrix is a performance measurement for machine learning classification problems where output can be two or more classes.



No. of actual positive cases that are correctly predicted as positive
No. of actual positive cases that are incorrectly predicted as negative
Predicted Class
Yes
No
Actual Class
Yes
TP
FN
No
FP
TN
No. of actual negative cases that are incorrectly predicted as positive
No. of actual negative cases that are correctly predicted as negative

Figure 16.7: The Confusion Matrix

Confusion Matrix Example:

	Positive (Predicted)	Negative (Predicted)
Positive (Actual)	100	50
Negative (Actual)	150	9700

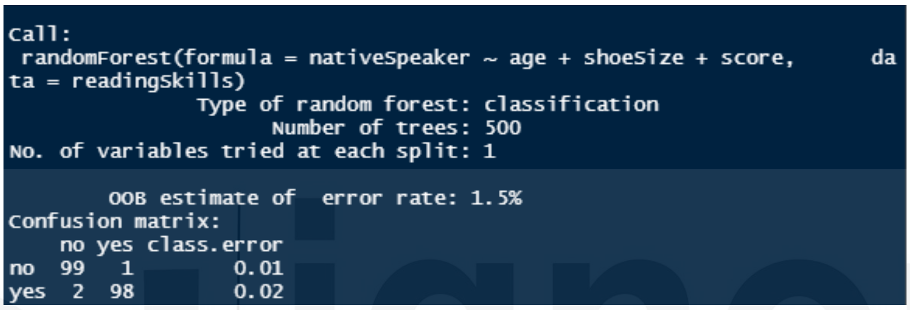
Figure 16.8: Example of Confusion Matrix

Please note the following for the confusion matrix of Figure 16.8

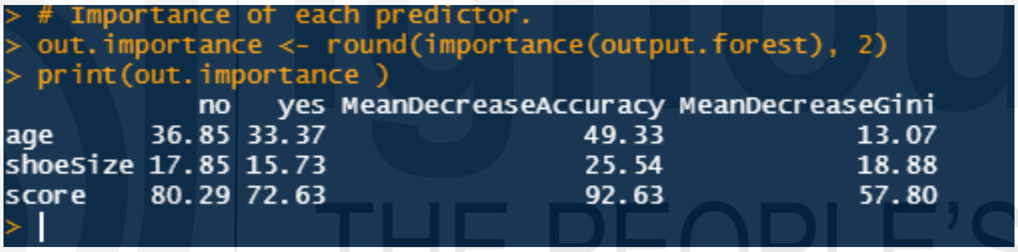
1. **Total number of observations** = (100 + 50 + 150 + 9700) = 10000

2. **Total number of positive cases (Actual)** = (100 + 50) = 150
3. **Total number of negative cases (Actual)** = (150 + 9700) = 9850
4. **TP = 100, i.e.**100 out of 150 positive cases are correctly predicted as positive.
5. **FN = 50, i.e**. 50 out of 150 positive cases are incorrectly predicted as negative.
6. **FP = 150, i.e.**150 out of 9850 negative cases are incorrectly predicted as positive.
7. **TN = 9700, i.e.**9700 out of 9850 negative cases are correctly predicted as negative**.**

Output:



```
Call:
 randomForest(formula = nativeSpeaker ~ age + shoeSize + score,      da
ta = readingSkills)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 1

        OOB estimate of  error rate: 1.5%
Confusion matrix:
    no yes class.error
no  99   1        0.01
yes  2  98        0.02
```



```
> # Importance of each predictor.
> out.importance <- round(importance(output.forest), 2)
> print(out.importance )
            no   yes MeanDecreaseAccuracy MeanDecreaseGini
age      36.85 33.37                49.33            13.07
shoeSize 17.85 15.73                25.54            18.88
score    80.29 72.63                92.63            57.80
> |
```

Figure 16.9: Output of Random Forest

Where, no is not a native speaker and yes is a native speaker. Mean Decrease Accuracy express how much accuracy a model loses excluding each variable and MeanDecreaseGini tells how each variable contributes to the homogeneity of the nodes.

From the Random Forest above, we can conclude that shoe size and score are important factors in determining if someone is native. As the values of MeanDecreaseGini is lower that means higher the purity and hence the 2 independent variables turns out to be important. Also, the model error is only 1%. This means that you can predict with 99% accuracy.

Check Your Progress 1

1. Can Random Forest Algorithm be used both for Continuous and Categorical Target Variables?

………………………………………………………………………………………

………………………………………………………………………………………

2.What is Out-of-Bag Error?

………………………………………………………………………………………

………………………………………………………………………………………

3. What does random refer to in 'Random Forest'?

………………………………………………………………………………………

………………………………………………………………………………………

16.4 CLASSIFICATION

The idea of a classification algorithm is very simple. Predict the target class by analysing the training dataset. Use the training dataset to get better boundary conditions that you can use to determine each target class. Once the constraints are determined, the next task is to predict the target class. This entire process is called classification.

The classification algorithm has some important points.

- **Classifier:** This is an algorithm that assigns input data to a specific category. Classification model. The classification model attempts to draw some conclusions from the input values given to the training. This inference predicts the class label / category of new data.
- **Characteristic**: This is an individually measurable property of the observed event.
- **Binary classification**: This is a classification task with two possible outcomes. For example, a gender classification with only two possible outcomes i.e. Men and women.
- **Multi-class classification:** This is a classification task where classification is done in three or more classes. Here is an example of a multiclass classification: A classifier can recognise a digit only as one of the digit classes say 0 or 1 or 2…or 9.
- **Multi-label classification:** This is a classification task where each sample is assigned a set of target labels. Here is an example of a multi-label classification: A news article that can be classified by a multi-label classifier as *people*, *places*, and *sports* at the same time.

In R, classification algorithms can be broadly divided into the following types.

Linear classifier

In machine learning, the main task of statistical classification is to use the properties of an object to find the class to which the object belongs. This task is solved by determining the classification based on the value of the linear combination of features. R has two linear classification algorithms:

- Logistic regression
- Naive Bayes classifier

Support vector machine

Support vector machines are supervised learning algorithms that analyse the data used for classification and regression analysis. In SVM, each data element is represented as a value for each attribute, that is, a point in n-dimensional space with a value at a particular coordinate. The least squares support vector machine is the most used classification algorithm in R.

Decision tree

The decision tree is a supervised learning algorithm used for classification and regression tasks. In R, the decision tree classifier is implemented using the R machine learning cullet package. The Random Forest algorithm is the most used decision tree algorithm in R.

16.5 CLUSTERING

Clustering in the R programming language is an unsupervised learning technique that divides a dataset into multiple groups and is called a cluster because of its similarities. After segmenting the data, multiple data clusters are generated. All objects in the cluster have common properties. Clustering is used in data mining and analysis to find similar datasets.

Clustering application in the R programming language

- **Marketing**: In R programming, clustering is useful for marketing. This helps identify market patterns and, therefore, find potential buyers. By identifying customer interests through clustering and displaying the same products of interest, you can increase your chances of buying a product.
- **Internet**: Users browse many websites based on their interests. Browsing history can be aggregated and clustered, and a user profile is generated based on the results of the clustering.
- **Games**: You can also use clustering algorithms to display games based on your interests.
- **Medicine**: In the medical field, every day there are new inventions of medicines and treatments. Occasionally, new species are discovered by researchers and scientists. Those categories can be easily found by using a clustering algorithm based on their similarity.

Clustering method

There are two types of clustering in R programming.

- **Hard clustering**: With this type of clustering, data points are assigned to only one cluster, whether they belong entirely to the cluster. The algorithm used for hard clustering is k-means clustering.
- **Soft clustering**: Soft clustering assigns the probabilities or possibilities of data points in a cluster rather than placing each data point in a cluster. All data points have a certain probability of being present in all clusters. The algorithm used for soft clustering is fuzzy clustering or soft k-means.

K-Means clustering in the R programming language

K-Means is an iterative hard clustering technique that uses an unsupervised learning algorithm. The total number of clusters is predefined by the user, and the data points are clustered based on the similarity of each data point. This algorithm also detects the center of gravity of the cluster.

Algorithm: Specifying the number of clusters (k): Let's look at an example of k = 2 and 5 data points. Randomly assign each data point to the cluster. Assume, the yellow and blue colors show two clusters with their respective random data points assigned. Calculate the centroid of a cluster: Remap each data point to the nearest cluster centroid. The blue data point is assigned to the yellow cluster because it is close to the center of gravity of the yellow cluster. Refactor the cluster centroid fuzzy clustering method or soft-k-means.

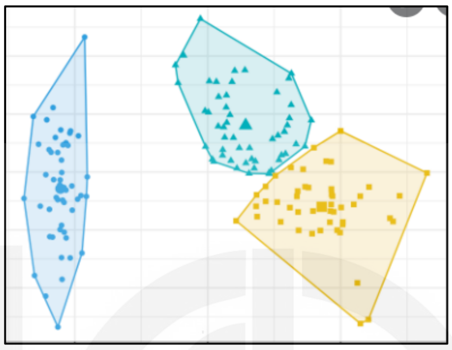
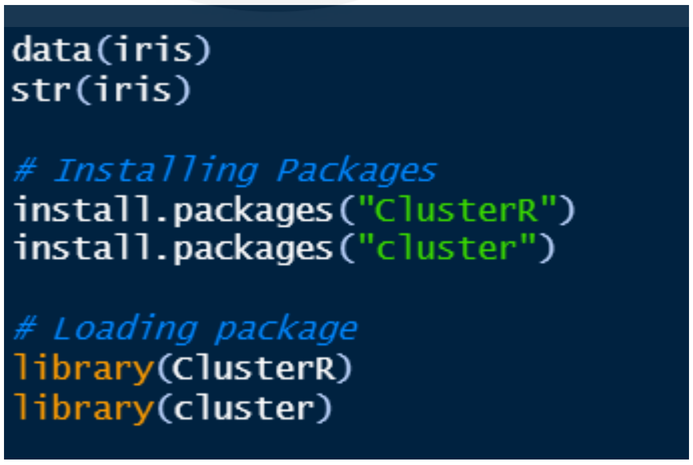


Figure 16.10: Clustering

Syntax: kmeans(x, centers, nstart)
where,

- x represents numeric matrix or data frame object
- centers represents the K value or distinct cluster centers
- nstart represents number of random sets to be chosen

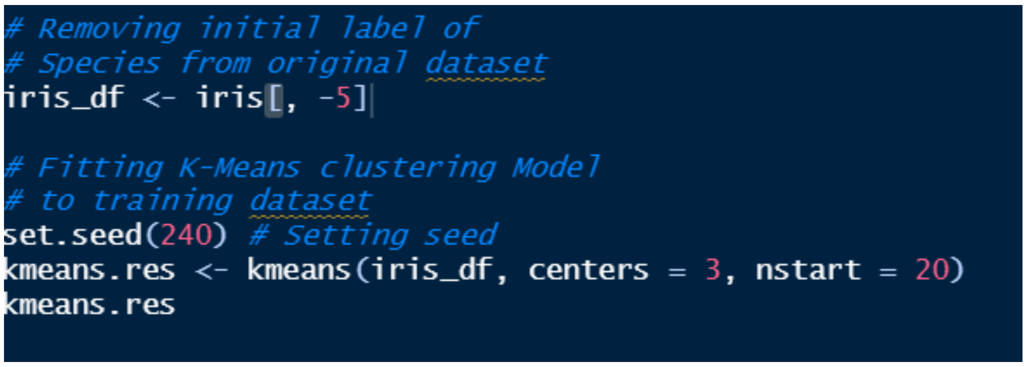
Input Data and loading the necessary packages in R. For the clustering, we are using iris data set. The dataset contains 3 classes each of around 50 instances and the class refers to a type of iris plant.



```
data(iris)
str(iris)

# Installing Packages
install.packages("ClusterR")
install.packages("cluster")

# Loading package
library(ClusterR)
library(cluster)
```

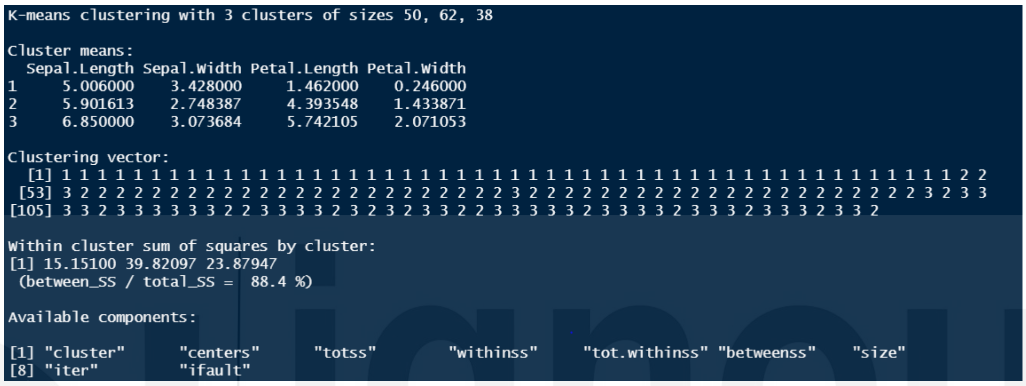
Fitting the K means clustering model

63



```
# Removing initial label of
# Species from original dataset
iris_df <- iris[, -5]

# Fitting K-Means clustering Model
# to training dataset
set.seed(240) # Setting seed
kmeans.res <- kmeans(iris_df, centers = 3, nstart = 20)
kmeans.res
```

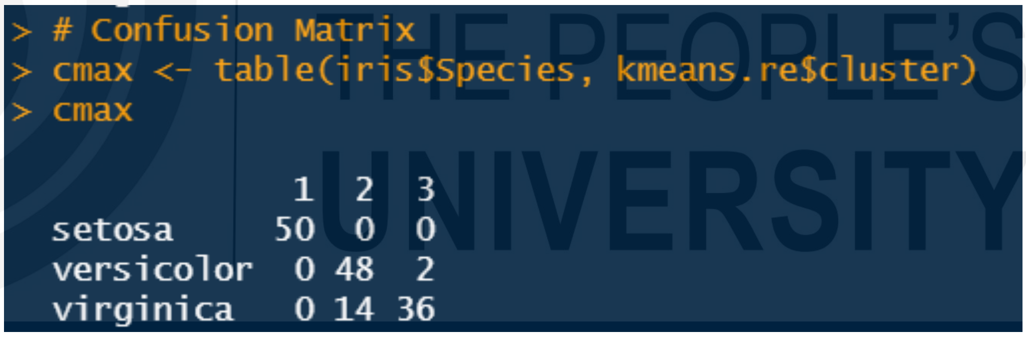
Result:



```
K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053

Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2
 [53] 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3
[105] 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"
[8] "iter"         "ifault"
```

The 3 clusters are made which are of 50, 62, and 38 sizes respectively. Within the cluster, the sum of squares is 88.4%.

Confusion Matrix:

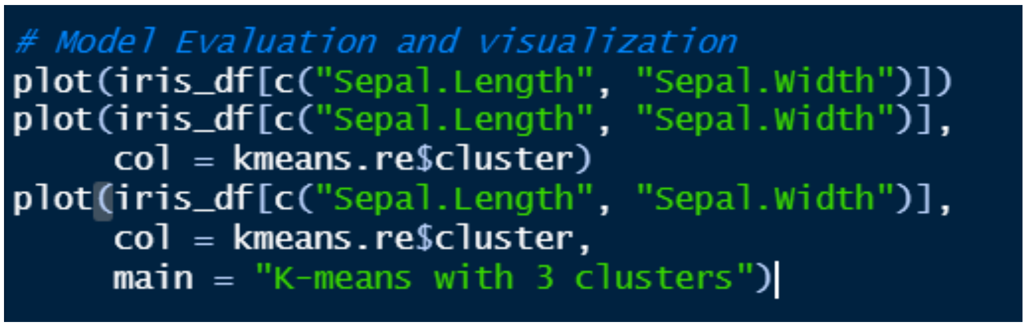


```
> # Confusion Matrix
> cmax <- table(iris$Species, kmeans.re$cluster)
> cmax

              1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0 14 36
```

Confusion matrix suggests that 50 setosa are correctly classified as setosa. Out of 62 versicolor, 14 are incorrectly classified as virginica and 48 correctly as versicolor. Out of 38 virginica, 2 are incorrectly classified as versicolor and 36 correctly classified as virginica.

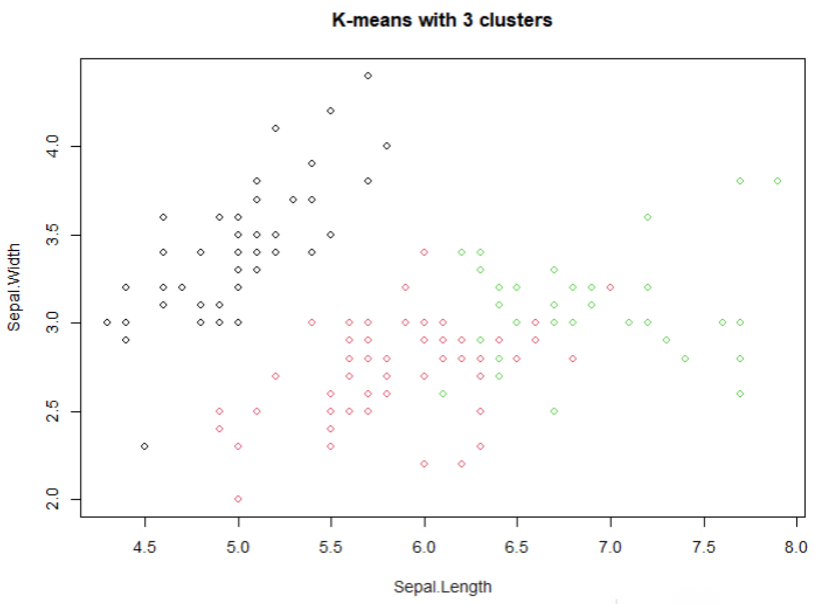
Model Evaluation and Visualization

Code:



```
# Model Evaluation and visualization
plot(iris_df[c("Sepal.Length", "Sepal.Width")])
plot(iris_df[c("Sepal.Length", "Sepal.Width")],
      col = kmeans.re$cluster)
plot(iris_df[c("Sepal.Length", "Sepal.Width")],
      col = kmeans.re$cluster,
      main = "K-means with 3 clusters")
```

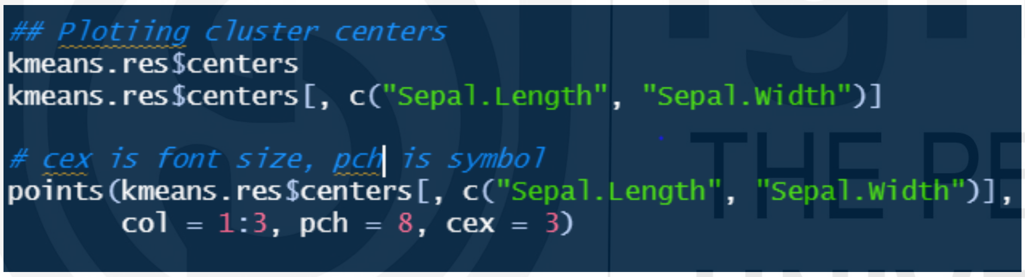
Output:



The plot shows 3 cluster plots with 3 different colors.

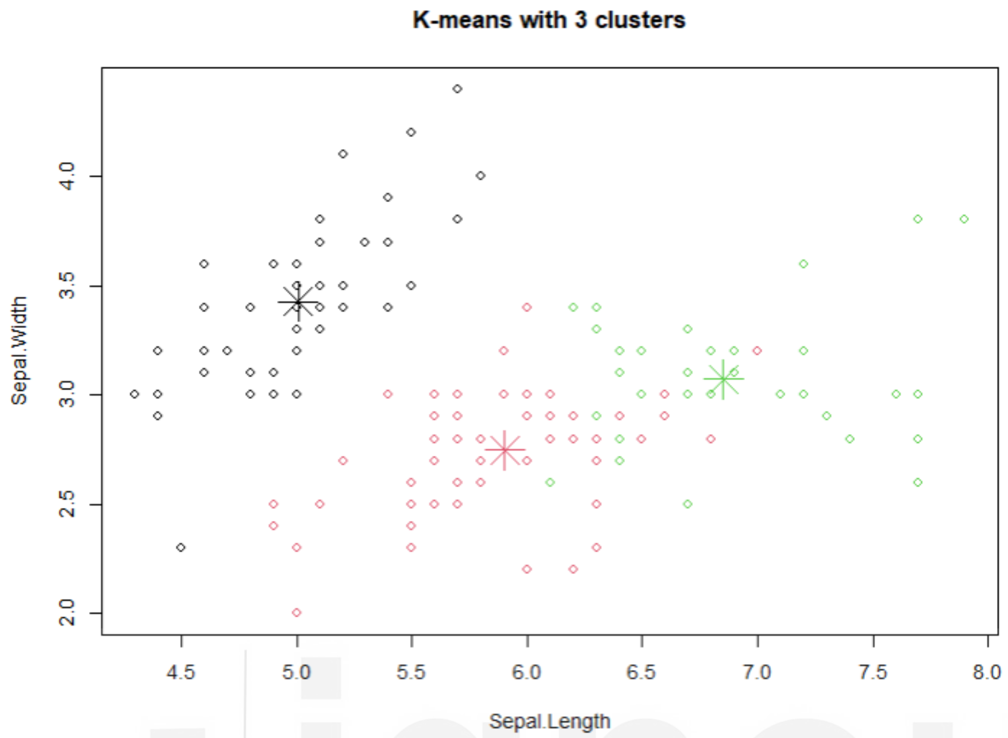
Plotting Cluster Centres

Code:



```
## Plotiing cluster centers
kmeans.res$centers
kmeans.res$centers[, c("Sepal.Length", "Sepal.Width")]

# cex is font size, pch is symbol
points(kmeans.res$centers[, c("Sepal.Length", "Sepal.Width")],
       col = 1:3, pch = 8, cex = 3)
```

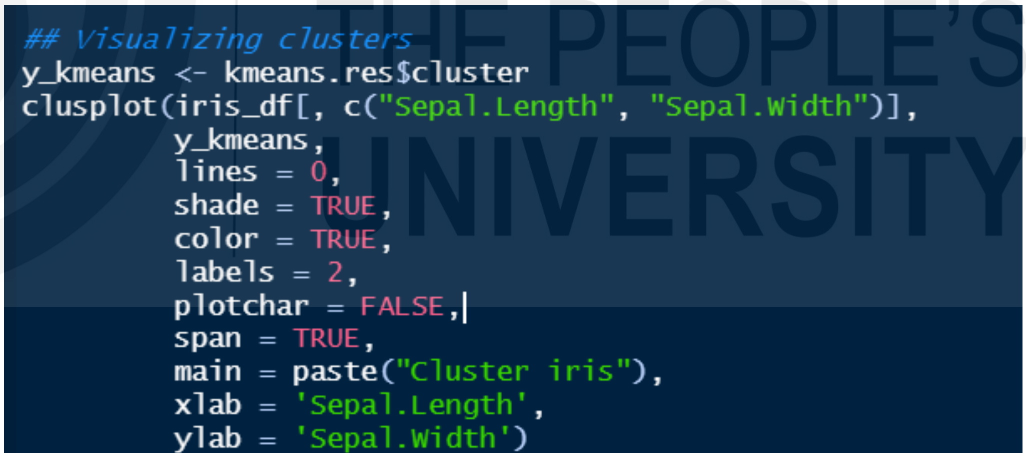
Output:



The plot shows the center of the clusters which are marked as cross sign with same color of the cluster.

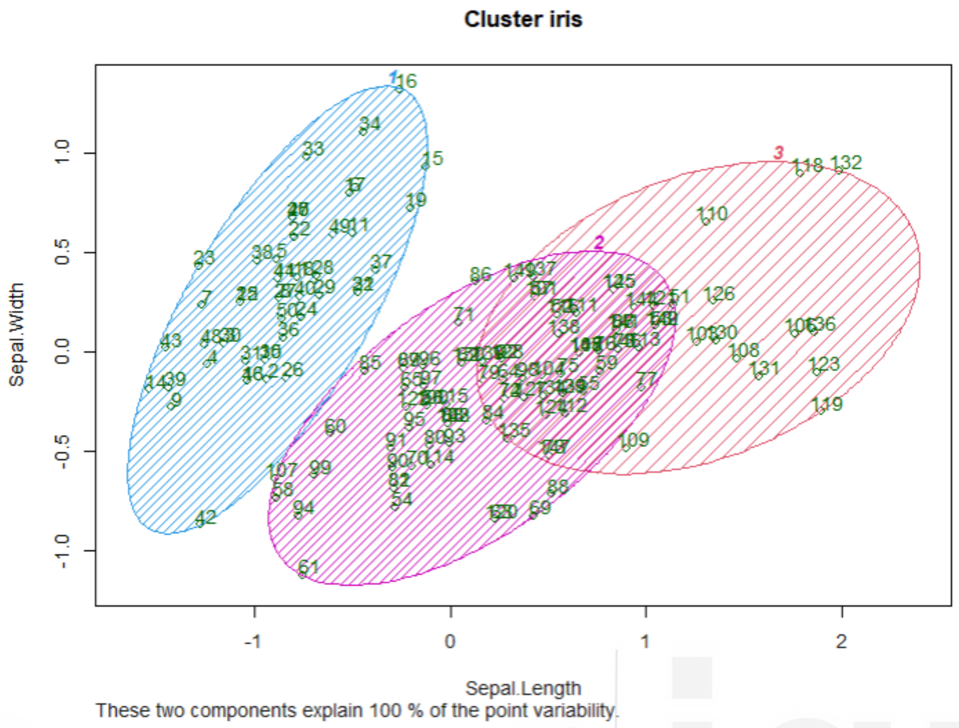
Visualizing Clusters

Code:



```
## Visualizing clusters
y_kmeans <- kmeans.res$cluster
clusplot(iris_df[, c("Sepal.Length", "Sepal.Width")],
         y_kmeans,
         lines = 0,
         shade = TRUE,
         color = TRUE,
         labels = 2,
         plotchar = FALSE,
         span = TRUE,
         main = paste("Cluster iris"),
         xlab = 'Sepal.Length',
         ylab = 'Sepal.Width')
```

Output:



The plot showing 3 clusters formed with varying sepal length and sepal width.

Check Your Progress 2

1. What is the difference between Classification and Clustering?

…………………………………………………………………………………

…………………………………………………………………………………

2. Can decision trees be used for performing clustering?

…………………………………………………………………………………

…………………………………………………………………………………

3. What is the minimum no. of variables/ features required to perform clustering?

…………………………………………………………………………………

…………………………………………………………………………………

16.6 ASSOCIATION RULES

Association Rule Mining in R Language is an Unsupervised Non-linear algorithm to discover how any item is associated with other. Frequent Mining shows which items appear together in a transaction. Major usage is in Retail, grocery stores, an online platform i.e. those having a large transactional database. The same way when any online social media or e-commerce websites know what you buy next using recommendations engines. The recommendations you get on item, while you check out the order is because of Association rule mining boarded on past user data. There are three common ways to measure association:

- Support

- Confidence
- Lift

Theory
In association rule mining, Support, Confidence, and Lift measure association.

[E1] Buy Product A => [E2] Buy Product B
Support (Rule) = P(E1 and E2) = Probability of Buying both the products A and B.
Confidence (Rule) = P(E2|E1) = Probability of buying the product B given that product A has already been bought.

Interpreting Support & Confidence of a Rule:

Computer => Antivirus software [support = 2%, confidence = 60%]
Computer: Antecedent**& Antivirus Software:** Consequence

Support: 2% of all the transaction under analysis show that computer and antivirus software are purchased together.
Confidence: 60% of the customers who purchased a computer also bought the software.

Lift: A measure of Association i.e. the occurrence of itemset A is independent of the occurrence of itemset B if
P(A and B) = P(A).P(B)

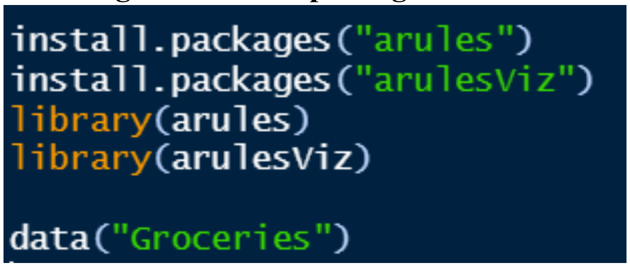
$$\frac{P(A \text{ and } B)}{P(A).P(B)} = 1$$

If <1, if buying A & buying B are negatively associated.
If =1, if buying A & buying B are not associated.
If >1, if buying A & buying B are positively associated.

Packages in R:
Installing Packages
install.packages("arules")
install.packages("arulesViz")

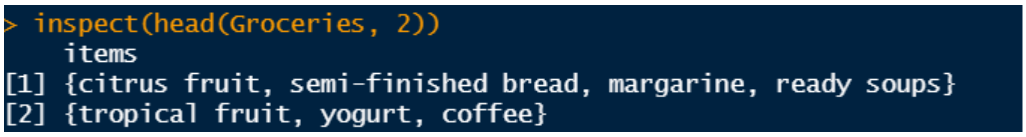
Syntax:
associa_rules = apriori(data = dataset, parameter = list(support = x,
confidence = y))

Installing the relevant packages and the dataset

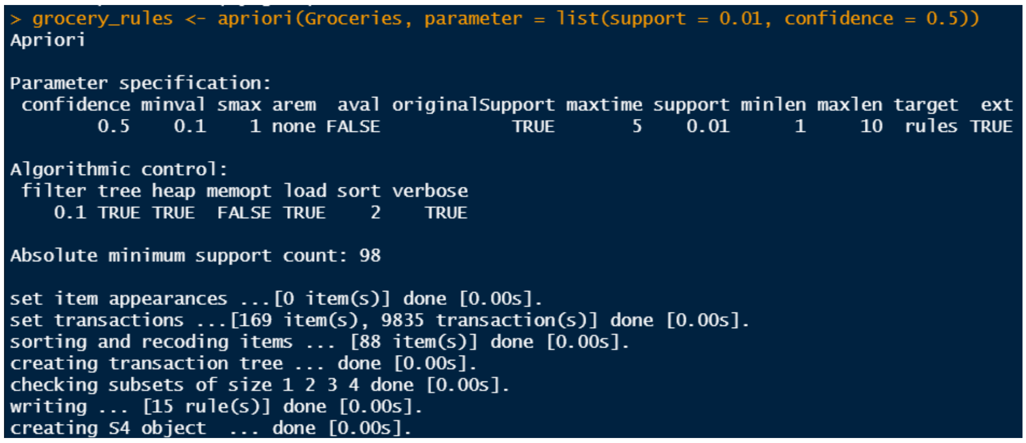


```
install.packages("arules")
install.packages("arulesViz")
library(arules)
library(arulesViz)

data("Groceries")
```

The first 2 transactions and the items involved in each transaction can be observed below.



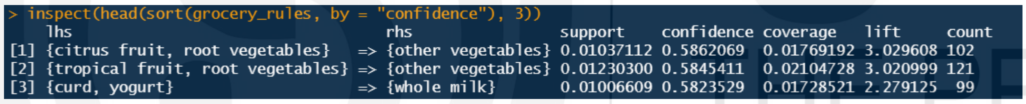
```
> inspect(head(Groceries, 2))
    items
[1] {citrus fruit, semi-finished bread, margarine, ready soups}
[2] {tropical fruit, yogurt, coffee}
```



```
> grocery_rules <- apriori(Groceries, parameter = list(support = 0.01, confidence = 0.5))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen maxlen target  ext
        0.5    0.1    1 none FALSE            TRUE       5    0.01      1     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 98

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [88 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 done [0.00s].
writing ... [15 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

The algorithm generated 15 rules with the given constraints.
Minval is the minimum value of support of an itemset which should be satisfied to be a part of a rule.
Smax: maximum support value.
Arem is additional rule evaluation parameter. We have fed Support and confidence as the constraints. There are several other rules using the arem parameter.

The top 3 rules sorted by confidence are shown below:



```
> inspect(head(sort(grocery_rules, by = "confidence"), 3))
    lhs                                  rhs                support    confidence coverage   lift     count
[1] {citrus fruit, root vegetables}   => {other vegetables} 0.01037112 0.5862069  0.01769192 3.029608 102
[2] {tropical fruit, root vegetables} => {other vegetables} 0.01230300 0.5845411  0.02104728 3.020999 121
[3] {curd, yogurt}                    => {whole milk}       0.01006609 0.5823529  0.01728521 2.279125  99
```

16.7 SUMMARY

This unit explores the concepts of advance data analysis and their application in R. It explains about the Decision Tree model and its application in R. It represents decisions and their results in a tree format. There are various types of decision trees that may fall under the 2 categories based on the target variables, i.e. categorical and continuous variable. Partitioning, pruning and the tree selection are the steps involved in the working of the decision tree. The other factors to consider when choosing a tree in R, are- entropy and information gain. The unit further explains in detail the concept of Random and its application in R. Random forests are a set of decision trees that are used in supervised learning algorithms for classification and regression analysis, but primarily for classification. It is a non-linear classification algorithm and takes samples from observations and random initial variables (columns) and attempts to build a model. In the subsequent sections, the unit explains the details of classification algorithm and its features and types. In R, the types of classification algorithms are- Linear classifier, Support vector machine and the decision tree. The linear classification algorithms can further be of 2 types- logistic regression and Naive Bayes classifier. It further explains the about Clustering and its application in R programming. Clustering in the R programming language is an unsupervised learning technique that divides a dataset into multiple groups and is called a cluster because of its similarities. It is used in data mining and analysis to find similar datasets ad has application in marketing, internet, gaming, medicine, etc. There are two types of clustering in R programming- namely, hard clustering and soft clustering. The unit also discusses K-Means clustering in the R programming language which is an iterative hard clustering technique that uses an unsupervised learning algorithm. The unit discusses the theory of the Association Rules and its application in R. It is an Unsupervised Non-linear algorithm to discover how any item is associated with other and has a major usage in Retail, grocery stores, an online platform i.e. those having a large transactional database. In association rule mining, support, confidence, and lift measure the association.

16.8 ANSWERS

Check Your Progress 1

1. Yes, Random Forest can be used for both continuous as well as categorical target (dependent) variables.
   A random forest i.e., the combination of decision trees, the classification model refers to the categorical dependent variable, and the regression model refers to the numeric or continuous dependent variable. But random forest is mainly used for Classification problems.

2. Out of Bag supports validation or test data. Random forests do not require a separate test dataset to validate the results. It is calculated internally during the execution of the algorithm in the following way: Because the forest is built on training data, each tree is tested on 1/3 (36.8%) of the samples that were not used to build that tree (similar to the validation dataset). This is known as the out-of-bag error estimation. This is simply an internal error estimate for the random forest you are building.

3. Random forest is one of the most popular and widely used machine learning algorithms in classification problems. It can also be used for regression problem statements, but it works mostly well with classification models.
   Improvements to the predictive model have become a deadly weapon for modern data scientists. The best part of the algorithm is that there are very few assumptions involved. Therefore, preparing the data is not too difficult and saves time.

Check Your Progress 2

1. **Classification** is taking data and putting it into **pre-defined categories** and in **Clustering** the set of categories, that you want to group the data into, **is not known** beforehand.

2.True, Decision trees can also be used to for clusters in the data, but clustering often generates natural clusters and is not dependent on any objective function.

3. At least a single variable is required to perform clustering analysis. Clustering analysis with a single variable can be visualized with the help of a histogram.

16.9 REFERENCES AND FURTHER READINGS

1. De Vries, A., & Meys, J. (2015). *R for Dummies*. John Wiley & Sons.
2. Peng, R. D. (2016). *R programming for data science* (pp. 86-181). Victoria, BC, Canada: Leanpub.
3. Schmuller, J. (2017). *Statistical Analysis with R For Dummies*. John Wiley & Sons.
4. Field, A., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage publications.
5. Lander, J. P. (2014). *R for everyone: Advanced analytics and graphics*. Pearson Education.
6. Lantz, B. (2019). *Machine learning with R: expert techniques for predictive modeling*. Packt publishing ltd.
7. Heumann, C., & Schomaker, M. (2016). *Introduction to statistics and data analysis*. Springer International Publishing Switzerland.
8. Davies, T. M. (2016). *The book of R: a first course in programming and statistics*. No Starch Press.
9. https://www.tutorialspoint.com/r/index.html
10. https://www.guru99.com/r-decision-trees.html
11. https://codingwithfun.com/p/r-language-random-forest-algorithm/
12. https://towardsdatascience.com/association-rule-mining-in-r-ddf2d044ae50
13. https://www.geeksforgeeks.org/k-means-clustering-in-r-programming/