Data Analytics in R

Module I: Introduction to R

Soumen Ghosh

Indian Institute of Information Technology Chittoor,
Sri City, Andhra Pradesh.

February 28, 2018

Basics of R

R, a dialect of S language, is developed as a general purpose language primarily meant for data analysis, statistical modeling, simulation and graphics. R project is widely used free software environment for statistical computing and graphics. The growth witnessed in usage of R in the last decade both from academia and industry has made R as the single most important tool for computational statistics, data visualization and data analysis.

Getting Help

help.start() # general help

```r
help("max")     # help about function max
?max            # same thing
apropos("max") # list all functions containing string max
```

```
## [1] "cummax"     "max"         "max.col"    "pmax"        "pmax.i
## [7] "varimax"    "which.max"
```

```r
example("max") # show an example of function max
```

```
##
## max> require(stats); require(graphics)
##
## max>  min(5:1, pi) #-> one number
## [1] 1
```

The Workspace I

```
getwd() # print the current working directory - cwd
```

```
## [1] "/home/soumen/0_HDD/ML_Training/pptMLinR/module1"
```

```
ls()    # list the objects in the current workspace
```

```
## [1] "cH"    "cut01" "D"     "n0"    "x"
```

setwd(mydirectory) # change to mydirectory

setwd(“c:/docs/mydir”) # note / instead of \ in windows

setwd(“/usr/rob/mydir”) # on linux

The Workspace II

Display last 25 commands

history()

Display all previous commands

history(max.show=Inf)

Save your command history

savehistory(file="myfile") # default is ".Rhistory"

Recall your command history

loadhistory(file="myfile") # default is ".Rhistory"

Save and Load the Workspace

Save the workspace to the file .RData in the cwd

```
save.image()
```

Save specific objects to a file

If you don’t specify the path, the cwd is assumed

```
save(object list,file=“myfile.RData”)
```

Load a workspace into the current session

If you don’t specify the path, the cwd is assumed

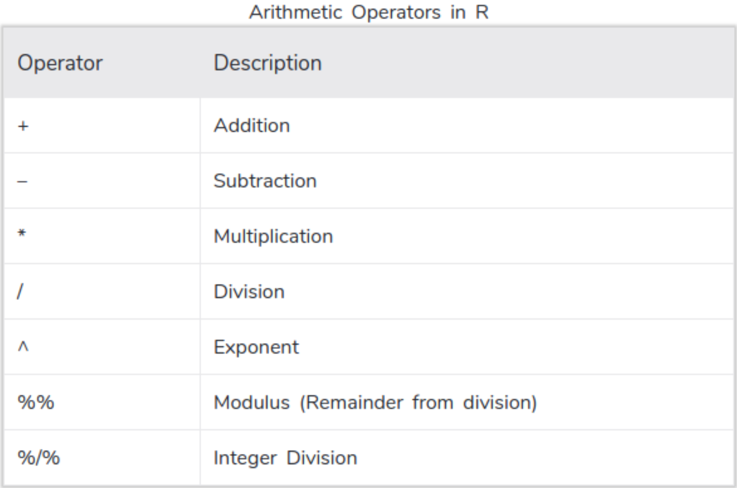
```
load(“myfile.RData”)
```

```
q() # quit R. You will be prompted to save the workspace.
```

Operators in R

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Arithmetic Operator

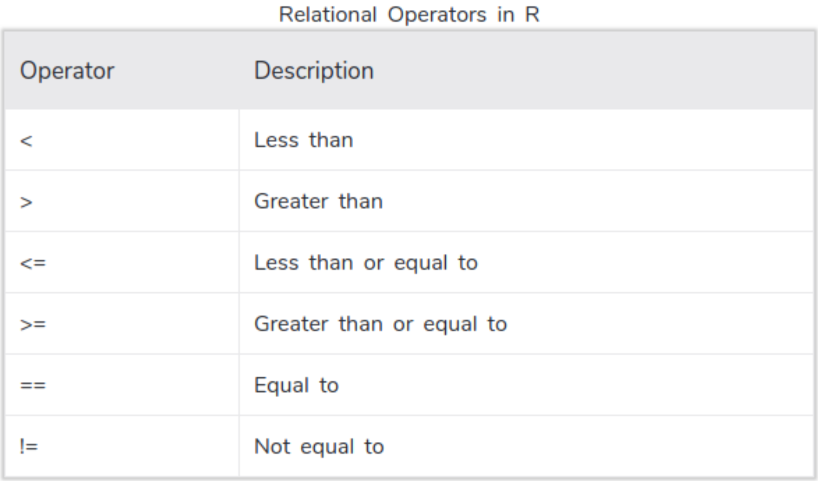


Arithmetic Operators in R

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

Figure 1:

Relational Operators

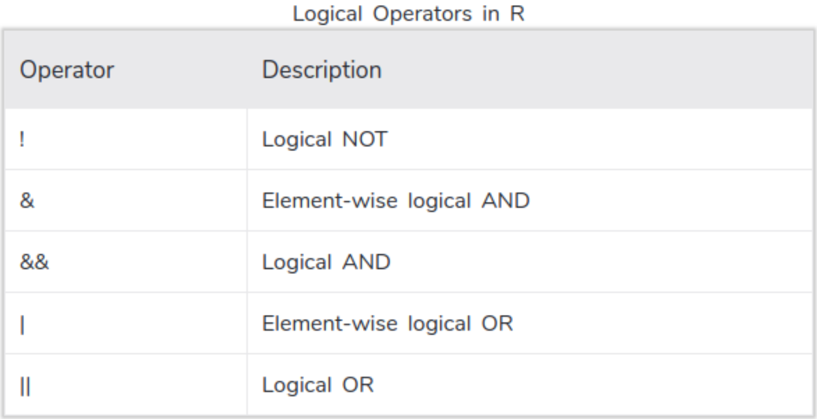


Relational Operators in R

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Figure 2:

Logical Operators

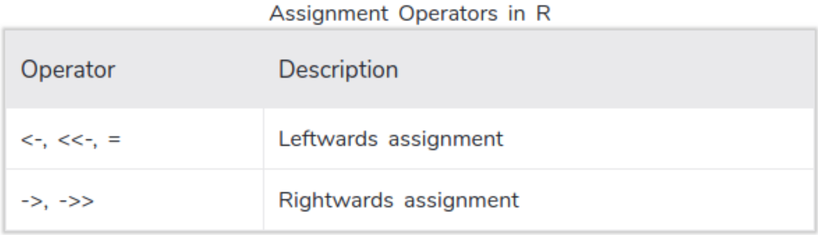


Logical Operators in R

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

Figure 3:

Assignment Operators



Assignment Operators in R

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

Figure 4:

Variable Types

- Logical
- Integer
- Numeric
- Character
- Complex

Logical

```r
x <- TRUE
print(x)
```

```
## [1] TRUE
```

```r
class(x)
```

```
## [1] "logical"
```

Integer

```r
x <- 10L
print(x)
```

```
## [1] 10
```

```r
class(x)
```

```
## [1] "integer"
```

Numeric

```r
x <- 10.59
print(x)
```

```
## [1] 10.59
```

```r
class(x)
```

```
## [1] "numeric"
```

Character

```
x <- 'string'
print(x)
```

```
## [1] "string"
```

```
class(x)
```

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

Complex

```
x <- 2 + 3i
print(x)
```

```
## [1] 2+3i
```

```
class(x)
```

```
## [1] "complex"
```

Getting Started with R I

```r
x <- sample(10)
print(x)
```

```
##  [1]  9  8 10  1  7  6  4  5  3  2
```

```r
length(x)
```

```
## [1] 10
```

```r
sum(x)
```

```
## [1] 55
```

```r
mean(x)
```

```
## [1] 5.5
```

Getting Started with R II

```r
print(x)
```

```
##  [1]  9  8 10  1  7  6  4  5  3  2
```

```r
unique(x)
```

```
##  [1]  9  8 10  1  7  6  4  5  3  2
```

```r
sort(x)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```r
rev(x)
```

```
##  [1]  2  3  5  4  6  7  1 10  8  9
```

Getting Started with R III

```
print(x)
```

```
##  [1]  9  8 10  1  7  6  4  5  3  2
```

```
rev(sort(x))
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```

```
max(x)
```

```
## [1] 10
```

```
min(x)
```

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

Getting Started with R IV

```r
print(x)
```

```
##  [1]  9  8 10  1  7  6  4  5  3  2
```

```r
which.max(x)
```

```
## [1] 3
```

```r
which.min(x)
```

```
## [1] 4
```

```r
which(x==5)
```

```
## [1] 8
```

Getting Started with R V

```
range(x)
```

```
## [1]  1 10
```

```
print(pi)
```

```
## [1] 3.141593
```

```
round(pi, 3)
```

```
## [1] 3.142
```

Getting Started with R VI

```
letters[3]
```

```
## [1] "c"
```

```
letters[3:6]
```

```
## [1] "c" "d" "e" "f"
```

```
LETTERS[sample(3)]
```

```
## [1] "B" "C" "A"
```

```
LETTERS[sample(10, 3)]
```

```
## [1] "I" "J" "G"
```

Useful Function

length(object) # number of elements or components

str(object) # structure of an object

class(object) # class or type of an object

names(object) # names

c(object,object,. . .) # combine objects into a vector

cbind(object, object, . . .) # combine objects as columns

rbind(object, object, . . .) # combine objects as rows

object # prints the object

ls() # list current objects

rm(object) # delete an object

newobject <- edit(object) # edit copy and save as newobject

fix(object) # edit in place

Data Types and Objects in R

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

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Strings I

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.

```r
a <- ’Start and end with single quote’
print(a)
```

```
## [1] "Start and end with single quote"
```

```r
b <- "Start and end with double quotes"
print(b)
```

```
## [1] "Start and end with double quotes"
```

```r
c <- "single quote ’ in between double quotes"
print(c)
```

Strings II

Invalid Strings e <- 'Mixed quotes"

print(e)

f <- ‘Single quote’ inside single quote’

print(f)

g <- “Double quotes” inside double quotes"

print(g)

String Manipulation

```r
a <- "Hello"
b <- ’How’
c <- "are you? "

print(paste(a,b,c))
```

```
## [1] "Hello How are you? "
```

```r
print(paste0(a,b,c))
```

```
## [1] "HelloHoware you? "
```

```r
print(paste(a,b,c, sep = "-"))
```

```
## [1] "Hello-How-are you? "
```

Formatting numbers & strings

```r
result <- format(23.47, nsmall = 5)
print(result)
```

```
## [1] "23.47000"
```

```r
result <- format(6)
print(result)
```

```
## [1] "6"
```

```r
result <- format(c(6, 13.14521), scientific = TRUE)
print(result)
```

```
## [1] "6.000000e+00" "1.314521e+01"
```

Formatting numbers & strings

```r
result <- format(13.7, width = 6)
print(result)
```

```
## [1] "  13.7"
```

```r
result <- format("Hello", width = 8, justify = "l")
print(result)
```

```
## [1] "Hello   "
```

```r
result <- format("Hello", width = 8, justify = "c")
print(result)
```

```
## [1] " Hello  "
```

```r
x <- "This is a Sample Text"
result <- nchar(x)
print(result)
```

```
## [1] 21
```

```r
toupper(x)
```

```
## [1] "THIS IS A SAMPLE TEXT"
```

```r
tolower(x)
```

```
## [1] "this is a sample text"
```

```r
substring(x,3,9)
```

```
## [1] "is is a"
```

Vector

```r
print("abc");
```

```
## [1] "abc"
```

```r
print(12.5)
```

```
## [1] 12.5
```

```r
print(63L)
```

```
## [1] 63
```

Multiple Elements Vector

```r
v <- 5:13
print(v)
```

```
## [1]  5  6  7  8  9 10 11 12 13
```

```r
v <- 6.6:12.6
print(v)
```

```
## [1]  6.6  7.6  8.6  9.6 10.6 11.6 12.6
```

```r
v <- 3.8:11.4
print(v)
```

```
## [1]  3.8  4.8  5.8  6.8  7.8  8.8  9.8 10.8
```

Multiple Elements Vector

Using sequence (Seq.) operator

```
print(seq(5, 9, by = 0.4))
```

```
##  [1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Using the c() function

```
s <- c(11, 33, 55, 99)
print(s)
```

```
## [1] 11 33 55 99
```

```
s1 <- c('apple','red',5,TRUE)
print(s1)
```

```
## [1] "apple" "red"   "5"     "TRUE"
```

Accessing Vector Elements

```
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(2,3,6)]
print(u)
```

```
## [1] "Mon" "Tue" "Fri"
```

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

```
## [1] "Sun" "Fri"
```

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

```
## [1] "Sun" "Tue" "Wed" "Fri" "Sat"
```

Vector Manipulation

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)

add.result <- v1+v2
print(add.result)
```

```
## [1]  7 19  4 13  1 13
```

```
sub.result <- v1-v2
print(sub.result)
```

```
## [1] -1 -3  4 -3 -1  9
```

```
multi.result <- v1*v2
print(multi.result)
```

Vector arithmetic operation

```r
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11,0,8,1,2)

multi.result <- v1*v2
print(multi.result)
```

```
## [1] 12 88  0 40  0 22
```

```r
divi.result <- v1/v2
print(divi.result)
```

```
## [1] 0.7500000 0.7272727       Inf 0.6250000 0.0000000 5.500
```

Vector element recycling

```
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)

add.result <- v1+v2
print(add.result)
```

```
## [1]  7 19  8 16  4 22
```

```
sub.result <- v1-v2
print(sub.result)
```

```
## [1] -1 -3  0 -6 -4  0
```

Sorting numerical vectors

```
v <- c(3,8,4,5,0,11, -9, 304)

sort.result <- sort(v)
print(sort.result)
```

```
## [1]  -9   0   3   4   5   8  11 304
```

```
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

```
## [1] 304  11   8   5   4   3   0  -9
```

Sorting character vectors

```r
v <- c("Red","Blue","yellow","violet")
sort.result <- sort(v)
print(sort.result)
```

```
## [1] "Blue"   "Red"    "violet" "yellow"
```

```r
revsort.result <- sort(v, decreasing = TRUE)
print(revsort.result)
```

```
## [1] "yellow" "violet" "Red"    "Blue"
```

Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types.

Syntax *matrix(data, nrow, ncol, byrow, dimnames)*

Following is the description of the parameters used

- data is the input vector which becomes the data elements of the matrix.
- nrow is the number of rows to be created.
- ncol is the number of columns to be created.
- byrow is a logical clue. If TRUE then the input vector elements are arranged by row.
- dimname is the names assigned to the rows and columns.

Createaing matrix I

```
M <- matrix(c(3:11), nrow = 3, byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    4    5
## [2,]    6    7    8
## [3,]    9   10   11
```

```
N <- matrix(c(3:11), nrow = 3, byrow = FALSE)
print(N)
```

```
##      [,1] [,2] [,3]
## [1,]    3    6    9
## [2,]    4    7   10
## [3,]    5    8   11
```

Createaing matrix II

```
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 Elements of a Matrix

```
print(P[1,3])
```

```
## [1] 5
```

```
print(P[4,2])
```

```
## [1] 13
```

```
print(P[2,])
```

```
## col1 col2 col3
##    6    7    8
```

```
print(P[,3])
```

```
## row1 row2 row3 row4
```

Matrix Computations I

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

```
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    9    4    6
```

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

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

Matrix Operations I

```
result <- matrix1 + matrix2
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]    8   -1    5
## [2,]   11   13   10
```

```
result <- matrix1 - matrix2
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]   -2   -1   -1
## [2,]    7   -5    2
```

Matrix Operations II

```
result <- matrix1 * matrix2
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]   15    0    6
## [2,]   18   36   24
```

```
result <- matrix1 / matrix2
print(result)
```

```
##      [,1]      [,2]      [,3]
## [1,]  0.6      -Inf 0.6666667
## [2,]  4.5 0.4444444 1.5000000
```

Matrix Operations III

```r
matrix3 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 3, ncol = 2)
print(matrix3)
```

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

```r
result <- matrix1 %*% matrix3
print(result)
```

```
##      [,1] [,2]
## [1,]   13   32
## [2,]   53  117
```

Matrix Operations IV

```
result <- t(matrix3)
print(result)
```

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

```
matrix4 <- rbind(result, c(3,7,2))
print(matrix4)
```

```
##      [,1] [,2] [,3]
## [1,]    5    2    0
## [2,]    9    3    4
## [3,]    3    7    2
```

Matrix Operations V

```r
#Inverse of a square matrix.
result <- solve(matrix4)
print(result)
```

```
##             [,1]        [,2]        [,3]
## [1,]  0.18032787  0.03278689 -0.06557377
## [2,]  0.04918033 -0.08196721  0.16393443
## [3,] -0.44262295  0.23770492  0.02459016
```

```r
d <- det(matrix4)
print(d)
```

```
## [1] -122
```

Common Matrices

Unit Matrix

```
U <- matrix(1,2,3)
print(U)
```

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

Zero Matrix

```
Z <- matrix(0,2,2)
print(Z)
```

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    0
```

Diagonal Matrix I

```
S <- matrix(c(2,3,-2,1,2,2,4,2,3),3,3)
print(S)
```

```
##      [,1] [,2] [,3]
## [1,]    2    1    4
## [2,]    3    2    2
## [3,]   -2    2    3
```

```
D <- diag(S)
print(D)
```

```
## [1] 2 2 3
```

Diagonal Matrix II

```
D <- diag(diag(S))
print(D)
```

```
##      [,1] [,2] [,3]
## [1,]    2    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

```
#Identity Matrix
I <- diag(c(1,1,1))
print(I)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    1    0
## [3,]    0    0    1
```

Row and Column Sum

```
a <- matrix(1:9, 3)
print(a)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
rowSums(a)
```

```
## [1] 12 15 18
```

```
colSums(a)
```

```
## [1]  6 15 24
```

Row and Column Sum

```
a <- matrix(1:9, 3)
print(a)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
apply(a, 1, sum)
```

```
## [1] 12 15 18
```

```
apply(a, 2, sum)
```

```
## [1]  6 15 24
```

Arrays

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

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

Creating Array I

```
vector1 <- c(5,9,3,10,11,12,13,14,15)
result <- array(vector1, 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
```

Creating Array II

```
vector1 <- c(5,9,3,10,11,12,13,14,15,16,17,18)
result <- array(vector1, dim = c(2,3,2))
print(result)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5    3   11
## [2,]    9   10   12
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]   13   15   17
## [2,]   14   16   18
```

Creating Array III

```r
column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2")
matrix.names <- c("Matrix1","Matrix2")
result <- array(vector1,dim = c(2,3,2),
          dimnames = list(row.names,column.names,matrix.names)
print(result)
```

```
## , , Matrix1
##
##      COL1 COL2 COL3
## ROW1    5    3   11
## ROW2    9   10   12
##
## , , Matrix2
##
##      COL1 COL2 COL3
```

Accessing Array Elements

```
print(result[2,,2])
```

```
## COL1 COL2 COL3
##   14   16   18
```

```
print(result[1,3,1])
```

```
## [1] 11
```

```
print(result[,,2])
```

```
##      COL1 COL2 COL3
## ROW1   13   15   17
## ROW2   14   16   18
```

Manipulating Array Elements

```
result[1,2,1] <- 111
print(result)
```

```
## , , Matrix1
##
##      COL1 COL2 COL3
## ROW1    5  111   11
## ROW2    9   10   12
##
## , , Matrix2
##
##      COL1 COL2 COL3
## ROW1   13   15   17
## ROW2   14   16   18
```

Manipulating Array Elements I

```r
matrix1 <- result[,,1]
matrix2 <- result[,,2]
result1 <- matrix1 + matrix2
print(result1)
```

```
##      COL1 COL2 COL3
## ROW1   18  126   28
## ROW2   23   26   30
```

Manipulating Array Elements II

```r
result2 <- apply(result, 2, sum)
print(result2)
```

```
## COL1 COL2 COL3
##   41  152   58
```

```r
result3 <- apply(result1, 1, sum)
print(result3)
```

```
## ROW1 ROW2
##  172   79
```

List

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

Creating List

```
list_data <- list("Red", 124, c(21,32,11), TRUE, 51.23)
print(list_data)
```

```
## [[1]]
## [1] "Red"
##
## [[2]]
## [1] 124
##
## [[3]]
## [1] 21 32 11
##
## [[4]]
## [1] TRUE
##
## [[5]]
```

Naming List Eliments

```
names(list_data) <- c("Character", "Integer", "Vector", "Logic

print(list_data)
```

```
## $Character
## [1] "Red"
##
## $Integer
## [1] 124
##
## $Vector
## [1] 21 32 11
##
## $Logical
## [1] TRUE
##
```

Accessing List Elements

```r
print(list_data$Character)
```

```
## [1] "Red"
```

```r
print(list_data[1])
```

```
## $Character
## [1] "Red"
```

```r
print(list_data$Vector)
```

```
## [1] 21 32 11
```

```r
print(list_data[3])
```

```
## $Vector
```

Manipulating List Elements

```
list_data[4] <- "New element"
print(list_data[4])
```

```
## $Logical
## [1] "New element"
```

```
list_data[2] <- NULL

print(list_data[3])
```

```
## $Logical
## [1] "New element"
```

Manipulating List Elements II

```
list_data[3] <- "updated element"
print(list_data)
```

```
## $Character
## [1] "Red"
##
## $Vector
## [1] 21 32 11
##
## $Logical
## [1] "updated element"
##
## $Double
## [1] 51.23
```

Merging Lists I

You can merge many lists into one list by placing all the lists inside one list() function.

```
list1 <- list(1,2)
list2 <- list("Sun","Mon","Tue")
merged.list <- c(list1,list2)
print(merged.list)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] "Sun"
##
```

Merging Lists II

```
list1 <- list(1,2)
list2 <- list("Sun","Mon","Tue")
nested.list <- list(list1,list2)
print(nested.list)
```

```
## [[1]]
## [[1]][[1]]
## [1] 1
##
## [[1]][[2]]
## [1] 2
##
##
## [[2]]
## [[2]][[1]]
## [1] "Sun"
```

Converting List to Vector I

```
list1 <- list(1,2,3)
print(list1)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```
v1 <- unlist(list1)
print(class(v1))
```

```
## [1] "numeric"
```

Converting List to Vector II

```r
list2 <- list(4:6)
print(list2)
```

```
## [[1]]
## [1] 4 5 6
```

```r
v2 <- unlist(list2)
result <- v1 + v2
print(result)
```

```
## [1] 5 7 9
```

Data Frames

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

Following are the characteristics of a data frame:

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

Create Data Frame

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

```
##   emp_id emp_name salary start_date
## 1      1     Rick 623.30 2012-01-01
## 2      2      Dan 515.20 2013-09-23
## 3      3 Michelle 611.00 2014-11-15
## 4      4     Ryan 729.00 2014-05-11
## 5      5     Gary 843.25 2015-03-27
```

Structure of Data Frame

```
class(emp.data)
```

```
## [1] "data.frame"
```

```
mode(emp.data)
```

```
## [1] "list"
```

```
str(emp.data)
```

```
## ’data.frame’:    5 obs. of  4 variables:
##  $ emp_id    : int  1 2 3 4 5
##  $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
##  $ salary    : num  623 515 611 729 843
##  $ start_date: Date, format: "2012-01-01" "2013-09-23" ...
```

Summary of Data Frame

```r
summary(emp.data)
```

```
##      emp_id    emp_name             salary        start_dat
##  Min.   :1   Length:5           Min.   :515.2   Min.   :201
##  1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:201
##  Median :3   Mode  :character   Median :623.3   Median :201
##  Mean   :3                      Mean   :664.4   Mean   :201
##  3rd Qu.:4                      3rd Qu.:729.0   3rd Qu.:201
##  Max.   :5                      Max.   :843.2   Max.   :201
```

Extract Data from Data Frame I

Extract Specific columns I

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

```
##   emp.data.emp_name emp.data.salary
## 1              Rick          623.30
## 2               Dan          515.20
## 3          Michelle          611.00
## 4              Ryan          729.00
## 5              Gary          843.25
```

```
class(result)
```

```
## [1] "data.frame"
```

Extract Data from Data Frame I

Extract Specific columns II

```r
result <- emp.data[,2:3]
print(result)
```

```
##   emp_name salary
## 1     Rick 623.30
## 2      Dan 515.20
## 3 Michelle 611.00
## 4     Ryan 729.00
## 5     Gary 843.25
```

```r
class(result)
```

```
## [1] "data.frame"
```

Extract Data from Data Frame II

Extract Specific rows

```
result <- emp.data[1:2,]
print(result)
```

```
##   emp_id emp_name salary start_date
## 1      1     Rick  623.3 2012-01-01
## 2      2      Dan  515.2 2013-09-23
```

```
class(result)
```

```
## [1] "data.frame"
```

Extract Data from Data Frame III

Extract Specific rows and columns togather I

```
result <- emp.data[1:2, 2:3]
print(result)
```

```
##   emp_name salary
## 1     Rick  623.3
## 2      Dan  515.2
```

```
class(result)
```

```
## [1] "data.frame"
```

Extract Data from Data Frame III

Extract Specific rows and columns togather II

```
result <- emp.data[c(1,3), c(2,4)]
print(result)
```

```
##   emp_name start_date
## 1     Rick 2012-01-01
## 3 Michelle 2014-11-15
```

```
class(result)
```

```
## [1] "data.frame"
```

Expand Data Frame

Add Column

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

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

Expand Data Frame

Add Row I

```
emp.newdata <-  data.frame(
   emp_id = c (6:8),
   emp_name = c("Rasmi","Pranab","Tusar"),
   salary = c(578.0,722.5,632.8),
   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-1
   dept = c("IT","Operations","Fianance"),
   stringsAsFactors = FALSE
)
print(emp.newdata)
```

```
##   emp_id emp_name salary start_date       dept
## 1      6    Rasmi  578.0 2013-05-21         IT
## 2      7   Pranab  722.5 2013-07-30 Operations
## 3      8    Tusar  632.8 2014-06-17   Fianance
```

Expand Data Frame

Add Row II

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

```
##   emp_id emp_name salary start_date       dept
## 1      1     Rick 623.30 2012-01-01         IT
## 2      2      Dan 515.20 2013-09-23 Operations
## 3      3 Michelle 611.00 2014-11-15         IT
## 4      4     Ryan 729.00 2014-05-11         HR
## 5      5     Gary 843.25 2015-03-27    Finance
## 6      6    Rasmi 578.00 2013-05-21         IT
## 7      7   Pranab 722.50 2013-07-30 Operations
## 8      8    Tusar 632.80 2014-06-17   Fianance
```

Factors

Factors are the data objects which are used to categorize the data and store it as levels. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male", "Female" and True, False etc. They are useful in data analysis for statistical modeling. An ordered factor is used to represent an ordinal variable.

Factors are created using the factor() function by taking a vector as input.

Create a Factor

```
data <- c("East","West","East","North","North","East","West",'
print(data)
```

```
##  [1] "East"  "West"  "East"  "North" "North" "East"  "West'
##  [9] "West"  "East"  "North"
```

```
print(is.factor(data))
```

```
## [1] FALSE
```

```
factor_data <- factor(data)
print(factor_data)
```

```
##  [1] East  West  East  North North East  West  West  West
## Levels: East North West
```

Create a Factor

```
print(is.factor(factor_data))
```

```
## [1] TRUE
```

```
str(factor_data)
```

```
##  Factor w/ 3 levels "East","North",..: 1 3 1 2 2 1 3 3 3 1
```

Factors in Data Frame I

```
height <- c(132,151,162,139,166,147,122)
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","m

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

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

Factors in Data Frame II

```
print(is.factor(input_data$gender))
```

```
## [1] TRUE
```

```
print(input_data$gender)
```

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

```
str(input_data$gender)
```

```
##  Factor w/ 2 levels "female","male": 2 2 1 1 2 1 2
```

Factors in Data Frame III

```
input_dataNew <- data.frame(height,weight,gender,
                 stringsAsFactors = FALSE)
print(is.factor(input_dataNew$gender))
```

```
## [1] FALSE
```

```
print(input_dataNew$gender)
```

```
## [1] "male"   "male"   "female" "female" "male"   "female" '
```

```
str(input_dataNew$gender)
```

```
##  chr [1:7] "male" "male" "female" "female" "male" "female"
```

Changing the Order of Levels I

```r
data <- c("East","West","East","North","North","East","West","
factor_data <- factor(data)
print(factor_data)
```

```
##  [1] East  West  East  North North East  West  West  West
## Levels: East North West
```

```r
str(factor_data)
```

```
##  Factor w/ 3 levels "East","North",..: 1 3 1 2 2 1 3 3 3 1
```

Changing the Order of Levels II

```
new_order_data <- factor(factor_data,levels = c("East","West",
print(new_order_data)
```

```
##  [1] East  West  East  North North East  West  West  West
## Levels: East West North
```

```
str(new_order_data)
```

```
##  Factor w/ 3 levels "East","West",..: 1 2 1 3 3 1 2 2 2 1 .
```

Ordinal Variable

```
data <- c("High", "Low", "Medium","High", "Medium", "High", "M
ordinalData <- ordered(data)
print(ordinalData)
```

```
## [1] High   Low    Medium High   Medium High   Medium High
## Levels: High < Low < Medium
```

```
str(ordinalData)
```

```
##  Ord.factor w/ 3 levels "High"<"Low"<"Medium": 1 2 3 1 3 1
```

Ordinal Variable II

```
factorData <- factor(data)
print(factorData)
```

```
## [1] High   Low    Medium High   Medium High   Medium High
## Levels: High Low Medium
```

```
str(factorData)
```

```
##  Factor w/ 3 levels "High","Low","Medium": 1 2 3 1 3 1 3 1
```

```
ordinalData <- ordered(factorData)
str(ordinalData)
```

```
##  Ord.factor w/ 3 levels "High"<"Low"<"Medium": 1 2 3 1 3 1
```

Ordinal Variable III

```
factorDataNew <- factor(data, levels = c("Low", "Medium", "Hi
print(factorDataNew)
```

```
## [1] High   Low    Medium High   Medium High   Medium High
## Levels: Low Medium High
```

```
str(factorDataNew)
```

```
##  Factor w/ 3 levels "Low","Medium",..: 3 1 2 3 2 3 2 3 1
```

```
ordinalDataNew <- ordered(factorDataNew)
str(ordinalDataNew)
```

```
##  Ord.factor w/ 3 levels "Low"<"Medium"<..: 3 1 2 3 2 3 2 3
```

Ordinal Variable IV

```
ordinalData <- factor(data, levels = c("Low", "Medium", "High"
                      ordered = TRUE)
print(ordinalData)
```

```
## [1] High   Low    Medium High   Medium High   Medium High
## Levels: Low < Medium < High
```

```
str(ordinalData)
```

```
##  Ord.factor w/ 3 levels "Low"<"Medium"<..: 3 1 2 3 2 3 2 3
```

Operation on Factor Variable

```r
factorDataNew[1] == factorDataNew[2]
```

```
## [1] FALSE
```

```r
factorDataNew[1] != factorDataNew[2]
```

```
## [1] TRUE
```

```r
factorDataNew[1] > factorDataNew[2]
```

```
## Warning in Ops.factor(factorDataNew[1], factorDataNew[2]):
## meaningful for factors
```

```
## [1] NA
```

Operation on Ordinal Variable

```r
print(ordinalDataNew)
```

```
## [1] High   Low    Medium High   Medium High   Medium High
## Levels: Low < Medium < High
```

```r
ordinalDataNew[3] == ordinalDataNew[5]
```

```
## [1] TRUE
```

```r
ordinalDataNew[1] > ordinalDataNew[2]
```

```
## [1] TRUE
```

```r
ordinalDataNew[3] >= ordinalDataNew[5]
```

```
## [1] TRUE
```

Generating Factor Levels

We can generate factor levels by using the gl() function. It takes two integers as input which indicates how many levels and how many times each level.

```
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

```
##  [1] Tampa   Tampa   Tampa   Tampa   Seattle Seattle Seattl
##  [9] Boston  Boston  Boston  Boston
## Levels: Tampa Seattle Boston
```

```
v <- gl(3, 2, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

```
## [1] Tampa   Tampa   Seattle Seattle Boston  Boston
## Levels: Tampa Seattle Boston
```

Control Structure in R

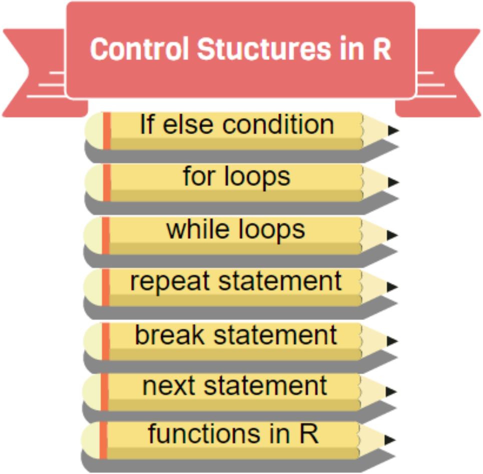


Figure 5:

Decision Making

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

- **if statement**
- **if else statement**
- **switch statement**

If Statement

```r
x <- 13
if(x %% 2 != 0) {
   print("X is an Odd number")
}
```

```
## [1] "X is an Odd number"
```

```r
if(x %% 2 == 0) {
   print("X is an Even number")
}
```

If-Else Statement

```r
x <- 14
if(x %% 2 == 0) {
  print("X is an Even number")
}else{
  print("X is an Odd number")
}
```

```
## [1] "X is an Even number"
```

Switch Statement

```r
x <- switch(
   3,
   "This is the Case1",
   "This is the Case2",
   "This is the Case3",
   "This is the Case4"
)
print(x)
```

```
## [1] "This is the Case3"
```

Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially. The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times.

- **for loop**
- **while loop**
- **repeat loop**

For Loop I

```
v <- c(1, 3, 5)
for ( i in v) {
   print(i)
}
```

```
## [1] 1
## [1] 3
## [1] 5
```

```
for ( i in 1:10) {
   print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
```

For Loop II

```r
v <- c('a','b','c','d','e','f')
for ( i in v) {
   print(i)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
## [1] "e"
## [1] "f"
```

While Loop

```
count <- 5
while (count > 0) {
   print(count)
   count = count - 1
}
```

```
## [1] 5
## [1] 4
## [1] 3
## [1] 2
## [1] 1
```

Repeat Loop

```r
count <- 1
repeat {
   print(count)
   count <- count + 1
   if(count > 5) {
      break
   }
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

Control Statements

Control statements change execution sequence from its normal sequence.

- **Break**
- **Next**

```
v <- LETTERS[1:4]
for ( i in v) {
   if (i == "D") {
      next
   }
   print(i)
}
```

```
## [1] "A"
## [1] "B"
## [1] "C"
```

Functions

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

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

Function Definition

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

function_name <- function(arg_1, arg_2, . . .) {

Function body

}

Built-in Function

```
print(seq(32,44))
```

```
##  [1] 32 33 34 35 36 37 38 39 40 41 42 43 44
```

```
print(mean(1:10))
```

```
## [1] 5.5
```

```
print(sum(1:5))
```

```
## [1] 15
```

User-defined Function

```r
funName <- function(a) {
   for(i in 1:a) {
      b <- i^2
      print(b)
   }
}

# Call the function new.function supplying 6 as an argument.
funName(5)
```

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

Calling a Function without an Argument

```r
funName <- function() {
   for(i in 1:5) {
      print(i^2)
   }
}

# Call the function without supplying an argument.
funName()
```

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

Calling a Function with Argument Values (by position and by name)

```
funName <- function(a,b,c) {
   result <- a * b + c
   print(result)
}

# Call the function by position of arguments.
funName(5,3,11)
```

```
## [1] 26
```

```
# Call the function by names of the arguments.
funName(c = 11, a = 5, b = 3)
```

```
## [1] 26
```

Calling a Function with Default Argument

```r
funName <- function(a = 3, b = 6) {
   result <- a * b
   print(result)
}

# Call the function without giving any argument.
funName()
```

```
## [1] 18
```

```r
# Call the function with giving new values of the argument.
funName(9,5)
```

```
## [1] 45
```

Lazy Evaluation of Function

```
funName <- function(a, b) {
  print(a^2)
  if(a>6){
    print(b)
  }
}

funName(5)
```

```
## [1] 25
```

Data Interfaces

R can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like:

- csv
- xlsx
- web
- xml etc.

Reading a CSV File I

```r
data <- read.csv("testData.csv")
print(data)
```

```
##   id     name salary start_date       dept
## 1  1     Rick 623.30 2012-01-01         IT
## 2  2      Dan 515.20 2013-09-23 Operations
## 3  3 Michelle 611.00 2014-11-15         IT
## 4  4     Ryan 729.00 2014-05-11         HR
## 5 NA     Gary 843.25 2015-03-27    Finance
## 6  6     Nina 578.00 2013-05-21         IT
## 7  7    Simon 632.80 2013-07-30 Operations
## 8  8     Guru 722.50 2014-06-17    Finance
```

Reading a CSV File II

```
data <- read.csv("testData.csv", header = FALSE)
print(data)
```

```
##   V1       V2     V3         V4         V5
## 1 id     name salary start_date       dept
## 2  1     Rick  623.3 2012-01-01         IT
## 3  2      Dan  515.2 2013-09-23 Operations
## 4  3 Michelle    611 2014-11-15         IT
## 5  4     Ryan    729 2014-05-11         HR
## 6        Gary 843.25 2015-03-27    Finance
## 7  6     Nina    578 2013-05-21         IT
## 8  7    Simon  632.8 2013-07-30 Operations
## 9  8     Guru  722.5 2014-06-17    Finance
```

Reading a CSV File III

```
data <- read.csv("testData.csv", header = TRUE)
print(paste(nrow(data), ncol(data)))
```

```
## [1] "8 5"
```

```
str(data)
```

```
## 'data.frame':    8 obs. of  5 variables:
##  $ id        : int  1 2 3 4 NA 6 7 8
##  $ name      : Factor w/ 8 levels "Dan","Gary","Guru",..: 6
##  $ salary    : num  623 515 611 729 843 ...
##  $ start_date: Factor w/ 8 levels "2012-01-01","2013-05-21"
##  $ dept      : Factor w/ 4 levels "Finance","HR",..: 3 4 3
```

Data Manipulations

```r
dataIT <- subset( data, dept == "IT")
print(dataIT)
```

```
##   id     name salary start_date dept
## 1  1     Rick  623.3 2012-01-01   IT
## 3  3 Michelle  611.0 2014-11-15   IT
## 6  6     Nina  578.0 2013-05-21   IT
```

```r
max(dataIT$salary)
```

```
## [1] 623.3
```

Loading Data

```r
data("iris")
dim(iris)
```

```
## [1] 150   5
```

```r
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 .
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",.
```

Return the First Part of an Object

```r
irisData <- iris
head(irisData)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Return the Last Part of an Object

```
irisData <- iris
tail(irisData)
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width   Spe
## 145          6.7         3.3          5.7         2.5 virgi
## 146          6.7         3.0          5.2         2.3 virgi
## 147          6.3         2.5          5.0         1.9 virgi
## 148          6.5         3.0          5.2         2.0 virgi
## 149          6.2         3.4          5.4         2.3 virgi
## 150          5.9         3.0          5.1         1.8 virgi
```

Writing into a CSV File

R can create csv file form existing data frame. The **write.csv()** function is used to create the csv file.

```r
write.csv(irisData,"output.csv", row.names = FALSE)
newData <- read.csv("output.csv")
head(newData)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Reading a xlsx File

```
install.packages(“xlsx”)

library(“xlsx”)

data <- read.xlsx(“testData.xlsx”, sheetIndex = 1)
```

Thank You