Loading and Handling Data in R

Challenges of Analytical Data Processing
- Expression, Variables and Functions
- Missing Values Treatment in R
- Using the ‘as’ Operator to Change the Structure of Data,
- Vectors
- Matrices

Factors
- List

Aggregating and Group Processing of a Variable,
- Simple Analysis Using R,
- Methods for Reading Data,

Comparison of R GUIs for Data Input

Expression, variables and functions

q Expressions

Operation	*Operator*	*Description*	*Example*
Addition	$x + y$	y added to x	`> 4 + 8` `[1] 12`
Subtraction	$x - y$	y subtracted from x	`> 10 - 3` `[1] 7`
Multiplication	$x * y$	x multiplied by y	`> 7 * 8` `[1] 56`
Division	x / y	x divided by y	`< 8/3` `[1] 2.666667`

Exponentiation	$x \wedge y$
	$x ** y$
		x raised to the power y	`> 2 ^ 5`
			`[1] 32`
			Or
			`>2 ** 5`
			`[1] 32`

Modulus	$x \%\% y$	Remainder of (x divided by y)	`> 5 %% 3` `[1] 2`
Integer Division	$x\%/\%y$	x divided by y but rounded down	`> 5 %/% 2` `[1] 2`
Computing the Square Root	sqrt(x)	Computing the square root of x	`> sqrt (25)` `[1] 5`

RGui (32-bit) - [R Console]

File Edit View Misc Packages Windows Help

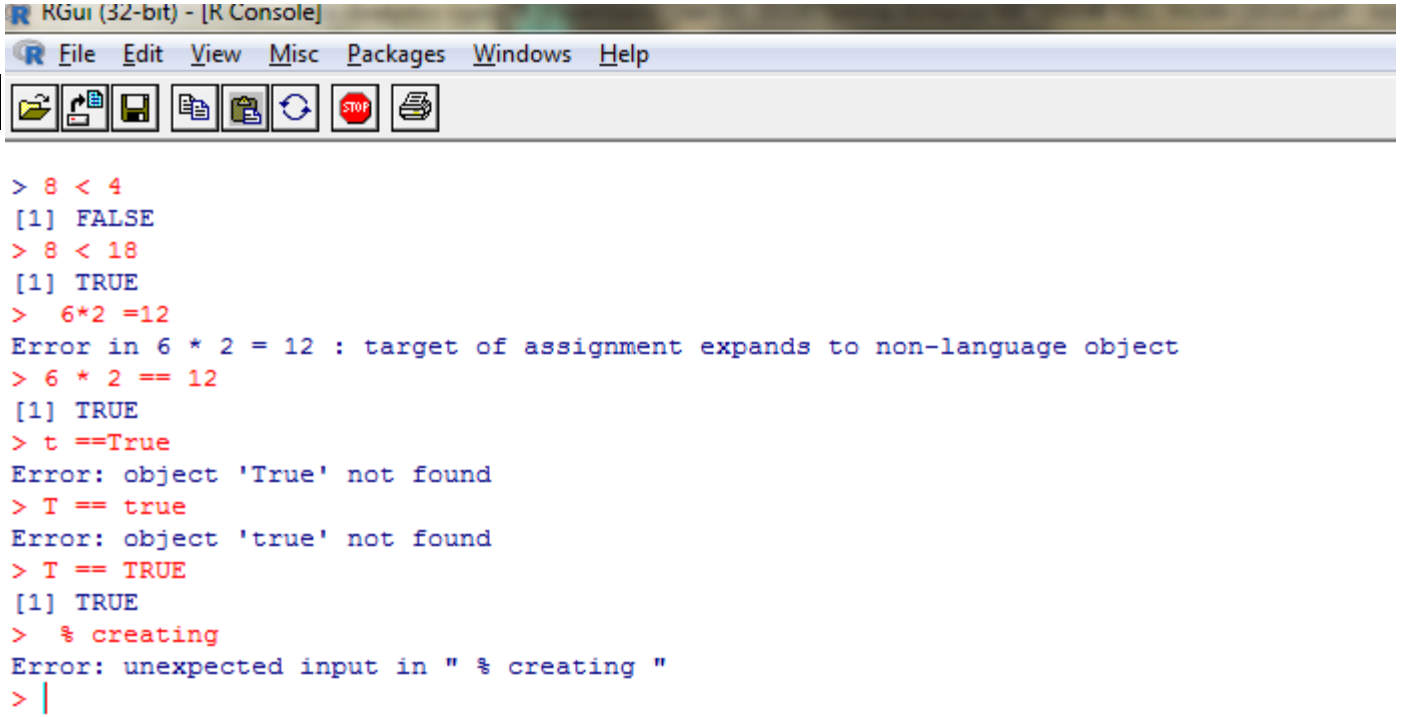
```
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 2+3
[1] 5
> 1013
[1] 1013
> 10-13
[1] -3
> x * y
Error: object 'x' not found
> x=2
> y=4
> x*y
[1] 8
> 2^5
[1] 32
> 2** 5
[1] 32
> 5 %% 3
[1] 2
> 5 %/% 2
[1] 2
> sqrt(50
+ )
[1] 7.071068
> sqrt(25)
[1] 5
>
```

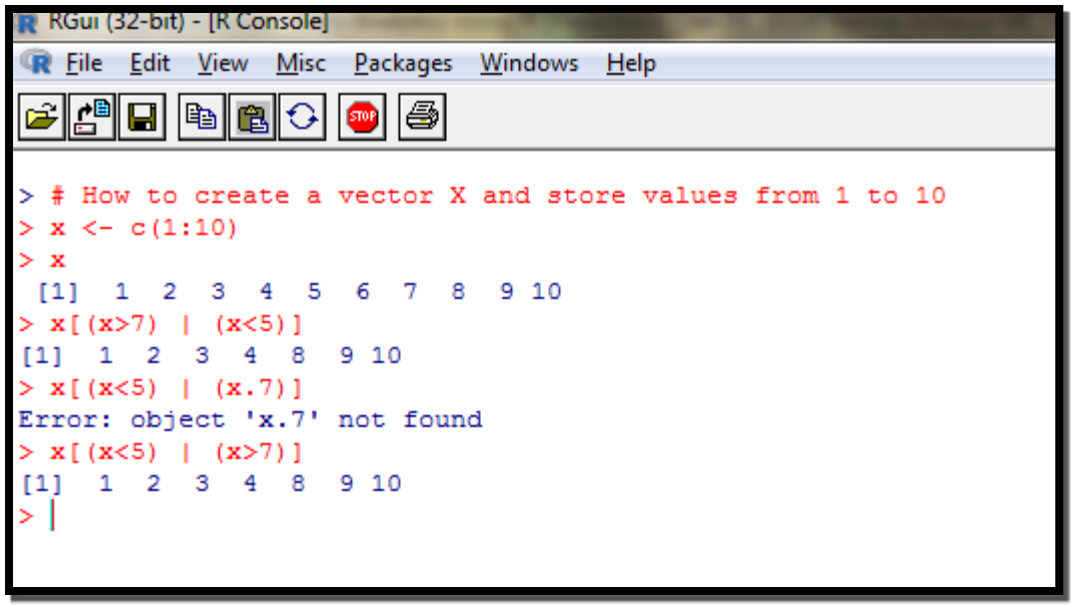
1:41 PM
9/23/2021

Logical Values

- Logical values are TRUE and FALSE or T and F. Note that these are case sensitive. The equality operator is ==.

- to Clear Screen use Ctrl





Part (i) Display 'TRUE' for elements whose values are more than 7, else display 'FALSE'.

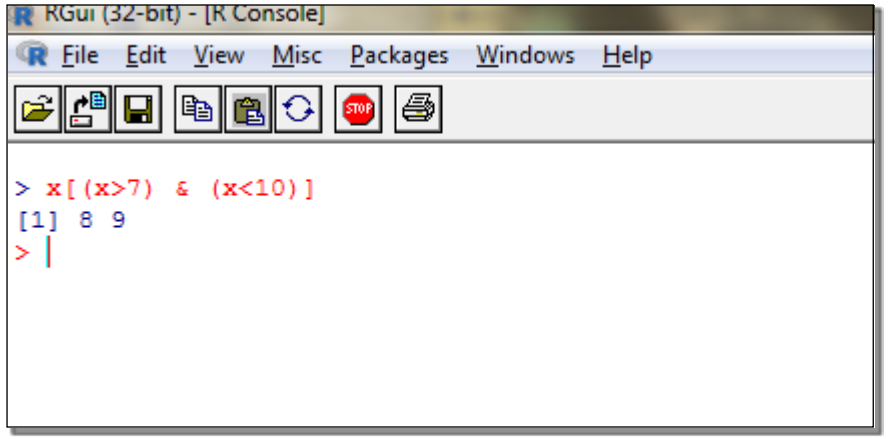
> x>7

[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE

Part (ii) Display 'TRUE' for elements whose values are less than 5, else display 'FALSE'.

> x<5

[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE

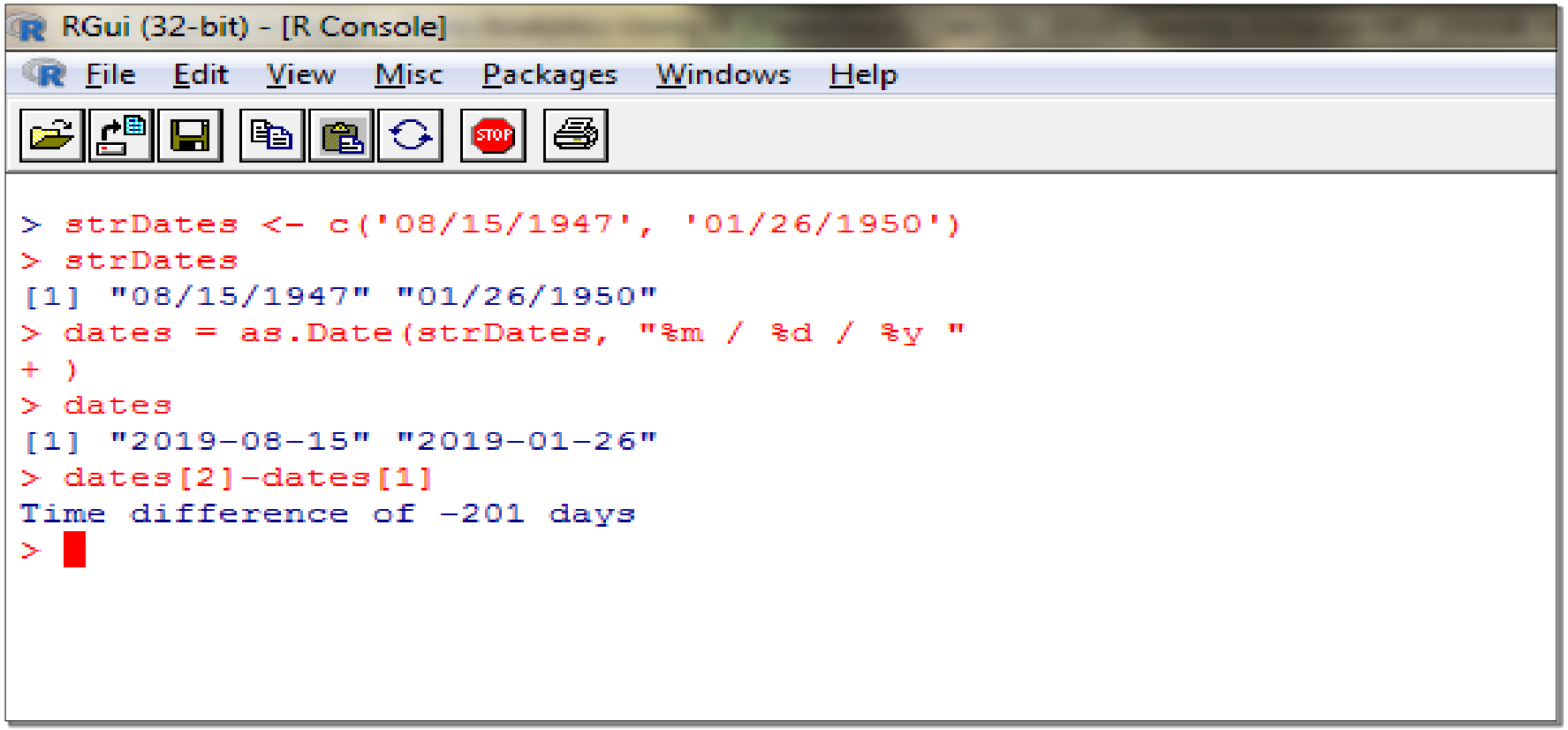

RGui (32-bit) - [R Console]
File Edit View Misc Packages Windows Help
> x[(x>7) & (x<10)]
[1] 8 9
>

Date Activity

Dates : The default format of date is YYYY-MM-DD.

Print system's date.	Sys.Date()	[1] "2017-01-13"
Print system's time.	Sys.time()	[1] "2017-01-13 10:54:37 IST"
Print the time zone	Sys.timezone()	[1] "Asia/Calcutta"
Print today's date.	today <- Sys.Date()	today [1] "2017-01-13"
Formate Date	format (today, format = "%B %d %Y")	"January 13 2017"
Store date as a text data type.	CustomDate = "2016-01-13" class (CustomDate)	[1] "2016-01-13" [1] "character"
Convert the date stored as text data type into a date data type.	CustDate = as.Date(CustomDate) class(CustDate) CustDate	[1] "Date" [1] "2016-01-13"

Find a Difference between Two Dates



Variables

S.No	Variables	Explain
1	Assign a value of 50 to the variable called 'Var'.	Var <-50 or Var=5
2	Print the value in the variable, 'Var	Var
3	Perform arithmetic operations on the variable, 'Var'.	Var + 10
4	Reassign a string value to the variable, 'Var'. > Var <- "R is a Statistical Programming Language" Print the value in the variable, 'Var'.	Var [1] "R is a Statistical Programming Language"
5	Reassign a logical value to the variable, 'Var'.	Var <- TRUE Var [1] TRUE

Loading and Handling Data in R

Challenges of Analytical Data Processing
- Expression, Variables and Functions
- Missing Values Treatment in R
- Using the ‘as’ Operator to Change the Structure of Data,
- Vectors
- Matrices

Factors
- List

Aggregating and Group Processing of a Variable,
- Simple Analysis Using R,
- Methods for Reading Data,

Comparison of R GUIs for Data Input

Vectors

A vector can have a list of values.
The values can be numbers, strings or logical.
All the values in a vector should be of the same data type.

A few points to remember about vectors in R are:

- Vectors are stored like arrays in C
- Vector indices begin at 1
- All vector elements must have the same mode such as integer, numeric (floating point number), character (string), logical (Boolean), complex, object, etc.

S.No	Vector	Example
1	Create a vector of numbers The c function (c is short for combine) creates a new vector consisting of three values, viz. 4, 7 and 8.	> c(4, 7, 8) [1] 4 7 8
2	Create a vector of string values.	> c("R", "SAS", "SPSS") [1] "R" "SAS" "SPSS"
3	Create a vector of logical values.	> c(TRUE, FALSE) [1] TRUE FALSE
4	A vector cannot hold values of different data types. Consider the example below on placing integer, string and Boolean values together in a vector.	> c(4, 8, "R", FALSE) [1] "4" "8" "R" "FALSE"
5	Declare a vector by the name, 'Project' of length 3 and store values in it. > Project <- vector(length = 3) > Project [1] <- "Finance Project" > Project [2] <- "Retail Project" > Project [3] <- "Energy Project"	Project [1] "Finance Project" "Retail Project" "Energy Project" > length (Project) [1] 3

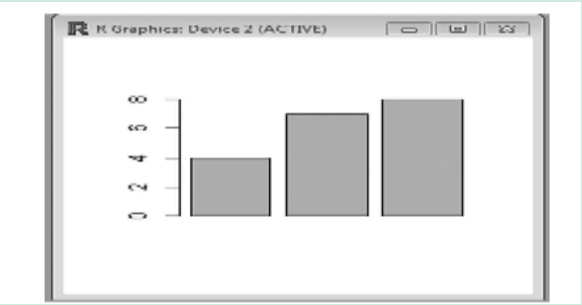
A sequence vector can be created with a start:end notation.

S.No	Sequence Vector	Example
1	Create a sequence of numbers between 1 and 5 (both inclusive).	> 1:5 [1] 1 2 3 4 5 > seq(1:5) [1] 1 2 3 4 5
2	The default increment with seq is 1. However, it also allows the use of increments other than 1.	> seq (1, 10, 2) [1] 1 3 5 7 9 > seq (from=1, to=10, by=2) [1] 1 3 5 7 9 > seq (1, 10, by=2) [1] 1 3 5 7 9
3	seq can also generate numbers in the descending order.	> 10:1 [1] 10 9 8 7 6 5 4 3 2 1 > seq (10, 1, by=−2) [1] 10 8 6 4 2

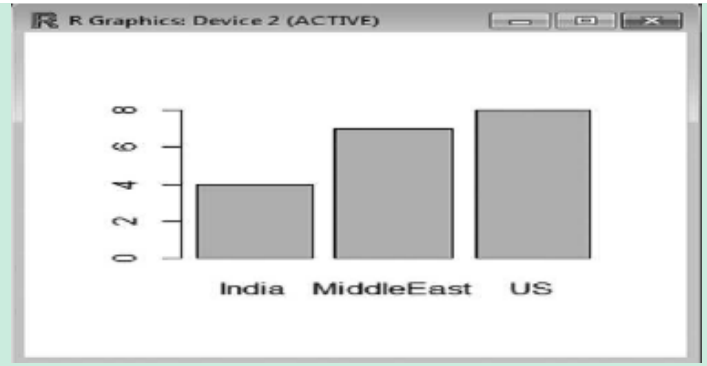
To access more than one value from the vector.

S.No	Rep Function	Example
1	Access the first and the fifth element from the vector, 'VariableSeq'.	> VariableSeq[c(1, 5)] [1] "R" "language"c
2	Access first to the fourth element from the vector, 'VariableSeq'.	> VariableSeq[1:4] [1] "R" "is" "a" "good programming"
3	Access the first, fourth and the fifth element from the vector, 'VariableSeq'.	> VariableSeq[c(1, 4:5)] [1] "R" "good programming" "language"
4	Retrieve all the values from the variable, 'VariableSeq'	> VariableSeq [1] "R" "is" "a" "good programming" [5] "language"

Vector Names : The names() function helps to assign names to the vector elements.

S.No	Function	Examples
	Creating a name to a vector values placeholder R is a programming language 1 2 3 4 5	> placeholder <- 1:5 > names(placeholder) <- c("r", "is", "a", "programming", "language")
	placeholder [3]	a 3
	Plot a bar graph using the barplot function. The barplot function uses a vector's values to plot a bar chart. > BarVector <- c(4, 7, 8) > barplot(BarVector)	

Vector Names : The names() function helps to assign names to the vector elements.

S.No	Function	Examples
	Let us use the name function to assign names to the vector elements. These names will be used as labels in the barplot. > names(BarVector) <- c("India", "MiddleEast", "US") Øbarplot(BarVector)	

S.No	Vector Math Function	Example
1	We can run other arithmetic operations on the vector as given:	> x – 1 [1] 4 7 8 > x * 2 [1] 10 16 18 > x / 2 [1] 2.5 4.0 4.5
2	Let us practice these arithmetic operations on two vectors.	> x [1] 5 8 9 > y <- c(1, 2, 3) > y [1] 1 2 3 > x + y [1] 6 10 12

Vector Math : Let us define a vector, 'x' with three values. Let us add a scalar value (single value) to the vector. This value will get added to each vector element

S.No	Vector Math Function	Example
1	value will get added to each vector element.	> x <- c(4, 7, 8) > x +1 [1] 5 8 9 the vector will retain its individual elements. > x [1] 4 7 8
2	If the vector needs to be updated with the new values, type the statement given below.	> x <- x + 1 > x [1] 5 8 9
3	Other Arithmetic operations	> x – 1 [1] 4 7 8 > x * 2 [1] 10 16 18 > x / 2 [1] 2.5 4.0 4.5

Vector Math : Let us define a vector, 'x' with three values. Let us add a scalar value (single value) to the vector. This value will get added to each vector element

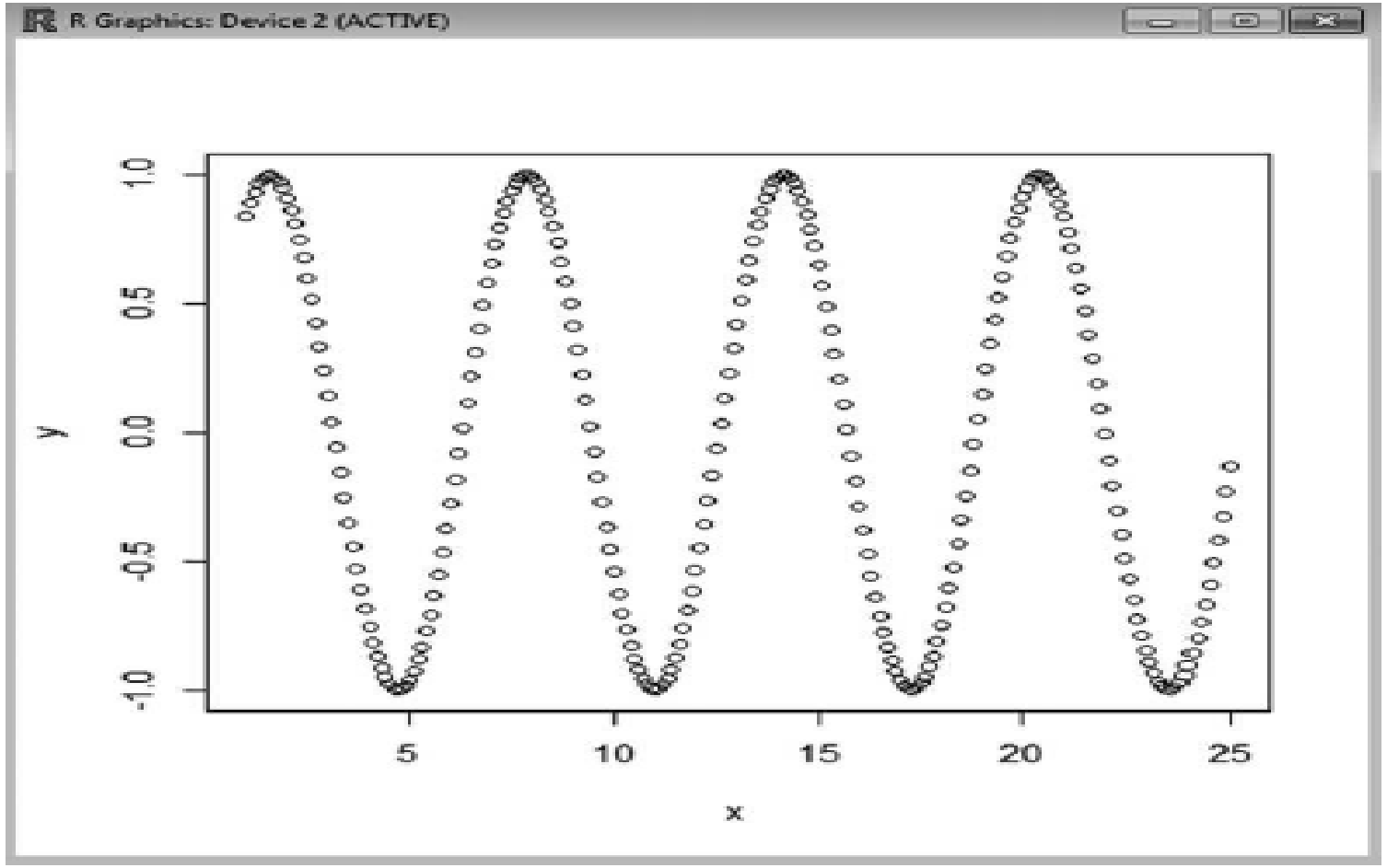
S.No	Vector Math Function	Example
1	Other arithmetic operations are:	> x – y [1] 4 6 6 > x * y [1] 5 16 27
2	Check if the two vectors are equal. The comparison takes place element by element.	> x [1] 5 8 9 > y [1] 1 2 3 > x==y [1] FALSE FALSE FALSE > x < y [1] FALSE FALSE FALSE > sin(x) [1] -0.9589243 0.9893582 0.4121185

Vector Recycling : If an operation is performed involving two vectors that requires them to be of the same length, the shorter one is recycled, i.e. repeated until it is long enough to m

S.No	Vector Recycling Function	Example
1	Add two vectors wherein one has length, 3 and the other has length, 6.	> c(1, 2, 3) + c(4, 5, 6, 7, 8, 9) [1] 5 7 9 8 10 12
2	Multiply the two vectors wherein one has length, 3 and the other has length, 6.	> c(1, 2, 3) * c(4, 5, 6, 7, 8, 9) [1] 4 10 18 7 16 27

Plot a Scatter Plot

The function to plot a scatter plot is 'plot'.
This function uses two vectors, i.e.
one for the x axis and another for the y axis.
The objective is to understand the relationship between numbers and their sines.
We will use two vectors.
Vector, **x which will have a sequence of values between 1 and 25**
at an interval of 0.1 and
vector, **y which stores the sines of all values held in vector, x.**


R Graphics: Device 2 (ACTIVE)
y
1.0
0.5
0.0
-0.5
-1.0
5
10
15
20
25
x

Session outcomes

At the end of the session student will be able to understand:

- Introduction of R
- Install R
- Install any R package

Aggregating and group processing of a variable

R provides some functions for aggregation operation.

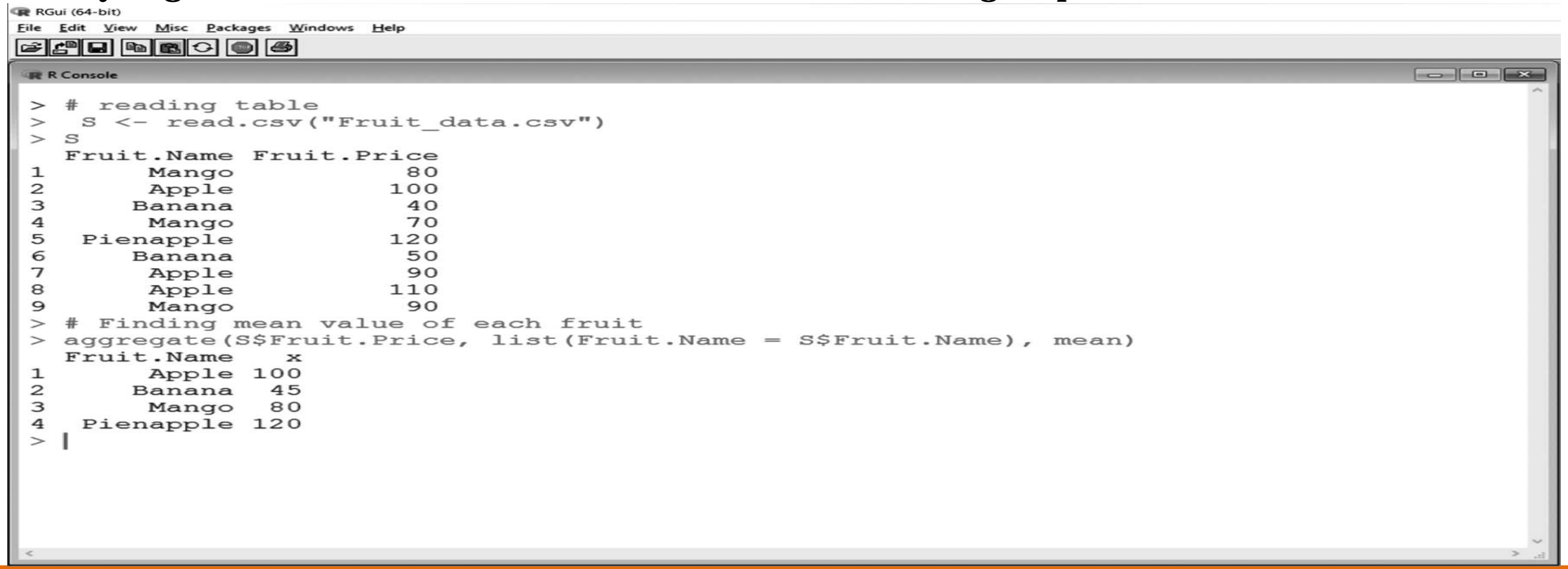
1. aggregate() Function: is an inbuilt function of R that aggregates data values. The function also splits data into groups after performing given statistical functions. The syntax of the aggregate() function is

aggregate(x, ...) or aggregate(x, by, FUN, ...)

where, x is an object, by argument defines the list of group elements of the specific variable of the dataset, FUN argument is a statistic function that returns a numeric value after given statistic operations and the dots '...' define the other optional argument

The following example reads a table, 'Fruit_data.csv' into object, 'S'. The aggregate() function computes the mean price of each type of fruit.

Here by argument is list(Fruit.Name = S$Fruit.Name) that groups the Fruit.Name columns

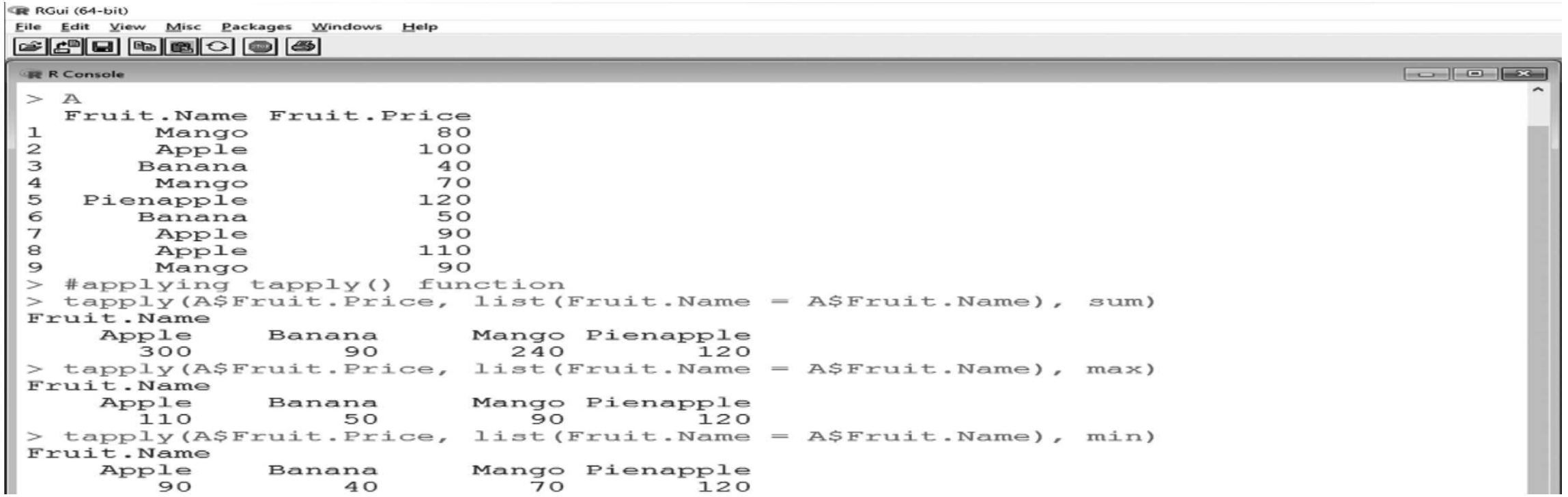


The tapply() function : is also an inbuilt function of R and works in a manner similar to the function aggregate(). The function aggregates the data values into groups after performing the given statistical functions.

The syntax of the tapply () function is
tapply (x, ...) or tapply(x, INDEX, FUN, ...)

where, x is an object that defines the summary variable, INDEX argument defines the list of group elements—also called group variable, FUN argument is a statistic function that returns a numeric value after given statistic operations and the dots '...' define the other optional argument.

The following example reads the table, 'Fruit_data.csv' into object, 'A'.
The tapply()function computes the sum and price of each type of fruit.
Here Fruit.Price is a summary variable and Fruit.Name is a grouping variable.
The FUN function is applied on the summary variable, Fruit.Price

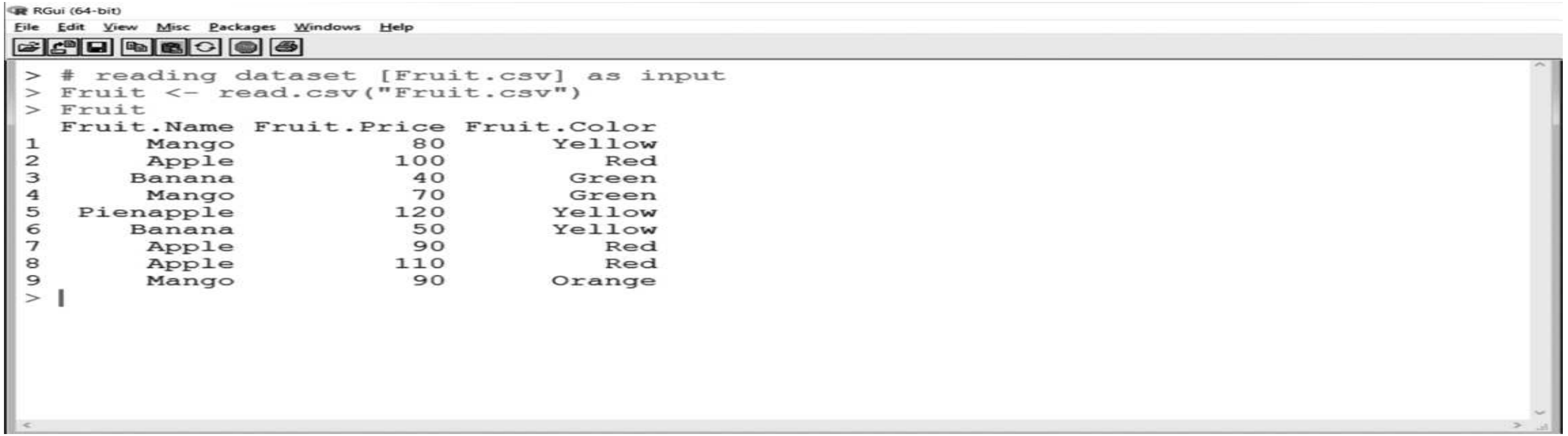



```
> A
  Fruit.Name Fruit.Price
1      Mango           80
2      Apple          100
3     Banana           40
4      Mango           70
5  Pienapple          120
6     Banana           50
7      Apple           90
8      Apple          110
9      Mango           90
> #applying tapply() function
> tapply(A$Fruit.Price, list(Fruit.Name = A$Fruit.Name), sum)
Fruit.Name
 Apple Banana Mango Pienapple
   300     90   240        120
> tapply(A$Fruit.Price, list(Fruit.Name = A$Fruit.Name), max)
Fruit.Name
 Apple Banana Mango Pienapple
   110     50    90        120
> tapply(A$Fruit.Price, list(Fruit.Name = A$Fruit.Name), min)
Fruit.Name
 Apple Banana Mango Pienapple
    90     40    70        120
```

Simple analysis using R

you will learn how to read data from a dataset, perform a common operation and see the output.

Input: is the first step in any processing, including analytical data processing. Here, the input is dataset, 'Fruit'. For reading the dataset into R, use read.table() or read.csv() function.



Describe Data Structure

After reading the dataset into the R workspace, the dataset can be described using different functions like names(), str(), summary(), head() and tail().

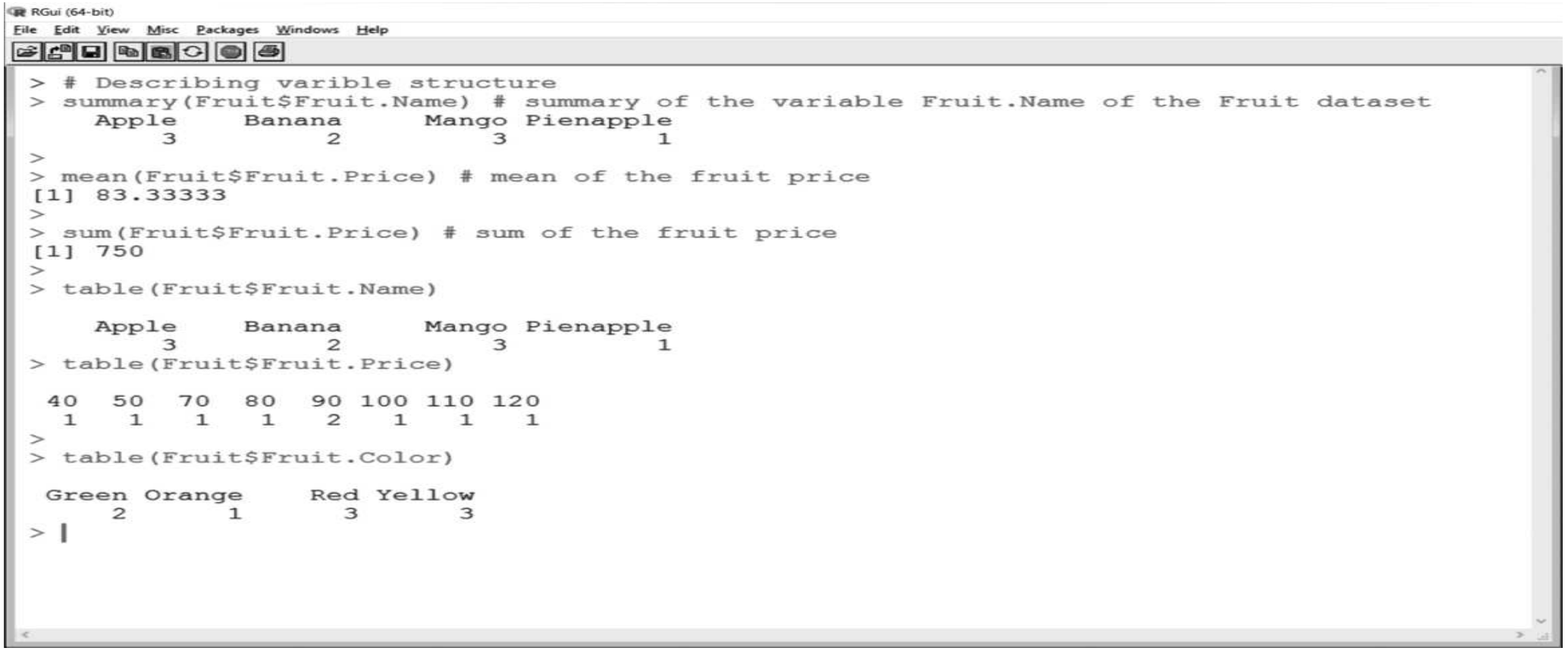
RGui (64-bit)

File Edit View Misc Packages Windows Help

```
> # Describing data structure
> names(Fruit)    # varibles of the dataset
[1] "Fruit.Name"  "Fruit.Price" "Fruit.Color"
>
> str(Fruit) # structure of the dataset
'data.frame':    9 obs. of  3 variables:
 $ Fruit.Name : Factor w/ 4 levels "Apple","Banana",..: 3 1 2 3 4 2 1 1 3
 $ Fruit.Price: int  80 100 40 70 120 50 90 110 90
 $ Fruit.Color: Factor w/ 4 levels "Green","Orange",..: 4 3 1 1 4 4 3 3 2
>
> head(Fruit,3) #reading top 3 rows of the dataset
  Fruit.Name Fruit.Price Fruit.Color
1      Mango          80       Yellow
2      Apple         100          Red
3     Banana          40        Green
>
> tail(Fruit,3) #reading bottom 3 rows of the dataset
  Fruit.Name Fruit.Price Fruit.Color
7      Apple          90          Red
8      Apple         110          Red
9      Mango          90       Orange
>
> summary(Fruit) #summary of the dataset
     Fruit.Name   Fruit.Price       Fruit.Color
 Apple     :3     Min.   : 40.00   Green :2
 Banana    :2     1st Qu.: 70.00   Orange:1
 Mango     :3     Median : 90.00   Red   :3
 Pienapple:1     Mean   : 83.33   Yellow:3
                 3rd Qu.:100.00
```

Describe Variable Structure

After describing the dataset, you can also describe the variables of the dataset using different functions. For describing the variables and performing operations on them, many functions are available.

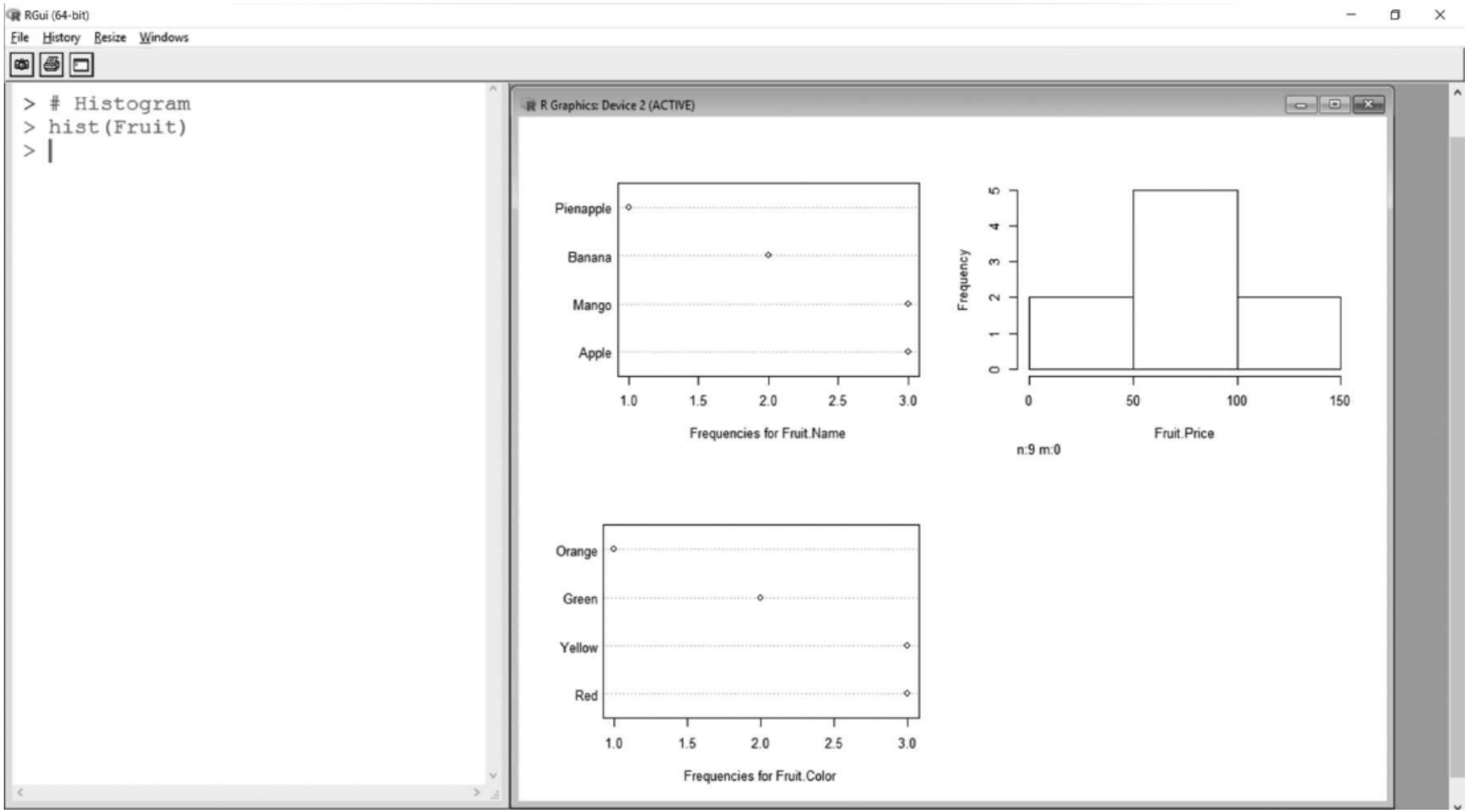


describes the histogram of the ‘Fruit’ dataset using the hist() function. A histogram is a graphical display of data that uses many bars of different heights.

The complete syntax for **hist() function** is:

hist(x, breaks = ‘Sturges’, freq = NULL, probability = !freq,include.lowest = TRUE, right = TRUE,
density = NULL, angle = 45, col = NULL, border = NULL, main = paste(‘Histogram of’ , xname),
xlim = range(breaks), ylim = NULL, xlab = xname, ylab, axes = TRUE, plot = TRUE, labels = FALSE,
nclass = NULL, warn.unused = TRUE, ...)

x is the vector for which a histogram is required.
freq is a logical value. If TRUE, the histogram graphic is a representation of frequencies, the counts component of the result. If FALSE, the probability densities and component density are plotted. main, xlab, ylab are arguments to title. plot is a logical value. If TRUE (default), a histogram is plotted, else a list of breaks and
counts is returned

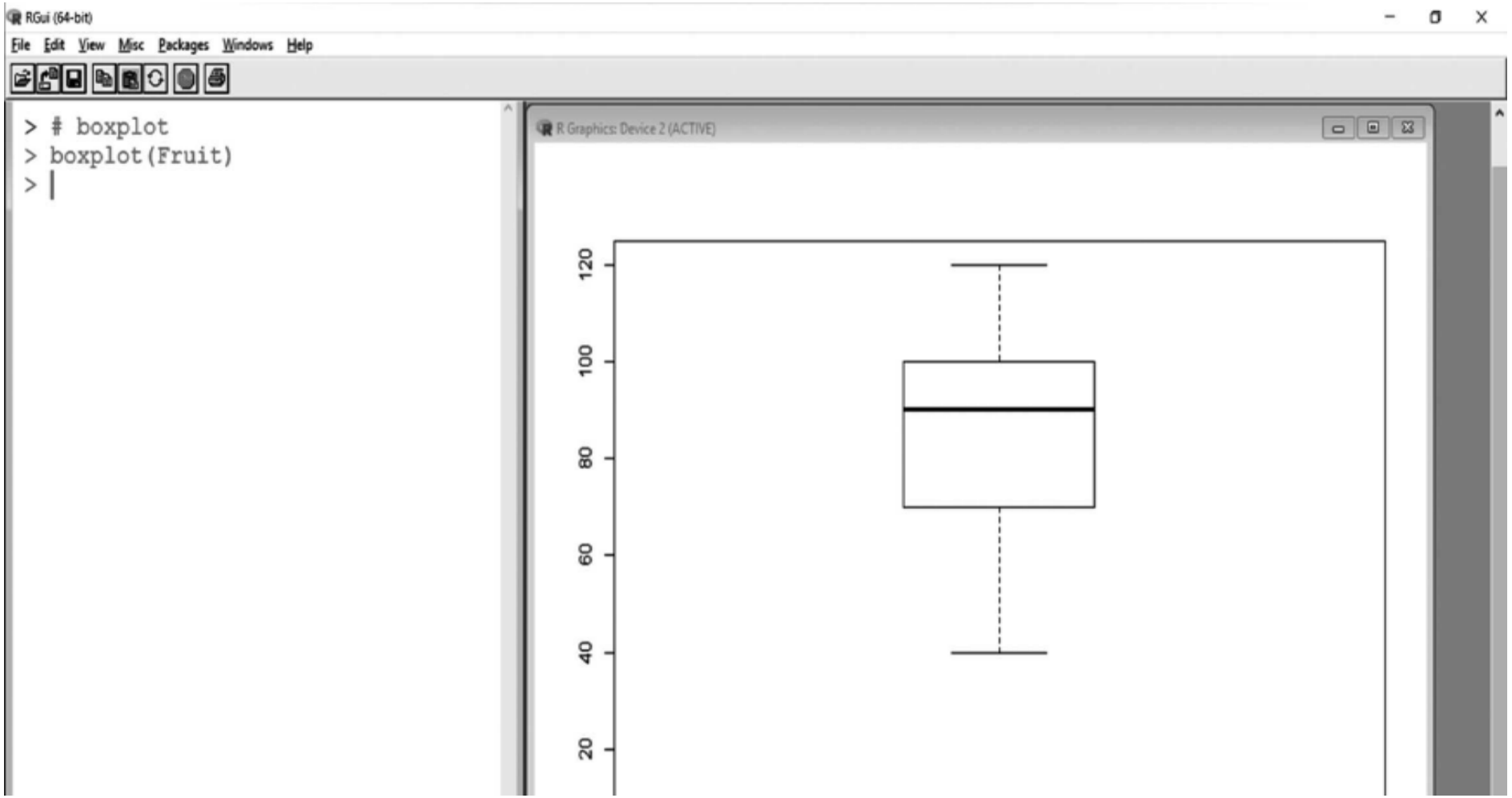

RGui (64-bit)
File History Resize Windows
> # Histogram
> hist(Fruit)
>
R Graphics: Device 2 (ACTIVE)
Pienapple
Banana
Mango
Apple
1.0 1.5 2.0 2.5 3.0
Frequencies for Fruit.Name
Frequency
0 1 2 3 4 5
0 50 100 150
Fruit.Price
n:9 m:0
Orange
Green
Yellow
Red
1.0 1.5 2.0 2.5 3.0
Frequencies for Fruit.Color

A box and whisker plot summarises the group values into boxes.

The syntax for boxplot( ) function is:

boxplot(x, ..., range = 1.5, width = NULL, varwidth = FALSE, notch = FALSE, outline = TRUE, names, plot = TRUE, border = par('fg'), col = NULL, log = '', pars = list(boxwex = 0.8, staplewex = 0.5, outwex = 0.5), horizontal = FALSE, add = FALSE, at = NULL)

where x is a numeric vector or a single list containing such vectors.
outline - If outline is not true, the outliers are not drawn.
range - This determines how far the plot whiskers extend out from the box.


RGui (64-bit)
File Edit View Misc Packages Windows Help
> # boxplot
> boxplot(Fruit)
>
R Graphics: Device 2 (ACTIVE)
120
100
80
60
40
20

Methods for reading data

CSV and Spreadsheets: Comma separated value (CSV) files and spreadsheets are used for storing small size data. R has an inbuilt function facility through which analysts can read both types of files.

Reading CSV Files

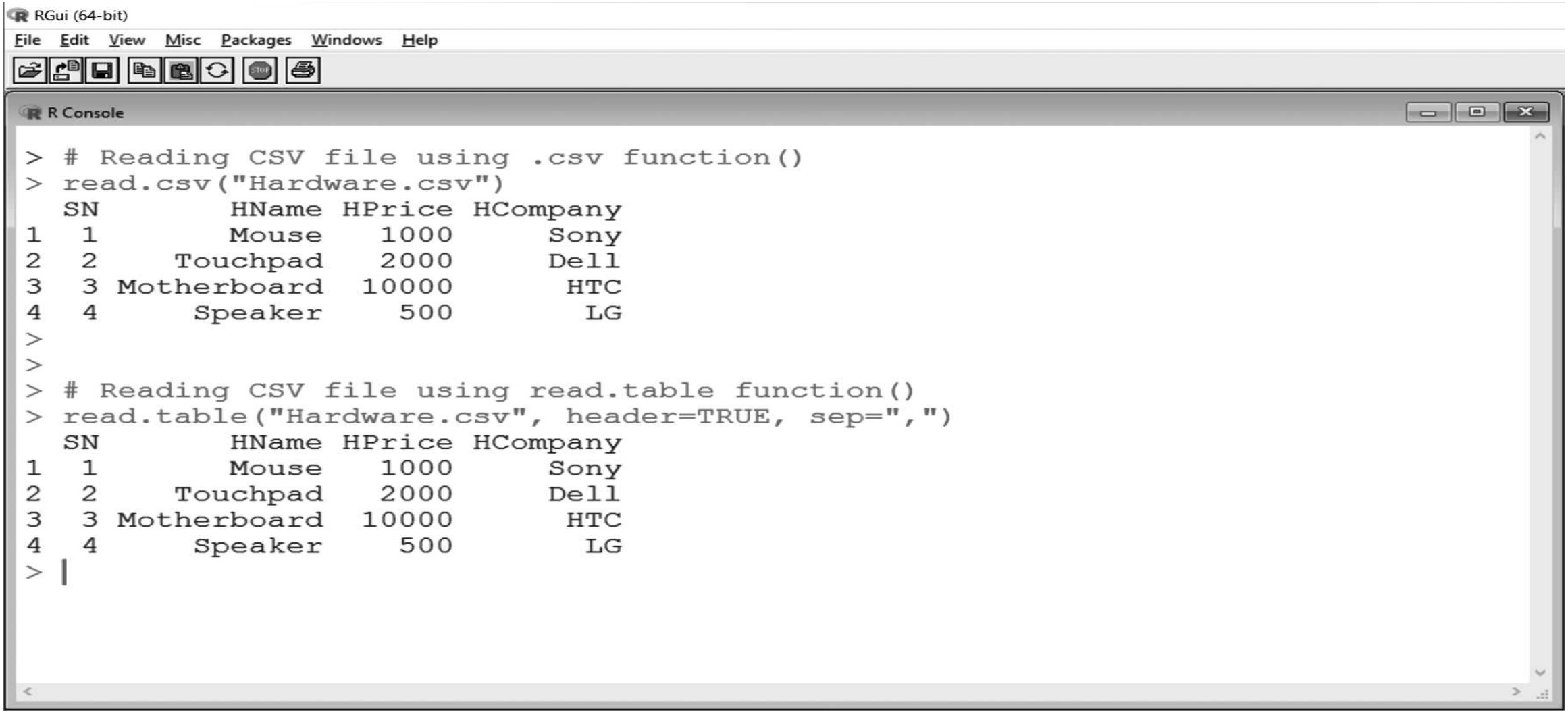
A CSV file uses **.csv extension** and stores data in a table structure format in any plain text. The following function reads data from a CSV file:

read.csv('filename')

where,

filename is the name of the CSV file that needs to be imported.

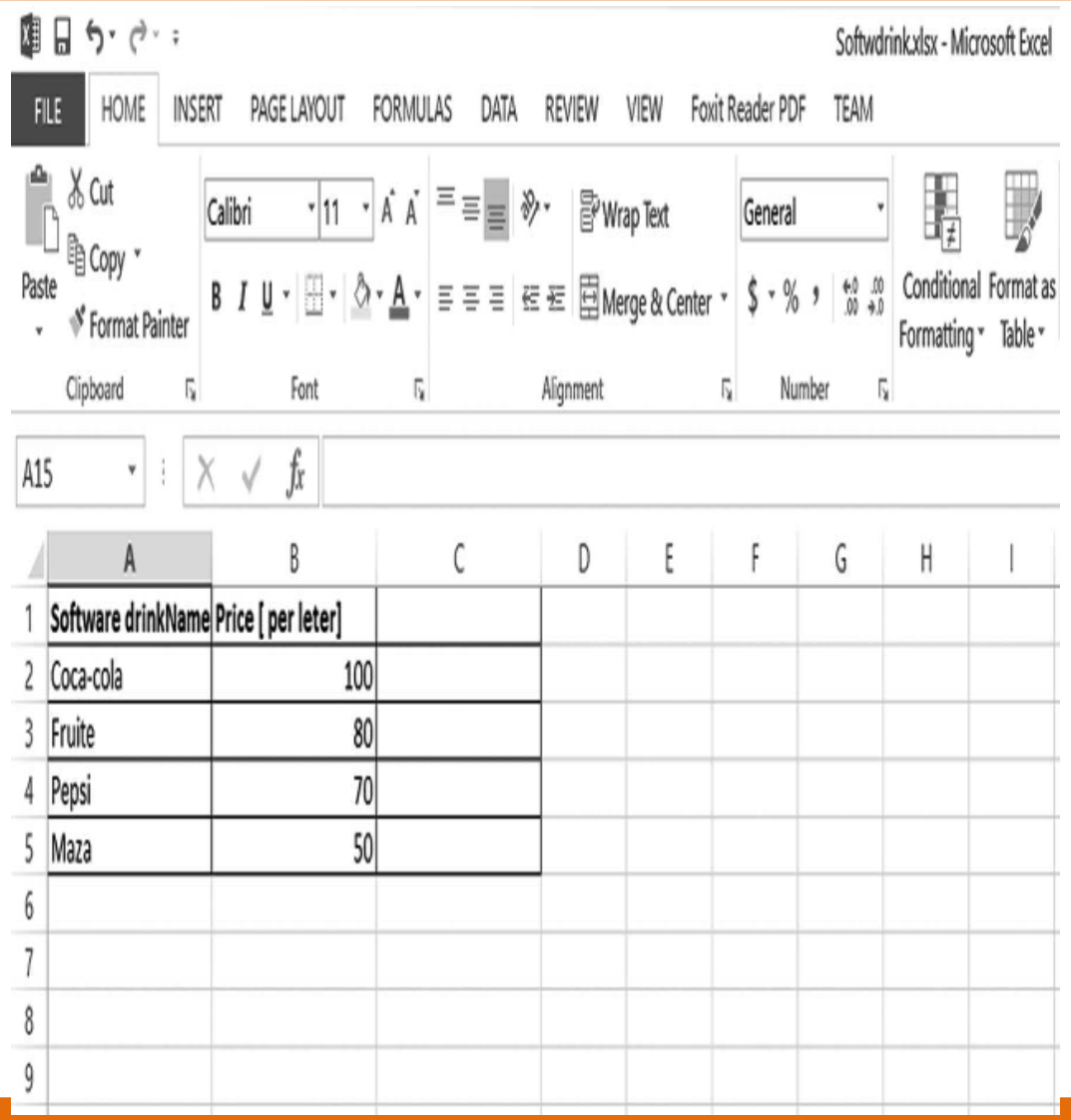
The read.table() function can also read data from CSV files. The syntax of the function is
read.table('filename', header=TRUE, sep=',',...)
where,
filename argument defines the path of the file to be read, header argument contains logical values TRUE and FALSE for defining whether the file has header names on the first line or not, sep argument defines the character used for separating each column of the file and the dots '...' define the other optional arguments.

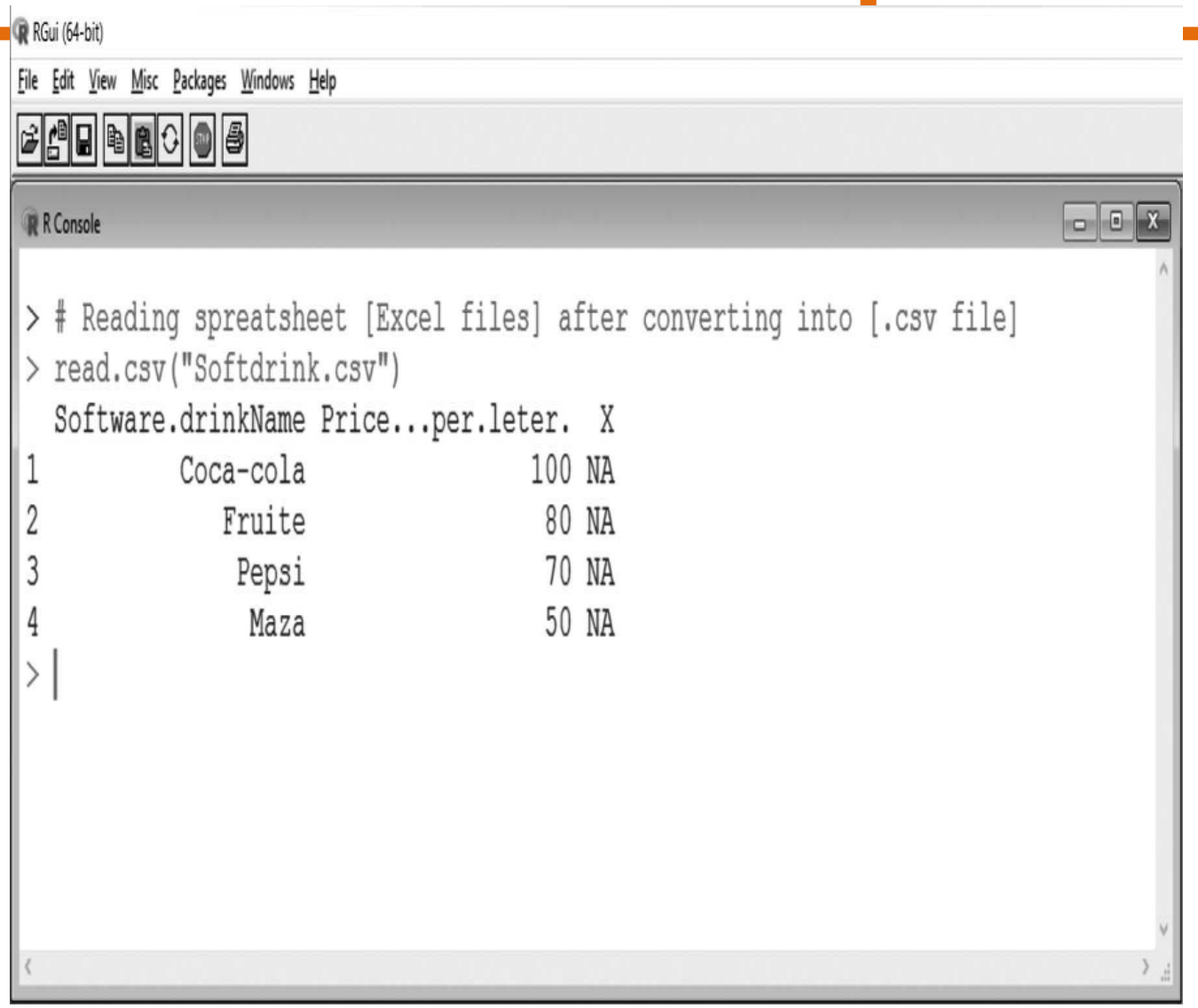

RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
> # Reading CSV file using .csv function()
> read.csv("Hardware.csv")
SN HName HPrice HCompany
1 1 Mouse 1000 Sony
2 2 Touchpad 2000 Dell
3 3 Motherboard 10000 HTC
4 4 Speaker 500 LG
>
>
> # Reading CSV file using read.table function()
> read.table("Hardware.csv", header=TRUE, sep=",")
SN HName HPrice HCompany
1 1 Mouse 1000 Sony
2 2 Touchpad 2000 Dell
3 3 Motherboard 10000 HTC
4 4 Speaker 500 LG
>

Reading Spreadsheets: A spreadsheet is a table that stores data in rows and columns. Many applications are available for creating a spreadsheet. Microsoft Excel is the most popular for creating an Excel file.

An Excel file **uses .xlsx** extension and stores data in a spreadsheet. In R, different **packages are available such as gdata, xlsx**, etc., that provide functions for reading Excel files. Importing such packages is necessary before using any inbuilt function of any package. The **read.xlsx() is an inbuilt function of 'xlsx' package for reading Excel files**. The **syntax of the read.xlsx() function is read.xlsx('filename',...)**

filename argument defines the path of the file to be read and the dots '...' define the other optional arguments.


Softwdrink.xlsx - Microsoft Excel
FILE
HOME
INSERT
PAGE LAYOUT
FORMULAS
DATA
REVIEW
VIEW
Foxit Reader PDF
TEAM
Cut
Copy
Format Painter
Paste
Calibri
11
Wrap Text
General
Merge & Center
Conditional Formatting
Format as Table
Clipboard
Font
Alignment
Number
A15
Software drinkName
Price [per leter]
Coca-cola 100
Fruite 80
Pepsi 70
Maza 50


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
> # Reading spreatsheet [Excel files] after converting into [.csv file]
> read.csv("Softdrink.csv")
Software.drinkName Price...per.leter. X
1 Coca-cola 100 NA
2 Fruite 80 NA
3 Pepsi 70 NA
4 Maza 50 NA
>

Reading Data from Packages

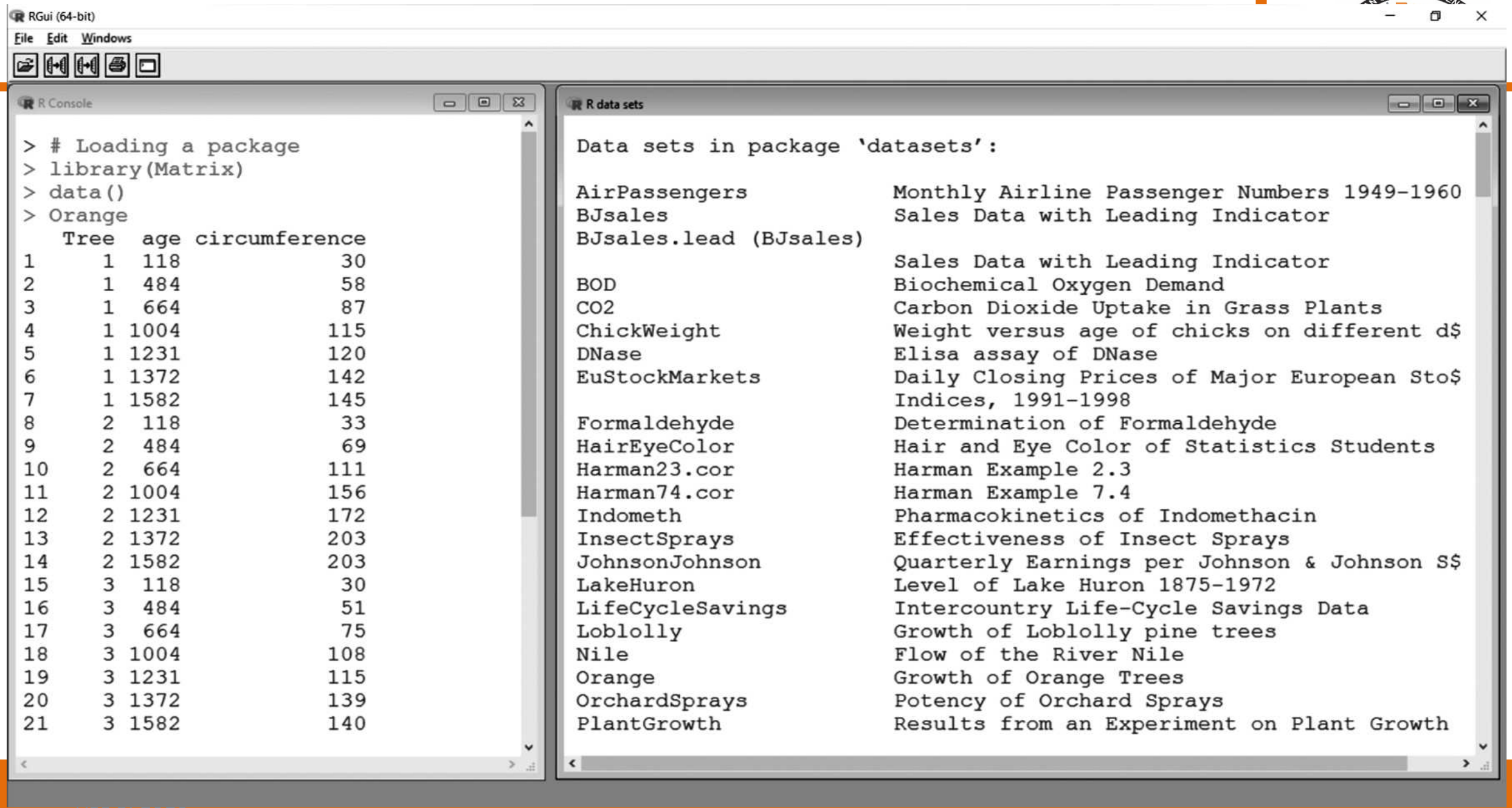
library() Function : loads packages into the R workspace. It is compulsory to import the package before reading the available dataset of that package. The syntax of the library() function is:

library(packagename)

Where, packagename argument is the name of the package to be read.

data() Function: lists all the available datasets of the loaded package into the R workspace. For loading a new dataset into the loaded packages, users need to pass the name of the new dataset into data() function. The syntax of the data() function is: **data(datasetname)**

Where, datasetname argument is the name of the dataset to be read.


RGui (64-bit)
File Edit Windows
R Console
> # Loading a package
> library(Matrix)
> data()
> Orange
Tree age circumference
1 1 118 30
2 1 484 58
3 1 664 87
4 1 1004 115
5 1 1231 120
6 1 1372 142
7 1 1582 145
8 2 118 33
9 2 484 69
10 2 664 111
11 2 1004 156
12 2 1231 172
13 2 1372 203
14 2 1582 203
15 3 118 30
16 3 484 51
17 3 664 75
18 3 1004 108
19 3 1231 115
20 3 1372 139
21 3 1582 140
R data sets
Data sets in package 'datasets':
AirPassengers Monthly Airline Passenger Numbers 1949-1960
BJsales Sales Data with Leading Indicator
BJsales.lead (BJsales)
Sales Data with Leading Indicator
BOD Biochemical Oxygen Demand
CO2 Carbon Dioxide Uptake in Grass Plants
ChickWeight Weight versus age of chicks on different d$
DNase Elisa assay of DNase
EuStockMarkets Daily Closing Prices of Major European Sto$
Indices, 1991-1998
Formaldehyde Determination of Formaldehyde
HairEyeColor Hair and Eye Color of Statistics Students
Harman23.cor Harman Example 2.3
Harman74.cor Harman Example 7.4
Indometh Pharmacokinetics of Indomethacin
InsectSprays Effectiveness of Insect Sprays
JohnsonJohnson Quarterly Earnings per Johnson & Johnson S$
LakeHuron Level of Lake Huron 1875-1972
LifeCycleSavings Intercountry Life-Cycle Savings Data
Loblolly Growth of Loblolly pine trees
Nile Flow of the River Nile
Orange Growth of Orange Trees
OrchardSprays Potency of Orchard Sprays
PlantGrowth Results from an Experiment on Plant Growth

Reading Data from Web/APIs

Nowadays most business organisations are using the Internet and cloud services for storing data. This online dataset is directly accessible through packages and application programming interfaces (APIs).

Different packages are available in R for reading from online datasets.

Packages	*Description*	*Download Link*
RCurl	The package permits download of files from the web server and post forms.	https://cran.r-project.org/web/packages/RCurl/index.html
Google Prediction API	It allows uploading of data to Google storage and then training them for Google Prediction API.	http://code.google.com/p/r-google-predictionapi-v121
Infochimps	The package provides the functions for accessing all API.	http://api.infochimps.com
HttpRequest	The package reads the web data with the help of an HTTP request protocol and implements the GET, POST request.	https://cran.r-project.org/web/packages/httpRequest/index.html
WDI	The package reads all World Bank data.	http://cransprojectorg/web/packages/WD1/index.html
XML	The package reads and creates an XML and HTML document with the help of an HTTP or FTP protocol.	http://cransprojectorg/web/packages/XML/index.html
Quantmod	The package reads finance data from Yahoo finance.	http://crans-projectorg/web/packages/quantmodfindex.html
ScrapeR	The package reads online data.	http://crans-projectorg/web/packages/scrapeR/index.html

The following example illustrates web scraping.

ØWeb scraping extracts data from any webpage of a website.

ØHere package **'RCurl'** is used for web scraping . At first, the package, 'RCurl' is imported into the workspace and then **getURL()** function of the package, 'RCurl' takes the required webpage.

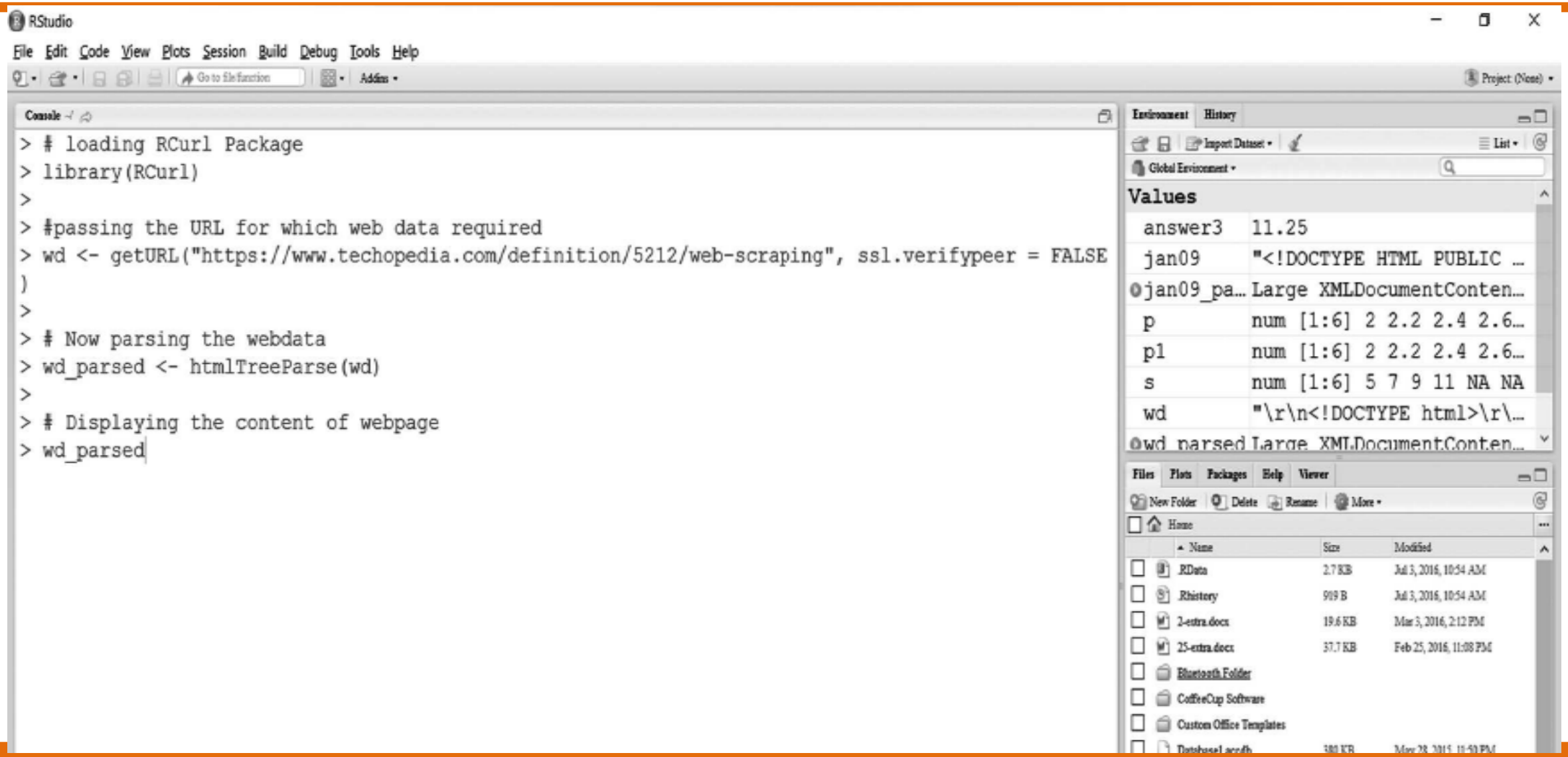
ØNow **htmlTreeParse() function parses** the content of the webpage.

Reading a JSON (Java Script Object Notation) Document

Step 1: Install rjson package.

> **install.packages("rjson")**

Installing package into 'C:/Users/seema_acharya/Documents/R/winlibrary/ 3.2'(as 'lib' is unspecified) trying URL 'https://cran.hafro.is/bin/windows/contrib/3.2/rjson_0.2.15.zip' Content type 'application/zip' length 493614 bytes (482 KB) downloaded 482 KB package 'rjson' successfully unpacked and MD5 sums checked


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Console
> # loading RCurl Package
> library(RCurl)
>
> #passing the URL for which web data required
> wd <- getURL("https://www.techopedia.com/definition/5212/web-scraping", ssl.verifypeer = FALSE
)
>
> # Now parsing the webdata
> wd_parsed <- htmlTreeParse(wd)
>
> # Displaying the content of webpage
> wd_parsed
Environment History
Import Dataset
List
Global Environment
Values
answer3 11.25
jan09 "<!DOCTYPE HTML PUBLIC …
jan09_pa… Large XMLDocumentConten…
p num [1:6] 2 2.2 2.4 2.6…
p1 num [1:6] 2 2.2 2.4 2.6…
s num [1:6] 5 7 9 11 NA NA
wd "\r\n<!DOCTYPE html>\r\…
wd_parsed Large XMLDocumentConten…
Files Plots Packages Help Viewer
New Folder Delete Rename More
Home
Name Size Modified
.RData 2.7 KB Jul 3, 2016, 10:54 AM
.Rhistory 919 B Jul 3, 2016, 10:54 AM
2-extra.docx 19.6 KB Mar 3, 2016, 2:12 PM
25-extra.docx 37.7 KB Feb 25, 2016, 11:08 PM
Bluetooth Folder
CoffeeCup Software
Custom Office Templates

Step 2: Input data.

Store the data given below in a text file (‘D:/Jsondoc.json’). Ensure that the file is saved with an extension of .json

```
{
‘EMPID’:[‘1001’,’2001’,’3001’,’4001’,’5001’,’6001’,’7001’,’8001’
],
‘Name’:[‘Ricky’,’Danny’,’Mitchelle’,’Ryan’,’Gerry’,’Nonita’,’Sim
on’,’Gallop’ ],
‘Dept’: [‘IT’,’Operations’,’IT’,’HR’,’Finance’,’IT’,’Operations’
,’Finance’]
}
```

A JSON document begins and ends with a curly brace ({}). A JSON document is a set of key value pairs. Each key:value pair is delimited using ‘,’ as a delimiter

Step 3: Read the JSON file, ‘d:/Jsondoc.json’.

```
> output <- fromJSON(file = “d:/Jsondoc.json”)
> output
$EMPID
[1] “1001” “2001” “3001” “4001” “5001” “6001” “7001” “8001”
$Name
[1] “Ricky” “Danny” “Mitchelle” “Ryan” “Gerry” “Nonita”
[7] “Simon” “Gallop”
$Dept
[1] “IT” “Operations” “IT” “HR” “Finance”
[6] “IT” “Operations” “Finance”
```

Step 4: Convert JSON to a data frame.

> JSONDataFrame <- as.data.frame(output)

Display the content of the data frame, ‘output’.

> JSONDataFrame

EMPID Name Dept

1 1001 Ricky IT

2 2001 Danny Operations

3 3001 Mitchelle IT

4 4001 Ryan HR

5 5001 Gerry Finance

6 6001 Nonita IT

7 7001 Simon Operations

8 8001 Gallop Finance

Reading an XML File

Step 1: Install an XML package.

> install.packages("XML")

Installing package into 'C:/Users/seema_acharya/Documents/R/winlibrary/ 3.2'(as 'lib' is unspecified)

trying URL 'https://cran.hafro.is/bin/windows/contrib/3.2/XML_3.98-1.3.zip' Content type 'application/zip' length 4299803 bytes (4.1 MB)

downloaded 4.1 M package 'XML' successfully unpacked and MD5 sums checked

Step 2: Input data.

Store the data below in a text file (XMLFile.xml in the D: drive). Ensure that the file is saved with an extension of .xml.

```
<RECORDS>
   <EMPLOYEE>
      <EMPID>1001</EMPID>
      <EMPNAME>Merrilyn</EMPNAME>
      <SKILLS>MongoDB</SKILLS>
      <DEPT>Computer Science</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <EMPID>1002</EMPID>
      <EMPNAME>Ramya</EMPNAME>
      <SKILLS>People Management</SKILLS>
      <DEPT>Human Resources</DEPT>
   </EMPLOYEE>

   <EMPLOYEE>
      <EMPID>1003</EMPID>
      <EMPNAME>Fedora</EMPNAME>
      <SKILLS>Recruitment</SKILLS>
      <DEPT>Human Resources</DEPT>
   </EMPLOYEE>
</RECORDS>
```

Reading an XML File: The xml file is read in R using the function xmlParse(). It is stored as a list in R.

Step 1: Begin by loading the required packages.

```
> library("XML")
Warning message:
package 'XML' was built under R version 3.2.3
> library ("methods")
> output <- xmlParse(file = "d:/XMLFile.xml")
> print(output)
```

```
<?xml version="1.0"?>
<RECORDS>
     <EMPLOYEE>
          <EMPID>1001</EMPID>
          <EMPNAME>Merrilyn</EMPNAME>
          <SKILLS>MongoDB</SKILLS>
          <DEPT>ComputerScience</DEPT>
     </EMPLOYEE>
     <EMPLOYEE>
          <EMPID>1002</EMPID>
          <EMPNAME>Ramya</EMPNAME>
          <SKILLS>PeopleManagement</SKILLS>
          <DEPT>HumanResources</DEPT>
     </EMPLOYEE>
     <EMPLOYEE>
          <EMPID>1003</EMPID>
          <EMPNAME>Fedora</EMPNAME>
          <SKILLS>Recruitment</SKILLS>
          <DEPT>HumanResources</DEPT>
     </EMPLOYEE>
</RECORDS>
```

Step 2: Extract the root node from the XML file.

```
> rootnode <- xmlRoot(output)
```

Find the number of nodes in the root.

```
> rootsize <- xmlSize(rootnode)
> rootsize
[1] 3
```

Let us display the details of the first node.

```
> print (rootnode[1])
$EMPLOYEE
<EMPLOYEE>
<EMPID>1001</EMPID>
<EMPNAME>Merrilyn</EMPNAME>
<SKILLS>MongoDB</SKILLS>
<DEPT>ComputerScience</DEPT>
</EMPLOYEE>
attr(, “class”)
[1] “XMLInternalNodeList” “XMLNodeList”
```

Let us display the details of the first element of the first node.

```
> print(rootnode[[1]][[1]])
<EMPID>1001</EMPID>
```

Let us display the details of the third element of the first node.

```
> print(rootnode[[1]][[3]])
<SKILLS>MongoDB</SKILLS>
```

Next, display the details of the third element of the second node.

```
> print(rootnode[[2]][[3]])
<SKILLS>PeopleManagement</SKILLS>
```

We can also display the value of 2nd element of the first node.

```
> output <-xmlValue(rootnode[[1]][[2]])
> output
[1] “Merrilyn”
```

Step 3: Convert the input xml file to a data frame using the xmlToDataFrame function.

> xmldataframe <- xmlToDataFrame("d:/XMLFile.xml")

Display the output of the data frame.

> xmldataframe

EMPID EMPNAME SKILLS DEPT

1 1001 Merrilyn MongoDB ComputerScience

2 1002 Ramya PeopleMananement HumanResources

3 1003 Fedora Recruitment HumanResources

Comparison of R GUIs for data input

- R is mainly used for statistical analytical data processing.

- Analytical data processing needs a large dataset that is stored in a tabular form. Sometimes it is difficult to use inbuilt functions of R for doing such analytical data processing operations in R console.

- Hence, to overcome this problem, GUI is developed for R. Graphical user interface is a graphical medium through which users interact with the language or perform operations.

- Different GUIs are available for data input in R.

- Each GUI has its own features.

GUI Name	Description	Download Weblink
RCommander (Rcmdr)	• RCommander was developed by John Fox and licensed under the GNU public license. • It comes with many plug-ins and has a very simple interface. • Users can install it like other packages of R within language.	http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/ Or https://cran.r-project.org/web/packages/Rcmdr/index.html
Rattle	• Dr. Graham Williams developed the Rattle GUI package written in R. • Data mining operation is the main application area of Rattle. • It offers statistical analysis, validation, testing and other operations.	http://rattle.togaware.com/ Or http://rattle.togaware.com/rattle-install-mswindows.html
RKWard	• RKWard community developed the RKWard package. • It provides a transparent front end and supports different features for doing analytical operations in R. • It supports different platforms, such as Windows, Linux, BSD, and OS X.	https://rkward.kde.org/ Or http://download.kde.org/stable/rkward/0.6.5/win32/install_rkward_0.6.5.exe
JGR (Java GUI for R)	• Markus Helbig, Simon Urbanek, and lan Fellows developed JGR. • JGR is a universal GUI for R that supports cross platform. • Users can use it as a replacement for the default R GUI on Windows.	http://www.rforge.net/JGR/ Or https://cran.r-project.org/web/packages/JGR/
Deducer	• Deducer is another simple GUI that has a menu system for doing common data operations, analytical processing and other operations. • It is mainly designed to use it with the Java-based R Console [JGR].	http://www.deducer.org/pmwiki/pmwiki.php?n=Main.DeducerManual Or http://www.deducer.org/pmwiki/index.php?n=Main.DownloadingAndInstallingDeducer

Using R With Databases

ØRODBC is a package of languages that interacts with a database. Michael Lapsley and Brian Ripley developed this package.

ØRODBC helps in accessing databases such as MS Access and Microsoft SQL Server through an ODBC interface. Its package has many inbuilt functions for performing database operations on the database.

Major functions of RODBC

Function	*Description*
`odbcConnect(dsn, uid= '', pwd= '')` where, dsn is domain name server, uid is the user ID and pwd is the password.	The function opens a connection to an ODBC database.
`sqlFetch(sqltable)` where, sqltable is name of the SQL table.	The function reads a table from an ODBC database into a data frame.
`sqlQuery(query)` where, query is the SQL query.	The function takes a query, sends to an ODBC database and returns its result.
`sqlSave(dataframe, tablename= 'sqltable')` where, data frame defines the data frame object and tablename argument is the name of the table.	The function writes or updates a data frame to a table in the ODBC database.
`sqlDrop(sqltable)` where, sqltable is the name of the SQL table.	The function removes a table from the ODBC database.
`odbcclose()`	The function closes the open connection.

Here is a sample code where package RODBC is used for reading data from a database.

```
>#  importing package
> library(RODBC)
> connect1 <- odbcConnect(dsn = 'servername', uid= '      '   , pwd= '      ')
#Open connection
> query1 <- 'Select * from lib.table where…'
> Demodb <- sqlQuery(connect1, query1, errors = TRUE)
> odbcClose(connection) #Close the connection
```

Using MySQL and R

- MySQL is an open source SQL database system.
- It is a small-sized popular database that is available for free download.
- For accessing MySQL database, users need to install the MySQL database system on their computers.
- MySQL database can be downloaded and installed from its official website.
- R also provides a package, 'RMySQL' used for accessing the database from the MySQL
- database. Like other packages, RMySQL has many inbuilt functions for interacting with
- a database.

Major functions of RMySQL

Function	*Description*
`dbConnect(MySQL(), uid= '', pwd= '', dbname = '',...)` where, `MySQL()` is MySQL driver, uid is the user ID, pwd is the password and dbname is the database name.	The function opens a connection to the MySQL database.
`dbDisconnect(connectionname)` where, Connectionname defines the name of the connection.	The function closes the open connection.
`dbSendQuery(connectionname, sql)` where, connectionname defines the name of the connection.	The function runs the SQL queries of the open connection.
`dbListTables(connectionname)` where, connectionname defines the name of the connection.	The function lists the tables of the database of the open connection.
`dbWriteTable(connectionname, name = 'table name', value = data.frame.name)` where, connectionname defines the name of the connection.	The function creates the table and alternatively writes or updates a data frame in the database.

A sample code to illustrate the use of RMySQL for reading data from a database is given below.

```
>#  importing package
>library(RMySQL)
#Open connection ‘connectm’
> connectm <- odbcConnect(MySQL(), uid= ‘’, pwd= ‘’,dbname = ‘’, host = ‘’)
> querym <- ‘Select * from lib.table where…’
> Demom<- dbSendQuery(connectm, querym)
#Close the connection ‘connect’
>dbDisconnect(connectm)
```

Using PostgreSQL and R

- PostgreSQL is an open source and customisable SQL database system.
- After MySQL, PostgreSQL database is used for business analytical processing.
- For accessing the PostgreSQL database, users need to install the PostgreSQL database system on their computer system.
- Please note that it requires a server.
- Users can get a server on rent, download and install the MySQL database from its official website.
- R has a package, 'RPostgreSQL' that is used for accessing the database from the PostgreSQL database.
- Like other packages, RPostgreSQLhas many inbuilt functions for interacting with its database.

Major functions of the RPostgreSQL

Function	*Description*
`dbConnect(driverobject, uid= '', pwd= '', dbname = '',…)` where, driverobject is an object of database driver, uid is the user ID, pwd is the password and dbname is the database name.	The function opens a connection to an RPostgreSQL database.
`dbDisconnect(connectionname)` where, Connectionname defines the name of the connection.	The function closes the open connection.

Using SQLite and R

- SQLite is a server-less, self-contained, transactional and zero-configuration SQL database system.
- It is an embedded SQL database engine that does not require any server, due to
- which it is called a serverless database.
- The database also supports all business analytical data processing.
- R has an RSQLite package that is used for accessing a database from the SQLite
- database.
- The RSQLite5 has many inbuilt functions for working with the database.
- Like other packages used for accessing a database, as explained in the previous sections, users can use the same methods—dbconnect() and dbDisconnect() for opening and closing the connection from the SQLite database, respectively.
- The only difference here is that users have to pass the SQLite database driver object in the dbConnect() function.

Using JasperDB and R

- JasperDB is another open source database system integrated with R.
- It was developed by the Jaspersoft community. It provides many business intelligence tools for analytical business processing.
- A Java library interface is used between JasperDB and R. It is called 'RevoConnectR for JasperReports Server'.
- The dashboard of the JasperReports Server provides many features through which R charts, an output of the RevoDeploy R, etc., are easily accessible.
- Like other packages, JasperDB has a package or web service framework called 'RevoDeployR' developed by Revolution Analytics.
- RevoDeploy R6 provides a set of web services with security features, scripts, APIs and libraries in a single server.
- It easily integrates with the dynamic R-based computations into web applications

Using Pentaho and R

Ø Pentaho is one of the most famous companies in the data integration field that develops different products and provides services for big data deployment and business analytics.
Ø The company provides different open source-based and enterprise-class platforms.
Ø Pentaho Data Integration (PDI) is one of the products of Pentaho7 used for accessing database and
Ø analytical data processing. It prepares and integrates data for creating a perfect picture
Ø of any business.
Ø The tool provides accurate and analytics-ready data reports to the end users, eliminates the coding complexity and uses big data in one place.
Ø R Script Executor is one of the inbuilt tools of the PDI tool for establishing a relationship between R and Pentaho Data Integration.
Ø Through R Script Executor, users can access data and perform analytical data operations. If users have R in their system already, then they just need to install PDI from its official website.
Ø The users need to configure environment variables, Spoon, DI Server, and Cluster nodes as well.
Ø Although users can try PDI and transform a database using R Script Executor, PDI is a paid tool for doing analytical data integration operation.

Thank You!

Topic: Introduction to R Language

Session no: 2

Course name: DATA ANALYTICS USING R

Semester: VII

Regulation : R18

Presented by

Loading and Handling Data in R

Challenges of Analytical Data Processing
- Expression, Variables and Functions
- Missing Values Treatment in R
- Using the ‘as’ Operator to Change the Structure of Data,
- Vectors
- Matrices

Factors
- List

Aggregating and Group Processing of a Variable,
- Simple Analysis Using R,
- Methods for Reading Data,

Comparison of R GUIs for Data Input

List : List is similar to C Struct.

S.No	Command	Example
1	Create a list in R. To create a list, 'emp' having three elements, 'EmpName', 'EmpUnit' and 'EmpSal'.	Øemp <- list ("EmpName="Alex", EmpUnit = "IT", EmpSal = 55000) > emp $EmpName [1] "Alex" $EmpUnit [1] "IT"
2	Emp list	EmpList <- list("Alex", "IT", 55000) EmpList [[1]] [1] "Alex" [[2]] [1] "IT" [[3]] [1] 55000

A list has elements. The elements in a list can have names, which are referred to as tags.
Elements can also have values.

S.No	Command	Example
1	Retrieve the names of the elements in the list 'emp'.	> names(emp) [1] "EmpName" "EmpUnit" "EmpSal"
2	Retrieve the values of the elements in the list 'emp'.	> unlist(emp) EmpName EmpUnit EmpSal "Alex" "IT" "55000"
3	Retrieve the value of the element 'EmpName' in the list 'emp'.	> unlist(emp["EmpName"]) EmpName "Alex"
4	Retrieve the value of the element 'EmpName' in the list 'emp'.	> emp[["EmpName"]] [1] "Alex

List : List is similar to C Struct.

	Add/Delete Element to or from a List	
1	Before adding an element to the list 'emp', let us verify what elements exist in the list.	> emp $EmpName [1] "Alex" $EmpUnit [1] "IT" $EmpSal [1] 55000
2	Add an element with the name 'EmpDesg' and value 'Software Engineer' to the list, 'emp'. > emp$EmpDesg = "Software Engineer"	> emp $EmpName [1] "Alex" $EmpUnit [1] "IT" $EmpSal [1] 55000 $EmpDesg [1] "Software Engineer"

List : List is similar to C Struct.

	Add/Delete Element to or from a List	
1	Delete an element with the name 'EmpUnit' and value 'IT' from the list, 'emp'. > emp$EmpUnit <- NULL	> emp $EmpName [1] "Alex" $EmpSal [1] 55000 $EmpDesg [1] "Software Engineer"
2	Determine the number of elements in the list, 'emp' **length() function** can be used to determine the number of elements present in the list. The list, 'emp' has three elements as shown:	> length(emp) [1] 3

	Recursive List: A recursive list means a list within a list. Create a list within a list. Let us begin with two lists, 'emp' and 'emp1'. The elements in both the lists are as shown below. We would like to combine both the lists into a single list called 'EmpList'. > EmpList <- list(emp, emp1)	> emp $EmpName [1] "Alex" 78 Data Analytics using R $EmpSal [1] 55000 $EmpDesg [1] "Software Engineer" > emp1 $EmpUnit [1] "IT" $EmpCity [1] "Los Angeles"

Task

TABLE 3.4 Functions for exploring a dataset

Functions	*Function Arguments*	*Description*
`names(dataset)`	• Dataset argument contains the name of the dataset.	The function displays the variables of the given dataset.
`summary(dataset)`	• Dataset argument contains the name of the dataset.	The function displays the summary of the given dataset.
`str(dataset)`	• Dataset argument contains the name of the dataset.	The function displays the structure of the given dataset.
`head(dataset, n)`	• Dataset argument contains the name of the dataset. • *n* is a numeric value to display the number of top rows.	The function displays the top rows according to the value of n. If the value of n is not provided in the function then by default the function displays the top 6 rows of the dataset.
`tail(dataset, n)`	• Dataset argument contains the name of the dataset. • *n* is a numeric value to display the number of bottom rows.	The function displays the top rows according to the value of n. If the value of n is not provided in the function then by default the function displays the bottom 6 rows of the dataset.
`class(dataset)`	• Dataset argument contains the name of the dataset.	The function displays the class of the dataset.
`dim(dataset)`	• Dataset argument contains the name of the dataset.	The function returns the dimension of the dataset which implies the total number of rows and columns of the dataset.
`table(dataset$variable names)`	• Dataset argument contains the name of the dataset. • Variable name contains the name of the variable names.	The function returns the number of categorical values after counting them.

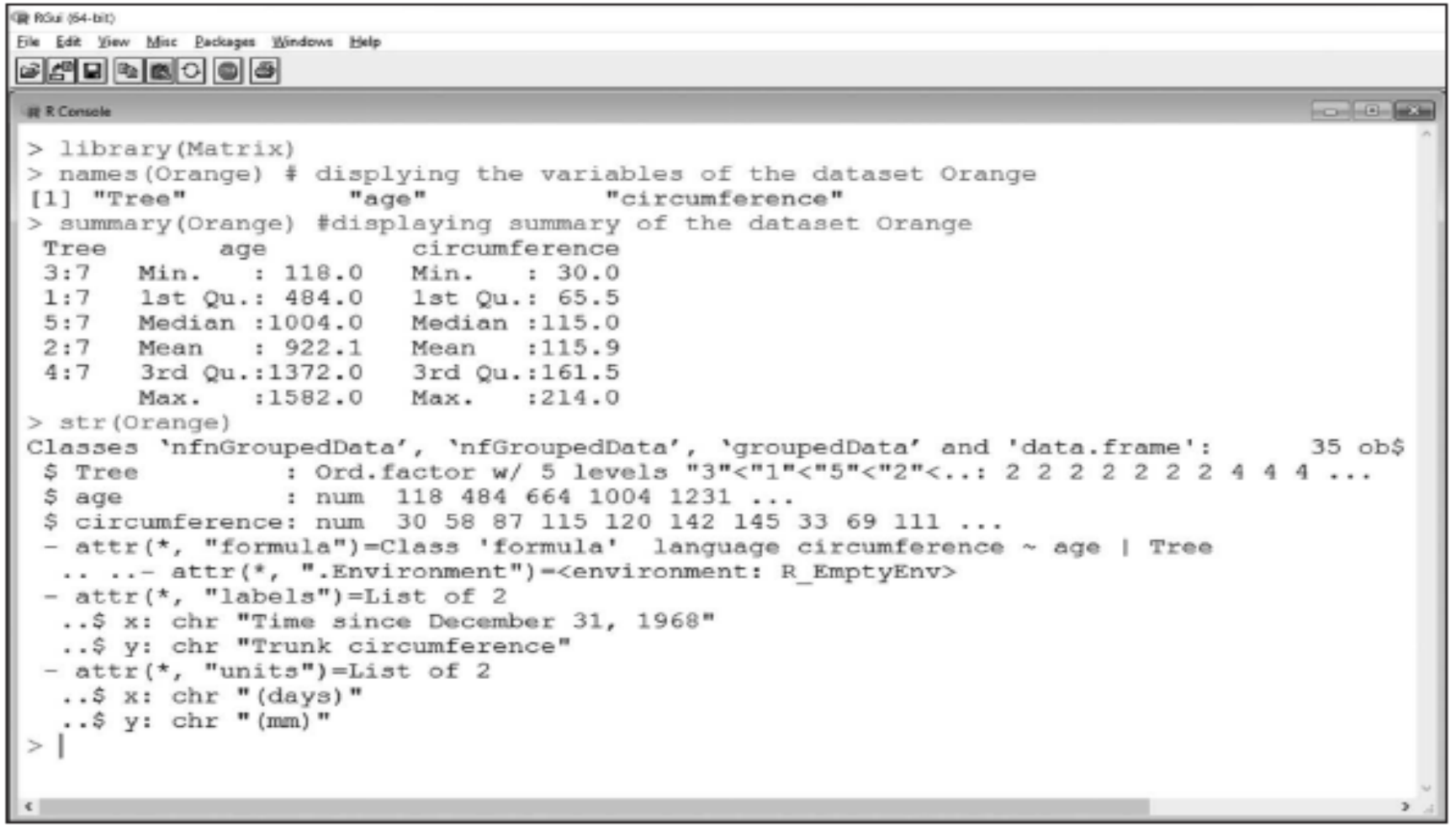


FIGURE 3.13 *Exploring a dataset using* `names()`, `summary()` *and* `str()` *functions*

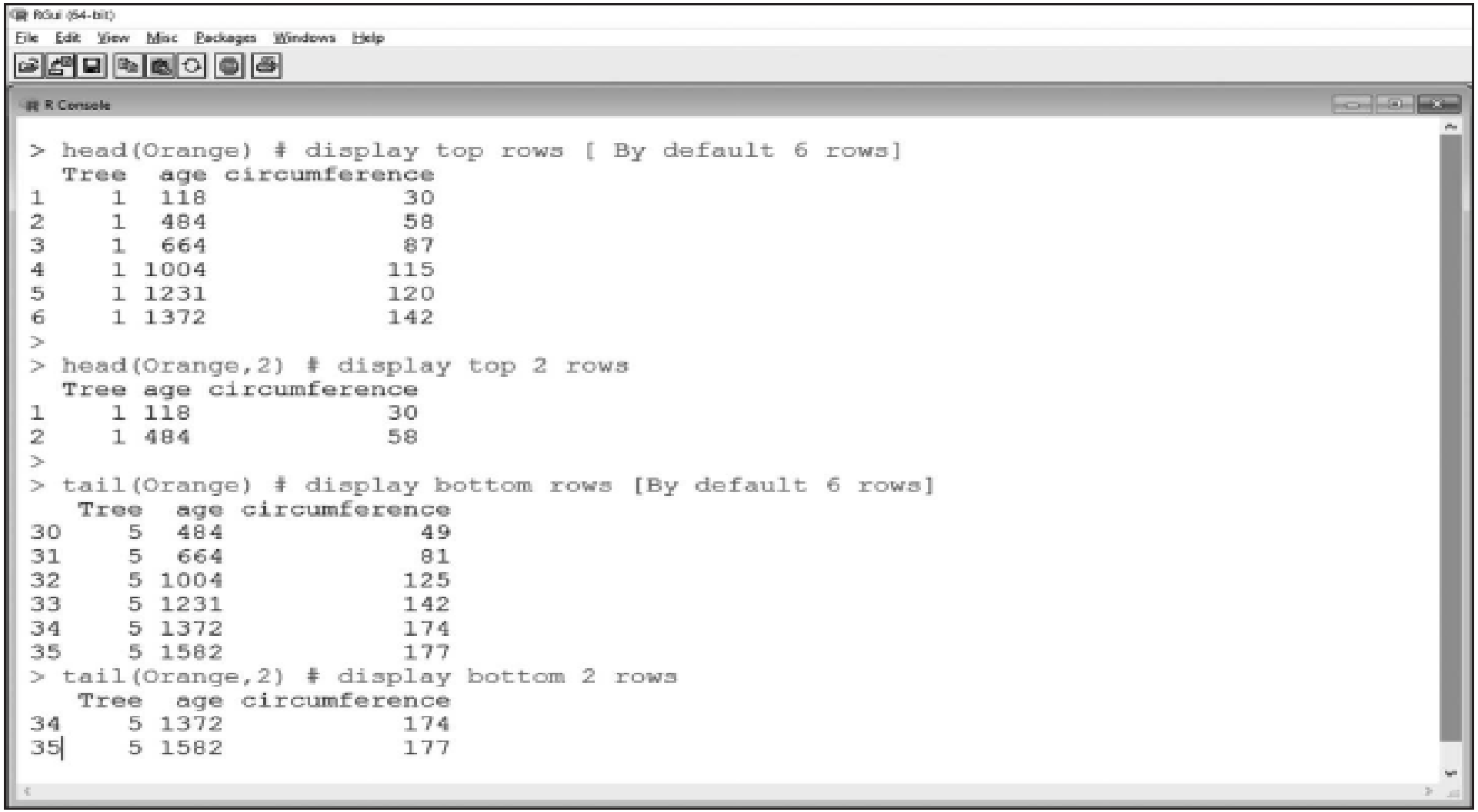


FIGURE 3.14 *Exploring a dataset using* `head()` *and* `tail()` *functions*

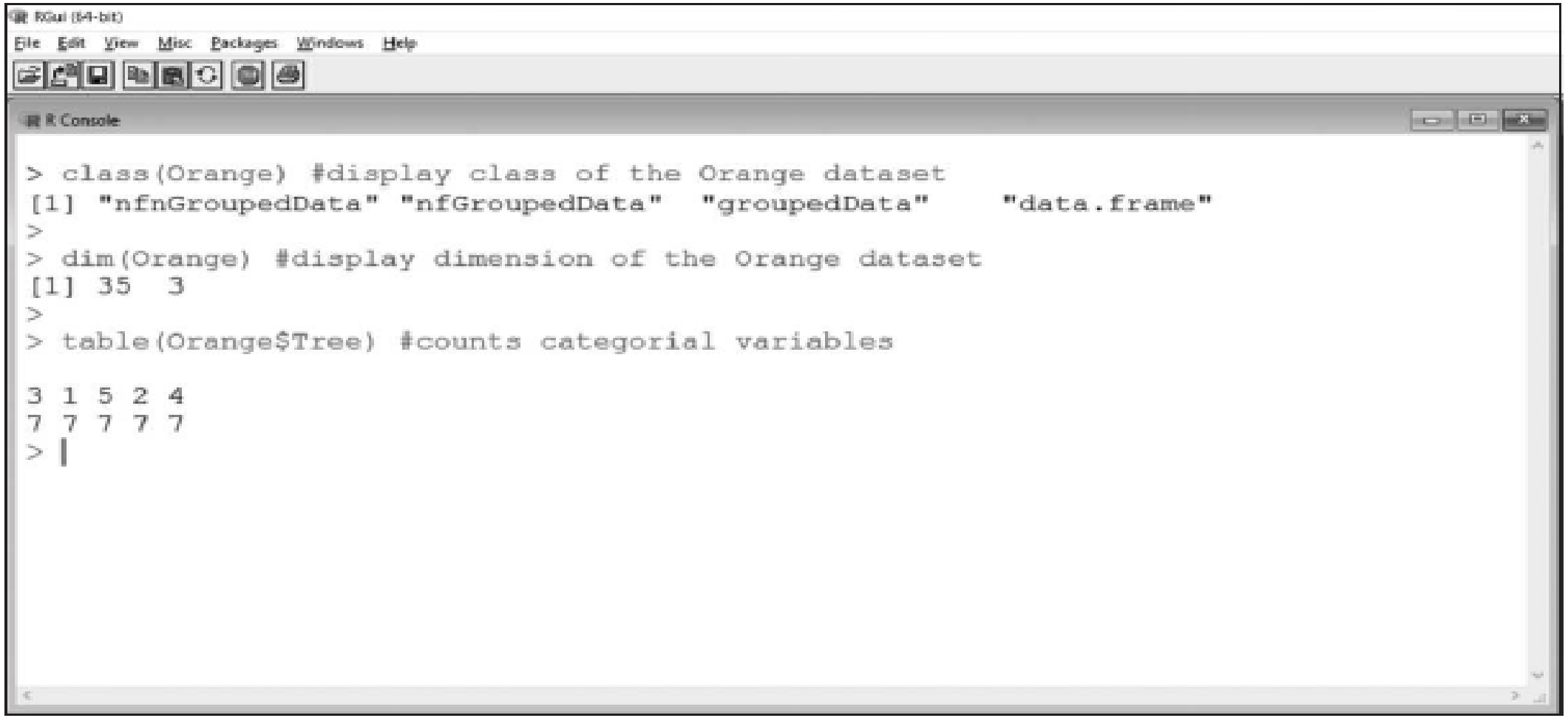


FIGURE 3.15 *Exploring a dataset using* **`class()`**, **`dim()`** *and* **`table()`** *functions*

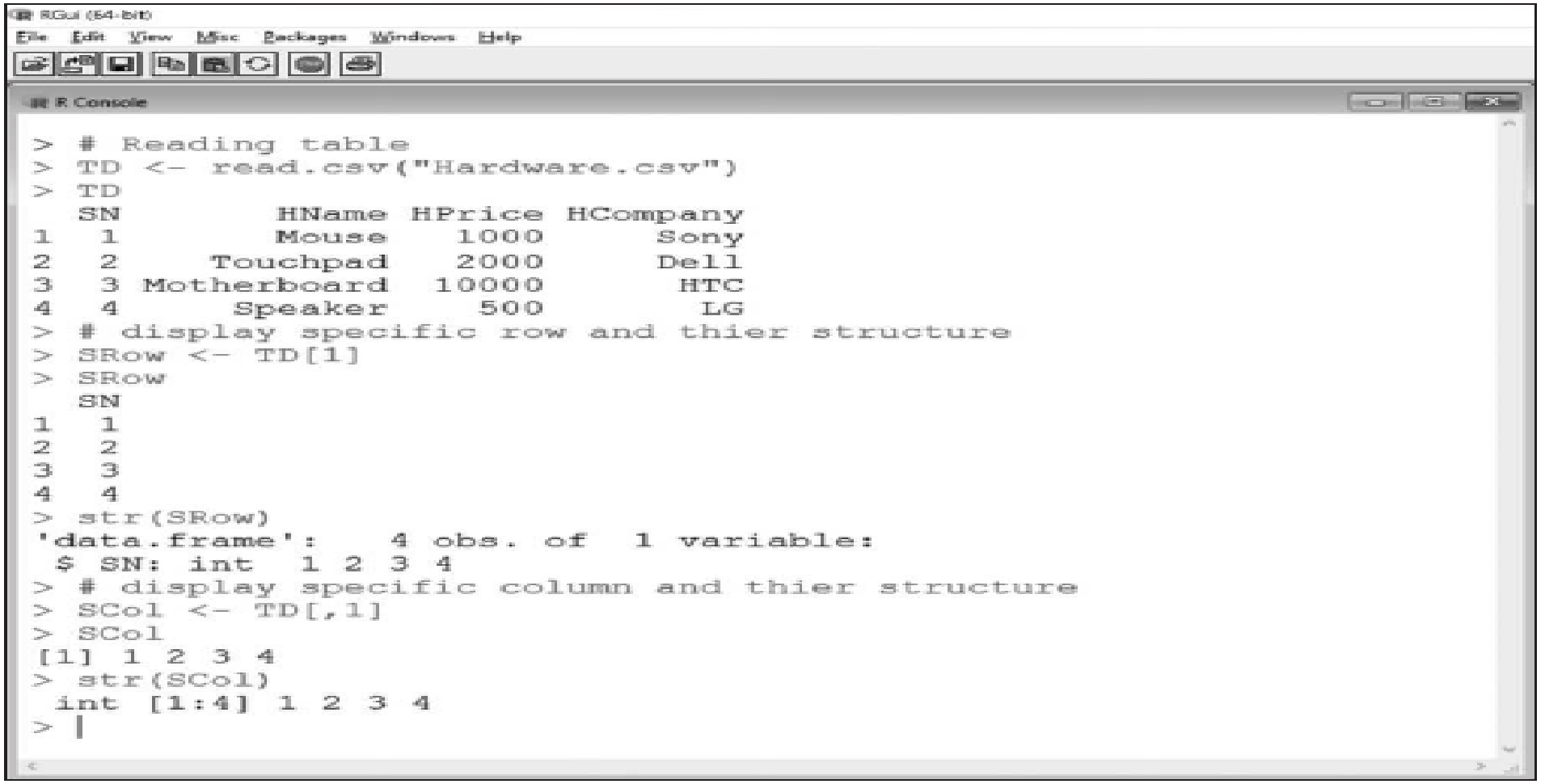
- Analytical data processing sometimes may require specific rows and columns of a dataset.

TABLE 3.5 Commands for accessing specific rows and columns of a dataset

Commands	*Command Arguments*	*Description*
`Tablename[n]`	*n* is a numeric value.	The command displays the rows according to the given value of argument *n* of the table.
`Tablename[, n]`	*n* is a numeric value.	The command displays the columns according to the given value of argument *n* of the table.

The following example reads a table, 'Hardware.csv' into object, 'TD' on the R workspace. The TD[1] and TD[, 1] commands displays rows and columns (Figure 3.16).

Merging Data




```
> # Reading table
> TD <- read.csv("Hardware.csv")
> TD
  SN         HName HPrice HCompany
1  1         Mouse   1000      Sony
2  2     Touchpad    2000      Dell
3  3 Motherboard  10000       HTC
4  4       Speaker    500        LG
> # display specific row and thier structure
> SRow <- TD[1]
> SRow
  SN
1  1
2  2
3  3
4  4
> str(SRow)
'data.frame':    4 obs. of  1 variable:
 $ SN: int  1 2 3 4
> # display specific column and thier structure
> SCol <- TD[,1]
> SCol
[1] 1 2 3 4
> str(SCol)
 int [1:4] 1 2 3 4
>
```

Conditional manipulation of a dataset

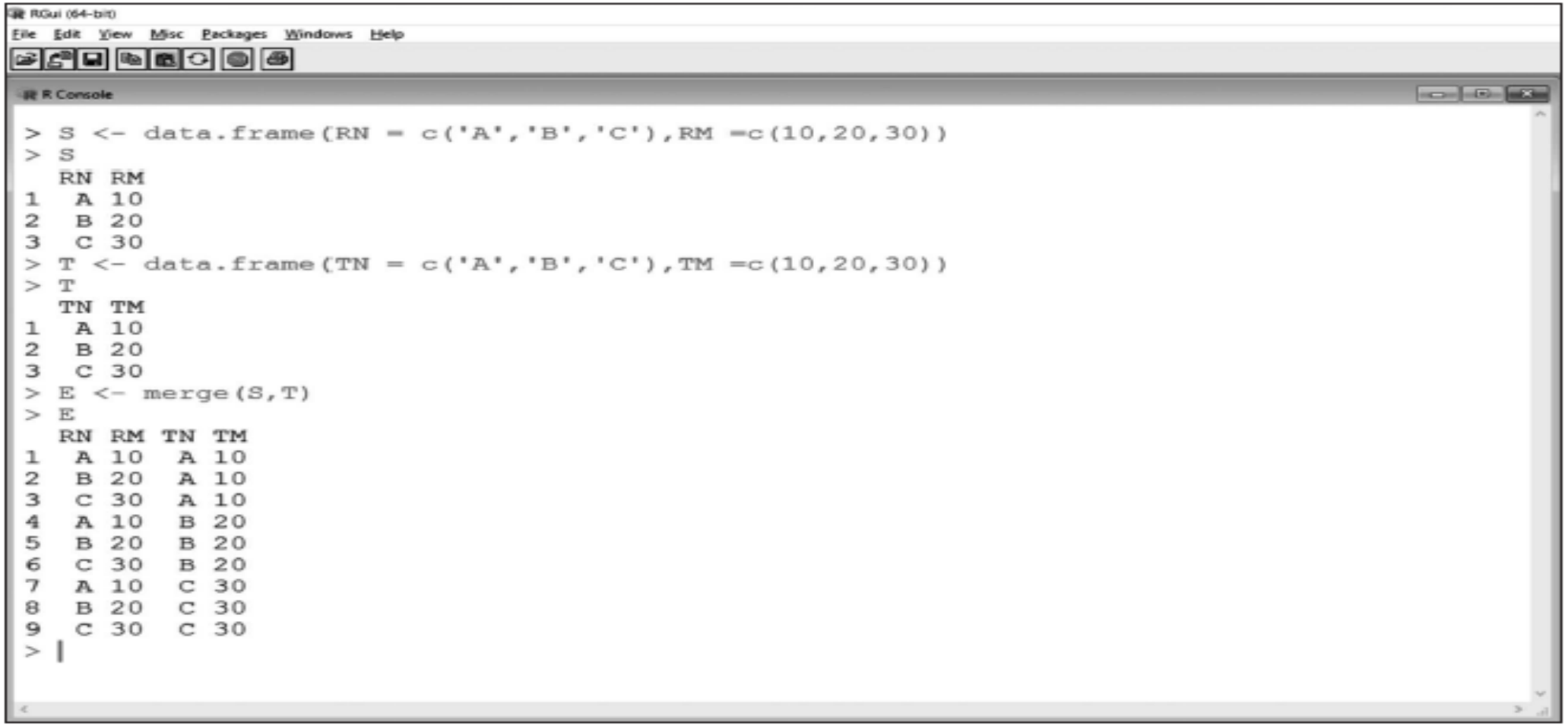


FIGURE 3.17 *Merging data*

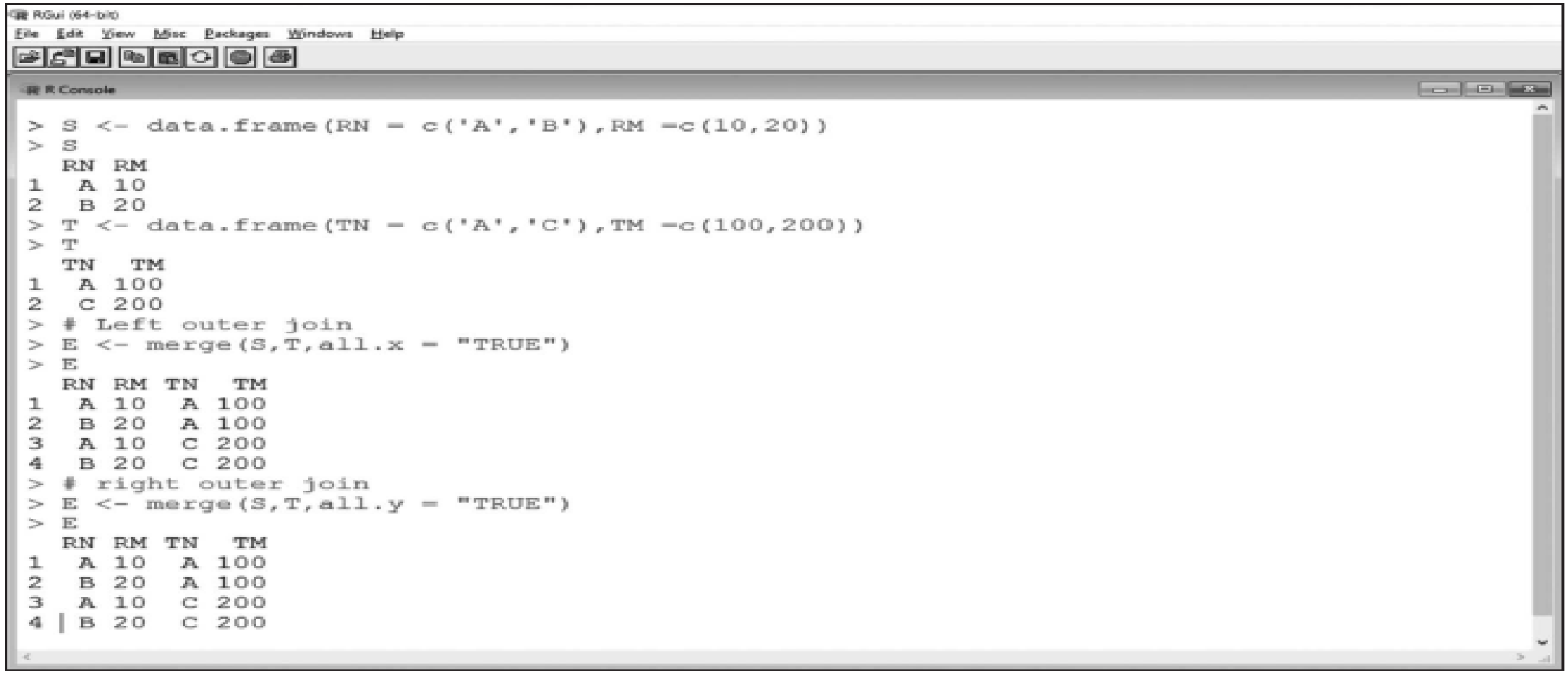


FIGURE 3.18 *Merging data using join condition*

Loading and Handling Data in R

Challenges of Analytical Data Processing
- Expression, Variables and Functions
- Missing Values Treatment in R
- Using the ‘as’ Operator to Change the Structure of Data,
- Vectors
- Matrices

Factors
- List

Aggregating and Group Processing of a Variable,
- Simple Analysis Using R,
- Methods for Reading Data,

Comparison of R GUIs for Data Input

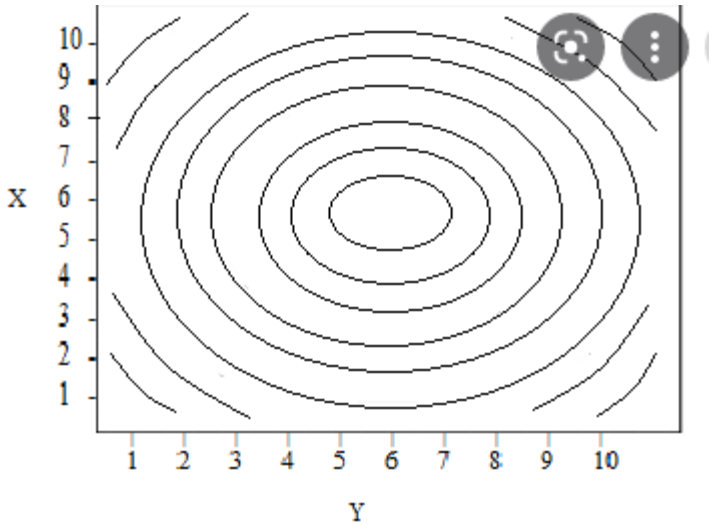
	Objective 1	
1	Let us create a matrix which is 3 rows by 4 columns and set all its elements to 1.	> matrix (1, 3, 4) [, 1] [, 2] [, 3] [, 4] [1,] 1 1 1 1 [2,] 1 1 1 1 [3,] 1 1 1 1
2	Use a vector to create an array, 3 rows high and 3 columns wide. Begin by creating a vector that has elements from 10 to 90 with an interval of 10. Validate by printing the value of vector a. Call the matrix function with vector, 'a' the number of rows and the number of columns.	Øa <- seq(10, 90, by = 10) > a [1] 10 20 30 40 50 60 70 80 90 > matrix (a, 3, 3) [, 1] [, 2] [, 3] [1,] 10 40 70 [2,] 20 50 80 [3,] 30 60 90

	Objective 2	
1	Re-shape the vector itself into an array using the dim function. Begin by creating a vector that has elements from 10 to 90 with an interval of 10	a <- seq (10, 90, by = 10)
2	Validate by printing the value of vector, a.	> a [1] 10 20 30 40 50 60 70 80 90
3	Assign new dimensions to vector, a by passing a vector having 3 rows and 3 columns (c (3, 3)).	> dim(a) <- c(3, 3)
	Print the values of vector, a. You will notice that the values have shifted to form 3 rows by 3 columns. The vector is no longer one dimensional. It has been converted into a two-dimensional matrix that is 3 rows high and 3 columns wide.	> a [, 1] [, 2] [, 3] [1,] 10 40 70 [2,] 20 50 80 [3,] 30 60 90

	Matrix Access	
1	Access the elements of a 3 *4 matrix. Create a matrix, 'mat', 3 rows high and 4 columns wide using a vector.	> x <- 1:12 > x [1] 1 2 3 4 5 6 7 8 9 10 11 12 > mat <- matrix (x, 3, 4) > mat [, 1] [, 2] [, 3] [, 4] [1,] 1 4 7 10 [2,] 2 5 8 11 [3,] 3 6 9 12
2	Access the element present in the second row and third column of the matrix, 'mat'.	> mat [2, 3] [1] 8

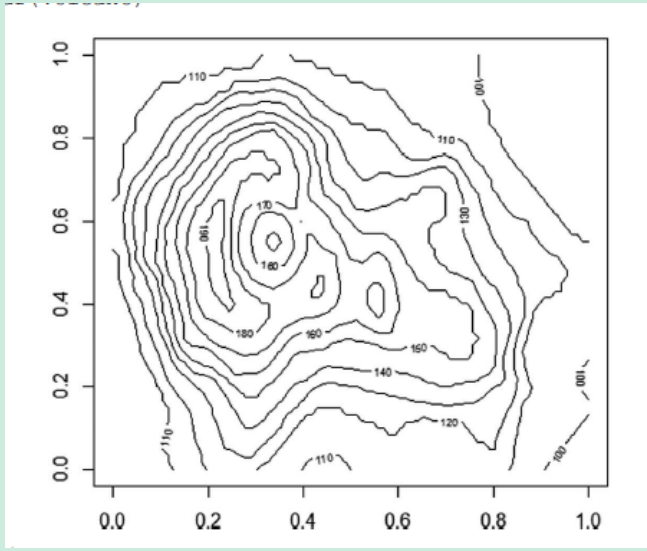
	Matrix Access	
1	Access the third row of an existing matrix. Let us begin by printing the values of an existing matrix, 'mat' To access the third row of the matrix, simply provide the row number and omit the column number. To access the second column of the matrix, simply provide the column number and omit the row number.	> mat [, 1] [, 2] [, 3] [, 4] [1,] 1 4 7 10 [2,] 2 5 8 11 [3,] 3 6 9 12 > mat [3,] [1] 3 6 9 12 > mat[, 2] [1] 4 5 6
2	To access the second and third columns of the matrix, simply provide the column numbers and omit the row number.	> mat[,2:3] [, 1] [, 2] [1,] 4 7 [2,] 5 8 [3,] 6 9

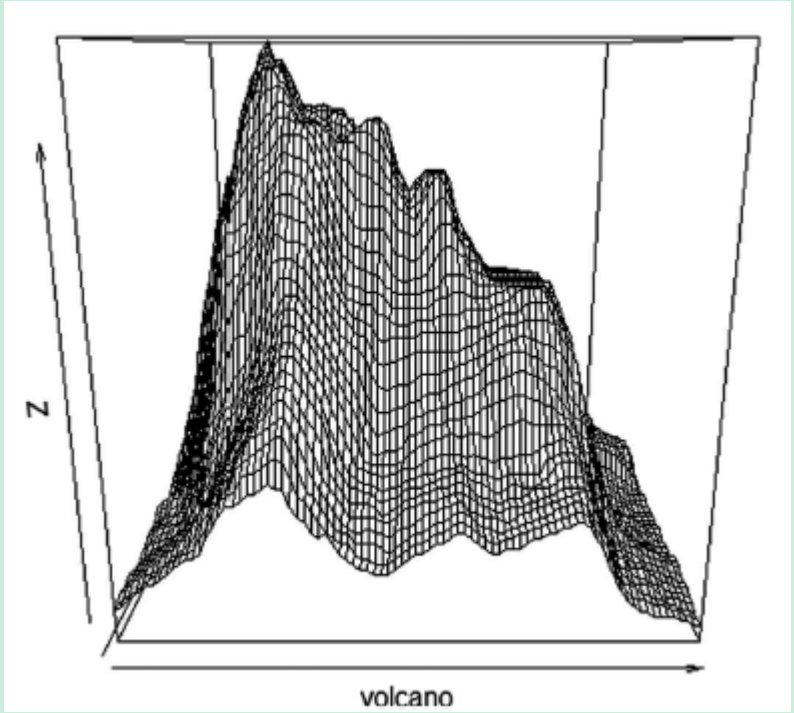
- Contour plots (sometimes called Level Plots) are **a way to show a three-dimensional surface on a two-dimensional plane**.
- It graphs two predictor variables X Y on the y-axis and a response variable Z as contours.
- These contours are sometimes called z-slices or iso- response values

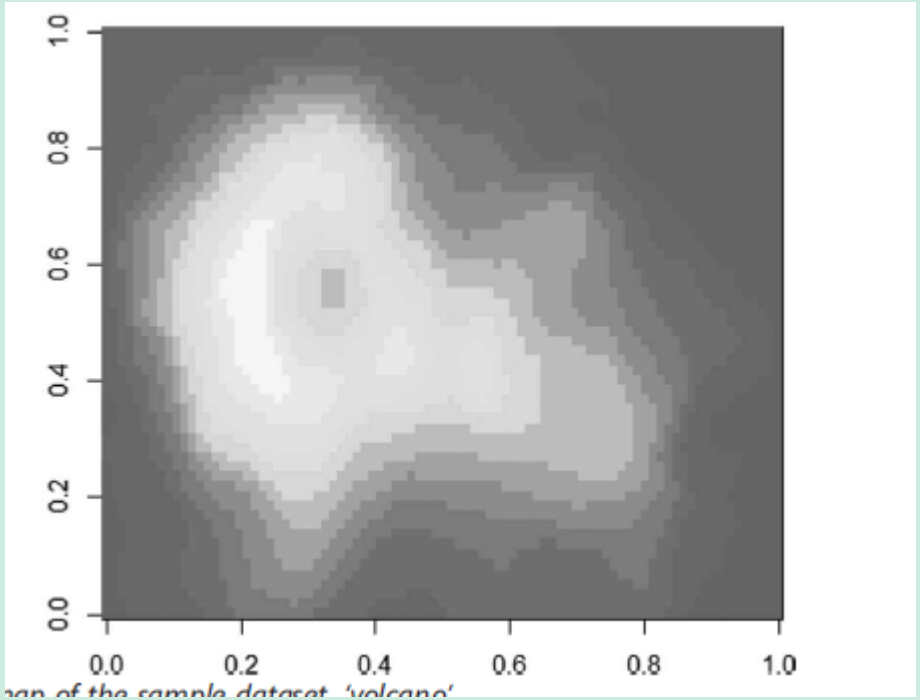


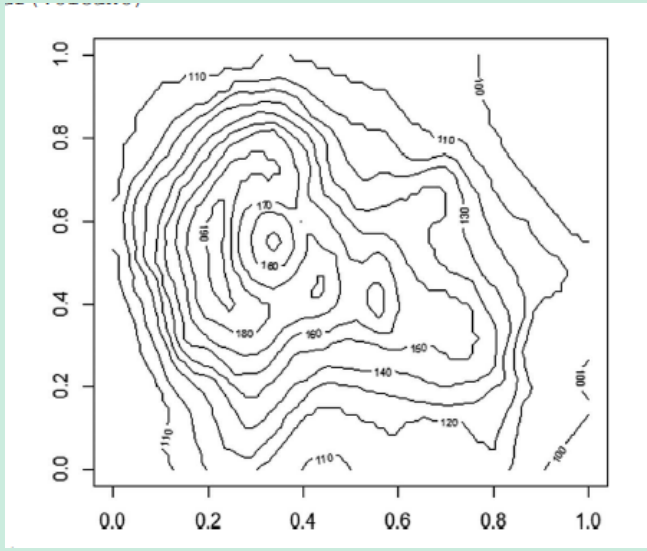
S.NO	Function	Example
	Create a matrix, 'mat' which is 9 rows high and 9 columns wide and assign the value '1' to all its elements.	Ømat <- matrix(1, 9, 9) > mat [, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [1,] 1 1 1 1 1 1 1 1 1 [2,] 1 1 1 1 1 1 1 1 1 [3,] 1 1 1 1 1 1 1 1 1 [4,] 1 1 1 1 1 1 1 1 1 [5,] 1 1 1 1 1 1 1 1 1 [6,] 1 1 1 1 1 1 1 1 1 [7,] 1 1 1 1 1 1 1 1 1

S.NO	Function	Example
	Create a matrix, 'mat' which is 9 rows high and 9 columns wide and assign the value '1' to all its elements.	Ømat <- matrix(1, 9, 9) > mat [, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [1,] 1 1 1 1 1 1 1 1 1 [2,] 1 1 1 1 1 1 1 1 1 [3,] 1 1 0 1 1 1 1 1 1 [4,] 1 1 1 1 1 1 1 1 1 [5,] 1 1 1 1 1 1 1 1 1 [6,] 1 1 1 1 1 1 1 1 1 [7,] 1 1 1 1 1 1 1 1 1

Dataset	Output
R includes some sample data sets. One of these is 'volcano', which is a 3D map of a dormant New Zealand volcano. Create a contour map of the volcano dataset (Figure 3.7). > contour(volcano)	

Dataset	Output
Let us create a 3D perspective map of the sample data set, 'volcano' (Figure 3.8). > persp(volcano)	

Dataset	Output
Create a heat map of the sample dataset, 'volcano' (Figure 3.9). > image(volcano)	

Dataset	Output
R includes some sample data sets. One of these is 'volcano', which is a 3D map of a dormant New Zealand volcano. Create a contour map of the volcano dataset (Figure 3.7). > contour(volcano)	

Factors :

> HouseColor <- c('red', 'green', 'blue', 'yellow', red', 'green', 'blue', 'blue')
> types <- factor(HouseColor)
> HouseColor
[1] "red" "green" "blue" "yellow" "red" "green" "blue" "blue"
> print(HouseColor)
[1] "red" "green" "blue" "yellow" "red" "green" "blue" "blue"
> print (types)
[1] red green blue yellow red green blue blue
Levels: blue green red yellow
Levels denotes the unique values. The above has four distinct values such as 'blue','green', 'red' and 'yellow'.
> as.integer(types)
[1] 3 2 1 4 3 2 1 1
The above output is explained as given below.
1 is the number assigned to blue. 2 is the number assigned to green. 3 is the number assigned to red.
4 is the number assigned to yellow.
> levels(types) [1] "blue" "green" "red" "yellow"