STATISTICAL PROGRAMMING IN R

K G Srinivasa
Associate Professor & Head
Department of Information Technology
Ch. B.P. Government Engineering College
Jaffarpur, Delhi

G M Siddesh
Associate Professor
Department of Information Science & Engineering
Ramaiah Institute of Technology
Bengaluru

Chetan Shetty
Assistant Professor
Department of Computer Science & Engineering
Ramaiah Institute of Technology
Bengaluru

B J Sowmya
Assistant Professor
Department of Computer Science & Engineering
Ramaiah Institute of Technology
Bengaluru

OXFORD
UNIVERSITY PRESS

Detailed Contents

Basics of R

1

Learning Objectives

A systematic study of this chapter will help the reader understand the following concepts:

- R Environment setup
- Installation of R and RStudio in Windows and Linux
- Basics of programming in R
- Data types, vectors, matrices, arrays, and classes
- Subsetting and coercion

1.1 INTRODUCTION

R, the language for statistical computing was initially developed by *Ross Ihaka* and *Robert Gentleman* at the University of Auckland in early 90s. It is considered as an open source implementation of the S language, which was developed by John Chambers in Bell Laboratories during the early 80s. R is highly extensible and provides solutions for a wide variety of statistical techniques and visualization capabilities. As it is open source and is supported by a large number of contributors, it is emerging as one of the most powerful solutions to big data analysis.

Like in every programming language, there are many pros and cons with R. The major advantages of using R are (a) it is open source and hence free, *(*b*)* it has top-notch functionalities for effective graphical representation of data, and (c) it is easy to use. In comparison with many other statistical software packages, R uses a command line interface, which allows users to code things in console and in the form of scripts. This might be frustrating for beginners, but makes work reproducible later. Users can wrap their work in R scripts and this can be easily shared with colleagues. Because of these advantages, R appeals to a large audience, both in academia and in business. Moreover, the ease of creating R packages to solve particular problems makes it more popular. The active R community has created thousands of well-documented R packages for a broad range of applications. R packages can be found for solving business analytics in financial sector and health care, high performance computing, distributed computing, statistics, and many more cutting edge research areas.

However, there are also some disadvantages. R seems to be relatively easy to learn in the beginning, but it is hard to really master it. Further, as R is command-based, it becomes highly inconvenient for many of the statisticians and other non-computer science professionals to use it. This steep learning curve sometimes results in poorly written R code that is very hard to read and to maintain. Such poorly written R code may slow down the process if users are working with large data sets.

1.2 R-ENVIRONMENT SETUP

R is one of the world's widely used open source programming tool and provides software environment for graphical representation, information reporting, and statistical analytics. R is licensed under GNU public license and is freely available for development and research purposes. It is also available under commercial license for advanced usage and requirements. The binary pre-compiled versions of R are available for all the well-known operating systems such as Windows, Mac, Linux, Fedora, Redhat, and openSUSE. R and its IDE, RStudio can be downloaded from www.r-project.org and www.rstudio.org. The steps involved in installing and configuring R and RStudio are explained in the following sections.

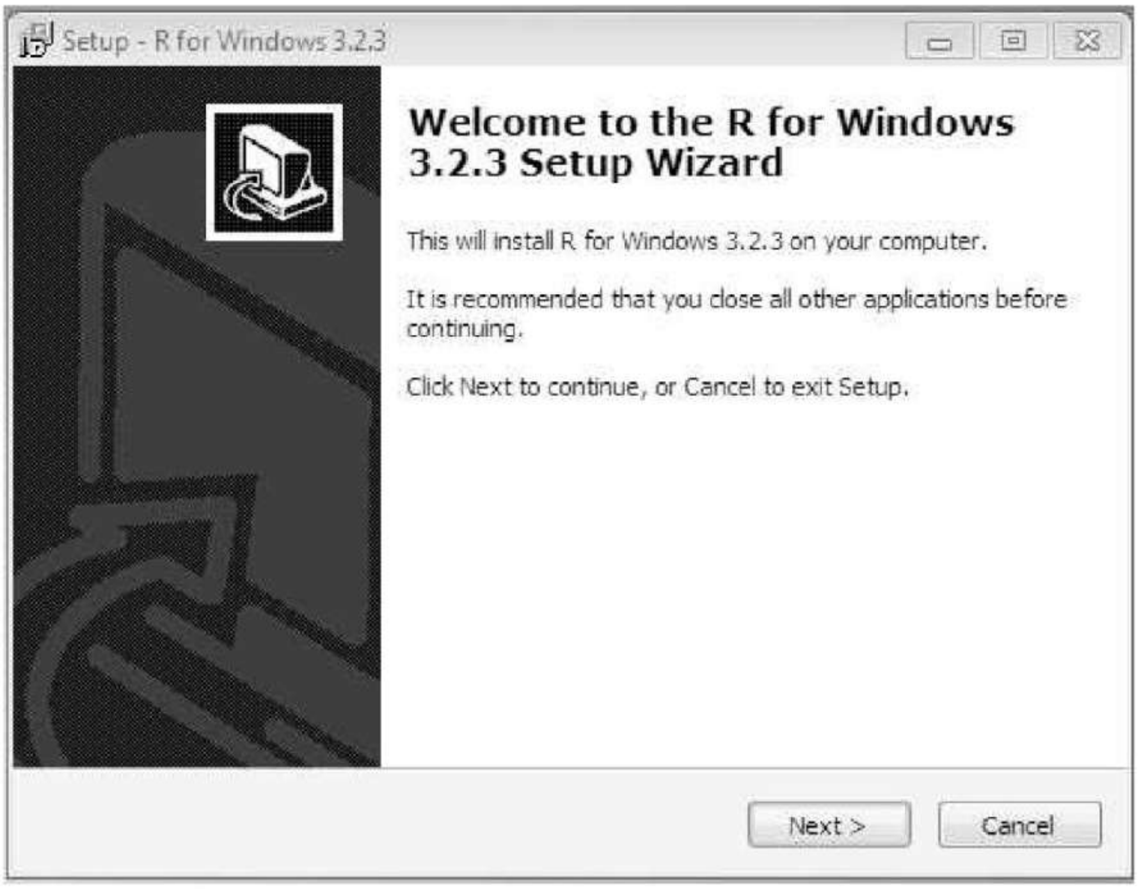
1.2.1 Installation of R

R can be installed from the mirror sites of CRAN (Comprehensive R Archive Network). The steps involved in installing R for windows and Linux operating system are described here.

In Windows

R can be installed in Windows 7/8/10/Vista and supports both the 32-bit and 64-bit versions. Go to the CRAN website and select the latest installer R 3.3.3 for Windows and download the.exe file. Double click on the download file and select Run as Administrator from the popup menu. Select the language to be used for installation and follow the directions. The installation folder for R can be found in C:\Programs\R. The steps for installing R with snapshots are detailed here:

1. Click on the link https://cran.r-project.org/bin/windows/base/ which redirects you to the download page.
2. Select the latest installer R-3.3.3 for installation and download the same. After download, clicking on the setup file opens the dialog box shown in Fig. 1.1.



Setup - R for Windows 3.2.3

Welcome to the R for Windows 3.2.3 Setup Wizard

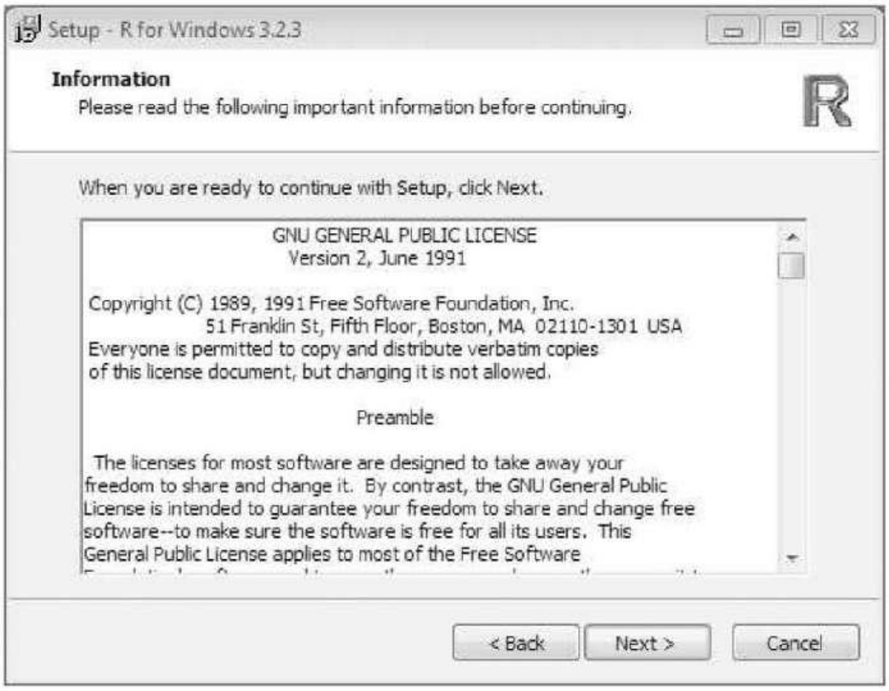
This will install R for Windows 3.2.3 on your computer.

It is recommended that you close all other applications before continuing.

Click Next to continue, or Cancel to exit Setup.

Fig. 1.1 Setup file for R

3. Click on the 'Next' button starts the installation process. This redirects you to the license window shown in Fig. 1.2; it is advisable to read the terms and conditions before selecting 'Next'.



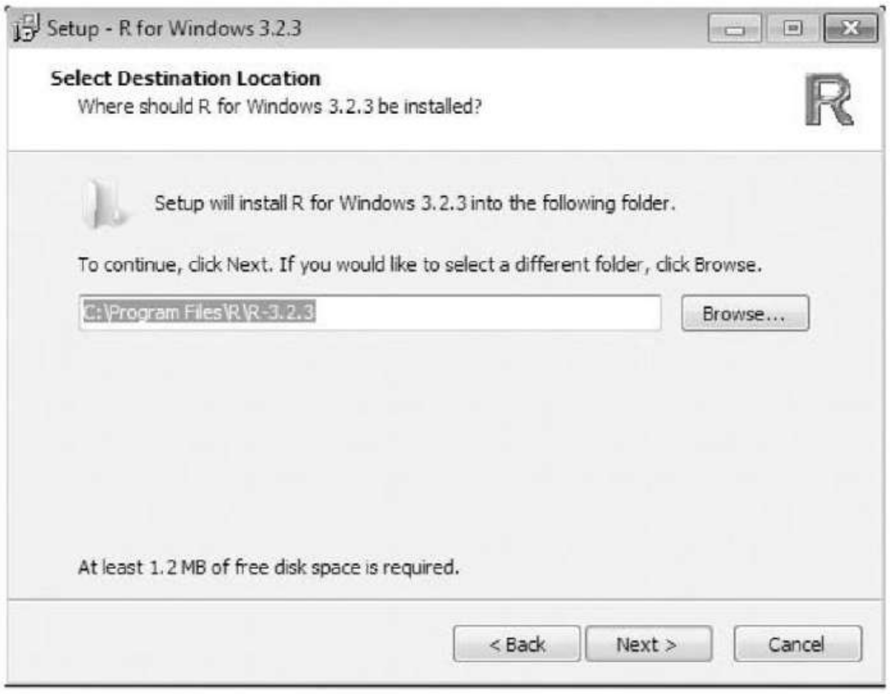
Information
Please read the following important information before continuing.

When you are ready to continue with Setup, click Next.

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.
           51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
```

Fig. 1.2 R licence agreement

4. After selecting the next button from the previous step the installation folder path is required (Fig. 1.3). Select the desired folder for installation; it is advisable to select the C directory for smooth running of the program.



Select Destination Location
Where should R for Windows 3.2.3 be installed?

Setup will install R for Windows 3.2.3 into the following folder.

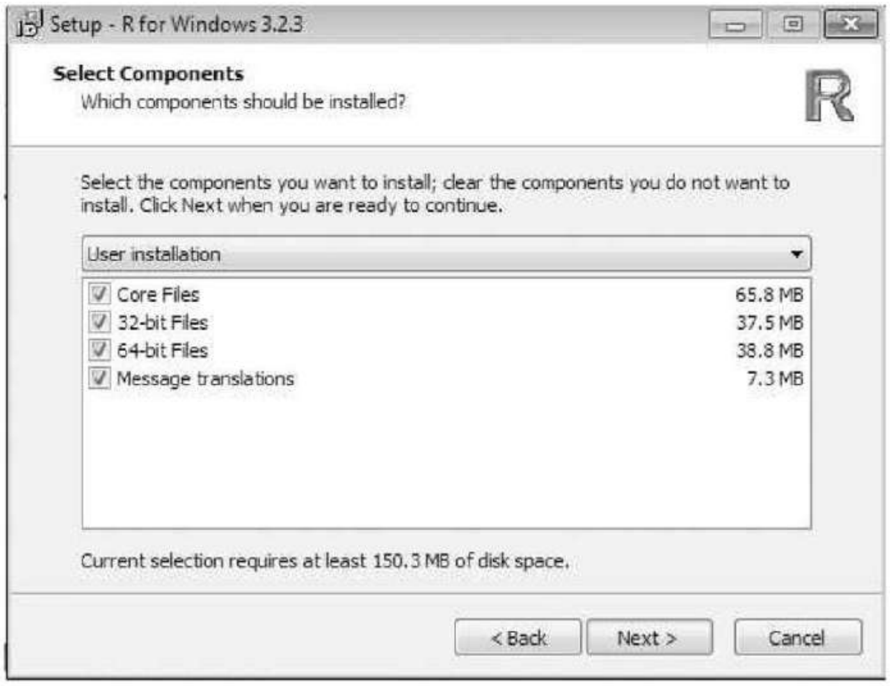
To continue, click Next. If you would like to select a different folder, click Browse.

C:\Program Files\R\R-3.2.3

At least 1.2 MB of free disk space is required.

Fig. 1.3 Selecting the installation folder

5. Next select the components for installation based on the requirements of your operating system (OS) to avoid unwanted use of disk space (Fig. 1.4).



Select Components
Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

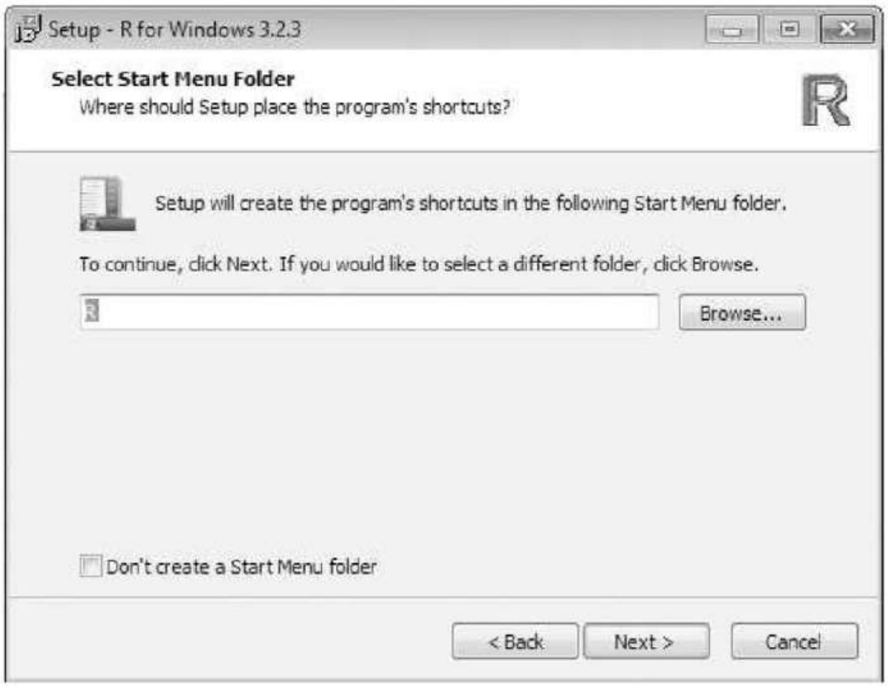
User installation

✓ Core Files	65.8 MB
✓ 32-bit Files	37.5 MB
✓ 64-bit Files	38.8 MB
✓ Message translations	7.3 MB

Current selection requires at least 150.3 MB of disk space.

Fig. 1.4 Selecting components for installation

6. In the next dialog box (Fig. 1.5), we need to select the start menu folder. Here, it is better to go with the default option given by the installer.



Select Start Menu Folder
Where should Setup place the program's shortcuts?

Setup will create the program's shortcuts in the following Start Menu folder.

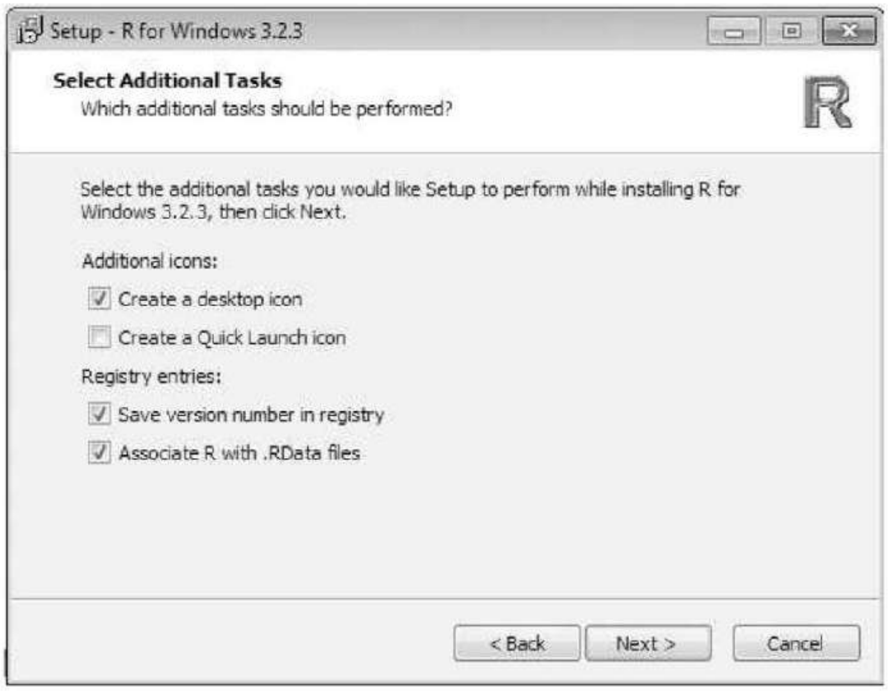
To continue, click Next. If you would like to select a different folder, click Browse.

R

☐ Don't create a Start Menu folder

Fig. 1.5 Selecting the Start menu folder

7. After setting up the Start menu folder, check the additional options for completing the setup as shown in Fig. 1.6.


Setup - R for Windows 3.2.3

Select Additional Tasks
Which additional tasks should be performed?

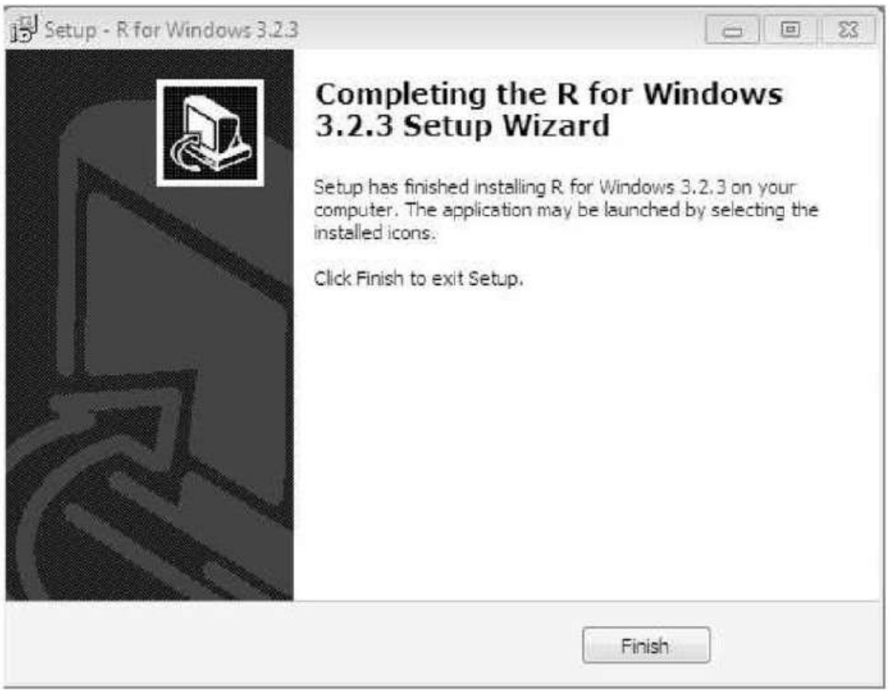
Select the additional tasks you would like Setup to perform while installing R for Windows 3.2.3, then click Next.

Additional icons:
- [✓] Create a desktop icon
- [ ] Create a Quick Launch icon

Registry entries:
- [✓] Save version number in registry
- [✓] Associate R with .RData files

Fig. 1.6 Additional options for setup

8. After clicking next from the previous step, the installation procedure ends and the window in Fig. 1.7 is displayed. Click Finish to exit from the installation window.


Setup - R for Windows 3.2.3

Completing the R for Windows 3.2.3 Setup Wizard

Setup has finished installing R for Windows 3.2.3 on your computer. The application may be launched by selecting the installed icons.

Click Finish to exit Setup.

Fig. 1.7 End of installation

In Ubuntu

R for Linux can be obtained as binaries with different versions and the installation varies with each version. The following are the steps involved in installation: Go to the CRAN website or follow the link https://cran.r-project.org/bin/linux/ and it will redirect you to the index page. The index page contains four different parent directories—debain, Redhat, sesu, and ubuntu.

Click on the desired version of Linux and it redirects you to the installation page which consists of the commands for installation. To install R-Base for Ubuntu 14.04 follow the commands given here:

1. Add R to your repository by typing the following command:

```
>sudo echo "dep http://cran.rstudio.com/bin/ubuntu trusty/" | sudo tee
-a etc/apt/spurces.list.
```

   Here *trusty* refers to Ubuntu 14.04; if any other version of Ubuntu is installed in your system, replace *trusty* with the respective version name from the CRAN website.

2. Add R to Ubuntu keyring.

```
>gpg –keyserver keyserver.ubuntu.com –recv-key E084DAB9
>gpg -a –export E084DAB9 |sudo apt-key add -
```

3. Finally install R-Base.

```
>sudo apt-get update
>sudo apt-get install r-base r-base-dev apt-get update
```

   An easy way to install R is by using the *yum* command. The command line interface (CLI) for the same is

```
$ yum install R
```

This command installs the core functionalities of the R programming language and also the standard package required for the working with R. After installing all the standard packages you can install the additional packages by launching R console using

```
$ R
```

This command initiates the R prompt '>' and the necessary packages can be installed by just typing them as commands. For example,

```
>install.packages("plotrix")
```

1.3 RSTUDIO

The Integrated Development Environment (IDE) for R is RStudio and it provides a variety of features such as an editor with direct code execution and syntax highlighting, a console, tools for plotting graphs, history lookup, debugging, and an environment for workspace creation. In this section the steps for installing RStudio for Linux and Windows environments are briefly explained.

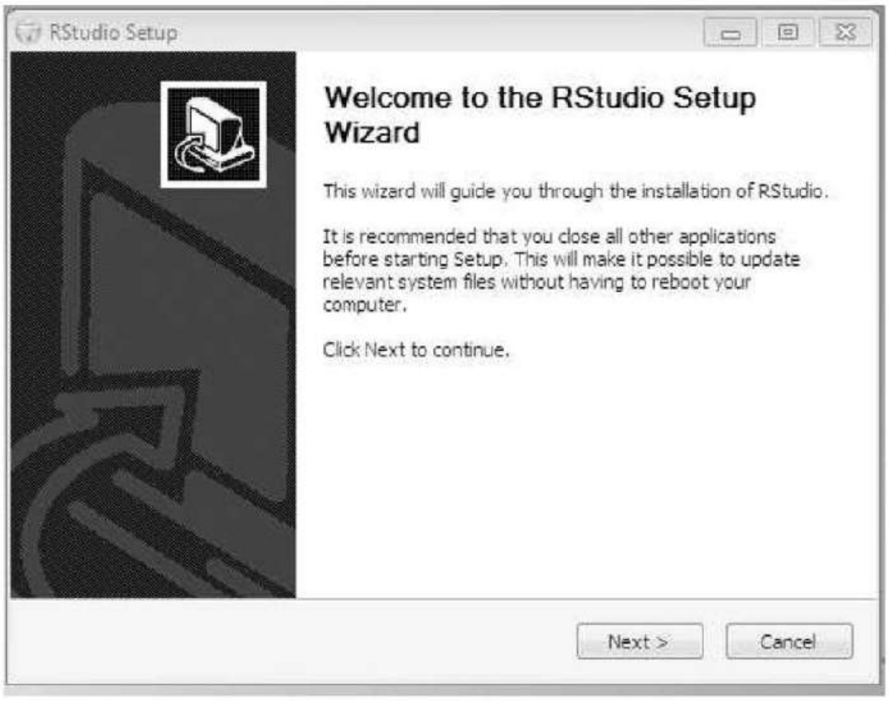
1.3.1 Installing and Configuring RStudio in Windows

RStudio can be installed in any of the Windows platforms such as Windows 7/8/10/Vista and can be configured within a few minutes. The basic requirement is R 2.11.1+ version. The following are the steps involved to setup RStudio:

1. Download the latest version of RStudio just by clicking on the link provided here https://www.rstudio.com/products/rstudio/download/ and it redirects you to download page. There are two versions of

RStudio available—desktop and server. Based on your usage and comfort, select the appropriate version to initiate your download. The latest desktop version for RStudio is 1.0.136.

2. Download the.exe file and double click on it to initiate the installation as shown in Fig. 1.8.



Welcome to the RStudio Setup Wizard

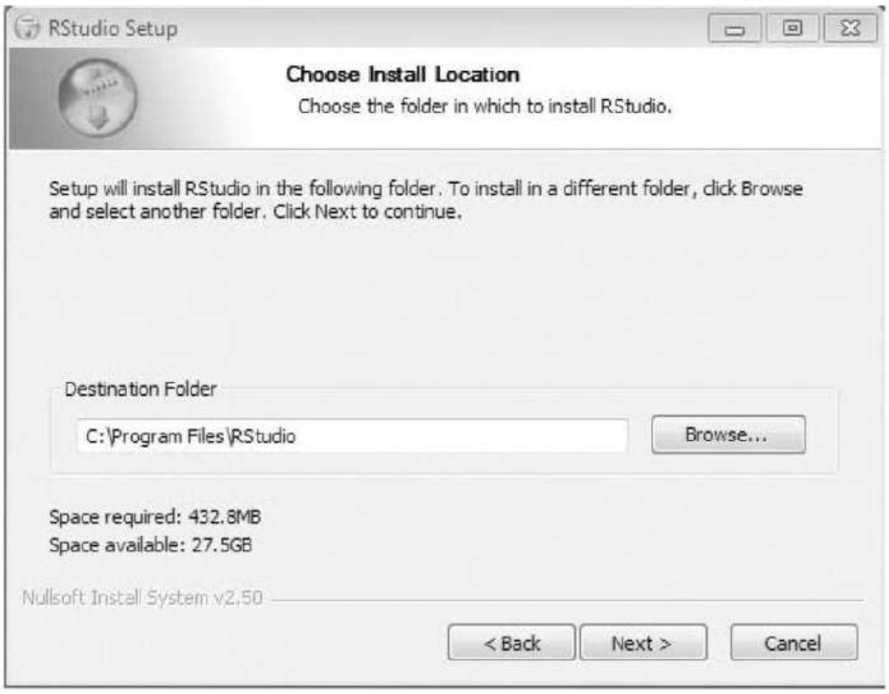
This wizard will guide you through the installation of RStudio.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Fig. 1.8 RStudio Setup

3. Click on the Next button and it redirects you to select the installation folder (Fig. 1.9). Select 'C:\' as your installation directory since R and RStudio must be installed in the same directory to avoid path issues for running R programs.



Choose Install Location

Choose the folder in which to install RStudio.

Setup will install RStudio in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Program Files\RStudio

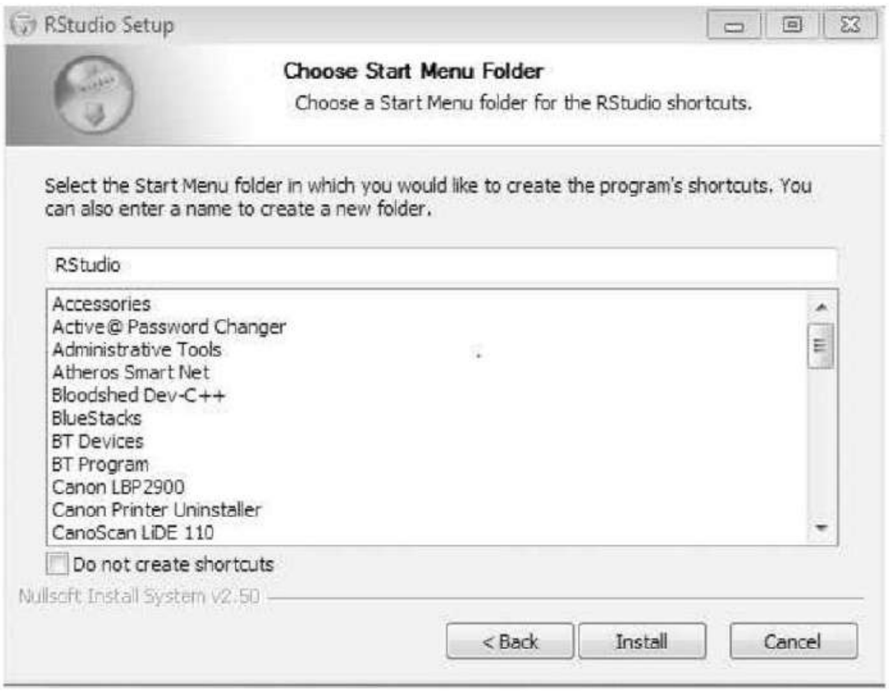
Space required: 432.8MB

Space available: 27.5GB

Nullsoft Install System v2.50

Fig. 1.9 Selecting the installation folder

4. Click Next to continue and a dialog box asking you to select the Start menu folder opens as shown in Fig. 1.10. Its is advisable to create your own folder to avoid any possible confusion and click on Install button to install RStudio.



Choose Start Menu Folder

Choose a Start Menu folder for the RStudio shortcuts.

Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

RStudio

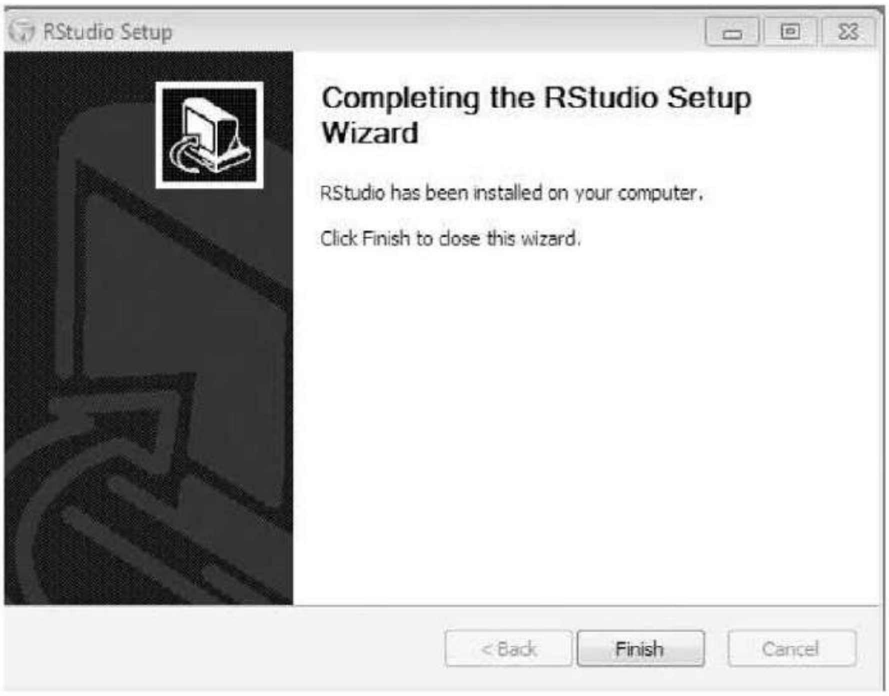
Accessories
Active@ Password Changer
Administrative Tools
Atheros Smart Net
Bloodshed Dev-C++
BlueStacks
BT Devices
BT Program
Canon LBP2900
Canon Printer Uninstaller
CanoScan LiDE 110

☐ Do not create shortcuts

Nullsoft Install System v2.50

Fig. 1.10 Choosing the Start menu folder

4. After completion of installation, the following window as shown in Fig. 1.11 appears. Click on 'Finish' to exit.



Completing the RStudio Setup Wizard

RStudio has been installed on your computer.

Click Finish to close this wizard.

Fig. 1.11 Installation complete

1.3.2 Installing and Configuring RStudio in Linux

After installing R-Base the next step is to install RStudio. It can be installed by using the following simple commands.

1. First install the core debain command for installation of the debain version of RStudio.

```
>sudo apt-get install gdebi-core
```

2. Using the wget command fetch the debain version of RStudio.

```
>wget https://download1.rstudio.org/rstudio-1.0.136-amd64.deb
```

3. After fetching RStudio, the following commands install RStudio with the standard packages.

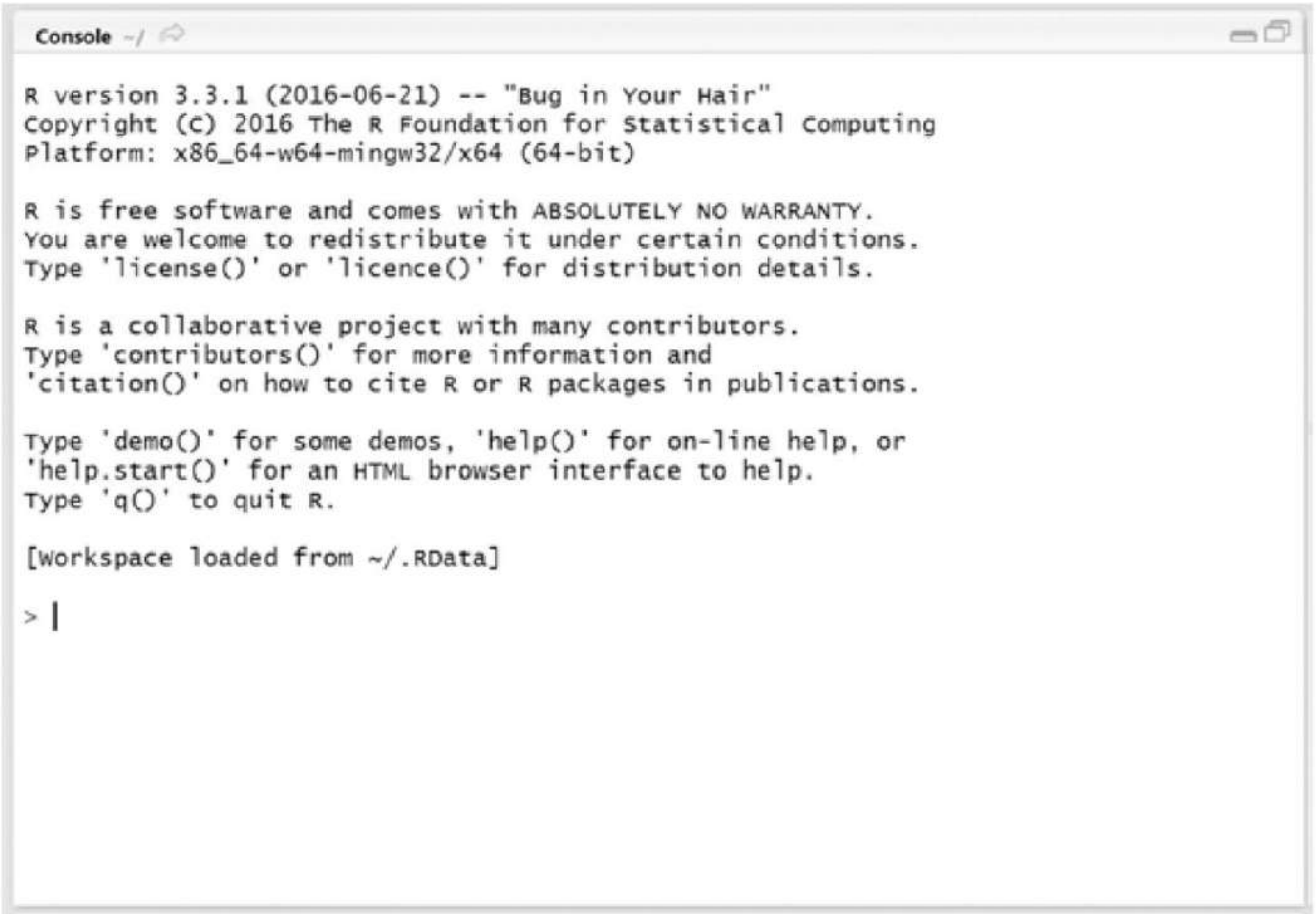
```
>sudo gdebi -n rstudio-1.0.136-amd64.deb
```

4. After installing RStudio, remove the installation file for saving disk space.

```
>rm rstudio-1.0.44-amd64.deb
```

1.4 PROGRAMMING WITH R

To begin with R, one needs to start working with R console where all the action takes place. Figure 1.12 gives a look and feel of the R console.



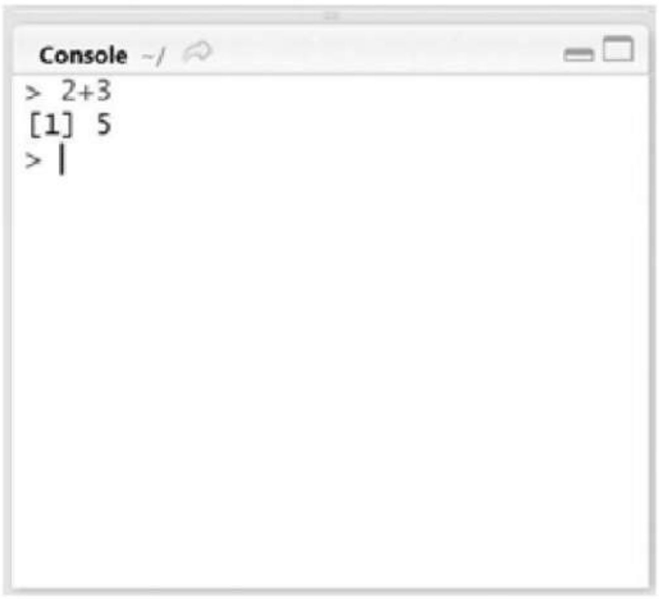
```
R version 3.3.1 (2016-06-21) -- "Bug in Your Hair"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

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.

[Workspace loaded from ~/.RData]

> |
```

Fig. 1.12 R console

R console is an execution window where users can execute R commands. Users need to type the required action on the command prompt and upon pressing the Enter key, R interprets the action typed by the user, executes the same, and gives the answer.

Example 1.1 Write the commands to perform basic arithmetic with R console.

Solution:
To calculate the sum of two numbers, say 1 and 2, the programmer needs to type 1 + 2 in the command prompt of the console. Then, R compiles what is typed on the command prompt, calculates the result, and prints the same as a numerical value (Fig. 1.13).

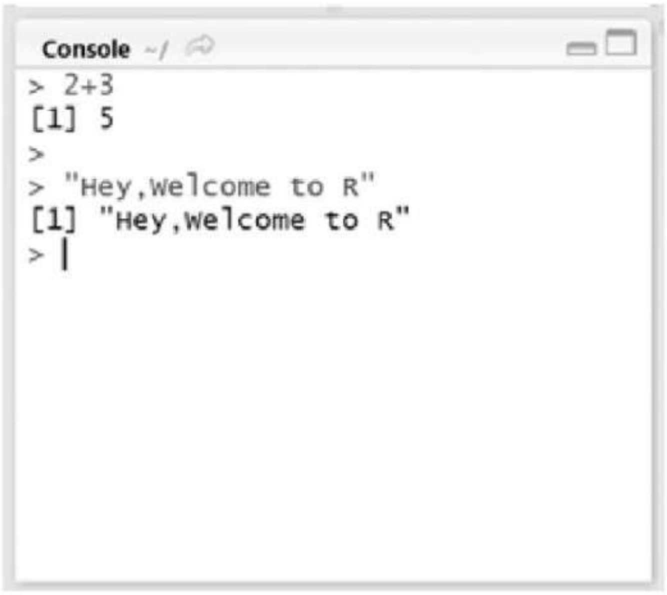


```
> 2+3
[1] 5
> |
```

Fig. 1.13 Basic arithmetic with R

Example 1.2 Display a string on R console.

Solution:
To display some text in the R console, programmers need to type the same within double quotes. R interprets the character string within double quotes and simply prints the same as the output.



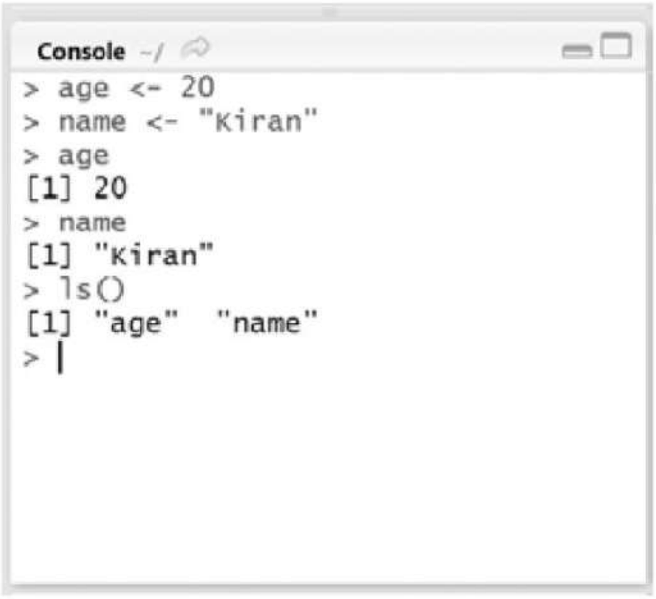
```
> 2+3
[1] 5
>
> "Hey,Welcome to R"
[1] "Hey,Welcome to R"
> |
```

Fig. 1.14 Displaying a string

As users assign values to variables in R console, these get accumulated in the R workspace. R workspace stores all the variables and their metadata information. Users can access the objects in the workspace using the *ls*() function. The usage of *ls* is shown in Example 1.5. *ls* lists all the stored variables that are created during the particular R session.

Example 1.5 Enumerate the process of listing the stored variables in R.

Solution:

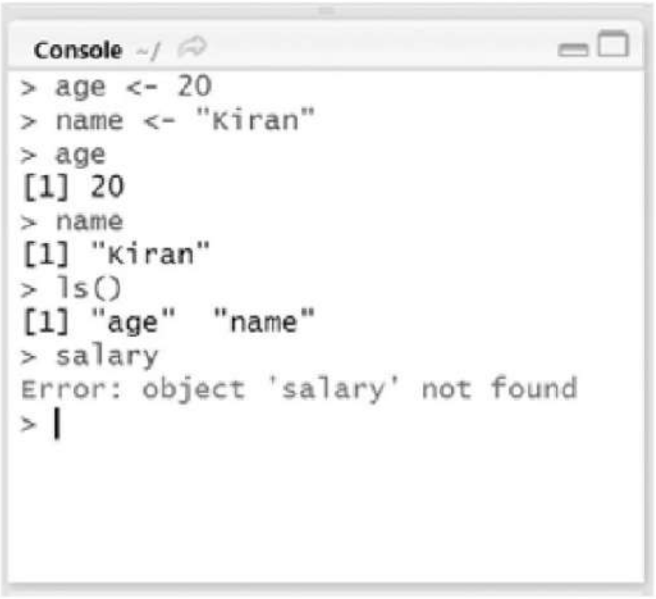


Console ~/

```
> age <- 20
> name <- "Kiran"
> age
[1] 20
> name
[1] "Kiran"
> ls()
[1] "age"  "name"
> |
```

Fig. 1.17 Listing the stored variables in R

If the user tries to fetch the value of a non-existing variable, then R throws an error as shown in Fig. 1.18, because, in this case, the requested variable *salary* is not defined in the workspace so far and thus not found.

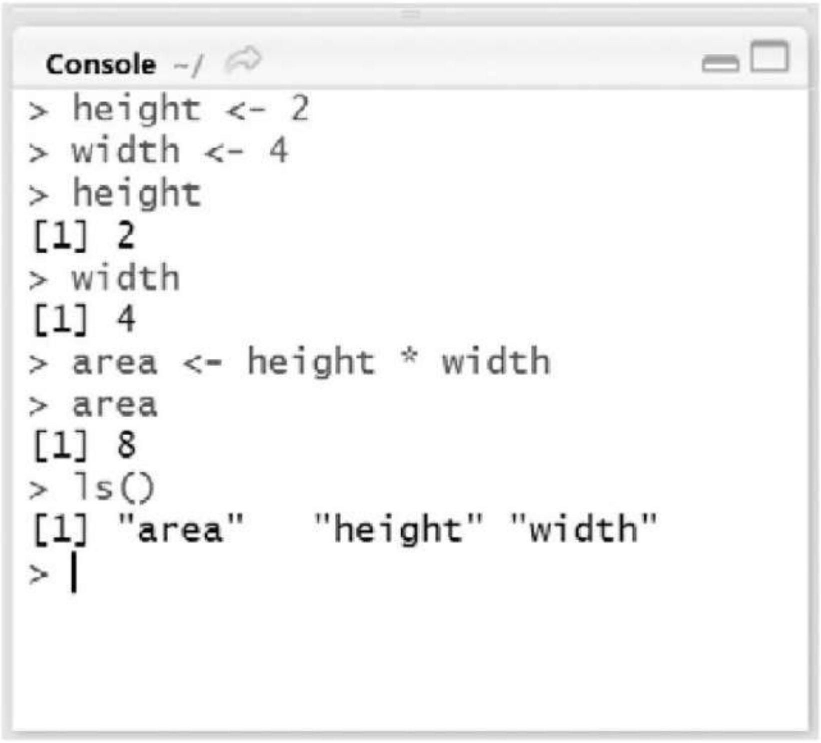


Console ~/

```
> age <- 20
> name <- "Kiran"
> age
[1] 20
> name
[1] "Kiran"
> ls()
[1] "age"  "name"
> salary
Error: object 'salary' not found
> |
```

Fig. 1.18 Error in R

The principle of using a workspace is to make the declared variables available for further use. Suppose the problem is to find out the area of an imaginary rectangle, which is nothing but the height multiplied by the width. Figure 1.19 illustrates the procedure of creating two variables *height* and *width,* assigning them with initial values, writing an expression for calculating the area of a rectangle, and finally executing the expression. So, now this workspace contains three objects namely *height*, *width*, and *area*. Executing the *ls* command lists all the declared objects in the workspace.

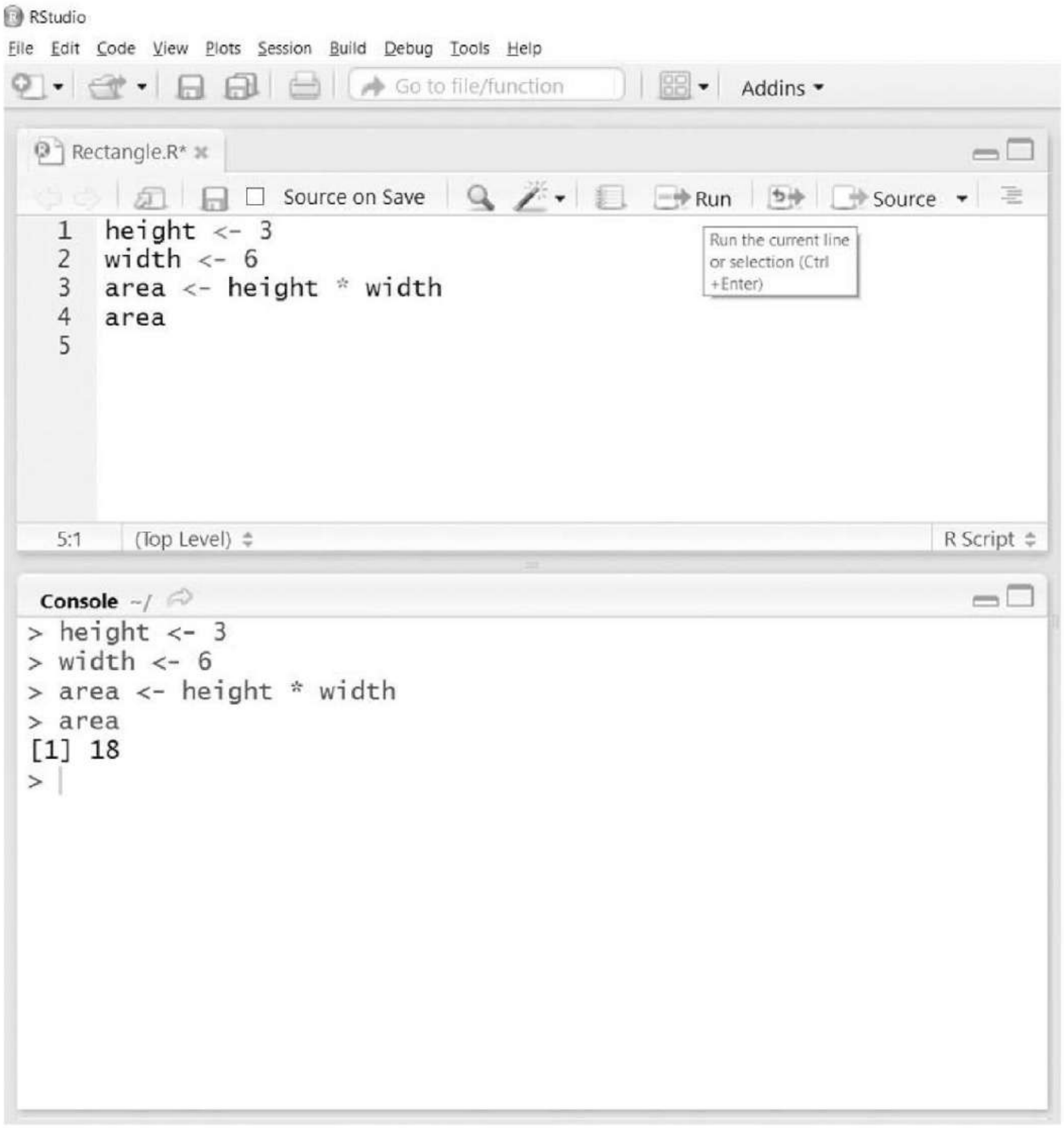


```
> height <- 2
> width <- 4
> height
[1] 2
> width
[1] 4
> area <- height * width
> area
[1] 8
> ls()
[1] "area"   "height" "width"
>
```

Fig. 1.19 Creating multiple variables

The basic concern of using the R console is the reuse and reassigning of the existing variables in the workspace. For example, if users want to recalculate the *area* of an imaginary rectangle when the *height* is 3 and the *width* is 6, the users need to reassign the variables *height* and *width*, and then recalculate the *area*. In many instances, the users may have to reuse the variables with different values. To facilitate this, programmers can write R scripts. An R script is simply a text file, containing a collection of successive lines of R code to solve a particular task. When using R, users need to build many scripts to make things easier for reuse, and hopefully automate parts of the work. RStudio was developed to make the job of R programmers easy. RStudio is the most popular IDE to write, reuse, and execute R programs. The *Run* button in RStudio is used to run the R script, which eventually interprets the R program line by line and executes every command, as in the case of R console. The most convenient part of RStudio is that, if users want to change the value of a variable, they can do it easily in the script file and again run the same. A sample R script to find the area of a rectangle written in RStudio and the console showing the results after running the R script through RStudio is shown in Fig. 1.20.

RStudio provides the true power of reproducibility. This also makes it easy to share R scripts among different users. In order to make others understand our code better, there is a need for documentation. RStudio allows programmers to write their comments as part of the code.


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Addins
Rectangle.R*
Source on Save
Run
Source
Run the current line or selection (Ctrl+Enter)
5:1
(Top Level)
R Script
Console ~/

```
height <- 3
width <- 6
area <- height * width
area
```

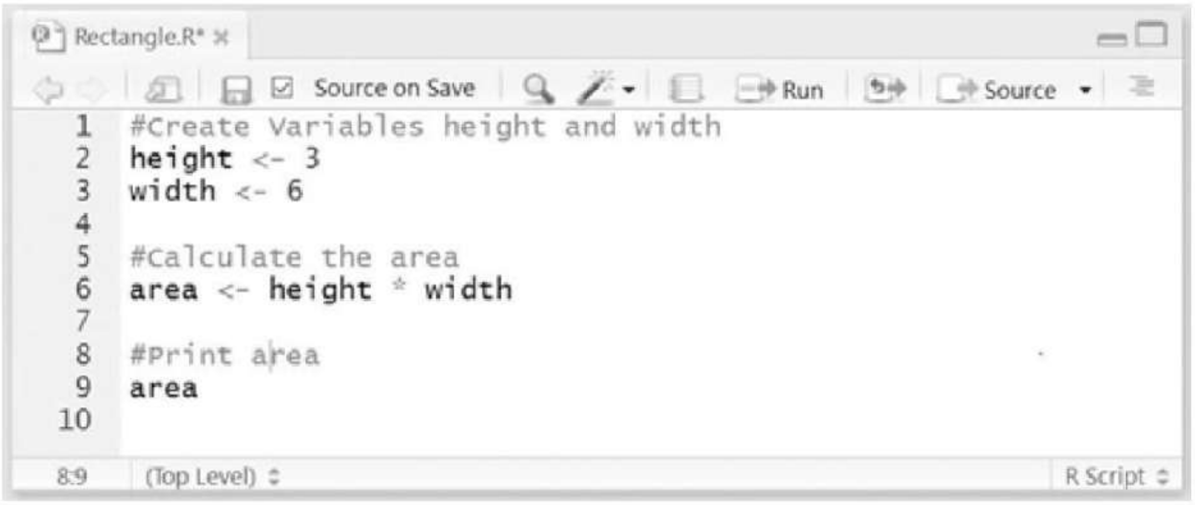
```
> height <- 3
> width <- 6
> area <- height * width
> area
[1] 18
>
```

Fig. 1.20 RStudio—R's most popular IDE

Example 1.6 Write an R script with comments.

Solution:

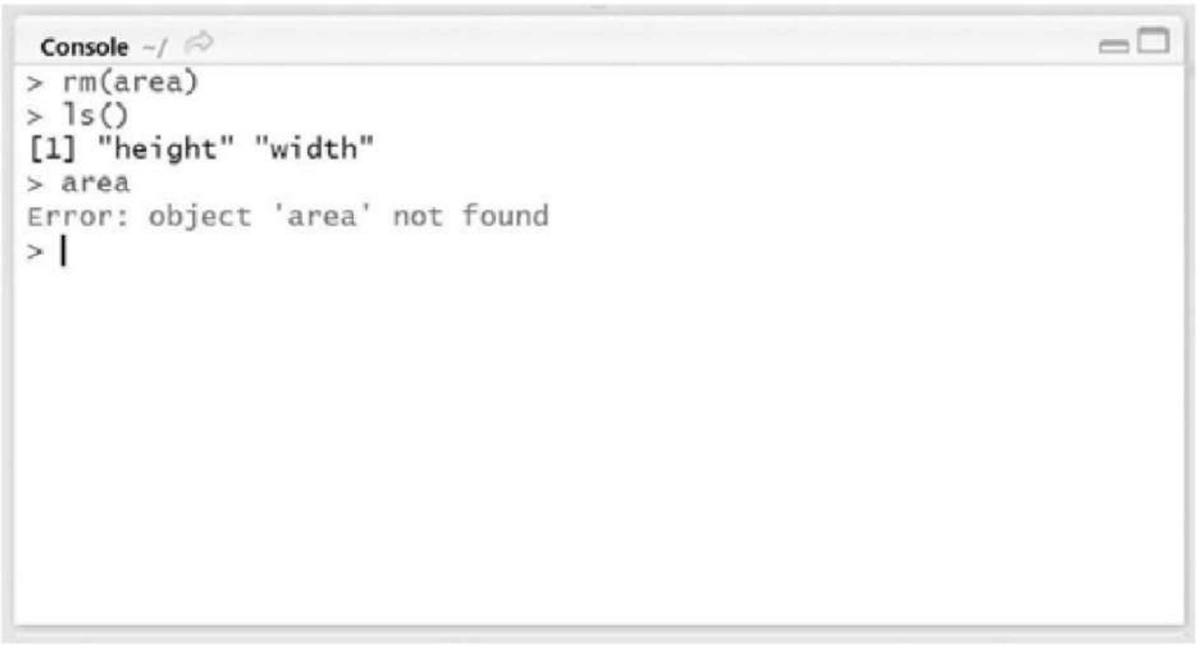
The comments in R start with the # symbol as shown in Fig. 1.21. If the user wants to run this script again, the lines starting with # do not affect R's execution. One has to be careful not to include the # symbol at the beginning of a proper R code as it will be considered as a comment only and not be executed by RStudio.



```
#Create Variables height and width
height <- 3
width <- 6

#Calculate the area
area <- height * width

#Print area
area

```

Fig. 1.21 Syntax for inserting comments

In case of using R for big data problems, it is common that the workspace consumes a lot of computer resources because of tons of data being involved. Therefore, it is always better to clean up the workspace periodically to make sure it does not get bloated. As a part of this cleaning process, the *rm*() command can be used to remove a particular variable as shown in Fig. 1.22. After you remove the variable *area* using the *rm*() function, and if you try to use it again, R throws an error as the requested object has already been deleted from the current workspace. If *ls* gets executed after *rm,* it lists only the remaining variables.



```
> rm(area)
> ls()
[1] "height" "width"
> area
Error: object 'area' not found
> |
```

Fig. 1.22 Removing a particular variable in R

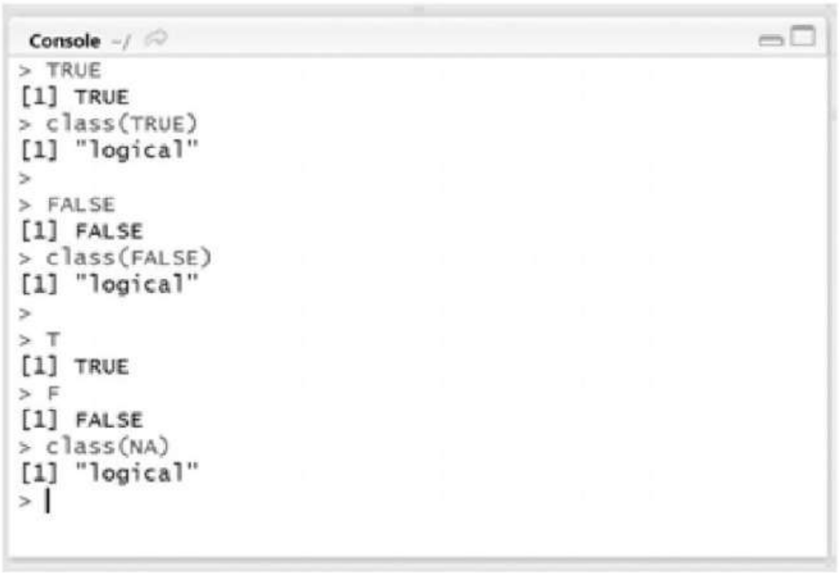
1.5 BASIC DATA TYPES

As in all programming languages, data types play a vital role in developing any application and the data becomes very important especially in statistical analysis. To begin with, let's discuss R fundamental data types, which are also known as atomic vector types that are used extensively in R programs.

R uses a function named *class*() to determine the type of a variable. In Example 1.7, *class*() is used to determine the type of the variable *TRUE*. As *TRUE* is logical, the function *class*(*TRUE*) returns *Logical*. Similarly, *class*(*NA*) returns *Logical* where *NA* denotes the missing value. The logical variables *TRUE* and *FALSE* can also be abbreviated as T and F respectively, but it is preferred to use their full versions.

Example 1.7 Write the commands in R console to determine the type of the variable.

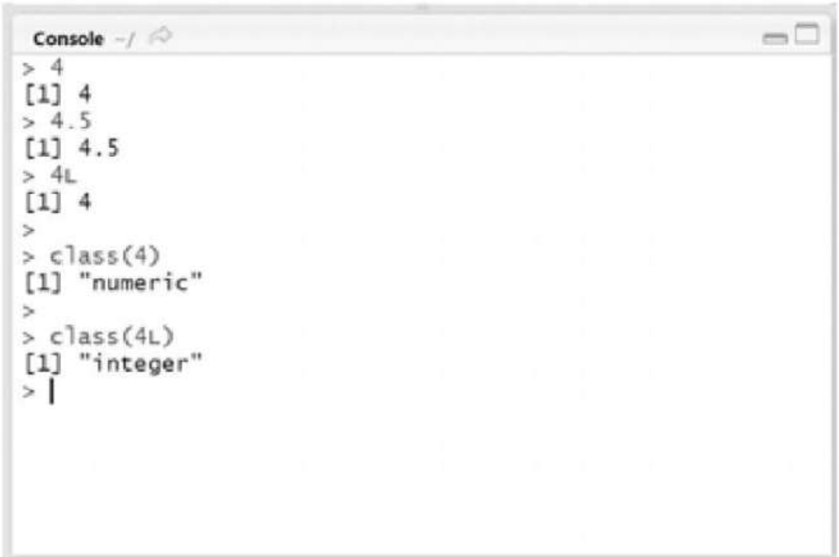
Solution:



```
Console ~/
> TRUE
[1] TRUE
> class(TRUE)
[1] "logical"
>
> FALSE
[1] FALSE
> class(FALSE)
[1] "logical"
>
> T
[1] TRUE
> F
[1] FALSE
> class(NA)
[1] "logical"
> |
```

Fig. 1.23 Determining the type of a variable

The numeric variables are used to represent numbers in R. Usually, all the arithmetic operations such as addition, subtraction, multiplication, and division can be directly applied on these numeric variables. *Integer* is a special type of numeric in R, which is used to represent natural numbers. To specify a digit as an integer, the programmer needs to add L next to it as shown in Fig. 1.24, but the users do not see the difference between the *integer* 4 and the *numeric* 4L from the output point of view. However, the *class*() function reveals the difference for the inputs 4 and 4L as shown in Fig. 1.24.



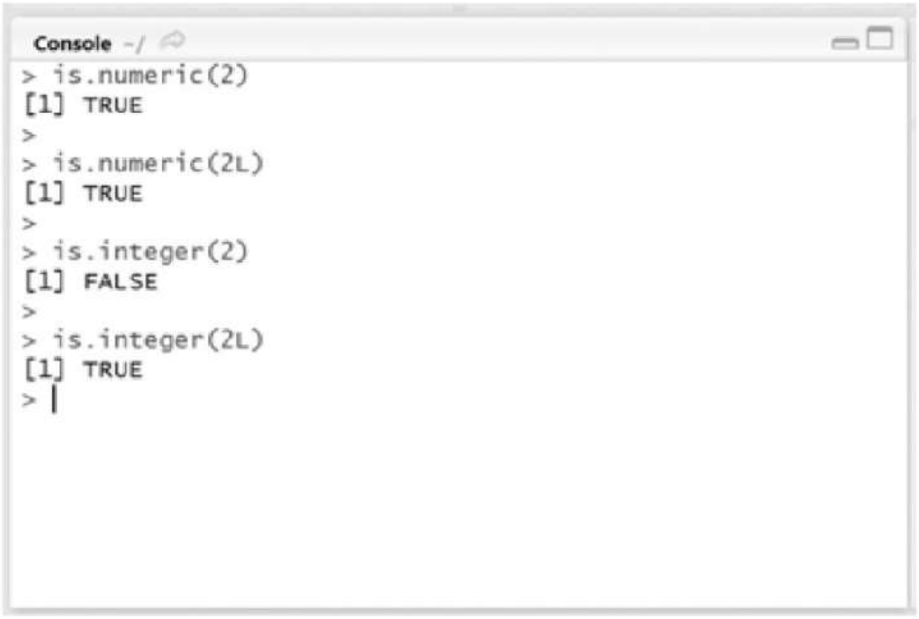
```
Console ~/
> 4
[1] 4
> 4.5
[1] 4.5
> 4L
[1] 4
>
> class(4)
[1] "numeric"
>
> class(4L)
[1] "integer"
> |
```

Fig. 1.24 Determining the integer type of the variable

Example 1.8 Enumerate the process to check whether a given input is numeric or integer using a function of R.

Solution:

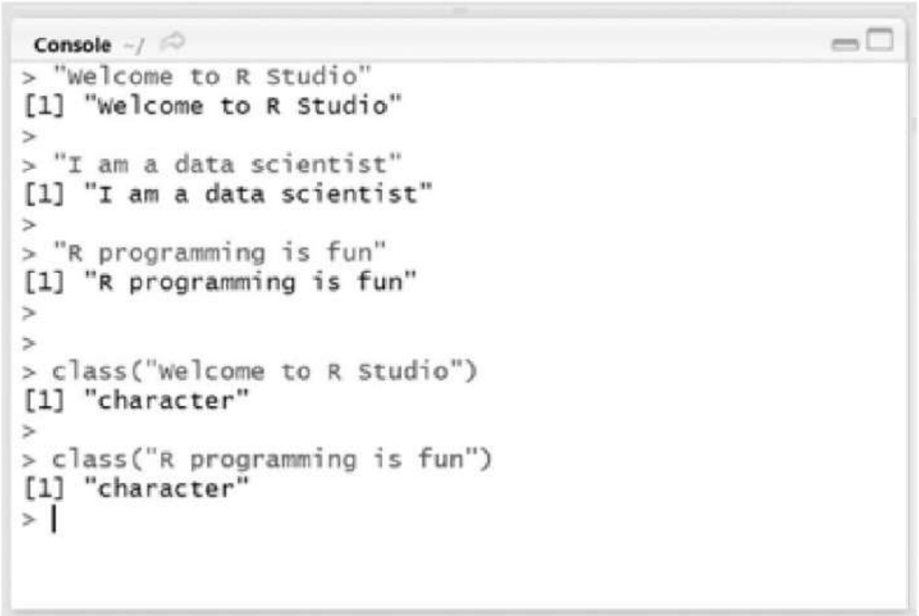
R provides a function for the programmers to check whether a given input is numeric or integer. This is known as the *is-dot-function*. Here, we use two functions—*is.numeric*() and *is.integer*()—as shown in Fig. 1.25, to check if a variable is a numeric or integer. In this particular case, it appears that both are *numeric* and the results also demonstrate the fact that all *integer* variables are *numeric*, but not all *numeric* variables are *integers*.



```
> is.numeric(2)
[1] TRUE
>
> is.numeric(2L)
[1] TRUE
>
> is.integer(2)
[1] FALSE
>
> is.integer(2L)
[1] TRUE
> |
```

Fig. 1.25 Checking if a given input is numeric or integer

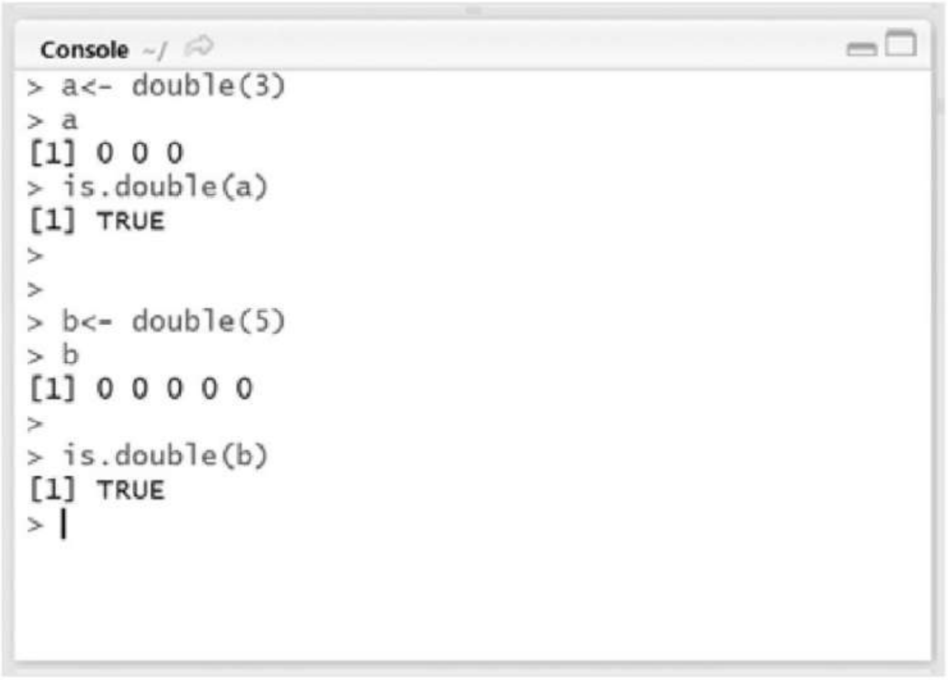
R allows the programmers to use the string of characters. Figure 1.26 displays the class of character strings with the type of object as *character*.



```
> "Welcome to R Studio"
[1] "Welcome to R Studio"
>
> "I am a data scientist"
[1] "I am a data scientist"
>
> "R programming is fun"
[1] "R programming is fun"
>
>
> class("Welcome to R Studio")
[1] "character"
>
> class("R programming is fun")
[1] "character"
> |
```

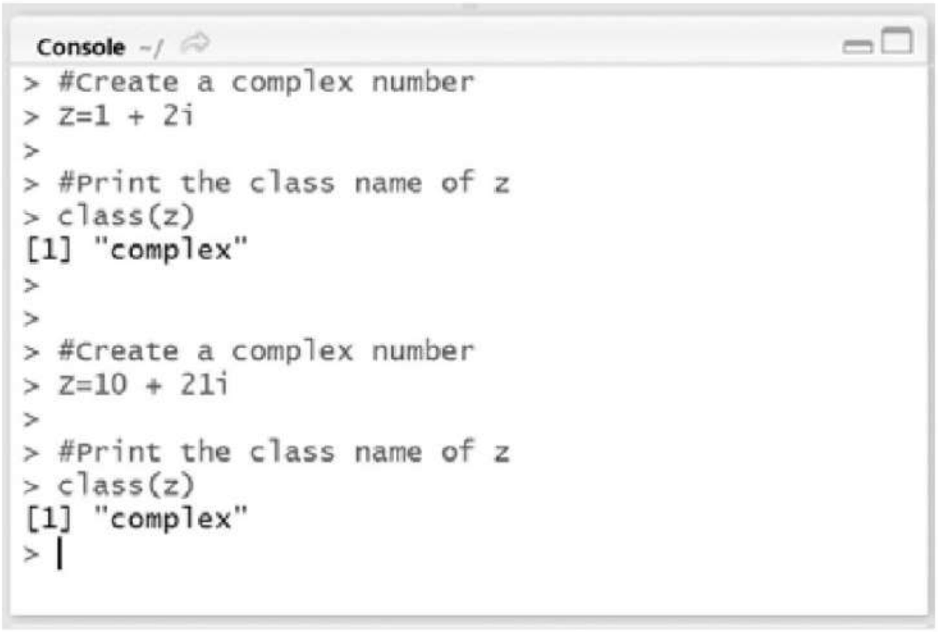
Fig. 1.26 Displaying the class of character strings

Apart from *integer* and *numeric*, R provides many other data types such as *double* for higher precision integer arithmetic, which creates a double-precision vector of the specified length. Initially, all the elements of the vector are initialized to 0. Figure 1.27 illustrates the *double* works. Similarly, Fig. 1.28 illustrates another data type called *complex* for handling complex numbers, which has two parts—a real value and an imaginary value.



```
> a<- double(3)
> a
[1] 0 0 0
> is.double(a)
[1] TRUE
>
>
> b<- double(5)
> b
[1] 0 0 0 0 0
>
> is.double(b)
[1] TRUE
> |
```

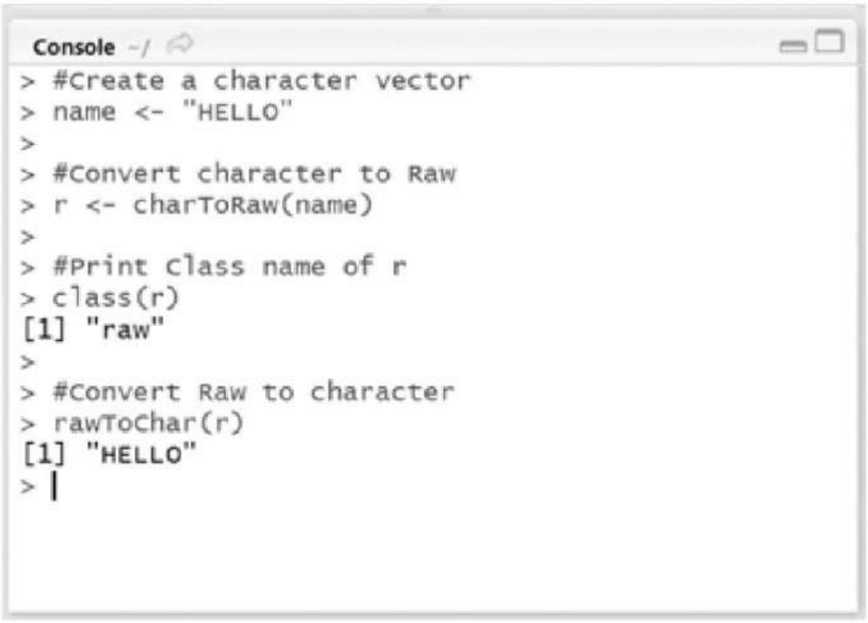
Fig. 1.27 Illustrating the usage of double data type



```
> #Create a complex number
> Z=1 + 2i
>
> #Print the class name of z
> class(z)
[1] "complex"
>
>
> #Create a complex number
> Z=10 + 21i
>
> #Print the class name of z
> class(z)
[1] "complex"
> |
```

Fig. 1.28 Illustrating the usage of complex data type

R provides flexibility to programmers to convert any given data type to a special data type called *raw*. The *raw* type is intended to hold any data as a sequence of bytes, where it is possible to extract sub-sequences of bytes and replace them as elements of a vector. In R, *raw* vectors are used to store fixed-length sequences of bytes. Figure 1.29 shows how a *character* vector is converted into a *raw* vector and vice versa.



```
> #Create a character vector
> name <- "HELLO"
>
> #Convert character to Raw
> r <- charToRaw(name)
>
> #Print Class name of r
> class(r)
[1] "raw"
>
> #Convert Raw to character
> rawToChar(r)
[1] "HELLO"
> |
```

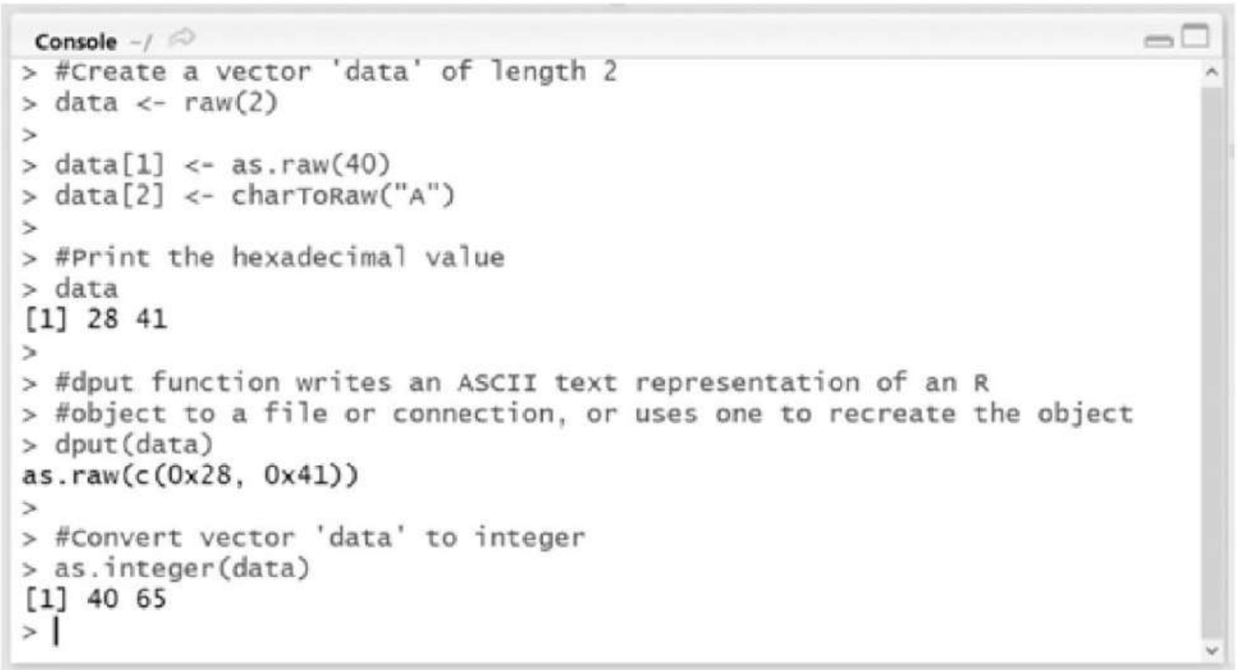
Fig. 1.29 Conversion of any given data type to raw data type

Example 1.9 clearly shows the difference between the *raw* vectors and others.

Example 1.9 Write the commands in R to differentiate between the *raw* vectors and other vectors.

Solution:

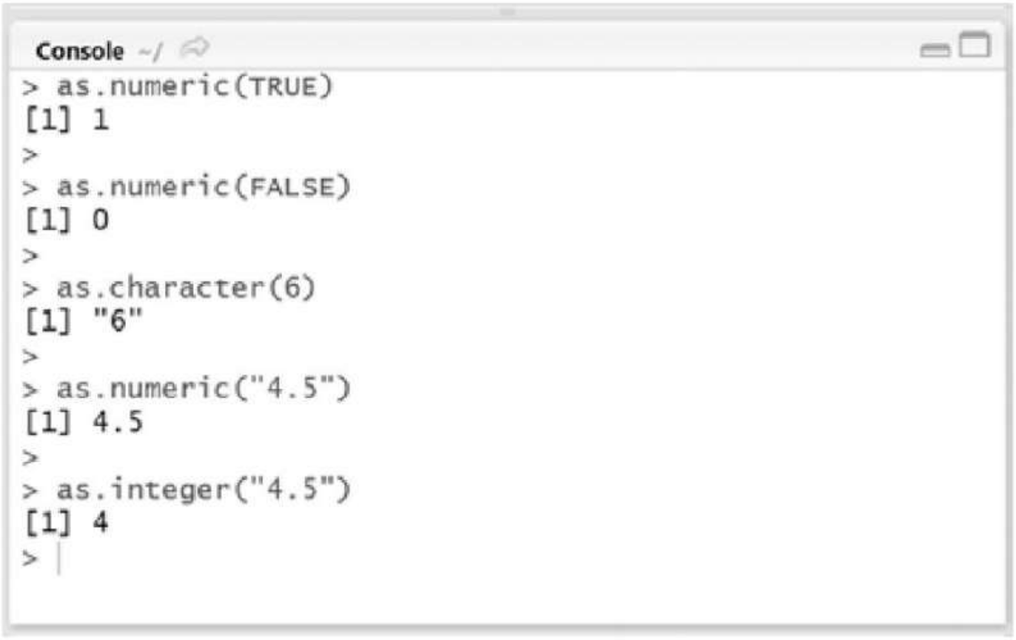
Here, as shown in Fig. 1.30, a random vector of two bytes is initially created and stored as *data*; the first byte of *data* is assigned the value 40 and the second byte is assigned the character A. If the variable *data* is printed on the command prompt, it displays the hexadecimal equivalents of the stored values. Another function *dput* is used to write an ASCII text representation of the variables. The variables can also be converted to other data types as in *as.integer(data)*, which converts the variable *data* into integer and displays. However, this is used only for display and the actual data type of *data* remains as *raw*.



```
> #Create a vector 'data' of length 2
> data <- raw(2)
>
> data[1] <- as.raw(40)
> data[2] <- charToRaw("A")
>
> #Print the hexadecimal value
> data
[1] 28 41
>
> #dput function writes an ASCII text representation of an R
> #object to a file or connection, or uses one to recreate the object
> dput(data)
as.raw(c(0x28, 0x41))
>
> #Convert vector 'data' to integer
> as.integer(data)
[1] 40 65
> |
```

Fig. 1.30 Difference between raw vector and others

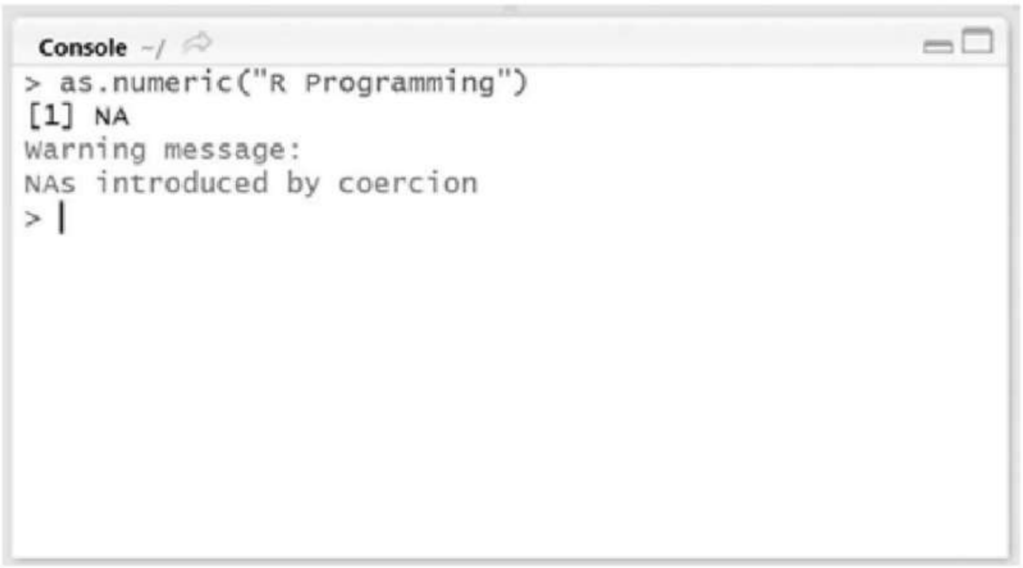
R gives the option to programmers to display a given data type as many other data types. Figure 1.31 shows how the function *as.numeric*() displays the logical values *TRUE* and *FALSE* as *numeric*. Similarly, the functions *as.character*() and *as.integer*() display the given input as character and integer, respectively.



```
Console ~/
> as.numeric(TRUE)
[1] 1
>
> as.numeric(FALSE)
[1] 0
>
> as.character(6)
[1] "6"
>
> as.numeric("4.5")
[1] 4.5
>
> as.integer("4.5")
[1] 4
> |
```

Fig. 1.31 Displaying a given data type as other data types

In programming, there are cases in which users may want to change the type of a variable. R provides a function called *coercion* to do this. By using the *as-dot* functions, one can *coerce* one type of variable to another type and many ways of transformation between the types are possible. Programmers can *coerce* numeric to character and vice versa. In some cases for example, if one wants to convert a character string "4.5" to an integer, R stores only 4 and implies some information loss as integers do not have the decimal part. However, *coercion* is not always possible. Figure 1.32 demonstrates a warning message when a user tries to convert a character string into a numeric. In these cases, coercion displays a warning message NA, that is, not available.



```
Console ~/
> as.numeric("R Programming")
[1] NA
Warning message:
NAs introduced by coercion
> |
```

Fig. 1.32 Demonstrating coercion displaying a warning message

1.6 VECTORS

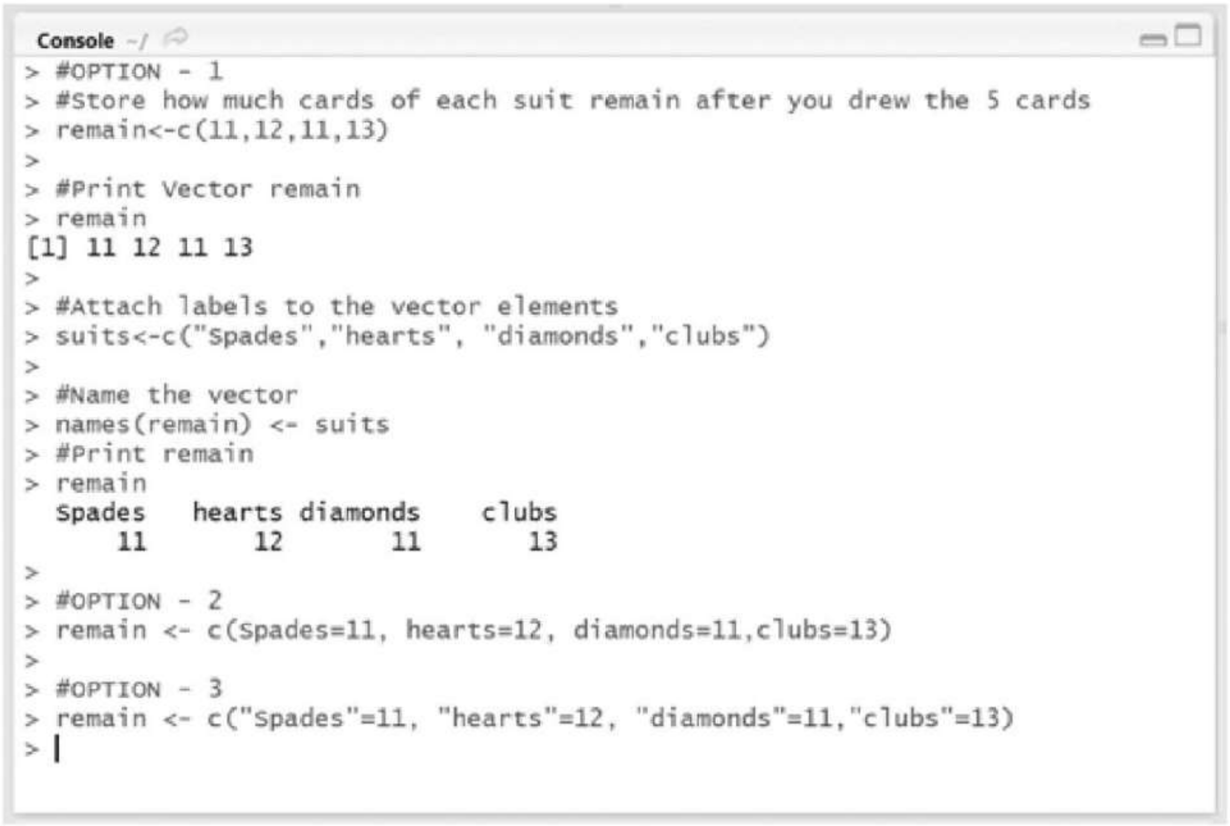
Vectors are the most basic R data objects. A vector is a sequence of data elements of the same data type. There are six types of atomic vectors—*logical*, *integer*, *double*, *complex*, *character*, and *raw*. Programmers can create *character* vectors, *numeric* vectors, *logical* vectors, and many more.

1.6.1 Creating and Naming Vectors

A function *c*() is used to create a vector in R, which further allows users to combine values into a vector.

Let us create vectors considering an example of a playing card game, where users need to record the suit of five cards drawn from a deck. A possible outcome of the corresponding vector will contain the type of the card drawn.

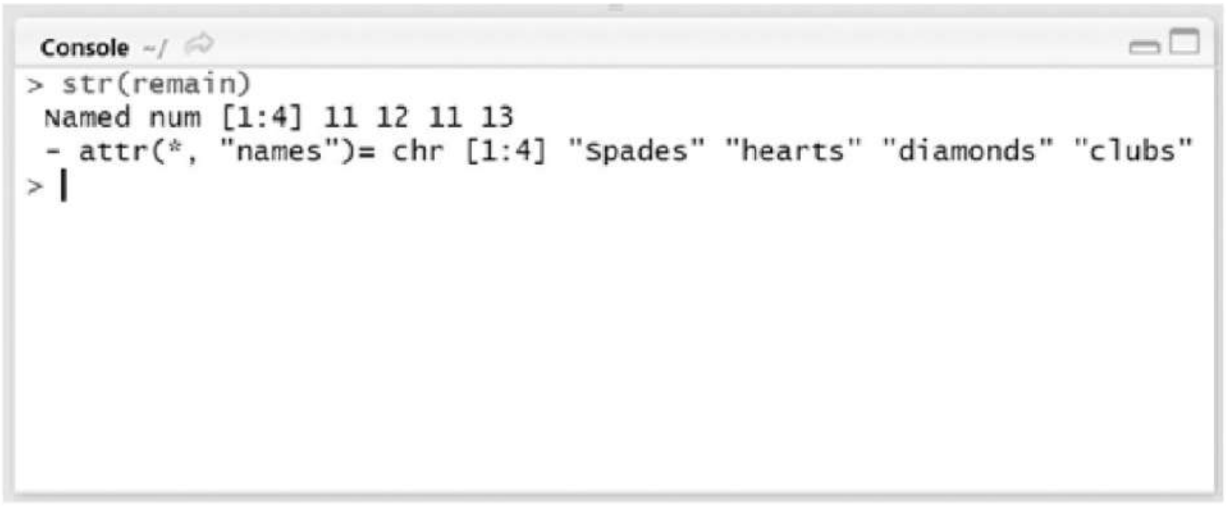
Figure 1.33 shows the various options for creating a new vector. Assume that the user has created a new character vector called *drawn_suits*. The users are allowed to ascertain whether the given variable is a vector or not by running a command called *is.vector*(); for example if users run *is.vector*(*drawn_suits*), the output *True* or *False* indicates whether *drawn_suits* is a vector or not. Figure 1.33 shows an example of creating a vector of integers to store the remaining number of cards from each suit after the user draws 5 cards, where the vector is assigned a new name *remain*. The figure indicates that there are 11 Spades, 12 Hearts, 11 Diamonds, and all 13 Clubs in the deck.



```
> #OPTION - 1
> #Store how much cards of each suit remain after you drew the 5 cards
> remain<-c(11,12,11,13)
>
> #Print Vector remain
> remain
[1] 11 12 11 13
>
> #Attach labels to the vector elements
> suits<-c("Spades","hearts", "diamonds","clubs")
>
> #Name the vector
> names(remain) <- suits
> #Print remain
> remain
  Spades   hearts diamonds    clubs
      11       12       11       13
>
> #OPTION - 2
> remain <- c(Spades=11, hearts=12, diamonds=11,clubs=13)
>
> #OPTION - 3
> remain <- c("Spades"=11, "hearts"=12, "diamonds"=11,"clubs"=13)
> |
```

Fig. 1.33 Creation of a new vector

R allows programmers to name the data elements of the vector. Naming in R is just a way of attaching labels to the vector elements to avoid any kind of ambiguity and to make the program more readable. R uses the *names*() function to name the vector elements. The basic way of naming the vector elements allows creating the character vector, namely *suits*, that contains the strings “spades”, “hearts”, “diamonds”, and “clubs”, and then setting the names of the elements in *remain* to the strings in *suits*. If the user prints the variable *remain* in the console, the names of the vector elements followed by the actual vector elements are printed on the console as shown in Fig. 1.33. Similarly, *Option – 2* and *Option – 3* illustrate the different ways of naming the data elements in one go without using the *names*() function. The result is same in all these three cases. There is one more function in R called *str*(), which displays the complete structure of the R object with the details of all the variables and their data types as illustrated in Fig 1.34.

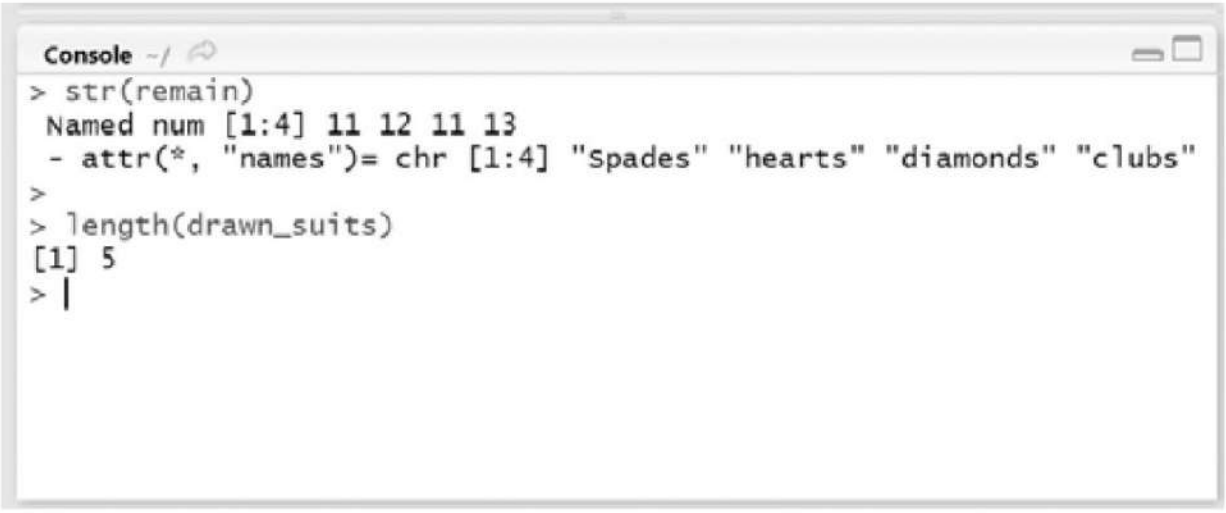


```
Console ~/
> str(remain)
 Named num [1:4] 11 12 11 13
 - attr(*, "names")= chr [1:4] "spades" "hearts" "diamonds" "clubs"
> 
```

Fig. 1.34 Naming the vector elements

Vector Length

A single variable in R is actually a vector of length 1. R provides a function utility named *length*() to determine the length of the vector. Figure 1.35 shows that the length of the vector *drawn_suits* is 5, as it contains five cards taken from a deck of four types.

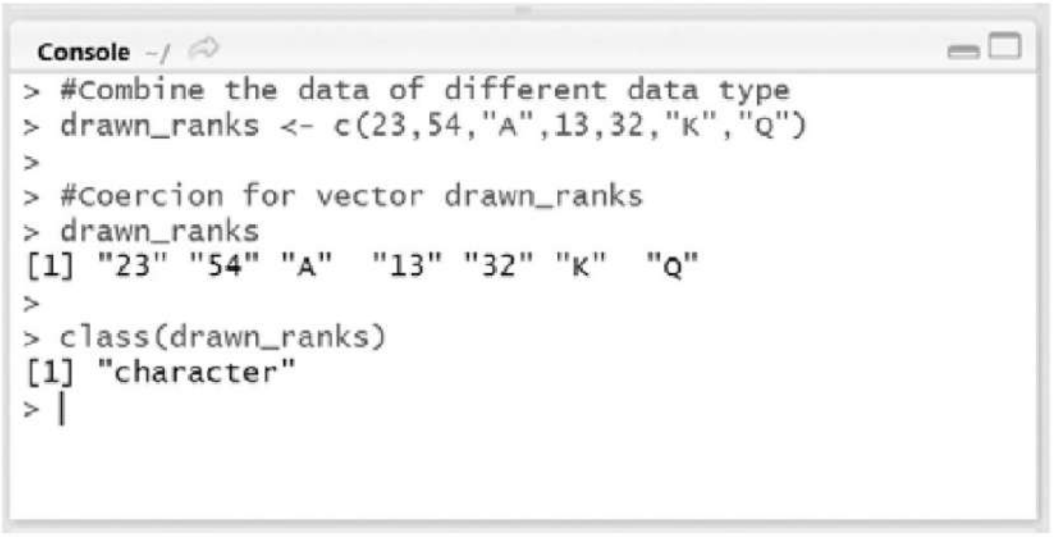


```
Console ~/
> str(remain)
 Named num [1:4] 11 12 11 13
 - attr(*, "names")= chr [1:4] "spades" "hearts" "diamonds" "clubs"
>
> length(drawn_suits)
[1] 5
> 
```

Fig. 1.35 Determining the length of a vector

Coercion of Vector Elements

A vector in R can only hold elements of the same type, which means that users cannot have a vector that contains both *logical* and *numeric* data types. If the user wants to build a mixed vector that contains both integers and characters, then automatically, R performs *coercion* to make sure that the vector contains elements of the same type. For example, assume that the user wants to record the suits drawn from a deck of cards and rank them as per their order in the game. Then the user might want to combine the result of drawing 8 cards in this manner, creating a vector called *drawn_ranks*. However, if the vector *drawn_ranks* is inspected, the whole vector is coerced to characters that is eventually converted into a homogeneous character vector as shown in Fig. 1.36. R handles this conversion automatically by upgrading all *logical* to *numeric* and *numeric* to character, as and when necessary. However, it may also be dangerous some times while handling vectors in calculations.



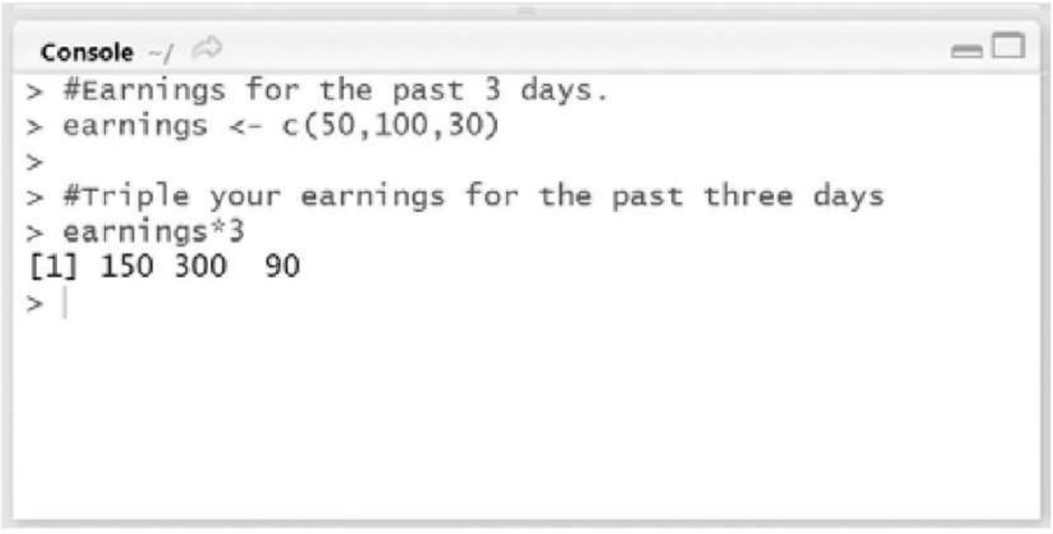
```
> #Combine the data of different data type
> drawn_ranks <- c(23,54,"A",13,32,"K","Q")
>
> #Coercion for vector drawn_ranks
> drawn_ranks
[1] "23" "54" "A"  "13" "32" "K"  "Q"
>
> class(drawn_ranks)
[1] "character"
> |
```

Fig. 1.36 Coercion of vector elements

1.6.2 Vector Arithmetic

R allows programmers to perform arithmetic calculations with vectors and the operations are applied element by element.

Assume we have a vector containing the gambling earnings for the past 3 days. Now a person promises to triple these earnings if we beat him in one round of poker. Now if we want to calculate the expected earnings for each of the past three days, we just need to multiply the earnings for the last three days by 3. Figure 1.37 shows this simple vector arithmetic by multiplying the vector variable *earnings* by 3, where the vector *earnings* has been created with the three days of earnings in ₹.



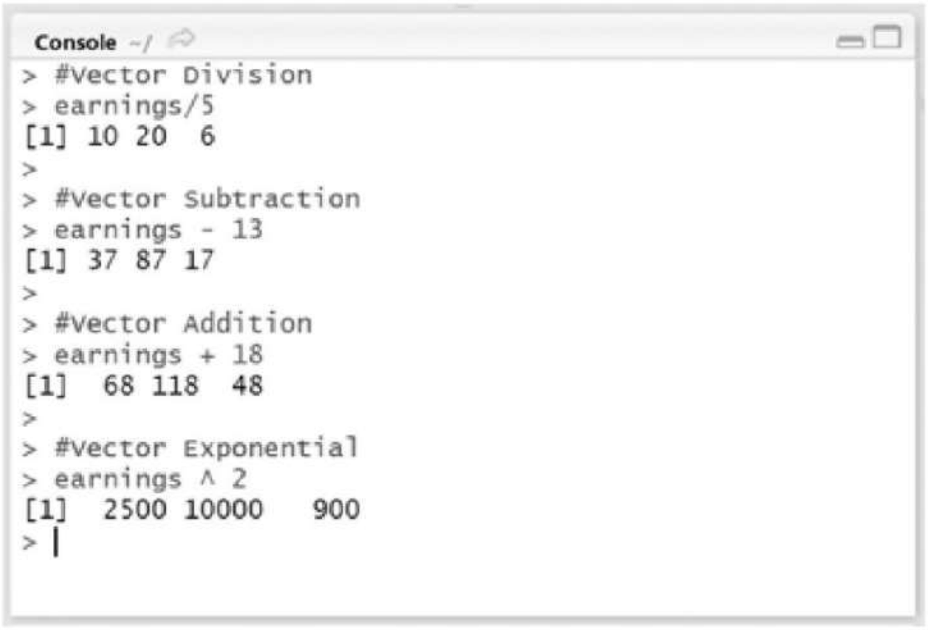
```
> #Earnings for the past 3 days.
> earnings <- c(50,100,30)
>
> #Triple your earnings for the past three days
> earnings*3
[1] 150 300  90
> |
```

Fig. 1.37 Vector arithmetic

Example 1.10 Illustrate summation, subtraction, multiplication, and division operations on vectors using the *earnings* vector from Fig. 1.37.

Solution:

R allows summation, subtraction, multiplication and division operations on vectors. Figure 1.38 illustrates the division of the *earnings* vector by 5, subtracting 13 from the *earnings* vector elements, adding 18 to the vector elements, and finding the exponential of 2 of the elements of *earnings*. The key point here is that the basic vector *earnings* will not change its original value due to these arithmetic operations.



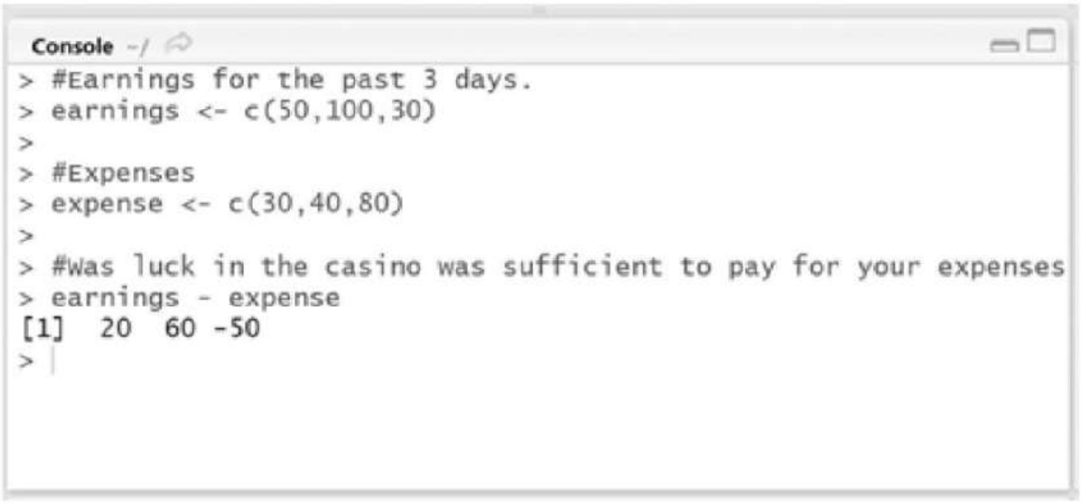
```
> #Vector Division
> earnings/5
[1] 10 20  6
>
> #Vector Subtraction
> earnings - 13
[1] 37 87 17
>
> #Vector Addition
> earnings + 18
[1]  68 118  48
>
> #Vector Exponential
> earnings ^ 2
[1]  2500 10000   900
> |
```

Fig. 1.38 Summation, subtraction, multiplication, and division operations on vectors

Example 1.11 Consider a scenario where the user decides to splurge his/her earnings by shopping, spends some money every day, and records the day-wise expenses in a vector. Calculate the day-wise savings using vector subtraction.

Solution:

Figure 1.39 shows the new vector balance after subtracting the *expense* from *earnings*. Here, the arithmetic is done element-wise, where the subtraction is performed element by element.



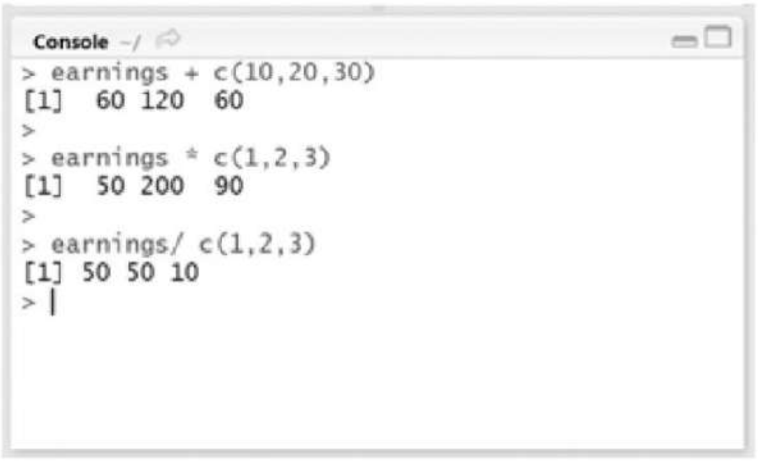
```
> #Earnings for the past 3 days.
> earnings <- c(50,100,30)
>
> #Expenses
> expense <- c(30,40,80)
>
> #Was luck in the casino was sufficient to pay for your expenses
> earnings - expense
[1]  20  60 -50
> |
```

Fig. 1.39 Subtraction of vectors

Example 1.12 Illustrate addition, multiplication, and division between vectors.

Solution:

Figure 1.40 shows vector arithmetic, where the operations of addition, multiplication, and division are performed between two vectors. In all these cases, one needs to ensure that the two vectors are of same data type and length.



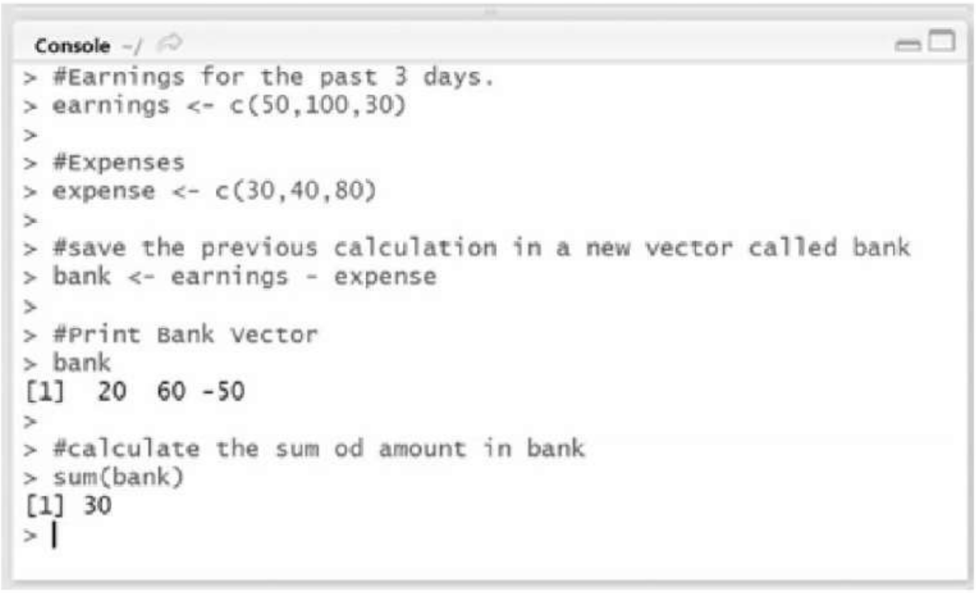
```
> earnings + c(10,20,30)
[1]  60 120  60
>
> earnings * c(1,2,3)
[1]  50 200  90
>
> earnings/ c(1,2,3)
[1] 50 50 10
> |
```

Fig. 1.40 Addition, multiplication, and division between vectors

Example 1.13 Enumerate multiplication and division operations between matrices and vectors in R console.

Solution:

Multiplication and division in R are different from the traditional matrix and vector operations, where the multiplication of two vectors can result in a single scalar or a matrix. Figure 1.41 shows users how to check their bank accounts' status after these three days of earnings in the city of Bangalore.



```
> #Earnings for the past 3 days.
> earnings <- c(50,100,30)
>
> #Expenses
> expense <- c(30,40,80)
>
> #save the previous calculation in a new vector called bank
> bank <- earnings - expense
>
> #Print Bank Vector
> bank
[1]  20  60 -50
>
> #calculate the sum od amount in bank
> sum(bank)
[1] 30
> |
```

Fig. 1.41 Multiplication and division between matrices and vectors

Users can save the previous calculation in a new vector called *bank* and use the *sum*() function on it to calculate the sum of all its elements. Instead of subtracting expenses from earnings to compare them, which gives users positive and negative numbers, users could also use relational operators to know when earnings