SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY

Unit-I-R Programming – SCS1621

Unit – I

Introduction to R - History and fundamentals of R, Installation and use of R / R Studio / R Shiny, Installing R packages, R – Nuts and Bolts -Getting Data In and Out - Control Structures and Functions-Loop Functions-Data Manipulation- String Operations- Matrix Operations.

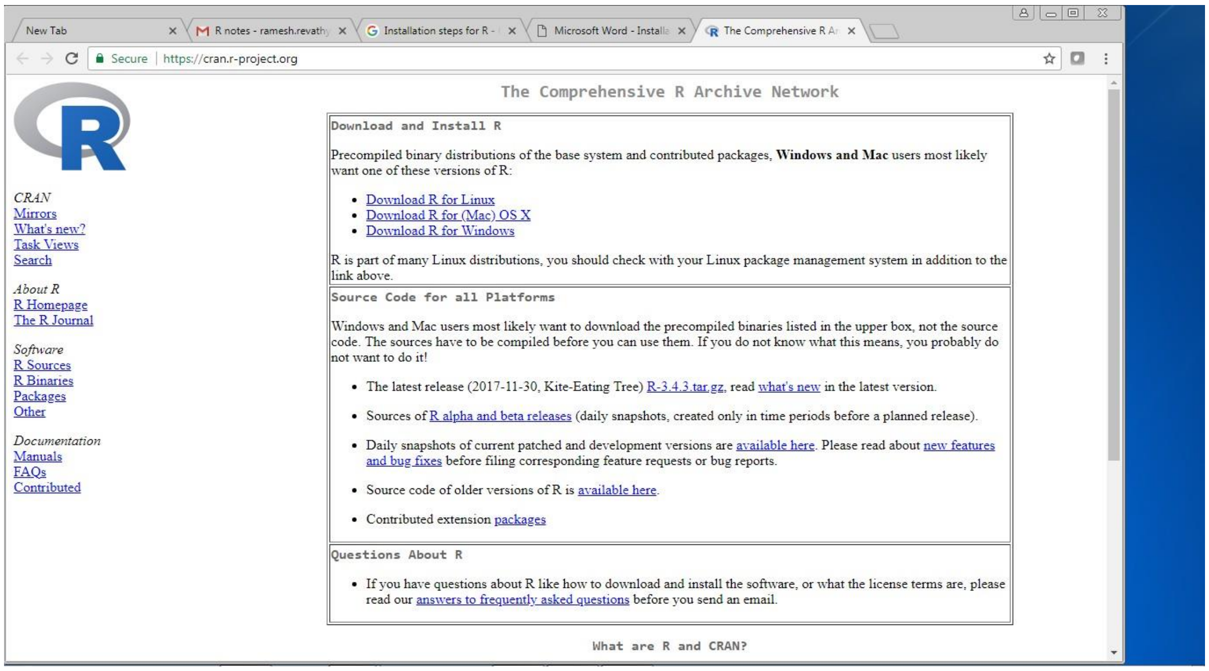
BASICS OF R

History of R

- **R** is a programming language and free software environment for statistical computing andgraphics that is supported by the R Foundation for Statistical Computing.
- R is an implementation of the S programming language combined with lexicalscoping semantics inspired by Scheme.
- S was created by John Chambers in 1976, while at Bell Labs.
- There are some important differences, but much of the code written for S runs unaltered.[15]
- R was created by Ross Ihaka and Robert Gentleman[16] at the University of Auckland, New Zealand, and is currently developed by the *R Development Core Team*, of which Chambersis a member.
- R is named partly after the first names of the first two R authors and partly as a play on the name of S.
- The project was conceived in 1992, with an initial version released in 1995 and a stable beta version in 2000.

Installation of R

1. Download the R installer from https://cran.r-project.org/


New Tab
R notes - ramesh.revathy
Installation steps for R -
Microsoft Word - Installa
The Comprehensive R Ar
Secure
https://cran.r-project.org

The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2017-11-30, Kite-Eating Tree) R-3.4.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

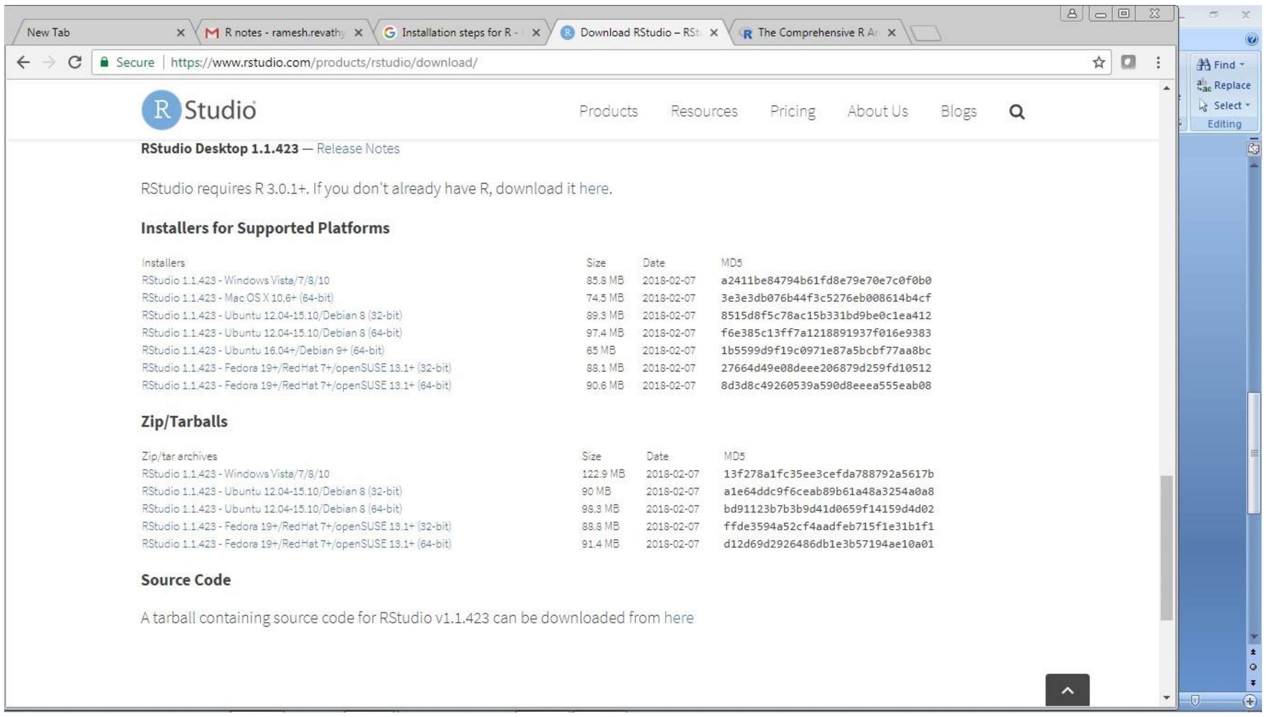
What are R and CRAN?

Fig 1.1:Installation of R

2.Run the installer. Default settings are fine. If you do not have admin rights on your laptop, thenask you local IT support. In that case, it is important that you also ask them to give you full permissions to the R directories. Without this, you will not be able to install additional packages later.

Installation of R Studio

1. Download RStudio: https://www.rstudio.com/products/rstudio/download/



RStudio Desktop 1.1.423 — Release Notes

RStudio requires R 3.0.1+. If you don't already have R, download it here.

Installers for Supported Platforms

Installers	Size	Date	MD5
RStudio 1.1.423 - Windows Vista/7/8/10	85.8 MB	2018-02-07	a2411be84794b61fd8e79e70e7c0f0b0
RStudio 1.1.423 - Mac OS X 10.6+ (64-bit)	74.5 MB	2018-02-07	3e3e3db076b44f3c5276eb008614b4cf
RStudio 1.1.423 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	89.3 MB	2018-02-07	8515d8f5c78ac15b331bd9be0c1ea412
RStudio 1.1.423 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	97.4 MB	2018-02-07	f6e385c13ff7a1218891937f016e9383
RStudio 1.1.423 - Ubuntu 16.04+/Debian 9+ (64-bit)	65 MB	2018-02-07	1b5599d9f19c0971e87a5bcbf77aa8bc
RStudio 1.1.423 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.1 MB	2018-02-07	27664d49e08deee206879d259fd10512
RStudio 1.1.423 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	90.6 MB	2018-02-07	8d3d8c49260539a590d8eeea555eab08

Zip/Tarballs

Zip/tar archives	Size	Date	MD5
RStudio 1.1.423 - Windows Vista/7/8/10	122.9 MB	2018-02-07	13f278a1fc35ee3cefda788792a5617b
RStudio 1.1.423 - Ubuntu 12.04-15.10/Debian 8 (32-bit)	90 MB	2018-02-07	a1e64ddc9f6ceab89b61a48a3254a0a8
RStudio 1.1.423 - Ubuntu 12.04-15.10/Debian 8 (64-bit)	98.3 MB	2018-02-07	bd91123b7b3b9d41d0659f14159d4d02
RStudio 1.1.423 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (32-bit)	88.8 MB	2018-02-07	ffde3594a52cf4aadfeb715f1e31b1f1
RStudio 1.1.423 - Fedora 19+/RedHat 7+/openSUSE 13.1+ (64-bit)	91.4 MB	2018-02-07	d12d69d2926486db1e3b57194ae10a01

Source Code

A tarball containing source code for RStudio v1.1.423 can be downloaded from here

Fig 1.2:Installation of R studio

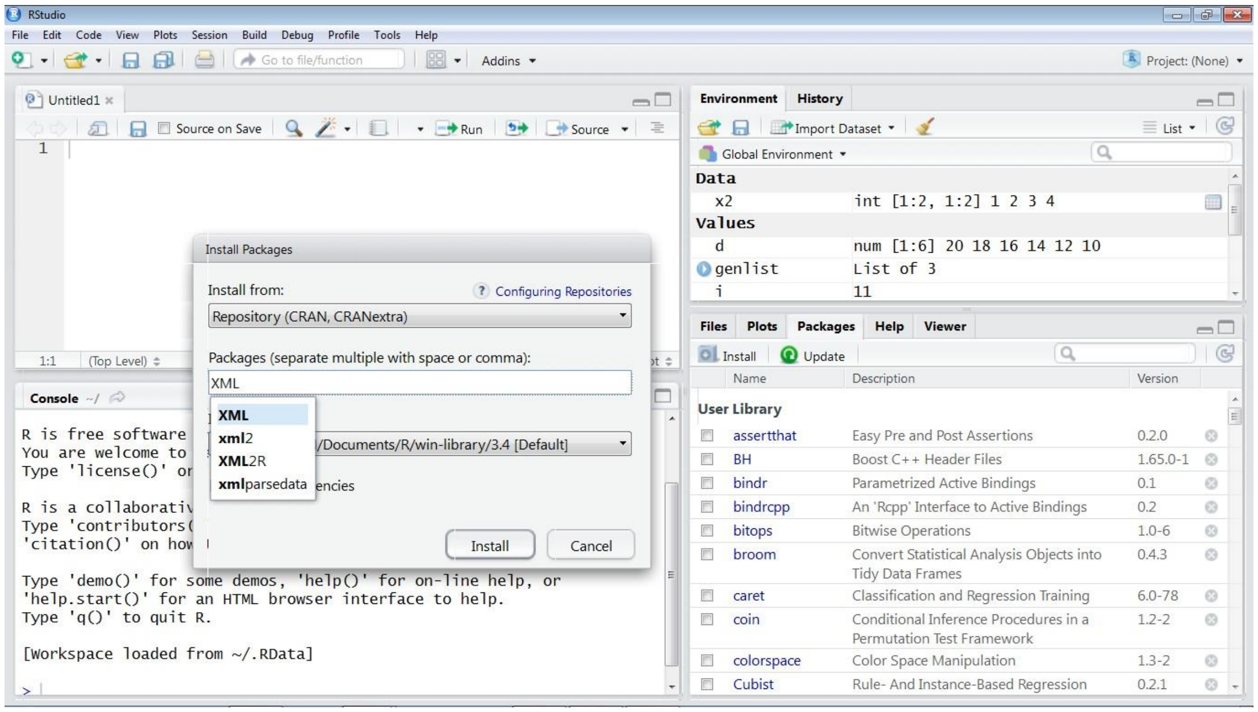
2. Once the installation of R has completed successfully, run the RStudio installer.
3. If you do not have administrative rights on your laptop, step 2 may fail. Ask your ITSupport or download a pre-built zip archive of RStudio which doesn't need installing.
4. The link for this is towards the bottom of the download page, highlighted in Image
5. Download the appropriate archive for your system (Windows/Linux only – the Macversion can be installed into your personal "Applications" folder without admin rights).
6. Double clicking on the zip archive should automatically unpack it on most Windowsmachines.

Installing R packages

There are three options to install packages.

Option 1:

Click on the tab ' Packages' then 'Install' as shown in figure 1.3. Click on Install toinstall R packages.



Install from: Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma): XML

/Documents/R/win-library/3.4 [Default]

```
R is free software
You are welcome to
Type 'license()' or

R is a collaborativ
Type 'contributors(
'citation()' on how

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]

>
```

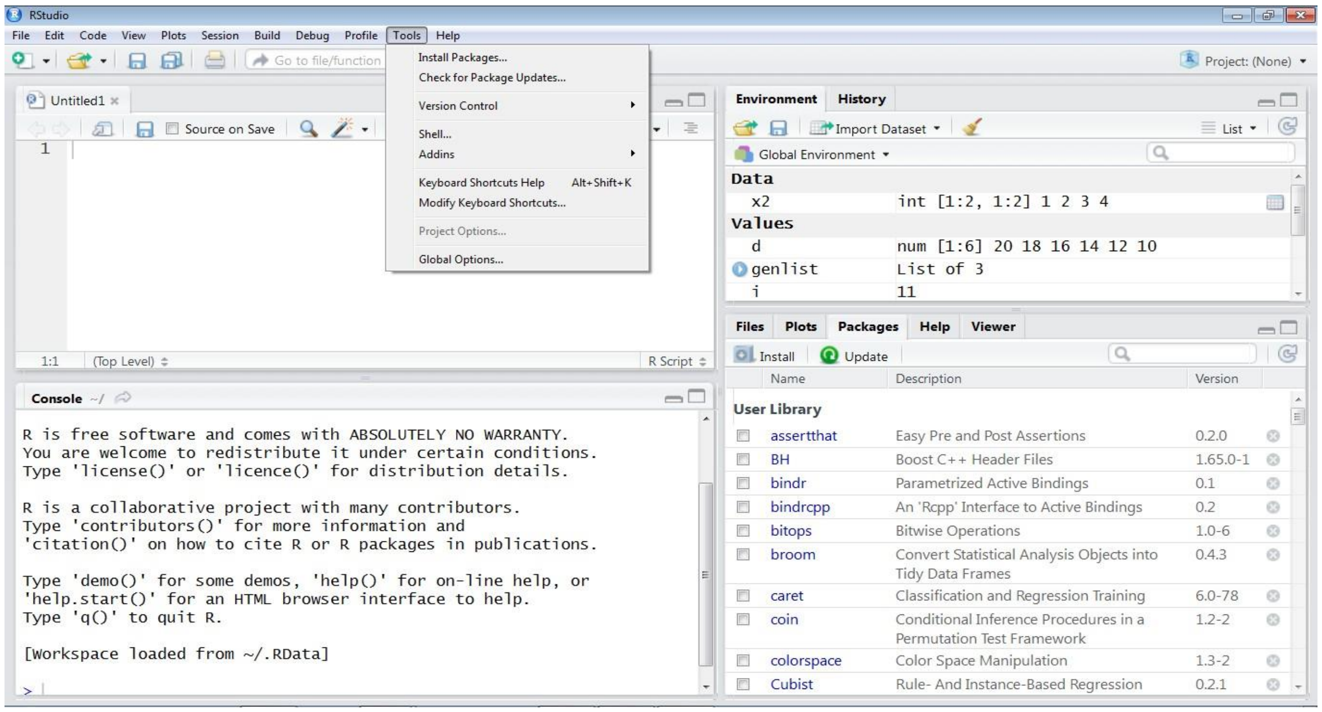
```
Data
  x2          int [1:2, 1:2] 1 2 3 4
Values
  d           num [1:6] 20 18 16 14 12 10
  genlist     List of 3
  i           11
```

Name	Description	Version
User Library		
assertthat	Easy Pre and Post Assertions	0.2.0
BH	Boost C++ Header Files	1.65.0-1
bindr	Parametrized Active Bindings	0.1
bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2
bitops	Bitwise Operations	1.0-6
broom	Convert Statistical Analysis Objects into Tidy Data Frames	0.4.3
caret	Classification and Regression Training	6.0-78
coin	Conditional Inference Procedures in a Permutation Test Framework	1.2-2
colorspace	Color Space Manipulation	1.3-2
Cubist	Rule- And Instance-Based Regression	0.2.1

Fig 1.3:Install Packages

Option 2:

Tools -> Install packages. As shown in Fig 1.4.



```
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]

>
```

```
Data
  x2          int [1:2, 1:2] 1 2 3 4
Values
  d           num [1:6] 20 18 16 14 12 10
  genlist     List of 3
  i           11
```

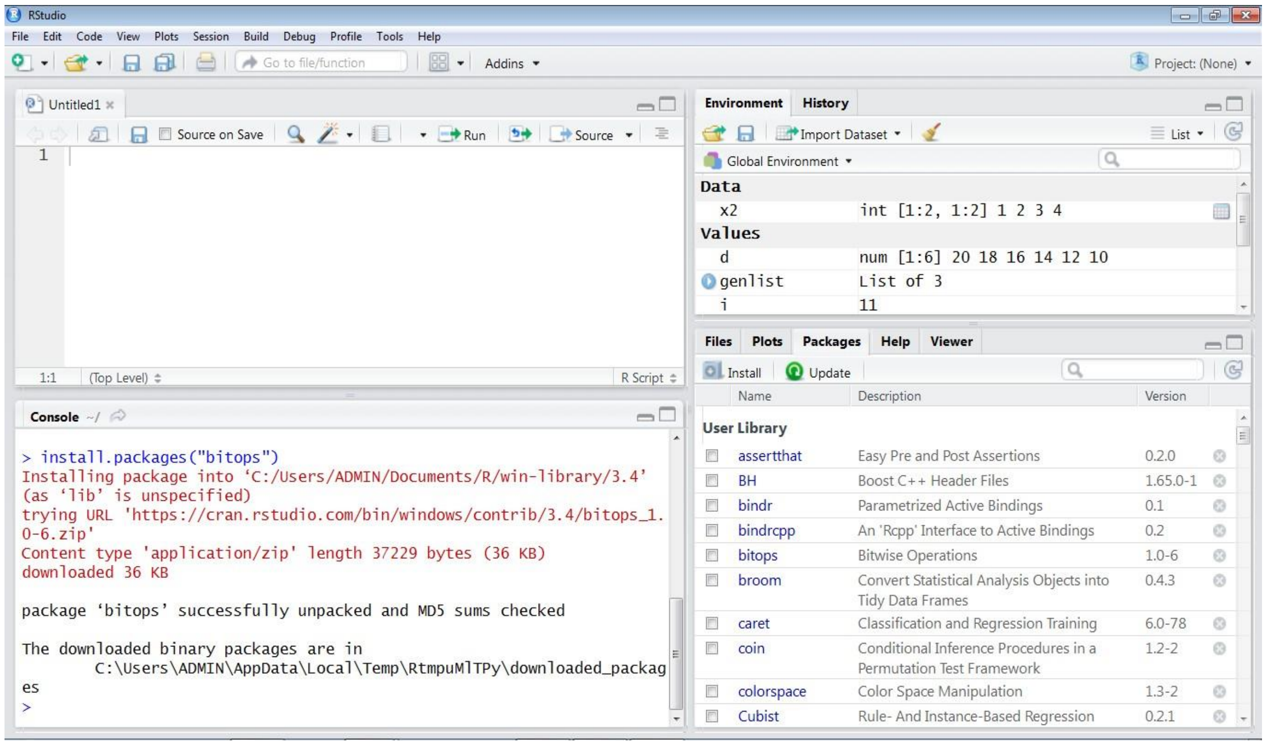
Name	Description	Version
User Library		
assertthat	Easy Pre and Post Assertions	0.2.0
BH	Boost C++ Header Files	1.65.0-1
bindr	Parametrized Active Bindings	0.1
bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2
bitops	Bitwise Operations	1.0-6
broom	Convert Statistical Analysis Objects into Tidy Data Frames	0.4.3
caret	Classification and Regression Training	6.0-78
coin	Conditional Inference Procedures in a Permutation Test Framework	1.2-2
colorspace	Color Space Manipulation	1.3-2
Cubist	Rule- And Instance-Based Regression	0.2.1

Fig 1.4:Installation of Packages

Option 3:

Use the following command to install packages from R console.

Install.packages("package name")



```
Data
  x2          int [1:2, 1:2] 1 2 3 4
Values
  d           num [1:6] 20 18 16 14 12 10
  genlist     List of 3
  i           11
```

```
> install.packages("bitops")
Installing package into 'C:/Users/ADMIN/Documents/R/win-library/3.4'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/3.4/bitops_1.
0-6.zip'
Content type 'application/zip' length 37229 bytes (36 KB)
downloaded 36 KB

package 'bitops' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
        C:\Users\ADMIN\AppData\Local\Temp\RtmpuMlTPy\downloaded_packag
es
>
```

Name	Description	Version
User Library		
assertthat	Easy Pre and Post Assertions	0.2.0
BH	Boost C++ Header Files	1.65.0-1
bindr	Parametrized Active Bindings	0.1
bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2
bitops	Bitwise Operations	1.0-6
broom	Convert Statistical Analysis Objects into Tidy Data Frames	0.4.3
caret	Classification and Regression Training	6.0-78
coin	Conditional Inference Procedures in a Permutation Test Framework	1.2-2
colorspace	Color Space Manipulation	1.3-2
Cubist	Rule- And Instance-Based Regression	0.2.1

Fig 1.5:Installation of packages

Note: Check that the packages are installed by typing 'library(<packagename>)'

Nuts and Bolts of R

- R is the most comprehensive statistical analysis package available.It incorporates all of the standard statistical tests, models, and analyses, as well as providing a comprehensive language for managing and manipulating data.
  - R is a programming language and environment developed for statistical analysis by practising statisticians and researchers. It reflects well on a very competent community of computational statisticians.
  - The graphical capabilities of R are outstanding, providing a fully programmable graphics language that surpasses most other statistical and

graphical packages. ˆ

- R is free and open source software, allowing anyone to use and, importantly, to modify it. R is licensed under the GNU General Public License, with copyright held by The R Foundation for Statistical Computing.
- R has no license restrictions (other than ensuring our freedom to use it at our own discretion), and so we can run it anywhere and at any time.
- Anyone can provide new packages, and the wealth of quality packages available for R is a testament to this approach to software development and sharing.
- R has over 4800 packages available from multiple repositories specializing in topics like econometrics, data mining, spatial analysis, and bio-informatics.
- R is cross-platform. R runs on many operating systems and different hardware. It is popularlyused on GNU/Linux, Macintosh, and Microsoft Windows, running on both 32 and 64 bit processors. ˆ
- R plays well with many other tools, importing data, for example, from CSV files, SAS, and SPSS, or directly from Microsoft Excel, Microsoft Access, Oracle, MySQL, and SQLite.
- It can also produce graphics output in PDF, JPG, PNG, and SVG formats, and table output for LATEX and HTML.
- R has active user groups .

Data In and Out

There are a few principal functions reading data into R.

- read.table(), read.csv(): for reading tabular data
- readLines( ) : for reading lines of a text file
- source () : for reading in R code files (inverse of dump)
- dget () : for reading in R code files (inverse of dput)
- load () : for reading in saved workspaces
- unserialize () : for reading single R objects in binary form

There are analogous functions for writing data to files

- write.table() : for writing tabular data to text files (i.e. CSV) or connections
- writeLines() : for writing character data line-by-line to a file or connection
- dump() : for dumping a textual representation of multiple R objects
- dput() : for outputting a textual representation of an R object
- save() :for saving an arbitrary number of R objects in binary format (possiblycompressed) to a file.
- Serialize() :for converting an R object into a binary format for outputting to a connection (orfile).

Control Structures in R

Control Structures can be divided into three categories.

1. Conditional

statements.2.Looping

Statements 3.Jump

Statements

Conditional Statements:

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

Example:If,If-Else,If-ElseIf Ladder,switch

If:

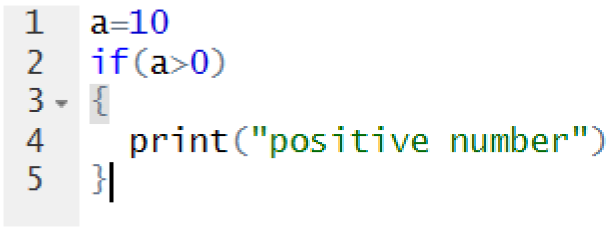
Description: An **if** statement consists of a Boolean expression followed by oneor more statements.

```
Syntax:      If(condition)
                 {
```

Statement

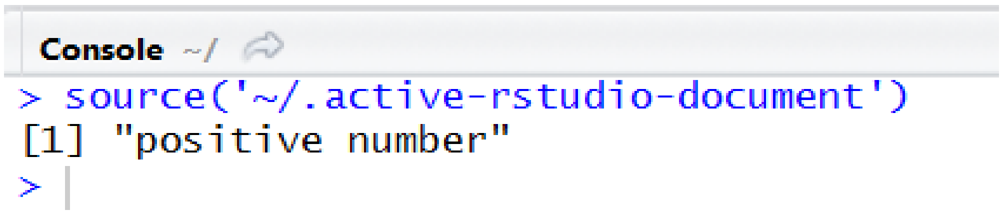
}

Example:Check for Positive Number



```
a=10
if(a>0)
{
  print("positive number")
}
```

Output :



```
> source('~/.active-rstudio-document')
[1] "positive number"
>
```

If-Else

Description: An **if…else** statement contains the same elements as an ifstatement and then some extra:

- The keyword else, placed after the first code block
- Second block of code, contained within braces, that has to be carried out ifand only if the result of the condition in the if() statement is FALSE

Syntax:if(condition)

{

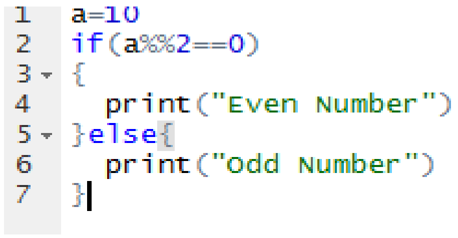
Statement1

}else{

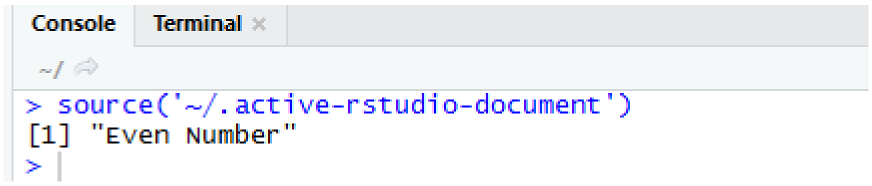
Statement 2

}

Example:Check for Odd or Even Number


```
a=10
if(a%%2==0)
{
  print("Even Number")
}else{
  print("Odd Number")
}
```

Output:



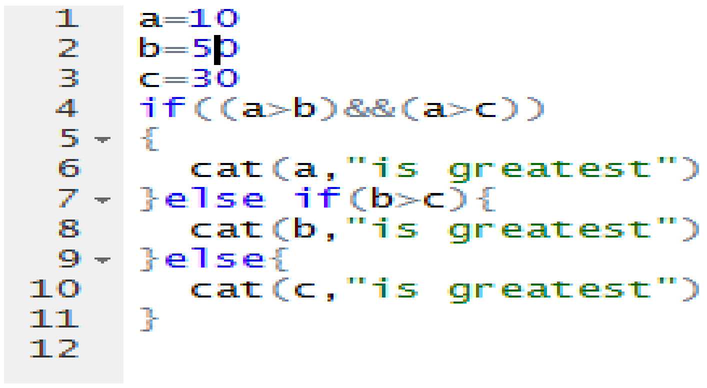
```
> source('~/.active-rstudio-document')
[1] "Even Number"
>
```

If-Else if Ladder

If-Else statements can be chained using If-Else if Ladder

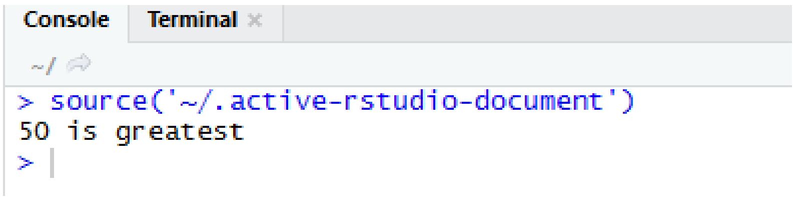
```
Syntax:if(condition)
            {
                    Statement1
             }else if{
                    Statement 2
             }else{
                    Statement 3
             }
```

Example: Greatest of three numbers



```
a=10
b=50
c=30
if((a>b)&&(a>c))
{
   cat(a,"is greatest")
}else if(b>c){
   cat(b,"is greatest")
}else{
   cat(c,"is greatest")
}
```

Output:



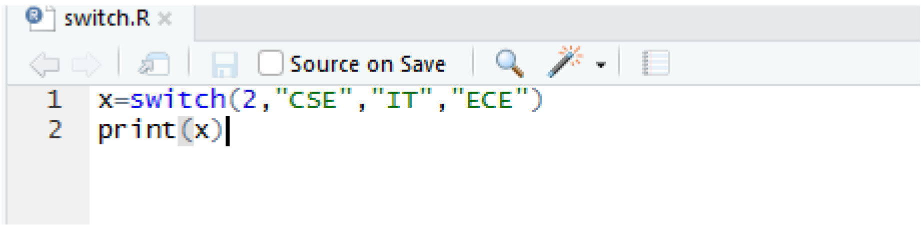
```
> source('~/.active-rstudio-document')
50 is greatest
> 
```

Switch:

A **switch** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

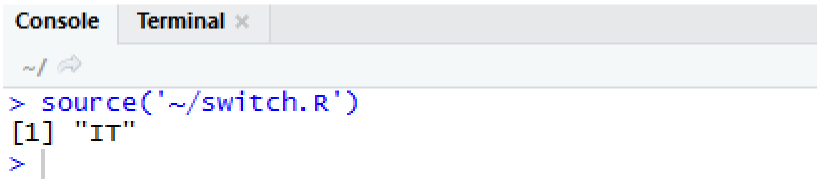
Syntax: switch(Expression, "Option 1", "Option 2", "Option 3"............................... "Option N")

Example:Print Department names using Switch



```
x=switch(2,"CSE","IT","ECE")
print(x)
```

Output:



```
> source('~/switch.R')
[1] "IT"
>
```

Looping Statements in R

There may be a situation when you need to execute a block of code several number of times. A loop statement allows us to execute a statement or group ofstatements multiple times.

Example:for,while,Repeat

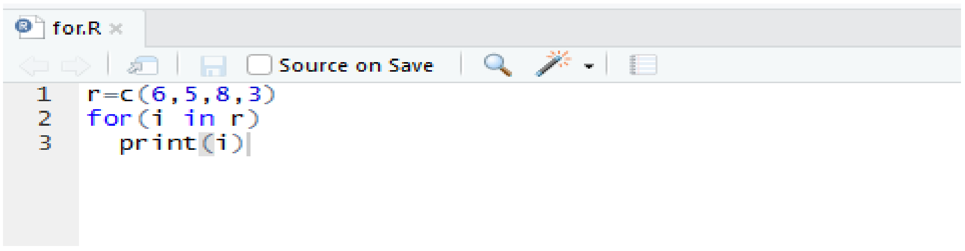
For:

Description: A **For loop** is a repetition control structure that allows you to efficiently write a loop that needs to execute a specific number of times.

Syntax: for(value in vector)

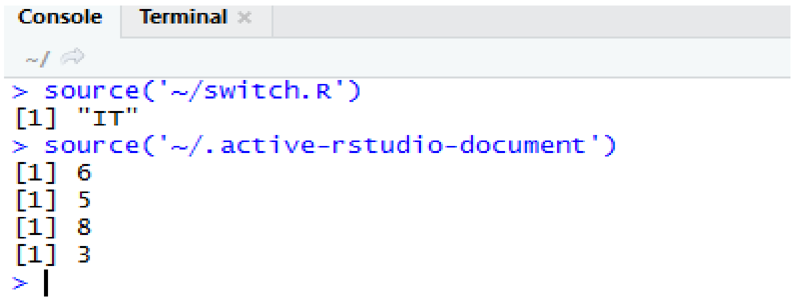
```
{
    Statements
}
```

Example: Printing elements of vector



```
r=c(6,5,8,3)
for(i in r)
  print(i)
```

Output:



```
> source('~/switch.R')
[1] "IT"
> source('~/.active-rstudio-document')
[1] 6
[1] 5
[1] 8
[1] 3
> 
```

While:

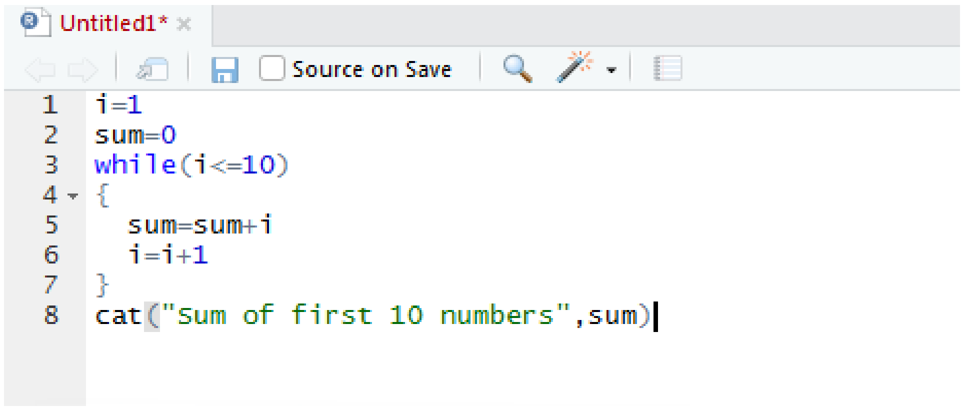
Description: The While loop executes the same code again and again until astop condition is met.

Syntax:while(condition){

Statement

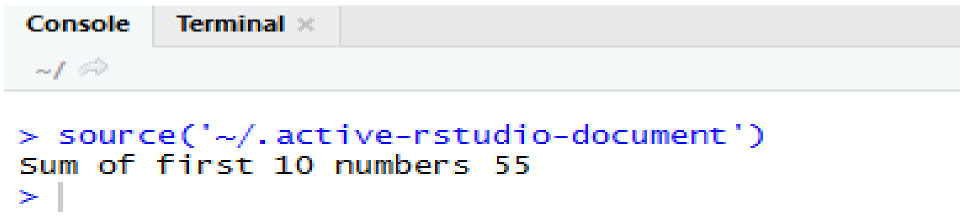
}

Example: Sum of Series



```
i=1
sum=0
while(i<=10)
{
  sum=sum+i
  i=i+1
}
cat("Sum of first 10 numbers",sum)
```

Output:



```
> source('~/.active-rstudio-document')
Sum of first 10 numbers 55
> 
```

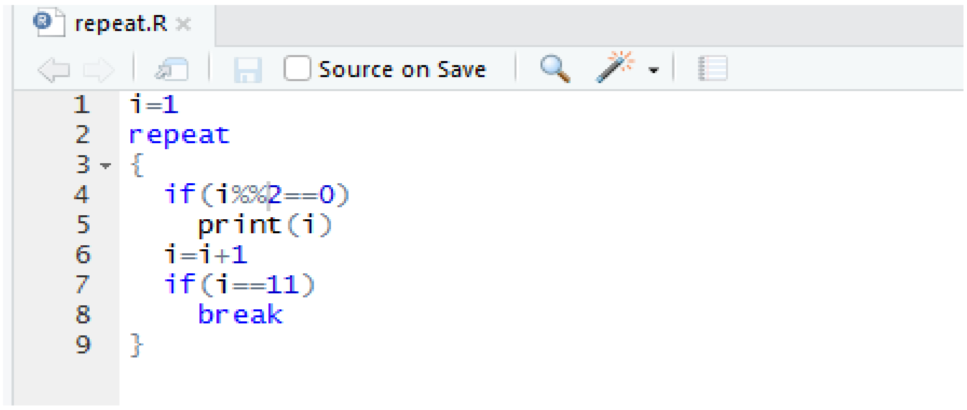
Repeat:

Description:The **Repeat loop** executes the same code again and again until astop condition is met.

Syntax:

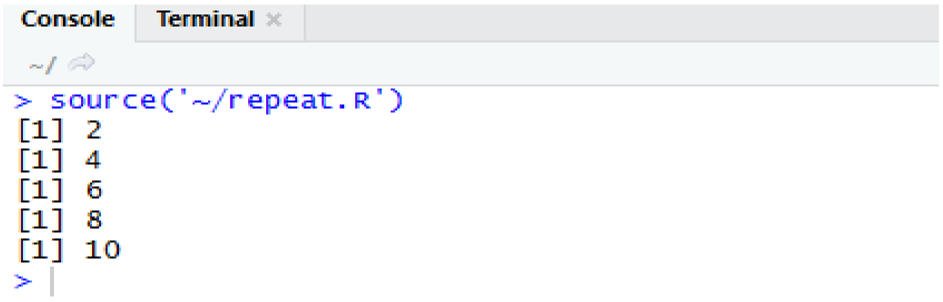
```
repeat
 {
      Commands
       If(condition)
             Break
  }
```

Example:Printing even numbers till 10



```
i=1
repeat
{
  if(i%%2==0)
    print(i)
  i=i+1
  if(i==11)
    break
}
```

Output:



```
> source('~/repeat.R')
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
> 
```

Jump Statements:

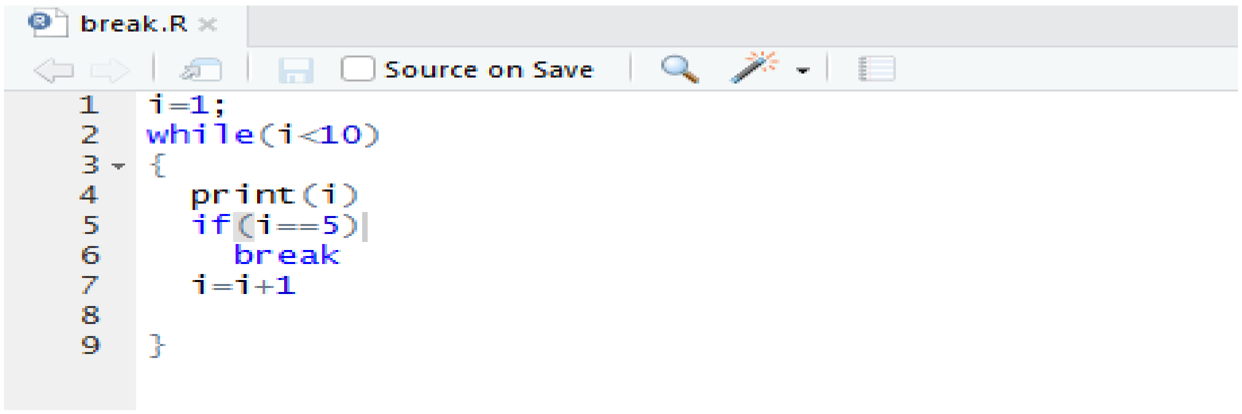
Description: Loop control statements change execution from its normalsequence
Example: break, next

Break:

Terminates the **loop** statement and transfers execution to the statement immediately following the loop.

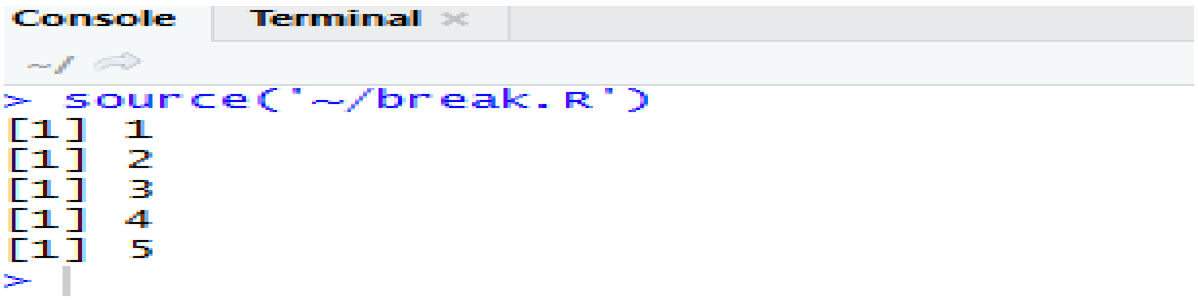
Syntax: break

Example :print numbers till 5



```
i=1;
while(i<10)
{
  print(i)
  if(i==5)
    break
  i=i+1

}
```

Output:



```
> source('~/break.R')
[1]  1
[1]  2
[1]  3
[1]  4
[1]  5
>
```

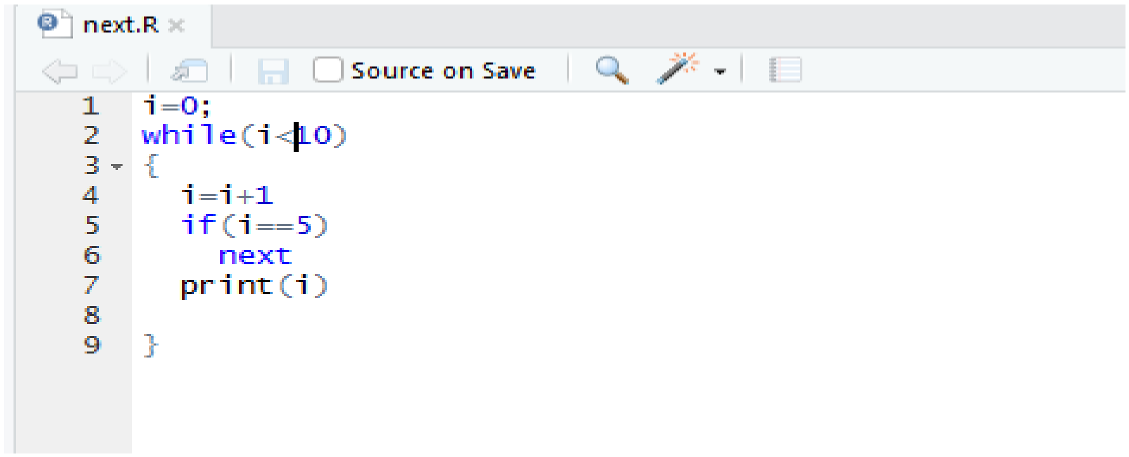
Next:

Description: The **next** statement in R programming language is useful when we want to skip the current iteration of a loop without terminating it. On encountering next, the R parser skips further evaluation and starts next iteration of the loop.

Syntax:

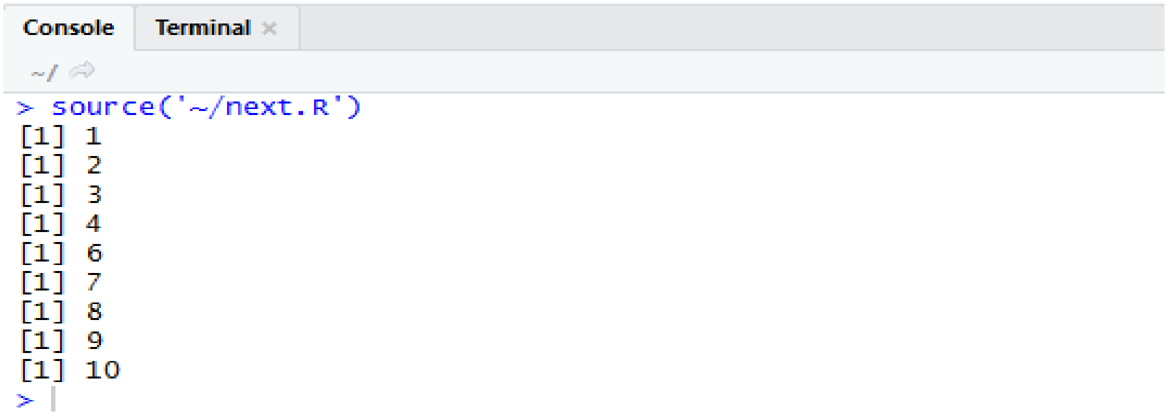
next

Example: Print numbers from 1 to 10 except 5



```
i=0;
while(i<10)
{
  i=i+1
  if(i==5)
    next
  print(i)

}
```

Output:



```
> source('~/next.R')
[1] 1
[1] 2
[1] 3
[1] 4
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
> 
```

Functions in R

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

Syntax:

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

Function name=function(ar1,ar2,……)

{

Function body

}

Function Components:

The different parts of a function are −

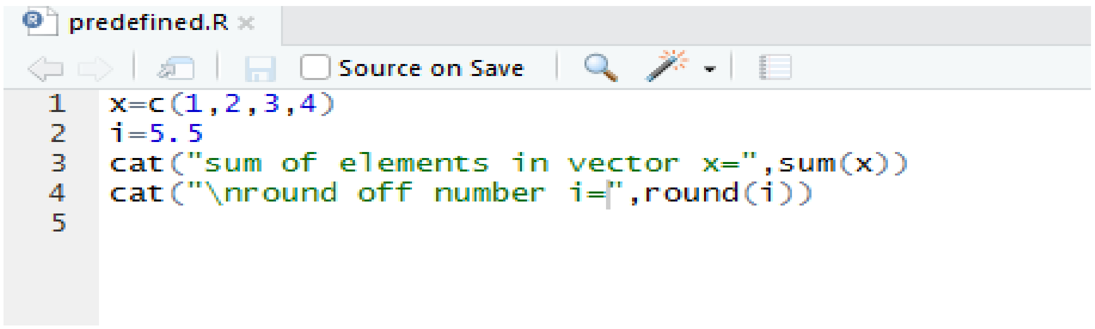
- **Function Name** − This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** − The function body contains a collection of statements that defines what the function does.
- **Return Value** − The return value of a function is the last expression in the function body to be evaluated.

Built-in Functions

R has many **in-built** functions which can be directly called in the program without defining them first.

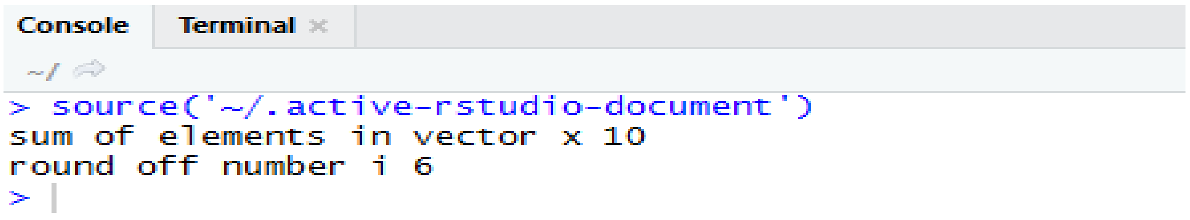
Eg: sum.seq,abs,round .

Example:



```
x=c(1,2,3,4)
i=5.5
cat("sum of elements in vector x=",sum(x))
cat("\nround off number i=",round(i))
```

Output:



```
> source('~/.active-rstudio-document')
sum of elements in vector x 10
round off number i 6
> 
```

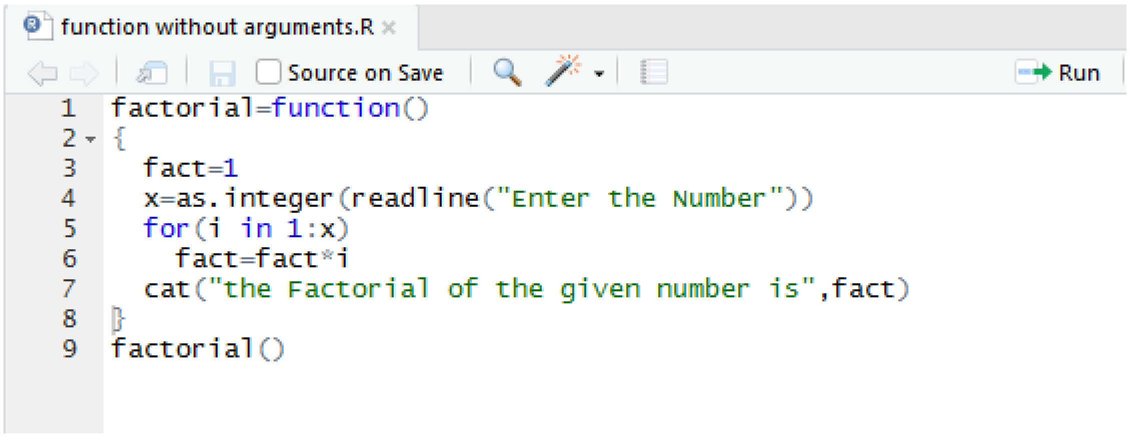
User Defined Function:

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Belowis an example of how a function is created and used.

Function without Arguments:

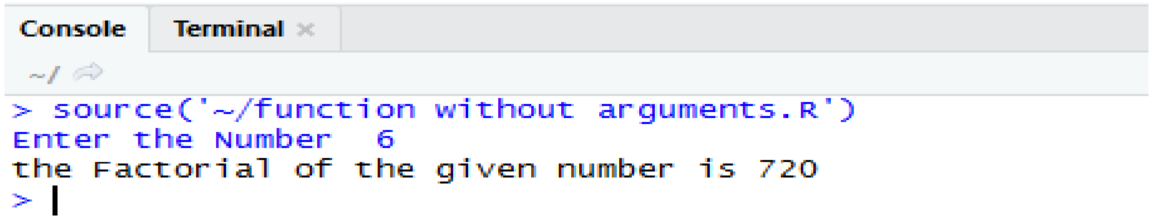
Here the function does not receive any arguments.

Example:



```r
factorial=function()
{
  fact=1
  x=as.integer(readline("Enter the Number"))
  for(i in 1:x)
    fact=fact*i
  cat("the Factorial of the given number is",fact)
}
factorial()
```

Output:

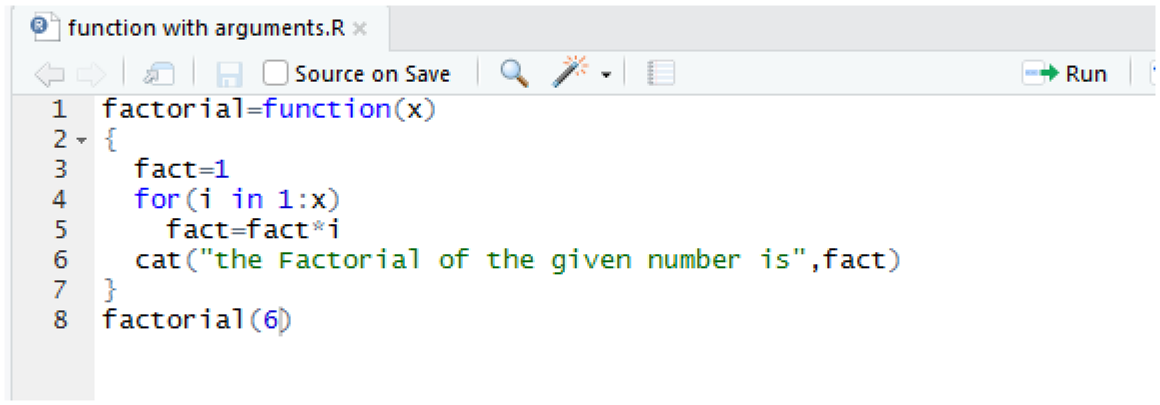


```
> source('~/function without arguments.R')
Enter the Number  6
the Factorial of the given number is 720
> 
```

Function with Arguments:

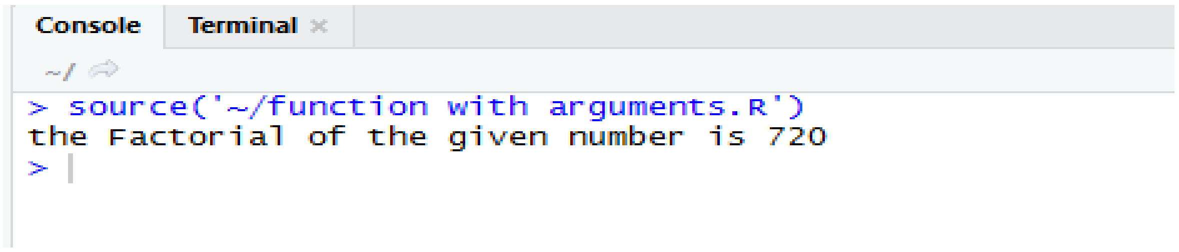
The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

Example:



```
factorial=function(x)
{
  fact=1
  for(i in 1:x)
    fact=fact*i
  cat("the Factorial of the given number is",fact)
}
factorial(6)
```

Output:

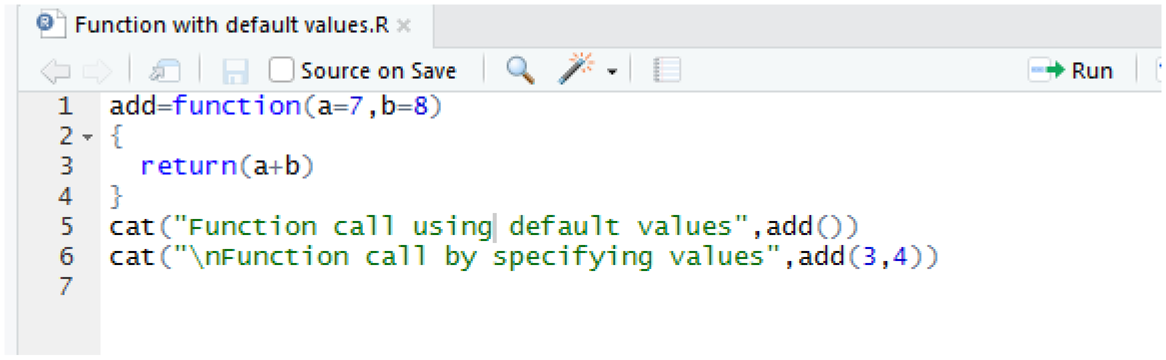


```
> source('~/function with arguments.R')
the Factorial of the given number is 720
> 
```

Function with Default Argument

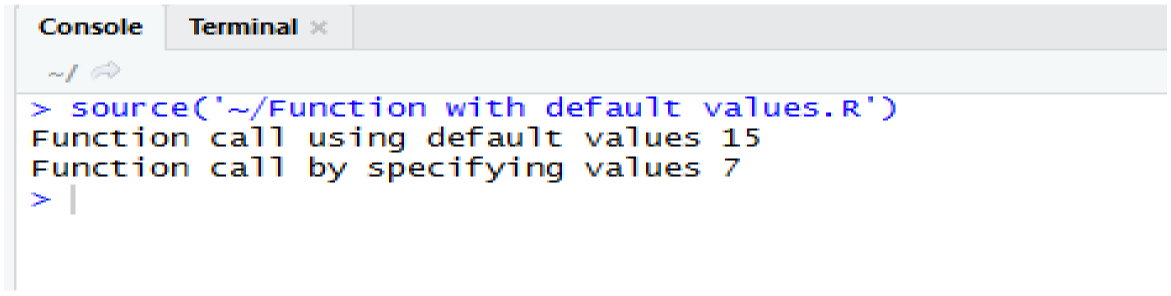
We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result. But we can also call such functions by supplying new values of the argument and get non default result.

Example:



```
add=function(a=7,b=8)
{
  return(a+b)
}
cat("Function call using default values",add())
cat("\nFunction call by specifying values",add(3,4))
```

Output:



```
> source('~/Function with default values.R')
Function call using default values 15
Function call by specifying values 7
> 
```

LOOPING FUNCTIONS IN R

The for, while loops can often be replaced by looping functions:

lapply:

Definition:Loop over a list and evaluate a function on each elementSyntax

: lapply(X, FUN, ...)

X

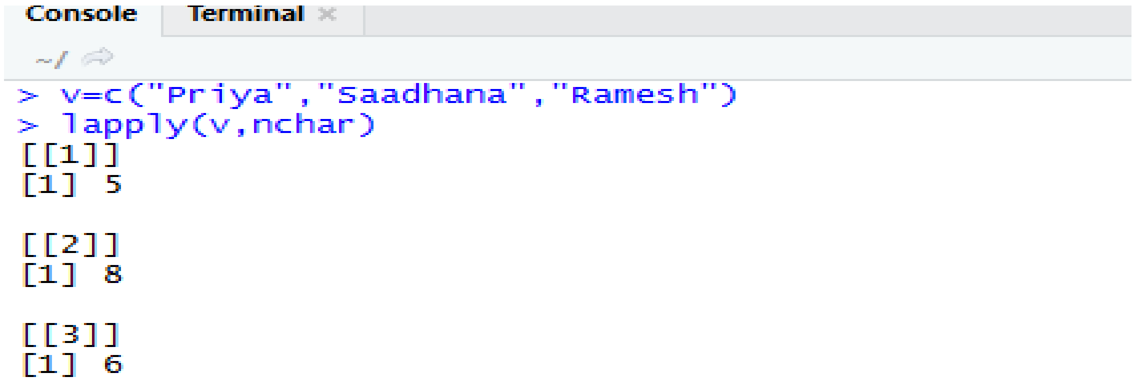
=Li

st FUN=A

Function

…. =other arguments

Example:



```
> v=c("Priya","Saadhana","Ramesh")
> lapply(v,nchar)
[[1]]
[1] 5

[[2]]
[1] 8

[[3]]
[1] 6
```

sapply:

Definition : same as lapply but try to simplify the result .

Syntax : lapply(X, FUN, ...)

X

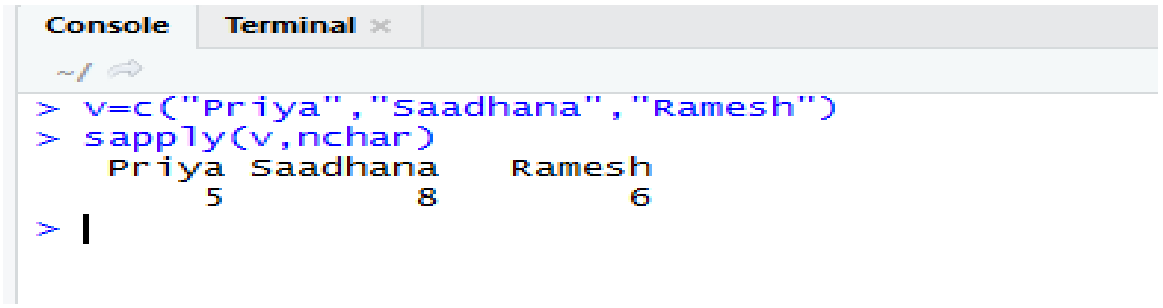
=Li

st FUN=A

Function

…. =other arguments

Example:



```
> v=c("Priya","Saadhana","Ramesh")
> sapply(v,nchar)
   Priya Saadhana    Ramesh
       5        8         6
> 
```

apply:

Definition: apply a function over the margins of an array

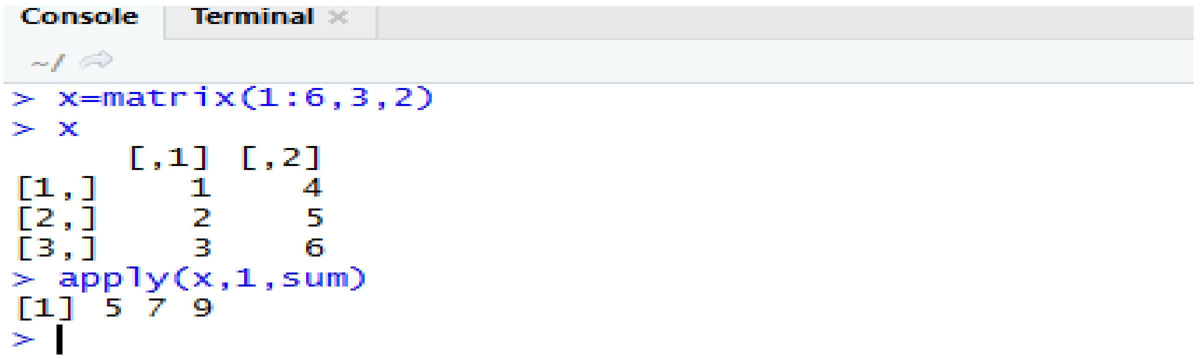
Syntax: apply(X, MARGIN, FUN, ...)

X =An Array

MARGIN =Integer vector indicating which margins should be "retained".

FUN	=Function to be applied
. . .	= other arguments to be passed to FUN

Example: Row wise sum in a given matrix using apply()



```
> x=matrix(1:6,3,2)
> x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> apply(x,1,sum)
[1] 5 7 9
> 
```

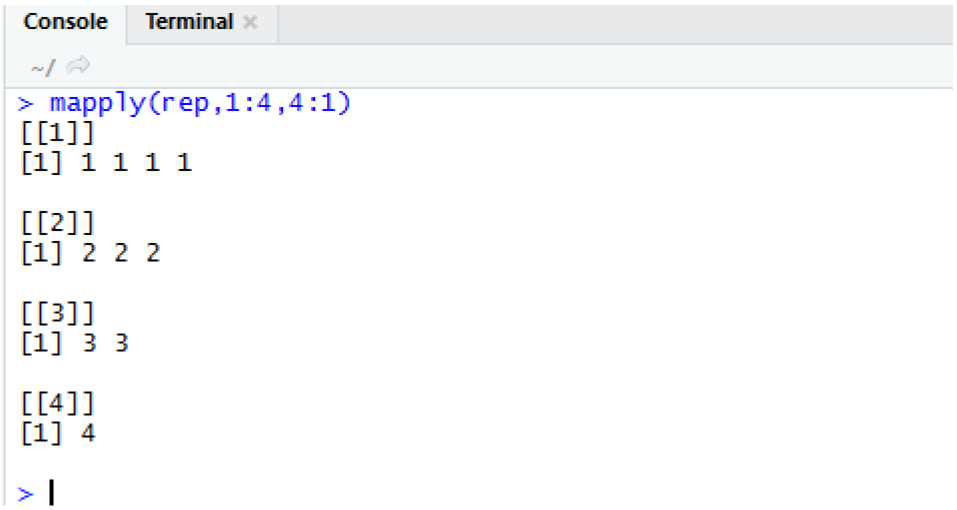
mapply:

Definition: Multivariate version of lapply

Syntax: mapply (FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)

FUN	: the function to be applied
. .	: arguments to apply over
MoreArgs	: a list of other arguments to
SIMPLIFY	: logical; whether the result should be simplified to a vector or matrix.

Example:



```
> mapply(rep,1:4,4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4

> 
```

Matrices in R

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. We use matrices containing numeric elements to be used in mathematical calculations.

Syntax:

The basic syntax for creating a matrix in R is −

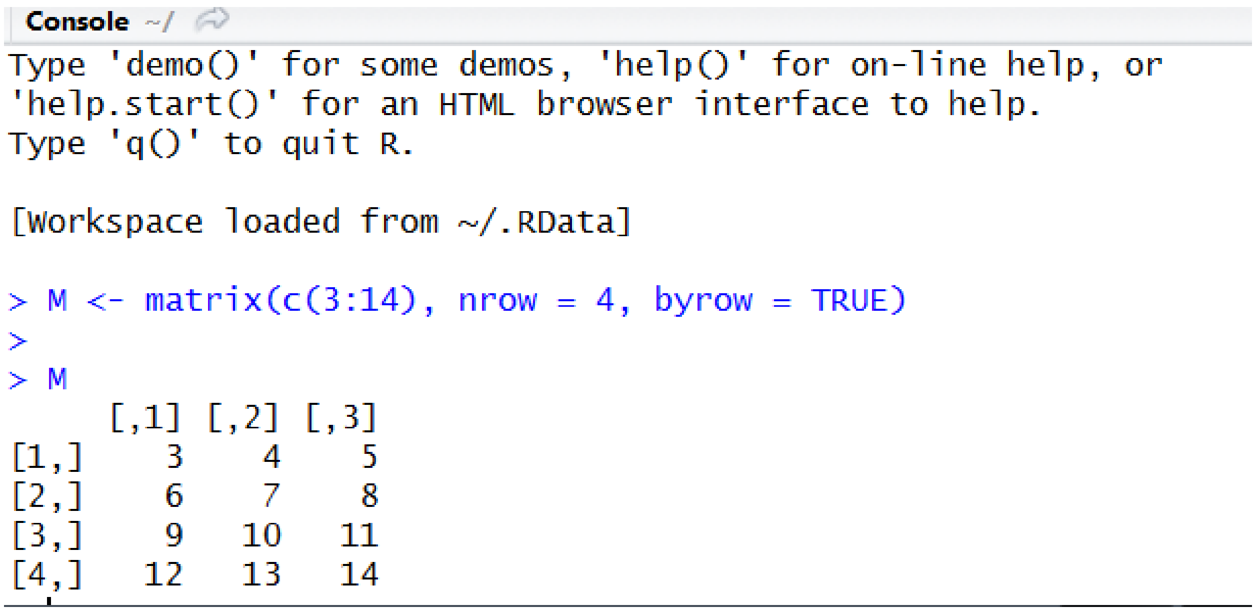
```
matrix(data, nrow, ncol, byrow, dimnames)
```

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

Matrix Creation:

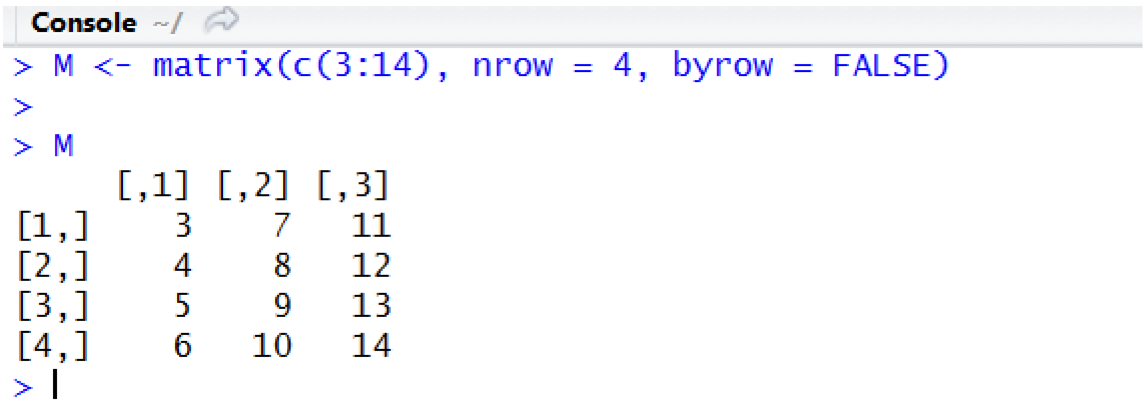
(i) Arrange elements sequentially by

row.Example:



```
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]

> M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
>
> M
     [,1] [,2] [,3]
[1,]    3    4    5
[2,]    6    7    8
[3,]    9   10   11
[4,]   12   13   14
```

(ii) Arrange elements sequentially by coloumn.

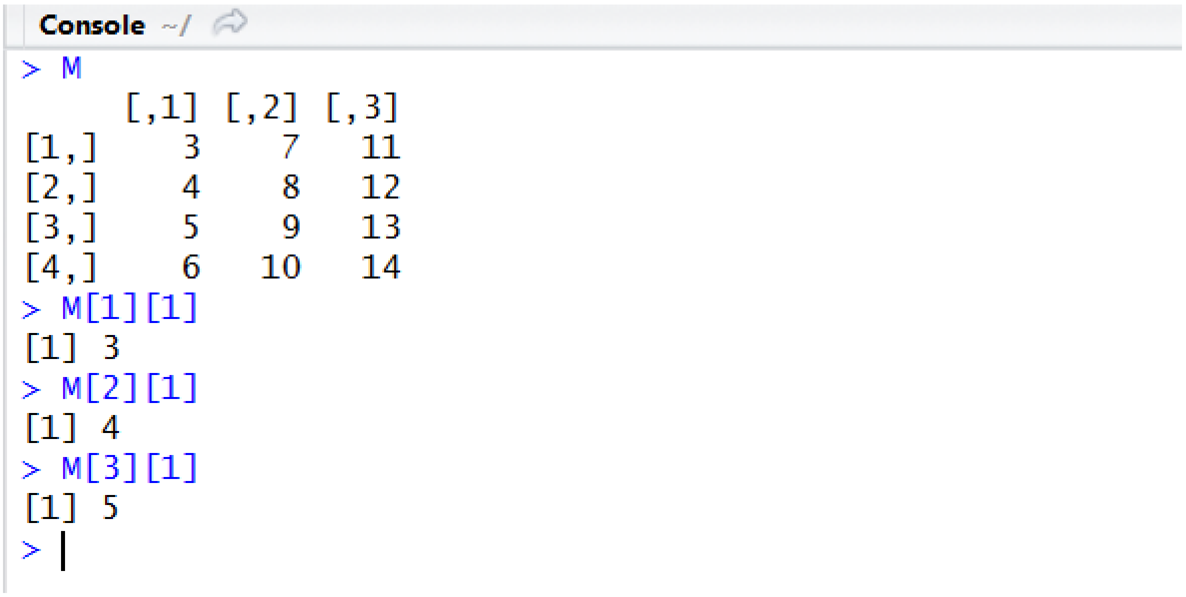


```
Console ~/
> M <- matrix(c(3:14), nrow = 4, byrow = FALSE)
>
> M
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
>
```

Accessing Elements of Matrix:

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

Example:

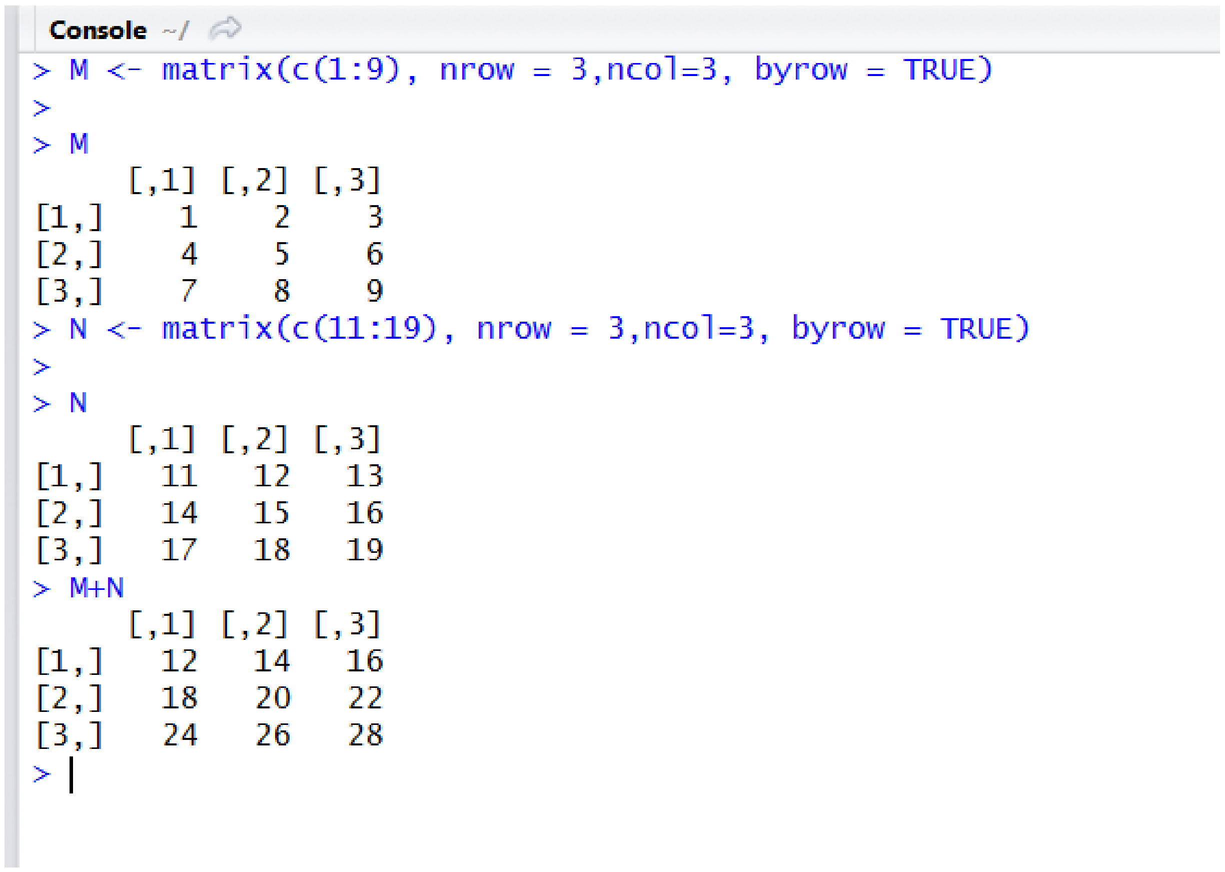


```
Console ~/
> M
     [,1] [,2] [,3]
[1,]    3    7   11
[2,]    4    8   12
[3,]    5    9   13
[4,]    6   10   14
> M[1][1]
[1] 3
> M[2][1]
[1] 4
> M[3][1]
[1] 5
>
```

Matrix Operations:

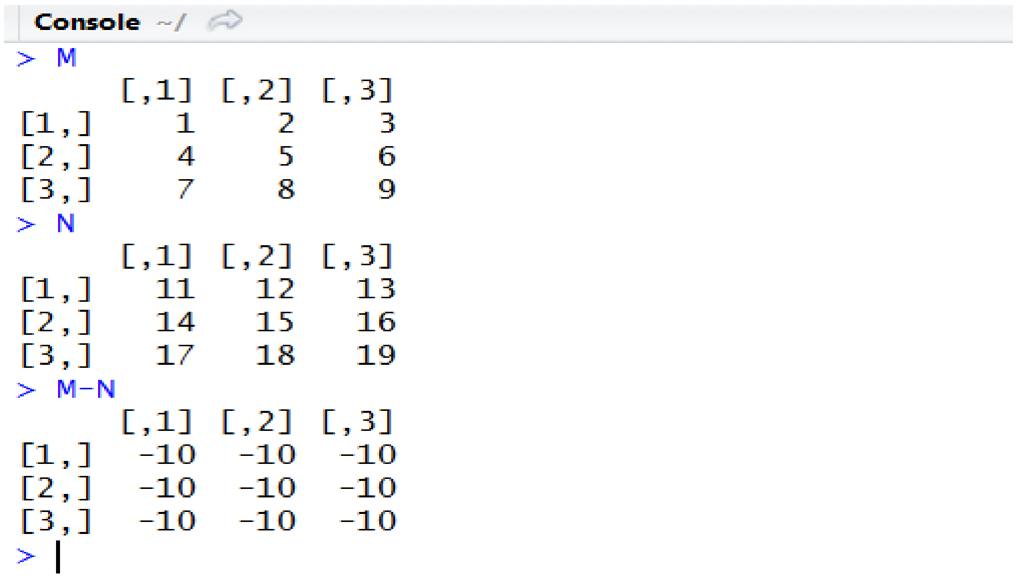
Various mathematical operations are performed on the matrices using the Roperators. The result of the operation is also a matrix.The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

Matrix Addition:



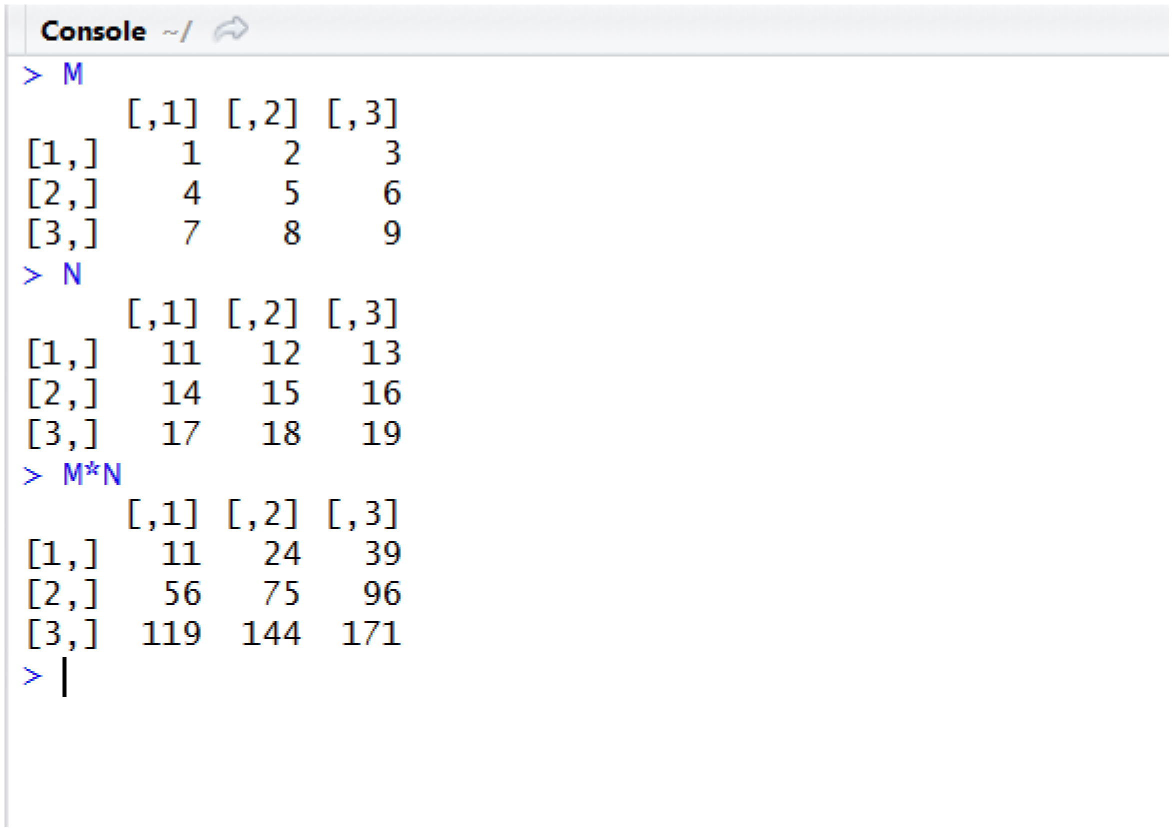
```
> M <- matrix(c(1:9), nrow = 3,ncol=3, byrow = TRUE)
>
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> N <- matrix(c(11:19), nrow = 3,ncol=3, byrow = TRUE)
>
> N
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   14   15   16
[3,]   17   18   19
> M+N
     [,1] [,2] [,3]
[1,]   12   14   16
[2,]   18   20   22
[3,]   24   26   28
>
```

Matrix Subtraction



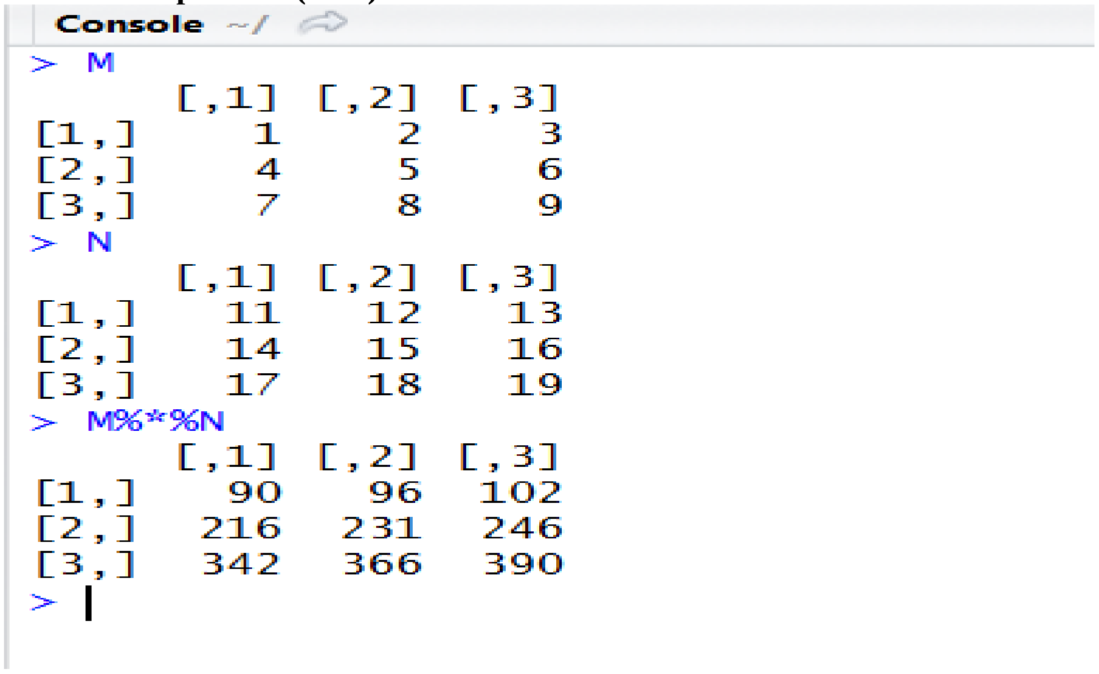
```
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> N
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   14   15   16
[3,]   17   18   19
> M-N
     [,1] [,2] [,3]
[1,]  -10  -10  -10
[2,]  -10  -10  -10
[3,]  -10  -10  -10
>
```

Matrix Multiplication(Elementwise)



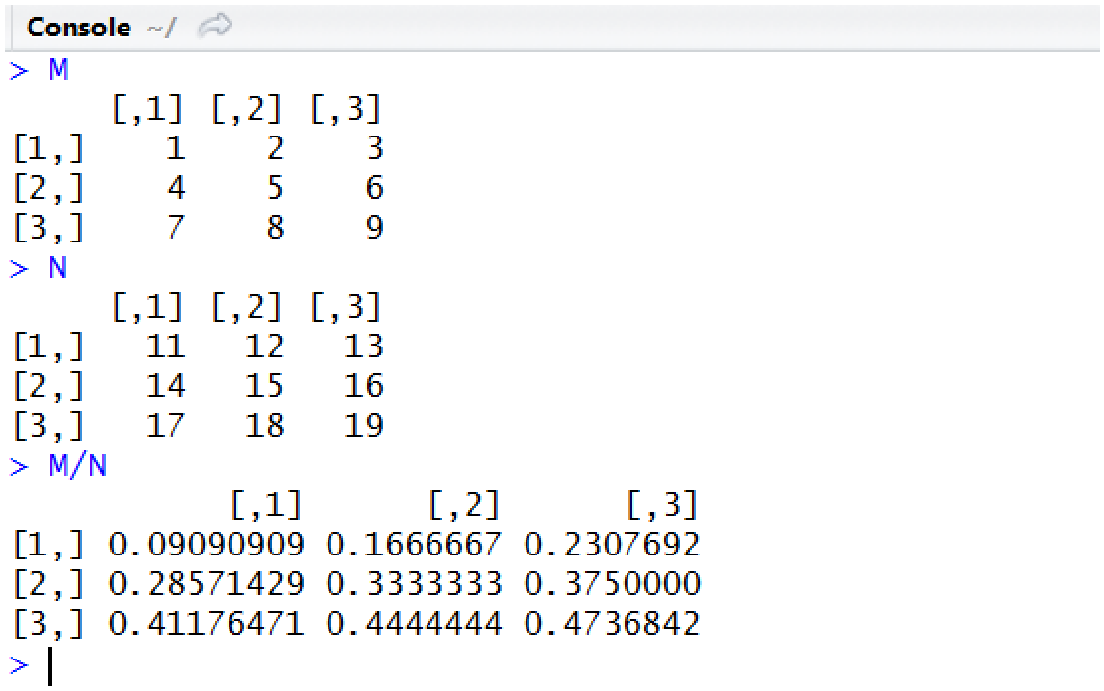
```
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> N
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   14   15   16
[3,]   17   18   19
> M*N
     [,1] [,2] [,3]
[1,]   11   24   39
[2,]   56   75   96
[3,]  119  144  171
> 
```

Matrix Multiplication(Real)



```
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> N
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   14   15   16
[3,]   17   18   19
> M%*%N
     [,1] [,2] [,3]
[1,]   90   96  102
[2,]  216  231  246
[3,]  342  366  390
> 
```

Matrix Division:



```
> M
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
> N
     [,1] [,2] [,3]
[1,]   11   12   13
[2,]   14   15   16
[3,]   17   18   19
> M/N
           [,1]      [,2]      [,3]
[1,] 0.09090909 0.1666667 0.2307692
[2,] 0.28571429 0.3333333 0.3750000
[3,] 0.41176471 0.4444444 0.4736842
> 
```

String Operations in R

String:

Any value written within a pair of single quote or double quotes in R istreated as a string.

Example:S=”Hello”

S1=’hai’

String Manipulation:

Concatenating Strings - paste() function:

Many strings in R are combined using the paste() function. It can takeany number of arguments to be combined together.

Syntax:

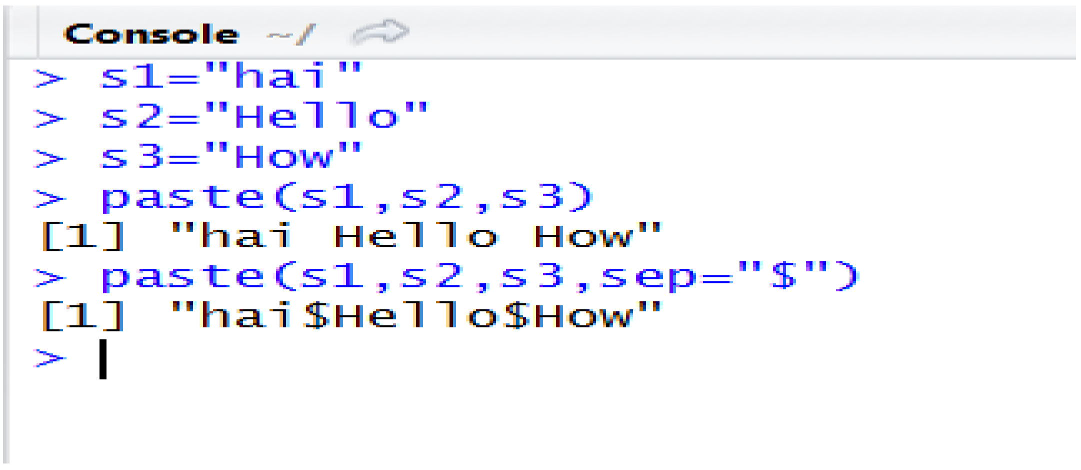
paste(..., sep = " ", collapse = NULL)

... represents any number of arguments to be combined.

sep represents any separator between the arguments. It is optional.

collapse is used to eliminate the space in between two strings. Butnot the space within two words of one string.

Example



```
> s1="hai"
> s2="Hello"
> s3="How"
> paste(s1,s2,s3)
[1] "hai Hello How"
> paste(s1,s2,s3,sep="$")
[1] "hai$Hello$How"
> 
```

Counting number of characters in a string - nchar() function

This function counts the number of characters including spaces in a string.

Syntax:

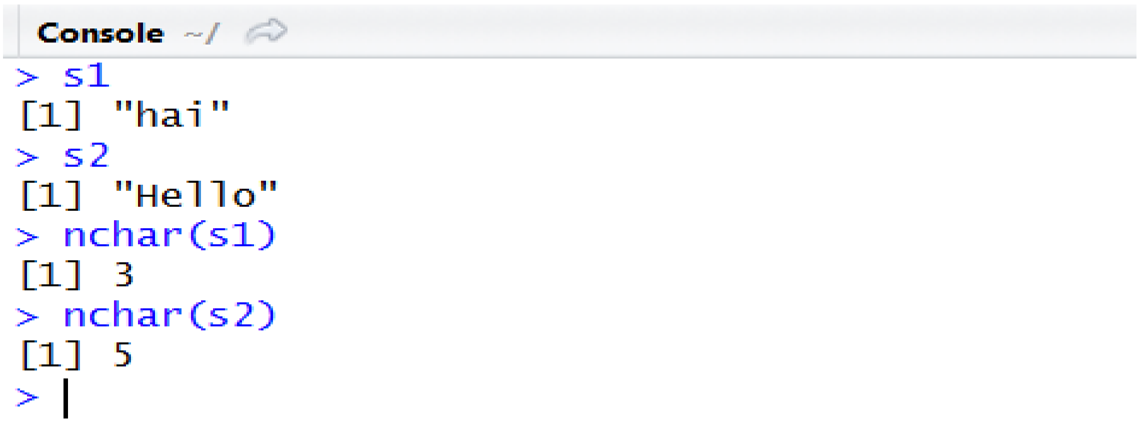
The basic syntax for nchar() function is −

```
nchar(x)
```

Following is the description of the parameters used −

- **x** is the vector input.

Example:



```
> s1
[1] "hai"
> s2
[1] "Hello"
> nchar(s1)
[1] 3
> nchar(s2)
[1] 5
> 
```

Changing the case - toupper() & tolower() functions

These functions change the case of characters of a string.

Syntax

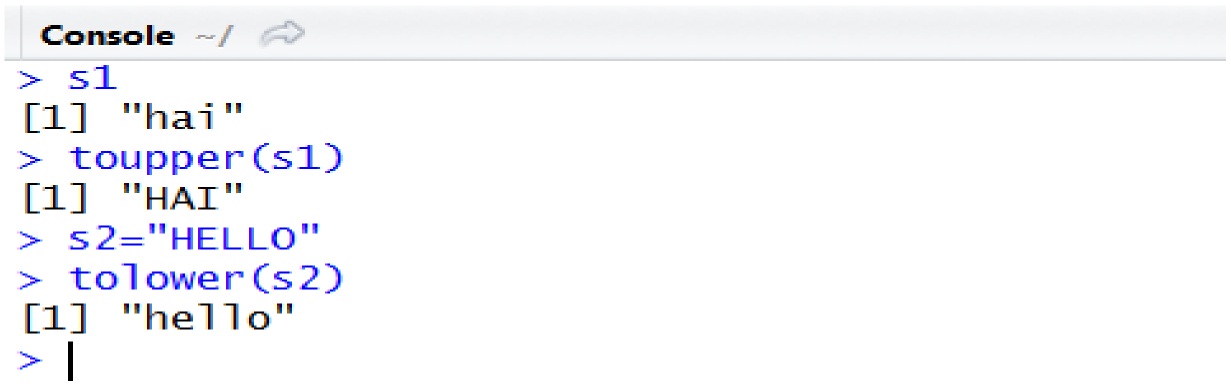
The basic syntax for toupper() & tolower() function is −

```
toupper(x)
tolower(x)
```

Following is the description of the parameters used −

- **x** is the vector input.

Example:



```
> s1
[1] "hai"
> toupper(s1)
[1] "HAI"
> s2="HELLO"
> tolower(s2)
[1] "hello"
>
```

Extracting parts of a string - substring() function

This function extracts parts of a String.

Syntax

The basic syntax for substring() function is −

```
substring(x,first,last)
```

Following is the description of the parameters used −

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

- **last** is the position of the last character to be extracted.

Replacement Functions:sub() and gsub()

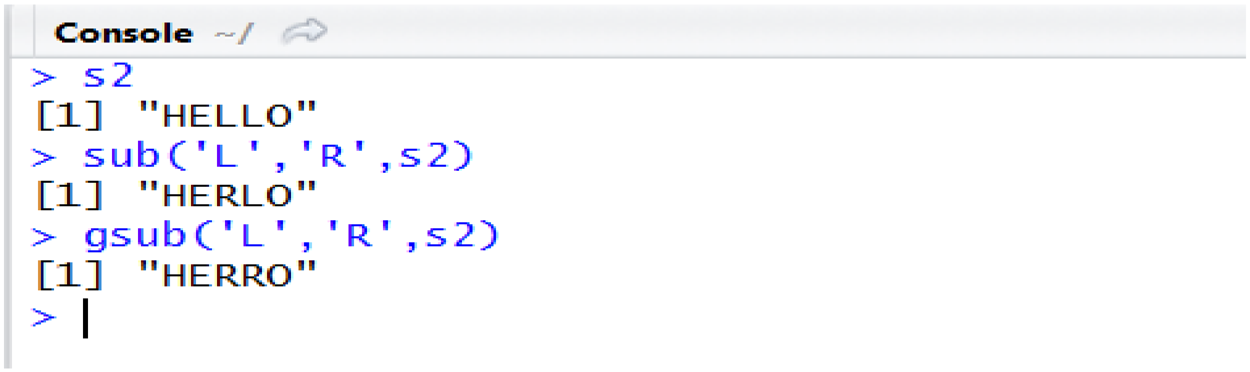
These are replacement functions, which replaces the occurrence of asubstring with other substring.

- sub() Function in R replaces the first instance of a substring
- gsub() function in R replaces all the instances of a substring

Syntax for sub() and gsub() function in R:

```
1. sub(old, new, string)

2. gsub(old, new, string)
```

Example:



```
Console ~/
> s2
[1] "HELLO"
> sub('L','R',s2)
[1] "HERLO"
> gsub('L','R',s2)
[1] "HERRO"
>
```

Pattern Matching Function:Grep() functionSyntax:

grep(value = FALSE) returns an integer vector of the indices of the elements of x that yielded a match .

grep(value = TRUE) returns a character vector containing the selectedelements of x.

Example:

SCHOOL OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY

Unit-II-R programming – SCS1621

R Data interfaces - CSV Files, XML files, Web Data- Data Preprocessing: Missing Values, Principle Component Analysis - Data Visualization – Charts & Graphs-Pie Chart, Bar Chart, Box plot, Histogram, Line graph, Scatter Plot.

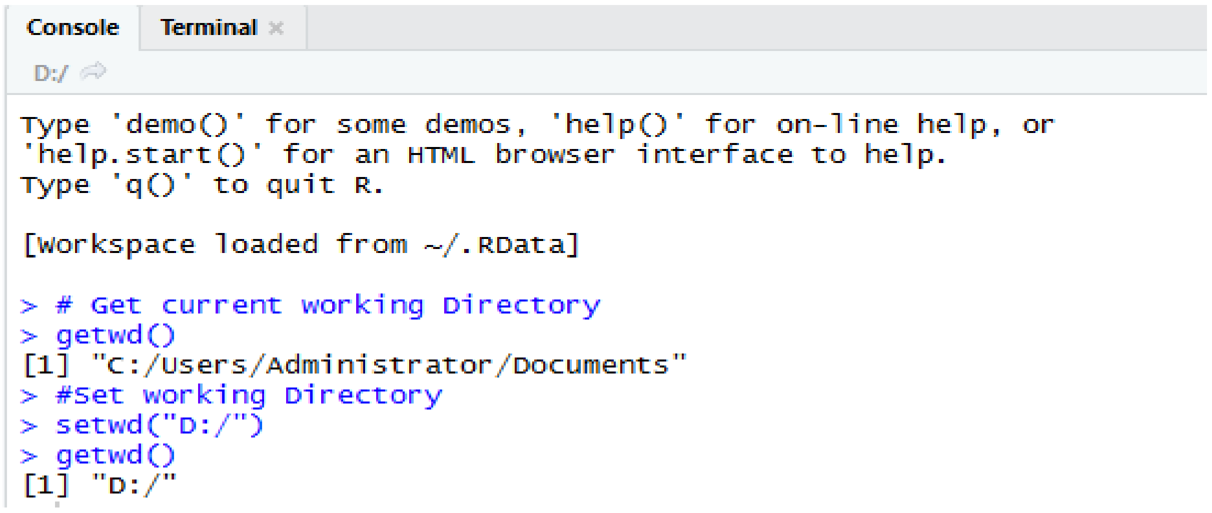
R DATA INTERFACES

we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

Getting and Setting the Working Directory

You can check which directory the R workspace is pointing to using the getwd() function. You can also set a new working directory using setwd()function.

Example:

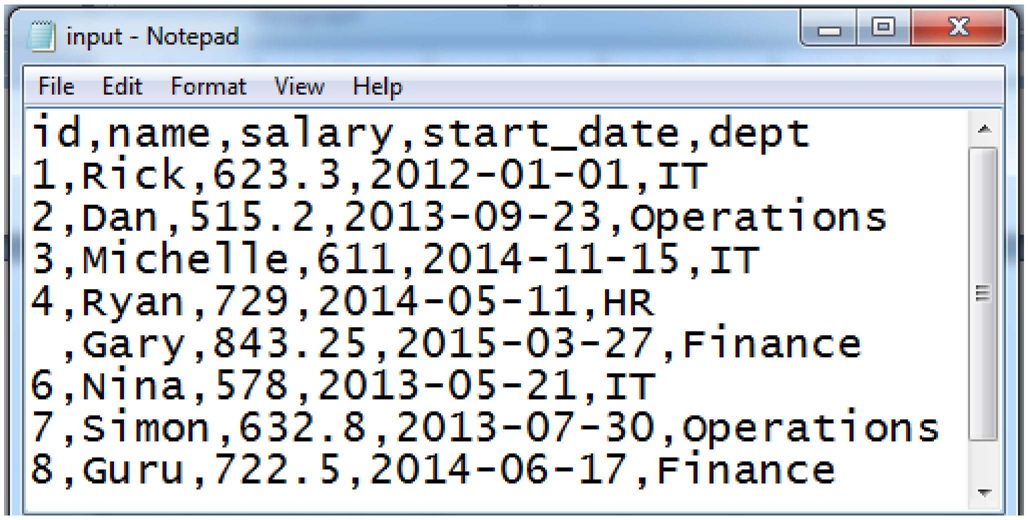


```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> # Get current working Directory
> getwd()
[1] "C:/Users/Administrator/Documents"
> #Set working Directory
> setwd("D:/")
> getwd()
[1] "D:/"
```

CSV Files

Input as CSV

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input.csv**.Create this file using windows notepad by copying and pasting this data. Save the fileas **input.csv** using the save As All files(*.*) option in notepad.

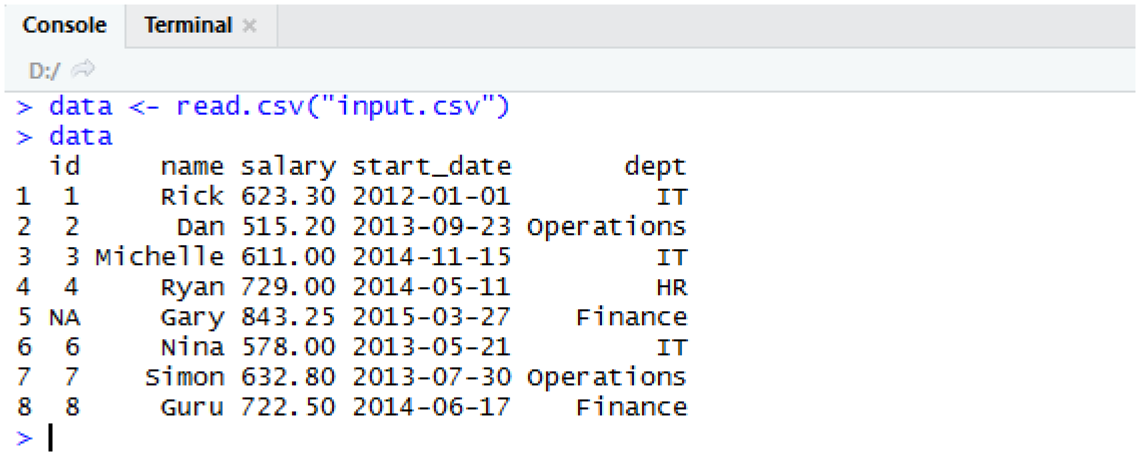


```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
 ,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

Reading a CSV File

read.csv() function is used to read a CSV file available in your currentworking directory .

Example:

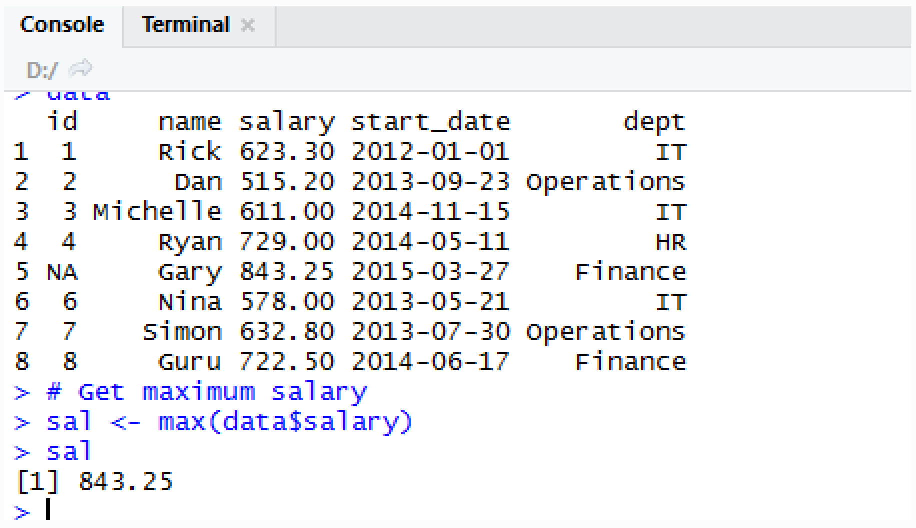


```
> data <- read.csv("input.csv")
> data
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance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

Analysing CSV file:

By default the read.csv() function gives the output as a data frame. Once weread data in a data frame, we can apply all the functions applicable to data frames.

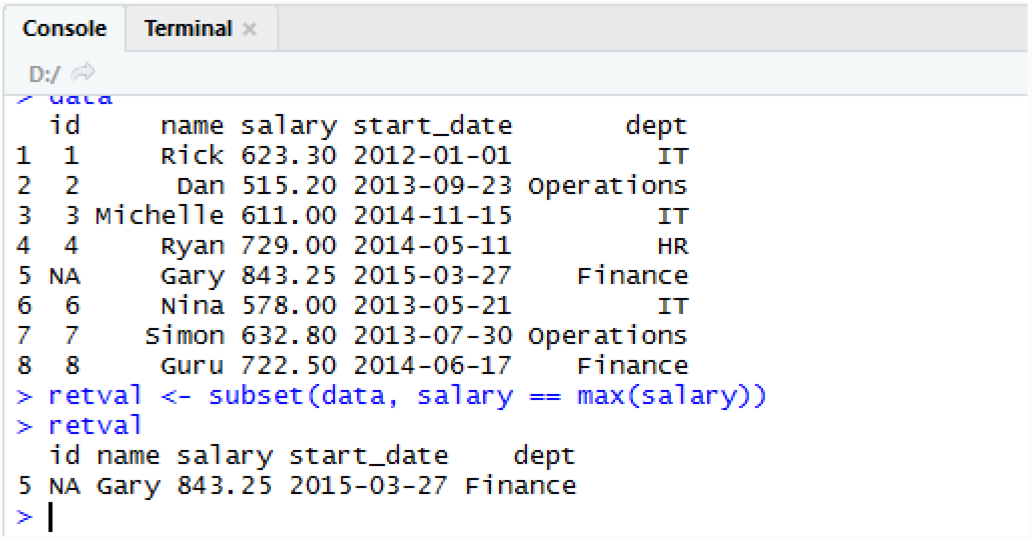
1.Get maximum salary.


Console Terminal

D:/

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

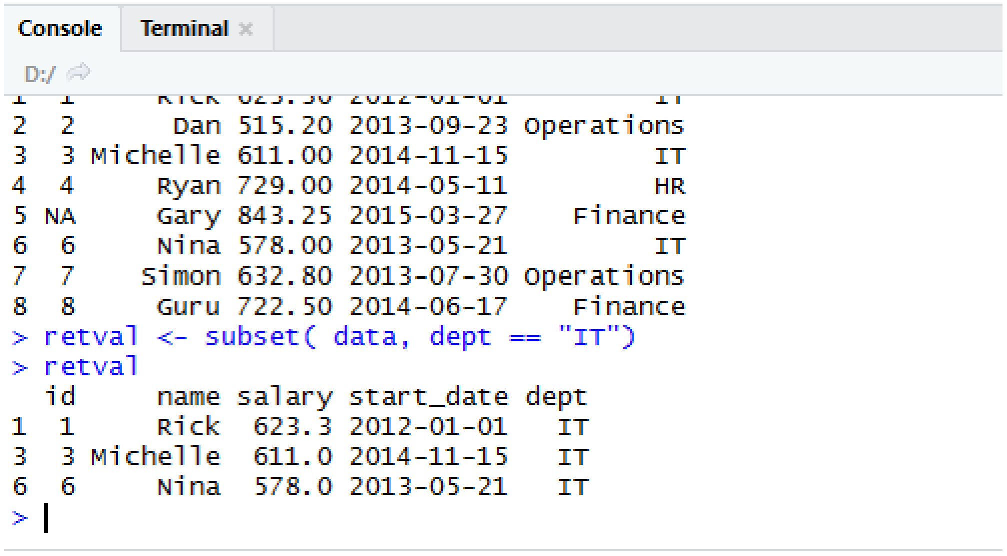
2. Get the details of the person with max salary


Console Terminal

D:/

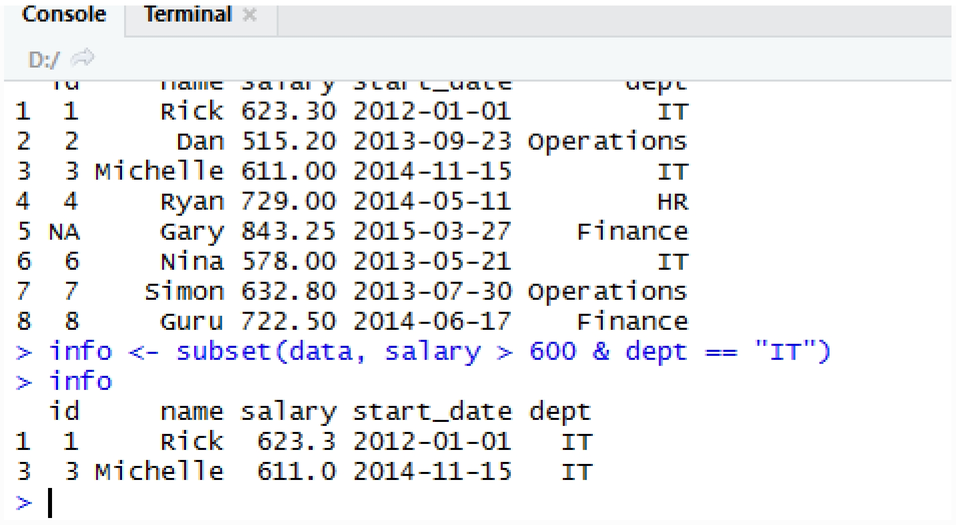
```
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21       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

3. Get all the people working in IT department



```
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> retval <- subset( data, dept == "IT")
> retval
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
6  6     Nina  578.0 2013-05-21   IT
> 
```

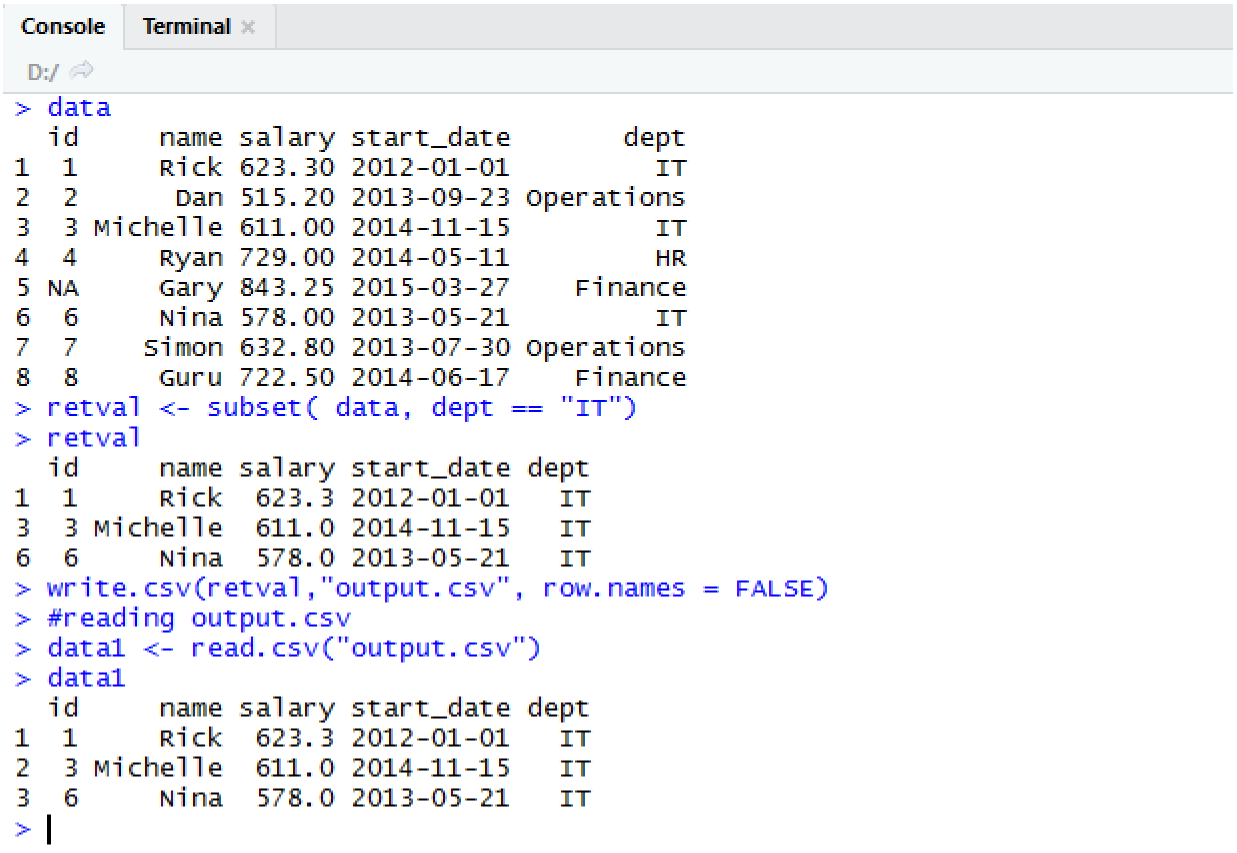
4. Get the persons in IT department whose salary is greater than 600



```
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> info <- subset(data, salary > 600 & dept == "IT")
> info
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
> 
```

Writing into a CSV File

R can create csv file form existing data frame. The **write.csv()** function isused to create the csv file. This file gets created in the working directory.


Console Terminal

D:/

```
> data
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> retval <- subset( data, dept == "IT")
> retval
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
6  6     Nina  578.0 2013-05-21   IT
> write.csv(retval,"output.csv", row.names = FALSE)
> #reading output.csv
> data1 <- read.csv("output.csv")
> data1
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
2  3 Michelle  611.0 2014-11-15   IT
3  6     Nina  578.0 2013-05-21   IT
> |
```

XML files

XML is a file format which shares both the file format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It stands for Extensible Markup Language (XML). Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in xml the markup tags describe the meaning of the data contained into he file.

Read a xml file in R using the "XML" package. This package can be installed using following command.

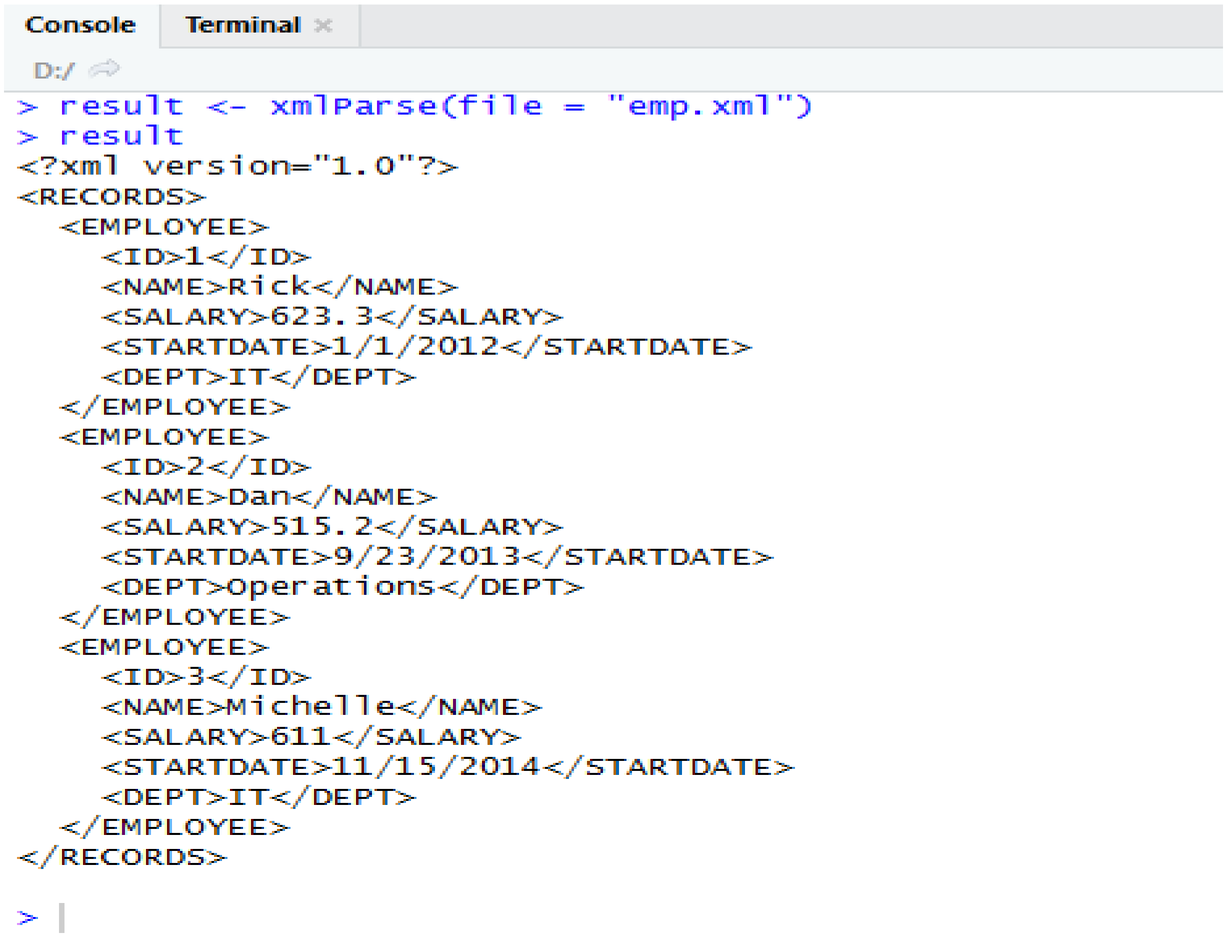
Install.packages(“XML”)

Input Data

Create a XMl file by copying the below data into a text editor like notepad. Save the file with a **.xml** extension and choosing the file type as **all files(*.*)**.

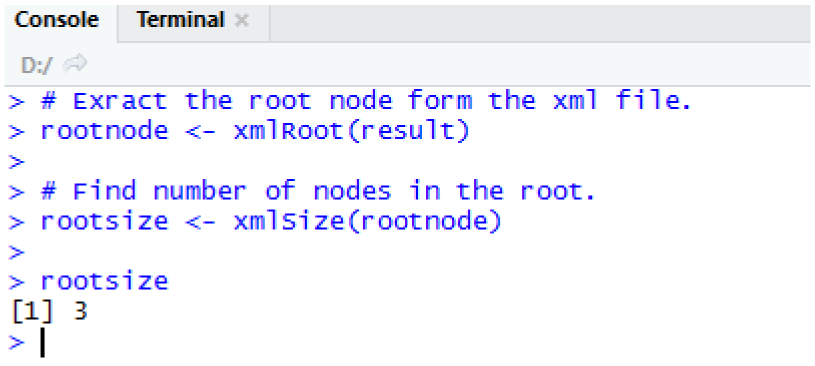
Reading XML File

The xml file is read by R using the function **xmlParse()**. It is stored as alist in R.



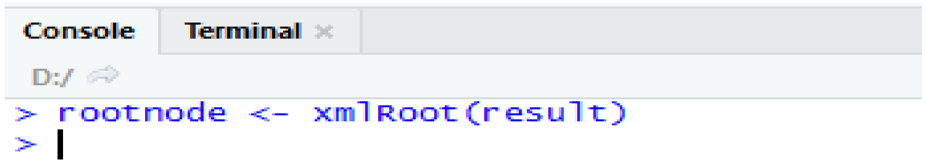
```
> result <- xmlParse(file = "emp.xml")
> result
<?xml version="1.0"?>
<RECORDS>
  <EMPLOYEE>
    <ID>1</ID>
    <NAME>Rick</NAME>
    <SALARY>623.3</SALARY>
    <STARTDATE>1/1/2012</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>2</ID>
    <NAME>Dan</NAME>
    <SALARY>515.2</SALARY>
    <STARTDATE>9/23/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>3</ID>
    <NAME>Michelle</NAME>
    <SALARY>611</SALARY>
    <STARTDATE>11/15/2014</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
</RECORDS>

> 
```

Get Number of Nodes Present in XML File



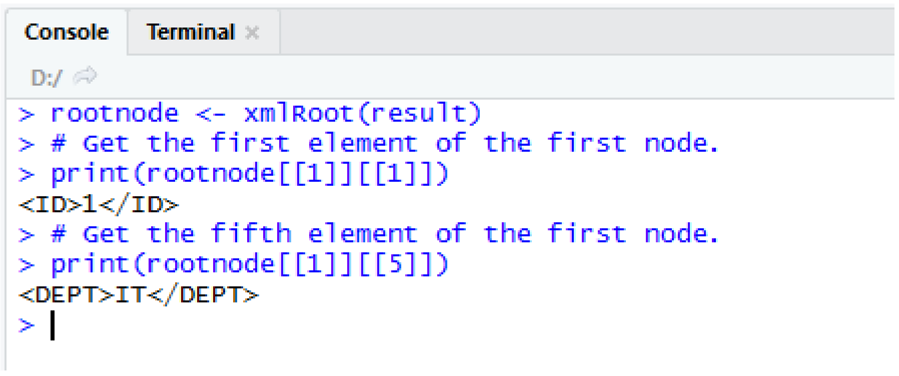
```
> # Exract the root node form the xml file.
> rootnode <- xmlRoot(result)
>
> # Find number of nodes in the root.
> rootsize <- xmlSize(rootnode)
>
> rootsize
[1] 3
> 
```

Details of the First Node



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

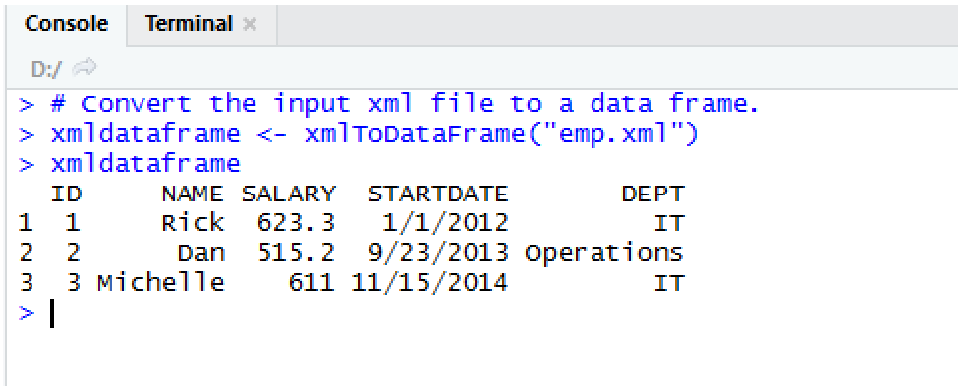
Get Different Elements of a Node



```
> rootnode <- xmlRoot(result)
> # Get the first element of the first node.
> print(rootnode[[1]][[1]])
<ID>1</ID>
> # Get the fifth element of the first node.
> print(rootnode[[1]][[5]])
<DEPT>IT</DEPT>
> 
```

XML to Data Frame

To handle the data effectively in large files we read the data in the xml fileas a data frame. Then process the data frame for data analysis.



```
> # Convert the input xml file to a data frame.
> xmldataframe <- xmlToDataFrame("emp.xml")
> xmldataframe
  ID     NAME SALARY  STARTDATE       DEPT
1  1     Rick  623.3   1/1/2012         IT
2  2      Dan  515.2  9/23/2013 Operations
3  3 Michelle    611 11/15/2014         IT
> 
```

WEB DATA

Many websites provide data for consumption by its users. For example the World Health Organization(WHO) provides reports on health and medical information in the form of CSV, txt and XML files. Using R programs, we canprogrammatically extract specific data from such websites. Some packages in R which are used to scrap data form the web are − "RCurl",XML", and "stringr". They are used to connect to the URL’s, identify required links for the files and download them to the local environment.

Install R Packages

The following packages are required for processing the URL’s and links to the files. If they are not available in your R Environment, you can install them using following commands.

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

Input Data

We will visit the URL weather data and download the CSV files using R for theyear 2015.

Example

We will use the function getHTMLLinks() to gather the URLs of the files. Then we will use the function download.file() to save the files to the local system. As we will be applying the same code again and again for multiple files, we will create a function to be called multiple times. The filenames arepassed as parameters in form of a R list object to this function.

```
# Read the URL.

url <- "http://www.geos.ed.ac.uk/~weather/jcmb_ws/"
```

```
# Gather the html links present in the webpage.links <- getHTMLLinks(url)

# Identify only the links which point to the JCMB 2015 files.filenames <- links[str_detect(links,

"JCMB_2015")]

# Store the file names as a list. filenames_list <- as.list(filenames)

# Create a function to download the files by passing the URL and filenamelist.

downloadcsv <- function (mainurl,filename) {filedetails <- str_c(mainurl,filename)

   download.file(filedetails,filename)

}

# Now apply the lapply function and save the files into the current Rworking directory.

lapply(filenames,downloadcsv,mainurl= "http://www.geos.ed.ac.uk/~weather/jcmb_ws/")
```

Verify the File Download

After running the above code, you can locate the following files in thecurrent R working directory.

"JCMB_2015.csv" "JCMB_2015_Apr.csv" "JCMB_2015_Feb.csv"
"JCMB_2015_Jan.csv" "JCMB_2015_Mar.csv"

R Charts and Graphs

R Programming language has numerous libraries to create charts andgraphs.

Pie Chart

A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the **pie()** function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

syntax

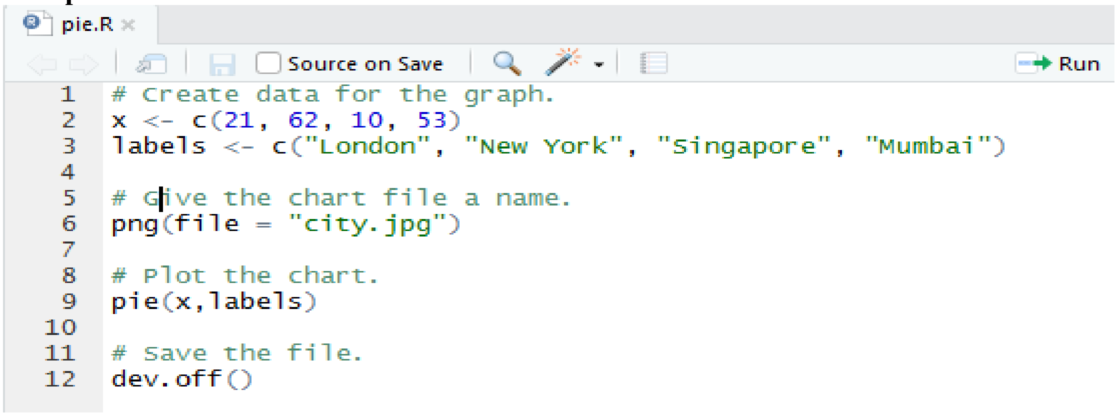
The basic syntax for creating a pie-chart using the R is −

```
pie(x, labels, radius, main, col, clockwise)
```

Following is the description of the parameters used −

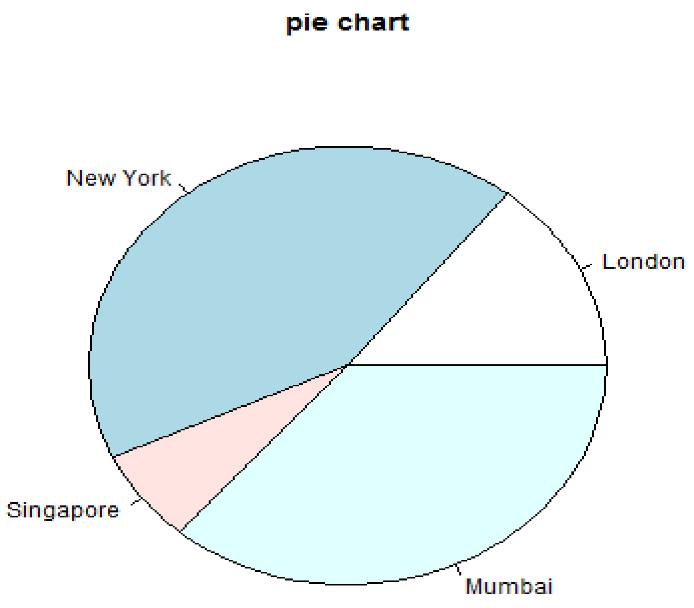
- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.
- **clockwise** is a logical value indicating if the slices are drawn clockwise oranti clockwise.

Example:



```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

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

# Plot the chart.
pie(x,labels)

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

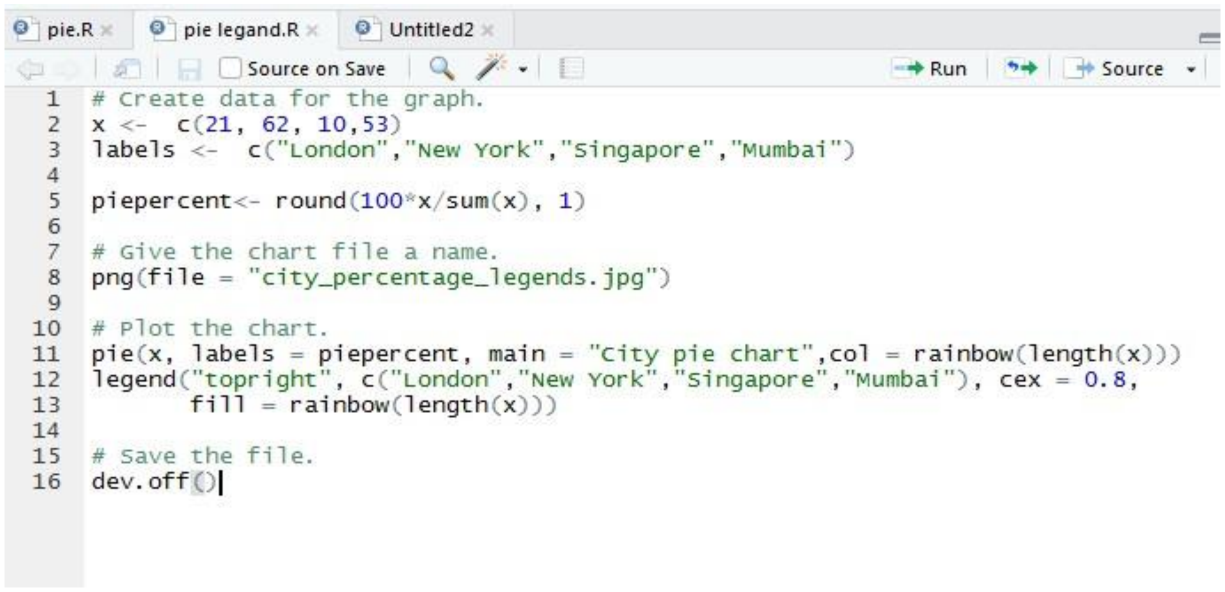
Output:



Slice Percentages and Chart Legend

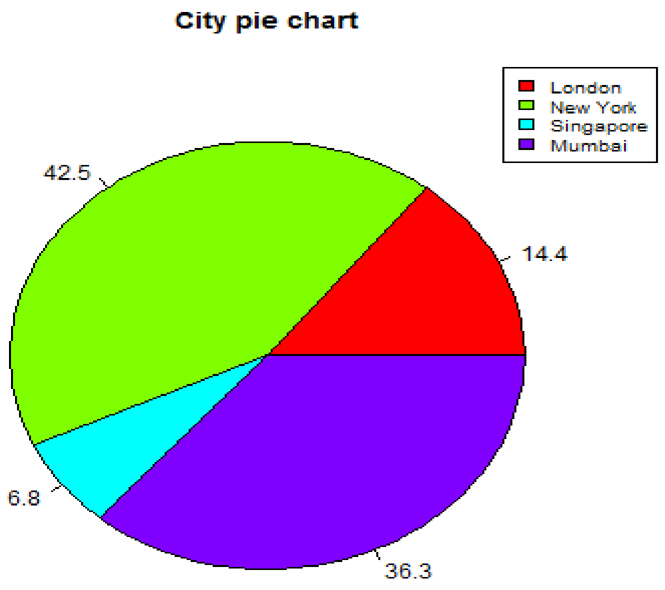
We can add slice percentage and a chart legend by creating additional chartvariables.

Example:



```r
# Create data for the graph.
x <-  c(21, 62, 10,53)
labels <-  c("London","New York","Singapore","Mumbai")

piepercent<- round(100*x/sum(x), 1)

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

# Plot the chart.
pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
       fill = rainbow(length(x)))

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

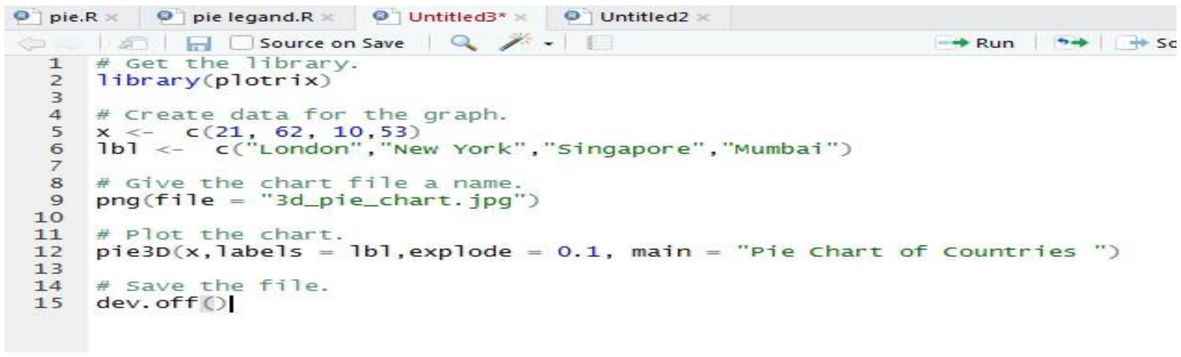
Output:



3D Pie Chart

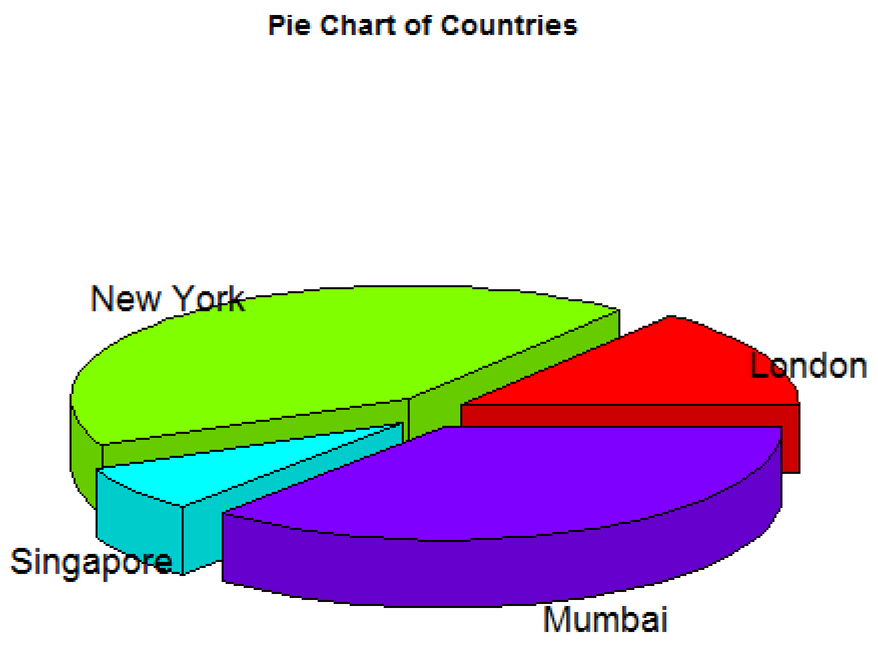
A pie chart with 3 dimensions can be drawn using additional packages. Thepackage **plotrix** has a function called **pie3D()** that is used for this.

Example:



```
# Get the library.
library(plotrix)

# Create data for the graph.
x <-   c(21, 62, 10,53)
lbl <-   c("London","New York","Singapore","Mumbai")

# Give the chart file a name.
png(file = "3d_pie_chart.jpg")

# Plot the chart.
pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")

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

Output:



BAR CHART

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function **barplot()** to create bar charts. R can draw both vertical and horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is −

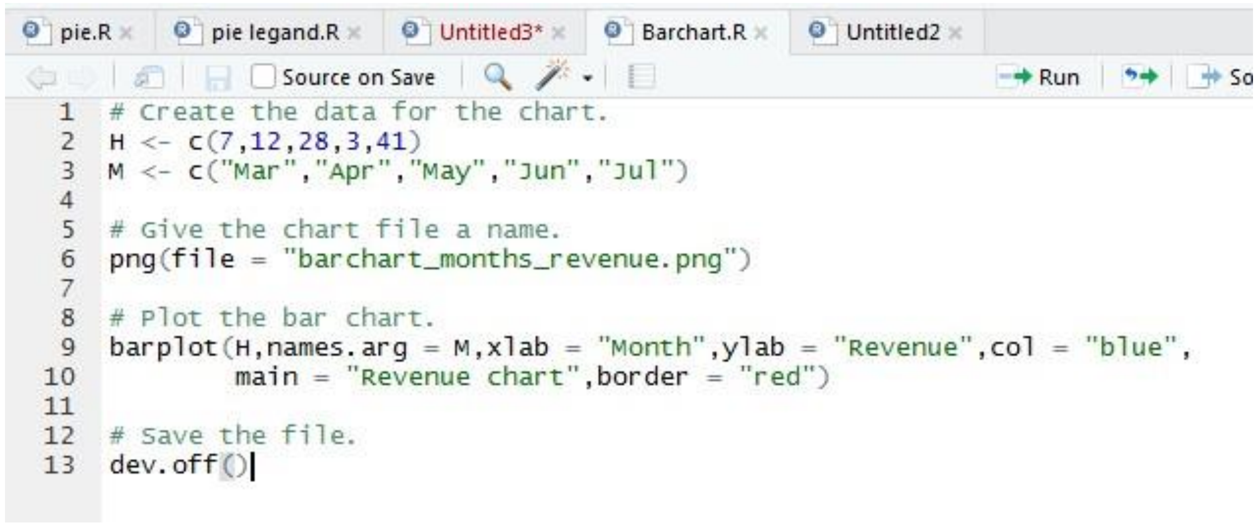
```
barplot(H, xlab, ylab, main, names.arg, col)
```

Following is the description of the parameters used −

- **H** is a vector or matrix containing numeric values used in bar chart.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the title of the bar chart.
- **names.arg** is a vector of names appearing under each bar.
- **col** is used to give colors to the bars in the graph.

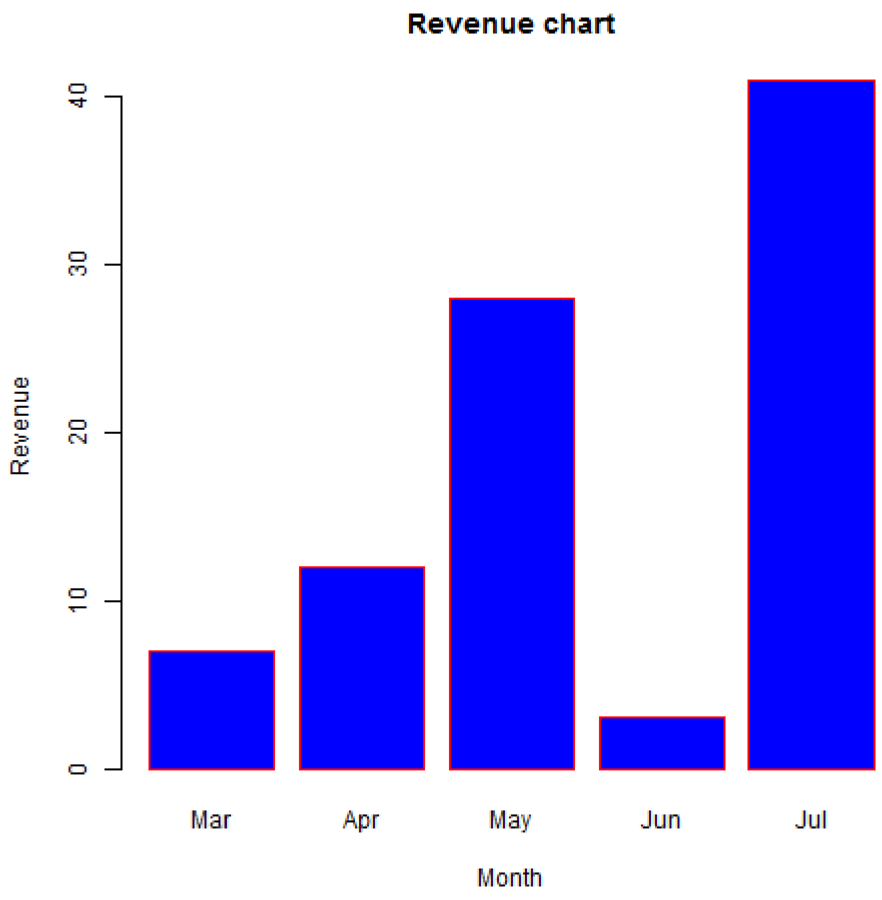
Example:

The following script will create and save the bar chart in the current R workingdirectory.



```
# Create the data for the chart.
H <- c(7,12,28,3,41)
M <- c("Mar","Apr","May","Jun","Jul")

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

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

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

Output:

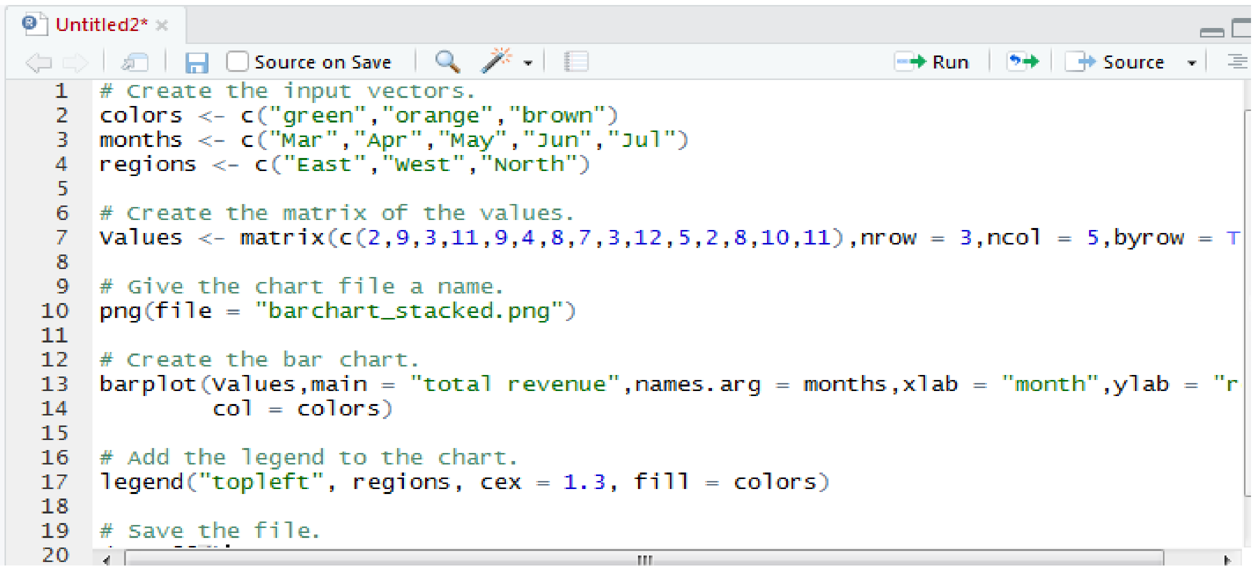


Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by usinga matrix as input values.

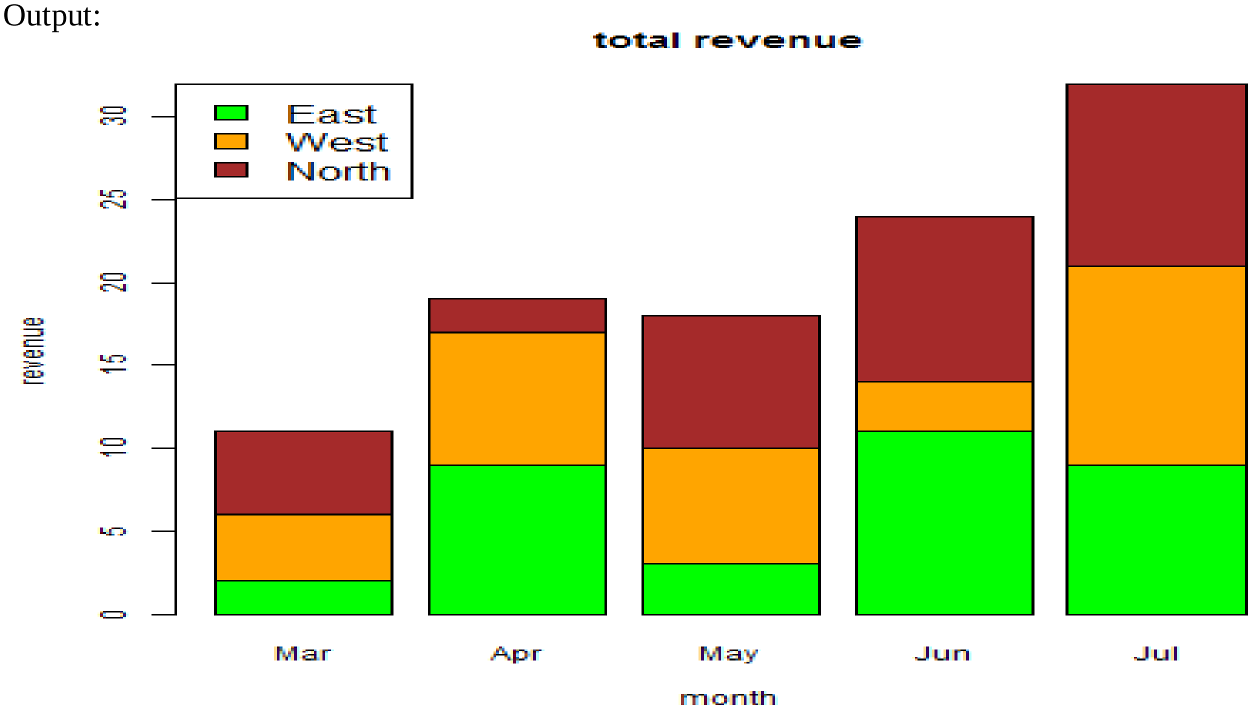
More than two variables are represented as a matrix which is used to createthe group bar chart and stacked bar chart.

Example:



```
# Create the input vectors.
colors <- c("green","orange","brown")
months <- c("Mar","Apr","May","Jun","Jul")
regions <- c("East","West","North")

# Create the matrix of the values.
Values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11),nrow = 3,ncol = 5,byrow = T

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

# Create the bar chart.
barplot(Values,main = "total revenue",names.arg = months,xlab = "month",ylab = "r
        col = colors)

# Add the legend to the chart.
legend("topleft", regions, cex = 1.3, fill = colors)

# Save the file.
```

Output:



HISTOGRAM

A histogram represents the frequencies of values of a variable bucketed intoranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist()** function. This function takes a vector asan input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is –

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

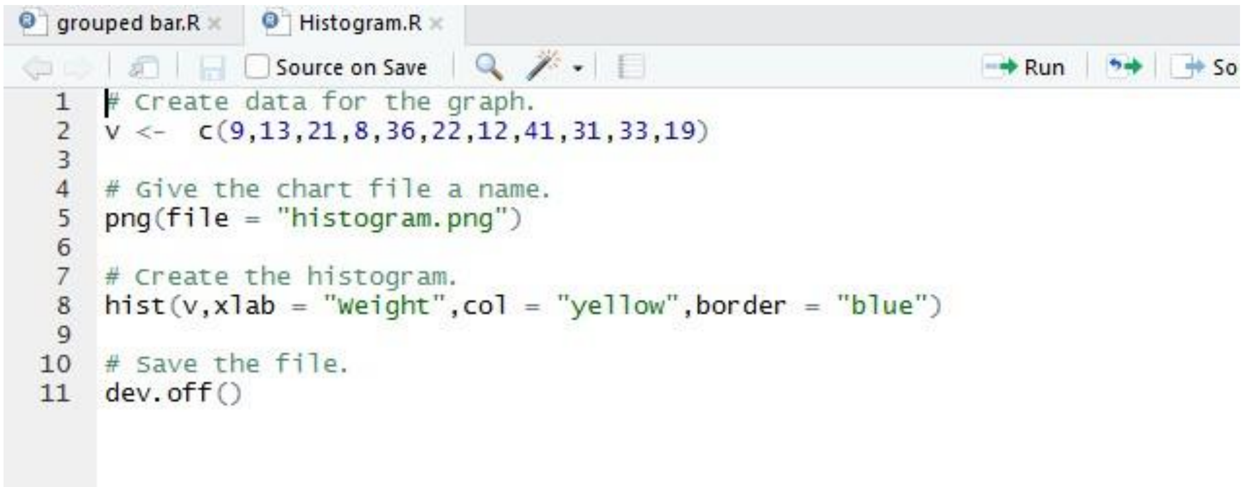
Following is the description of the parameters used −

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

Example

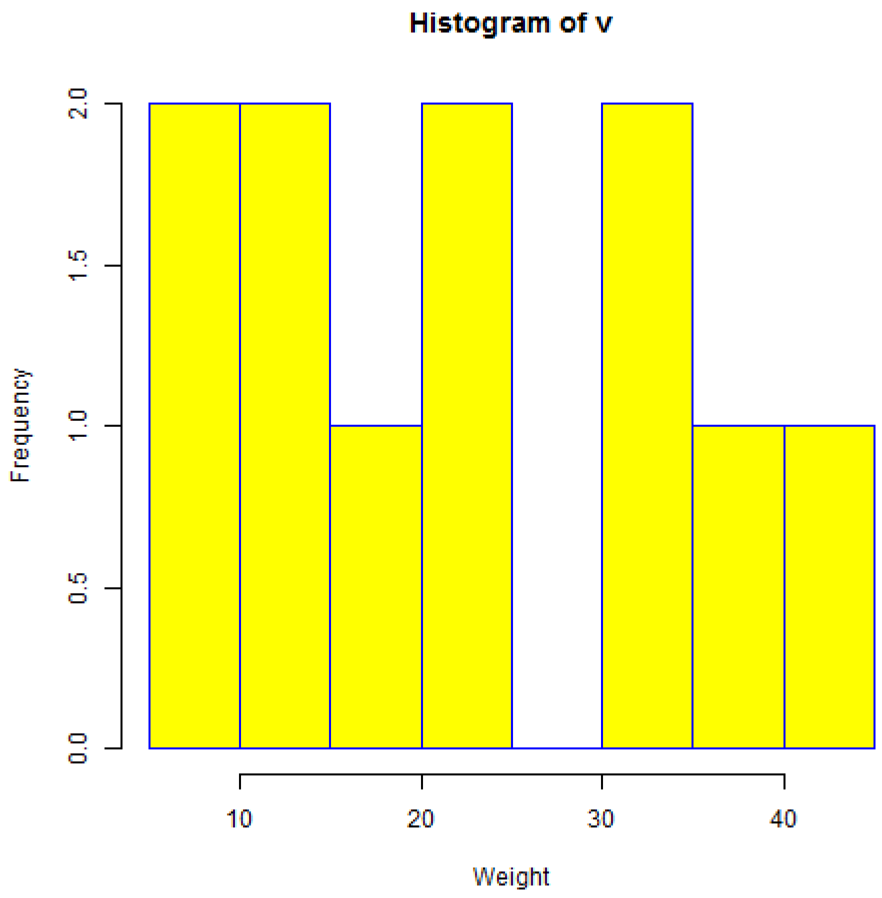
A simple histogram is created using input vector, label, col and borderparameters.

The script given below will create and save the histogram in the current Rworking directory.



```
# Create data for the graph.
v <-  c(9,13,21,8,36,22,12,41,31,33,19)

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

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

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

Output:

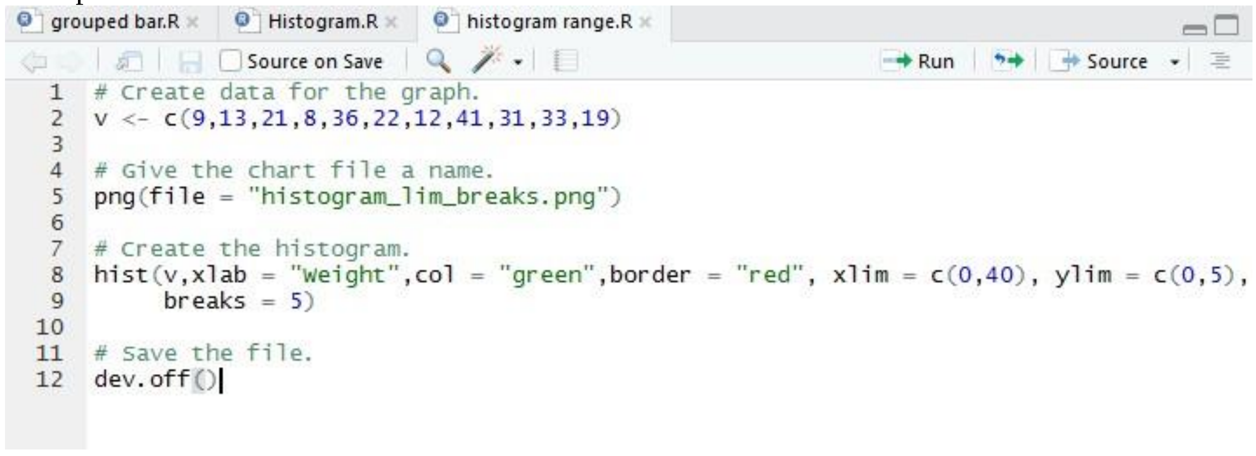


Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use thexlim and ylim parameters.

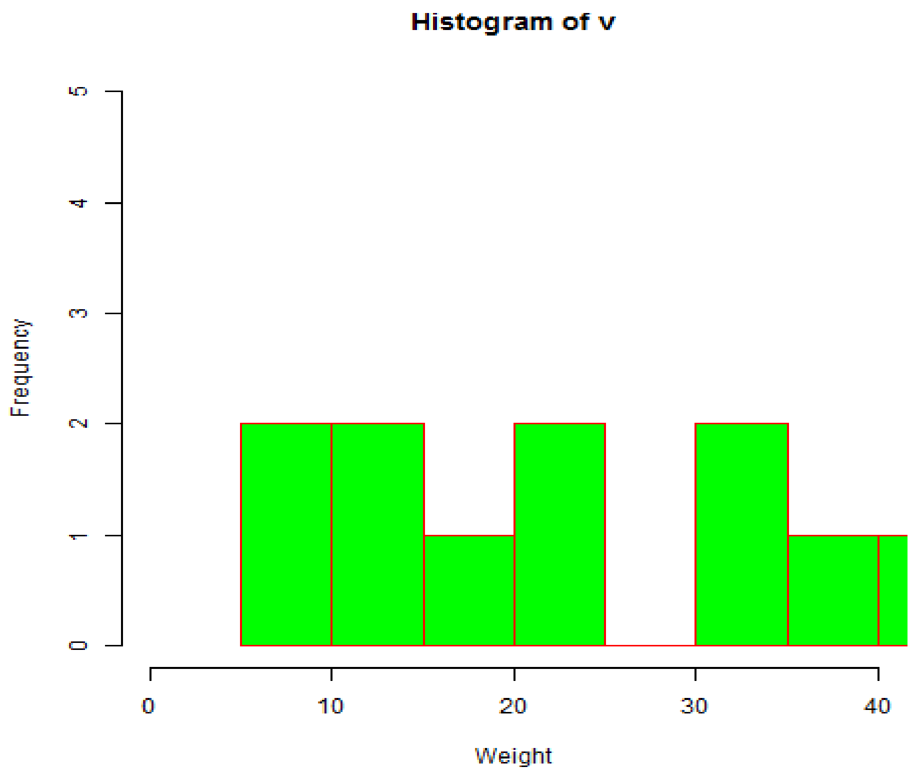
The width of each of the bar can be decided by using breaks.

Example:



```
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

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

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
     breaks = 5)

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

Output:



LINE GRAPH

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is −

```
plot(v,type,col,xlab,ylab)
```

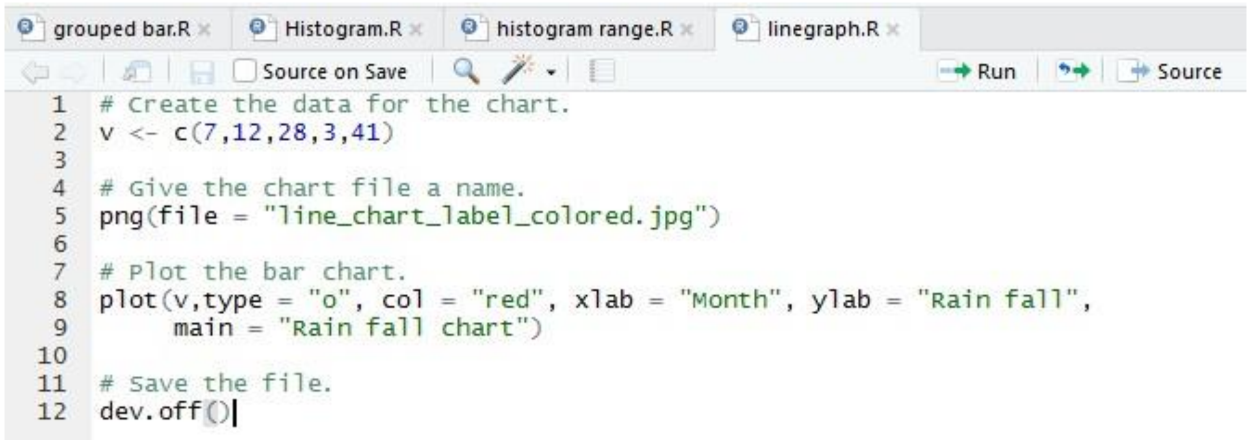
Following is the description of the parameters used −

- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only thelines and "o" to draw both points and lines.

- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

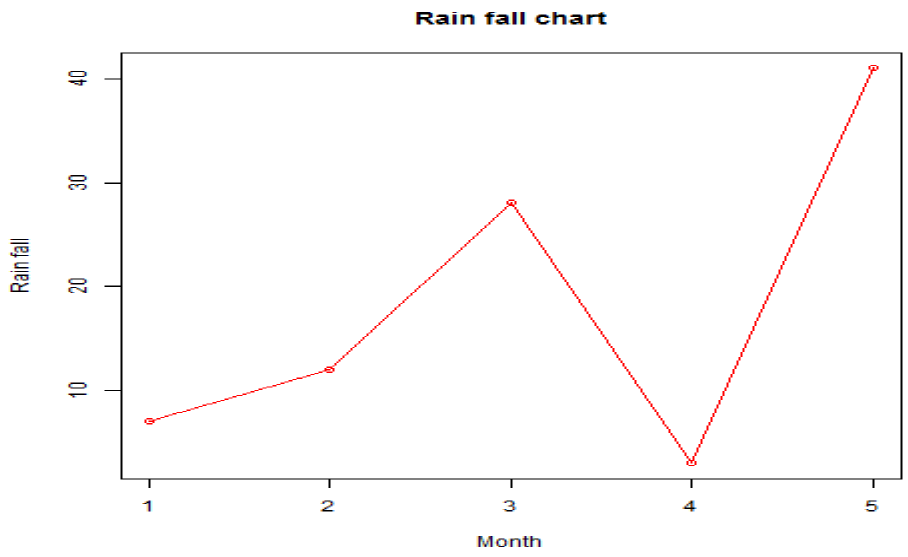
Example

A simple line chart is created using the input vector and the type parameteras "O". The below script will create and save a line chart in the current R working directory. The features of the line chart has been expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.



```r
# Create the data for the chart.
v <- c(7,12,28,3,41)

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

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

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

Output:

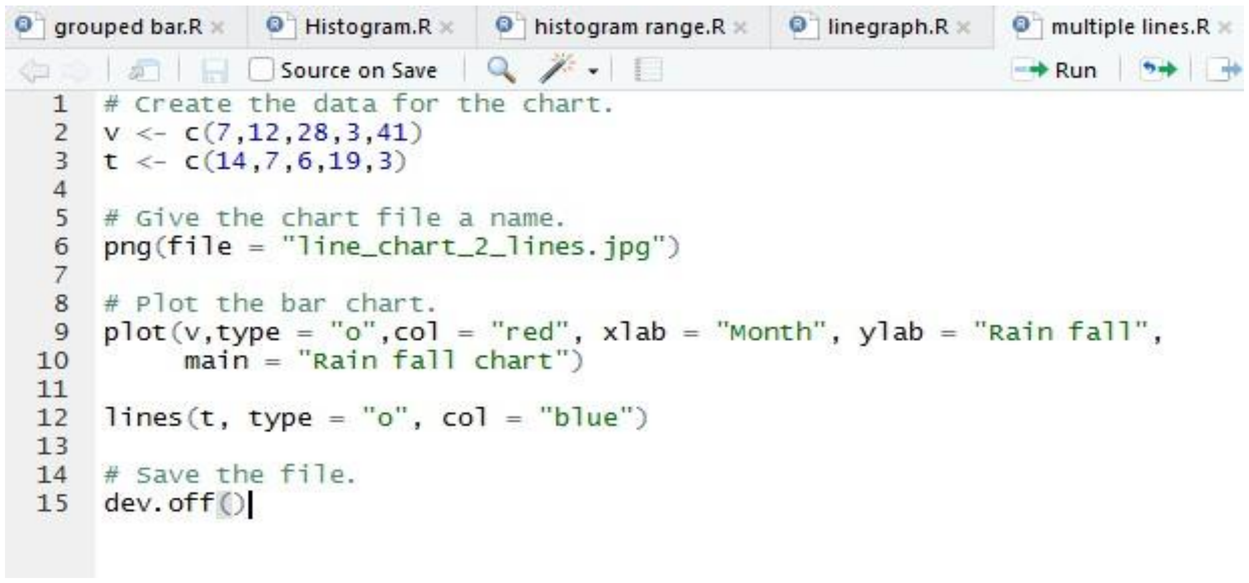


Multiple Lines in a Line Chart

More than one line can be drawn on the same chart by using the **lines()**function.

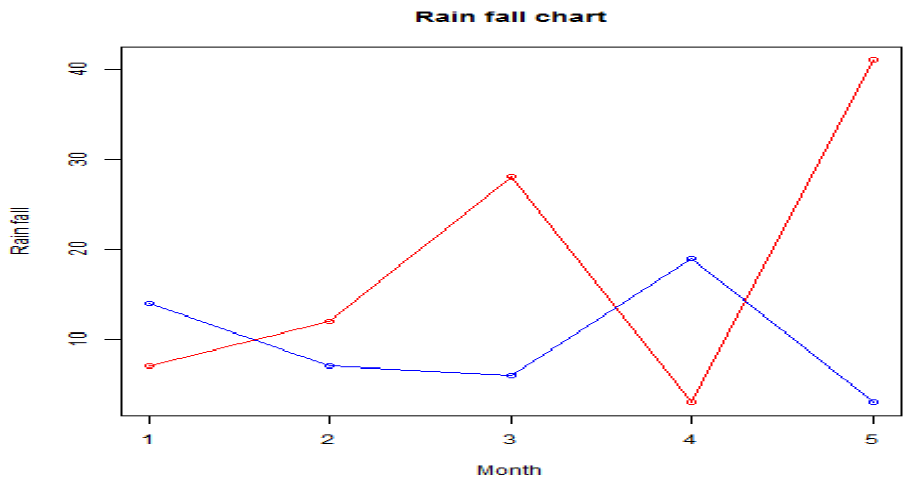
After the first line is plotted, the lines() function can use an additionalvector as input to draw the second line in the chart,

Example:



```
# Create the data for the chart.
v <- c(7,12,28,3,41)
t <- c(14,7,6,19,3)

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

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

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

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

Output:



Scattar Plots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in thehorizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is –

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

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

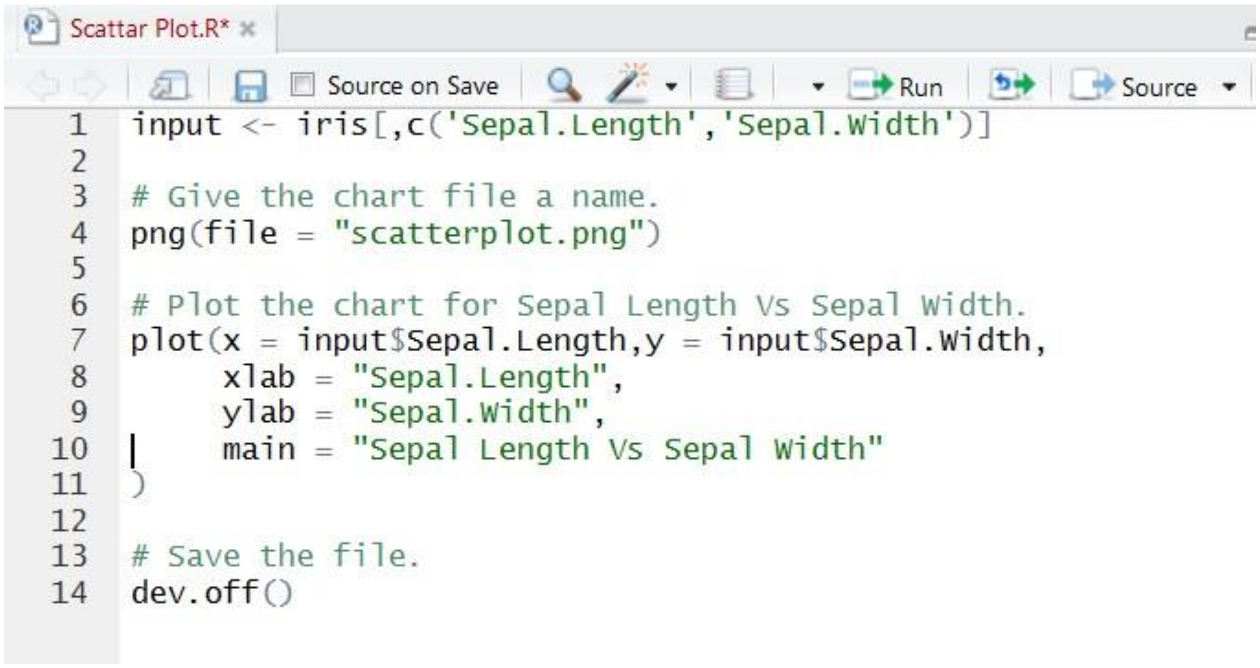
Example

We use the data set **"iris"** available in the R environment to create a basicscatter plot. Let's use the columns "Sepal length" and "Sepal Width" in iris.

Creating the Scatterplot

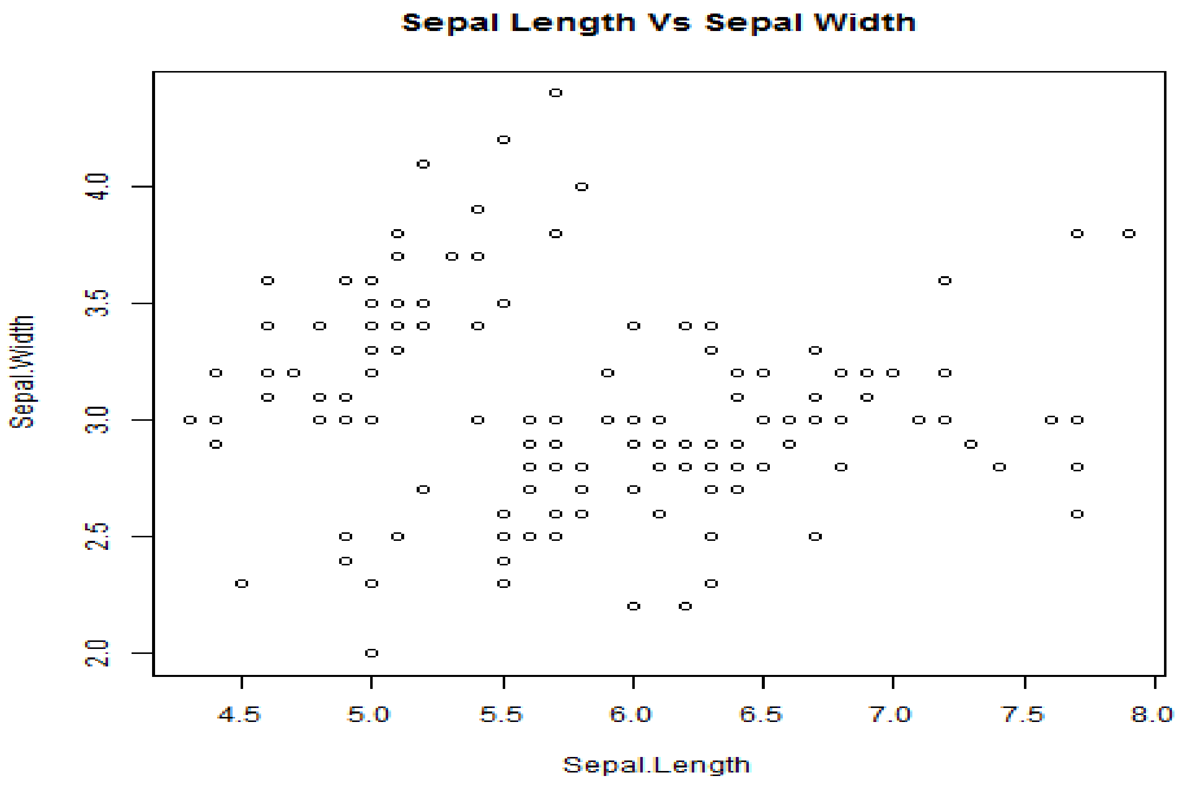
The below script will create a scatterplot graph for the relation between"Sepal length" and "Sepal Width" in iris dataset.

Example:



```r
input <- iris[,c('Sepal.Length','Sepal.Width')]

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

# Plot the chart for Sepal Length Vs Sepal Width.
plot(x = input$Sepal.Length,y = input$Sepal.Width,
     xlab = "Sepal.Length",
     ylab = "Sepal.Width",
     main = "Sepal Length Vs Sepal Width"
)

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

Output:



Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is −

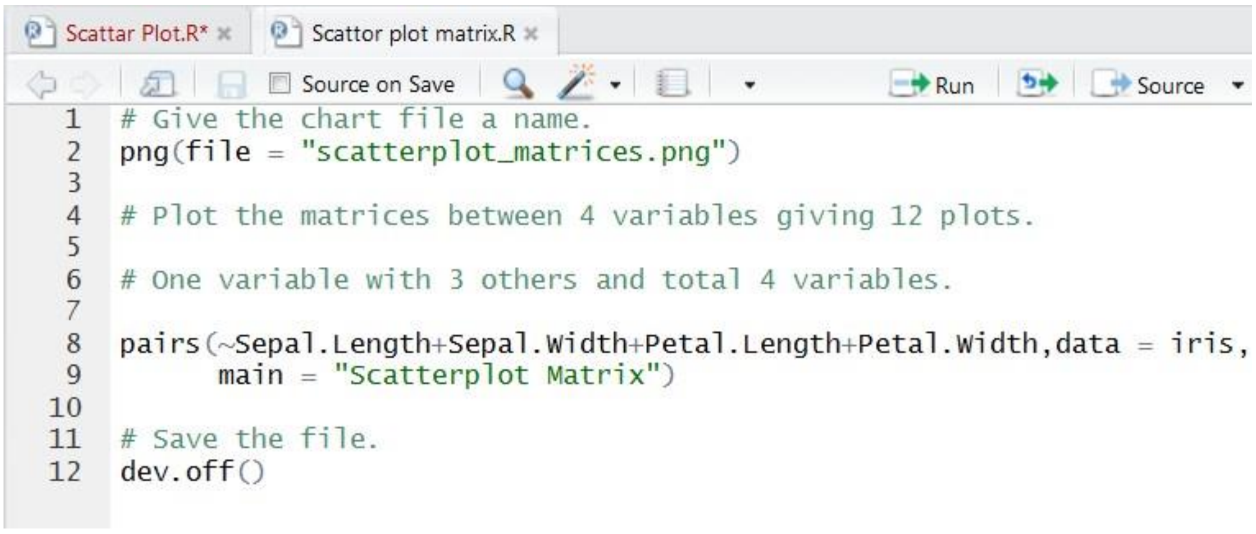
```
pairs(formula, data)
```

Following is the description of the parameters used −

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

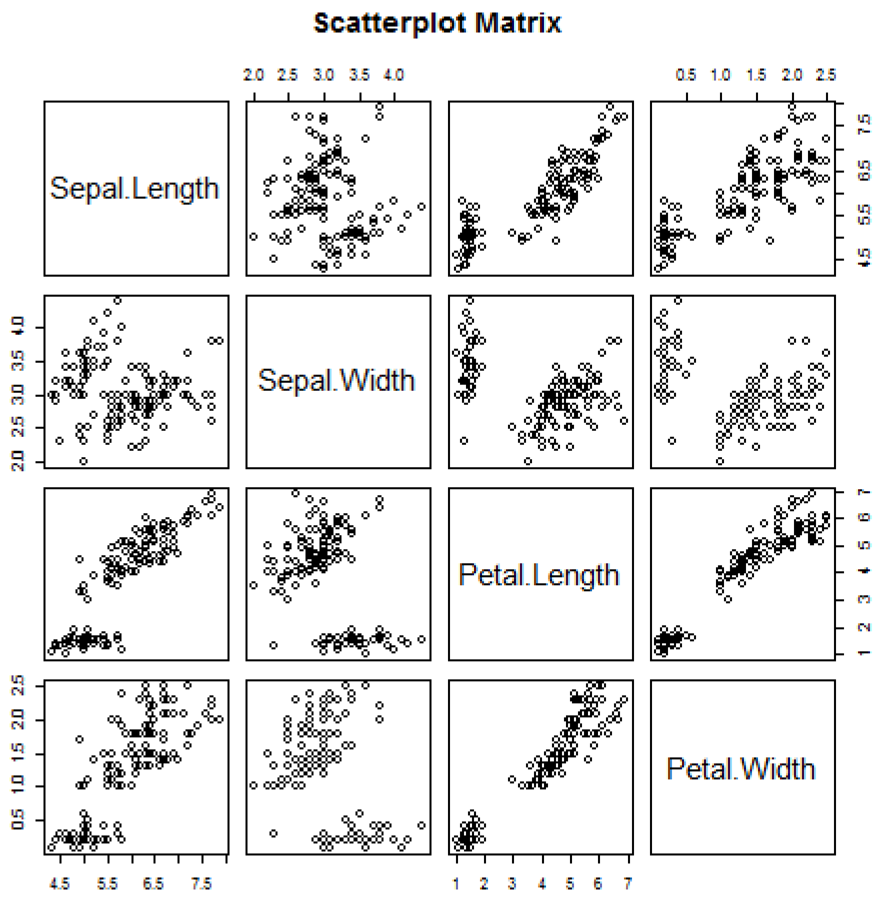
Example

Each variable is paired up with each of the remaining variable. A scatterplot is plottedfor each pair.



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

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,data = iris,
      main = "Scatterplot Matrix")

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

Output:



Box Plot

Boxplots are a measure of how well distributed is the data in a data set. It divides the data set into three quartiles. This graph represents the minimum, maximum, median, first quartile and third quartile in the data set. It is also useful in comparing the distribution of data across data sets by drawing boxplots for each of them.

Boxplots are created in R by using the **boxplot()** function.

Syntax

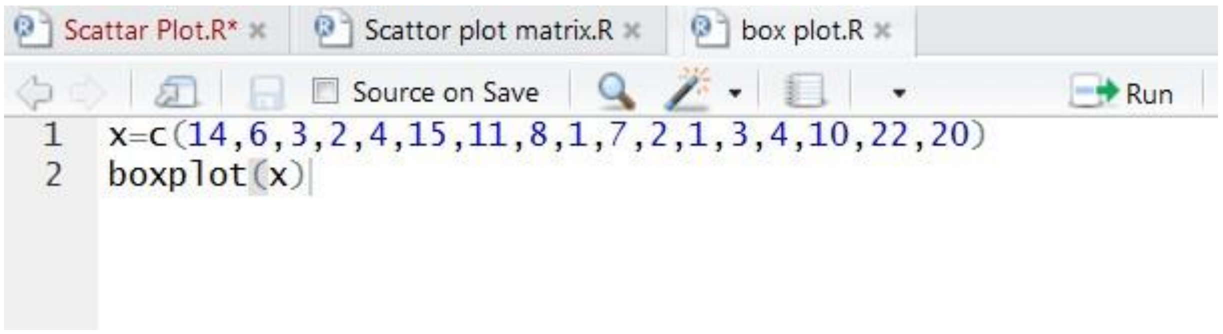
The basic syntax to create a boxplot in R is −

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

Following is the description of the parameters used –

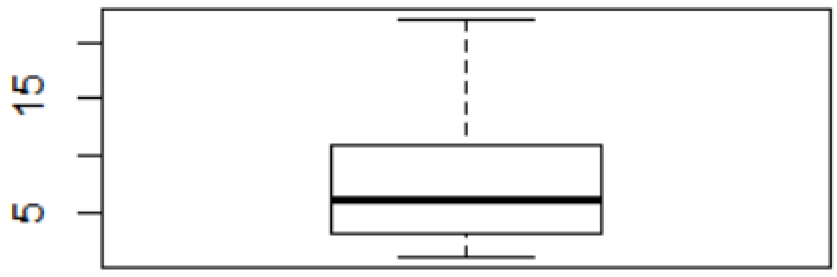
- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate tothe sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.
-

Example 1:

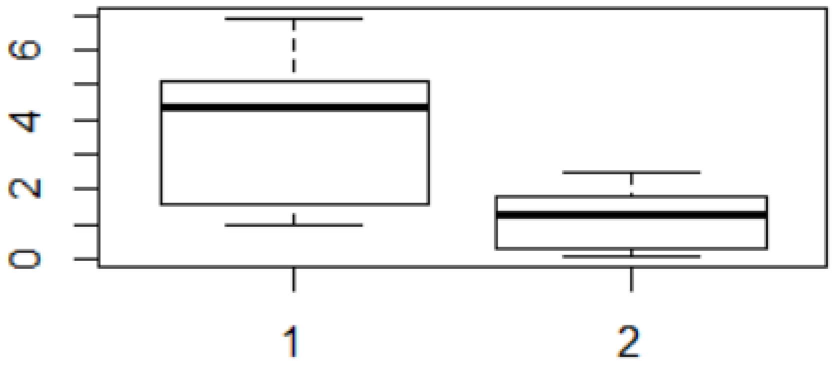


```
x=c(14,6,3,2,4,15,11,8,1,7,2,1,3,4,10,22,20)
boxplot(x)
```

Output:



Example 2: boxplot(iris$Petal.Length,iris$Petal.Width)**Output:**



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY

Unit-III-R Programming – SCS1621

Statistical Modeling in R - Descriptive statistics-R Packages: Regression (MASS package) - Distribution (STATS package) - ANOVA - Time Series Analysis

Statistical Modeling in R

A **statistical model** is a mathematical model that embodies a set of statistical assumptions concerning the generation of some sample data and similar data from a larger population. A statistical model represents, often in considerably idealized form, the data-generating process.

There are several standard statistical models to fit the data using R. The key to modeling in R is the formula object, which provides a shorthand method to describe the exact model to be fit to the data. Modeling functions in R typically require a formula object as an argument. The modeling functions return a model object that contains all the information about the fit

Statistical analysis in R is performed by using many in-built functions. st of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

Descriptive Statistics

Descriptive statistics are brief descriptive coefficients that summarize a given data set, which can be either a representation of the entire population or a sample of it. Descriptive statistics are broken down into measures of central tendency and measures of variability, or spread. Measures of central tendency include the mean, median and mode, while measures of variability includethe standard deviation or variance, the minimum and maximum variables.

Measures of central tendency describe the center position of a distribution for a data set. A person analyzes the frequency of each data point in the distribution and describes it using the mean, median or mode, which measure the most common patterns of the data set being analyzed.

Measures of variability, or the measures of spread, aid in analyzing how spread-out the distribution is for a set of data. For example, while the measures of central tendency may give a person the average of a data set, it doesn't describe how the data is distributed within the set. So, while the average of the data may be 65 out of 100, there can still be data points at both 1 and 100. Measures of variability help communicate this by describing the shape and spread of the data set. Range, quartiles, absolute deviation and variance are all examples of measures ofvariability.

Mean

It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function mean () is used to calculate this in R.Syntax

The basic syntax for calculating mean in R is ,

mean (x, trim = 0, na.rm = FALSE, ...)

x -> is the input vector.

trim ->is used to drop some observations from both end of the sorted vector.

na.rm ->is used to remove the missing values from the input vector.

Median

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

The basic syntax for calculating median in R is

median(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Maximum:

It represents maximum value in the given data set.

The basic syntax for calculating Maximum in R is −

max(x, na.rm = FALSE)

x->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Minimum:
It represents minimum value in the given data set. The basic syntax for calculating Minimum in R

max(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Range:

The difference between the maximum and minimum data entries in the set.

Range = (Max. data entry) – (Min. data entry)The basic syntax for calculating Range in R is −

range(x, na.rm = FALSE)

x ->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

The standard deviation

It measures variability and consistency of the sample or population. In most real-world applications, consistency is a great advantage. In statistical data analysis, less variation is often better.

The basic syntax for calculating Range in R is –

sd(x, na.rm = FALSE)

x->is the input vector.

na.rm ->is used to remove the missing values from the input vector.

Example:

# Create a vector.

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

Find Mean.

 print("Mean :",mean(x)

)# Find Median

print("Median :",median(x))

Find Standard Deviation.

print("Standard Deviation",sd(x))

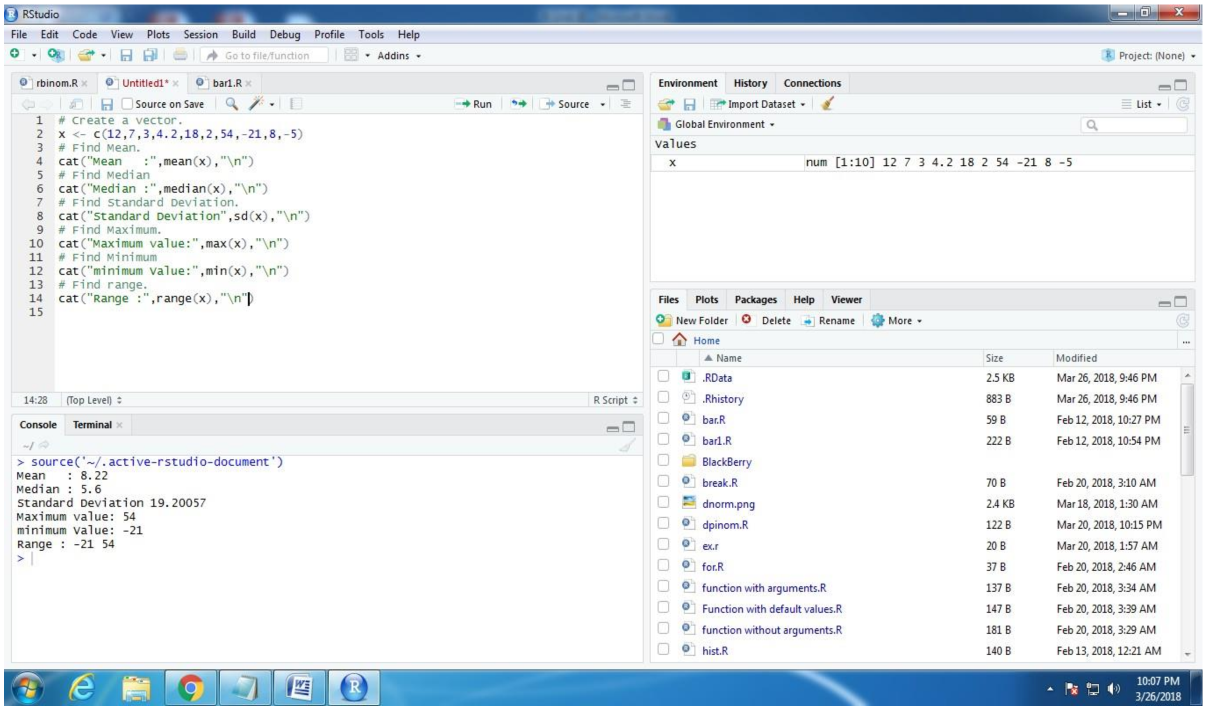
# Find Maximum. print("Maximum value:",max(x))

# Find Minimum print("minimum Value:",min(x))

Find range.

print("Range :",range(x))

Output:



```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)
# Find Mean.
cat("Mean    :",mean(x),"\n")
# Find Median
cat("Median :",median(x),"\n")
# Find Standard Deviation.
cat("Standard Deviation",sd(x),"\n")
# Find Maximum.
cat("Maximum value:",max(x),"\n")
# Find Minimum
cat("minimum Value:",min(x),"\n")
# Find range.
cat("Range :",range(x),"\n")
```

```
> source('~/.active-rstudio-document')
Mean    : 8.22
Median : 5.6
Standard Deviation 19.20057
Maximum value: 54
minimum Value: -21
Range : -21 54
> 
```

```
Values
  x                     num [1:10] 12 7 3 4.2 18 2 54 -21 8 -5
```

Name	Size	Modified
.RData	2.5 KB	Mar 26, 2018, 9:46 PM
.Rhistory	883 B	Mar 26, 2018, 9:46 PM
bar.R	59 B	Feb 12, 2018, 10:27 PM
bar1.R	222 B	Feb 12, 2018, 10:54 PM
BlackBerry		
break.R	70 B	Feb 20, 2018, 3:10 AM
dnorm.png	2.4 KB	Mar 18, 2018, 1:30 AM
dpinom.R	122 B	Mar 20, 2018, 10:15 PM
ex.r	20 B	Mar 20, 2018, 1:57 AM
for.R	37 B	Feb 20, 2018, 2:46 AM
function with arguments.R	137 B	Feb 20, 2018, 3:34 AM
Function with default values.R	147 B	Feb 20, 2018, 3:39 AM
function without arguments.R	181 B	Feb 20, 2018, 3:29 AM
hist.R	140 B	Feb 13, 2018, 12:21 AM

Regression

Linear Regression

Regression analysis is a very widely used statistical tool to establish a relationship model between two variables. One of these variable is called predictor variable whose value is gathered through experiments. The other variable is called response variable whose value is derived from the predictor variable.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve.

The general mathematical equation for a linear regression is −

```
y = ax + b
```

Following is the description of the parameters used −

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

StepstoEstablishaRegression

A simple example of regression is predicting weight of a person when his height is known. Todo this we need to have the relationship between height and weight of a person.

The steps to create the relationship is −

- Carry out the experiment of gathering a sample of observed values of height andcorresponding weight.

- Create a relationship model using the **lm()** functions in R.
- Find the coefficients from the model created and create the mathematical equation usingthese
- Get a summary of the relationship model to know the average error in prediction. Alsocalled **residuals**.
- To predict the weight of new persons, use the **predict()** function in R.

Input Data

Below is the sample data representing the observations −

```
# Values of height
151, 174, 138, 186, 128, 136, 179, 163, 152, 131

# Values of weight.
63, 81, 56, 91, 47, 57, 76, 72, 62, 48
```

lm() Function

This function creates the relationship model between the predictor and the

response variable.Syntax

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

```
lm(formula,data)
```

Following is the description of the parameters used −

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

Create Relationship Model & get the Coefficients

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function

.relation <- lm(y~x) print(relation)

When we execute the above code, it produces the following result −

```
Call:
lm(formula = y

Coefficien
ts:              x
(Intercept     0.67
```

Get the Summary of the Relationship

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

print(summary(relation))

When we execute the above code, it produces the following result −

Output:

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

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

Coefficients:
        Estimate Std. Error t value
Pr(>|t|) (Intercept) -38.45509
                       8.04901  -4.778
0.00139 **
         0.674610.05191  12.997 1.16e-06 ***
```

```
Residual standard error: 3.253 on 8 degrees of
freedom Multiple R-squared:  0.9548,
                              Adjusted R-squared:
```

predict() FunctionSyntax

The basic syntax for predict() in linear regression is –

predict(object, newdata)

Following is the description of the parameters used −

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

Predict the weight of new persons

```
# The predictor vector.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector.

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm() function.

relation <- lm(y~x)

# Find weight of a person with height 170

a <- data.frame(x = 170)

result <-  predict(relation,a)

print(result)
```

When we execute the above code, it produces the following result −

```
       1
76.22869
```

Visualize the Regression Graphically

# Create the predictor and response variable.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

relation <- lm(y~x)

# Give the chart file a name.

png(file = "linearregression.png")

# Plot the chart.

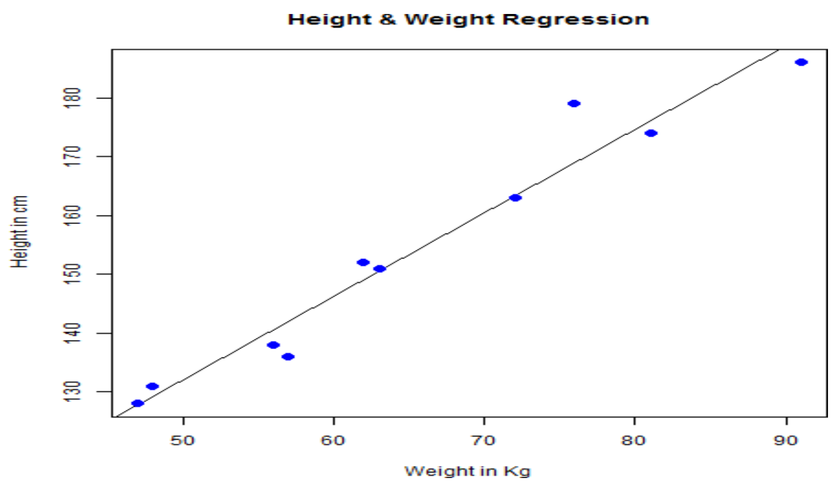
plot(y,x,col = "blue",main = "Height & Weight Regression",

abline(lm(x~y)), cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")

# Save the file.

dev.off()

When we execute the above code, it produces the following result −



R-Multiple Regression

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multiple regression we have more than one predictor variable and one response variable.

The general mathematical equation for multiple regression is −

```
y = a + b1x1 + b2x2 +...bnxn
```

Following is the description of the parameters used −

- **y** is the response variable.
- **a, b1, b2...bn** are the coefficients.
- **x1, x2, ...xn** are the predictor variables.

We create the regression model using the **lm()** function in R. The model determines the value ofthe coefficients using the input data. Next we can predict the value of the response variable for agiven set of predictor variables using these coefficients.

lm() Function

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

Syntax

The basic syntax for **lm()** function in multiple regression is −

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

Following is the description of the parameters used −

- **formula** is a symbol presenting the relation between the response variable and predictorvariables.
- **data** is the vector on which the formula will be applied.

Example:

Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"), horse power("hp"), weight of the car("wt") and some more parameters.The goal of the model is to establish the relationship between "mpg" as a response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars data set for this purpose.

input <- cars[,c("mpg","disp","hp","wt")]

Create Relationship Model & get the Coefficients

input <- mtcars[,c("mpg","disp","hp","wt")]

# Create the relationship model.

model <- lm(mpg~disp+hp+wt, data = input)

# Show the model.

print(model)

# Get the Intercept and coefficients as vector elements.

When we execute the above code, it produces the following result −

```
Call:
lm(formula = mpg ~ disp + hp + wt,

Coefficien
ts:           dis       h        w
(Intercept   -0.000937  -0.031157  -
```

Create Equation for Regression Model

Based on the above intercept and coefficient values, we create the mathematical equation.

```
Y = a+Xdisp.x1+Xhp.x2+Xwt.x3or
Y = 37.15+(-0.000937)*x1+(-0.0311)*x2+(-3.8008)*x3
```

Apply Equation for predicting New Values

We can use the regression equation created above to predict the mileage when a new set of values for displacement, horse power and weight is provided.

For a car with disp = 221, hp = 102 and wt = 2.91 the predicted mileage is

```
Y = 37.15+(-0.000937)*221+(-0.0311)*102+(-3.8008)*2.91 = 22.7104
```

R-Logistic Regression

The Logistic Regression is a regression model in which the response variable (dependent variable) has categorical values such as True/False or 0/1. It actually measures the probability of a binary response as the value of response variable based on the mathematical equation relating it with the predictor variables.

The general mathematical equation for logistic regression is −

```
y = 1/(1+e^-(a+b1x1+b2x2+b3x3+...))
```

Following is the description of the parameters used −

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

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

The basic syntax for **glm()** function in logistic regression is −

```
glm(formula,data,family)
```

Following is the description of the parameters used −

- **formula** is the symbol presenting the relationship between the variables.
- **data** is the data set giving the values of these variables.
- **family** is R object to specify the details of the model. It's value is binomial for logisticregression.

Example

The in-built data set "mtcars" describes different models of a car with their various engine specifications. In "mtcars" data set, the transmission mode (automatic or manual) is described by the column am which is a binary value (0 or 1). We can create a logistic regression model between the columns "am" and 3 other columns - hp, wt and cyl.

```
# Select some columns form mtcars. input

<- mtcars[,c("am","cyl","hp","wt")]
```

CreateRegressionModel

We use the **glm()** function to create the regression model and get its summary for analysis.

```
input <- mtcars[,c("am","cyl","hp","wt")]

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

print(summary(am.data))
```

When we execute the above code, it produces the following result −

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

   Min     1      Media      3       Ma
-          -0.14907  -0.01464

Coefficien


       Estimate                 Std. Error          z value               Pr(>|z|)
       (Intercept)              8.11637             2.428                 0.0152 *
        19.70288
       cyl         0.48760      1.07162             0.455                  0.6491
       hp         0.03259       0.01886             1.728                 0.0840 .
Sig    wt       -9.14947        4.15332             -2.203                0.0276 *
       ---
0.05 ‘.’ 0.1 ‘ ’ 1 (Dispersion parameter for

binomial family taken to be 1)

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

Conclusion

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

DISTRIBUTION

Normal Distribution:

The normal distribution is the most widely known and used of all distributions. Because the normal distribution approximates many natural phenomena so well, it has developed into a standard of reference for many probability problems.

The normal distribution is defined by the following probability density function,

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

where μ is the population mean

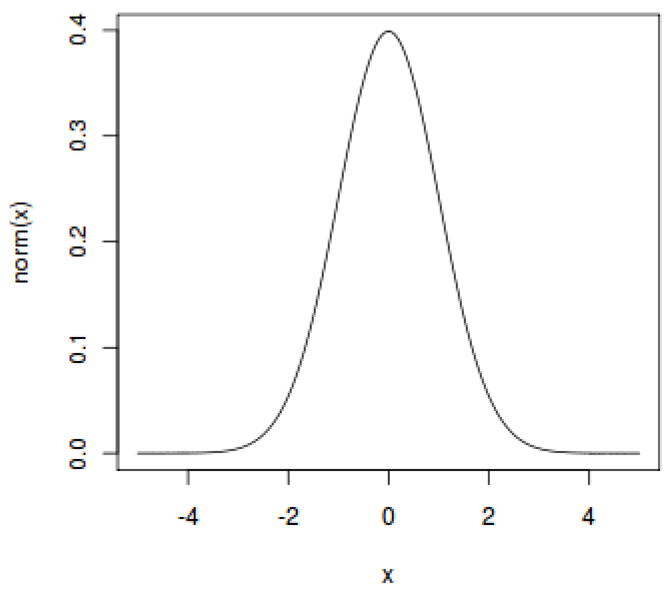
σ is the standard Deviation

Properties of a normal distribution

- The mean, mode and median are all equal.
- The curve is symmetric at the center (i.e. around the mean, μ).
- Exactly half of the values are to the left of center and exactly half the values are to theright.
- The total area under the curve is 1.

Standard Normal Distribution

In particular, the normal distribution with $\mu = 0$ and $\sigma = 1$ is called the standard normal distribution, and is denoted as N (0,1). It can be graphed as follows.



Examples of data which follows a Normal Distribution:

- heights of people
- size of things produced by machines
- errors in measurements
- blood pressure
- marks on a test

R has four in built functions to generate normal distribution. They are described below.

```
dnorm(x, mean, sd)
pnorm(x, mean, sd)
qnorm(p, mean, sd)
rnorm(n, mean, sd)
```

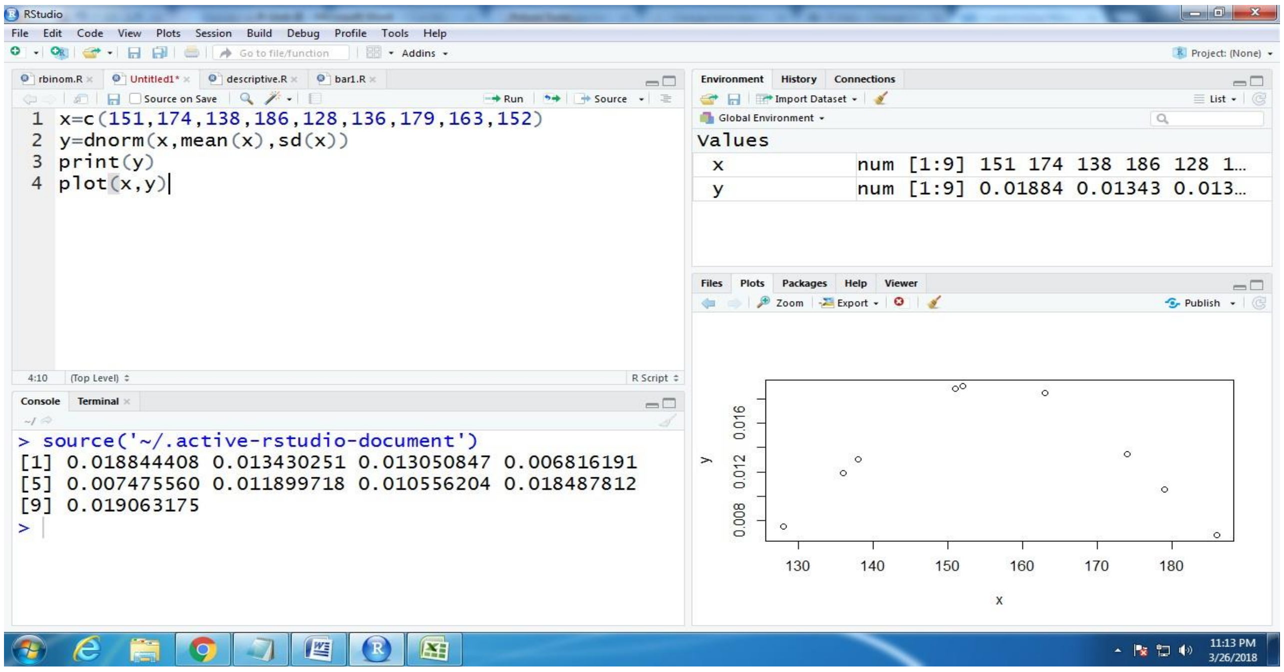
Following is the description of the parameters used in above functions −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations(sample size).
- **mean** is the mean value of the sample data. It's default value is zero.
- **sd** is the standard deviation. It's default value is 1.

dnorm()

This function gives height of the probability distribution at each point for a given mean and standard deviation.

Example and Output:



```
x=c(151,174,138,186,128,136,179,163,152)
y=dnorm(x,mean(x),sd(x))
print(y)
plot(x,y)
```

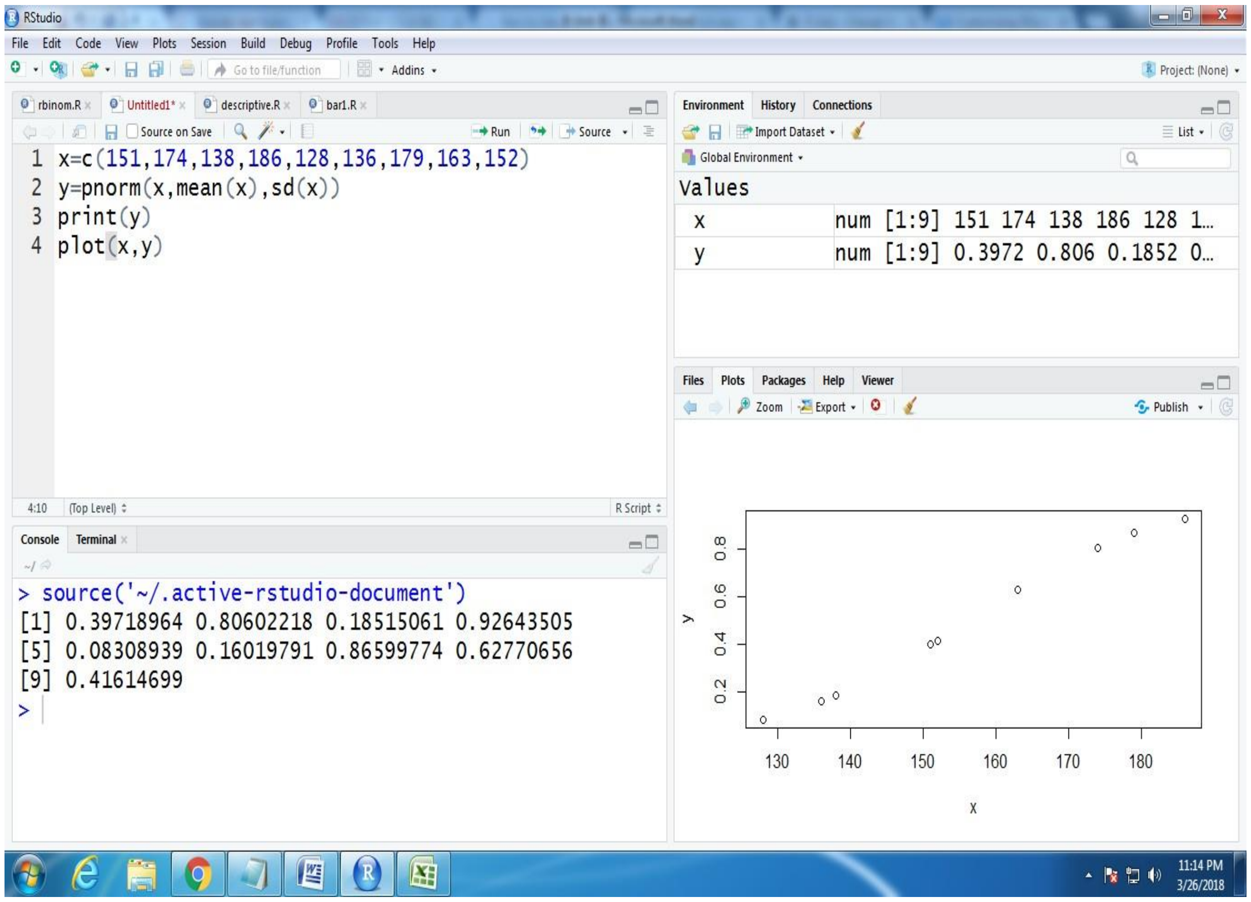
```
Values
 x     num [1:9] 151 174 138 186 128 1…
 y     num [1:9] 0.01884 0.01343 0.013…
```

```
> source('~/.active-rstudio-document')
[1] 0.018844408 0.013430251 0.013050847 0.006816191
[5] 0.007475560 0.011899718 0.010556204 0.018487812
[9] 0.019063175
>
```

pnorm()

This function gives the probability of a normally distributed random number to be less that the value of a given number. It is also called "Cumulative Distribution Function".

Example and Output:



```
x=c(151,174,138,186,128,136,179,163,152)
y=pnorm(x,mean(x),sd(x))
print(y)
plot(x,y)
```

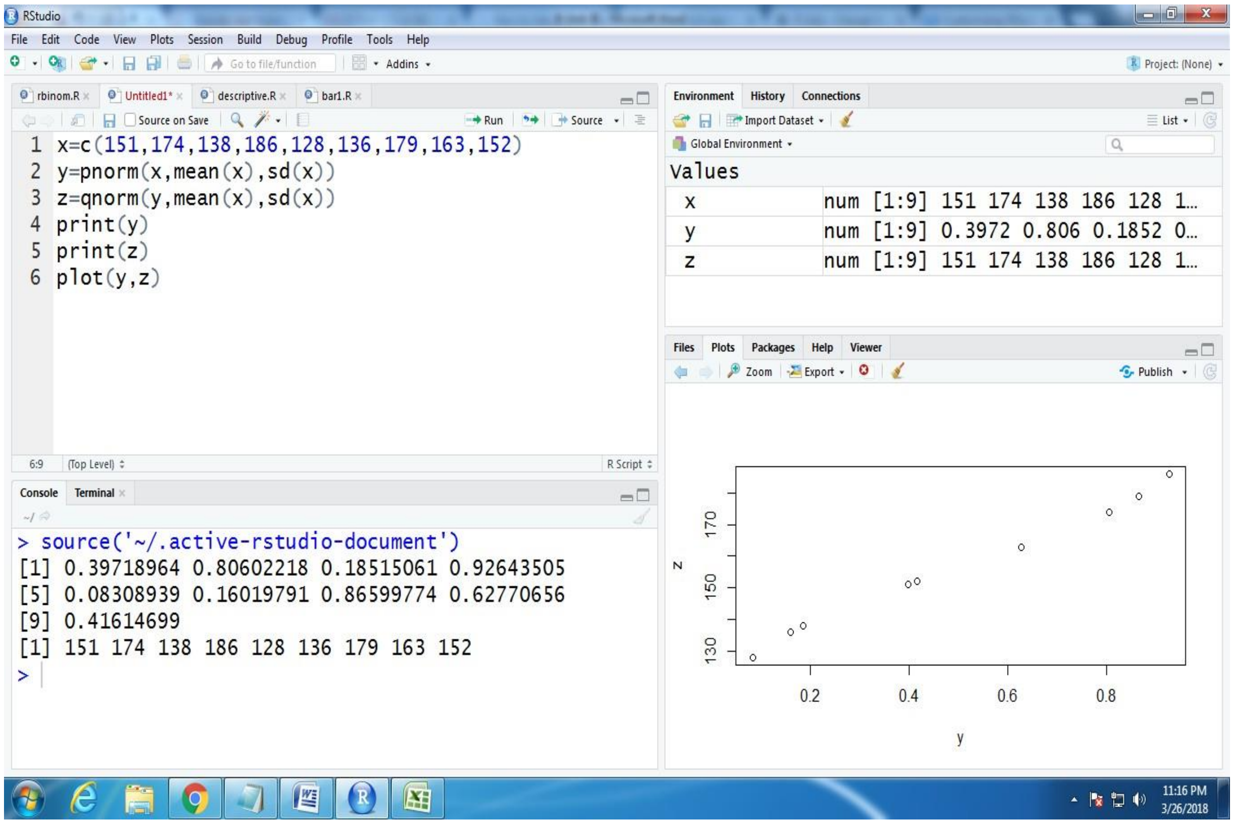
```
Values
x    num [1:9] 151 174 138 186 128 1…
y    num [1:9] 0.3972 0.806 0.1852 0…
```

```
> source('~/.active-rstudio-document')
[1] 0.39718964 0.80602218 0.18515061 0.92643505
[5] 0.08308939 0.16019791 0.86599774 0.62770656
[9] 0.41614699
>
```

qnorm()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

Example and Output:



```
x=c(151,174,138,186,128,136,179,163,152)
y=pnorm(x,mean(x),sd(x))
z=qnorm(y,mean(x),sd(x))
print(y)
print(z)
plot(y,z)
```

```
Values
x    num [1:9] 151 174 138 186 128 1…
y    num [1:9] 0.3972 0.806 0.1852 0…
z    num [1:9] 151 174 138 186 128 1…
```

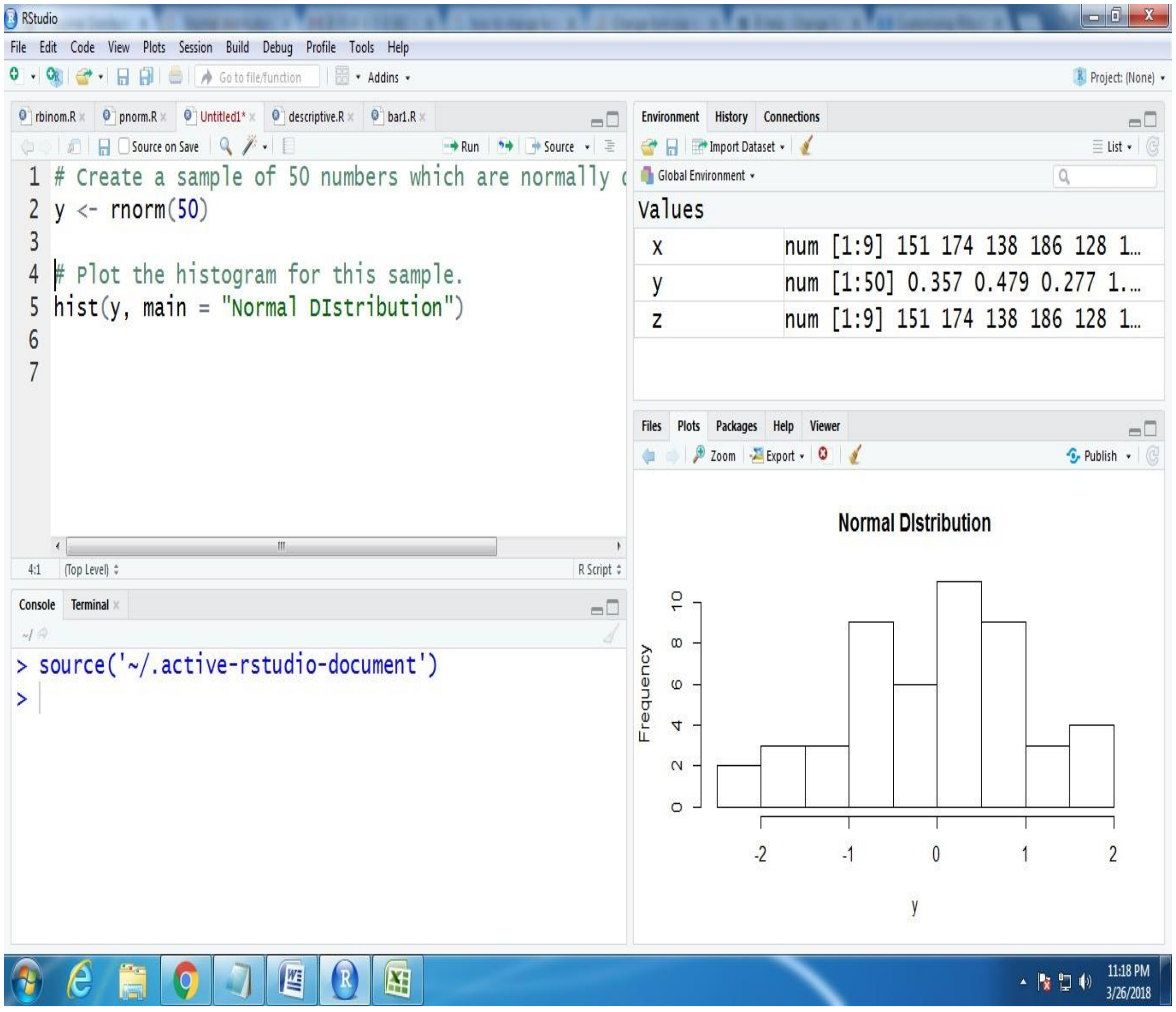
```
> source('~/.active-rstudio-document')
[1] 0.39718964 0.80602218 0.18515061 0.92643505
[5] 0.08308939 0.16019791 0.86599774 0.62770656
[9] 0.41614699
[1] 151 174 138 186 128 136 179 163 152
>
```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
rbinom.R Untitled1* descriptive.R bar1.R
Source on Save
Run
Source
Environment History Connections
Import Dataset
Global Environment
Files Plots Packages Help Viewer
Zoom Export Publish
Console Terminal
z
y
130
150
170
0.2
0.4
0.6
0.8
11:16 PM
3/26/2018

rnorm()

This function is used to generate random numbers whose distribution is normal. It takes the sample size as input and generates that many random numbers. We draw a histogram to show the distribution of the generated numbers.

Example and Output:



```
# Create a sample of 50 numbers which are normally (
y <- rnorm(50)

# Plot the histogram for this sample.
hist(y, main = "Normal DIstribution")
```

```
Values
x    num [1:9] 151 174 138 186 128 1…
y    num [1:50] 0.357 0.479 0.277 1.…
z    num [1:9] 151 174 138 186 128 1…
```

```
> source('~/.active-rstudio-document')
>
```

BINOMIAL DISTRIBUTION

A **binomial distribution** can be thought of as simply the probability of a SUCCESS or FAILURE outcome in an experiment or survey that is repeated multiple times. The binomial is a type of distribution that has **two possible outcomes** (the prefix “bi” means two, or twice). For example, a coin toss has only two possible outcomes: heads or tails and taking a test could have two possible outcomes: pass or fail.

Characteristics of Binomial Distribution

The number of observations or trials is fixed. In other words, you can only figure outthe probability of something happening if you do it a certain number of times. This is common sense — if you toss a coin once, your probability of getting a tails is 50%. If you toss a coin a20 times, your probability of getting a tails is very, very close to 100%.

Each observation or trial is independent. In other words, none of your trials have an effect on the probability of the next trial.

The **probability of success** (tails, heads, fail or pass) is exactly the same from one trial to another.

The binomial distribution formula is:

$$b(x; n, P) = nCx * Px * (1 - P)n - x$$

Where :

b=binomial probability

x=total number of “successes” (pass or fail, heads or tails etc.)

P=probability of a success on an individual trial

n= number of trials

Example:

A coin is tossed 10 times. What is the probability of getting exactly 6 heads?

$b(x; n, P) = {}_nC_x * P^x * (1 – P)^{n – x}$

The number of trials (n) = 10
The odds of success (p) = 0.5q = 0.5
x = 6
P(x=6) = $_{10}C_6$ * 0.5^6 * 0.5^4 = 210 * 0.015625 * 0.0625 = 0.205078125

R has four in-built functions to generate binomial distribution. They are described below.

```
dbinom(x,  size,  prob)

pbinom(x,  size,  prob)

qbinom(p, size, prob)

rbinom(n, size, prob)
```

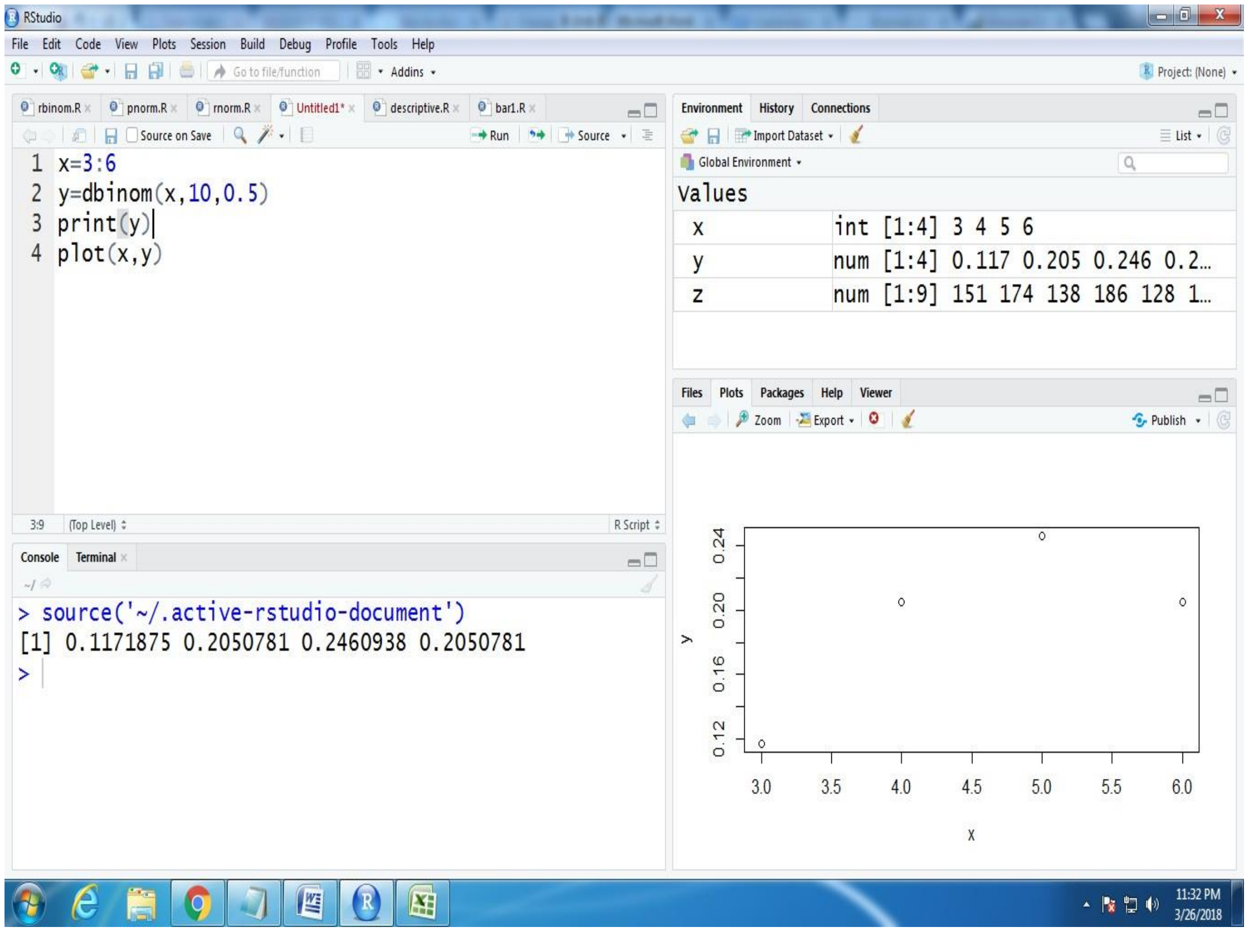
Following is the description of the parameters used −

- **x** is a vector of numbers.
- **p** is a vector of probabilities.
- **n** is number of observations.
- **size** is the number of trials.
- **prob** is the probability of success of each trial.

dbinom()

This function gives the probability density distribution at each point.

Example and Output:



```
x=3:6
y=dbinom(x,10,0.5)
print(y)
plot(x,y)
```

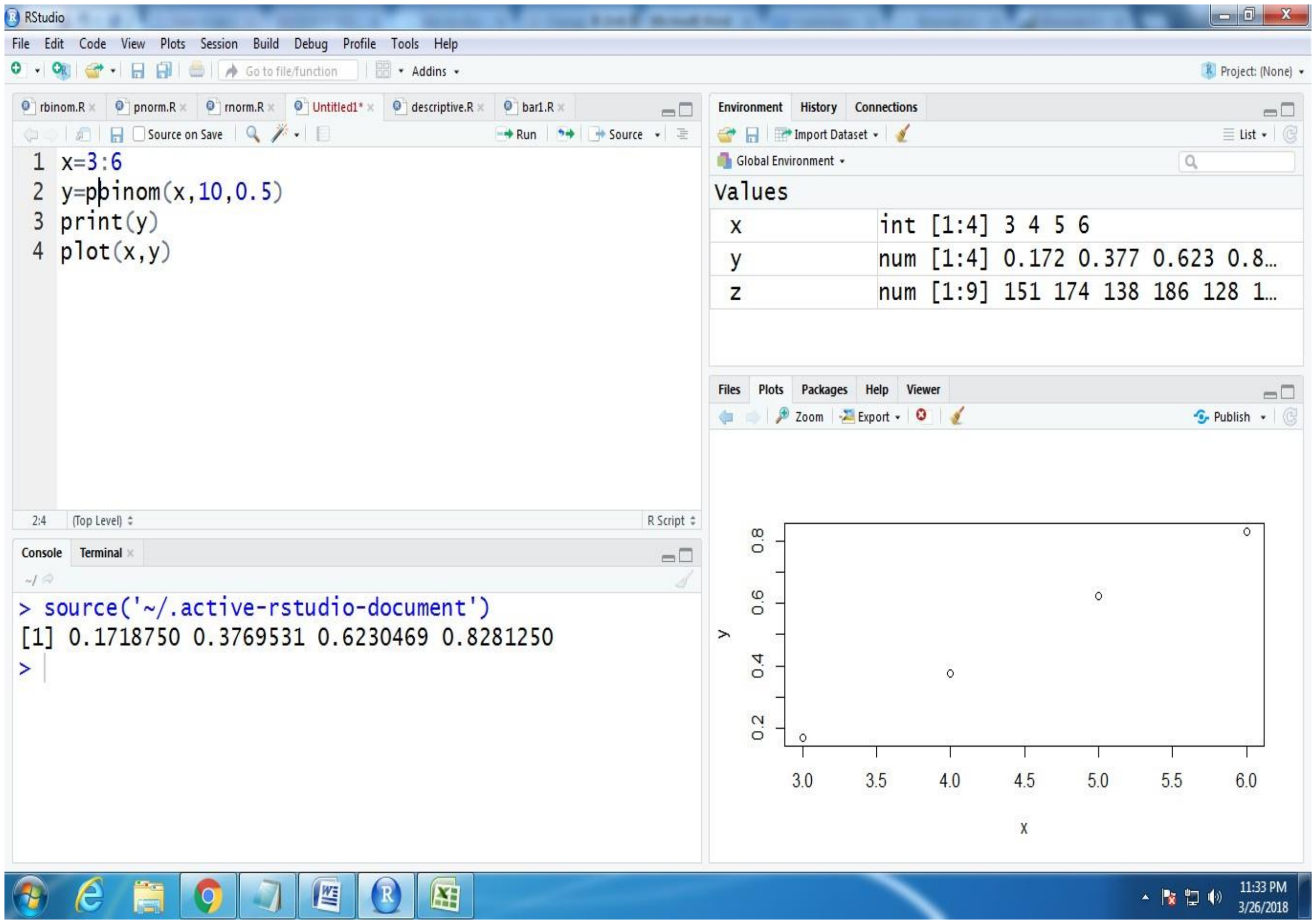
```
Values
x      int [1:4] 3 4 5 6
y      num [1:4] 0.117 0.205 0.246 0.2…
z      num [1:9] 151 174 138 186 128 1…
```

```
> source('~/.active-rstudio-document')
[1] 0.1171875 0.2050781 0.2460938 0.2050781
>
```

pbinom()

This function gives the cumulative probability of an event. It is a single value representing theprobability.

Example and Output:



```
x=3:6
y=pbinom(x,10,0.5)
print(y)
plot(x,y)
```

```
Values
x     int [1:4] 3 4 5 6
y     num [1:4] 0.172 0.377 0.623 0.8…
z     num [1:9] 151 174 138 186 128 1…
```

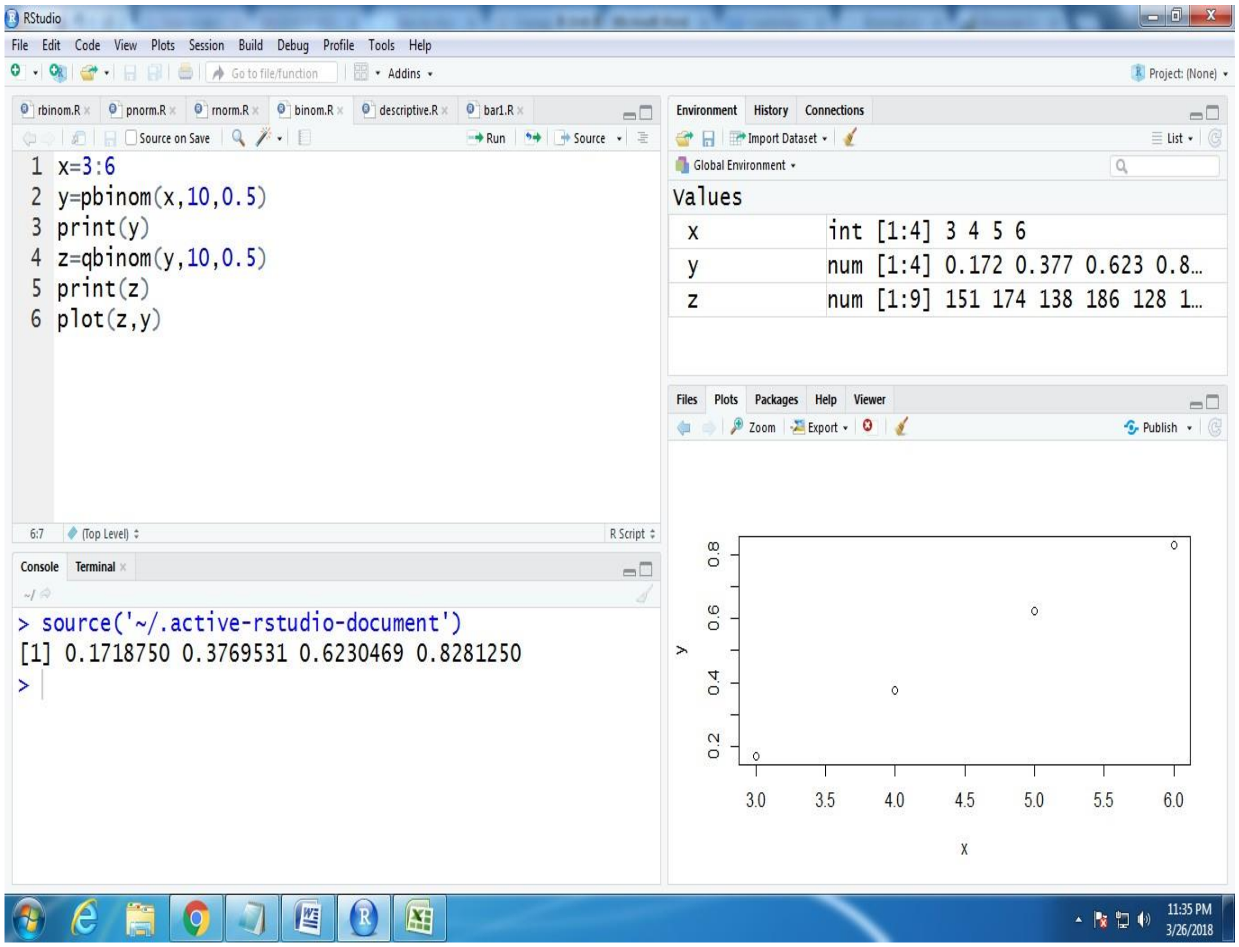
```
> source('~/.active-rstudio-document')
[1] 0.1718750 0.3769531 0.6230469 0.8281250
>
```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
rbinom.R pnorm.R rnorm.R Untitled1* descriptive.R bar1.R
Source on Save
Run
Source
Environment History Connections
Import Dataset
Global Environment
Files Plots Packages Help Viewer
Zoom Export Publish
Console Terminal
y
x
0.2 0.4 0.6 0.8
3.0 3.5 4.0 4.5 5.0 5.5 6.0
11:33 PM
3/26/2018

qbinom()

This function takes the probability value and gives a number whose cumulative value matchesthe probability value.

Example and output:



```
x=3:6
y=pbinom(x,10,0.5)
print(y)
z=qbinom(y,10,0.5)
print(z)
plot(z,y)
```

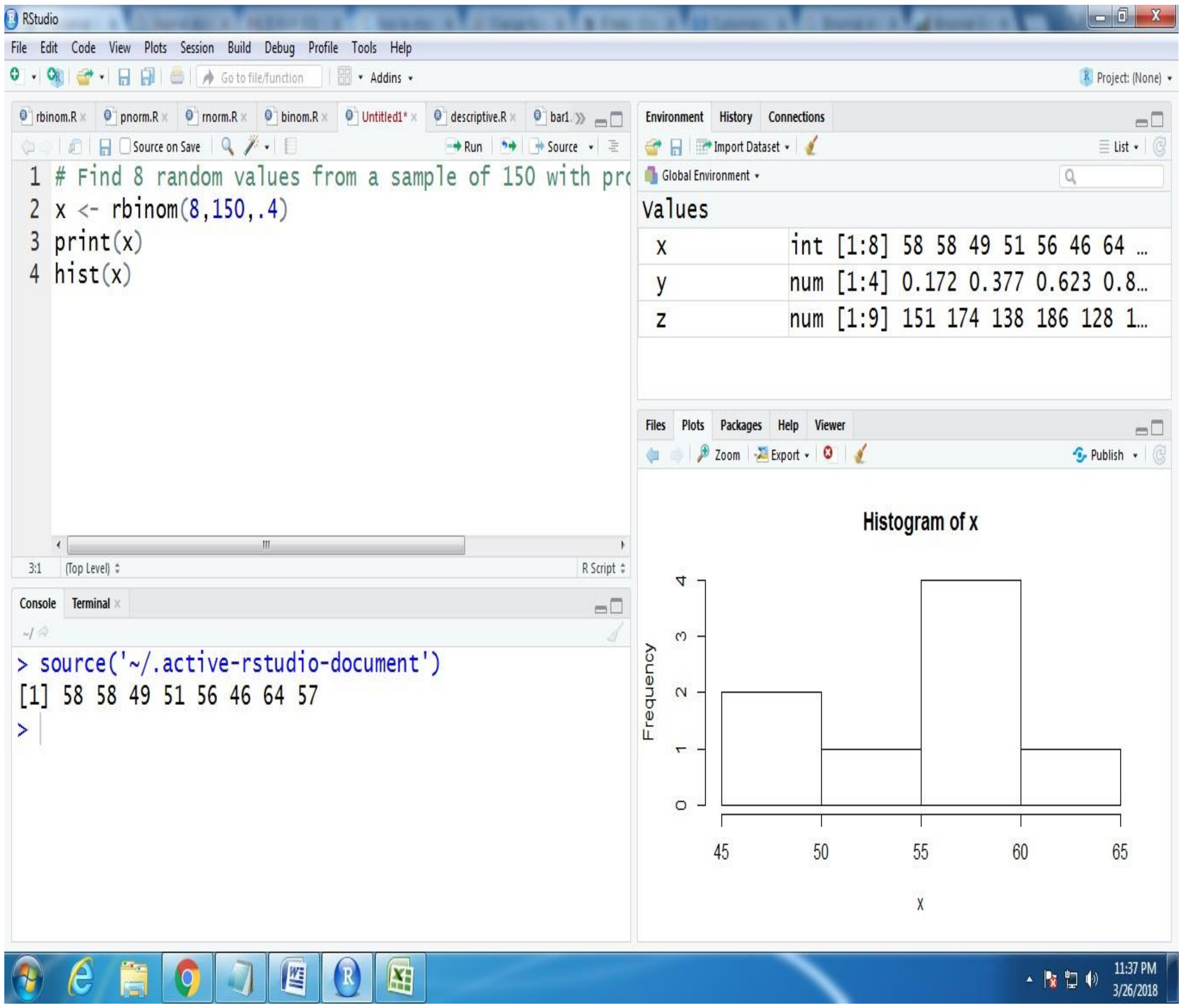
```
> source('~/.active-rstudio-document')
[1] 0.1718750 0.3769531 0.6230469 0.8281250
>
```

```
Values
x    int [1:4] 3 4 5 6
y    num [1:4] 0.172 0.377 0.623 0.8…
z    num [1:9] 151 174 138 186 128 1…
```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
rbinom.R pnorm.R rnorm.R binom.R descriptive.R bar1.R
Environment History Connections
Import Dataset
Global Environment
Files Plots Packages Help Viewer
Zoom Export Publish
Console Terminal
y
x
0.2 0.4 0.6 0.8
3.0 3.5 4.0 4.5 5.0 5.5 6.0
11:35 PM
3/26/2018

rbinom()

This function generates required number of random values of given probability from a givensample.



```
# Find 8 random values from a sample of 150 with pro
x <- rbinom(8,150,.4)
print(x)
hist(x)
```

```
x    int [1:8] 58 58 49 51 56 46 64 ...
y    num [1:4] 0.172 0.377 0.623 0.8...
z    num [1:9] 151 174 138 186 128 1...
```

```
> source('~/.active-rstudio-document')
[1] 58 58 49 51 56 46 64 57
>
```

Time series Analysis

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called **time-series object**. It is also a R data object like a vector or data frame.The time series object is created by using the **ts()** function.

Syntax

The basic syntax for **ts()** function in time series analysis is −

```
timeseries.object.name <-  ts(data, start, end, frequency)
```

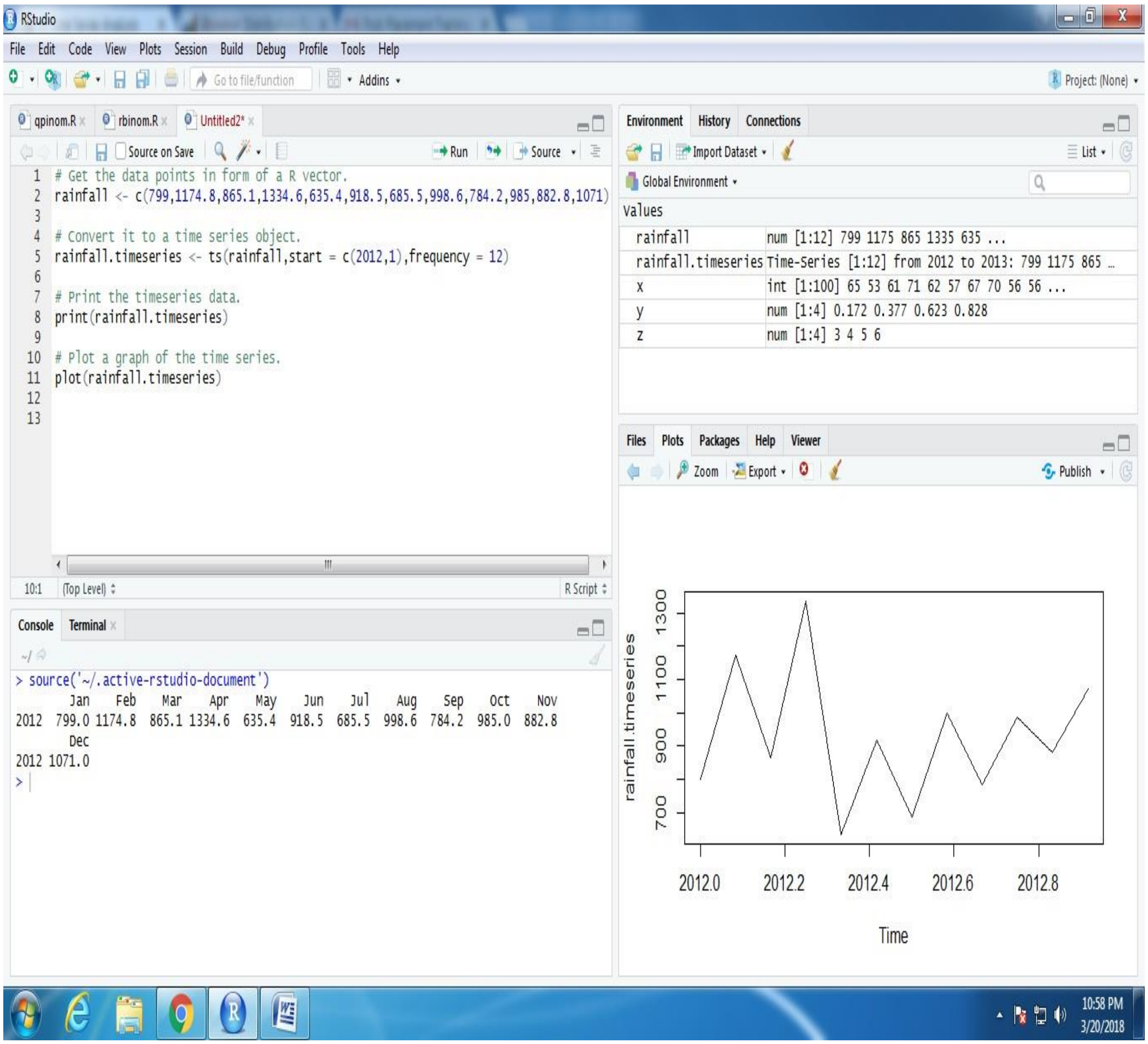
Following is the description of the parameters used −

- **data** is a vector or matrix containing the values used in the time series.
- **start** specifies the start time for the first observation in time series.
- **end** specifies the end time for the last observation in time series.
- **frequency** specifies the number of observations per unit time.

Except the parameter "data" all other parameters are optional.

Example:

Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it.



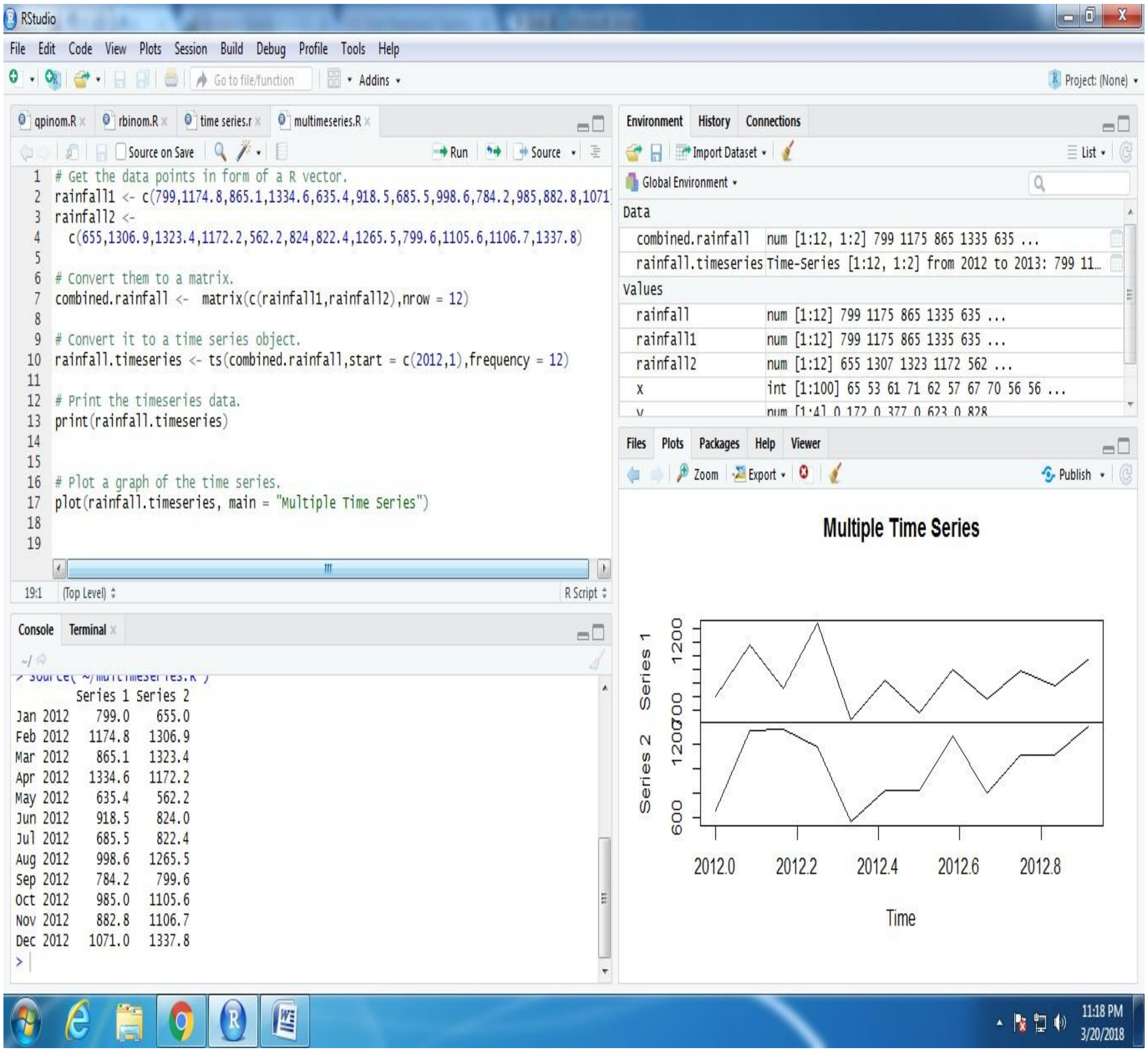
```
# Get the data points in form of a R vector.
rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

# Convert it to a time series object.
rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)

# Print the timeseries data.
print(rainfall.timeseries)

# Plot a graph of the time series.
plot(rainfall.timeseries)
```

```
values
rainfall            num [1:12] 799 1175 865 1335 635 ...
rainfall.timeseries Time-Series [1:12] from 2012 to 2013: 799 1175 865 …
x                   int [1:100] 65 53 61 71 62 57 67 70 56 56 ...
y                   num [1:4] 0.172 0.377 0.623 0.828
z                   num [1:4] 3 4 5 6
```

```
> source('~/.active-rstudio-document')
        Jan    Feb    Mar    Apr    May    Jun    Jul    Aug    Sep    Oct    Nov
2012  799.0 1174.8  865.1 1334.6  635.4  918.5  685.5  998.6  784.2  985.0  882.8
        Dec
2012 1071.0
>
```

Multiple Time Series

We can plot multiple time series in one chart by combining both the series into a matrix.



```r
# Get the data points in form of a R vector.
rainfall1 <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071
rainfall2 <-
  c(655,1306.9,1323.4,1172.2,562.2,824,822.4,1265.5,799.6,1105.6,1106.7,1337.8)

# Convert them to a matrix.
combined.rainfall <-  matrix(c(rainfall1,rainfall2),nrow = 12)

# Convert it to a time series object.
rainfall.timeseries <- ts(combined.rainfall,start = c(2012,1),frequency = 12)

# Print the timeseries data.
print(rainfall.timeseries)


# Plot a graph of the time series.
plot(rainfall.timeseries, main = "Multiple Time Series")
```

```
         Series 1 Series 2
Jan 2012    799.0    655.0
Feb 2012   1174.8   1306.9
Mar 2012    865.1   1323.4
Apr 2012   1334.6   1172.2
May 2012    635.4    562.2
Jun 2012    918.5    824.0
Jul 2012    685.5    822.4
Aug 2012    998.6   1265.5
Sep 2012    784.2    799.6
Oct 2012    985.0   1105.6
Nov 2012    882.8   1106.7
Dec 2012   1071.0   1337.8
>
```

```
Data
combined.rainfall   num [1:12, 1:2] 799 1175 865 1335 635 ...
rainfall.timeseries Time-Series [1:12, 1:2] from 2012 to 2013: 799 11…
Values
rainfall            num [1:12] 799 1175 865 1335 635 ...
rainfall1           num [1:12] 799 1175 865 1335 635 ...
rainfall2           num [1:12] 655 1307 1323 1172 562 ...
x                   int [1:100] 65 53 61 71 62 57 67 70 56 56 ...
```

ANOVA

One Way Anova:

We are often interested in determining whether the means from more than two populations or groups are equal or not. To test whether the difference in means is statistically significant wecan perform analysis of variance (ANOVA) using the R function aov(). If the ANOVA F-test shows there is a significant difference in means between the groups we may want to perform multiple comparisons between all pair-wise means to determine how they differ.

Analysis of Variance

The first step in our analysis is to graphically compare the means of the variable of interest across groups. It is possible to create side-by-side boxplots of measurements organized in groups using the function plot(). Simply type

plot(response ~ factor, data=data_name)

response -> name of the response variable

factor->the variable that separates the data into groups.

Both variables should be contained in a data frame called data_name.

Example:

A drug company tested three formulations of a pain relief medicine for migraine headache sufferers. For the experiment 27 volunteers were selected and 9 were randomly assigned to one of three drug formulations. The subjects were instructed to take the drug during their next migraine headache episode and to report their pain on a scale of 1 to 10 (10 being most pain).

Drug A	4	[illegible]	4	[illegible]	[illegible]	4	[illegible]	4
Drug B	6	[illegible]	4	[illegible]	4	6	[illegible]	6
Drug C	6	[illegible]	6	[illegible]	[illegible]	5	[illegible]	5

To make side-by-side boxplots of the variable pain grouped by the variable drug we must firstread in the data into the appropriate format.

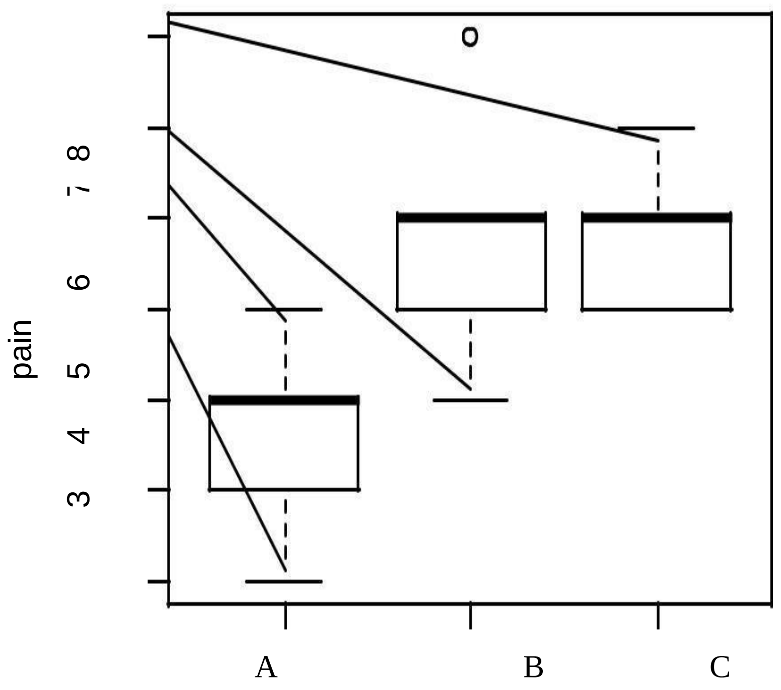
```
> pain = c(4, 5, 4, 3, 2, 4, 3, 4, 4, 6, 8, 4, 5, 4, 6, 5, 8, 6, 6, 7, 6, 6, 7, 5, 6, 5, 5)
> drug = c(rep("A",9), rep("B",9), rep("C",9))
>migraine = data.frame(pain,drug)
```

Note the command rep("A",9) constructs a list of nine A‟s in a row. The variable drug is therefore a list of length 27 consisting of nine A‟s followed by nine B‟s followed by nine C‟s.If we print the data frame migraine we can see the format the data should be on in order to make side-by-side boxplots and perform ANOVA (note the output is cut-off between observations 6- 25 for space purposes).

We can now make the boxplots by typing:

```
>           plot(pain ~ drug, data=migraine)
```

The output of this program is shown below:



From the boxplots it appears that the mean pain for drug A is lower than that for drugs B andC.Next, the R function aov() can be used for fitting ANOVA models. The general form is

aov(response ~ factor, data=data_name)

Once the ANOVA model is fit, one can look at the results using the summary() function. This

P roduces the standard ANOVA table.

<u>Ex.</u> Drug company example continued.

```
>results = aov(pain ~ drug, data=migraine)
>summary(results)
                      Df      Sum Sq     Mean       F          Pr(>F)
                                          Sq         value
          drug                 28.22     14.1111    11.906     0.0002559 ***
                                   2
          Residuals 24        28.44      1.1852
                                  4
          ---
          Signif.             0 „***‟ 0.001 „**‟ 0.01 „*‟ 0.05 „.‟ 0.1 „ ‟ 1
           codes:
```

Studying the output of the ANOVA table above we see that the F-statistic is 11.91 with a p-value equal to 0.0003. We clearly reject the null hypothesis of equal means for all three drug groups.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY

Unit-IV-R Programming – SCS1621

Machine Learning in R - Classification: Decision Trees, Random Forest, SVM – Clustering - Association Rule Mining - Outlier Detection.

Machine learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Some machine learning methods

Machine learning algorithms are often categorized as supervised or unsupervised.

Supervised machine learning algorithms can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

In contrast, **unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn't figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi- supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiringunlabeled data generally doesn't require additional resources.

Reinforcement machine learning algorithms is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

Classification

Classification is a data mining function that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify loan applicants as low, medium, or high credit risks.

Following are the examples of cases where the data analysis task is
Classification −

- A bank loan officer wants to analyze the data in order to know which customer (loan applicant) are risky or which are safe.
- A marketing manager at a company needs to analyze a customer with a given profile, who will buy a new computer.

In both of the above examples, a model or classifier is constructed to predict the categorical labels. These labels are risky or safe for loan application data and yes or no for marketing data.

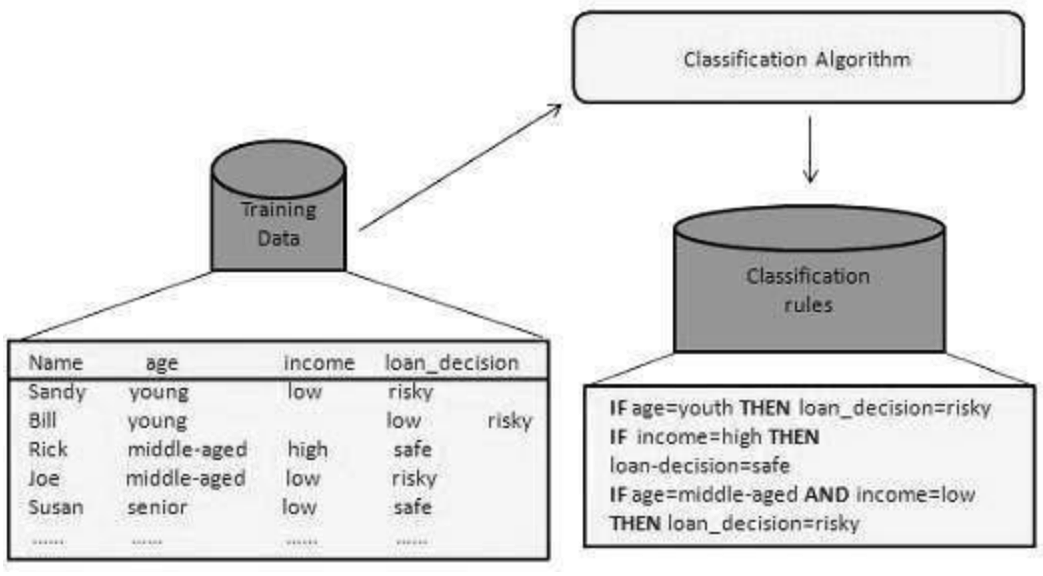
How Does ClassificationWorks?

With the help of the bank loan application that we have discussed above, let us understand the working of classification. The Data Classification process includes two steps −

- Building the Classifier or Model
- Using Classifier for Classification

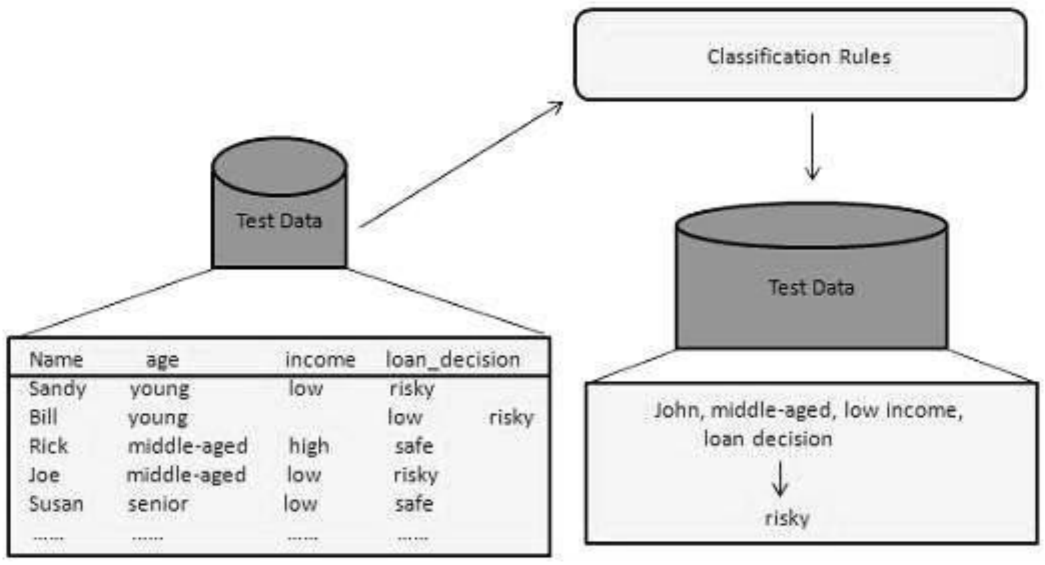
Building the Classifier or Model

- This step is the learning step or the learning phase.
- In this step the classification algorithms build the classifier.
- The classifier is built from the training set made up of database tuples and their associated class labels.
- Each tuple that constitutes the training set is referred to as a category or class. These tuples can also be referred to as sample, object or data points.


Classification Algorithm
Training Data
Classification rules
Name age income loan_decision
Sandy young low risky
Bill young low risky
Rick middle-aged high safe
Joe middle-aged low risky
Susan senior low safe
IF age=youth THEN loan_decision=risky
IF income=high THEN loan-decision=safe
IF age=middle-aged AND income=low THEN loan_decision=risky

Using Classifier for Classification

In this step, the classifier is used for classification. Here the test data is used to estimate the accuracy of classification rules. The classification rules can be applied to the new data tuples if the accuracy is considered acceptable.

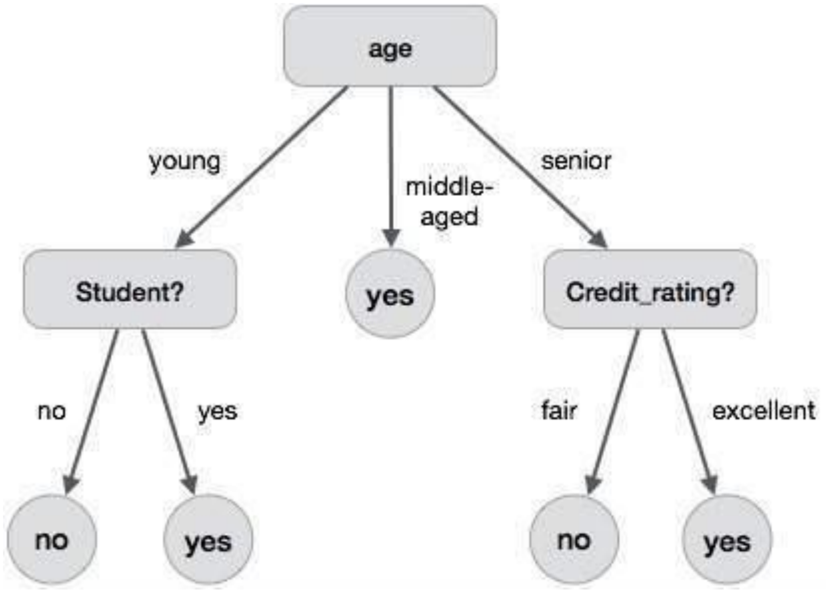

Classification Rules
Test Data
Test Data
Name age income loan_decision
Sandy young low risky
Bill young low risky
Rick middle-aged high safe
Joe middle-aged low risky
Susan senior low safe
John, middle-aged, low income, loan decision
risky

Decision Tree

A decision tree is a structure that includes a root node, branches, and leaf nodes. Each internal node denotes a test on an attribute, each branch denotes the outcome of a test, and each leaf node holds a class label. The topmost node in the tree is the root node.

The following decision tree is for the concept buy_computer that indicates whether a customer at a company is likely to buy a computer or not. Each internal node represents a test on an attribute. Each leaf node represents a class.

The benefits of having a decision tree are as follows −



- It does not require any domain knowledge.
- It is easy to comprehend.
- The learning and classification steps of a decision tree are simple and fast.

Implementation of Decision Tree in R

The R package **"party"** is used to create decision trees.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

```
install.packages("party")
```

The package "party" has the function **ctree()** which is used to create and analyze decision tree.

Syntax

The basic syntax for creating a decision tree in R is −

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

Following is the description of the parameters used −

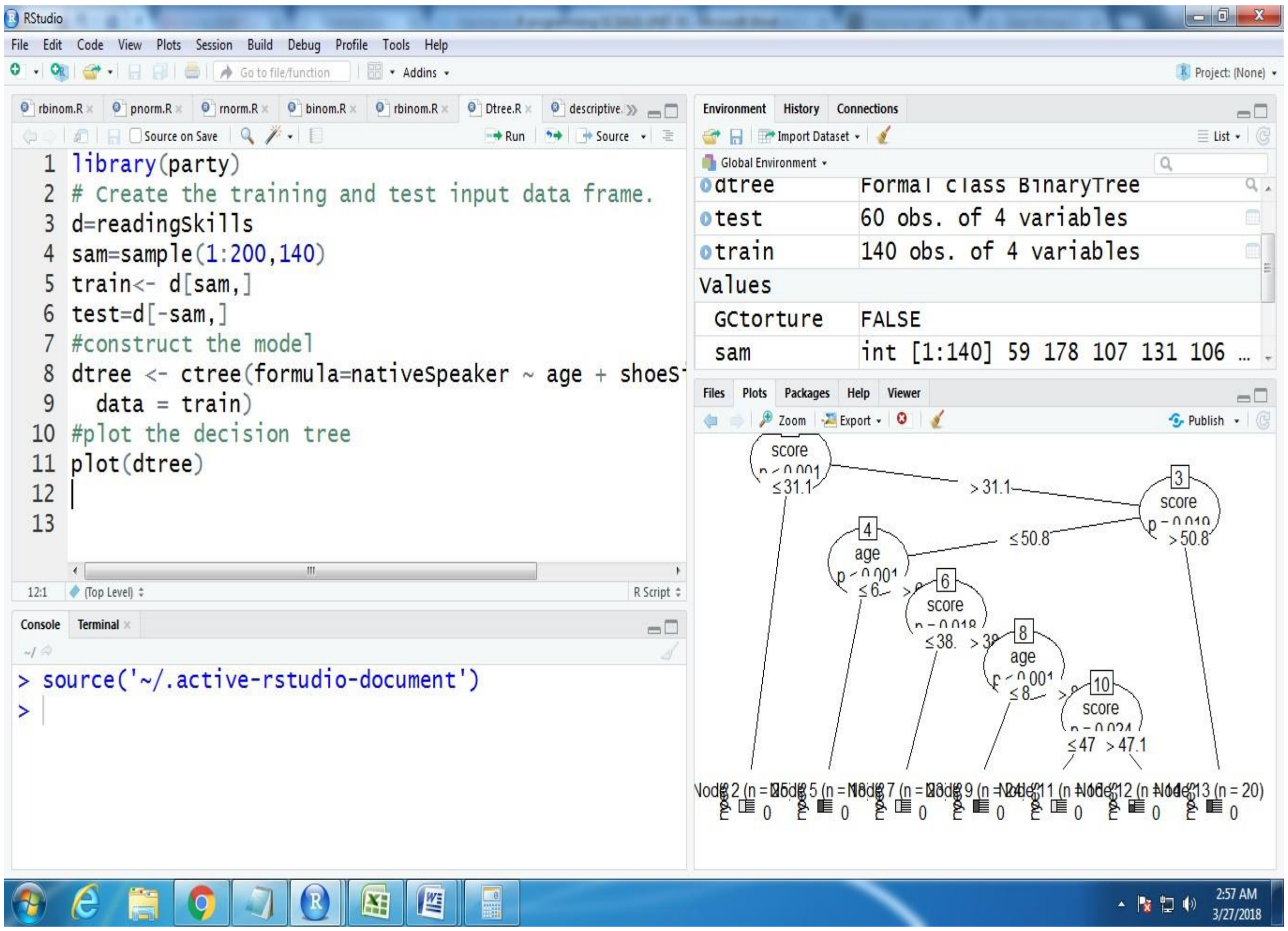
- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.

Input Data

We will use the R in-built data set named **readingSkills** to create a decision tree. It describes the score of someone's readingSkills if we know the variables "age","shoesize","score" and whether the person is a native speaker or not.

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


> dim(readingSkills) [1]
200          4
```

Constructing Decision Tree Model in R

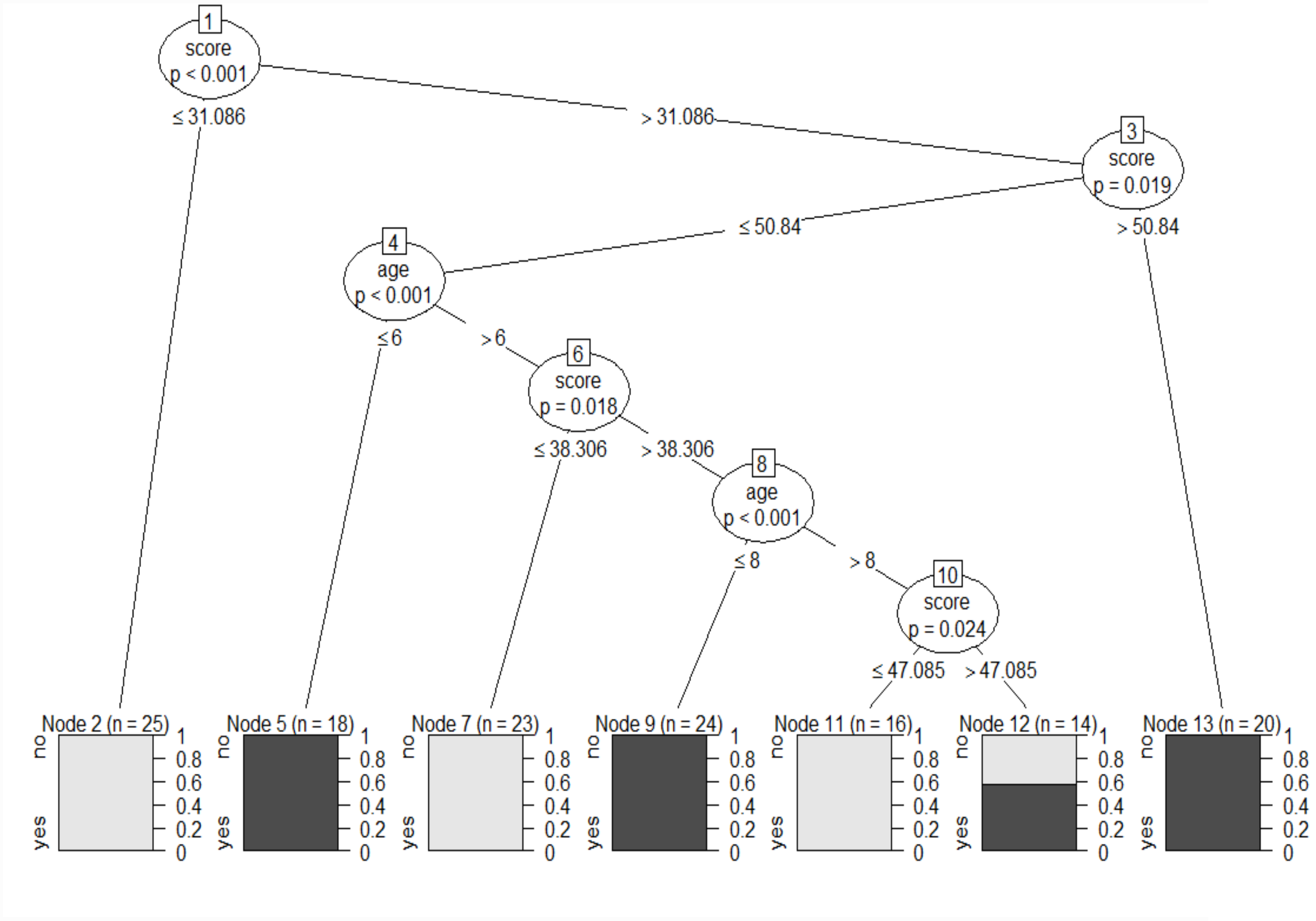


```
library(party)
# Create the training and test input data frame.
d=readingSkills
sam=sample(1:200,140)
train<- d[sam,]
test=d[-sam,]
#construct the model
dtree <- ctree(formula=nativeSpeaker ~ age + shoeS
  data = train)
#plot the decision tree
plot(dtree)
```

```
> source('~/.active-rstudio-document')
>
```

```
dtree    Formal class BinaryTree
test     60 obs. of 4 variables
train    140 obs. of 4 variables
Values
GCtorture  FALSE
sam      int [1:140] 59 178 107 131 106 ...
```

Output Decision Tree Structure:



Prediction Using Test data:

```
> pred=predict(dtree,test)
> pred
 [1] yes yes yes yes no no no no yes no  yes
[12] no no no no yes yes yes no no no  no
[23] yes yes no no no yes no yes yes no
[34] yes yes yes no yes yes no no yes no  yes
[45] yes yes no yes yes yes no yes yes yes yes
[56] no yes yes no yes
Levels: no yes
```

```
> table(pred)
pred
 no   yes
 27 33
```

Checking Accracy of Constrcted Model:

```
> acc=addmargins(table(pred,test$nativeSpeaker))
> acc

 pred no yes Sum no 27
             0      27
   yes  3      30        33

   Sum 30   30          60
```

Accuracy=Total number of correctly classified observation/Total Observation

```
> value=57/60
> value
[1] 0.95
```

Conclusion:

Constructed Decision Tree 95% accurate

Random Forest

In the random forest approach, a large number of decision trees are created. Every observation is fed into every decision tree. The most common outcome for each observation is used as the final output. A new observation is fed into all the trees and taking a majority vote for each classification model.

An error estimate is made for the cases which were not used while building the tree. That is called an **OOB (Out-of-bag)** error estimate which is mentioned as a percentage.

The R package **"randomForest"** is used to create random forests.

Install R Package

Use the below command in R console to install the package. You also have to install the dependent packages if any.

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

The package "randomForest" has the function **randomForest()** which is used to create and analyze random forests.

Syntax

The basic syntax for creating a random forest in R is −

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

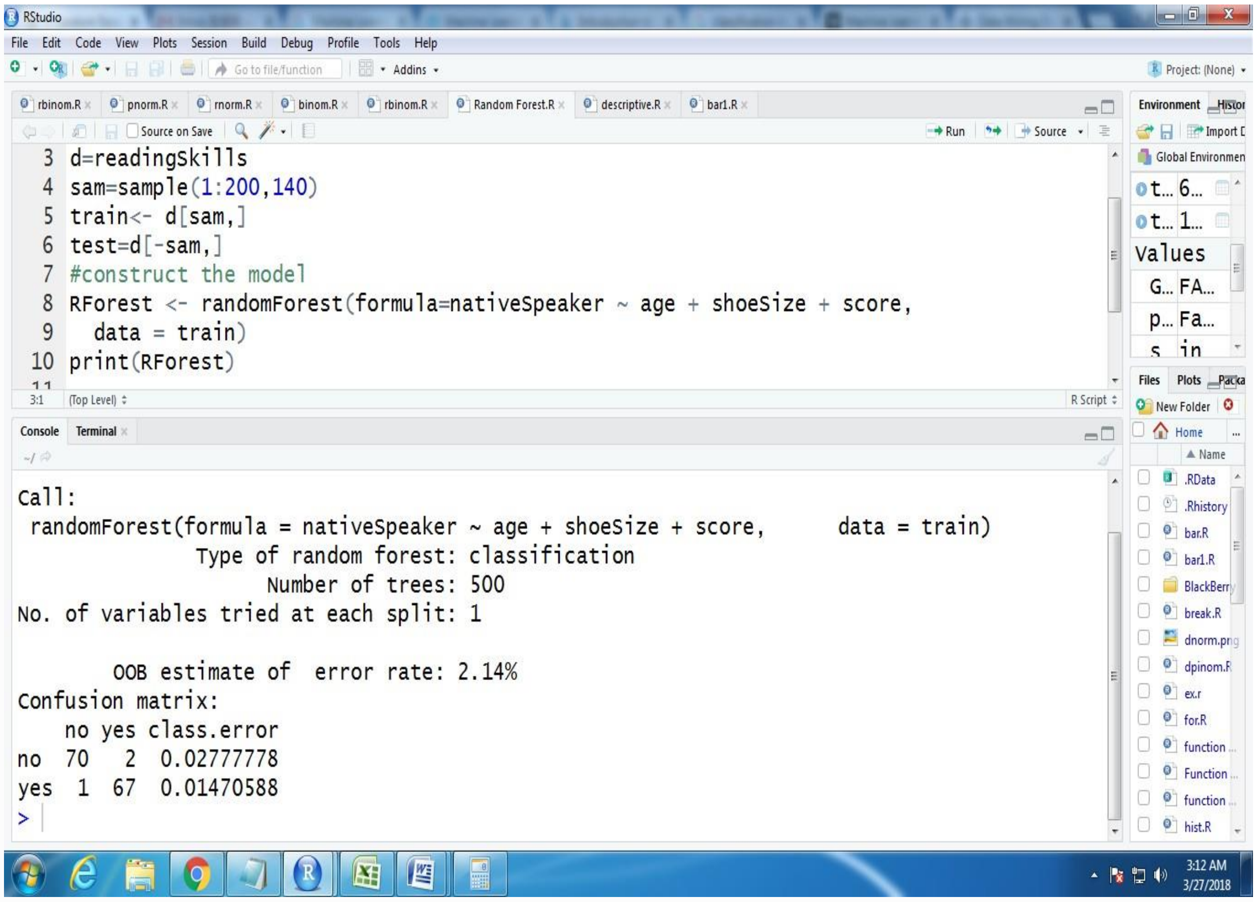
Following is the description of the parameters used −

- **formula** is a formula describing the predictor and response variables.
- **data** is the name of the data set used.

Input Data

We will use the R in-built data set named reading Skills to create a decision tree. It describes the score of someone's reading Skills if we know the variables "age", "shoesize", "score" and whether the person is a native speaker.

Construct Random Forest Model:



```
d=readingSkills
sam=sample(1:200,140)
train<- d[sam,]
test=d[-sam,]
#construct the model
RForest <- randomForest(formula=nativeSpeaker ~ age + shoeSize + score,
  data = train)
print(RForest)
```

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

        OOB estimate of  error rate: 2.14%
Confusion matrix:
    no yes class.error
no  70   2  0.02777778
yes  1  67  0.01470588
>
```

Prediction using Constructed Random Forest Model using Test Data:

> pred=predict(RForest,test)

> pred

1 2 4 5 8 9 10 23 27 30 35 39 42 43 44 46 47 48 57 60 61 62

yes yes yes yes yes yes no no yes yes yes yes no no no yes yes no yes no yes no 68 70
71 72 76 77 79 84 86 89 96 100 101 105 107 109 120 129 131 132 133 141

no yes yes no yes yes no no yes yes no yes yes no yes no no yes no no yes no 142 147

152 154 166 170 173 174 187 188 190 191 193 194 197 200

```
 no yes yes yes no no yes no yes yes yes no no no yes  no
Levels: no yes

> table(pred)
pred
 no yes
 27 33
```

Checking Accuracy of the constructed Model

```
> acc=addmargins(table(pred,test$nativeSpeaker))
> acc

pred no yes Sum no27
     0    27
  yes     1     32        33
  Sum 28       32       60
```

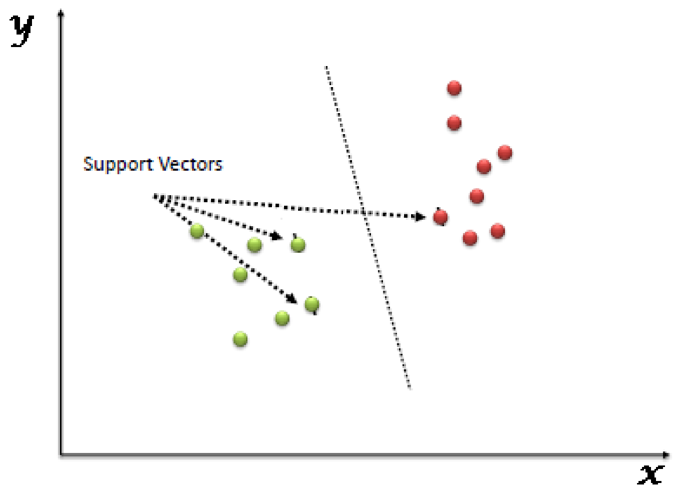
Calculate Accuracy Value from the above table:

```
> accvalue=(27+32)/60
> accvalue [1]
0.9833333
```

Conclusion: Hence the constructed model is 98% Accurate.

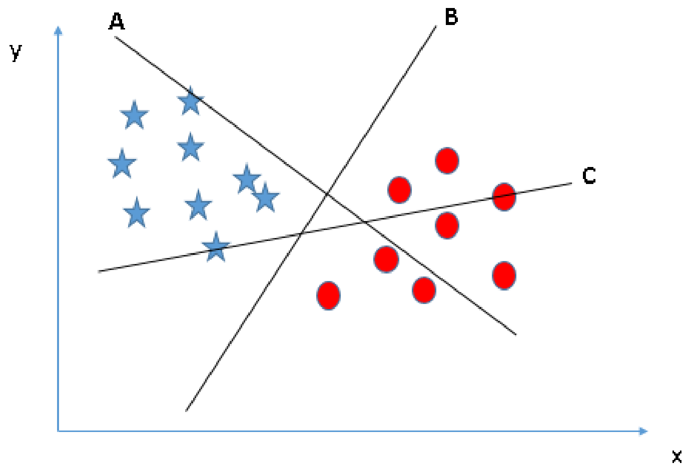
Support Vector Machine

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In this algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper- plane that differentiate the two classes very well (look at the below snapshot).



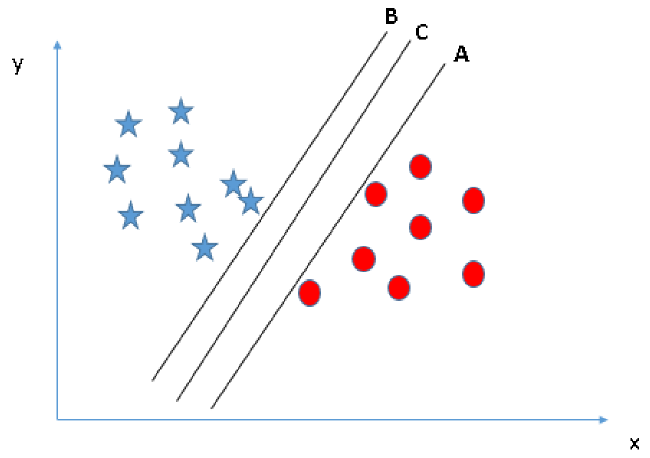
Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line).

Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.

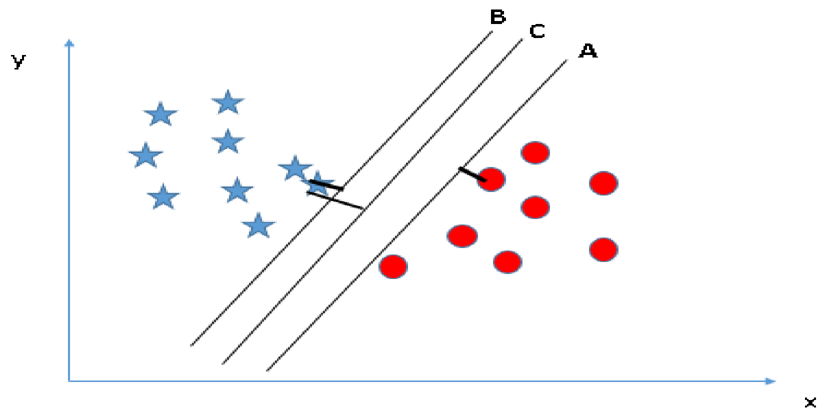


"Select the hyper-plane which segregates the two classes better". In this scenario, hyper-plane "B" has excellently performed this job.

Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?

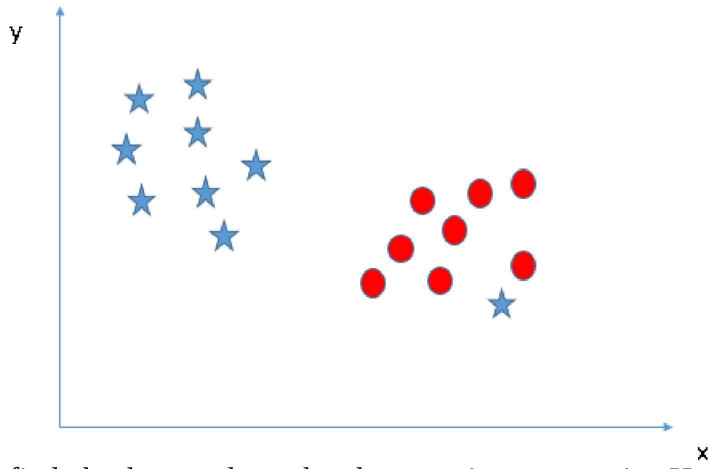


Here, maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**. Consider the below snapshot:

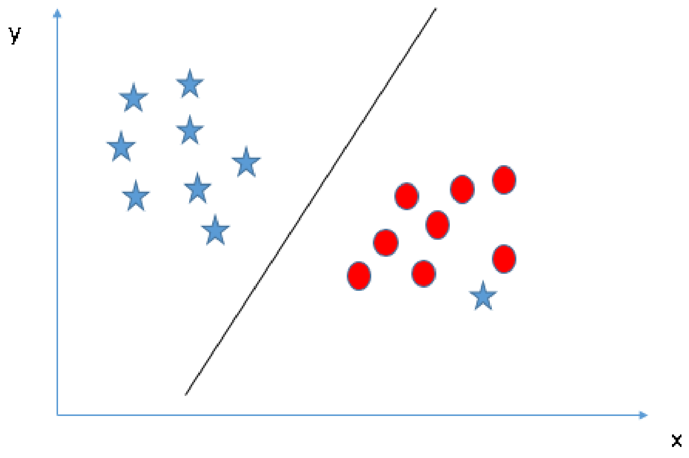


In the above figure, margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C. Another lightning reason for selecting the hyper-plane with higher margin is robustness. If we select a hyper-plane having low margin then there is high chance of miss- classification.

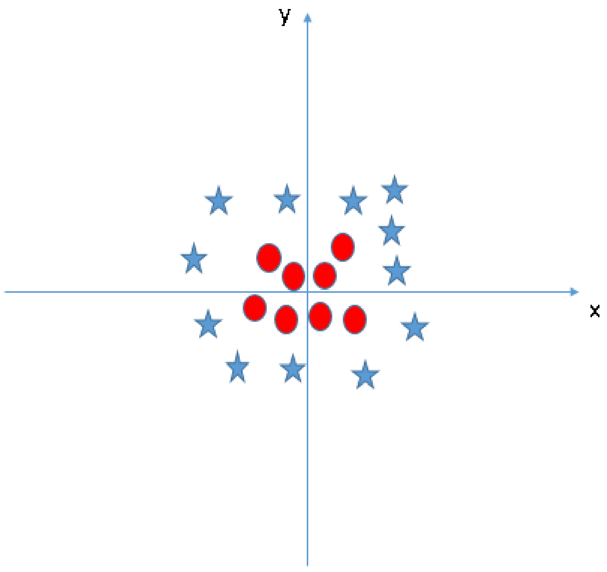
Can we classify two classes (Scenario-3): In below figure , It is unable to segregate the two classes using a straight line, as one of star lies in the territory of other(circle) class as an outlier. Here one star at other end is like an outlier for star class. SVM has a feature to ignore outliers



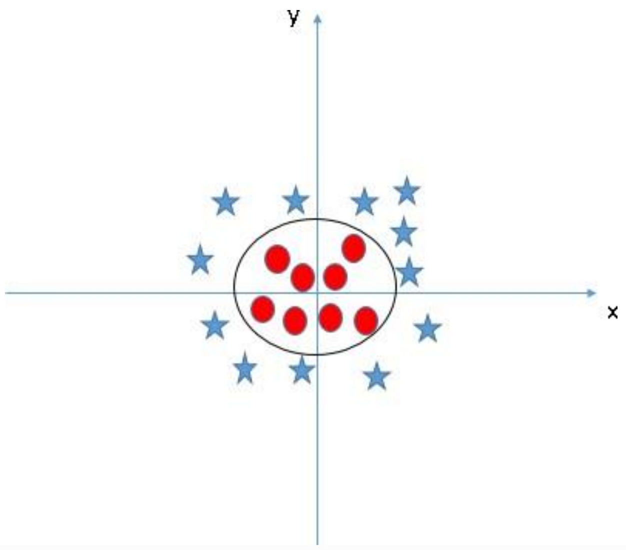
and find the hyper-plane that has maximum margin. Hence, we can say, SVM is robust to outliers.



Find the hyper-plane to segregate to classes (Scenario-4): In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only\looked at the linear hyper-plane.



When we look at the hyper-plane in original input space it looks like a circle:



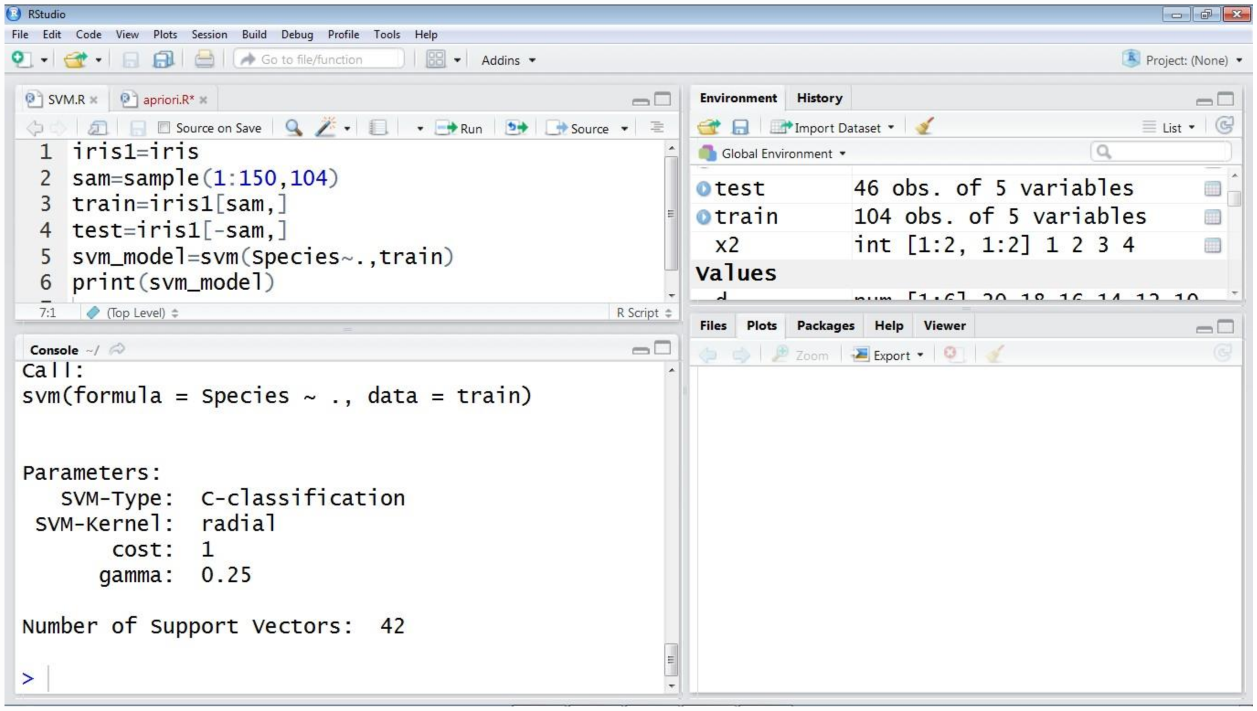
Support Vector Implementation in R

Install the required package e1071 for SVM implementation. Syntax:

Install the pakage e1071.

```
>install.packages(“e1071”)
>library(e1071)
```

SVM Model Construction for iris dataset and it’s output:



```
iris1=iris
sam=sample(1:150,104)
train=iris1[sam,]
test=iris1[-sam,]
svm_model=svm(Species~.,train)
print(svm_model)
```

test 46 obs. of 5 variables
train 104 obs. of 5 variables
x2 int [1:2, 1:2] 1 2 3 4
Values

```
Call:
svm(formula = Species ~ ., data = train)


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.25

Number of Support Vectors:  42

>
```

Predicting values for test data using constrcted model:

```
> pred=predict(svm_model,test)
> table(pred) pred
      setosa  versicolor       virginica
      15        19             12
```

Checking Accuracy of the constrcted Model:

```
> addmargins(table(pred,test$Species))
```

pred	setosa	versicolor	virginica	Sum
setosa	15	0	0	15
versicolor	0	16	3	19
virginica	0	0	12	12
Sum	15	16	15	46

Calculating Accuracy value from the above table:

```
> AccValue=(15+16+12)/46
> AccValue [1]
0.9347826
```

Conclusion:

The Constructed model is 93% Accurate.

Clustering

Cluster is a group of objects that belongs to the same class. In other words, similar objects are grouped in one cluster and dissimilar objects are grouped in another cluster.

What is Clustering?

Clustering is the process of making a group of abstract objects into classes of similar objects.

- A cluster of data objects can be treated as one group.
- While doing cluster analysis, we first partition the set of data into groups based on data similarity and then assign the labels to the groups.
- The main advantage of clustering over classification is that, it is adaptable to changes and helps single out useful features that distinguish different groups.

Applications ofCluster Analysis

- Clustering analysis is broadly used in many applications such as market research, pattern recognition, data analysis, and image processing.
- Clustering can also help marketers discover distinct groups in their customer base. And they can characterize their customer groups based on the purchasing patterns.
- In the field of biology, it can be used to derive plant and animal taxonomies, categorize genes with similar functionalities and gain insight into structures inherent to populations.
- Clustering also helps in identification of areas of similar land use in an earth observation database. It also helps in the identification of groups of houses in a city according to house type, value, and geographic location.
- Clustering also helps in classifying documents on the web for information discovery.

- Clustering is also used in outlier detection applications such as detection of credit card fraud.
- As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.

Requirements of Clustering in Data Mining

The following points throw light on why clustering is required in data mining

- **Scalability** − We need highly scalable clustering algorithms to deal with large databases.
- **Ability to deal with different kinds of attributes** − Algorithms should be capable to be applied on any kind of data such as interval-based (numerical) data, categorical, and binary data.
- **Discovery of clusters with attribute shape** − The clustering algorithm should be capable of detecting clusters of arbitrary shape. They should not be bounded to only distance measures that tend to find spherical cluster of small sizes.
- **High dimensionality** − The clustering algorithm should not only be able to handle low-dimensional data but also the high dimensional space.
- **Ability to deal with noisy data** − Databases contain noisy, missing or erroneous data. Some algorithms are sensitive to such data and may lead to poor quality clusters.
- **Interpretability** − The clustering results should be interpretable, comprehensible, and usable.

Clustering Methods

Clustering methods can be classified into the following categories −

- Partitioning Method
- Hierarchical Method
- Density-based Method
- Grid-Based Method
- Model-Based Method
- Constraint-based Method

Partitioning Method

Suppose we are given a database of ‘n’ objects and the partitioning method constructs ‘k’ partition of data. Each partition will represent a cluster and k
≤ n. It means that it will classify the data into k groups, which satisfy the following
requirements −

- Each group contains at least one object.
- Each object must belong to exactly one group.

Hierarchical Methods

This method creates a hierarchical decomposition of the given set of data objects. We can classify hierarchical methods on the basis of how the hierarchical decomposition is formed. There are two approaches here −

- Agglomerative Approach
- Divisive Approach

Agglomerative Approach

This approach is also known as the bottom-up approach. In this, we start with each object forming a separate group. It keeps on merging the objects or groups that are close to one another. It keep on doing so until all of the groups are merged into one or until the termination condition holds.

Divisive Approach

This approach is also known as the top-down approach. In this, we start with all of the objects in the same cluster. In the continuous iteration, a cluster is split up into smaller clusters. It is down until each object in one cluster or the termination condition holds. This method is rigid, i.e., once a merging or splitting is done, it can never be undone.

Approaches to Improve Quality of Hierarchical Clustering

Here are the two approaches that are used to improve the quality of
hierarchical clustering −

- Perform careful analysis of object linkages at each hierarchical partitioning.
- Integrate hierarchical agglomeration by first using a hierarchical agglomerative algorithm to group objects into micro-clusters, and then performing macro- clustering on the micro-clusters.

Density-based Method

This method is based on the notion of density. The basic idea is to continue growing the given cluster as long as the density in the neighborhood exceeds some threshold, i.e., for each data point within a given cluster, the radius of a given cluster has to contain at least a minimum number of points.

Grid-based Method

In this, the objects together form a grid. The object space is quantized into finite number of cells that form a grid structure.

Advantage

- The major advantage of this method is fast processing time.
- It is dependent only on the number of cells in each dimension in the quantized space.

Model-based methods

In this method, a model is hypothesized for each cluster to find the best fit of data for a given model. This method locates the clusters by clustering the density function. It reflects spatial distribution of the data points.

This method also provides a way to automatically determine the number of clusters based on standard statistics, taking outlier or noise into account. It therefore yields robust clustering methods.

Constraint-based Method

In this method, the clustering is performed by the incorporation of user or application-oriented constraints. A constraint refers to the user expectation or the properties of desired clustering results. Constraints provide us with an interactive way of communication with the clustering process. Constraints can be specified by the user or the application requirement.

K-Means Clustering

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize those items into groups. To achieve this, we will use the k Means algorithm; an unsupervised learning algorithm.

(It will help if you think of items as points in an n-dimensional space). The algorithm will categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.

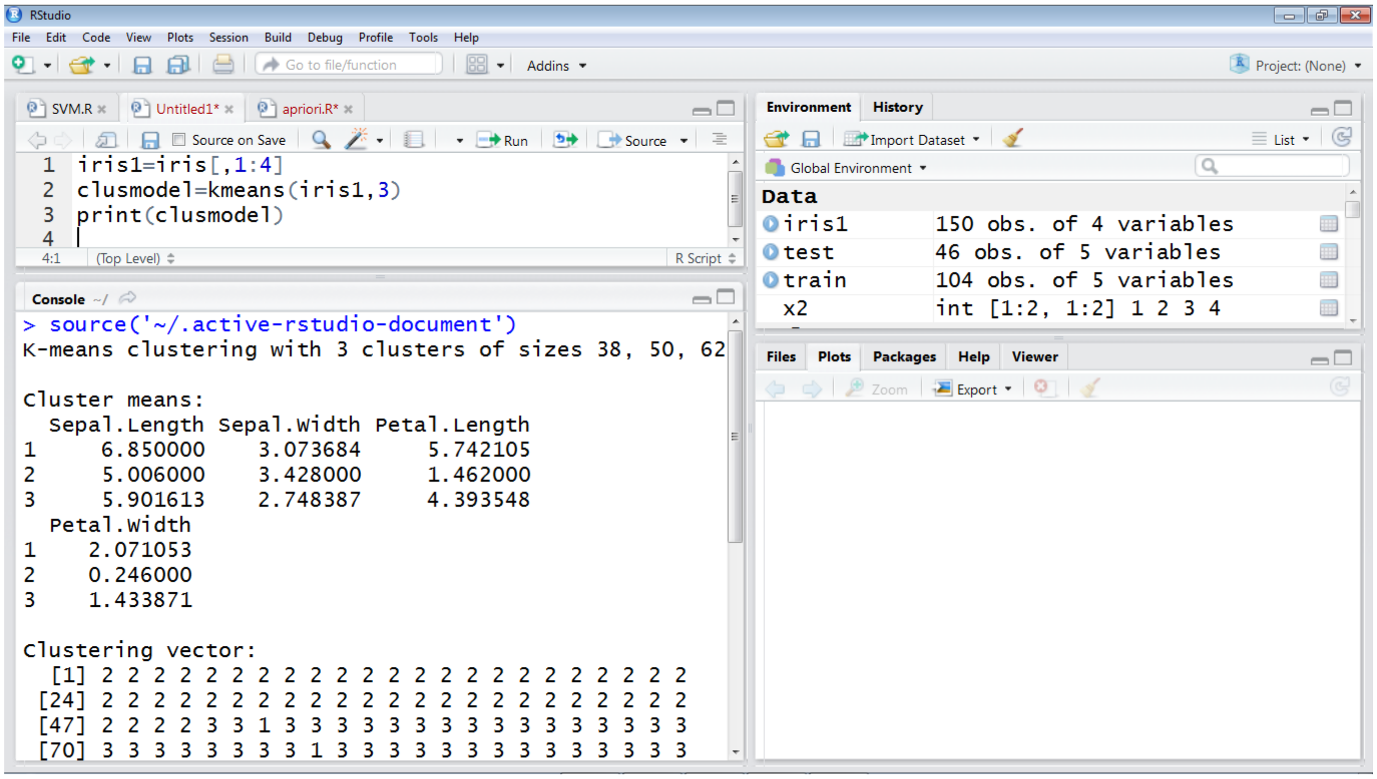
The algorithm works as follows:

1. First we initialize k points, called means, randomly.
2. We categorize each item to its closest mean and we update the mean’s coordinates, which are the averages of the items categorized in that mean so far.
3. We repeat the process for a given number of iterations and at the end, we have our clusters.

The “points” mentioned above are called means, because they hold the mean values of the items categorized in it. To initialize these means, we have a lot of options. An intuitive method is to initialize the means at random items in the data set. Another method is to initialize the means at random values between the boundaries of the data set .

K Means Clustering Implementaion in R.

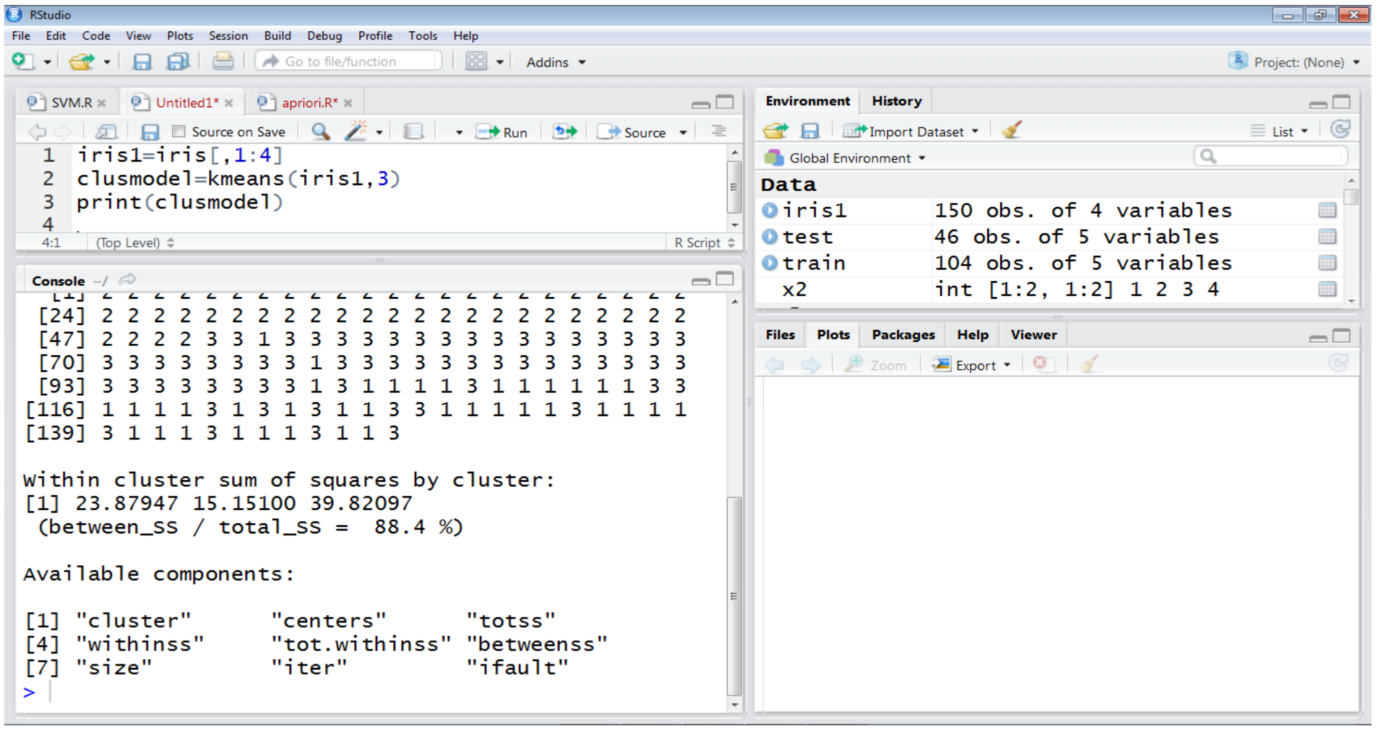
Example:Cluster the iris data set using K mean clustering algorithm.



```
iris1=iris[,1:4]
clusmodel=kmeans(iris1,3)
print(clusmodel)
```

iris1	150 obs. of 4 variables
test	46 obs. of 5 variables
train	104 obs. of 5 variables
x2	int [1:2, 1:2] 1 2 3 4

```
> source('~/.active-rstudio-document')
K-means clustering with 3 clusters of sizes 38, 50, 62

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

Clustering vector:
  [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [24] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [47] 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [70] 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
```

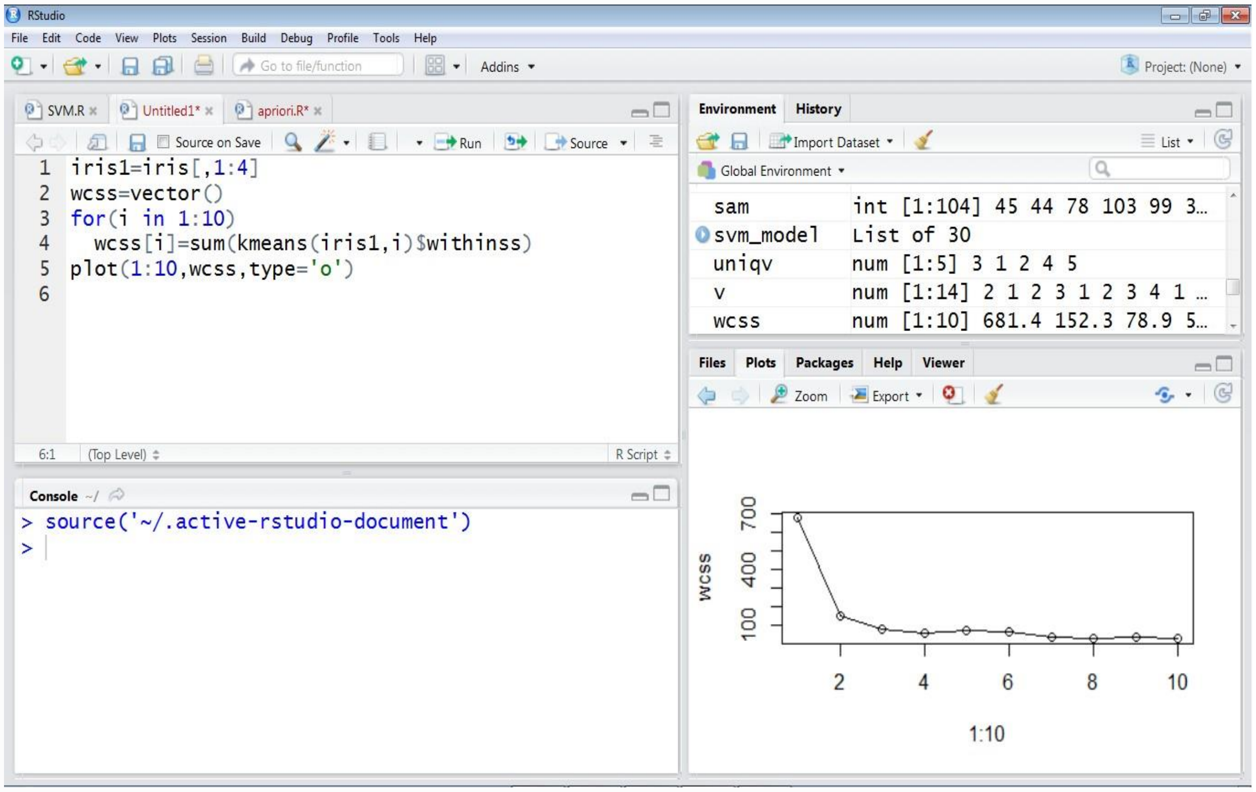


```
 [24] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [47] 2 2 2 2 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [70] 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [93] 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1 1 1 3 3 3
[116] 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1
[139] 3 1 1 1 3 1 1 1 3 1 1 3

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

Available components:

[1] "cluster"      "centers"      "totss"
[4] "withinss"     "tot.withinss" "betweenss"
[7] "size"         "iter"         "ifault"
>
```

Find Optimal Number of clusters using Elbow Method:

One method to validate the number of clusters is the *elbow method*. The
Idea of the elbow method is to run k-means clustering on the dataset for a range of values of *k* (say, *k* from 1 to 10 in the examples above), and for each value of *k* calculate the sum of squared errors (SSE).

Then, plot a line chart of the SSE for each value of *k*. If the line chart looks like an arm, then the "elbow" on the arm is the value of *k* that is the best. The idea is that we want a small SSE, but that the SSE tends to decrease toward 0 as we increase *k* (the SSE is 0 when *k* is equal to the number of data points in the dataset, because then each data point is its own cluster, and there is no error between it and the center of its cluster). So our goal is to choose a small value of *k* that still has a low SSE, and the elbow usually represents where we start to have diminishing returns by increasing *k*.



```
iris1=iris[,1:4]
wcss=vector()
for(i in 1:10)
  wcss[i]=sum(kmeans(iris1,i)$withinss)
plot(1:10,wcss,type='o')
```

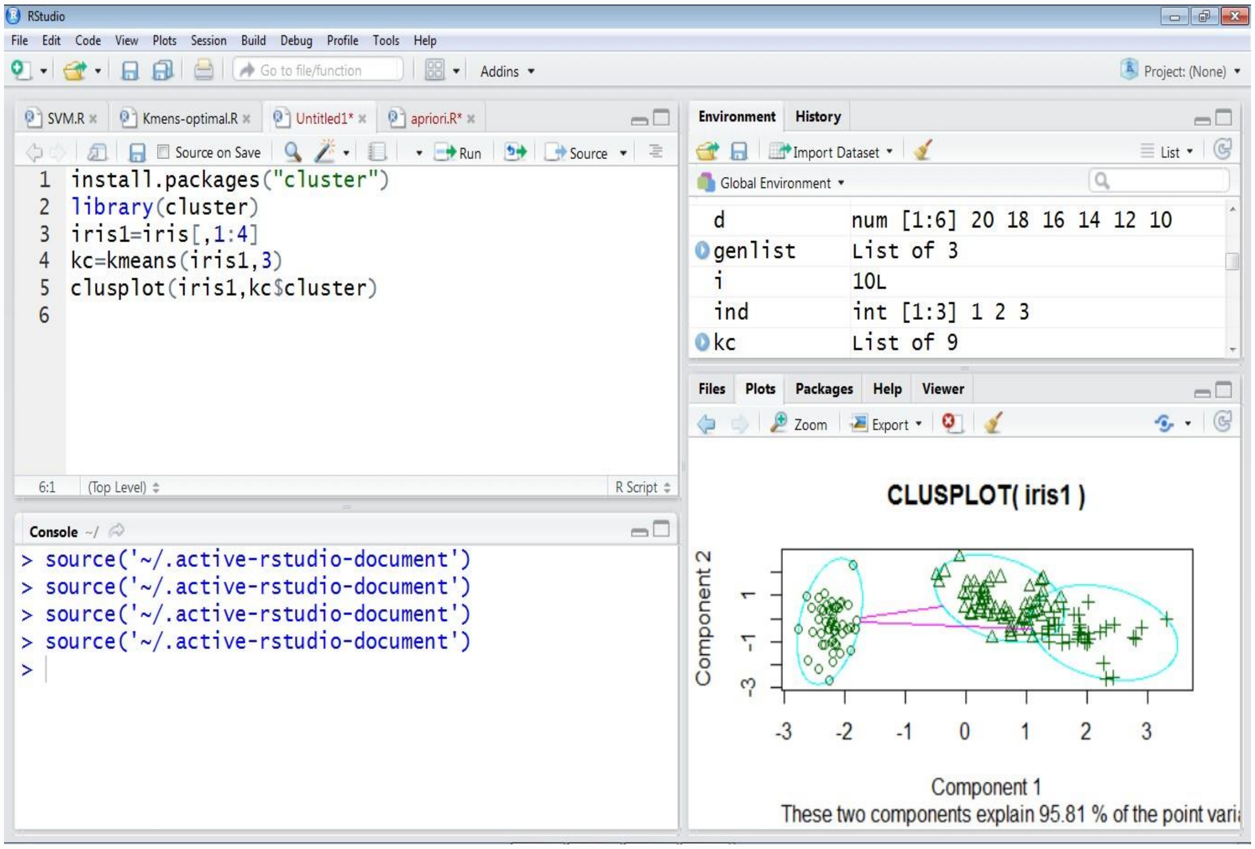
```
> source('~/.active-rstudio-document')
>
```

```
sam        int [1:104] 45 44 78 103 99 3...
svm_model  List of 30
uniqv      num [1:5] 3 1 2 4 5
v          num [1:14] 2 1 2 3 1 2 3 4 1 ...
wcss       num [1:10] 681.4 152.3 78.9 5...
```

Optimal number of clusters in the above example is 3.

Plotting cluster output:

Install the package cluster and use the function clus plot()to visualize clustering results.



```
install.packages("cluster")
library(cluster)
iris1=iris[,1:4]
kc=kmeans(iris1,3)
clusplot(iris1,kc$cluster)
```

```
d         num [1:6] 20 18 16 14 12 10
genlist   List of 3
i         10L
ind       int [1:3] 1 2 3
kc        List of 9
```

```
> source('~/.active-rstudio-document')
> source('~/.active-rstudio-document')
> source('~/.active-rstudio-document')
> source('~/.active-rstudio-document')
>
```

Association Rule Mining

Association rule learning is a popular and well researched method for discovering interesting relations between variables in large databases. It is the way of analyzing and presenting strong rules discovered in databases using different measures of interestingness. Based on the concept of strong rules, discover regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets. For example, the rule $\{\mathrm{onions}, \mathrm{potatoes}\} \Rightarrow \{\mathrm{burger}\}$ found in the sales data of a supermarket would indicate that if a customer buys onions and potatoes together, he or she is likely to also buy hamburger meat. Such information can be used as the basis for decisions about marketing activities such as, e.g., promotional pricing or product placements. In addition to the above example from market basket analysis association rules are employed today in many application areas including Web usage mining, intrusion detection and bioinformatics. As opposed to sequence mining, association rule learning typically does not consider the order of items either within a transaction or across transactions.

Apriori Algorithm

The most famous algorithm for association rule learning is Apriori. It was
proposed by Agrawal and Srikant in 1994. The input of the algorithm is a dataset of transactions where each transaction is a set of items. The output is a collection of association rules for which support and confidence are greater than some specified threshold. The name comes from the Latin phrase a priori (literally, "from what is before") because of one smart observation behind the algorithm: if the item set is infrequent, then we can be sure in advance that all its subsets are also infrequent.

You can implement Apriori with the following steps:

1. Count the support of all item sets of length 1, or calculate the frequency of every item in the dataset.
2. Drop the item sets that have support lower than the threshold.
3. Store all the remaining item sets.
4. Extend each stored item set by one element with all possible extensions. This step is known as candidate generation.

5. Calculate the support value of each candidate.
6. Drop all candidates below the threshold.
7. Drop all stored items from step 3 that have the same support as their extensions.
8. Add all the remaining candidates to storage.
9. Repeat steps 4 to step 8 until there are no more extensions with support greater than the threshold.

This is not a very efficient algorithm if you have a lot of data, but mobile applications are not recommended for use with big data anyway. This algorithm was influential in its time, and is also elegant and easy to understand today.

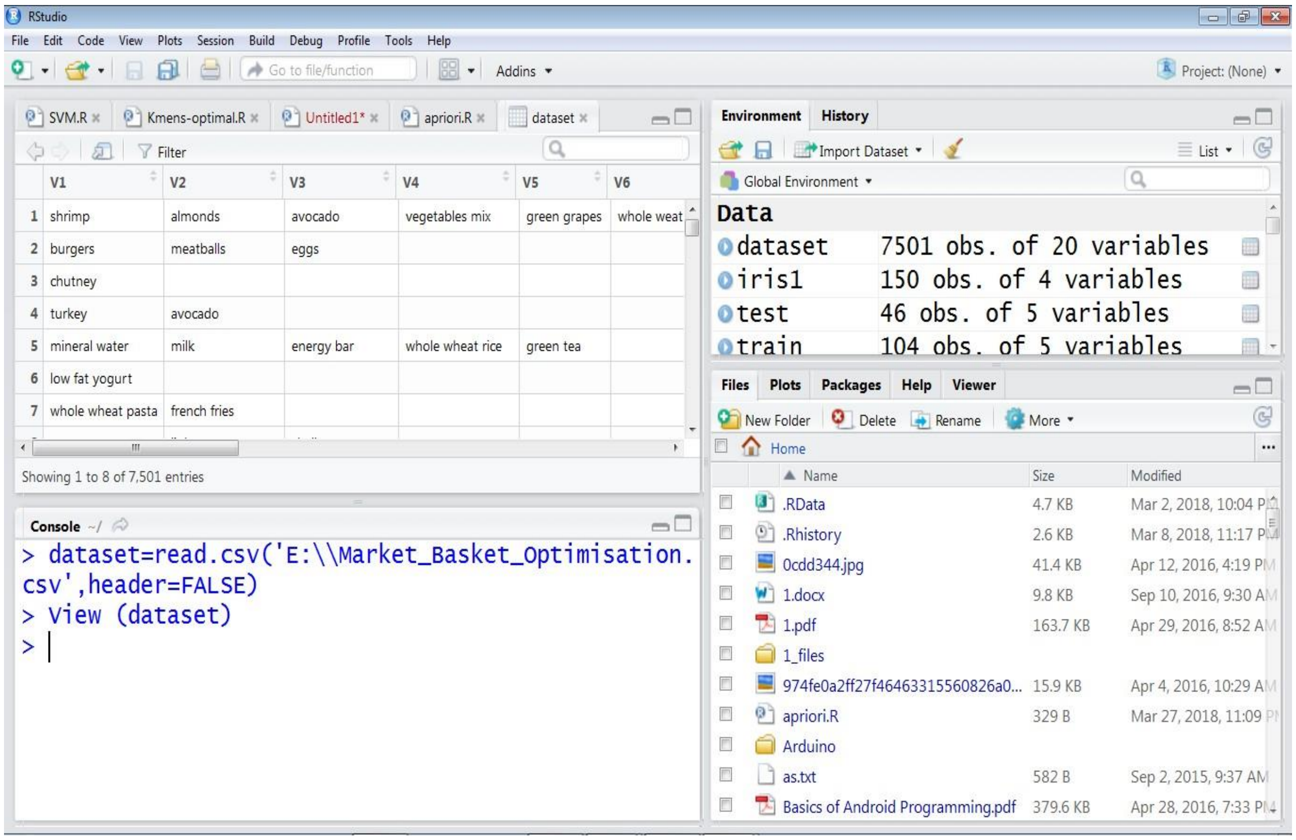
Implementation of Apriori Algorithm in R import the package and use the package arules:

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

Load the data set

Market _Basket_Optimaisation data set should be downloaded from the below website.

www.superdatascience/machinelearing



	V1	V2	V3	V4	V5	V6
1	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat
2	burgers	meatballs	eggs			
3	chutney					
4	turkey	avocado				
5	mineral water	milk	energy bar	whole wheat rice	green tea	
6	low fat yogurt					
7	whole wheat pasta	french fries				

Showing 1 to 8 of 7,501 entries

```
> dataset=read.csv('E:\\Market_Basket_Optimisation.
csv',header=FALSE)
> View (dataset)
> |
```

Data

```
dataset    7501 obs. of 20 variables
iris1      150 obs. of 4 variables
test       46 obs. of 5 variables
train      104 obs. of 5 variables
```

Name	Size	Modified
.RData	4.7 KB	Mar 2, 2018, 10:04 P
.Rhistory	2.6 KB	Mar 8, 2018, 11:17 P
0cdd344.jpg	41.4 KB	Apr 12, 2016, 4:19 P
1.docx	9.8 KB	Sep 10, 2016, 9:30 A
1.pdf	163.7 KB	Apr 29, 2016, 8:52 A
1_files		
974fe0a2ff27f46463315560826a0...	15.9 KB	Apr 4, 2016, 10:29 A
apriori.R	329 B	Mar 27, 2018, 11:09
Arduino		
as.txt	582 B	Sep 2, 2015, 9:37 AM
Basics of Android Programming.pdf	379.6 KB	Apr 28, 2016, 7:33 P

Convert the dataset into sparse Matrix:

```
>dataset=read.transactions('E:\\Market_Basket_Optimisation.csv',sep=",",rm.dup licates=TRUE)
  distribution of transactions with duplicates: 15
> dataset
  transactions in sparse format with 7501 transactions (rows) and 119 items (columns)
```

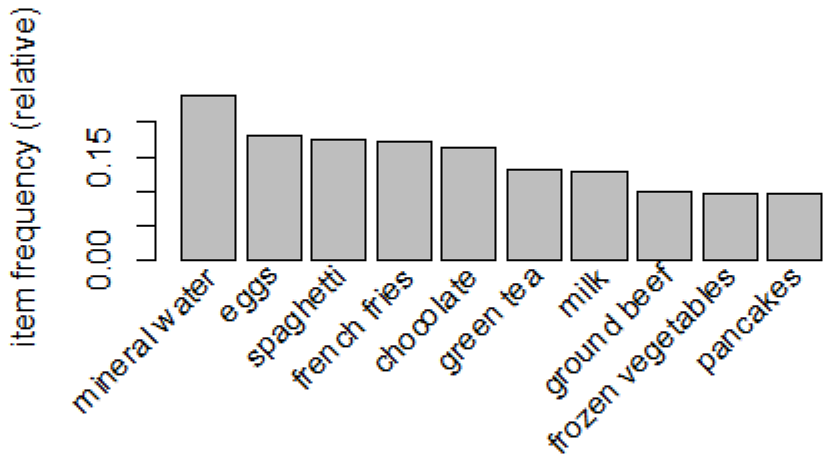
Get the Summary of the given data set:

```
> summary(dataset)
     transactions as itemMatrix in sparse format with 7501 rows
 (elements/itemsets/transactions) and119 columns (items) and a density of 0.03288973

    most frequent items:
      mineral water            eggs            spaghetti   french fries
              1788             1348              1306     1282
        chocolate             (Other)
              1229             22405
```

```
element (itemset/transaction) length distribution:

sizes
    1       2 3             4       5       6       7       8       9      10       1      12       1
                                                                                    1                3
    1       1358            8       6       4       3       3       2     139       1      67       4
    7       1044            1       6       9       9       2       5               0                0
    5                       6       7       3       1       4       9               2
    4
    1       15 16           1       1       2
    4                       8       9       0
    2       17 4            1       2       1
    2

   Min.   1st Qu.  Median    Mean   3rd Qu.    Max.
  1.000    2.000    3.000    3.914    5.000   20.000

includes extended item information - examples: labels
1                 almonds
2  antioxydant juice
3               asparagus
```

Plot ten items with Highest frequency :

> itemFrequencyPlot(dataset,topN=10)



Generate Association Rules with support =0.003 and confidence=0.8

>rules=apriori(data=dataset,parameter=list(support=0.003,confiden ce=0.8))

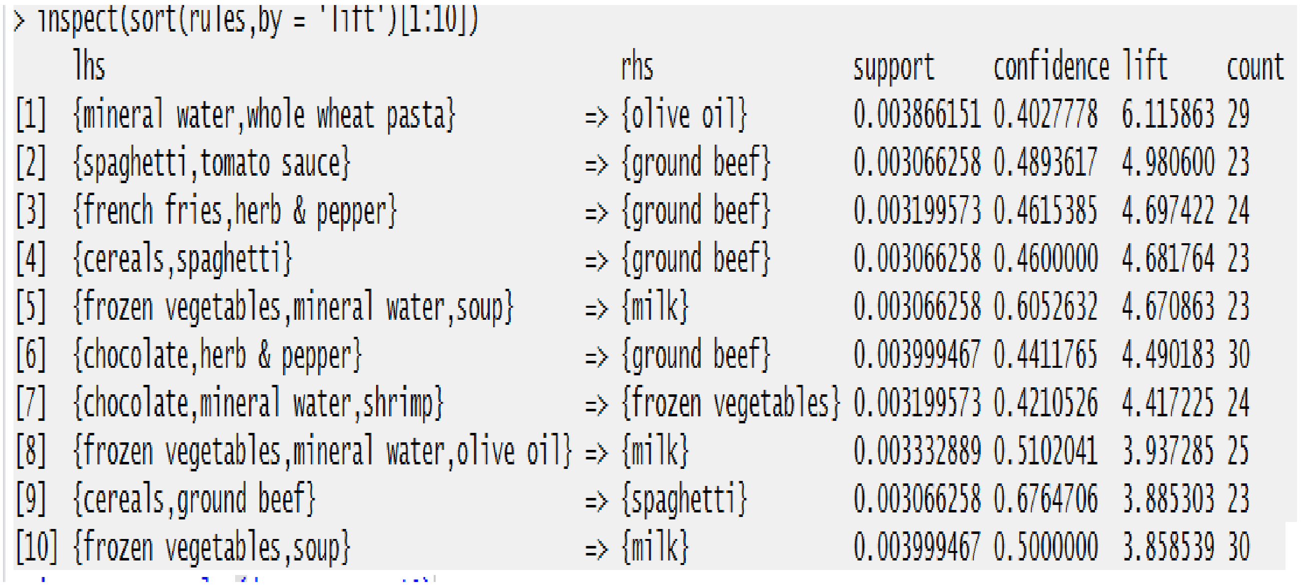
```
Parameter specification:
  confidence minval smax arem      aval originalSupport maxtime support
         0.8    0.1    1 none FALSE            TRUE       5   0.003
  minlen maxlen target      ext
       1     10  rules FALSE

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

```
Absolute minimum support count: 22

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s]. sorting and
recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.00s]. checking subsets
of size 1 2 3 4 5 done [0.00s].
writing ... [0 rule(s)] done [0.00s].
creating S4 object         ... done [0.00s].
```

Generate Association Rules with support =0.003 and confidence=0.4

```
>
rules=apriori(data=dataset,parameter=list(support=0.003,confiden ce=0.4))
Apriori

Parameter specification:
 confidence minval smax arem      aval originalSupport maxtime support
        0.4     0.1    1 none FALSE            TRUE       5   0.003
 minlen maxlen target       ext
      1     10  rules FALSE

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

Absolute minimum support count: 22

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[119 item(s), 7501 transaction(s)] done [0.00s]. sorting and
recoding items ... [115 item(s)] done [0.00s].
creating transaction tree ... done [0.00s]. checking subsets
of size 1 2 3 4 5 done [0.00s].
writing ... [281 rule(s)] done [0.00s].
creating S4 object         ... done [0.00s].
```

Print the top 10 rules



```
> inspect(sort(rules,by = 'lift')[1:10])
     lhs                                           rhs                 support     confidence lift     count
[1]  {mineral water,whole wheat pasta}          => {olive oil}         0.003866151 0.4027778  6.115863 29
[2]  {spaghetti,tomato sauce}                   => {ground beef}       0.003066258 0.4893617  4.980600 23
[3]  {french fries,herb & pepper}               => {ground beef}       0.003199573 0.4615385  4.697422 24
[4]  {cereals,spaghetti}                        => {ground beef}       0.003066258 0.4600000  4.681764 23
[5]  {frozen vegetables,mineral water,soup}     => {milk}              0.003066258 0.6052632  4.670863 23
[6]  {chocolate,herb & pepper}                  => {ground beef}       0.003999467 0.4411765  4.490183 30
[7]  {chocolate,mineral water,shrimp}           => {frozen vegetables} 0.003199573 0.4210526  4.417225 24
[8]  {frozen vegetables,mineral water,olive oil} => {milk}             0.003332889 0.5102041  3.937285 25
[9]  {cereals,ground beef}                      => {spaghetti}         0.003066258 0.6764706  3.885303 23
[10] {frozen vegetables,soup}                   => {milk}              0.003999467 0.5000000  3.858539 30
```

Outlier Analysis

Outlier:

An outlier is a data object that deviates significantly from the rest of the objects, as if it were generated by a different mechanism. Data objects that are not outliers as “normal” or expected data. Similarly, we may refer to outliers as “abnormal” data.

Outlier Analysis:

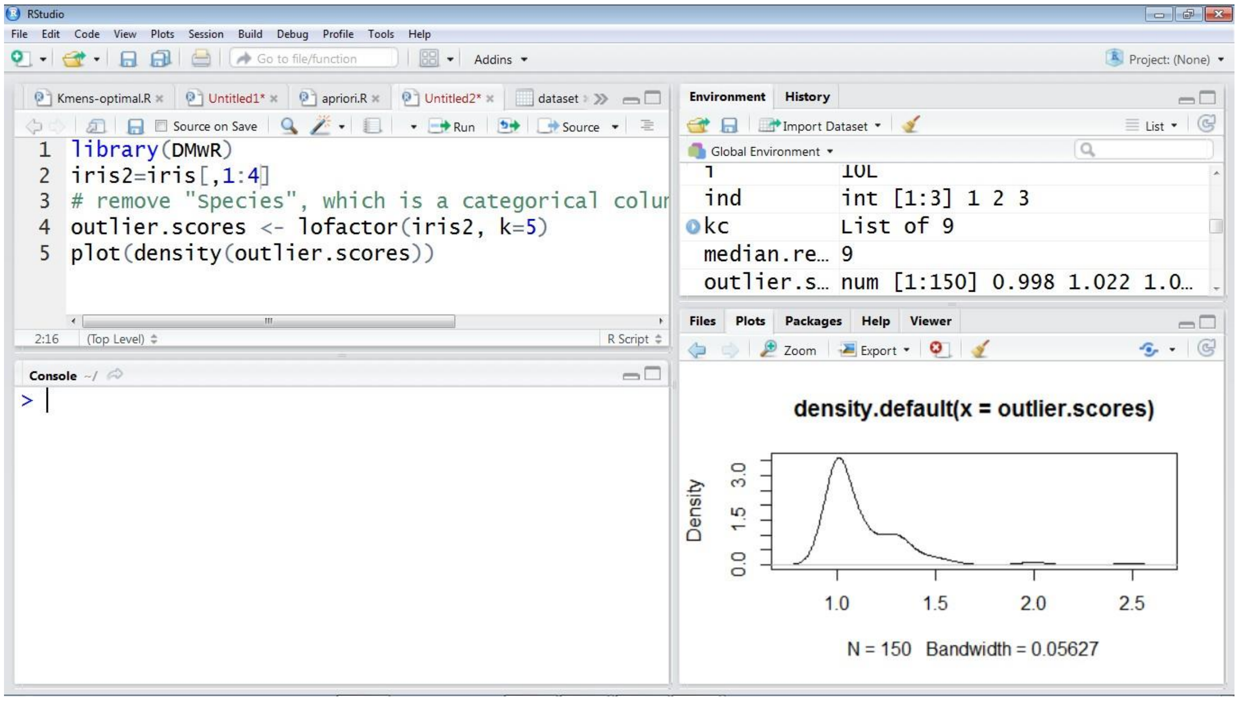
The outliers may be of particular interest, such as in the case of fraud detection, where outliers may indicate fraudulent activity. Thus, outlier detection and analysis is an interesting data mining task, referred to as outlier mining or outlier analysis.

Implementation of Outlier Analysis in R

The LOF algorithm

LOF (Local Outlier Factor) is an algorithm for identifying density-based local outliers [Breunig et al., 2000]. With LOF, the local density of a point is compared with that of its neighbors. If the former is signi.cantly lower than the latter (with an LOF value greater than one), the point is in a sparser region than its neighbors, which suggests it be an outlier.

Function lofactor(data, k) in packages DMwR and dprep calculates local outlier factors using the LOF algorithm, where k is the number of neighbors used in the calculation of the local outlier factors.

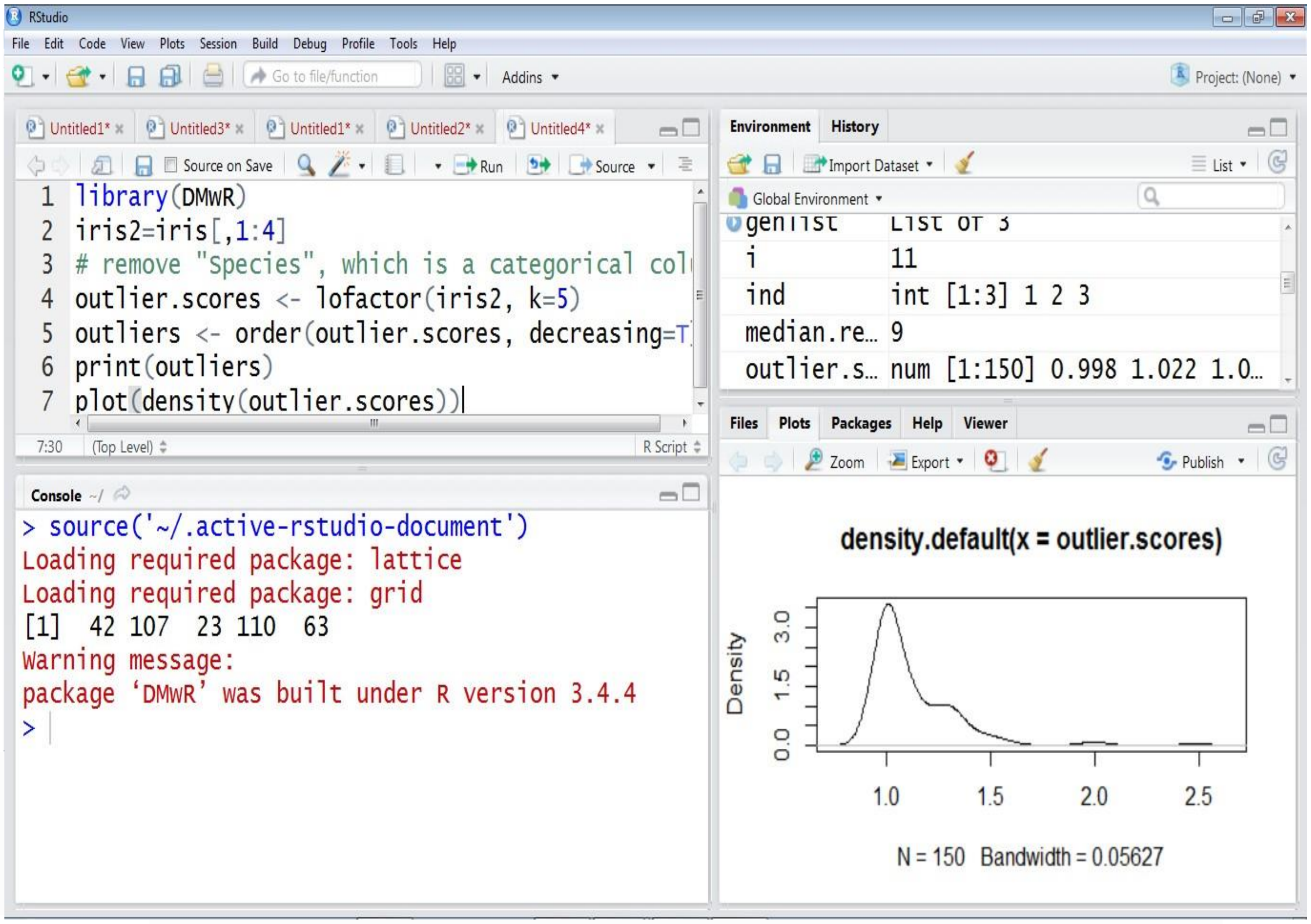


```
library(DMwR)
iris2=iris[,1:4]
# remove "Species", which is a categorical colu
outlier.scores <- lofactor(iris2, k=5)
plot(density(outlier.scores))
```

```
i            10L
ind          int [1:3] 1 2 3
kc           List of 9
median.re…   9
outlier.s…   num [1:150] 0.998 1.022 1.0…
```

```
>
```

Print the top 5 outliers:



```
library(DMwR)
iris2=iris[,1:4]
# remove "Species", which is a categorical col
outlier.scores <- lofactor(iris2, k=5)
outliers <- order(outlier.scores, decreasing=T
print(outliers)
plot(density(outlier.scores))
```

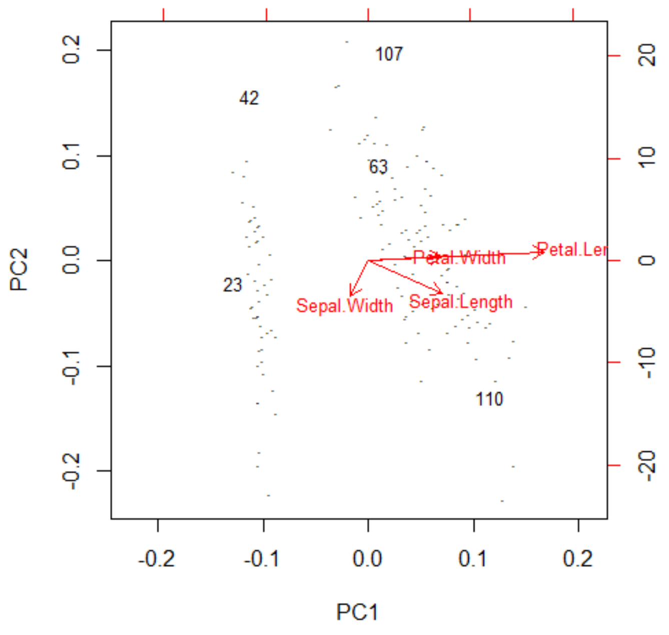
```
> source('~/.active-rstudio-document')
Loading required package: lattice
Loading required package: grid
[1]  42 107  23 110  63
Warning message:
package ‘DMwR’ was built under R version 3.4.4
>
```

genlist	List of 3
i	11
ind	int [1:3] 1 2 3
median.re...	9
outlier.s...	num [1:150] 0.998 1.022 1.0...

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled1* Untitled3* Untitled1* Untitled2* Untitled4*
Source on Save
Run
Source
7:30 (Top Level) R Script
Console ~/
Environment History
Import Dataset
List
Global Environment
Files Plots Packages Help Viewer
Zoom Export Publish
density.default(x = outlier.scores)
Density
0.0 1.5 3.0
1.0 1.5 2.0 2.5
N = 150 Bandwidth = 0.05627

Visualize Outliers with Plots

Next, we show outliers with a biplot of the first two principal components.



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF COMPUTING
DEPARTMENT OF INFORMATION TECHNOLOGY

Unit-V- R Programming – SCS1621

Overview of R Shiny - R Hadoop - Case Study - Hypothesis Generation, Importing Data set and Basic Data Exploration, Feature Engineering, Model Building.

Overview of R Shiny

Writing codes for plotting graphs in R time & again can get very tiring. Also, it is very difficult to create an interactive visualization for story narration using above packages. These problems can be resolved by dynamically creating interactive plots in R using Shiny with minimal effort.

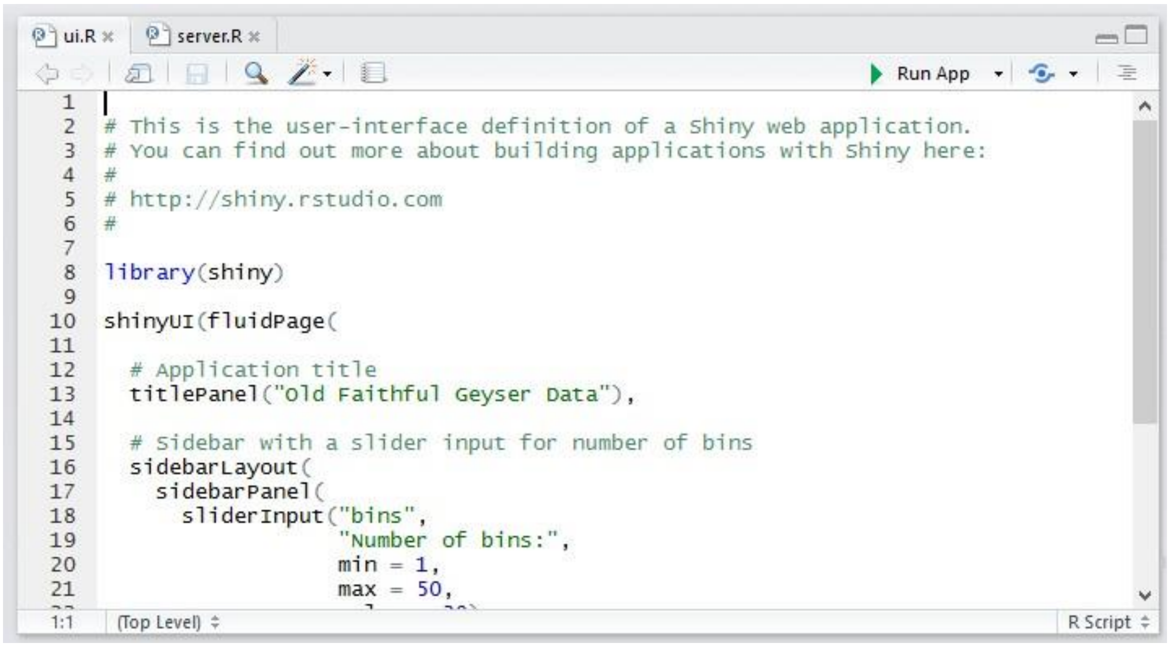
If you use R, chances are that you might have come across Shiny. It is an open package from RStudio, used to build interactive web pages with R. It provides a very powerful way to share your analysis in an interactive manner with the community.

Shiny is an open package from RStudio, which provides a web application framework to create interactive web applications (visualization) called "Shiny apps". The ease of working with Shiny has what popularized it among R users. These web applications seamlessly display R objects (like plots, tables etc.) and can also be made live to allow access to anyone.

Shiny provides automatic reactive binding between inputs and outputs which we will be discussing in the later parts of this article. It also provides extensive pre-built widgets which make it possible to build elegant and powerful applications with minimal effort.

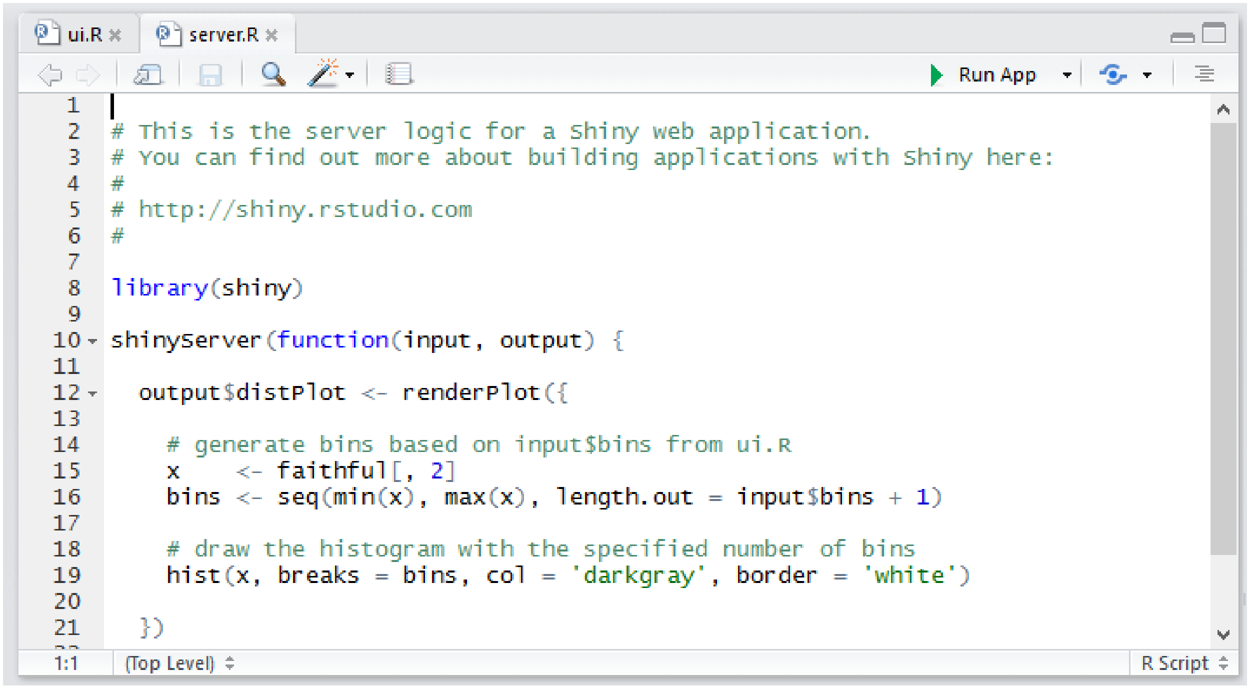
Components of R Shiny

1.UI.R: This file creates the user interface in a shiny application. It provides interactivity to the shiny app by taking the input from the user and dynamically displaying the generated output on the screen.



```

# This is the user-interface definition of a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyUI(fluidPage(

  # Application title
  titlePanel("Old Faithful Geyser Data"),

  # Sidebar with a slider input for number of bins
  sidebarLayout(
    sidebarPanel(
      sliderInput("bins",
                  "Number of bins:",
                  min = 1,
                  max = 50,
```

2. **Server.R:** This file contains the series of steps to convert the input given by user into the desired output to be displayed.

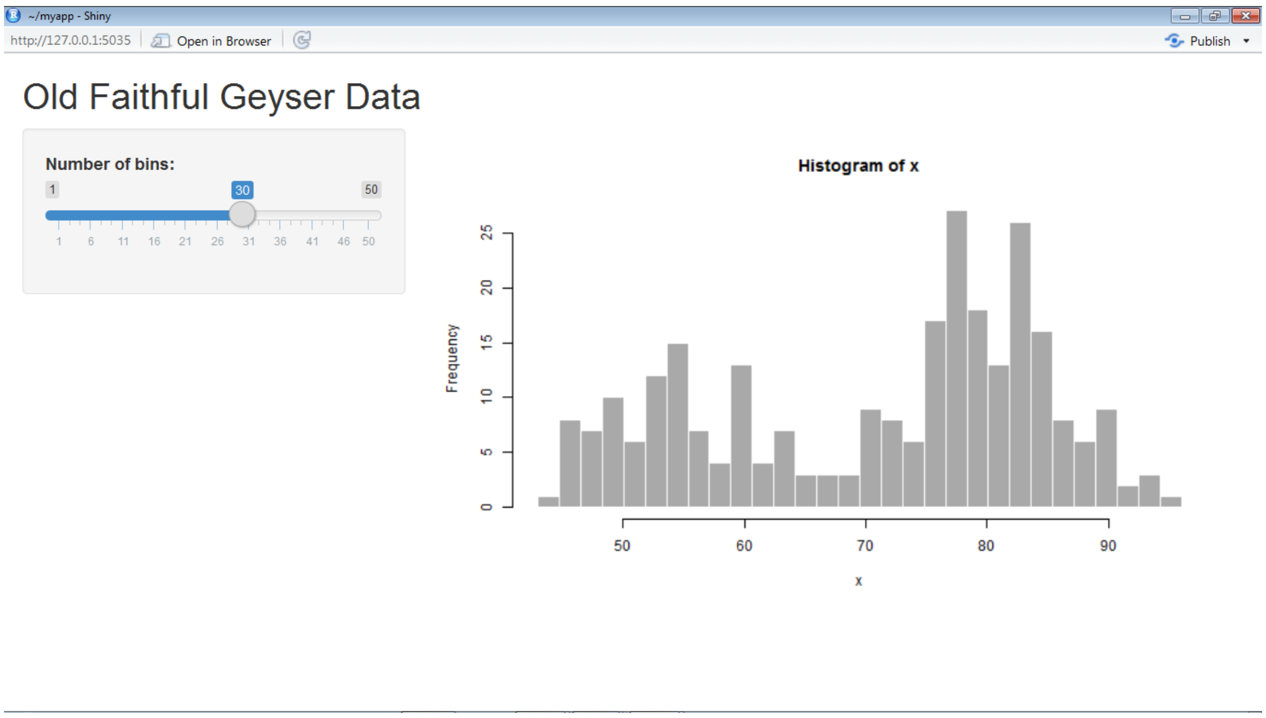


```

# This is the server logic for a Shiny web application.
# You can find out more about building applications with Shiny here:
#
# http://shiny.rstudio.com
#

library(shiny)

shinyServer(function(input, output) {

  output$distPlot <- renderPlot({

    # generate bins based on input$bins from ui.R
    x    <- faithful[, 2]
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

    # draw the histogram with the specified number of bins
    hist(x, breaks = bins, col = 'darkgray', border = 'white')

  })
```

Creation of Simple Creating simple RShiny Application

Writing "ui.R"

If you are creating a shiny application, the best way to ensure that the application interface runs smoothly on different devices with different screen resolutions is to create it using fluid page. This ensures that the page is laid out dynamically based on the resolution of each device.



The user interface can be broadly divided into three categories:

- **Title Panel:** The content in the title panel is displayed as metadata, as in top left corner of above image which generally provides name of the application and some other relevant information.

- **Sidebar Layout:** Sidebar layout takes input from the user in various forms like text input, checkbox input, radio button input, drop down input, etc. It is represented in dark background in left section of the above image.

- **Main Panel:** It is part of screen where the output(s) generated as a result of performing a set of operations on input(s) at the server.R is / are displayed.

Let’s understand UI.R and Server.R with an example:

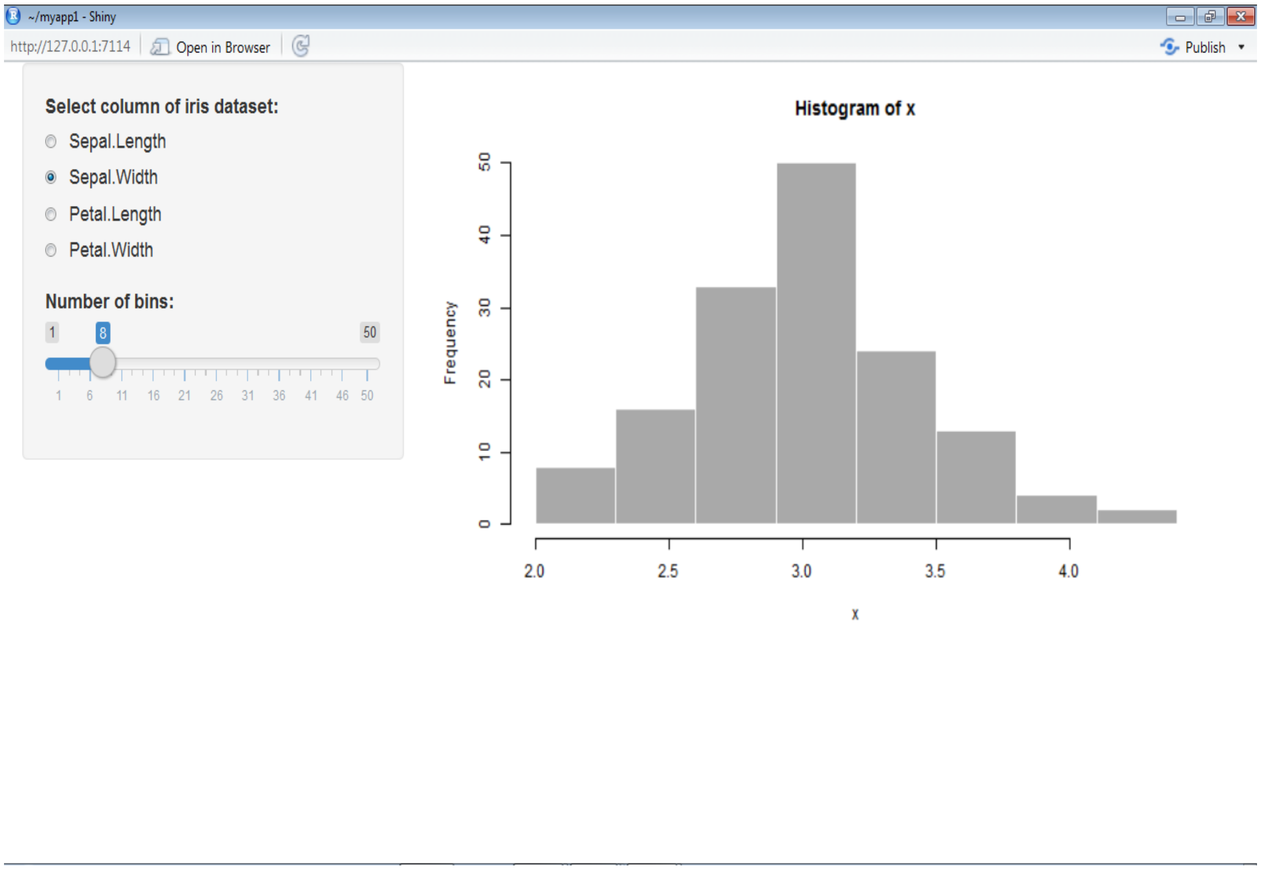
```
#UI.R

#loading shiny library

library(shiny)

shinyUI(fluidPage(

#fluid page for dynamically adapting to screens of different resolutions.

 titlePanel("Iris Dataset"),

 sidebarLayout(

 sidebarPanel  (

      #implementing radio buttons

      radioButtons("p", "Select column of iris dataset:",

                   list("Sepal.Length"='a', "Sepal.Width"='b', "Petal.Length"='c', "Petal.Width"='d')),

                  #slider input for bins of histogram

                  sliderInput("bins", "Number of bins:", min = 1, max = 50, value = 30)
  ),

 mainPanel(plotOutput("distPlot")))

  ))
```

Writing SERVER.R

This acts as the brain of web application. The server.R is written in the form of a function which maps input(s) to the output(s) by some set of logical operations. The inputs taken in ui.R file are accessed using $ operator (input$InputName). The outputs are also referred using the $ operator (output$OutputName). We will be discussing a few examples of server.R in the coming sections of the article for better understanding.

```
#SERVER.R
library(shiny)

#writing server function
shinyServer(function(input, output) {

#referring output distPlot in ui.r as output$distPlot
  output$distPlot <- renderPlot({

#referring input p in ui.r as input$p
    if(input$p=='a'){
      i<-1
    }

    if(input$p=='b'){
      i<-2
    }

    if(input$p=='c'){
      i<-3
    }

    if(input$p=='d'){
      i<-4
    }

    x    <- iris[, i]

#referring input bins in ui.r as input$bins
    bins <- seq(min(x), max(x), length.out = input$bins + 1)

#producing histogram as output
    hist(x, breaks = bins, col = 'darkgray', border = 'white')
  })

})
```

Output:

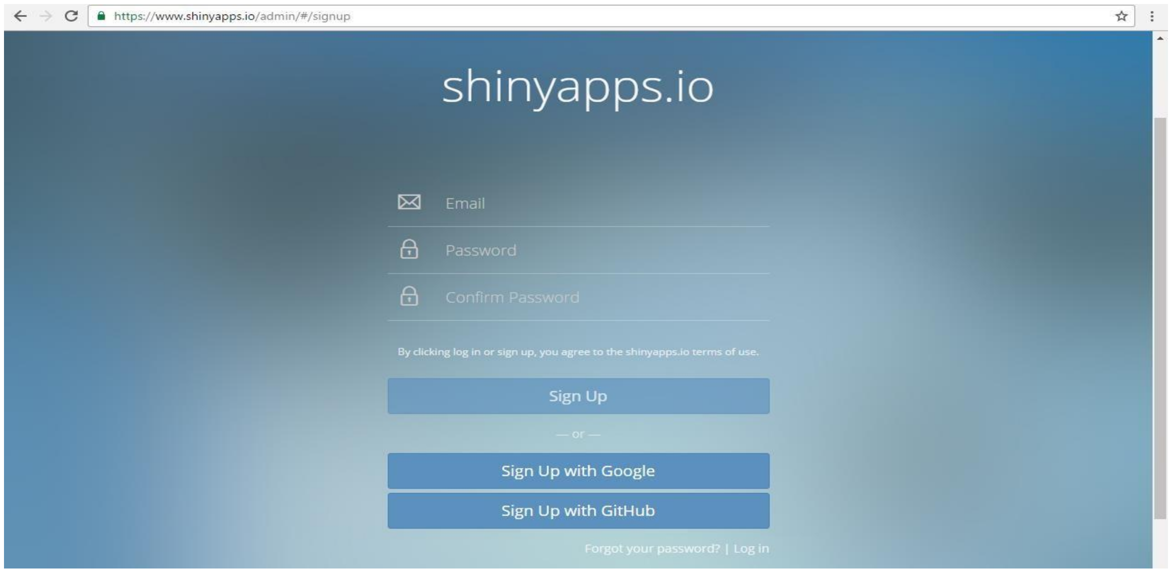


Deploying the Shiny app on the Web

The shiny apps which you have created can be accessed and used by anyone only if, it is deployed on the web. You can host your shiny application on “Shinyapps.io”. It provides free of cost platform as a service [PaaS] for deployment of shiny apps, with some restrictions though like only 25 hours of usage in a month, limited memory space, etc. You can also use your own server for deploying shiny apps.

Steps for using shiny cloud:

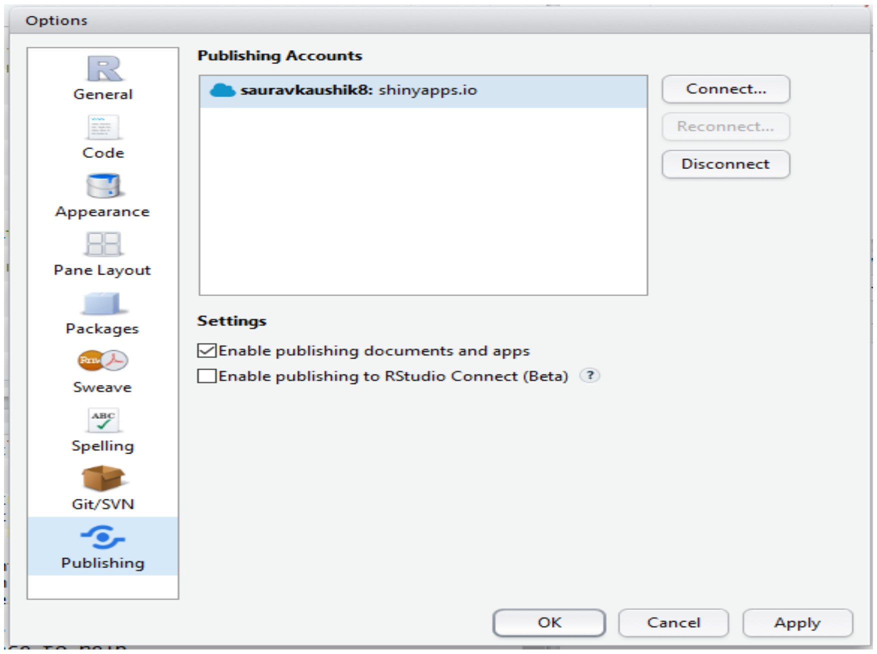
Step 1: Sign up on shinyapps.io


https://www.shinyapps.io/admin/#/signup
shinyapps.io
Email
Password
Confirm Password
By clicking log in or sign up, you agree to the shinyapps.io terms of use.
Sign Up
— or —
Sign Up with Google
Sign Up with GitHub
Forgot your password? | Log in

Step 2: Go to Tools in R Studio.

Step 3: Open global options.

Step 4: Open publishing tab



sauravkaushik8: shinyapps.io

Settings

[✓] Enable publishing documents and apps

[ ] Enable publishing to RStudio Connect (Beta)

Step 5: Manage your account(s).

Shiny App for displaying summary of the given data set:

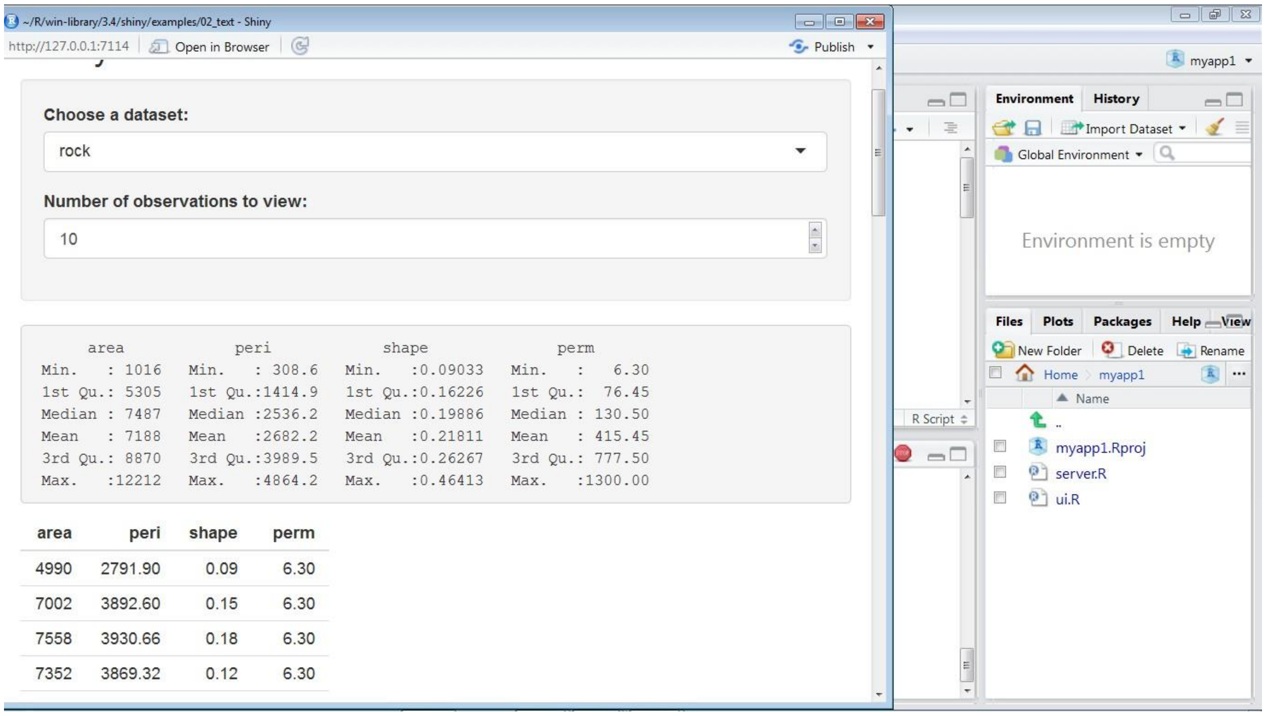
Server.R

```
library(shiny)
library(datasets)

# Define server logic required to summarize and view the selected dataset
shinyServer(function(input, output) {

  # Return the requested dataset
  datasetInput <- reactive({


    switch(input$dataset,
           "rock" = rock,
           "pressure" = pressure,
           "cars" = cars)
  })

  # Generate a summary of the dataset
  output$summary <- renderPrint({
  dataset <- datasetInput()
  summary(dataset)
  })

  # Show the first "n" observations
  output$view <- renderTable({
  head(datasetInput(), n = input$obs)
  })
})
```

ui.R

```
library(shiny)

# Define UI for dataset viewer application
shinyUI(fluidPage(

  # Application title
  titlePanel("Shiny Text"),
 # Sidebar with controls to select a dataset and specify the number
 # of observations to view
 sidebarLayout(
  sidebarPanel(
    selectInput("dataset", "Choose a dataset:",
            choices = c("rock", "pressure", "cars")),

    numericInput("obs", "Number of observations to view:", 10)
  ),

  # Show a summary of the dataset and an HTML table with the requested
  # number of observations
  mainPanel(
    verbatimTextOutput("summary"),

    tableOutput("view")
  )

 )
))
```

Output:



Choose a dataset:

rock

Number of observations to view:

10

```
     area            peri             shape              perm
Min.   : 1016   Min.   : 308.6   Min.   :0.09033   Min.   :   6.30
1st Qu.: 5305   1st Qu.:1414.9   1st Qu.:0.16226   1st Qu.:  76.45
Median : 7487   Median :2536.2   Median :0.19886   Median : 130.50
Mean   : 7188   Mean   :2682.2   Mean   :0.21811   Mean   : 415.45
3rd Qu.: 8870   3rd Qu.:3989.5   3rd Qu.:0.26267   3rd Qu.: 777.50
Max.   :12212   Max.   :4864.2   Max.   :0.46413   Max.   :1300.00
```

area	peri	shape	perm
4990	2791.90	0.09	6.30
7002	3892.60	0.15	6.30
7558	3930.66	0.18	6.30
7352	3869.32	0.12	6.30

R HADOOP

R is an amazing data science programming tool to run statistical data analysis on models and translating the results of analysis into colourful graphics. There is no doubt that R is the most preferred programming tool for statisticians, data scientists, data analysts and data architects but it falls short when working with large datasets. One major drawback with R programming language is that all objects are loaded into the main memory of a single machine. Large datasets of size petabytes cannot be loaded into the RAM memory; this is when Hadoop integrated with R language, is an ideal solution. To adapt to the in-memory, single machine limitation of R programming language, data scientists have to limit their data analysis to a sample of data from the large data set. This limitation of R programming language comes as a major hindrance when

dealing with big data. Since, R is not very scalable, the core R engine can process only limited amount of data.To the contrary, distributed processing frameworks like Hadoop are scalable for complex operations and tasks on large datasets (petabyte range) but do not have strong statistical analytical capabilities. As Hadoop is a popular framework for big data processing, integrating R with Hadoop is the next logical step. Using R on Hadoop will provide highly scalable data analytics platform which can be scaled depending on the size of the dataset. Integrating Hadoop with R lets data scientists run R in parallel on large dataset as none of the data science libraries in R language will work on a dataset that is larger than its memory. Big Data analytics with R and Hadoop competes with the cost value return offered by commodity hardware cluster for vertical scaling.

Methods of Integrating R and Hadoop Together

Data analysts or data scientists working with Hadoop might have R packages or R scripts that they use for data processing. To use these R scripts or R packages with Hadoop, they need to rewrite these R scripts in Java programming language or any other language that implements Hadoop MapReduce. This is a burdensome process and could lead to unwanted errors. To integrate Hadoop with R programming language, we need to use a software that already is written for R language with the data being stored on the distributed storage Hadoop. There are many solutions for using R language to perform large computations but all these solutions require that the data be loaded into the memory before it is distributed to the computing nodes. This is not an ideal solution for large datasets. Here are some commonly used methods to integrate Hadoop with R to make the best use of the analytical capabilities of R for large datasets-

1) RHADOOP –Install R on Workstations and Connect to Data in Hadoop

The most commonly used open source analytics solution to integrate R programming language with Hadoop is RHadoop. RHadoop developed by Revolution Analytics lets users directly ingest data from HBase database subsystems and HDFS file systems. Rhadoop package is the 'go-to' solution for using R on Hadoop because of its simplicity and cost advantage. Rhadoop is a

collection of 5 different packages which allows Hadoop users to manage and analyse data using R programming language. RHadoop package is compatible with open source Hadoop and as well with popular Hadoop distributions- Cloudera, Hortonworks and MapR.

rhbase – rhbase package provides database management functions for HBase within R using Thrift server. This package needs to be installed on the node that will run R client. Using rhbase, data engineers and data scientists can read, write and modify data stored in HBase tables from within R.

rhdfs –rhdfs package provides R programmers with connectivity to the Hadoop distributed file system so that they read, write or modify the data stored in Hadoop HDFS.

plyrmr – This package supports data manipulation operations on large datasets managed by Hadoop. Plyrmr (plyr for MapReduce) provides data manipulation operations present in popular packages like reshape2 and plyr. This package depends on Hadoop MapReduce to perform operations but abstracts most of the MapReduce details.

ravro –This package lets users read and write Avro files from local and HDFS file systems.

rmr2 (Execute R inside Hadoop MapReduce) – Using this package, R programmers can perform statistical analysis on the data stored in a Hadoop cluster. Using rmr2 might be a cumbersome process to integrate R with Hadoop but many R programmers find using rmr2 much easier than depending on Java based Hadoop mappers and reducers. rmr2 might be a little tedious but it eliminates data movement and helps parallelize computation to handle large datasets.

2) RHIPE – Execute R inside Hadoop Map Reduce

RHIPE (**"R and Hadoop Integrated Programming Environment")** is an R library that allows users to run Hadoop MapReduce jobs within R programming language. R programmers just have to write R map and R reduce functions and the RHIPE library will transfer them and invoke the corresponding Hadoop Map and Hadoop Reduce tasks. RHIPE uses a protocol buffer encoding scheme to transfer the map and reduce inputs. The advantage of using RHIPE over other parallel R packages is, that it integrates well with Hadoop and provides a data distribution scheme

using HDFS across a cluster of machines - which provides fault tolerance and optimizes processor usage.

3) R and Hadoop Streaming

Hadoop Streaming API allows users to run Hadoop MapReduce jobs with any executable script that reads data from standard input and writes data to standard output as mapper or reducer. Thus, Hadoop Streaming API can be used along R programming scripts in the map or reduce phases. This method to integrate R, Hadoop does not require any client side integration because streaming jobs are launched through Hadoop command line. MapReduce jobs submitted undergo data transformation through UNIX standard streams and serialization to ensure Java complaint input to Hadoop, irrespective of the language of the input script provided by the programmer.

4) RHIVE –Install R on Workstations and Connect to Data in Hadoop

If you want your Hive queries to be launched from R interface then RHIVE is the go-to package with functions for retrieving metadata like database names, column names, and table names from Apache Hive. RHIVE provides rich statistical libraries and algorithms available in R programming language to the data stored in Hadoop by extending HiveQL with R language functions. RHIVE functions allow users to apply R statistical learning models to the data stored in Hadoop cluster that has been catalogued using Apache Hive. The advantage of using RHIVE for Hadoop R integration is that it parallelizes operations and avoids data movement because data operations are pushed down into Hadoop.

5) ORCH – Oracle Connector for Hadoop

ORCH can be used on non-oracle Hadoop clusters or on any other Oracle big appliance. Mappers and Reducers are written in R programming language and MapReduce jobs are executed from the R environments through a high level interface. With ORCH for R Hadoop integration, R programmers do not have to learn a new programming language like Java for getting into the details of Hadoop environment like Hadoop Cluster hardware or software. ORCH connector also allows users to test the ability of Map Reduce programs locally, through the same function call, much before they are deployed to the Hadoop cluster.

The number of open source options for performing big data analytics with R and Hadoop is continuously expanding but for simple Hadoop MapReduce jobs, R and Hadoop Streaming still proves to be the best solution. The combination of R and Hadoop together is a must have toolkit for professionals working with big data to create fast, predictive analytics combined with performance, scalability and flexibility you need.

Most Hadoop users claim that the advantage of using R programming language is its exhaustive list of data science libraries for statistics and data visualization. However, the data science libraries in R language are non-distributed in nature which makes data retrieval a time consuming affair. However, this is an in-built limitation of R programming language, but if we just ignore it, then R and Hadoop together can make big data analytics an ecstasy!

Case Study

Data set used: https://archive.ics.uci.edu/ml/machine-learning- databases/00275/

Step 1. Hypothesis Generation

Before exploring the data to understand the relationship between variables, I'd recommend you to focus on hypothesis generation first. Now, this might sound counter-intuitive for solving a data science problem. Before exploring data, think about the business problem, gain the domain knowledge.

How does it help? This practice usually helps you form better features later on, which are not biased by the data available in the dataset. At this stage, you are expected to posses structured thinking i.e. a thinking process which takes into consideration all the possible aspects of a particular problem.

Here are some of the hypothesis which I thought could influence the demand of bikes:

Hourly trend: There must be high demand during office timings. Early morning and late evening can have different trend (cyclist) and low demand during 10:00 pm to 4:00 am.

Daily Trend: Registered users demand more bike on weekdays as compared to weekend or holiday.

Rain: The demand of bikes will be lower on a rainy day as compared to a sunny day. Similarly, higher humidity will cause to lower the demand and vice versa.

Temperature: In India, temperature has negative correlation with bike demand. But, after looking at Washington's temperature graph, I presume it may have positive correlation.

Pollution: If the pollution level in a city starts soaring, people may start using Bike (it may be influenced by government / company policies or increased awareness).

Time: Total demand should have higher contribution of registered user as compared to casual because registered user base would increase over time.

Traffic: It can be positively correlated with Bike demand. Higher traffic may force people to use bike as compared to other road transport medium like car, taxi etc

Step 2. Understanding the Data Set

The dataset shows hourly rental data for two years (2011 and 2012). The training data set is for the first 19 days of each month. The test dataset is from 20th day to month's end. We are required to predict the total count of bikes rented during each hour covered by the test set.

In the training data set, they have separately given bike demand by registered, casual users and sum of both is given as count.

Training data set has 12 variables (see below) and Test has 9 (excluding registered, casual and count).

Independent Variables

datetime: date and hour in "mm/dd/yyyy hh:mm" format

season: Four categories-> 1 = spring, 2 = summer, 3 = fall, 4 = winter

holiday: whether the day is a holiday or not (1/0)

workingday: whether the day is neither a weekend nor holiday (1/0)

weather: Four Categories of weather

1-> Clear, Few clouds, Partly cloudy, Partly cloudy

2-> Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist

3-> Light Snow and Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds

4-> Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog

temp: hourly temperature in Celsius

atemp: "feels like" temperature in Celsius

humidity: relative humidity

windspeed: wind speed

Dependent Variables

registered: number of registered user

casual: number of non-registered user

count: number of total rentals (registered + casual)

3. Importing Data set and Basic Data Exploration

For this solution, R (R Studio 0.99.442) in Windows Environment has been used.

Below are the steps to import and perform data exploration.

1. **Import Train and Test Data Set**

```
setwd("E:/kaggle data/bike sharing")

train=read.csv("train_bike.csv")

test=read.csv("test_bike.csv")
```

2. **Combine both Train and Test Data set (to understand the distribution of independent variable together).**

```
test$registered=0

test$casual=0

test$count=0

data=rbind(train,test)
```

Before combing test and train data set, I have made the structure similar for both.

3. **Variable Type Identification**

```
str(data)
```

```
'data.frame': 17379 obs. of 12 variables:

$ datetime : Factor w/ 17379 levels "2011-01-01 00:00:00",..: 1 2 3 4 5 6 7 8 9 10
...

$ season    : int  1 1 1 1 1 1 1 1 1 1 ...

$ holiday   : int  0 0 0 0 0 0 0 0 0 0 ...

$ workingday: int 0 0 0 0 0 0 0 0 0 0 ...

$ weather : int 1 1 1 1 1 2 1 1 1 1 ...

$ temp      : num 9.84 9.02 9.02 9.84 9.84 ...

$ atemp     : num 14.4 13.6 13.6 14.4 14.4 ...

$ humidity : int 81 80 80 75 75 75 80 86 75 76 ...
$ windspeed : num 0 0 0 0 0 ...
$ casual    : num 3 8 5 3 0 0 2 1 1 8 ...
$ registered: num 13 32 27 10 1 1 0 2 7 6 ...
$ count     : num 16 40 32 13 1 1 2 3 8 14 ...
```

Find missing values in data set if any.

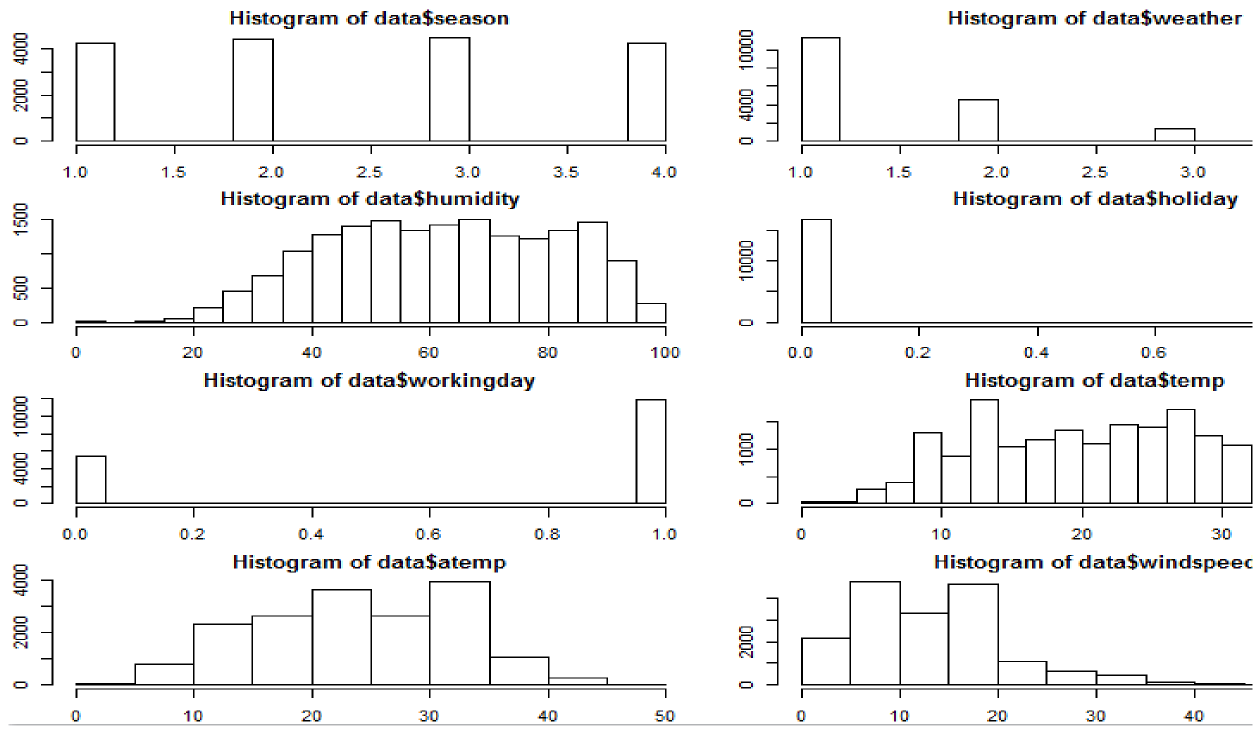
```
table(is.na(data))
```

```
FALSE
208548
```

From Above you can see that it has returned no missing values in the data frame.

4. **Understand the distribution of numerical variables and generate a frequency table for numeric variables. plot a histogram for each numerical variables and analyze the distribution.**

```
par(mfrow=c(4,2))
par(mar = rep(2, 4))
hist(data$season)
hist(data$weather)
hist(data$humidity)
hist(data$holiday)
hist(data$workingday)
hist(data$temp)
hist(data$atemp)
hist(data$windspeed)
```



Few inferences can be drawn by looking at the these histograms:

- Season has four categories of almost equal distribution
- Weather 1 has higher contribution i.e. mostly clear weather.

prop.table(table(data$weather))

```
1    2    3    4

0.66 0.26 0.08 0.00
```

As expected, mostly working days and variable holiday is also showing a similar inference. You can use the code above to look at the distribution in detail. Here you can generate a variable for weekday using holiday and working day. Incase, if both have zero values, then it must be a working day.Variables temp, atemp, humidity and windspeed looks naturally distributed.

Convert discrete variables into factor (season, weather, holiday, workingday)

```
data$season=as.factor(data$season)

data$weather=as.factor(data$weather)

data$holiday=as.factor(data$holiday)

data$workingday=as.factor(data$workingday)
```

4. Hypothesis Testing (using multivariate analysis)

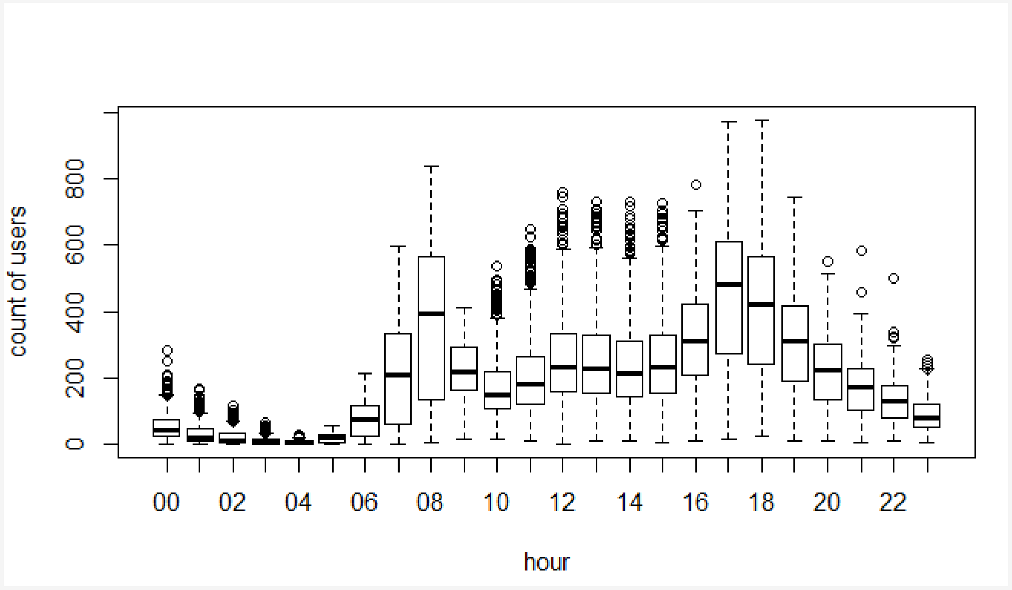
Till now, we have got a fair understanding of the data set. Now, let's test the hypothesis which we had generated earlier. Here I have added some additional hypothesis from the dataset. Let's test them one by one:

- **Hourly trend**: We don't have the variable 'hour' with us right now. But we can extract it using the datetime column.

```
data$hour=substr(data$datetime,12,13)

data$hour=as.factor(data$hour)
```

Let's plot the hourly trend of count over hours and check if our hypothesis is correct or not. We will separate train and test data set from combined one.

```
train=data[as.integer(substr(data$datetime,9,10))<20,]

test=data[as.integer(substr(data$datetime,9,10))>19,]

boxplot(train$count~train$hour,xlab="hour", ylab="count of users")
```



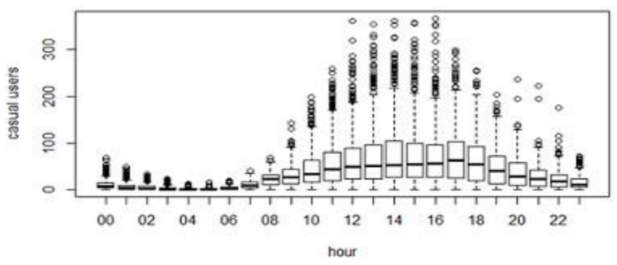
Above, you can see the trend of bike demand over hours. Quickly, I'll segregate the bike demand in three categories:

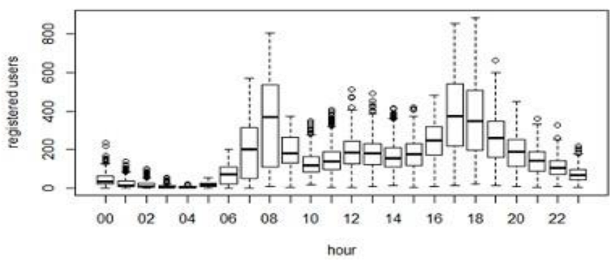
High : 7-9 and 17-19 hours

Average : 10-16 hours

Low : 0-6 and 20-24 hours

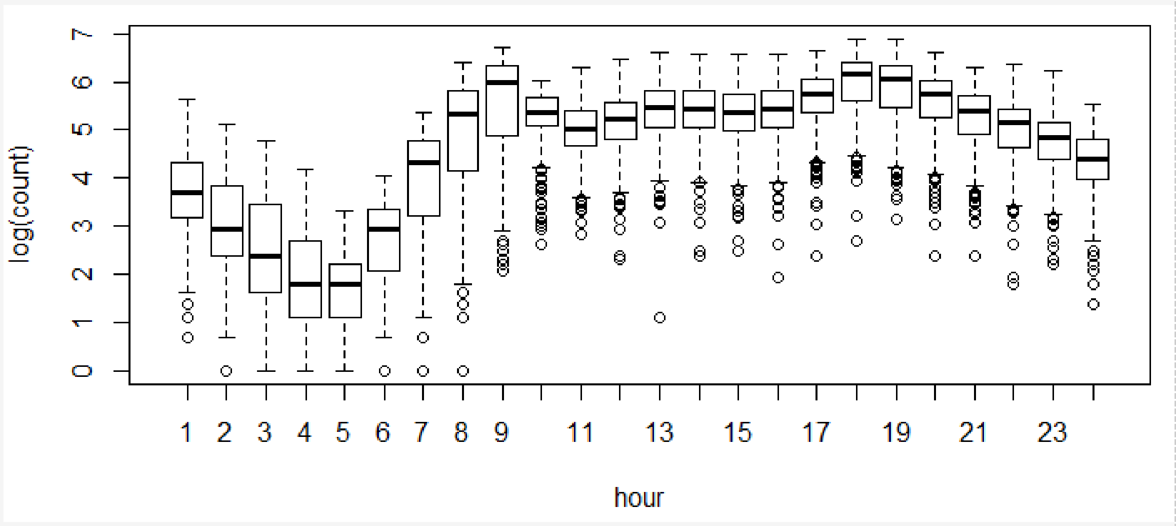
Here we have analyzed the distribution of total bike demand. Let’s look at the distribution of registered and casual users separately.





Above you can see that registered users have similar trend as count. Whereas, casual users have different trend. Thus, we can say that ‘hour’ is significant variable and our hypothesis is ‘true’.

You might have noticed that there are a lot of outliers while plotting the count of registered and casual users. These values are not generated due to error, so we consider them as natural outliers. They might be a result of groups of people taking up cycling (who are not registered). To treat such outliers, we will use logarithm transformation. Let’s look at the similar plot after log transformation.

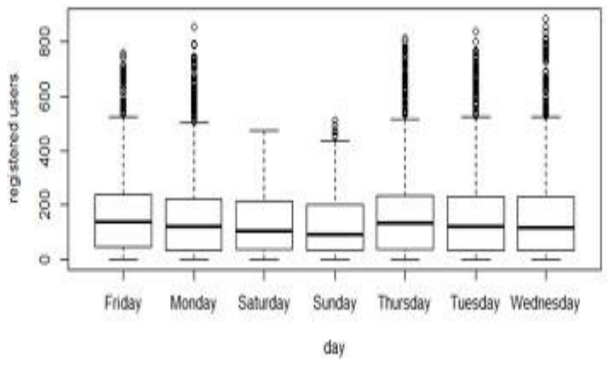


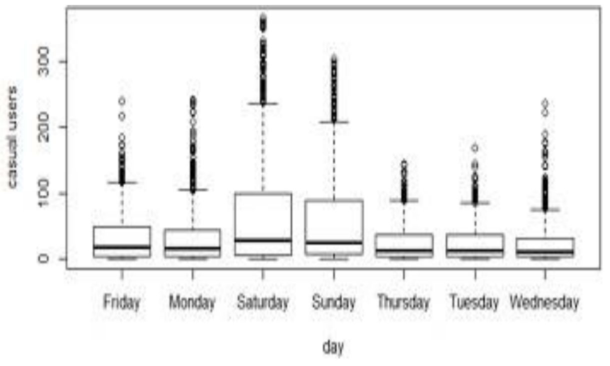
```
boxplot(log(train$count)~train$hour,xlab="hour",ylab="log(count)")
```

Daily Trend: Like Hour, we will generate a variable for day from datetime variable and after that we'll plot it.

```
date=substr(data$datetime,1,10)

days<-weekdays(as.Date(date))

data$day=days
```

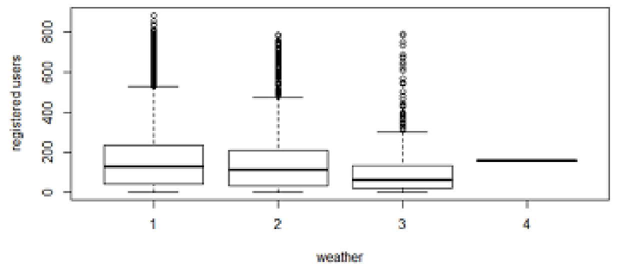
Plot shows registered and casual users' demand over days.

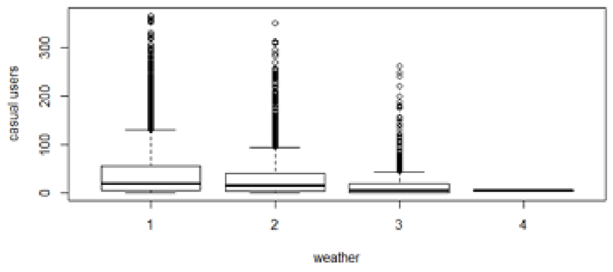




While looking at the plot, I can say that the demand of causal users increases over weekend.

Rain: We don't have the 'rain' variable with us but have 'weather' which is sufficient to test our hypothesis. As per variable description, weather 3 represents light rain and weather 4 represents

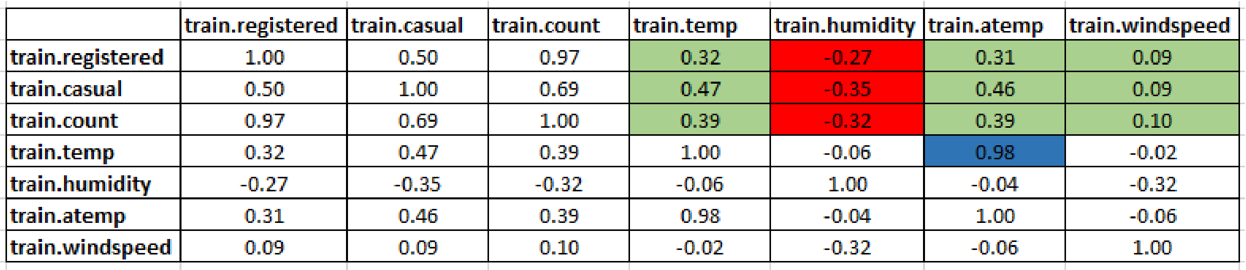




heavy rain. Take a look at the plot: It is clearly satisfying our hypothesis.

Temperature, Windspeed and Humidity: These are continuous variables so we can look at the correlation factor to validate hypothesis.

sub=data.frame(train$registered,train$casual,train$count,train$temp,train$humidity,train$atemp,train$windspeed)
cor(sub)



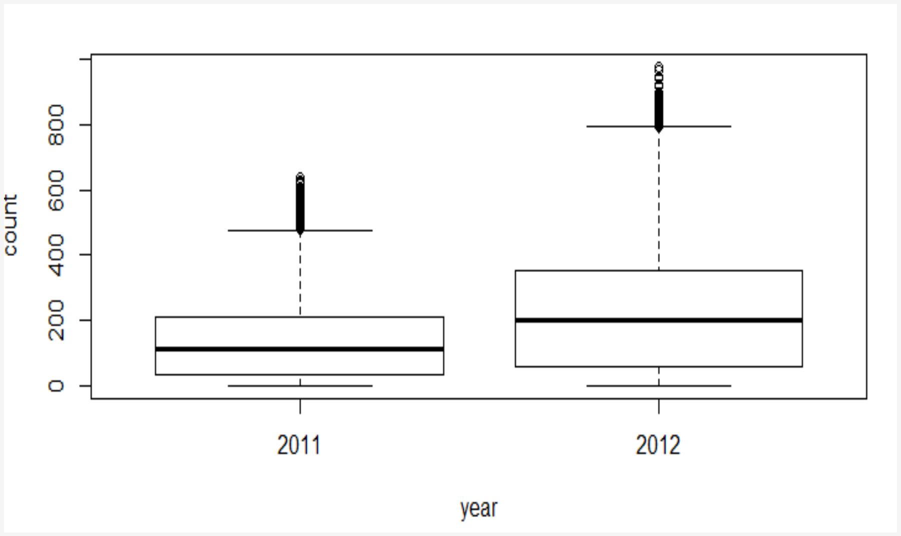
	train.registered	train.casual	train.count	train.temp	train.humidity	train.atemp	train.windspeed
train.registered	1.00	0.50	0.97	0.32	-0.27	0.31	0.09
train.casual	0.50	1.00	0.69	0.47	-0.35	0.46	0.09
train.count	0.97	0.69	1.00	0.39	-0.32	0.39	0.10
train.temp	0.32	0.47	0.39	1.00	-0.06	0.98	-0.02
train.humidity	-0.27	-0.35	-0.32	-0.06	1.00	-0.04	-0.32
train.atemp	0.31	0.46	0.39	0.98	-0.04	1.00	-0.06
train.windspeed	0.09	0.09	0.10	-0.02	-0.32	-0.06	1.00

Here are a few inferences you can draw by looking at the above histograms:

- Variable temp is positively correlated with dependent variables (casual is more compare to registered)
- Variable atemp is highly correlated with temp.
- Wind speed has lower correlation as compared to temp and humidity

Time: Let's extract year of each observation from the date time column and see the trend of bike demand over year.

```
data$year=substr(data$datetime,1,4) data$year=as.factor(data$year)

train=data[as.integer(substr(data$datetime,9,10))<20,]

test=data[as.integer(substr(data$datetime,9,10))>19,]

boxplot(train$count~train$year,xlab="year", ylab="count")
```



We can see that 2012 has higher bike demand as compared to 2011.

Pollution & Traffic: We don't have the variable related with these metrics in our data set so we cannot test this hypothesis.

5. Feature Engineering

In addition to existing independent variables, we will create new variables to improve the prediction power of model. Initially, you must have noticed that we generated new variables like hour, month, day and year.

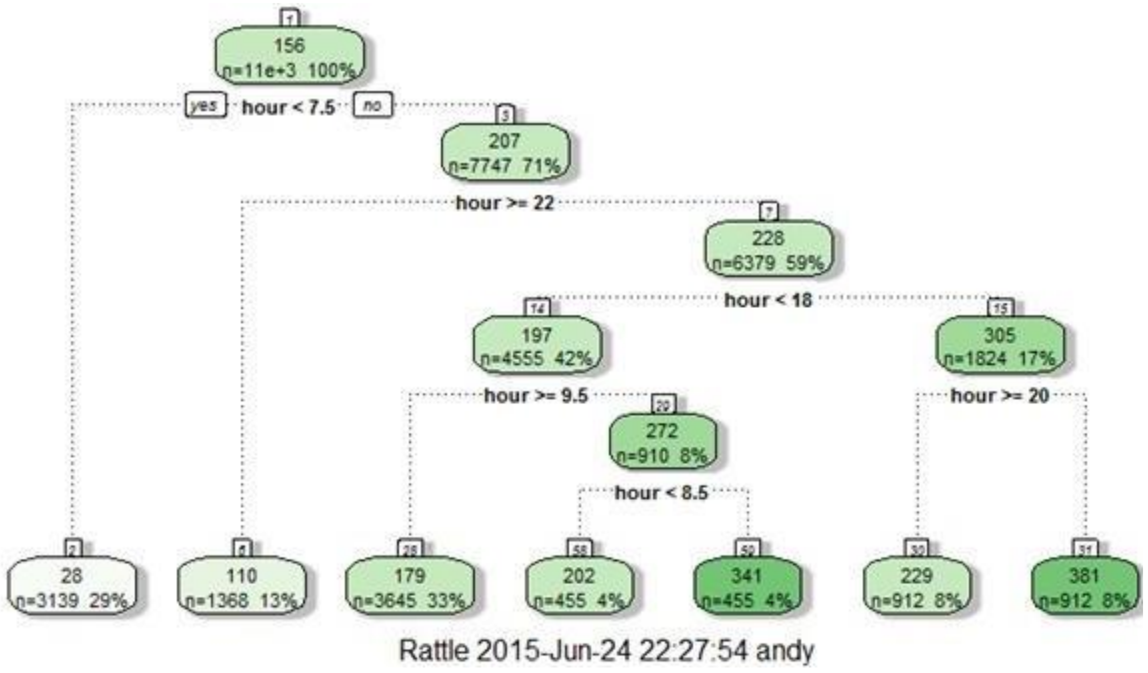
Here we will create more variables, let’s look at the some of these:

Hour Bins: Initially, we have broadly categorize the hour into three categories. Let’s create bins for the hour variable separately for casual and registered users. Here we will use decision tree to find the accurate bins.

```
train$hour=as.integer(train$hour) # convert hour to

integer test$hour=as.integer(test$hour) # modifying in

both train and test data set
```

We use the library rpart for decision tree algorithm.

```
library(rpart)

library(rattle) #these libraries will be used to get a good visual
plot for the decision tree model.

library(rpart.plot) library(RColorBrewer)

d=rpart(registered~hour,data=train)

fancyRpartPlot(d)
```



Now, looking at the nodes we can create different hour bucket for registered users.

```
data=rbind(train,test)

data$dp_reg=0

data$dp_reg[data$hour<8]=1

data$dp_reg[data$hour>=22]=2

data$dp_reg[data$hour>9 &

data$hour<18]=3

data$dp_reg[data$hour==8]=4

data$dp_reg[data$hour==9]=5 data$dp_reg[data$hour==20 |

data$hour==21]=6 data$dp_reg[data$hour==19 | data$hour==18]=7
```

Similarly, we can create day_part for casual users also (dp_cas).

Temp Bins: Using similar methods, we have created bins for temperature for both registered and casuals users. Variables created are (temp_reg and temp_cas).

Year Bins: We had a hypothesis that bike demand will increase over time and we have proved it also. Here I have created 8 bins (quarterly) for two years. Jan-Mar 2011 as 1.Oct-Dec2012 as 8.

```
data$year_part[data$year=='2011']=1

data$year_part[data$year=='2011' &

data$month>3]=2

data$year_part[data$year=='2011' &

data$month>6]=3

data$year_part[data$year=='2011' &

data$month>9]=4

data$year_part[data$year=='2012']=5

data$year_part[data$year=='2012' &

data$month>3]=6
```

```
data$year_part[data$year=='2012' &

data$month>6]=7

data$year_part[data$year=='2012' &

data$month>9]=8 table(data$year_part)
```

Day Type: Created a variable having categories like “weekday”, “weekend” and “holiday”.

```
data$day_type=""

data$day_type[data$holiday==0 & data$workingday==0]="weekend"

data$day_type[data$holiday==1]="holiday"

data$day_type[data$holiday==0 &

data$workingday==1]="working day"
```

Weekend: Created a separate variable

for weekend (0/1)

```
data$weekend=0

data$weekend[data$day=="Sunday" | data$day=="Saturday" ]=1
```

6. Model Building

Before executing the random forest model code, I have followed following steps:

- Convert discrete variables into factor (weather, season, hour, holiday, working day, month, day)

```
train$hour=as.factor(train$hour)

test$hour=as.factor(test$hour)
```

- As we know that dependent variables have natural outliers so we will predict log of dependent variables.
- Predict bike demand registered and casual users separately.
  y1=log(casual+1) and y2=log(registered+1),

Here we have added 1 to deal with zero values in the casual and registered columns.

```
#predicting the log of registered users. set.seed(415)
```

```
fit1 <- randomForest(logreg ~ hour +workingday+day+holiday+ day_type
+temp_reg+humidity+atemp+windspeed+season+weather+dp_reg+weekend+yea
r+year_part, data=train,importance=TRUE, ntree=250)

pred1=predict(fit1,test)

test$logreg=pred1

#predicting the log of casual users. set.seed(415)

fit2 <- randomForest(logcas ~hour +
day_type+day+humidity+atemp+temp_cas+windspeed+season+weather+holiday
+workingday+dp_cas+weekend+year+year_part, data=train,importance=TRUE,
ntree=250)

pred2=predict(fit2,test)

test$logcas=pred2
```

Re-transforming the predicted variables and then writing the output of count to the file submit.csv

```
test$registered=exp(test$logreg)-1

test$casual=exp(test$logcas)-1

test$count=test$casual+test$registered

s<-data.frame(datetime=test$datetime,count=test$count)

write.csv(s,file="submit.csv",row.names=FALSE)
```