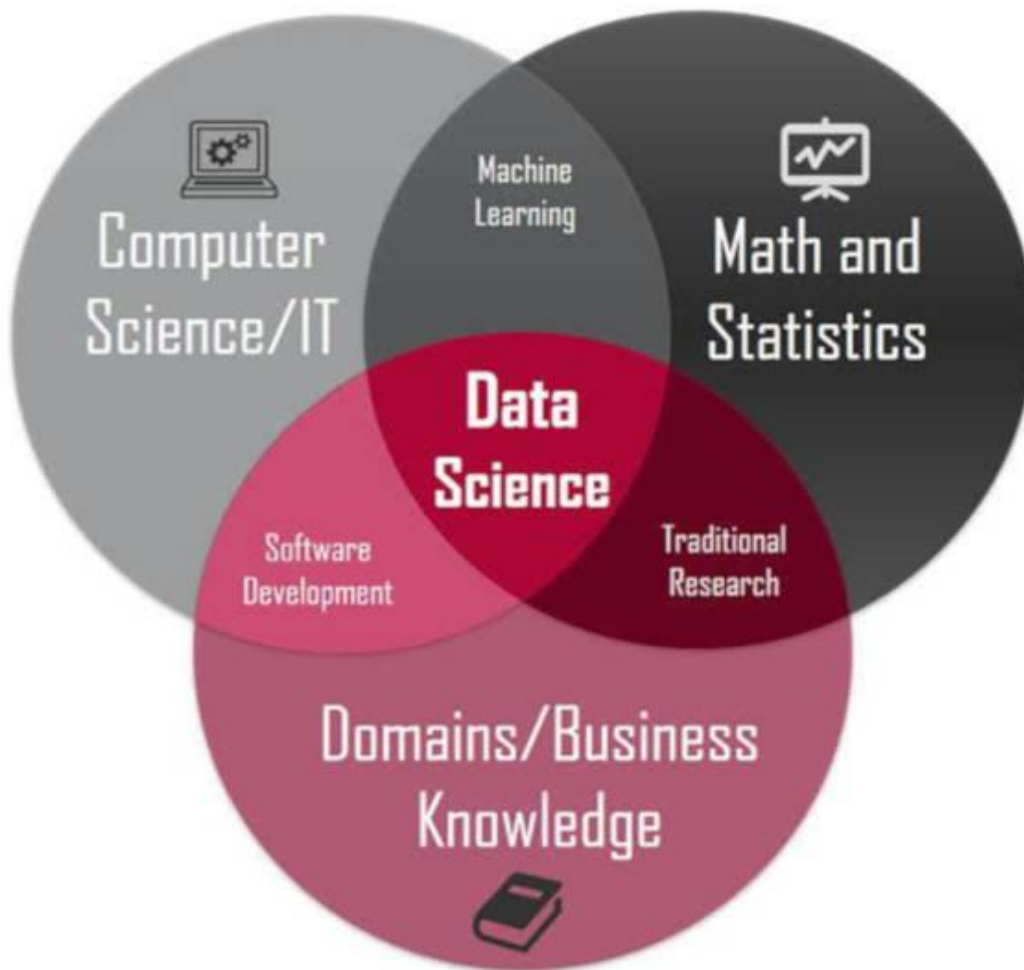


Introduction to R

Data Science is the field of study with Data

Data Science uses Scientific methods, processes, algorithms and systems to extract knowledge and insights from structured and unstructured data.

Data science is a blend of various tools, algorithms and Machine Learning principles to discover hidden patterns from the raw data.



Data Science Life Cycle



1. Business Understanding – Ask Relevant Questions and define objectives for the problem that needs to be tackled.

2. Data Mining – Gather and scrap the data necessary for the project.

3. Data Cleaning- Fix the inconsistencies within the data and handle the missing values.

4. Data Exploration – Form hypothesis about the defined problem by visually analyzing data.

5. Feature Engineering – Select important features and construct more meaningful once using the raw data.

6. Predictive Modelling – Train Machine Learning models, evaluate their performance and use them to make predictions.

7. Data Visualization- Communicate the findings with key stakeholders using plots and interactive visualizations.

Languages for Data Science

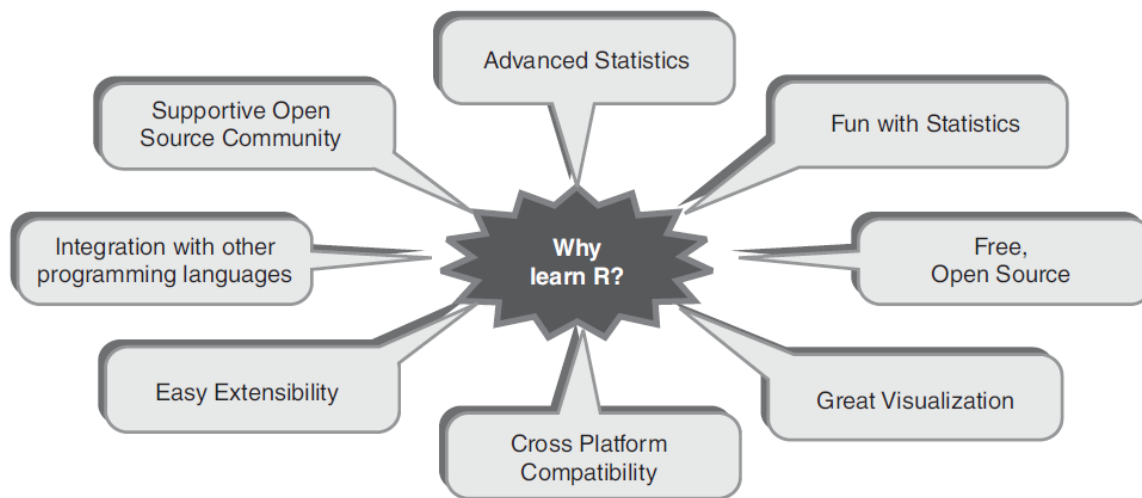
- R
- Python
- Julia
- Scala
- MATLAB

R Programming Language

It was designed by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team.

- **R** is an open-source programming language that is widely used for **statistical analysis, Graphics representation and reporting.**
- R is **a scripting** or programming language which provides an environment for statistical computing, data science and graphics.
- **R** generally comes with the Command-line interface.
- **R is an Interpreted language**
- **R runs on all platforms, R** is available across widely used platforms like Windows, Linux, and macOS.
- **R** programming language is the latest cutting-edge tool.

Why – R Language



- R programming is used as a **leading tool** for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
- It's a **platform-independent language**. This means it can be applied to all operating system.
- It is available for Windows, Mac and a wide variety of Unix platforms.
- It's an **open-source free language**. That means anyone can install it in any organization without purchasing a license.

- R programming language is not only a statistic package but also allows us to **integrate** with other languages (C, C++, Java, Python). Thus, you can easily interact with many data sources and statistical packages.
- The R programming language has **a vast community** of users and it's growing day by day.
- **Development** is in rapid scale.
- **Academia** – Many researchers and scholars use R for experimenting with data science.
- **Data Wrangling**- Cleaning messy and complex data sets, R has extensive library of tools for data manipulation and analysis.
- **Data Visualization**- Graphical form, R has tools for Visualization, Analysis and Representation.
- **Machine Learning**- Train the algorithm and predict tools
- It provides techniques for **various statistical analyses** like classical tests and classification, time series analysis, clustering, linear and non-linear modelling and graphical operations.

- The techniques supported by R are highly extensible.
- Superior support for **graphics**. It can provide well-developed and high-quality plots from data analysis. R has excellent tools for **creating graphics** such as bar charts, scatter plots, multipanel, lattice charts, etc.
- The plots can contain **mathematical formulae and symbols**, if necessary, and users have full control over the selection and use of symbols in the graphics.
- Hence, other than robustness, **user-experience and user-friendliness** are two key aspects of R.
- it is a general programming language so that you can automate your analyses and **create new functions**.
- It has an object oriented and functional programming structure along with support from a robust and vibrant community.
- R has a flexible **analysis tool kit**, which makes it easy to access data in various formats, manipulate it (transform, merge, aggregate, etc).

- R can easily **import data** from MS Excel, MS Access, MySQL, SQLite, Oracle etc. It can easily connect to databases using ODBC (Open Database Connectivity Protocol) and ROracle package.

Features of R Programming Language

1. Statistical Features of R:

- **Basic Statistics:** The most common basic statistics terms are the **mean, mode, and median**. These are all known as “Measures of Central Tendency.” So using the R language we can measure central tendency very easily.
- **Static graphics:** R is rich with facilities for creating and developing interesting static graphics. R contains functionality for many plot types including **graphic maps, mosaic plots, biplots, and so on**.
- **Probability distributions:** Probability distributions play a vital role in statistics and by using R we can easily handle various types of probability distribution such as **Binomial Distribution, Normal Distribution, Chi-squared Distribution** and many more.

- **Data analysis:** It provides a large, coherent and integrated collection of tools for data analysis.

2. Programming Features of R:

- **R Packages:** One of the major features of R is it has a wide availability of libraries. R has CRAN(Comprehensive R Archive Network), which is a repository holding more than **10, 000 packages** in CRAN.
- **Distributed Computing:** Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance. Two new packages **ddR and multidplyr** used for distributed programming in R were released in November 2015.

3. Effective Data handling & Storage Facility

4. Graphical Representation and Analysis

Advantages of R:

- R is the most comprehensive statistical analysis package. As new technology and concepts often appear first in R.
- As R programming language is an open source. Thus, you can run R anywhere and at any time.
- R programming language is suitable for GNU/Linux and Windows operating system.
- R programming is cross-platform which runs on any operating system.
- In R, everyone is welcome to provide new packages, bug fixes, and code enhancements.

Disadvantages of R:

- In the R programming language, the **standard of some packages is less than perfect.**
- Although, R commands give little pressure to **memory management**. So R programming language may consume all available memory.
- R programming language is **much slower than** other programming languages such as Python and MATLAB.

Applications of R:

- We use R for **Data Science**. It gives us a broad variety of libraries related to statistics. It also provides the environment for statistical computing and design.
- R is used by many **quantitative analysts** as its programming tool. Thus, it helps in data importing and cleaning.

Programming in R:

The integrated development suite for R language can be downloaded from the **Comprehensive R Archive Network (CRAN)**

1. Download R interpreter
2. Download R studio

After writing the program save the file with the extension .r.

To run the program use the following command on the command line:

R file_name.r or cntrl+enter

R Studio

R studio is the most widely used IDE for writing, testing and executing R codes.

This is a user-friendly and open source solution.

The various parts in a typical screen of an R studio IDE are

- **Console**, where users write a command and see the output
- **Workspace tab**, where users can see active objects from the code written in the console
- **History tab**, which shows a history of commands used in the code.
- **File tab**, where folders and files can be seen in the default workspace
- **Plot tab**, which shows graphs
- **Packages tab**, which shows add-ons and packages required for running specific process(s)
- **Help tab**, which contains the information on IDE, commands, etc.

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

Project: (None)

Environment History Connections Tutorial

R Global Environment

values

a	89
b	19.5
c	108.5
x	-5

Files Plots Packages Help Viewer

New Folder Delete Rename More

Home

	Name	Size	Modified
<input type="checkbox"/>	.RData	2.6 KB	Aug 31, 2021, 11:53 PM
<input type="checkbox"/>	.Rhistory	422 B	Aug 31, 2021, 11:53 PM
<input type="checkbox"/>	Custom Office Templates		
<input type="checkbox"/>	desktop.ini	402 B	Jul 5, 2021, 1:50 PM
<input type="checkbox"/>	My Music		
<input type="checkbox"/>	My Pictures		
<input type="checkbox"/>	My Videos		
<input type="checkbox"/>	Python Scripts		
<input type="checkbox"/>	R		
<input type="checkbox"/>	Zoom		

```
1 x<--5
2 if(x>0)
3 {
4   print("Non negaitve")
5 } else
6 {
7   print(" negaitve")
8 }
9
10
11
12
13
14
```

7:21 (Top Level) R Script

Console Terminal Jobs

```
R 4.1.1 ~ /
> x<--5
> if(x>0)
+ {
+   print("Non negaitve")
+ } else
+ {
+   print(" negaitve")
+ }
+ [1] " negaitve"
>
```

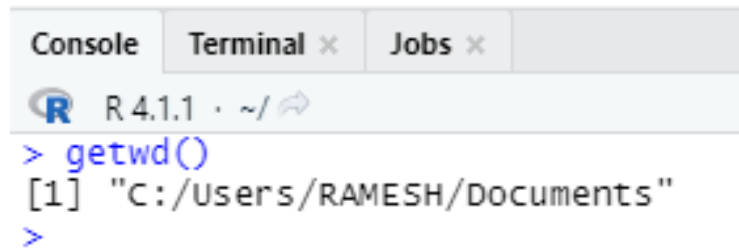
Getting Started with R

Working with Directory

getwd() : Returns the absolute filepath of the current working directory. This function has no arguments.

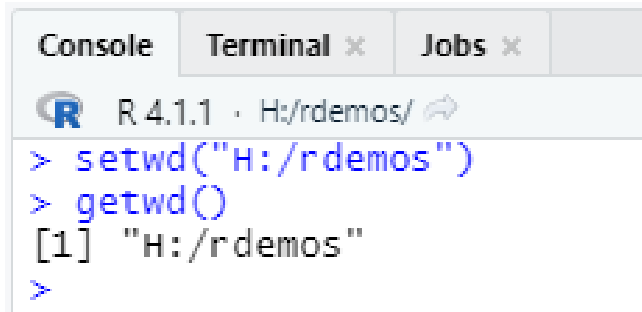
>getwd()

The **getwd()** function can return NULL if the working directory is not available.



A screenshot of the R console interface. At the top, there are tabs for 'Console', 'Terminal x', and 'Jobs x'. Below the tabs, the R logo and version 'R 4.1.1' are shown, followed by the current directory '~/' and a refresh icon. The command prompt shows the execution of `> getwd()`, which returns `[1] "C:/Users/RAMESH/Documents"`. The prompt ends with a greater-than sign `>`.

setwd() : Resets the current working directory to another location as per the user's preference.



A screenshot of the R console interface. At the top, there are tabs for 'Console', 'Terminal x', and 'Jobs x'. Below the tabs, the R logo and version 'R 4.1.1' are shown, followed by the current directory 'H:/rdemos/' and a refresh icon. The command prompt shows the execution of `> setwd("H:/rdemos")`, followed by `> getwd()`, which returns `[1] "H:/rdemos"`. The prompt ends with a greater-than sign `>`.

dir() : returns a character vector of the names of files or directories in the named directory.

**dir(path = ".", pattern = NULL, all.files = FALSE,
full.names = FALSE, recursive = FALSE,
ignore.case = FALSE, include.dirs = FALSE, no.. =
FALSE)**

or

**list.files(path = ".", pattern = NULL, all.files = FALSE,
full.names = FALSE, recursive = FALSE,
ignore.case = FALSE, include.dirs = FALSE, no.. =
FALSE)**

Example:

1. **>dir() => Ouput: character(0)**
>list.files() => Output: character(0)

The above command implies that there are no files or directories in the current directory.

2. To display the files and directories in the current directory, use path= "."

>dir(path=".")

3. To display the list of all files and directories in a specific path.

> dir (path="D:/ramesh/rdemos")

4. To display the complete or absolute path of all files and directories in the specified path

>dir(path="D:/ramesh/rdemos", full.names = TRUE)

5. To look for a specific pattern

> dir(path="D:/ramesh/rdemos", pattern="^D")

6. To display a recursive list of files or directories in the specified path

>dir(path="D:/ramesh/rdemos",recursive=TRUE, include.dirs=TRUE)

Data Types in R

- **R is a programming language.**
- **R makes use of variables to store varied information. This means that when variables are created, locations are reserved in the computer's memory to hold the related values.**
- **The number of locations or size of memory reserved is determined by the data type of the variables.**
- **Data type essentially means the kind of value which can be stored, such as boolean, numbers, characters, etc.**
- **In R, however, variables are not declared as data types.**
- **Variables in R are used to store some R objects and the data type of the R object becomes the data type of the variable.**

The most popular R (Data Structures) objects are:

- I. Vector**
- II. List**
- III. Matrix**
- IV. Array**
- V. Factor**
- VI. Data Frames**

1. vector

A vector is the simplest of all R objects.

It has varied data types. All other R objects are based on these atomic vectors.

class() function can be used to check the data type.

typeof() function can be used to check the data type.

Data types supported by R are:



➤ **Logical:** TRUE / T and FALSE / F are logical values.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> class(TRUE)
[1] "logical"
> TRUE
[1] TRUE
> class(T)
[1] "logical"
> class(F)
[1] "logical"
> class(FALSE)
[1] "logical"
```

➤ **Numeric:** Represents numeric data of integer or real type.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> 123
[1] 123
> class(123)
[1] "numeric"
> x<-12
> class(x)
[1] "numeric"
> class(3.142)
[1] "numeric"
>
```

- **Integer:** Integer data type is a sub class of numeric data type. Notice the use of “L “as a suffix to a numeric value in order for it to be considered an “integer”.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> 123L
[1] 123
> class(123L)
[1] "integer"
>
```

*****Integers are numeric but NOT all numbers are integers.**

is.numeric(): can be used to test numeric or not.

is.integer(): can be used to test integer or not.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> x<-23L
> is.numeric(x)
[1] TRUE
> is.integer(x)
[1] TRUE
> y<-34
> is.integer(y)
[1] FALSE
.
```

- **Character:** Represents characters (String).
is.character(): can be used to test string or not.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> "welcome to R programming"
[1] "welcome to R programming"
> class("welcome to R ")
[1] "character"
> is.character("welcome")
[1] TRUE
> is.character(12)
[1] FALSE
> x<-"vardhaman"
> class(x)
[1] "character"
>
```

- **Double:** Represents double precision floating point values. By default, numbers are of “double” type.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> x<-32.34
> typeof(x)
[1] "double"
> y<-32
> typeof(y)
[1] "double"
> |
```

is.double(): can be used to test double or not.

- **Complex:** Represents complex values.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> 3+4i
[1] 3+4i
> y<-4+5i
> class(y)
[1] "complex"
> typeof(y)
[1] "complex"
> is.complex(y)
[1] TRUE
> is.double(y)
[1] FALSE
> |
```

is.complex(): can be used to test complex or not.

- **Raw:** Represents each byte has hexadecimal value of corresponding ASCII code.

raw vector is printed with each byte separately represented as a pair of hex digits.

is.raw(): can be used to test raw data or not.

```
Console Terminal x Jobs x
R 4.1.1 · H:/rdemos/
> charToRaw("welcome")
[1] 77 65 6c 63 6f 6d 65
> charToRaw("hi")
[1] 68 69
> charToRaw("hi")
[1] 68 69
> class(charToRaw("welldone"))
[1] "raw"
```

Type Coercion

Type coercion allows to convert one data type to another

```
Console Terminal x Jobs x
R 4.1.1 · ~/
> as.character(58)
[1] "58"
> as.integer("58")
[1] 58
> as.integer("welcome")
[1] NA
Warning message:
NAS introduced by coercion
> as.integer(TRUE)
[1] 1
> as.integer(FALSE)
[1] 0
> as.numeric(TRUE)
[1] 1
> as.integer(78.85)
[1] 78
> as.complex(4)
[1] 4+0i
> as.double(78)
[1] 78
> as.double("39.45")
[1] 39.45
> as.logical(1)
[1] TRUE
```

Variables and Assignment in R

Variables are used to store the information to be manipulated and referenced in the R program.

The R variable can store an **atomic vector**, a group of atomic vectors, or a combination of many R objects.

R is a **dynamically typed**, means it check the type of data type when the statement is run.

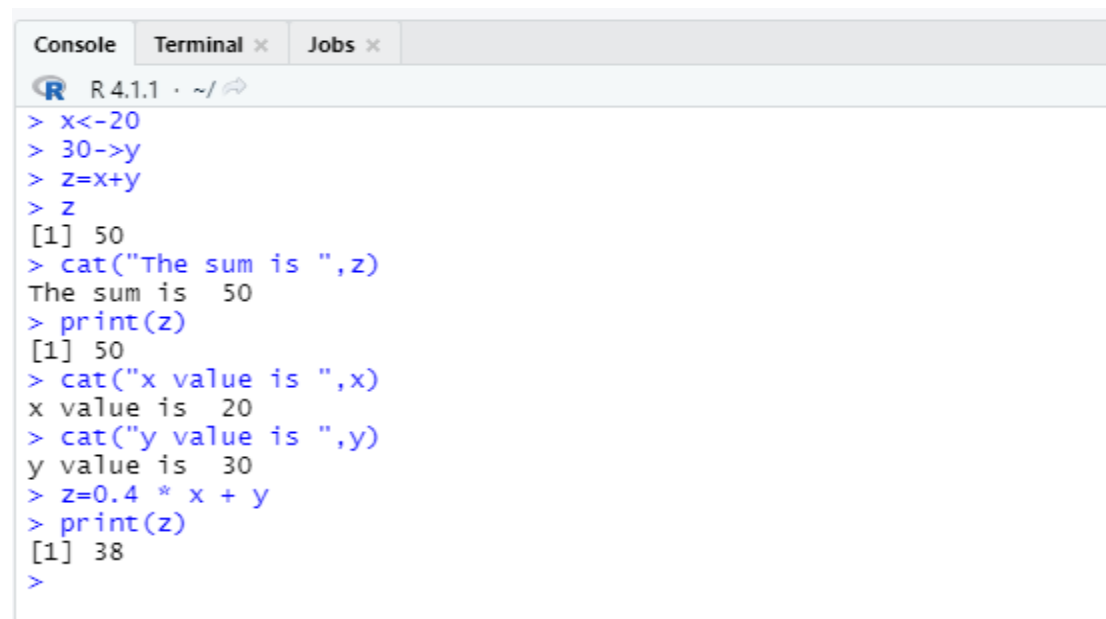
Eg: `var_name`, `var.name`, `var_name2`

In R programming, three operators which we can use to assign the values to the variable. We can use **leftward (<-)**, **rightward (->)**, and **equal_to (=)** operator.

There are two functions which are used to print the value of the variable i.e., **print()** and **cat()**.

The **cat()** function combines multiples values into a continuous print output.

The **print()** function output only a single value.



```
Console Terminal x Jobs x
R 4.1.1 ~ /
> x<-20
> 30->y
> z=x+y
> z
[1] 50
> cat("The sum is ",z)
The sum is 50
> print(z)
[1] 50
> cat("x value is ",x)
x value is 20
> cat("y value is ",y)
y value is 30
> z=0.4 * x + y
> print(z)
[1] 38
>
```

When a value is assigned to a variable, it does not display anything on the console. To get the value, type the name of the variable at the prompt.

```
>len
```

```
>wid
```

Use the **ls()** function to list all the objects in the working environment.

```
>ls()
```

```
Console Jobs x
R 4.1.1 · ~/
> ls()
[1] "a"      "area" "b"      "c"      "len"    "wid"    "x"
> |
```

Keywords in R Programming

In programming, a keyword is a word which is reserved by a program because it has a special meaning.

A keyword can be a command or a parameter.

A keyword can't be used as a variable name.

Keywords are also called as "reserved names."

if	else	repeat
while	function	for
next	break	TRUE
FALSE	NULL	Inf
NaN	NA	NA_integer_
NA_real_	NA_complex_	NA_character_

Operators in R

Arithmetic Operators:

Operator	Meaning
+	This operator is used to add two vectors in R.
-	This operator is used to subtract a vector from another one.
*	This operator is used to multiply two vectors with each other.
/	This operator divides the vector from another one.
%%	This operator is used to find the remainder of the first vector with the second vector.
/%	This operator is used to find the division of the first vector with the second(quotient).Only Integer part is the result
^	This operator raised the first vector to the exponent of the second vector.

```
Console Terminal x Jobs x
R 4.1.1 ~ /
> a<-4
> b<-5
> print(a+b)
[1] 9
> print(a-b)
[1] -1
> print(a*b)
[1] 20
> print(a/b)
[1] 0.8
> print(a%%b)
[1] 4
> print(a/%b)
[1] 0
> print(9/%4)
[1] 2
> print(a^b)
[1] 1024
> |
```

C() -> function is combine to perform operation at a time on all arguments.


```
Console Terminal x Jobs x
R 4.1.1 ~ /
> a=c(2,3,4)
> b=c(4,6,8)
> print(a+b)
[1] 6 9 12
> print(a-b)
[1] -2 -3 -4
> print(a*b)
[1] 8 18 32
> print(a/b)
[1] 0.5 0.5 0.5
> print(a%b)
[1] 2 3 4
> print(a/%b)
[1] 0 0 0
> print(b/%a)
[1] 2 2 2
> print(a^b)
[1] 16 729 65536
> |
```

Logical Operators:

Operator	Meaning
&	Logical AND operator. (look for every element in vector)
	Logical OR operator. (look for every element in vector)
!	Logical NOT operator.
/	This operator divides the vector from another one.
&&	AND Operator (look only for first element in vector)
	OR operator (look only for first element in vector)

```
> a<-10
> b<-0
> print(a&b)
[1] FALSE
> print(a|b)
[1] TRUE
> print(!b)
[1] TRUE
> print(a&& b)
[1] FALSE
> print(a|| b)
[1] TRUE
>
```

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Console Terminal x Jobs x
R 4.1.1 ~ /
> a <- c(3, 0, TRUE, 2+2i)
> b <- c(2, 4, TRUE, 2+3i)
> print(a&&b)
[1] TRUE
> print(a&b)
[1] TRUE FALSE TRUE TRUE
> print(a || b)
[1] TRUE
>
> print(a|b)
[1] TRUE TRUE TRUE TRUE
> print(!a)
[1] FALSE TRUE FALSE FALSE
> a <- c(3, 0, TRUE, 2+2i)
> b <- c(0, 4, TRUE, 2+3i)
> print(a&&b)
[1] FALSE
> |

```

Assignment Operators:

Operator	Meaning
<- or = or <<-	left assignment operators.
-> or ->>	right assignment operators..

```

RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function Addins

Console Terminal x Jobs x
R 4.1.1 ~ /
> a<-34.5
> b<<-45.6
> c=78.5
> print(a,b,c)
Error in print.default(a, b, c) : invalid printing digits 45
> print(a)
[1] 34.5
> print(b)
[1] 45.6
> print(c)
[1] 78.5
> 99 -> a
> 100 ->>b
> print(a)
[1] 99
> print(b)
[1] 100
> a<-(3,4,5+6i)
Error: unexpected ',' in "a<-(3,"
> a<-c(3,4,5+6i)
> print(a)
[1] 3+0i 4+0i 5+6i

```

Miscellaneous Operators:

Operator	Meaning
:	used to create the series of numbers in sequence for a vector.
%in%	to identify if an element belongs to a vector.

```
Console Terminal x Jobs x
R 4.1.1 · ~/
> d<- 5:20
> print(d)
[1] 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
> x1<- 5
> x2<- 16
> print(x1%in%d)
[1] TRUE
> print(x2%in%d)
[1] TRUE
> print(30%in%d)
[1] FALSE
```

Conditional Statements in R

1. if statement

if(boolean_expression) {

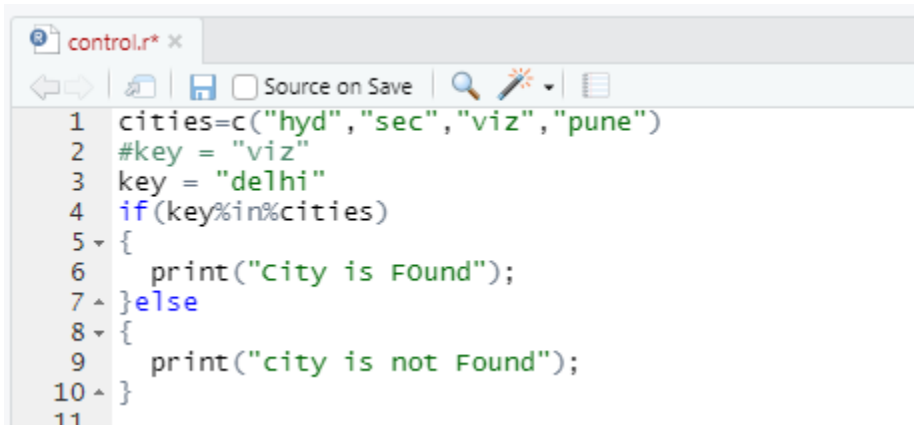
// If the boolean expression is true, then statement(s) will be executed.

}

```
control.r* x
Source on Save
1 a=10
2 b=20
3 if(a<b)
4   print(a)
5
6 |
```

2. if-else statement

```
if(boolean_expression) {  
    // statement(s) will be executed if the boolean expression is true.  
} else {  
    // statement(s) will be executed if the boolean expression is false.  
}
```

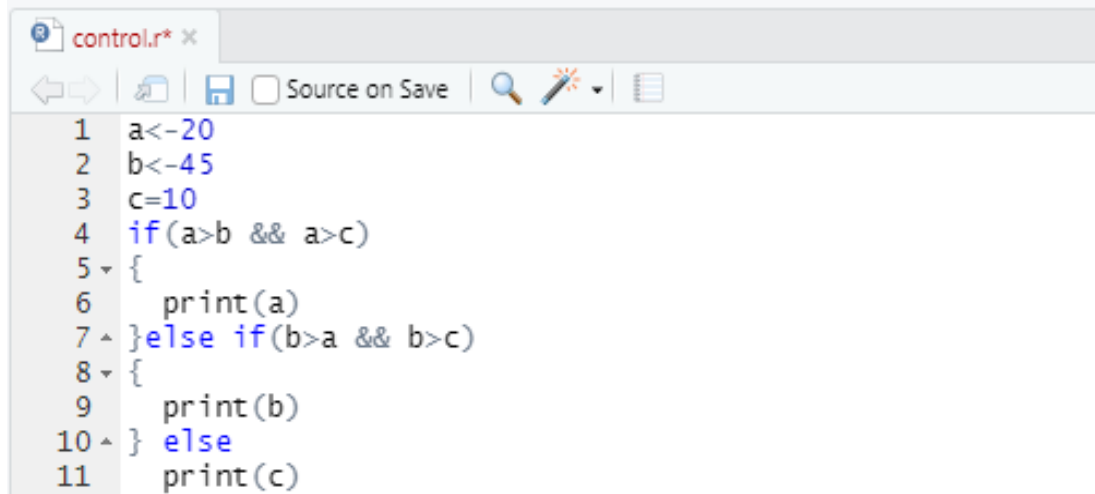


The screenshot shows a code editor window titled 'control.r*'. The code is as follows:

```
1 cities=c("hyd","sec","viz","pune")  
2 #key = "viz"  
3 key = "delhi"  
4 if(key%in%cities)  
5 {  
6     print("City is Found");  
7 }else  
8 {  
9     print("city is not Found");  
10 }  
11
```

3. else-if statement

```
if(boolean_expression 1) {  
    // This block executes when the boolean expression 1 is true.  
} else if( boolean_expression 2) {  
    // This block executes when the boolean expression 2 is true.  
} else if( boolean_expression 3) {  
    // This block executes when the boolean expression 3 is true.  
} else {  
    // This block executes when none of the above condition is true.  
}
```

A screenshot of an R script editor window titled 'control.r*'. The window has a toolbar with icons for back, forward, save, source on save, search, and a list. The script content is as follows:

```
1 a<-20
2 b<-45
3 c=10
4 if(a>b && a>c)
5 {
6   print(a)
7 } else if(b>a && b>c)
8 {
9   print(b)
10 } else
11   print(c)
```

4. switch statement

- If expression type is a character string, the string is matched to the listed cases.
- If there is more than one match, the first match element is used.
- No default case is available.
- If no case is matched, an unnamed case is used.

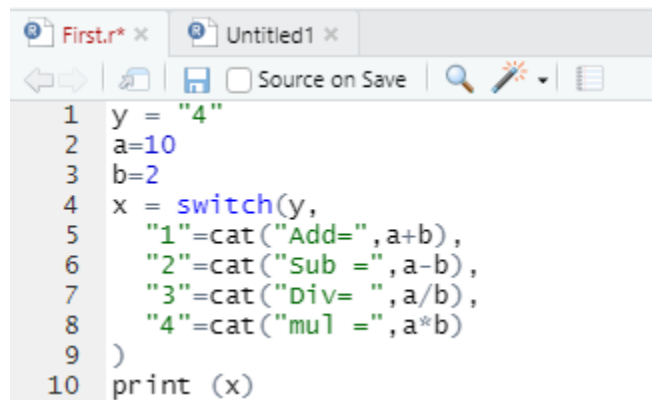
switch statement is used in two forms.

1. Based on Index

switch(expression, case1, case2, case3....)

Eg: `fruit <- switch(3,"apple","banana","papaya","watermelon")`
`print(fruit) #papaya`

2. Based on matching value



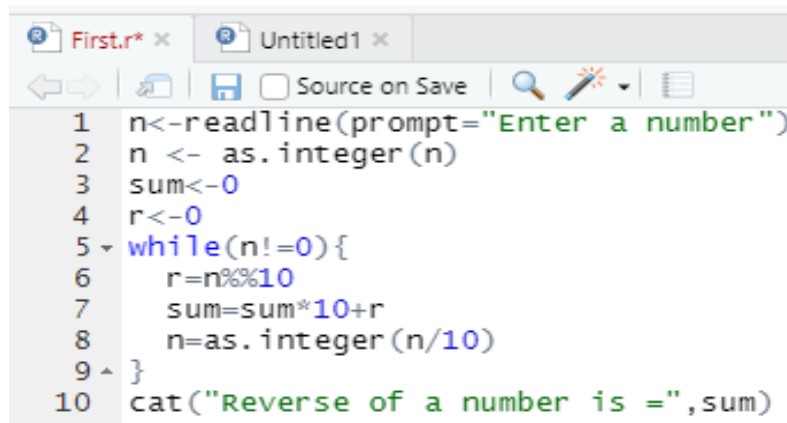
```
1 y = "4"
2 a=10
3 b=2
4 x = switch(y,
5   "1"=cat("Add=",a+b),
6   "2"=cat("Sub =",a-b),
7   "3"=cat("Div= ",a/b),
8   "4"=cat("mul =",a*b)
9 )
10 print (x)
```

***If no case matches. unnamed case is invoked and results in null.

Iterative/Loop Statements in R

1. while loop

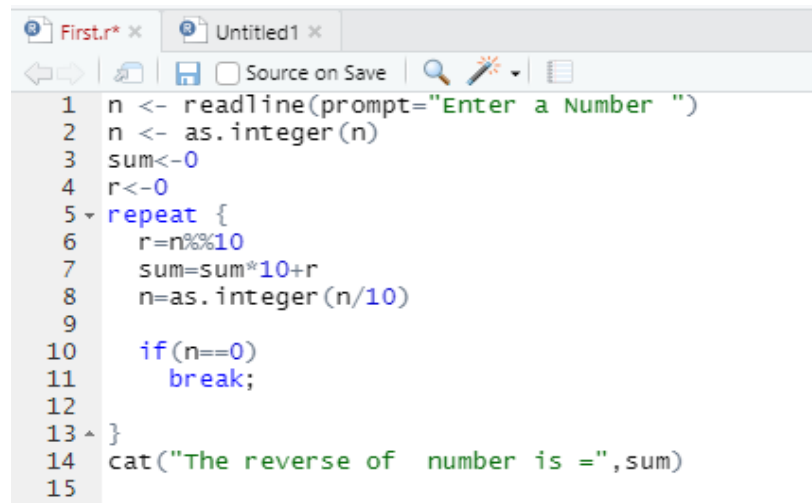
```
while (test_expression) {
  statements
}
```



```
1 n<-readline(prompt="Enter a number")
2 n <- as.integer(n)
3 sum<-0
4 r<-0
5 while(n!=0){
6   r=n%%10
7   sum=sum*10+r
8   n=as.integer(n/10)
9 }
10 cat("Reverse of a number is =",sum)
```

2. repeat loop

```
repeat {
  commands / statements
  if(condition) {
    break
  }
}
```



```

1 n <- readline(prompt="Enter a Number ")
2 n <- as.integer(n)
3 sum<-0
4 r<-0
5 repeat {
6   r=n%%10
7   sum=sum*10+r
8   n=as.integer(n/10)
9
10  if(n==0)
11    break;
12
13 ^ }
14 cat("The reverse of number is =",sum)
15

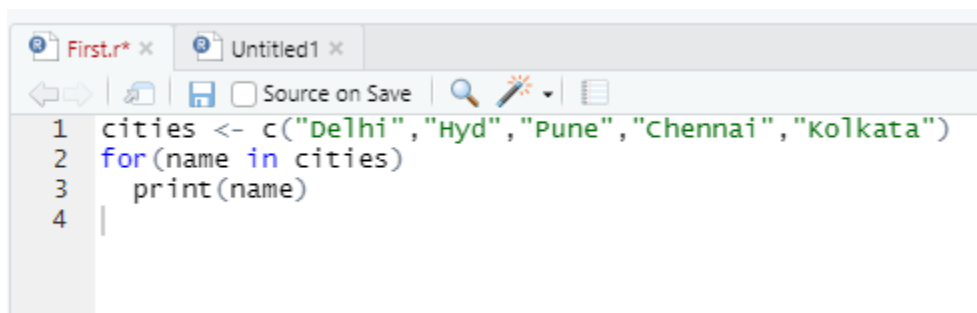
```

3. for loop

```

for (var in vector) {
  statements
}

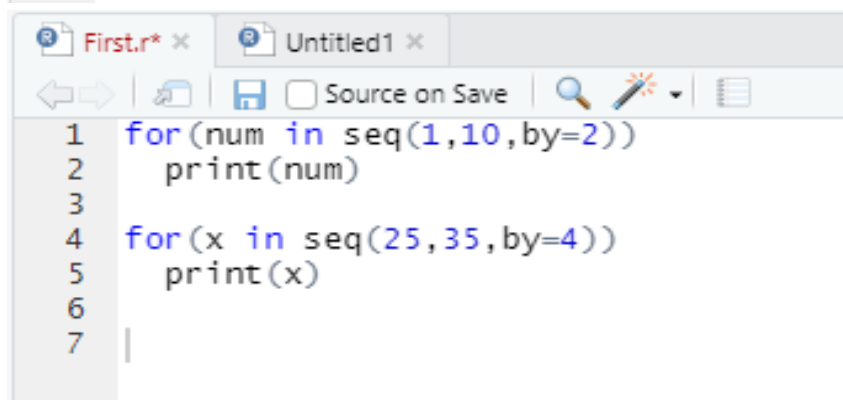
```



```

1 cities <- c("Delhi","Hyd","Pune","Chennai","kolkata")
2 for(name in cities)
3   print(name)
4

```



```

1 for(num in seq(1,10,by=2))
2   print(num)
3
4 for(x in seq(25,35,by=4))
5   print(x)
6
7

```

next statement

The next statement is used to skip any remaining statements in the loop and continue executing. i.e a next statement is a statement which skips the current iteration of a loop without terminating it

```
First.r* x  Untitled1 x
< >  Save Source on Save  Search  Run  Help
1 x<-1:10
2 print(x)
3 for(i in x)
4 {
5   if(i%%3==0)
6     next
7   print(i)
8 }
```

break statement

In the R language, the break statement is used to break the execution and for an immediate exit from the loop. In nested loops, break exits from the innermost loop only and control transfer to the outer loop.

```
First.r* x  Untitled1 x
< >  Save Source on Save  Search  Run  Help
1 x<-1:10
2 print(x)
3 for(i in x)
4 {
5   if(i%%3==0)
6     break
7   print(i)
8 }
```


Handling Packages in R

A package in R is the fundamental unit of shareable code.

A package is a collection of the following elements:

- Functions
- Data sets
- Compiled code
- Documentation for the package and for the functions inside
- Tests – few tests to check if everything works as it should.

The directory where packages are stored is called a **library**. R comes with a standard set of packages. Others are available for download and installation as per requirement.

10,000 plus packages available in CRAN. This is also one of the reasons behind the huge popularity and success of R.

One can develop their **own R package**, as it is open source any R user can then download, install and learn to use the package.

Packages, therefore allow for an easy, transparent and cross-platform extension of the R base system.

R is an **open source language**; thus, new packages are being developed and updated by developers daily. Some of these packages

may not work properly or may have bugs. Hence, it is not a good idea to use every new and updated package on R development environment. This can affect the stability of the development environment.

A stable environment requires the **sandboxing technique** - a security mechanism often used to execute untested or untrusted programs or code from unverified or untrusted third parties, users, etc., without damaging/maligning the host machine or operating system or production environment.

Sandboxing technique A stable environment to test new packages or update a package before installing it in the development environment.

The command **.libPaths()** can be used to get or set the path of the package library.

It is used to change the default path of library.

.libPaths() //get path of all libraries

.libpaths("new path for libraries to set") //change the path of all libraries.

R can be extended easily with the help of a rich set of packages.

The commonly used packages are

<i>Data Management</i>	<i>Data Visualisation</i>	<i>Data Products</i>	<i>Data Modelling and Simulation</i>
dplyr, tidyr, foreign, haven etc.	ggplot, ggvis, lattice, igraph etc.	shiny, slidify, knitr, markdown etc.	MASS, forecast, bootstrap, broom, nlme, ROCR, party etc.

<i>Package Name</i>	<i>Description</i>	<i>Available At</i>
arm	It is used for hierarchical or multi-level regression models.	http://cran.r-project.org/web/packages/arm/
lme4	It contains functions for generating generalised and linear mixed-effects models.	http://cran.r-project.org/web/packages/lme4/
Rcurl	It provides an interface of R to the package library, libcurl. The interface helps in interacting with the HTTP protocols for importing raw data from the web.	http://www.omegahat.org/Rcurl/
RJSONIO	It provides a set of functions to read and write JSON for analysing data from different web-based APIs.	http://www.omegahat.org/RJSONIO/
XML	It provides functions and facilities for analysing HTML and XML documents to extract structured data from web-based sources.	http://www.omegahat.org/RXML/

<i>Author(s)</i>	<i>Package Name</i>	<i>Description</i>	<i>Available At</i>
Gabor Csardi	igraph	It contains routines for network analysis and making simple graphs to represent social networks.	http://igraph.sourceforge.net/
Hadley Wickham	ggplot	It contains a set of grammar rules for implementing graphics in R. The package is used for creating high-quality graphics.	http://cran.r-project.org/web/packages/ggplot2/index.html
Hadley Wickham	lubridate	The package provides functions to use dates in R in an easier way.	https://github.com/hadley/lubridate
Hadley Wickham	reshape	It contains a set of tools for manipulation, aggregation and management of data in R.	http://had.co.nz/plyr/
Ingo Feinerer	tm	It contains functions to perform text mining in R. Text mining helps to work with unstructured data.	http://www.spatstat.org/spatstat/
Jerome Friedman, Trevor Hastie, and Rob Tibshirani	glmnet	It helps to work with the elastic-net and also regularised and generalised linear models.	http://had.co.nz/ggplot2/

Installing an R Package

First Approach:

1. On the R Console window, Install R package by choosing “Install package(s)” from the “Packages” menu at the top of the R console. (In R studio: tools-> Install Packages or Use directly packages tab)
2. Choose the name of the package to install. **E.g. ggplot2**
3. The package is now installed. When we want to use the “ggplot2”. first we have to load the package by typing into the R console:

library(“ggplot2”).

4. We can get help on a package by
help (package = “ggplot2”)

Second Approach:

install.packages(): to install one or more packages.

>install.packages(“ggplot2”)

>install.packages(c(“ggplot”, “tidyr”, “dplyr”))

Basic Commands to work with Packages

installed.packages() : check for all installed packages on the machine.

> installed.packages()

remove.packages(): used to uninstall packages, specify the name of package as argument.

> remove.packages("package1","package2"..)

packageDescription(): It has details such as what the package does, who is the author, what is the version for the documentation, the date, the type of license its use, and the package dependencies, etc. of a specified argument as file name.

> packageDescription ("package name")

Eg: > packageDescription("stats")

help (): to get help on a specified package

>help (package = "ggplot2")

>help ("ggplot2")

>help(package = "datasets") -> lists all data sets

library (): to load a package into memory before when using it frequently.

> library("datasets")

#Loads data sets library.

>AirPassengers

loads data set AirPassengers

or

>datasets::AirPassengers

#to access dataset directly

find.package(): to search or find a package, if exist its path is returned.

>find.package("ggplot2")

Commands for Data Exploration

1. Load Internal Dataset

To Check the “datasets” package is installed or not.

>installed.packages()

To Check the path of “datasets” package in system.

>find.package(“datasets”)

Load the package “datasets”

>library(“datasets”)

2. Accessing a data set (e.g. mtcars)

>mtcars => shows the data set.

[, 1]	mpg	Miles/(US) gallon
[, 2]	cyl	Number of cylinders
[, 3]	disp	Displacement (cu.in.)
[, 4]	hp	Gross horsepower
[, 5]	drat	Rear axle ratio
[, 6]	wt	Weight (1000 lbs)
[, 7]	qsec	1/4 mile time
[, 8]	vs	V/S
[, 9]	am	Transmission (0 = automatic, 1 = manual)
[,10]	gear	Number of forward gears
[,11]	carb	Number of carburetors

The dataset “mtcars” has 32 observations on 11 variables.

3. Accessing data set characteristics using commands.

summary(): command includes functions like min, max, median, mean, etc., for each variable present in the given data frame.

>summary(“datasets”)

```
Console Jobs x
R 4.1.1 ~ /
> summary(mtcars)
      mpg      cyl      disp      hp      drat
Min.   :10.40  Min.   :4.000  Min.   : 71.1  Min.   : 52.0  Min.   :2.760
1st Qu.:15.43  1st Qu.:4.000  1st Qu.:120.8  1st Qu.: 96.5  1st Qu.:3.080
Median :19.20  Median :6.000  Median :196.3  Median :123.0  Median :3.695
Mean   :20.09  Mean   :6.188  Mean   :230.7  Mean   :146.7  Mean   :3.597
3rd Qu.:22.80  3rd Qu.:8.000  3rd Qu.:326.0  3rd Qu.:180.0  3rd Qu.:3.920
Max.   :33.90  Max.   :8.000  Max.   :472.0  Max.   :335.0  Max.   :4.930

      wt      qsec      vs      am      gear
Min.   :1.513  Min.   :14.50  Min.   :0.0000  Min.   :0.0000  Min.   :3.000
1st Qu.:2.581  1st Qu.:16.89  1st Qu.:0.0000  1st Qu.:0.0000  1st Qu.:3.000
Median :3.325  Median :17.71  Median :0.0000  Median :0.0000  Median :4.000
Mean   :3.217  Mean   :17.85  Mean   :0.4375  Mean   :0.4062  Mean   :3.688
3rd Qu.:3.610  3rd Qu.:18.90  3rd Qu.:1.0000  3rd Qu.:1.0000  3rd Qu.:4.000
Max.   :5.424  Max.   :22.90  Max.   :1.0000  Max.   :1.0000  Max.   :5.000

      carb
Min.   :1.000
1st Qu.:2.000
Median :2.000
Mean   :2.812
3rd Qu.:4.000
Max.   :8.000
>
```

str(): command displays the internal structure of a data frame. It is a diagnostic function and displays one line per basic object.

```
Console Jobs x
R 4.1.1 ~ /
> str(mtcars)
'data.frame':   32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num   6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num   3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num   2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num   0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num   1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num   4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num   4 4 1 1 2 1 4 2 2 4 ...
```

rnorm(): To generate a vector of normally distributed random numbers.

e.g. Generate a vector of 100 normally distributed random numbers with standard mean and sd arguments used are 2 and 4.


```
Console Jobs x
R 4.1.1 · ~/
> x<-rnorm(100,2,4)
> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-8.4399 -0.4396  1.4089  1.4602  3.7581  9.1764
> x
 [1] -0.14245951  7.71816476  2.65690535 -2.20972229  3.56292934  4.00454617
 [7]  1.00171151  3.58473223  4.10434750  5.98980088  4.57219621  2.57732984
[13]  0.95856578  3.52130423  0.92366451  4.97501317  2.19201853  8.43208919
[19]  0.23438256 -2.23414990  0.73913596  1.82040736  6.28841414 -0.42439418
[25] -0.41853702 -0.99560601 -3.63456899  5.02285909  4.91362563  1.64627617
[31]  2.68016540 -0.48510162 -1.98730931  3.71924497  4.35979457 -8.43990234
[37]  2.31963217  6.60790853  1.91354511 -1.98726290  2.47411457  1.24250116
[43] -1.74411094 -0.59146334  3.72417413  2.83376602  6.17987118  4.85840562
[49]  0.14551917  2.46803472  2.95606667 -0.19741537 -0.16285132  2.62255635
[55]  5.12197615  1.01543527  7.55122241  9.17638743 -7.81198675 -2.63855210
[61]  6.11435525  3.53922664 -2.50714769  2.22648736  1.21276679 -3.37262437
[67]  6.02903525  6.97290582  2.35518821 -0.76510912  1.75888150  0.63258525
[73]  0.77745860 -3.21004082 -0.12426911  2.98987551  0.54323027  0.91767876
[79] -6.19805552  0.43800336  7.47600935  0.31299912  3.33586825 -4.80101678
[85] -2.92704462  0.83706231 -5.81213318  1.41904282  1.39884655  5.21277831
[91] -0.71438813  6.99207116  3.86000984 -1.65476793 -4.25718601 -3.03940996
[97] -0.04568901  5.86753407 -6.69216841 -0.38418487
.

Console Jobs x
R 4.1.1 · ~/
> m<-matrix(rnorm(9),3,3)
> m
      [,1]      [,2]      [,3]
[1,] -0.4902098  0.5071346 -0.1615156
[2,]  1.0880484 -0.3885631  0.2799333
[3,] -1.1428472 -0.2819964 -2.9212243
> str(m)
 num [1:3, 1:3] -0.49 1.088 -1.143 0.507 -0.389 ...
> summary(m)
      v1      v2      v3
Min.   :-1.1428  Min.   :-0.38856  Min.   :-2.92122
1st Qu.: -0.8165  1st Qu.: -0.33528  1st Qu.: -1.54137
Median : -0.4902  Median : -0.28200  Median : -0.16152
Mean   : -0.1817  Mean   : -0.05448  Mean   : -0.93427
3rd Qu.:  0.2989  3rd Qu.:  0.11257  3rd Qu.:  0.05921
Max.   :  1.0880  Max.   :  0.50713  Max.   :  0.27993
```

head(): command displays the first “n” observations from the given data frame. The default value for n is 6. We can specify number of observations required with “n”.

```

Console Jobs x
R 4.1.1 · ~/
> df<-head(mtcars)
> df
      mpg  cyl  disp  hp drat   wt  qsec vs  am  gear  carb
Mazda RX4    21.0   6  160  110 3.90 2.620 16.46 0   1    4    4
Mazda RX4 wag 21.0   6  160  110 3.90 2.875 17.02 0   1    4    4
Datsun 710    22.8   4  108   93 3.85 2.320 18.61 1   1    4    1
Hornet 4 Drive 21.4   6  258  110 3.08 3.215 19.44 1   0    3    1
Hornet Sportabout 18.7   8  360  175 3.15 3.440 17.02 0   0    3    2
Valiant      18.1   6  225  105 2.76 3.460 20.22 1   0    3    1
> df1<-head(mtcars,n=4)
> df1
      mpg  cyl  disp  hp drat   wt  qsec vs  am  gear  carb
Mazda RX4    21.0   6  160  110 3.90 2.620 16.46 0   1    4    4
Mazda RX4 wag 21.0   6  160  110 3.90 2.875 17.02 0   1    4    4
Datsun 710    22.8   4  108   93 3.85 2.320 18.61 1   1    4    1
Hornet 4 Drive 21.4   6  258  110 3.08 3.215 19.44 1   0    3    1
>

```

tail(): command displays the last “n” observations from a given data frame. The default value for n is 6. We can specify number of observations required with “n”.

```

Console Jobs x
R 4.1.1 · ~/
> tail(mtcars)
      mpg  cyl  disp  hp drat   wt  qsec vs  am  gear  carb
Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.7 0   1    5    2
Lotus Europa  30.4   4  95.1 113 3.77 1.513 16.9 1   1    5    2
Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5 0   1    5    4
Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5 0   1    5    6
Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6 0   1    5    8
Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6 1   1    4    2
> df<-tail(mtcars,4)
> df
      mpg  cyl  disp  hp drat   wt  qsec vs  am  gear  carb
Ford Pantera L 15.8   8 351 264 4.22 3.17 14.5 0   1    5    4
Ferrari Dino   19.7   6 145 175 3.62 2.77 15.5 0   1    5    6
Maserati Bora  15.0   8 301 335 3.54 3.57 14.6 0   1    5    8
Volvo 142E     21.4   4 121 109 4.11 2.78 18.6 1   1    4    2
> |

```

ncol(): returns the number of columns in the given dataset.

nrow(): returns the number of rows in the given dataset.

```
Console Jobs x
R 4.1.1 · ~/
> ncol(mtcars)
[1] 11
> rows<-nrow(mtcars)
> rows
[1] 32
>
```

edit(): command helps with the dynamic editing or data manipulation of a dataset. When this command is invoked, a dynamic data editor window opens with a tabular view of the dataset. The changes can be saved to a new data frame object.

```
Console Jobs x
R 4.1.1 · ~/
> df<-edit(mtcars)
> head(df,4)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	8	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	6	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1

```
> |
```

The rounded values are changed in original data set.

fix(): command saves the changes in the dataset itself, so there is no need to assign any variable to it.

>edit() -> Do changes in original dataset

>fix() -> Fix the changes in original dataset.

***To read help on any command in R, the user can type "?" followed by the function name on the console.

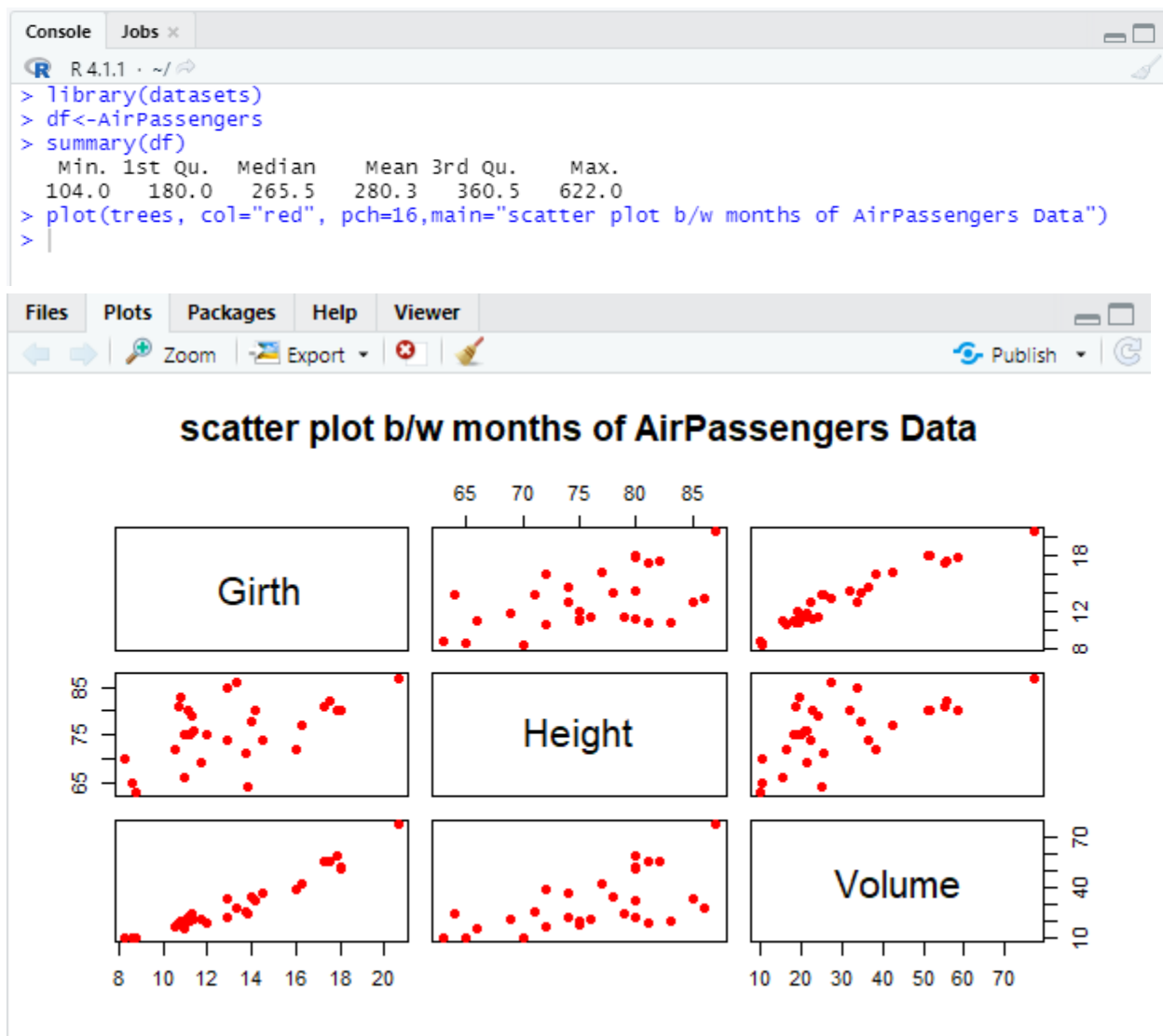
>?edit()

data(): function lists the available datasets.

>data() -> all data sets

>data("AirPassengers") -> returns AirPassengers dataset

plot(): between the variables of a data set.



save.image(): writes an external representation of R objects to the specified file. At a later point in time when it is required to read back the objects.

Syntax

save.image(file = ".RData", version = NULL, ascii = FALSE, safe = TRUE)

The file is to be given an extension of RData.

Note: The "R" and "D" in "RData" should be in capitals.

If `ascii = TRUE`, will save an ascii representation of the file. The default is `ascii = FALSE`. With `ascii` being set to false, a binary representation of the file is saved. `version` is used to specify the current workspace format version. The value of `NULL` specifies the current default format. `safe` is set to a logical value. A value of `TRUE` means that a temporary file is used to create the saved workspace. This temporary file is renamed to file if the save succeeds.

1. Name a few packages used for data management in R.

Ans: dplyr, tidyr, foreign, haven, etc.

2. Name a few packages used for data visualisation in R.

Ans: ggplot, ggvis, lattice, igraph, etc.

3. Name a few packages used for developing data produces in R.

Ans: shiny, slidify, knitr, markdown, etc.

4. Name a few packages used for data modelling and simulation in R.

Ans: MASS, forecast, bootstrap, broom, nlme, ROCR, party, etc.

5. How can the default path to package library be changed in R?

Ans: To change the default package library in R, users need to follow the following steps on the console of R IDE:

Step 1: Check the current path to the package library

> .libPaths()

Step 2: Change the path using the following command.

> .libPaths("write the desired path here")

6. What is the command to check and install the "dplyr" package?

Ans: if (!require("dplyr ")) {install.packages("dplyr")}

7. How can we install multiple packages in R?

Ans: To install multiple packages in R the command is,
>install.packages(c("ggplot","tidyr","dplyr"))