What is R?

Introduction to R

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, …) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

The R environment

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

The term "environment" is intended to characterize it as a fully planned and coherent system, rather than an incremental accretion of very specific and inflexible tools, as is frequently the case with other data analysis software.

R, like S, is designed around a true computer language, and it allows users to add additional functionality by defining new functions. Much of the system is itself written in the R dialect of S, which makes it easy for users to follow the algorithmic choices made. For computationally-intensive tasks, C, C++ and Fortran code can be linked and called at run time. Advanced users can write C code to manipulate R objects directly.

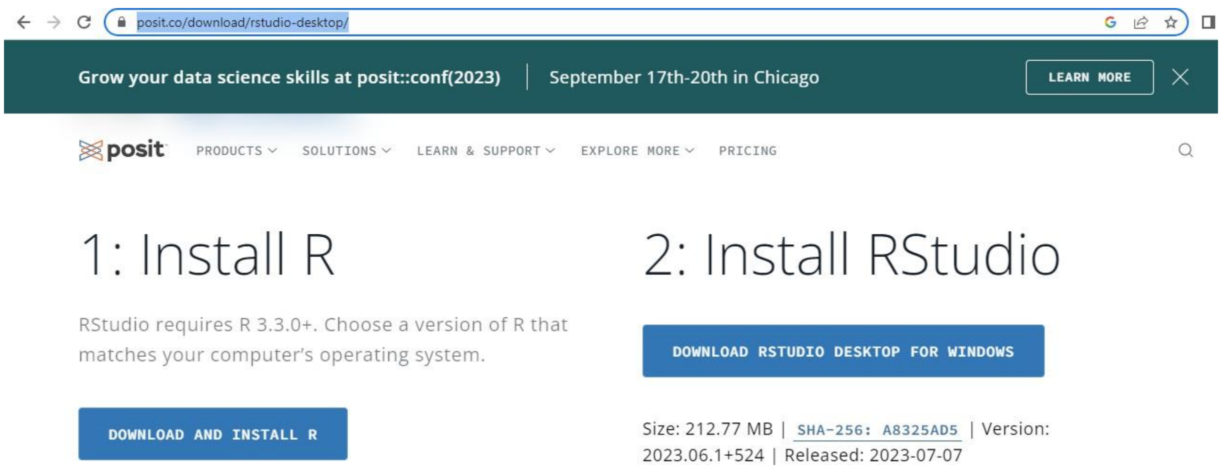
Many users think of R as a statistics system. We prefer to think of it as an environment within which statistical techniques are implemented. R can be extended (easily) via *packages*. There are about eight packages supplied with the R distribution and many more are available through the CRAN family of Internet sites covering a very wide range of modern statistics.

R has its own LaTeX-like documentation format, which is used to supply comprehensive documentation, both on-line in a number of formats and in hardcopy.

Installation of R:

Step1:

https://posit.co/download/rstudio-desktop/


posit.co/download/rstudio-desktop/
Grow your data science skills at posit::conf(2023)
September 17th-20th in Chicago
LEARN MORE
posit
PRODUCTS
SOLUTIONS
LEARN & SUPPORT
EXPLORE MORE
PRICING
DOWNLOAD AND INSTALL R
DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

1: Install R

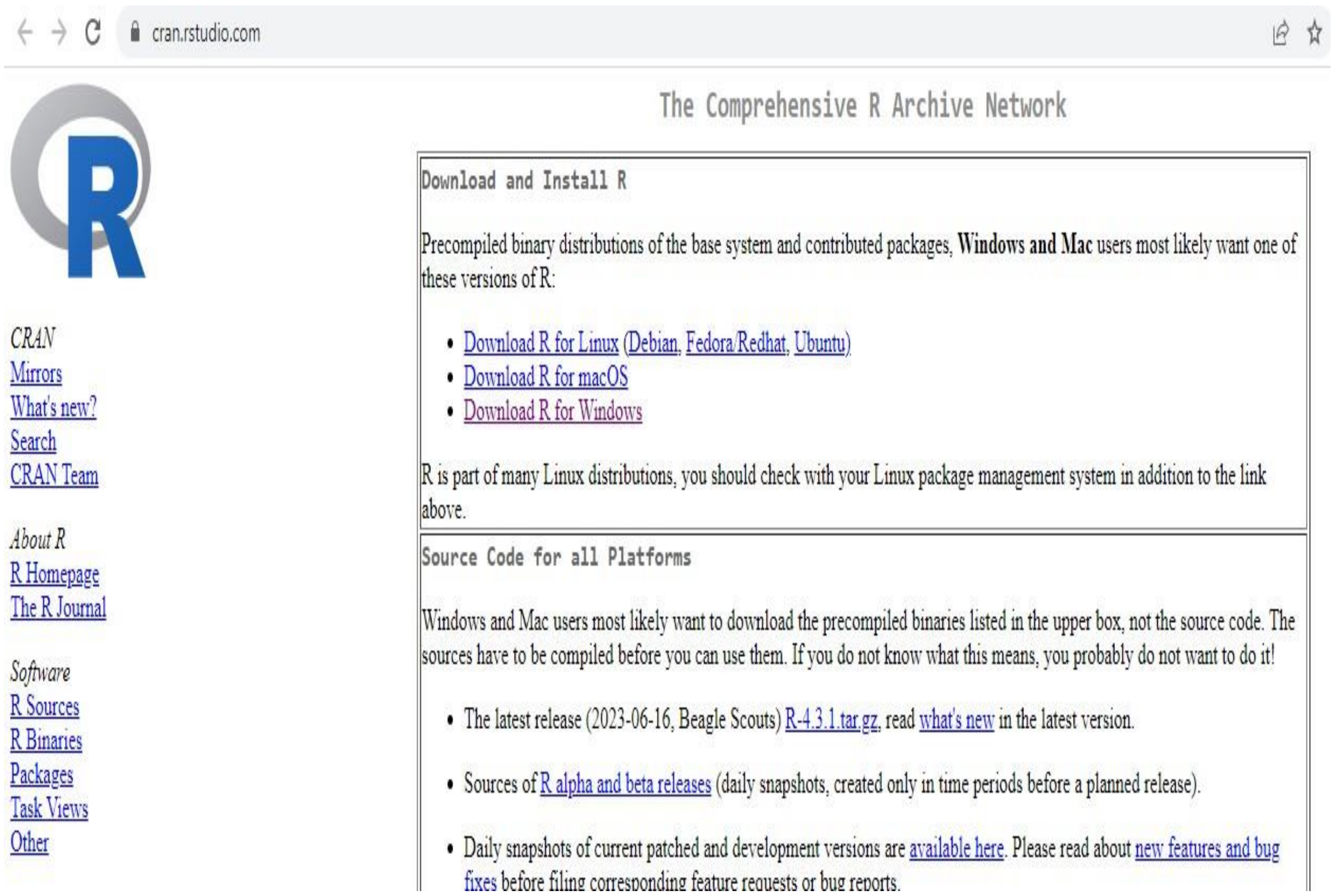
RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

2: Install RStudio

Size: 212.77 MB | SHA-256: A8325AD5 | Version: 2023.06.1+524 | Released: 2023-07-07

Click on Download and install R

PROG NO:		PG NO:
DATE:		R NO:



The Comprehensive R Archive Network

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 (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

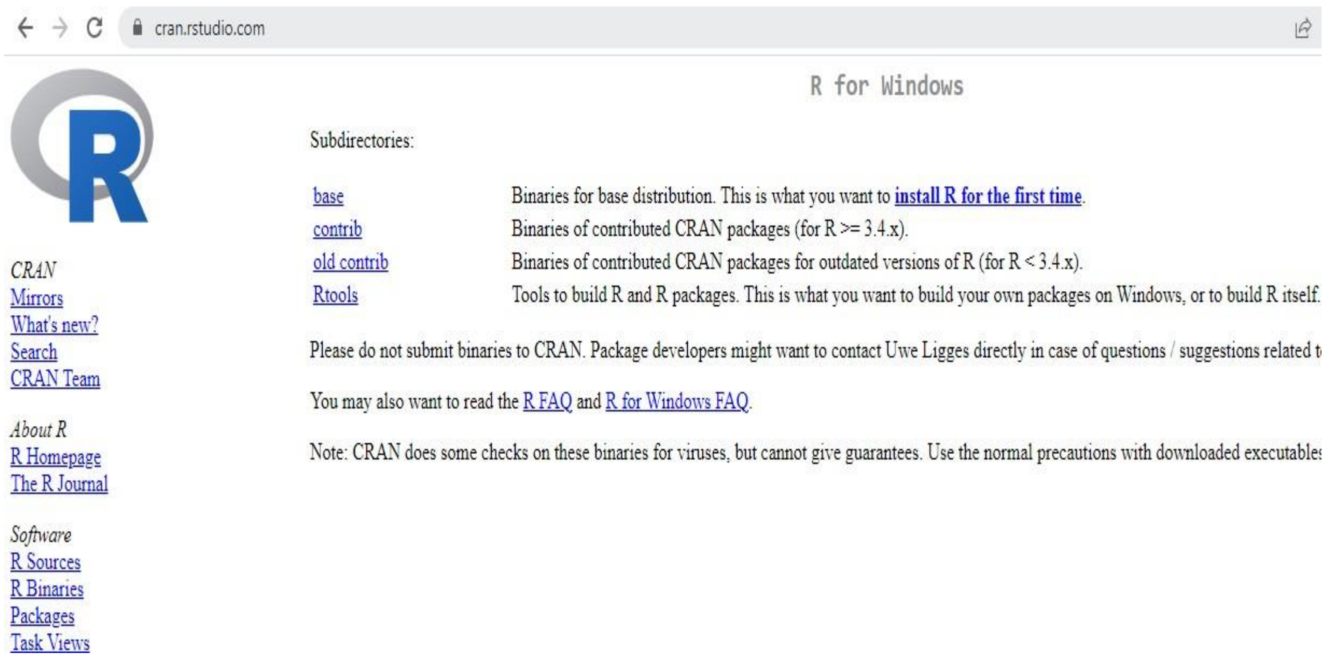
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

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 (2023-06-16, Beagle Scouts) R-4.3.1.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.

Click on Download R for Windows


cran.rstudio.com

R for Windows

Subdirectories:

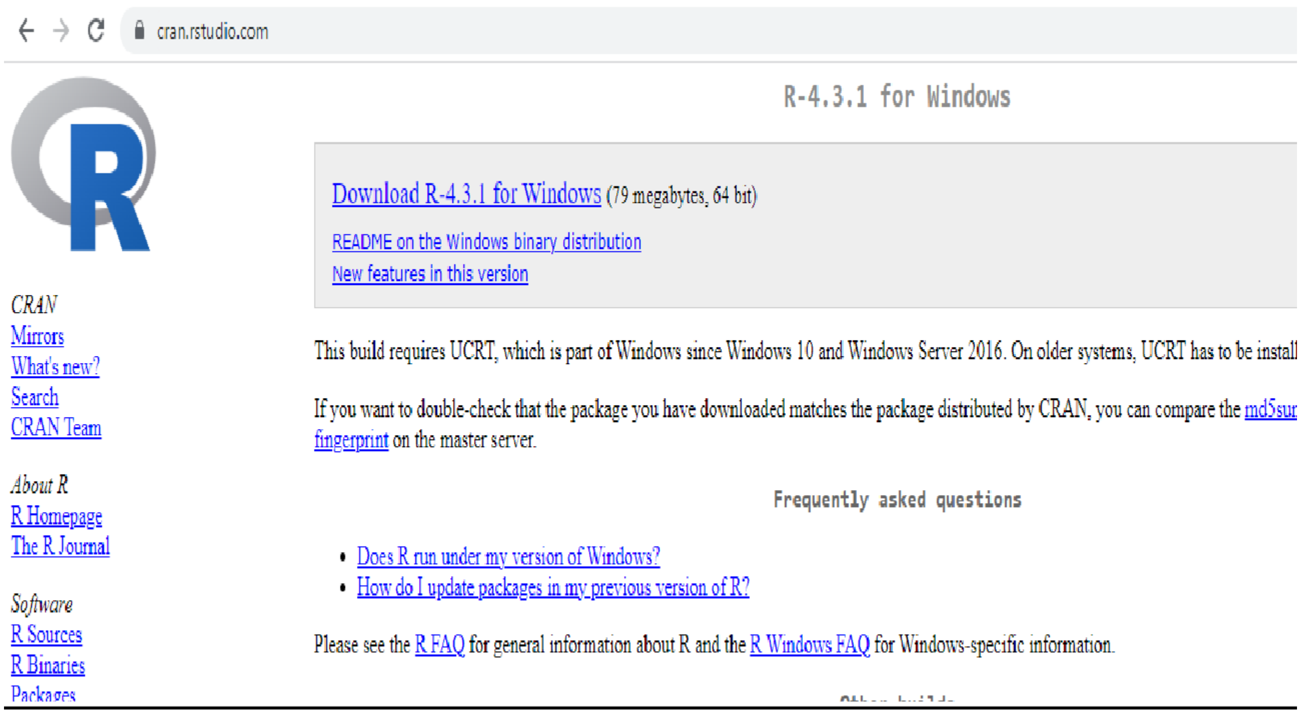
base	Binaries for base distribution. This is what you want to **install R for the first time**.
contrib	Binaries of contributed CRAN packages (for R >= 3.4.x).
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x).
Rtools	Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself.

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related t

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executable

Click on **Install R for the first time**



cran.rstudio.com

R-4.3.1 for Windows

Download R-4.3.1 for Windows (79 megabytes, 64 bit)
README on the Windows binary distribution
New features in this version

This build requires UCRT, which is part of Windows since Windows 10 and Windows Server 2016. On older systems, UCRT has to be install

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sur fingerprint on the master server.

Frequently asked questions

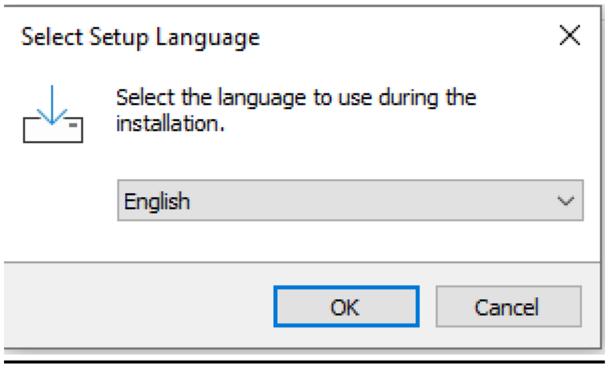
- Does R run under my version of Windows?
- How do I update packages in my previous version of R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

Click on **Download R-4.3.1 for Windows**

https://cran.rstudio.com/bin/windows/base/R-4.3.1-win.exe

Click on **Yes**



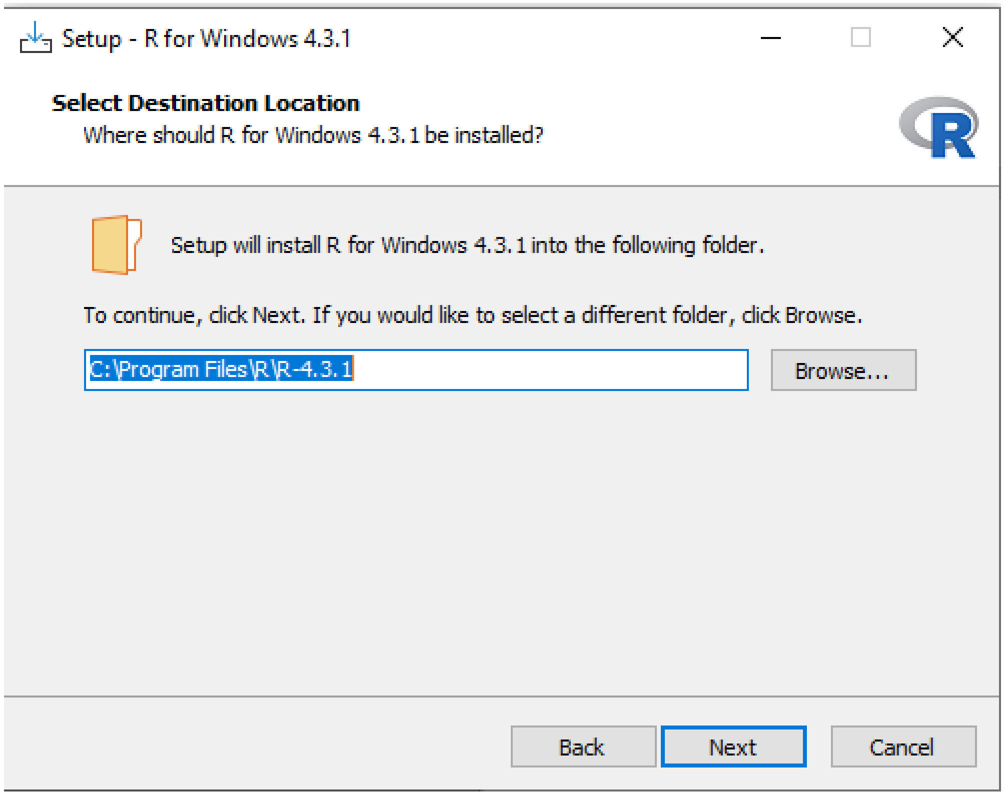
Select Setup Language

Select the language to use during the installation.

English

Click on Ok,Next

PROG NO: DATE:		PG NO: R NO:


Setup - R for Windows 4.3.1

Select Destination Location

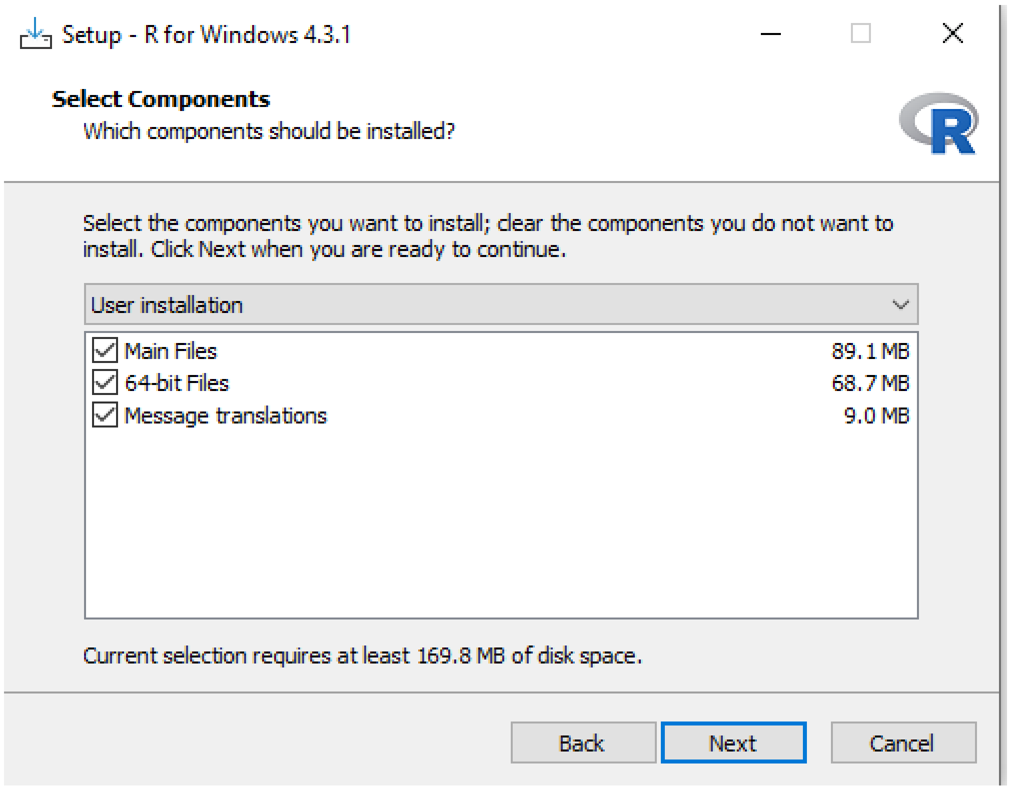
Where should R for Windows 4.3.1 be installed?

Setup will install R for Windows 4.3.1 into the following folder.

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

C:\Program Files\R\R-4.3.1

PROG NO:	PG NO:
DATE:	R NO:



Setup - R for Windows 4.3.1

Select Components
Which components should be installed?

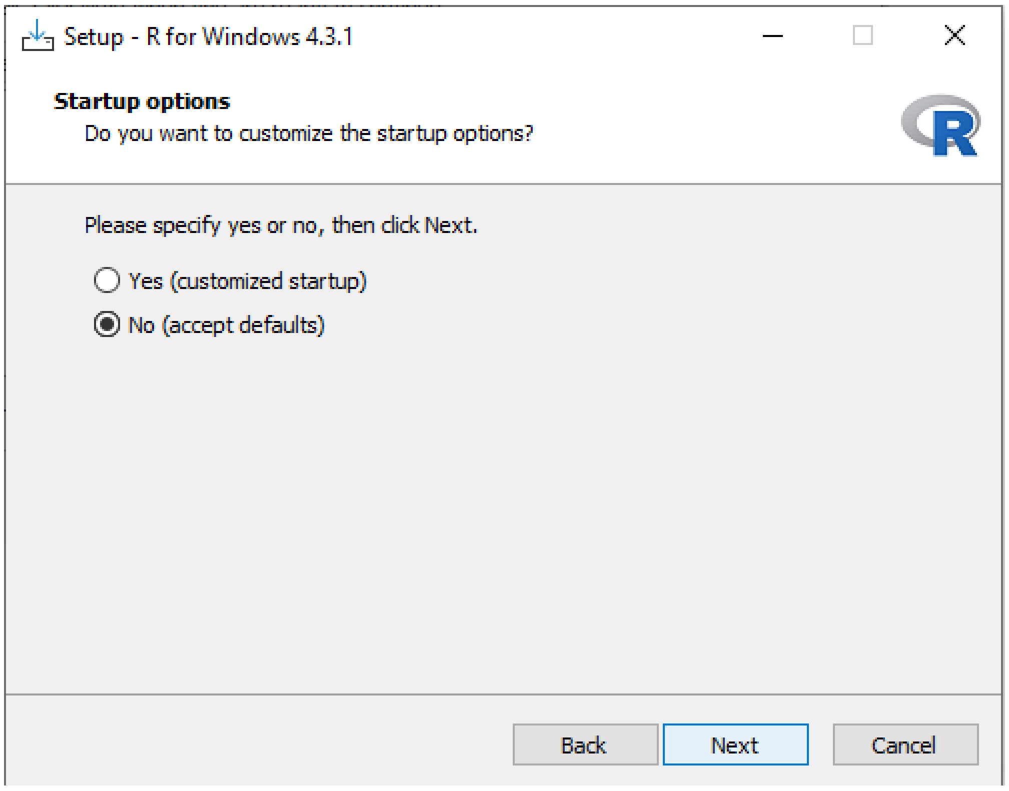
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

User installation

☑ Main Files	89.1 MB
☑ 64-bit Files	68.7 MB
☑ Message translations	9.0 MB

Current selection requires at least 169.8 MB of disk space.

PROG NO:		PG NO:
DATE:		R NO:

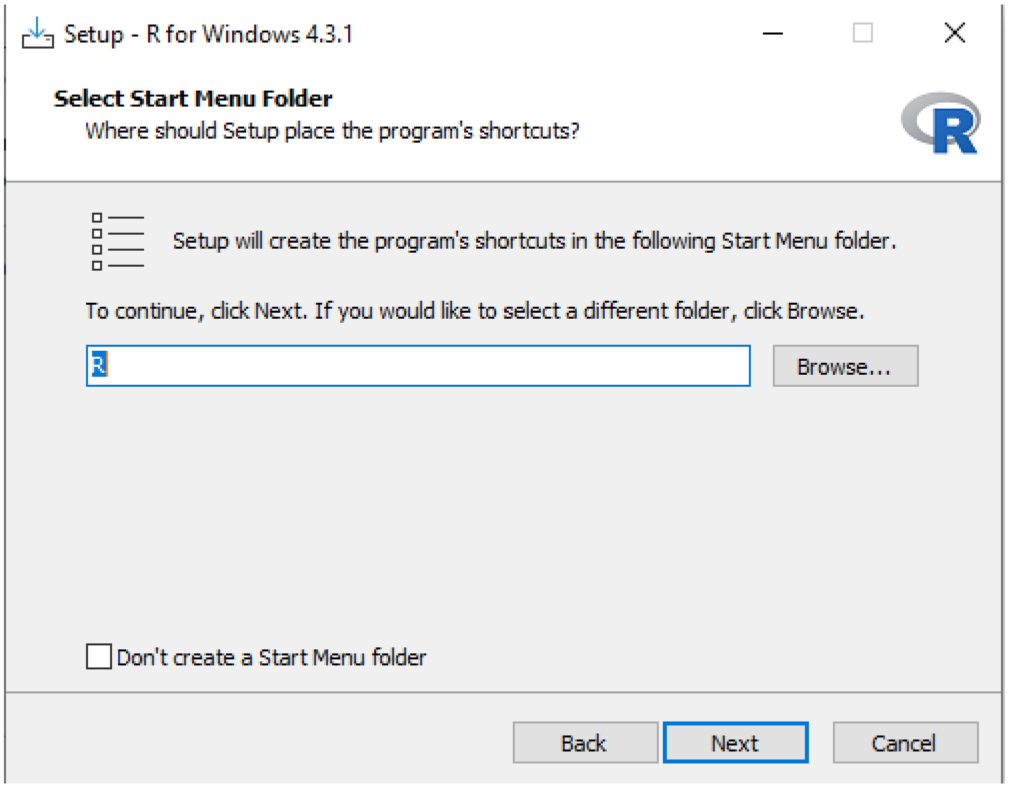

Setup - R for Windows 4.3.1

Startup options

Do you want to customize the startup options?

Please specify yes or no, then click Next.

Yes (customized startup) [✓ No (accept defaults)]


Setup - R for Windows 4.3.1

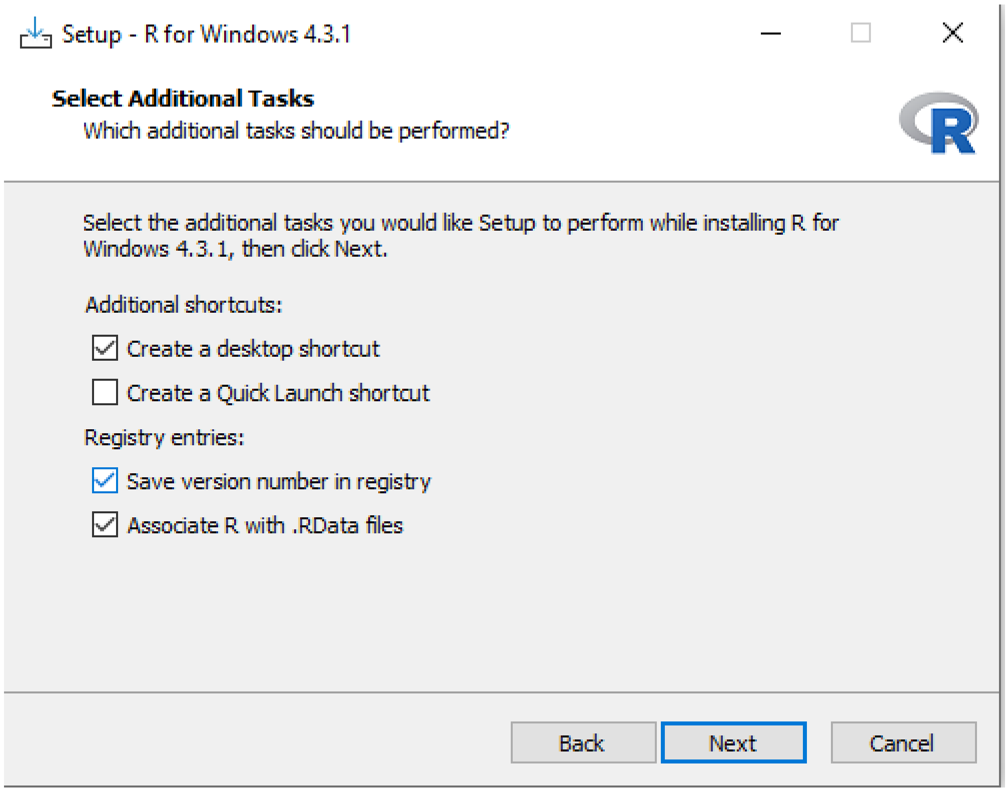
Select Start Menu Folder
Where should Setup place the program's shortcuts?

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

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

R

☐ Don't create a Start Menu folder



Setup - R for Windows 4.3.1

Select Additional Tasks

Which additional tasks should be performed?

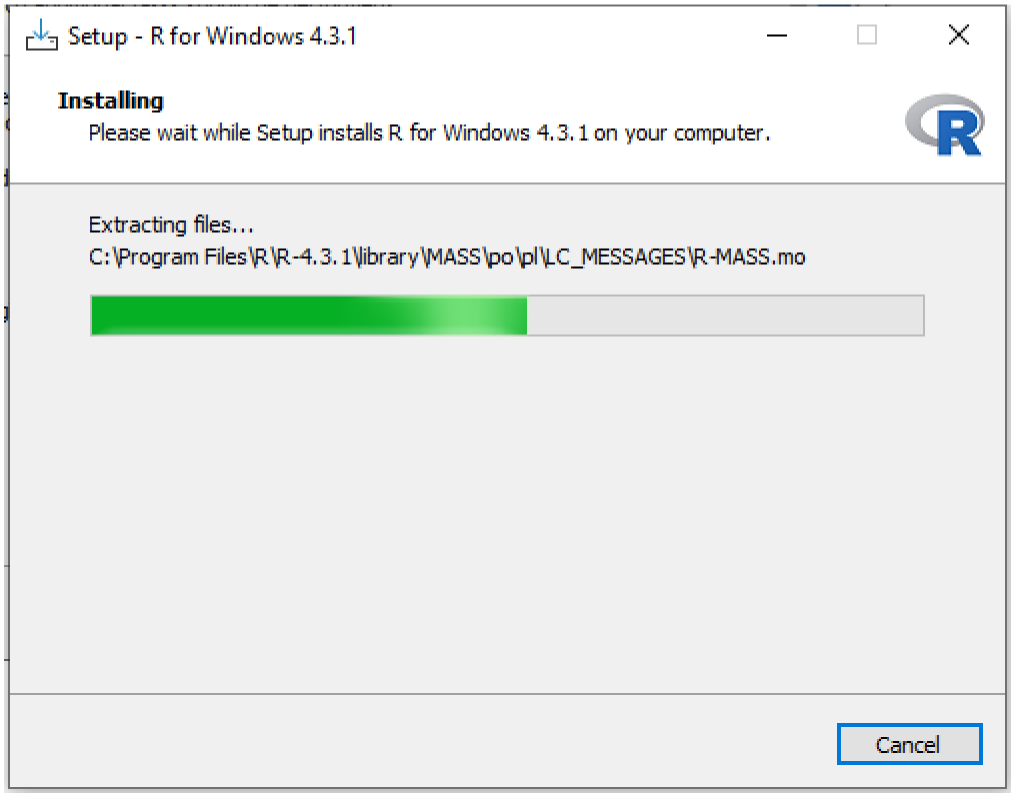
Select the additional tasks you would like Setup to perform while installing R for Windows 4.3.1, then click Next.

Additional shortcuts:

- ☑ Create a desktop shortcut
- ☐ Create a Quick Launch shortcut

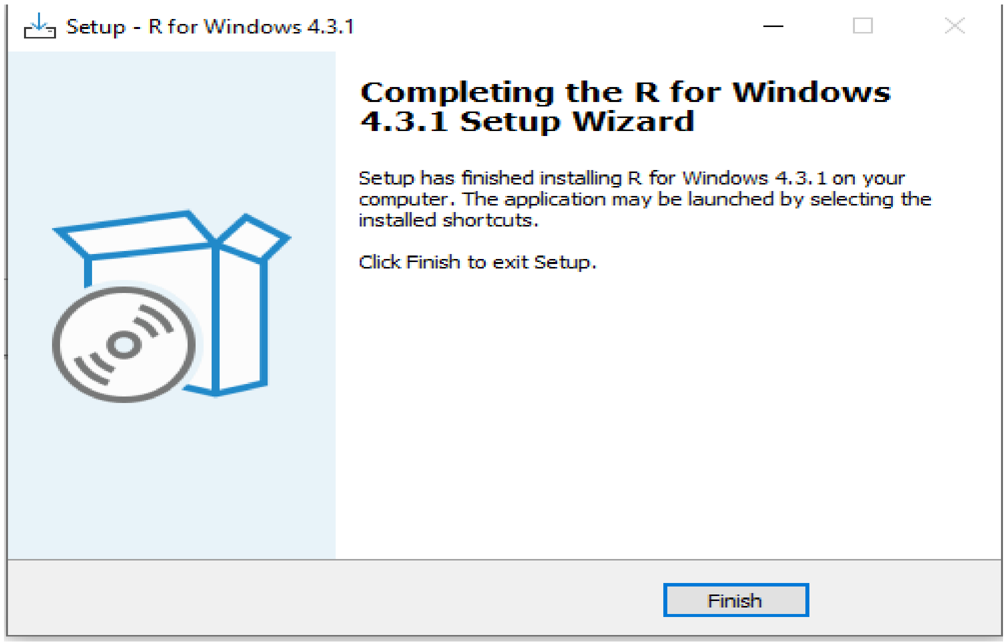
Registry entries:

- ☑ Save version number in registry
- ☑ Associate R with .RData files


Setup - R for Windows 4.3.1

Installing
Please wait while Setup installs R for Windows 4.3.1 on your computer.

Extracting files...
C:\Program Files\R\R-4.3.1\library\MASS\po\pl\LC_MESSAGES\R-MASS.mo


Setup - R for Windows 4.3.1

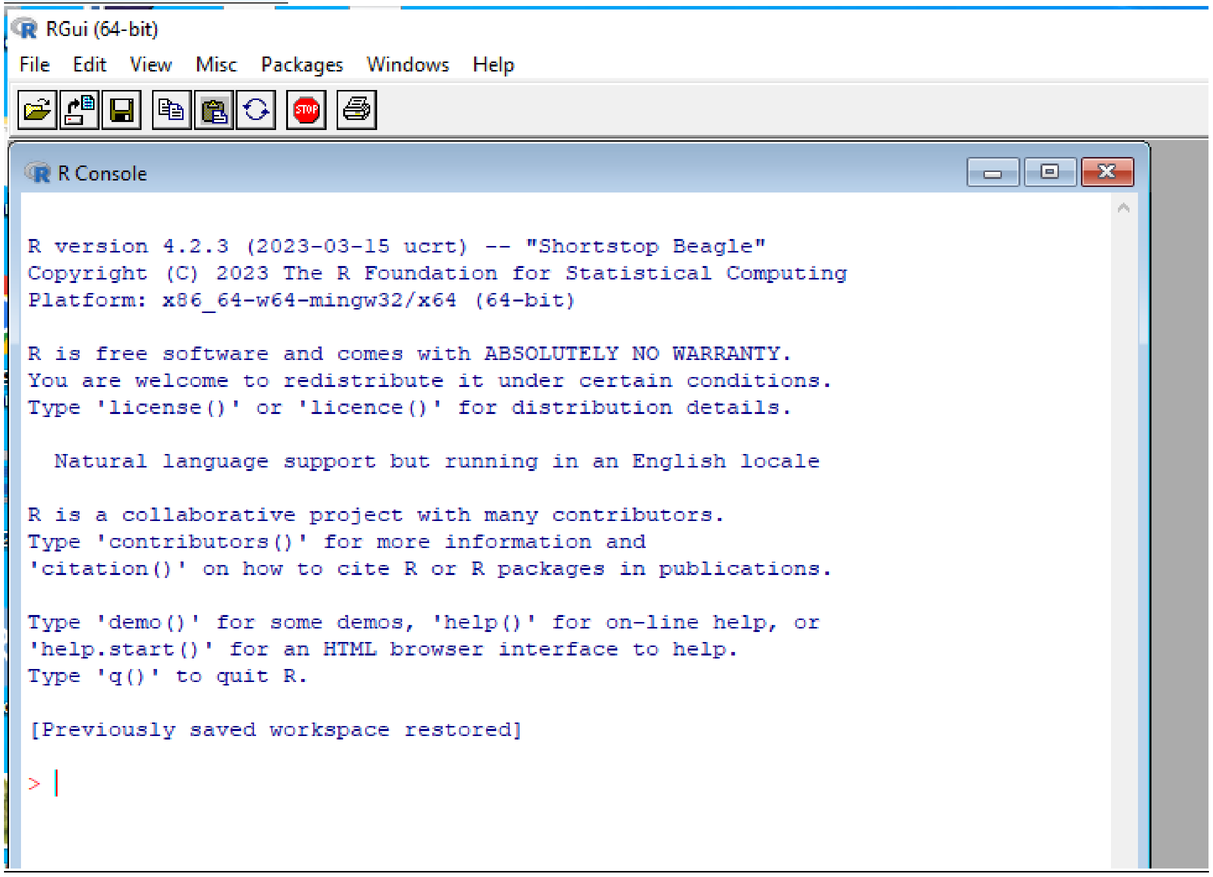
Completing the R for Windows 4.3.1 Setup Wizard

Setup has finished installing R for Windows 4.3.1 on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

Click on Finish

PROG NO: DATE:		PG NO: R NO:



```
R version 4.2.3 (2023-03-15 ucrt) -- "Shortstop Beagle"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

  Natural language support but running in an English locale

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

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

[Previously saved workspace restored]

> |
```

R studio installation:

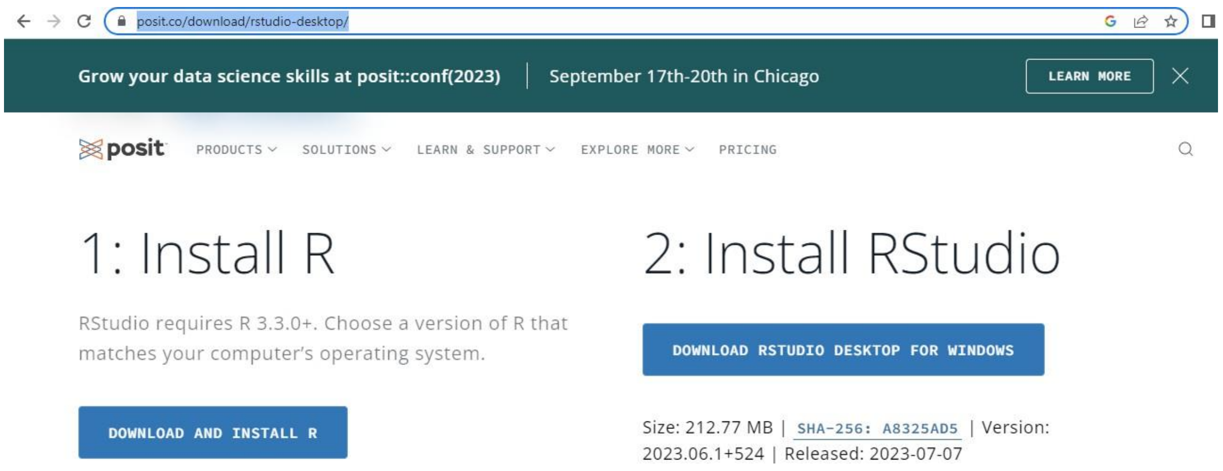
RStudio is a flexible and multifunctional open-source IDE (integrated development environment) that is extensively used as a graphical front-end to work with R of version 3.0.1 or higher. In addition, it's also adapted to many other programming languages, such as Python or SQL.

RStudio offers numerous helpful features:

- A user-friendly interface
- The ability to write and save reusable scripts

- Easy access to all the imported data and created objects (like variables, functions, etc.)
- Exhaustive help on any object
- Code autocompletion
- The ability to create projects to organize and share your work with your collaborators more efficiently
- Plot previewing
- Easy switching between terminal and console
- Operational history tracking
- Plenty of **articles** from RStudio Support on how to use the IDE

https://posit.co/download/rstudio-desktop/


posit.co/download/rstudio-desktop/
Grow your data science skills at posit::conf(2023) | September 17th-20th in Chicago
LEARN MORE
posit
PRODUCTS
SOLUTIONS
LEARN & SUPPORT
EXPLORE MORE
PRICING

1: Install R

RStudio requires R 3.3.0+. Choose a version of R that matches your computer's operating system.

DOWNLOAD AND INSTALL R

2: Install RStudio

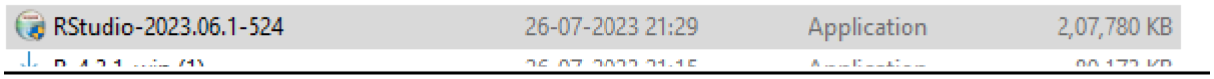
DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 212.77 MB | SHA-256: A8325AD5 | Version: 2023.06.1+524 | Released: 2023-07-07

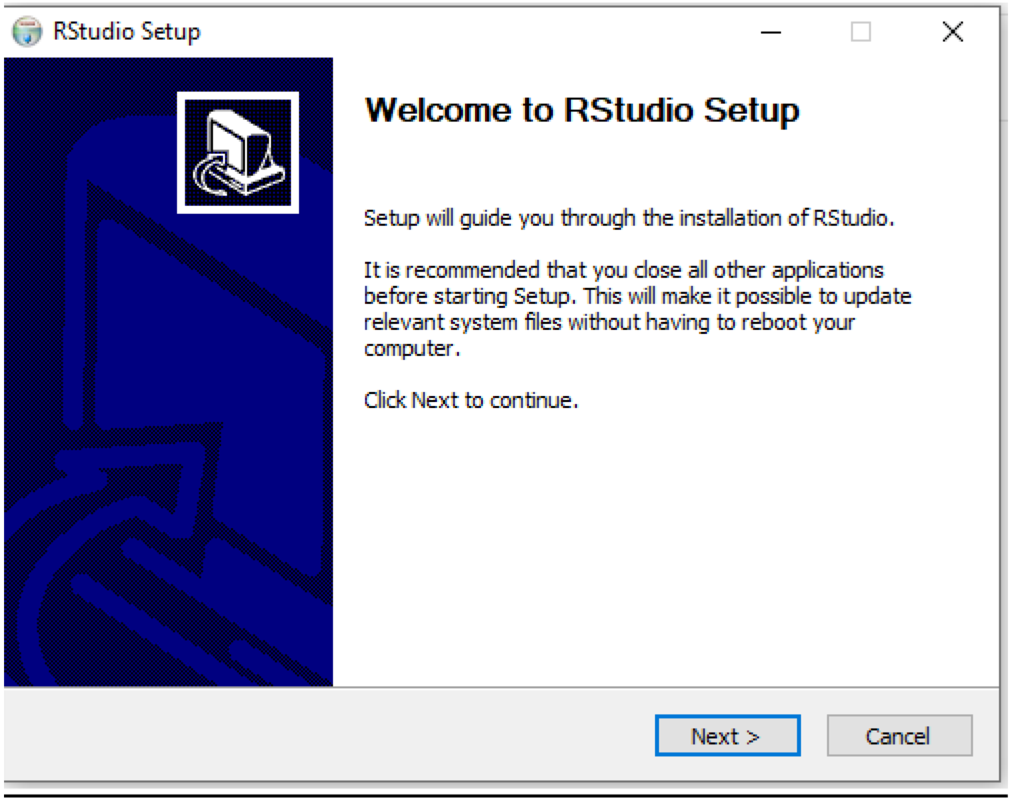
Click on 2:Install RStudio

Download RSTUDIO DESKTOP FOR WINDOWS

PROG NO:		PG NO:
DATE:		R NO:



RStudio-2023.06.1-524 26-07-2023 21:29 Application 2,07,780 KB



RStudio Setup

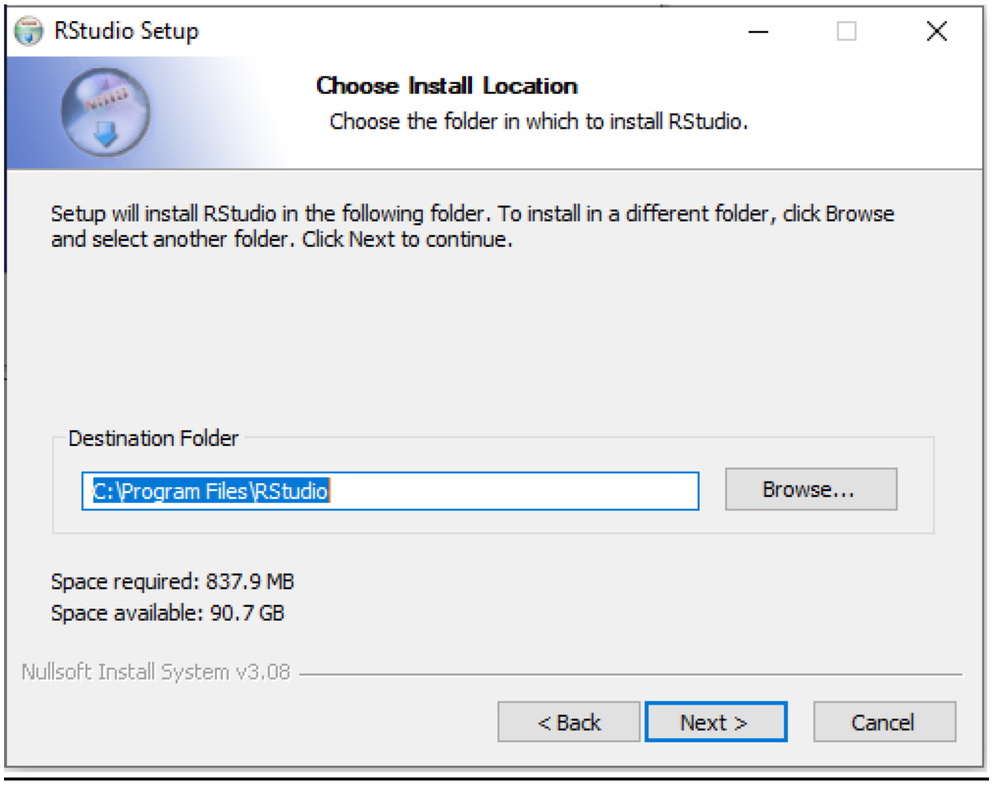
Welcome to RStudio Setup

Setup will guide you through the installation of RStudio.

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

Click Next to continue.

PROG NO:		PG NO:
DATE:		R NO:


RStudio Setup

Choose Install Location

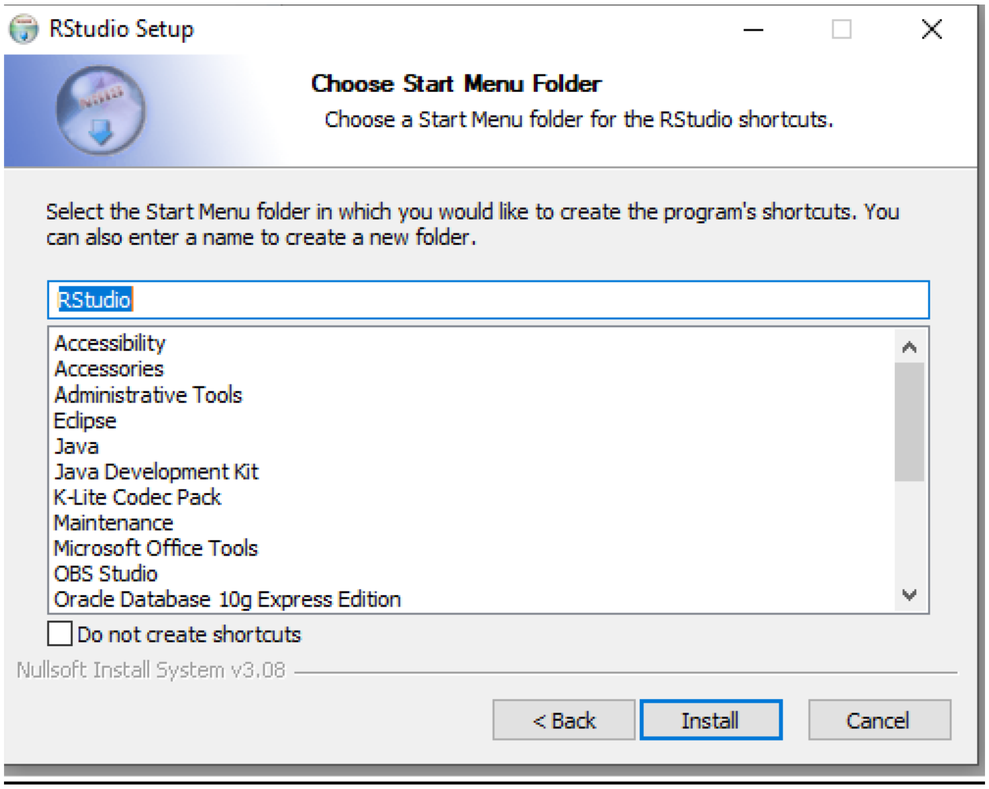
Choose the folder in which to install RStudio.

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

Destination Folder: C:\Program Files\RStudio

Space required: 837.9 MB
Space available: 90.7 GB

Nullsoft Install System v3.08


RStudio Setup

Choose Start Menu Folder
Choose a Start Menu folder for the RStudio shortcuts.

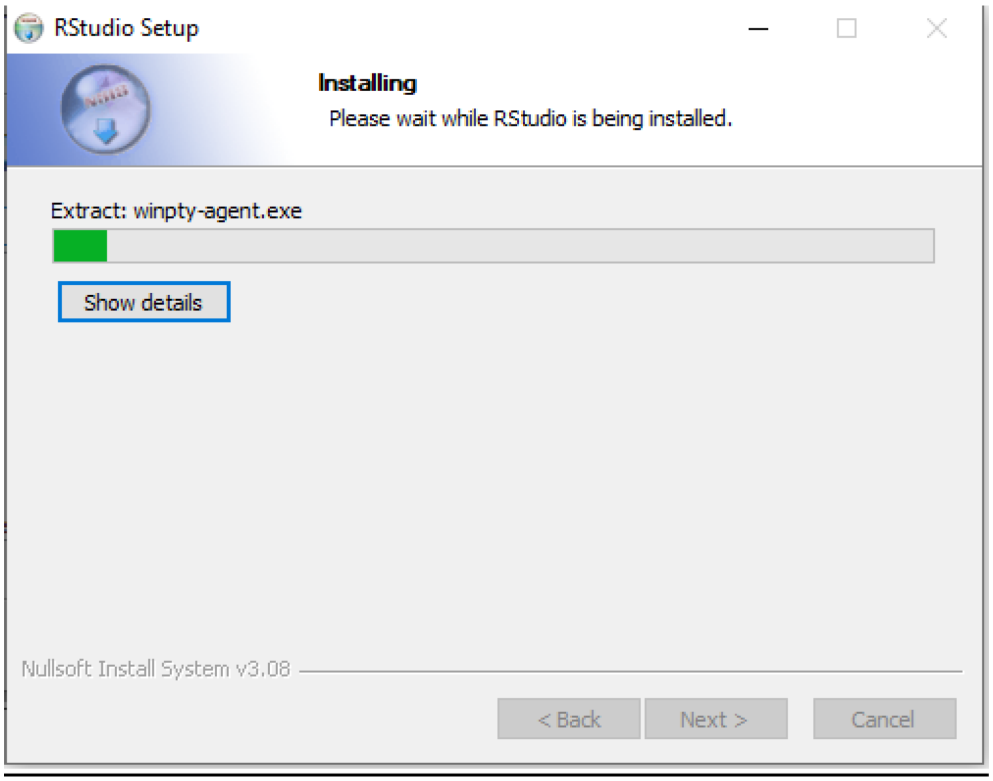
Select the Start Menu folder in which you would like to create the program's shortcuts. You can also enter a name to create a new folder.

RStudio

Accessibility
Accessories
Administrative Tools
Eclipse
Java
Java Development Kit
K-Lite Codec Pack
Maintenance
Microsoft Office Tools
OBS Studio
Oracle Database 10g Express Edition

[ ] Do not create shortcuts

Nullsoft Install System v3.08

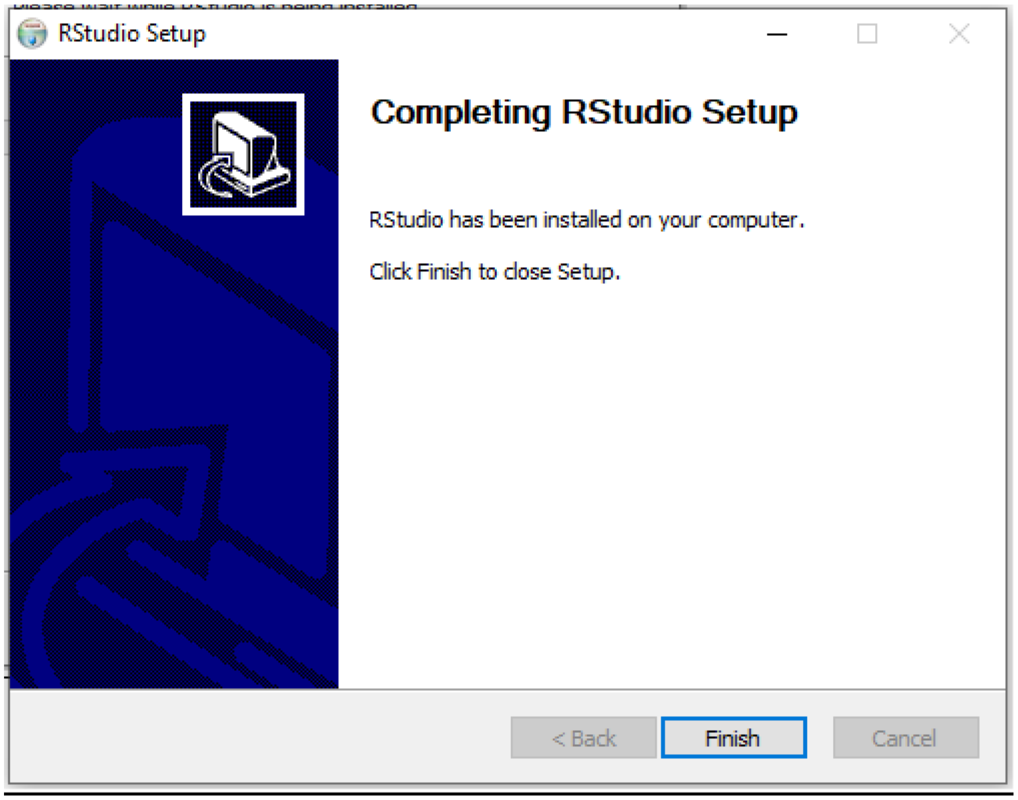


Installing
Please wait while RStudio is being installed.

Extract: winpty-agent.exe

Nullsoft Install System v3.08

PROG NO:	PG NO:
DATE:	R NO:



Completing RStudio Setup

RStudio has been installed on your computer.

Click Finish to close Setup.

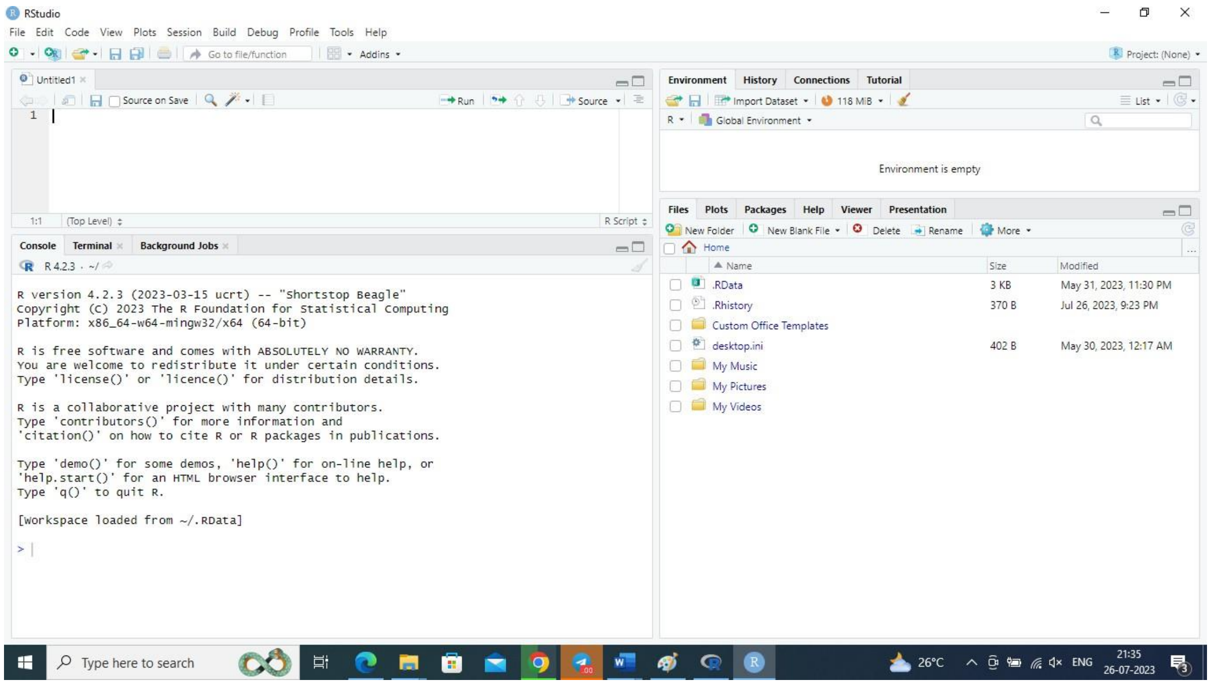
PBR VITS (Autonomous), KAVALI
DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE

PROG NO:

DATE:

PG NO:

R NO:



```
R version 4.2.3 (2023-03-15 ucrt) -- "Shortstop Beagle"
Copyright (C) 2023 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

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

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

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

[workspace loaded from ~/.RData]

> 
```

Name	Size	Modified
.RData	3 KB	May 31, 2023, 11:30 PM
.Rhistory	370 B	Jul 26, 2023, 9:23 PM
Custom Office Templates		
desktop.ini	402 B	May 30, 2023, 12:17 AM
My Music		
My Pictures		
My Videos		

R - Basic Syntax

As a convention, we will start learning R programming by writing a "Hello, World!" program. Depending on the needs, you can program either at R command prompt or you can use an R script file to write your program. Let's check both one by one.

R Command Prompt

Once you have R environment setup, then it's easy to start your R command prompt by just typing the following command at your command prompt −

$ R

This will launch R interpreter and you will get a prompt > where you can start typing your program as follows −

```
> myString <- "Hello, World!"
> print ( myString)
[1] "Hello, World!"
```

Here first statement defines a string variable myString, where we assign a string "Hello, World!" and then next statement print() is being used to print the value stored in variable myString.

Comments

Comments are like helping text in your R program and they are ignored by the interpreter while executing your actual program. Single comment is written using # in the beginning of the statement as follows −

# My first program in R Programming

R does not support multi-line comments but you can perform a trick which is something as follows −

```
if(FALSE) {
  "This is a demo for multi-line comments and it should be put inside either a
     single OR double quote"
}

myString <- "Hello, World!"
print ( myString)
```

```
[1] "Hello, World!"
```

Though above comments will be executed by R interpreter, they will not interfere with your actual program. You should put such comments inside, either single or double quote.

R - Data Types

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

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

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

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Data Type	Example	Verify

Logical	TRUE, FALSE	`v <- TRUE`
		`print(class(v))`
		it produces the following result −
		[1] "logical"

Numeric	12.3, 5, 999	`v <- 23.5`
		`print(class(v))`
		it produces the following result −
		[1] "numeric"

Integer	2L, 34L, 0L	`v <- 2L`
		`print(class(v))`
		it produces the following result −
		`[1] "integer"`

Complex	3 + 2i	`v <- 2+5i`
		`print(class(v))`
		it produces the following result −
		`[1] "complex"`

Character	'a' , '"good", "TRUE", '23.4'	`v <- "TRUE"`
		`print(class(v))`
		it produces the following result −
		`[1] "character"`

Raw	"Hello" is stored as 48 65 6c 6c 6f	`v <- charToRaw("Hello")`
		`print(class(v))`
		it produces the following result −
		`[1] "raw"`

In R programming, the very basic data types are the R-objects called **vectors** which hold elements of different classes as shown above. Please note in R the number of classes is not confined to only the above six types. For example, we can use many atomic vectors and create an array whose class will become array.

Vectors

When you want to create vector with more than one element, you should use **c()** function which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

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

```
[1] "red"    "green"  "yellow"
[1] "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.
list1 <- list(c(2,5,3),21.3,sin)

# Print the list.
print(list1)
```

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

```
[[1]]
[1] 2 5 3

[[2]]
[1] 21.3

[[3]]
function (x)  .Primitive("sin")
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

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

```
     [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('green','yellow'),dim = c(3,3,2))
print(a)
```

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

```
, , 1

     [,1]     [,2]     [,3]
[1,] "green"  "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green"

, , 2

     [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green"
[3,] "yellow" "green"  "yellow"
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor()** function. The **nlevels** functions gives the count of levels.

```
# Create a vector.
apple_colors <- c('green','green','yellow','red','red','red','green')

# Create a factor object.
factor_apple <- factor(apple_colors)

# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

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

```
[1] green  green  yellow red    red    red    green
Levels: green red yellow
[1] 3
```

R - Data Frames:

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

Following are the characteristics of a data frame.

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

Create Data Frame

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
  emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
  salary = c(623.3,515.2,611.0,729.0,843.25),

  start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
    "2015-03-27")),
  stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

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

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

Get the Structure of the Data Frame

The structure of the data frame can be seen by using **str()** function.

```
# Create the data frame.
emp.data <- data.frame(
  emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(emp.data)
```

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

```
'data.frame':   5 obs. of  4 variables:
 $ emp_id    : int  1 2 3 4 5
 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary    : num  623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying **summary()** function.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the summary.
print(summary(emp.data))
```

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

```
     emp_id    emp_name             salary        start_date
 Min.   :1   Length:5           Min.   :515.2   Min.   :2012-01-01
 1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
 Median :3   Mode  :character   Median :623.3   Median :2014-05-11
 Mean   :3                      Mean   :664.4   Mean   :2014-01-14
 3rd Qu.:4                      3rd Qu.:729.0   3rd Qu.:2014-11-15
 Max.   :5                      Max.   :843.2   Max.   :2015-03-27
```

Extract Data from Data Frame

Extract specific column from a data frame using column name.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01","2013-09-23","2014-11-15","2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Extract Specific columns.
result <- data. frame(emp .data$ emp_ name, emp. data$ salary)
print(result)
```

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

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

Extract the first two rows and then all columns

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Extract first two rows.
result  <-  emp.data[1:2,]
print(result)
```

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

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

Extract 3rd and 5th row with 2nd and 4th column

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

               start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)

# Extract 3rd and 5th row with 2nd and 4th column.
result <- emp.data[c(3,5),c(2,4)]
print(result)
```

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

```
  emp_name start_date
3 Michelle 2014-11-15
5     Gary 2015-03-27
```

Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   stringsAsFactors = FALSE
)

# Add the "dept" coulmn.
emp.data$dept <- c("IT","Operations","IT","HR","Finance")
v <- emp.data
print(v)
```

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

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

Add Row

To add more rows permanently to an existing data frame, we need to bring in the new rows in the same structure as the existing data frame and use the **rbind()** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

```
# Create the first data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   dept = c("IT","Operations","IT","HR","Finance"),
   stringsAsFactors = FALSE
)

# Create the second data frame
emp.newdata <-     data.frame(
   emp_id = c (6:8),
   emp_name = c("Rasmi","Pranab","Tusar"),
   salary = c(578.0,722.5,632.8),
   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
   dept = c("IT","Operations","Fianance"),
   stringsAsFactors = FALSE
)

# Bind the two data frames.
emp.finaldata <- rbind(emp.data,emp.newdata)
print(emp.finaldata)
```

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

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

Subsetting :

Subsetting in R is a useful indexing feature for accessing object elements. It can be used to select and filter variables and observations. You can use brackets to select rows and columns from your dataframe.

Subset rows of a data.frame with indices:

Let’s select rows 1 and 3 from df using brackets:

```
df[ c(1, 3), ]

x x2 y z

1 1 7 -0.2707606 6

3 4 10 -1.3473558 7
```

Subset rows of a data.frame:

Let’s select the rows of df where the x column is greater than 5 or

is equal to 2. Without any index for columns, all columns are

returned:

```
df[ df$x > 5 | df$x == 2, ]

x x2 y z

2 2 6 -1.1179372 4

4 10 13 0.4832675 10

5 10 13 0.1523950 5
```

Subset rows of a data.frame:

We can subset both rows and colums at the same time:

```
df[ df$x > 5 | df$x == 2, c("y", "z")]

y z

2 -1.1179372 4

4 0.4832675 10

5 0.1523950 5
```

Subset rows of a data.frame: dplyr

The command in dplyr for subsetting rows is filter. Try

?filter

filter(df, x > 5 | x == 2)

```
x x2 y z
1 2 6 -1.1179372 4
2 10 13 0.4832675 10
3 10 13 0.1523950 5
```

Note, no $ or subsetting is necessary. R "knows" x refers to a column of df.

Subset rows of a data.frame: dplyr

By default, you can separate conditions by commas, and filter assumes these statements are joined by &

filter(df, x > 2 & y < 0)

```
x x2 y z
1 4 10 -1.347356 7
```

filter(df, x > 2, y < 0)

```
x x2 y z
1 4 10 -1.347356 7
```

Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. R language is rich in built-in operators and provides following types of operators.

Types of Operators

We have the following types of operators in R programming −

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

Arithmetic Operators

Following table shows the arithmetic operators supported by R language. The operators act on each element of the vector.

Operator	Description	Example

+	Adds two vectors	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v+t)`
		it produces the following result −
		`[1] 10.0  8.5  10.0`

−	Subtracts second vector from the first	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v-t)`
		it produces the following result −
		`[1] -6.0  2.5  2.0`

*	Multiplies both vectors	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`

		`print(v*t)` it produces the following result − [1] 16.0 16.5 24.0

/	Divide the first vector with the second	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v/t)`
		When we execute the above code, it produces the following result −
		[1] 0.250000 1.833333 1.500000

%%	Give the remainder of the first vector with the second	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v%%t)`
		it produces the following result −
		[1] 2.0 2.5 2.0

%/%	The result of division of first vector with second (quotient)	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v%/%t)`
		it produces the following result −
		[1] 0 1 1

^	The first vector raised to the exponent of second vector	`v <- c( 2,5.5,6)`
		`t <- c(8, 3, 4)`
		`print(v^t)`
		it produces the following result −
		[1] 256.000 166.375 1296.000

Relational Operators

Following table shows the relational operators supported by R language. Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

>	Checks if each element of the first vector is greater than the corresponding element of the second vector.	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v>t)
		it produces the following result −
		[1] FALSE TRUE FALSE FALSE

<	Checks if each element of the first vector is less than the corresponding element of the second vector.	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v < t)
		it produces the following result −
		[1] TRUE FALSE TRUE FALSE

==	Checks if each element of the first vector is equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v == t)
		it produces the following result −
		[1] FALSE FALSE FALSE TRUE

<=	Checks if each element of the first vector is less than or equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v<=t)
		it produces the following result −
		[1] TRUE FALSE TRUE TRUE

>=	Checks if each element of the first vector is greater than or equal to the corresponding element of the second vector.	v <- c(2,5.5,6,9)
		t <- c(8,2.5,14,9)
		print(v>=t)
		it produces the following result −
		[1] FALSE TRUE FALSE TRUE

!=	Checks if each element of the first vector is	v <- c(2,5.5,6,9)

	unequal to the corresponding element of the second vector.	t <- c(8,2.5,14,9)
		print(v!=t)
		it produces the following result –
		[1] TRUE TRUE TRUE FALSE

Logical Operators

Following table shows the logical operators supported by R language. It is applicable only to vectors of type logical, numeric or complex. All numbers greater than 1 are considered as logical value TRUE.Each element of the first vector is compared with the corresponding element of the second vector. The result of comparison is a Boolean value.

Operator	Description	Example

&	It is called Element-wise Logical AND operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if both the elements are TRUE.	v <- c(3,1,TRUE,2+3i)
		t <- c(4,1,FALSE,2+3i)
		print(v&t)
		it produces the following result –
		[1] TRUE TRUE FALSE TRUE

|	It is called Element-wise Logical OR operator. It combines each element of the first vector with the corresponding element of the second vector and gives a output TRUE if one the elements is TRUE.	v <- c(3,0,TRUE,2+2i)
		t <- c(4,0,FALSE,2+3i)
		print(v|t)
		it produces the following result –
		[1] TRUE FALSE TRUE TRUE

!	It is called Logical NOT operator. Takes each element of the vector and gives the opposite logical value.	v <- c(3,0,TRUE,2+2i)
		print(!v)
		it produces the following result –
		[1] FALSE TRUE FALSE FALSE

The logical operator && and || considers only the first element of the vectors and give a vector of single element as output.

Operator	Description	Example

&&	Called Logical AND operator. Takes first element of both the vectors and gives the TRUE only if both are TRUE.	`v <- c(3,0,TRUE,2+2i)`
		`t <- c(1,3,TRUE,2+3i)`
		`print(v&&t)`
		it produces the following result –
		[1] TRUE

||	Called Logical OR operator. Takes first element of both the vectors and gives the TRUE if one of them is TRUE.	`v <- c(0,0,TRUE,2+2i)`
		`t <- c(0,3,TRUE,2+3i)`
		`print(v||t)`
		it produces the following result –[1] FALSE

Assignment Operators

These operators are used to assign values to vectors.

Operator	Description	Example

<−
or
=
or
<<−
	Called Left Assignment	`v1 <- c(3,1,TRUE,2+3i)`
		`v2 <<- c(3,1,TRUE,2+3i)`
		`v3 = c(3,1,TRUE,2+3i)`
		`print(v1)`
		`print(v2)`
		`print(v3)`
		it produces the following result –
		[1] 3+0i 1+0i 1+0i 2+3i
		[1] 3+0i 1+0i 1+0i 2+3i
		[1] 3+0i 1+0i 1+0i 2+3i

->
or
->>
	Called Right Assignment	`c(3,1,TRUE,2+3i) -> v1`
		`c(3,1,TRUE,2+3i) ->> v2`
		`print(v1)`
		`print(v2)`

		it produces the following result − [1] 3+0i 1+0i 1+0i 2+3i [1] 3+0i 1+0i 1+0i 2+3i

Miscellaneous Operators

These operators are used to for specific purpose and not general mathematical or logical computation.

Operator	Description	Example

:	Colon operator. It creates the series of numbers in sequence for a vector.	`v <- 2:8`
		`print(v)`
		it produces the following result −
		`[1] 2 3 4 5 6 7 8`

%in%	This operator is used to identify if an element belongs to a vector.	`v1 <- 8`
		`v2 <- 12`
		`t <- 1:10`
		`print(v1 %in% t)`
		`print(v2 %in% t)`
		it produces the following result −
		`[1] TRUE`
		`[1] FALSE`

%*%	This operator is used to multiply a matrix with its transpose.	`M = matrix( c(2,6,5,1,10,4), nrow = 2,ncol = 3,byrow = TRUE)`
		`t = M %*% t(M)`
		`print(t)`
		it produces the following result −
		`     [,1] [,2]`
		`[1,]   65   82`
		`[2,]   82  117`

R - Variables

A variable provides us with named storage that our programs can manipulate. A variable in R can store an atomic vector, group of atomic vectors or a combination of many Robjects. A valid variable name consists of letters, numbers and the dot or underline characters. The variable name starts with a letter or the dot not followed by a number.

Variable Name	Validity	Reason
var_name2.	valid	Has letters, numbers, dot and underscore
var_name%	Invalid	Has the character '%'. Only dot(.) and underscore allowed.
2var_name	invalid	Starts with a number
.var_name, var.name	valid	Can start with a dot(.) but the dot(.)should not be followed by a number.
.2var_name	invalid	The starting dot is followed by a number making it invalid.
_var_name	invalid	Starts with _ which is not valid

Variable Assignment

The variables can be assigned values using leftward, rightward and equal to operato r. The values of the variables can be printed using **print()** or **cat()** function. The **cat()** function co mbines multiple items into a continuous print output.

```
# Assignment using equal operator.
var.1 = c(0,1,2,3)

# Assignment using leftward operator.
var.2 <- c("learn","R")

# Assignment using rightward operator.
c(TRUE,1) -> var.3

print(var.1)
```

```
cat ("var.1 is ", var.1 ,"\n")
cat ("var.2 is ", var.2 ,"\n")
cat ("var.3 is ", var.3 ,"\n")
```

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

```
[1] 0 1 2 3
var.1 is  0 1 2 3
var.2 is  learn R
var.3 is  1 1
```

Note − The vector c(TRUE,1) has a mix of logical and numeric class. So logical class is coerced to numeric class making TRUE as 1.

Data Type of a Variable

In R, a variable itself is not declared of any data type, rather it gets the data type of the R - object assigned to it. So R is called a dynamically typed language, which means that we can change a variable's data type of the same variable again and again when using it in a program.

```
var_x <- "Hello"
cat("The class of var_x is ",class(var_x),"\n")

var_x <- 34.5
cat("  Now the class of var_x is ",class(var_x),"\n")

var_x <- 27L
cat("   Next the class of var_x becomes ",class(var_x),"\n")
```

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

```
The class of var_x is  character
   Now the class of var_x is  numeric
      Next the class of var_x becomes  integer
```

Finding Variables

To know all the variables currently available in the workspace we use the **ls()** function. Also the ls() function can use patterns to match the variable names.

```
print(ls())
```

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

```
[1] "my var"     "my_new_var" "my_var"     "var.1"
[5] "var.2"      "var.3"      "var.name"   "var_name2."
[9] "var_x"      "varname"
```

Note − It is a sample output depending on what variables are declared in your environment.

The ls() function can use patterns to match the variable names.

```
# List the variables starting with the pattern "var".
print(ls(pattern = "var"))
```

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

```
[1] "my var"     "my_new_var" "my_var"     "var.1"
[5] "var.2"      "var.3"      "var.name"   "var_name2."
[9] "var_x"      "varname"
```

The variables starting with **dot(.)** are hidden, they can be listed using "all.names = TRUE" argument to ls() function.

```
print(ls(all.name = TRUE))
```

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

```
[1] ".cars"        ".Random.seed" ".var_name"    ".varname"     ".varname2"
[6] "my var"       "my_new_var"   "my_var"       "var.1"        "var.2"
[11]"var.3"        "var.name"     "var_name2."   "var_x"
```

Deleting Variables

Variables can be deleted by using the **rm()** function. Below we delete the variable var.3. On printing the value of the variable error is thrown.

```
rm(var.3)
print(var.3)
```

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

```
[1] "var.3"
Error in print(var.3) : object 'var.3' not found
```

All the variables can be deleted by using the **rm()** and **ls()** function together.

```
rm(list = ls())
print(ls())
```

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

```
character(0)
```

PROG NO:		PG NO:
DATE:		R NO:

Viewing Data in R

There are many ways to view data in R. A few of the few common methods are detailed below.

Viewing defined objects via the list function

The list (ls) function returns all of the defined objects (data.frames, vectors, constants, etc) in the current workspace. It does not provide any information regarding the structure or contents of the objects.

```
ls()
```

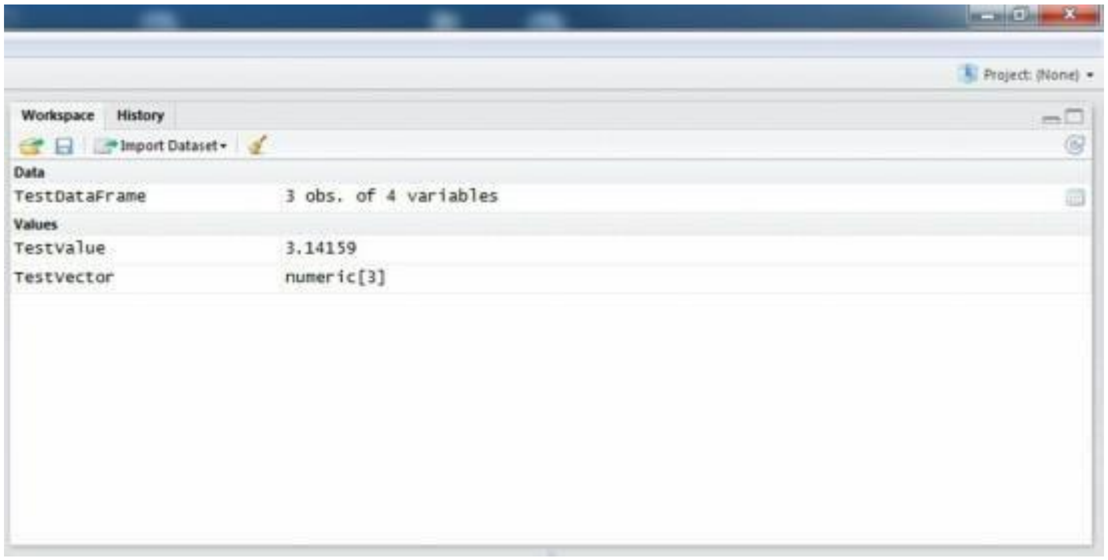
EXAMPLE:

```
> ls()
[1] "TestDataFrame" "TestValue"     "TestVector"
```

Viewing defined objects in RStudio

In RStudio, the workspace tab in the Top-Right frame lists all defined objects (along with their structure). The image below in shows an example where the following objects are shown:

- TestDataFrame = Dataframe with 4 columns and 3 rows
- TestValue = A constant value equal to 3.14159
- TestVector = A numeric vector with 3 entries.



```
Data
TestDataFrame        3 obs. of 4 variables
Values
TestValue            3.14159
TestVector           numeric[3]
```

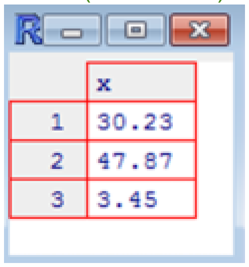
Viewing contents of a defined object

The contents of a defined object can be viewed via the view function. The object contents will be shown in a new window.

```
View(ObjectName)
```

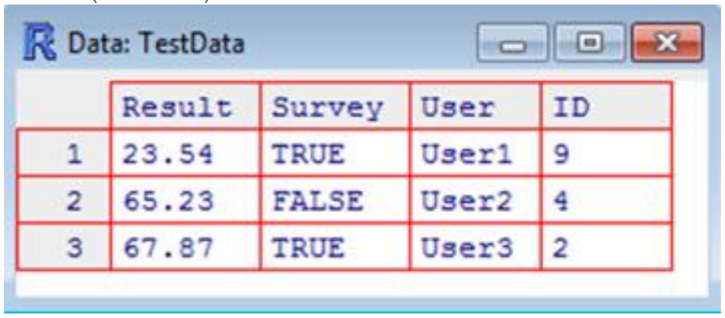
EXAMPLE:

```
> View(TestVector)
```



	x
1	30.23
2	47.87
3	3.45

```
> View (TestData)
```



Data: TestData

	Result	Survey	User	ID
1	23.54	TRUE	User1	9
2	65.23	FALSE	User2	4
3	67.87	TRUE	User3	2

Viewing the mode of an object

The mode of an object provides information regarding what type of data is contained within the object. The mode of an object can be viewed using the mode function.

```
mode(ObjectName)
```

EXAMPLE:

```
> mode(TestVector)
[1] "numeric"


>mode(TestData)
[1] "list"
```

Viewing the class of an object

For simple vectors the class will be the same as the mode. However, there are other possible values for "matrix", "array", "factor" and "data.frame" objects. The class of an object can be viewed with the class function.

```
class(ObjectName)
```

EXAMPLE:

```
> class(TestVector)
[1] "numeric"
```

```
>class(TestData)
[1] "data.frame"
```

Viewing the length of an object

The length of an object can be viewed using the length function. The length function returns the number of entries for a vector, and the number of variables for a data frame.

```
length(ObjectName)
```

EXAMPLE:

```
> length(TestVector)
[1] 3

>length(TestData)
[1] 4
```

Data Manipulation in R with data.table

Data manipulation is a crucial step in the data analysis process, as it allows us to prepare and organize our data in a way that is suitable for the specific analysis or visualization. There are many different tools and techniques for data manipulation, depending on the type and structure of the data, as well as the specific goals of the manipulation.

The **data.table** package is an **R** package that provides an enhanced version of the **data.frame** class in **R.** It's syntax and features make it easier and faster to manipulate and work with large datasets.

The **date.table** is one of the most downloaded packages by developers and an ideal choice for Data Scientists.

Installating data.table package

Installing data.table package is as simple as installing other packages. You can use the below commands in **CRAN's** command line tool to install this package −

Installing 'data.table' package using CRAN

```
install.packages('data.table')
```

Installing dev version from Gitlab

```
install.packages("data.table",
repos="https://Rdatatable.gitlab.io/data.table")
```

Importing Datasets

In R programming language, we have tons of built-in datasets that one may use as demo data to demonstrate how the R functions work.

One such popular inbuilt dataset is **“Iris”** dataset. This dataset provides us the measurement of four different attributes of 50 flowers (three different species).

The way we deal with datasets in **data.table** is quite different from dealing datasets in **data.frame.** Let’s go deep into this and get some insights.

The data.table provides us fread() function (fast read) which is basically data.table’s version of read.csv() function. Similar to read.csv() function it can read a file stored locally as well as capable enough to read files hosted on a website.

Example

Consider the below program that imports iris data stored as a CSV file on the internet

```
#  Importing  library

library(data.table)

# Creating a dataset

myDataset                                                                                    <-
fread("https://raw.githubusercontent.com/gexijin/learnR/master/datasets/iris.csv")

# print the iris dataset

print(myDataset)
```

Output

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

As you see from the above output, the imported data is directly stored as a data.table.

The data.table generally inherits from a data.frame class and therefore is a data.frame by itself. Therefore, those functions that accept a data.frame will get the job done for data.table as well.

Displaying IRIS Dataset
Example

```
# Importing library

library(data.table)

# Creating a dataset

myDataset <- fread(
"https://raw.githubusercontent.com/gexijin/learnR/master/datasets/iris.cs
v")
# print the iris dataset
print(myDataset)
```

Output

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
  1:       5.1         3.5          1.4         0.2    setosa
  2:       4.9         3.0          1.4         0.2    setosa
  3:       4.7         3.2          1.3         0.2    setosa
  4:       4.6         3.1          1.5         0.2    setosa
  5:       5.0         3.6          1.4         0.2    setosa
 ---
146:       6.7         3.0          5.2         2.3 virginica
147:       6.3         2.5          5.0         1.9 virginica
148:       6.5         3.0          5.2         2.0 virginica
149:       6.2         3.4          5.4         2.3 virginica
150:       5.9         3.0          5.1         1.8 virginica
```

There are 150 rows and 5 columns in the Iris data set.

Let’s print first six rows from the iris dataset

```
head(myDataset)
```

Output

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1:       5.1         3.5          1.4         0.2  setosa
2:       4.9         3.0          1.4         0.2  setosa
3:       4.7         3.2          1.3         0.2  setosa
4:       4.6         3.1          1.5         0.2  setosa
```

```
5:      5.0      3.6      1.4      0.2  setosa
6:      5.4      3.9      1.7      0.4  setosa
```

R Plotting

Plot

The `plot()` function is used to draw points (markers) in a diagram.

The function takes parameters for specifying points in the diagram.

Parameter 1 specifies points on the **x-axis**.

Parameter 2 specifies points on the **y-axis**.

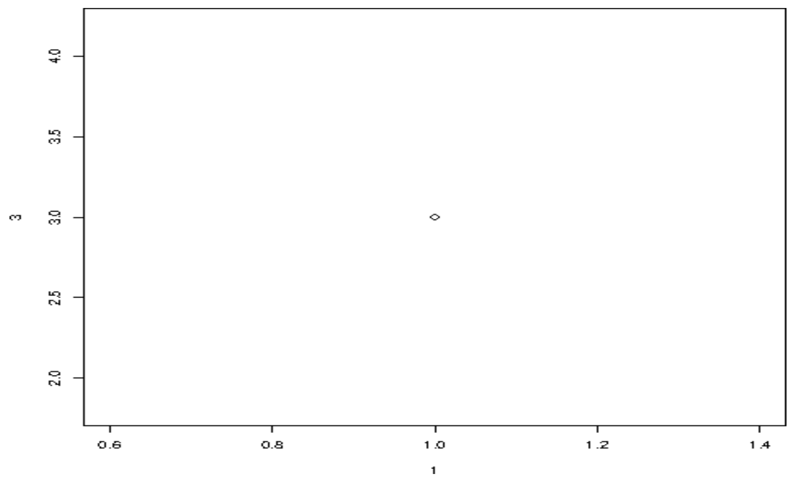
At its simplest, you can use the `plot()` function to plot two numbers against each other:

Example

Draw one point in the diagram, at position (1) and position (3):

```
plot(1, 3)
```

Result:



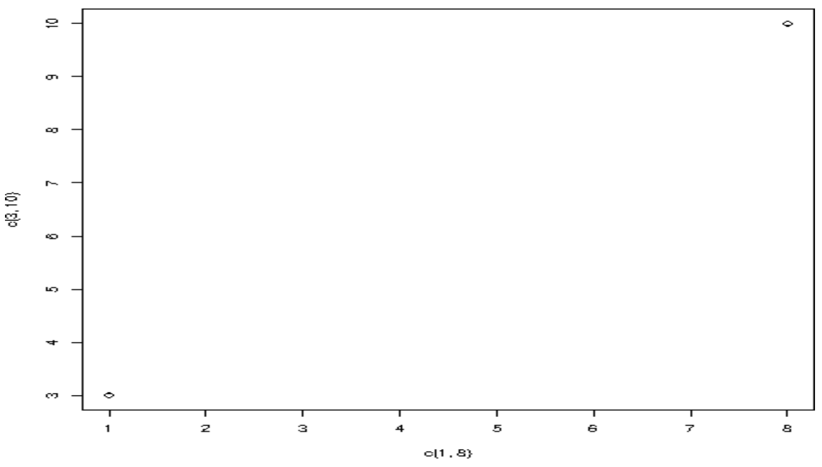
To draw more points, use vectors:

Example

Draw two points in the diagram, one at position (1, 3) and one in position (8, 10):

```
plot(c(1, 8), c(3, 10))
```

Result:



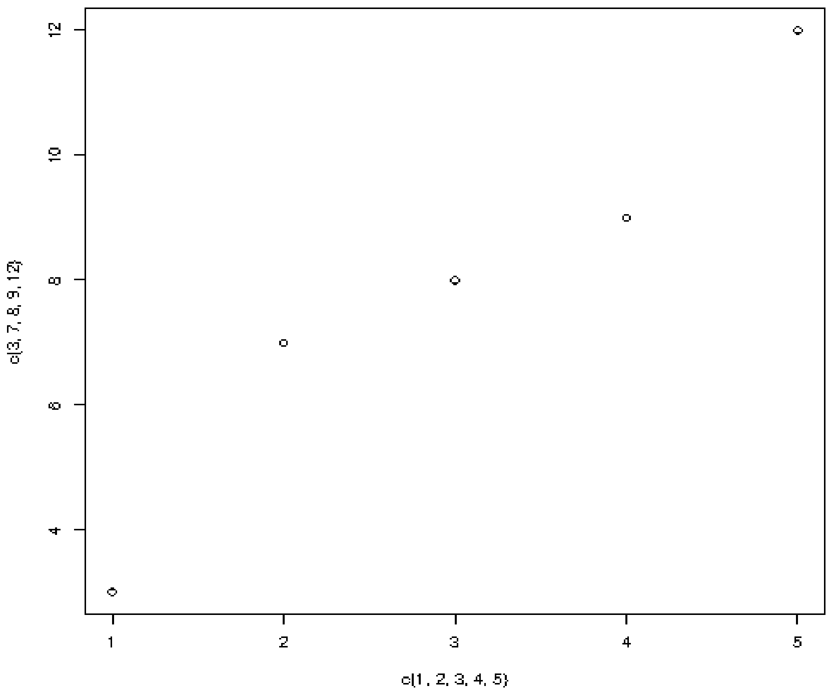
Multiple Points

You can plot as many points as you like, just make sure you have the same number of points in both axis:

Example

```
plot(c(1, 2, 3, 4, 5), c(3, 7, 8, 9, 12))
```

Result:



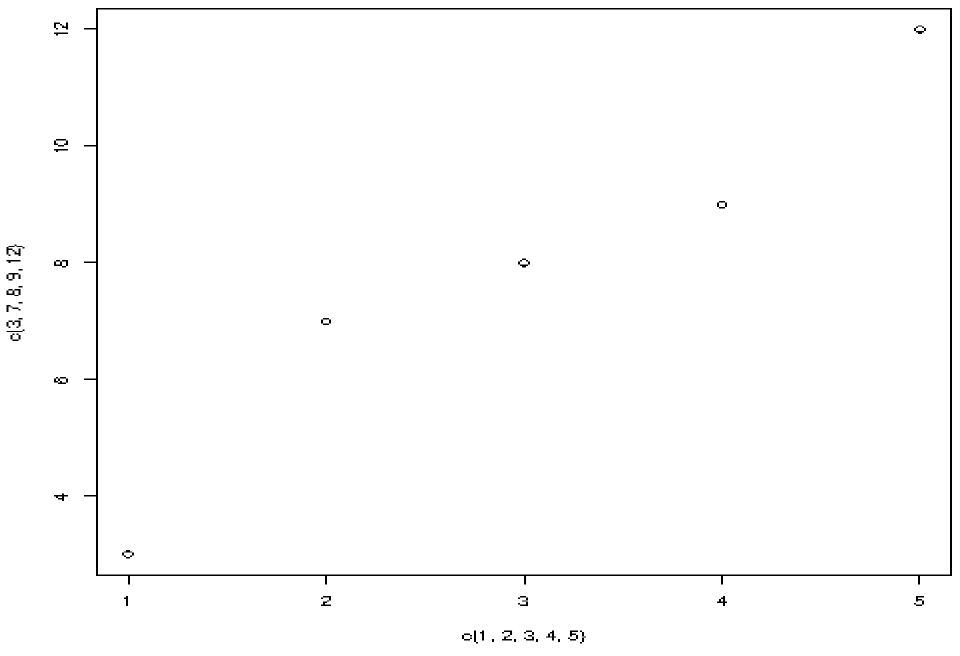
PROG NO:		PG NO:
DATE:		R NO:

For better organization, when you have many values, it is better to use variables:

Example

```
x <- c(1, 2, 3, 4, 5)
y <- c(3, 7, 8, 9, 12)

plot(x, y)
```

Result:



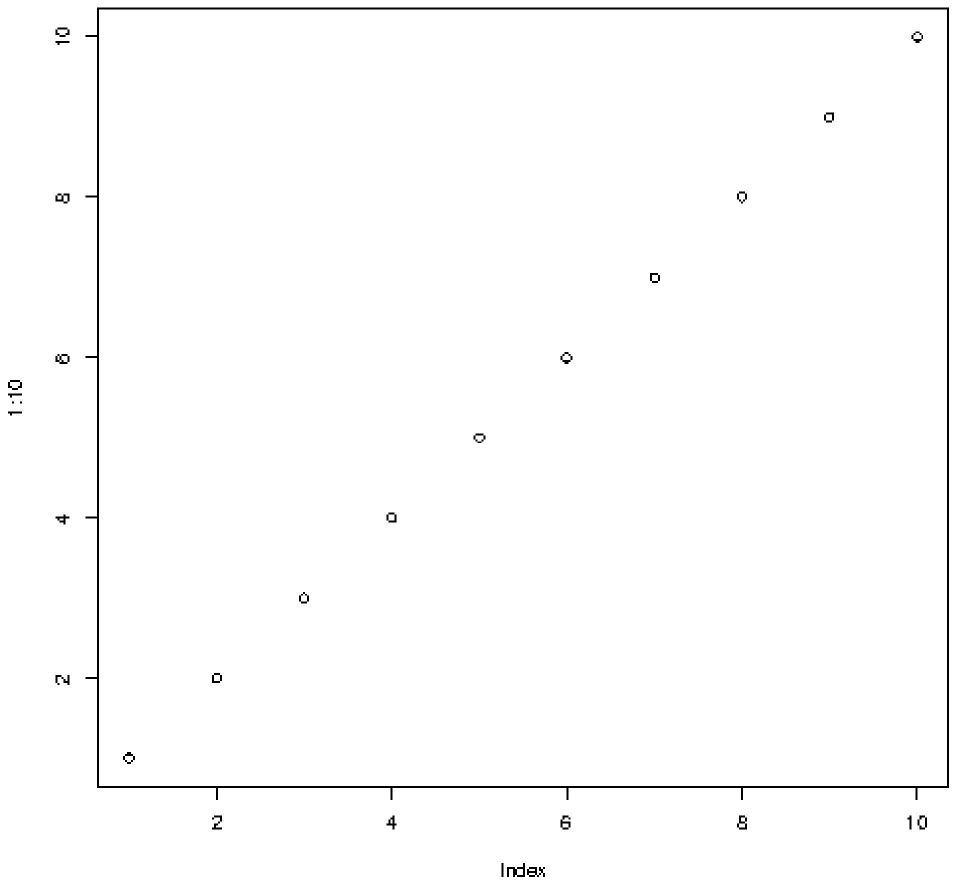
Sequences of Points

If you want to draw dots in a sequence, on both the **x-axis** and the **y-axis**, use the `:` operator:

Example

```
plot(1:10)
```

Result:



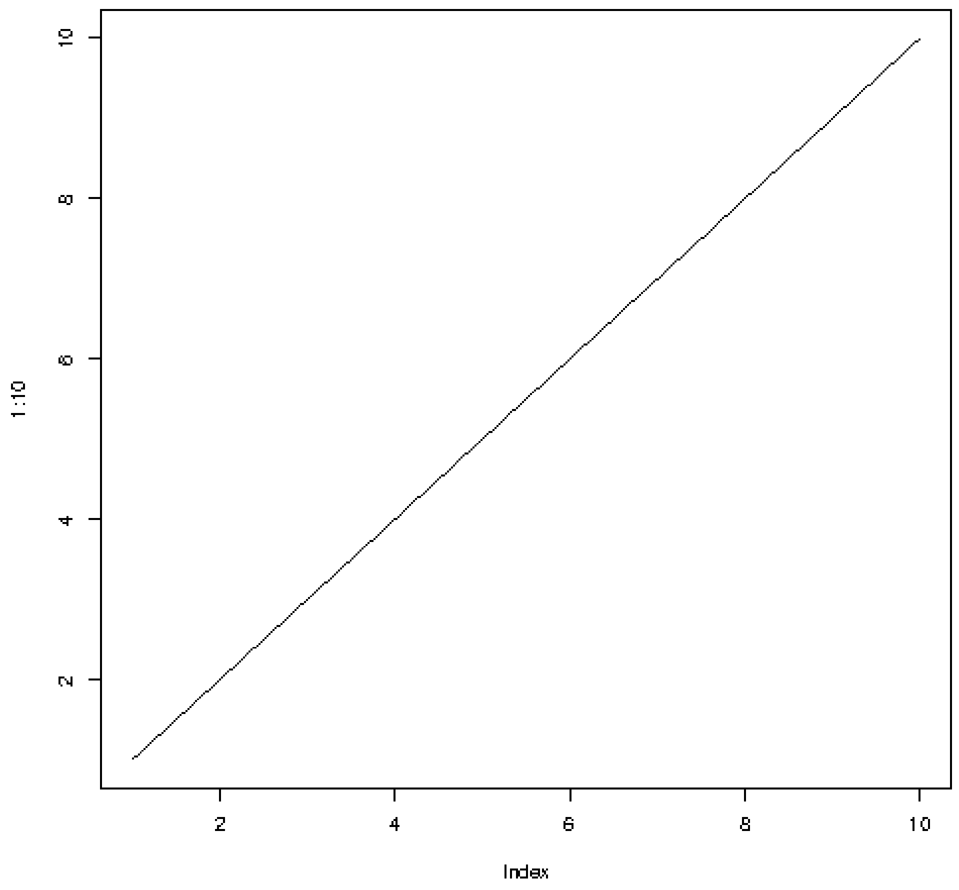
Draw a Line

The `plot()` function also takes a `type` parameter with the value `l` to draw a line to connect all the points in the diagram:

Example

```
plot(1:10, type="l")
```

Result:



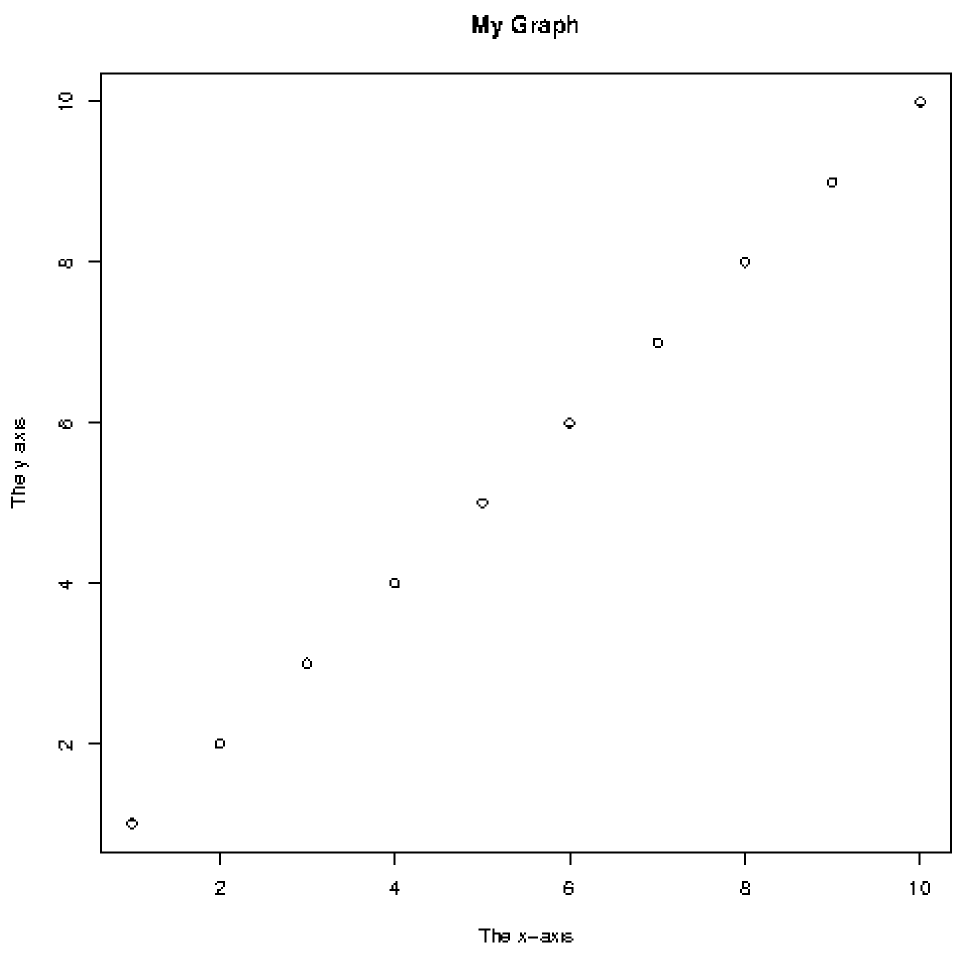
Plot Labels

The `plot()` function also accept other parameters, such as `main`, `xlab` and `ylab` if you want to customize the graph with a main title and different labels for the x and y-axis:

Example

```
plot(1:10, main="My Graph", xlab="The x-axis", ylab="The y axis")
```

Result:



Graph Appearance

There are many other parameters you can use to change the appearance of the points.

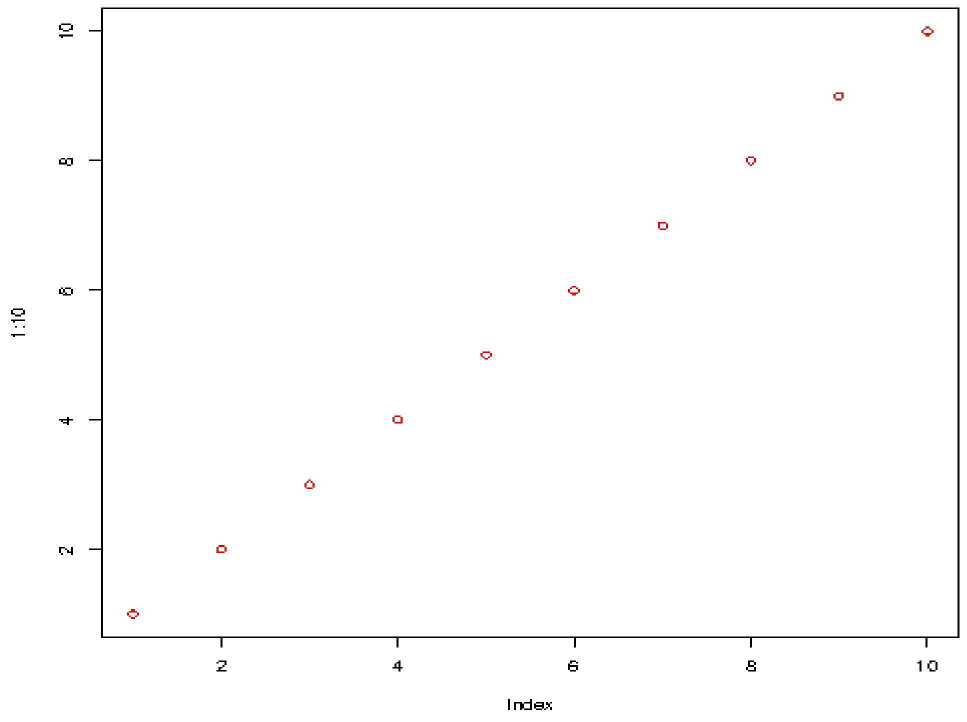
Colors

Use `col="color"` to add a color to the points:

Example

```
plot(1:10, col="red")
```

Result:



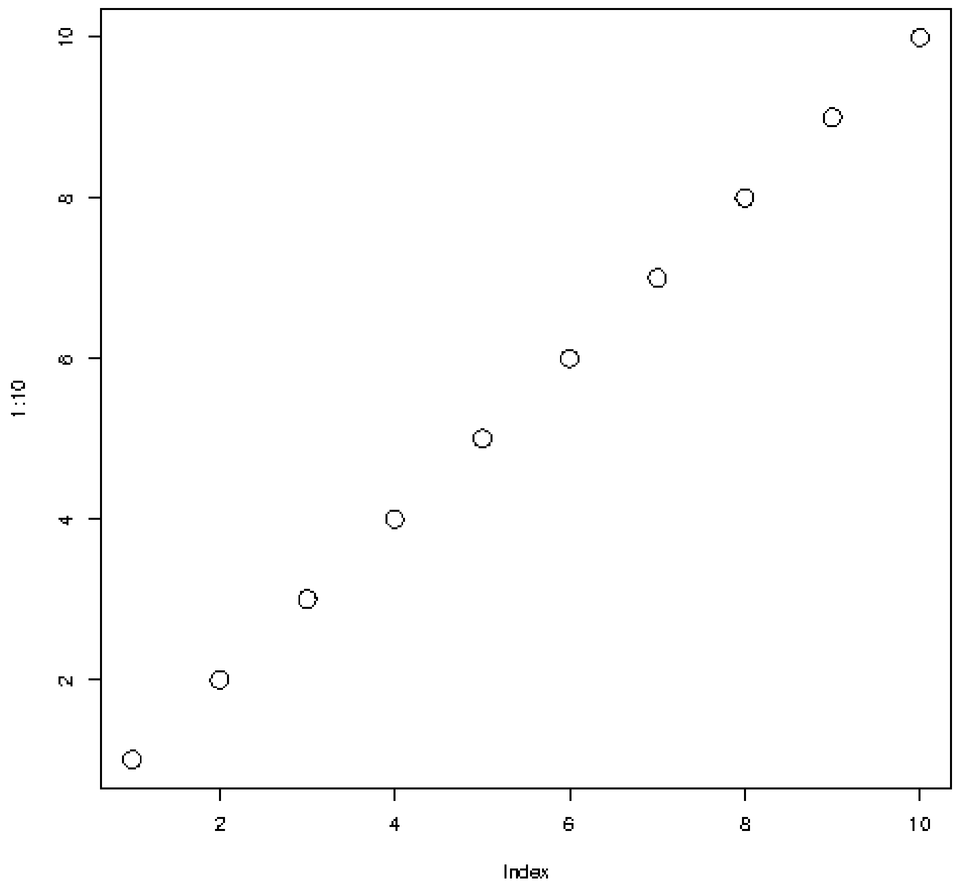
Size

Use `cex=number` to change the size of the points (`1` is default, while `0.5` means 50% smaller, and `2` means 100% larger):

Example

```
plot(1:10, cex=2)
```

Result:



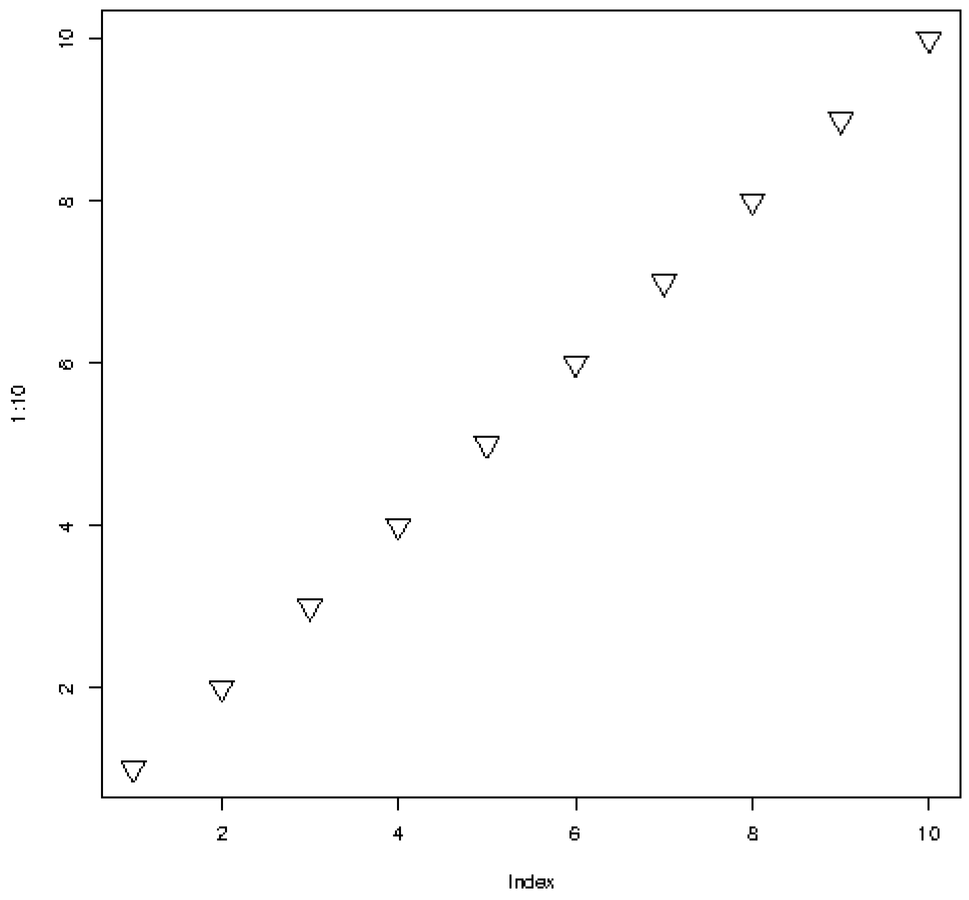
Point Shape

Use `pch` with a value from 0 to 25 to change the point shape format:

Example

```
plot(1:10, pch=25, cex=2)
```

Result:



The values of the `pch` parameter ranges from 0 to 25, which means that we can choose up to 26 different types of point shapes:

R - CSV Files

In R, 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.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

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.

```
# Get and print current working directory.
print(getwd())

# Set current working directory.
setwd("/web/com")

# Get and print current working directory.
++-
print(getwd())
```

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

```
[1] "/web/com/1441086124_2016"
[1] "/web/com"
```

This result depends on your OS and your current directory where you are working.

file.create("C:/Users/LAB10/Documents/R_csv files/CSV/input5.csv")

Input as CSV File

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**.

You can create this file using windows notepad by copying and pasting this data. Save the file as **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
5,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

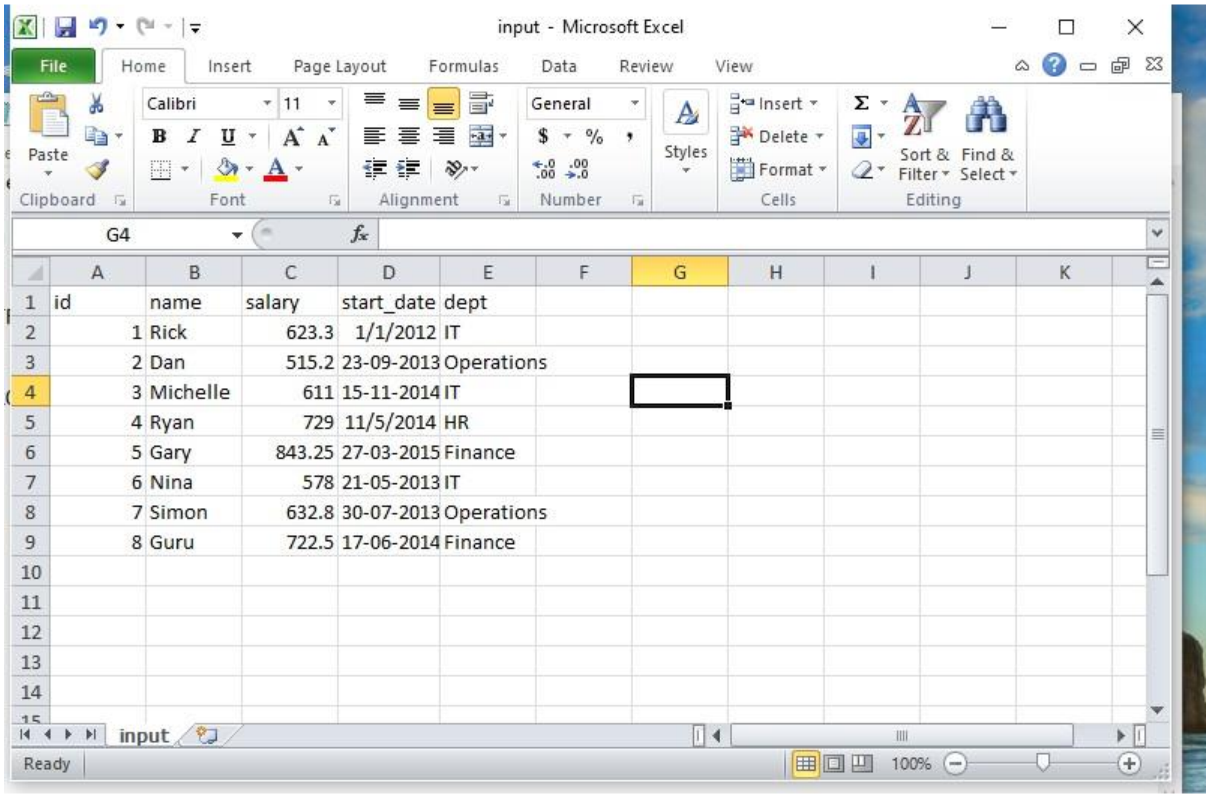
Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory −

```
data <- read.csv("input.csv")
print(data)
```

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

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

PROG NO:	PG NO:
DATE:	R NO:



	A	B	C	D	E
1	id	name	salary	start_date	dept
2	1	Rick	623.3	1/1/2012	IT
3	2	Dan	515.2	23-09-2013	Operations
4	3	Michelle	611	15-11-2014	IT
5	4	Ryan	729	11/5/2014	HR
6	5	Gary	843.25	27-03-2015	Finance
7	6	Nina	578	21-05-2013	IT
8	7	Simon	632.8	30-07-2013	Operations
9	8	Guru	722.5	17-06-2014	Finance

Analyzing the CSV File

By default the **read.csv()** function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```
data <- read.csv("input.csv")

print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

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

```
[1] TRUE
[1] 5
[1] 8
```

Once we read data in a data frame, we can apply all the functions applicable to data frames as explained in subsequent section.

Get the maximum salary

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)
```

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

```
[1] 843.25
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)

# Get the person detail having max salary.
retval <- subset(data, salary == max(salary))
print(retval)
```

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

```
     id    name  salary  start_date    dept
5    NA    Gary  843.25  2015-03-27    Finance
```

Get all the people working in IT department

```
# Create a data frame.
data <- read.csv("input.csv")

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

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

```
   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
```

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

```
# Create a data frame.
data <- read.csv("input.csv")

info <- subset(data, salary > 600 & dept == "IT")
print(info)
```

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

```
   id  name      salary  start_date  dept
1   1  Rick      623.3   2012-01-01  IT
3   3  Michelle  611.0   2014-11-15  IT
```

Get the people who joined on or after 2014

```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))
print(retval)
```

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

```
   id  name     salary  start_date  dept
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
8   8  Guru     722.50  2014-06-17  Finance
```

Writing into a CSV File

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

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```

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

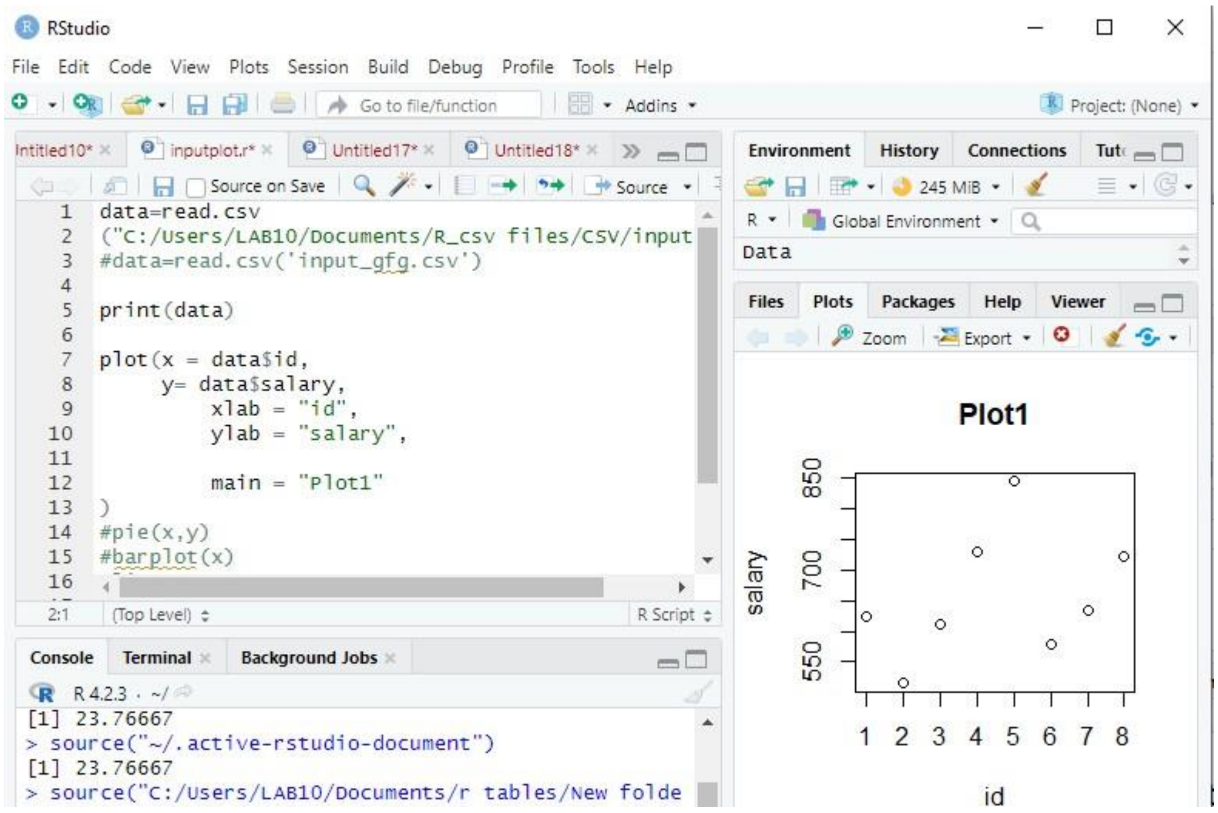
```
  X    id  name      salary  start_date    dept
1 3     3  Michelle  611.00  2014-11-15    IT
2 4     4  Ryan      729.00  2014-05-11    HR
3 5    NA  Gary      843.25  2015-03-27    Finance
4 8     8  Guru      722.50  2014-06-17    Finance
```

Here the column X comes from the data set newper. This can be dropped using additional parameters while writing the file.

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv", row.names = FALSE)
newdata <- read.csv("output.csv")
print(newdata)
```

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

```
    id   name      salary  start_date    dept
1    3   Michelle  611.00  2014-11-15    IT
2    4   Ryan      729.00  2014-05-11    HR
3   NA   Gary      843.25  2015-03-27    Finance
4    8   Guru      722.50  2014-06-17    Finance
```


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
Untitled10* inputplot.r* Untitled17* Untitled18*
Source on Save
Source
Environment History Connections
245 MiB
Global Environment
Files Plots Packages Help Viewer
Zoom Export
Plot1
salary
id
Console Terminal Background Jobs
R 4.2.3 · ~/
2:1 (Top Level) R Script

```
data=read.csv
("C:/Users/LAB10/Documents/R_csv files/CSV/input
#data=read.csv('input_gfg.csv')

print(data)

plot(x = data$id,
      y= data$salary,
          xlab = "id",
          ylab = "salary",

          main = "Plot1"
)
#pie(x,y)
#barplot(x)
```

```
Data
```

```
[1] 23.76667
> source("~/.active-rstudio-document")
[1] 23.76667
> source("C:/Users/LAB10/Documents/r tables/New folde
```

Working with CSV files in R Programming

CSV files are basically the text files wherein the values of each row are separated by a delimiter, as in a comma or a tab. In this article, we will use the following sample CSV file:

sample.csv

```
id, name, department, salary, projects

1,   A,     IT,         60754,   4

2,   B,     Tech,       59640,   2

3,   C,     Marketing, 69040,   8

4,   D,     Marketing, 65043,   5

5,   E,     Tech,       59943,   2
```

```
6,    F,      IT,         65000,    5
7,    G,      HR,         69000,    7
```

Reading a CSV file

The contents of a CSV file can be read as a data frame in R using the read.csv(…) function. The CSV file to be read should be either present in the current working directory or the directory should be set accordingly using the setwd(…) command in R. The CSV file can also be read from a URL using **`read.csv()`** function.

Examples:

```
csv_data <- read.csv(file = 'sample.csv')
print(csv_data)
  # print number of columns
print (ncol(csv_data))
  # print number of rows
print(nrow(csv_data))
```

Output:

```
      id,    name,     department,     salary,      projects
      1       A          HR           60754         14
      2       B         Tech          59640         3
      3       C       Marketing       69040         8
      4       D          HR           65043         5
      5       E         Tech          59943         2
      6       F          IT           65000         5
      7       G          HR           69000         7
 [1] 4
 [1] 7
```

The header is by default set to a TRUE value in the function. The head is not included in the count of rows, therefore this CSV has 7 rows and 4 columns.

Querying with CSV files

SQL queries can be performed on the CSV content, and the corresponding result can be retrieved using the subset(csv_data,) function in R. Multiple queries can be applied in the function at a time where each query is separated using a logical operator. The result is stored as a data frame in R.

Examples:

```
csv_data <- read.csv(file ='sample.csv')
min_pro <- min(csv_data$projects)
print (min_pro)
```

Output:

```
2
```

Aggregator functions (min, max, count etc.) can be applied on the CSV data. Here the **min()** function is applied on projects column using $ symbol. The minimum number of projects which is 2 is returned.

```
csv_data <- read.csv(file ='sample.csv')
new_csv <- subset(csv_data, department == "HR" & projects <10)
print (new_csv)
```

Output:

```
       id,    name,    department,     salary,      projects
4       4       D          HR           65043          5
7       7       G          HR           69000          7
```

The subset of the data that is created is stored as a data frame satisfying the conditions specified as the arguments of the function. The employees D and G are HR and have the number of projects<10. The row numbers are retained in the resultant data frame.

Writing into a CSV file
The contents of the data frame can be written into a CSV file. The CSV file is stored in the current working directory with the name specified in the function write.csv(data frame, output CSV name) in R.

Examples:

```
csv_data <- read.csv(file ='sample.csv')
new_csv <- subset(csv_data, department == "HR" & projects <10)
write.csv(new_csv, "new_sample.csv")
new_data <-read.csv(file ='new_sample.csv')
print(new_data)
```

Output:

```
      X    id,   name,    department,    salary,      projects
1     4      4      D          HR          65043           5
2     7      7      G          HR          69000           7
```

The column X contains the row numbers of the original CSV file. In order to remove it, we can specify an additional argument in the write.csv() function that set row names to FALSE.

```
csv_data <- read.csv(file ='sample.csv')
new_csv <- subset(csv_data, department == "HR" & projects <10)
write.csv(new_csv, "new_sample.csv", row.names = FALSE)
new_data <-read.csv(file ='new_sample.csv')
print(new_data)
```

Output:

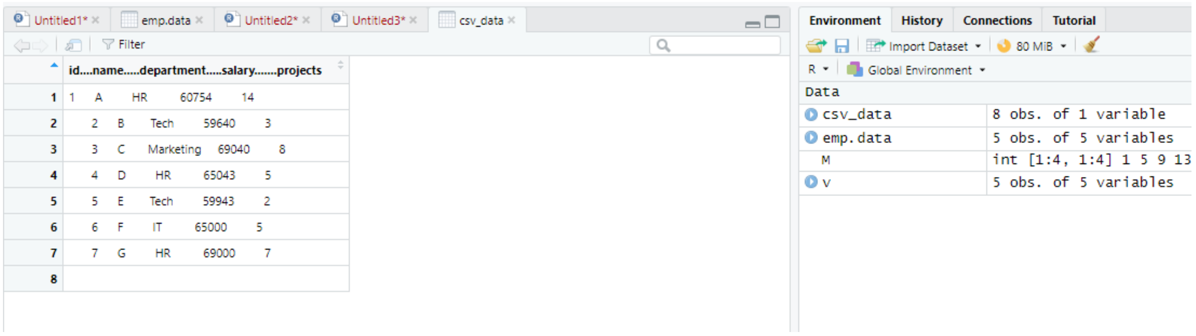
```
      id,   name,    department,    salary,      projects
1     4       D          HR          65043          5
2     7       G          HR          69000          7
```

The original row numbers are removed from the new CSV.

R PROGRAM:

```
csv_data <- read.csv(file = 'C:/Users/DELL/Desktop/lab mmm/sample.csv')
print(csv_data)
# print number of columns
print (ncol(csv_data))
# print number of rows
print(nrow(csv_data))
```

OUTPUT:



	id....name.....department.....salary.......projects
1	1 A HR 60754 14
2	2 B Tech 59640 3
3	3 C Marketing 69040 8
4	4 D HR 65043 5
5	5 E Tech 59943 2
6	6 F IT 65000 5
7	7 G HR 69000 7
8	

```
Data
csv_data    8 obs. of 1 variable
emp.data    5 obs. of 5 variables
M           int [1:4, 1:4] 1 5 9 13
v           5 obs. of 5 variables
```

PROG NO:		PG NO:
DATE:		R NO:

Table Creation in R

Method 1: Create a table from scratch

We can create a table by using as.table() function, first we create a table using matrix and then assign it to this method to get the table format.

Syntax:

```
as.table(data)
```

Example:

In this example, we will create a matrix and assign it to a table in the R language.

```
# create matrix with 4 columns and 4 rows

data= matrix(c(1:16), ncol=4, byrow=TRUE)

 # specify the column names and row names of matrix

colnames(data) = c('col1','col2','col3','col4')

rownames(data) <- c('row1','row2','row3','row4')

final=as.table(data)

 Final
```

Output:

```
      col1 col2 col3 col4

row1     1    2    3    4
row2     5    6    7    8
row3     9   10   11   12
```

Method 2: Create a table from an existing dataframe

We can create from the existing dataframe using table() function

Syntax:

```
table(dataframe$column_name, dataframe$column_name)
```

where,

- dataframe is the input dataframe
- column_name is the column names to be created as tables from the dataframe

Example:

In this example, we will be creating a table from an existing data frame using the table function in the R language.

```
# create dataframe with 4 columns and 4 rows

data= data.frame(col1=c(1:4),col2=c(5:8),

                  col3=c(9:12),col4=c(13:16))

 # assign to table from dataframe

final=table(data$col1,data$col2)

 # display

Final
```

Output:

```
   5 6 7 8
 1 1 0 0 0
 2 0 1 0 0
 3 0 0 1 0
 4 0 0 0 1
```

Bar Charts in R

Introduction Bar Charts in R

Bar Charts in R are the commonly used chart to create a graphical representation of the dataset. The Bar chart is represented as vertical or horizontal bars where the bar length or height indicates the count or frequency or any other calculated measure of the variable. As a best practice, a vector or a matrix can be used as input to the bar chat creation function in R for plotting bar charts. In addition, there are various labels, and color assignment features are available with the bar plot function, which is used to create the bar charts.

Syntax

The Basic syntax to create a Bar chart in R is shown below.

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

Description of the Parameters are:

H denotes height (vector or matrix). If H is a vector, the values determine the heights of the bars. If it is a matrix with option false corresponds to sub bars, and true denotes to create a horizontal bar.

- **xlab:** Label for X-axis
- **ylab:** Label for Y-axis
- **main:** Heading of the bar chart
- **names. arg:** Label to the bars a character vector.
- **col:** It gives color to the bars in the chart.

How to create a simple Bar Chart in R?

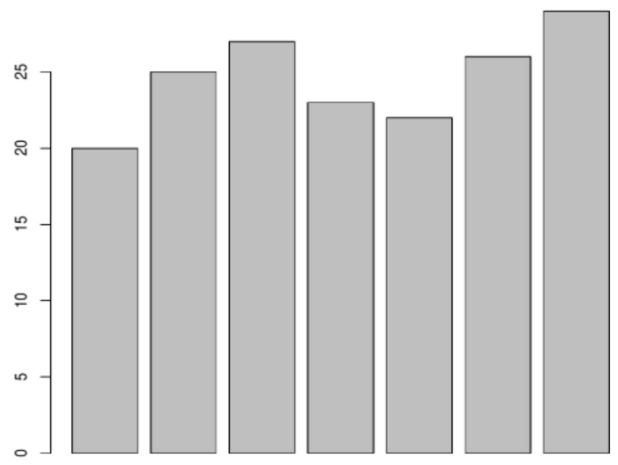
Here we shall discuss how to create Bar charts using function barplot () in R, which is very easy to implement with vertical and horizontal bars. In the below example, we will see creating charts using vectors.

```
temp <- c(20, 25, 27, 23, 22, 26, 29)
```

```
barplot(temp)
```

Output:

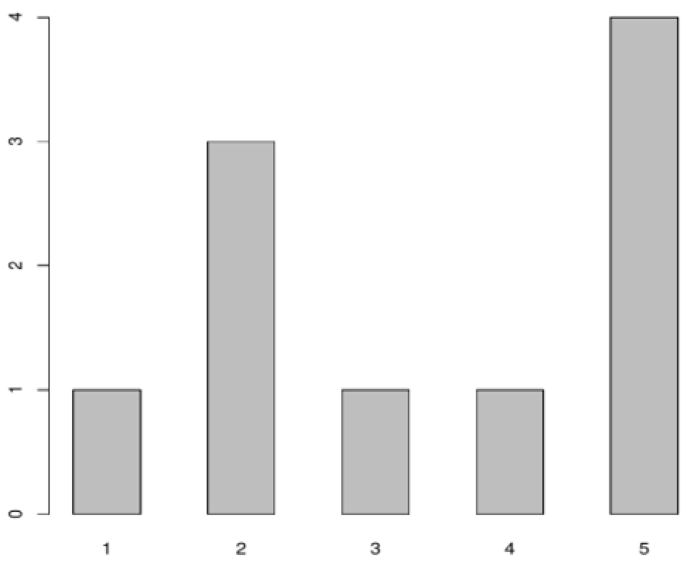
The bar Plot should look like this:



The next example comes with initializing some vector of numbers and creating a table () command to count them. The width of the bar can be adjusted using a parameter width () and space by space () in the barplot.

```
// Vector numbers are created using function c ()

x<- c (1,2,2,2,3,5,5,5,5,4)

cnt <- table(x)

cnt

x

barplot (cnt , space =1.0)
```

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



Creating a Bar chart using R built-in data set with a Horizontal bar. To do so, make horiz = TRUE or else vertical bars are drawn when horiz= FALSE (default option).

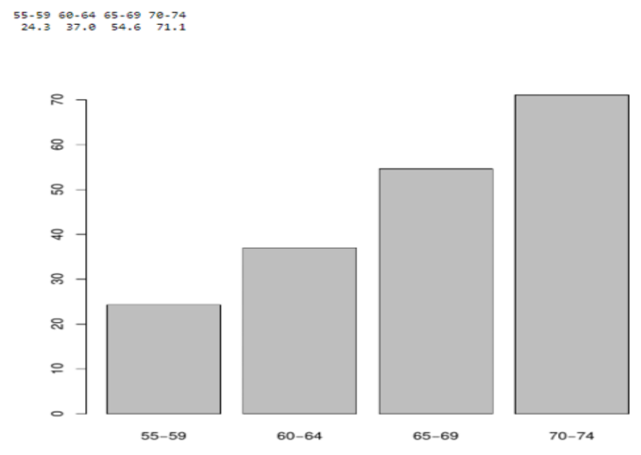
We shall consider a R data set as:

```
Rural Male Rural Female Urban Male Urban Female

## 50-54      11.7         8.7      15.4         8.4

## 55-59      18.1         11.7      24.3         13.6
## 60-64    26.9    20.3    37.0    19.3

## 65-69    41.0    30.9    54.6    35.1

## 70-74    66.0    54.3    71.1    50.0
```

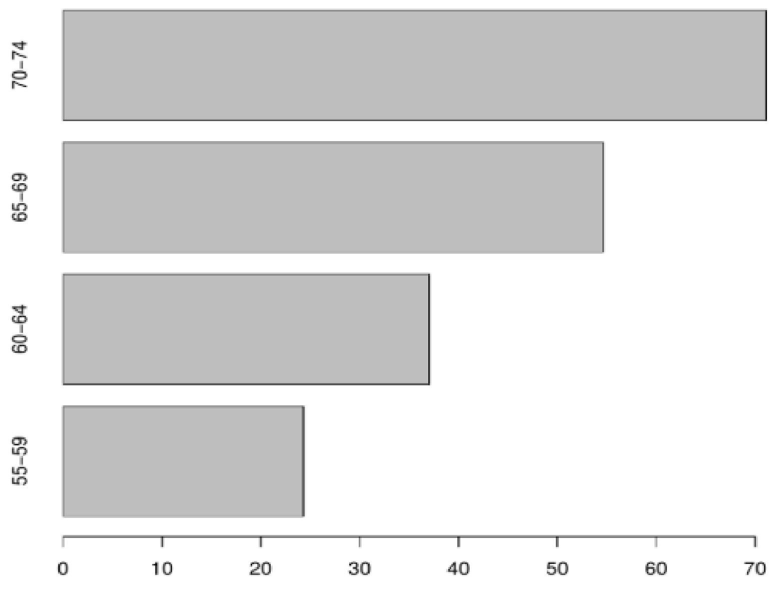
Here comes an example to plot the built-in data set of R.

```
a<- VADeaths [2:5, "Urban Male"] barplot(a)

# Horizontal bar plot

barplot (a, horiz = TRUE)
```

Output:



```
55-59 60-64 65-69 70-74
 24.3  37.0  54.6  71.1
```



Creating a Bar Chart with Labels & Title

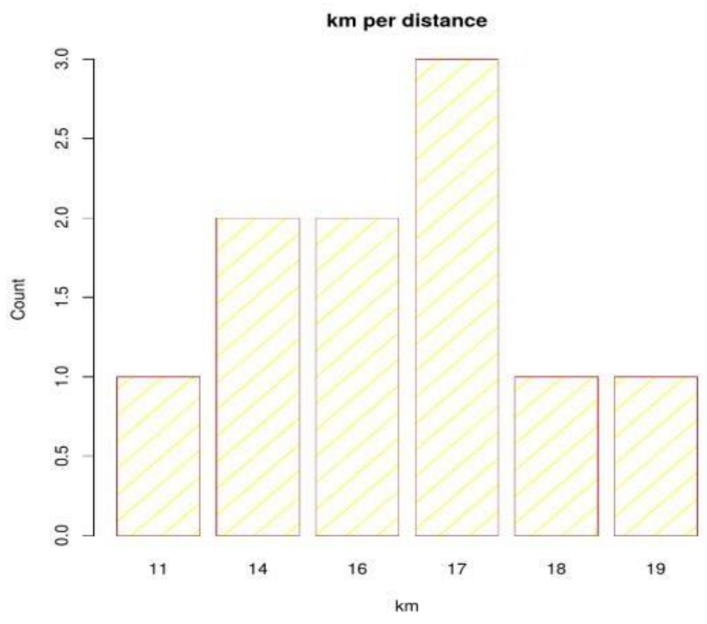
The bar chart could look more elegant by adding more parameters to the bar plot.

Assigning titles and labels

Titles here are assigned using main arguments as “ Km per distance”, and x-axis as “km and y-axis as “ count” (labels) and the parameter col is for adding colors to the bar(either in hexadecimal or RGB format). also, care should be taken a number of bars should be equal to the number of colours assigned in the character vector; if not, the colors get repeated, density is for shading lines on the bars. Titles and labels can be modified and added to the bar charts.

The following example plots kilometer per count using different parameters.

```
km <- c(11,14,14,16,17,19,17,16,17,18)

table (km)

km

barplot(table(km),

main="km per distance",

xlab="km",

ylab="Count",

border="brown",

col="yellow",

density=5)
```

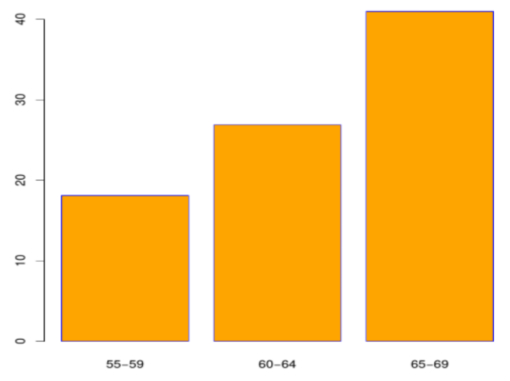


Assigning and changing colors

```
x <- VADeaths [2:4, "Rural Male"] barplot (x, col = "orange", border = "blue")
```

The bar chart for the above code is given here:

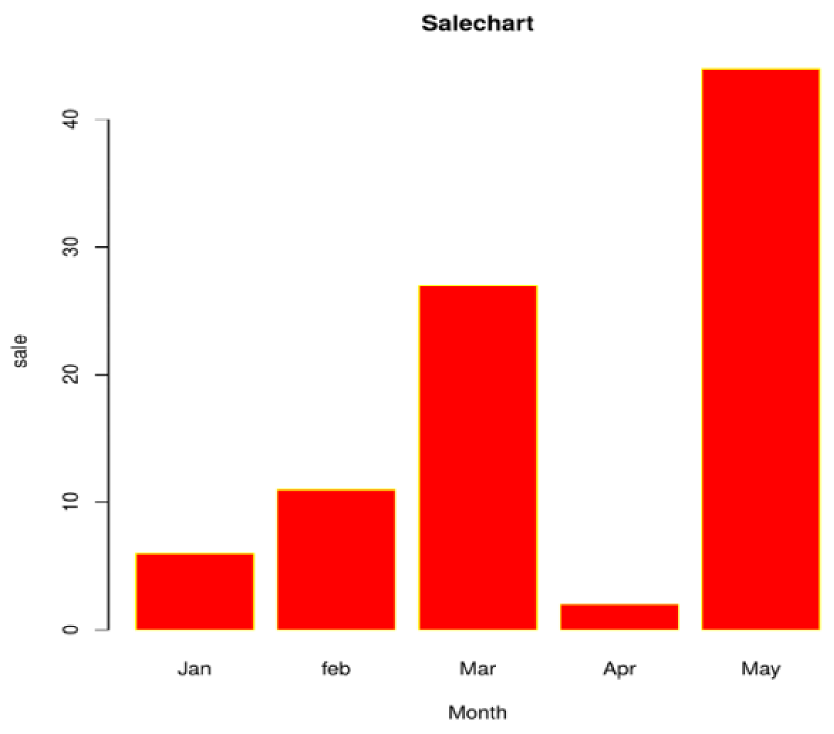
```
55-59 60-64 65-69
 18.1  26.9  41.0
```



And each of the bars can be assigned different colors. Here, we will fix some labels.

```
H <- c (6,11,27,2,44)

D <- c("Jan","feb","Mar","Apr","May")

barplot(H,names.arg=D,xlab="Month",ylab="sale",col="Red",main="Salechart",border="yel

low")
```

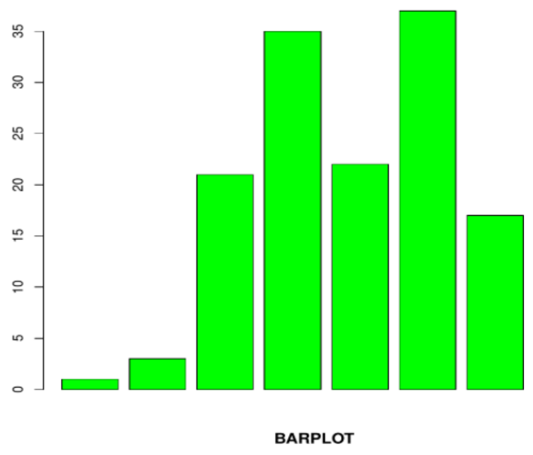
When executed, we get the following Output:

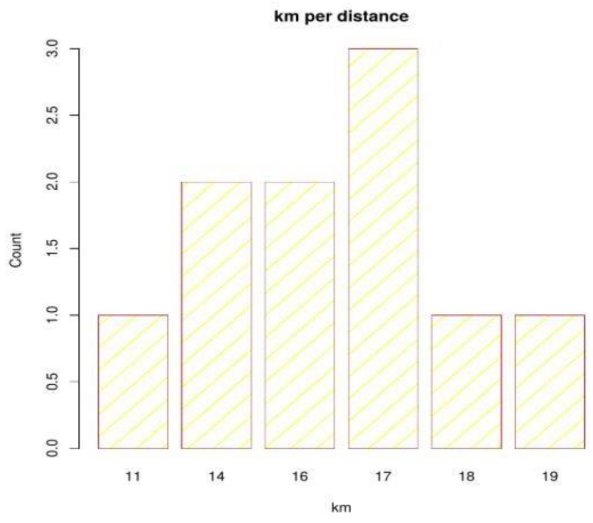


Using various Arguments

```
B <- c (1, 3, 21, 35, 22, 37, 17)

barplot (B, col="green")

barplot (B, main="BARPLOT", xlab="LETTERS", ylab="VALUES",

names.arg=c("A","B","C","D","E","F","G"),

border="yellow", density=c (90, 70, 50, 40, 30, 20, 10))
```

PROG NO:		PG NO:
DATE:		R NO:





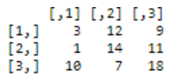
Using Matrix

```
mt <- c (3, 1, 10, 12, 14, 7, 9, 11, 18)
```

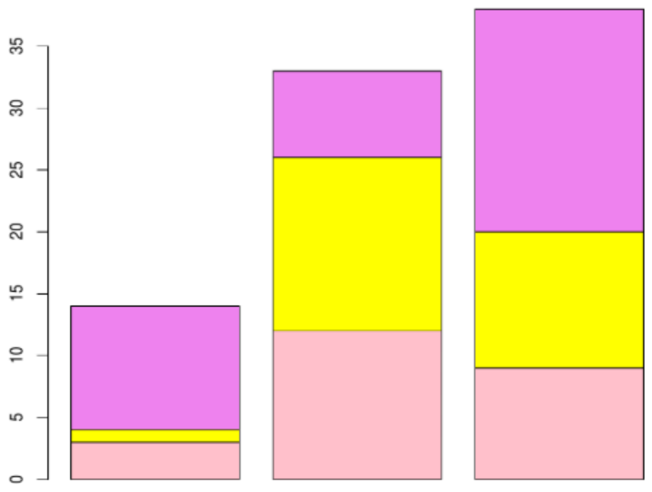
```
val <- matrix (mt, nrow = 3, ncol = 3)
```

```
val
```

```
barplot (val, col = c ("pink", "yellow", "violet"))
```



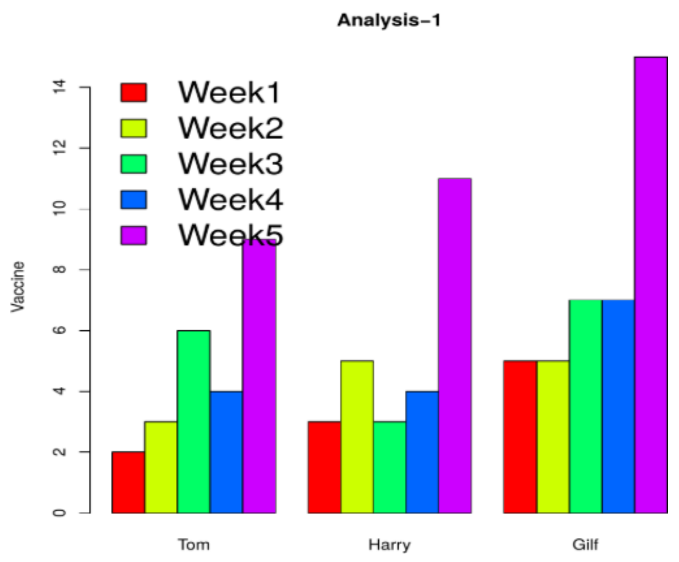
```
     [,1] [,2] [,3]
[1,]    3   12    9
[2,]    1   14   11
[3,]   10    7   18
```



Multiple comparisons

In the below example, we have created a matrix for three vectors representing five points, and a comparison between them is done using a bar chart. Here, we are using the legend function to display the legends. Bty argument is meant for legend borders. The data has been plotted as follows.

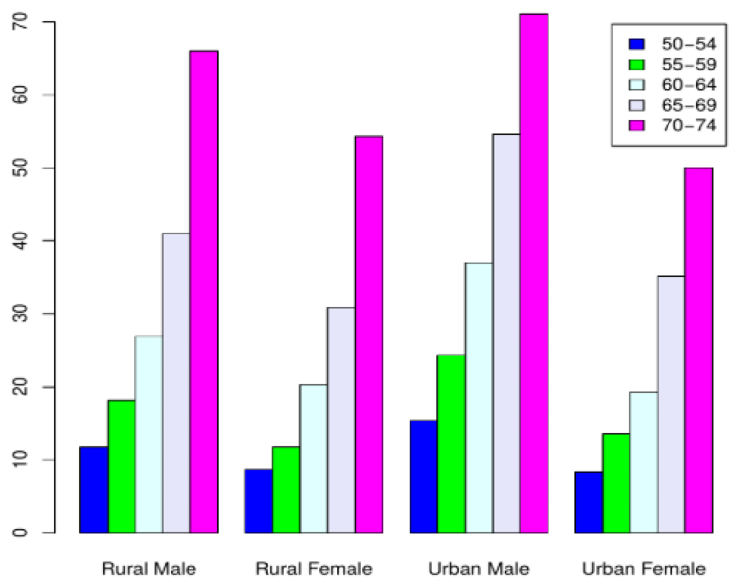
```
A <- c (2,3,6,4,9)

B <- c (3,5,3,4,11)

C <- c (5,5,7,7,15)

data<- data.frame(A, B, C)

names(data)<- c("Tom","Harry","Gilf")

barplot(height=as.matrix(data),main="Analysis-1",ylab="Vaccine",

beside=TRUE,col=rainbow (5))

legend

("topleft",c("Week1","Week2","Week3","Week4","Week5"),cex=2.0,bty="n",fill=rainbow

(5))
```



Grouped Bar Plots

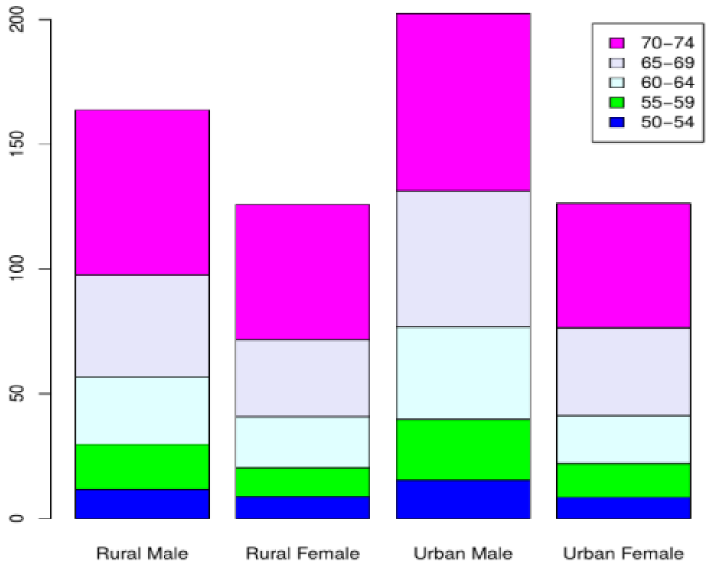
Bar charts are created for all the columns. (columns are grouped together). Group chart makes use of matrix as input values.

```
barplot (VADeaths,col = c("blue", "green", "lightcyan","lavender", "magenta"),
legend = rownames(VADeaths),beside = TRUE)
```



// Now Making beside = FALSE

```
barplot (VADeaths, col = c("blue","green","light cyan","lavender","magenta"),

legend = rownames(VADeaths), beside = FALSE)
```



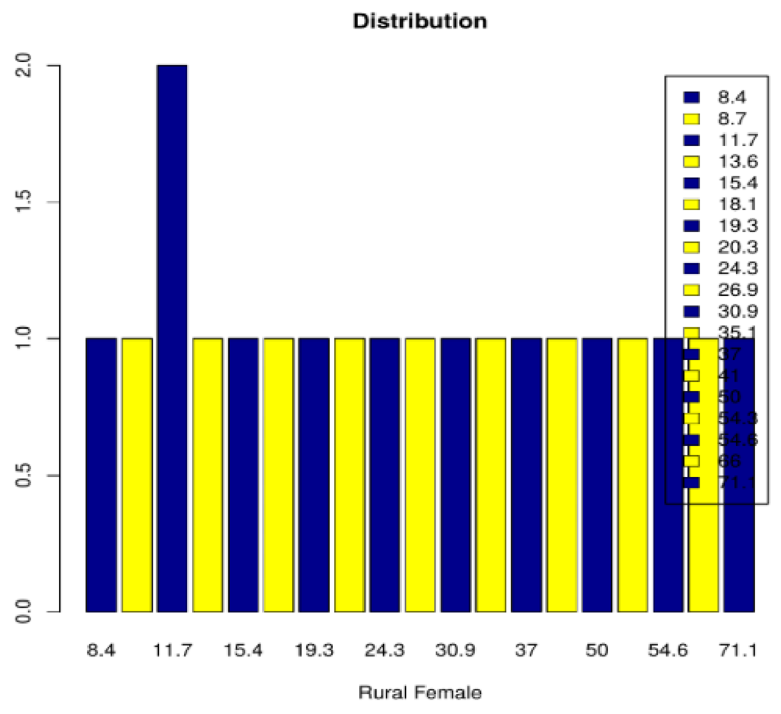
Stacked Bar Plot

Instead of assigning the bars continuously, it is effective to stack them in order.

Example:

```
counts <- table (VADeaths)

barplot(counts, main="Distribution",

xlab="Rural Female",col=c("darkblue","yellow"), legend = rownames(counts))
```

PROG NO:		PG NO:
DATE:		R NO:



Introduction to Pie Chart in R

R offers a basic chart feature known as the Pie Chart, which displays data value proportions in a circular chart symbol. Labeling the sections of the chart representing different data values with meaningful names is possible. Pie charts are generally preferred for small-size vector variables. Based on the **R packages**, pie charts can have two-dimensional or three-dimensional views. Pie is the function in R language which supports two-dimensional pie charts. Pie charts are handy for **data analysis**. The pie function in R allows users to assign a meaningful title to pie charts using the “main” parameter.

Pie charts facilitate an easy understanding of patterns in data, compared to numeric figures, which often require more time for comprehension. For example, if we plot the above example as a pie chart, we can understand the amount and proportion of production within a minute.
There are various packages for plotting pie charts in R, and among those many options, we shall focus on two methods in this article.

Syntax

The above section briefly explains the pie chart and its use. In this section, we shall learn about pie charts in R specifically. For those new to R, it is a **programming language** mainly used for data analysis and machine learning. R is exceptionally rich in functionality and provides hundreds of libraries for various use cases.

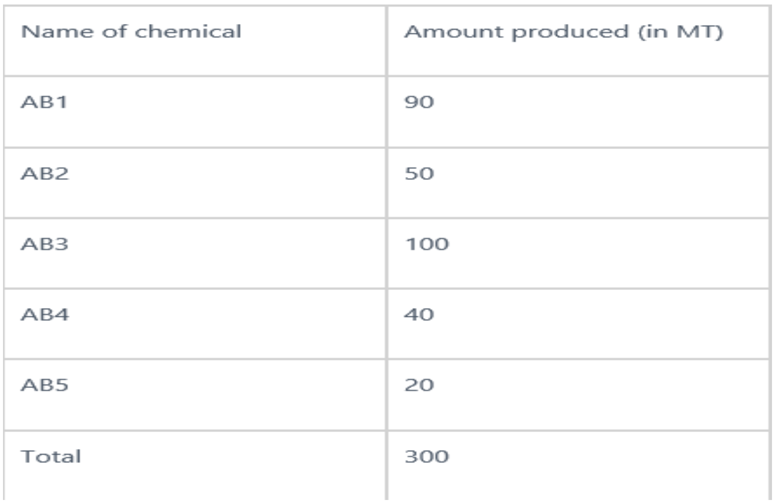
A simple in-built function in R can create it, and the syntax for creating it is as follows: `pie(x, labels, radius, main, col, clockwise)`

Where,Here x is a vector containing the numeric values present in the pie chart, such as those production figures in the above example.

- Labels take a name for those values in X, such as the name of chemicals.
- The radius argument is for the radius of the circle of the pie chart. Its typical value lies between −1 and +1.
- The main argument presents the title of the chart.
- col argument can describe the colors on the chart.
- Clockwise is a logical value that takes either True or False, indicating if the slices of charts are present clockwise or anti-clockwise.**Note:** That X is a mandatory argument, and the rest are all optional.

How to create a pie chart in R?

Now that we understand the syntax of the pie chart let's build a pie chart. We will again use the same example in the introduction section above. First of all, let's convert the example above into the form of a table for easy understanding.

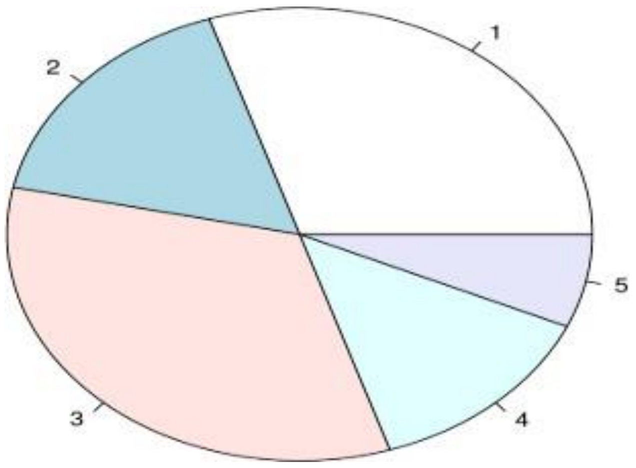


Name of chemical	Amount produced (in MT)
AB1	90
AB2	50
AB3	100
AB4	40
AB5	20
Total	300

First, we use the following two lines of R code to convert the table above into two vectors, one for the name of the chemical and the other for the volume of the chemical.

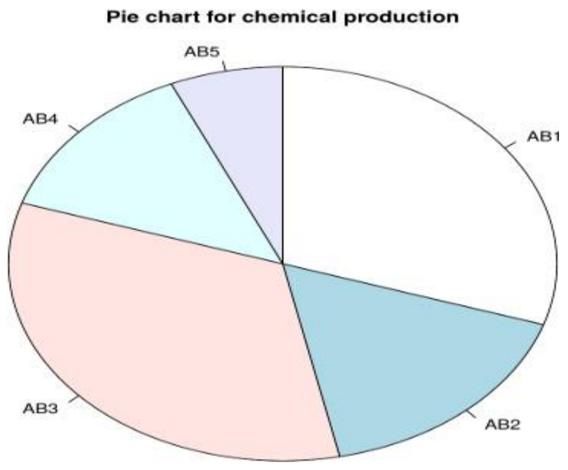
Now, we plot a simple pie chart by only providing the x value in the syntax above:

- chem <- c("AB1","AB2","AB3","AB4","AB5")
- vol <- c(90,50,100,40,20)
- pie(vol) Its output is the figure below:



If you observe the output, it is not very clear as to what exactly is what. So to make it more intuitive, we input a few more arguments in the pie function and run again.

- chem <- c("AB1","AB2","AB3","AB4","AB5")
- vol <- c(90,50,100,40,20)
- pie(x=vol, labels = chem, radius = 1,main = "Pie chart for chemical production", clockwise = T)



This picture is better to understand as it contains the name of the chemicals and a title. Please note that the color scheme in both charts is coming by default, which we can change as per our need or wish. We will do that in the section below.

How to change pie charts and fill color?

In this section, let's learn how to change pie charts.

First, let's show the number of chemicals in the chart instead of the name of the chemicals.

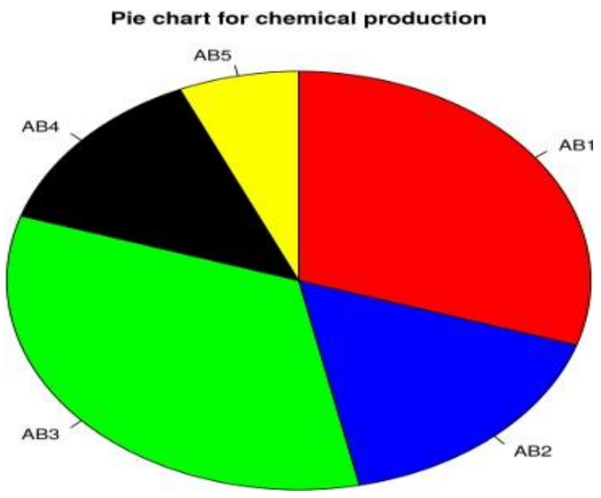
```
pie(x=vol, labels = vol, radius = 1,main = “Pie chart for chemical production”, clockwise = T)
```

Run it yourself and see the output.

Next, let's change the color of the charts.

- chem <- c("AB1","AB2","AB3","AB4","AB5")
- vol <- c(90,50,100,40,20)
- pie(x=vol, labels = chem, radius = 1,main = "Pie chart for chemical production", col=c("red"," blue"," green"," black"," yellow"),clockwise = T)

Here we specified the colors that we wanted. The output is as below:



PROG NO:		PG NO:
DATE:		R NO:

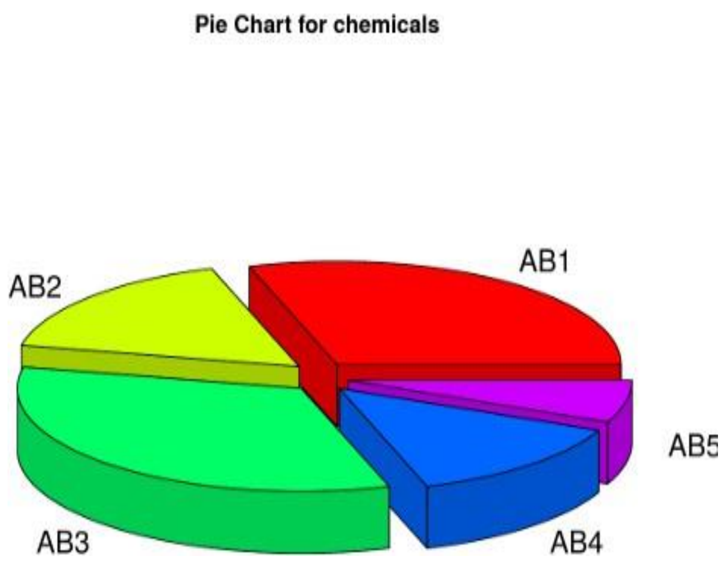
How to create a 3D pie chart?

In this section, we will learn how to build a 3D pie chart in R. for creating a 3d pie chart; we need to install a library first as it differs from an essential inbuilt function.

You should install the library plotrix before running the code for the pie chart. To install the library, run the following command in R.

- packages(“plotrix”) After that, run the following two lines to get a 3d plot.
- chem <- c(“AB1”,”AB2”,”AB3”,”AB4”,”AB5”)
- vol <- c(90,50,100,40,20)
- library(plotrix)
- pie3D(vol,labels = chem,explode = 0.1, main = “Pie Chart for chemicals ”)

The output is as below:



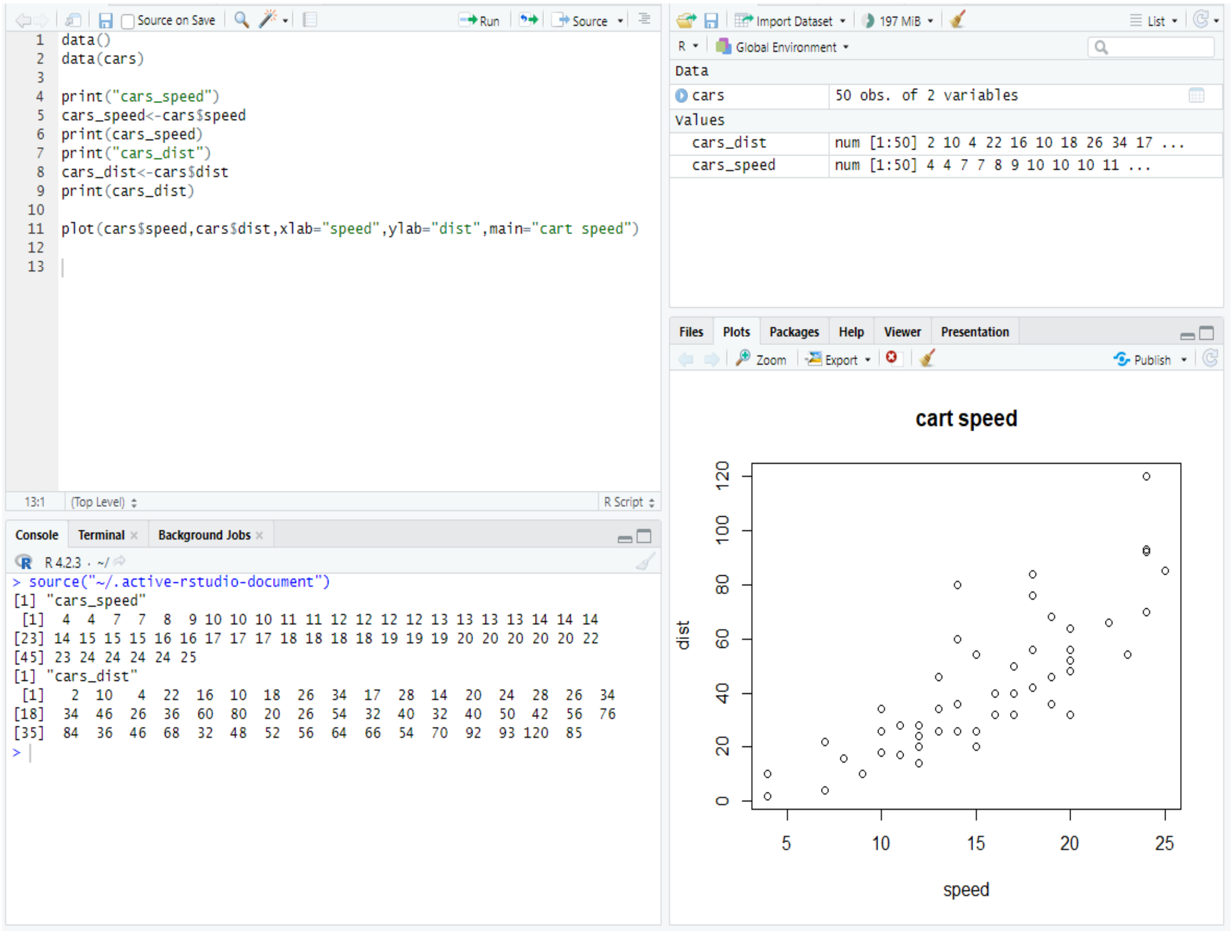
PBR VITS (Autonomous), KAVALI

DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE

PROG NO:		PG NO:
DATE:		R NO:

WORKING WITH DIFFERENT DATA SETS:

CARS :



```r
data()
data(cars)

print("cars_speed")
cars_speed<-cars$speed
print(cars_speed)
print("cars_dist")
cars_dist<-cars$dist
print(cars_dist)

plot(cars$speed,cars$dist,xlab="speed",ylab="dist",main="cart speed")
```

```
Data
cars          50 obs. of 2 variables
Values
  cars_dist   num [1:50] 2 10 4 22 16 10 18 26 34 17 ...
  cars_speed  num [1:50] 4 4 7 7 8 9 10 10 10 11 ...
```

```
> source("~/.active-rstudio-document")
[1] "cars_speed"
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14
[23] 14 15 15 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22
[45] 23 24 24 24 24 25
[1] "cars_dist"
 [1]   2  10   4  22  16  10  18  26  34  17  28  14  20  24  28  26  34
[18]  34  46  26  36  60  80  20  26  54  32  40  32  40  50  42  56  76
[35]  84  36  46  68  32  48  52  56  64  66  54  70  92  93 120  85
> 
```

PBR VITS (Autonomous), KAVALI
DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE

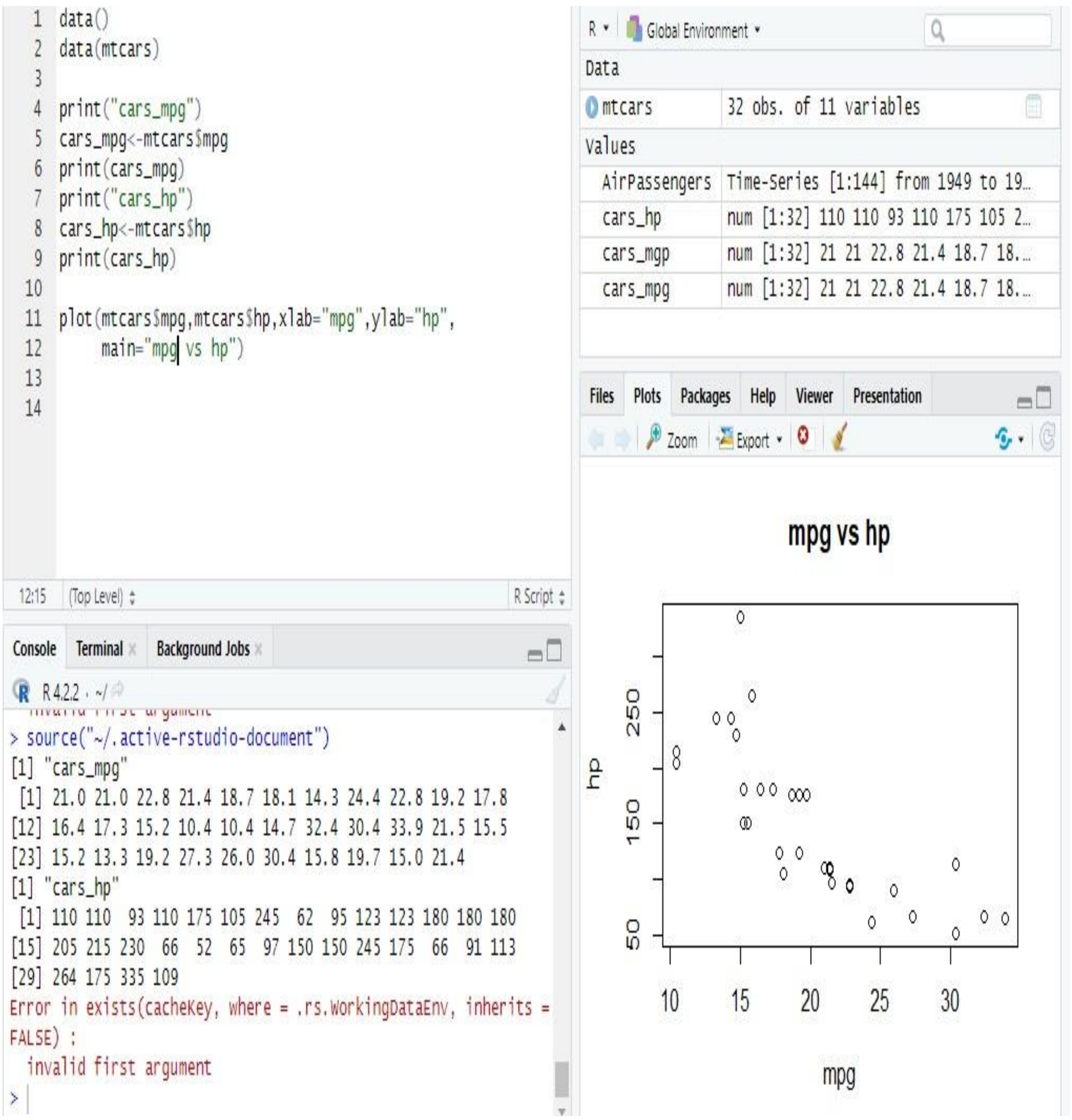
PROG NO:

DATE:

PG NO:

R NO:

MTCARS:



```r
data()
data(mtcars)

print("cars_mpg")
cars_mpg<-mtcars$mpg
print(cars_mpg)
print("cars_hp")
cars_hp<-mtcars$hp
print(cars_hp)

plot(mtcars$mpg,mtcars$hp,xlab="mpg",ylab="hp",
     main="mpg vs hp")
```

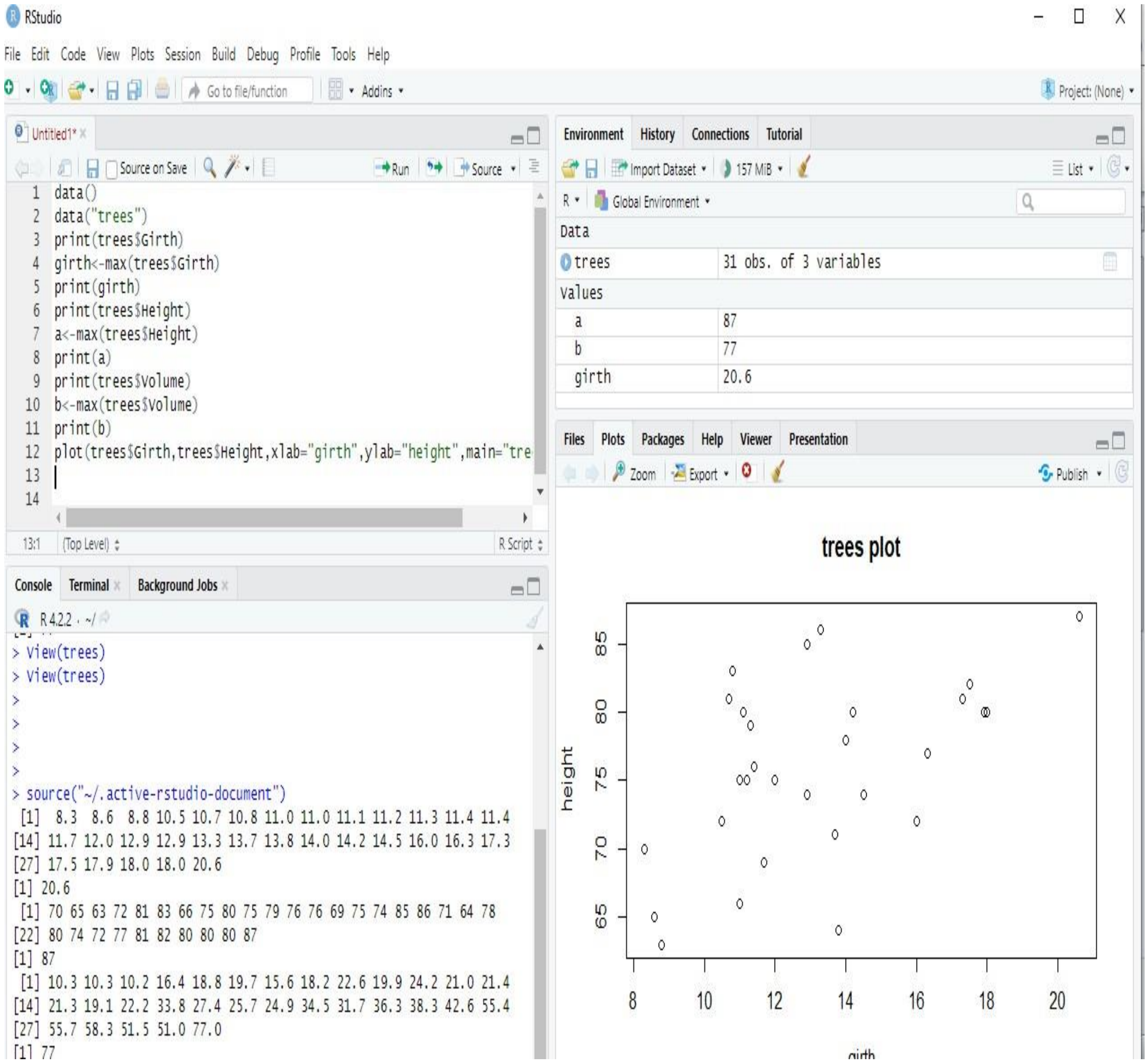
```
Data
mtcars          32 obs. of 11 variables
Values
AirPassengers   Time-Series [1:144] from 1949 to 19…
cars_hp         num [1:32] 110 110 93 110 175 105 2…
cars_mgp        num [1:32] 21 21 22.8 21.4 18.7 18.…
cars_mpg        num [1:32] 21 21 22.8 21.4 18.7 18.…
```

```
> source("~/.active-rstudio-document")
[1] "cars_mpg"
 [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8
[12] 16.4 17.3 15.2 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5
[23] 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4
[1] "cars_hp"
 [1] 110 110  93 110 175 105 245  62  95 123 123 180 180 180
[15] 205 215 230  66  52  65  97 150 150 245 175  66  91 113
[29] 264 175 335 109
Error in exists(cacheKey, where = .rs.WorkingDataEnv, inherits =
FALSE) :
  invalid first argument
>
```

PBR VITS (Autonomous), KAVALI
DEPARTMENT OF CSE-ARTIFICIAL INTELLIGENCE

PROG NO:		PG NO:
DATE:		R NO:

TREES:



```r
data()
data("trees")
print(trees$Girth)
girth<-max(trees$Girth)
print(girth)
print(trees$Height)
a<-max(trees$Height)
print(a)
print(trees$Volume)
b<-max(trees$Volume)
print(b)
plot(trees$Girth,trees$Height,xlab="girth",ylab="height",main="tre
```

Data	
trees	31 obs. of 3 variables
Values	
a	87
b	77
girth	20.6

```
> View(trees)
> View(trees)
>
>
>
>
> source("~/.active-rstudio-document")
 [1]  8.3  8.6  8.8 10.5 10.7 10.8 11.0 11.0 11.1 11.2 11.3 11.4 11.4
[14] 11.7 12.0 12.9 12.9 13.3 13.7 13.8 14.0 14.2 14.5 16.0 16.3 17.3
[27] 17.5 17.9 18.0 18.0 20.6
[1] 20.6
 [1] 70 65 63 72 81 83 66 75 80 75 79 76 76 69 75 74 85 86 71 64 78
[22] 80 74 72 77 81 82 80 80 80 87
[1] 87
 [1] 10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 24.2 21.0 21.4
[14] 21.3 19.1 22.2 33.8 27.4 25.7 24.9 34.5 31.7 36.3 38.3 42.6 55.4
[27] 55.7 58.3 51.5 51.0 77.0
[1] 77
```

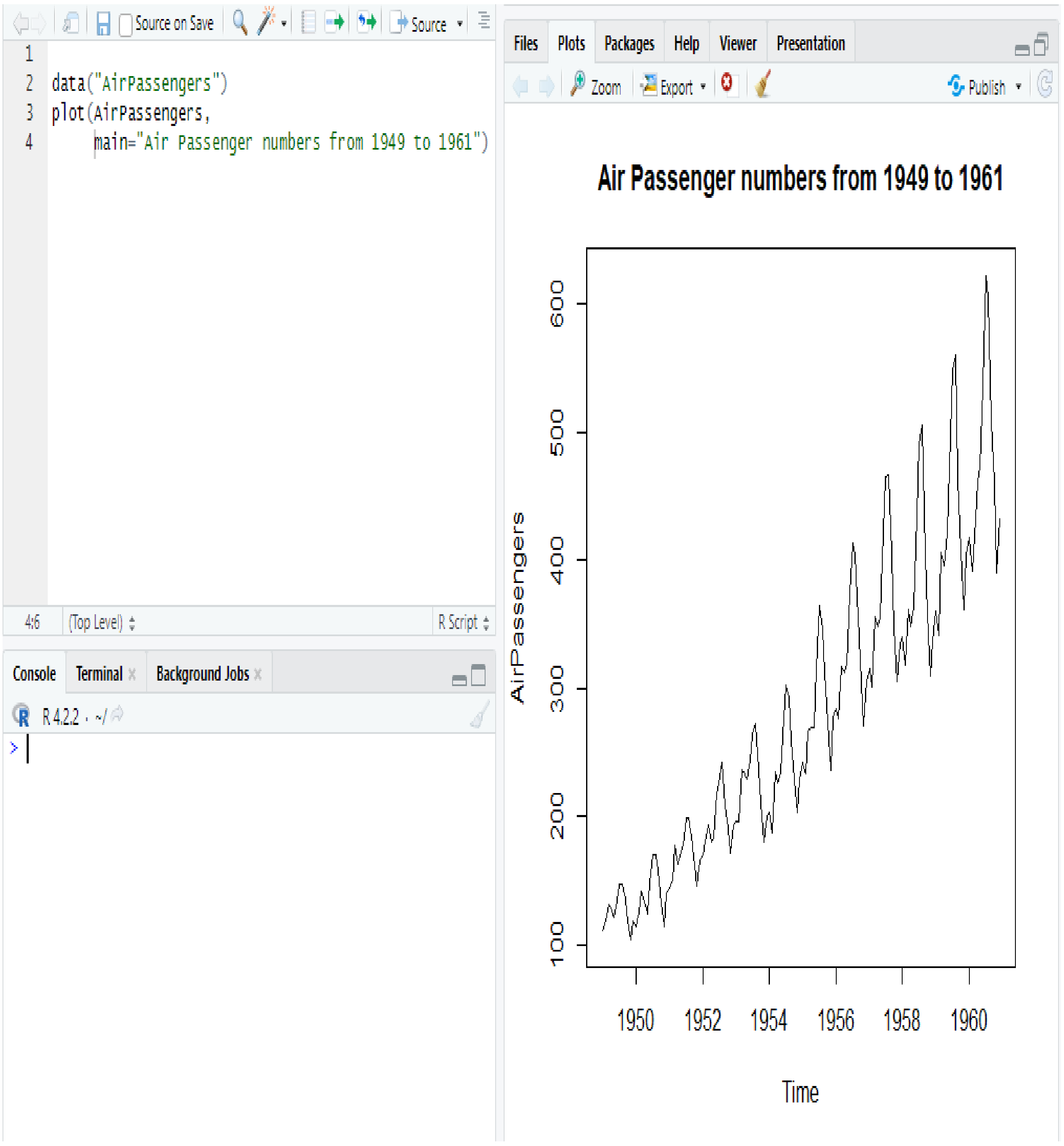
PROG NO:

DATE:

PG NO:

R NO:

AIRPASSENGERS:



```
data("AirPassengers")
plot(AirPassengers,
     main="Air Passenger numbers from 1949 to 1961")
```

How to Perform Univariate Analysis in R (With Examples)

The term univariate analysis refers to the analysis of one variable. You can remember this because the prefix "uni" means "one."

There are three common ways to perform univariate analysis on one variable:

1. Summary statistics – Measures the center and spread of values.

2. Frequency table – Describes how often different values occur.

3. Charts – Used to visualize the distribution of values.

This tutorial provides an example of how to perform univariate analysis for the following variable:

```
#create variable with 15 values
x <- c(1, 1, 2, 3.5, 4, 4, 4, 5, 5, 6.5, 7, 7.4, 8, 13, 14.2)
```

Summary Statistics

We can use the following syntax to calculate various summary statistics for our variable:

```
#find mean
mean(x)
[1] 5.706667

#find median
median(x)

[1] 5

#find range
max(x) - min(x)

[1] 13.2
```

```
#find interquartile range (spread of middle 50% of values)
IQR(x)

[1] 3.45

#find standard deviation
sd(x)

[1] 3.858287
```

Frequency Table

We can use the following syntax to produce a frequency table for our variable:

```
#produce frequency table
table(x)

  1    2  3.5    4    5  6.5    7  7.4    8   13  14.2
  2    1    1    3    2    1    1    1    1    1     1
```

This tells us that:

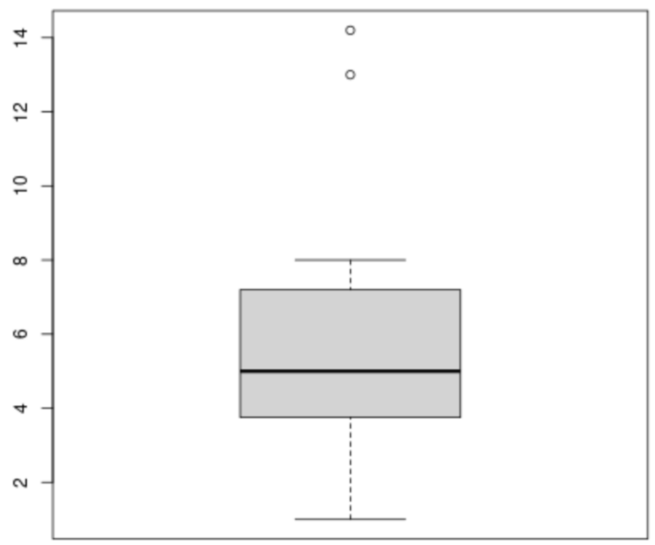
- The value **1** occurs 2 times
- The value **2** occurs 1 time
- The value **3.5** occurs 1 time

And so on.

Charts

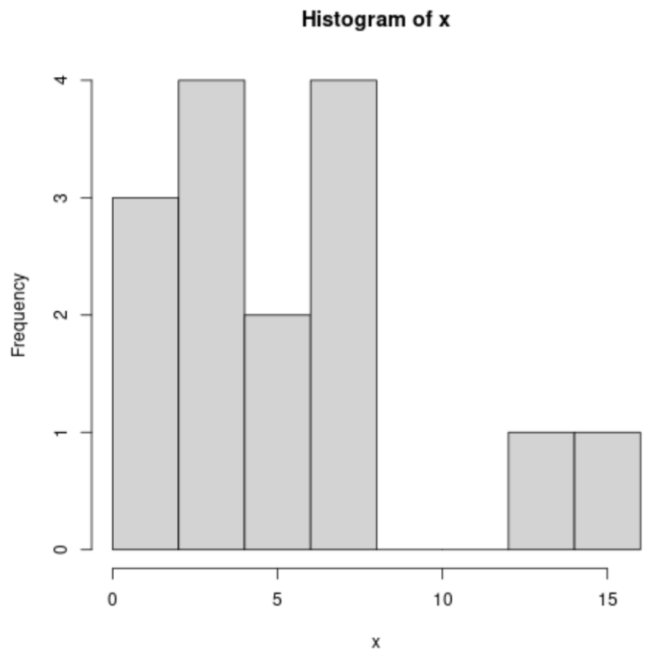
We can produce a boxplot using the following syntax:

```
#produce boxplot
boxplot(x)
```



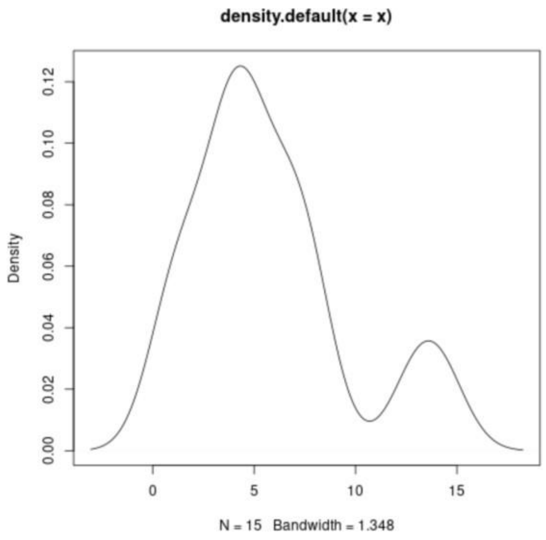
We can produce a histogram using the following syntax:

```
#produce histogram
hist(x)
```



We can produce a density curve using the following syntax:

```
#produce density curve
plot(density(x))
```



Each of these charts give us a unique way to visualize the distribution of values for our variable.

Central Tendency in R Programming

Central Tendency is one of the feature of descriptive statistics. Central tendency tells about how the group of data is clustered around the centre value of the distribution. Central tendency performs the following measures:

- Arithmetic Mean
- Geometric Mean
- Harmonic Mean
- Median
- Mode

Arithmetic Mean

The arithmetic mean is simply called the average of the numbers which represents the central value of the data distribution. It is calculated by adding all the values and then dividing by the total number of observations.

Formula:

$$X = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

where,

X *indicates the arithmetic mean*

indicates value in data vector

n *indicates total number of observations*

In R language, arithmetic mean can be calculated by **mean()** function.

Syntax: *mean(x, trim, na.rm = FALSE)*

Parameters:

x: *Represents object*

trim: *Specifies number of values to be removed from each side of object before calculating the mean. The value is between 0 to 0.5*

na.rm: *If TRUE then removes the NA value from* ***x***

Example:

```
# Defining vector

x <- c(3, 7, 5, 13, 20, 23, 39, 23, 40, 23, 14, 12, 56, 23)



# Print mean

print(mean(x))
```

Output:

```
[1] 21.5
```

Geometric Mean

The geometric mean is a type of mean that is computed by multiplying all the data values and thus, shows the central tendency for given data distribution.

Formula:

$$X = \left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

where,

X *indicates geometric mean*

indicates value in data vector

n *indicates total number of observations*

prod() and **length()** function helps in finding the geometric mean for given set of numbers as there is no direct function for geometric mean.

Syntax:

```
prod(x)^(1/length(x))
```

where,

prod() *function returns the product of all values present in vector* ***x***

length() *function returns the length of vector* ***x***

Example:

```
# Defining vector

x <- c(1, 5, 9, 19, 25)

# Print Geometric Mean

print(prod(x)^(1 / length(x)))
```

Output:

```
[1] 7.344821
```

Harmonic Mean

Harmonic mean is another type of mean used as another measure of central tendency. It is computed as reciprocal of the arithmetic mean of reciprocals of the given set of values.

Formula:

$$X = \frac{N}{\sum_{i=1}^{N} \frac{1}{x_i}}$$

where,

X *indicates harmonic mean*

indicates value in data vector

n *indicates total number of observations*

Example:

Modifying the code to find the harmonic mean of given set of values.

```
# Defining vector

x <- c(1, 5, 8, 10)
```

```
# Print Harmonic Mean

print(1 / mean(1 / x))
```

Output:

```
[1] 2.807018
```

Median

Median in statistics is another measure of central tendency which represents the middlemost value of a given set of values.

In R language, median can be calculated by **median()** function.
Syntax: *median(x, na.rm = FALSE)*
Parameters:
x: *It is the data vector*
na.rm: *If TRUE then removes the NA value from **x***

Example:

```
# Defining vector

x <- c(3, 7, 5, 13, 20, 23, 39,

       23, 40, 23, 14, 12, 56, 23)

# Print Median

median(x)
```

Output:

```
[1] 21.5
```

Mode

The mode of a given set of values is the value that is repeated most in the set. There can exist multiple mode values in case if there are two or more values with matching maximum frequency.

Example 1: Single-mode value

In R language, there is no function to calculate mode. So, modifying the code to find out the mode for a given set of values.

```
# Defining vector

x <- c(3, 7, 5, 13, 20, 23, 39,

       23, 40, 23, 14, 12, 56,

       23, 29, 56, 37, 45, 1, 25, 8)



# Generate frequency table

y <- table(x)

# Print frequency table

print(y)

# Mode of x

m <- names(y)[which(y == max(y))]

# Print mode

print(m)
```

Output:

```
x
 1  3  5  7  8 12 13 14 20 23 25 29 37 39 40 45 56
 1  1  1  1  1  1  1  1  1  4  1  1  1  1  1  1  2
[1] "23"
```

Example 2: Multiple Mode values

```
# Defining vector
x <- c(3, 7, 5, 13, 20, 23, 39, 23, 40,
       23, 14, 12, 56, 23, 29, 56, 37,
       45, 1, 25, 8, 56, 56)


# Generate frequency table
y <- table(x)
# Print frequency table
print(y)
# Mode of x
m <- names(y)[which(y == max(y))]
# Print mode
```

```
print(m)
```

Output:

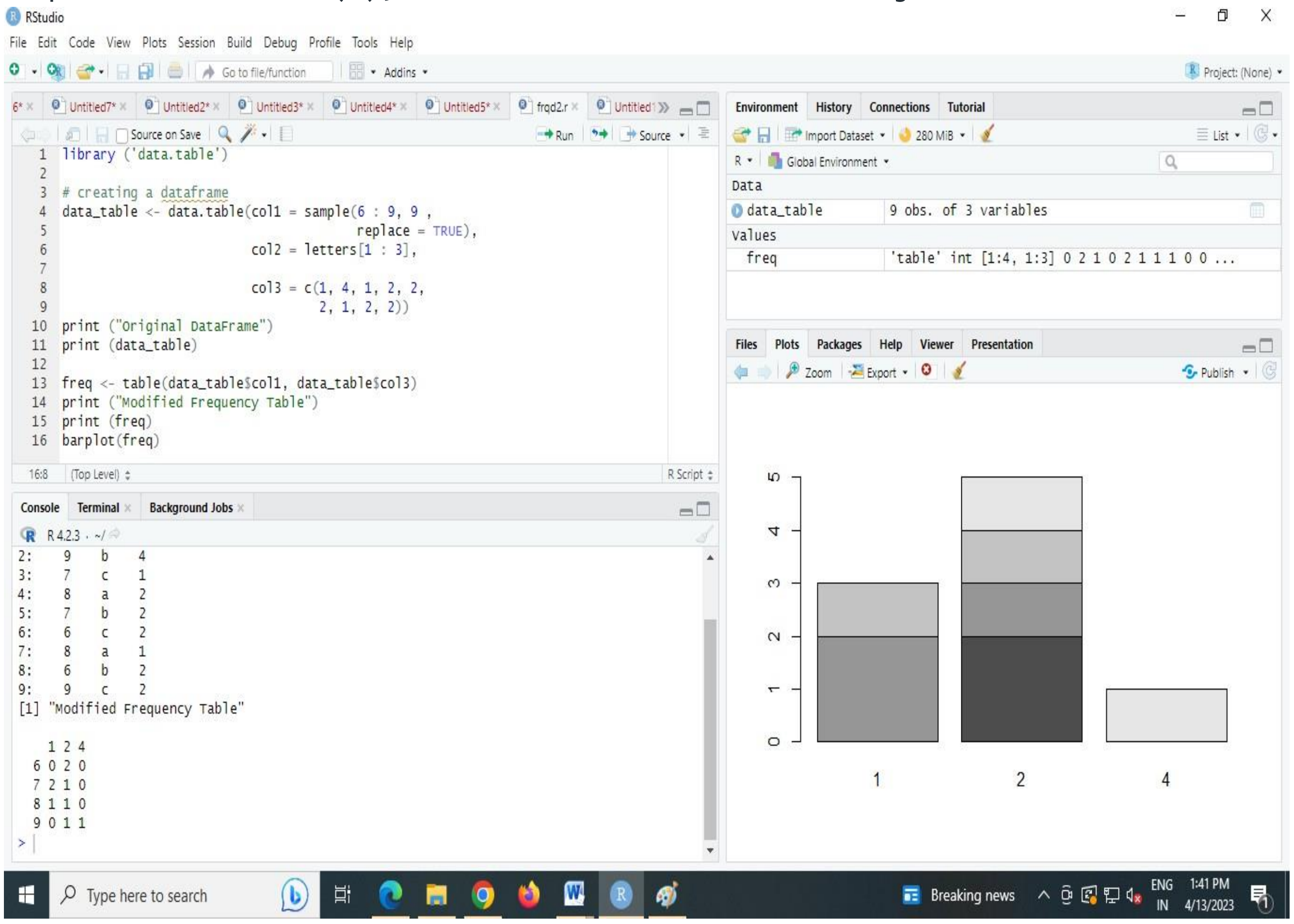
```
x
 1  3  5  7  8 12 13 14 20 23 25 29 37 39 40 45 56 
 1  1  1  1  1  1  1  1  1  4  1  1  1  1  1  1  4 
[1] "23" "56"
```

Frequency Distribution table

In this article, we are going to see how to make a frequency distribution table using R Programming Language.

The table() method in R is used to compute the frequency counts of the variables appearing in the specified column of the dataframe. The result is returned to the form of a two-row tabular structure, where the first row indicates the value of the column and the next indicates its corresponding frequencies.

```
frq-table <- table (x), where x is the data.table object
```



```r
library ('data.table')

# creating a dataframe
data_table <- data.table(col1 = sample(6 : 9, 9 ,
                                       replace = TRUE),
                         col2 = letters[1 : 3],

                         col3 = c(1, 4, 1, 2, 2,
                                  2, 1, 2, 2))
print ("Original DataFrame")
print (data_table)

freq <- table(data_table$col1, data_table$col3)
print ("Modified Frequency Table")
print (freq)
barplot(freq)
```

```
2:    9    b    4
3:    7    c    1
4:    8    a    2
5:    7    b    2
6:    6    c    2
7:    8    a    1
8:    6    b    2
9:    9    c    2
[1] "Modified Frequency Table"

    1 2 4
  6 0 2 0
  7 2 1 0
  8 1 1 0
  9 0 1 1
>
```

Environment: data_table — 9 obs. of 3 variables; freq — 'table' int [1:4, 1:3] 0 2 1 0 2 1 1 1 0 0 ...

PROG NO:

DATE:

PG NO:

R NO:

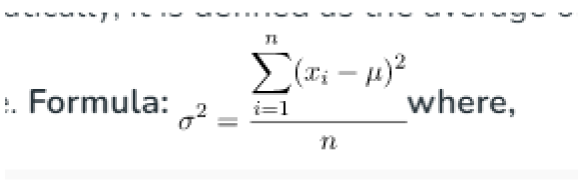
Variability in R Programming

Variability (also known as **Statistical Dispersion**) is another feature of descriptive statistics. Measures of central tendency and variability together comprise of descriptive statistics. Variability shows the spread of a data set around a point. **Example:** Suppose, there exist 2 data sets with the same mean value:
A = 4, 4, 5, 6, 6 Mean(A) = 5 B = 1, 1, 5, 9, 9 Mean(B) = 5

So, to differentiate among the two data sets, R offers various measures of variability.

Variance
Variance is a measure that shows how far is each value from a particular point, preferably mean value. Mathematically, it is defined as the average of squared differences from the mean

value. **Formula:**



. Formula: $\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \mu)^2}{n}$ where,

where,

*specifies variance of the data set specifies value in data set specifies the mean of data set **n** specifies total number of observations*

In the R language, there is a standard built-in function to calculate the variance of a data set.

***Syntax:** var(x) **Parameter: x:** It is data vector*

Example:

-

```
# Defining vector

x <- c(5, 5, 8, 12, 15, 16)
```

```
 # Print variance of x

print(var(x))
```

Output:

```
[1] 23.76667
```

Multivariate Analysis in R

Analyzing data sets with numerous variables is a crucial statistical technique known as multivariate analysis. Many different multivariate analysis procedures can be carried out using the well-liked programming language R. A number of libraries and functions are available in the well-liked programming language R for carrying out multivariate analysis. In this post, we'll go through various functions and methods for implementing multivariate analysis in R Programming Language.

- **Multivariate analysis:** The statistical analysis of data sets with several variables is referred to as multivariate analysis. In order to comprehend the underlying structure of the data and to find patterns and interactions between variables, multivariate analysis is performed.
- **Multivariate data:** Data sets with multiple variables are referred to as multivariate data. Multivariate data can be quantitative or categorical, and it is possible to analyze it using a number of different statistical methods.
- **Dimensionality reduction:** Dimensionality reduction is the technique of minimizing information loss while minimizing the number of variables in a data set. Multivariate analysis frequently uses dimensionality reduction to streamline the data and make it simpler to analyze.
- **Exploratory and confirmatory analysis:** Without having any preconceived notions, exploratory analysis is used to examine and comprehend the dataset. A specific hypothesis is validated through confirmatory analysis.

- R

```r
# Load the iris data set
data(iris)


# Select the variables to include
# in the PCA analysis
vars <- c("Sepal.Length", "Sepal.Width",
          "Petal.Length", "Petal.Width")


# Subset the data to include
# only the selected variables
data_subset <- iris[, vars]

# Scale the data
data_scaled <- scale(data_subset)

# Perform PCA
pca <- prcomp(data_scaled,
              center = TRUE, scale. = TRUE)

# Print the summary of the PCA results
summary(pca)
```

Output:

```
Importance of components:
                          PC1    PC2     PC3     PC4
Standard deviation     1.7084 0.9560 0.38309 0.14393
Proportion of Variance 0.7296 0.2285 0.03669 0.00518
Cumulative Proportion  0.7296 0.9581 0.99482 1.00000
```

The results of the PCA are summarized in this output, which also includes the standard deviation, variance proportion, and cumulative proportion for each principal component. The first principal component accounts for 72.96 percent of the total variation in the data, whereas the second and third components each account for 22.8 percent and 3.6 percent of the variance. The data may be efficiently reduced to three dimensions because the cumulative proportion reveals that the first three components account for more than 99% of the overall variance in the data.

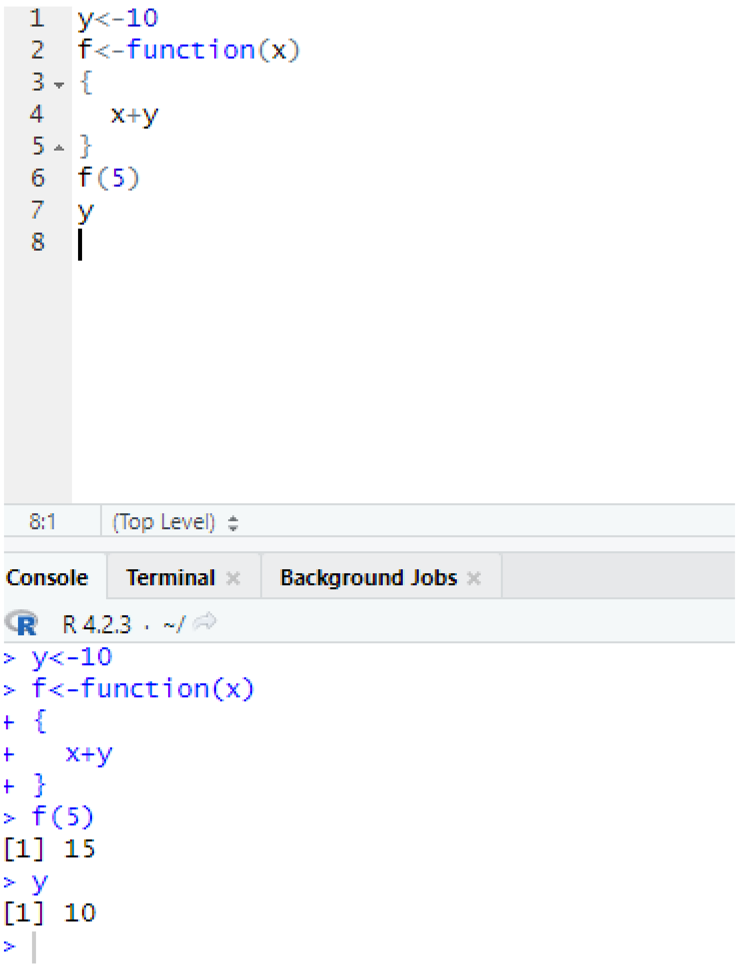
PROG NO:

DATE:

PG NO:

R NO:

How to write functions



```r
y<-10
f<-function(x)
{
  x+y
}
f(5)
y
```

8:1 (Top Level)

Console | Terminal | Background Jobs

R 4.2.3 · ~/

```
> y<-10
> f<-function(x)
+ {
+   x+y
+ }
> f(5)
[1] 15
> y
[1] 10
>
```

Statistical functions

R standard installation contains wide range of statistical functions. In this tutorial, we will briefly look at the most important function.

Basic statistic functions

Operator	Description
mean(x)	Mean of x
median(x)	Median of x
var(x)	Variance of x
sd(x)	Standard deviation of x
scale(x)	Standard scores (z-scores) of x
quantile(x)	The quartiles of x
summary(x)	Summary of x: mean, min, max etc..

```
speed <- dt$speed
speed
# Mean speed of cars dataset
mean(speed)
```

Output:

```
## [1] 15.4
# Median speed of cars dataset
median(speed)
```

Output:

```
## [1] 15
# Variance speed of cars dataset
var(speed)
```

Output:

```
## [1] 27.95918
# Standard deviation speed of car   dataset
sd(speed)
```

Output:

```
## [1] 5.287644
# Standardize vector speed of car   dataset
```

```
head(scale(speed), 5)
```

Output:

```
##            [,1]
## [1,] -2.155969
## [2,] -2.155969
## [3,] -1.588609
## [4,] -1.588609
## [5,] -1.399489
# Quantile speed of cars dataset
quantile(speed)
```

Output:

```
##   0%  25%  50%  75% 100%
##    4   12   15   19   25
# Summary speed of cars dataset
summary(speed)
```

Output:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     4.0    12.0    15.0    15.4    19.0    25.0
```

Up to this point, we have learned a lot of R built-in functions.

Note: Be careful with the class of the argument, i.e. numeric, Boolean or string. For instance, if we need to pass a string value, we need to enclose the string in quotation mark: "ABC" .

Contingency Tables in R Programming

Contingency tables are very useful to condense a large number of observations into smaller to make it easier to maintain tables. A contingency table shows the distribution of a variable in the rows and another in its columns. Contingency tables are not only useful for condensing data, but they also show the relations between variables. They are a way of summarizing categorical variables. A contingency table that deals with a single table are called a **complex or a flat contingency table.**

Making Contingency tables

A contingency table is a way to redraw data and assemble it into a table. And, it shows the layout of the original data in a manner that allows the reader to gain an overall summary of the original data. The **table()** function is used in R to create a contingency table. The **table()** function is one of the most versatile functions in R. It can take any data structure as an argument and turn it into a table. The more complex the original data, the more complex is the resulting contingency table.

Creating contingency tables from Vectors

In R a vector is an ordered collection of basic data types of a given length. The only key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures. It is the simplest data object from which you can create a contingency table.

Example:

```r
# R program to illustrate
# Contingency Table

# Creating a vector
vec = c(2, 4, 3, 1, 6, 3, 2, 1, 4, 5)

# Creating contingency table from vec using table()
conTable = table(vec)
print(conTable)
```

Output:

```
vec
1 2 3 4 5 6
2 2 2 2 1 1
```

In the given program what happens is first when we execute table() command on the vector it sorts the vector value and also prints the frequencies of every element given in the vector.

Creating contingency tables from Data
Now we will see a simple example that provides a data frame containing character values in one column and also containing a factor in one of its columns. This one column of factors contains character variables. In order to create our contingency table from data, we will make use of the table(). In the following example, the table() function returns a contingency table. Basically, it returns a tabular result of the categorical variables.

Example:

```r
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Creating contingency table from data using table()
conTable = table(df)
print(conTable)
```

Output:

```
            Gender
 Name     Female  Male
 Amiya      0     1
 Asish      0     1
 Rosy       1     0
```

Creating custom contingency tables

The contingency table in R can be created using only a part of the data which is in contrast with collecting data from all the rows and columns. We can create a custom contingency table in R using the following ways:

- Using Columns of a Data Frame in a Contingency Table
- Using Rows of a Data Frame in a Contingency Table
- By Rotating Data Frames in R
- Creating Contingency Tables from Matrix Objects in R

- **Using Columns of a Data Frame in a Contingency Table:** With the help of table() command, we are able to specify the columns with which the contingency tables can be created. In order to do so, you only need to pass the name of vector objects in the parameter of table() command.
  Example:

```r
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Creating contingency table by selecting a column
conTable = table(df$Name)
print(conTable)
```

Output:

```
Amiya Asish  Rosy

    1     1     1
```

From the output, you can notice that the table() command sorts the name in alphabetical order along with their frequencies of occurrence.

- **Using Rows of a Data Frame in a Contingency Table:** We can’t create a contingency table using rows of a data frame directly as we did in “using column” part. With the help

of the matrix, we can create a contingency table by looking at the rows of a data frame.
Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Creating contingency table by selecting rows
conTable = table(as.matrix(df[2:3, ]))
print(conTable)
```

Output:

```
Asish Female    Male    Rosy

  1       1       1       1
```

- **By Rotating Data Frames in R:** We can also create a contingency table by rotating a data frame in R. We can perform a rotation of the data, that is, transpose of the data using the **t()** command.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Rotating the data frame
newDf = t(df)

# Creating contingency table by rotating data frame
conTable = table(newDf)
```

```
print(conTable)
```

Output:

```
newDf

Amiya   Asish Female   Male    Rosy

  1       1       1      2       1
```

- **Creating Contingency Tables from Matrix Objects in R** A matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. Matrices are two-dimensional, homogeneous data structures. We can create a contingency table by using this matrix object.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a matrix
A = matrix(
  c(1, 2, 4, 1, 5, 6, 2, 4, 7),
  nrow = 3,
  ncol = 3
)

# Creating contingency table using matrix object
conTable = table(A)
print(conTable)
```

Output:

```
A

1 2 4 5 6 7

2 2 2 1 1 1
```

Converting Objects into tables
As mentioned above, a table is a special type of data object which is similar to the matrix but also possesses several differences.

- **Converting Matrix Objects into tables:** We can directly convert a matrix object into table by using the **as.table()** command. Just pass the matrix object as a parameter to the **as.table()** command.
  Example:

```
# R program to illustrate
# Contingency Table

# Creating a matrix
A = matrix(
  c(1, 2, 4, 1, 5, 6, 2, 4, 7),
  nrow = 3,
  ncol = 3
)

# Converting Matrix Objects into tables
newTable = as.table(A)
print(newTable)
```

Output:

```
  A B C
A 1 1 2
B 2 5 4
C 4 6 7
```

Converting Data frame Objects into table

We can’t directly convert a data frame object into table by using the **as.table()** command. In the case of a data frame, the object can be converted into the matrix, and then it can be converted into the table using **as.table()** command

Example:

```
# R program to illustrate
# Contingency Table

# Creating a data frame
df = data.frame(
  "Name" = c("Amiya", "Rosy", "Asish"),
  "Gender" = c("Male", "Female", "Male")
)

# Converting data frame object to matrix object
modifiedDf = as.matrix(df)

# Converting Matrix Objects into tables
newTable = as.table(modifiedDf)
print(newTable)
```

Output:

```
  Name  Gender
A Amiya Male
B Rosy  Female
C Asish Male
```

How to Perform a Chi-Square Goodness of Fit Test in R

A **Chi-Square Goodness of Fit Test** is used to determine whether or not a categorical variable follows a hypothesized distribution.

This tutorial explains how to perform a Chi-Square Goodness of Fit Test in R.

Example: Chi-Square Goodness of Fit Test in R

A shop owner claims that an equal number of customers come into his shop each weekday. To test this hypothesis, a researcher records the number of customers that come into the shop in a given week and finds the following:

- **Monday:** 50 customers
- **Tuesday:** 60 customers
- **Wednesday:** 40 customers
- **Thursday:** 47 customers
- **Friday:** 53 customers

Use the following steps to perform a Chi-Square goodness of fit test in R to determine if the data is consistent with the shop owner's claim.

Step 1: Create the data.

First, we will create two arrays to hold our observed frequencies and our expected proportion of customers for each day:

```
observed <- c(50, 60, 40, 47, 53)
expected <- c(.2, .2, .2, .2, .2) #must add up to 1
```

Step 2: Perform the Chi-Square Goodness of Fit Test.

Next, we can perform the Chi-Square Goodness of Fit Test using the **chisq.test()** function, which uses the following syntax:

chisq.test(x, p)

where:

- **x:** A numerical vector of observed frequencies.
- **p:** A numerical vector of expected proportions.

The following code shows how to use this function in our example:

```
#perform Chi-Square Goodness of Fit Test
chisq.test(x=observed, p=expected)
```

OUTPUT:

```
        Chi-squared test for given probabilities

data:  observed
X-squared = 4.36, df = 4, p-value = 0.3595
```

The Chi-Square test statistic is found to be **4.36** and the corresponding p-value is **0.3595**.

Note that the p-value corresponds to a Chi-Square value with n-1 degrees of freedom (dof), where n is the number of different categories. In this case, dof = 5-1 = 4.

You can use the Chi-Square to P Value Calculator to confirm that the p-value that corresponds to $X^2 = 4.36$ with dof = 4 is **0.35947**.

Recall that a Chi-Square Goodness of Fit Test uses the following null and alternative hypotheses:

- **H_0: (null hypothesis)** A variable follows a hypothesized distribution.
- **H_1: (alternative hypothesis)** A variable does not follow a hypothesized distribution.

Since the p-value (.35947) is not less than 0.05, we fail to reject the null hypothesis. This means

we do not have sufficient evidence to say that the true distribution of customers is different from the distribution that the shop owner claimed.

Programming in R:

1. **Write a R program to take input from the user(name and age)and display the values.also print the version of R installation**

```
name=readline(prompt="input your name:")
age=readline(prompt="input your age:")
print(paste("my name is",name,"and i am",age,"years old"))
print(R.version.string)
```

#OUTPUT:

```
input your name:
input your age:
[1] "my name is  and i am  years old"[1]
 "R version 4.1.2 (2021-11-01)"
```

2. Write a R program to get the details of the objects in memory.

```
name="python";

n1=10;

n2=0.5;

nums=c(10,20,30,40,50,60)

print(ls())

print("details of the object in memory:")

print(ls.str())

#OUTPUT:

[1] "n1"   "n2"   "name" "nums"

[1] "details of the object in memory:"

n1 :  num 10

n2 :  num 0.5

name :  chr "python"

nums :  num [1:6] 10 20 30 40 50 60
```

PROG NO:		PG NO:
DATE:		R NO:

3. Write a R program to create a sequence of numbers from 20 to 50 and find the mean the mean of numbers from 20 to 60 and sum of numbers from 51 to 91.

```
print("sequence of numbers from 20 to 50:")

print(seq(20,50))

print("mean of numbers from 20 to 60:")

print(mean(20:60))

print("sum of numbers from 51 to 91:")

print(sum(51:91))

#OUTPUT:

[1] "sequence of numbers from 20 to 50:"

 [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44

[26] 45 46 47 48 49 50

[1] "mean of numbers from 20 to 60:"

[1] 40

[1] "sum of numbers from 51 to 91:"

[1] 2911
```

4. Write a R program to create a vector which contains 10 random integers values between -50 and +50.

```
v=sample(-50:50,10,replace=TRUE)
print("content of the vector:")
print("10 random integer values between -50 and +50:")
print(v)


#OUTPUT:
[1] "content of the vector:"
[1] "10 random integer values between -50 and +50:"
 [1]  -6  12 -34  26  11 -19  13  20   0  19
```

5. Write a R program to get the first 10 fibonacci numbers.

```
fibonacci<-numeric(10)
fibonacci[1]<-fibonacci[2]<-1
for(i in 3:10)fibonacci[i]<-fibonacci[i-2]+fibonacci[i-1]
print("first 10 fibonacci numbers:")
print(fibonacci)
#OUTPUT:
[1] "first 10 fibonacci numbers:"
 [1]  1  1  2  3  5  8 13 21 34 55
```

6. Write a R program to extract first 10 english letters in loowercase and last 10 letters in uppercase and extract letters between 22nd to 24th letters in uppercase.

```
print("first 10 letters in lower case:")
t=head(letters,10)
print(t)
print("last 10 letters in uppercase:")
t=tail(LETTERS,10)
print(t)
print("letters between 22nd to 24th letter in uppercase:")
e=tail(LETTERS[22:24])
print(e)
```

```
#OUTPUT:
[1] "first 10 letters in lower case:"
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
[1] "last 10 letters in uppercase:"
 [1] "Q" "R" "S" "T" "U" "V" "W" "X" "Y" "Z"
[1] "letters between 22nd to 24th letter in uppercase:"
[1] "V" "W" "X"
```

PROG NO: DATE:		PG NO: R NO:

7. Write a R program to find th efactors of a given numbers.

```
print_factors=function(n)
{
  print(paste("The factors of ",n,"are:"))
  for(i in 1:n)
  {
    if((n%%i)==0)
    {
    print(i)
    }
  }
}
print_factors(4)
print_factors(7)
print_factors(12)
#OUTPUT:
[1] "The factors of  4 are:"
[1] 1
[1] 2
[1] 4
[1] "The factors of  7 are:"
[1] 1
[1] 7
[1] "The factors of  12 are:"
[1] 1
[1] 2
```

8. Write a R program to find the maximum and the minimum value of a given vector.

```
nums=c(10,20,30,40,50,60)
print("original vector:")
print(paste("maximum value of the said vector :",max(nums)))
print(paste("minimum value of the said vector:",min(nums)))
#OUTPUT:
[1] "original vector:"
[1] "maximum value of the said vector : 60"
[1] "minimum value of the said vector: 10"
```

9. Write a R program to get the unique elements of a given string & unique numbers of vectors.

```
str1="The quick brown fox jumps over the lazy dog"
print("original vector(string)")
print(str1)
print("inique elements of the said vector:")
print(unique (tolower(str1)))
nums=c(1,2,3,4,5,6)
print("original vector(number)")
print(nums)
print("unique elements of the said vector:")
print(unique(nums))
#OUTPUT:[1] "original vector(string)"
[1] "The quick brown fox jumps over the lazy dog"
[1] "inique elements of the said vector:"
[1] "the quick brown fox jumps over the lazy dog"
[1] "original vector(number)"
[1] 1 2 3 4 5 6
```

```
[1] "unique elements of the said vector:"
[1] 1 2 3 4 5 6
```

10. Write a R program to create an array ,passing in a vector & values & a vector of dimensions .also provide names

```
a=array(
6:30,
dim=c(4,3,2),
dimnames=list(
c("col1","col2","col3","col4"),
c("row1","row2","row3")
)
)
print(a)
#OUTPUT:
 row1 row2 row3
col1    6   10   14
col2    7   11   15
col3    8   12   16
col4    9   13   17
     row1 row2 row3
col1   18   22   26
col2   19   23   27
col3   20   24   28
col4   21   25   29
```

11. #Create a vector for matrix elements.

```
vector1=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)

matrix1<-matrix(vector1,nrow=4,ncol=4)

print(matrix1)

mul_vec=c(1,2,3,4)

print("Result")

print(matrix1*mul_vec)
```

```
#OUTPUT:
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    2    6   10   14
[3,]    3    7   11   15
[4,]    4    8   12   16
[1] "Result"
     [,1] [,2] [,3] [,4]
[1,]    1    5    9   13
[2,]    4   12   20   28
[3,]    9   21   33   45
[4,]   16   32   48   64
```

PROG NO: DATE:		PG NO: R NO:

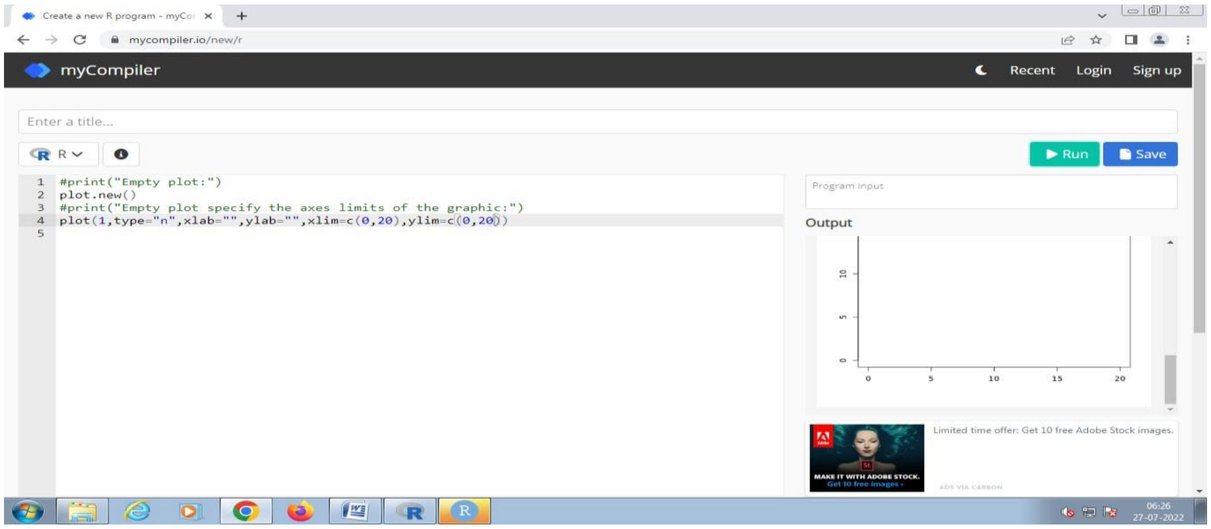
12. Write a R Program to create a list of elements using vectors,matrices and a functions.Print the content of the list.

```
l=list(
c(1,2,2,5,7,12),
month.abb,
matrix(c(3,-8,1,-3),nrow=2),
asin
)
print("counter of the list:")
print(l)
#OUTPUT:
 [1] "counter of the list:"
[[1]]
[1]  1  2  2  5  7 12


[[2]]
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"


[[3]]
     [,1] [,2]
[1,]    3    1
[2,]   -8   -3
[[4]]
function (x)  .Primitive("asin")
```

PROG NO:		PG NO:
DATE:		R NO:

13. Write a R Program to draw an empty plot and an empty plot specify the axes limits of the graph.

```
#print("Empty plot:")

plot.new()

#print("Empty plot specify the axes limits of the graphic:")

plot(1,type="n",xlab="",ylab="",xlim=c(0,20),ylim=c(0,20))
```



```
#print("Empty plot:")
plot.new()
#print("Empty plot specify the axes limits of the graphic:")
plot(1,type="n",xlab="",ylab="",xlim=c(0,20),ylim=c(0,20))
```

14. Write a R Program to create a simple bar plot of 5 subject marks.

marks=c(70,95,80,74)

barplot(marks,

main="comparing marks of 5 subjects",

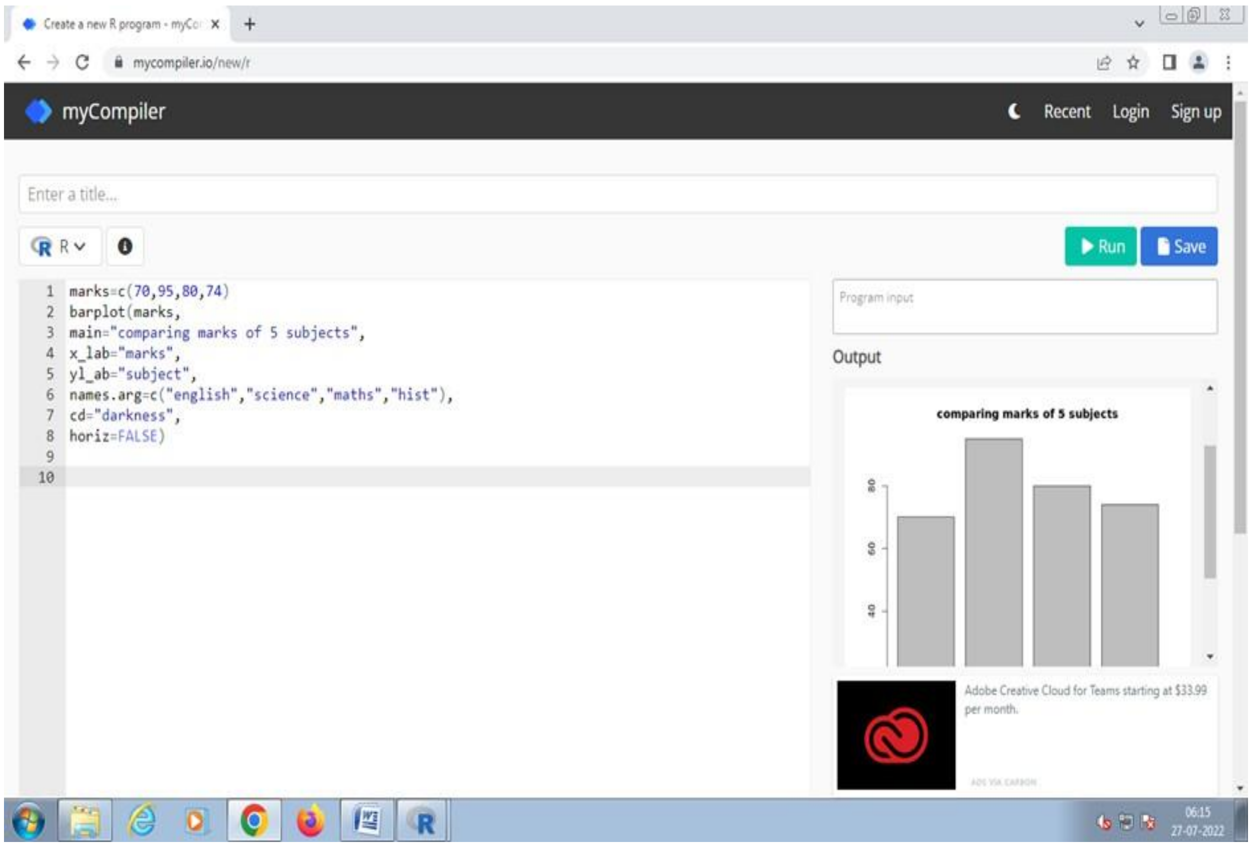
x_lab="marks",

yl_ab="subject",

names.arg=c("english","science","maths","hist"),

cd="darkness",

horiz=FALSE)


Create a new R program - myCo
mycompiler.io/new/r
myCompiler
Recent
Login
Sign up
Enter a title...
R
Run
Save
Program input
Output
comparing marks of 5 subjects
80
60
40
Adobe Creative Cloud for Teams starting at $33.99 per month.
06:15
27-07-2022

```
marks=c(70,95,80,74)
barplot(marks,
main="comparing marks of 5 subjects",
x_lab="marks",
yl_ab="subject",
names.arg=c("english","science","maths","hist"),
cd="darkness",
horiz=FALSE)
```

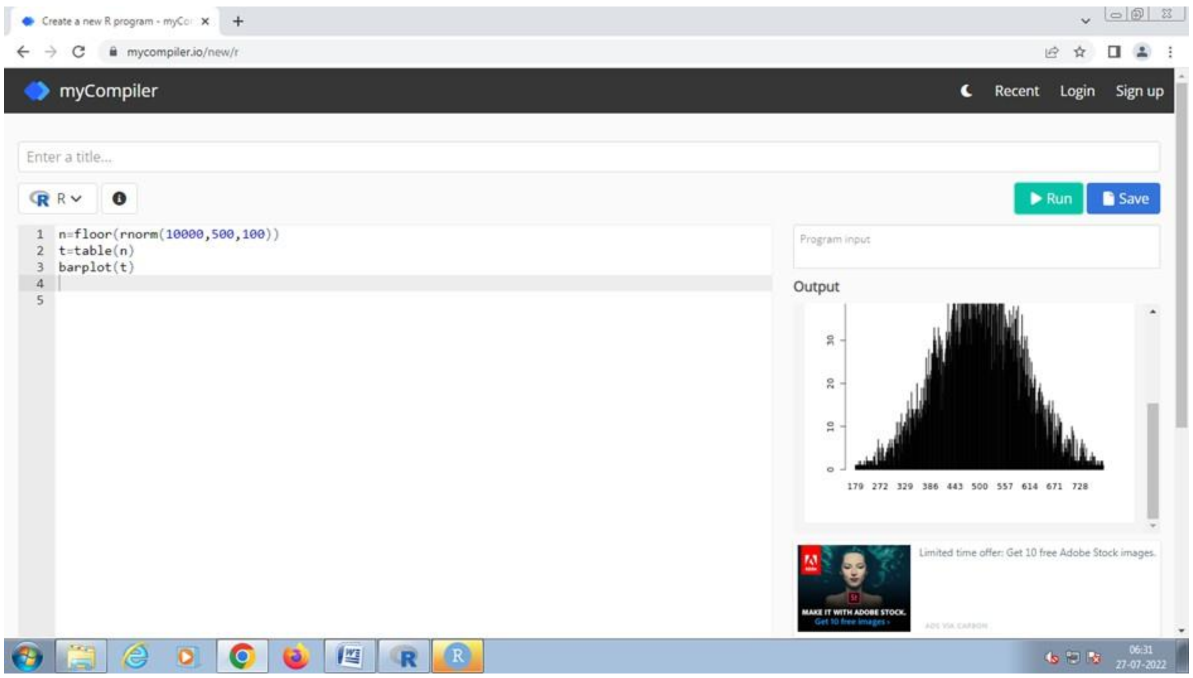
PROG NO: DATE:		PG NO: R NO:

15. Write a R program to create bell curve of a random normal distribution.

Sample Solution :

R Programming Code :

```
n = floor(rnorm(10000, 500, 100))

t = table(n)

barplot(t)
```



```
n=floor(rnorm(10000,500,100))
t=table(n)
barplot(t)
```

16. Write a R Program to compute sum,mean and product of a given elements.

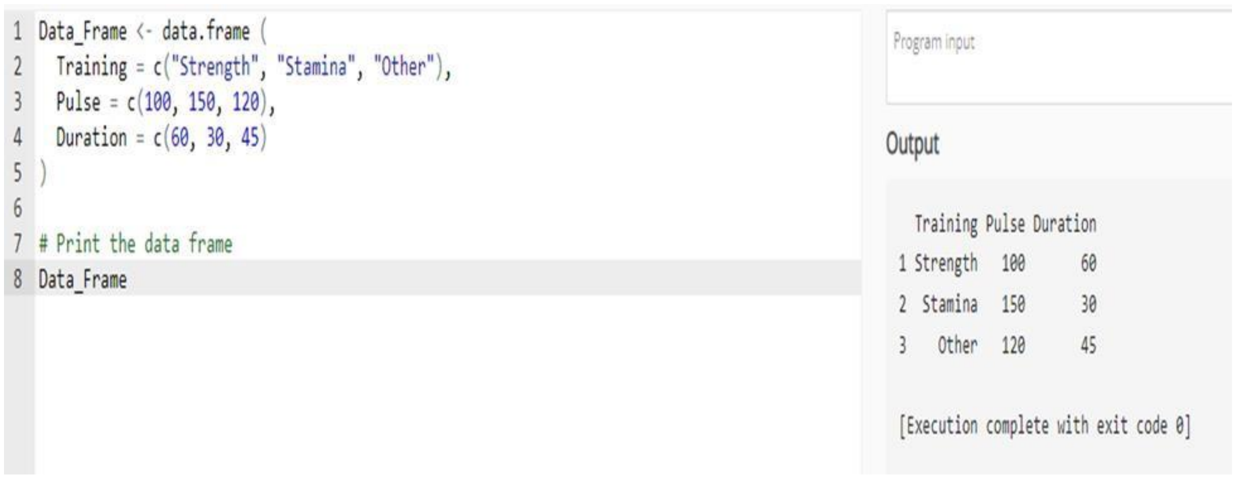
```
nums=c(10,20,30)

print("original vector:")

print(nums)

print(paste("sum of vector elements:",sum(nums)))

print(paste("mean of vector elements:",mean(nums)))

print(paste("product of vector elements:",prod(nums)))
```

```
#OUTPUT:

[1] "original vector:"

[1] 10 20 30

[1] "sum of vector elements: 60"

[1] "mean of vector elements: 20"

[1] "product of vector elements: 6000
```

17. Write a R Program to create a Dataframes which contain details of 5 employees and display the details.

```
employees=data.frame(name=c("Anastasia s","Dima R","Katherine S","James A","Laura Martin"),

gender=c("M","M","F","F","M"),

age=c(23,22,25,26,32),

designetion=c("clerk","manager","executive","CEO","assistant"),

SSN=c("123","124","125","126","127")

)

print("details of the employees:")

print(employees)

#OUTPUT:

[1] "details of the employees:"

        name gender age designetion SSN

1  Anastasia s     M  23      clerk 123
```

18.Write a R program to create DataFrame for 3 rows and 3 columns



```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
Data_Frame
```

Program input

Output

```
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45

[Execution complete with exit code 0]
```