Unit-3: Introduction to R and working with Data

3.1 Overview of R and its applications in data analysis and statistics

R is a programming language and software environment for statistical analysis, graphics representation and reporting. So, it can be said that R is a statistical computing and graphics system. The widespread usage of R in fields such as data science, machine learning, and statistical modelling has made it one of the most sought-after programming languages.

It is used by statisticians, data scientists, and researchers. The R Language offers an extensive suite of packages and libraries tailored for data manipulation, statistical modelling, and visualization.

R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

R programming language is an implementation of the S programming language. So it is said that the modern R, a part of the GNU free software project, was based on S.

R is comprised of two parts: the R language itself and a run-time environment.

R is an interpreted language, which means that users access its functions through a command-line interpreter. Moreover, multiple third-party graphical user interfaces are available, such as RStudio—an integrated development environment—and Jupyter—a notebook interface. It is written primarily in C, Fortran, and R itself. Precompiled executables are provided for various operating systems.

Unlike languages such as Python and Java, R is not a general-purpose programming language. Instead, it's considered a domain-specific language (DSL), meaning its functions and use are designed for a specific area of use or domain. R is equipped with a large set of functions that enable data visualizations, so users can analyze data, model it as required, and then create graphics.

Evolution of R

R was initially written by Ross Ihaka and Robert Gentleman at the Department of Statistics of the University of Auckland in Auckland, New Zealand. R made its first appearance in 1993.

A large group of individuals has contributed to R by sending code and bug reports.

Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

The R language got its name for two reasons: because R is the first letter in the inventors' names, and because R is a play on the name of its parent language S, which was originally developed by Bell Telephone Laboratories.

Features of R

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,

- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.
- R supports both procedural programming and object-oriented programming. Procedural programming includes the procedure, records, modules, and procedure calls. While object-oriented programming language includes class, objects, and generic functions.
- R is an open-source language and it is available for free for everyone to use for statistical and graphical purposes.

Why Use R Language?

The R Language is a powerful tool widely used for data analysis, statistical computing, and machine learning. Here are several reasons why professionals across various fields prefer R:

1. Comprehensive Statistical Analysis:

R language is specifically designed for statistical analysis and provides a vast array of statistical techniques and tests, making it ideal for data-driven research.

2. Extensive Packages and Libraries:

The R Language boasts a rich ecosystem of packages and libraries that extend its capabilities, allowing users to perform advanced data manipulation, visualization, and machine learning tasks with ease.

3. Strong Data Visualization Capabilities:

R language excels in data visualization, offering powerful tools like ggplot2 and plotly, which enable the creation of detailed and aesthetically pleasing graphs and plots.

4. Open Source and Free:

As an open-source language, R is free to use, which makes it accessible to everyone, from individual researchers to large organizations, without the need for costly licenses.

5. Platform Independence:

The R Language is platform-independent, meaning it can run on various operating systems, including Windows, macOS, and Linux, providing flexibility in development environments.

6. Integration with Other Languages:

R can easily integrate with other programming languages such as C, C++, Python, and Java, allowing for seamless interaction with different data sources and statistical packages.

7. Growing Community and Support:

R language has a large and active community of users and developers who contribute to its continuous improvement and provide extensive support through forums, mailing lists, and online resources.

8. High Demand in Data Science:

R is one of the most requested programming languages in the Data Science job market, making it a valuable skill for professionals looking to advance their careers in this field.

Advantages of R language

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

Disadvantages of R language

- In the R programming language, the standard of some packages is less than perfect.
- Although, R commands give little pressure on memory management. So R programming language may consume all available memory.
- In R basically, nobody to complain if something doesn’t work.
- R programming language is much slower than other programming languages such as Python and MATLAB.

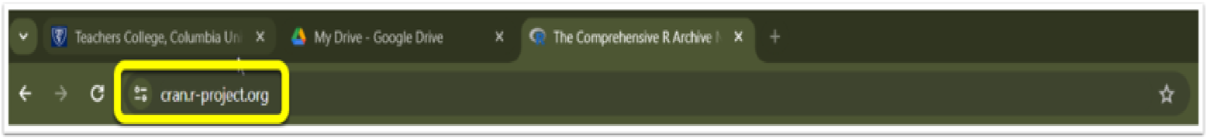
Applications of R language

- We use R for Data Science. It gives us a broad variety of libraries related to statistics. It also provides the environment for statistical computing and design.
- R is used by many quantitative analysts as its programming tool. Thus, it helps in data importing and cleaning.
- R is the most prevalent language. So many data analysts and research programmers use it.
- It is used as a fundamental tool for finance.
- Tech giants like Google, Facebook, Bing, Twitter, Accenture, Wipro, and many more using R nowadays.

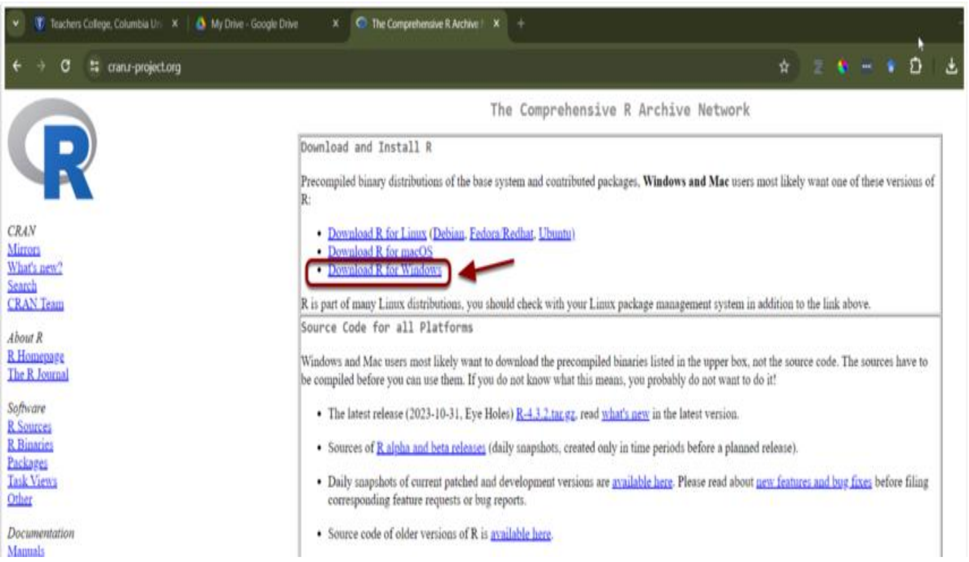
3.2 Installing R and Rstudio

1. To install R, go to cran.r-project.org

```
cran.r-project.org
```



2. Click Download R for Windows.



The Comprehensive R Archive Network

CRAN
Mirrors
What's new?
Search
CRAN Team

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Task Views
Other

Documentation
Manuals

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

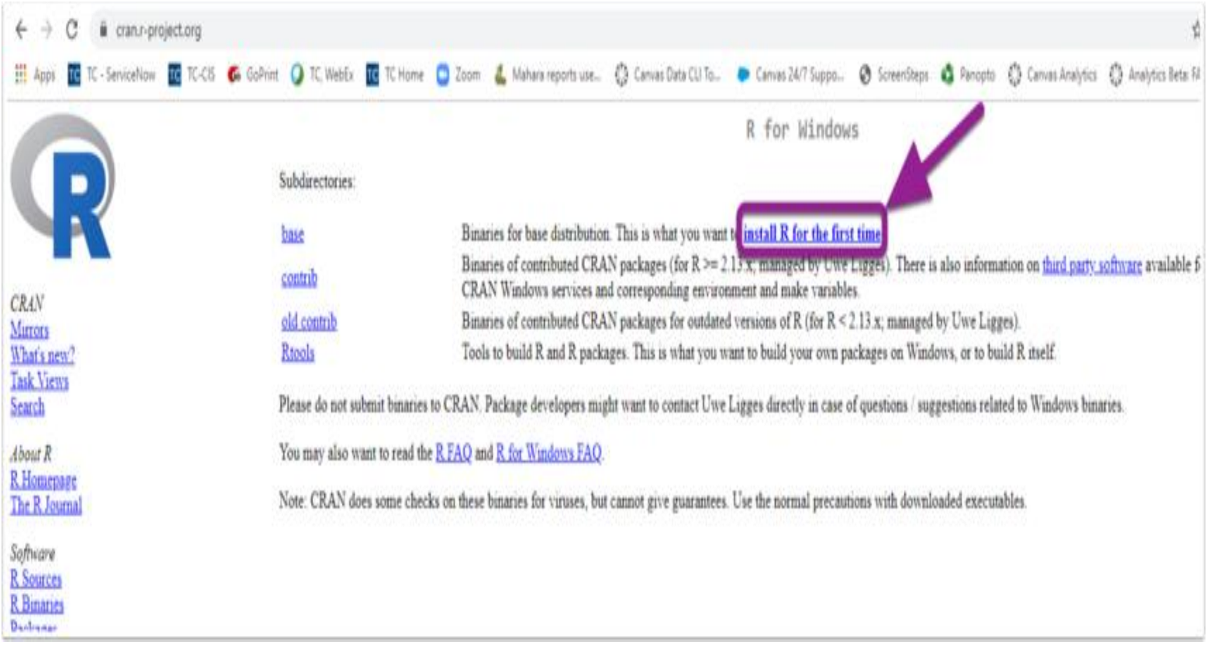
R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

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 (2023-10-31, Eye Holes) R-4.3.2.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.

3. Click on install R for the first time.



R for Windows

CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries

Subdirectories:

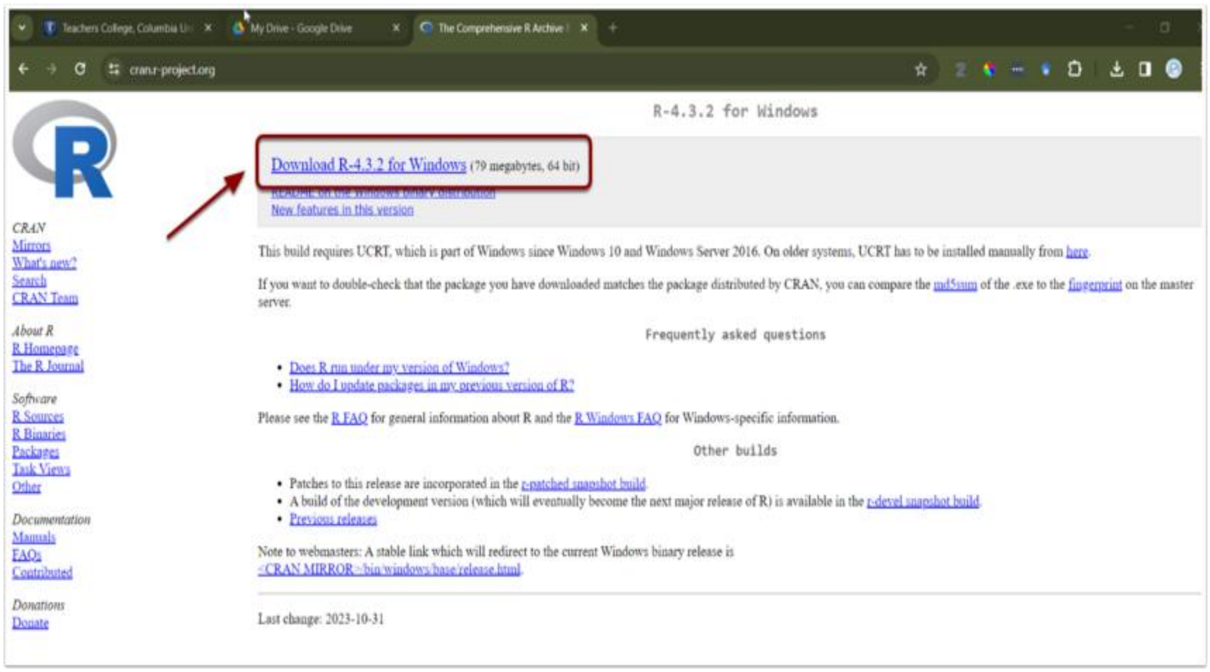
base	Binaries for base distribution. This is what you want to install R for the first time.
contrib	Binaries of contributed CRAN packages (for R >= 2.13.x; managed by Uwe Ligges). There is also information on third party software available for CRAN Windows services and corresponding environment and make variables.
old contrib	Binaries of contributed CRAN packages for outdated versions of R (for R < 2.13.x; managed by Uwe Ligges).
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 to Windows binaries.

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 executables.

4. Click Download R for Windows. Open the downloaded file.



R-4.3.2 for Windows

Download R-4.3.2 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 installed manually from here.

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the 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.

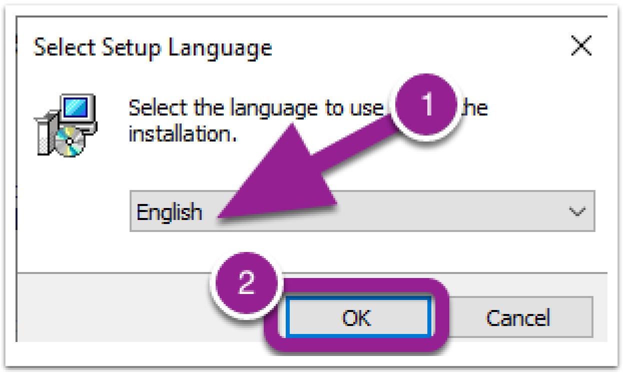
Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is
<CRAN MIRROR>/bin/windows/base/release.html.

Last change: 2023-10-31

5. Select the language you would like to use during the installation. Then click OK.

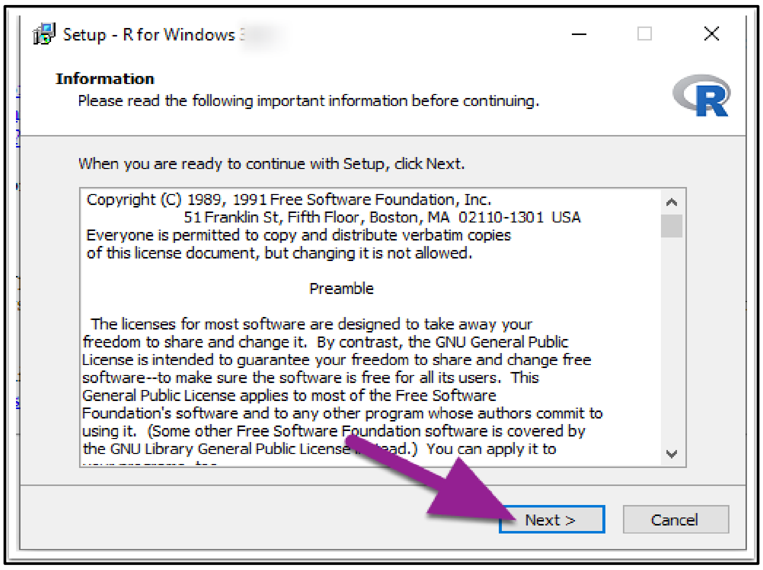


Select Setup Language

Select the language to use during the installation.

English

6. Click Next.



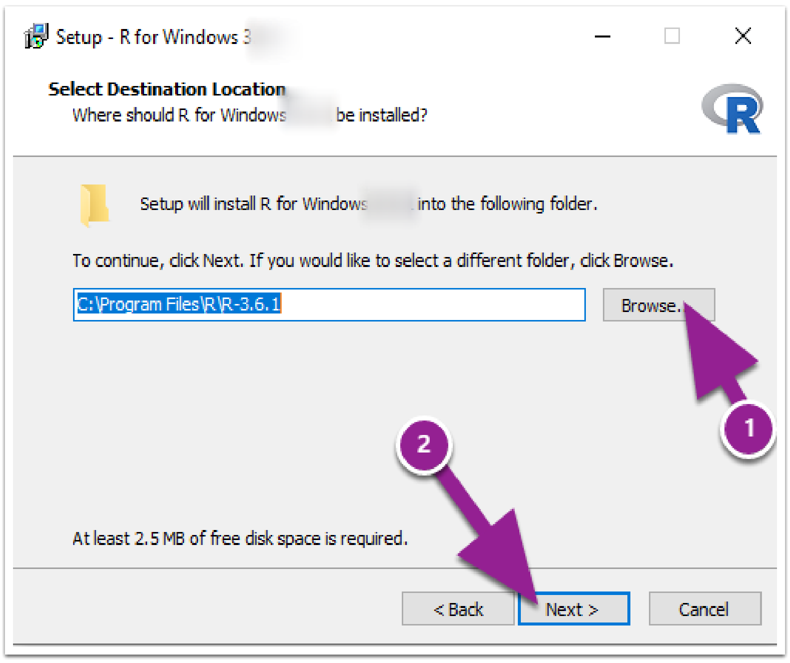
Setup - R for Windows

Information
Please read the following important information before continuing.

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

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

                            Preamble

 The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Library General Public License instead.)  You can apply it to
```

7. Select where you would like R to be installed. It will default to your Program Files on your C Drive. Click Next.


Setup - R for Windows 3

Select Destination Location

Where should R for Windows be installed?

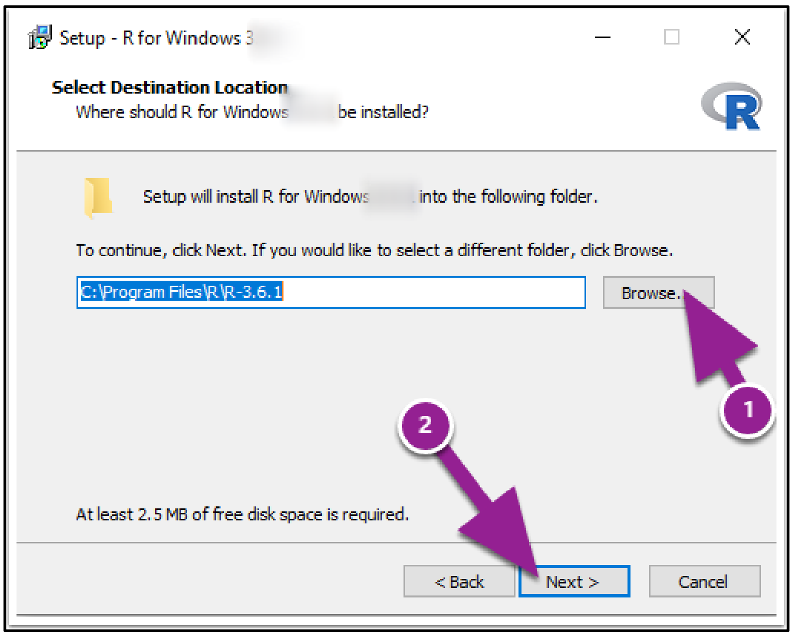
Setup will install R for Windows into the following folder.

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

C:\Program Files\R\R-3.6.1

At least 2.5 MB of free disk space is required.

8. You can then choose which installation you would like.


Setup - R for Windows 3

Select Destination Location

Where should R for Windows be installed?

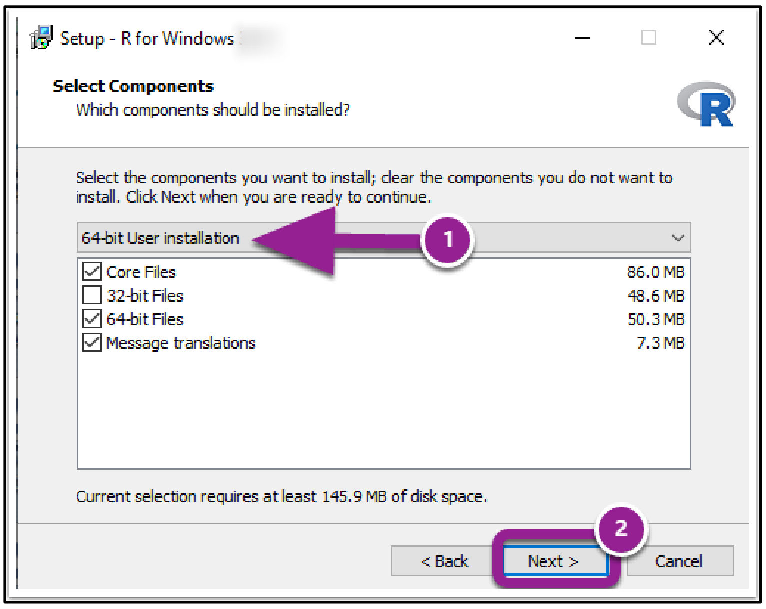
Setup will install R for Windows into the following folder.

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

C:\Program Files\R\R-3.6.1

At least 2.5 MB of free disk space is required.

9. (Optional) If your computer is a 64-bit, you can choose the 64-bit User Installation. Then click Next.


Setup - R for Windows

Select Components

Which components should be installed?

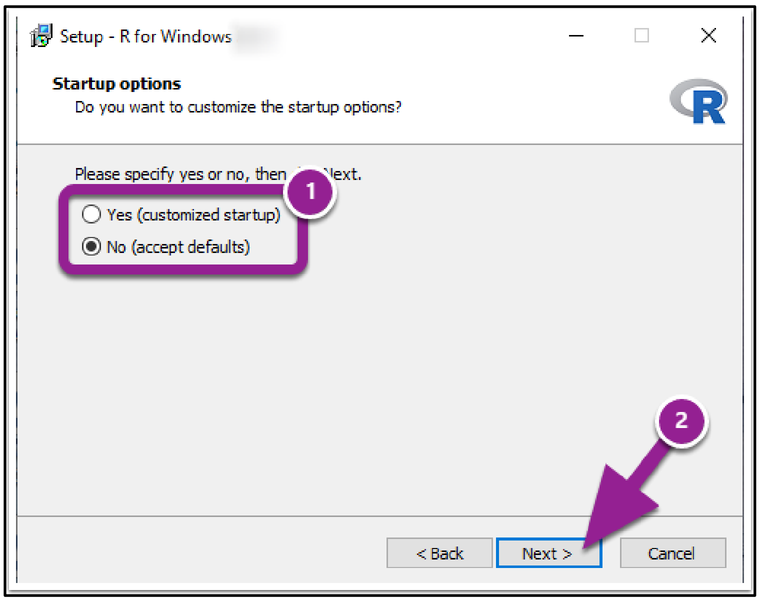
Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

64-bit User installation

[✓] Core Files	86.0 MB
[ ] 32-bit Files	48.6 MB
[✓] 64-bit Files	50.3 MB
[✓] Message translations	7.3 MB

Current selection requires at least 145.9 MB of disk space.

10. Then specify if you want to customized your startup or just use the defaults. Then click Next.


Setup - R for Windows

Startup options

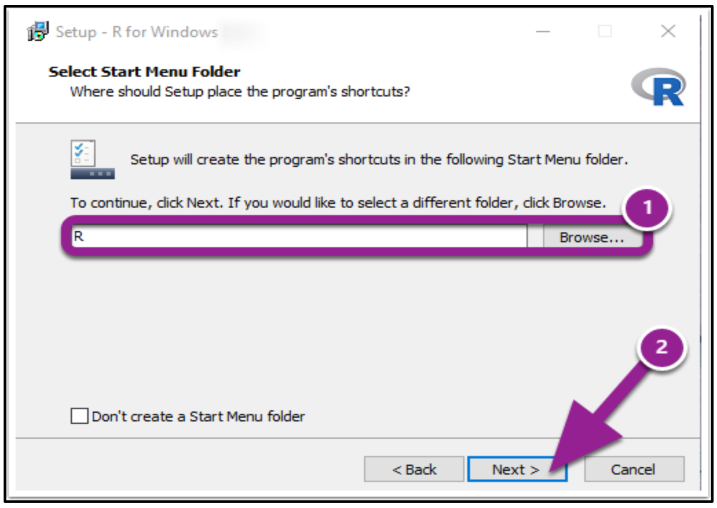
Do you want to customize the startup options?

Please specify yes or no, then click Next.

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

11. Then you can choose the folder that you want R to be saved within or the default if the R folder that was created. Once you have finished, click Next.

You can also choose if you do not want a Start Menu folder at the bottom.


Setup - R for Windows

Select Start Menu Folder

Where should Setup place the program's shortcuts?

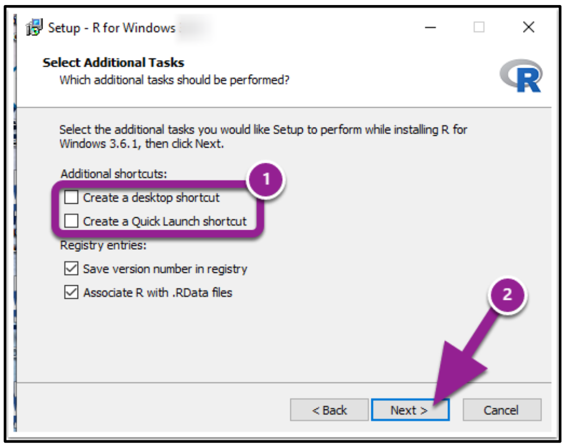
Setup will create the program's shortcuts in the following Start Menu folder.

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

R

[ ] Don't create a Start Menu folder

12. You can then select additional shortcuts if you would like. Click Next.


Setup - R for Windows

Select Additional Tasks

Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing R for Windows 3.6.1, then click Next.

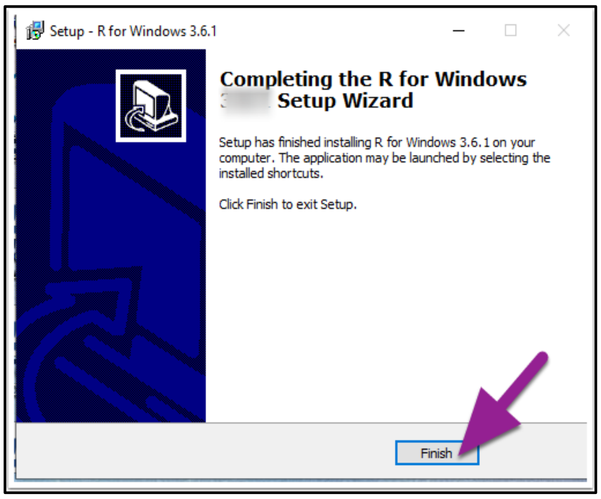
Additional shortcuts:

- [ ] Create a desktop shortcut
- [ ] Create a Quick Launch shortcut

Registry entries:

- [x] Save version number in registry
- [x] Associate R with .RData files

13. Click Finish.

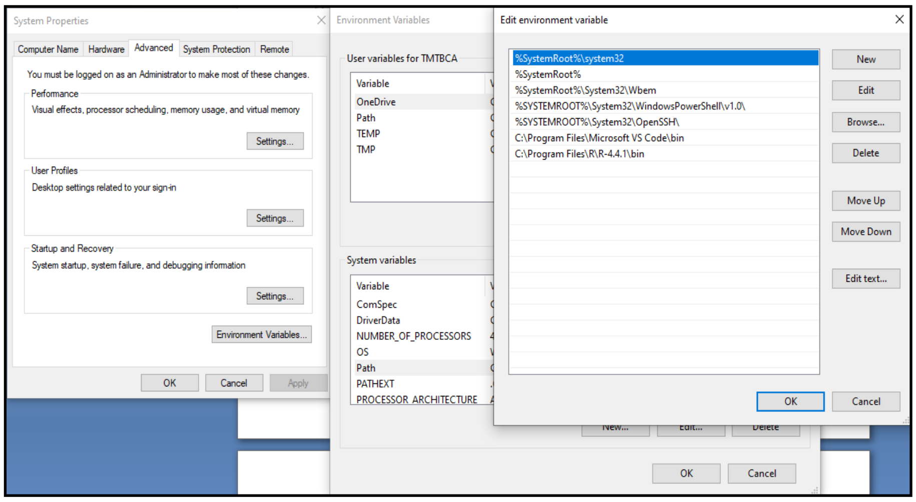

Setup - R for Windows 3.6.1

Completing the R for Windows Setup Wizard

Setup has finished installing R for Windows 3.6.1 on your computer. The application may be launched by selecting the installed shortcuts.

Click Finish to exit Setup.

14. Set environment variable


System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

Environment Variables

User variables for TMTBCA

Variable
OneDrive
Path
TEMP
TMP

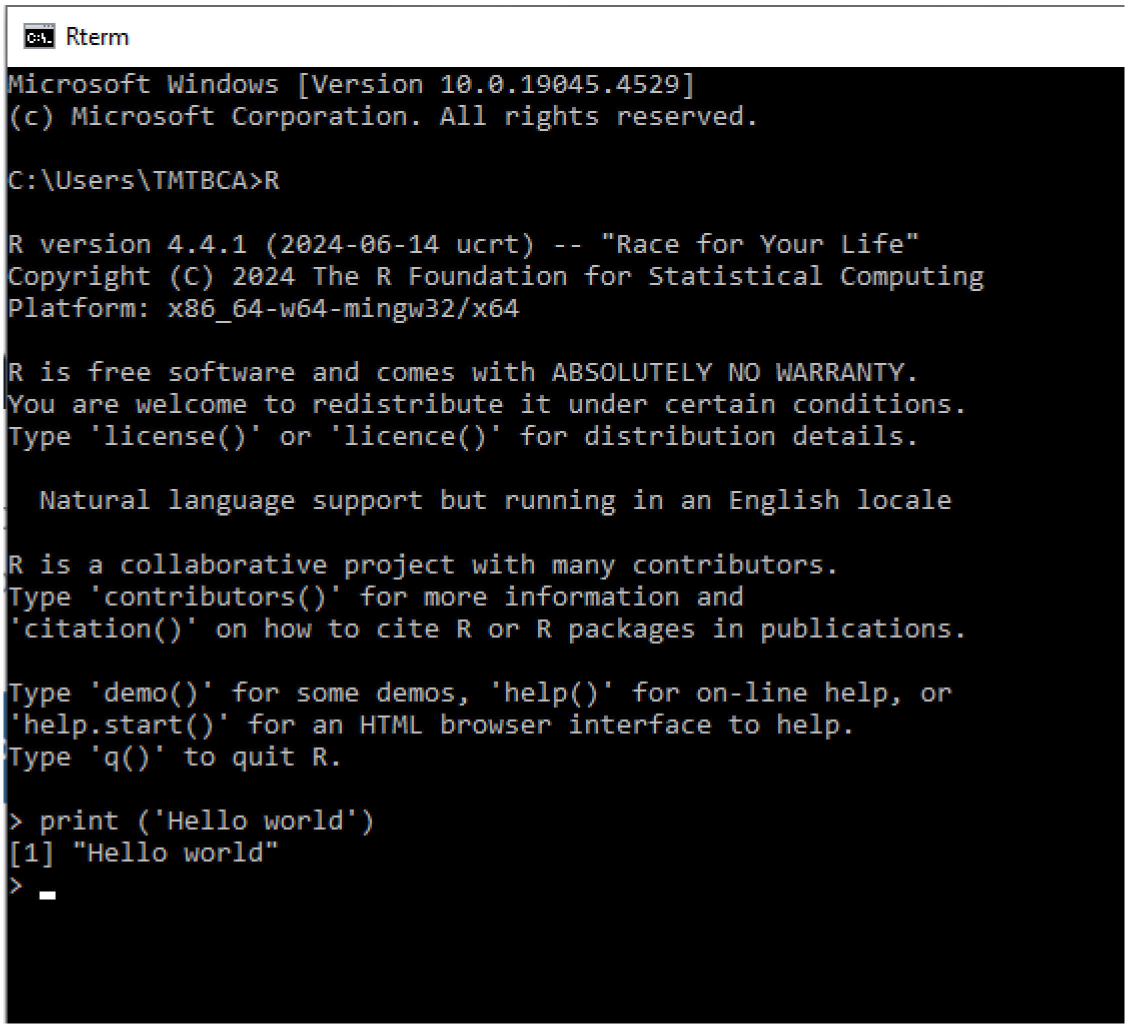
System variables

Variable
ComSpec
DriverData
NUMBER_OF_PROCESSORS
OS
Path
PATHEXT
PROCESSOR_ARCHITECTURE

Edit environment variable

```
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Microsoft VS Code\bin
C:\Program Files\R\R-4.4.1\bin
```

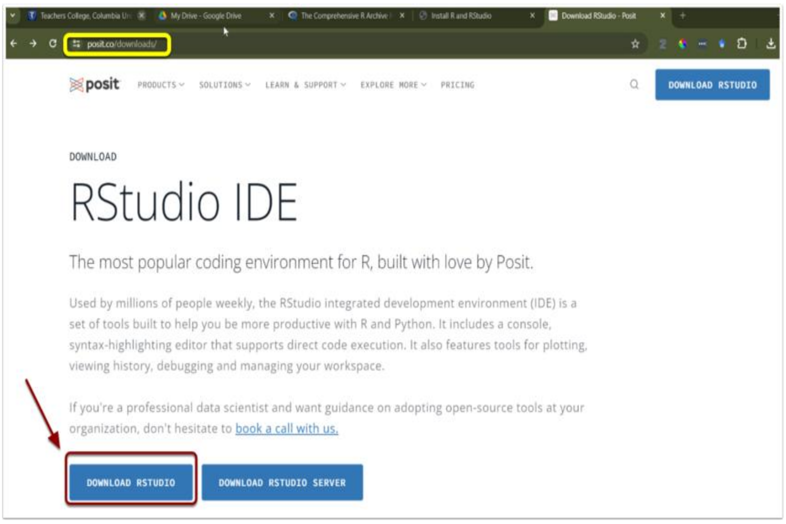
15. Open Command Prompt with cmd command and Type 'R' and press enter



```
Rterm
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\TMTBCA>R

R version 4.4.1 (2024-06-14 ucrt) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64

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.

> print ('Hello world')
[1] "Hello world"
> 
```

16. Next, download RStudio. Go to https://posit.co/downloads/

`https://posit.co/downloads/`



DOWNLOAD

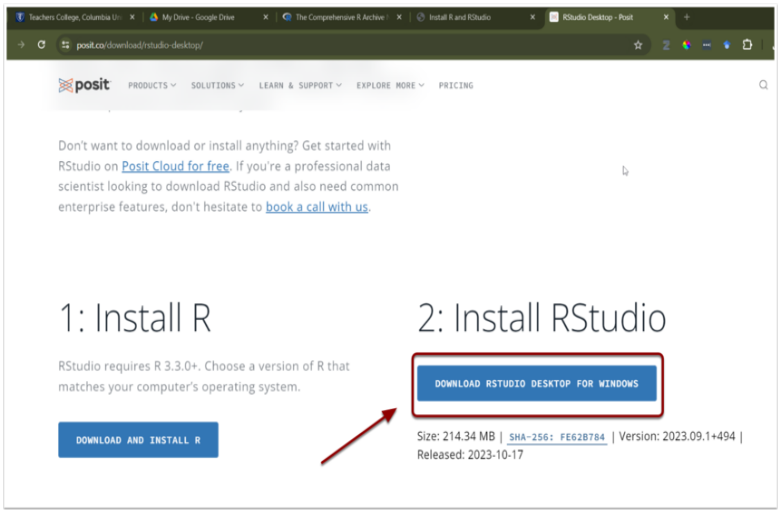
RStudio IDE

The most popular coding environment for R, built with love by Posit.

Used by millions of people weekly, the RStudio integrated development environment (IDE) is a set of tools built to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution. It also features tools for plotting, viewing history, debugging and managing your workspace.

If you're a professional data scientist and want guidance on adopting open-source tools at your organization, don't hesitate to book a call with us.

17. Click Download RStudio.



Don't want to download or install anything? Get started with RStudio on Posit Cloud for free. If you're a professional data scientist looking to download RStudio and also need common enterprise features, don't hesitate to book a call with us.

1: Install R

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

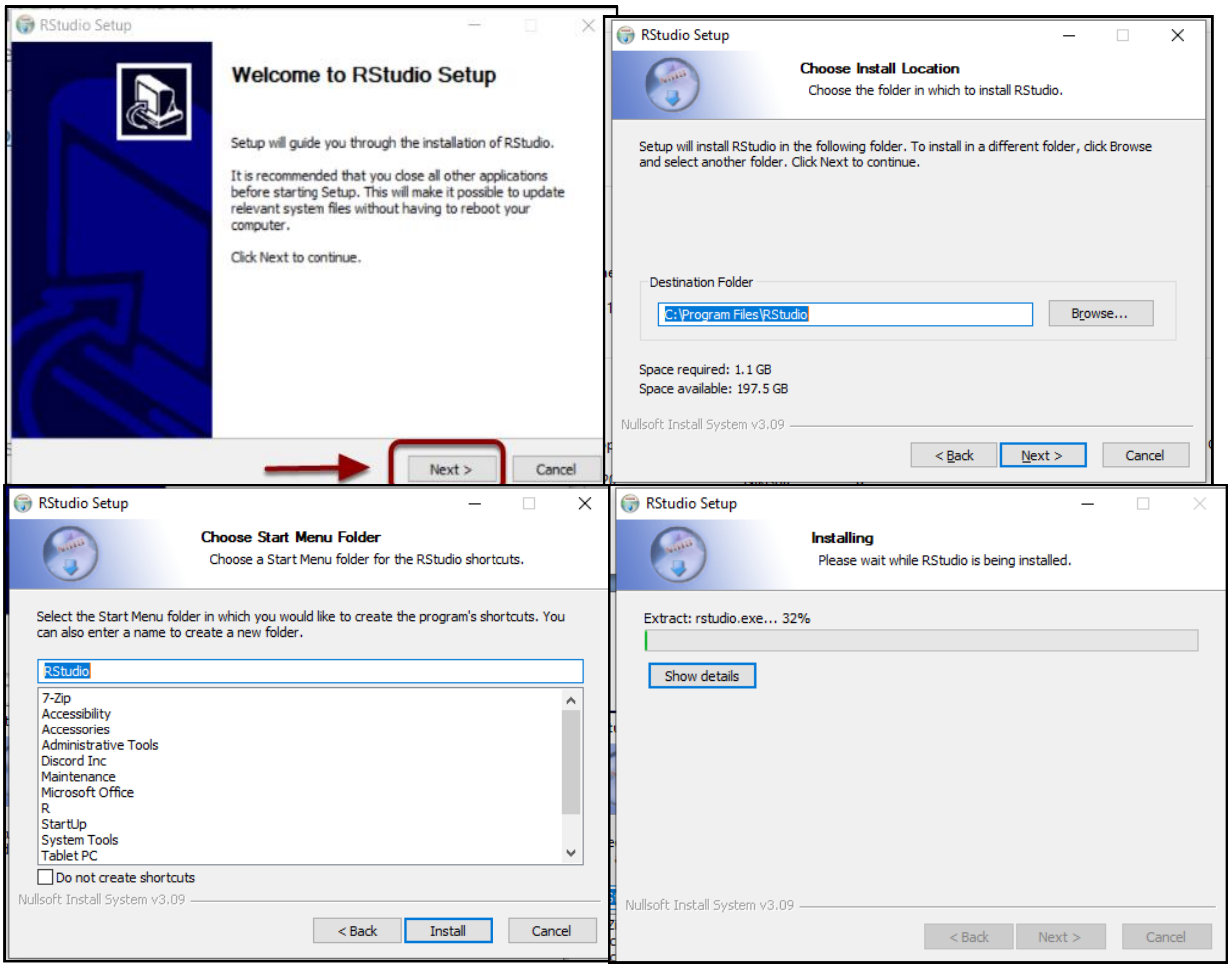
DOWNLOAD AND INSTALL R

2: Install RStudio

DOWNLOAD RSTUDIO DESKTOP FOR WINDOWS

Size: 214.34 MB | SHA-256: FE628784 | Version: 2023.09.1+494 |
Released: 2023-10-17

18. Once the packet has downloaded, the Welcome to RStudio Setup Wizard will open. Click Next and go through the installation steps.



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.

Next > Cancel

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
Browse...

Space required: 1.1 GB
Space available: 197.5 GB

Nullsoft Install System v3.09

< Back Next > Cancel

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

7-Zip
Accessibility
Accessories
Administrative Tools
Discord Inc
Maintenance
Microsoft Office
R
StartUp
System Tools
Tablet PC

Do not create shortcuts

Nullsoft Install System v3.09

< Back Install Cancel

RStudio Setup

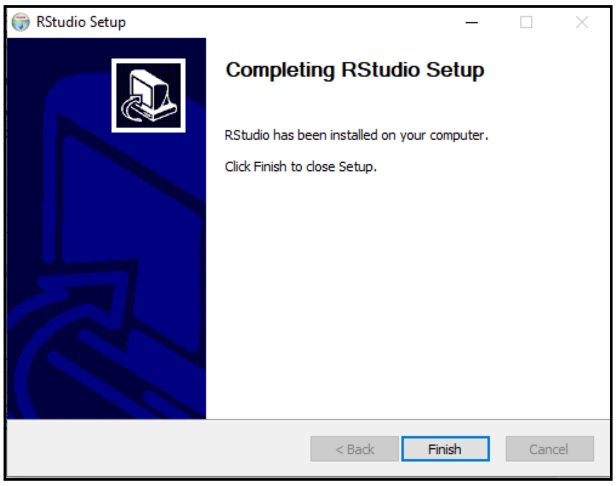
Installing
Please wait while RStudio is being installed.

Extract: rstudio.exe... 32%

Show details

Nullsoft Install System v3.09

< Back Next > Cancel



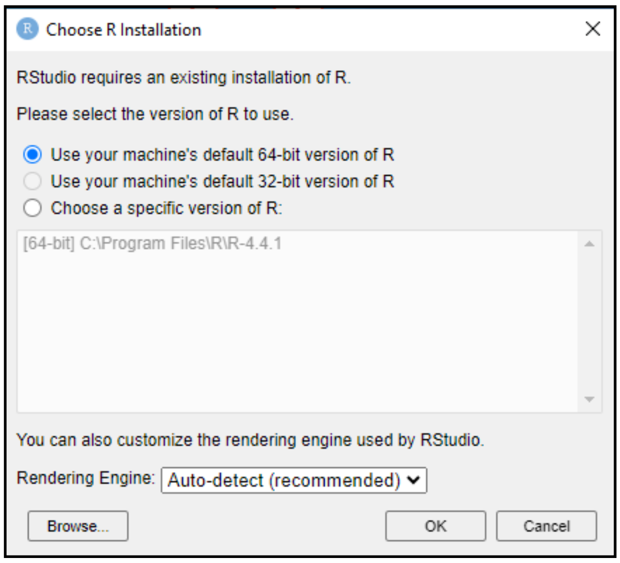
RStudio Setup

Completing RStudio Setup

RStudio has been installed on your computer.

Click Finish to close Setup.

19. After the Setup Wizard finishing the installation, RStudio will open.



Choose R Installation

RStudio requires an existing installation of R.

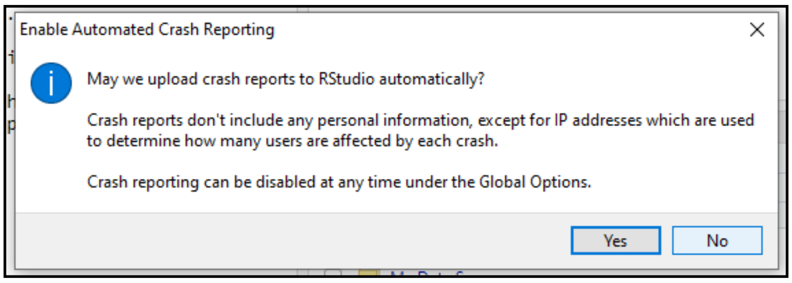
Please select the version of R to use.

- [✓ Use your machine's default 64-bit version of R]
- Use your machine's default 32-bit version of R
- Choose a specific version of R:

[64-bit] C:\Program Files\R\R-4.4.1

You can also customize the rendering engine used by RStudio.

Rendering Engine: Auto-detect (recommended)



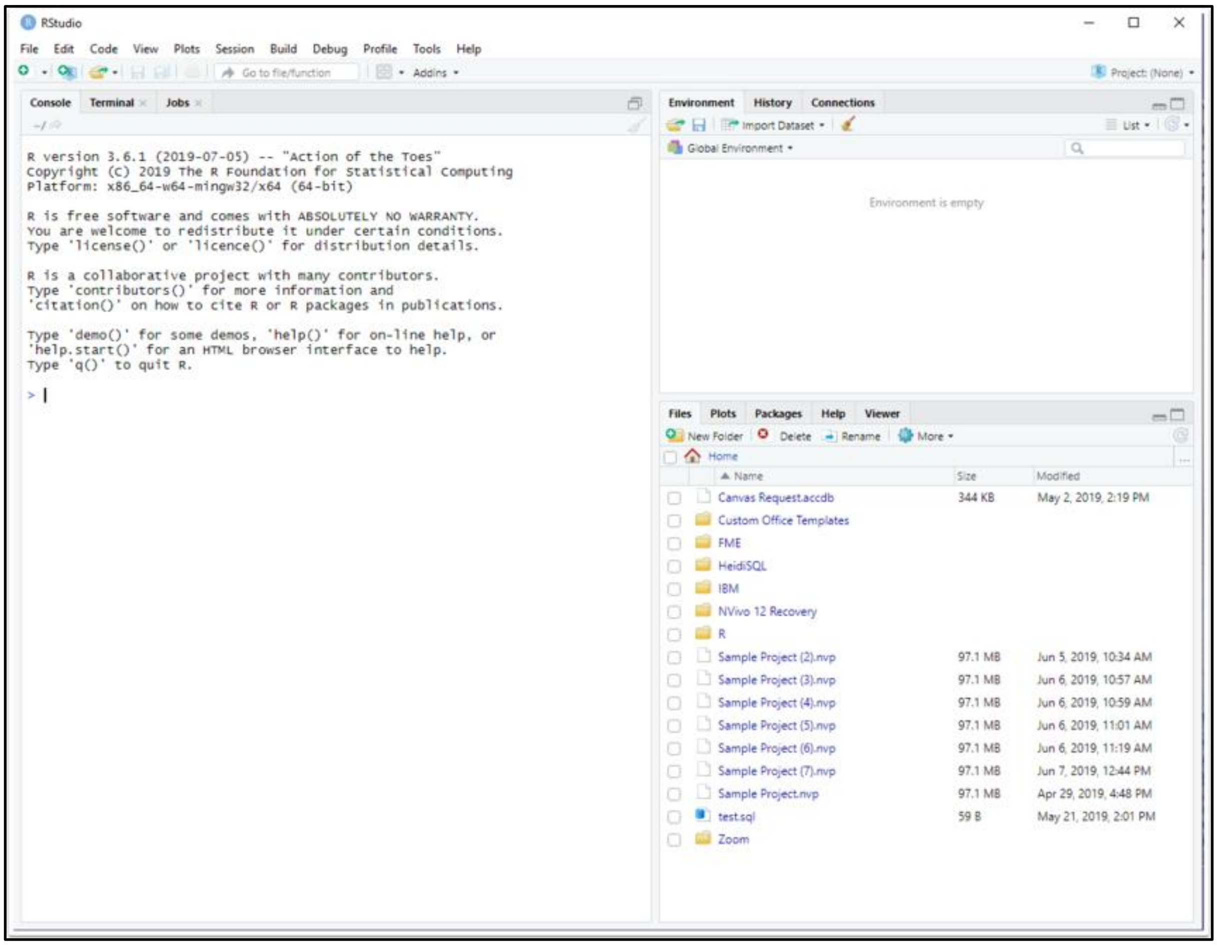
Enable Automated Crash Reporting

May we upload crash reports to RStudio automatically?

Crash reports don't include any personal information, except for IP addresses which are used to determine how many users are affected by each crash.

Crash reporting can be disabled at any time under the Global Options.

20. Open R Studio



```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 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.

> 
```

Name	Size	Modified
Canvas Request.accdb	344 KB	May 2, 2019, 2:19 PM
Custom Office Templates		
FME		
HeidiSQL		
IBM		
NVivo 12 Recovery		
R		
Sample Project (2).nvp	97.1 MB	Jun 5, 2019, 10:34 AM
Sample Project (3).nvp	97.1 MB	Jun 6, 2019, 10:57 AM
Sample Project (4).nvp	97.1 MB	Jun 6, 2019, 10:59 AM
Sample Project (5).nvp	97.1 MB	Jun 6, 2019, 11:01 AM
Sample Project (6).nvp	97.1 MB	Jun 6, 2019, 11:19 AM
Sample Project (7).nvp	97.1 MB	Jun 7, 2019, 12:44 PM
Sample Project.nvp	97.1 MB	Apr 29, 2019, 4:48 PM
test.sql	59 B	May 21, 2019, 2:01 PM
Zoom		

Why use R Studio?

R Studio is preferred for several reasons:

- **Integrated Development Environment (IDE)**: It provides a user-friendly interface designed specifically for R programming, offering features like syntax highlighting, code completion, and debugging tools that enhance productivity and code quality, making it ideal after completing the R language installation.
- **Project Management**: R Studio facilitates efficient project organization with its project-based workflow, allowing users to manage multiple scripts, data files, and plots within a cohesive workspace, which is essential for organizing projects after the R language installation.
- **Data Visualization**: It supports powerful data visualization capabilities through integrated plotting tools and compatibility with popular visualization libraries like ggplot2. It enables users to create insightful graphs and charts effortlessly, enhancing data representation after the R language installation.
- **Package Management**: R Studio simplifies package management with tools like the Package Manager and integrated CRAN repository access, making it easy to install, update, and manage R packages crucial for various analytical tasks, supporting package management post R language installation.
- **Markdown and R Markdown Support**: It seamlessly integrates with Markdown and R Markdown, enabling users to create dynamic reports, presentations, and documents that combine code, visualizations, and narrative text in a single file, facilitating report generation after R language installation.

- **Collaboration and Sharing**: R Studio facilitates collaboration by supporting version control systems like Git and enabling seamless sharing of projects and analyses through R Studio Server and R Studio Cloud, promoting collaboration after the R language installation.

3.3 Basic R syntax, variables, and data types.

Datatypes

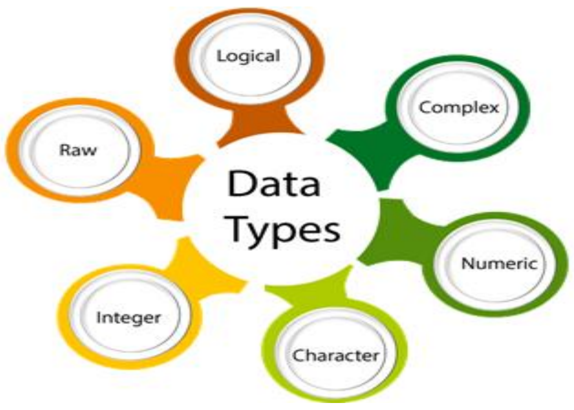
Different forms of data that can be saved and manipulated are defined and categorized using data types in computer languages, including R. Each R data type has unique properties and associated operations.

R Data types are used to specify the kind of data that can be stored in a variable.

For effective memory consumption and precise computation, the right data type must be selected.

Each R data type has its own set of regulations and restrictions. The operating system allocates memory based on the data type of the variable and decides what can be stored in the reserved memory.

There are the following data types which are used in R programming:



Data type	Example	Description
Logical	True, False	It is a special data type for data with only two possible values which can be construed as true/false.
Numeric	12,32,112,5432	Decimal value is called numeric in R, and it is the default computational data type.
Integer	3L, 66L, 2346L	Here, L tells R to store the value as an integer,
Complex	Z=1+2i, t=7+3i	A complex value in R is defined as the pure imaginary value i.
Character	'a', '"good"', "TRUE", '35.4'	In R programming, a character is used to represent string values. We convert objects into character values with the help of as.character() function.
Raw	Single_raw<- as.raw(255)	A raw data type is used to holds raw bytes.To save and work with data at the byte level in R, use the raw data type. By displaying a series of unprocessed bytes, it enables low-level operations on binary data.

R Variables – Creating, Naming and Using Variables in R

A variable is a memory allocated for the storage of specific data and the name associated with the variable is used to work around this reserved block.

The name given to a variable is known as its variable name. Usually, a single variable stores only the data belonging to a certain data type.

The name is so given to them because when the program executes there is subject to change hence it varies from time to time.

Variables in R

R Programming Language is a dynamically typed language, i.e. the R Language Variables are not declared with a data type rather they take the data type of the R-object assigned to them.

This feature is also shown in languages like Python and PHP.

Creating Variables in R Language

Let's look at ways of declaring and initializing variables in R language:

R supports three ways of variable assignment:

- Using equal operator- an equal sign to assign values to variables.
- Using the leftward operator- data is copied from right to left.
- Using the rightward operator- data is copied from left to right.

Syntax for creating R Variables

Types of Variable Creation in R:

Using equal to operators

```
variable_name = value
```

using leftward operator

```
variable_name<- value
```

using rightward operator

```
value ->variable_name
```

Let's look at the example of creating Variables in R:

```
# using equal to operator

var1 = "hello"

print(var1)


# using leftward operator

var2 <- "hello"

print(var2)


# using rightward operator

"hello" -> var3

print(var3)
```

Output

```
[1] "hello"

[1] "hello"

[1] "hello"
```

Nomenclature (Naming convention) of R Variables

The following rules need to be kept in mind while naming a R variable:

- A valid variable name consists of a combination of alphabets, numbers, dot(.), and underscore(_) characters. Example: var.1_ is valid
- Apart from the dot and underscore operators, no other special character is allowed. Example: var$1 or var#1 both are invalid
- Variables can start with alphabets or dot characters. Example: .var or var is valid
- The variable should not start with numbers or underscore. Example: 2var or _var is invalid.
- If a variable starts with a dot the next thing after the dot cannot be a number. Example: .3var is invalid
- The variable name should not be a reserved keyword in R. Example: TRUE, FALSE,etc.

```
# R program to demonstrate rules for naming the variables

# Correct naming

b2 = 7

# Correct naming

Amiya_Profession = "Student"

# Correct naming

Amiya.Profession = "Student"

# Wrong naming

2b = 7

# Wrong naming

Amiya@Profession = "Student"
```

Important Methods for R Variables

R provides some useful methods to perform operations on variables. These methods are used to determine the data type of the variable, finding a variable, deleting a variable, etc. Following are some of the methods used to work on variables:

1. class() function

This built-in function is used to determine the data type of the variable provided to it.

The R variable to be checked is passed to this as an argument and it prints the data type in return.

Syntax

```
class(variable)
```

Example

```
var1 = "hello"

print(class(var1))
```

Output

```
[1] "character"
```

2. ls() function

This built-in function is used to know all the present variables in the workspace.

This is generally helpful when dealing with a large number of variables at once and helps prevents overwriting any of them.

Syntax

```
ls()
```

Example

```
# using equal to operator

var1 = "hello"

# using leftward operator

var2 <- "hello"

# using rightward operator

"hello" -> var3

print(ls())
```

Output

```
[1] "var1" "var2" "var3"
```

3. rm() function

This is again a built-in function used to delete an unwanted variable within your workspace.

This helps clear the memory space allocated to certain variables that are not in use thereby creating more space for others. The name of the variable to be deleted is passed as an argument to it.

Syntax

```
rm(variable)
```

Example

```
# using equal to operator
var1 = "hello"
# using leftward operator
var2 <- "hello"
# using rightward operator
"hello" -> var3
# Removing variable
rm(var3)
print(var3)
```

Output

```
Error in print(var3) : object 'var3' not found
Execution halted
```

Example

```
# numeric
x <- 10.5
class(x)
# integer
x <- 1000L
class(x)
# complex
x <- 9i + 3
class(x)
# character/string
x <- "R is exciting"
class(x)
# logical/boolean
x <- TRUE
class(x)
```

Scope of Variables in R programming

The location where we can find a variable and also access it if required is called the scope of a variable. There are mainly two types of variable scopes:

1. Global Variables

Global variables are those variables that exist throughout the execution of a program. It can be changed and accessed from any part of the program.

As the name suggests, Global Variables can be accessed from any part of the program.

They are available throughout the lifetime of a program.

They are declared anywhere in the program outside all of the functions or blocks.

Declaring global variables

Global variables are usually declared outside of all of the functions and blocks. They can be accessed from any portion of the program.

# R program to illustrateusage of global variables

```
# global variable

global = 5

# global variable accessed from

# within a function

display = function(){

print(global)

}

display()

# changing value of global variable

global = 10

display()
```

Output

```
[1] 5

[1] 10
```

In the above code, the variable 'global' is declared at the top of the program outside all of the functions so it is a global variable and can be accessed or updated from anywhere in the program.

2. Local Variables

Local variables are those variables that exist only within a certain part of a program like a function and are released when the function call ends. Local variables do not exist outside the block in which they are declared, i.e. they cannot be accessed or used outside that block.

Declaring local variables

Local variables are declared inside a block.

# R program to illustrateusage of local variables

```
age=20
```

```
func = function(){
# this variable is local to the
# function func() and cannot be
# accessed outside this function
age = 18
cat("\nAge is:",age)
}
func()
cat("\nAge is:",age)
```

Output

```
Age is:
[1] 18
[2] 20
```

Accessing Global Variables

Global Variables can be accessed from anywhere in the code unlike local variables that have a scope restricted to the block of code in which they are created. Example:

```
f = function() {
# a is a local variable here
a <-1
}
f()
# Can't access outside the function
print(a) # This'll give error
```

Output:

```
Error in print(a) : object 'a' not found
```

In the above code, we see that we are unable to access variable “a” outside the function as it’s assigned by an assignment operator(<-) that makes “a” as a local variable. To make assignments to global variables, a super assignment operator(<<-) is used. How super assignment operator works? When using this operator within a function, it searches for the variable in the parent environment frame, if not found it keeps on searching the next level until it reaches the global environment. If the variable is still not found, it is created and assigned at the global level. Example:

# R program to illustrateScope of variables

```
outer_function = function(){
inner_function = function(){
        # Note that "<<-" operator here
        # makes a as global variable
        a <<- 10
        print(a)
}
inner_function()
print(a)
}
outer_function()
# Can access outside the function
print(a)
```

Output:

```
[1] 10
[1] 10
[1] 10
```

When the statement “a <<- 10” is encountered within inner_function(), it looks for the variable “a” in the outer_function() environment. When the search fails, it searches in R_GlobalEnv. Since “a” is not defined in this global environment as well, it is created and assigned there which is now referenced and printed from within inner_function() as well as outer_function().

Difference between local and global variables in R

- **Scope** A global variable is defined outside of any function and may be accessed from anywhere in the program, as opposed to a local variable.
- **Lifetime** A local variable’s lifetime is constrained by the function in which it is defined. The local variable is destroyed once the function has finished running. A global variable, on the other hand, doesn’t leave memory until the program is finished running or the variable is explicitly deleted.
- **Naming conflicts** If the same variable name is used in different portions of the program, they may occur since a global variable can be accessed from anywhere in the program. Contrarily, local variables are solely applicable to the function in which they are defined, reducing the likelihood of naming conflicts.
- **Memory usage** Because global variables are kept in memory throughout program execution, they can eat up more memory than local variables. Local variables, on the other hand, are created and destroyed only when necessary, therefore they normally use less memory.

In R, variables are the containers for storing data values. They are reference, or pointers, to an object in memory which means that whenever a variable is assigned to an instance, it gets mapped to that instance. A variable in R can store a vector, a group of vectors or a combination of many R objects. c( ) method is used to combine values into vectors or list.

Example:

```
# R program to demonstrate variable assignment
# Assignment using equal operator
var1 = c(0, 1, 2, 3)
print(var1)
# Assignment using leftward operator
var2 <- c("Python", "R")
print(var2)
# A Vector Assignment
a = c(1, 2, 3, 4)
print(a)
b = c("Debi", "Sandeep", "Subham", "Shiba")
print(b)
# A group of vectors Assignment using list
c = list(a, b)
print(c)
```

Output:

```
[1] 0 1 2 3
[1] "Python" "R"
[1] 1 2 3 4
[1] "Debi"    "Sandeep" "Subham"  "Shiba"
[[1]]
[1] 1 2 3 4
[[2]]
[1] "Debi"    "Sandeep" "Subham"  "Shiba"
```

Basic R Syntax

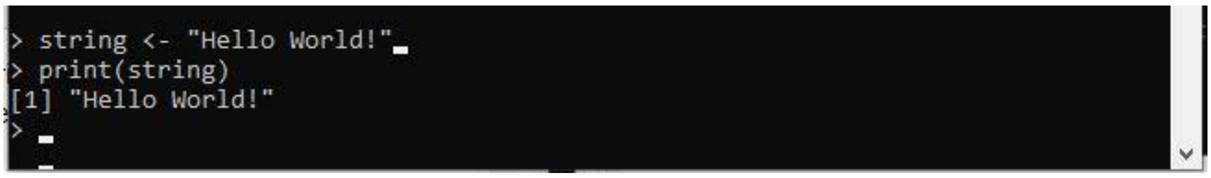
R Programming is a very popular programming language which is broadly used in data analysis. The way in which we define its code is quite simple. The "Hello World!" is the basic program for all the languages, and now we will understand the syntax of R programming with "Hello world" program. We can write our code either in command prompt, or we can use an R script file.

R Command Prompt

It is required that we have already installed the R environment set up in our system to work on the R command prompt. After the installation of R environment setup, we can easily start R command prompt by typing R in our Windows command prompt. When we press enter after typing R, it will launch interpreter, and we will get a prompt on which we can code our program.

"Hello, World!" Program

The code of "Hello World!" in R programming can be written as:



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

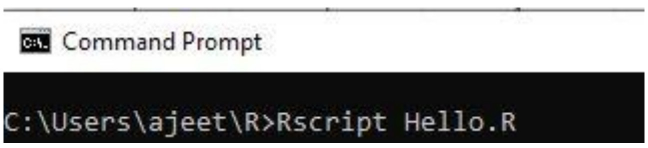
In the above code, the first statement defines a string variable string, where we assign a string "Hello World!". The next statement print() is used to print the value which is stored in the variable string.

R Script File

The R script file is another way on which we can write our programs, and then we execute those scripts at our command prompt with the help of R interpreter known as Rscript. We make a text file and write the following code. We will save this file with .R extension as:

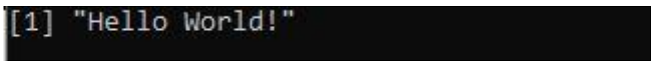
```
Hello.R

string <-"Hello World!"

print(string)
```

To execute this file in Windows and other operating systems, the process will remain the same as mentioned below.



```
C:\Users\ajeet\R>Rscript Hello.R
```

When we press enter it will give us the following output:



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

Comments

In R programming, comments are the programmer readable explanation in the source code of an R program. The purpose of adding these comments is to make the source code easier to understand. These comments are generally ignored by compilers and interpreters.

In R programming there is only single-line comment. R doesn't support multi-line comment. But if we want to perform multi-line comments, then we can add our code in a false block.

Single-line comment

```
#My First program in R programming

string <-"Hello World!"

print(string)
```

Syntax

To output text in R, use single or double quotes:

Example

```
"Hello World!"
```

To output numbers, just type the number (without quotes):

Example

```
5

10

25
```

To do simple calculations, add numbers together:

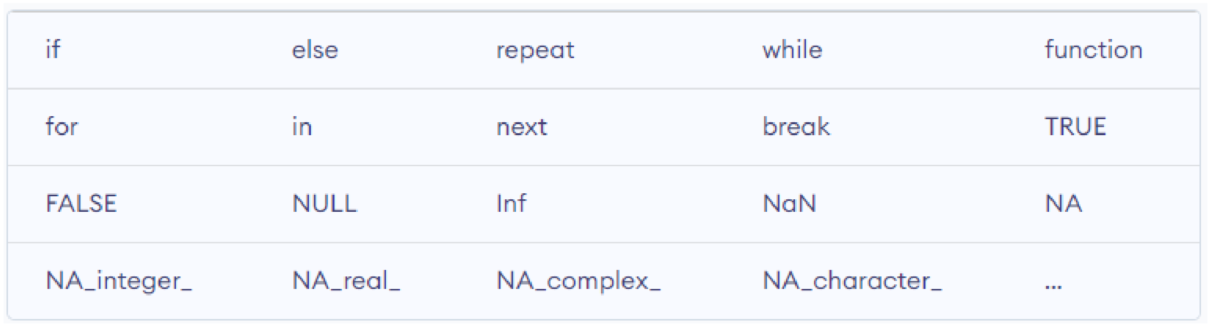
Example

```
5 + 5
```

R Reserved Words

Reserved words in R programming are a set of words that have special meaning and cannot be used as an identifier (variable name, function name etc.).

Here is a list of reserved words in the R's parser.



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

- This list can be viewed by typing `help  (reserved)`or`?eeserved` at the R command prompt as follows. (Press q to exit from help and return to R prompt)
- Among these words, if, else, repeat, while, function, for, in, next and break are used for conditions, loops and user defined functions.
- They form the basic building blocks of programming in R.
- TRUE and FALSE are the logical constants in R.
- NULL represents the absence of a value or an undefined value.
- Inf is for "Infinity", for example when 1 is divided by 0 whereas NaN is for "Not a Number", for example when 0 is divided by 0.
- NA stands for "Not Available" and is used to represent missing values.
- R is a case sensitive language. Which means that TRUE and True are not the same.
- While the first one is a reserved word denoting a logical constant in R, the latter can be used as a variable name.

Example

```
True <- 1
TRUE
True
```

Output

```
[1] TRUE
[1] 1
```

Following are some most important keywords and statements along with their examples:

Conditional Statement

if: If statement is one of the Decision-making statements in the R programming language. It is one of the easiest decision-making statements. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.

Example:

```
# R program to illustrate if statement
# assigning value to variable a
a <- 5
# condition
if( a > 0 )
{
     print("Positive Number") # Statement
}
```

Output:

```
Positive Number
```

else: It is similar to if statement but when the test expression in if condition fails, then statements in else condition are executed.

Example:

```
x <- 5
# Check value is less than or greater than 10
if(x > 10)
{
     print(paste(x, "is greater than 10"))
}
else
{
     print(paste(x, "is less than 10"))
}
```

Output:

```
[1] "5 is less than 10"
```

Looping Structure

while: It is a type of control statement which will run a statement or a set of statements repeatedly unless the given condition becomes false. It is also an entry controlled loop, in this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

Example:

```
# R program to demonstrate the use of while loop
val = 1
# using while loop
while (val<= 5 )
{
     # statements
     print(val)
     val = val + 1
}
```

Output:

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

Repeat

It is a simple loop that will run the same statement or a group of statements repeatedly until the stop condition has been encountered. Repeat loop does not have any condition to terminate the loop, a programmer must specifically place a condition within the loop's body and use the declaration of a break statement to terminate this loop. If no condition is present in the body of the repeat loop then it will iterate infinitely.

Example:

```
# R program to demonstrate the use of repeat loop
val = 1
# using repeat loop
repeat
{
     # statements
     print(val)
     val = val + 1
     # checking stop condition
     if(val> 5)
     {
          # using break statement
          # to terminate the loop
          break
     }
}
```

Output:

```
[1] 1
[1] 2
```

```
[1] 3
[1] 4
[1] 5
```

For In Loop

for: It is a type of control statement that enables one to easily construct a loop that has to run statements or a set of statements multiple times. For loop is commonly used to iterate over items of a sequence. It is an entry controlled loop, in this loop the test condition is tested first, then the body of the loop is executed, the loop body would not be executed if the test condition is false.

Example:

```
# R program to demonstrate the use of for loop
# using for loop
for (val in 1:5)
{
     # statement
     print(val)
}
```

Output:

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

Creating User Defined Function

function: Functions are useful when you want to perform a certain task multiple number of times. In R functions are created using function keyword.

Example:

```
# A simple R function to check
# whether x is even or odd
evenOdd = function(x){
if(x %% 2 == 0)
     return("even")
else
     return("odd")
}
print(evenOdd(4))
print(evenOdd(3))
```

Output:

```
[1] "even"
[1] "odd"
```

Next and Break Statement

Next statement in R is used to skip any remaining statements in the loop and continue the execution of the program. In other words, it is a statement that skips the current iteration without loop termination.

Example:

```
# R program to illustrate next in for loop
val<- 6:11
# Loop
for (i in val)
{
     if (i == 8)
     {
          # test expression
          next
     }
     print(i)
}
```

Output:

```
[1] 6
[1] 7
[1] 9
[1] 10
[1] 11
```

break: The break keyword is a jump statement that is used to terminate the loop at a particular iteration.

Example:

```
# R Break Statement Example
a<-1
while (a < 10)
{
     print(a)
     if(a == 5)
          break
     a = a + 1
}
```

Output:

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

Boolean Statements

TRUE/FALSE: The TRUE and FALSE keywords are used to represent a Boolean true and Boolean false. If the given statement is true, then the interpreter returns true else the interpreter returns false.

Example:

```
# A simple R program
# to illustrate TRUE / FALSE
# Sample values
x = 4
y = 3
# Comparing two values
z = x > y
p = x < y
# print the logical value
print(z)
print(p)
```

Output:

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

NULL, NaN and NA

NULL: In R, NULL represents the null object. NULL is used to represent missing and undefined values. NULL is the logical representation of a statement which is neither TRUE nor FALSE.

Example:

```
# A simple R program
# to illustrate NULL
v = as.null(c(1, 2, 3, 4))
#as.null() ignore any arguments passed in it and returns NULL.
print(v)
```

Output:

```
NULL
```

NaN: NaN represents value other than number. NaN keyword means ‘Not a Number’.

Example:

```
# A simple R program
# To check NaN
y = c(1, NaN, 3)
print(is.nan(y))
```

Output:

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

NA: NA stands for “Not Available” and is used to represent missing values. There are also constants NA_integer_, NA_real_, NA_complex_ and NA_character_ of the other atomic vector types which support missing values and all of these are reserved words in the R language.

Example:

```
# A simple R program
# to illustrate NA
# To check NA
x = c(1, NA, 2, 3)
print(is.na(x))
```

Output:

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

Inf : In R is.finite and is.infinite return a vector of the same length as x, where x is an R object to be tested. This indicating which elements are finite (not infinite and not missing) or infinite. Inf and -Inf keyword mean positive and negative infinity.

```
# to illustrate Inf
# To check Inf
x = c(Inf, 2, 3)
print(is.finite(x))
```

Output:

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

Type Conversion

You can convert from one type to another with the following functions:

as.numeric()

as.integer()

as.complex()

Example

```
x <- 1L # integer

y <- 2 # numeric

# convert from integer to numeric:

a <- as.numeric(x)

# convert from numeric to integer:

b <- as.integer(y)

# print values of x and y

x

y

# print the class name of a and b

class(a)
class(b)
```

Math

In R, you can use operators to perform common mathematical operations on numbers.

The + operator is used to add together two values:

Example

```
10 + 5
```

And the - operator is used for subtraction:

Example

```
10 - 5
```

Built-in Math Functions

R also has many built-in math functions that allows you to perform mathematical tasks on numbers.

For example, the min() and max() functions can be used to find the lowest or highest number in a set:

Example

```
max(5, 10, 15)
min(5, 10, 15)
```

sqrt()

The sqrt() function returns the square root of a number:

Example

```
sqrt(16)
```

abs()

The abs() function returns the absolute (positive) value of a number:

Example

```
abs(-4.7)
```

ceiling() and floor()

The ceiling() function rounds a number upwards to its nearest integer, and the floor() function rounds a number downwards to its nearest integer, and returns the result:

Example

```
ceiling(1.4)
```

output `2`

```
floor(1.4)
```

Output `1`

String Literals

Strings are used for storing text.

A string is surrounded by either single quotation marks, or double quotation marks:

"hello" is the same as 'hello':

Example

```
"hello"
'hello'
```

Assign a String to a Variable

Assigning a string to a variable is done with the variable followed by the <- operator and the string:

Example

```
str <- "Hello"
str # print the value of str
```

Multiline Strings

You can assign a multiline string to a variable like this:

Example

```
str <- "This is
a test."
str # print the value of str
```

However, note that R will add a "\n" at the end of each line break. This is called an escape character, and the n character indicates a new line.

If you want the line breaks to be inserted at the same position as in the code, use the cat() function:

Example

```
str <- "This is
a test."
cat(str)
```

String Length

There are many useful string functions in R.

For example, to find the number of characters in a string, use the nchar() function:

Example

```
str <- "Hello World!"
```

```
nchar(str)
```

Check a String

The two functions grep() and grepl() let you check whether a pattern is present in a character string or vector of a character string, but they both return different outputs:

- Grep() return vector of indices of the element if a pattern exists in that vector.
- Grepl() return TRUE if the given pattern is present in the vector. Otherwise, it return FALSE

The grep() is a built-in function in R. It searches for matches for certain character patterns within a character vector. The grep() takes pattern and data as an argument and return a vector of indices of the character string.

Syntax:

```
 grep("pattern", x)
```

Parameter:

Pattern- The pattern that matches with given vector element

x – specified character vector

Example:

```
#Program to show usage of grep()

# code

x <- c('Geeks', 'GeeksforGeeks', 'Geek', 'Geeksfor', 'Gfg')

# calling grep() function

grep('Geek', x)
```

Output

```
[1] 1 2 3 4
```

The **grepl**() stands for "grep logical". In R it is a built-in function that searches for matches of a string or string vector. The grepl() method takes a pattern and data and returns TRUE if a string contains the pattern, otherwise FALSE.

Syntax:

```
grep("pattern", x)
```

Parameter:

Pattern- The pattern that matches with given vector element

x – specified character vector

Example

```
str <- "Hello World!"
```

```
grepl("H", str)

grepl("Hello", str)

grepl("X", str)
```

Output

```
TRUE

TRUE

FALSE
```

Example:

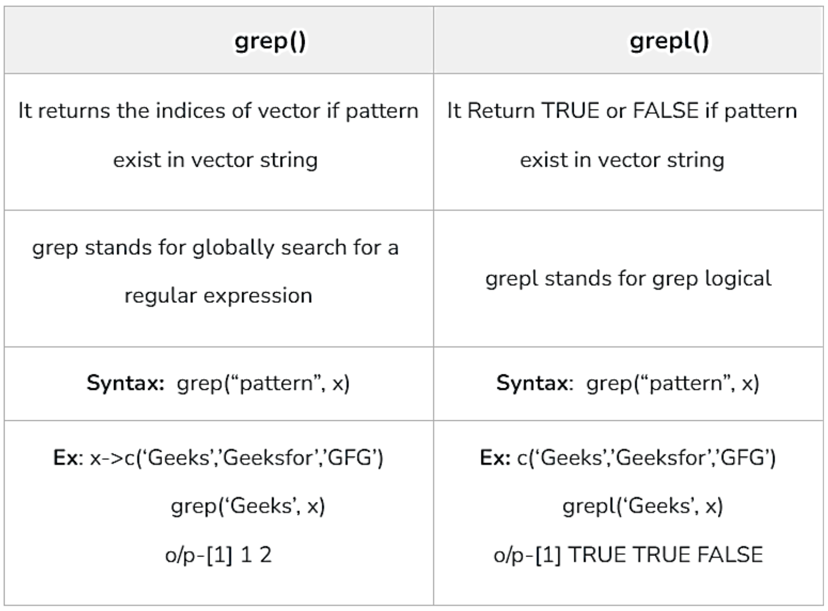
```
#Program to show the usage of grepl()

# Code

x <- c('Geeks', 'GeeksforGeeks', 'Geek',  'Geeksfor', 'Gfg')

# calling grepl() function

grepl('for', x)
```

Output:

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

Both functions need a pattern and x argument, where the pattern is the regular expression you want to match for, and the x argument is the character vector from which you can match the pattern string.

grep() and grepl() functions are help you to search data in the fastest manner when there is a huge amount of data present.



grep()	grepl()
It returns the indices of vector if pattern exist in vector string	It Return TRUE or FALSE if pattern exist in vector string
grep stands for globally search for a regular expression	grepl stands for grep logical
Syntax: grep("pattern", x)	**Syntax**: grep("pattern", x)
Ex: x->c('Geeks','Geeksfor','GFG') grep('Geeks', x) o/p-[1] 1 2	**Ex:** c('Geeks','Geeksfor','GFG') grepl('Geeks', x) o/p-[1] TRUE TRUE FALSE

Combine Two Strings

Use the paste() function to merge/concatenate two strings:

Example

```
str1 <- "Hello"

str2 <- "World"

paste(str1, str2)
```

Escape Characters

To insert characters that are illegal in a string, you must use an escape character.

An escape character is a backslash \ followed by the character you want to insert.

An example of an illegal character is a double quote inside a string that is surrounded by double quotes:

Example

```
str <- "We are the so-called "Vikings", from the north."

str
```

Result:

```
Error: unexpected symbol in "str <- "We are the so-called "Vikings"
```

To fix this problem, use the escape character \":

Example

The escape character allows you to use double quotes when you normally would not be allowed:

```
str <- "We are the so-called \"Vikings\", from the north."

str

cat(str)
```

Note that auto-printing the str variable will print the backslash in the output. You can use the cat() function to print it without backslash.

Other escape characters in R:

Code	Result
\\	Backslash
\n	New Line
\r	Carriage Return
\t	Tab
\b	Backspace

Booleans (Logical Values)

In programming, you often need to know if an expression is true or false.

You can evaluate any expression in R, and get one of two answers, TRUE or FALSE.

When you compare two values, the expression is evaluated and R returns the logical answer:

Example

```
10 > 9    # TRUE because 10 is greater than 9

10 == 9   # FALSE because 10 is not equal to 9

10 < 9    # FALSE because 10 is greater than 9
```

You can also compare two variables:

Example

```
a <- 10

b <- 9

a > b
```

You can also run a condition in an if statement, which you will learn much more about in the if..else chapter.

Example

```
a <- 200

b <- 33

if (b > a) {

  print ("b is greater than a")

} else {

  print("b is not greater than a")

}
```

Operators of R

Arithmetic Operators

Arithmetic operators are used with numeric values to perform common mathematical operations:

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y

/	Division	x / y
^	Exponent	x ^ y
%%	Modulus	x %% y
%/%	Integer Division	x%/%y

Assignment Operators

Assignment operators are used to assign values to variables:

Example

```
my_var<- 3

my_var<<- 3

3 -> my_var

3 ->> my_var

my_var # print my_var
```

Note: <<- is a global assigner. You will learn more about this in the Global Variable chapter.

It is also possible to turn the direction of the assignment operator.

x <- 3 is equal to 3 -> x

Comparison Operators

Comparison operators are used to compare two values:

Operator	**Name**	**Example**
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Logical Operators

Logical operators are used to combine conditional statements:

Operator	**Description**
&	Element-wise Logical AND operator. It returns TRUE if both elements are TRUE

&&	Logical AND operator - Returns TRUE if both statements are TRUE
|	Elementwise- Logical OR operator. It returns TRUE if one of the statement is TRUE
||	Logical OR operator. It returns TRUE if one of the statement is TRUE.
!	Logical NOT - returns FALSE if statement is TRUE

Example

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1 & list2)
```
Output : FALSE   TRUE

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1|list2)
```
Output : TRUE TRUE

```
list1 <- c(0,FALSE)
print(!list1)
```
Output : TRUE TRUE

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1[1] && list2[1])
```
Output : FALSE

```
list1 <- c(TRUE, 0.1)
list2 <- c(0,4+3i)
print(list1[1]||list2[1])
```
Output : TRUE

Example

```
# R program to illustrate
# the use of Logical operators
vec1 <- c(0,2)
```

```
vec2 <- c(TRUE,FALSE)
# Performing operations on Operands
cat ("Element wise AND :", vec1 & vec2, "\n")
cat ("Element wise OR :", vec1 | vec2, "\n")
cat ("Logical AND :", vec1[1] && vec2[1], "\n")
cat ("Logical OR :", vec1[1] || vec2[1], "\n")
cat ("Negation :", !vec1)
```

Output

```
Element wise AND : FALSE FALSE
Element wise OR : TRUE TRUE
Logical AND : FALSE
Logical OR : TRUE
Negation : TRUE FALSE
```

Miscellaneous Operators

Miscellaneous operators are used to manipulate data:

Operator	Description	Example
:	Creates a series of numbers in a sequence	x <- 1:10
%in%	Find out if an element belongs to a vector	x %in% y
%*%	Matrix Multiplication	x <- Matrix1 %*% Matrix2

Example

```
val<- 0.1
 list1 <- c(TRUE, 0.1,"apple")
 print (val %in% list1)
```

Output : TRUE

Checks for the value 0.1 in the specified list. It exists, therefore, prints TRUE.

Example

```
 mat = matrix(c(1,2,3,4,5,6),nrow=2,ncol=3)
         print (mat)
         print( t(mat))
         pro = mat %*% t(mat)
         print(pro)
```

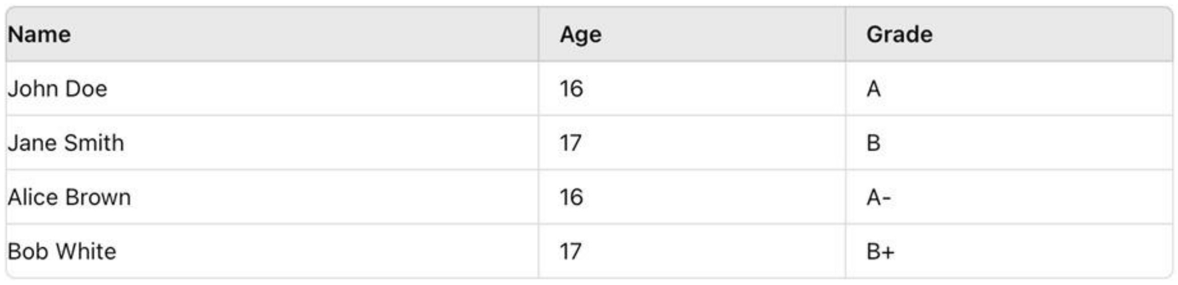
Output :
```
[,1] [,2] [,3]      #original matrix of order 2x3
[1,]   1    3    5
[2,]   2    4    6
     [,1] [,2]             #transposed matrix of order 3x2
[1,]    1    2
[2,]    3    4
[3,]    5    6
     [,1] [,2]            #product matrix of order 2x2
[1,]   35   44
[2,]   44   56
```

3.4 Importing data into R from different file formats (CSV, Excel, etc.).

What is CSV File?

A CSV (Comma-Separated Values) file is a simple text file used to store tabular data, such as a spreadsheet or database. Each line in a CSV file corresponds to a row in the table, and each value in that line is separated by a comma (or another delimiter like a semicolon or tab).

For Example Consider following Table:



Name	Age	Grade
John Doe	16	A
Jane Smith	17	B
Alice Brown	16	A-
Bob White	17	B+

If we want to create CSV file for it, it can be written as follows:

Name,Age,Grade

John Doe,16,A

Jane Smith,17,B

Alice Brown,16,A-

Bob White,17,B+

How to Import a CSV File into R ?

A CSV file is used to store contents in a tabular-like format, which is organized in the form of rows and columns. The column values in each row are separated by a delimiter string. The CSV files can be loaded into the working space and worked using both in-built methods and external package imports.

Method 1: Using read.csv() method

The read.csv() method in base R is used to load a .csv file into the present script and work with it. The contents of the csv can be stored into the variable and further manipulated. Multiple files can also be accessed in different variables. The output is returned in the form of a data frame, where row numbers are assigned integers beginning with 1.

`Syntax: read.csv(path, header = TRUE, sep = ",")`

Arguments :

path : The path of the file to be imported

header : By default : TRUE . Indicator of whether to import column headings.

sep = "," : The separator for the values in each row.

Example:

```
# specifying the path
path <- "D:\\sem3A\\R\\Script\\Emp.csv"
# reading contents of csv file
content <- read.csv(path)
# contents of the csv file
print (content)
```

Output:

```
  ID Name     Post Age
1  5    H       CA  67
2  6    K      SDE  39
3  7    Z    Admin  28
```

In case, the header is set to FALSE, the column names are ignored, and default variables names are displayed for each column beginning from V1.

```
path <- "D:\\sem3A\\R\\Script\\Emp.csv"
# reading contents of csv file
content <- read.csv(path, header = FALSE)
# contents of the csv file
print (content)
```

Output:

```
   V1  V2       V3  V4
1  5    H       CA  67
2  6    K      SDE  39
3  7    Z    Admin  28
```

Method 2: Using read_csv() method

The “readr” package in R is used to read large flat files into the working space with increase speed and efficiency.

install.packages("readr")

The read_csv() method reads a csv file reading one line at a time. The data using this method is read in the form of a tibble, of the same dimensions as of the table stored in the .csv file. Only ten rows of the tibble are displayed on the screen and rest are available after expanding, which increases the readability of the large files. This method is more efficient since it returns more information about the column types. It also displays progress tracker for the percentage of file read into the system currently if the progress argument is enabled, therefore being more robust. This method is also faster in comparison to the base R read.csv() method.

```
Syntax: read_csv (file-path , col_names , n_max , col_types , progress )
```

Arguments :

file-path : The path of the file to be imported

col_names : By default, it is TRUE. If FALSE, the column names are ignored.

n_max : The maximum number of rows to read.

col_types : If any column succumbs to NULL, then the col_types can be specified in a compact string format.

progress : A progress meter to analyse the percentage of file read into the system

```
library("readr")

# specifying the path

path <- “D:\\sem3A\\R\\Script\\Emp.csv”

# reading contents of csv file

content <- read_csv(path, col_names = TRUE)

# contents of the csv file

print (content)
```

Importing Data from a Text File

We can easily import or read .txt file using basic R function read.table(). read.table() is used to read a file in table format. This function is easy to use and flexible.

Syntax:

```
# read data stored in .txt file

x<-read.table(“file_name.txt”, header=TRUE/FALSE)

# Simple R program to read txt file

x<-read.table("D:\\sem3A\\R\\Script\\Emp.txt", header=FALSE)

# print x

print(x)
```

Output:

```
   V1 V2 V3

1 100 a1 b1

2 200 a2 b2
```

```
3 300 a3 b3
```

If the header argument is set at TRUE, which reads the column names if they exist in the file.

Importing Data from a delimited file

R has a function read.delim() to read the delimited files into the list. The file is by default separated by a tab which is represented by sep="", that separated can be a comma(,), dollar symbol($), etc.

Syntax: read.delim("file_name.txt", sep="", header=TRUE)

```
x <- read.delim("D:\\sem3A\\R\\Script\\Emp.txt", sep="|",
                 header=TRUE)
# print x
print(x)
# print type of x
typeof(x)
```

Output:

```
   X.V1.V2.V3
1 1, 100, a1, b1
2 2, 200, a2, b2
3 3, 300, a3, b3
[1] "list
```

How to import an Excel File into R ?

Method 1: Using read_excel()

In this approach to import the Excel file in the R, the user needs to call the read_excel() function from readxl library of the R language with the name of the file as the parameter. readxl() package can import both .xlsx and .xls files. This package is pre-installed in R-Studio. With the use of this function, the user will be able to import the Excel file in R.

Syntax: read_excel(filename, sheet, dtype = "float32")

Parameters:

filename:-File name to read from.

sheet:-Name of the sheet in Excel file.

dtype:-Numpy data type.

Returns:

The variable is treated to be a data frame.

```
library(readxl)
gfg_data=read_excel('Emp.xlsx')
```

```
gfg_data
```

Method 2: Using read.xlsx()

Read.xlsx() is present in xlsx package.

Syntax: read.xlsx(path)

Example

```
library(xlsx)

gfg_data=read.xlsx('Emp.xlsx')

gfg_data
```

3.5 read, write and view data using data frames.

Data Frames are data displayed in a format as a table.

Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.

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.

Use the data.frame() function to create a data frame:

Example:

```
# Create a data frame

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
```

Output:

```
   Training Pulse Duration

1 Strength   100       60
```

```
2   Stamina    150        30
3     Other    120        45
```

Summarize the Data

Use the summary() function to summarize the data from a Data Frame:

Example

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

Output:

```
   Training Pulse Duration
1  Strength    100        60
2   Stamina    150        30
3     Other    120        45
Training      Pulse             Duration
 Other   :1    Min.   :100.0    Min.   :30.0
 Stamina :1    1st Qu.:110.0    1st Qu.:37.5
 Strength:1    Median :120.0    Median :45.0
               Mean   :123.3    Mean   :45.0
               3rd Qu.:135.0    3rd Qu.:52.5
               Max.   :150.0    Max.   :60.0
```

Access Items

We can use single brackets [ ] to access data of an element, double brackets [[ ]] or $ to access columns from a data frame:

Example

```
Data_Frame<- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data_Frame[1]
Data_Frame[["Training"]]
Data_Frame$Training
```

Output:

```
  Training
1 Strength
2  Stamina
3    Other
[1] Strength Stamina  Other
[1] Strength Stamina  Other
```

Add Rows

Use the rbind() function to add new rows in a Data Frame:

Example

```
Data_Frame<- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Add a new row
New_row_DF<- rbind(Data_Frame, c("Strength", 110, 110))
# Print the new row
New_row_DF
```

Output:

```
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4 Strength   110      110
```

Add Columns

Use the cbind() function to add new columns in a Data Frame:

Example

```
Data_Frame<- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Add a new column
New_col_DF<- cbind(Data_Frame, Steps = c(1000, 6000, 2000))
# Print the new column
New_col_DF
```

Output:

```
  Training Pulse Duration Steps
1 Strength   100       60  1000
2  Stamina   150       30  6000
3    Other   120       45  2000
```

Remove Rows and Columns

Use the c() function to remove rows and columns in a Data Frame:

Example

```
Data_Frame<- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
# Remove the first row and column
Data_Frame_New<- Data_Frame[-c(1), -c(1)]
# Print the new data frame
Data_Frame_New
```

Output:

```
   Pulse Duration
2   150       30
```

```
3   120       45
```

Amount of Rows and Columns

Use the dim() function to find the amount of rows and columns in a Data Frame:

Example

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

Output:

```
[1] 3 3
```

You can also use the ncol() function to find the number of columns and nrow() to find the number of rows:

Example

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

Output:

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

Data Frame Length

Use the length() function to find the number of columns in a Data Frame (similar to ncol()):

Example

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

Output:

```
[1] 3
```

Combining Data Frames

Use the rbind() function to combine two or more data frames in R vertically:

Example

```
Data_Frame1 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data_Frame2 <- data.frame (
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)
New_Data_Frame<- rbind(Data_Frame1, Data_Frame2)
New_Data_Frame
```

Output:

```
  Training Pulse Duration
1 Strength   100       60
2  Stamina   150       30
3    Other   120       45
4  Stamina   140       30
5  Stamina   150       30
6 Strength   160       20
```

And use the cbind() function to combine two or more data frames in R horizontally:

Example

```
Data_Frame3 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data_Frame4 <- data.frame (
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)
New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)
New_Data_Frame1
```

Output:

```
  Training Pulse Duration Steps Calories
1 Strength   100       60  3000      300
2  Stamina   150       30  6000      400
3    Other   120       45  2000      300
```