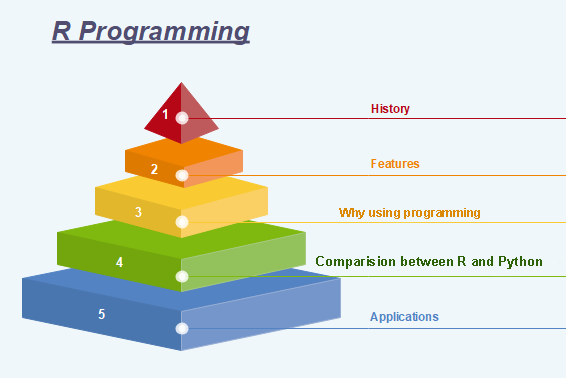
What is R Programming

**"R is an interpreted computer programming language which was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand."** The ***R Development Core Team*** currently develops R. It is also a software environment used to analyze **statistical information**, **graphical representation**, **reporting**, and **data modeling**. R is the implementation of the **S programming** language, which is combined with **lexical scoping semantics**.7M of Java

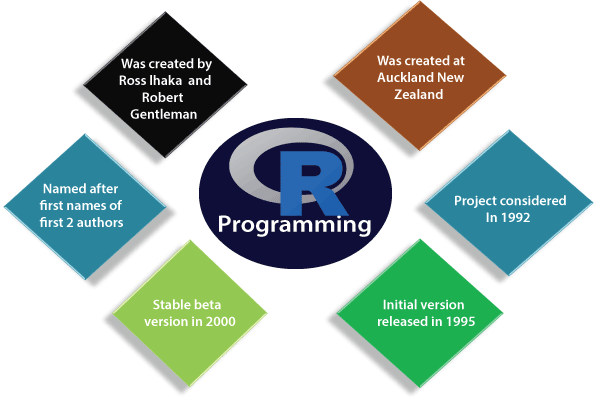
R not only allows us to do branching and looping but also allows to do modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python, and FORTRAN languages to improve efficiency.

In the present era, R is one of the most important tool which is used by researchers, data analyst, statisticians, and marketers for retrieving, cleaning, analyzing, visualizing, and presenting data.



History of R Programming

The history of R goes back about 20-30 years ago. R was developed by Ross lhaka and Robert Gentleman in the University of Auckland, New Zealand, and the R Development Core Team currently develops it. This programming language name is taken from the name of both the developers. The first project was considered in 1992. The initial version was released in 1995, and in 2000, a stable beta version was released.



The following table shows the release date, version, and description of R language:

|  |  |  |
| --- | --- | --- |
| **Version-Release** | **Date** | **Description** |
| 0.49 | 1997-04-23 | First time R's source was released, and CRAN (Comprehensive R Archive Network) was started. |
| 0.60 | 1997-12-05 | R officially gets the GNU license. |
| 0.65.1 | 1999-10-07 | update.packages and install.packages both are included. |
| 1.0 | 2000-02-29 | The first production-ready version was released. |
| 1.4 | 2001-12-19 | First version for Mac OS is made available. |
|  |  |  |
| 2.0 | 2004-10-04 | The first version for Mac OS is made available. |
| 2.1 | 2005-04-18 | Add support for UTF-8encoding, internationalization, localization etc. |
| 2.11 | 2010-04-22 | Add support for Windows 64-bit systems. |
| 2.13 | 2011-04-14 | Added a function that rapidly converts code to byte code. |
| 2.14 | 2011-10-31 | Added some new packages. |
| 2.15 | 2012-03-30 | Improved serialization speed for long vectors. |
| 3.0 | 2013-04-03 | Support for larger numeric values on 64-bit systems. |
| 3.4 | 2017-04-21 | The just-in-time compilation (JIT) is enabled by default. |
| 3.5 | 2018-04-23 | Added new features such as compact internal representation of integer sequences, serialization format etc. |

Features of R programming

R is a domain-specific programming language which aims to do data analysis. It has some unique features which make it very powerful. The most important arguably being the notation of vectors. These vectors allow us to perform a complex operation on a set of values in a single command. There are the following features of R programming:

1. It is a simple and effective programming language which has been well developed.
2. It is data analysis software.
3. It is a well-designed, easy, and effective language which has the concepts of user-defined, looping, conditional, and various I/O facilities.
4. It has a consistent and incorporated set of tools which are used for data analysis.
5. For different types of calculation on arrays, lists and vectors, R contains a suite of operators.
6. It provides effective data handling and storage facility.
7. It is an open-source, powerful, and highly extensible software.
8. It provides highly extensible graphical techniques.
9. It allows us to perform multiple calculations using vectors.
10. R is an interpreted language.

Why use R Programming?

There are several tools available in the market to perform data analysis. Learning new languages is time taken. The data scientist can use two excellent tools, i.e., R and Python. We may not have time to learn them both at the time when we get started to learn data science. Learning statistical modeling and algorithm is more important than to learn a programming language. A programming language is used to compute and communicate our discovery.

The important task in data science is the way we deal with the data: clean, feature engineering, feature selection, and import. It should be our primary focus. Data scientist job is to understand the data, manipulate it, and expose the best approach. For machine learning, the best algorithms can be implemented with R. **Keras** and **TensorFlow** allow us to create high-end machine learning techniques. R has a package to perform **Xgboost**. Xgboost is one of the best algorithms for **Kaggle competition**.

R communicate with the other languages and possibly calls Python, Java, C++. The big data world is also accessible to R. We can connect R with different databases like **Spark** or **Hadoop**.

In brief, R is a great tool to investigate and explore the data. The elaborate analysis such as clustering, correlation, and data reduction are done with R.

Comparison between R and Python

Data science deals with identifying, extracting, and representing meaningful information from the data source. R, Python, SAS, SQL, Tableau, MATLAB, etc. are the most useful tools for data science. R and Python are the most used ones. But still, it becomes confusing to choose the better or the most suitable one among the two, R and Python.

|  |  |  |
| --- | --- | --- |
| **Comparison Index** | **R** | **Python** |
| **Overview** | "R is an interpreted computer programming language which was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand ." The R Development Core Team currently develops R. R is also a software environment which is used to analyze statistical information, graphical representation, reporting, and data modeling. | Python is an Interpreted high-level programming language used for general-purpose programming. Guido Van Rossum created it, and it was first released in 1991. Python has a very simple and clean code syntax. It emphasizes the code readability and debugging is also simple and easier in Python. |
| **Specialties for data science** | R packages have advanced techniques which are very useful for statistical work. The CRAN text view is provided by many useful R packages. These packages cover everything from Psychometrics to Genetics to Finance. | For finding outliers in a data set both R and Python are equally good. But for developing a web service to allow peoples to upload datasets and find outliers, Python is better. |
| **Functionalities** | For data analysis, R has inbuilt functionalities | Most of the data analysis functionalities are not inbuilt. They are available through packages like Numpy and Pandas |
| **Key domains of application** | Data visualization is a key aspect of analysis. R packages such as ggplot2, ggvis, lattice, etc. make data visualization easier. | Python is better for deep learning because Python packages such as Caffe, Keras, OpenNN, etc. allows the development of the deep neural network in a very simple way. |
| **Availability of packages** | There are hundreds of packages and ways to accomplish needful data science tasks. | Python has few main packages such as viz, Sccikit learn, and Pandas for data analysis of machine learning, respectively. |

Applications of R

There are several-applications available in real-time. Some of the popular applications are as follows:

* Facebook
* Google
* Twitter
* HRDAG
* Sunlight Foundation
* RealClimate
* NDAA
* XBOX ONE
* ANZ
* FDA

Installation of R

**R programming** is a very popular language and to work on that we have to install two things, i.e., R and RStudio. R and RStudio works together to create a project on R.

Installing R to the local computer is very easy. First, we must know which operating system we are using so that we can download it accordingly.

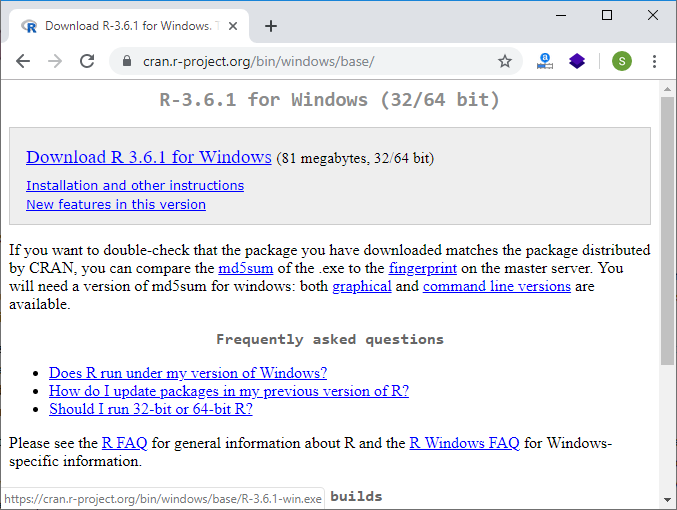
* The official site [https://cloud.r-project.org](https://cloud.r-project.org/) provides binary files for major operating systems including Windows, Linux, and Mac OS. In some Linux distributions, R is installed by default, which we can verify from the console by entering R.
* To install R, either we can get it from the site [https://cloud.r-project.org](https://cloud.r-project.org/) or can use commands from the terminal.
* 6.2M
* 91
* SQL CREATE TABLE

Install R in Windows

There are following steps used to install the R in Windows:

**Step 1:**

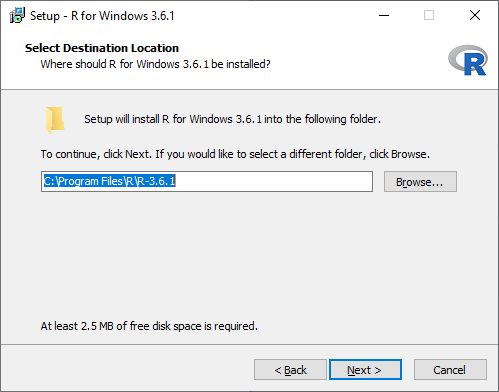
* First, we have to download the R setup from <https://cloud.r-project.org/bin/windows/base/>.



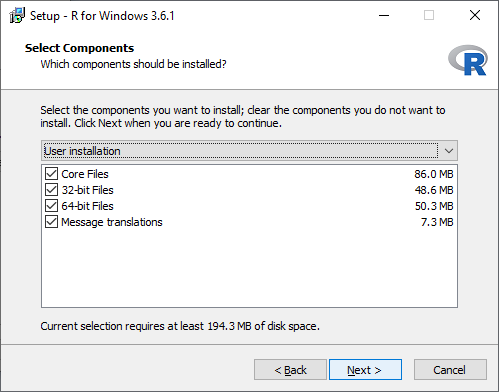
**Step 2:**

When we click on **Download R 3.6.1 for windows**, our downloading will be started of R setup. Once the downloading is finished, we have to run the setup of R in the following way:

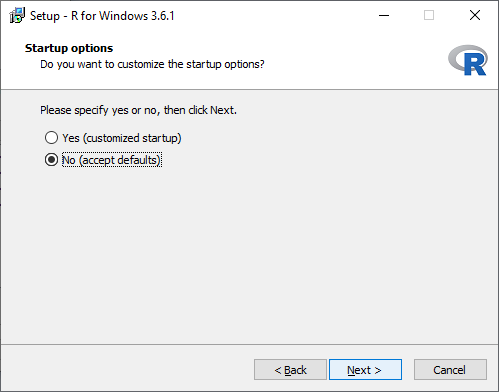
1) Select the path where we want to download the R and proceed to Next.



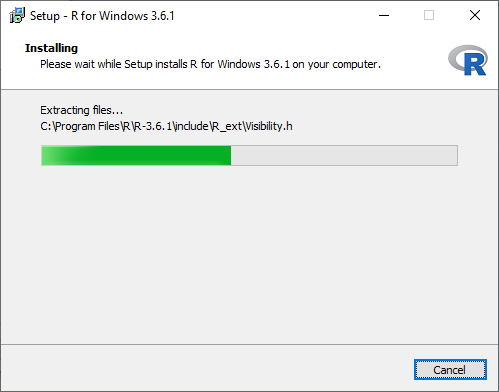
2) Select all components which we want to install, and then we will proceed to **Next**.



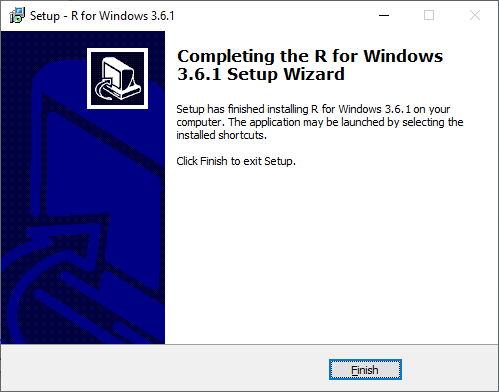
3) In the next step, we have to select either customized startup or accept the default, and then we proceed to **Next**.



4) When we proceed to next, our installation of R in our system will get started:



5) In the last, we will click on finish to successfully install R in our system.



RStudio IDE

RStudio is an integrated development environment which allows us to interact with R more readily. RStudio is similar to the standard RGui, but it is considered more user-friendly. This IDE has various drop-down menus, Windows with multiple tabs, and so many customization processes. The first time when we open RStudio, we will see three Windows. The fourth Window will be hidden by default. We can open this hidden Window by clicking the **File** drop-down menu, then **New File** and then **R Script**.

|  |  |  |
| --- | --- | --- |
| **RStudio Windows/Tabs** | **Location** | **Description** |
| Console Window | Lower-left | The location where commands are entered and output is printed. |
| Source Tabs | Upper-left | Built-in test editor |
| Environment Tab | Upper-left | An interactive list of loaded R objects. |
| History Tab | Upper-left | List of keystrokes entered into the console. |
| Files Tab | Lower-right | File explorer to navigate C drive folders. |
| Plots Tab | Lower-right | Output location for plots. |
| Packages Tab | Lower-right | List of installed packages. |
| Help Tab | Lower-right | Output location for help commands and help search Window. |
| Viewer Tab | Lower-right | Advanced tab for local web content. |

Installation of RStudio

**RStudio Desktop** is available for both Windows and Linux. The open-source RStudio Desktop installation is very simple to install on both operating systems. The licensed version of RStudio has some more features than open-source. Before installing RStudio, let's see what are the additional features in the license version of RStudio.

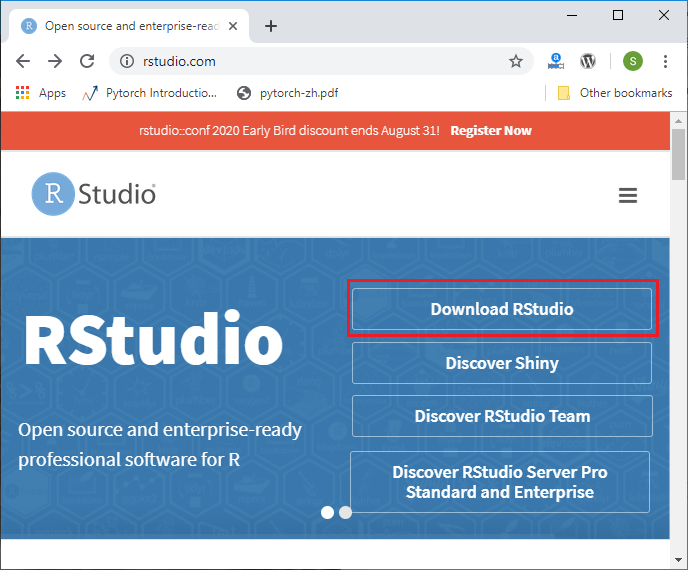
|  |  |  |
| --- | --- | --- |
| **Factor** | **Open-Source** | **Commercial License** |
| **Overview** | 1) Access RStudio locally | All of the features of open-source are include with 1) There is a commercial license for organizations which are not able to use AGPL software. 2) It provides access to priority support. |
| 2) Code completion, syntax highlighting, and smart indentation |
| 3) Can execute R code directly from the source editor |
| 4) Quickly jump to function definitions. |
| 5) Easily manage multiple working directories using projects. |
| 6) Integrated R help and documentation. |
| 7) Provide interactive debugger to diagnose and fix errors quickly. |
| 8) Extensive package deployment tools. |
| **Support** | It supports for community forums only. | 1) It supports priority email. 2) It supports for an 8-hour response during business hour. |
| **License** | AGPL v3 | RStudio License Agreement |
| **Pricing** | Free | $995/year |

Installation on Windows/Linux

On Windows and Linux, it is quite simple to install RStudio. The process of installing RStudio in both the OS is the same. There are the following steps to install RStudio in our Windows/Linux:

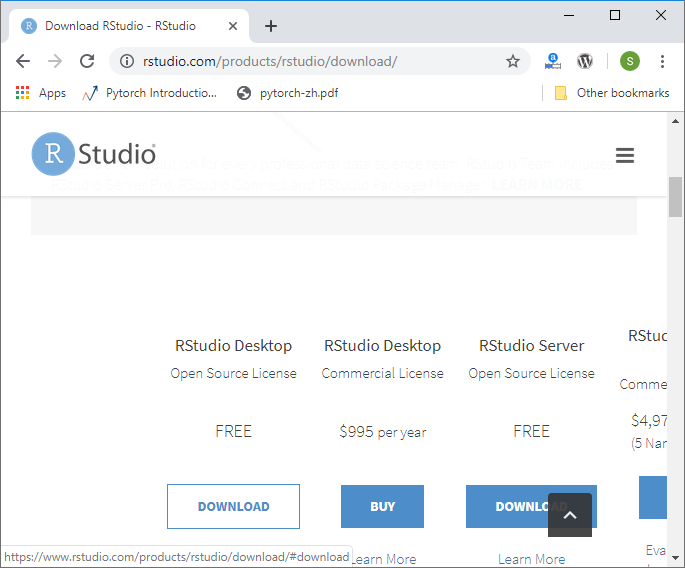
**Step 1:**

In the first step, we visit the RStudio official site and click on **Download RStudio**.



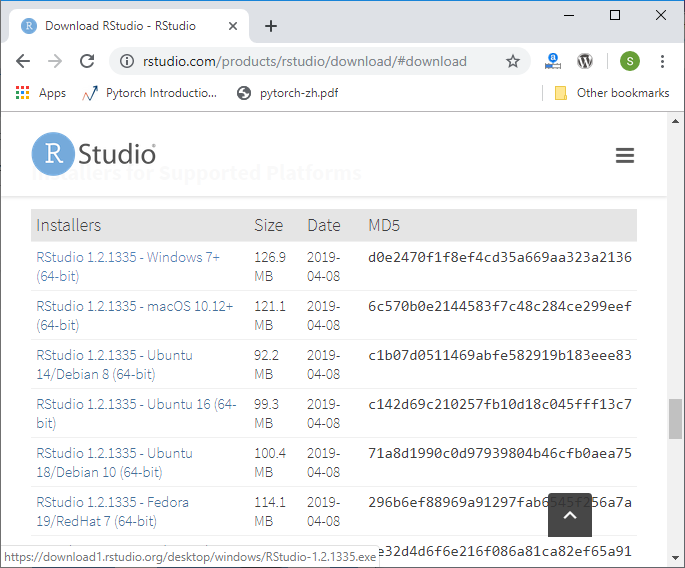
**Step 2:**

In the next step, we will select the RStudio desktop for open-source license and click on download.



**Step 3:**

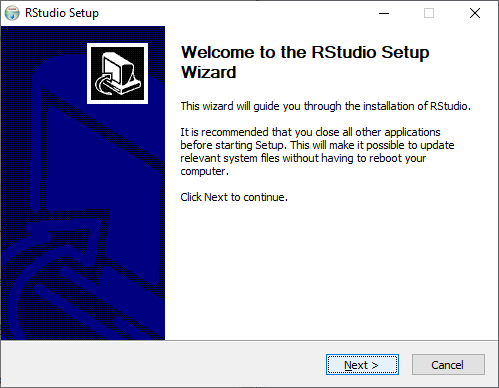
In the next step, we will select the appropriate installer. When we select the installer, our downloading of RStudion setup will start.



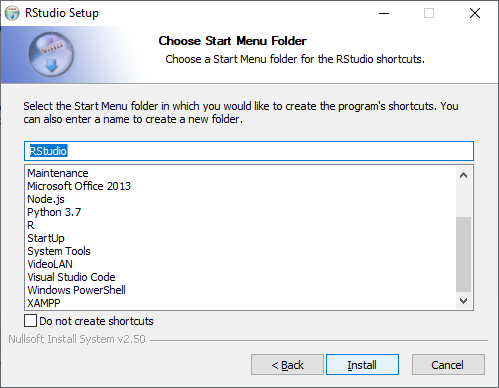
**Step 4:**

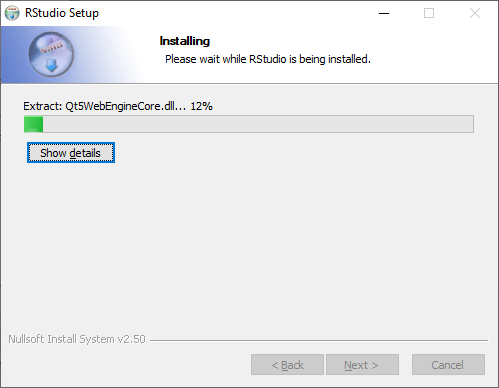
In the next step, we will run our setup in the following way:

1) Click on Next.

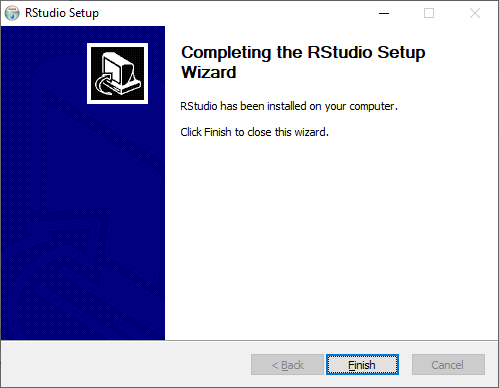


2) Click on Install.

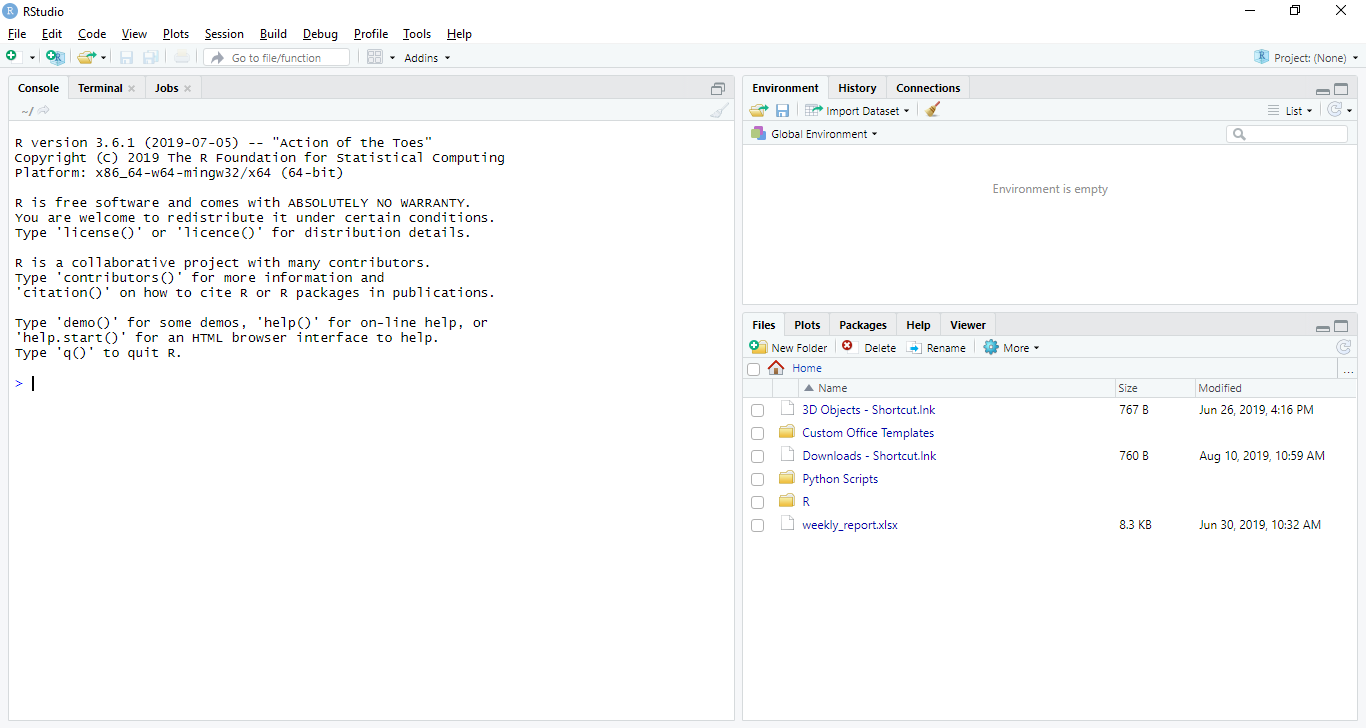




3) Click on finish.



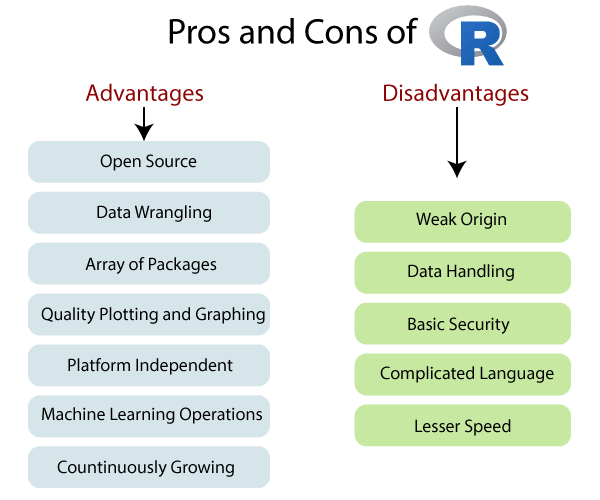
4) RStudio is ready to work.



# R Advantages and Disadvantages

R is the most popular programming language for statistical modeling and analysis. Like other programming languages, R also has some advantages and disadvantages. It is a continuously evolving language which means that many cons will slowly fade away with future updates to R.

There are the following pros and cons of R



Pros

1) Open Source

An open-source language is a language on which we can work without any need for a license or a fee. R is an open-source language. We can contribute to the development of R by optimizing our packages, developing new ones, and resolving issues.

2) Platform Independent

R is a platform-independent language or cross-platform programming language which means its code can run on all operating systems. R enables programmers to develop software for several competing platforms by writing a program only once. R can run quite easily on Windows, Linux, and Mac.

Hello Java Program for Beginners

3) Machine Learning Operations

R allows us to do various machine learning operations such as classification and regression. For this purpose, R provides various packages and features for developing the artificial neural network. R is used by the best data scientists in the world.

4) Exemplary support for data wrangling

R allows us to perform data wrangling. R provides packages such as dplyr, readr which are capable of transforming messy data into a structured form.

5) Quality plotting and graphing

R simplifies quality plotting and graphing. R libraries such as ggplot2 and plotly advocates for visually appealing and aesthetic graphs which set R apart from other programming languages.

6) The array of packages

R has a rich set of packages. R has over 10,000 packages in the CRAN repository which are constantly growing. R provides packages for data science and machine learning operations.

7) Statistics

R is mainly known as the language of statistics. It is the main reason why R is predominant than other programming languages for the development of statistical tools.

8) Continuously Growing

R is a constantly evolving programming language. Constantly evolving means when something evolves, it changes or develops over time, like our taste in music and clothes, which evolve as we get older. R is a state of the art which provides updates whenever any new feature is added.

Cons

1) Data Handling

In R, objects are stored in physical memory. It is in contrast with other programming languages like Python. R utilizes more memory as compared to Python. It requires the entire data in one single place which is in the memory. It is not an ideal option when we deal with Big Data.

2) Basic Security

R lacks basic security. It is an essential part of most programming languages such as Python. Because of this, there are many restrictions with R as it cannot be embedded in a web-application.

3) Complicated Language

R is a very complicated language, and it has a steep learning curve. The people who don't have prior knowledge or programming experience may find it difficult to learn R.

4) Weak Origin

The main disadvantage of R is, it does not have support for dynamic or 3D graphics. The reason behind this is its origin. It shares its origin with a much older programming language "S."

5) Lesser Speed

R programming language is much slower than other programming languages such as MATLAB and Python. In comparison to other programming language, R packages are much slower.

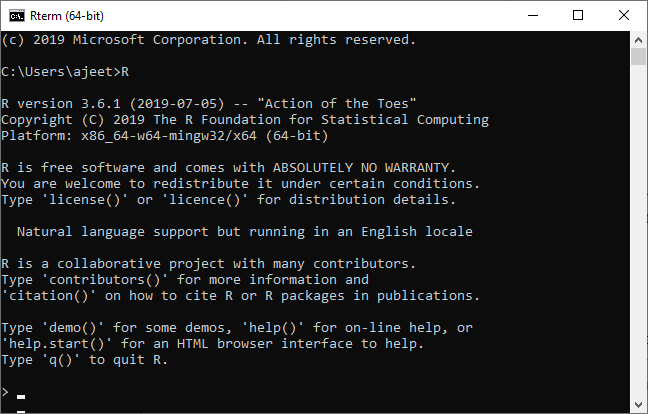
In R, algorithms are spread across different packages. The programmers who have no prior knowledge of packages may find it difficult to implement algorithms.

Syntax of R Programming

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:

9.5M

175

Prime Ministers of India | List of Prime Minister of India (1947-2020)



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

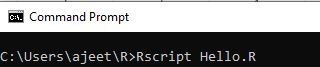
The R script file is another way on which we can write our programs, and then we execute those scripts at our command prompt with the help of R interpreter known as **Rscript**. We make a text file and write the following code. We will save this file with .R extension as:

**Demo.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.



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

Syntax of R Programming

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)

**The trick for multi-line comment**

#Trick for multi-line comment

**if**(FALSE) {

   "R **is** an interpreted computer programming language which was created by

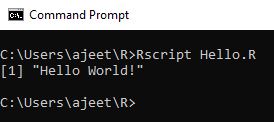
  Ross Ihaka **and** Robert Gentleman at the University of Auckland, New Zealand "

}

#My First program in R programming

string <-"Hello World!"

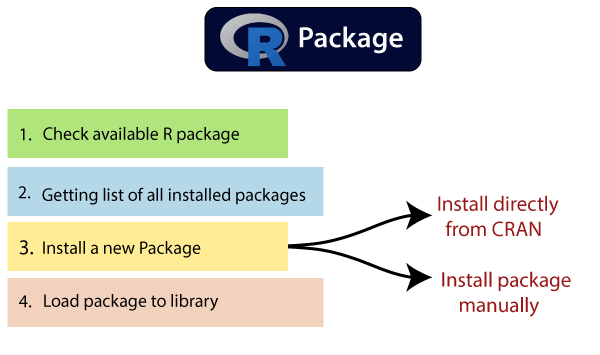
**print**(string)



R Packages

R packages are the collection of R functions, sample data, and compile codes. In the R environment, these packages are stored under a directory called "**library**." During installation, R installs a set of packages. We can add packages later when they are needed for some specific purpose. Only the default packages will be available when we start the R console. Other packages which are already installed will be loaded explicitly to be used by the R program.

There is the following list of commands to be used to check, verify, and use the R packages.



Check Available R Packages

To check the available R Packages, we have to find the library location in which R packages are contained. R provides libPaths() function to find the library locations.

1. libPaths()

When the above code executes, it produces the following project, which may vary depending on the local settings of our PCs & Laptops.

[1] "C:/Users/ajeet/OneDrive/Documents/R/win-library/3.6"

[2] "C:/Program Files/R/R-3.6.1/library"

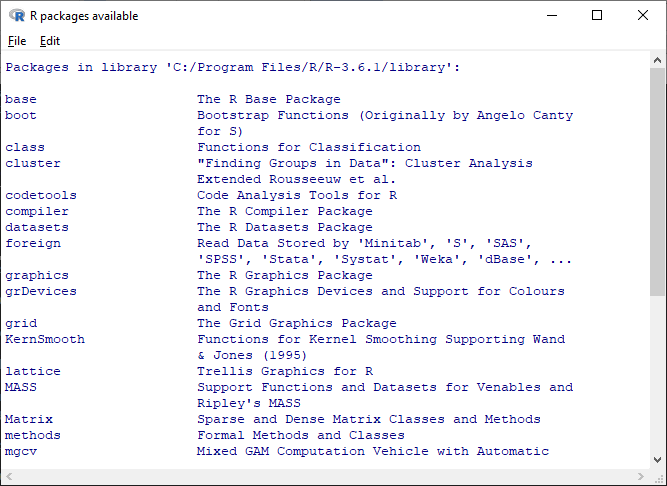
Getting the list of all the packages installed

R provides library() function, which allows us to get the list of all the installed packages.

1. library()

When we execute the above function, it produces the following result, which may vary depending on the local settings of our PCs or laptops.

Packages in library 'C:/Program Files/R/R-3.6.1/library':



Like library() function, R provides search() function to get all packages currently loaded in the R environment.

1. search()

When we execute the above code, it will produce the following result, which may vary depending on the local settings of our PCs and laptops:

[1] ".GlobalEnv" "package:stats" "package:graphics"

[4] "package:grDevices" "package:utils" "package:datasets"

[7] "package:methods" "Autoloads" "package:base"

Install a New Package

In R, there are two techniques to add new R packages. The first technique is installing package directly from the CRAN directory, and the second one is to install it manually after downloading the package to our local system.

**Install directly from CRAN**

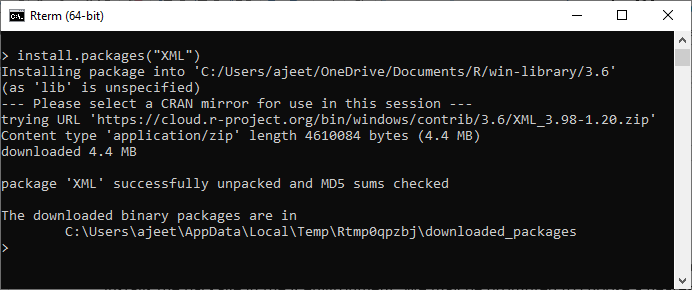
The following command is used to get the packages directly from CRAN webpage and install the package in the R environment. We may be prompted to choose the nearest mirror. Choose the one appropriate to our location.

1. install.packages("Package Name")

The syntax of installing XML package is as follows:

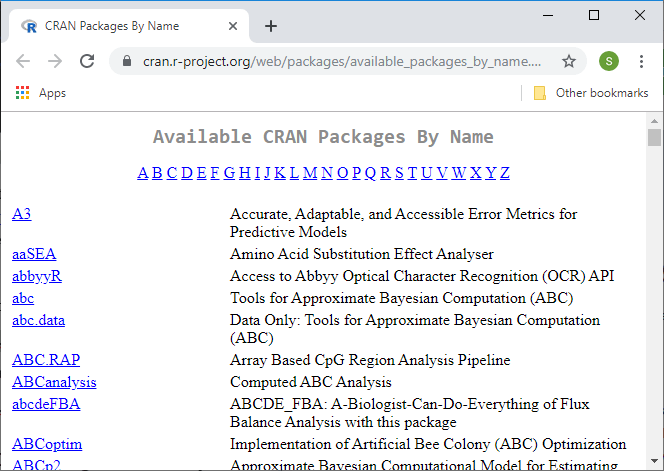
1. install.packages("XML")

**Output**



**Install package manually**

To install a package manually, we first have to download it from <https://cran.r-project.org/web/packages/available_packages_by_name.html>. The required package will be saved as a .zip file in a suitable location in the local system.



Once the downloading has finished, we will use the following command:

1. install.packages(file\_name\_with\_path, repos = NULL, type = "source")

Install the package named "XML"

1. install.packages("C:\Users\ajeet\OneDrive\Desktop\graphics\xml2\_1.2.2.zip", repos = NULL, type = "source")

Load Package to Library

We cannot use the package in our code until it will not be loaded into the current R environment. We also need to load a package which is already installed previously but not available in the current environment.

There is the following command to load a package:

1. library("package Name", lib.loc = "path to library")

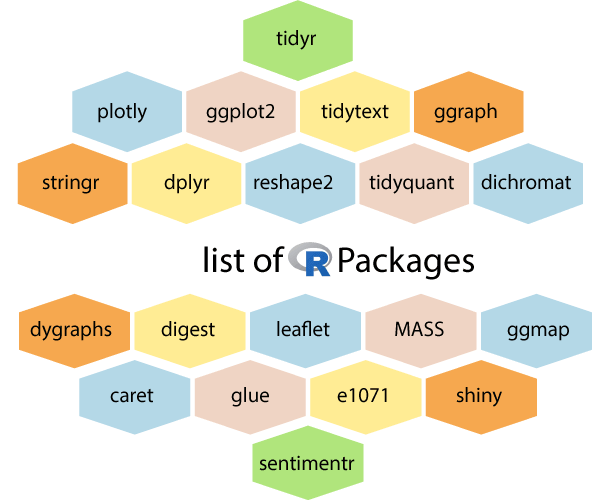
Command to load the XML package

1. install.packages("C:\Users\ajeet\OneDrive\Desktop\graphics\xml2\_1.2.2.zip", repos = NULL,   type = "source")

List of R packages

R is the language of data science which includes a vast repository of packages. These packages appeal to different regions which use R for their data purposes. CRAN has 10,000 packages, making it an ocean of superlative statistical work. There are lots of packages in R, but we will discuss the important one.

There are some mostly used and popular packages which are as follows:



1) tidyr

The word tidyr comes from the word tidy, which means clear. So the **tidyr** package is used to make the data' tidy'. This package works well with dplyr. This package is an evolution of the reshape2 package.

2) ggplot2

R allows us to create graphics declaratively. R provides the **ggplot** package for this purpose. This package is famous for its elegant and quality graphs which sets it apart from other visualization packages.

3) ggraph

R provides an extension of ggplot known as **ggraph**. The limitation of **ggplot** is the dependency on tabular data is taken away in ggraph.

4) dplyr

R allows us to perform data wrangling and data analysis. R provides the **dplyr** library for this purpose. This library facilitates several functions for the data frame in R.

5) tidyquant

The tidyquant is a financial package which is used for carrying out quantitative financial analysis. This package adds to the **tidyverse** universe as a financial package which is used for importing, analyzing and visualizing the data.

6) dygraphs

The dygraphs package provides an interface to the main JavaScript library which we can use for charting. This package is essentially used for plotting time-series data in R.

7) leaflet

For creating interactive visualization, R provides the **leaflet** package. This package is an open-source JavaScript library. The world's popular websites like the New York Times, Github and Flicker, etc. are using leaflet. The leaflet package makes it easier to interact with these sites.

8) ggmap

For delineating spatial visualization, the **ggmap** package is used. It is a mapping package which consists of various tools for geolocating and routing.

9) glue

R provides the **glue** package to perform the operations of data wrangling. This package is used for evaluating R expressions which are present within the string.

10) shiny

R allows us to develop interactive and aesthetically pleasing web apps by providing a **shiny** package. This package provides various extensions with HTML widgets, CSS, and JavaScript.

11) plotly

The plotly package provides online interactive and quality graphs. This package extends upon the JavaScript library **-plotly.js**.

12) tidytext

The **tidytext** package provides various functions of text mining for word processing and carrying out analysis through ggplot, dplyr, and other miscellaneous tools.

13) stringr

The stringr package provides simplicity and consistency to use wrappers for the '**stringi**' package. The stringi package facilitates common string operations.

14) reshape2

This package facilitates flexible reorganization and aggregation of data using melt () and decast () functions.

15) dichromat

The R dichromat package is used to remove Red-Green or Blue-Green contrasts from the colors.

16) digest

The digest package is used for the creation of cryptographic hash objects of R functions.

17) MASS

The **MASS** package provides a large number of statistical functions. It provides datasets that are in conjunction with the book "Modern Applied Statistics with S."

18) caret

R allows us to perform classification and regression tasks by providing the caret package. **CaretEnsemble** is a feature of caret which is used for the combination of different models.

19) e1071

The **e1071** library provides useful functions which are essential for data analysis like Naive Bayes, Fourier Transforms, SVMs, Clustering, and other miscellaneous functions.

20) sentimentr

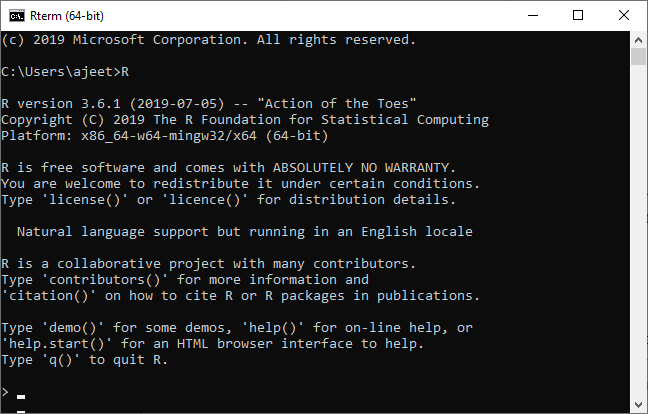
The sentiment package provides functions for carrying out sentiment analysis. It is used to calculate text polarity at the sentence level and to perform aggregation by rows or grouping variables.

Syntax of R Programming

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:

Features of Java - Javatpoint



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.

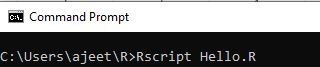
R Script File

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:

**Demo.R**

1. string <-"Hello World!"
2. **print**(string)

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



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

Syntax of R Programming

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.

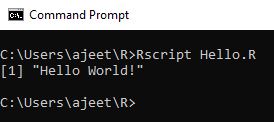
In R programming there is only single-line comment. R doesn't support multi-line comment. But if we want to perform multi-line comments, then we can add our code in a false block.

**Single-line comment**

1. #My First program in R programming
2. string <-"Hello World!"
3. **print**(string)

**The trick for multi-line comment**

1. #Trick for multi-line comment
2. **if**(FALSE) {
3. "R **is** an interpreted computer programming language which was created by
4. Ross Ihaka **and** Robert Gentleman at the University of Auckland, New Zealand "
5. }
6. #My First program in R programming
7. string <-"Hello World!"
8. **print**(string)

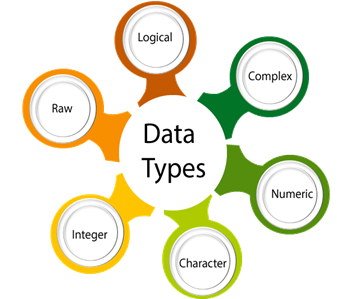


Data Types in R Programming

In programming languages, we need to use various variables to store various information. Variables are the reserved memory location to store values. As we create a variable in our program, some space is reserved in memory.

In R, there are several data types such as integer, string, etc. 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:



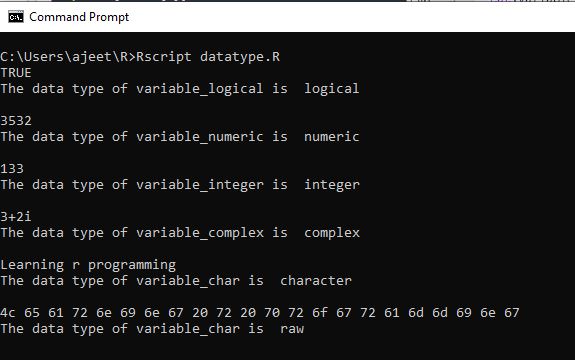
|  |  |  |
| --- | --- | --- |
| **Data type** | **Example** | **Description** |
| **Logical** | True, False | It is a special data type for data with only two possible values which can be construed as true/false. |
| **Numeric** | 12,32,112,5432 | Decimal value is called numeric in R, and it is the 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** |  | A raw data type is used to holds raw bytes. |

Let's see an example for better understanding of data types:

Exception Handling in Java - Javatpoint

1. #Logical Data type
2. variable\_logical<- TRUE
3. cat(variable\_logical,"\n")
4. cat("The data type of variable\_logical is ",**class**(variable\_logical),"\n\n")
6. #Numeric Data type
7. variable\_numeric<- 3532
8. cat(variable\_numeric,"\n")
9. cat("The data type of variable\_numeric is ",**class**(variable\_numeric),"\n\n")
11. #Integer Data type
12. variable\_integer<- 133L
13. cat(variable\_integer,"\n")
14. cat("The data type of variable\_integer is ",**class**(variable\_integer),"\n\n")
16. #Complex Data type
17. variable\_complex<- 3+2i
18. cat(variable\_complex,"\n")
19. cat("The data type of variable\_complex is ",**class**(variable\_complex),"\n\n")
21. #Character Data type
22. variable\_char<- "Learning r programming"
23. cat(variable\_char,"\n")
24. cat("The data type of variable\_char is ",**class**(variable\_char),"\n\n")
26. #Raw Data type
27. variable\_raw<- charToRaw("Learning r programming")
28. cat(variable\_raw,"\n")
29. cat("The data type of variable\_char is ",**class**(variable\_raw),"\n\n")

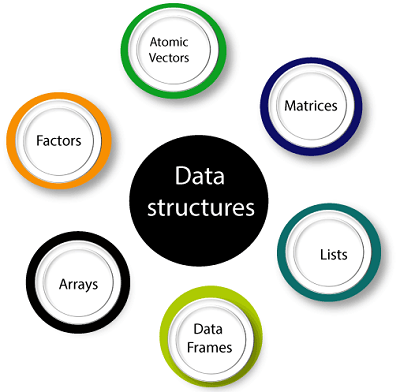
When we execute the following program, it will give us the following output:



Data Structures in R Programming

Data structures are very important to understand. Data structure are the objects which we will manipulate in our day-to-day basis in R. Dealing with object conversions is the most common sources of despairs for beginners. We can say that everything in R is an object.

R has many data structures, which include:



1. Atomic vector
2. List
3. Array
4. Matrices
5. Data Frame
6. Factors

Vectors

A vector is the basic data structure in R, or we can say vectors are the most basic R data objects. There are six types of atomic vectors such as logical, integer, character, double, and raw. **"A vector is a collection of elements which is most commonly of mode character, integer, logical or numeric"** A vector can be one of the following two types:

1. Atomic vector
2. Lists

List

In R, **the list** is the container. Unlike an atomic vector, the list is not restricted to be a single mode. A list contains a mixture of data types. The list is also known as generic vectors because the element of the list can be of any type of R object. **"A list is a special type of vector in which each element can be a different type."**

C++ vs Java

We can create a list with the help of list() or as.list(). We can use vector() to create a required length empty list.

Arrays

There is another type of data objects which can store data in more than two dimensions known as arrays. **"An array is a collection of a similar data type with contiguous memory allocation."** Suppose, if we create an array of dimension (2, 3, 4) then it creates four rectangular matrices of two rows and three columns.

In R, an array is created with the help of **array()** function. This function takes a vector as an input and uses the value in the dim parameter to create an array.

Matrices

A matrix is an R object in which the elements are arranged in a two-dimensional rectangular layout. In the matrix, elements of the same atomic types are contained. For mathematical calculation, this can use a matrix containing the numeric element. A matrix is created with the help of the matrix() function in R.

**Syntax**

The basic syntax of creating a matrix is as follows:

1. matrix(data, no\_row, no\_col, by\_row, dim\_name)

Data Frames

A **data frame** is a two-dimensional array-like structure, or we can say it is a table in which each column contains the value of one variable, and row contains the set of value from each column.

There are the following characteristics of a data frame:

1. The column name will be non-empty.
2. The row names will be unique.
3. A data frame stored numeric, factor or character type data.
4. Each column will contain same number of data items.

Factors

**Factors** are also data objects that are used to categorize the data and store it as levels. Factors can store both strings and integers. Columns have a limited number of unique values so that factors are very useful in columns. It is very useful in data analysis for statistical modeling.

Factors are created with the help of **factor()** function by taking a vector as an input parameter.

Variables in R Programming

Variables are used to store the information to be manipulated and referenced in the R program. The R variable can store an atomic vector, a group of atomic vectors, or a combination of many R objects.

Language like C++ is statically typed, but R is a dynamically typed, means it check the type of data type when the statement is run. A valid variable name contains letter, numbers, dot and underlines characters. A variable name should start with a letter or the dot not followed by a number.

|  |  |  |
| --- | --- | --- |
| **Name of variable** | **Validity** | **Reason for valid and invalid** |
| **\_var\_name** | Invalid | Variable name can't start with an underscore(\_). |
| **var\_name, var.name** | Valid | Variable can start with a dot, but dot should not be followed by a number. In this case, the variable will be invalid. |
| **var\_name%** | Invalid | In R, we can't use any special character in the variable name except dot and underscore. |
| **2var\_name** | Invalid | Variable name cant starts with a numeric digit. |
| **.2var\_name** | Invalid | A variable name cannot start with a dot which is followed by a digit. |
| **var\_name2** | Valid | The variable contains letter, number and underscore and starts with a letter. |

Assignment of variable

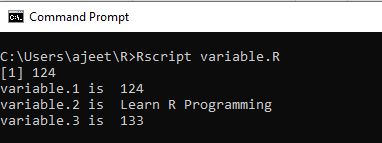
In R programming, there are three operators which we can use to assign the values to the variable. We can use leftward, rightward, and equal\_to operator for this purpose.

There are two functions which are used to print the value of the variable i.e., print() and cat(). The cat() function combines multiples values into a continuous print output.

SQL CREATE TABLE

1. # Assignment using equal operator.
2. variable.1 = 124
4. # Assignment using leftward operator.
5. variable.2 <- "Learn R Programming"
7. # Assignment using rightward operator.
8. 133L -> variable.3
10. **print**(variable.1)
11. cat ("variable.1 is ", variable.1 ,"\n")
12. cat ("variable.2 is ", variable.2 ,"\n")
13. cat ("variable.3 is ", variable.3 ,"\n")

When we execute the above code in our R command prompt, it will give us the following output:



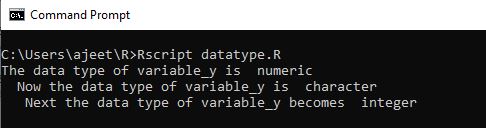
Data types of variable

R programming is a dynamically typed language, which means that we can change the data type of the same variable again and again in our program. Because of its dynamic nature, a variable is not declared of any data type. It gets the data type from the R-object, which is to be assigned to the variable.

We can check the data type of the variable with the help of the class() function. Let's see an example:

1. variable\_y<- 124
2. cat("The data type of variable\_y is ",**class**(variable\_y),"\n")
4. variable\_y<- "Learn R Programming"
5. cat("  Now the data type of variable\_y is ",**class**(variable\_y),"\n")
6. variable\_y<- 133L
7. cat("   Next the data type of variable\_y becomes ",**class**(variable\_y),"\n")

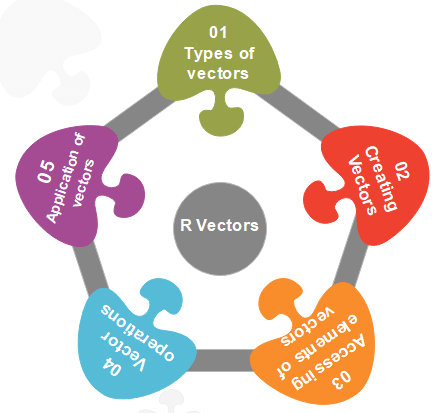
When we execute the above code in our R command prompt, it will give us the following output:



R Vector

A **vector** is a basic data structure which plays an important role in R programming.

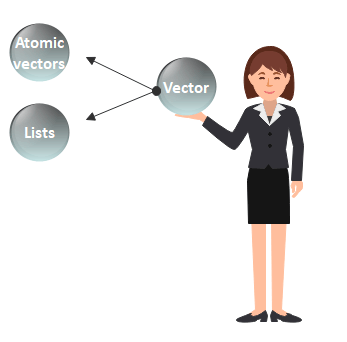
In R, a sequence of elements which share the same data type is known as vector. A vector supports logical, integer, double, character, complex, or raw data type. The elements which are contained in vector known as **components** of the vector. We can check the type of vector with the help of the **typeof()** function.



The length is an important property of a vector. A vector length is basically the number of elements in the vector, and it is calculated with the help of the length() function.

Vector is classified into two parts, i.e., **Atomic vectors** and **Lists**. They have three common properties, i.e., **function type, function length**, and **attribute function**.

Skip Ad



There is only one difference between atomic vectors and lists. In an atomic vector, all the elements are of the same type, but in the list, the elements are of different data types. In this section, we will discuss only the atomic vectors. We will discuss lists briefly in the next topic.

How to create a vector in R?

In R, we use c() function to create a vector. This function returns a one-dimensional array or simply vector. The c() function is a generic function which combines its argument. All arguments are restricted with a common data type which is the type of the returned value. There are various other ways to create a vector in R, which are as follows:

1) Using the colon(:) operator

We can create a vector with the help of the colon operator. There is the following syntax to use colon operator:

1. z**<-x:y**

This operator creates a vector with elements from x to y and assigns it to z.

**Example:**

1. a**<-4:-10**
2. a

**Output**

[1] 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10

2) Using the seq() function

In R, we can create a vector with the help of the seq() function. A sequence function creates a sequence of elements as a vector. The seq() function is used in two ways, i.e., by setting step size with ?by' parameter or specifying the length of the vector with the 'length.out' feature.

**Example:**

1. seq\_vec**<-seq**(1,4,by=0.5)
2. seq\_vec
3. class(seq\_vec)

**Output**

[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0

**Example:**

1. seq\_vec**<-seq**(1,4,length.out=6)
2. seq\_vec
3. class(seq\_vec)

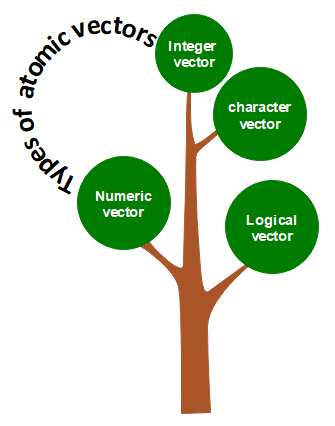
**Output**

[1] 1.0 1.6 2.2 2.8 3.4 4.0

[1] "numeric"

Atomic vectors in R

In R, there are four types of atomic vectors. Atomic vectors play an important role in Data Science. Atomic vectors are created with the help of **c()** function. These atomic vectors are as follows:



Numeric vector

The decimal values are known as numeric data types in R. If we assign a decimal value to any variable d, then this d variable will become a numeric type. A vector which contains numeric elements is known as a numeric vector.

**Example:**

1. d**<-45.5**
2. num\_vec**<-c**(10.1, 10.2, 33.2)
3. d
4. num\_vec
5. class(d)
6. class(num\_vec)

**Output**

[1] 45.5

[1] 10.1 10.2 33.2

[1] "numeric"

[1] "numeric"

Integer vector

A non-fraction numeric value is known as integer data. This integer data is represented by "Int." The Int size is 2 bytes and long Int size of 4 bytes. There is two way to assign an integer value to a variable, i.e., by using as.integer() function and appending of L to the value.

A vector which contains integer elements is known as an integer vector.

**Example:**

1. d**<-as.integer**(5)
2. e**<-5L**
3. int\_vec**<-c**(1,2,3,4,5)
4. int\_vec**<-as.integer**(int\_vec)
5. int\_vec1**<-c**(1L,2L,3L,4L,5L)
6. class(d)
7. class(e)
8. class(int\_vec)
9. class(int\_vec1)

**Output**

[1] "integer"

[1] "integer"

[1] "integer"

[1] "integer"

Character vector

A character is held as a one-byte integer in memory. In R, there are two different ways to create a character data type value, i.e., using as.character() function and by typing string between double quotes("") or single quotes('').

A vector which contains character elements is known as an integer vector.

**Example:**

1. d**<-**'shubham'
2. e**<-**"Arpita"
3. f**<-65**
4. f**<-as.character**(f)
5. d
6. e
7. f
8. char\_vec**<-c**(1,2,3,4,5)
9. char\_vec**<-as.character**(char\_vec)
10. char\_vec1**<-c**("shubham","arpita","nishka","vaishali")
11. char\_vec
12. class(d)
13. class(e)
14. class(f)
15. class(char\_vec)
16. class(char\_vec1)

**Output**

[1] "shubham"

[1] "Arpita"

[1] "65"

[1] "1" "2" "3" "4" "5"

[1] "shubham" "arpita" "nishka" "vaishali"

[1] "character"

[1] "character"

[1] "character"

[1] "character"

[1] "character"

Logical vector

The logical data types have only two values i.e., True or False. These values are based on which condition is satisfied. A vector which contains Boolean values is known as the logical vector.

**Example:**

1. d**<-as.integer**(5)
2. e**<-as.integer**(6)
3. f**<-as.integer**(7)
4. g**<-d>**e
5. h**<-e<f**
6. g
7. h
8. log\_vec**<-c**(d**<e**, d**<f**, e**<d**,e**<f**,f**<d**,f**<e**)
9. log\_vec
10. class(g)
11. class(h)
12. class(log\_vec)

**Output**

[1] FALSE

[1] TRUE

[1] TRUE TRUE FALSE TRUE FALSE FALSE

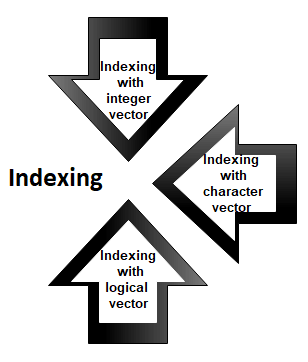
[1] "logical"

[1] "logical"

[1] "logical"

Accessing elements of vectors

We can access the elements of a vector with the help of vector indexing. Indexing denotes the position where the value in a vector is stored. Indexing will be performed with the help of integer, character, or logic.



1) Indexing with integer vector

On integer vector, indexing is performed in the same way as we have applied in C, C++, and java. There is only one difference, i.e., in C, C++, and java the indexing starts from 0, but in R, the indexing starts from 1. Like other programming languages, we perform indexing by specifying an integer value in square braces [] next to our vector.

**Example:**

1. seq\_vec**<-seq**(1,4,length.out=6)
2. seq\_vec
3. seq\_vec[2]

**Output**

[1] 1.0 1.6 2.2 2.8 3.4 4.0

[1] 1.6

2) Indexing with a character vector

In character vector indexing, we assign a unique key to each element of the vector. These keys are uniquely defined as each element and can be accessed very easily. Let's see an example to understand how it is performed.

**Example:**

1. char\_vec**<-c**("shubham"=22,"arpita"=23,"vaishali"=25)
2. char\_vec
3. char\_vec["arpita"]

**Output**

shubham arpita vaishali

22 23 25

arpita

23

3) Indexing with a logical vector

In logical indexing, it returns the values of those positions whose corresponding position has a logical vector TRUE. Let see an example to understand how it is performed on vectors.

**Example:**

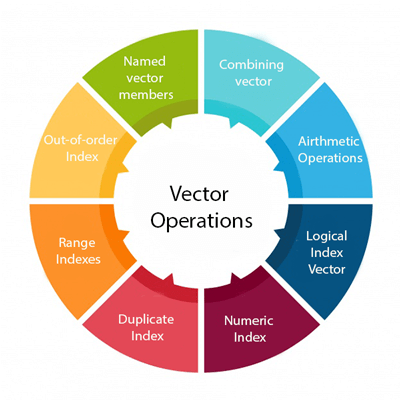
1. a**<-c**(1,2,3,4,5,6)
2. a[c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE)]

**Output**

[1] 1 3 4 6

Vector Operation

In R, there are various operation which is performed on the vector. We can add, subtract, multiply or divide two or more vectors from each other. In data science, R plays an important role, and operations are required for data manipulation. There are the following types of operation which are performed on the vector.



1) Combining vectors

The c() function is not only used to create a vector, but also it is also used to combine two vectors. By combining one or more vectors, it forms a new vector which contains all the elements of each vector. Let see an example to see how c() function combines the vectors.

**Example:**

1. p**<-c**(1,2,4,5,7,8)
2. q**<-c**("shubham","arpita","nishka","gunjan","vaishali","sumit")
3. r**<-c**(p,q)

**Output**

[1] "1" "2" "4" "5" "7" "8"

[7] "shubham" "arpita" "nishka" "gunjan" "vaishali" "sumit"

2) Arithmetic operations

We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors. We can add, subtract, multiply, or divide two vectors. Let see an example to understand how arithmetic operations are performed on vectors.

**Example:**

1. a**<-c**(1,3,5,7)
2. b**<-c**(2,4,6,8)
3. a+b
4. a-b
5. a\*b
6. a/b
7. a

**Output**

[1] 3 7 11 15

[1] -1 -1 -1 -1

[1] 2 12 30 56

[1] 0.5000000 0.7500000 0.8333333 0.8750000

[1] 1 3 5 7

3) Logical Index vector

With the help of the logical index vector in R, we can form a new vector from a given vector. This vector has the same length as the original vector. The vector members are TRUE only when the corresponding members of the original vector are included in the slice; otherwise, it will be false. Let see an example to understand how a new vector is formed with the help of logical index vector.

**Example:**

1. a**<-c**("Shubham","Arpita","Nishka","Vaishali","Sumit","Gunjan")
2. b**<-c**(TRUE,FALSE,TRUE,TRUE,FALSE,FALSE)
3. a[b]

**Output**

[1] "Shubham" "Nishka" "Vaishali"

4) Numeric Index

In R, we specify the index between square braces [ ] for indexing a numerical value. If our index is negative, it will return us all the values except for the index which we have specified. For example, specifying [-3] will prompt R to convert -3 into its absolute value and then search for the value which occupies that index.

**Example:**

1. q**<-c**("shubham","arpita","nishka","gunjan","vaishali","sumit")
2. q[2]
3. q[-4]
4. q[15]

**Output**

[1] "arpita"

[1] "shubham" "arpita" "nishka" "vaishali" "sumit"

[1] NA

5) Duplicate Index

An index vector allows duplicate values which means we can access one element twice in one operation. Let see an example to understand how duplicate index works.

**Example:**

1. q**<-c**("shubham","arpita","nishka","gunjan","vaishali","sumit")
2. q[c(2,4,4,3)]

**Output**

[1] "arpita" "gunjan" "gunjan" "nishka"

6) Range Indexes

Range index is used to slice our vector to form a new vector. For slicing, we used colon(:) operator. Range indexes are very helpful for the situation involving a large operator. Let see an example to understand how slicing is done with the help of the colon operator to form a new vector.

**Example:**

1. q**<-c**("shubham","arpita","nishka","gunjan","vaishali","sumit")
2. b**<-q**[2:5]
3. b

**Output**

[1] "arpita" "nishka" "gunjan" "vaishali"

7) Out-of-order Indexes

In R, the index vector can be out-of-order. Below is an example in which a vector slice with the order of first and second values reversed.

**Example:**

1. q**<-c**("shubham","arpita","nishka","gunjan","vaishali","sumit")b**<-q**[2:5]
2. q[c(2,1,3,4,5,6)]

**Output**

[1] "arpita" "shubham" "nishka" "gunjan" "vaishali" "sumit"

8) Named vectors members

We first create our vector of characters as:

1. z=c("TensorFlow","PyTorch")
2. z

**Output**

[1] "TensorFlow" "PyTorch"

Once our vector of characters is created, we name the first vector member as "Start" and the second member as "End" as:

1. names(z)=c("Start","End")
2. z

**Output**

Start End

"TensorFlow" "PyTorch"

We retrieve the first member by its name as follows:

1. z["Start"]

**Output**

Start

"TensorFlow"

We can reverse the order with the help of the character string index vector.

1. z[c("Second","First")]

**Output**

Second First

"PyTorch" "TensorFlow"

Applications of vectors

1. In machine learning for principal component analysis vectors are used. They are extended to eigenvalues and eigenvector and then used for performing decomposition in vector spaces.
2. The inputs which are provided to the deep learning model are in the form of vectors. These vectors consist of standardized data which is supplied to the input layer of the neural network.
3. In the development of support vector machine algorithms, vectors are used.
4. Vector operations are utilized in neural networks for various operations like image recognition and text processing.

R Lists

In R, lists are the second type of vector. Lists are the objects of R which contain elements of different types such as number, vectors, string and another list inside it. It can also contain a function or a matrix as its elements. A list is a data structure which has components of mixed data types. We can say, a list is a generic vector which contains other objects.

Example

1. vec **<-** c(3,4,5,6)
2. char\_vec**<-c**("shubham","nishka","gunjan","sumit")
3. logic\_vec**<-c**(TRUE,FALSE,FALSE,TRUE)
4. out\_list**<-list**(vec,char\_vec,logic\_vec)
5. out\_list

**Output:**

[[1]]

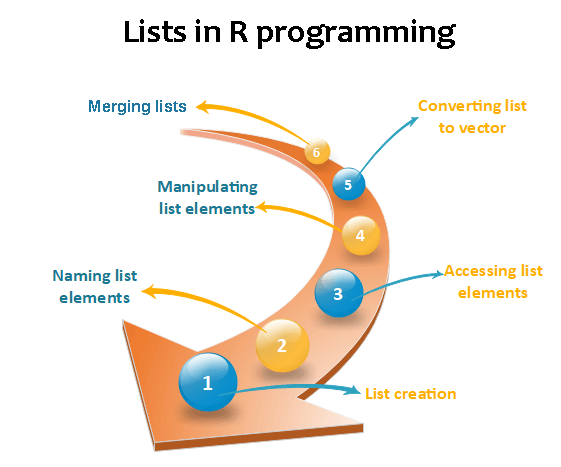
[1] 3 4 5 6

[[2]]

[1] "shubham" "nishka" "gunjan" "sumit"

[[3]]

[1] TRUE FALSE FALSE TRUE



Lists creation

The process of creating a list is the same as a vector. In R, the vector is created with the help of c() function. Like c() function, there is another function, i.e., list() which is used to create a list in R. A list avoid the drawback of the vector which is data type. We can add the elements in the list of different data types.

**Syntax**

Skip Ad

1. list()

**Example 1:** Creating list with same data type

1. list\_1**<-list**(1,2,3)
2. list\_2**<-list**("Shubham","Arpita","Vaishali")
3. list\_3**<-list**(c(1,2,3))
4. list\_4**<-list**(TRUE,FALSE,TRUE)
5. list\_1
6. list\_2
7. list\_3
8. list\_4

**Output:**

[[1]]

[1] 1

[[2]]

[1] 2

[[3]]

[1] 3

[[1]]

[1] "Shubham"

[[2]]

[1] "Arpita"

[[3]]

[1] "Vaishali"

[[1]]

[1] 1 2 3

[[1]]

[1] TRUE

[[2]]

[1] FALSE

[[3]]

[1] TRUE

**Example 2:** Creating the list with different data type

1. list\_data**<-list**("Shubham","Arpita",c(1,2,3,4,5),TRUE,FALSE,22.5,12L)
2. print(list\_data)

In the above example, the list function will create a list with character, logical, numeric, and vector element. It will give the following output

**Output:**

[[1]]

[1] "Shubham"

[[2]]

[1] "Arpita"

[[3]]

[1] 1 2 3 4 5

[[4]]

[1] TRUE

[[5]]

[1] FALSE

[[6]]

[1] 22.5

[[7]]

[1] 12

Giving a name to list elements

R provides a very easy way for accessing elements, i.e., by giving the name to each element of a list. By assigning names to the elements, we can access the element easily. There are only three steps to print the list data corresponding to the name:

1. Creating a list.
2. Assign a name to the list elements with the help of names() function.
3. Print the list data.

Let see an example to understand how we can give the names to the list elements.

**Example**

1. # Creating a list containing a vector, a matrix and a list.
2. list\_data **<-** list(c("Shubham","Nishka","Gunjan"), matrix(c(40,80,60,70,90,80), nrow = 2),
3. list("BCA","MCA","B.tech"))
5. # Giving names to the elements in the list.
6. names(list\_data) **<-** c("Students", "Marks", "Course")
8. # Show the list.
9. print(list\_data)

**Output:**

$Students

[1] "Shubham" "Nishka" "Gunjan"

$Marks

[,1] [,2] [,3]

[1,] 40 60 90

[2,] 80 70 80

$Course

$Course[[1]]

[1] "BCA"

$Course[[2]]

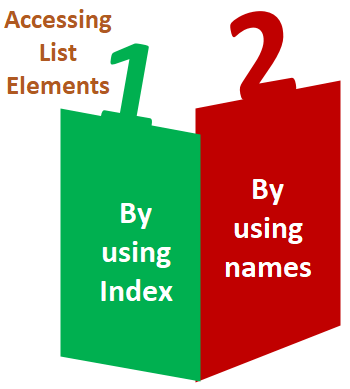
[1] "MCA"

$Course[[3]]

[1] "B. tech."

Accessing List Elements

R provides two ways through which we can access the elements of a list. First one is the indexing method performed in the same way as a vector. In the second one, we can access the elements of a list with the help of names. It will be possible only with the named list.; we cannot access the elements of a list using names if the list is normal.



Let see an example of both methods to understand how they are used in the list to access elements.

**Example 1:** Accessing elements using index

1. # Creating a list containing a vector, a matrix and a list.
2. list\_data **<-** list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80), nrow = 2),
3. list("BCA","MCA","B.tech"))
4. # Accessing the first element of the list.
5. print(list\_data[1])
7. # Accessing the third element. The third element is also a list, so all its elements will be printed.
8. print(list\_data[3])

**Output:**

[[1]]

[1] "Shubham" "Arpita" "Nishka"

[[1]]

[[1]][[1]]

[1] "BCA"

[[1]][[2]]

[1] "MCA"

[[1]][[3]]

[1] "B.tech"

**Example 2:** Accessing elements using names

1. # Creating a list containing a vector, a matrix and a list.
2. list\_data **<-** list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80), nrow = 2),list("BCA","MCA","B.tech"))
3. # Giving names to the elements in the list.
4. names(list\_data) **<-** c("Student", "Marks", "Course")
5. # Accessing the first element of the list.
6. print(list\_data["Student"])
7. print(list\_data$Marks)
8. print(list\_data)

**Output:**

$Student

[1] "Shubham" "Arpita" "Nishka"

[,1] [,2] [,3]

[1,] 40 60 90

[2,] 80 70 80

$Student

[1] "Shubham" "Arpita" "Nishka"

$Marks

[,1] [,2] [,3]

[1,] 40 60 90

[2,] 80 70 80

$Course

$Course[[1]]

[1] "BCA"

$Course[[2]]

[1] "MCA"

$Course[[3]]

[1] "B. tech."

Manipulation of list elements

R allows us to add, delete, or update elements in the list. We can update an element of a list from anywhere, but elements can add or delete only at the end of the list. To remove an element from a specified index, we will assign it a null value. We can update the element of a list by overriding it from the new value. Let see an example to understand how we can add, delete, or update the elements in the list.

**Example**

1. # Creating a list containing a vector, a matrix and a list.
2. list\_data **<-** list(c("Shubham","Arpita","Nishka"), matrix(c(40,80,60,70,90,80), nrow = 2),
3. list("BCA","MCA","B.tech"))
5. # Giving names to the elements in the list.
6. names(list\_data) **<-** c("Student", "Marks", "Course")
8. # Adding element at the end of the list.
9. list\_data[4] **<-** "Moradabad"
10. print(list\_data[4])
12. # Removing the last element.
13. list\_data[4] **<-** NULL
15. # Printing the 4th Element.
16. print(list\_data[4])
18. # Updating the 3rd Element.
19. list\_data[3] **<-** "Masters of computer applications"
20. print(list\_data[3])

**Output:**

[[1]]

[1] "Moradabad"

$<NA>

NULL

$Course

[1] "Masters of computer applications"

Converting list to vector

There is a drawback with the list, i.e., we cannot perform all the arithmetic operations on list elements. To remove this, drawback R provides unlist() function. This function converts the list into vectors. In some cases, it is required to convert a list into a vector so that we can use the elements of the vector for further manipulation.

The unlist() function takes the list as a parameter and change into a vector. Let see an example to understand how to unlist() function is used in R.

**Example**

1. # Creating lists.
2. list1 **<-** list(10:20)
3. print(list1)
5. list2 **<-list**(5:14)
6. print(list2)
8. # Converting the lists to vectors.
9. v1 **<-** unlist(list1)
10. v2 **<-** unlist(list2)
12. print(v1)
13. print(v2)
15. adding the vectors
16. result **<-** v1+v2
17. print(result)

**Output:**

[[1]]

[1] 1 2 3 4 5

[[1]]

[1] 10 11 12 13 14

[1] 1 2 3 4 5

[1] 10 11 12 13 14

[1] 11 13 15 17 19

Merging Lists

R allows us to merge one or more lists into one list. Merging is done with the help of the list() function also. To merge the lists, we have to pass all the lists into list function as a parameter, and it returns a list which contains all the elements which are present in the lists. Let see an example to understand how the merging process is done.

**Example**

1. # Creating two lists.
2. Even\_list **<-** list(2,4,6,8,10)
3. Odd\_list **<-** list(1,3,5,7,9)
5. # Merging the two lists.
6. merged.list **<-** list(Even\_list,Odd\_list)
8. # Printing the merged list.
9. print(merged.list)

**Output:**

[[1]]

[[1]][[1]]

[1] 2

[[1]][[2]]

[1] 4

[[1]][[3]]

[1] 6

[[1]][[4]]

[1] 8

[[1]][[5]]

[1] 10

[[2]]

[[2]][[1]]

[1] 1

[[2]][[2]]

[1] 3

[[2]][[3]]

[1] 5

[[2]][[4]]

[1] 7

[[2]][[5]]

[1] 9

R Arrays

In R, arrays are the data objects which allow us to store data in more than two dimensions. In R, an array is created with the help of the **array()** function. This array() function takes a vector as an input and to create an array it uses vectors values in the **dim** parameter.

**For example**- if we will create an array of dimension (2, 3, 4) then it will create 4 rectangular matrices of 2 row and 3 columns.

R Array Syntax

There is the following syntax of R arrays:

1. array\_name **<-** array(data, dim= (row\_size, column\_size, matrices, dim\_names))

**data**

10 Sec

Exception Handling in Java - Javatpoint

The data is the first argument in the array() function. It is an input vector which is given to the array.

**matrices**

In R, the array consists of multi-dimensional matrices.

**row\_size**

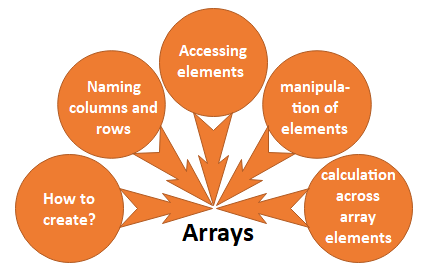
This parameter defines the number of row elements which an array can store.

**column\_size**

This parameter defines the number of columns elements which an array can store.

**dim\_names**

This parameter is used to change the default names of rows and columns.



How to create?

In R, array creation is quite simple. We can easily create an array using vector and array() function. In array, data is stored in the form of the matrix. There are only two steps to create a matrix which are as follows

1. In the first step, we will create two vectors of different lengths.
2. Once our vectors are created, we take these vectors as inputs to the array.

Let see an example to understand how we can implement an array with the help of the vectors and array() function.

**Example**

1. #Creating two vectors of different lengths
2. vec1 **<-c**(1,3,5)
3. vec2 **<-c**(10,11,12,13,14,15)
5. #Taking these vectors as input to the array
6. res **<-** array(c(vec1,vec2),dim=c(3,3,2))
7. print(res)

**Output**

, , 1

[,1] [,2] [,3]

[1,] 1 10 13

[2,] 3 11 14

[3,] 5 12 15

, , 2

[,1] [,2] [,3]

[1,] 1 10 13

[2,] 3 11 14

[3,] 5 12 15

Naming rows and columns

In R, we can give the names to the rows, columns, and matrices of the array. This is done with the help of the dim name parameter of the array() function.

It is not necessary to give the name to the rows and columns. It is only used to differentiate the row and column for better understanding.

Below is an example, in which we create two arrays and giving names to the rows, columns, and matrices.

**Example**

1. #Creating two vectors of different lengths
2. vec1 **<-c**(1,3,5)
3. vec2 **<-c**(10,11,12,13,14,15)
5. #Initializing names for rows, columns and matrices
6. col\_names **<-** c("Col1","Col2","Col3")
7. row\_names **<-** c("Row1","Row2","Row3")
8. matrix\_names **<-** c("Matrix1","Matrix2")
10. #Taking the vectors as input to the array
11. res **<-** array(c(vec1,vec2),dim=c(3,3,2),dimnames=list(row\_names,col\_names,matrix\_names))
12. print(res)

**Output**

, , Matrix1

Col1 Col2 Col3

Row1 1 10 13

Row2 3 11 14

Row3 5 12 15

, , Matrix2

Col1 Col2 Col3

Row1 1 10 13

Row2 3 11 14

Row3 5 12 15

Accessing array elements

Like C or C++, we can access the elements of the array. The elements are accessed with the help of the index. Simply, we can access the elements of the array with the help of the indexing method. Let see an example to understand how we can access the elements of the array using the indexing method.

**Example**

1. , , Matrix1
2. Col1 Col2 Col3
3. Row1    1   10   13
4. Row2    3   11   14
5. Row3    5   12   15
7. , , Matrix2
8. Col1 Col2 Col3
9. Row1    1   10   13
10. Row2    3   11   14
11. Row3    5   12   15
13. Col1 Col2 Col3
14. 5   12   15
16. [1] 13
18. Col1 Col2 Col3
19. Row1    1   10   13
20. Row2    3   11   14
21. Row3    5   12   15

Manipulation of elements

The array is made up matrices in multiple dimensions so that the operations on elements of an array are carried out by accessing elements of the matrices.

**Example**

1. #Creating two vectors of different lengths
2. vec1 **<-c**(1,3,5)
3. vec2 **<-c**(10,11,12,13,14,15)
5. #Taking the vectors as input to the array1
6. res1 **<-** array(c(vec1,vec2),dim=c(3,3,2))
7. print(res1)
9. #Creating two vectors of different lengths
10. vec1 **<-c**(8,4,7)
11. vec2 **<-c**(16,73,48,46,36,73)
13. #Taking the vectors as input to the array2
14. res2 **<-** array(c(vec1,vec2),dim=c(3,3,2))
15. print(res2)
17. #Creating matrices from these arrays
18. mat1 **<-** res1[,,2]
19. mat2 **<-** res2[,,2]
20. res3 **<-** mat1+mat2
21. print(res3)

**Output**

, , 1

[,1] [,2] [,3]

[1,] 1 10 13

[2,] 3 11 14

[3,] 5 12 15

, , 2

[,1] [,2] [,3]

[1,] 1 10 13

[2,] 3 11 14

[3,] 5 12 15

, , 1

[,1] [,2] [,3]

[1,] 8 16 46

[2,] 4 73 36

[3,] 7 48 73

, , 2

[,1] [,2] [,3]

[1,] 8 16 46

[2,] 4 73 36

[3,] 7 48 73

[,1] [,2] [,3]

[1,] 9 26 59

[2,] 7 84 50

[3,] 12 60 88

Calculations across array elements

For calculation purpose, r provides **apply()** function. This apply function contains three parameters i.e., x, margin, and function.

This function takes the array on which we have to perform the calculations. The basic syntax of the apply() function is as follows:

1. apply(x, margin, fun)

Here, x is an array, and a margin is the name of the dataset which is used and fun is the function which is to be applied to the elements of the array.

**Example**

1. #Creating two vectors of different lengths
2. vec1 **<-c**(1,3,5)
3. vec2 **<-c**(10,11,12,13,14,15)
5. #Taking the vectors as input to the array1
6. res1 **<-** array(c(vec1,vec2),dim=c(3,3,2))
7. print(res1)
9. #using apply function
10. result **<-** apply(res1,c(1),sum)
11. print(result)

**Output**

, , 1

[,1] [,2] [,3]

[1,] 1 10 13

[2,] 3 11 14

[3,] 5 12 15

, , 2

[,1] [,2] [,3]

[1,] 1 10 13

[2,] 3 11 14

[3,] 5 12 15

[1] 48 56 64

R Matrix

In R, a two-dimensional rectangular data set is known as a matrix. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation.

In the R matrix, elements are arranged in a fixed number of rows and columns. The matrix elements are the real numbers. In R, we use matrix function, which can easily reproduce the memory representation of the matrix. In the R matrix, all the elements must share a common basic type.

Example

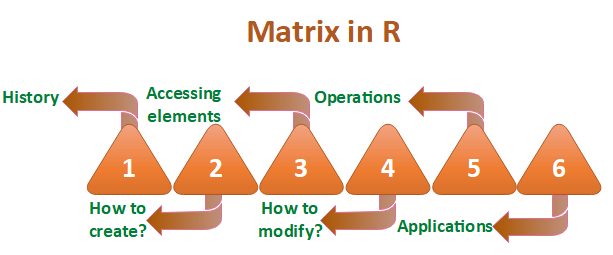
1. matrix1**<-matrix**(c(11, 13, 15, 12, 14, 16),nrow =2, ncol =3, byrow = TRUE)
2. matrix1

**Output**

[,1] [,2] [,3]

[1,] 11 13 15

[2,] 12 14 16



History of matrices in R

The word "Matrix" is the Latin word for womb which means a place where something is formed or produced. Two authors of historical importance have used the word "Matrix" for unusual ways. They proposed this axiom as a means to reduce any function to one of the lower types so that at the "bottom" (0order) the function is identical to its extension.

OOPs Concepts in Java

Any possible function other than a matrix from the matrix holds true with the help of the process of generalization. It will be true only when the proposition (which asserts function in question) is true. It will hold true for all or one of the value of argument only when the other argument is undetermined.

How to create a matrix in R?

Like vector and list, R provides a function which creates a matrix. R provides the matrix() function to create a matrix. This function plays an important role in data analysis. There is the following syntax of the matrix in R:

1. matrix(data, nrow, ncol, byrow, dim\_name)

**data**

The first argument in matrix function is data. It is the input vector which is the data elements of the matrix.

**nrow**

The second argument is the number of rows which we want to create in the matrix.

**ncol**

The third argument is the number of columns which we want to create in the matrix.

**byrow**

The byrow parameter is a logical clue. If its value is true, then the input vector elements are arranged by row.

**dim\_name**

The dim\_name parameter is the name assigned to the rows and columns.

Let's see an example to understand how matrix function is used to create a matrix and arrange the elements sequentially by row or column.

**Example**

1. #Arranging elements sequentially by row.
2. P **<-** matrix(c(5:16), nrow = 4, byrow = TRUE)
3. print(P)
5. # Arranging elements sequentially by column.
6. Q **<-** matrix(c(3:14), nrow = 4, byrow = FALSE)
7. print(Q)
9. # Defining the column and row names.
10. row\_names = c("row1", "row2", "row3", "row4")
11. ccol\_names = c("col1", "col2", "col3")
13. R **<-** matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(row\_names, col\_names))
14. print(R)

**Output**

[,1] [,2] [,3]

[1,] 5 6 7

[2,] 8 9 10

[3,] 11 12 13

[4,] 14 15 16

[,1] [,2] [,3]

[1,] 3 7 11

[2,] 4 8 12

[3,] 5 9 13

[4,] 6 10 14

col1 col2 col3

row1 3 4 5

row2 6 7 8

row3 9 10 11

row4 12 13 14

Accessing matrix elements in R

Like C and C++, we can easily access the elements of our matrix by using the index of the element. There are three ways to access the elements from the matrix.

1. We can access the element which presents on nth row and mth column.
2. We can access all the elements of the matrix which are present on the nth row.
3. We can also access all the elements of the matrix which are present on the mth column.

Let see an example to understand how elements are accessed from the matrix present on nth row mth column, nth row, or mth column.

**Example**

1. # Defining the column and row names.
2. row\_names = c("row1", "row2", "row3", "row4")
3. ccol\_names = c("col1", "col2", "col3")
4. #Creating matrix
5. R **<-** matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row\_names, col\_names))
6. print(R)
8. #Accessing element present on 3rd row and 2nd column
9. print(R[3,2])
11. #Accessing element present in 3rd row
12. print(R[3,])
14. #Accessing element present in 2nd column
15. print(R[,2])

**Output**

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 13

row4 14 15 16

[1] 12

col1 col2 col3

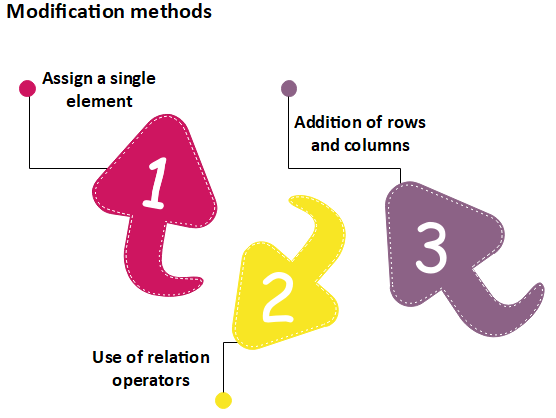
11 12 13

row1 row2 row3 row4

6 9 12 15

Modification of the matrix

R allows us to do modification in the matrix. There are several methods to do modification in the matrix, which are as follows:



Assign a single element

In matrix modification, the first method is to assign a single element to the matrix at a particular position. By assigning a new value to that position, the old value will get replaced with the new one. This modification technique is quite simple to perform matrix modification. The basic syntax for it is as follows:

1. matrix[n, m]**<-y**

Here, n and m are the rows and columns of the element, respectively. And, y is the value which we assign to modify our matrix.

Let see an example to understand how modification will be done:

**Example**

1. # Defining the column and row names.
2. row\_names = c("row1", "row2", "row3", "row4")
3. ccol\_names = c("col1", "col2", "col3")
5. R **<-** matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row\_names, col\_names))
6. print(R)
8. #Assigning value 20 to the element at 3d roe and 2nd column
9. R[3,2]**<-20**
10. print(R)

**Output**

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 13

row4 14 15 16

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 20 13

row4 14 15 16

Use of Relational Operator

R provides another way to perform matrix medication. In this method, we used some relational operators like >, <, ==. Like the first method, the second method is quite simple to use. Let see an example to understand how this method modifies the matrix.

**Example 1**

1. # Defining the column and row names.
2. row\_names = c("row1", "row2", "row3", "row4")
3. ccol\_names = c("col1", "col2", "col3")
5. R **<-** matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row\_names, col\_names))
6. print(R)
8. #Replacing element that equal to the 12
9. R[R==12]**<-0**
10. print(R)

**Output**

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 13

row4 14 15 16

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 0 13

row4 14 15 16

**Example 2**

1. # Defining the column and row names.
2. row\_names = c("row1", "row2", "row3", "row4")
3. ccol\_names = c("col1", "col2", "col3")
5. R **<-** matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row\_names, col\_names))
6. print(R)
8. #Replacing elements whose values are greater than 12
9. R[R**>**12]**<-0**
10. print(R)

**Output**

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 13

row4 14 15 16

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 0

row4 0 0 0

Addition of Rows and Columns

The third method of matrix modification is through the addition of rows and columns using the cbind() and rbind() function. The cbind() and rbind() function are used to add a column and a row respectively. Let see an example to understand the working of cbind() and rbind() functions.

**Example 1**

1. # Defining the column and row names.
2. row\_names = c("row1", "row2", "row3", "row4")
3. ccol\_names = c("col1", "col2", "col3")
5. R **<-** matrix(c(5:16), nrow = 4, byrow = TRUE, dimnames = list(row\_names, col\_names))
6. print(R)
8. #Adding row
9. rbind(R,c(17,18,19))
11. #Adding column
12. cbind(R,c(17,18,19,20))
14. #transpose of the matrix using the t() function:
15. t(R)
17. #Modifying the dimension of the matrix using the dim() function
18. dim(R)**<-c**(1,12)
19. print(R)

**Output**

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 13

row4 14 15 16

col1 col2 col3

row1 5 6 7

row2 8 9 10

row3 11 12 13

row4 14 15 16

17 18 19

col1 col2 col3

row1 5 6 7 17

row2 8 9 10 18

row3 11 12 13 19

row4 14 15 16 20

row1 row2 row3 row4

col1 5 8 11 14

col2 6 9 12 15

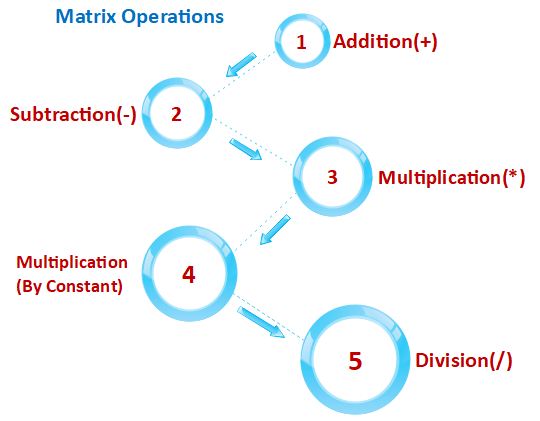
col3 7 10 13 16

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]

[1,] 5 8 11 14 6 9 12 15 7 10 13 16

Matrix operations

In R, we can perform the mathematical operations on a matrix such as addition, subtraction, multiplication, etc. For performing the mathematical operation on the matrix, it is required that both the matrix should have the same dimensions.



Let see an example to understand how mathematical operations are performed on the matrix.

**Example 1**

1. R **<-** matrix(c(5:16), nrow = 4,ncol=3)
2. S **<-** matrix(c(1:12), nrow = 4,ncol=3)
4. #Addition
5. sum**<-R**+S
6. print(sum)
8. #Subtraction
9. sub**<-R-S**
10. print(sub)
12. #Multiplication
13. mul**<-R**\*S
14. print(mul)
16. #Multiplication by constant
17. mul1**<-R**\*12
18. print(mul1)
20. #Division
21. div**<-R**/S
22. print(div)

**Output**

[,1] [,2] [,3]

[1,] 6 14 22

[2,] 8 16 24

[3,] 10 18 26

[4,] 12 20 28

[,1] [,2] [,3]

[1,] 4 4 4

[2,] 4 4 4

[3,] 4 4 4

[4,] 4 4 4

[,1] [,2] [,3]

[1,] 5 45 117

[2,] 12 60 140

[3,] 21 77 165

[4,] 32 96 192

[,1] [,2] [,3]

[1,] 60 108 156

[2,] 72 120 168

[3,] 84 132 180

[4,] 96 144 192

[,1] [,2] [,3]

[1,] 5.000000 1.800000 1.444444

[2,] 3.000000 1.666667 1.400000

[3,] 2.333333 1.571429 1.363636

[4,] 2.000000 1.500000 1.333333

Applications of matrix

1. In geology, Matrices takes surveys and plot graphs, statistics, and used to study in different fields.
2. Matrix is the representation method which helps in plotting common survey things.
3. In robotics and automation, Matrices have the topmost elements for the robot movements.
4. Matrices are mainly used in calculating the gross domestic products in Economics, and it also helps in calculating the capability of goods and products.
5. In computer-based application, matrices play a crucial role in the creation of realistic seeming motion.

R Data Frame

A data frame is a two-dimensional array-like structure or a table in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a special case of the list in which each component has equal length.

A data frame is used to store data table and the vectors which are present in the form of a list in a data frame, are of equal length.

In a simple way, it is a list of equal length vectors. A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.

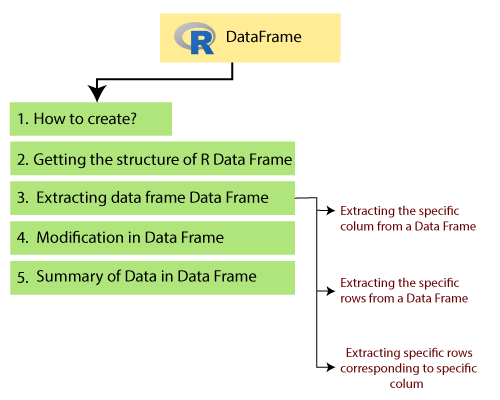
There are following characteristics of a data frame.

13M

305

C++ vs Java

* The columns name should be non-empty.
* The rows name should be unique.
* The data which is stored in a data frame can be a factor, numeric, or character type.
* Each column contains the same number of data items.



How to create Data Frame

In R, the data frames are created with the help of frame() function of data. This function contains the vectors of any type such as numeric, character, or integer. In below example, we create a data frame that contains employee id (integer vector), employee name(character vector), salary(numeric vector), and starting date(Date vector).

**Example**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,915.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. # Printing the data frame.
12. print(emp.data)

**Output**

employee\_idemployee\_namesalstarting\_date

1 1 Shubham623.30 2012-01-01

2 2 Arpita915.20 2013-09-23

3 3 Nishka611.00 2014-11-15

4 4 Gunjan729.00 2014-05-11

5 5 Sumit843.25 2015-03-27

Getting the structure of R Data Frame

In R, we can find the structure of our data frame. R provides an in-build function called str() which returns the data with its complete structure. In below example, we have created a frame using a vector of different data type and extracted the structure of it.

**Example**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. # Printing the structure of data frame.
12. str(emp.data)

**Output**

'data.frame': 5 obs. of 4 variables:

$ employee\_id : int 1 2 3 4 5

$ employee\_name: chr "Shubham" "Arpita" "Nishka" "Gunjan" ...

$ sal : num 623 515 611 729 843

$ starting\_date: Date, format: "2012-01-01" "2013-09-23" ...

Extracting data from Data Frame

The data of the data frame is very crucial for us. To manipulate the data of the data frame, it is essential to extract it from the data frame. We can extract the data in three ways which are as follows:

1. We can extract the specific columns from a data frame using the column name.
2. We can extract the specific rows also from a data frame.
3. We can extract the specific rows corresponding to specific columns.

Let's see an example of each one to understand how data is extracted from the data frame with the help these ways.

Extracting the specific columns from a data frame

**Example**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name= c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. # Extracting specific columns from a data frame
12. final **<-** data.frame(emp.data$employee\_id,emp.data$sal)
13. print(final)

**Output**

emp.data.employee\_idemp.data.sal

1 1 623.30

2 2 515.20

3 3 611.00

4 4 729.00

5 5 843.25

Extracting the specific rows from a data frame

**Example**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. # Extracting first row from a data frame
12. final **<-** emp.data[1,]
13. print(final)

16. # Extracting last two row from a data frame
17. final **<-** emp.data[4:5,]
18. print(final)

**Output**

employee\_id employee\_name sal starting\_date

1 1 Shubham 623.3 2012-01-01

employee\_id employee\_name sal starting\_date

4 4 Gunjan 729.00 2014-05-11

5 5 Sumit 843.25 2015-03-27

Extracting specific rows corresponding to specific columns

**Example**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. # Extracting 2nd and 3rd row corresponding to the 1st and 4th column
12. final **<-** emp.data[c(2,3),c(1,4)]
13. print(final)

**Output**

employee\_id starting\_date

2 2 2013-09-23

3 3 2014-11-15

Modification in Data Frame

R allows us to do modification in our data frame. Like matrices modification, we can modify our data frame through re-assignment. We cannot only add rows and columns, but also we can delete them. The data frame is expanded by adding rows and columns.

We can

1. Add a column by adding a column vector with the help of a new column name using cbind() function.
2. Add rows by adding new rows in the same structure as the existing data frame and using rbind() function
3. Delete the columns by assigning a NULL value to them.
4. Delete the rows by re-assignment to them.

Let's see an example to understand how rbind() function works and how the modification is done in our data frame.

**Example: Adding rows and columns**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. print(emp.data)
13. #Adding row in the data frame
14. x **<-** list(6,"Vaishali",547,"2015-09-01")
15. rbind(emp.data,x)
17. #Adding column in the data frame
18. y **<-** c("Moradabad","Lucknow","Etah","Sambhal","Khurja")
19. cbind(emp.data,Address=y)

**Output**

employee\_id employee\_name sal starting\_date

1 1 Shubham 623.30 2012-01-01

2 2 Arpita 515.20 2013-09-23

3 3 Nishka 611.00 2014-11-15

4 4 Gunjan 729.00 2014-05-11

5 5 Sumit 843.25 2015-03-27

employee\_id employee\_name sal starting\_date

1 1 Shubham 623.30 2012-01-01

2 2 Arpita 515.20 2013-09-23

3 3 Nishka 611.00 2014-11-15

4 4 Gunjan 729.00 2014-05-11

5 5 Sumit 843.25 2015-03-27

6 6 Vaishali 547.00 2015-09-01

employee\_id employee\_name sal starting\_date Address

1 1 Shubham 623.30 2012-01-01 Moradabad

2 2 Arpita 515.20 2013-09-23 Lucknow

3 3 Nishka 611.00 2014-11-15 Etah

4 4 Gunjan 729.00 2014-05-11 Sambhal

5 5 Sumit 843.25 2015-03-27 Khurja

**Example: Delete rows and columns**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. print(emp.data)
13. #Delete rows from data frame
14. emp.data**<-emp.data**[-1,]
15. print(emp.data)
17. #Delete column from the data frame
18. emp.data$starting\_date**<-NULL**
19. print(emp.data)

**Output**

employee\_idemployee\_namesalstarting\_date

1 1 Shubham623.30 2012-01-01

2 2 Arpita515.20 2013-09-23

3 3 Nishka611.00 2014-11-15

4 4 Gunjan729.00 2014-05-11

5 5 Sumit843.25 2015-03-27

employee\_idemployee\_namesalstarting\_date

2 2 Arpita515.20 2013-09-23

3 3 Nishka611.00 2014-11-15

4 4 Gunjan729.00 2014-05-11

5 5 Sumit843.25 2015-03-27

employee\_idemployee\_namesal

1 1 Shubham623.30

2 2 Arpita515.20

3 3 Nishka611.00

4 4 Gunjan729.00

5 5 Sumit843.25

Summary of data in Data Frames

In some cases, it is required to find the statistical summary and nature of the data in the data frame. R provides the summary() function to extract the statistical summary and nature of the data. This function takes the data frame as a parameter and returns the statistical information of the data. Let?s see an example to understand how this function is used in R:

**Example**

1. # Creating the data frame.
2. emp.data**<-** data.frame(
3. employee\_id = c (1:5),
4. employee\_name = c("Shubham","Arpita","Nishka","Gunjan","Sumit"),
5. sal = c(623.3,515.2,611.0,729.0,843.25),
7. starting\_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
8. "2015-03-27")),
9. stringsAsFactors = FALSE
10. )
11. print(emp.data)
13. #Printing the summary
14. print(summary(emp.data))

**Output**

employee\_idemployee\_namesalstarting\_date

1 1 Shubham623.30 2012-01-01

2 2 Arpita515.20 2013-09-23

3 3 Nishka611.00 2014-11-15

4 4 Gunjan729.00 2014-05-11

5 5 Sumit843.25 2015-03-27

employee\_idemployee\_namesalstarting\_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

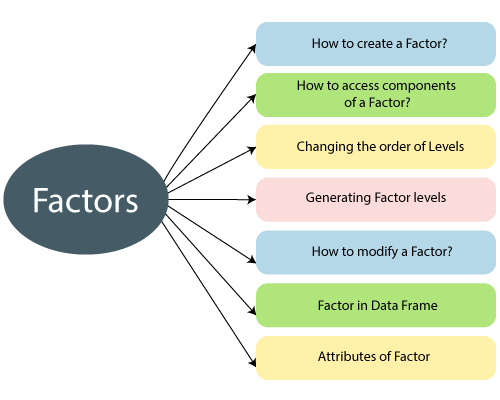
Mean :3 Mean :664.4 Mean :2014-01-14

3rd Qu.:4 3rd Qu.:729.0 3rd Qu.:2014-11-15

Max. :5 Max. :843.2 Max. :2015-03-27

R factors

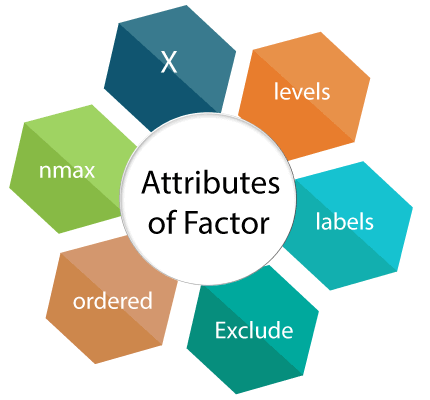
The factor is a data structure which is used for fields which take only predefined finite number of values. These are the variable which takes a limited number of different values. These are the data objects which are used to categorize the data and to store it on multiple levels. It can store both integers and strings values, and are useful in the column that has a limited number of unique values.



Factors have labels which are associated with the unique integers stored in it. It contains predefined set value known as levels and by default R always sorts levels in alphabetical order.

Attributes of a factor

There are the following attributes of a factor in R



1. **X**  
   It is the input vector which is to be transformed into a factor.
2. **levels**  
   It is an input vector that represents a set of unique values which are taken by x.
3. **labels**  
   It is a character vector which corresponds to the number of labels.
4. **Exclude**  
   It is used to specify the value which we want to be excluded,
5. **ordered**  
   It is a logical attribute which determines if the levels are ordered.
6. **nmax**  
   It is used to specify the upper bound for the maximum number of level.

How to create a factor?

In R, it is quite simple to create a factor. A factor is created in two steps

10M

185

Prime Ministers of India | List of Prime Minister of India (1947-2020)

1. In the first step, we create a vector.
2. Next step is to convert the vector into a factor,

R provides factor() function to convert the vector into factor. There is the following syntax of factor() function

1. factor\_data**<-** factor(vector)

Let's see an example to understand how factor function is used.

**Example**

1. # Creating a vector as input.
2. data **<-** c("Shubham","Nishka","Arpita","Nishka","Shubham","Sumit","Nishka","Shubham","Sumit","Arpita","Sumit")
4. print(data)
5. print(is.factor(data))
7. # Applying the factor function.
8. factor\_data**<-** factor(data)
10. print(factor\_data)
11. print(is.factor(factor\_data))

**Output**

[1] "Shubham" "Nishka" "Arpita" "Nishka" "Shubham" "Sumit" "Nishka"

[8] "Shubham" "Sumit" "Arpita" "Sumit"

[1] FALSE

[1] Shubham Nishka Arpita Nishka Shubham Sumit Nishka Shubham Sumit

[10] Arpita Sumit

Levels: Arpita Nishka Shubham Sumit

[1] TRUE

Accessing components of factor

Like vectors, we can access the components of factors. The process of accessing components of factor is much more similar to the vectors. We can access the element with the help of the indexing method or using logical vectors. Let's see an example in which we understand the different-different ways of accessing the components.

**Example**

1. # Creating a vector as input.
2. data **<-** c("Shubham","Nishka","Arpita","Nishka","Shubham","Sumit","Nishka","Shubham","Sumit","Arpita","Sumit")
4. # Applying the factor function.
5. factor\_data**<-** factor(data)
7. #Printing all elements of factor
8. print(factor\_data)
10. #Accessing 4th element of factor
11. print(factor\_data[4])
13. #Accessing 5th and 7th element
14. print(factor\_data[c(5,7)])
16. #Accessing all elemcent except 4th one
17. print(factor\_data[-4])
19. #Accessing elements using logical vector
20. print(factor\_data[c(TRUE,FALSE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE)])

**Output**

[1] Shubham Nishka Arpita Nishka Shubham Sumit Nishka Shubham Sumit

[10] Arpita Sumit

Levels: Arpita Nishka Shubham Sumit

[1] Nishka

Levels: Arpita Nishka Shubham Sumit

[1] Shubham Nishka

Levels: Arpita Nishka Shubham Sumit

[1] Shubham Nishka Arpita Shubham Sumit Nishka Shubham Sumit Arpita

[10] Sumit

Levels: Arpita Nishka Shubham Sumit

[1] Shubham Shubham Sumit Nishka Sumit

Levels: Arpita Nishka Shubham Sumit

Modification of factor

Like data frames, R allows us to modify the factor. We can modify the value of a factor by simply re-assigning it. In R, we cannot choose values outside of its predefined levels means we cannot insert value if it's level is not present on it. For this purpose, we have to create a level of that value, and then we can add it to our factor.

Let's see an example to understand how the modification is done in factors.

**Example**

1. # Creating a vector as input.
2. data **<-** c("Shubham","Nishka","Arpita","Nishka","Shubham")
4. # Applying the factor function.
5. factor\_data**<-** factor(data)
7. #Printing all elements of factor
8. print(factor\_data)
10. #Change 4th element of factor with sumit
11. factor\_data[4] **<-**"Arpita"
12. print(factor\_data)
14. #change 4th element of factor with "Gunjan"
15. factor\_data[4] **<-** "Gunjan"    # cannot assign values outside levels
16. print(factor\_data)
18. #Adding the value to the level
19. levels(factor\_data) **<-** c(levels(factor\_data),"Gunjan")#Adding new level
20. factor\_data[4] **<-** "Gunjan"
21. print(factor\_data)

**Output**

[1] Shubham Nishka Arpita Nishka Shubham

Levels: Arpita Nishka Shubham

[1] Shubham Nishka Arpita Arpita Shubham

Levels: Arpita Nishka Shubham

Warning message:

In `[<-.factor`(`\*tmp\*`, 4, value = "Gunjan") :

invalid factor level, NA generated

[1] Shubham Nishka Arpita Shubham

Levels: Arpita Nishka Shubham

[1] Shubham Nishka Arpita Gunjan Shubham

Levels: Arpita Nishka Shubham Gunjan

Factor in Data Frame

When we create a frame with a column of text data, R treats this text column as categorical data and creates factor on it.

**Example**

# Creating the vectors for data frame.

1. height **<-** c(132,162,152,166,139,147,122)
2. weight **<-** c(40,49,48,40,67,52,53)
3. gender **<-** c("male","male","female","female","male","female","male")
5. # Creating the data frame.
6. input\_data**<-** data.frame(height,weight,gender)
7. print(input\_data)
9. # Testing if the gender column is a factor.
10. print(is.factor(input\_data$gender))
12. # Printing the gender column to see the levels.
13. print(input\_data$gender)

**Output**

height weight gender

1 132 40 male

2 162 49 male

3 152 48 female

4 166 40 female

5 139 67 male

6 147 52 female

7 122 53 male

[1] TRUE

[1] male male female female male female male

Levels: female male

Changing order of the levels

In R, we can change the order of the levels in the factor with the help of the factor function.

**Example**

1. data **<-** c("Nishka","Gunjan","Shubham","Arpita","Arpita","Sumit","Gunjan","Shubham")
2. # Creating the factors
3. factor\_data**<-** factor(data)
4. print(factor\_data)
6. # Apply the factor function with the required order of the level.
7. new\_order\_factor**<-** factor(factor\_data,levels = c("Gunjan","Nishka","Arpita","Shubham","Sumit"))
8. print(new\_order\_factor)

**Output**

[1] Nishka Gunjan Shubham Arpita Arpita Sumit Gunjan Shubham

Levels: Arpita Gunjan Nishka Shubham Sumit

[1] Nishka Gunjan Shubham Arpita Arpita Sumit Gunjan Shubham

Levels: Gunjan Nishka Arpita Shubham Sumit

Generating Factor Levels

R provides gl() function to generate factor levels. This function takes three arguments i.e., n, k, and labels. Here, n and k are the integers which indicate how many levels we want and how many times each level is required.

There is the following syntax of gl() function which is as follows

1. gl(n, k, labels)
2. n indicates the number of levels.
3. k indicates the number of replications.
4. labels is a vector of labels for the resulting factor levels.

**Example**

1. gen\_factor**<-** gl(3,5,labels=c("BCA","MCA","B.Tech"))
2. gen\_factor

**Output**

[1] BCA BCA BCA BCA BCA MCA MCA MCA MCA MCA

[11] B.Tech B.Tech B.Tech B.Tech B.Tech

Levels: BCA MCA B.Tech