# BIG DATA ANALYTICS USING R

## B.A / B.Com (Hons) THIRD YEAR

## SEMESTER – V

### Lesson Writers


**Dr. U. Surya Kameswari**
**M.Sc CS, M.Tech IT., Ph.D**
Assistant Professor
Department of Computer Science and Engineering University College of sciences
Acharya Nagarjuna University

**Dr. B. Reddaiah, M.E., Ph.D.**
Associate Professor
Department of Computer Science and Technology
Yogi Vemana University
Kadapa-516005

**Mr. G V Suresh, M.Tech., (Ph.D)**
Associate professor
Department of Computer Science and Engineering Lakireddy Balireddy College of Engineering (Autonomous)
Mylavaram

**Mrs. A. Sarvani, M.Tech., (Ph.D)**
Associate professor
Department of Information Technology
Lakireddy Balireddy College of Engineering (Autonomous)
Mylavaram


### Editor


**Dr. K. Lavanya B.E., M.Tech, Ph.D**
Assistant professor
Department of Computer Science and Engineering
University College of sciences
Acharya Nagarjuna University
Email: dr.lavanyakampa@gmail.com


### Director


**Prof.V.VENKATESWARLU**
**M.A.(Soc), M.S.W., M.Phil., Ph.D.**
**CENTRE FOR DISTANCE EDUCATION**
**ACHARYA NAGARJUNA UNIVERSITY**
**NAGARJUNA NAGAR-522510**
**website: anucde.info**
**e-mail: anucdedirector@gmail.com**

B.Com (CA): Big Data Analytics Using R

First Edition: 2024

No. of Copies



Published by

**Prof.V.Venkateswarlu**

Director

Centre for Distance Education

Acharya Nagarjuna University

Nagarjuna Nagar-522510

Printed at

## *FOREWORD*

***Since its establishment in 1976, Acharya Nagarjuna University has been forging ahead in the path of progress and dynamism, offering a variety of courses and research contributions. I am extremely happy that by gaining 'A' grade from the NAAC in the year 2016, Acharya Nagarjuna University is offering educational opportunities at the UG, PG levels apart from research degrees to students from over 443 affiliated colleges spread over the two districts of Guntur and Prakasam.***

***The University has also started the Centre for Distance Education in 2003-04 with the aim of taking higher education to the door step of all the sectors of the society. The centre will be a great help to those who cannot join in colleges, those who cannot afford the exorbitant fees as regular students, and even to housewives desirous of pursuing higher studies. Acharya Nagarjuna University has started offering B.A., and B.Com courses at the Degree level and M.A., M.Com., M.Sc., M.B.A., and L.L.M., courses at the PG level from the academic year 2003-2004 onwards.***

***To facilitate easier understanding by students studying through the distance mode, these self-instruction materials have been prepared by eminent and experienced teachers. The lessons have been drafted with great care and expertise in the stipulated time by these teachers. Constructive ideas and scholarly suggestions are welcome from students and teachers involved respectively. Such ideas will be incorporated for the greater efficacy of this distance mode of education. For clarification of doubts and feedback, weekly classes and contact classes will be arranged at the UG and PG levels respectively.***

***It is my aim that students getting higher education through the Centre for Distance Education should improve their qualification, have better employment opportunities and in turn be part of country's progress. It is my fond desire that in the years to come, the Centre for Distance Education will go from strength to strength in the form of new courses and by catering to larger number of people. My congratulations to all the Directors, Academic Coordinators, Editors and Lesson-writers of the Centre who have helped in these endeavours.***

**Prof. K.Gangadhar Rao**
**Vice-Chancellor**
**Acharya Nagarjuna University**

# Course-6A: Big Data Analytics Using R ----Lab (Practical)

**LAB Exercise 1:**

# Download and install R-Programming environment

**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* 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.

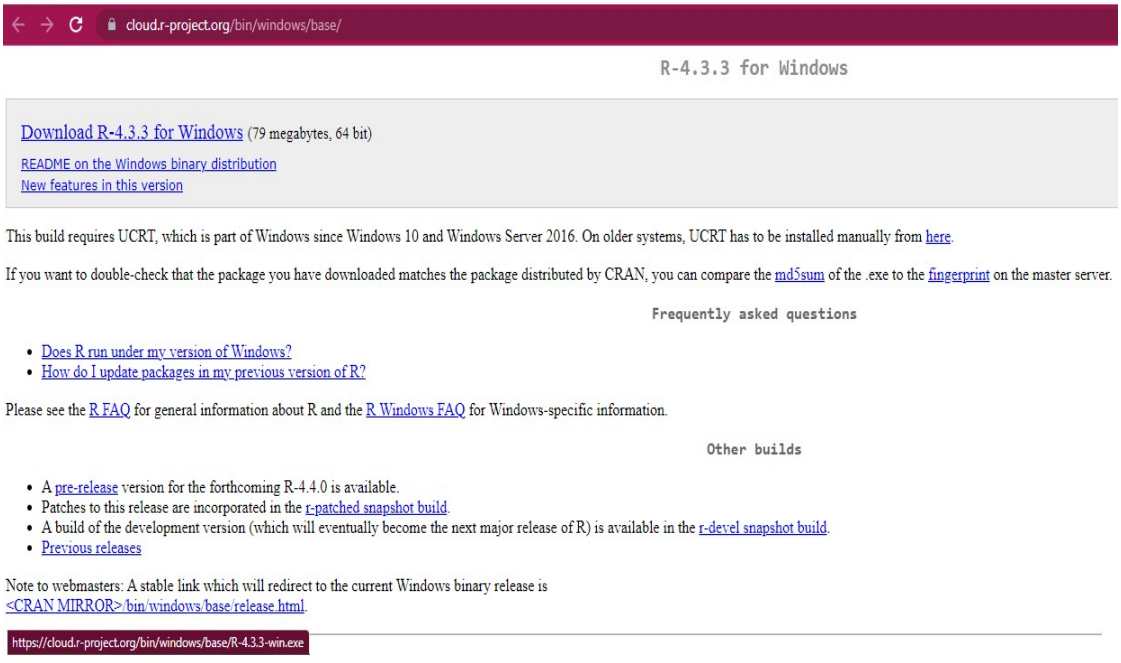
To install R, either we can get it from the site *https://cloud.r-project.org* or can use commands from the terminal.

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

## 1.1 Installation of R for Windows Setup

### Step 1:

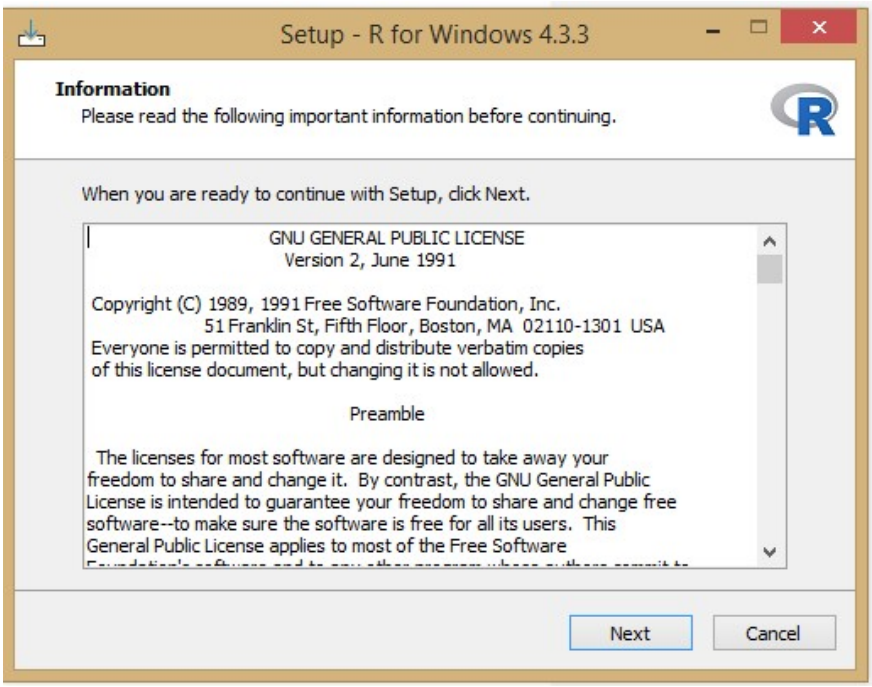
First, we have to download the R setup from *https://cloud.r-project.org/bin/windows/base/*.



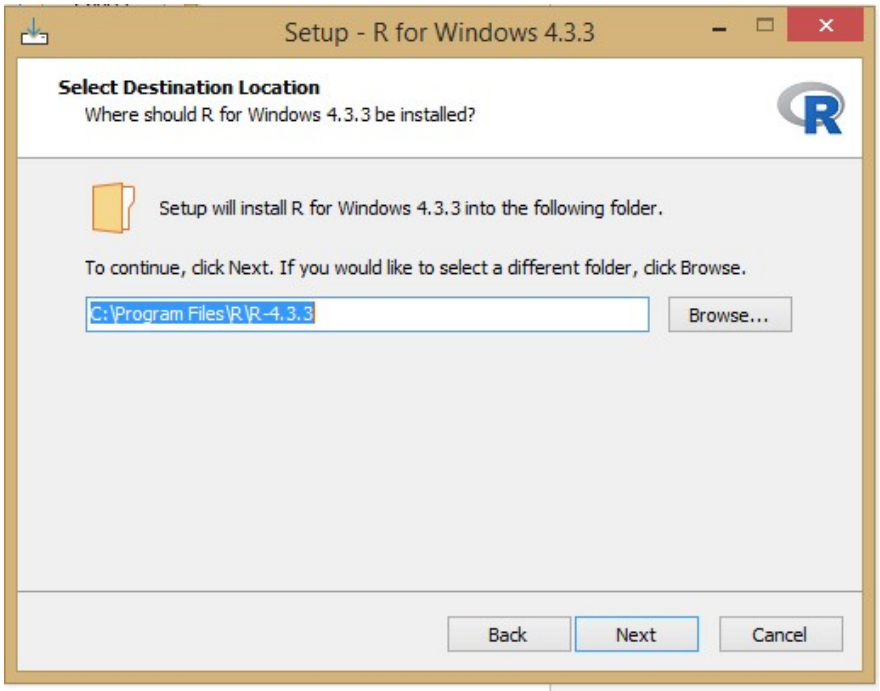
**Step 2:**

When we click on **Download R 4.3.3 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:

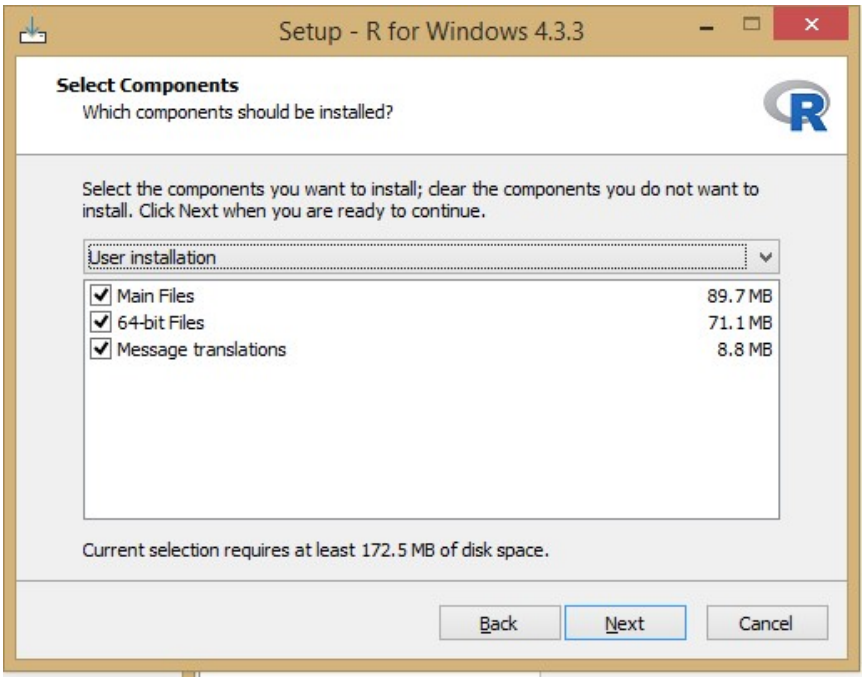
1) Select the path where we want to download the R and proceed to Next.



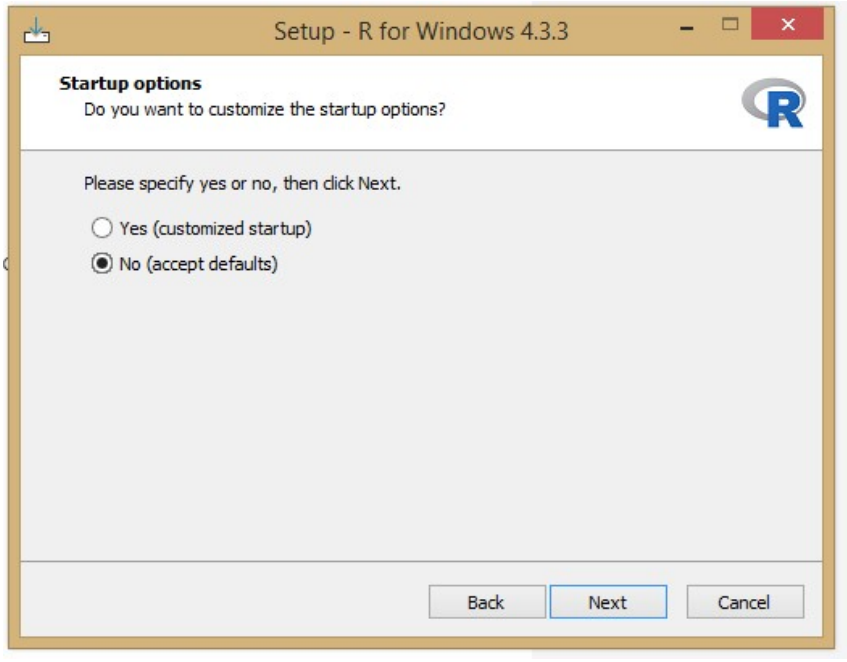

Click on *Next*

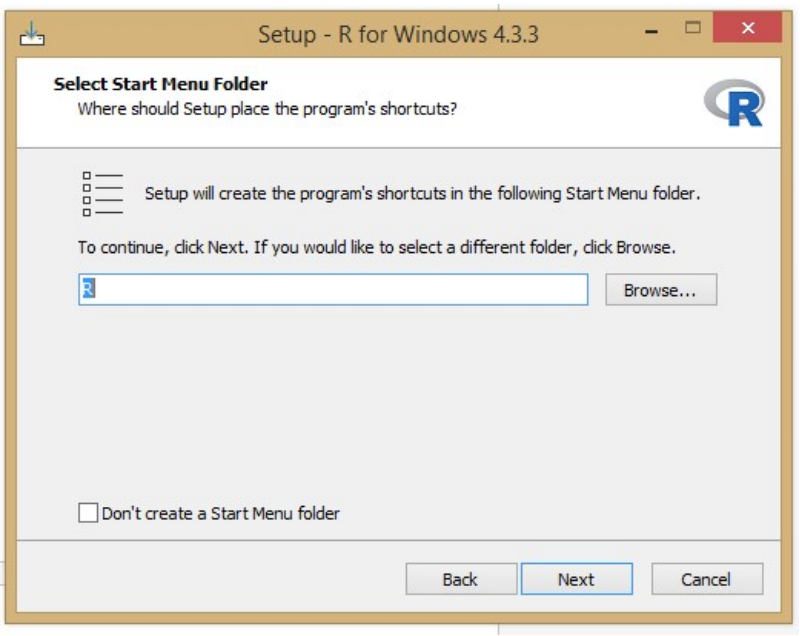


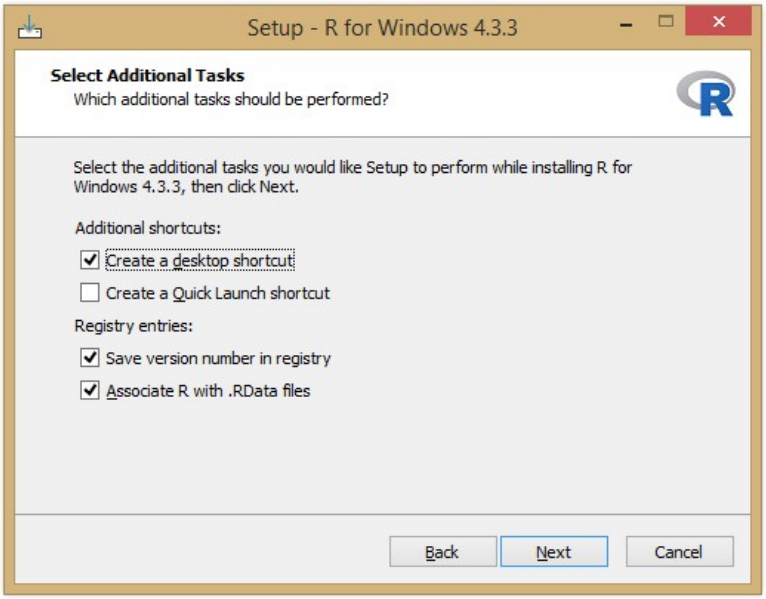
Select all components which we want to install, and then we will proceed to *Next.*



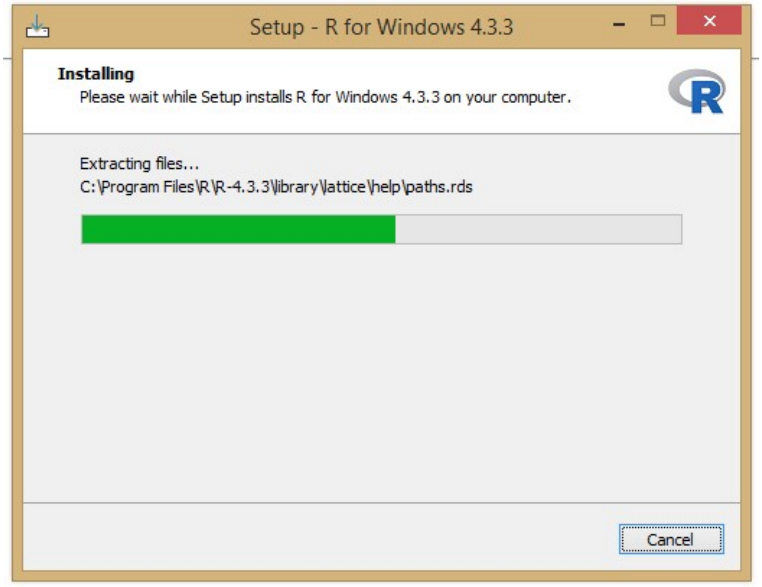

In the next step, we have to select either customized startup or accept the default, and then we proceed to *Next.*



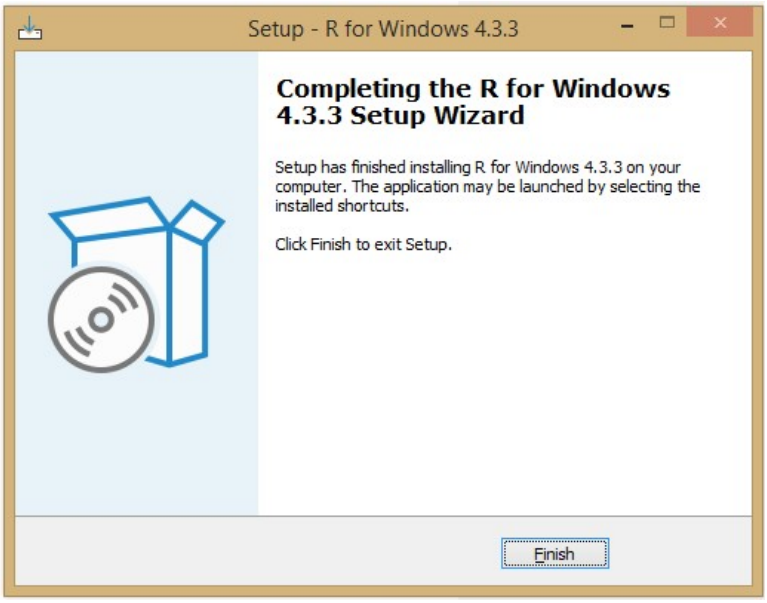



When we proceed to next, our installation of R in our system will get started:




In the last, we will click on *Finish* to successfully install R in our system.




### 1.2 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.

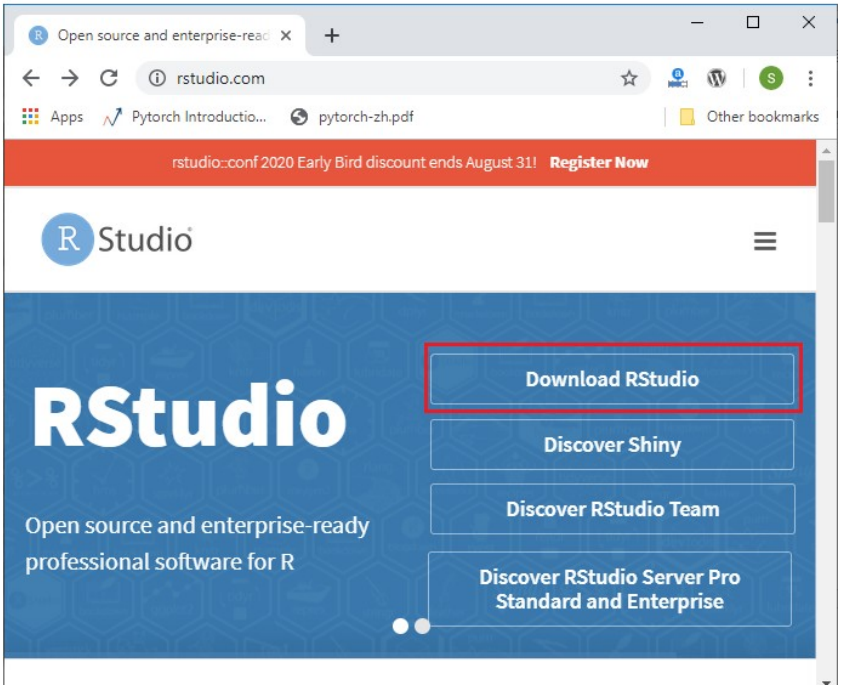
**Installation of RStudio**

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.

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:

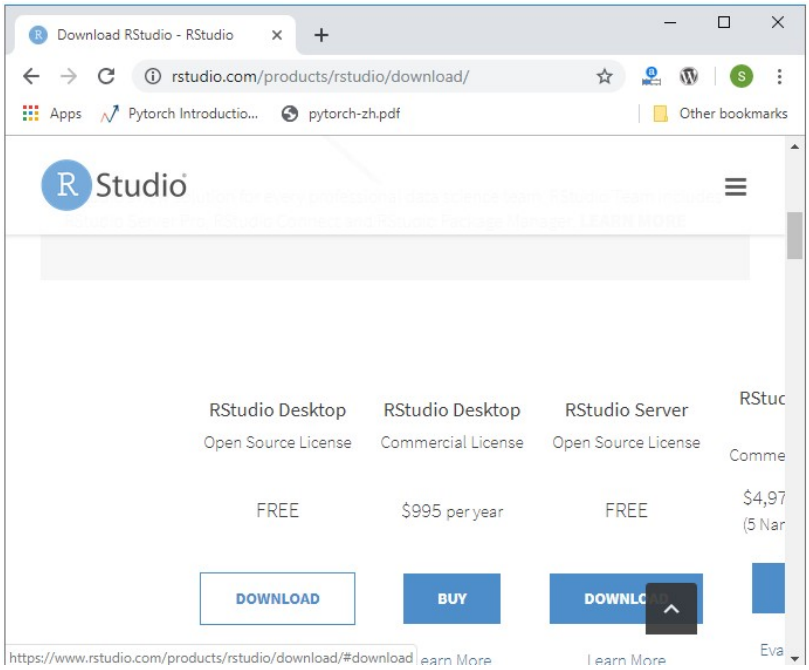
**Step 1:**

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



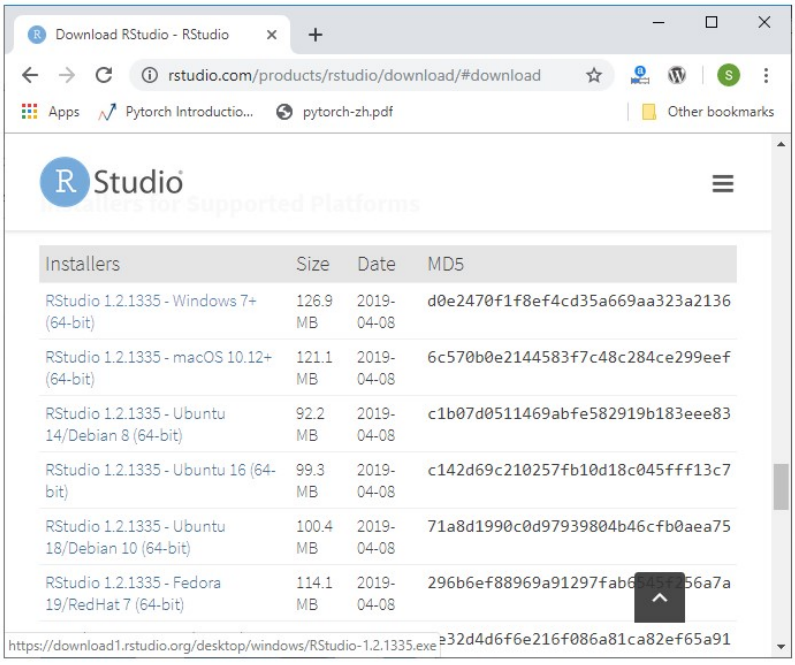

**Step 2:**

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




**Step 3:**

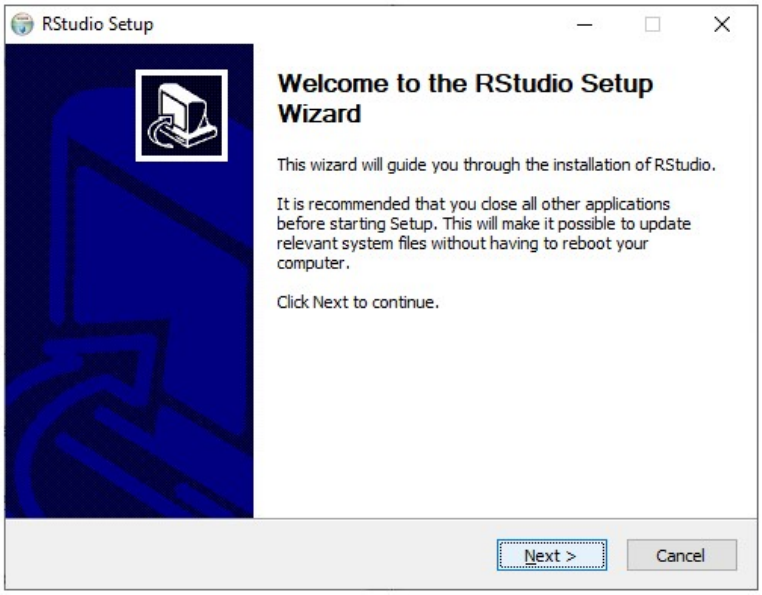
In the next step, we will select the appropriate installer. When we select the installer, our downloading of RStudion setup will start.



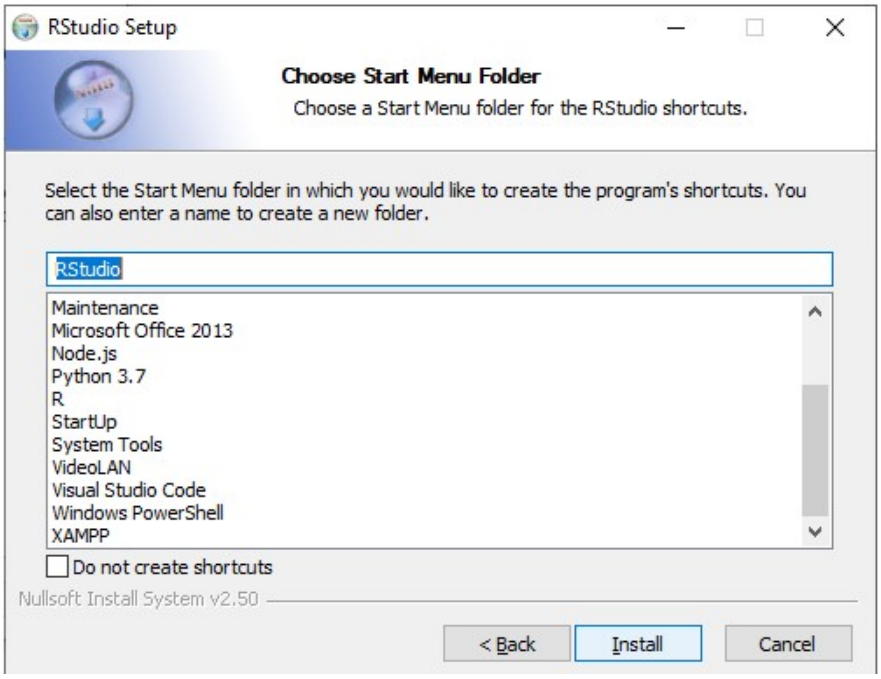

**Step 4:**

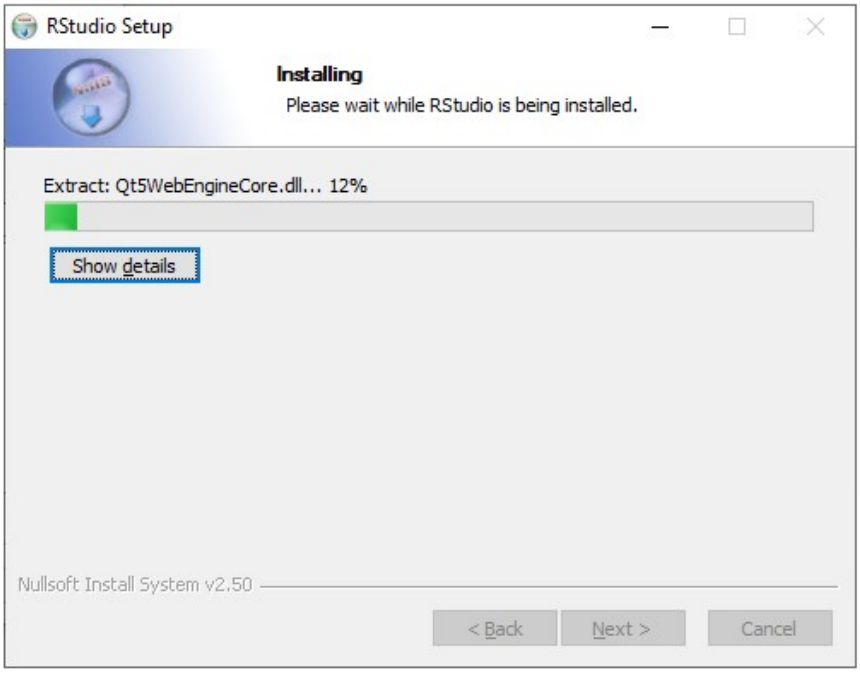
In the next step, we will run our setup in the following way:

1) Click on Next.

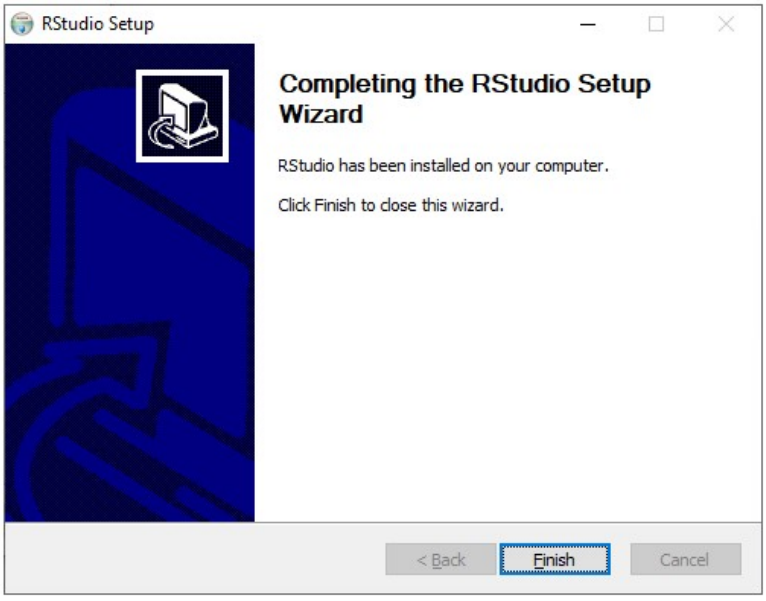



2) Click on Install.



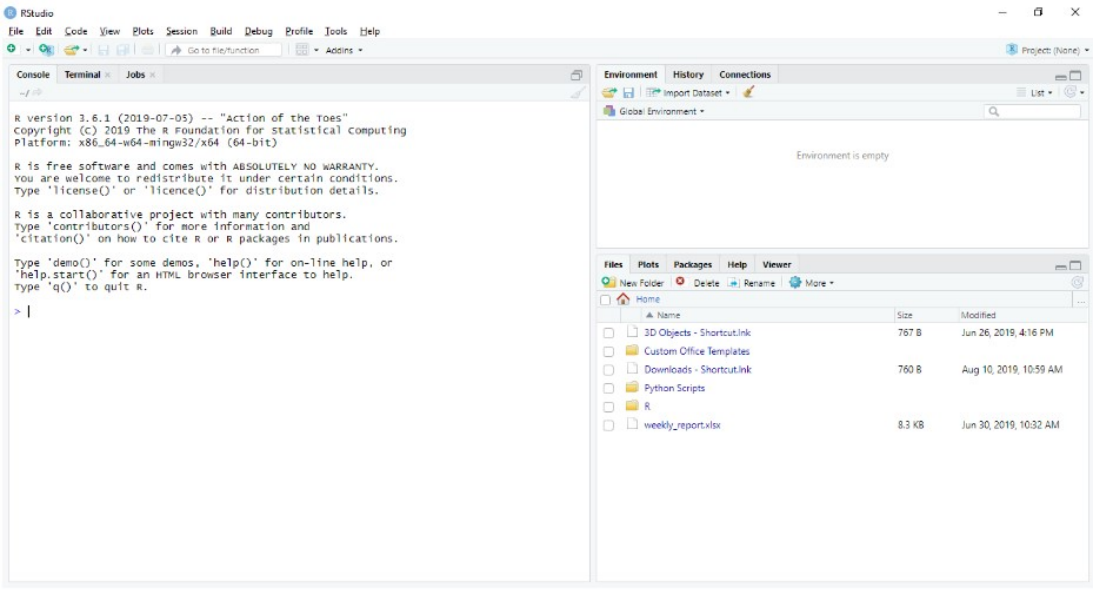



3) Click on finish.




4) RStudio is ready to work.

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*



**LAB EXERCISE 2:**

## Create a vector in R and perform operations on it.

**PROGRAM DESCRIPTION**

Vectors are the most basic R data objects and there are six types of atomic vectors. They are logical,integer, double, complex, character and raw.

**SOURCE CODE**

```
# Creating a vector

vect<- c(2, 4, 6, 8, 10)


# Displaying the original vector

print("Original Vector:")
print(vect)


# Accessing elements in the vector

print("Accessing elements in the vector:")
print("First element:", vect[1])
print("Third element:", vect[3])


# Modifying an element in the vector

print("Modifying an element in the vector:")
vect[2] <- 12
print(vect)


# Adding new elements to the vector

print("Adding new elements to the vector:")
vect<- c(vect, 14, 16)
print(vect)


# Removing elements from the vectorprint("Removing elements from the vector:")
```

```
vect<- vect[-3]
print(vect)


# Performing operations on the vectorprint("Performing operations on the vector:")

sum_result<- sum(vect)
mean_result<- mean(vect)
print(paste('Sum:',sum_result))
print(paste("Mean:",mean_result))
```

**OUTPUT**

```
[1] "Original Vector:"
[1]  2  4  6  8 10
[1] "Accessing elements in the vector:"
[1] "First element:"
[1] "Third element:"
[1] "Modifying an element in the vector:"
[1]  2 12  6  8 10
[1] "Adding new elements to the vector:"
[1]  2 12  6  8 10 14 16
[1]  2 12  8 10 14 16
[1] "Sum: 62"
[1] "Mean: 10.3333333333333"
```

**LAB EXERCISE 3:**

## Create integer, complex, logical, character data type objects in R and print their values and their class using print and class functions.

---

**PROGRAM DESCRIPTION**

Atomic data types are the object types which you can create (atomic) vectors with them.

There are several functions that can show you the data type of an R object, such as type of, mode, storage. Mode, class and str.

If we want to print the R data type, we recommend using the type of function.

There are other functions that allow you to check if some object belongs to some data type, returning TRUE or FALSE. As a rule, these functions start with is followed by the data type.

**SOURCE CODE**

```
# creating objects of different data types
integer_object<- 15L
complex_object<- 8 + 2i
logical_object<- TRUE
character_object<- "Acharya Nagarjuna University"

# Printing values and classes

print(integer_object)
print(class(integer_object))

print(complex_object)
print(class(complex_object))

print(logical_object)
```

```
print(class(logical_object))

print(character_object)
print(class(character_object))
```

**OUTPUT**

```
[1] 15
[1] "integer"
[1] 8+2i
[1] "complex"
[1] TRUE
[1] "logical"
[1] " Acharya Nagarjuna University "
[1] "character"
```

**LAB EXERCISE 4:**

## Write code in R to demonstrate sum(), min(), max() and seq() functions

**PROGRAM DESCRIPTION**

**sum():** This function calculates the sum of all the elements in a numeric vector. It's useful for obtaining the total of a set of values, such as sales figures, test scores, or any other numeric data.

**min():** The min() function returns the smallest value in a vector. It's handy for finding the minimum value in a dataset, which could represent, for example, the lowest temperature in a week or the cheapest product price in a list.

**max():** Conversely, max() returns the largest value in a vector. It's useful for identifying the maximum value within a dataset, such as the highest temperature in a week or the most expensive item in a list.

**seq():** This function generates a sequence of numbers according to specified parameters. It's commonly used to create sequences of integers, either incrementing or decrementing, which can be useful for generating indices, iterating over loops, or creating numeric ranges for analysis or visualization.

**SOURCE CODE**

```
# Create a numeric vector
numeric_vector<- c(3, 7, 1, 9, 4, 6)

# Calculate the sum of elements in the vector
total<- sum(numeric_vector)
print(paste("Sum of elements:", total))

# Find the minimum value in the vector
minimum<- min(numeric_vector)
```

```
print(paste("Minimum value:", minimum))

# Find the maximum value in the vector
maximum<- max(numeric_vector)
print(paste("Maximum value:", maximum))

# Generate a sequence of numbers from 1 to 10
sequence<- seq(1, 10)
print(paste("Sequence from 1 to 10:", sequence))
```

**OUTPUT**

```
[1] "Sum of elements: 30"
[1] "Minimum value: 1"
[1] "Maximum value: 9"
[1] "Sequence from 1 to 10: 1"  "Sequence from 1 to 10: 2"
[3] "Sequence from 1 to 10: 3"  "Sequence from 1 to 10: 4"
[5] "Sequence from 1 to 10: 5"  "Sequence from 1 to 10: 6"
[7] "Sequence from 1 to 10: 7"  "Sequence from 1 to 10: 8"
[9] "Sequence from 1 to 10: 9"  "Sequence from 1 to 10: 10"
```

**LAB EXERCISE 5:**

## Write code in R to manipulate text in R using grep(), toupper(), tolower() and substr() functions.

---

**PROGRAM DESCRIPTION**

**grep()** : is used to search for a pattern within a character vector or a file. It returns the indices of the elements in the vector that contain the specified pattern.

**toupper():** converts all the characters in a character vector to uppercase. It's useful when you want to standardize the case of text data for consistency or comparison purposes.

**tolower()** : converts all the characters in a character vector to lowercase. It's helpful for standardizing text data to a uniform case.

**substr():** extracts substrings from elements in a character vector. It takes arguments specifying the vector, the starting position, and optionally, the number of characters to extract. It's useful for extracting specific portions of text, such as extracting a portion of a string based on position.

**SOURCE CODE**

```
# Sample text
text<- "Acharya Nagarjuna University is the one of the best University in India which is a state university established in 1976"

# grep(): Finding patterns in text
pattern<- "University"
matching_indices<- grep(pattern, text)
print(paste("Indices of 'University' in text:", matching_indices))
```

```
# toupper(): Converting text to uppercase
uppercase_text<- toupper(text)
print(paste("Uppercase text:", uppercase_text))

# tolower(): Converting text to lowercase
lowercase_text<- tolower(text)
print(paste("Lowercase text:", lowercase_text))

# substr(): Extracting substrings
substring<- substr(text, start = 1, stop = 12)
print(paste("Substring:", substring))
```

**OUTPUT**

```
[1] "Indices of 'University' in text: 1"
[1] "Uppercase text: ACHARYA NAGARJUNA UNIVERSITY IS THE ONE OF THE BEST UNIVERSITY IN INDIA WHICH IS A STATE UNIVERSITY ESTABLISHED IN 1976"
[1] "Lowercase text: acharyanagarjuna university is the one of the best university in india which is a state university established in 1976"
[1] "Substring: Acharya Naga"
```

**LAB EXERCISE 6:**

# Create data frame in R and perform operations on it.

---

**PROGRAM DESCRIPTION**

A data frame is a fundamental data structure in R used for storing and manipulating structured data. It allows for the representation and analysis of tabular data, where each column can have a different data type.

Common operations on data frames include creating, accessing, modifying, and analyzing data. This involves tasks such as selecting specific columns or rows, adding new columns, filtering data based on conditions, sorting, merging, and performing summary statistics. These operations enable users to manage and extract meaningful insights from their data efficiently.

**SOURCE CODE:**

```
# Create a data frame
students<- data.frame(
  Name = c("Srinivas", "Surya", "Shourya", "Rohit", "Arya"),
  Age = c(22,24,20,22,21),
  Grade = c("A", "C", "B", "A", "B")
)

# Print the data frame

print("Original Data Frame:")
print(students)

# Accessing specific columns

print("Names of the students:")
print(students$Name)
```

```
# Adding a new column

students$Gender<- c("Female", "Male", "Male", "Male", "Female")
print("Data Frame with Gender:")
print(students)

# Filtering rows based on condition

print("Students with Grade A:")
print(subset(students, Grade == "A"))

# Sorting the data frame by age

print("Data Frame sorted by Age:")
print(students[order(students$Age), ])

# Calculating summary statistics

print("Summary Statistics:")
print(summary(students$Age))
```

**OUTPUT:**

```
[1] "Original Data Frame:"
     Name Age Grade
1 Srinivas  22     A
2    Surya  24     C
3  Shourya  20     B
4    Rohit  22     A
5     Arya  21     B

[1] "Names of the students:"

[1] "Srinivas" "Surya"    "Shourya"  "Rohit"    "Arya"
```

```
[1] "Data Frame with Gender:"
  Name Age Grade Gender
1 Srinivas  22     A Female
2    Surya  24     C   Male
3  Shourya  20     B   Male
4    Rohit  22     A   Male
5     Arya  21     B Female

[1] "Students with Grade A:"
  Name Age Grade Gender
1 Srinivas  22     A Female
4    Rohit  22     A   Male

[1] "Data Frame sorted by Age:"
  Name Age Grade Gender
3  Shourya  20     B   Male
5     Arya  21     B Female
1 Srinivas  22     A Female
4    Rohit  22     A   Male
2    Surya  24     C   Male

[1] "Summary Statistics:"
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   20.0    21.0    22.0    21.8    22.0    24.0
```

**LAB EXERCISE 7:**

# Import data into R from text and excel files using read.table () and read.csv () functions.

---

**PROGRAM DESCRIPTION**

Upon the termination of a program, all data is lost. Our data will remain intact even if the program terminates if it is saved to a file. If we are required to submit a substantial quantity of data, the process will consume many hours. But in the event that we possess a file encompassing the entirety of the data, we can effortlessly retrieve its contents by executing a few commands in R. It is simple and error-free to transfer data from one computer to another. In order to enable the storage of those assets in a multitude of formats. The data could potentially be stored in a tabular format (e.g., comma-separated values) or a text file (i.e., txt), csv, or on the cloud or the internet.

**read.table():** it is a general function that can be used to read a file in table format. The data will be imported as a data frame.

Syntax: *read.table(file, header = FALSE, sep = "", dec = ".")*

**read.csv()** is used for reading "comma separated value" files (".csv"). In this also the data will be imported as a data frame.

Syntax: *read.csv(file, header = TRUE, sep = ",", dec = ".", ...)*

**SOURCE CODE: Importing data from a text file using read.table():**

```
# Import data from a text file
Text_data<- read.table("StudentsPerformance.txt", header = TRUE, sep = "\t")
head(text_data)
```

**OUTPUT:**

| gender | race.ethnicity | parental.level.of.education | lunch | test.preparation.course | math.score | reading.score | writing.score |
|---|---|---|---|---|---|---|---|
| female | group B | bachelor's degree | standard | none | 72 | 72 | 74 |
| female | group C | some college | standard | completed | 69 | 90 | 88 |
| female | group B | master's degree | standard | none | 90 | 95 | 93 |
| male | group A | associate's degree | free/reduced | none | 47 | 57 | 44 |
| male | group C | some college | standard | none | 76 | 78 | 75 |
| female | group B | associate's degree | standard | none | 71 | 83 | 78 |

**SOURCE CODE: Importing data from a CSV file using read.csv():**

```
# Import data from a CSV file
csv_data<- read.csv("StudentsPerformance.csv")
head(csv_data)
```

**OUTPUT:**

| gender | race.ethnicity | parental.level.of.education | lunch | test.preparation.course | math.score | reading.score | writing.score |
|---|---|---|---|---|---|---|---|
| female | group B | bachelor's degree | standard | none | 72 | 72 | 74 |
| female | group C | some college | standard | completed | 69 | 90 | 88 |
| female | group B | master's degree | standard | none | 90 | 95 | 93 |
| male | group A | associate's degree | free/reduced | none | 47 | 57 | 44 |
| male | group C | some college | standard | none | 76 | 78 | 75 |
| female | group B | associate's degree | standard | none | 71 | 83 | 78 |

**LAB EXERCISE 8:**

## Write code in R to find out whether number is prime or not.

**PROGRAM DESCRIPTION**

The code defines a function is_prime() that takes a number num as input and returns TRUE if the number is prime, and FALSE otherwise.

The function checks whether the input number is less than or equal to 1, in which case it returns FALSE, as numbers less than or equal to 1 are not prime.

It checks if the number is 2, which is a prime number, and returns TRUE if it is.

If the number is even and greater than 2, it returns FALSE, as even numbers greater than 2 are not prime.

It then iterates through odd numbers from 3 up to the square root of the input number (sqrt(num)), checking if the number is divisible by any of them. If it is, it returns FALSE.

If the number is not divisible by any number between 3 and the square root of the input number, it returns TRUE, indicating that the number is prime.

The code then tests the function with a specific number (num = 17 in this case) and prints whether the number is prime or not.

**SOURCE CODE**

```
# Function to check if a number is prime

is_prime<- function(num) {
if (num<= 1) {
return(FALSE)  # Numbers less than or equal to 1 are not prime
  }
if (num == 2) {
return(TRUE)   # 2 is a prime number
```

```
  }
if (num %% 2 == 0) {
return(FALSE)  # Even numbers greater than 2 are not prime
  }
for (i in 3:sqrt(num)) {
if (num %% i == 0) {
return(FALSE)  # If num is divisible by any number between 3 and sqrt(num), it's not prime
    }
  }
return(TRUE)  # If num is not divisible by any number between 3 and sqrt(num), it's prime
}
# Test the function

num<- as.integer(readline(prompt = "Enter a number: "))
if (is_prime(num)) {
print(paste(num, "is a prime number"))
} else {
print(paste(num, "is not a prime number"))
}
```

**OUTPUT:**

Test case 1:

Enter a number: 26

It is not a prime number

Test case 2:

Enter a number: 17

17 is a prime number

**LAB EXERCISE 9:**

## Print numbers from 1 to 100 using while loop and for loop in R.

**PROGRAM DESCRIPTION**

Using WHILE loop:

- Initialize a variable*num* to store the starting number, which is 1.
- Use a while loop to iterate as long as *num* is less than or equal to 100.
- Within each iteration, print the current value of *num*.
- Increment*num* by 1 at the end of each iteration to move to the next number.

Using FOR loop:

- Use a *for* loop to iterate over a sequence of numbers from 1 to 100 (1:100).
- In each iteration, the variable *num* takes on the next value from the sequence.
- Within each iteration, print the current value of *num*

**SOURCE CODE : Using a while loop to print numbers from 1 to 100**

```
i<- 1  # Initialize counter
while (i<= 100) {
  #print(i)
cat(i, '\t')
i<- i + 1  # Increment counter
}
```

**OUTPUT:**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 |
| 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 |
| 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | | | | |

**SOURCE CODE :Using a for loop to print numbers from 1 to 100**

```
for (i in 1:100) {
cat(i, '\t')
}
```

**OUTPUT:**

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 |
| 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 | 71 | 72 |
| 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 | 81 | 82 | 83 | 84 |
| 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 | 93 | 94 | 95 | 96 |
| 97 | 98 | 99 | 100 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 |
| 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 |
| 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| 69 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |
| 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 90 | 91 | 92 |
| 93 | 94 | 95 | 96 | 97 | 98 | 99 | 100 | | | | |

**LAB EXERCISE 10:**

## Write a program to import data from csv file and print the data on the console.

---

**PROGRAM DESCRIPTION**

A CSV (Comma-Separated Values) file is a plain text file format commonly used for storing tabular data. In a CSV file, each line represents a row of data, and the values within each row are separated by commas (or other delimiters like tabs or semicolons). The first row often contains column headers, providing names for each column in the dataset.

To create a CSV (Comma-Separated Values) file, you can use a text editor or spreadsheet software.

The read.sv()function is used to read data from a CSV (Comma-Separated Values) file and create a data frame. It is part of the base R package and is commonly used for importing tabular data stored in CSV format.

**SOURCE CODE**

# Read data from a CSV file named "StudentPerformance.csv" in the current directory

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

# View the structure of the data frame

```
str(df)
```

```
'data.frame':       1000 obs. of  8 variables:
 $ gender                    : Factor w/ 2 levels "female","male": 1 1 1 2 2 1 1 2 2 1 ...
 $ race.ethnicity            : Factor w/ 5 levels "group A","group B",..: 2 3 2 1 3 2 2 2 4 2 ...
 $ parental.level.of.education: Factor w/ 6 levels "associate's degree",..: 2 5 4 1 5 1 5 5 3 3 ...
x $ lunch                     : Factor w/ 2 levels "free/reduced",..: 2 2 2 1 2 2 2 1 1 1 ...
```

```
$ test.preparation.course    : Factor w/ 2 levels "completed","none": 2 1 2 2 2 2 1 2 1 2 ...
$ math.score                 : int  72 69 90 47 76 71 88 40 64 38 ...
$ reading.score              : int  72 90 95 57 78 83 95 43 64 60 ...
$ writing.score              : int  74 88 93 44 75 78 92 39 67 50 .
```

# View the first few rows of the data frame

```
head(df)
```

| gender | race.ethnicity | parental.level.of.education | lunch | test.preparation.course | math.score | reading.score | writing.score |
|---|---|---|---|---|---|---|---|
| female | group B | bachelor's degree | standard | none | 72 | 72 | 74 |
| female | group C | some college | standard | completed | 69 | 90 | 88 |
| female | group B | master's degree | standard | none | 90 | 95 | 93 |
| male | group A | associate's degree | free/reduced | none | 47 | 57 | 44 |
| male | group C | some college | standard | none | 76 | 78 | 75 |
| female | group B | associate's degree | standard | none | 71 | 83 | 78 |

**LAB EXERCISE 3:**

## Write a program to demonstrate histogram in R.

---

**PROGRAM DESCRIPTION**

Histograms can be created using the hist() function in R programming language. This function takes in a vector of values for which the histogram is plotted.

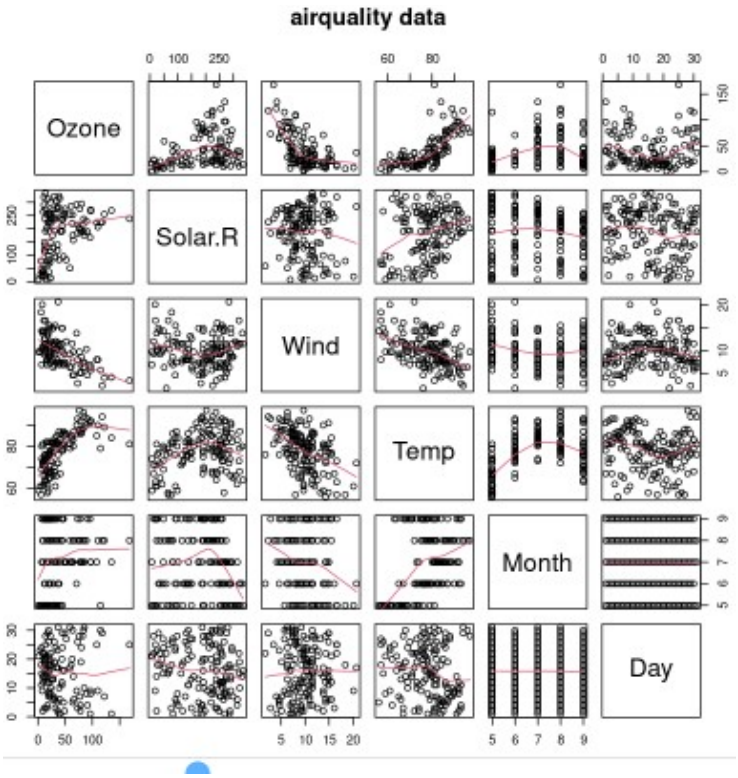
For this exercise we use the built-in dataset*airquality* which has Daily readings of the following air quality values for May 1, 1973 (a Tuesday) to September 30, 1973.
Ozone: Mean ozone in parts per billion from 1300 to 1500 hours at Roosevelt Island
Solar.R: Solar radiation in Langleys in the frequency band 4000--7700 Angstroms from 0800 to 1200 hours at Central Park
Wind: Average wind speed in miles per hour at 0700 and 1000 hours at LaGuardia Airport
Temp: Maximum daily temperature in degrees Fahrenheit at La Guardia Airport.

**Dataset Link:**

https://www.rdocumentation.org/packages/datasets/versions/3.6.2/topics/airquality

**SOURCE CODE**

```
if(!require('datasets')) {
install.packages('datasets')
library('datasets')
}

require(graphics)
pairs(airquality, panel = panel.smooth, main = "airquality data")
```

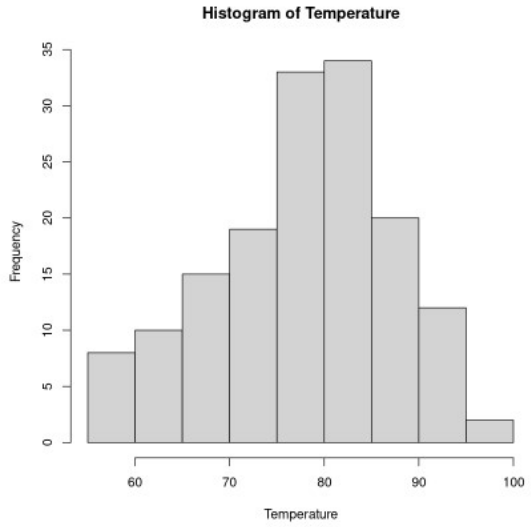



```
str(airquality)


OUTPUT:
'data.frame':       153 obs. of  6 variables:
 $ Ozone  :int  41 36 12 18 NA 28 23 19 8 NA ...
 $ Solar.R: int  190 118 149 313 NA NA 299 99 19 194 ...
 $ Wind   : num  7.4 8 12.6 11.5 14.3 14.9 8.6 13.8 20.1 8.6 ...
 $ Temp   : int  67 72 74 62 56 66 65 59 61 69 ...
 $ Month  :int  5 5 5 5 5 5 5 5 5 5 ...
 $ Day    : int  1 2 3 4 5 6 7 8 9 10 ...
```

We will use the temperature parameter which has 154 observations in degrees Fahrenheit.

**Simple histogram**

```
Temperature <- airquality$Temp
hist(Temperature)
```



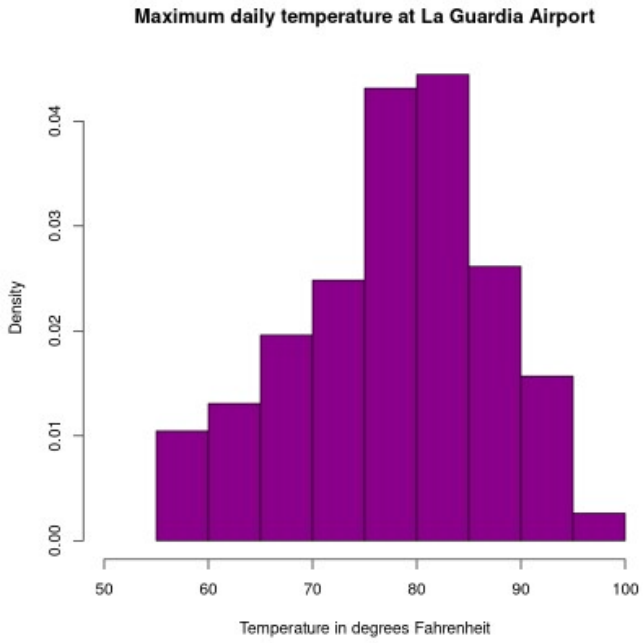

We can see above that there are 9 cells with equally spaced breaks. In this case, the height of a cell is equal to the number of observations falling in that cell.

We can pass in additional parameters to control the way our plot looks.Some of the frequently used ones are, main to give the title, xlab and ylab to provide labels for the axes, xlim and ylim to provide range of the axes, col to define color etc. Additionally, with the argument freq=FALSE we can get the probability distribution instead of the frequency.

**Histogram with added parameters**

```
hist(Temperature,
main="Maximum daily temperature at La Guardia Airport",
xlab="Temperature in degrees Fahrenheit",
xlim=c(50,100),
col="darkmagenta",
```

```
freq=FALSE
)
```



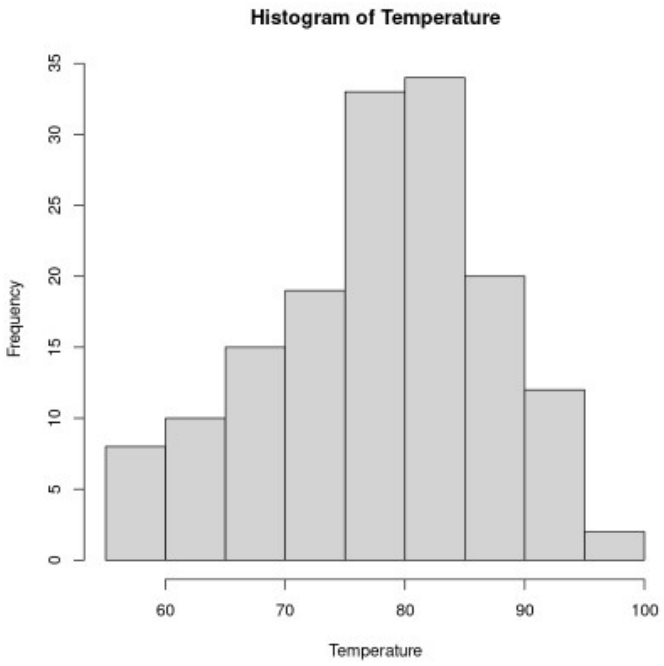

We can observe that the y axis is labeled density instead of frequency. In this case, the total area of the histogram is equal to 1.

**Return Value of hist()**

The hist() function returns a list with 6 components.

```
# create a histogram of the "Temperature" variable
h <- hist(Temperature)


# print the histogram object
print(h)


OUTPUT:
$breaks
 [1]  55  60  65  70  75  80  85  90  95 100
```

```
$counts
[1]  8 10 15 19 33 34 20 12  2

$density
[1] 0.010457516 0.013071895 0.019607843 0.024836601 0.043137255 0.044444444
[7] 0.026143791 0.015686275 0.002614379

$mids
[1] 57.5 62.5 67.5 72.5 77.5 82.5 87.5 92.5 97.5

$xname
[1] "Temperature"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"
```



We can observe that an object of class histogram is returned which has:

*breaks*- places where the breaks occur,

*counts*- the number of observations falling in that cell,

*density*- the density of cells, mids-the midpoints of cells,

*xname*- the x argument name and

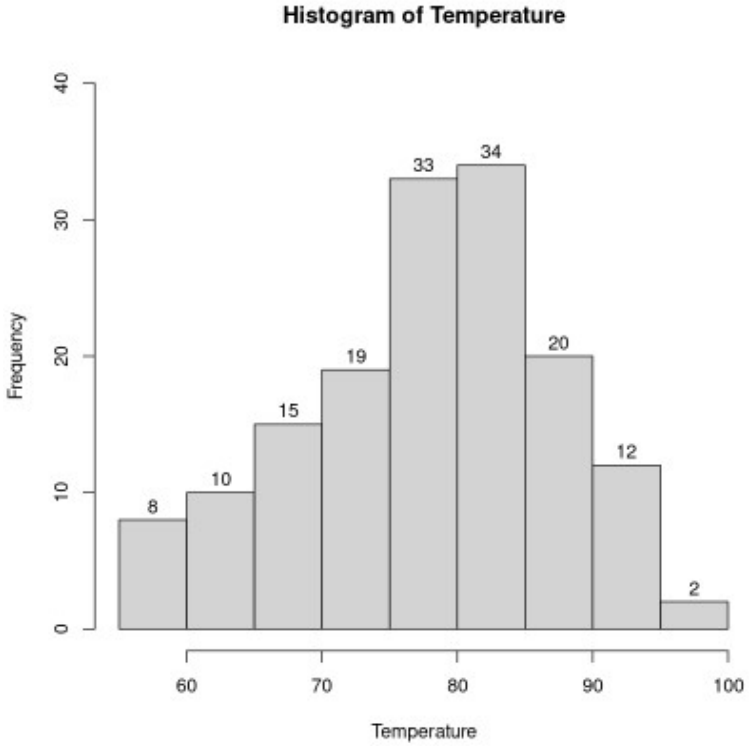
*equidist*- a logical value indicating if the breaks are equally spaced or not.

We can use these values for further processing.

For example, in the following example we use the return values to place the counts on top of each cell using the text() function.

**Use Histogram return values for labels using text()**

```
h <- hist(Temperature,ylim=c(0,40))
text(h$mids,h$counts,labels=h$counts, adj=c(0.5, -0.5))
```

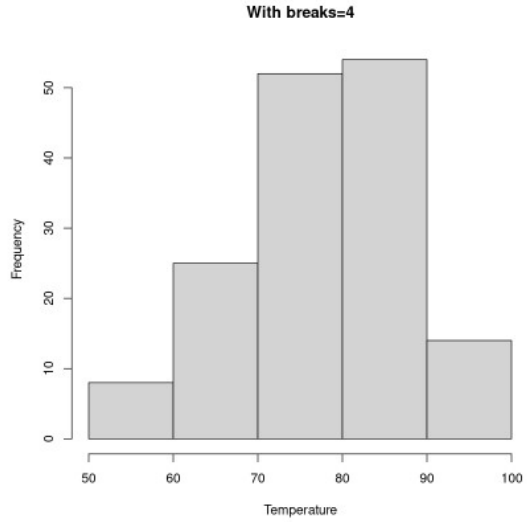


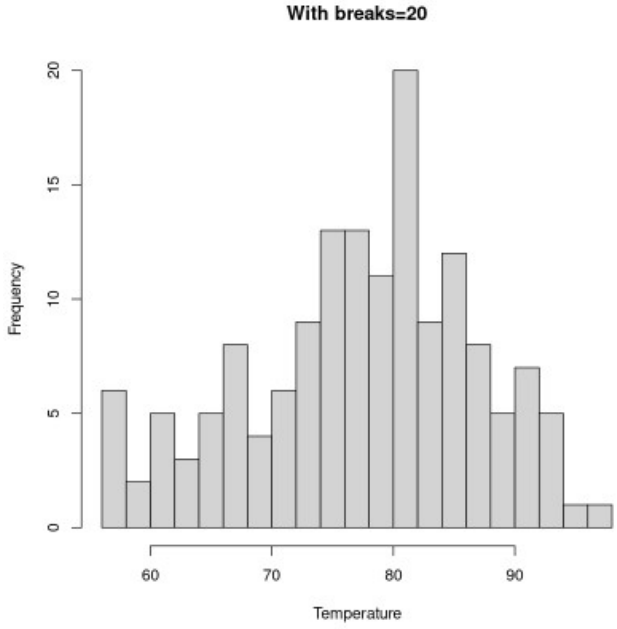
**Defining the Number of Breaks**

With the breaks argument we can specify the number of cells we want in the histogram. However, this number is just a suggestion.

R calculates the best number of cells, keeping this suggestion in mind. Following are two histograms on the same data with different numbers of cells.

```
hist(Temperature, breaks=4, main="With breaks=4")
hist(Temperature, breaks=20, main="With breaks=20")
```



In the above figure we see that the actual number of cells plotted is greater than we had specified.

We can also define breakpoints between the cells as a vector. This makes it possible to plot a histogram with unequal intervals. In such a case, the area of the cell is proportional to the number of observations falling inside that cell.

**Histogram with non-uniform width**

```
hist(Temperature,
main="Maximum daily temperature at La Guardia Airport",
xlab="Temperature in degrees Fahrenheit",
xlim=c(50,100),
col="chocolate",
border="brown",
breaks=c(55,60,70,75,80,100)
)
```