ADITYA

COLLEGE OF ENGINEERING

Aditya Nagar, ADB Road, Surampalem - 533437

Department of

Name : Roll No. : | | | | | | | | | | |

Certified that this is the bonafide record of practical work done by

Mr. /Ms. .....

a student of .....with PIN No. .....

in the ..... Laboratory during the year .....

No. of Practicals Conducted : ☐ No. of Practicals Attended : ☐

Signature - Faculty Incharge **Signature - Head of the Department**

Submitted for the Practical examinationheld on .....

EXAMINER - 1 **EXAMINER - 2**

VISION & MISSION OF THE INSTITUTE

VISION

To induce higher planes of learning by imparting technical education with

- International standards
- Applied research
- Creative Ability
- Value based instruction and to emerge as a premiere institute.

MISSION

Achieving academic excellence by providing globally acceptable technical education by forecasting technology through

- Innovative Research and development
- Industry Institute Interaction
- Empowered Manpower

VISION & MISSION OF THE DEPARTMENT

VISION

To be a recognized Computer Science and Engineering hub striving to meet the growing needs of the Industry and Society.

MISSION

M1: Imparting Quality Education through state-of-the-art infrastructure with industry Collaboration

M2: Enhance Teaching Learning Process to disseminate knowledge.

M3: Organize Skill based, Industrial and Societal Events for overall Development.

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Pointer

S No	Date	Name of the Experiment	Page No	Remark

Experiment-1:

Installing R and RStudio
Basic functionality of R, variable, data types in R

Install R on Windows:

Step-1: Go to CRAN R Project Website.
Step-2: Click on the Download R for Windows Link.
Step-3: Click on the base Subdirectory link (or) install R for the first time link. Step-4: Click Download R.X.X.X for Windows (Stand for the latest version eg.3.8.5) and Save the Executable file .EXE File
Step5: Run the EXE file and Follow the Installation Instructions.
5(a): Select the desired language and then click next. 5(b): Read the License Agreement and then click next.
5(c): Select the components you wish to install (it's Recommended to install all components) and click next.
5(d): Enter / Browse the Folder / path you wish to install R into and confirm by clicking next.
5(e): Select additional tasks to line creating desktop shortcuts etc.., then click next.
5(f): Wait for the installation process until completed.
5(g): Click finish to complete Installation.

Install R Studio on Windows:

Step-1: With R-base installed, Let's Move on to the installing R studio to begin, go to download RStudio and click on the download button for R Studio Desktop.
Step-2: Click on the link for the windows version of R Studio and Save .EXE File.
Step-3: Run the .EXE and Follow the Installation Instructions
=> Click next on the welcome Window
=> Enter / Browse the path to the installation folder and click next to proceed
=> Select the folder for start menu shortcuts (or) Click on do not create shortcut and then click next.
=> Wait for Installation process until to completed
=> Click finish to end the installation.

Basic Functionality of R:

What is R?

R is a popular programming language used for statistical computing and graphical presentation. Its most common use is to analyze and visualize data.

Why Use R?

- It is a great resource for data analysis, data visualization, data science and machine learning
- It provides many statistical techniques (such as statistical tests, classification, clustering and data reduction)
- It is easy to draw graphs in R, like pie charts, histograms, box plot, scatter plot, etc++
- It works on different platforms (Windows, Mac, Linux)
- It is open-source and free
- It has a large community support
- It has many packages (libraries of functions) that can be used to solve different problems

VARIABLES IN R:

Creating Variables in R:

Variables are containers for storing data values.

R does not have a command for declaring a variable. A variable is created the moment you first assign a value to it. To assign a value to a variable, use the <- sign. To output (or print) the variable value, just type the variable name:

EXAMPLE:

```
Name= readline(prompt= "Enter Name:")
Branch= readline(prompt= "Enter Branch:")
College= readline(prompt= "Enter college:")
print(paste("My Name is",Name,"i am studying in",Branch,
        "Engineering in",College,"College"))
print(R.version.string)
```

OUTPUT:

```
Enter Name: Rani
Enter Age:19
Enter Gender: female
Enter City: surampalem
```

[1] "My Name is Rani and My Age is 19 i am female i am living in surampalem City"
[1] "R version 4.3.2 (2023-10-31 ucrt)"

DATA TYPES IN R:

Data Types:

In programming, data type is an important concept.

Variables can store data of different types, and different types can do different things.

In R, variables do not need to be declared with any particular type, and can even change type after they have been set:

Basic Data Types

Basic data types in R can be divided into the following types:

- numeric - (10.5, 55, 787)
- integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
- complex - (9 + 3i, where "i" is the imaginary part)
- character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- logical (a.k.a. boolean) - (TRUE or FALSE)

We can use the class() function to check the data type of a variable:

EXAMPLE:

```
# numeric

x <- 10.5

class(x)

# integer

x <- 1000L

class(x)

# complex
```

```
x <- 9i + 3
class(x)

# character/string
x <- "R is exciting"
class(x)
# logical
x <- TRUE
class(x)
```

OUTPUT:

“numeric”

“integer”

“complex”

“character”

“logical”

Experiment- 2:

a) Implement R script to show the usage of various operators available in R language

Arithmetic operators:- These are used to perform basic mathematical operations like addition, subtraction, multiplication, division, modulus and etc;

1)Addition operator(+):-

It consists of 6 types of operators those are as fallows

1) Addition operator(+)

2) Subtraction operator(-)

3)Multiplication operator(*)

4) Division operator(/)

5) Power operator(^)

6) Modulo operator(%%)

Program:-

```
vec1 <- c(0, 2)
vec2 <- c(2, 3)

# Performing operations on Operands
cat ("Addition of vectors :", vec1 + vec2, "\n")
cat ("Subtraction of vectors :", vec1 - vec2, "\n")
cat ("Multiplication of vectors :", vec1 * vec2, "\n")
cat ("Division of vectors :", vec1 / vec2, "\n")
cat ("Modulo of vectors :", vec1 %% vec2, "\n")
cat ("Power operator :", vec1 ^ vec2)
```

Output:-

```
Addition of vectors : 2 5
Subtraction of vectors : -2 -1
Multiplication of vectors : 0 6
Division of vectors : 0 0.6666667
Modulo of vectors :  0 2
Power operator : 0 8
```

Logical Operators:- These operators are used to perform logical operations between 2 expressions. These will return Boolean expressions

Those are as follows:

1) Logical AND(&&)
2) Logical OR(||)
3) NOT Operator(!)

Program:-

```
vec1 <- c(0,2)
vec2 <- c(TRUE,FALSE)

# Performing operations on Operands
cat ("Element wise AND :", vec1 & vec2, "\n")
cat ("Element wise OR :", vec1 | vec2, "\n")
cat ("Logical AND :", vec1 && vec2, "\n")
cat ("Logical OR :", vec1 || vec2, "\n")
cat ("Negation :", !vec1)
```

Output:

Element wise AND : FALSE FALSE

Element wise OR: TRUE TRUE

Logical AND: FALSE

Logical OR: TRUE

Negation: TRUE FALSE

Relational Operators: Theese are used to find the reltionships between values and expressions.it also returns Boolean values

The fallowing are the relational operators

1) Less than(<)

2) Greater than(>)

3) Greater than or equal to(>=)

4) Less than or equal to(<=)

5) Not Equal to(!=)

Program:

```
vec1 <- c(0, 2)

vec2 <- c(2, 3)

cat ("Vector1 less than Vector2 :", vec1 < vec2, "\n")

cat ("Vector1 less than equal to Vector2 :", vec1 <= vec2, "\n")

cat ("Vector1 greater than Vector2 :", vec1 > vec2, "\n")
```

```
cat ("Vector1 greater than equal to Vector2 :", vec1 >= vec2, "\n")
cat ("Vector1 not equal to Vector2 :", vec1 != vec2, "\n")
```

Output:

```
Vector 1 less than Vector 2: TRUE TRUE
Vector 1 less than equal to Vector 2 : TRUE TRUE
Vector 1 greater  than Vector 2: FALSE FALSE
Vector 1 greater than equal to Vector 2 : FALSE FALSE
Vector 1 not equal to Vector 2 : TRUE TRUE
```

b) Implement R script to read person‘s age from keyboard and display whether heis eligible for voting or not.

Program:

```
{
age <- as.integer(readline(prompt = "Enter your age :"))
if (age >= 18) {
print(paste("You are valid for voting :", age))
} else{
print(paste("You are not valid for voting :", age))
}
}
```

Output:

Enter your age:17

You are not valid for

voting:17Enter your

age:19

You are valid for voting

c) Implement R script to find biggest number between two numbers. Program:

```
x <- as.integer(readline(prompt = "Enter first number :"))
y <- as.integer(readline(prompt = "Enter
second number :"))
if (x > y)
 {
print(paste("Greatest is :", x))
}
Else
 {
print(paste("Greatest is :", y))
}
}
```

Output:

Enter first
number:12
Enter second
number:16
[1]"Greatest is: 16"

d) Implement R script to check the given year is leap year or not.

Program:

```
{
year = as.integer(readline(prompt="Enter a
year: "))if((year %% 4) == 0) {
if((year %% 100) == 0) {
if((year %% 400) == 0) {
print(paste(year,"is a leap
year"))
}
 else {
        print(paste(year,"is not a leap year"))
}

else {
        print(paste(year,"is a leap year"))
        }
}
else
{
        print(paste(year,"is not a leap year"))
        }

}
```

Output:

Enter a year:2022

[1] "2022 ia not a leap year"

Experiment:- 3

a) Implement R Script to generate first N natural numbers.

Program:

```
n = readline("Enter the range: ")
n = as.integer(n)
cat("The first",n,"natural numbers:/n")
for (i in 1:n)
{
print(i)
}
```

Output :

```
Enter the range: 5
The first 5 natural number:
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

b) Implement R Script to check given number is palindrome or not.

Program:

```
n = readline("Enter a Number: ")
n = as.integer(n)
rev = 0
num = n
while (n> 0){
d = n %% 10
rev = rev * 10 + d
n = n %% 10
}
if(rev == num){
cat ("Number is Palindrome: ",num)
}
else {
cat ("Number is not Palindrome:",num)
}
```

Output :

Enter a Number : 121
Number is
Palindrome : 121
Enter a Number : 125
Number is not Palindrome :125

c) Implement R script to print factorial of a number.

Program :

```
n = readline("Enter a
Number: ")
n = as.integer(n)
fact <- 1
if(n==0||n=)
{
cat("Factorial of",n,"is : 1")
}
Else
 {
for ( i in 1 : n)
{
fact = fact *i
}
cat (n,"Factorial is: ",fact)
}
```

Output:

```
Enter a Number:
5
5Factorialis:120
Enter a Number:
0
0 Factorial is :1
```

d) Implement R Script to check given number is Armstrong or not.

Program :

```
n = raedline("Enter a
Number")
n = as.integer(n)
rev = 0
num = n
while(n >0)
{
d = n %%
10
rev = rev + d * d
* dn = n %% 10
}
if(rev == num )
{
print(cat(num,"is Armstrong
number"))
}
Else
 {
print(cat(num,"is not an Armstrong number"))
}
```

Output:

```
Enter a Number : 123
123 is not an Armstrong
numberEnter a Number :
153
153 is a Armstrong Number.
```

Experiment:-4

a) Implement R Script to perform various operations on string using string libraries.

Program :

```
n = readline("Enter a String: ")
rev = ""
for i in n :
rev = i + rev
if(rev == n){
cat ("String is Palindrome: ",n)
}
else
{
 cat ("String is not Palindrome:",n)
}
```

Output :

Enter a String : malayalam

String is Palindrome : malayalam

Enter a String : satya

String is not Palindrome : satya

b) Implement R Script to check given string is palindrome or not.

Program :

```
n = readline("Enter a
String: ")rev = ""
for i in n :
rev = i +
revif(rev
== n){
cat ("String is Palindrome: ",n)
}
else {
cat ("String is not Palindrome:",n)
}
```

Output :

```
Enter a String : malayalam
String is Palindrome :
malayalamEnter a String :
satya
String is not Palindrome : satya
```

c) Implement R script to accept line of text and find the number of characters, number of vowels and number of blank spaces in it.

Program:

```
text <- readline(prompt = "Enter a line of text: ")
results <- count_text(text)
cat("Number of characters:", results$characters, "\n")
cat("Number of vowels:", results$vowels, "\n")
cat("Number of blank spaces:", results$spaces, "\n")
```

Output:

```
Enter a line of text: Hello World
Number of Characters : 10
Number of vowels : 3
Number of blank spaces : 1
```

Experiment:-5

a) Implement R Script to create a list.

Program:

```
list1 <- list(1, "pradeep", c(1,2,3),
6:10) list1Output: [[1]]
[1] 1 [[2]]
[1] "pradeep"
[[3]][1] 1 2 3
[[4]]
[1] 6 7 8 9 10
```

Second method:

```
list1 <- list(1, "pradeep", c(1,2,3), 6:10) str(list1)
```

Output:

```
List of 4
$ : num 1
$ : chr "pradeep"
$ : num [1:3] 1 2 3
$ : int [6:10] 6 7 8 9 10
```

b) Implement R Script to access elements in the list.

Program:

```
x <- list(TRUE, 14, "pradeep")

print(x[1])

print(x[2])

print(x[3])
```

Output:

```
[[1]]

[1] TRUE [[1]]

[1] 14

[1] "pradeep"
```

c) Implement R Script to merge two or more lists. Implement R Script to perform matrixoperation

Program:

```
c(1,2,3,4,5,6,7,8),
nrow = 4,
ncol = 2,
byrow = FALSE )
```

Output:

```
[,1][,2]
[,1] 1 5
[2,] 2 6
[3,] 3 7
[4,] 4 8
```

Experiment 6:-

Implement R script to perform following operations:

Descrpition:

With R, it's Important that one understand that there is a difference between the actual R object and the manner in which that R object is printed to the console. Often, the printed output may have additional bells and whistles to make the output more friendly to the users. However, thesebells and whistles are not inherently part of the object

R has five basic or "atomic" classes of objects:

• character

• numeric (real numbers)

• integer

• complex

• logical (True/False)

The most basic type of R object is a vector. Empty vectors can be created with the vector() function. There is really only one rule about vectors in R, which is that A vector can only containobjects of the same class. But of course, like any good rule, there is an exception, which is a list, which we will get to a bit later. A list is represented as a vector but can contain objects of different classes. Indeed, that's usually why we use them.

There is also a class for "raw" objects, but they are not commonly used directly in data analysis

a) Various operations on vectors.

Program:

```
X <- c(1, 4, 5, 2, 6, 7)
Print('using c function')
print(X)
```

Output:

```
[1] "using c
 function"[1] 1 4
 5 2 6 7
```

Program:

```
Y <- seq(1, 10, length.out = 5)
print('using seq() function')
print(Y)
```

Output:

```
[1] "using seq() function"
[1] 1.00 3.25 5.50 7.75 10.00
```

b) Finding the sum and average of given numbers using arrays.

Array:

Of course, if we have a data set consisting of more than two pieces of categorical informationabout each subject, then a matrix is not sufficient. The generalization of matrices to higher dimensions is the array. Arrays are defined much like matrices, with a call to the array() command. Here is a 2 × 3 × 3 array:

```
#sum,min&max of an array

>Vec1<-c(1,2,3,4)

>array1<-array(vec1)

>print(array)
[1] 1 2 3 4
>print(array1)

#sum of elements

>sum(array1)

>[1]10

#Minimum:

>min(array1)
[1]1
#Maximum:

>max(array1)
[1]4
#AVERAGE:
print(mean(array1)
[1]2.5
```

Output:

Array is :[1] 1 2 3 4
Minimum Element is : [1]
1Maximum Element is :[1]
4Sum of Array is :10
Average is : 2.5

c) To display elements of list in reverse order.

Program:

```
x <- list("c", "b",
"a")
result = rev(x)
print(result)
```

Output:

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

d) Finding the minimum and maximum elements in the array.

Program:

```
x <- c(8, 2, 5, 4, 9, 6, 54, 18)
range()
```

Output

```
2  54
```

Experiment:-7

a) Implement R Script to perform various operations on matrices

Descrpition:

Matrices are much used in statistics, and so play an important role in R. To create a matrix use thefunction matrix (), specifying elements by column first:

Program:

```
one <- matrix (1:10, nrow=10)
one
two <- matrix(51:60,nrow=2) three <-matrix(61:70,nrow=2) two
three
two+three
mat <- matrix(5:10,nrow=2,ncol=3,byrow=TRUE)
 mat
mat2 <- matrix(1:9,nrow=3,dimnames=list(c("r1","r2","r3"),c("c1","c2","c3"))) mat2 mat2[2,2]
mat2[,1]
mat2[2,]
 dim(mat2)
```

OUTPUT:

```
> matrix(

+ c(1,2,3,4,5,6,7,8),
+ nrow = 4,
+ ncol = 2,
+ byrow = FALSE )
[,1][,2]
[,1] 1 5

[,2] 2 6

[,3] 3 7

[,4] 4 8

> two <- matrix(51:60,nrow=2)
> three <-matrix(61:70,nrow=2)
> two
[,1] [,2] [,3] [,4] [,5]
[1,] 51 53 55 57 59
[2,] 52 54 56 58 60
```

```
> three
[,1] [,2] [,3] [,4] [,5]
[1,] 61 63 65 67 69
[2,] 62 64 66 68 70


> two+three
[,1] [,2] [,3] [,4] [,5]
[1,] 112 116 120 124 128
[2,] 114 118 122 126 130

> mat <- matrix(5:10,nrow=2,ncol=3,byrow=TRUE)

> mat
[,1] [,2] [,3]
[1,] 5 6 7
[2,] 8 9 10
> mat2 <- matrix(1:9,nrow=3,dimnames=list(c("r1","r2","r3"),c("c1","c2","c3")))
> mat2 c1 c2 c3

r1 1 4 7
r2 2 5 8
r3 3 6 9

> mat2[2,2]
[1] 5
> mat2[,1] r1 r2 r3
1 2 3
> mat2[2,]
c1 c2 c3 2 5 8
> dim(mat2) [1] 3 3
```

b) Implement R Script to extract the data from data frames.

Program:

```
emp.data <- data.frame( emp_id = c (1:5),
emp_name =
c("Rick","Dan","Michelle","Ryan","Gary"),salary =
c(623.3,515.2,611.0,729.0,843.25),
start _date = as. Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
"2015-03-27")),
Strings As Factors = FALSE
)
# Print the data
frame.Print (emp
.data) str(emp. data)
summary(emp
.data)
result <-
data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

Output:

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

'data .frame': 5 obs. of 4 variables:
$ emp_ id : int 1 2 3 4 5
$ emp_ name : chr "Rick" "Dan" "Michelle" "Ryan" ...
$ salary : num 623 515 611 729 843
$ start_ date: Date, format: "2012-01-01" "2013-09-23" ...
> summary(emp. data)

emp_ id emp _name salary start_date
Min. :1 Length:5 Min. :515.2 Min. :2012-01-01
1st Qu.:2
Class :character
1st Qu.:611.0
1st Qu.:2013-09-23
```

```
 Median  :3
Mode :character
r 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
> result <- data. frame(emp .data $emp_ name ,emp .data $salary)

> print(result)

emp. data.emp_name emp.data.salary
1 Rick 623.30
2 Dan 515.20
3 Michelle 611.00
4 Ryan 729.00
5 Gray 843.25
```

c) Write R script to display file contents.

Program:

```
file. show("pradeep.txt")
file.show("pradeep.txt")
file.show("pradeep.txt")
file.remove("pradeep.txt")
```

Output:

```
TRUE
TRUE
TRUE
TRUE
```

d) Write R script to copy file contents from one file to another

Program:

```
1:Create a new directory dir.create("newdir") newDirPath <- "newdir"
2: Create a new file files <- c("data.txt") file.create(files)
    newFilePath <- "data.txt"

3: Copy a file from one folder to another dir.create("newdir")
    newDirPath <- "newdir"

  files <- c("data.txt") file.create(files) newFilePath <- "info.txt"
   file.copy(newFilePath, newDirPath)
```

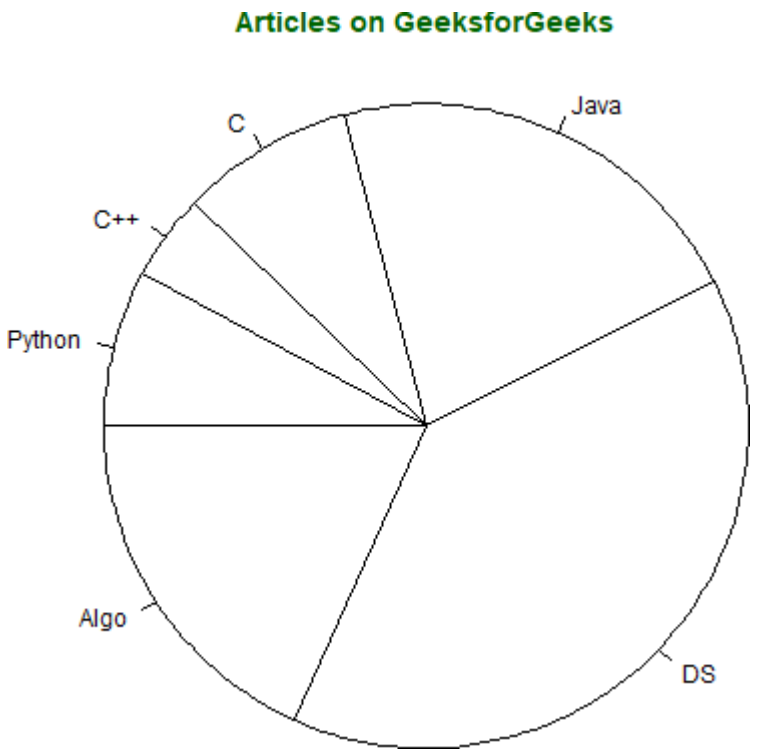
Output:

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

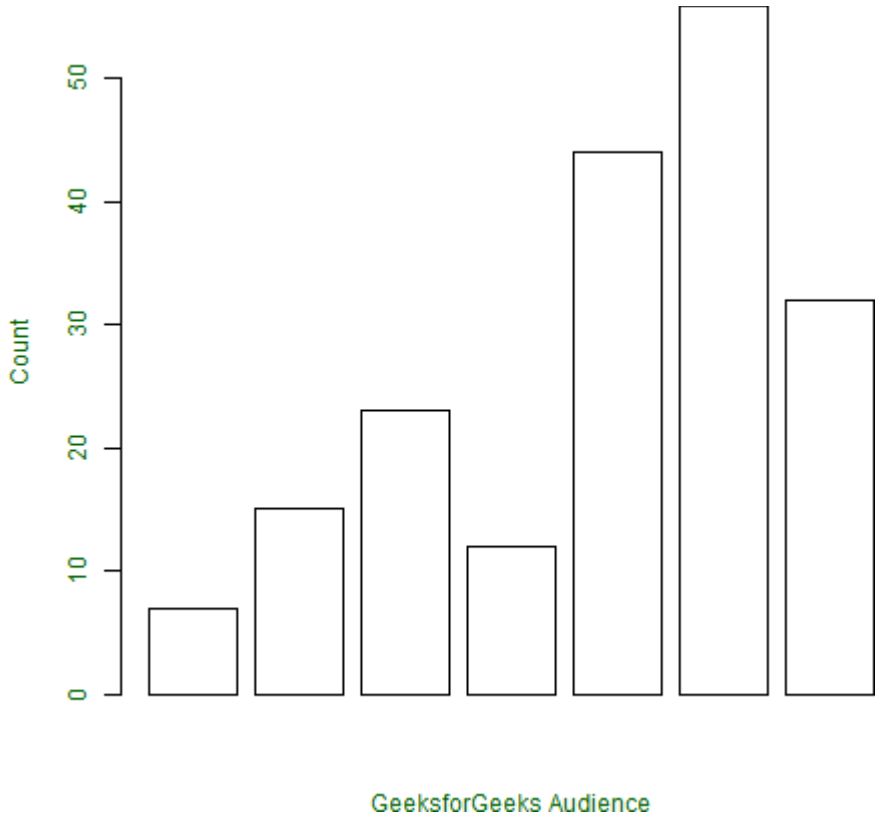
Experiment:-8

a) Implement R Script to create a Pie chart, Bar Chart, scatter plot and Histogram(Introductiontoggplot2 graphics).

```
# defining vector x with number of articles
x <- c(210, 450, 250, 100, 50, 90)

 # defining labels for each value in x

names(x) <- c("Algo", "DS", "Java", "C", "C++", "Python")#
 output to be present as PNG file
png(file = "piechart.png")

 # creating pie chart


  pie(x, labels = names(x), col = "white",
  main = "Articles on GeeksforGeeks", radius = -1,
  col.main = "darkgreen")

# saving the file
dev.off()
```




```
# defining vector
x <- c(7, 15, 23, 12, 44, 56, 32)

# output to be present as PNG
file
png(file = "barplot.png")

# plotting vector
barplot(x, xlab = "GeeksforGeeks Audience",
        ylab = "Count", col = "white",
        col.axis = "darkgreen",
        col.lab = "darkgreen")

# saving the file
dev.off()
```



taking input from dataset Orange already

 # present in R

orange <- Orange[, c('age', 'circumference')]

 # output to be present as PNG

file

png(file = "plot.png")

 # plotting

plot(x = orange$age, y = orange$circumference, xlab = "Age",

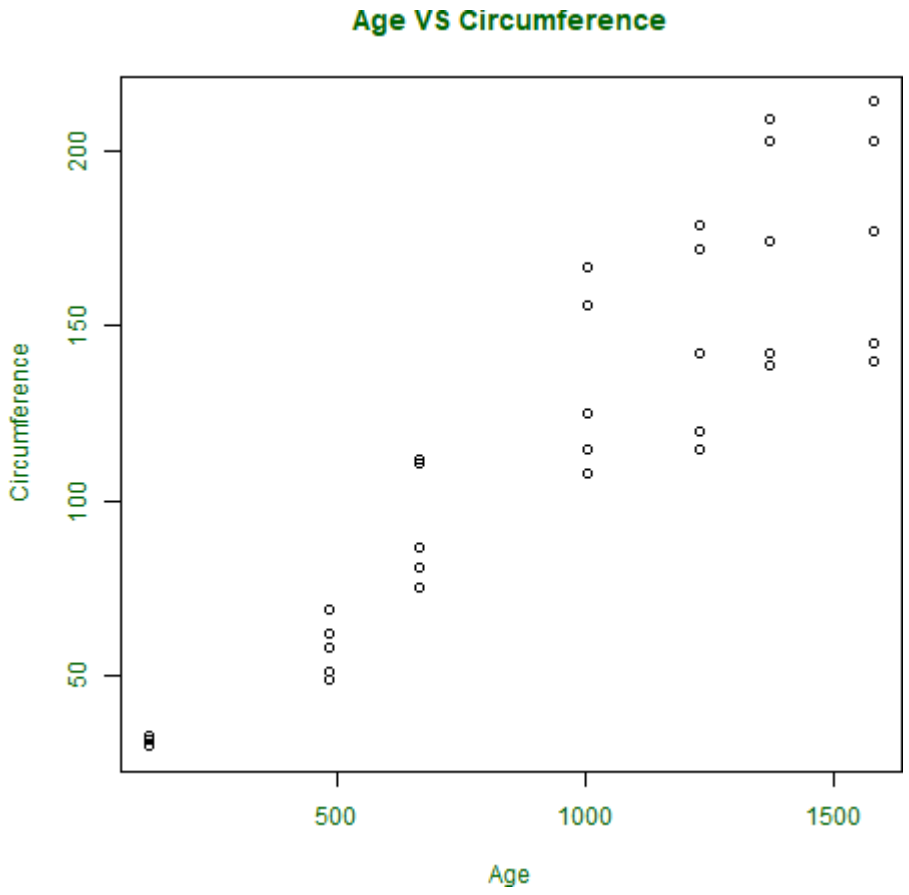
ylab = "Circumference", main = "Age VS Circumference",

col.lab = "darkgreen", col.main = "darkgreen",

col.axis = "darkgreen")

 # saving the file

dev.off()



x <- c(42, 21, 22, 24, 25, 30, 29, 22,

23, 23, 24, 28, 32, 45, 39, 40)

 # output to be present as PNG

file

png(file = "boxplot.png")

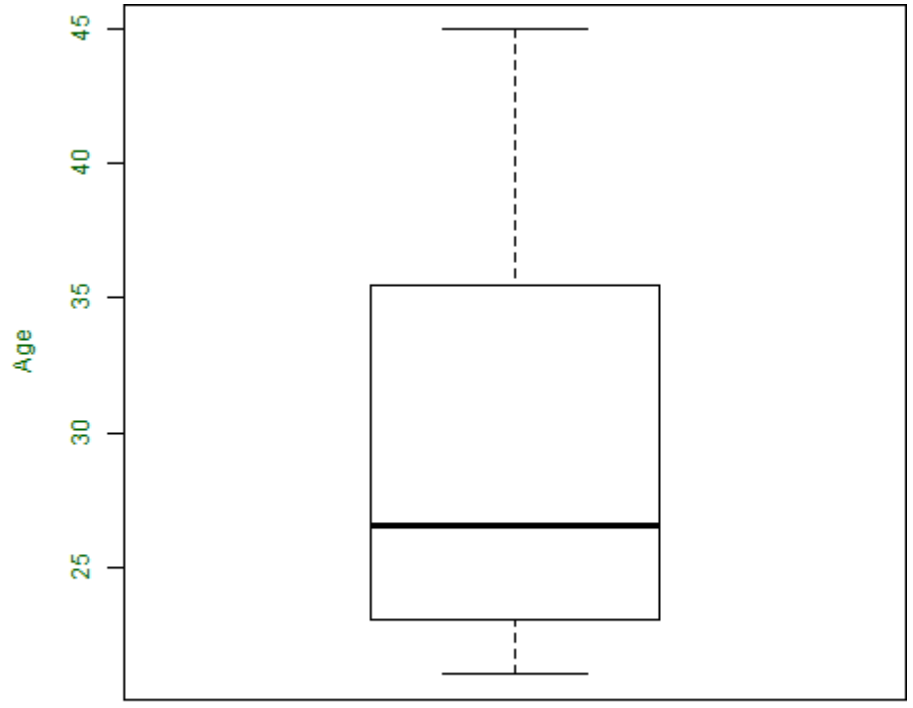
 # plotting

boxplot(x, xlab = "Box Plot", ylab = "Age",

col.axis = "darkgreen", col.lab = "darkgreen")

 # saving the file

dev.off()



Box Plot

+

b) Implement R script to perform mean, median, mode, range, summary, variance,standard deviation operations

PROGRAM:

MEAN:

```
Data_Cars <- mtcars mean(Data_Cars$wt)
```

OUTPUT:

[1] 3.21725

MEDIAN:

```
Data_Cars <- mtcars median(Data_Cars$wt)
```

OUTPUT:

[1] 3.325

MODE:

```
Data_Cars <- mtcars names(sort(-table(Data_Cars$wt)))[1]
```

OUTPUT:

[1] "3.44"

VARIANCE:

```
# R program to get variance of a list # Taking a list of elements
list = c(2, 4, 4, 4, 5, 5, 7, 9)
# Calculating variance using var()
print(var(list))
```

OUTPUT:

[1] 4.571429

STANDARD DEVIATION:

```
 # R program to get
 # standard deviation of a
list# Taking a list of
elements list = c(2, 4, 4, 4,
5, 5, 7, 9)
 # Calculating
 standard# deviation
 using sd()
 print(sd(list)
)
```

OUTPUT

2.13809

c) Introduction to ggplot2 graphics.

Introduction:

ggplot2 is a powerful and widely used R package for creating statistical graphics. It offers a grammar-based approach, making it intuitive and flexible for creating a wide variety of plots. Here's an introduction to get you started:

1. The Grammar of Graphics:

ggplot2 builds upon the "Grammar of Graphics" framework, which proposes that a graphic can be decomposed into several fundamental components:

- **Data:** The raw data you want to visualize.
- **Aesthetic Mappings:** How your data maps to visual properties like color, size, shape, etc.
- **Geometries:** The geometric shapes used to represent your data (e.g., points, lines, bars, etc.).
- **Scales:** Define the mapping between data values and visual properties (e.g., color scale, size scale, etc.).
- **Coordinates:** The coordinate system used to position your data points (e.g., cartesian, polar, etc.).
- **Stats:** Statistical transformations applied to data (e.g., mean, median, etc.).
- **Facets:** Panels dividing the plot to display data subsets based on different variables.
- **Annotations:** Additional elements like labels, titles, and legends.

By combining these components in a specific order, you can create a wide range of visualizations.

2. Basic ggplot2 Workflow:

1. **Load the ggplot2 package:** Use `library(ggplot2)` to import the package.
2. **Prepare your data:** Ensure your data is in a tidy format, meaning each row represents a single observation and each column represents a single variable.
3. **Create the ggplot object:** Use `ggplot(data, aes(...))` where `data` is your data frame and `aes(...)` defines the aesthetic mappings.
4. **Add geometric layers:** Use functions like `geom_point()`, `geom_line()`, `geom_bar()`, etc. to define the geometric shapes used to represent your data.
5. **Customize the plot:** Use various options and arguments to customize visual elements like colors, scales, labels, titles, etc.

Experiment-9

a) Implement R Script to perform Normal, Binomial distributions.

The binomial distribution model deals with finding the probability of success of an event which has only two possible outcomes in a series of experiments.

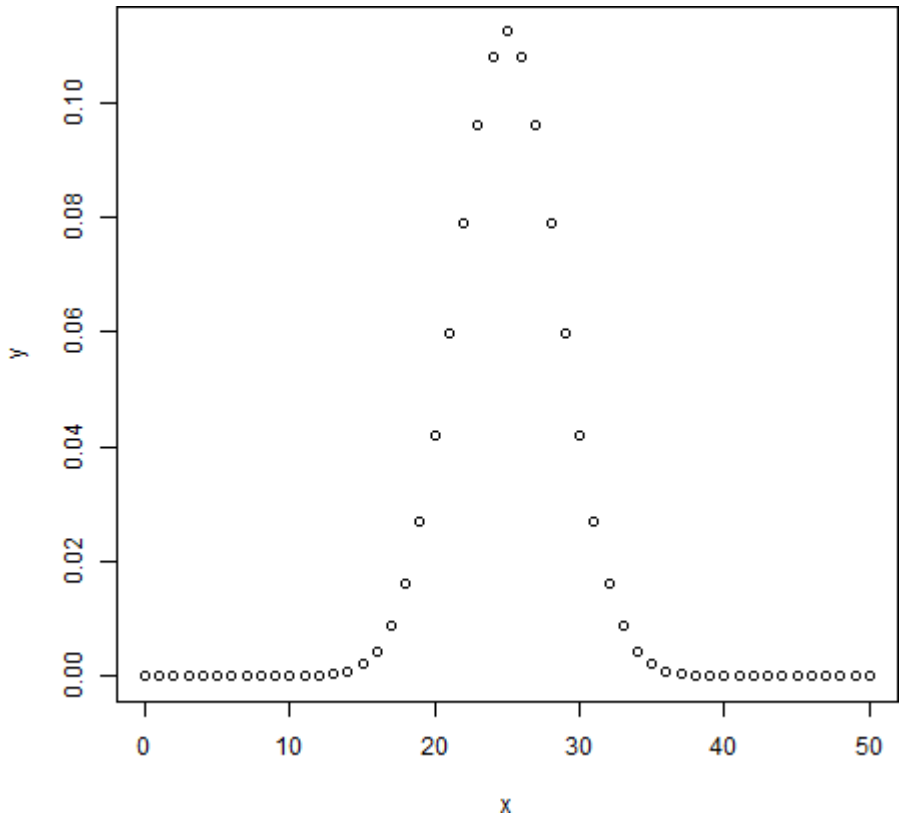
For example, tossing of a coin always gives a head or a tail. The probability of finding exactly 3 heads in tossing a coin repeatedly for 10 times is estimated during the binomial distribution. R has four in-built functions to generate binomial distribution. They are describe below.

```
dbinom(x,   size,   prob)
pbinom(x,   size,   prob)
qbinom(p,   size,   prob)
rbinom(n, size, prob)
```

Following is the description of the parameters used − x is a vector of numbers. p is a vector of probabilities. n is number of observations. size is the number of trials. prob is the probability of success of each trial. dbinom() This function gives the probability density distribution at each point.

Program:-

```
# Create a sample of 50 numbers which are incremented by 1.x <-
seq(0,50,by = 1)
#Create the binomial distribution.
y <- dbinom(x,50,0.5)

# Give the chart file a name.png(file
= "dbinom.png")

# Plot the graph for this sample.
plot(x,y)

# Save the file.
dev.off()
```




```
# Probability of getting 26 or less heads from a 51 tosses of a coin.
x <- pbinom(26,51,0.5)

print(x)
```

OUTPUT:-

```
[1] 0.610116
```

qbinom()

This function takes the probability value and gives a number whose cumulative value matches the probability value.

Program:-

```
# How many heads will have a probability of 0.25 will come out when a coin
# is tossed 51 times.
x <- qbinom(0.25,51,1/2)

print(x)
```

Output:-

```
 [1] 23
```

rbinom()

This function generates required number of random values of given probability from a given sample.

Program:-

```
# Find 8 random values from a sample of 150 with probability of 0.4.
x <- rbinom(8,150,.4)

print(x)
```

output:-

```
[1] 58 61 59 66 55 60 61 6
```

b) Implement R Script to perform correlation, Linear and multiple regression.

In Linear Regression these two variables are related through an equation, where exponent (power) of both these variables is 1. Mathematically a linear relationship represents a straight line when plotted as a graph. A non-linear relationship where the exponent of any variable is not equal to 1 creates a curve. The general mathematical equation for a linear regression is − y
= ax + b lm() Function This function creates the relationship model between the predictor andthe response variable.
Syntax The basic syntax for lm() function in linear regression is − lm(formula,data) predict() Function
Syntax The basic syntax for predict() in linear regression is − predict(object, newdata) Following is the description of the parameters used − object is the formula which is already created using the lm() function. newdata is the vector containing the new value for predictorvariable. Predict the weight of new persons

Program:-

```
# The predictor vector.

x <- c(151, 174, 138, 186, 128, 136, 179, 163, 152, 131)

# The resposne vector

. y <- c(63, 81, 56, 91, 47, 57, 76, 72, 62, 48)

# Apply the lm()

function.relation <-

lm(y~x)

# Find weight of a person with height

170

.a <- data.frame(x = 170)

result <-

predict(relation,a)

print(result)
```

Output:-

When we execute the above code, it produces the following

result −176.22869

MULTIPLE REGRESSION:

Multiple regression is an extension of linear regression into relationship between more than two variables. In simple linear relation we have one predictor and one response variable, but in multipleregression we have more than one predictor variable and one response variable. The general mathematical equation for multiple regression is –

y = a + b1x1 + b2x2 +...bnxn

Following is the description of the parameters used –

y is the response variable. a, b1, b2...bn are the

coefficients.x1, x2, ...xn are the predictor

variables.

We create the regression model using the lm() function in R.

The model determines the value of the coefficients using the input data. Next we can predict thevalue of the response variable for a given set of predictor variables using these coefficients.

lm() Function

This function creates the relationship model between the predictor and the

response variable. SyntaxThe basic syntax for lm() function in multiple

regression is –

lm(y ~ x1+x2+x3...,data)

Following is the description of the parameters used –

formula is a symbol presenting the relation between the response variable and predictor variables. datais the vector on which the formula will be applied.

Example

Input Data Consider the data set "mtcars" available in the R environment. It gives a comparison between different car models in terms of mileage per gallon (mpg), cylinder displacement("disp"),horse power("hp"), weight of the car("wt") and some more parameters.

The goal of the model is to establish the relationship between "mpg" as a

response variable with "disp","hp" and "wt" as predictor variables. We create a subset of these variables from the mtcars dataset for this purpose.

```
input <-

mtcars[,c("mpg","disp","hp","wt")

]print(head(input))
```

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

	mpg	disp	hp	wt
Mazda RX4	21.0	160	110	2.620
Mazda RX4 Wag	21.0	160	110	2.875
Datsun 710	22.8	108	93	2.320
Hornet 4 Drive	21.4	258	110	3.215
Hornet Sportabout	18.7	360	175	3.440
Valiant	18.1	225	105	3.460

Experiment -10

Introduction to Non-Tabular Data Types: Time series, spatial data, Network data. DataTransformations: Converting Numeric Variables into Factors, Date Operations, String Parsing, Geocoding.

A.Non-Tabular Data Types:

TIME SERIES:-

Time Series in R is used to see how an object behaves over a period of time. In R, it can be easilydone by ts() function with some parameters. Time series takes the data vector and each data is connected with timestamp value as given by the user

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language usesmany functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called time-series object. It is also a R data object like a vector or data frame.

The time series object is created by using the ts() function.

Syntax:-

The basic syntax for ts() function in time series analysis is –timeseries.object.name <- ts(data, start, end, frequency) Following is the description of the parameters used –

• data is a vector or matrix containing the values used in the time series.

• start specifies the start time for the first observation in time series.

• end specifies the end time for the last observation in time series.

• frequency specifies the number of observations per unittime.Except the parameter "data" all other parameters are optional.

EXAMPLE PROGRAM:

```
# Get the data points in form of a R vector

. rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

 # Convert it to a time series object.

 rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12)
 # Print the timeseries data
print(rainfall.timeseries

 # Give the chart file a name.
 png(file = "rainfall.png")
 # Plot a graph of the time series.
 plot(rainfall.timeseries)
 # Save the file.
dev.off()
```

OUTPUT:-

```
 Jan       Feb      Mar       Apr       May      Jun       Jul       Aug      Sep
 2012  799.0   1174.8    865.1     1334.6 635.4     918.5     685.5   998.6 784.2
Oct        Nov      Dec
2012     985.0     882.8 1071.0
```

SPATIAL DATA:-

Spatial data is any type of data that directly or indirectly references a specific geographical area orlocation. Sometimes called geospatial data or geographic information, spatial data can also numerically represent a physical object in a geographic coordinate system. However, spatial data ismuch more than a spatial component of a map

. • ggmap: extends the plotting package ggplot2 for maps

• rgdal: R's interface to the popular C/C++ spatial data processing library gdal

• rgeos: R's interface to the powerful vector processing library geos

• maptools: provides various mapping functions

• dplyr and tidyr: fast and concise data manipulation packages

• tmap: a new packages for rapidly creating beautiful maps

Data Transformations:

Converting Numeric Variables into Factors For converting a numeric into factor we use cut() function. cut() divides the range of numeric vector(assume x) which is to be converted by cutting intointervals and codes its value (x) according to which interval they fall

Level one corresponds to the leftmost, level two corresponds to the next leftmost, and so on.

Syntax:cut.default(x, breaks, labels = NULL, include.lowest = FALSE, right = TRUE, dig.lab = 3)

Creating vectors

age <- c(40, 49, 48, 40, 67, 52, 53)

salary <- c(103200, 106200, 150200, 10606, 10390, 14070, 10220)

gender <- c("male", "male", "transgender","female", "male", "female", "transgender")# Creating data frame named employee

employee<- data.frame(age, salary, gender)

Creating a factor corresponding to age # with three equally spaced levelswfact = cut(employee$age, 3) table(wfact)

Output:

wfact (40,49] (49,58] (58,67] 4 2 1

Date

operations:

EXAMPLE

1. mydate <- as.Date('1970-01-01')

2. mydate

3. [1] "1970-01-01"

4.

class(mydate)

5. [1] "Date"

Note, that the datatype of the mydate variable is "Date" which confirms that R has internally storedthe date that we passed as a character string to the as.Date() function.

1. mydate <- mydate + 7

2. mydate

3. [1] "1970-01-08"

4.

class(mydate)

5. [1] "Date"

6.

as.numeric(mydate)

7. [1]

7 Adding seven days gives us the 8th of January 1970 and then converting it to numeric correctly outputs 7 days since origin. The datatype of mydate is still a "Date".The key point to note here is thatinternally R stores all dates as the number of days from the origin date of 1st January 1970 and the datatype of all these dates is "Date".

STRING PARSING:

strsplit() method in R Programming Language is used to split the string by using a delimiter.strsplit() Syntax:

Syntax:

strsplit(string, split)

R program to split a string

Given String gfg < - "Geeks For Geeks"

Using strsplit() method answer < -

strsplit(gfg, " ")print(answer)

Output:

[1] "Geeks" "For"

"Geeks"

GEO CODING:

Geocoding can be simply achieved in R using the geocode() function from the ggmap library. Thegeocode function uses Googles Geocoding API to turn addresses from text to latitude and longitude pairs very simply

EXPERIMENT-11

Introduction Dirty data problems: Missing values, data manipulation, duplicates, forms of datadates, outliers, spelling

Missing Values in R:

is.na() Function for Finding Missing values:

A logical vector is returned by this function that indicates all the NA values present. It returns aBoolean value. If NA is present in a vector it returns TRUE else FALSE.

example:

x<- c(NA, 3, 4, NA, NA, NA) is.na(x)

output:

[1] TRUE FALSE FALSE TRUE TRUE TRUE

is.nan()Function for Finding Missing values:

A logical vector is returned by this function that indicates all the NaN values present. It returns aBoolean value. If NaN is present in a vector it returns TRUE else FALSE.

example:

x<- c(NA, 3, 4, NA, NA, 0 / 0, 0 / 0) is.nan(x)

output:

[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE

DUPLICATE VALUES IN R:

1.duplicated(): The R function duplicated() returns a logical vector where TRUE specifieswhich elements of a vector or data frame are duplicates

. example:

x <- c(1, 1, 4, 5, 4, 6)

duplicated(x)

output

[1] FALSE TRUE FALSE FALSE TRUE FALSE

E t duplicate elements:

x x <- c(1, 1, 4, 5, 4, 6)

t x[duplicated(x)]

r [1] 1 4

a

c

REMOVING DUPLICATES IN R:

If you want to remove duplicated elements, use !duplicated(), where ! is a logical negation:
EXAMPLE:

x <- c(1, 1, 4, 5, 4, 6) x

[!duplicated(x)]

OUTPUT:

[1] 1 4 5 6

forms of DATA DATES IN R:

1) **Sys.Date():**
In R programming, if you use Sys.Date() function, it will give you the systemdate.
syntax:Sys.Date()
output:
[1] "2022-04-20"

1) **Sys.timezone() :**

a function named Sys.timezone() that allows us to get the timezone based on the location atwhich the user is running the code on the system.
syntax:
Sys.timezone
()output:
[1] "Asia/Calcutt

a"

[2] 2)Sys.time() :

we have the Sys.time() function. Which, if used, will return the current date as well as the time of thesystem with the timezone details.

syntax:

Sys.time()

output:

[1] "2022-04-20 11:02:56 IST"

3) as.date():

as.Date() function allows us to create a date value (without time) in R programming. It allows thevarious input formats of the date value as well through the format = argument.

example:

```
mydate<-as.date("2014-04-30")
        mydate
```

ouptut:

[1] "2014-04-30"

Experiment:-12

Data sources: SQLite examples for relational databases, Loading SPSS and SAS files, Reading from Excel and Google Spreadsheets, API and web scraping examples.

Data sources:

SQLite examples for relational databases, Loading SPSS and SAS files, Reading from Google Spreadsheets, API and web scraping examples.

Popular examples of standard relational databases include Microsoft SQL Server, Oracle Database, MySQL and IBM DB2. Cloud-based relational databases, or database as a service, are also widely used because they enable companies to outsource database maintenance, patching and infrastructure support requirements

SQLite Database is an open-source database provided in Android which is used to store data inside the user's device in the form of a Text file. We can perform so many operations on this data such as adding new data, updating, reading, and deleting this data

. .Creating a database

Now we're ready to create a database. Before we do that though, you'll probably want to change the working directory of SQLite. We can do that with the .cd command. For example, let's say we have a folder on our desktop called 'data', then we would change our working directory as follows using the SQLite command line interface:

```
sqlite> .cd 'C:\Users\clayford\Desktop\data'
```

Loading SPSS and SAS files:

The easiest way to import SPSS files into R is to use the read_sav() function from the haven library. This function uses the following basic syntax:

```
data <- read_sav('C:/Users/User_Name/file_name.sav')
```

The following step-by-step example shows how to import a SPSS file into R in practice.

Step 1: Download a SPSS File For this example, we'll download the SPSS file called healthdata.sav from this page.

Step 2: Install haven Package Next, we'll install the haven package in R:

```
install.packages('haven')
```

We'll then load the package

```
library(haven)
```

Step 3: Import the SPSS File

Next, we'll use the read_sav() function to import the SPSS file:

```
data <- read_sav('C:/Users/bob/Downloads/healthdata.sav')
```

Once we've imported the SPSS file, we can get a quick summary of the data:

```
#view class of data
class(data) [1] "tbl_df" "tbl" "data.frame"
#display dimensions of data frame dim(data)
```

[1] 185 3

#view first six rows of data

head(data)

CD EXERC HEALTH

1 1 [ordered] 3 6

2 2 [did not order] 3 7

3 2 [did not order] 5 6

4 2 [did not order] 5 3

5 1 [ordered] 5 6

6 2 [did not order] 2 3

The easiest way to import SAS files into R is to use the read_sas() function from the haven library. This function uses the following basic syntax:

data <- read_sas('C:/Users/User_Name/file_name.sas7bdat')

The following step-by-step example shows how to import a SAS file into R in practice.

Step 1: Download a SAS Data File

For this example, we'll download the SAS file called cola.sas7bdat from this page.

Step 2: Install haven Package

Next, we'll install the haven package in R:

install.packages('haven')

We'll then load the package: library(haven)

Step 3: Import the SAS File

Next, we'll use the read_sas() function to import the SAS file:

data <- read_sas('C:/Users/bob/Downloads/cola.sas7bdat')

Once we've imported the SAS file, we can get a quick summary of the data:

#view class of data

class(data)

[1] "tbl_df" "tbl" "data.frame"

#display dimensions of data frame dim(data)

[1] 5466 5

#view first six rows of data

head(data)

ID	CHOICE	PRICE	FEATURE	DISPLAY
1	1	0 1.79	0	0
2	1	0 1.79	0	0
3	1	1 1.79	0	0
4	2	0 1.79	0	0
5	2	0 1.79	0	0
6	2	1 0.890	1	1

Reading from Google Spreadsheets:

You can read google sheets data in R using the package 'googlesheets4'. This package will allow you to get into sheets using R.

First you need to install the ‘googlesheets4’ package in R and then you have to load the library to proceed further

. #Install the required package install.packages('googlesheets4')

#Load the required library library(googlesheets4)

1. Setup the Authorization

You cannot read the data from google sheets right away. As Gsheets are web-based spreadsheets, they will be associated with your google mail. So, you have to allow R to access the Google sheets.

You would have used functions like read.csv or read.table to read data into R. But, here you don’t need to mention the file type. All you need is to copy the google Sheets link from the browser and paste it here and run the code

. Once you run the below code, you can see an interface for the further process.

#Read google sheets data into R

x <- read_sheet('https://docs.google.com/spreadsheets/d/1J9-ZpmQT_oxLZ4kfe5gRvBs7vZhEGhSCIpNS78XOQUE/edit?usp=sharing')

Is it OK to cache OAuth access credentials in the folder

1: Yes

2: No

You have to select option 1: YES to continue to the authorization process.

As a first step, if you are having multiple G accounts logged in, it will ask you to continue with your account as shown below.

• You have to select your account to authorise R to access the G sheets. This process is followed by multiple authorizations. You have to allow R to in all those steps.

• In the below picture, you will be shown the permissions you are giving to the Tidyverse API. Click "Allow" and you are done.

• After the successful authorization, you can see the completion message.

• After this, you will see a successful authorization message in the R studio as shown below.

2. Reading the Data into R

It's great that you have completed the authorization process and it went successfully. Now let's see how we can read the data into R from Google sheets. #Reads data into R

df <- read_sheet('https://docs.google.com/spreadsheets/d/1J9-ZpmQT_oxLZ4kfe5gRvBs7vZhEGhSCIpNS78XOQUE/edit?usp=sharing')

#Prints the data

df

A tibble: 1,000 x 20

months_loan_dura~ credit_history purpose amount savings_balance employment_leng~

<chr>	<dbl><chr><chr>	<dbl><chr>
1< 0 DM	6 critic~ radio~	1169 unknown
2 1 - 200 DM	48 repaid radio~	5951 < 100 DM
3 unknown	12 critic~ educa~	2096 < 100 DM
4 < 0 DM	42 repaid furni~	7882 < 100 DM
5 < 0 DM	24 delayed car (~	4870 < 100 DM
6 unknown	36 repaid educa~	9055 unknown
7 unknown	24 repaid furni~	2835 501 - 1000 DM
8 1 - 200 DM	36 repaid car (~	6948 < 100 DM
9 unknown	12 repaid radio~	3059 > 1000 DM
10 1 - 200 DM	30 critic~	car (~ 5234 < 100 DM

... with 990 more rows, and 14 more variables: installment_rate ,

personal_status , other_debtors , residence_history

, # property , age , installment_plan , housing , # existing_credits , default , dependents , telephone , # foreign_worker , job

Here you can see, how R can read the data from Google sheets using the function 'read_sheet' function.

I am also adding the dataframe here for your reference / understanding.

1. Reading Google sheets into R using Sheet ID

You don't need to copy the sheet link to read the data. You can only copy the sheet ID and can use that with the read_sheet function. It will read the data as usual.

If you are not aware of sheet ID, I have added a sheet link and I have highlighted the Sheet ID with color. You can copy this ID can follow the same process.
https://docs.google.com/spreadsheets/d/1J9-ZpmQT_oxLZ4kfe5gRvBs7vZhEGhSCIpNS78XOQUE/edit#gid=0

You can find the discussed code below.

#Reads the data with Sheet ID into R

df <- read_sheet('1J9-ZpmQT_oxLZ4kfe5gRvBs7vZhEGhSCIpNS78XOQUE')

#Prints the data

df

This code will give the same output i.e. data. I have used credit data for the whole illustration. You can use any data for this purpose. I hope from now, reading google sheets into R is not an issue for you.

API and web scraping:

Example:

R program to illustrate

Web Scraping

Import rvest library library(rvest)

Reading the HTML code from the website webpage = read_html("https://www.geeksforgeeks.org / data-structures-in-r-programming")

Using CSS selectors to scrape the heading section heading = html_node(webpage, '.entry-title' # Converting the heading data to text text = html_text(heading) print(text)

Using CSS selectors to scrape # all the paragraph section

Note that we use html_nodes() here paragraph = html_nodes(webpage, 'p')

```
# Converting the heading data to tex
t pText = html_text(paragraph)
 # Print the top 6 data
print(head(pText))
```

output:

[1] “Data Structures in R Programming”

[1] “A data structure is a particular way of organizing data in a computer so that it can be used effectively. The idea is to reduce the space and time complexities of different tasks. Data structures in R programming are tools for holding multiple values. ” [2] “R’s base data structures are often organized by their dimensionality (1D, 2D, or nD) and whether they’re homogeneous (all elements must be of the identical type) or heterogeneous (the elements are often of various types). This gives rise to the five data types which are most frequently utilized in data analysis. the subsequent table shows a transparent cut view of those data structures.” [3] “The most essential data structures used in R include:” [4] “” [5] “A vector is an ordered collection of basic data types of a given length. The only

key thing here is all the elements of a vector must be of the identical data type e.g homogeneous data structures. Vectors are one-dimensional data structures.” [6]

“Example:”

Example for API:

```
 # Installing the packages
install.packages("httr")
install.packages("jsonlite")
# Loading packages
library(httr)
library(jsonlite)
 # Initializing API Call
call <- http://www.omdbapi.com/?i=tt3896198&apikey=948d3551&plot=full&r=json
```

Getting details in AP

I get_movie_details <- GET(url = call)

Getting status of HTTP Call status_code(get_movie_details)

Content in the API str(content(get_movie_details))

Converting content to text get_movie_text <- content(get_movie_details, "text", encoding = "UTF-8") get_movie_text

Parsing data in JSON

get_movie_json <- fromJSON(get_movie_text, flatten = TRUE) get_movie_json

Converting into dataframe get_movie_dataframe <- as.data.frame(get_movie_json)

Output:

• Status