ITA0448 – STATISTICS WITH R PROGRAMMING

Lab manual Day 1

P.Barath

192121147

BASIC OPERATIONS IN R

1.Write The Commands To Perform Basic Arithmetic In R.

Program:

```
a<-16
b<-3
add<-a+b
sub=a-b
multi=a*b
division=a/b
integer_division=a%/%b
exponent=a^b

print(paste("addition of two numbers 16 and 3 is:",add))
print(paste("Subtracting Number 3 from 16 is : ", sub))
print(paste("Multiplication of two numbers 16 and 3 is : ", multi))
print(paste("Division of two numbers 16 and 3 is : ", division))
print(paste("Integer Division of two numbers 16 and 3 is : ", Integer_Division))
print(paste("Exponent of two numbers 16 and 3 is : ", exponent))
```

output:

```
> a<-16
> b<-3
> add<-a+b
> sub=a-b
> multi=a*b
> division=a/b
> integer_division=a%/%b
> exponent=a^b
>
> print(paste("addition of two numbers 16 and 3 is:",add))
[1] "addition of two numbers 16 and 3 is: 19"
> print(paste("Subtracting Number 3 from 16 is : ", sub))
[1] "Subtracting Number 3 from 16 is :  13"
> print(paste("Multiplication of two numbers 16 and 3 is : ", multi))
[1] "Multiplication of two numbers 16 and 3 is :  48"
> print(paste("Division of two numbers 16 and 3 is : ", division))
[1] "Division of two numbers 16 and 3 is :  5.33333333333333"
> print(paste("Integer Division of two numbers 16 and 3 is : ", Integer_Division))
```

2. Display a String on R Console.

program

```
print("hello, world!")
```

```
> print("hello, world!")
```

Output:

```
[1] "hello, world!"
```

3.Declare Variables In R And Also Write The Commands For Retrieving The Value Of The Stored Variables In R Console.

Program:

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

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

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

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

output:

```
> #assignment using equal operator.
> var.1=c(0,1,2,3)
>
> #assignment using leftward operator.
> var.2<-c("learn","R")
```

```
>
> #assgnment using rightward operator.
> c(TRUE,1)-> var.3
>
> print(var.1)
[1] 0 1 2 3
> cat("var.1 is", var.1,"\n")
var.1 is 0 1 2 3
> cat("var.2 is", var.2,"\n")
var.2 is learn R
> cat("var.3 is", var.3,"\n")
var.3 is 1 1
```

4. Write R script to calculate the area of Rectangle.

Program:

```
length <- 5
width <- 10

area <- length * width

cat("the area of the rectangle is",area,"\n")
```

output:

```
> length <- 5
> width <- 10
```

```
>
> area <- length * width
>
> cat("the area of the rectangle is",area,"\n")
the area of the rectangle is 50
```

5.Write Commands In R Console To Determine The Type Of Variable

program

```
x <- "ms dhoni"
typeof(x)

x <- 5
typeof(x)
```

output:

```
> x <- "ms dhoni"
> typeof(x)
[1] "character"
>
> x <- 5
> typeof(x)
[1] "double"
```

6.Enumerate The Process To Check Whether A Given Input Is Numeric , Integer , Double, Complex in R.

Program

```
x <- 3.14
```

```
is.numeric(x)

x <- 5L
is.integer(x)

x <-343.3
is.double(x)

x <- 2+4i
is.complex(x)
```

output:

```
> x <- 3.14
> is.numeric(x)
[1] TRUE
>
> x <- 5L
> is.integer(x)
[1] TRUE
>
> x <-343.3
> is.double(x)
[1] TRUE
>
> x <- 2+4i
```

```
> is.complex(x)
[1] TRUE
```

7. Illustration of Vector Arithmetic.

Program:

```
v1<-c(1,2,3)
v2<-c(4,5,6)

# vector addition

v3<- v1+v2
print(v3)

# vector sub
v4 <- v1-v2
print(v4)

#vector mul

v5 <- v1 * v2
print(v5)
```

output:

```
> v1<-c(1,2,3)
> v2<-c(4,5,6)
```

```
>

> # vector addition

>

> v3<- v1+v2

> print(v3)

[1] 5 7 9

>

> # vector sub

> v4 <- v1-v2

> print(v4)

[1] -3 -3 -3

>

> #vector mul

>

> v5 <- v1 * v2

> print(v5)

[1]  4 10 18
```

8. Write an R Program to Take Input From User.

Input name as “Jack” and age as 17.

The program should display the output as

“Hai , Jack next year you will be 18 years old”

Program:

```
# Prompt the user to enter their name and age
```

```
name <- readline(prompt = "Please enter your name: ")

age <- readline(prompt = "Please enter your age: ")


# Convert the age to a numeric value

age <- as.numeric(age)


# Add one to the age to calculate the next year's age

next_year_age <- age + 1


# Display the message to the user

message(paste("Hi,", name, "next year you will be", next_year_age, "years old."))
```

output:

```
>"Hi,", name, "next year you will be", next_year_age, "years old."
```

DATA STRUCTURES IN R

1) Perform Matrix Addition & Subtraction in R

Program:

```
# Create two matrices
mat1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
mat2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)

# Add the matrices
mat_sum <- mat1 + mat2

# Subtract the matrices
mat_diff <- mat1 - mat2
```

```
# Print the results
print(mat_sum)
print(mat_diff)
```

Output:

```
> # Create two matrices
> mat1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
> mat2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)
>
> # Add the matrices
> mat_sum <- mat1 + mat2
>
> # Subtract the matrices
> mat_diff <- mat1 - mat2
>
> # Print the results
> print(mat_sum)
     [,1] [,2]
[1,]    6   10
[2,]    8   12
> print(mat_diff)
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
>



>
```

2) Perform Scalar multiplication and matrix multiplication in R

Program:

```
# Create a matrix

mat <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)


# Perform scalar multiplication

scalar_mult <- 2 * mat
```

Perform matrix multiplication

mat_mult <- mat %*% mat

Print the results

print(scalar_mult)

print(mat_mult)

output:

```
> # Create a matrix
> mat <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
>
> # Perform scalar multiplication
> scalar_mult <- 2 * mat
>
> # Perform matrix multiplication
> mat_mult <- mat %*% mat
>
> # Print the results
> print(scalar_mult)
     [,1] [,2]
[1,]    2    6
[2,]    4    8
> print(mat_mult)
     [,1] [,2]
[1,]    7   15
[2,]   10   22
>


>
```

3) Find Transpose of matrix in R.

Program:

```
# Create a matrix

mat <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)


# Find the transpose of the matrix

mat_transpose <- t(mat)


# Print the results

print(mat)

print(mat_transpose)
```

output:

```
> # Create a matrix
> mat <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
>
> # Find the transpose of the matrix
> mat_transpose <- t(mat)
>
> # Print the results
> print(mat)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> print(mat_transpose)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
>



>
```

4) Perform the operation of combining matrices in R using cbind() and rbind()

functions.

Program:

```
# Create two matrices

mat1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)

mat2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)


# Combine the matrices horizontally (column-wise)

mat_combined1 <- cbind(mat1, mat2)


# Combine the matrices vertically (row-wise)

mat_combined2 <- rbind(mat1, mat2)


# Print the results

print(mat_combined1)

print(mat_combined2)
```

output:

```
> # Create two matrices
> mat1 <- matrix(c(1, 2, 3, 4), nrow = 2, ncol = 2)
> mat2 <- matrix(c(5, 6, 7, 8), nrow = 2, ncol = 2)
>
> # Combine the matrices horizontally (column-wise)
> mat_combined1 <- cbind(mat1, mat2)
>
> # Combine the matrices vertically (row-wise)
> mat_combined2 <- rbind(mat1, mat2)
>
> # Print the results
> print(mat_combined1)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> print(mat_combined2)
     [,1] [,2]
```

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


>
```

5) Deconstruct a matrix in R

Program:

```
# Create a matrix

mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)


# Extract the elements of the matrix

elements <- mat[1:6]


# Extract the rows of the matrix

rows <- mat[c(1, 2), ]


# Extract the columns of the matrix

columns <- mat[, c(2, 3)]


# Print the results

print(mat)

print(elements)

print(rows)
```

print(columns)

output:

```
> # Create a matrix
> mat <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)
>
> # Extract the elements of the matrix
> elements <- mat[1:6]
>
> # Extract the rows of the matrix
> rows <- mat[c(1, 2), ]
>
> # Extract the columns of the matrix
> columns <- mat[, c(2, 3)]
>
> # Print the results
> print(mat)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> print(elements)
[1] 1 2 3 4 5 6
> print(rows)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> print(columns)
     [,1] [,2]
[1,]    3    5
[2,]    4    6
>
```

6) Perform array manipulation in R .

Program:

Create the vectors with different length

vector1 <- c(1, 2, 3)

vector2 <- c(10, 15, 3, 11, 16, 12)

taking this vector as input

result <- array(c(vector1, vector2), dim = c(3, 3, 2))

print(result)

output:

```
# Create the vectors with different length
> vector1 <- c(1, 2, 3)
> vector2 <- c(10, 15, 3, 11, 16, 12)
>
> # taking this vector as input
> result <- array(c(vector1, vector2), dim = c(3, 3, 2))
> print(result)
, , 1

     [,1] [,2] [,3]
[1,]    1   10   11
[2,]    2   15   16
[3,]    3    3   12

, , 2

     [,1] [,2] [,3]
[1,]    1   10   11
[2,]    2   15   16
[3,]    3    3   12

>



>
```

7) Perform calculations across array elements in an array using the apply() function.

Program:

Create a matrix

mat <- matrix(1:9, nrow = 3)

```
# Apply the sum function to each row of the matrix

row_sums <- apply(mat, 1, sum)


# Print the row sums

print(row_sums)

# Create a matrix

mat <- matrix(rnorm(16), nrow = 4)


# Apply the mean function to each column of the matrix

col_means <- apply(mat, 2, mean)


# Print the column means

print(col_means)
```

output

```
> # Create a matrix
> mat <- matrix(1:9, nrow = 3)
>
> # Apply the sum function to each row of the matrix
> row_sums <- apply(mat, 1, sum)
>
> # Print the row sums
> print(row_sums)
[1] 12 15 18
> # Create a matrix
> mat <- matrix(rnorm(16), nrow = 4)
>
> # Apply the mean function to each column of the matrix
> col_means <- apply(mat, 2, mean)
>
> # Print the column means
> print(col_means)
[1]  0.05163526  0.22976812 -0.05427408  0.05352512
>
```

```
>
```

8) Demonstrate Factor data structure in R.

Program:

```
# Create a vector of categorical data
grades <- c("A", "B", "A", "C", "B", "B", "A")

# Convert the vector to a factor
grades_factor <- factor(grades)

# Print the levels of the factor
print(levels(grades_factor))

# Print the counts of each level
print(table(grades_factor))

# Rename the levels of the factor
levels(grades_factor) <- c("Excellent", "Good", "Fair")

# Print the renamed levels
print(levels(grades_factor))

# Print the counts of each renamed level
print(table(grades_factor))
```

output:

```
> # Create a vector of categorical data
> grades <- c("A", "B", "A", "C", "B", "B", "A")
>
> # Convert the vector to a factor
> grades_factor <- factor(grades)
>
> # Print the levels of the factor
> print(levels(grades_factor))
[1] "A" "B" "C"
>
> # Print the counts of each level
> print(table(grades_factor))
grades_factor
A B C
3 3 1
>
> # Rename the levels of the factor
> levels(grades_factor) <- c("Excellent", "Good", "Fair")
>
> # Print the renamed levels
> print(levels(grades_factor))
[1] "Excellent" "Good"      "Fair"
>
> # Print the counts of each renamed level
> print(table(grades_factor))
grades_factor
Excellent      Good      Fair
        3         3         1
>



>
```

9) Create a data frame and print the structure of the data frame in R.

Program:

```
# Create a data frame with two columns

df <- data.frame(name = c("Alice", "Bob", "Charlie"), age = c(25, 30, 35))



# Print the structure of the data frame

str(df)
```

output

```
> # Create a data frame with two columns
> df <- data.frame(name = c("Alice", "Bob", "Charlie"), age = c(25, 30, 35))
>
> # Print the structure of the data frame
> str(df)
'data.frame':	3 obs. of  2 variables:
 $ name: chr  "Alice" "Bob" "Charlie"
 $ age : num  25 30 35
>



>
```

10) Demonstrate the creation of S3 class in R.

Program:

```
# Define a custom S3 class

myclass <- function(x) {

  class(x) <- c("myclass", class(x))

  x

}


# Create an object of the custom class

myobject <- myclass(5)


# Print the class of the object

class(myobject)


# Define a method for the custom class

print.myclass <- function(x, ...) {

  cat("This is an object of class 'myclass':\n")
```

```
  print(as.numeric(x))

}

# Call the method on the object

print(myobject)
```

output

```
> # Define a custom S3 class
> myclass <- function(x) {
+     class(x) <- c("myclass", class(x))
+     x
+ }
>
> # Create an object of the custom class
> myobject <- myclass(5)
>
> # Print the class of the object
> class(myobject)
[1] "myclass" "numeric"
>
> # Define a method for the custom class
> print.myclass <- function(x, ...) {
+     cat("This is an object of class 'myclass':\n")
+     print(as.numeric(x))
+ }
>
> # Call the method on the object
> print(myobject)
This is an object of class 'myclass':
[1] 5
>



>
```

11) Demonstrate the creation of S4 class in R.

Program:

```
# Define a custom S4 class
```

```
setClass("Person",
  representation(
    name = "character",
    age = "numeric"
  )
)

# Create an object of the custom class
person <- new("Person", name = "Alice", age = 25)

# Print the object
person

# Define a method for the custom class
setMethod("print", "Person",
  function(x) {
    cat("Name: ", x@name, "\n")
    cat("Age: ", x@age, "\n")
  }
)

# Call the method on the object
print(person)
```

output

```
> # Define a custom S4 class
> setClass("Person",
+          representation(
+              name = "character",
+              age = "numeric"
+          )
+ )
>
> # Create an object of the custom class
> person <- new("Person", name = "Alice", age = 25)
>
> # Print the object
> person
An object of class "Person"
Slot "name":
[1] "Alice"

Slot "age":
[1] 25

>
> # Define a method for the custom class
> setMethod("print", "Person",
+           function(x) {
+               cat("Name: ", x@name, "\n")
+               cat("Age: ", x@age, "\n")
+           }
+ )
>
> # Call the method on the object
> print(person)
Name:  Alice
Age:  25
>



>
```

12) Demonstrate the creation of Reference class in R by defining a class called students with fields – Name, Age , GPA. Also illustrate how the fields of the object can be accessed using the $ operator. Modify the Name field by reassigning the name to Paul.

Program:

```
# Define a Reference Class called "students"

students <- setRefClass("students",

  fields = list(
```

```
    Name = "character",
    Age = "numeric",
    GPA = "numeric"
  )
)

# Create an object of the Reference Class
s <- students(Name = "John", Age = 20, GPA = 3.5)

# Access fields using the $ operator
s$Name # "John"
s$Age # 20
s$GPA # 3.5

# Modify the Name field
s$Name <- "Paul"
s$Name # "Paul"
```

output:

```
> # Define a Reference Class called "students"
> students <- setRefClass("students",
+                         fields = list(
+                           Name = "character",
+                           Age = "numeric",
+                           GPA = "numeric"
+                         )
+ )
>
> # Create an object of the Reference Class
> s <- students(Name = "John", Age = 20, GPA = 3.5)
>
> # Access fields using the $ operator
```

```
> s$Name # "John"
[1] "John"
> s$Age # 20
[1] 20
> s$GPA # 3.5
[1] 3.5
>
> # Modify the Name field
> s$Name <- "Paul"
> s$Name # "Paul"
[1] "Paul"
>



>
```