# DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

## R Programming Lab Manual

Department of Artificial Intelligence and Machine Learning
DAYANANDA SAGAR ACADEMY OF TECHNOLOGY AND MANAGEMENT

Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore-560 082
(Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Delhi) CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi,

NAAC A+
2022-2023

List of R lab programs

1. Illustrate the if-else statement and how it operates on vectors of variable length.
2. Illustrate for loop and stop on condition. Print the error message.
3. Illustrate while loop and stop on condition. Print the error message.
4. Apply the predefined mean function using two-dimension data set.
5. Apply for ragged array apply.
6. Compute mean values for vector aggregates defined by factors tapply and supply.
7. Compute the sum of each list component and return the result as a list.
8. Compute the sum of each vector component and return the result as a list.
9. Illustrate return and break.
10. Illustrate for computing the mean for any custom selection of columns without compromising the speed performance.
11. Illustrate the function in R programming.
12. Illustrate the function with an optional argument.
13. Download and install R-Programming environment and install basic packages using install.packages() command in R.
14. Learn all the basics of R-Programming (Data types, Variables, Operators etc,.)
15. Write a program to find list of even numbers from 1 to n using R-Loops.
16. Create a function to print squares of numbers in sequence.
17. Write a program to join columns and rows in a data frame using cbind() and rbind() in R.
18. Implement different String Manipulation functions in R.
19. Implement different data structures in R (Vectors, Lists, Data Frames)
20. Write a program to read a csv file and analyze the data in the file in R.

Reference Books:


1. Sams Teach Yourself R Andy Nicholls,RichardPugh,AimeeGott,Pearson Publication.
2. www.statmethods.net/about/books.html
3. W.J. Owen, The R Guide
4. D. Rossiter, Introduction to the R Project for Statistical Computing for Use at the ITC
5. W.N. Venebles& D. M. Smith, An Introduction to R**
6. M. Crawley, Basic Statistics: An Introduction using R
7. P. Dalgaard, Introductory Statistics with R

Lab Manual:

1. **Illustrate the if-else statement and how it operates on vectors of variable length.**

   **Solution:**

```
# Example code using if-else statement on vectors of variable length

vector1 <- c(1, 2, 3, 4, 5)
vector2 <- c(6, 7, 8, 9)
vector3 <- c(10, 11)

# Assuming we want to compare the lengths of the vectors and perform
different operations based on the results

# Case 1: If vector1 is longer than vector2
if (length(vector1) > length(vector2)) {
  result <- vector1 * 2  # Multiply each element of vector1 by 2
  print("Result:", result)
}
# Case 2: If vector1 is shorter than vector2
else if (length(vector1) < length(vector2)) {
  result <- vector2 + 5  # Add 5 to each element of vector2
  print("Result:", result)
}
# Case 3: If vector1 and vector2 have the same length
else {
  result <- vector1 - vector2  # Subtract corresponding elements of
vector1 and vector2
  print("Result:", result)
}
```

2. **Illustrate for loop and stop on condition. Print the error message.**
   **Solution:**

   **vector <- c(1, 2, 3, 4, 5)**

```
# Iterate over each element in the vector
for (i in vector) {
  # Check if the element is greater than 3
  if (i > 3) {
    stop("Error: Element greater than 3 found!")  # Print error
message and stop the loop
  } else {
    print(i)  # Print the element if it's not greater than 3
  }
}
```

3. **Illustrate while loop and stop on condition. Print the error message.**
   **Solution:**

```
# Example code using while loop and stop on condition

counter <- 1

# Execute the loop while the counter is less than or equal to 5
while (counter <= 5) {
  # Check if the counter is equal to 3
  if (counter == 3) {
    stop("Error: Counter value is 3!")  # Print error message and stop the
loop
  } else {
    print(counter)  # Print the counter value if it's not equal to 3
  }

  counter <- counter + 1  # Increment the counter
}
```

4. **Apply the predefined mean function using two dimension data set.**
   **Solution:**

```
# Create a two-dimensional dataset
dataset <- matrix(c(1, 2, 3, 4, 5, 6, 7, 8, 9), nrow = 3, ncol = 3)
```

```
# Calculate the mean of the entire dataset
mean_value <- mean(dataset)
print(mean_value)

# Calculate the mean along the rows (axis 1)
row_means <- apply(dataset, 1, mean)
print(row_means)

# Calculate the mean along the columns (axis 2)
col_means <- apply(dataset, 2, mean)
print(col_means)
```

5. **Apply for ragged array apply.**
   **Solution:**

```
lapply()

# Create a ragged array
ragged_array <- list(
  list(1, 2, 3),
  list(4, 5),
  list(6, 7, 8, 9)
)

# Define a custom function to calculate the sum of a sublist
sum_sublist <- function(sublist) {
  sum_val <- sum(sublist)
  return(sum_val)
}

# Apply the sum_sublist function to each sublist using lapply()
result <- lapply(ragged_array, sum_sublist)
print(result)
```

6. **Compute mean values for vector aggregates defined by factors tapply and supply.**
   **Solution:**

```
defined by factors

# Create a vector
values <- c(12, 15, 10, 8, 20, 16)
```

```
# Create a factor defining groups
groups <- factor(c("A", "B", "A", "B", "B", "A"))

# Using tapply() function
tapply_result <- tapply(values, groups, mean)
print(tapply_result)

# Using aggregate() function
data <- data.frame(values, groups)
aggregate_result <- aggregate(values ~ groups, data, mean)
print(aggregate_result)
```

7. **Compute the sum of each list component and return the result as a list.**
   **Solution:**

```
# Create a list
my_list <- list(a = c(1, 2, 3),
                b = c(4, 5, 6),
                c = c(7, 8, 9))

# Compute the sum of each component using lapply()
result <- lapply(my_list, sum)
print(result)
```

8. **Compute the sum of each vector component and return the result as a list.**
   **Solution:**

```
# Create a vector
my_vector <- c(1, 2, 3, 4, 5)

# Compute the sum of each component using lapply()
```

```
result <- lapply(my_vector, sum)
print(result)
```

9. **Illustrate return and break.**
   **Solution:**

   1) **Return:**

```
# Code illustrating the return statement

my_function <- function(a, b) {
  if (a > b) {
    return(a - b)
  } else {
    return(b - a)
  }
}

result <- my_function(5, 3)
print(result)
```

   2) **Break:**

```
# Code illustrating the break statement

my_vector <- c(1, 2, 3, 4, 5)

for (num in my_vector) {
  if (num == 3) {
    break
  }
  print(num)
}
```

10. **Illustrate for computing the mean for any custom selection of columns without compromising the speed performance.**
    **Solution:**

```
# Code for computing the mean of custom selection of columns without
# compromising speed performance
```

```
# Create a matrix
data <- matrix(1:1000000, nrow = 1000, ncol = 1000)

# Select the columns for which you want to compute the mean
selected_columns <- c(100, 500, 700)

# Compute the mean for the selected columns
mean_values <- colMeans(data[, selected_columns])

# Print the mean values
print(mean_values)
```

**11. Illustrate the function in R programming.**

**Solution:**

```
# Define a function to calculate the square of a number
square <- function(x) {
  result <- x^2
  return(result)
}

# Call the function and store the result
num <- 5
square_result <- square(num)

# Print the result
print(square_result)
```

**12. Illustrate the function with an optional argument.**

**Solution:**

```
# Define a function to calculate the power of a number
power <- function(x, exponent = 2) {
  result <- x^exponent
  return(result)
```

```
}

# Call the function without specifying the exponent argument
num <- 3
power_result <- power(num)

# Print the result
print(power_result)

# Call the function with a custom exponent argument
custom_power_result <- power(num, 3)

# Print the result
print(custom_power_result)
```

**13. Download and install R-Programming environment and install basic packages using install.packages() command in R.**

**Solution**:

```
# Install the "dplyr" package for data manipulation
install.packages("dplyr")

# Install the "ggplot2" package for data visualization
install.packages("ggplot2")

# Install the "readr" package for data import
install.packages("readr")

# Install the "tidyr" package for data tidying
install.packages("tidyr")

# Install the "stringr" package for string manipulation
install.packages("stringr")
```

**14. Learn all the basics of R-Programming (Data types, Variables, Operators etc,.)**

**Solution:**

```
# Numeric data type
numeric_var <- 10
print(numeric_var)

# Character data type
```

```
character_var <- "Hello, World!"
print(character_var)

# Logical data type
logical_var <- TRUE
print(logical_var)

# Vector data type
vector_var <- c(1, 2, 3, 4, 5)
print(vector_var)

# Variable assignment and reassignment
x <- 5
y <- 3
z <- x + y
print(z)

# Arithmetic operators
addition <- x + y
subtraction <- x - y
multiplication <- x * y
division <- x / y
exponentiation <- x^y
modulo <- x %% y
print(addition)
print(subtraction)
print(multiplication)
print(division)
print(exponentiation)
print(modulo)
```

**15.** **Write a program to find list of even numbers from 1 to n using R-Loops.**

**Solution:**

```
# Function to find even numbers
find_even_numbers <- function(n) {
  even_numbers <- c()  # Empty vector to store even numbers
```

```
  for (i in 1:n) {
    if (i %% 2 == 0) {  # Check if number is even
      even_numbers <- c(even_numbers, i)  # Add even number to
the vector
    }
  }

  return(even_numbers)
}

# Enter the value of n
n <- 10

# Call the function to find even numbers from 1 to n
even_numbers_list <- find_even_numbers(n)

# Print the list of even numbers
print(even_numbers_list)
```

16. **Create a function to print squares of numbers in sequence.**

**Solution:**

```
# Function to print squares of numbers in a sequence
print_squares <- function(start, end) {
  for (num in start:end) {
    square <- num^2
    print(square)
  }
}

# Main program
start <- 1
end <- 5

print_squares(start, end)
```

**17. Write a program to join columns and rows in a data frame using cbind() and rbind() in R.**

**Solution:**

```
# Create sample data frames
df1 <- data.frame(A = 1:5, B = 6:10)
df2 <- data.frame(A = 11:15, B = 16:20)
```

```
# Join columns using cbind()
combined_df <- cbind(df1, df2)
print(combined_df)

# Join rows using rbind()
combined_rows <- rbind(df1, df2)
print(combined_rows)
```

**18.** **Implement different String Manipulation functions in R.**

**Solution:**

```
# String manipulation functions in R

# nchar(): Returns the number of characters in a string
string <- "Hello, World!"
char_count <- nchar(string)
print(char_count)

# tolower(): Converts a string to lowercase
lowercase_string <- tolower(string)
print(lowercase_string)

# toupper(): Converts a string to uppercase
uppercase_string <- toupper(string)
print(uppercase_string)

# substr(): Extracts a substring from a string
substring <- substr(string, start = 7, stop = 12)
print(substring)

# strsplit(): Splits a string into substrings based on a specified delimiter
split_string <- strsplit(string, split = ", ")
print(split_string)

# paste(): Concatenates multiple strings together
string1 <- "Hello"
string2 <- "World!"
concatenated_string <- paste(string1, string2)
print(concatenated_string)

# gsub(): Replaces occurrences of a pattern in a string with a new value
replaced_string <- gsub("Hello", "Hi", string)
print(replaced_string)
```

**19.** **Implement different data structures in R (Vectors, Lists, Data Frames)**

**Solutions:**

```
# Vectors
vector1 <- c(1, 2, 3, 4, 5)   # Numeric vector
vector2 <- c("apple", "banana", "orange")   # Character vector
vector3 <- c(TRUE, FALSE, TRUE)   # Logical vector

# Lists
list1 <- list(1, "apple", TRUE)   # List with different data types
list2 <- list(a = 10, b = "banana", c = FALSE)   # List with named
elements

# Data Frames
df <- data.frame(Name = c("John", "Emily", "Michael"),
                 Age = c(25, 30, 28),
                 Grade = c("A", "B", "A+"),
                 stringsAsFactors = FALSE)   # Data frame with
columns

# Accessing elements in data structures
# Vectors
print(vector1[3])   # Accessing the third element of vector1
print(vector2[2:3])   # Accessing the second and third elements of
vector2

# Lists
print(list1[[2]])   # Accessing the second element of list1
print(list2$b)   # Accessing the element with name "b" in list2

# Data Frames
print(df$Age)   # Accessing the "Age" column of df
print(df[1, ])   # Accessing the first row of df

# Adding elements to lists
list1[[4]] <- 10   # Adding a numeric element to list1
list2$d <- TRUE   # Adding a named logical element to list2

# Modifying elements in data structures
vector1[2] <- 6   # Modifying the second element of vector1
list1[[3]] <- "orange"   # Modifying the third element of list1
df$Grade[2] <- "A-"   # Modifying the second element of the
"Grade" column in df

# Printing data structures
print(vector1)
```

```
print(list1)
print(df)
```

**20. Write a program to read a csv file and analyze the data in the file in R.**

**Solution:**

```
Read CSV file
data <- read.csv("data.csv")

# Display the structure of the data
str(data)

# Display the summary statistics of numeric columns
summary(data$NumericColumn)

# Calculate the mean of a numeric column
mean_value <- mean(data$NumericColumn)
print(paste("Mean:", mean_value))

# Calculate the maximum value of a numeric column
max_value <- max(data$NumericColumn)
print(paste("Maximum Value:", max_value))

# Calculate the minimum value of a numeric column
min_value <- min(data$NumericColumn)
print(paste("Minimum Value:", min_value))

# Count the frequency of values in a categorical column
frequency <- table(data$CategoricalColumn)
print("Frequency of CategoricalColumn:")
print(frequency)
```