1. Write an R program for different types of data structures in R.

→
```
my_vector <- c(1, 2, 3, 4, 5)
print("Numeric vector")
print(my_vector)
cv <- c("apple", "banana", "cherry")
print("character vector")
print(cv)
my_matrix <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 2,
    byrow = TRUE)
print("Matrix")
print(my_matrix)
my_list <- list(name = "Tony", age = 20, scores = c(80, 90, 95))
print("List")
print(my_list)
my_data <- data.frame(name = c("Tony", "Yusra", "Rumi",
"Madiha"), age = c(25, 30, 35, 27), scores = c
(80, 90, 95, 90))
print("Data Frame")
print(my_data)
class(my_vector)
```

Output :-

```
[1] "Numeric Vector"
[1] 1 2 3 4 5
[1] "Character Vector"
[1] "apple", "banana", "cherry"
[1] "Matrix"
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[1] "List"
$name
[1] "Tony"
$age
[1] 20
$scores
[1] 80 90 95
[1] "Data Frame"
  name  age scores
1 Tony   25     80
2 Yusra  30     90
3 Rumi   35     85
4 Madiha 27     90
```

2. Write an R program that includes variables, constants, data types.

→
```
name <- "Taniya"
age <- 20
score <- 95.5
PI <- 3.14159
GRAVITY <- 9.81
is_student <- TRUE
grades <- c(80, 90, 85)
student_info <- list(name = "Taniya", age = 20, scores = grades)
cat("Data type of 'name' :", (typeof(name)), "\n")
cat("Data type of 'age':", (typeof(age)), "\n")
cat("Data type of 'score':", (typeof(score)), "\n")
cat("Data type of 'is-student':", (typeof(is-student)), "\n")
cat("Data type of 'grades':", (typeof(grades)), "\n")
cat("Data type of 'student-info':", (typeof(student_info)),
"\n")

cat("constant value of 'PI' :", PI, "\n")
cat("constant value of 'GRAVITY' :", GRAVITY, "\n")
```

Output :-

Data type of 'name' : character
Data type of 'age' : double
Data type of 'score' : double
Data type of 'is_student' : logical
Data type of 'grades' : double
Data type of 'student_info' : list
Constant value of 'PI' : 3.14159
Constant value of 'GRAVITY' : 9.81

3. Write an R program that include different operators, control structures, default values for arguments, returning complex objects.

→
```
calculate_area <- function ( radius = 5, shape = 'circle')
{
if ( shape == 'circle')
{
area <- pi * radius * 2
}
 else
cat (" Unsupported shape", shape, \n")
return ( NULL)
}
msg <- if else (area > 10, "Large Area", "Small Area")
if (area > 10)
  msg <- " large area"
else
   msg <- " small area"
return ( list (shape = shape, radius = radius , area = area,
        Msg = msg))
}
 circle_result <- calculate_area()
 Square_result <- calculate_area (radius=4, shape='Square')
 cat ("Circle result ", "\n")
 print ( circle_result)
 cat (" square result", "\n")
 print ( square_result)
```

Output :-

Unsupported shape square

Circle result

$ shape

[1] "circle"

$ radius

[1] 5

$ area

[1] 31.41593

$ Msg

[1] "large Area"

Square Result

NULL

4. Write an R program for quick sort implementation, binary search tree.

→
```
quickSort <- function (arr)
{
if (length(arr) <= 1)
{
return (arr)
}
pivot <- arr [1]
smaller <- arr [arr< pivot]
equal <- arr [arr == pivot]
greater <- arr [ arr > pivot]
return (c(quickSort (smaller), equal, quicksort (greater)))
}
print (" Quick sort")
print (" Before sort")
my_array <- c [ 5, 2, 8, 3, 1, 9)
print (my-array)
sorted_array <- quicksort (my-array)
print (" After sort")
print (" sorted_array")
Node <- function (value)
{
list (
value= value,
left = NULL,
right = NULL)
}
```

```
insert <- function (root, value)
{
  if (is.null (root))
{
  return (Node (value))
}
  if (value < root$value)
{
  root$left <- insert (root$left, value)
}
  else if
  (value > root $value)
{
  root $right <- insert (root $right, value)
}
  return (root)
}
  inorder <- function (root)
{
  if (!is.null (root))
{
  inorder [root $left)
     print (root $ value)
     inorder ( root $right)
}
}
  print ("Binary search tree")
  my-tree <- NULL
```

```
keys <- c(5,2,8,3,1,9)
for (key in keys)
{
  my_tree <- insert(my_tree, key)
}
  inorder(my_tree)
```

Output :-

```
[1] "Quick sort"
[1] "Before sort"
[1] 5 2 8 3 1 9
[1] "After sort"
[1] 1 2 3 5 8 9
[1] "Binary search tree"
[1] 1
[1] 2
[1] 3
[1] 5
[1] 8
[1] 9
```

5. Write an R program for calculating Cumulative sums and products, minima, maxima and calculus.

→
```
numbers <- c(2, 4, 1, 8, 5, 7)
cumulative_sum <- cumsum(numbers)
cat("cumulative sum:", cumulative_sum, "\n")
cumulative_product <- cumprod(numbers)
cat("cumulative product:", cumulative_product, "\n")
minimum_value <- min(numbers)
maximum_value <- max(numbers)
cat("Minimum value", minimum_value, "\n")
cat("Maximum value", maximum_value, "\n")
differentiate <- diff(numbers)
cat("Differentiation (First Difference):", differentiate, "\n")
integrate <- cumsum(numbers)
cat("Integration (cumulative sum):", integrate, "\n")
```

Output :-

Cumulative Sum : 2 6 7 15 20 27
Cumulative product : 2 8 8 64 320 2240
Minimum value 1
Maximum value 8
Differentiation (First Difference) : 2 -3 7 -3 2
Integration (cumulative sum) : 2 6 7 15 20 27

6. Write an R program for finding stationary distribution of markovchains.

→
```
install.packages("markovchain")
library(markovchain)
transition_matrix <- matrix(c(0.7, 0.3, 0.2, 0.8),
        nrow = 2, byrow = TRUE)
mc <- new("markovchain", states = c("state 1", "state 2"),
        transitionMatrix = transition_matrix)
stationary_dist <- steadyStates(mc)
print(stationary_dist)
```

Output :-

	State 1	State 2
[1.]	0.4	0.6

7. Write an R program that includes linear algebra operations on vectors and matrices.

→
```
vector1 <- c(1,2,3)
vector2 <- c(4,5,6)
vector_sum <- vector1 + vector2
cat("vector Addition :", "\n")
cat(vector_sum)
vector_diff <- vector1 - vector2
cat("\n vector Substraction :", "\n")
cat(vector_diff)
Scalar <- 2
vector_scalar_product <- scalar * vector1
cat("\n vector scalar product : \n")
cat(vector_scalar_product)
matrix1 <- matrix(c(1,2,3,4), nrow=2, ncol=2)
matrix2 <- matrix(c(5,6,7,8), nrow=2, ncol=2)
matrix_sum <- matrix1 + matrix2
cat("\n Matrix Addition : \n")
cat(matrix_sum)
matrix_diff <- matrix1 - matrix2
cat("\n Matrix substraction : \n")
cat(matrix_diff)
matrix_product <- matrix1 %*% (matrix2)
cat("\n Matrix Multiplication : \n")
cat(matrix_product)
matrix_det <- det(matrix1)
cat("\n Matrix Determinant : \n")
cat(matrix_det)
```

```
matrix_transpose <- t(matrix1)
cat("\n Matrix transpose :\n")
cat(matrix_transpose)
matrix_inv <- solve(matrix1)
print("\n Matrix inverse :")
print(matrix_inv)
```

Output :-

vector Addition :

5 7 9

vector subtraction :

-3 -3 -3

vector scalar product:

2 4 6

Matrix Addition :

6 8 10 12

Matrix Subtraction :

-4 -4 -4 -4

Matrix Multiplication :

23 34 31 46

Matrix Determinant :

-2

Matrix Transpose :

1 3 2 4 [1] " In Matrix Inverse :"

[,1] [,2]

[1,] -2 1.5

[2,] 1 -0.5

8 Write an R program for any virual representation of an object with creating graphs using graphic functions : Plot(), Hist(), Linechart(), Pie(), Boxplot(), Scatter plots().

→
```
data <- c (10, 15, 20, 25, 30, 35, 40, 45, 50)
plot (data, type = "l" main = "line chart", xlab = "x-axis",
  ylab = "y-axis")
hist ( data. main = "Histogram", xlab = "values", ylab =
  "Frequency")
categories <- c ( "A", "B", "C")
values <- c ( 30, 40, 50)
pie (values, labels = categories, main = "Pie chart", col = c
  ("red", "green", "blue"))
data <- list (A = c ( 2, 4, 6, 8) , B = c (1, 3, 5, 7))
boxplot ( data, main = "Boxplot", xlab = "Groups", ylab =
  "values")

x <- c (1, 2, 3, 4, 5)
y <- c (10, 20, 30, 20, 50)
plot ( x, y, pch = 19, col = "blue", main = "Scatter Plot",
  xlab = "x-axis", ylab = "y-axis")
```

9. Write an R program for with any dataset containing data frames objects, indexing and sub-setting data frames and emptoy manipulating and analyzing data.

→
```
emp_data <- data.frame(
    EmployeeId = c(1, 2, 3, 4, 5),
    FirstName = c("Taniya", "Yusra", "Fatima", "Rahaf", "Aysha"),
    LastName = c("Khairdari", "Mulla", "Arshi", "Damdi", "Taniya"),
    Age = c(30, 25, 28, 35, 32),
    Department = c("HR", "Marketing", "Finance", "HR", "IT"),
    Salary = c(50,000, 55,000, 60,000, 52,000, 70,000)
)

cat("Employee Data : \n")
print(emp_data)
cat("\n Subset and indexing")
hr_emp <- emp_data[emp_data$Department == "HR",]
cat("HR Employees : \n")
print(hr_emp)
old_emp <- emp_data[emp_data$Age >= 30,]
cat("Employee aged 30 or older : \n")
print(old_emp)
high_sal_emp <- emp_data[emp_data$Salary >= 55000,]
cat("Employee aged 30 or older : \n")
print(high_sal_emp)

cat("\n Data Manipulation and Analysis : \n")
avg_sal <- mean(emp_data$salary)
```

```
cat ("Average Salary:", avg-sal, "\n")
max-age <- max (emp-data $ Age)
cat ("\nMaximum Age :", max-age, "\n")
dept_count <- Table (emp-data $ Department)
cat ("\n Number of Employees in each Department")
print ( dept_count)
dept_payroll <- tapply (emp-data $ salary, emp-data
$ Department, sum)
cat ("Total payroll in each Department :\n")
print ( dept-payroll)
```

Output :-

Employee Data :

	Employee Id	FirstName	LastName	Age	Department	Salary
1	1	Taniya	Kharoori	30	HR	50,000
2	2	Yusra	Mulla	25	Marketing	55,000
3	3	Fatima	Asthi	28	Finance	60,000
4	4	Rahaf	Damda	35	HR	52,000
5	5	Aysha	Taniya	32	IT	70,000

Subset and Indexing HR Employees :-

	EmployeeId	FirstName	LastName	Age	Department	Salary
1	1	Taniya	Kharoori	30	HR	50,000
4	4	Rahaf	Damda	35	HR	52,000

Employee aged 30 or older :-

	EmployeeId	FirstName	LastName	Age	Department	Salary
1	1	Taniya	Kharoori	30	HR	50,000
4	4	Rahaf	Damda	35	HR	52,000
5	5	Aysha	Taniya	32	IT	70,000

Employee aged 30 or older :-

	EmployeeId	FirstName	LastName	Age	Department	Salary
2	2	Yusra	Mulla	25	Marketing	55,000
3	3	Fatima	Asthi	28	Finance	60,000
5	5	Aysha	Taniya	32	IT	70,000

Data Manipulation and Analysis :

Average Salary : 57,400

Maximum Age : 35

Number of Employee in each Department

Finance	HR	IT	Marketing
1	2	1	1

Total Payroll in Each Department :

Finance	HR	IT	Marketing
60,000	102000	70,000	55,000

10. Write a program to create any application of Linear Regression in multivariate context for predictive purpose.

→
```
data <-
data.frame ( Salary = c( 50,000, 60,000, 75000 , 80,000,
95,000 , 1,10,000 , 1,20,000 . 1,30,000 ),
Experience = c ( 1, 2, 3, 4, 5 . 6, 7, 8 ),
Education = c (12, 14, 16, 16, 18, 20, 20, 22)
)

model <- lm ( Salary - Experience + Education , data = data )
model
new_data <-
data.fram ( Experience = c ( 9,10 ), Education = c ( 22, 24 )
)

predicted_salaries <- predict ( model , new data = new_data )
cat ( " Predicted salaries:\n " )
print ( predicted_salaries )
```

Output :-

Predicted Salaries :

1	2
139333.3	152500.0