R Programming Lab Manual

V sem

BCA

Prepared by Muhammad Yousuf

Program Name	**B.C.A**	Semester	**V**
Course Title	**R Programming Lab**		
Course Code:	**DSC14-Lab**	No. of Credits	**02**
Contact hours	**04 Hours per week**	Duration of SEA/Exam	**1:30 hours**
Formative Assessment Marks	**25**	Summative Assessment Marks	**25**

Overview
The following program problematic comprises of R programming basics and application of several Statistical Techniques using it. The module aims to provide exposure in terms of Statistical Analysis, Hypothesis Testing, Regression and Correlation using R programming language.

Learning Objectives
The objective of this Laboratory to make students exercise the fundamentals of statistical analysis in R environment. They would be able to analysis data for the purpose of exploration using Descriptive and Inferential Statistics. Students will understand Probability and SamplingDistributions and learn the creative application of Linear Regression in multivariate context for predictive purpose.

Course Outcomes:

- Install, Code and Use R Programming Language in R Studio IDE to perform basic tasks on Vectors, Matrices and Data frames. Explore fundamentals of statistical analysis in R environment.
- Describe key terminologies, concepts and techniques employed in Statistical Analysis.
- Define Calculate, Implement Probability and Probability Distributions to solve a wide variety of problems.
- Conduct and interpret a variety of Hypothesis Tests to aid Decision Making.
- Understand, Analyse, and Interpret Correlation Probability and Regression to analyse the underlying relationships between different variables.

1. Write a R program for different types of data structures in R.
2. Write a R program that include variables, constants, data types.
3. Write a R program that include different operators, control structures, default values for arguments, returning complex objects.
4. Write a R program for quick sort implementation, binary search tree.
5. Write a R program for calculating cumulative sums, and products minima maxima and calculus.
6. Write a R program for finding stationary distribution of markanov chains.
7. Write a R program that include linear algebra operations on vectors and matrices.
8. Write a R program for any visual representation of an object with creating graphs using graphic functions: Plot(),Hist(),Linechart(),Pie(),Boxplot(),Scatterplots().
9. Write a R program for with any dataset containing data frame objects, indexing and subsetting data frames, and employ manipulating and analyzing data.
10. Write a program to create an any application of Linear Regression in multivariate context for predictive purpose.

1. Write a R program for different types of data structures in R.
2. Write a R program that include variables, constants, data types.
3. Write a R program that include different operators, control structures, default values for arguments, returning complex objects.
4. Write a R program for quick sort implementation, binary search tree.
5. Write a R program for calculating cumulative sums, and products minima maxima and calculus.
6. Write a R program for finding stationary distribution of markanov chains.
7. Write a R program that include linear algebra operations on vectors and matrices.
8. Write a R program for any visual representation of an object with creating graphs using graphic functions: Plot(),Hist(),Linechart(),Pie(),Boxplot(),Scatterplots().
9. Write a R program for with any dataset containing dataframe objects, indexing and subsetting
data frames, and employ manipulating and analyzing data.
10. Write a program to create an any application of Linear Regression in multivariate context for predictive purpose.

To run the program directly without downloading the r-studios/ r-software. Visit this website https://www.mycompiler.io/new/r

---

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

You can run this code in an R environment to see how these data structures work. Each data structure has its own use cases and properties, and you can perform various operations on them to manipulate and analyze data.

Program

```
# Vector
 my_vector <- c(1, 2, 3, 4, 5)
print(my_vector)
# List
my_list <- list(name = "John", age = 30, city = "New York")
print(my_list)

# Matrix
my_matrix <- matrix(1:6, nrow = 2, ncol = 3)
print(my_matrix)

# Data Frame
my_df <- data.frame(Name = c("Alice", "Bob", "Charlie"),
                    Age = c(25, 30, 22))
print(my_df)

# Array
my_array <- array(1:12, dim = c(2, 3, 2))
print(my_array)

# Factor
my_factor <- factor(c("High", "Low", "Medium", "High", "Low"))
print(my_factor)

# DataFrame with time-series
date <- as.Date(c("2023-01-01", "2023-01-02", "2023-01-03"))
value <- c(100, 110, 105)
df_time_series <- data.frame(Date = date, Value = value)
print(df_time_series)
```

Output

```
[1] 1 2 3 4 5
$name
[1] "John"

$age
[1] 30

$city
[1] "New York"

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
     Name Age
1   Alice  25
2     Bob  30
3 Charlie  22
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

[1] High   Low    Medium High
Low
Levels: High Low Medium
        Date Value
1 2023-01-01   100
2 2023-01-02   110
3 2023-01-03   105
```

2. Write a R program that include variables, constants, data types.

In this program, we define variables (e.g., name, age, height, is_student) and constants (e.g., PI, G). We also demonstrate different data types such as character vectors, integer vectors, double vectors, and logical vectors. The cat function is used to print the values of these variables, constants, and data types.	
Program	output

# Variables 	
name <- "Alice"
age <- 25
height <- 165.5
is_student <- TRUE
 	
# Constants 	
PI <- 3.14159265359
G <- 9.81
 	
# Data Types 	
char_vector <- c("apple", "banana", "cherry")
int_vector <- c(1, 2, 3, 4, 5)
double_vector <- c(1.5, 2.7, 3.0)
logical_vector <- c(TRUE, FALSE, TRUE, FALSE)
 	
# Print variables, constants, and data types 	
cat("Name:", name, "\n")
cat("Age:", age, "\n")
cat("Height:", height, "\n")
cat("Is Student:", is_student, "\n")
 	
cat("PI Constant:", PI, "\n")
cat("Gravity Constant:", G, "\n")
 	
cat("Character Vector:", char_vector, "\n")
cat("Integer Vector:", int_vector, "\n")
cat("Double Vector:", double_vector, "\n")
cat("Logical Vector:", logical_vector, "\n")
	Name: Alice
	Age: 25
	Height: 165.5
	Is Student: TRUE
	PI Constant: 3.141593
	Gravity Constant: 9.81
	Character Vector: apple banana cherry
	Integer Vector: 1 2 3 4 5
	Double Vector: 1.5 2.7 3
	Logical Vector: TRUE FALSE TRUE FALSE
	
	[Execution complete with exit code 0]

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

<table>
<tr><td colspan="2">This program defines a function calculate_area with default argument values and returns a complex object (a list). It also includes control structures (if-else statements and a for loop), logical operators, and demonstrates working with complex objects (lists of lists).</td></tr>
<tr><td>program</td><td>Output</td></tr>
<tr><td><pre>
# Function with default argument values
calculate_area <- function(shape = "circle", radius = 1, length = 1, width = 1) {
 if (shape == "circle") {
  area <- pi * radius^2
 } else if (shape == "rectangle") {
  area <- length * width
 } else {
  area <- 0
 }
 return(list(shape = shape, area = area))
}

# Calculate areas using the function
circle_area <- calculate_area("circle", radius = 5)
rect_area <- calculate_area("rectangle", length = 4, width = 6)
default_area <- calculate_area()

# Print the results
cat("Circle Area:", circle_area$area, "for shape:", circle_area$shape, "\n")
cat("Rectangle Area:", rect_area$area, "for shape:", rect_area$shape, "\n")
cat("Default Area:", default_area$area, "for shape:", default_area$shape, "\n")

# Conditional statements
grade <- 85
if (grade >= 90) {
 cat("A\n")
} else if (grade >= 80) {
 cat("B\n")
} else if (grade >= 70) {
 cat("C\n")
} else {
 cat("F\n")
}

# Loop
for (i in 1:5) {
 cat("Iteration:", i, "\n")
}

# Logical operators
is_sunny <- TRUE
is_warm <- TRUE

if (is_sunny && is_warm) {
 cat("It's a nice day!\n")
}

# Complex objects (list of lists)
student1 <- list(name = "Alice", age = 25)
student2 <- list(name = "Bob", age = 22)
students <- list(student1, student2)

# Accessing complex object elements
cat("First student's name:", students[[1]]$name, "\n")
cat("Second student's age:", students[[2]]$age, "\n")
</pre></td><td><pre>
Circle Area: 78.53982 for shape: circle
Rectangle Area: 24 for shape: rectangle
Default Area: 3.141593 for shape: circle
B
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Iteration: 5
It's a nice day!

[Execution complete with exit code 0]
</pre></td></tr>
</table>

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

The first part of the code implements the Quick Sort algorithm, and the second part implements a Binary Search Tree (BST) with insertion and in-order traversal to print elements in sorted order. You can modify and extend these implementations as needed.

Quick Sort Implementation:

```
# Quick Sort function
quick_sort <- function(arr) {
 if (length(arr) <= 1) {
  return(arr)
 }

 pivot <- arr[1]
 less <- arr[arr < pivot]
 equal <- arr[arr == pivot]
 greater <- arr[arr > pivot]

 return(c(quick_sort(less), equal,
quick_sort(greater)))
}

# Example usage
unsorted_array <- c(9, 7, 5, 11, 12, 2, 14, 3, 10, 6)
sorted_array <- quick_sort(unsorted_array)
cat("QUICK SORT is in Sorted Array :",
sorted_array, "\n")
```

Binary Search Tree (BST) Implementation:

```
# Define a Node structure for the Binary Search Tree
Node <- function(key) {
 return(list(key = key, left = NULL, right = NULL))
}

# Insert a value into the BST
insert <- function(root, key) {
 if (is.null(root)) {
  return(Node(key))
 }

 if (key < root$key) {
  root$left <- insert(root$left, key)
 } else if (key > root$key) {
  root$right <- insert(root$right, key)
 }

 return(root)
}

# In-order traversal to print BST elements in sorted order
inorder_traversal <- function(root) {
 if (!is.null(root)) {
  inorder_traversal(root$left)
  cat(root$key, " ")
  inorder_traversal(root$right)
 }
}

# Example usage
bst_root <- NULL
bst_root <- insert(bst_root, 10)
bst_root <- insert(bst_root, 5)
bst_root <- insert(bst_root, 15)
bst_root <- insert(bst_root, 3)
bst_root <- insert(bst_root, 7)
bst_root <- insert(bst_root, 12)
bst_root <- insert(bst_root, 18)

cat("BINARY SEARCH TREE >>>In-order Traversal
(Sorted Order): ")
inorder_traversal(bst_root)
```

Output :
QUICK SORT is in Sorted Array: 2 3 5 6 7 9 10 11 12 14

[Execution complete with exit code 0]

Output:
BINARY SEARCH TREE >>>In-order Traversal (Sorted Order): 3 5 7 10 12 15 18
[Execution complete with exit code 0]

5. Write a R program for calculating cumulative sums, and products minima maxima and calculus.

<table>
<tr><td colspan="2">In this program:

We calculate the cumulative sum and product of a vector.
We find the minimum and maximum values in the vector.
We perform basic calculus operations, including finding the derivative of a function and calculating the integral of a function over a specified range. To use the Deriv and pracma libraries, you may need to install and load them using install.packages and library functions.</td></tr>
<tr><td>Program</td><td>Output</td></tr>
<tr><td><pre>
# Create a sample vector
values <- c(1, 2, 3, 4, 5)

# Calculate cumulative sum
cumulative_sum <- cumsum(values)
cat("Cumulative Sum:", cumulative_sum, "\n")

# Calculate cumulative product
cumulative_product <- cumprod(values)
cat("Cumulative Product:", cumulative_product, "\n")

# Find the minimum and maximum values
min_value <- min(values)
max_value <- max(values)
cat("Minimum Value:", min_value, "\n")
cat("Maximum Value:", max_value, "\n")

# Basic calculus operations
# Define a function
f <- function(x) {
  return(2 * x^2 + 3 * x + 1)
}

# Calculate the derivative (first order)
library(Deriv)
derivative <- Deriv(f, "x")
cat("Derivative of 2x^2 + 3x + 1:", derivative(2), "\n")

# Calculate the integral
library(pracma)
integral <- integral(f, lower = 1, upper = 2)
cat("Integral of 2x^2 + 3x + 1 from 1 to 2:", integral, "\n")
</pre></td><td>Cumulative Sum: 1 3 6 10 15
Cumulative Product: 1 2 6 24 120
Minimum Value: 1
Maximum Value: 5
Error in library(Deriv) : there is no package called 'Deriv'
Execution halted

[Execution complete with exit code 1]</td></tr>
</table>

6. Write a R program for finding stationary distribution of markanov chains.

Finding the stationary distribution of a Markov chain typically involves solving a set of linear equations. You can use the markovchain and solve functions in R to find the stationary distribution.
Here's an example R program:

```
# Load the markovchain package
library(markovchain)

# Define the transition matrix of the Markov chain
# Replace this with your own transition matrix
P <- matrix(c(0.7, 0.3, 0.2, 0.8), nrow = 2, byrow = TRUE)

# Create a markovchain object
mc <- new("markovchain", states = c("State1", "State2"), transitionMatrix = P)

# Find the stationary distribution
stationary_distribution <- steadyStates(mc)

# Print the stationary distribution
cat("Stationary Distribution:")
print(stationary_distribution)
```

Note: Make sure you have an active internet connection, as this command will download the package from the Comprehensive R Archive Network (CRAN) and install it on your machine.

First Dowload & install R packages, you can use the `install.packages("markovchain")` function.
Second download & install the "markovchain" package in R. Here's how you can install the "markovchain" package you can use the `install.packages("markovchain")` function

Third : After the installation is complete, you can load the package into your R session using the `library(markovchain)` function

Keep in mind that you only need to install a package once, but you'll need to load it in each new R session where you want to use its functions.

Define the transition matrix P of your Markov chain. Make sure it represents the transitions between your states correctly.
Create a markovchain object with the transition matrix.
Use the steadyStates function to find the stationary distribution.
Print the stationary distribution.
Make sure to replace the example transition matrix with your own transition matrix based on your specific Markov chain.

Output :

```
Stationary Distribution:> print(stationary_distribution)
     State1 State2
[1,]    0.4    0.6
```

7. Write a R program that include linear algebra operations on vectors and matrices.

In this program, we perform the following linear algebra operations:

Vector addition and subtraction.
Vector dot product.
Matrix addition and subtraction.
Matrix multiplication (using %*% for matrix multiplication).
Matrix determinant calculation.
Matrix inverse calculation (using solve).
You can run this code in an R environment to see the results of these linear algebra operations on vectors and matrices.

Program	output

<pre># Create vectors 	
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
# Create matrices 	
matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
matrix2 <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 2)
 	
# Vector addition 	
vector_sum <- vector1 + vector2
cat("Vector Addition:", vector_sum, "\n")
 	
# Vector subtraction 	
vector_diff <- vector1 - vector2
cat("Vector Subtraction:", vector_diff, "\n")
 	
# Vector dot product 	
dot_product <- sum(vector1 * vector2)
cat("Vector Dot Product:", dot_product, "\n")
 	
# Matrix addition 	
matrix_sum <- matrix1 + matrix2
cat("Matrix Addition:\n")
print(matrix_sum)
 	
# Matrix subtraction 	
matrix_diff <- matrix1 - matrix2
cat("Matrix Subtraction:\n")
print(matrix_diff)
 	
# Matrix multiplication 	
matrix_product <- matrix1 %*% t(matrix2)
cat("Matrix Multiplication:\n")
print(matrix_product)
 	
# Matrix determinant 	
matrix_det <- det(matrix1)
cat("Matrix Determinant:", matrix_det, "\n")
 	
# Matrix inverse 	
matrix_inv <- solve(matrix1)
cat("Matrix Inverse:\n")
print(matrix_inv)</pre>
	<pre>Vector Addition: 5 7 9
	Vector Subtraction: -3 -3 -3
	Vector Dot Product: 32
	Matrix Addition:
	     [,1] [,2] [,3]
	[1,] 8 12 16
	[2,] 10 14 18
	Matrix Subtraction:
	     [,1] [,2] [,3]
	[1,] -6 -6 -6
	[2,] -6 -6 -6
	
	Matrix Multiplication:
	     [,1] [,2]
	[1,] 89 98
	[2,] 116 128
	Error in determinant.matrix(x, logarithm = TRUE, ...) :
	  'x' must be a square matrix
	Calls: det -> determinant -> determinant.matrix
	Execution halted [Execution complete with exit code 1]</pre>

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

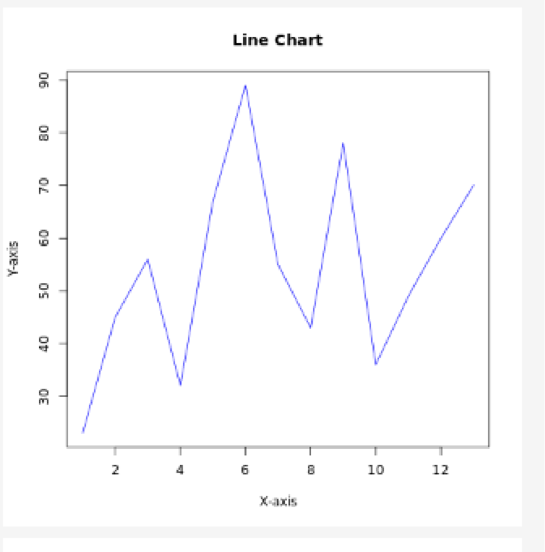
<table>
<tr>
<td>

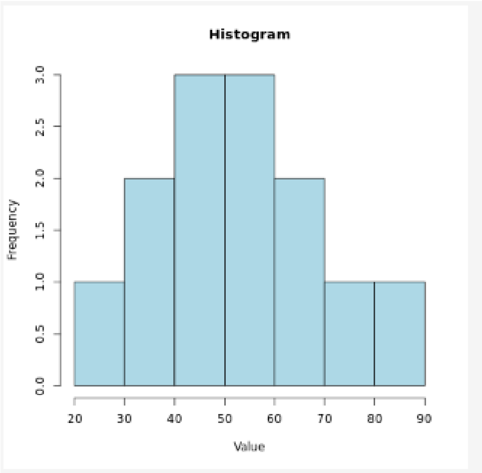
```
# Create a sample dataset
data <- c(23, 45, 56, 32, 67, 89, 55, 43, 78, 36, 49, 60, 70)

# Create a basic line chart
plot(data, type = "l", col = "blue", xlab = "X-axis", ylab = "Y-axis", main = "Line Chart")

# Create a histogram
hist(data, col = "lightblue", xlab = "Value", ylab = "Frequency", main = "Histogram")

# Create a pie chart
pie_data <- c(20, 30, 40, 10)
pie(pie_data, labels = c("A", "B", "C", "D"), col = rainbow(length(pie_data)), main = "Pie
Chart")

# Create a boxplot
boxplot(data, col = "lightgreen", xlab = "Value", main = "Box Plot")

# Create a scatterplot
x <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
y <- c(2, 4, 5, 7, 8, 10, 11, 14, 15, 17)
plot(x, y, col = "red", xlab = "X-axis", ylab = "Y-axis", main = "Scatterplot")

# Add a regression line to the scatterplot
abline(lm(y ~ x), col = "blue")

# Create a legend for the scatterplot
legend("topleft", legend = "Regression Line", col = "blue", lty = 1)
```

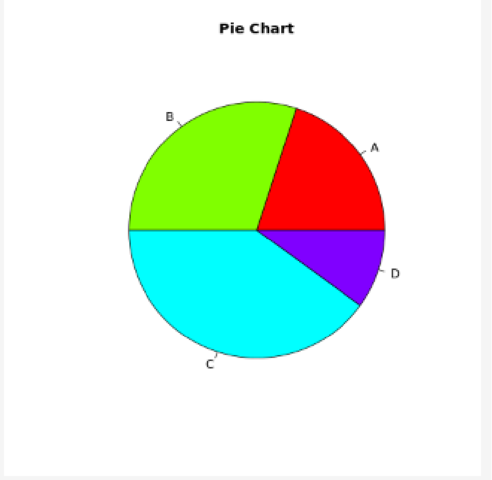
</td>
<td>
This program creates various types of graphs:

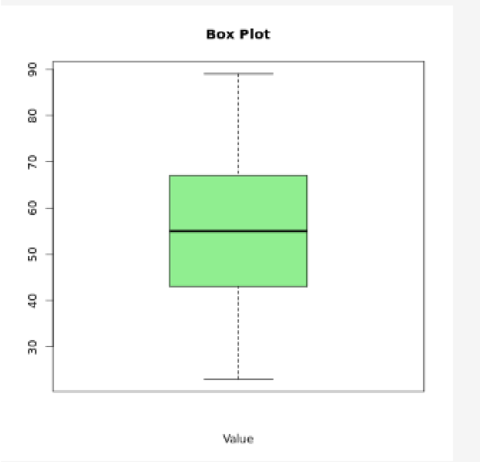
A line chart using plot().
A histogram using hist().
A pie chart using pie().
A box plot using boxplot().
A scatterplot using plot() and adds a regression line to it using abline(). Finally, a legend is added to the scatterplot.
You can run this code in an R environment to visualize the different graph types.
</td>
</tr>
</table>

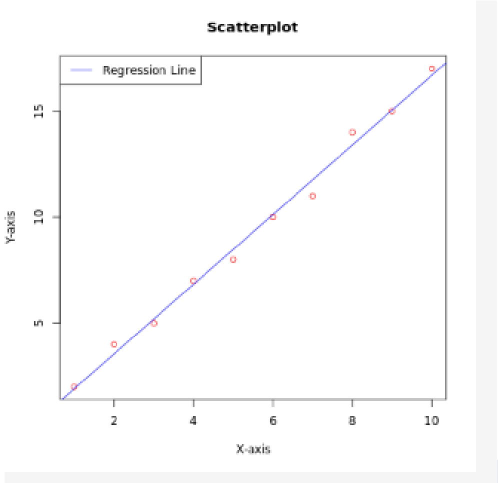
Output :


Line Chart
Y-axis
X-axis


Histogram
Frequency
Value


Pie Chart
A
B
C
D


Box Plot
Value


Scatterplot
Regression Line
Y-axis
X-axis

9. Write a R program for with any dataset containing dataframe objects, indexing and subsetting data frames, and employ manipulating and analyzing data.

In this program:

We create a sample DataFrame called data.
We select specific columns from the DataFrame.
We subset rows based on a condition (age < 30).
We change a specific value in the DataFrame.
We add a new column (Salary) to the DataFrame.
We calculate the average age and maximum salary.
We use the dplyr library to group the data by the "City" column and calculate summary statistics for each group.
You can run this code in an R environment to manipulate and analyze the sample data in the DataFrame.

Program

```r
# Create a sample DataFrame
data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David", "Eve"),
  Age = c(25, 30, 22, 28, 35),
  City = c("New York", "San Francisco", "Los Angeles", "Chicago", "Miami")
)

# Display the entire DataFrame
print(data)

# Select specific columns
selected_columns <- data[c("Name", "Age")]
print("Selected Columns:")
print(selected_columns)
# Select rows based on a condition
young_people <- data[data$Age < 30, ]
print("Young People:")
print(young_people)

# Change a specific value in the DataFrame
data[1, "Age"] <- 26

# Add a new column to the DataFrame
data$Salary <- c(55000, 60000, 48000, 65000, 70000)
print("DataFrame with Salary:")
print(data)

# Calculate the average age
average_age <- mean(data$Age)
cat("Average Age:", average_age, "\n")

# Calculate the maximum salary
max_salary <- max(data$Salary)
cat("Maximum Salary:", max_salary, "\n")

# Group data by a column and calculate summary statistics
library(dplyr)
grouped_data <- data %>%
  group_by(City) %>%
  summarise(Average_Age = mean(Age), Max_Salary = max(Salary))
print("Grouped Data:")
print(grouped_data)
```

output

```
Name Age        City
1  Alice  25     New York
2    Bob  30 San Francisco
3 Charlie  22  Los Angeles
4  David  28      Chicago
5    Eve  35       Miami
[1] "Selected Columns:"
    Name Age
1  Alice  25
2    Bob  30
3 Charlie  22
4  David  28
5    Eve  35
[1] "Young People:"
    Name Age      City
1  Alice  25   New York
3 Charlie  22 Los Angeles
4  David  28    Chicago
[1] "DataFrame with Salary:"
    Name Age        City Salary
1  Alice  26     New York  55000
2    Bob  30 San Francisco  60000
3 Charlie  22  Los Angeles  48000
4  David  28      Chicago  65000
5    Eve  35       Miami  70000
Average Age: 28.2
Maximum Salary: 70000

Attaching package: 'dplyr'

The following objects are masked from
'package:stats':
   filter, lag

The following objects are masked from
'package:base':
   intersect, setdiff, setequal, union
[1] "Grouped Data:"
# A tibble: 5 x 3
  City          Average_Age Max_Salary
  <chr>               <dbl>      <dbl>
1 Chicago                28      65000
2 Los Angeles            22      48000
3 Miami                  35      70000
4 New York               26      55000
5 San Francisco          30      60000

[Execution complete with exit code 0]
```

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

Creating a linear regression model in a multivariate context involves predicting a dependent variable based on multiple independent variables. Below is an example program in R that demonstrates how to build a multivariate linear regression model for predictive purposes. In this example, I'll use the built-in "mtcars" dataset, which contains information about various car models.

```r
# Load the mtcars dataset
data(mtcars)

# Explore the first few rows of the dataset
head(mtcars)

# Split the dataset into training and testing sets
set.seed(123)  # Set seed for reproducibility
sample_index <- sample(1:nrow(mtcars), 0.7 * nrow(mtcars))  # 70% for training, 30% for testing

train_data <- mtcars[sample_index, ]
test_data <- mtcars[-sample_index, ]

# Build a multivariate linear regression model
model <- lm(mpg ~., data = train_data)  # Assuming "mpg" is the dependent variable

# Summary of the model
summary(model)

# Make predictions on the test set
predictions <- predict(model, newdata = test_data)

# Evaluate the model
mse <- mean((predictions - test_data$mpg)^2)  # Mean Squared Error
# Print the Mean Squared Error
cat("Mean Squared Error:", mse, "\n")
```

This example uses the "mpg" (miles per gallon) variable as the dependent variable and includes all other variables in the dataset as independent variables. You may need to adjust the code based on your specific dataset and the variable you want to predict.

OUTPUT:

Terminal 1

```
C:\Users\Md_Yousuf\Documents>Rscript test.R
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1

Call:
lm(formula = mpg ~ ., data = train_data)

Residuals:
    Min      1Q  Median      3Q     Max
-3.7715 -1.4518 -0.4919  1.1869  4.6713

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 21.130670  26.526254   0.797    0.443
cyl         -0.888146   1.626880  -0.546    0.596
disp         0.025947   0.030443   0.852    0.412
hp          -0.031020   0.040364  -0.769    0.458
drat         0.009658   2.491820   0.004    0.997
wt          -4.881273   2.735728  -1.784    0.102
qsec         0.929945   0.986188   0.943    0.366
vs           0.542141   3.056351   0.177    0.862
am           3.129479   3.322083   0.942    0.366
gear        -0.310675   2.299091  -0.135    0.895
carb         0.628132   1.262164   0.498    0.629

Residual standard error: 3.162 on 11 degrees of freedom
Multiple R-squared:  0.8822,    Adjusted R-squared:  0.7751
F-statistic: 8.239 on 10 and 11 DF,  p-value: 0.0008402

Mean Squared Error: 5.205016
```