**Answer:-1**

# Step 1: Create the vector

vec <- c(5, 7, 9, 11, 13, 13, 11, 9, 7, 5)

# Step 2: Calculate the sum

sum\_vec <- sum(vec)

# Step 3: Calculate the mean

mean\_vec <- mean(vec)

# Step 4: Find the highest and lowest values

highest\_value <- max(vec)

lowest\_value <- min(vec)

# Step 5: Get the length of the vector

length\_vec <- length(vec)

# Step 6: Calculate variance and standard deviation

variance\_vec <- var(vec) # Sample variance

std\_dev\_vec <- sd(vec) # Sample standard deviation

# For population variance and standard deviation:

variance\_population <- var(vec) \* (length\_vec - 1) / length\_vec

std\_dev\_population <- sqrt(variance\_population)

# Step 7: Sort the vector in decreasing order

sorted\_vec <- sort(vec, decreasing = TRUE)

# Print the results

cat("Sum:", sum\_vec, "\n")

cat("Mean:", mean\_vec, "\n")

cat("Highest Value:", highest\_value, "\n")

cat("Lowest Value:", lowest\_value, "\n")

cat("Length:", length\_vec, "\n")

cat("Variance (sample):", variance\_vec, "\n")

cat("Standard Deviation (sample):", std\_dev\_vec, "\n")

cat("Variance (population):", variance\_population, "\n")

cat("Standard Deviation (population):", std\_dev\_population, "\n")

cat("Sorted Vector:", sorted\_vec, "\n")

**Output:-**

Sum: 90

Mean: 9

Highest Value: 13

Lowest Value: 5

Length: 10

Variance (sample): 8.888889

Standard Deviation (sample): 2.981424

Variance (population): 8

Standard Deviation (population): 2.828427

Sorted Vector: 13 13 11 11 9 9 7 7 5 5

**Answer:-2**

# Create a vector of the first 50 even numbers starting from 2

even\_numbers <- seq(2, 100, by = 2)

# Create a vector with values from 30 down to 1

countdown <- seq(30, 1)

# Print the vectors

cat("Even Numbers:", even\_numbers, "\n")

cat("Countdown:", countdown, "\n")

**Output:-**

Even Numbers: 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100

Countdown: 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

**Answer:-3**

# Create a vector of size 10 with the 5th and 7th values as NA

vector\_with\_na <- c(1, 2, 3, 4, NA, 6, NA, 8, 9, 10)

# Use is.na() to find locations of missing data

missing\_locations <- is.na(vector\_with\_na)

# Print the vector and the missing locations

cat("Vector:", vector\_with\_na, "\n")

cat("Missing locations (TRUE indicates NA):", missing\_locations, "\n")

**Output:-**

Vector: 1 2 3 4 NA 6 NA 8 9 10

Missing locations (TRUE indicates NA): FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE FALSE FALSE

**Answer:-4**

# Create a character vector of size 5

char\_vector <- c("This", "is", "a", "character", "vector")

# Find the index of the value "is" using which()

index\_which <- which(char\_vector == "is")

# Alternatively, find the index using match()

index\_match <- match("is", char\_vector)

# Print the results

cat("Character Vector:", char\_vector, "\n")

cat("Index of 'is' using which():", index\_which, "\n")

cat("Index of 'is' using match():", index\_match, "\n")

**Output:-**

Character Vector: This is a character vector

Index of 'is' using which(): 2

Index of 'is' using match(): 2

**Answer:-5**

# Step 1: Create a 7-point scale

seven\_point\_scale <- 1:7

names(seven\_point\_scale) <- c("Bad", "Somewhat Bad", "Not Good", "Ok", "Good", "Very Good", "Excellent")

# Step 2: Input feedback from 5 students

# For this example, let's assume the feedback given is:

student\_feedback <- c("Good", "Ok", "Very Good", "Bad", "Excellent")

# Step 3: Convert feedback to corresponding scale values

numeric\_feedback <- seven\_point\_scale[student\_feedback]

# Step 4: Calculate the average feedback

average\_feedback <- mean(numeric\_feedback)

# Step 5: Print results

cat("Seven Point Scale:", seven\_point\_scale, "\n")

cat("Student Feedback:", student\_feedback, "\n")

cat("Numeric Feedback:", numeric\_feedback, "\n")

cat("Average Feedback:", average\_feedback, "\n")

**Output:-**

Seven Point Scale: 1 2 3 4 5 6 7

Student Feedback: Good Ok Very Good Bad Excellent

Numeric Feedback: 5 4 6 1 7

Average Feedback: 4.6

**Answer:-6**

# Step 1: Create two strings

string1 <- "Hello"

string2 <- "World"

# Step 2: Concatenate the strings

concatenated\_string <- paste(string1, string2)

# Step 3: Print the result

cat("Concatenated String:", concatenated\_string, "\n")

**Output:-**

Concatenated String: Hello World

**Answer:-7**

# Define the string with punctuation marks

stringName <- "Hello, world! This is a test. Let's see how, well, it works: amazing, isn't it?"

# Remove punctuation marks

clean\_string <- gsub("[[:punct:]]", "", stringName)

# Split the cleaned string into words

words <- unlist(strsplit(clean\_string, " "))

# Remove any empty strings that may result from splitting

words <- words[words != ""]

# Number of words

num\_words <- length(words)

# Count distinct words and their occurrences

word\_counts <- table(tolower(words)) # Convert to lowercase to avoid case sensitivity

distinct\_words <- length(word\_counts)

# Display results

list(

Cleaned\_String = clean\_string,

Total\_Words = num\_words,

Distinct\_Words = distinct\_words,

Word\_Frequencies = word\_counts

)

**Output:-**

$Cleaned\_String

[1] "Hello world This is a test Lets see how well it works amazing isnt it"

$Total\_Words

[1] 15

$Distinct\_Words

[1] 14

$Word\_Frequencies

a amazing hello how is isnt it lets see test

1 1 1 1 1 1 2 1 1 1

this well works world

1 1 1 1

**Answer:-9**

# Create two 5x5 matrices

matrix1 <- matrix(1:25, nrow = 5, ncol = 5) # Matrix with values from 1 to 25

matrix2 <- matrix(25:1, nrow = 5, ncol = 5) # Matrix with values from 25 to 1

# Display the matrices

print("Matrix 1:")

print(matrix1)

print("Matrix 2:")

print(matrix2)

# Addition of two matrices

matrix\_addition <- matrix1 + matrix2

print("Matrix Addition:")

print(matrix\_addition)

# Subtraction of two matrices

matrix\_subtraction <- matrix1 - matrix2

print("Matrix Subtraction:")

print(matrix\_subtraction)

# Element-wise multiplication of two matrices

matrix\_multiplication <- matrix1 \* matrix2

print("Matrix Element-wise Multiplication:")

print(matrix\_multiplication)

# Matrix multiplication (matrix product)

matrix\_product <- matrix1 %\*% matrix2

print("Matrix Product:")

print(matrix\_product)

**Output:-**

[1] "Matrix 1:"

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

[1,] 1 6 11 16 21

[2,] 2 7 12 17 22

[3,] 3 8 13 18 23

[4,] 4 9 14 19 24

[5,] 5 10 15 20 25

[1] "Matrix 2:"

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

[1,] 25 20 15 10 5

[2,] 24 19 14 9 4

[3,] 23 18 13 8 3

[4,] 22 17 12 7 2

[5,] 21 16 11 6 1

[1] "Matrix Addition:"

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

[1,] 26 26 26 26 26

[2,] 26 26 26 26 26

[3,] 26 26 26 26 26

[4,] 26 26 26 26 26

[5,] 26 26 26 26 26

[1] "Matrix Subtraction:"

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

[1,] -24 -14 -4 6 16

[2,] -22 -12 -2 8 18

[3,] -20 -10 0 10 20

[4,] -18 -8 2 12 22

[5,] -16 -6 4 14 24

[1] "Matrix Element-wise Multiplication:"

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

[1,] 25 120 165 160 105

[2,] 48 133 168 153 88

[3,] 69 144 169 144 69

[4,] 88 153 168 133 48

[5,] 105 160 165 120 25

[1] "Matrix Product:"

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

[1,] 1215 940 665 390 115

[2,] 1330 1030 730 430 130

[3,] 1445 1120 795 470 145

[4,] 1560 1210 860 510 160

[5,] 1675 1300 925 550 175

**Answer:-10**

# Define a matrix

matrix1 <- matrix(1:9, nrow = 3, ncol = 3) # 3x3 matrix

# Display the original matrix

print("Original Matrix:")

print(matrix1)

# Transpose the matrix

matrix\_transpose <- t(matrix1)

# Display the transposed matrix

print("Transposed Matrix:")

print(matrix\_transpose)

**Output:-**

[1] "Original Matrix:"

[,1] [,2] [,3]

[1,] 1 4 7

[2,] 2 5 8

[3,] 3 6 9

[1] "Transposed Matrix:"

[,1] [,2] [,3]

[1,] 1 2 3

[2,] 4 5 6

[3,] 7 8 9

**Answer:-11**

# Define a square matrix

matrix1 <- matrix(c(4, 7, 2, 6), nrow = 2, ncol = 2) # 2x2 matrix

# Display the original matrix

print("Original Matrix:")

print(matrix1)

# Calculate the inverse of the matrix

matrix\_inverse <- solve(matrix1)

# Display the inverse matrix

print("Inverse of Matrix:")

print(matrix\_inverse)

**Output:-**

[1] "Original Matrix:"

[,1] [,2]

[1,] 4 2

[2,] 7 6

[1] "Inverse of Matrix:"

[,1] [,2]

[1,] 0.6 -0.2

[2,] -0.7 0.4

**Answer:-12**

# Create a list of factors

factor\_list <- factor(c("apple", "banana", "apple", "orange", "banana", "apple", "grape", "orange", "apple"))

# Display the list of factors

print("List of Factors:")

print(factor\_list)

# Find occurrences of each factor

factor\_counts <- table(factor\_list)

# Display the occurrences of each factor

print("Occurrences of Each Factor:")

print(factor\_counts)

**Output:-**

[1] "List of Factors:"

[1] apple banana apple orange banana apple grape orange apple

Levels: apple banana grape orange

[1] "Occurrences of Each Factor:"

factor\_list

apple banana grape orange

4 2 1 2

**Answer:-13**

# Function to find the largest and smallest values in a 3x3x3 array

find\_min\_max <- function(arr) {

# Find the smallest value

min\_value <- min(arr)

# Find the largest value

max\_value <- max(arr)

# Return the results as a list

return(list(Smallest = min\_value, Largest = max\_value))

}

# Create a 3x3x3 array

array\_data <- array(1:27, dim = c(3, 3, 3))

# Call the function and pass the array as an argument

result <- find\_min\_max(array\_data)

# Display the results

print("Smallest and Largest values in the array:")

print(result)

**Output:-**

[1] "Smallest and Largest values in the array:"

$Smallest

[1] 1

$Largest

[1] 27

**Answer:-14**

# Define a symmetric matrix

matrix\_sym <- matrix(c(4, 1, 1,

1, 3, 0,

1, 0, 2),

nrow = 3, ncol = 3)

# Display the symmetric matrix

print("Symmetric Matrix:")

print(matrix\_sym)

# Calculate the eigenvalues and eigenvectors

eigen\_result <- eigen(matrix\_sym)

# Display the eigenvalues

print("Eigenvalues:")

print(eigen\_result$values)

# Display the eigenvectors

print("Eigenvectors:")

print(eigen\_result$vectors)

**Output:-**

[1] "Symmetric Matrix:"

[,1] [,2] [,3]

[1,] 4 1 1

[2,] 1 3 0

[3,] 1 0 2

[1] "Eigenvalues:"

[1] 4.879385 2.652704 1.467911

[1] "Eigenvectors:"

[,1] [,2] [,3]

[1,] 0.8440296 0.2931284 0.4490988

[2,] 0.4490988 -0.8440296 -0.2931284

[3,] 0.2931284 0.4490988 -0.8440296

**Answer:-15**

# Define the states for 20 students (assuming they are from 5 different states)

states <- c("Texas", "California", "Texas", "New York", "Florida",

"California", "Texas", "Florida", "California", "New York",

"Texas", "California", "Florida", "New York", "California",

"Texas", "Florida", "New York", "Florida", "Texas")

# Display the states list

print("States of 20 Students:")

print(states)

# Convert the states list to a factor

states\_factor <- factor(states)

# Compute the frequency of each state using table() on the factor

state\_frequencies <- table(states\_factor)

# Display the frequencies of each state

print("Frequency of Each State:")

print(state\_frequencies)

**Output:-**

[1] "States of 20 Students:"

[1] "Texas" "California" "Texas" "New York" "Florida"

[6] "California" "Texas" "Florida" "California" "New York"

[11] "Texas" "California" "Florida" "New York" "California"

[16] "Texas" "Florida" "New York" "Florida" "Texas"

[1] "Frequency of Each State:"

states\_factor

California Florida New York Texas

5 5 4 6

**Answer:-16**

# Define a list of incomes

incomes <- c(45000, 70000, 120000, 30000, 50000, 80000, 450000, 25000, 67000, 95000,

53000, 120000, 15000, 32000, 58000, 40000, 75000, 135000, 280000, 48000)

# Define income brackets

income\_brackets <- c(10000, 50000, 100000, 150000, 200000, 500000)

# Create factor classes for incomes using the cut() function

income\_classes <- cut(incomes, breaks = income\_brackets, right = FALSE,

labels = c("10000-50000", "50000-100000", "100000-150000",

"150000-200000", "200000-500000"))

# Display the income classes

print("Income Classes:")

print(income\_classes)

# Create a frequency table for the income classes

income\_frequencies <- table(income\_classes)

# Display the frequency distribution

print("Frequency Distribution of Income Classes:")

print(income\_frequencies)

**Output:-**

[1] "Income Classes:"

[1] 10000-50000 50000-100000 100000-150000 10000-50000 50000-100000

[6] 50000-100000 200000-500000 10000-50000 50000-100000 50000-100000

[11] 50000-100000 100000-150000 10000-50000 10000-50000 50000-100000

[16] 10000-50000 50000-100000 100000-150000 200000-500000 10000-50000

5 Levels: 10000-50000 50000-100000 100000-150000 ... 200000-500000

[1] "Frequency Distribution of Income Classes:"

income\_classes

10000-50000 50000-100000 100000-150000 150000-200000 200000-500000

7 8 3 0 2