Experiment No 6

Title: Write a R Program to Convert a given matrix into 1 dimensional array, Create an 3 dimensional array of 24 elements using the dim () function.

Objective:

The objective of this practical is to demonstrate how to manipulate matrices and arrays in R. Specifically, it teaches how to convert a matrix into a 1-dimensional array and how to create a 3-dimensional array using the `dim()` function. This enhances understanding of data structures and their dimensional properties in R, which is crucial for handling complex datasets.

Theory:

R-matrix is a two-dimensional arrangement of data in rows and columns.
In a matrix, rows are the ones that run horizontally and columns are the ones that run vertically. In R programming, matrices are two-dimensional, homogeneous data structures. These are some examples of matrices:

$$\begin{pmatrix} 1 & 5 & 3 \\ 4 & 9 & 2 \\ 5 & 6 & 7 \end{pmatrix} \quad \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad \begin{bmatrix} 1 & 4 & 5 \end{bmatrix}$$

Creating a Matrix in R
To create a matrix in R you need to use the function called **matrix()**.
The arguments to this **matrix()** are the set of elements in the vector. You have to pass how many numbers of rows and how many numbers of columns you want to have in your matrix.

Note: *By default, matrices are in column-wise order.*

Syntax to Create R-Matrix

matrix(data, nrow, ncol, byrow, dimnames)
Parameters:

- data – values you want to enter
- nrow – no. of rows
- ncol – no. of columns
- byrow – logical clue, if 'true' value will be assigned by rows

- dimnames – names of rows and columns

Get or Set Dimensions of a Matrix in R Programming –dim() function

The dim() function in R is used to get or set the dimensions of an object. This function is particularly useful when working with matrices, arrays, and data frames. Below, we will discuss how to use dim() to both retrieve and set dimensions, with examples for better understanding.
Syntax: dim(x)
Parameters:
x: array, matrix or data frame.

The dim() function in R is a versatile tool for getting and setting the dimensions of matrices, arrays, and data frames. Whether you need to reshape a vector into a matrix or simply check the size of an existing matrix, dim() provides an easy and efficient way to handle dimensions in R.

Practical Approach:

R Program to Convert a given matrix into 1 dimensional array

```
rows=c("r1","r2")

cols=c("c1","c2","c3","c4")

M=matrix(c(2:9),nrow=2,byrow=TRUE,dimnames=list(rows,cols))

print("Original matrix:")

print(M)

output=as.vector(M)

print("1D array :")

print(output)
```

Output:

```
[1] "Original matrix:"
  c1 c2 c3 c4
r1  2  3  4  5
r2  6  7  8  9
[1] "1D array :"
[1] 2 6 3 7 4 8 5 9
```

R program to Create an 3 dimensional array of 24 elements using the dim () function.

```
# Create two vectors

data1 <- c(1,2,3,4,5,6)

data2 <- c(60, 18, 12, 13, 14, 19)


# pass these vectors as input to the array.

# 4 rows,2 columns and 3 arrays

result <- array(c(data1, data2), dim = c(4,2,3))

print(result)
```

Output:

```
, , 1

     [,1] [,2]
[1,]    1    5
[2,]    2    6
[3,]    3   60
[4,]    4   18

, , 2

     [,1] [,2]
[1,]   12    1
[2,]   13    2
[3,]   14    3
[4,]   19    4

, , 3

     [,1] [,2]
[1,]    5   12
[2,]    6   13
[3,]   60   14
[4,]   18   19
```

Outcome:

By completing this practical, we will learn how to change a matrix into a simple list of numbers and create a 3D array in R. This helps you understand how to work with data in different shapes and sizes, making it easier to manage and analyze information.

Online Reference Websites:

- **https://www.w3resource.com/r-programming-exercises/matrix/index.php**
- **https://campus.datacamp.com/courses/php-15602560-statistical-programming-in-r/basics-of-r-programming?ex=13**
- **https://www.coursesidekick.com/mathematics/270914**
- **https://www.programiz.com/r/matrix**

Viva Questions:

1.	What is a matrix in R, and how is it different from a data frame?
2.	How do you create a matrix in R?
3.	What function is used to convert a matrix into a 1-dimensional array?
4.	How can you access specific elements or rows and columns of a matrix in R?
5.	How do you find the dimensions (rows and columns) of a matrix in R?
6.	What is the difference between `matrix()` and `array()` in R?
7.	How do you perform element-wise addition or multiplication on two matrices in R?
8.	How can you transpose a matrix in R?
9.	Can a matrix in R hold data types other than numeric? Why or why not?
10.	How would you bind two matrices together by rows or columns in R?

Conclusion:

This practical enhances our ability to work with matrices and arrays in R, allowing for more flexible data manipulation. Understanding these structures is crucial for efficient data analysis and handling in R programming.

Experiment No 9

Title: Write an R Program to read data from the file and writing output back to specified file.

Objective:

Objective of this practical is to:

- To learn how to read data from external files into R.
- To practice writing data from R back into a specified output file.
- To apply file I/O operations for data analysis and reporting tasks in R.

Theory:

File reading in R

One of the important formats to store a file is in a text file. R provides various methods that one can read data from a text file.

- **read.delim()**: This method is used for reading "tab-separated value" files (".txt"). By default, point (".") is used as decimal point.

Syntax: read.delim(file, header = TRUE, sep = "\t", dec = ".", …)
Parameters:
file: the path to the file containing the data to be read into R.
header: a logical value. If TRUE, read.delim() assumes that your file has a header row, so row 1 is the name of each column. If that's not the case, you can add the argument header = FALSE.
sep: the field separator character. "\t" is used for a tab-delimited file.
dec: the character used in the file for decimal points.

- **read_tsv():** This method is also used for to read a tab separated ("\t") values by using the help of readr package.

Syntax: read_tsv(file, col_names = TRUE)
Parameters:
file: the path to the file containing the data to be read into R.
col_names: Either TRUE, FALSE, or a character vector specifying column names. If TRUE, the first row of the input will be used as the column names.

Reading one line at a time

read_lines(): This method is used for the reading line of your own choice whether it's one or two or ten lines at a time. To use this method we have to import **reader** package.
Syntax: read_lines(file, skip = 0, n_max = -1L)

Parameters:

file: file path
skip: Number of lines to skip before reading data
n_max: Numbers of lines to read. If n is -1, all lines in the file will be read.

Reading the whole file

read_file(): This method is used for reading the whole file. To use this method we have to import reader package.

Syntax: read_lines(file)

file: the file path

Reading a file in a table format

Another popular format to store a file is in a tabular format. R provides various methods that one can read data from a tabular formatted data file.

- **read.table()**: read.table() is a general function that can be used to read a file in table format. The data will be imported as a data frame.

Syntax: read.table(file, header = FALSE, sep = "", dec = ".")

Parameters:

file: the path to the file containing the data to be imported into R.
header: logical value. If TRUE, read.table() assumes that your file has a header row, so row 1 is the name of each column. If that's not the case, you can add the argument header = FALSE.
sep: the field separator character
dec: the character used in the file for decimal points.

- **read.csv():** read.csv() is used for reading "comma separated value" files (".csv"). In this also the data will be imported as a data frame.

Syntax: read.csv(file, header = TRUE, sep = ",", dec = ".", …)

Parameters:

file: the path to the file containing the data to be imported into R.
header: logical value. If TRUE, read.csv() assumes that your file has a header row, so row 1 is the name of each column. If that's not the case, you can add the argument header = FALSE.
sep: the field separator character
dec: the character used in the file for decimal points.

R – Writing to Files

Writing Data to CSV files in R Programming Language

CSV stands for Comma Separated Values. These files are used to handle a large amount of statistical data. Following is the syntax to write to a CSV file:

Syntax:

```
write. csv(my_data, file = "my_data.csv")
write. csv2(my_data, file = "my_data.csv")
```

Here,
csv() and csv2() are the function in R programming.

- write.csv() uses “.” for the decimal point and a comma (“, ”) for the separator.
- write.csv2() uses a comma (“, ”) for the decimal point and a semicolon (“;”) for the separator.

Writing Data to text files

Text files are commonly used in almost every application in our day-to-day life as a step for the “Paperless World”. Well, writing to .txt files is very similar to that of the CSV files. Following is the syntax to write to a text file:

Syntax:

```
write.table(my_data, file = "my_data.txt", sep = "")
```

Writing Data to Excel files

To write data to excel we need to install the package known as “xlsx package”, it is basically a java based solution for reading, writing, and committing changes to excel files. It can be installed as follows:

```
install.packages("xlsx")
```

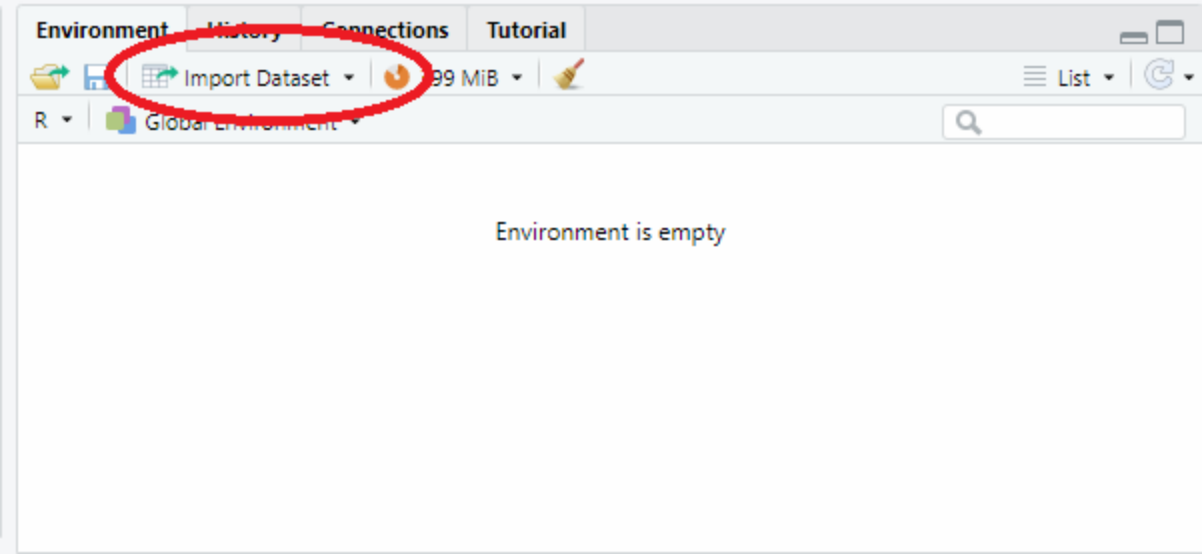
```
library("xlsx")
 write.xlsx(my_data, file = "result.xlsx",
 sheetName = "my_data", append = FALSE).
```

Using R-Studio

Here we are going to import data through R studio with the following steps.

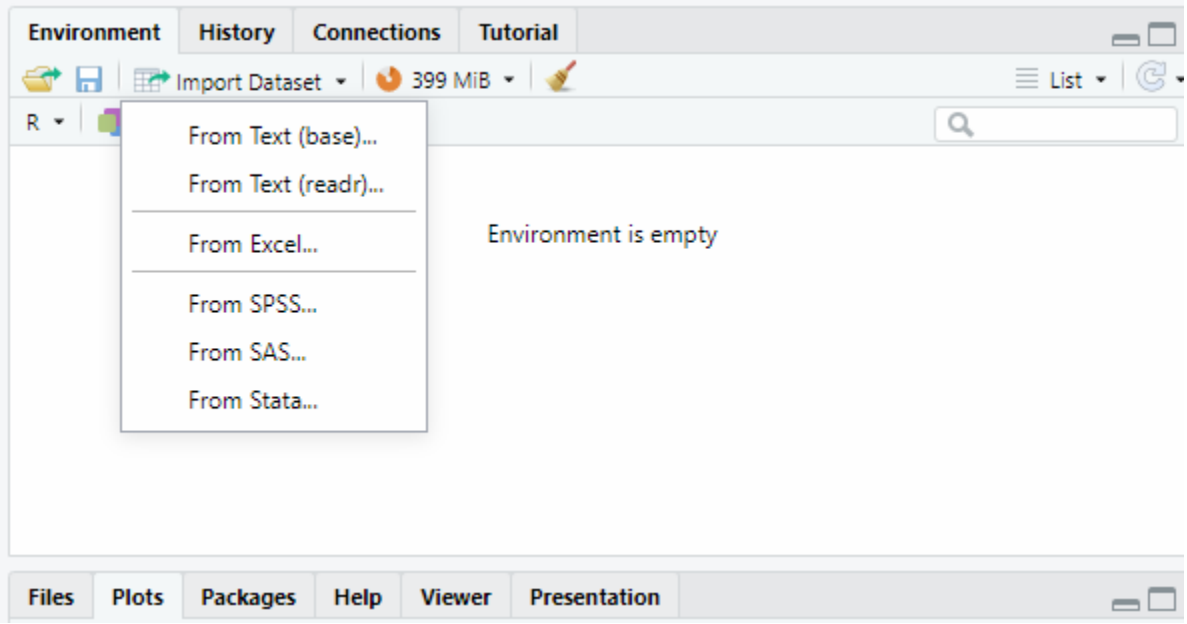
Steps:

- From the Environment tab click on the Import Dataset Menu.



Importing Data in R Script

- Select the file extension from the option.



Importing Data in R Script

- In the third step, a pop-up box will appear, either enter the file name or browse the desktop.
- The selected file will be displayed on a new window with its dimensions.
- In order to see the output on the console, type the filename.

Practical Approach:

R Program to read data from the file and writing output back to specified file.

```
# 1. Reading data from a file
input_file <- "input_data.txt"   # Specify the input file path

# Check if the file exists
if (file.exists(input_file)) {
  data <- read.table(input_file, header = TRUE)  # Reading the file (assuming it's a table with a header)
  cat("Data read from file:\n")
  print(data)
} else {
  cat("Input file does not exist!\n")
}
# 2. Writing data to another file
output_file <- "output_data.txt"   # Specify the output file path

# Writing data to the specified file
write.table(data, file = output_file, row.names = FALSE, col.names = TRUE)

cat("\nData has been written to", output_file, "\n")
```

Outcome:

The program successfully reads data from an input file, processes the data (if required), and writes the output to a specified file, ensuring data handling between files is completed efficiently.

Online Reference Websites:

- **https://www.datacamp.com/tutorial/r-data-import-tutorial**

- **http://www.sthda.com/english/wiki/best-practices-in-preparing-data-files-for-importing-into-r**

Viva Questions:

1.	What function is used to read data from a CSV file in R?
2.	How do you write data to a file in R, ensuring no row names are included in the output?
3.	Can you explain how to specify a custom file path when reading or writing files in R?
4.	What are some common file formats supported by R for reading and writing data?
5.	How would you handle missing data when reading a file in R?
6.	How do you ensure that data types are preserved correctly when writing data to a file in R?
7.	What are some potential errors you might encounter when reading or writing files, and how would you handle them in R?

Conclusion:

This practical demonstrates the ability to read data from a file and write it back to another file in R. It highlights the importance of correctly handling file I/O operations, ensuring smooth data flow between files, and preparing the data for further processing or storage. Understanding these operations is fundamental for effective data manipulation in R.