Assignment 1: R Programming Basics Program using List, Matrix, String and Factors Program using data frame and visualization

Objectives:

Students will be able to

- Understand data structures available in R programming.
- Understand how data structures are used for writing programmes in R
- Learn working of R programming for data visualization.
- How to implement data structures in R, step by step
- Understand use of inbuilt data sets and how to perform operations on it.

Prerequisites:

- ✓ Knowledge of C, CPP Programming languages
- ✓ https://www.statmethods.net/r-tutorial/index.html
- ✓ https://www.listendata.com/2014/06/getting-started-with-r.html

Introduction :

R programming is one of the most popular languages used in data science, statistical computations or scientific research. R programming is very efficient and user-friendly widely used in machine learning. It provides flexibility in doing big statistical operations with a few lines of code. R programming provides following data structures

1. Vectors
2. Lists
3. Arrays
4. Matrix
5. Data Frames
6. Factors

What is R Vector?

A vector is a sequence of elements that share the same data type. These elements are known as components of a vector.

R vector is the basic data structure, which plays an essential role in R programming. Vectors in R are same as the arrays in C language which hold multiple data values of the same type. One major key point is that in R the indexing of the vector will start from '1' and not from '0'. We can create numeric vectors and character vectors as well.

R vector comes in two parts: **Atomic vectors** and **Lists.**

- These data structures differ in the type of their elements: All elements of an atomic vector must be of the same type, whereas the elements of a list can have different types.
- They have three common properties, i.e., **function type, function length**, and **attribute function**.

Types of Atomic Vector

1. Numeric Data Type - Decimal values are referred to as numeric data types in R.

2. Integer Data Type - A numeric value with no fraction called integer data is represented by “Int”.

```
# R program to create numeric Vectors
# Creation of vectors using c() function.
v1 <- c(1,2,3,4)
typeof(v1)          # display type of vector
```

Output

```
[1] "double"
```

```
v2<- c(1.1, 2.1, 3.1, 4.1)
typeof(v2)
```

Output

```
[1] "double"
```

```
# by using 'L' we can specify that we want integer values.
        v3 <- c(1L, 2L, 3L, 4L)
        typeof(v3)
```

Output

```
[1] "integer"
```

3. Character Data Type - The character is held as the one-byte integer in memory

```
# R program to create Character Vectors

# by default numeric values are converted into characters
        v4 <- c('Lab', '357', 'data mining', '35')
        typeof(v4) # displaying type of vector
```

Output

```
[1] " character"
```

4. Logical Data Type - A logical data type returns either of the two values – TRUE or FALSE based on which condition is satisfied and NA for NULL values.
   For Example:

```
# R program to create Logical Vectors

#Creating logical vector using c() function
    V5 <- c(TRUE, FALSE, TRUE, NA)
    typeof(V5)
```

Output:

```
        [1] "logical"
```

How to create a vector in R?

Different ways to create vector in R

1. **Using c() function** - In R, we use c() function to create a vector. This function returns a one-dimensional array or simply vector. The c() function is a generic function which combines its argument. All arguments are restricted with a common data type which is the type of the returned value. There are various other ways to create a vector in R, which are as follows:

```
# we can use the c function to combine the values as a vector.
# By default the type will be double
V1 <- c(1, 2, 3, 4)
cat('using c function', V1, '\n')

Output:
using c function 1 2 3 4
```

2. **Using the ':' Operator:** We can create a vector with the help of the colon operator to create vector of continuous values

```
V2 <- 1:5
cat('using colon', V2)
Output:
using colon 1 2 3 4 5
```

3. **Using the seq() function -** In R, we can create a vector with the help of the seq() function. A sequence function creates a sequence of elements as a vector. There are also two ways in this. The first way is to set the step size and the second method is by setting the length of the vector.

```
Seq_V <- seq(1, 4, length.out = 6)
cat('using seq() function', Seq_V, '\n')
Output:
        using seq() function 1 1.6 2.2 2.8 3.4 4
Seq_V2 <- seq(1, 4, by = 0.5)
cat('using seq() function by', Seq_V2, '\n')
Output:
        using seq() function by 1 1.5 2.0 2.5 3.0 4
```

How to Access Elements of R Vectors?

With the help of vector indexing, we can access the elements of vectors. Indexing denotes the position where the values in a vector are stored. This indexing can be performed with the help of integer, character or logic.

```
# R program to access elements of a Vector
# accessing elements with an index number.
X <- c(1, 5, 10, 1, 12)
cat('Using Subscript operator', X[2], '\n')
Output:
Using Subscript operator 5
# by passing a range of values inside the vector index.
Y <- c(14, 18, 12, 11, 17)
cat('Using combine() function', Y[c(4, 1)], '\n')
Output:
Using combine() function 11 14
# using logical expressions
Z <- c(5, 2, 1, 4, 4, 3)
cat('Using Logical indexing', Z[Z>4])
Output:
Using Logical indexing 5
```

```
#indexing using character vector
char_vec<-c("shubham"=101,"ram"=102,"varsha"=103)
cat('Using character vector', char_vec[“ram”])
Output:
Using character vector 102
#indexing using logical vector it returns the values of those positions whose
#corresponding position has a logical vector TRUE.
a<-c(1,2,3,4,5,6)
a[c(TRUE,FALSE,TRUE,TRUE,FALSE,TRUE)]
Output:
[1] 1 3 4 6
```

Vector Operations

In R, there are various operation which is performed on the vector. We can add, subtract, multiply or divide two or more vectors from each other. In data science, R plays an important role, and operations are required for data manipulation. There are the following types of operation which are performed on the vector.

1. **Combining vector in R :** The c() function is not only used to create a vector, but also it is also used to combine two vectors. By combining one or more vectors, it forms a new vector which contains all the elements of each vector. Let see an example to see how c() function combines the vectors.

```
num = c(1, 2, 3, 4)
str = c("one", "two", "three", "Four")
c(num,str)
Output:

[1] "1"     "2"     "3"     "4"     "one"   "two"   "three" "Four"
```

2. **Arithmetic Operation on Vectors in R:** We can perform all the arithmetic operation on vectors. The arithmetic operations are performed member-by-member on vectors. We can add, subtract, multiply, or divide two vectors.

```
a<-c(1,3,5,7)
b<-c(2,4,6,8)
a+b  #addition
Output
            [1]  3  7 11 15
a-b  #subtraction
Output
            [1] -1 -1 -1 -1
a/b  #division #a % / % b is used to give integer division
Output
            [1] 0.5000000 0.7500000 0.8333333 0.8750000
a%%b  #Reminder operation
Output
[1] 1 3 5 7
```

3. **Logical Index Vector:** With the help of the logical index vector in R, we can form a new vector from a given vector. This vector has the same length as the original vector. The

vector members are TRUE only when the corresponding members of the original vector are included in the slice; otherwise, it will be false.

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
L<-c(TRUE,FALSE,TRUE,TRUE,FALSE,FALSE)
names[b]

Output
            [1] "Ram"   "Nisha" "Siya"
```

4. **Numeric Index**: In R, we specify the index between square braces [ ] for indexing a numerical value. If our index is negative, it will return us all the values except for the index which we have specified. For example, specifying [-3] will prompt R to convert -3 into its absolute value and then search for the value which occupies that index.

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[2]
Output
          [1] "Aryan"

names[-4]
Output
             [1] "Ram"    "Aryan"  "Nisha"  "Radha"  "Gunjan"

names[15]
Output
            NA
```

5. **Duplicate Index**: The index vector allows duplicate values. Hence, the following retrieves a member twice in one operation

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[c(2,4,4,3)]
Output
          [1] "Aryan" "Siya"  "Siya"  "Nisha"
```

6. **Range Index:** Range index is used to slice our vector to form a new vector. For slicing, we used colon(:) operator. Range indexes are very helpful for the situation involving a large operator.

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[1:3]
Output
[1] "Ram"   "Aryan" "Nisha"
```

7. **Out of Order Index:** In R, the index vector can be out-of-order. Below is an example in which a vector slice with the order of first and second values reversed

```
names<-c("Ram","Aryan","Nisha","Siya","Radha","Gunjan")
names[c(2,1,3,4,5,6)]
Output
            [1] "Aryan"  "Ram"    "Nisha"  "Siya"   "Radha"  "Gunjan"
```

8. **Sorting element of the vector**: **sort()** function is used with the help of which we can sort the values in ascending or descending order.

```
# R program to sort elements of a Vector
 # Creation of Vector
num <- c(8, 2, 7, 1, 11, 2)

# Sort in ascending order
Asc_num <- sort(num)
cat('ascending order', Asc_num, '\n')
```

Output

```
ascending order 1  2  2  7  8 11
```

```
# sort in descending order by setting decreasing as TRUE
 Desc_num<- sort(num, decreasing = TRUE)
cat('descending order', Desc_num)
```

Output

```
descending order 11  8  7  2  2  1
```

9. **Names vectors members**

```
 # We first create our vector of characters as
lib=c("TensorFlow","PyTorch")
lib
```

Output

```
[1] "TensorFlow" "PyTorch"
```

```
# Once our vector of characters is created, we name the first vector member as
"Start" and the #second member as "End" as:
names(lib)=c("Start","End")
lib
```

Output

```
      Start         End
"TensorFlow"    "PyTorch"
```

```
#We retrieve the first member by its name as follows:
lib["Start"]
```

Output

```
 Start
"TensorFlow"
```

Applications of vectors

1. In machine learning for principal component analysis vectors are used. They are extended to eigenvalues and eigenvector and then used for performing decomposition in vector spaces.

2. The inputs which are provided to the deep learning model are in the form of vectors. These vectors consist of standardized data which is supplied to the input layer of the neural network.
3. In the development of support vector machine algorithms, vectors are used.
4. Vector operations are utilized in neural networks for various operations like image recognition and text processing.

What is R List?

In R, lists are the second type of vector. Lists are the objects of R which contain elements of different types such as number, vectors, string and another list inside it. It can also contain a function or a matrix as its elements. A list is a data structure which has components of mixed data types. We can say, a list is a generic vector which contains other objects.

How to create List?

The process of creating a list is the same as a vector. In R, the vector is created with the help of c() function. Like c() function, there is another function, i.e., list() which is used to create a list in R. A list avoid the drawback of the vector which is data type. We can add the elements in the list of different data types.

```
# Create a list containing strings, numbers, vectors and a logical values.
list_data <- list("Ram", "Sham", c(1,2,3), TRUE, 1.03)
print(list_data)

Output
$Rscript main.r
[[1]]
[1] "Ram"
[[2]]
[1] "Sham"
[[3]]
[1] 1 2 3
[[4]]
[1] TRUE
[[5]]
[1] 1.03
```

Giving a name to list elements

R provides a easy way for accessing elements, i.e., by giving the name to each element of a list. By assigning names to the elements, we can access the element easily. There are only three steps to print the list data corresponding to the name:

1. Creating a list.
2. Assign a name to the list elements with the help of names() function.
3. Print the list data.

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),
   list("BCA","MCA","BSc"))
```

```
# Giving names to the elements in the list.
names(list_data) <- c("Students", "Marks", "Course")
  # Show the list.
print(list_data)
```

Output

```
$Students
[1] "Ram"  "Sham" "Raj"
$Marks
     [,1] [,2] [,3]
[1,]   40   60   90
[2,]   80   70   80
$Course
$Course[[1]]
[1] "BCA"
$Course[[2]]
[1] "MCA"
$Course[[3]]
[1] "BSc"
```

How to access R List elements?

R provides two ways to access the elements of a list.

1. First one is the indexing method performed in the same way as a vector.
2. In the second one, we can access the elements of a list with the help of names. It will be possible only with the named list.; we cannot access the elements of a list using names if the list is normal.

Example 1. Accessing elements of List using index

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),    list("BCA","
MCA","BSc"))
# Accessing the first element of the list.
print(list_data[1])
 Output
[[1]]
[1] "Ram"  "Sham" "Raj"
# Accessing the third element. The third element is also a list, so all its elements will be printed.
print(list_dat
Output

[[1]]
[[1]][[1]]
[1] "BCA"
[[1]][[2]]
[1] "MCA"
[[1]][[3]]
[1] "BSc"
```

Example 2. Accessing elements of List using name

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),    list("BCA","MCA","BSc"))
# Giving names to the elements in the list.
names(list_data) <- c("Student", "Marks", "Course")
# Accessing the first element of the list.
print(list_data["Student"])
Output
          $Student
          [1] "Ram"  "Sham" "Raj"

print(list_data$Marks)
Output

     [,1] [,2] [,3]
[1,]   40   60   90
[2,]   80   70   80

print(list_data)
Output

  $Student
[1] "Ram"  "Sham" "Raj"
$Marks
     [,1] [,2] [,3]
[1,]   40   60   90
[2,]   80   70   80
$Course
$Course[[1]]
[1] "BCA"
$Course[[2]]
[1] "MCA"
$Course[[3]]
[1] "BSc"
```

R allows us to add, delete, or update elements in the list. We can update an element of a list from anywhere, but elements can add or delete only at the end of the list. To remove an element from a specified index, we will assign it a null value. We can update the element of a list by overriding it from the new value.

```
# Creating a list containing a vector, a matrix and a list.
list_data <- list(c("Ram","Sham","Raj"), matrix(c(40,80,60,70,90,80), nrow = 2),    list("BCA","MCA"
,"BSc"))
# Giving names to the elements in the list.
names(list_data) <- c("Student", "Marks", "Course")
# Adding element at the end of the list(you can use append function also).
list_data[4] <- "Pune"  #list_data <- append(list_data,"Pune")
print(list_data[4])
Output
[[1]]
[1] "Pune"

# Removing the last element.
list_data[4] <- NULL
  # Printing the 4th Element.
print(list_data[4])
Output
$<NA>
NULL
  # Updating the 3rd Element.
list_data[3] <- "BCA Science"
print(list_data[3])
Output
$Course
[1] "BCA Science"
```

Converting list to vector : There is a drawback with the list, i.e., we cannot perform all the arithmetic operations on list elements. To remove this, drawback R provides unlist() function. This function converts the list into vectors. In some cases, it is required to convert a list into a vector so that we can use the elements of the vector for further manipulation. The unlist() function takes the list as a parameter and change into a vector.

```
# Creating lists.
list1 <- list(1:5)
print(list1)
Output
        [[1]]
        [1] 1 2 3 4 5
list2 <-list(11:15)
print(list2)
Output
        [[1]]
        [1] 11 12 13 14 15
# Converting the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)
print(v1)
Output
        [1] 1 2 3 4 5
 print(v2)
Output
        [1] 11 12 13 14 15
#adding the vectors
result <- v1+v2
print(result)
Output
        [1] 12 14 16 18 20
```

How to merge lists in R programming?

R allows us to merge one or more lists into one list. Merging is done with the help of the list() function also. To merge the lists, we have to pass all the lists into list function as a parameter, and it returns a list which contains all the elements which are present in the lists.

```
# Creating two lists.
Even_list <- list(2,4,6,8,10)
Odd_list <- list(1,3,5,7,9)

# Merging the two lists.
merged.list <- list(Even_list,Odd_list)

# Printing the merged list.
print(merged.list)
```

Output

```
[[1]]
[[1]][[1]]
[1] 2

[[1]][[2]]
[1] 4

[[1]][[3]]
[1] 6

[[1]][[4]]
[1] 8

[[1]][[5]]
[1] 10


[[2]]
[[2]][[1]]
[1] 1

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

[[2]][[3]]
[1] 5

[[2]][[4]]
[1] 7

[[2]][[5]]
[1] 9
```

What is array in R programming?

In R, arrays are the data objects which allow us to store data in more than two dimensions. In R, an array is created with the help of the **array()** function. This array() function takes a vector as an input and to create an array it uses vectors values in the **dim** parameter.

For example- if we will create an array of dimension (2, 3, 4) then it will create 4 rectangular matrices of 2 row and 3 columns.

```
#Create two vectors of different lengths.
 vector1 <- c(1,2,3)
 vector2 <- c(10,11,12,13,14,15)

 # Take these vectors as input to the array.
 result <- array(c(vector1,vector2),dim = c(3,3,2))
 print(result)
```

Output

```
, , 1

     [,1] [,2] [,3]
[1,]    1   10   13
[2,]    2   11   14
[3,]    3   12   15

, , 2

     [,1] [,2] [,3]
[1,]    1   10   13
[2,]    2   11   14
[3,]    3   12   15
```

Naming Columns and Rows

We can give names to the rows, columns and matrices in the array by using the **dimnames** parameter.

```
#Create two vectors of different lengths.
 vector1 <- c(1,2,3)
 vector2 <- c(10,11,12,13,14,15)

column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
 # Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames =
list(row.names,column.names,matrix.names))
print(result)
```

Output

```
, , Matrix1
     COL1 COL2 COL3
ROW1    1   10   13
ROW2    2   11   14
ROW3    3   12   15

, , Matrix2

     COL1 COL2 COL3
ROW1    1   10   13
ROW2    2   11   14
ROW3    3   12   15
```

Accessing Array elements

```
#Create two vectors of different lengths.
 vector1 <- c(1,2,3)
 vector2 <- c(10,11,12,13,14,15)
 column.names <- c("COL1","COL2","COL3")
row.names <- c("ROW1","ROW2","ROW3")
matrix.names <- c("Matrix1","Matrix2")
 # Take these vectors as input to the array.
result <- array(c(vector1,vector2),dim = c(3,3,2),dimnames =
list(row.names,column.names,matrix.names))

# Print the third row of the second matrix of the array.
print(result[3,,2])
Output
COL1 COL2 COL3
  3   12   15
 # Print the element in the 1st row and 3rd column of the 1st matrix.
 print(result[1,3,1])
 Output
 [1] 13
 # Print the 2nd Matrix.
 print(result[,,2])
 Output
     COL1 COL2 COL3
 ROW1    1   10   13
 ROW2    2   11   14
 ROW3    3   12   15
```

Matrix in R

In R, a two-dimensional rectangular data set is known as a matrix. A matrix is created with the help of the vector input to the matrix function. On R matrices, we can perform addition, subtraction, multiplication, and division operation.

In the R matrix, elements are arranged in a fixed number of rows and columns. The matrix elements are the real numbers. In R, we use matrix function, which can easily reproduce the memory representation of the matrix. In the R matrix, all the elements must share a common basic type.

```
#Syntax to create matrix
#matrix(data, nrow, ncol, byrow, dim_name)
matrix1<-matrix(c(11, 13, 15, 12, 14, 16),nrow =2, ncol =3, byrow = TRUE)
matrix1
Output
    [,1] [,2] [,3]
[1,]   11   13   15
[2,]   12   14   16
# Elements are arranged sequentially by row.
M <- matrix(c(1:12), nrow = 4, byrow = TRUE)
print(M)
```

```
Output
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12

# Elements are arranged sequentially by column.
N <- matrix(c(1:12), nrow = 4, byrow = FALSE)
print(N)
Output
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12

# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")
P <- matrix(c(1:12), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
Output
     col1 col2 col3
row1    1    2    3
row2    4    5    6
row3    7    8    9
row4   10   11   12
```

Accessing matrix elements

There are three ways to access the elements from the matrix.

1. We can access the element which presents on nth row and mth column.
2. We can access all the elements of the matrix which are present on the nth row.
3. We can also access all the elements of the matrix which are present on the mth column.

```
# Defining the column and row names.
row_names = c("row1", "row2", "row3", "row4")
col_names = c("col1", "col2", "col3")
#Creating matrix
R <- matrix(c(1:12), nrow = 4, byrow = TRUE, dimnames = list(row_names, col_names))
print(R)
 Output
      col1 col2 col3
row1    1    2    3
row2    4    5    6
row3    7    8    9
row4   10   11   12

#Accessing element present on 3rd row and 2nd column
print(R[3,2])
 Output
            [1] 8
```

```
#Accessing element present in 3rd row
print(R[3,])
Output
            col1 col2 col3
             7    8    9
  #Accessing element present in 2nd column
print(R[,2])
Output
            row1 row2 row3 row4
              2    5    8   11
```

Matrix Computations : Various mathematical operations are performed on the matrices using the R operators. The result of the operation is also a matrix. The dimensions (number of rows and columns) should be same for the matrices involved in the operation.

```
> # Create two 2x3 matrices.
> matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
> print(matrix1)
Output
     [,1] [,2] [,3]
[1,]    3   -1    2
[2,]    9    4    6
 > matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
> print(matrix2)
Output
     [,1] [,2] [,3]
[1,]    5    0    3
[2,]    2    9    4
> # Add the matrices.
> result <- matrix1 + matrix2
> cat("Result of addition","\n")
Result of addition
> print(result)
Output
     [,1] [,2] [,3]
[1,]    8   -1    5
[2,]   11   13   10
> # Subtract the matrices
> result <- matrix1 - matrix2
> cat("Result of subtraction","\n")
Result of subtraction
> print(result)
Output
     [,1] [,2] [,3]
[1,]   -2   -1   -1
[2,]    7   -5    2
> # Multiply the matrices.
> result <- matrix1 * matrix2
> cat("Result of multiplication","\n")
Result of multiplication
> print(result)
Output
     [,1] [,2] [,3]
[1,]   15    0    6
[2,]   18   36   24
> result <- matrix1 / matrix2
> cat("Result of division","\n")
Result of division
> print(result)
Output
     [,1]      [,2]      [,3]
[1,]  0.6      -Inf 0.6666667
[2,]  4.5 0.4444444 1.5000000
```

Applications of matrix

1. In geology, Matrices takes surveys and plot graphs, statistics, and used to study in different fields.
2. Matrix is the representation method which helps in plotting common survey things.
3. In robotics and automation, Matrices have the topmost elements for the robot movements.
4. Matrices are mainly used in calculating the gross domestic products in Economics, and it also helps in calculating the capability of goods and products.
5. In computer-based application, matrices play a crucial role in the creation of realistic seeming motion.

Data Frames in R

- A data frame is a two-dimensional structure similar to array or a table in which a column contains values of one variable, and rows contains one set of values from each column. A data frame is a special case of the list in which each component has equal length.
- A data frame is used to store data table and the vectors which are present in the form of a list in a data frame, are of equal length.
- In a simple way, it is a list of equal length vectors. A matrix can contain one type of data, but a data frame can contain different data types such as numeric, character, factor, etc.
- There are following characteristics of a data frame.
    - The columns name should be non-empty.
    - The rows name should be unique.
    - The data which is stored in a data frame can be a factor, numeric, or character type.
    - Each column contains the same number of data items

How to create data frame?

In R, the data frames are created with the help of frame() function of data. This function contains the vectors of any type such as numeric, character, or integer. In below example, we create a data frame that contains employee id (integer vector), employee name(character vector), salary(numeric vector), and starting date(Date vector).

```
# Creating the data frame.
emp.data<- data.frame(
employee_id = c (101:105),
employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
sal = c(40000, 35000, 20000, 25000, 30000),

starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-01",
    "2020-03-05")),
stringsAsFactors = FALSE
)
# Printing the data frame.
print(emp.data)
```

Output

	employee_id	employee_name	sal	starting_date
1	101	Ram	40000	2020-01-01
2	102	Sham	35000	2019-09-01
3	103	Neha	20000	2021-01-01
4	104	Siya	25000	2019-05-01
5	105	Sumit	30000	2020-03-05

Note : The argument 'stringsAsFactors' is an argument to the 'data.frame()' function in R. It is a logical that indicates whether strings in a data frame should be treated as factor variables or as just plain strings

Extracting data from data frame

The data of the data frame is very crucial for us. To manipulate the data of the data frame, it is essential to extract it from the data frame. We can extract the data in three ways which are as follows:

1. We can extract the specific columns from a data frame using the column name.

```
    # Creating the data frame.
    emp.data<- data.frame(
    employee_id = c (101:105),
    employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
    sal = c(40000, 35000, 20000, 25000, 30000),

    starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-
    01", "2020-03-05")),
    stringsAsFactors = FALSE
    )
# Extracting specific columns from a data frame
final <- data.frame(emp.data$employee_id,emp.data$sal)
print(final)
```

Output

	emp.data.employee_id	emp.data.sal
1	101	40000
2	102	35000
3	103	20000
4	104	25000
5	105	30000

2. We can extract the specific rows also from a data frame.

```
# Creating the data frame.
      emp.data<- data.frame(
      employee_id = c (101:105),
      employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
      sal = c(40000, 35000, 20000, 25000, 30000),
      starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-01", "2020-03-05")),
      stringsAsFactors = FALSE
      )
      # Extracting first row from a data frame
      final <- emp.data[1,]
      print(final)
```

Output

```
   employee_id employee_name   sal   starting_date
1        101          Ram     40000    2020-01-01
```

```
1.          # Extracting last two row from a data frame
2.          final <- emp.data[4:5,]
3.          print(final)
```

Output

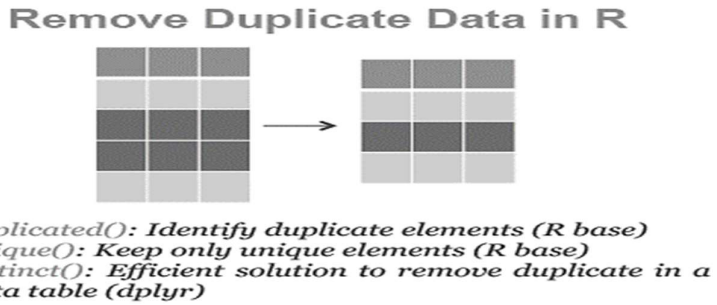
```
              employee_id employee_name   sal   starting_date
4        104          Siya      25000    2019-05-01
5        105         Sumit       30000    2020-03-05
```

3. We can extract the specific rows corresponding to specific columns

```
# Creating the data frame.
emp.data<- data.frame(
employee_id = c (101:105),
employee_name = c("Ram","Sham","Neha","Siya","Sumit"),
sal = c(40000, 35000, 20000, 25000, 30000),
starting_date = as.Date(c("2020-01-01", "2019-09-01", "2021-01-01", "2019-05-01", "2020-03-05")),
stringsAsFactors = FALSE
)
# Extracting 2nd and 3rd row corresponding to the 1st and 4th column
final <- emp.data[c(2,3),c(1,4)]
print(final)
```

Output

```
  employee_id starting_date
2        102    2019-09-01
3        103    2021-01-01
```



```
> companies <- data.frame(Shares = c("TCS", "Reliance", "HDFC Bank", "Infosys", "Reliance"),
+ Price = c(3200, 1900, 1500, 2200, 1900))
> companies
     Shares Price
1       TCS  3200
2  Reliance  1900
3 HDFC Bank  1500
4    Infosys 2200
5  Reliance  1900
> cat("After removing Duplicates ", "\n")
After removing Duplicates
> companies[duplicated(companies),]
    Shares Price
5 Reliance  1900
> unique(companies)
     Shares Price
1       TCS  3200
2  Reliance  1900
3 HDFC Bank  1500
4   Infosys  2200
```

What is Factors in R?

- The factor is a data structure which is used for fields which take only predefined finite number of values.
- These are the variable which takes a limited number of different values.
- These are the data objects which are used to store categorical data as levels.
- It can store both integers and strings values, and are useful in the column that has a limited number of unique values.
- By default R always sorts levels in alphabetical order.

Attributes of Factor

- **X**
  It is the input vector which is to be transformed into a factor.
- **levels**
  It is an input vector that represents a set of unique values which are taken by x.
- **labels**
  It is a character vector which corresponds to the number of labels.
- **Exclude**
  It is used to specify the value which we want to be excluded,
- **ordered**
  It is a logical attribute which determines if the levels are ordered.
- **nmax**
  It is used to specify the upper bound for the maximum number of level.

How to create factor in R ?
Factors are created using the **factor ()** function by taking a vector as input

Example :

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name)
Output

[1] Ram  Siya Raj  Sham Ram
Levels: Raj Ram Sham Siya
```

Notice that in the output, we have four levels “Raj”, “Ram” ,”Sham” and “Siya”. This does not contain the redundant “Ram” that was occurring twice in our input vector.

In order to add this missing level to our factors, we use the “levels” attribute as follows:

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name, levels=c(“Ram”,”Siya”,”Raj”,”Sham”))
Output

[1] Ram  Siya Raj  Sham Ram
Levels: Raj Ram Sham Siya
```

In order to provide abbreviations or ‘labels’ to our levels, we make use of the labels argument as follows –

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name, levels=c("Ram","Siya","Raj","Sham"),labels = c("R1","S1","R2","S2"))
Output

[1] R1 S1 R2 S2 R1
Levels: R1 S1 R2 S2
```

if you want to exclude any level from your factor, you can make use of the exclude argument.

```
# Create vector of names
stud_name<-c("Ram","Siya","Raj","Sham",”Ram”)
#Convert vector into factor
factor(stud_name, levels=c("Ram","Siya","Raj","Sham"),exclude="Ram")
Output
[1] <NA> Siya Raj  Sham <NA>
Levels: Siya Raj Sham
```

Accessing component of Factor in R

There are various ways to access the elements of a factor in R. Some of the ways are as follows:

```
# Create vector of compass
>compass <- c("East","West","East","North")
>data <- factor(compass)
>data
```

Output

```
[1] East  West  East  North
Levels: East North West

>data[4] #Accessing the 4th element
```

Output

```
[1] North
Levels: East North West

>data[c(2,3)] #Accessing the 2nd & 3rd element
```

Output

```
[1] West East
Levels: East North West
```

```
data[-1] #Accessing everything except 1st element
```

Output

```
[1] West  East  North
Levels: East North West

data[c(TRUE, FALSE, TRUE, TRUE)] #Accessing using Logical Vector
```

Output

```
[1] East  East  North
Levels: East North West
```

The levels() method of the specific factor vector can be used to extract the individual levels in the form of independent strings. It returns the number of elements equivalent to the length of the vector.

Syntax:
levels (fac-vec)[fac-vec]

Since, these levels are each returned as strings, therefore they belong to the same data type. Hence, they are combined now to form an atomic vector using the c() method.

Syntax:
c (vec1 , vec2)

where vec1 and vec2 are vectors belonging to same data type

```
> fac1 <- factor(letters[1:3])
> print ("Factor1 : ")
[1] "Factor1 : "
> print (fac1)
[1] a b c
Levels: a b c
> sapply(fac1,class)
[1] "factor" "factor" "factor"
> fac2 <- factor(c(1:4))
> print ("Factor2 : ")
[1] "Factor2 : "
> print (fac2)
[1] 1 2 3 4
Levels: 1 2 3 4
> sapply(fac2,class)
[1] "factor" "factor" "factor" "factor"
>
> # extracting levels of factor1
> level1 <- levels(fac1)[fac1]
>
> # extracting levels of factor2
> level2 <- levels(fac2)[fac2]
```

```
> # combine into one factor
> combined <- factor(c( level1,level2 ))
> print ("Combined Factor : ")
[1] "Combined Factor : "
> print (combined)
[1] a b c 1 2 3 4
Levels: 1 2 3 4 a b c
>
> sapply(combined,class)
[1] "factor" "factor" "factor" "factor" "factor" "factor" "factor"
```

Factor Functions in R

Some of these functions; *is.factor(), as.factor(), is.ordered(), as.ordered().*

- *is.factor()* checks if the input is present in the form of factor and returns a Boolean value (TRUE or FALSE).
- as.factor() takes the input (usually a vector) and converts it into a factor.
- is.ordered() checks if the factor is ordered and returns boolean TRUE or FALSE.
- The as.ordered() function takes an unordered function and returns a factor that is arranged in order.

Working with Dataset

1. **Dataset built into R**

This dataset is built into R ,to get information on the dataset is by typing ? before the dataset. Information will appear on the 4th panel under the Help tab.

```
?women
```

To see the whole dataset, use the command, View(*data_frame*). In this case, it would be View(women). A new tab in the Source panel called women will appear. You should see 15 rows and 2 columns of data entries.

2. **View part of dataset**

   If you want to see only a portion of the dataset, the function head( ) or tail( ) will do the job.

   The function head(*data_frame[,nL]*) will show the first 6 rows of the dataset. (nL : where is total number of rows you want followed by L)

```
head(women)
```

   The function tail(*data_frame*) will show the last 6 rows of the dataset.

```
tail(women)
```

3. **Viewing entries tied to the variable**
   Suppose we want to work on the data of women having height greater than 60

```
ans <- women[women$height>60,]
ans
```

4. **Ordering data frame by variable**
   Use the function called order( ) to order the data frame in descending or ascending order. The syntax in this case is: *data_frame*[order(*data_frame*$*variable*),]

   Suppose we want to order the women data in ascending order of weight of the women

```
data=women
sorted_data=data[order(data$weight),]
sorted_data
```

   To order the data frame, in descending order by weight of the women, put negative sign in front of the target vector.

```
data=women
sorted_data=data[order(-data$weight),]
sorted_data
```

R Data Visualization

In R, we can create visually appealing data visualizations by writing few lines of code. For this purpose, we use the diverse functionalities of R. Data visualization is an efficient technique for

gaining insight about data through a visual medium. With the help of visualization techniques, a human can easily obtain information about hidden patterns in data that might be neglected.

By using the data visualization technique, we can work with large datasets to efficiently obtain key insights about it.

Standard Graphics

R standard graphics are available through package graphics, include several functions which provide statistical plots, like:

- Scatterplots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is −

plot(x, y, main, xlab, ylab, xlim, ylim, axes)

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

```
input <- mtcars[,c('wt','mpg')]
print(head(input))
# Get the input values.
input <- mtcars[,c('wt','mpg')]

 # Give the chart file a name.
 png(file = "scatterplot.png")

 # Plot the chart for cars with weight between 2.5 to 5 and mileage #between 15 and 30.
 plot(x = input$wt,y = input$mpg,
     xlab = "Weight",
     ylab = "Milage",
     xlim = c(2.5,5),
     ylim = c(15,30),
     main = "Weight vs Milage"
 )

 # Save the file.
 dev.off()
```

Pie charts

The Pie charts are created with the help of pie () function, which takes positive numbers as vector input. Additional parameters are used to control labels, colors, titles, etc.

There is the following syntax of the pie() function:

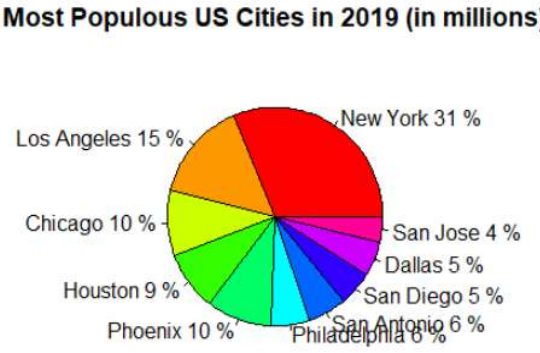
pie(X, Labels, Radius, Main, Col, Clockwise)

Here,

1. **X** is a vector that contains the numeric values used in the pie chart.
2. **Labels** are used to give the description to the slices.
3. **Radius** describes the radius of the pie chart.
4. **Main** describes the title of the chart.
5. **Col** defines the color palette.
6. **Clockwise** is a logical value that indicates the clockwise or anti-clockwise direction in which slices are drawn.

```
Cities <- c("New York", "Los Angeles", "Chicago", "Houston", "Phoenix",
+            "Philadelphia", "San Antonio", "San Diego", "Dallas", "San Jose")
> Population <- c(8.60, 4.06, 2.68, 2.40, 2.71, 1.58, 1.57, 1.45, 1.40, 1.03 )
> top_ten <- data.frame(Cities, Population)
> top_ten
pct <- round(100*top_ten$Population/sum(top_ten$Population))
> # Draw pie chart
> pie(top_ten$Population,
+     labels = paste(top_ten$Cities, sep = " ", pct, "%"),
+     col = rainbow(length(top_ten$Population)),
+     main = "Most Populous US Cities in 2019 (in millions)")
```

Output:



- **Bar plots**

  The barplot default is a vertical bar graph with black borders and gray filling. The bar graph is drawn in the same order as the data frame entries.

To add a title, labels on the axes and color to your bar graph, we use the following arguments.

- main = "*Header of the graph*"
- xlab = "*x-axis label*"
- ylab = "*y-axis label*"
- name.arg = *vector* (used for labelling each of the bar graphs)
- border = "*bar graph border color*"
- col = "*color to fill bar graph*"

```
barplot(height = IB$Users,
        main = "2018 Internet Browser Users (in million)",
        xlab = "Internet Browser",
        ylab = "Users",
        names.arg = IB$Browser,
        border = "dark blue",
        col = "pink")
```

```
Browser <- c("Chrome", "Edge", "Firefox", "IE",
+              "Opera", "Safari", "Others")
> Users <- c(2502.4, 150.78, 395.83, 238.05, 86.49, 387.65, 134.8)
> IB <- data.frame(Browser, Users)
> IB
  > barplot(height = IB$Users,
+         main = "2018 Internet Browser Users (in million)",
+         xlab = "Internet Browser",
+         ylab = "Users",
+         names.arg = IB$Browser,
+         border = "dark blue",
+         col = "pink")
```

Output: