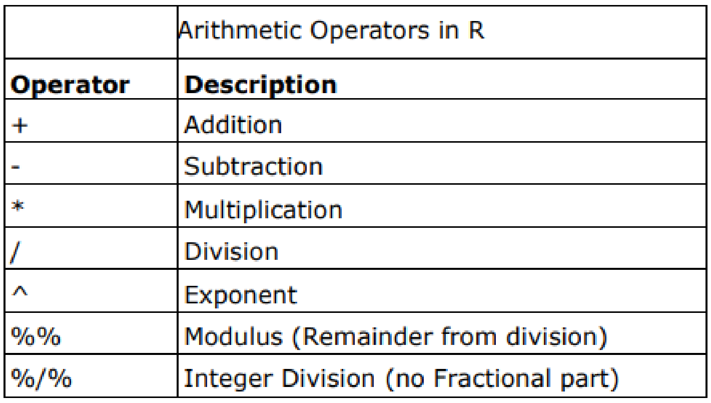
Exercise 1:

1a) Write a program to illustrate basic arithmetic in R

Arithmetic Operators:These operators are used to carry out mathematical operations like addition and multiplication. Here is a list of arithmetic operators available in R.



	Arithmetic Operators in R
Operator	**Description**
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division (no Fractional part)

Example:

```
> x <-5
> y <-16
 >x+y
[1]21
>x-y
 [1]-11
>x*y
[1]80
>y/x
[1]3.2
 >y%/%x
[1]3
>y%%x
[1]1
```

```
>y^x
[1]1048576
```

1b) Write a program to illustrate Variable assignment in R.

Variables in R :

Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.

Rules for writing Identifiers in R :

1. Identifiers can be a combination of letters, digits, period (.) and underscore (_).

2. It must start with a letter or a period. If it starts with a period, it cannot be followed by a digit.

3. Reserved words in R cannot be used as identifiers.

Valid identifiers in R :

total, Sum, .fine.with.dot, this_is_acceptable, Number5

Invalid identifiers in R :

tot@l, 5um, _fine, TRUE, .0ne

example :

```
var1 = "hello"

print(var1)

# using leftward operator

var2 < - "hello"

print(var2)

# using rightward operator

"hello" -> var3

print(var3)
```

output:

```
[1] "hello"
[1] "hello"
[1] "hello"
```

1c) Write a program to illustrate data types in R.

Data types in R:

To make the best of the R language, you'll need a strong understanding of the basic data types and data structures and how to operate on those.

• character

• numeric (real or decimal)

• integer

• logical

• complex

Example	**Type**
"a", "swc"	character
2, 15.5	numeric
2 (Must add a L at end to denote integer)	integer
TRUE, FALSE	logical
1+4i	complex

Example :

```
>TRUE
```

```
[1] TRUE
>class(TRUE)
[1] "logical"
>class(FALSE)
[1] "logical"
> T
[1] TRUE
> F
[1] FALSE
>class(2)
[1] "numeric"
>class(2L)
[1] "integer"
>class(2i)
[1] "complex"
>class("i love R programming")
[1] "character"
>class(2.5)
[1] "numeric"
>is.numeric(2)
[1] TRUE
>is.numeric(2.5)
 [1] TRUE
>is.numeric(22i)
[1] FALSE
>is.integer(22i)
[1] FALSE
```

```
>is.integer(2)
```

[1] FALSE

```
>class("integer is numeric numeric is not always integer")
```

[1] "character

Exercise 2:

2a) Write a program to illustrate if-else-else if in R

Decision making is an important part of programming. This can be achieved in R programming using the conditional if...else statement.

A) if statement :

Syntax of if statement :

```
if(test_expression ){ statement }
```

If the test_expression is TRUE, the statement gets executed. But if it's FALSE, nothing happens. Here, test_expression can be a logical or numeric vector, but only the first element is taken into consideration. In the case of numeric vector, zero is taken as FALSE, rest as TRUE.

Example of if statement.

```
 x <-5
if(x >0){
 print("Positive number")
 }
```

Output:

```
 [1] "Positive number"
```

B) if...else statement :

Syntax of if...else statement

```
if(test_expression){ statement1 }else{ statement2 }
```

The else part is optional and is evaluated if test_expression is FALSE. It is important to note that else must be in the same line as the closing braces of the if statements.

Example of if...else statement

```
 x <--5
if(x >0){
 print("Non-negative number")
} else{ print("Negative number")
 }
```

Output :

```
[1] "Negative number"
```

C) Nested else..if statement

Syntax of nested else if statement :

```
if( test_expression1){
 statement1
 } elseif( test_expression2) {
 statement2
} elseif( test_expression3) {
statement3
 } else
statement4
```

Only one statement will get executed depending upon the test_expressions.

Example of nested if...else statement :

```
x <-0
 if(x 0){
print("Positive number")
}else
 print("Zero")
```

Output :

```
[1] "Zero"
```

2b) Write a program to illustrate While and For Loops in R

While Loop :

In R programming, while loops are used to loop until a specific condition is met.

Syntax of while loop :

```
while(test_expression){
 statement
}
```

Here, test_expression is evaluated and the body of the loop is entered if the result is TRUE.
The statements inside the loop are executed and the flow returns to evaluate the
test_expression again. This is repeated each time until test_expression evaluates to FALSE, in which case, the loop exits.

Example of while loop :

```
i <-1
while(i <6) {
 print(i)
i = i+1
}
```

Output :

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

In the above example, iis initialized to 1. Here, the test_expression is i < 6 which evaluates to
TRUE since 1 is less than 6. So, the body of the loop is entered and is printed and incremented. Incrementing is important as this will eventually meet the exit condition. Failing to do so will result into an infinite loop. In the next iteration, the value of i is 2 and the loop continues. This will continue until itakes the value 6. The condition 6 < 6 will give FALSE and
the loop finally exits

For Loop :

A for loop is used to iterate over a vector, in R programming.

Syntax of for loop

```
for(valin sequence){
statement
}
```

Here, sequence is a vector and val takes on each of its value during the loop. In each iteration, statement is evaluated.

Example of for loop :

Below is an example to count the number of even numbers in a vector.

```
x <-c(2,5,3,9,8,11,6)
count<-0
for(valin x){
if(val%%2==0)
count = count+1
}
print(count)
```

Output

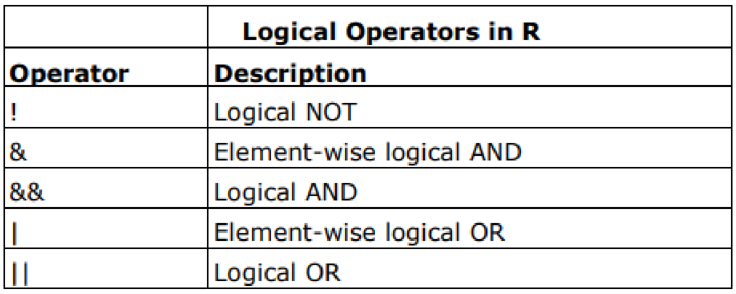
```
[1] 3
```

In the above example, the loop iterates 7 times as the vector x has 7 elements. In each
iteration, val takes on the value of corresponding element of x. We have used a counter to
count the number of even numbers in x. We can see that x contains 3 even numbers

2c) Write a program to illustrate Logical Operators in R?

Logical Operators:

Logical operators are used to carry out Boolean operations like AND, OR etc.



	Logical Operators in R
Operator	**Description**
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

Operators & and | perform element-wise operation producing result having length of the longer operand.

But && and || examines only the first element of the operands resulting into a single length logical vector.

Zero is considered FALSE and non-zero numbers are taken as TRUE.

An example run.

```
> x <-c(TRUE,FALSE,0,6)
> y <-c(FALSE,TRUE,FALSE,TRUE)
>!x
[1]FALSE TRUETRUE FALSE
>x&y
[1] FALSE FALSEFALSE TRUE
>x&&y
[1] FALSE
>x|y
[1] TRUETRUE FALSE TRUE
> x||y
[1] TRUE
```

Experiment 3:

3a) Write a program to illustrate Creating and Naming a Vector in R

Creating (Declaration of) a Vector:

Vectors are generally created using the **c()** function. Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different. Coercion is from lower to higher types from logical to integer to double to character.

Example:

```
> x <-c(1,5,4,9,0)
>typeof(x)
[1]"double"
>length(x)
[1]5
> x <-c(1,5.4, TRUE,"hello")
>x
[1]"1""5.4""TRUE""hello"
>typeof(x)
[1]"character"
```

If we want to create a vector of consecutive numbers, the : operator is very helpful. More complex

sequences can be created using the seq() function, like defining number of points in an interval, or the

step size.

```
> x <-1:7;
 x
[1] 1234567
> y <-2:-2;
 y
[1] 210-1-2
>seq(1,3,by=0.2) # specify step size
[1] 1.01.21.41.61.82.02.22.42.62.83.0
>seq(1,5,length.out=4) # specify length of the vector
[1] 1.0000002.3333333.6666675.000000
```

Naming a Vector in R :

The Naming of the Vector is Based on the Variable Creation ,While Creating vector name we have to follow the Rules of Variables.

Example:

```
> x <-5
> variable <-16
> a1 <- 79
```

3b) Write a program to illustrate create a matrix and naming matrix in R.

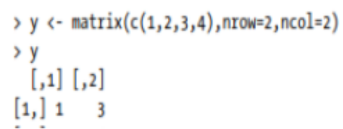
Creating Matrices: A Matrix is created using the matrix() function.

Dimension of the matrix can be defined by passing appropriate value for arguments nrow and ncol. Providing value for both the dimension is not necessary.

Syntax: matrix(data, nrow, ncol, byrow)

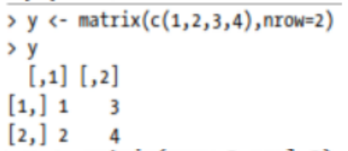
Following is the description of the parameters used −

- **data** is the input vector which becomes the data elements of the matrix.
- **nrow** is the number of rows to be created.
- **ncol** is the number of columns to be created.
- **byrow** is a logical clue. If TRUE then the input vector elements are arranged by row



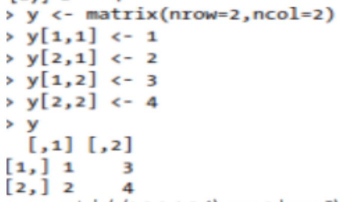
```
> y <- matrix(c(1,2,3,4),nrow=2,ncol=2)
> y
     [,1] [,2]
[1,]    1    3
```

Now y is a vector with 1,2,3,4 as elements having 2 rows and 2 columns.



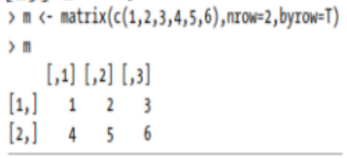
```
> y <- matrix(c(1,2,3,4),nrow=2)
> y
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

If one of the dimensions is provided, the other is inferred from length of the data.



```
> y <- matrix(nrow=2,ncol=2)
> y[1,1] <- 1
> y[2,1] <- 2
> y[1,2] <- 3
> y[2,2] <- 4
> y
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

We can also assign the elements into the matrix by using indexing. But before that we need to declare that as matrix.



```
> m <- matrix(c(1,2,3,4,5,6),nrow=2,byrow=T)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

By default matrix is column wise. If we want to declare as the row wise we need to mention it while declaring 'byrow=T'.

naming matrix in R:

The Naming of the Matrix is Based on the Variable Creation ,While Creating Matrix name we have to follow the Rules of Variables.

Example:

> x <- matrix(nrow=5,ncol=2)

> variable <-matrix(c(1,2,3,4,5),nrows=2,ncol=2)

> a1 <- matrix(c(1,2,3,4,5),nrows=2)

3c) Write a program to illustrate Add Column and Add Row in Matrix in R.

If we want to Add row and column to the matrix .We have to use nrow and ncol in variable declaration.

```
#Creating a Matrix
MatrixA <- matrix(data = 1:9, nrow = 3, ncol = 3)
#Printing Matrix
MatrixA
```

Output

```
>MatrixA
[,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
```

We use function rbind() to add the row to any existing matrix. We use function cbind() to add the column to any existing matrix.

rbind():

```r
#Creating a new Matrix using rbind()
MatrixB <- rbind(MatrixA, c(10,11,12))
#Printing that Matrix
MatrixB
```

Output

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

Here, a new matrix named MatrixB has been created which is the combination of a new row with values 10, 11, and 12 in the previous matrix with the name MatrixA

cbind():

```r
#Creating a new Matrix using cbind()
MatrixC <- cbind(MatrixA, c(10, 11, 12))
#Printing Matrix
MatrixC
```

Output:

```
> Matrix C
[,1] [,2] [,3] [,4]
[1,] 1 4 7 10
```

```
[2,] 2 5 8 11
[3,] 3 6 9 12
```

3d) Write a program to illustrate Selection of elements in matrices in R

Selection of matrix elements

Similar to vectors, you can use the square brackets [ ] to select one or multiple elements from a matrix. Whereas vectors have one dimension, matrices have two dimensions. You should therefore use a comma to separate that what to select from the rows from that what you want to select from the columns. For example:

my_matrix[1,2] selects from the first row the second element.

my_matrix[1:3,2:4] selects rows 1,2,3 and columns 2,3,4.

If you want to select all elements of a row or a column, no number is needed before or after the comma:

my_matrix[,1] selects all elements of the first column.

my_matrix[1,] selects all elements of the first row.

Example:

```
MatrixA <-
 matrix(data = 1,2,3,4,5,6,7,8,9, nrow = 3, ncol = 3)
```

```
MatrixA[1:3,2:4]
```

Output:

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

Experiment 4:

4a) How to create a Matrix in R.

Matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know 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:

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

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.

Example:

```
A = matrix(

c(1, 2, 3, 4, 5, 6, 7, 8, 9),

nrow = 3,

ncol = 3,

byrow = TRUE

)

# Naming rows
```

rownames(A) = c("a", "b", "c")

# Naming columns

colnames(A) = c("c", "d", "e")

cat("The 3x3 matrix:\n")

print(A)

Output:

```
The 3x3 matrix:
  c d e
a 1 2 3
b 4 5 6
c 7 8 9
```

4b) Print Dimension of the matrix with dim()

The **dim()** function in R is used to get or set the dimensions of a matrix, array, or data frame. It returns the number of rows and columns as a vector or sets them if you assign a value. For example, dim(m) returns the dimensions of m, and dim(m) <- c(3, 4) changes m to have 3 rows and 4 columns.

Syntax :

# To get the dimension value

dim(data)

# To set the dimension value

dim(data) <- value

Example:

Dimension of matrix can be modified as well, using the dim() function.

>x

o/p:

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

```
[1,]135
[2,]246
>dim(x)<- c(3,2);
 x # change to 3X2 matrix
```

o/p:

```
[,1][,2]
[1,]14
[2,]25
[3,]36
>dim(x)<- c(1,6);
x # change to 1X6 matrix
```

o/p:

```
[,1][,2][,3][,4][,5][,6]
[1,]123456
```

4c) Construct a matrix with 5 rows that contains the numbers 1 up to 10 and by row = FALSE

We can create a matrix with the function matrix(). Following is a function to create a matrix in R which takes three arguments:

matrix(data, nrow, ncol, byrow = FALSE)

Arguments:

data: The collection of elements that R will arrange into the rows and columns of the matrix

nrow: Number of rows

ncol: Number of columns

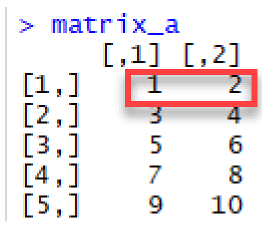
byrow: The rows are filled from the left to the right. We use `byrow = FALSE` (default values), if we want the matrix to be filled by the columns i.e. the values are filled top to bottom.

Example:

matrix_b <-matrix(1:10, byrow = FALSE, nrow = 5)

matrix_b

Output:



```
> matrix_a
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
[4,]    7    8
[5,]    9   10
```

4e) Add a Column to a Matrix with the cbind().

Use the `cbind()` function to add additional columns in a Matrix:

Example:

thismatrix <- matrix(c("apple", "banana", "cherry", "orange","grape", "pineapple", "pear", "melon", "fig"), nrow = 3, ncol = 3)

newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry"))

# Print the new matrix

Newmatrix

Ouput:

```
     [,1]    [,2]        [,3]    [,4]
[1,] "apple"  "orange"    "pear"  "strawberry"
[2,] "banana" "grape"     "melon" "blueberry"
[3,] "cherry" "pineapple" "fig"   "raspberry"
```

4f) Slice a Matrix

We can select elements one or many elements from a matrix in R programming by using the square brackets [ ]. This is where slicing comes into the picture.

For example:

matrix_c[1,2] selects the element at the first row and second column.

matrix_c[1:3,2:3] results in a R slice matrix with the data on the rows 1, 2, 3 and columns 2, 3,

matrix_c[,1] selects all elements of the first column.

matrix_c[1,] selects all elements of the first row.

4g) Write a program to illustrate Compare Matrices and Compare Vectors

A matrix is nothing but a vector but with two additional attributes: number of rows and columns. Let us check the class of the matrix.

`> length(z)`
`[1] 8`
	We can get the length of the matrix using length() function.

`> class(z)`
`[1] "matrix"`
`> attributes(z)`
`$dim`
`[1] 4 2`
	In addition to the length matrix has addition attributes. To get that we use attributes() function to get the dimensions of the matrix i.e., no of rows and columns.
	To know the class of the matrix we use class() function.

`> dim(z)`
`[1] 4 2`
	We can get the number of rows and columns of the matrix by just dim() function. Here in z there are 4 rows and 2 columns are there.

`> nrow(z)`
`[1] 4`
`> ncol(z)`
`[1] 2`
	In order to get either number of rows or number of columns we have ncol() and nrow() functions.

Exercise 5 :

5a) Write a program to illustrate Factors in R.

Factors in R Programming Language are data structures that are implemented to categorize the data or represent categorical data and store it on multiple levels.
They can be stored as integers with a corresponding label to every unique integer. Though factors may look similar to character vectors, they are integers and care must be taken while using them as strings. The factor accepts only a restricted number of distinct values.

Attributes of Factors in R Language

- **x:** It is the vector that needs to be converted into a factor.

- **Levels:** It is a set of distinct values which are given to the input vector x.
- **Labels:** It is a character vector corresponding to the number of labels.
- **Exclude:** This will mention all the values you want to exclude.
- **Ordered:** This logical attribute decides whether the levels are ordered.
- **nmax:** It will decide the upper limit for the maximum number of levels.

. Example:

```
x < -c("female", "male", "male", "female")

print(x)

# Converting the vector x into a factor

# named gender

gender < -factor(x)

print(gender)
```

Output:

```
[1] "female" "male"   "male"   "female"

[1] female male   male   female

Levels: female male
```

5b) Case Study of why you need use a factor in R.

The term factor refers to a statistical data type used to store categorical variables. The difference between a categorical variable and a continuous variable is that a categorical variable can belong to a **limited number of categories**. A continuous variable, on the other hand, can correspond to an infinite number of values.

It is important that R knows whether it is dealing with a continuous or a categorical variable, as the statistical models you will develop in the future treat both types differently. (You will see later why this is the case.)

A good example of a categorical variable is sex. In many circumstances you can limit the sex categories to "Male" or "Female". (Sometimes you may need different categories. For example, you may need to consider chromosomal variation, hermaphroditic animals, or different cultural norms, but you will always have a finite number of categories.)

5c) Write a program to illustrate Ordered Factors in R.

Ordering Factor Levels

Ordered factors is an extension of factors. It arranges the levels in increasing order. We use two functions: factor() along with argument ordered().

Syntax: factor(data, levels =c(“”), ordered =TRUE)

Parameter:

data: input vector with explicitly defined values.

levels(): Mention the list of levels in c function.

ordered: It is set true for enabling ordering.

Example:

```
# creating size vector

size = c("small", "large", "large", "small","medium", "large", "medium",
"medium")

# converting to factor

size_factor <- factor(size)

print(size_factor)

# ordering the levels
```

```
ordered.size <- factor(size, levels = c(
  "small", "medium", "large"), ordered = TRUE)
print(ordered.size)
```

Output:

```
[1] small  large  large  small  medium large  medium medium
Levels: large medium small
[1] small  large  large  small  medium large  medium medium


Levels: small < medium < large
```

Exercise-6:

6a) How to create a Data Frame.

Data frame is like a matrix, with a two dimensional rows and columns structure. However it differ from matrix that each column may have different modes. For instance one column may contains number another may contains character strings. A data frame is a list, with the components of that list being equal-length vectors.

Creating a Data Frame :

We can create a data frame using the data.frame() function. For example, the above shown data

frame can be created as follows.

```
> x <-data.frame("SN"=1:2,"Age"=c(21,15),"Name"=c("John","Dora"))

>str(x) # structure of x
```

o/p:

```
'data.frame':2 obs.of 3 variables:
```

$ SN :int12

$ Age:num2115

$ Name:Factor w/2 levels "Dora","John":21

Notice above that the third column, Name is of type factor, instead of a character vector. By

default, data.frame() function converts character vector into factor. To suppress this behavior, we

can pass the argument stringsAsFactors=FALSE.

6b) Slice DataFrame

Slicing the Data Frame

Slicing the Data Frame gives the required rows and columns. This can be done by three ways. They are listed below-

Slicing with [,]

Slicing with logical vectors.

Slicing with subset().

Slicing with [,]

Slicing the data frame with [,] returns data of specified rows and columns. **The syntax of this is mentioned below-**

dataframeName[fromRow : toRow , columnNumber]

Example:

# create a data frame

```
stats <- data.frame(player=c('A', 'B', 'C', 'D'),
                runs=c(100, 200, 408, NA),
                wickets=c(17, 20, NA, 5))
print("stats Dataframe")
stats
# fetch 2,3 rows and 1,2 columns
stats[2:3,c(1,2)]
# fetch 1:3 rows of 1st column
cat("players - ")
stats[1:3,1]
```

Output:

```
"stats Dataframe"
  player runs wickets
1      A  100      17
2      B  200      20
3      C  408      NA
4      D   NA       5
  player runs
2      B  200
3      C  408
players - [1] "A" "B" "C"
```

6c) Append a Column to Data Frame

We can add a column to a data frame using $ symbol.

syntax:

 dataframe_name $ column_name = c(value 1,value 2 . . . , value n)

Here c() function is a vector holds values .we can pass any type of data with similar type.

Steps for adding a column to a dataframe.

1.Create a data frame.

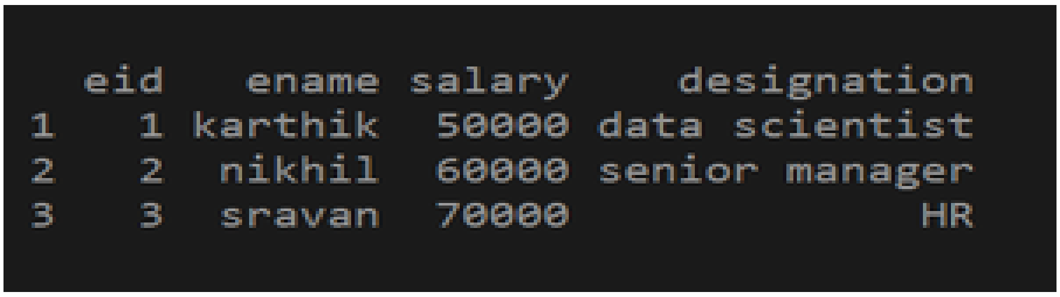
2.Use the $ symbol as shown in the above syntax to add a column to a data frame.

3.Print the updated data frame to see the changes.

Example:

```
df2 = data.frame(eid = c(1, 2, 3),

                 ename = c("karthik", "nikhil", "sravan"),

                 salary = c(50000, 60000, 70000))

  df2$designation = c("data scientist", "senior manager", "HR")

print(df2)
```

Output:



```
  eid   ename salary    designation
1   1 karthik  50000 data scientist
2   2  nikhil  60000 senior manager
3   3  sravan  70000             HR
```

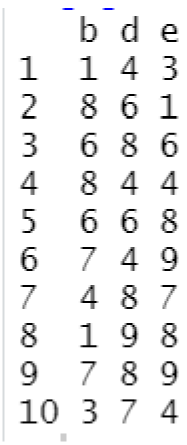
6d) Select a Column To DataFrame

Method 1: Selecting specific Columns Using Base R by column name
In this approach to select a specific column, the user needs to write the name of the column name in the square bracket with the name of the given data frame as per the requirement to get those specific columns needed by the user.

Example:

```
gfg < - data.frame(a=c(5, 1, 1, 5, 6, 7, 5, 4, 7, 9),

                   b=c(1, 8, 6, 8, 6, 7, 4, 1, 7, 3),

                   c=c(7, 1, 8, 9, 4, 1, 5, 6, 3, 7),

                   d=c(4, 6, 8, 4, 6, 4, 8, 9, 8, 7),

                   e=c(3, 1, 6, 4, 8, 9, 7, 8, 9, 4)

gfg[c('b', 'd', 'e')]
```

Output:



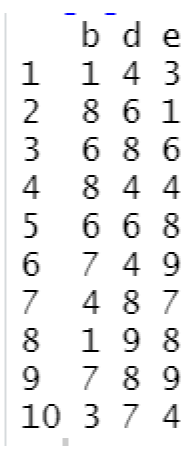
```
   b d e
1  1 4 3
2  8 6 1
3  6 8 6
4  8 4 4
5  6 6 8
6  7 4 9
7  4 8 7
8  1 9 8
9  7 8 9
10 3 7 4
```

Method 2: Selecting specific Columns Using Base R by column index
In this approach to select the specific columns, the user needs to use the square brackets with the data frame given, and. With it, the user also needs to use the index of columns inside of the square bracket where the indexing starts with 1, and as per the requirements of the user has to give the required column index to inside the brackets

Example:

```
gfg < - data.frame(a=c(5, 1, 1, 5, 6, 7, 5, 4, 7, 9),
                   b=c(1, 8, 6, 8, 6, 7, 4, 1, 7, 3),
                   c=c(7, 1, 8, 9, 4, 1, 5, 6, 3, 7),
                   d=c(4, 6, 8, 4, 6, 4, 8, 9, 8, 7),
                   e=c(3, 1, 6, 4, 8, 9, 7, 8, 9, 4))
  gfg[c(2, 4, 5)]
```

Output:



```
   b d e
1  1 4 3
2  8 6 1
3  6 8 6
4  8 4 4
5  6 6 8
6  7 4 9
7  4 8 7
8  1 9 8
9  7 8 9
10 3 7 4
```

6e) Subset a Data Frame

By Using subset function we can skip the NA values in the data frame same like that of na.rm=true.

When looking to create more complex subsets or a subset based on a condition, the next step up is to use the **subset()** function. For example, what if you wanted to look at debt from someone named Dan. You could just use the brackets to select their debt and total it up, but it isn't a very robust way of doing things, especially with potential changes to the data set.

Example:

```
# This works, but is not informative nor robust

debt[1:3, ]

# Much more informative!
```

subset(debt, name == "Dan")

Output:

```
     name    payment

1     Dan       100

2     Dan       200

3     Dan       150
```

Accessing and Subsetting Dataframes

Moving to this next example, what if you are only interested in the cash flows from company A?

subset(cash, company == "A")

Output:

```
         Company           cash_flow            year

1         A                  1000              1

2         A                  4000              3

3         A                   550            4
```

6f) Write a program to illustrate Data Frames Selection of elements in a DataFrame

Similar to vectors and matrices, you select elements from a data frame with the help of square brackets [ ]. By using a comma, you can indicate what to select from the rows and the columns respectively.

For example:

my_df[1,2] selects the value at the first row and second column in my_df.

my_df[1:3,2:4] selects rows 1, 2, 3 and columns 2, 3, 4 in my_df.

Sometimes you want to select all elements of a row or column. For example, my_df[1,] selects all elements of the first row. Let us now apply this technique on planets_df!

Example:

```
df<-data.frame(11:13,14:16,17:19,20:22)

  df

  c(df[df==11], df[df==13],df[df==22])
```

Output:

```
 #      X11.13     X14.16     X17.19    X20.22

 #  1     11            14           17          20

 #  2     12            15          18           21

 #  3     13            16           19          22

         >[1] 11 13 22
```

6g) Write a program to illustrate Sorting a DataFrame

Using order() function

This function is used to sort the dataframe based on the particular column in the dataframe

Syntax: *order(dataframe$column_name,decreasing = TRUE))*

Example:

```
data = data.frame(rollno = c(1, 5, 4, 2, 3),subjects = c("java", "python", "php",
"sql","c"))

print(data)

print("sort the data in decreasing order based on subjects ")
```

print(data[order(data$subjects, decreasing = TRUE),]

print("sort the data in decreasing order based on rollno ")

print(data[order(data$rollno, decreasing = TRUE),]   )

Output:

```
rollno subjects
1      1     java
2      5   python
3      4      php
4      2      sql
5      3        c
[1] "sort the data in decreasing order based on subjects
"
  rollno subjects
4      2      sql
2      5   python
3      4      php
1      1     java
5      3        c
[1] "sort the data in decreasing order based on rollno "
  rollno subjects
2      5   python
3      4      php
5      3        c
4      2      sql
1      1     java
```

6h) Merge DataFrames in R: Full and Partial Match

Full Match:

A full match returns values that have a counterpart in the destination table. The values that are not match won't be return in the new data frame. The partial match, however, return the missing values as NA.

Example:

```
producers <- data.frame(
    surname =
c("Spielberg","Scorsese","Hitchcock","Tarantino","Polansk
i"),
    nationality = c("US","US","UK","US","Poland"),
    stringsAsFactors=FALSE)
movies <- data.frame(
    surname = c("Spielberg",
            "Scorsese",
                "Hitchcock",
                 "Hitchcock",
                "Spielberg",
                "Tarantino",
                "Polanski"),
    title = c("Super 8",
            "Taxi Driver",
            "Psycho",
            "North by Northwest",
            "Catch Me If You Can",
            "Reservoir Dogs","Chinatown"),
            stringsAsFactors=FALSE)
m1 <- merge(producers, movies, by.x = "surname")
m1
dim(m1)
```

Output:

```
surname          nationality      title
```

```
1 Hitchcock       UK          Psycho
2 Hitchcock       UK          North by Northwest
3 Polanski        Poland            Chinatown
4 Scorsese        US          Taxi Driver
5 Spielberg       US          Super 8
6 Spielberg       US          Catch Me If You Can
7 Tarantino       US          Reservoir Dogs
```

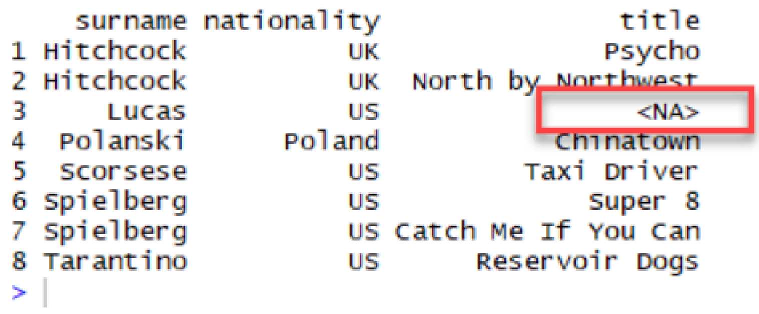
Partial match:

It is not surprising that two dataframes do not have the same common key variables. In the full matching, the dataframe returns only rows found in both x and y data frame. With partial merging, it is possible to keep the rows with no matching rows in the other data frame. These rows will have NA in those columns that are usually filled with values from y. We can do that by setting all.x= TRUE.

Example:

```
add_producer <-  c('Lucas', 'US')

producers <- rbind(producers, add_producer)

m3 <-merge(producers, movies, by.x = "surname", by.y = "name", all.x = TRUE)

m3
```

Output:



```
    surname nationality               title
1 Hitchcock          UK              Psycho
2 Hitchcock          UK  North by Northwest
3     Lucas          US                <NA>
4  Polanski      Poland           Chinatown
5  Scorsese          US         Taxi Driver
6 Spielberg          US             Super 8
7 Spielberg          US Catch Me If You Can
8 Tarantino          US      Reservoir Dogs
> 
```

Exercise 7:

7a) Write a program to illustrate List? Why would you need a list.

A list in R is a generic object consisting of an ordered collection of objects.

Example:

```
empId = c(1, 2, 3, 4)

empName = c("Debi", "Sandeep", "Subham", "Shiba")

numberOfEmp = 4

empList = list(empId, empName, numberOfEmp)

print(empList)
```

Output:

```
[[1]]

[1] 1 2 3 4

[[2]]

[1] "Debi"    "Sandeep" "Subham"  "Shiba"

[[3]]

[1] 4
```

Need of List in R

Lists are one-dimensional, heterogeneous data structures. The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.

A list is a vector but with heterogeneous data elements. A list in R is created with the use of list() function. R allows accessing elements of a list with the use of the index value. In R, the indexing of a list starts with 1 instead of 0 like other programming languages.

7b) Write a program to illustrate Adding more elements into a list?

Appending Data to the List

Appending to a list means adding a value to the last of an already existing list. We are going to append two lists by using append() function

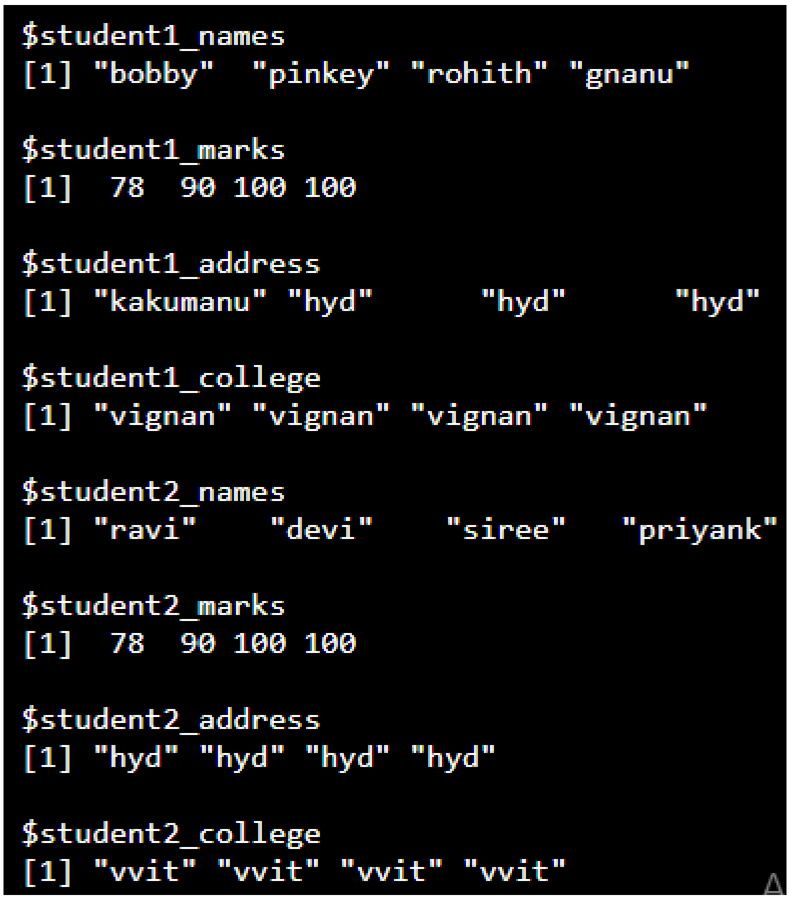
Syntax: append(list1,list2)

Example:

```
# vector with names
names=c("bobby","pinkey","rohith","gnanu")

# vector with marks
marks=c(78,90,100,100)

# address vector
address=c("kakumanu","hyd","hyd","hyd")

# college values
college=c("vignan","vignan","vignan","vignan")

# pass these vectors as inputs to the list
student1=list(student1_names=names,student1_marks=marks,
student1_address=address,student1_college=college)

# vector with names
names=c("ravi","devi","siree","priyank")

# vector with marks
marks=c(78,90,100,100)

# address vector
```

```
address=c("hyd","hyd","hyd","hyd")

# college values
college=c("vvit","vvit","vvit","vvit")

# pass these vectors as inputs to the list
# address vector
student2=list(student2_names=names,student2_marks=marks,
student2_address=address,student2_college=college)

# append list 1 and list 2
print(append(student1,student2))
```

Output:



```
$student1_names
[1] "bobby"   "pinkey" "rohith"  "gnanu"

$student1_marks
[1]  78  90 100 100

$student1_address
[1] "kakumanu" "hyd"      "hyd"      "hyd"

$student1_college
[1] "vignan" "vignan" "vignan" "vignan"

$student2_names
[1] "ravi"    "devi"    "siree"   "priyank"

$student2_marks
[1]  78  90 100 100

$student2_address
[1] "hyd" "hyd" "hyd" "hyd"

$student2_college
[1] "vvit" "vvit" "vvit" "vvit"
```

Exercise 8:

8a) Write a program to illustrate Function Inside Function In R.

A function is a set of statements organized together to perform a specific task. R has a large

number of in-built functions and the user can create their own functions.

Function Definition

An R function is created by using the keyword function. The basic syntax of an R function definition is as follows –

```
function_name<-function(arg_1, arg_2,...){

Function body }
```

Function Components

The different parts of a function are –

• **Function Name** – This is the actual name of the function. It is stored in R environment as an object with this name.

• **Arguments** – An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.

• **Function** Body – The function body contains a collection of statements that defines what the function does. Return Value – The return value of a function is the last expression in the function body to be evaluated.

Example:

```
Outer_func <- function(x) {

  Inner_func <- function(y) {

    a <- x + y
    return(a)
  }
  return (Inner_func)
}
output <- Outer_func(3) # To call the Outer_func
output(5)
```

Output:

```
[1] 8
```

8b) **Write a program to illustrate some built in Mathematical Functions.**

Built-in Math Functions

R also has many built-in math functions that allows you to perform mathematical tasks on numbers.

For example, the **min()** and **max()** functions can be used to find the lowest or highest number in a set:

```
>max(5, 10, 15)
[1] 15
>min(5, 10, 15)
[1] 5
```

sqrt()

The sqrt() function returns the square root of a number:

```
> sqrt(16)
[1] 4
```

abs()

The abs() function returns the absolute (positive) value of a number:

```
>abs(-4.6)
[1] 4.7
```

ceiling() and floor()

The ceiling() function rounds a number upwards to its nearest integer, and the floor() function rounds a number downwards to its nearest integer, and returns the result:

```
ceiling(1.4)
[1] 2
floor(1.4)
[1] 1
```

8c) Write a program to calculate mean,mode,SD and variance.

Mean: Calculate sum of all the values and divide it with the total number of values in the data set.

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
> mean.result = mean(x) # calculate mean
> print (mean.result)
[1] 2.8
```

Mode: The most occurring number in the data set. For calculating mode, there is no default function in R. So, we have to create our own custom function

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
> mode.result = mode(x)
> print (mode.result)
[1] 1
```

Variance: How far a set of data values are spread out from their mean.

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
> variance.result = var(x) # calculate variance
> print (variance.result)
[1] 2.484211
```

Standard Deviation: A measure that is used to quantify the amount of variation or dispersion of a set of data values.

```
x <- c(1,2,3,4,5,1,2,3,1,2,4,5,2,3,1,1,2,3,5,6)
> sd.result = sqrt(var(x)) # calculate standard deviation
> print (sd.result)
[1] 1.576138
```