Topic 1: Introduction to R language

2023-08-08

In this topic in week 2 , you will learn about :

- Data frames and list
- Extracting elements from vectors

The readings in this tutorial follow *R for Data Science*, section 5.2.

Objects in R

Objectives

- To introduce data types in R and show how these data types are used in data structures.
- Learn how to create vectors of different types.
- Be able to check the type of vector.
- Learn about missing data and other special values.
- Get familiar with the different data structures (lists, matrices, data frames).

Everything in R is an object.

R has 5 basic data types.

- character
- numeric (real or decimal)
- integer
- logical
- complex

A variable can store different types of values such as numbers, characters etc. The **variables are assigned with R-Objects** and the **data type of the R-object** becomes the **data type of the variable.**

Elements of these data types may be combined to form data structures, such as atomic vectors. When we call a vector atomic, we mean that the vector only holds data of a single data type.

R provides many functions to examine features of vectors and other objects, for example

class( ) - what kind of object is it (high-level)?

```r
x <- 3L
class(x)
```

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

typeof( ) - what is the object's data type (low-level)?

```r
x <- 3L
typeof(x)
```

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

length( ) - how long is it? What about two dimensional objects?

```r
x <- 3L
length(x)
```

```
## [1] 1
```

attributes( ) - does it have any metadata?

```r
x <- 3L
attributes(x)
```

```
## NULL
```

Below are examples of atomic character vectors, numeric vectors, integer vectors, etc.

- character: <“a”>, “abc”
- numeric: 3, 13.5
- integer: (the L tells R to store this as an integer)
- logical: TRUE, FALSE
- complex: 1+3i (complex numbers with real and imaginary parts)

R has many data structures. The frequently used ones are:

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Vectors

A vector is a basic data structure which plays an important role in R programming. In R, a sequence of elements which share the same data type is known as vector. A vector supports logical, integer, double, character, complex, or raw data type.

```r
# Atomic vector of type character.
print("ali");
```

```
## [1] "ali"
```

```r
# Atomic vector of type double.
print(2.5)
```

```
## [1] 2.5
```

```r
# Atomic vector of type integer.
print(3L)
```

```
## [1] 3
```

```r
# Atomic vector of type logical.
print(TRUE)
```

```
## [1] TRUE
```

```r
# Atomic vector of type complex.
print(1+3i)
```

```
## [1] 1+3i
```

```r
# Atomic vector of type raw.
print(charToRaw('hello'))
```

```
## [1] 68 65 6c 6c 6f
```

1. Multiple Elements Vector.

Using colon operator (:) with numeric data

```r
# Creating a sequence from 4 to 23.
v <- 4:12
print(v)
```

```
## [1]  4  5  6  7  8  9 10 11 12
```

```r
# Creating a sequence from 3.6 to 9.6.
v <- 3.6:9.6
print(v)
```

```
## [1] 3.6 4.6 5.6 6.6 7.6 8.6 9.6
```

```r
# If the final element specified does not belong to the sequence then it is discarded.
v <- 4.8:10.4
print(v)
```

```
## [1] 4.8 5.8 6.8 7.8 8.8 9.8
```

Using sequence (seq.) operator

```r
# Create vector with elements from 5 to 9 incrementing by 0.4.
print(seq(5, 9, by = 0.4))
```

```
##  [1] 5.0 5.4 5.8 6.2 6.6 7.0 7.4 7.8 8.2 8.6 9.0
```

Using the c( ) function.

The non-character values are coerced to character type if one of the elements is a character.

```r
# The logical and numeric values are converted to characters.
s <- c('apple','red',4,TRUE)
print(s)
```

```
## [1] "apple" "red"   "4"     "TRUE"
```

2. Accessing Vector Elements

Elements of a Vector are accessed using indexing. The [ ] **brackets** are used for indexing. Indexing starts with position 1. Giving a negative value in the index drops that element from result.**TRUE, FALSE** or **0** and **1** can also be used for indexing.

```r
# Accessing vector elements using position.
t <- c("Sun","Mon","Tue","Wed","Thurs","Fri","Sat")
u <- t[c(1,2,5)]
print(u)
```

```
## [1] "Sun"   "Mon"   "Thurs"
```

```r
# Accessing vector elements using logical indexing.
v <- t[c(TRUE,FALSE,FALSE,FALSE,FALSE,TRUE,FALSE)]
print(v)
```

```
## [1] "Sun" "Fri"
```

```r
# Accessing vector elements using negative indexing.
x <- t[c(-3,-6)]
print(x)
```

```
## [1] "Sun"   "Mon"   "Wed"   "Thurs" "Sat"
```

```r
# Accessing vector elements using 0/1 indexing.
y <- t[c(0,0,0,0,0,0,1)]
print(y)
```

```
## [1] "Sun"
```

3. Vector Manipulation.

Vector arithmetic.

Two vectors of same length can be added, subtracted, multiplied or divided giving the result as a vector output.

```r
# Create two vectors.
v1 <- c(2,8,4,4,0,10)
v2 <- c(3,11,0,7,1,3)

# Vector addition.
add <- v1+v2
print(add)
```

```
## [1]  5 19  4 11  1 13
```

```r
# Vector subtraction.
sub <- v1-v2
print(sub)
```

```
## [1] -1 -3  4 -3 -1  7
```

```r
# Vector multiplication.
multi <- v1*v2
print(multi)
```

```
## [1]  6 88  0 28  0 30
```

```r
# Vector division.
divi <- v1/v2
print(divi)
```

```
## [1] 0.6666667 0.7272727       Inf 0.5714286 0.0000000 3.3333333
```

Vector Element Recycling

If we apply arithmetic operations to two vectors of unequal length, then the elements of the shorter vector are recycled to complete the operations.

```r
v1 <- c(3,8,4,5,0,11)
v2 <- c(4,11)
# V2 becomes c(4,11,4,11,4,11)

add.result <- v1+v2
print(add.result)
```

```
## [1]  7 19  8 16  4 22
```

```r
sub.result <- v1-v2
print(sub.result)
```

```
## [1] -1 -3  0 -6 -4  0
```

Vector Element Sorting

Elements in a vector can be sorted using the sort() function.

```r
v <- c(3,8,4,5,0,11, -9, 304)

# Sort the elements of the vector.
sort1 <- sort(v)
print(sort1)
```

```
## [1]  -9   0   3   4   5   8  11 304
```

```r
# Sort the elements in the reverse order.
revsort <- sort(v, decreasing = TRUE)
print(revsort)
```

```
## [1] 304  11   8   5   4   3   0  -9
```

```r
# Sorting character vectors.
v <- c("Red","Blue","yellow","violet")
sort2 <- sort(v)
print(sort2)
```

```
## [1] "Blue"   "Red"    "violet" "yellow"
```

```r
# Sorting character vectors in reverse order.
revsort <- sort(v, decreasing = TRUE)
print(revsort)
```

```
## [1] "yellow" "violet" "Red"    "Blue"
```

Lists

Lists are the R objects which contain elements of different types like numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using **list()** function.

1. Creating a List

Following is an example to create a list containing strings, numbers, vectors and a logical values.

```r
# Create a list containing strings, numbers, vectors and a logical
# values.
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
print(list_data)
```

```
## [[1]]
## [1] "Red"
##
## [[2]]
## [1] "Green"
##
## [[3]]
## [1] 21 32 11
##
```

```
## [[4]]
## [1] TRUE
##
## [[5]]
## [1] 51.23
##
## [[6]]
## [1] 119.1
```

2. Naming List Elements

The list elements can be given names and they can be accessed using these names.

```r
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Show the list.
print(list_data)
```

```
## $`1st Quarter`
## [1] "Jan" "Feb" "Mar"
##
## $A_Matrix
##      [,1] [,2] [,3]
## [1,]    3    5   -2
## [2,]    9    1    8
##
## $`A Inner list`
## $`A Inner list`[[1]]
## [1] "green"
##
## $`A Inner list`[[2]]
## [1] 12.3
```

3. Accessing List Elements

Elements of the list can be accessed by the index of the element in the list. In case of named lists it can also be accessed using the names.

We continue to use the list in the above example

```r
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Access the first element of the list.
print(list_data[1])
```

```
## $`1st Quarter`
## [1] "Jan" "Feb" "Mar"
```

```r
# Access the thrid element. As it is also a list, all its elements will be printed.
print(list_data[3])
```

```
## $`A Inner list`
## $`A Inner list`[[1]]
## [1] "green"
##
## $`A Inner list`[[2]]
## [1] 12.3
```

```r
# Access the list element using the name of the element.
print(list_data$A_Matrix)
```

```
##      [,1] [,2] [,3]
## [1,]    3    5   -2
## [2,]    9    1    8
```

4. Manipulating List Elements

We can add, delete and update list elements as shown below. We can add and delete elements only at the end of a list. But we can update any element.

```r
# Create a list containing a vector, a matrix and a list.
list_data <- list(c("Jan","Feb","Mar"), matrix(c(3,9,5,1,-2,8), nrow = 2),
   list("green",12.3))

# Give names to the elements in the list.
names(list_data) <- c("1st Quarter", "A_Matrix", "A Inner list")

# Add element at the end of the list.
list_data[4] <- "New element"
print(list_data[4])
```

```
## [[1]]
## [1] "New element"
```

```r
# Remove the last element.
list_data[4] <- NULL

# Print the 4th Element.
print(list_data[4])
```

```
## $<NA>
## NULL
```

```r
# Update the 3rd Element.
list_data[3] <- "updated element"
print(list_data[3])
```

```
## $`A Inner list`
## [1] "updated element"
```

5. Merging Lists

You can merge many lists into one list by placing all the lists inside one list() function.

```r
# Create two lists.
list1 <- list(1,2,3)
list2 <- list("Sun","Mon","Tue")
```

```r
# Merge the two lists.
merged.list <- c(list1,list2)

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

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] "Sun"
##
## [[5]]
## [1] "Mon"
##
## [[6]]
## [1] "Tue"
```

6. Converting List to Vector

A list can be converted to a vector so that the elements of the vector can be used for further manipulation. All the arithmetic operations on vectors can be applied after the list is converted into vectors. To do this conversion, we use the unlist() function. It takes the list as input and produces a vector.

```r
# Create lists.
list1 <- list(1:5)
print(list1)
```

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

```r
list2 <-list(10:14)
print(list2)
```

```
## [[1]]
## [1] 10 11 12 13 14
```

```r
# Convert the lists to vectors.
v1 <- unlist(list1)
v2 <- unlist(list2)

print(v1)
```

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

```r
print(v2)
```

```
## [1] 10 11 12 13 14
```

```r
# Now add the vectors
result <- v1+v2
print(result)
```

```
## [1] 11 13 15 17 19
```

Matrices

Matrices are the R objects in which the elements are arranged in a two-dimensional rectangular layout. They contain elements of the same atomic types. Though we can create a matrix containing only characters or only logical values, they are not of much use. We use matrices containing numeric elements to be used in mathematical calculations.

A Matrix is created using the **matrix()** function.

Syntax

The basic syntax for creating a matrix in R is

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

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.
- dimname is the names assigned to the rows and columns.

Example

Create a matrix taking a vector of numbers as input.

```
# Elements are arranged sequentially by row.
M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
print(M)
```

```
##      [,1] [,2] [,3]
## [1,]    3    4    5
## [2,]    6    7    8
## [3,]    9   10   11
## [4,]   12   13   14
```

```
# Elements are arranged sequentially by column.
N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
print(N)
```

```
##      [,1] [,2] [,3]
## [1,]    3    7   11
## [2,]    4    8   12
## [3,]    5    9   13
## [4,]    6   10   14
```

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
print(P)
```

```
##      col1 col2 col3
## row1    3    4    5
## row2    6    7    8
## row3    9   10   11
## row4   12   13   14
```

Accessing Elements of a Matrix

Elements of a matrix can be accessed by using the column and row index of the element. We consider the matrix P above to find the specific elements below.

```
# Define the column and row names.
rownames = c("row1", "row2", "row3", "row4")
colnames = c("col1", "col2", "col3")

# Create the matrix.
P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))

# Access the element at 3rd column and 1st row.
print(P[1,3])
```

```
## [1] 5
```

```
# Access the element at 2nd column and 4th row.
print(P[4,2])
```

```
## [1] 13
```

```
# Access only the  2nd row.
print(P[2,])
```

```
## col1 col2 col3
##    6    7    8
```

```
# Access only the 3rd column.
print(P[,3])
```

```
## row1 row2 row3 row4
##    5    8   11   14
```

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.

1. Matrix Addition Subtraction

```
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    9    4    6
```

```
matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)
```

```
##      [,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
```

```r
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]    8   -1    5
## [2,]   11   13   10
```

```r
# Subtract the matrices
result <- matrix1 - matrix2
cat("Result of subtraction","\n")
```

```
## Result of subtraction
```

```r
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]   -2   -1   -1
## [2,]    7   -5    2
```

2. Matrix Multiplication Division

```r
# Create two 2x3 matrices.
matrix1 <- matrix(c(3, 9, -1, 4, 2, 6), nrow = 2)
print(matrix1)
```

```
##      [,1] [,2] [,3]
## [1,]    3   -1    2
## [2,]    9    4    6
```

```r
matrix2 <- matrix(c(5, 2, 0, 9, 3, 4), nrow = 2)
print(matrix2)
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    3
## [2,]    2    9    4
```

```r
# Multiply the matrices.
result <- matrix1 * matrix2
cat("Result of multiplication","\n")
```

```
## Result of multiplication
```

```r
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]   15    0    6
## [2,]   18   36   24
```

```r
# Divide the matrices
result <- matrix1 / matrix2
cat("Result of division","\n")
```

```
## Result of division
```

```r
print(result)
```

```
##      [,1]      [,2]      [,3]
## [1,]  0.6      -Inf 0.6666667
## [2,]  4.5 0.4444444 1.5000000
```

Arrays

Arrays are the R data objects which can store data in more than two dimensions. For example If we create an array of dimension (2, 3, 4) then it creates 4 rectangular matrices each with 2 rows and 3 columns. Arrays can store only data type.

An array is created using the array() function. It takes vectors as input and uses the values in the dim parameter to create an array.

Example

The following example creates an array of two 3x3 matrices each with 3 rows and 3 columns.

```r
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
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)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

1. Naming Columns and Rows

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

```r
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
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)
```

```
## , , Matrix1
##
##      COL1 COL2 COL3
## ROW1    5   10   13
## ROW2    9   11   14
## ROW3    3   12   15
##
```

```
## , , Matrix2
##
##      COL1 COL2 COL3
## ROW1    5   10   13
## ROW2    9   11   14
## ROW3    3   12   15
```

2. Accessing Array Elements

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
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])
```

```
## 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])
```

```
## [1] 13
```

```
# Print the 2nd Matrix.
print(result[,,2])
```

```
##      COL1 COL2 COL3
## ROW1    5   10   13
## ROW2    9   11   14
## ROW3    3   12   15
```

3. Manipulating Array Elements

As array is made up matrices in multiple dimensions, the operations on elements of array are carried out by accessing elements of the matrices.

```
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

# Take these vectors as input to the array.
array1 <- array(c(vector1,vector2),dim = c(3,3,2))

# Create two vectors of different lengths.
vector3 <- c(9,1,0)
vector4 <- c(6,0,11,3,14,1,2,6,9)
array2 <- array(c(vector1,vector2),dim = c(3,3,2))

# create matrices from these arrays.
matrix1 <- array1[,,2]
```

```r
matrix2 <- array2[,,2]

# Add the matrices.
result <- matrix1+matrix2
print(result)
```

```
##      [,1] [,2] [,3]
## [1,]   10   20   26
## [2,]   18   22   28
## [3,]    6   24   30
```

4. Calculations Across Array Elements

We can do calculations across the elements in an array using the apply() function.

Syntax

apply(x, margin, fun)

Following is the description of the parameters used

- x is an array.
- margin is the name of the data set used.
- fun is the function to be applied across the elements of the array.

Example

We use the **apply()** function below to calculate the sum of the elements in the rows of an array across all the matrices

```r
# Create two vectors of different lengths.
vector1 <- c(5,9,3)
vector2 <- c(10,11,12,13,14,15)

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

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

```r
# Use apply to calculate the sum of the rows across all the matrices.
result <- apply(new.array, c(1), sum)
print(result)
```

```
## [1] 56 68 60
```

Data Frame

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

1. Create Data Frame

```r
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)
# Print the data frame.
print(student.data)
```

```
##   std_id std_name pocketmoney start_date
## 1      1      Ali      623.30 2012-01-01
## 2      2      Abu      515.20 2013-09-23
## 3      3    Ahmad      611.00 2014-11-15
## 4      4     Siti      729.00 2014-05-11
## 5      5     Emma      843.25 2015-03-27
```

2. Get the Structure of the Data Frame

The structure of the data frame can be seen by using str() function.

```r
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)
# Get the structure of the data frame.
str(student.data)
```

```
## 'data.frame':    5 obs. of  4 variables:
##  $ std_id     : int  1 2 3 4 5
##  $ std_name   : chr  "Ali" "Abu" "Ahmad" "Siti" ...
##  $ pocketmoney: num  623 515 611 729 843
##  $ start_date : Date, format: "2012-01-01" "2013-09-23" ...
```

3. Summary of Data in Data Frame

The statistical summary and nature of the data can be obtained by applying summary() function.

```r
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)
# Print the summary.
print(summary(student.data))
```

```
##      std_id    std_name          pocketmoney      start_date
##  Min.   :1   Length:5           Min.   :515.2   Min.   :2012-01-01
##  1st Qu.:2   Class :character   1st Qu.:611.0   1st Qu.:2013-09-23
##  Median :3   Mode  :character   Median :623.3   Median :2014-05-11
##  Mean   :3                      Mean   :664.4   Mean   :2014-01-14
##  3rd Qu.:4                      3rd Qu.:729.0   3rd Qu.:2014-11-15
##  Max.   :5                      Max.   :843.2   Max.   :2015-03-27
```

4. Extract Data from Data Frame

Extract specific column from a data frame using column name.

```r
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)
# Extract Specific columns.
result <- data.frame(student.data$std_name,student.data$pocketmoney)
print(result)
```

```
##   student.data.std_name student.data.pocketmoney
## 1                   Ali                   623.30
## 2                   Abu                   515.20
## 3                 Ahmad                   611.00
## 4                  Siti                   729.00
## 5                  Emma                   843.25
```

Extract the first two rows and then all columns

```r
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)
# Extract first two rows.
result <- student.data[1:2,]
print(result)
```

```
##   std_id std_name pocketmoney start_date
## 1      1      Ali       623.3 2012-01-01
## 2      2      Abu       515.2 2013-09-23
```

Extract 3rd and 5th row with 2nd and 4th column

```
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)

# Extract 3rd and 5th row with 2nd and 4th column.
result <- student.data[c(3,5),c(2,4)]
print(result)
```

```
##   std_name start_date
## 3    Ahmad 2014-11-15
## 5     Emma 2015-03-27
```

5. Expand Data Frame

A data frame can be expanded by adding columns and rows.

Add Column

Just add the column vector using a new column name.

```
# Create the data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),

   stringsAsFactors = FALSE
)
```

```r
# Add the "program" column.
student.data$program <- c("CS","BM","Stat","Banking","AM")
v <- student.data
print(v)
```

```
##   std_id std_name pocketmoney start_date program
## 1      1      Ali      623.30 2012-01-01      CS
## 2      2      Abu      515.20 2013-09-23      BM
## 3      3    Ahmad      611.00 2014-11-15    Stat
## 4      4     Siti      729.00 2014-05-11 Banking
## 5      5     Emma      843.25 2015-03-27      AM
```

Add Row

To **add more rows permanently** to an existing data frame, we need to bring in the new rows in the **same structure** as the existing data frame and use the **rbind()** function.

In the example below we create a data frame with new rows and merge it with the existing data frame to create the final data frame.

```r
# Create the first data frame.
student.data <- data.frame(
   std_id = c (1:5),
   std_name = c("Ali","Abu","Ahmad","Siti","Emma"),
   pocketmoney = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
      "2015-03-27")),
   program = c("CS","BM","Stat","Banking","AM"),
   stringsAsFactors = FALSE
)

# Create the second data frame
student.newdata <-  data.frame(
   std_id = c (6:8),
   std_name = c("Hairi","Alimah","Maria"),
   pocketmoney = c(578.0,722.5,632.8),
   start_date = as.Date(c("2013-05-21","2013-07-30","2014-06-17")),
   program = c("CS","Stat","AM"),
   stringsAsFactors = FALSE
)

# Bind the two data frames.
std.finaldata <- rbind(student.data,student.newdata)
print(std.finaldata)
```

```
##   std_id std_name pocketmoney start_date program
## 1      1      Ali      623.30 2012-01-01      CS
## 2      2      Abu      515.20 2013-09-23      BM
## 3      3    Ahmad      611.00 2014-11-15    Stat
## 4      4     Siti      729.00 2014-05-11 Banking
## 5      5     Emma      843.25 2015-03-27      AM
## 6      6    Hairi      578.00 2013-05-21      CS
## 7      7   Alimah      722.50 2013-07-30    Stat
## 8      8    Maria      632.80 2014-06-17      AM
```

Factors

Factors are the data objects which are used to **categorize the data** and store it as **levels**. They can store both strings and integers. They are useful in the columns which have a limited number of unique values. Like "Male,"Female" and True, False etc. They are useful in data analysis for statistical modeling.

Factors are created using the **factor ()** function by taking a vector as input.

Example

```
# Create a vector as input.
data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)
```

```
##  [1] "East"  "West"  "East"  "North" "North" "East"  "West"  "West"  "West"
## [10] "East"  "North"
```

```
print(is.factor(data))
```

```
## [1] FALSE
```

```
# Apply the factor function.
factor_data <- factor(data)

print(factor_data)
```

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

```
print(is.factor(factor_data))
```

```
## [1] TRUE
```

1. Factors in Data Frame

On creating any data frame with a column of text data, R treats the text column as categorical data and creates factors on it.

```
# Create the vectors for data frame.
height <- c(132,151,162,139,166,147,122)
weight <- c(48,49,66,53,67,52,40)
gender <- c("male","male","female","female","male","female","male")

# Create the data frame.
input_data <- data.frame(height,weight,gender)
print(input_data)
```

```
##   height weight gender
## 1    132     48   male
## 2    151     49   male
## 3    162     66 female
## 4    139     53 female
## 5    166     67   male
## 6    147     52 female
## 7    122     40   male
```

```
# Test if the gender column is a factor.
print(is.factor(input_data$gender))
```

```
## [1] FALSE
```

```r
# Print the gender column so see the levels.
print(input_data$gender)
```

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

2. Changing the Order of Levels

The order of the levels in a factor can be changed by applying the factor function again with new order of the levels.

```r
data <- c("East","West","East","North","North","East","West",
   "West","West","East","North")
# Create the factors
factor_data <- factor(data)
print(factor_data)
```

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

```r
# Apply the factor function with required order of the level.
new_order_data <- factor(factor_data,levels = c("East","West","North"))
print(new_order_data)
```

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

3. Generating Factor Levels

We can generate factor levels by using the gl() function. It takes two integers as input which indicates how many levels and how many times each level.

Syntax

gl(n, k, labels)

Following is the description of the parameters used

- **n** is a integer giving the number of levels.
- **k** is a integer giving the number of replications.
- **labels** is a vector of labels for the resulting factor levels.

Example

```r
v <- gl(3, 4, labels = c("Tampa", "Seattle","Boston"))
print(v)
```

```
##  [1] Tampa   Tampa   Tampa   Tampa   Seattle Seattle Seattle Seattle Boston
## [10] Boston  Boston  Boston
## Levels: Tampa Seattle Boston
```