DATA FRAME and R Functions

A **data frame** is used for storing data tables. It is a list of vectors of equal length. For example, the following variable df is a data frame containing three vectors n, s, b.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)      # df is a data frame
```

How to create a Data Frame in R?

We can create a data frame using the data.frame() function.

```
>x <- data.frame(“n" = c(2,3,5), “s" = c("aa", "bb", "cc") , “b" = c(TRUE, FALSE, TRUE))
>str(x)
```

Output:

```
>'data.frame':   3 obs. of  3 variables:
> $ n: num  2 3 5
> $ s: Factor w/ 3 levels "aa","bb","cc": 1 2 3
> $ b: logi  TRUE FALSE TRUE
```

Notice above that the third column, **s** 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.

```
>x <- data.frame("n" = c(2,3,5), "s" = c("aa", "bb", "cc") , "b" = c(TRUE, FALSE, TRUE),stringsAsFactors=FALSE)
```

```
> str(x)
'data.frame':   3 obs. of  3 variables:
 $ n: num  2 3 5
 $ s: chr  "aa" "bb" "cc"
 $ b: logi  TRUE FALSE TRUE
```

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.

```
# Create the data frame.

emp.data <- data.frame( emp_id = c (1:5),
                     emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
                     salary = 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(emp.data)

  emp_id  emp_name   salary   start_date
1   1       Rick      623.30   2012-01-01
2   2       Dan       515.20   2013-09-23
3   3       Michelle  611.00   2014-11-15
4   4       Ryan      729.00   2014-05-11
5   5       Gary      843.25   2015-03-27
```

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

```
# Print the summary.

print(summary(emp.data))

 emp_id      emp_name            salary          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
```

# Extract Specific columns.

```
result <- data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

```
      emp.data.emp_name    emp.data.salary
1           Rick               623.30
2            Dan               515.20
3        Michelle              611.00
4           Ryan               729.00
5    Gary                      843.25
```

# Extract first two rows.

```
 result <- emp.data[1:2,]
print(result)
```

```
   emp_id  emp_name  salary   start_date
1     1        Rick        623.3     2012-01-01
2     2        Dan         515.2     2013-09-23
```

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.

```
# Add the "dept" coulmn.

emp.data$dept <- c("IT","Operations","IT","HR","Finance")

v <- emp.data
print(v)
```

Output:

```
  emp_id emp_name salary   start_date      dept
1    1      Rick       623.30  2012-01-01       IT
2    2      Dan        515.20  2013-09-23  Operations
3    3      Michelle   611.00  2014-11-15       IT
4    4      Ryan       729.00  2014-05-11       HR
5    5      Gary       843.25  2015-03-27    Finance
```

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

```
# Create the second data frame
emp.newdata <- data.frame( emp_id = c (6:8), emp_name = c("Rasmi","Pranab","Tusar"),
                           salary = c(578.0,722.5,632.8), start_date = as.Date(c("2013-
                                    05-21","2013-07-30","2014-06-17")),
                           dept = c("IT","Operations","Fianance"),
                           stringsAsFactors = FALSE )

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

```
  emp_id emp_name      salary   start_date    dept
1   1     Rick         623.30   2012-01-01     IT
2   2     Dan          515.20   2013-09-23  Operations
3   3     Michelle     611.00   2014-11-15     IT
4   4     Ryan         729.00   2014-05-11     HR
5   5     Gary         843.25   2015-03-27   Finance
6   6      Rasmi       578.00   2013-05-21     IT
7   7      Pranab      722.50   2013-07-30   Operations
8   8     Tusar        632.80   2014-06-17   Fianance
```

Subsetting Data Frames

Data frames possess the characteristics of both lists and matrices:
•if you subset with a single vector, they behave like lists and will return the selected columns with all rows.
• if you subset with two vectors, they behave like matrices and can be subset by row and column.

```
# subsetting by row numbers
> emp.finaldata[2:4,]
  emp_id emp_name salary  start_date     dept
2     2        Dan         515.2   2013-09-23    Operations
3     3        Michelle  611.0   2014-11-15       IT
4    4         Ryan        729.0   2014-05-11       HR
```

# subsetting by col numbers

```
> emp.finaldata[, 2:3]
     emp_name        salary
1     Rick           623.30
2      Dan           515.20
3    Michelle        611.00
4     Ryan           729.00
5     Gary           843.25
6    Rasmi           578.00
7    Pranab          722.50
8    Tusar           632.80
```

# subset for both rows and columns

```
> emp.finaldata[1:3, 2:3]
     emp_name        salary
1     Rick           623.3
2      Dan           515.2
3     Michelle       611.0
```

Using Subset() function

In the previous example, we selected rows and column without condition. It is possible to **subset** based on whether or not a certain condition was true.

We use the subset() function.

subset(x, condition)

arguments: -
x: data frame used to perform the subset
condition: define the conditional statement

```
# salary > 700
> subset(emp.finaldata, salary>700)
   emp_id emp_name          salary     start_date          dept
4     4         Ryan               729.00     2014-05-11          HR
5     5         Gary               843.25     2015-03-27           Finance
7     7         Pranab             722.50     2013-07-30          Operations
```

R-functions

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.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

An R function is created by using the keyword **function**.

syntax of an R function−

```
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.

R has many **in-built** functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as **user defined** functions.

Built-in Function

Simple examples of in-built functions are **seq()**, **mean()**, **max()**, **sum(x)** and **paste(...)** etc.

```
# Create a sequence of numbers from 32 to 44.
print(seq(32,44))

# Find mean of numbers from 25 to 82.
print(mean(25:82))

# Find sum of numbers frm 41 to 68.
print(sum(41:68))
```

Output:

```
[1] 32 33 34 35 36 37 38 39 40 41 42 43 44
[1] 53.5
[1] 1526
```

User-defined Function

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
        for(i in 1:a) {
                b <- i^2
                print(b)
                }
            }

# Call the function new.function supplying 6 as an argument.

 new.function(6)
```

Output:

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

Calling a Function without an Argument

```
# Create a function without an argument.
 new.function <- function() {
                for(i in 1:5) {
                                print(i^2)
                                 }
                         }

# Call the function without supplying an argument.
new.function()
```

Output:

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
```

Calling a Function with Argument Values (by position and by name)

```
# Create a function with arguments.
new.function <- function(a,b,c) {
            result <- a * b + c
            print(result)
            }

# Call the function by position of arguments.
new.function(5,3,11)

# Call the function by names of the arguments.
new.function(a = 11, b = 5, c = 3)
```

Output:

```
[1] 26
[1] 58
```

Calling a Function with Default Argument

```
# Create a function with arguments.
new.function <- function(a = 3, b = 6) {
                        result <- a * b print(result)
                        }

# Call the function without giving any argument.
new.function()

# Call the function with giving new values of the argument.
new.function(9,5)
```

Output:

```
[1] 18
[1] 45
```

Lazy Evaluation of Function

Arguments to functions are evaluated lazily, which means so they are evaluated only when needed by the function body.

```
# Create a function with arguments.
new.function <- function(a, b) {
            print(a^2)
            print(a)
            print(b) }

# Evaluate the function without supplying one of the arguments.
new.function(6)
```

Output:

```
[1] 36
 [1] 6
Error in print(b) : argument "b" is missing, with no default
```