Unit -4

Data Frames:

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.

Create Data Frame

```
# 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)
```

When we execute the above code, it produces the following result −

```
  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
```

Get the Structure of the Data Frame

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

```
# 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
)
# Get the structure of the data frame.
str(emp.data)
```

When we execute the above code, it produces the following result −

```
'data.frame':   5 obs. of  4 variables:
 $ emp_id    : int  1 2 3 4 5
 $ emp_name  : chr  "Rick" "Dan" "Michelle" "Ryan" ...
 $ salary    : num  623 515 611 729 843
 $ start_date: Date, format: "2012-01-01" "2013-09-23" "2014-
11-15" "2014-05-11" ...
```

Summary of Data in Data Frame

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

```
# 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 summary.
print(summary(emp.data))
```

When we execute the above code, it produces the following result −

```
     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 Data from Data Frame

Extract specific column from a data frame using column name.

```
# 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
)
# Extract Specific columns.
result <- data.frame(emp.data$emp_name,emp.data$salary)
print(result)
```

When we execute the above code, it produces the following 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 the first two rows and then all columns

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

When we execute the above code, it produces the following result −

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

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

```
# 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
)

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

When we execute the above code, it produces the following result −

```
  emp_name start_date
3 Michelle 2014-11-15
5     Gary 2015-03-27
```

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.
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
)

# Add the "dept" coulmn.
emp.data$dept <- c("IT","Operations","IT","HR","Finance")
v <- emp.data
print(v)
```

When we execute the above code, it produces the following result −

```
  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 example 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 first 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")),
   dept = c("IT","Operations","IT","HR","Finance"),
   stringsAsFactors = FALSE
)

# 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)
```

When we execute the above code, it produces the following result −

```
  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
```

Data Reshaping:

Data Reshaping in R is about changing the way data is organized into rows and columns. Most of the time data processing in R is done by taking the input data as a data frame. It is easy to extract data from the rows and columns of a data frame but there are situations when we need the data frame in a format that is different from format in which we received it. R has many functions to split, merge and change the rows to columns and vice-versa in a data frame.

Joining Columns and Rows in a Data Frame

We can join multiple vectors to create a data frame using the **cbind()**function. Also we can merge two data frames using **rbind()** function.

```r
# Create vector objects.
city <- c("Tampa","Seattle","Hartford","Denver")
state <- c("FL","WA","CT","CO")
zipcode <- c(33602,98104,06161,80294)

# Combine above three vectors into one data frame.
addresses <- cbind(city,state,zipcode)

# Print a header.
cat("# # # # The First data frame\n")

# Print the data frame.
print(addresses)

# Create another data frame with similar columns
```

```
new.address <- data.frame(
   city = c("Lowry","Charlotte"),
   state = c("CO","FL"),
   zipcode = c("80230","33949"),
   stringsAsFactors = FALSE
)

# Print a header.
cat("# # # The Second data frame\n")

# Print the data frame.
print(new.address)

# Combine rows form both the data frames.
all.addresses <- rbind(addresses,new.address)

# Print a header.
cat("# # # The combined data frame\n")

# Print the result.
print(all.addresses)
```

When we execute the above code, it produces the following result −

```
# # # # The First data frame
     city       state zipcode
[1,] "Tampa"    "FL"  "33602"
[2,] "Seattle"  "WA"  "98104"
[3,] "Hartford" "CT"   "6161"
[4,] "Denver"   "CO"  "80294"

# # # The Second data frame
       city       state   zipcode
1      Lowry      CO      80230
2      Charlotte  FL      33949

# # # The combined data frame
       city      state zipcode
1      Tampa     FL    33602
2      Seattle   WA    98104
3      Hartford  CT     6161
4      Denver    CO    80294
5      Lowry     CO    80230
6     Charlotte  FL    33949
```

Merging Data Frames

We can merge two data frames by using the **merge()** function. The data frames must have same column names on which the merging happens.

In the example below, we consider the data sets about Diabetes in Pima Indian Women available in the library names "MASS". we merge the two data sets based on the values of blood pressure("bp") and body mass index("bmi"). On choosing these two columns for merging, the records where values of these two variables match in both data sets are combined together to form a single data frame.

```
library(MASS)
merged.Pima <- merge(x = Pima.te, y = Pima.tr,
   by.x = c("bp", "bmi"),
   by.y = c("bp", "bmi")
)
print(merged.Pima)
nrow(merged.Pima)
```

When we execute the above code, it produces the following result −

```
   bp  bmi npreg.x glu.x skin.x ped.x age.x type.x npreg.y glu.y
skin.y ped.y
1   60 33.8       1   117     23 0.466    27     No       2   125
20 0.088
2   64 29.7       2    75     24 0.370    33     No       2   100
23 0.368
3   64 31.2       5   189     33 0.583    29    Yes       3   158
13 0.295
4   64 33.2       4   117     27 0.230    24     No       1    96
27 0.289
5   66 38.1       3   115     39 0.150    28     No       1   114
36 0.289
6   68 38.5       2   100     25 0.324    26     No       7   129
49 0.439
7   70 27.4       1   116     28 0.204    21     No       0   124
20 0.254
8   70 33.1       4    91     32 0.446    22     No       9   123
44 0.374
9   70 35.4       9   124     33 0.282    34     No       6   134
23 0.542
10  72 25.6       1   157     21 0.123    24     No       4    99
17 0.294
11  72 37.7       5    95     33 0.370    27     No       6   103
32 0.324
12  74 25.9       9   134     33 0.460    81     No       8   126
38 0.162
13  74 25.9       1    95     21 0.673    36     No       8   126
38 0.162
14  78 27.6       5    88     30 0.258    37     No       6   125
31 0.565
15  78 27.6      10   122     31 0.512    45     No       6   125
31 0.565
```

```
16 78 39.4       2   112     50 0.175    24     No        4   112
40 0.236
17 88 34.5       1   117     24 0.403    40    Yes        4   127
11 0.598
   age.y type.y
1     31     No
2     21     No
3     24     No
4     21     No
5     21     No
6     43    Yes
7     36    Yes
8     40     No
9     29    Yes
10    28     No
11    55     No
12    39     No
13    39     No
14    49    Yes
15    49    Yes
16    38     No
17    28     No
[1] 17
```

Melting and Casting

One of the most interesting aspects of R programming is about changing the shape of the data in multiple steps to get a desired shape. The functions used to do this are called **melt()** and **cast()**.

We consider the dataset called ships present in the library called "MASS".

```
library(MASS)
print(ships)
```

When we execute the above code, it produces the following result −

```
     type year   period   service   incidents
1     A   60     60         127         0
2     A   60     75          63         0
3     A   65     60        1095         3
4     A   65     75        1095         4
5     A   70     60        1512         6
.............
.............
8     A   75     75        2244         11
9     B   60     60       44882         39
10    B   60     75       17176         29
11    B   65     60       28609         58
.............
```

```
.............
17     C    60      60       1179           1
18     C    60      75        552           1
19     C    65      60        781           0
.............
.............
```

Melt the Data

Now we melt the data to organize it, converting all columns other than type and year into multiple rows.

```
molten.ships <- melt(ships, id = c("type","year"))
print(molten.ships)
```

When we execute the above code, it produces the following result −

```
      type year  variable  value
1       A   60    period     60
2       A   60    period     75
3       A   65    period     60
4       A   65    period     75
.............
.............
9       B   60    period     60
10      B   60    period     75
11      B   65    period     60
12      B   65    period     75
13      B   70    period     60
............
............
41      A   60    service   127
42      A   60    service    63
43      A   65    service  1095
............
............
70      D   70    service  1208
71      D   75    service     0
72      D   75    service  2051
73      E   60    service    45
74      E   60    service     0
75      E   65    service   789
............
............
101     C   70    incidents    6
102     C   70    incidents    2
103     C   75    incidents    0
104     C   75    incidents    1
105     D   60    incidents    0
106     D   60    incidents    0
............
............
```

Cast the Molten Data

We can cast the molten data into a new form where the aggregate of each type of ship for each year is created. It is done using the **cast()** function.

```
recasted.ship <- cast(molten.ships, type+year~variable,sum)
print(recasted.ship)
```

When we execute the above code, it produces the following result −

```
     type year  period  service  incidents
1      A   60     135      190       0
2      A   65     135     2190       7
3      A   70     135     4865      24
4      A   75     135     2244      11
5      B   60     135    62058      68
6      B   65     135    48979     111
7      B   70     135    20163      56
8      B   75     135     7117      18
9      C   60     135     1731       2
10     C   65     135     1457       1
11     C   70     135     2731       8
12     C   75     135      274       1
13     D   60     135      356       0
14     D   65     135      480       0
15     D   70     135     1557      13
16     D   75     135     2051       4
17     E   60     135       45       0
18     E   65     135     1226      14
19     E   70     135     3318      17
20     E   75     135      542       1
```

In R, we can read data from files stored outside the R environment. We can also write data into files which will be stored and accessed by the operating system. R can read and write into various file formats like csv, excel, xml etc.

In this chapter we will learn to read data from a csv file and then write data into a csv file. The file should be present in current working directory so that R can read it. Of course we can also set our own directory and read files from there.

Getting and Setting the Working Directory

You can check which directory the R workspace is pointing to using the **getwd()** function. You can also set a new working directory using **setwd()**function.

```
# Get and print current working directory.
print(getwd())

# Set current working directory.
setwd("/web/com")

# Get and print current working directory.
print(getwd())
```

When we execute the above code, it produces the following result −

```
[1] "/web/com/1441086124_2016"
[1] "/web/com"
```

This result depends on your OS and your current directory where you are working.

Input as CSV File

The csv file is a text file in which the values in the columns are separated by a comma. Let's consider the following data present in the file named **input.csv**.

You can create this file using windows notepad by copying and pasting this data. Save the file as **input.csv** using the save As All files(*.*) option in notepad.

```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
5,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

Reading a CSV File

Following is a simple example of **read.csv()** function to read a CSV file available in your current working directory −

```
data <- read.csv("input.csv")
print(data)
```

When we execute the above code, it produces the following result −

```
      id,    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     NA     Gary      843.25     2015-03-27       Finance
6      6     Nina      578.00     2013-05-21       IT
7      7     Simon     632.80     2013-07-30       Operations
8      8     Guru      722.50     2014-06-17       Finance
```

Analyzing the CSV File

By default the **read.csv()** function gives the output as a data frame. This can be easily checked as follows. Also we can check the number of columns and rows.

```
data <- read.csv("input.csv")

print(is.data.frame(data))
print(ncol(data))
print(nrow(data))
```

When we execute the above code, it produces the following result −

```
[1] TRUE
[1] 5
[1] 8
```

Once we read data in a data frame, we can apply all the functions applicable to data frames as explained in subsequent section.

Get the maximum salary

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)
print(sal)
```

When we execute the above code, it produces the following result −

```
[1] 843.25
```

Get the details of the person with max salary

We can fetch rows meeting specific filter criteria similar to a SQL where clause.

```
# Create a data frame.
data <- read.csv("input.csv")

# Get the max salary from data frame.
sal <- max(data$salary)

# Get the person detail having max salary.
retval <- subset(data, salary == max(salary))
print(retval)
```

When we execute the above code, it produces the following result −

```
      id    name  salary  start_date    dept
5     NA    Gary  843.25  2015-03-27    Finance
```

Get all the people working in IT department

```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset( data, dept == "IT")
print(retval)
```

When we execute the above code, it produces the following result −

```
      id    name      salary   start_date   dept
1     1     Rick      623.3    2012-01-01   IT
3     3     Michelle  611.0    2014-11-15   IT
6     6     Nina      578.0    2013-05-21   IT
```

Get the persons in IT department whose salary is greater than 600

```
# Create a data frame.
data <- read.csv("input.csv")

info <- subset(data, salary > 600 & dept == "IT")
print(info)
```

When we execute the above code, it produces the following result −

```
      id    name      salary   start_date   dept
1     1     Rick      623.3    2012-01-01   IT
3     3     Michelle  611.0    2014-11-15   IT
```

Get the people who joined on or after 2014

```
# Create a data frame.
data <- read.csv("input.csv")

retval <- subset(data, as.Date(start_date) > as.Date("2014-01-
01"))
print(retval)
```

When we execute the above code, it produces the following result −

```
        id    name      salary    start_date     dept
3        3    Michelle 611.00    2014-11-15     IT
4        4    Ryan     729.00    2014-05-11     HR
5       NA    Gary     843.25    2015-03-27     Finance
8        8    Guru     722.50    2014-06-17     Finance
```

Writing into a CSV File

R can create csv file form existing data frame.
The **write.csv()** function is used to create the csv file. This file gets created in the working directory.

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-
01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv")
newdata <- read.csv("output.csv")
print(newdata)
```

When we execute the above code, it produces the following result −

```
  X      id    name      salary    start_date     dept
1 3      3     Michelle  611.00    2014-11-15     IT
2 4      4     Ryan      729.00    2014-05-11     HR
3 5     NA     Gary      843.25    2015-03-27     Finance
4 8      8     Guru      722.50    2014-06-17     Finance
```

Here the column X comes from the data set newper. This can be dropped using additional parameters while writing the file.

```
# Create a data frame.
data <- read.csv("input.csv")
retval <- subset(data, as.Date(start_date) > as.Date("2014-01-
01"))

# Write filtered data into a new file.
write.csv(retval,"output.csv", row.names = FALSE)
newdata <- read.csv("output.csv")
print(newdata)
```

When we execute the above code, it produces the following result −

```
      id    name      salary    start_date     dept
1      3    Michelle  611.00    2014-11-15     IT
2      4    Ryan      729.00    2014-05-11     HR
3     NA    Gary      843.25    2015-03-27     Finance
4      8    Guru      722.50    2014-06-17     Finance
```

Reading the Excel File

The input.xlsx is read by using the **read.xlsx()** function as shown below. The result is stored as a data frame in the R environment.

```
# Read the first worksheet in the file input.xlsx.
data <- read.xlsx("input.xlsx", sheetIndex = 1)
print(data)
```

When we execute the above code, it produces the following result −

```
      id,    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      NA     Gary       843.25     2015-03-27     Finance
6       6     Nina       578.00     2013-05-21     IT
7       7     Simon      632.80     2013-07-30     Operations
8       8     Guru       722.50     2014-06-17     Finance
```