Date:

Practical No 1

Aim: Practical of Data collection, Data curation and management for Unstructured data (NoSQL)

Importing Data in R

1. Objective – Importing Data in R

In this practical, data will be imported from external sources into R programming language.

2. Importing Data in R

a. Using the Combine Command

c() function use to concatenate or combine items in R as specified below:

>c(item.1, item.2, item.n)

(The c() function combines all the specified items in one object)

>sample.name = c(item.1, item.2, item.n)

(The concatenated values can assign to a named object, as shown in the command)

b. Entering Numerical Items as Data

Numerical data can simply enhance by typing the values separated by commas into the c() command. Let us create a data set. Below command use for the same:

>data1 = c(3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9)

It creates a new object to hold the data and then type the values in the parenthesis. The values are separated by using commas. The result is not automatically displayed. To see the dataset, type its name in the R console as follows:

>data1

This command will display entries of data

```
> data1
 [1] 3 5 7 5 3 2 6 8 5 6 9
```

Displays the contents of object data1

As R supports different types of data, all data types can import into it for computation. Existing data objects can incorporate with existing values to make new ones, simply by incorporating them as if they were values themselves.

e.g.

```
data1 = c(3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9)
data1

data2 = c(data1, 4, 5, 7, 3, 4)
data2
```

```
> data1 = c(3, 5, 7, 5, 3, 2, 6, 8, 5, 6, 9)
> data1
 [1] 3 5 7 5 3 2 6 8 5 6 9
>
> data2 = c(data1, 4, 5, 7, 3, 4)
> data2
 [1] 3 5 7 5 3 2 6 8 5 6 9 4 5 7 3 4
```

c. Entering Text Items as Data

Data that is not numerical can differentiate from numbers by using quotes. There is no difference between using single and double quotes; R converts them all to double. Either or both can use as long as the surrounding quotes for any single item match. As numerical data can import, text values can also import and manipulate in R.

```
day1 = c('Mon', 'Tue', 'Wed', 'Thu')
day1
day1 = c(day1, 'Fri')
day1
```

```
> day1 = c('Mon', 'Tue', 'Wed', 'Thu')
> day1
[1] "Mon" "Tue" "Wed" "Thu"
> day1 = c(day1, 'Fri')
> day1
[1] "Mon" "Tue" "Wed" "Thu" "Fri"
```

When text and numbers are combined, entire data object becomes a text variable and the numbers are also converted to text.
The c()command is a quick way of getting a series of values stored in a data object. This command is useful when the samples are small, but it can be tedious when a lot of typing involves.

d. Using the scan() Command

When using the c() command, you may find typing all the commas to separate the values a little tedious. Instead, you can use the scan()command to do the same job, but without the commas. In addition to using the scan()command to enter text into datasets, it can use with the clipboard and take data from files.

Unlike the c()command, the scan()command uses empty parentheses. The command then prompts you to enter the desired data. The entered data can store in a new variable.
e.g.

```
>file_name = scan()
```

This is the syntax for using scan command. One can also use the scan() command to enter text into datasets. Simply entering the items in quotes will generate an error message. The modified syntax for entering text as data is as follows:

```
>mn <- scan(what = 'character')
Jan Feb Mar
Apr May
>mn
```

[1] “Jan” “Feb” “Mar” “Apr” “May”

what: type of data, including logical, integer, numeric, complex, character, raw
In R, the user must specify that the items entered are characters, and not numbers. To do so, the (what = ‘character’) part must add.

e. Using the Clipboard to Make Data

Another way of importing data interactively into R is to use the Clipboard to copy and paste data. The scan() command can use with programs, such as a spreadsheet for entering data into R.
The steps to import data are:

1. If the spreadsheet data is in the form of numbers, simply type the command in R as usual before switching to the spreadsheet containing the data.
2. Highlight the necessary cells in the spreadsheet and copy them to the clipboard.
3. Then return to R and paste the data from the clipboard into R. As usual, R waits until a blank line is entered before ending the data entry so you can continue to copy and paste more data as required.
4. Enter a blank line to complete data entry.

e.g.

```
> fn = scan()
```

```
1: 23
2: 43
3: 6
4: 77
5: 8
6: 97
7: 45
8:
Read 7 items
```

If the data is text, add the what = 'character' instruction to the scan() command. If the file can open in a spreadsheet, proceed with the aforementioned four steps. Now, if the file opens in a text editor or word processor, see how the data items are separated before continuing. If the data is separated by simple spaces, simply copy and paste. If the data is separated by some other character, R needs to be told which character is used as the separator.

f. Using Scan() to retrieve data from CSV file

The scan() command can use to retrieve data from a CSV file, as follows:

```
fn = scan(sep = ',')
```

The separator must enclose in quotes. You need to press enter to finish the data entry.

g. Reading a File of Data from a Disk

The scan() command can use to retrieve data file in the memory of the system. Scan() can read data into a vector or list from the console or file. To read a file with the scan()command, simply add file = 'filename' to the command as shown below:

```
Object_Name = scan(file = 'File_Name.txt')
```

The filename must enclosed in quotes.

R looks for the data file in the default directory. To get the current working directory, getwd() command is used as below:

```
> getwd()
[1] "C:/Users/Administrator/Documents"
```

We can alter the working directory in R. In case you want to load files by just typing their names from any directory, the task becomes easier if we permanently set the working directory as different directory. We can alter the directory with the setwd() command:

```
> setwd("C:/Users/Administrator/Desktop")
> getwd()
[1] "C:/Users/Administrator/Desktop"
```

In the Windows and Mac operating systems, there is an alternative method that enables file selection. The instruction file.choose()can include as part of scan()command. This opens a browser-type window where users can navigate and select the file to read.

```
> Object_Name = scan(file.choose())
```

h. Reading Bigger Data Files

Let us now see how to read bigger data files in R:

The scan()command is helpful in reading simple vectors. It is possible to enter large amounts of data directly into R from complicated data files that contain multiple items. It is more likely that the data would be stored in a spreadsheet. R provides the means to read data that is stored in a range of text formats, all of which the spreadsheet is able to create.

- Command to read from CSV file: > read.csv() or read.csv2()
- Command to read from tables: > read.table()
- Command to read from Tab separated value files: > delim()

The difference between read.csv() and read.csv2() in R is in their usage. The former function use if the separator is a ‘,’ while the latter use if the separator is ‘;’ to separate the values in your data file.

i. Missing Values in Data Files

In the real world, samples are often of unequal size. So now we are going to see how R handles missing values in data files:
Let us consider two samples, mow and unmow.
The mow sample contains five values, whereas the unmow sample contains four values. When this data read into R from a spreadsheet or text file, the program recognizes multiple columns of data and sets them accordingly.
R converts data into a neat rectangular item and fills in any gaps with NA.
NOTE: The NA item is a special object in its own right as “Not Applicable” or “Not Available.”

For example:

```
> gf = read.csv (file.choose())
> gf
  Mow Unmow
1  43    65
2   6    35
3  77    21
4   8    20
5  97    NA
```

The dataset has been called grass and R has filled in the gap by using NA.
R always pads out the shorter samples by using NA to produce a rectangular object. This is called a data frame in R. Thus R data frame is an important kind of object because it uses so often in statistical data manipulation.

3. Conclusion

In this R practical, we discussed the process of importing data in R. Also, we saw different command for importing data in R and understand all with the help of examples. Still, if you have a query regarding Importing Data in R, ask in the comment tab.

R - CSV Files

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
```

Microsoft Excel is the most widely used spreadsheet program which stores data in the .xls or .xlsx format. R can read directly from these files using some excel specific packages. Few such packages are - XLConnect, xlsx, gdata etc. We will be using xlsx package. R can also write into excel file using this package.

Install xlsx Package

You can use the following command in the R console to install the "xlsx" package. It may ask to install some additional packages on which this package is dependent. Follow the same command with required package name to install the additional packages.

```
install.packages("xlsx")
```

Verify and Load the "xlsx" Package

Use the following command to verify and load the "xlsx" package.

```
# Verify the package is installed.
any(grepl("xlsx",installed.packages()))

# Load the library into R workspace.
library("xlsx")
```

When the script is run we get the following output.

```
[1] TRUE
Loading required package: rJava
Loading required package: methods
Loading required package: xlsxjars
```

Input as xlsx File

Open Microsoft excel. Copy and paste the following data in the work sheet named as sheet1.

id	name	salary	start_date	dept
1	Rick	623.3	1/1/2012	IT
2	Dan	515.2	9/23/2013	Operations
3	Michelle	611	11/15/2014	IT
4	Ryan	729	5/11/2014	HR
5	Gary	43.25	3/27/2015	Finance
6	Nina	578	5/21/2013	IT
7	Simon	632.8	7/30/2013	Operations
8	Guru	722.5	6/17/2014	Finance

Also copy and paste the following data to another worksheet and rename this worksheet to "city".

name	city
Rick	Seattle
Dan	Tampa
Michelle	Chicago
Ryan	Seattle
Gary	Houston
Nina	Boston
Simon	Mumbai
Guru	Dallas

Save the Excel file as "input.xlsx". You should save it in the current working directory of the R workspace.

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
```

R - Binary Files

A binary file is a file that contains information stored only in form of bits and bytes.(0's and 1's). They are not human readable as the bytes in it translate to characters and symbols which contain many other non-printable characters. Attempting to read a binary file using any text editor will show characters like Ø and ð.

The binary file has to be read by specific programs to be useable. For example, the binary file of a Microsoft Word program can be read to a human readable form only by the Word program. Which indicates that, besides the human readable text, there is a lot more information like formatting of characters and page numbers etc., which are also stored along with alphanumeric characters. And finally a binary file is a continuous sequence of bytes. The line break we see in a text file is a character joining first line to the next.

Sometimes, the data generated by other programs are required to be processed by R as a binary file. Also R is required to create binary files which can be shared with other programs.

R has two functions **WriteBin()** and **readBin()** to create and read binary files.

Syntax

```
writeBin(object, con)
readBin(con, what, n )
```

Following is the description of the parameters used −

- **con** is the connection object to read or write the binary file.
- **object** is the binary file which to be written.
- **what** is the mode like character, integer etc. representing the bytes to be read.
- **n** is the number of bytes to read from the binary file.

Example

We consider the R inbuilt data "mtcars". First we create a csv file from it and convert it to a binary file and store it as a OS file. Next we read this binary file created into R.

Writing the Binary File

We read the data frame "mtcars" as a csv file and then write it as a binary file to the OS.

```
# Read the "mtcars" data frame as a csv file and store only the columns
   "cyl", "am" and "gear".
write.table(mtcars, file = "mtcars.csv",row.names = FALSE, na = "",
   col.names = TRUE, sep = ",")

# Store 5 records from the csv file as a new data frame.
new.mtcars <- read.table("mtcars.csv",sep = ",",header = TRUE,nrows = 5)

# Create a connection object to write the binary file using mode "wb".
```

```
write.filename = file("/web/com/binmtcars.dat", "wb")

# Write the column names of the data frame to the connection object.
writeBin(colnames(new.mtcars), write.filename)

# Write the records in each of the column to the file.
writeBin(c(new.mtcars$cyl,new.mtcars$am,new.mtcars$gear), write.filename)

# Close the file for writing so that it can be read by other program.
close(write.filename)
```

Reading the Binary File

The binary file created above stores all the data as continuous bytes. So we will read it by choosing appropriate values of column names as well as the column values.

```
# Create a connection object to read the file in binary mode using "rb".
read.filename <- file("/web/com/binmtcars.dat", "rb")

# First read the column names. n = 3 as we have 3 columns.
column.names <- readBin(read.filename, character(),  n = 3)

# Next read the column values. n = 18 as we have 3 column names and 15 values.
read.filename <- file("/web/com/binmtcars.dat", "rb")
bindata <- readBin(read.filename, integer(),  n = 18)

# Print the data.
print(bindata)

# Read the values from 4th byte to 8th byte which represents "cyl".
cyldata = bindata[4:8]
print(cyldata)

# Read the values form 9th byte to 13th byte which represents "am".
amdata = bindata[9:13]
print(amdata)

# Read the values form 9th byte to 13th byte which represents "gear".
geardata = bindata[14:18]
print(geardata)

# Combine all the read values to a dat frame.
finaldata = cbind(cyldata, amdata, geardata)
colnames(finaldata) = column.names
print(finaldata)
```

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

```
 [1]    7108963 1728081249    7496037          6          6          4
 [7]          6          8          1          1          1          0
[13]          0          4          4          4          3          3

[1] 6 6 4 6 8

[1] 1 1 1 0 0

[1] 4 4 4 3 3
```

```
    cyl am gear
[1,]   6  1    4
[2,]   6  1    4
[3,]   4  1    4
[4,]   6  0    3
[5,]   8  0    3
```

R - XML Files

XML is a file format which shares both the file format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It stands for Extensible Markup Language (XML). Similar to HTML it contains markup tags. But unlike HTML where the markup tag describes structure of the page, in xml the markup tags describe the meaning of the data contained into he file.

You can read a xml file in R using the "XML" package. This package can be installed using following command.

```
install.packages("XML")
```

Input Data

Create a XMl file by copying the below data into a text editor like notepad. Save the file with a **.xml** extension and choosing the file type as **all files(*.*)**.

```
<RECORDS>
  <EMPLOYEE>
    <ID>1</ID>
    <NAME>Rick</NAME>
    <SALARY>623.3</SALARY>
    <STARTDATE>1/1/2012</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>2</ID>
    <NAME>Dan</NAME>
    <SALARY>515.2</SALARY>
    <STARTDATE>9/23/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>3</ID>
    <NAME>Michelle</NAME>
    <SALARY>611</SALARY>
    <STARTDATE>11/15/2014</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>4</ID>
    <NAME>Ryan</NAME>
    <SALARY>729</SALARY>
    <STARTDATE>5/11/2014</STARTDATE>
    <DEPT>HR</DEPT>
  </EMPLOYEE>
```

```
  <EMPLOYEE>
    <ID>5</ID>
    <NAME>Gary</NAME>
    <SALARY>843.25</SALARY>
    <STARTDATE>3/27/2015</STARTDATE>
    <DEPT>Finance</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>6</ID>
    <NAME>Nina</NAME>
    <SALARY>578</SALARY>
    <STARTDATE>5/21/2013</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>7</ID>
    <NAME>Simon</NAME>
    <SALARY>632.8</SALARY>
    <STARTDATE>7/30/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>

  <EMPLOYEE>
    <ID>8</ID>
    <NAME>Guru</NAME>
    <SALARY>722.5</SALARY>
    <STARTDATE>6/17/2014</STARTDATE>
    <DEPT>Finance</DEPT>
  </EMPLOYEE>

</RECORDS>
```

Reading XML File

The xml file is read by R using the function **xmlParse()**. It is stored as a list in R.

```
# Load the package required to read XML files.
library("XML")

# Also load the other required package.
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

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

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

```
1
Rick
623.3
1/1/2012
IT
```

```
2
Dan
515.2
9/23/2013
Operations

3
Michelle
611
11/15/2014
IT

4
Ryan
729
5/11/2014
HR

5
Gary
843.25
3/27/2015
Finance

6
Nina
578
5/21/2013
IT

7
Simon
632.8
7/30/2013
Operations

8
Guru
722.5
6/17/2014
Finance
```

Get Number of Nodes Present in XML File

```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.
rootnode <- xmlRoot(result)

# Find number of nodes in the root.
rootsize <- xmlSize(rootnode)
```

```
# Print the result.
print(rootsize)
```

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

```
output
[1] 8
```

Details of the First Node

Let's look at the first record of the parsed file. It will give us an idea of the various elements present in the top level node.

```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.
rootnode <- xmlRoot(result)

# Print the result.
print(rootnode[1])
```

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

```
$EMPLOYEE
  1
  Rick
  623.3
  1/1/2012
  IT


attr(,"class")
[1] "XMLInternalNodeList" "XMLNodeList"
```

Get Different Elements of a Node

```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Give the input file name to the function.
result <- xmlParse(file = "input.xml")

# Exract the root node form the xml file.
rootnode <- xmlRoot(result)

# Get the first element of the first node.
print(rootnode[[1]][[1]])

# Get the fifth element of the first node.
print(rootnode[[1]][[5]])

# Get the second element of the third node.
print(rootnode[[3]][[2]])
```

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

```
1
IT
Michelle
```

XML to Data Frame

To handle the data effectively in large files we read the data in the xml file as a data frame. Then process the data frame for data analysis.

```
# Load the packages required to read XML files.
library("XML")
library("methods")

# Convert the input xml file to a data frame.
xmldataframe <- xmlToDataFrame("input.xml")
print(xmldataframe)
```

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

```
    ID    NAME     SALARY    STARTDATE       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
```

As the data is now available as a dataframe we can use data frame related function to read and manipulate the file.

R - JSON Files

JSON file stores data as text in human-readable format. Json stands for JavaScript Object Notation. R can read JSON files using the rjson package.

Install rjson Package

In the R console, you can issue the following command to install the rjson package.

```
install.packages("rjson")
```

Input Data

Create a JSON file by copying the below data into a text editor like notepad. Save the file with a **.json** extension and choosing the file type as **all files(*.*)**.

```
{
  "ID":["1","2","3","4","5","6","7","8" ],
  "Name":["Rick","Dan","Michelle","Ryan","Gary","Nina","Simon","Guru" ],
  "Salary":["623.3","515.2","611","729","843.25","578","632.8","722.5" ],

  "StartDate":[ "1/1/2012","9/23/2013","11/15/2014","5/11/2014","3/27/2015","5/21/2013",
    "7/30/2013","6/17/2014"],
  "Dept":[ "IT","Operations","IT","HR","Finance","IT","Operations","Finance"]
}
```

Read the JSON File

The JSON file is read by R using the function from **JSON()**. It is stored as a list in R.

```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
```

```
result <- fromJSON(file = "input.json")

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

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

```
$ID
[1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"

$Name
[1] "Rick"    "Dan"     "Michelle" "Ryan"    "Gary"    "Nina"    "Simon"   "Guru"

$Salary
[1] "623.3"  "515.2"  "611"    "729"    "843.25" "578"    "632.8"  "722.5"

$StartDate
[1] "1/1/2012"   "9/23/2013"  "11/15/2014" "5/11/2014"  "3/27/2015"  "5/21/2013"
   "7/30/2013"  "6/17/2014"

$Dept
[1] "IT"         "Operations" "IT"         "HR"         "Finance"    "IT"
   "Operations" "Finance"
```

Convert JSON to a Data Frame

We can convert the extracted data above to a R data frame for further analysis using the **as.data.frame()** function.

```
# Load the package required to read JSON files.
library("rjson")

# Give the input file name to the function.
result <- fromJSON(file = "input.json")

# Convert JSON file to a data frame.
json_data_frame <- as.data.frame(result)

print(json_data_frame)
```

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
```