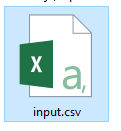
**#6. R - CSV Files**

**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

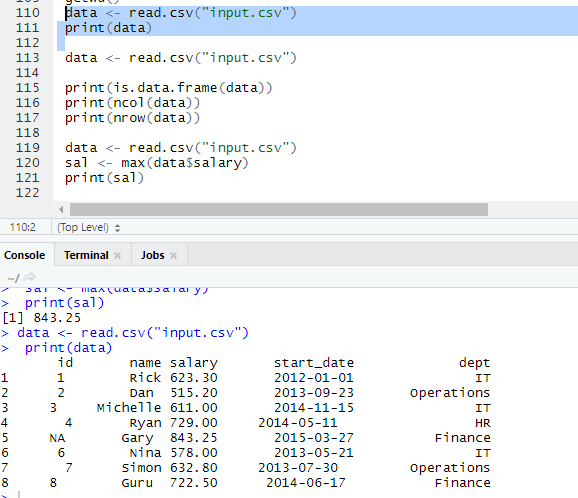
4 4 Ryan 729.00 2014-05-11 HR

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))

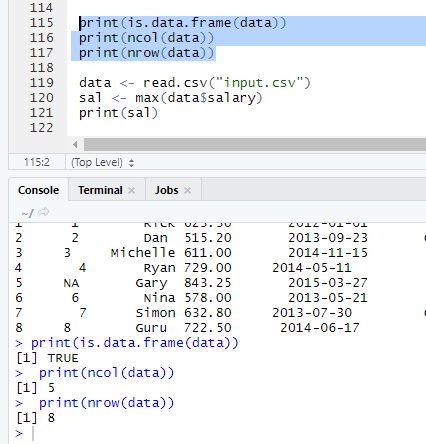
print(nrow(data))

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")

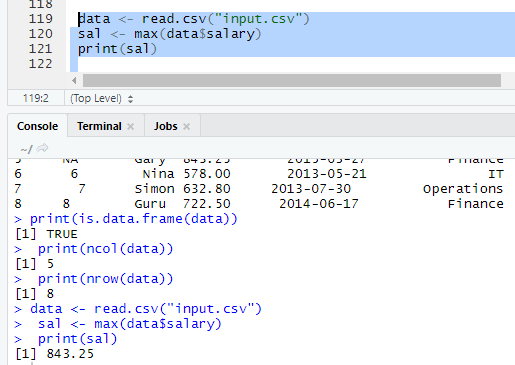
# Get the max salary from data frame.

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.

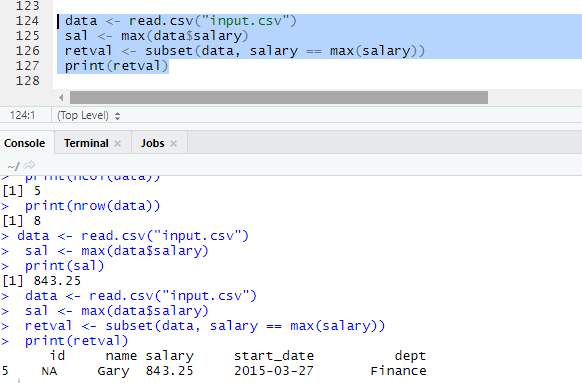
retval <- subset(data, salary == max(salary))

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)

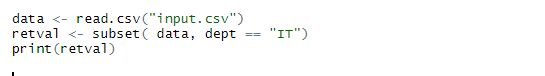
When we execute the above code, it produces the following result −

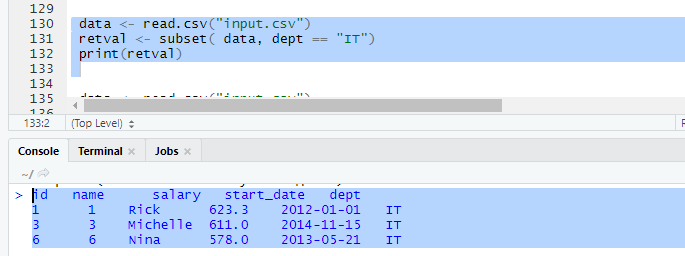
id name salary start\_date dept

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")

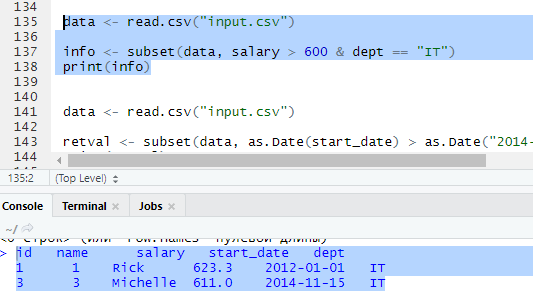
print(info)

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 −

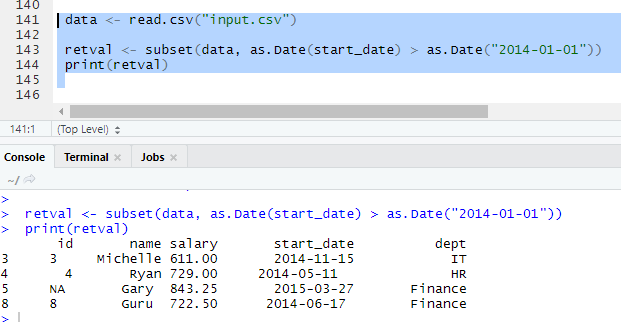
id name salary start\_date dept

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 −

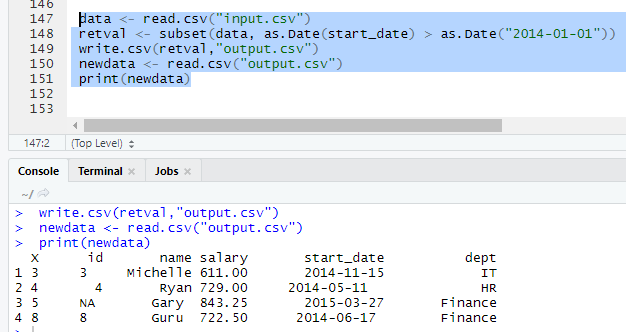
X id name salary start\_date dept

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 −

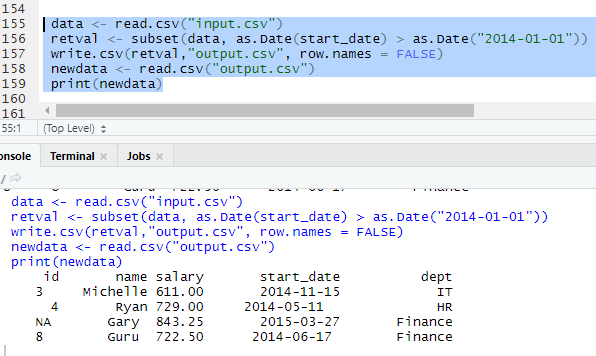
id name salary start\_date dept

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



**R - Excel File**

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

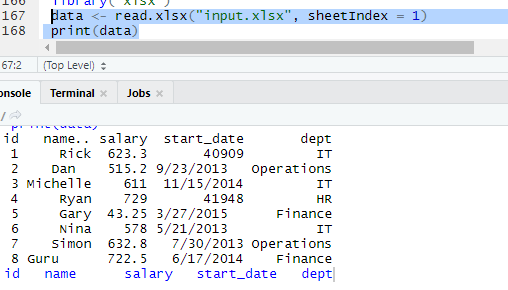
4 4 Ryan 729.00 2014-05-11 HR

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.

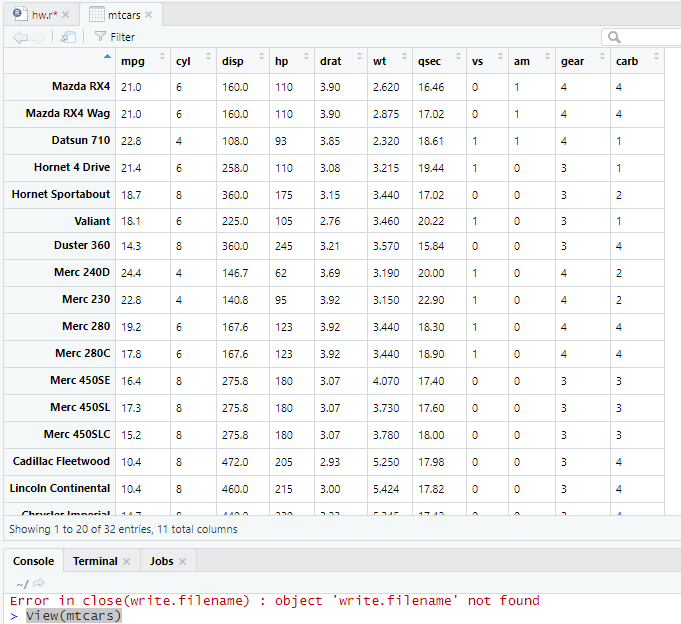
**Example**

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.

View(mtcars)



# Read the "mtcars" data frame as a csv file and store only the columns

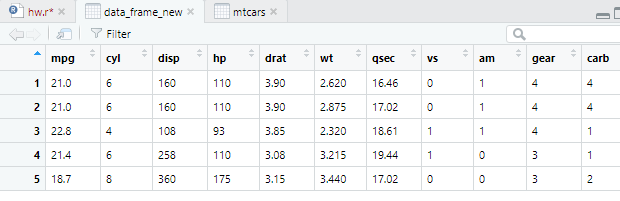
"cyl", "am" and "gear".

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)

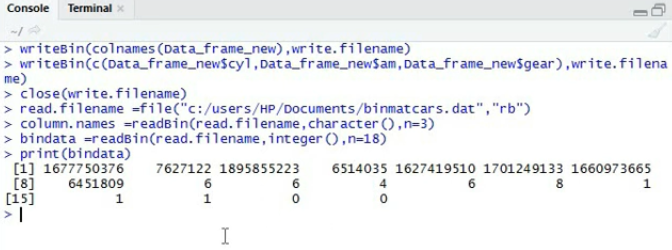
# Next read the column values. n = 18 as we have 3 column names and 15 values.

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

As we can see, we got the original data back by reading the binary file in R.

**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

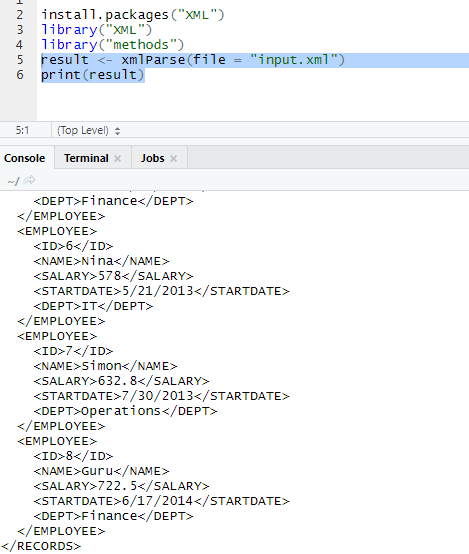
8

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)

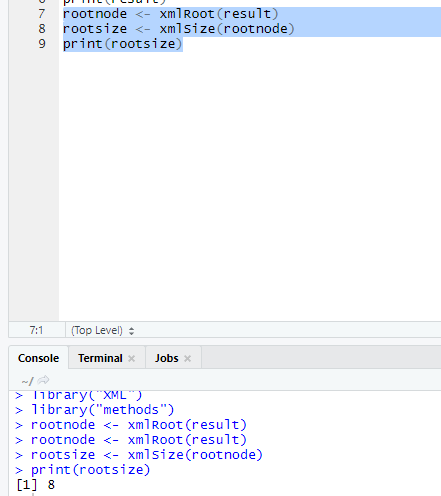
# Print the result.

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

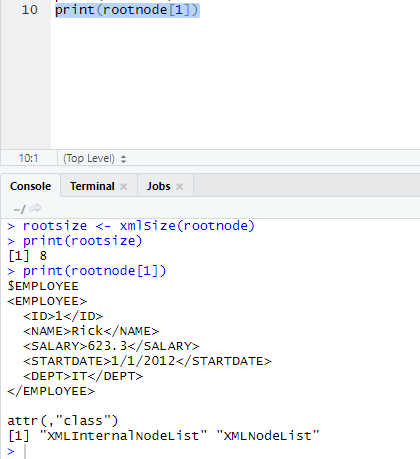
1

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.

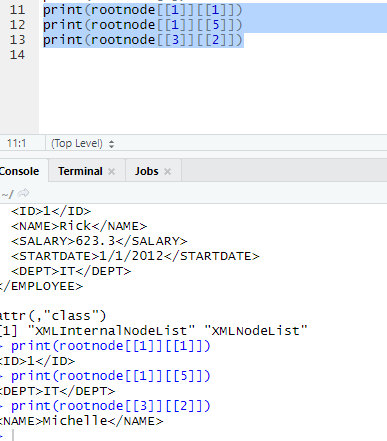
print(rootnode[[3]][[2]])

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

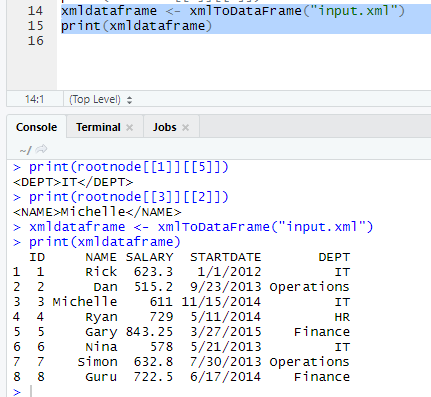
4 4 Ryan 729.00 2014-05-11 HR

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

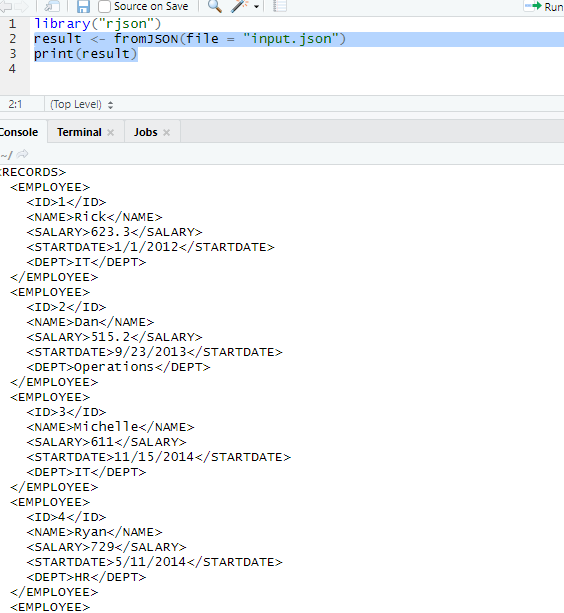
[1] "1/1/2012" "9/23/2013" "11/15/2014" "5/11/2014" "3/27/2015" "5/21/2013"

"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

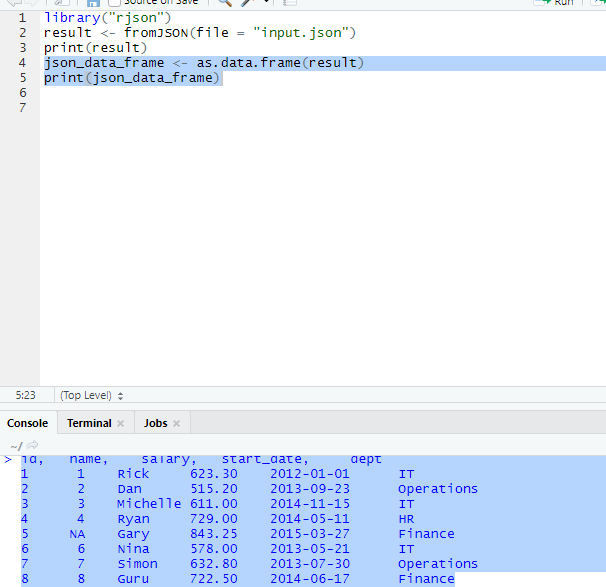
4 4 Ryan 729.00 2014-05-11 HR

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



**R - Web Data**

Many websites provide data for consumption by its users. For example the World Health Organization(WHO) provides reports on health and medical information in the form of CSV, txt and XML files. Using R programs, we can programmatically extract specific data from such websites. Some packages in R which are used to scrap data form the web are − "RCurl",XML", and "stringr". They are used to connect to the URL’s, identify required links for the files and download them to the local environment.

**Install R Packages**

The following packages are required for processing the URL’s and links to the files. If they are not available in your R Environment, you can install them using following commands.

install.packages("RCurl")

install.packages("XML")

install.packages("stringr")

install.packages("plyr")

**Input Data**

We will visit the URL [weather data](https://www.geos.ed.ac.uk/~weather/jcmb_ws/" \t "_blank) and download the CSV files using R for the year 2015.

**Example**

We will use the function **getHTMLLinks()** to gather the URLs of the files. Then we will use the function **download.file()** to save the files to the local system. As we will be applying the same code again and again for multiple files, we will create a function to be called multiple times. The filenames are passed as parameters in form of a R list object to this function.

# Read the URL.

url <- "http://www.geos.ed.ac.uk/~weather/jcmb\_ws/"

# Gather the html links present in the webpage.

links <- getHTMLLinks(url)

# Identify only the links which point to the JCMB 2015 files.

filenames <- links[str\_detect(links, "JCMB\_2015")]

# Store the file names as a list.

filenames\_list <- as.list(filenames)

# Create a function to download the files by passing the URL and filename list.

downloadcsv <- function (mainurl,filename) {

filedetails <- str\_c(mainurl,filename)

download.file(filedetails,filename)

}

# Now apply the l\_ply function and save the files into the current R working directory.

l\_ply(filenames,downloadcsv,mainurl = "http://www.geos.ed.ac.uk/~weather/jcmb\_ws/")

**Verify the File Download**

After running the above code, you can locate the following files in the current R working directory.

"JCMB\_2015.csv" "JCMB\_2015\_Apr.csv" "JCMB\_2015\_Feb.csv" "JCMB\_2015\_Jan.csv"

"JCMB\_2015\_Mar.csv"

