MARUDHAR KESARI JAIN COLLEGE FOR WOMEN (AUTONOMOUS)

VANIYAMBADI

PG Department of Computer Applications

I MCA – Semester - II

E-Notes (Study Material)

Core Course -1: Data Analytics and Visualization	**Code: 24PCAC21**
Unit: 2 - Reading and getting data into R (External Data): Using CSV files, XML files, Web Data, JSON files, Databases, Excel files. Working with R Charts and Graphs: Histograms, Boxplots, Bar Charts, Line Graphs, Scatterplots, Pie Charts.	
Learning Objectives: To learn about the getting data In and Out of R	
Course Outcome: To understand the getting data In and Out of R	

Overview:

- Reading and getting data into R
- CSv , Excel, Json files
- Working with R Charts and Graphs

1. Introduction to Reading and getting data into R (External Data)

It refers to the process of importing data from various external sources (outside of R) into the R environment for analysis. R is often used for data analysis, and to work with real-world data, you typically need to load data from different file formats, databases, web APIs, or cloud-based services.

External data could come in many forms, such as:

- **CSV files**: Comma-separated values files are a common and simple format for storing data.
- **Excel files**: Data stored in .xls or .xlsx formats.
- **Databases**: Data stored in relational databases like MySQL, PostgreSQL, or SQLite.
- **JSON or XML files**: These are often used for web APIs and complex data structures.
- **Google Sheets**: Data stored on cloud-based spreadsheets.
- **Web scraping/API**: Data collected directly from websites or APIs (like social media, financial data, etc.).

Why it's important:

- **Data access**: Often, the data you need for analysis is not directly in R. You may have it in files on your computer, on the web, or stored in a database.
- **Data integration**: You might need to pull data from multiple sources into R to combine and analyze it in a single workflow.
- **Automation**: By learning how to read external data, you can automate the process of data collection and analysis (e.g., pulling daily stock prices or retrieving the latest sales data).

Steps in Reading and Getting Data into R:

1. **Locate the Data**: Identify where your data resides (local machine, web, database, etc.).
2. **Select the Appropriate Method**: Choose the right function or package based on the data format (CSV, Excel, database, etc.).
3. **Load the Data into R**: Use a function (like read.csv(), read_excel(), dbReadTable(), etc.) to read and import the data into an R object (usually a data frame).
4. **Examine the Data**: Once the data is loaded, you can use functions like head(), summary(), and str() to inspect its contents and structure.

2. Reading Tabular Data Files:

If you have a .txt or a tab-delimited text file, or a file with table like structure then you can easily import it by using the basic R function read.table(). It's good to know that the read.table() function is the most important and commonly used function to import simple data files into R. It is easy and flexible. Reads a file in table format and creates a data frame from it.

Syntax: read.table(file, header = FALSE, sep = "", quote = "\"", dec = ".", row.names, col.names, ……)

Here

• file: You have to specify the file name, or Full path along with file name. You can also use the URL of the external (online) txt files. For example, sampleFile.txt or "C:/Users/Suresh/Documents/R Programs/sampleFile.txt"

• header: If the text file contains Columns names as the First Row then please specify TRUE otherwise, FALSE

• sep: It is a short form of separator. You have to specify the character that is separating the fields. ", "means data is separated by comma. The default separator is "white space", that is one or more spaces, tabs, carriage return etc.

• quote: the set of quoting characters. To disable quoting altogether, use quote = "".If your character values (ex: Last-Name, Occupation, Education column etc) are enclosed in

quotes then you have to specify the quote type. For double quotes we use: quote = “\””.

• dec: the character used in the file for decimal points.

• row.names: A Character vector that contains the row names for the returned data frame

Example 1: A data table can resides in a text file. The cells inside the table are separated by blank characters. Here is an example of a table with 4 rows and 3 columns.

100 a1 b1

200 a2 b2

300 a3 b3

400 a4 b4

Now copy and paste the table above in a file named "mydata.txt" with a text editor. Then load the data into the workspace with the function read.table.

> mydata = read.table("mydata.txt") # read text file

> mydata

print data frame

V1 V2 V3

1 100 a1 b1

2 200 a2 b2

3 300 a3 b3

4 400 a4 b4

>mydata1=read.table(“rain.txt

Example 2:

rain<- read.table("C:/Users/SUNITHA/Desktop/rain.txt",header=TRUE,sep=",")

o/p: rain

month rain_mm flow_cmm

1 1 128 15000

2 2 98 12000

3 3 92 11000

4 4 77 9800

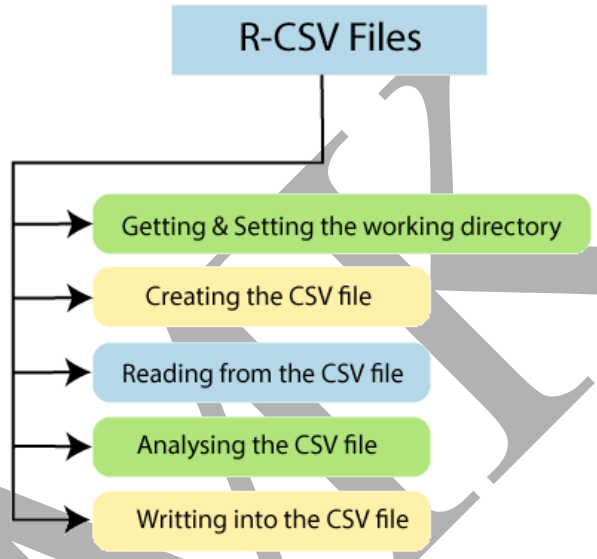
5 5 68 7600

3. CSV Files

A **Comma-Separated Values (CSV) file** is a plain text file which contains a list of data. These files are often used for the exchange of data between different applications. For example, databases and contact managers mostly support CSV files.

These files can sometimes be called **character-separated values** or **comma-delimited files**. They often use the comma character to separate data, but sometimes use other characters such as semicolons. The idea is that we can export the complex data from one application to a CSV file, and then importing the data in that CSV file to another application.

Storing data in excel spreadsheets is the most common way for data storing, which is used by the data scientists. There are lots of packages in R designed for accessing data from the excel spreadsheet. Users often find it easier to save their spreadsheets in comma-separated value files and then use R's built-in functionality to read and manipulate the data.



3.1 Getting and setting the working directory

In R, getwd() and setwd() are the two useful functions. The getwd() function is used to check on which directory the R workspace is pointing. And the setwd() function is used to set a new working directory to read and write files from that directory.

Let's see an example to understand how getwd() and setwd() functions are used.

Example

```
# Getting and printing current working directory.
print(getwd())
# Setting the current working directory.
setwd("C:/Users/ajeet")
# Getting and printingthe current working directory.
print(getwd())
```

3.2 Creating a CSV File

A text file in which a comma separates the value in a column is known as a CSV file. Let's start by creating a CSV file with the help of the data, which is mentioned below by saving with .csv extension using the save As All files(*.*) option in the notepad.

Example: record.csv

```
id,name,salary,start_date,dept
1,Shubham,613.3,2012-01-01,IT
2,Arpita,525.2,2013-09-23,Operations
3,Vaishali,63,2014-11-15,IT
4,Nishka,749,2014-05-11,HR
5,Gunjan,863.25,2015-03-27,Finance
6,Sumit,588,2013-05-21,IT
7,Anisha,932.8,2013-07-30,Operations
8,Akash,712.5,2014-06-17,Finance
```

Reading a CSV file

R has a rich set of functions. R provides read.csv() function, which allows us to read a CSV file available in our current working directory. This function takes the file name as an input and returns all the records present on it.

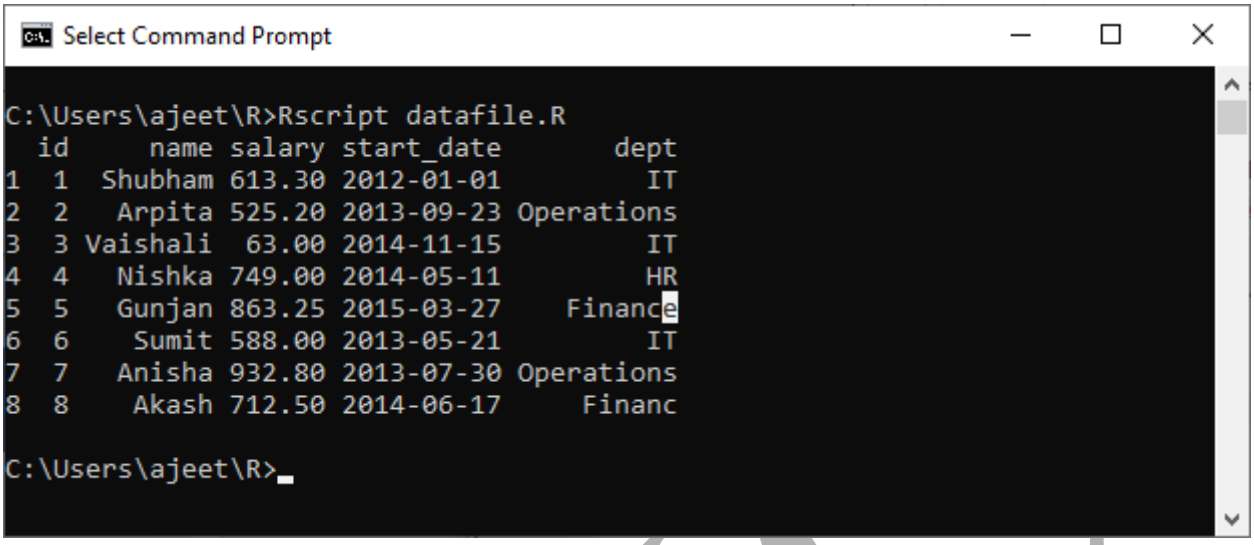
Let's use our record.csv file to read records from it using read.csv() function.

Example

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

When we execute above code, it will give the following output

Output



Analyzing the CSV File

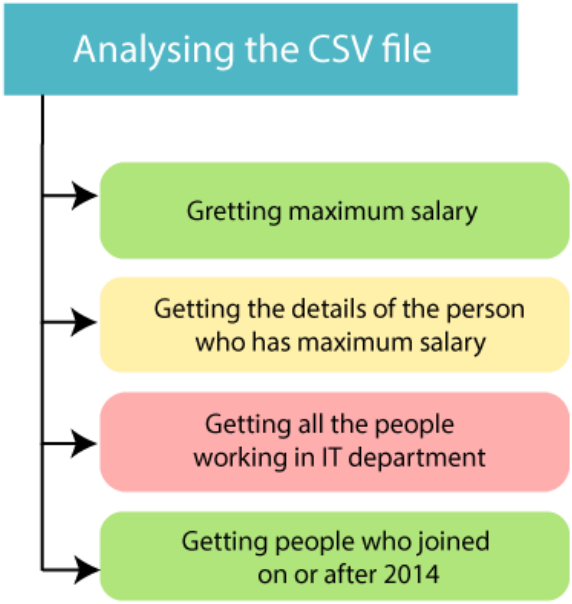
When we read data from the .csv file using **read.csv()** function, by default, it gives the output as a data frame. Before analyzing data, let's start checking the form of our output with the help of **is.data.frame()** function. After that, we will check the number of rows and number of columns with the help of **nrow()** and **ncol()** function.

Example

```
1. csv_data<- read.csv("record.csv")
2. print(is.data.frame(csv_data))
3. print(ncol(csv_data))
4. print(nrow(csv_data))
```

When we run above code, it will generate the following output:

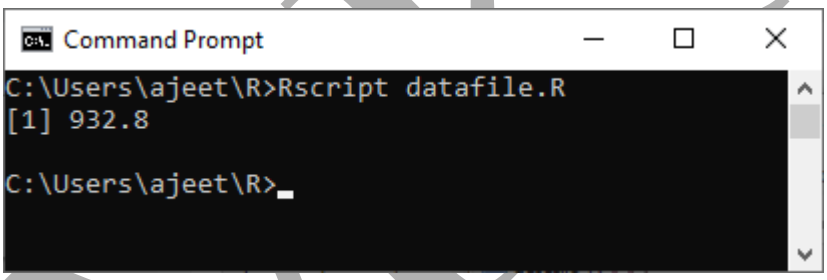
Output



Example: Getting the maximum salary

```
# Creating a data frame.
csv_data<- read.csv("record.csv")

# Getting the maximum salary from data frame.
max_sal<- max(csv_data$salary)
print(max_sal)
```



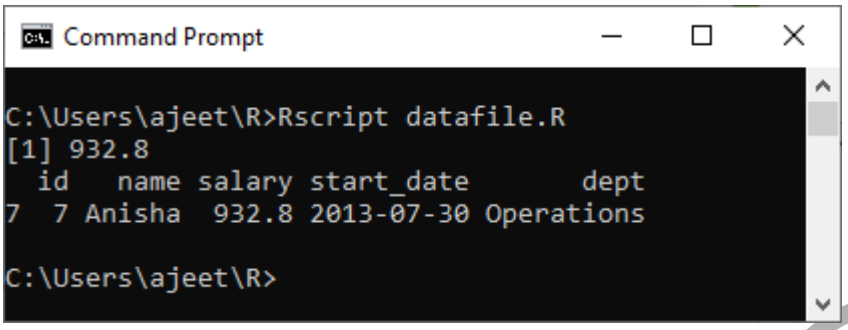
Example: Getting the details of the person who have a maximum salary

```
# Creating a data frame.
csv_data<- read.csv("record.csv")

# Getting the maximum salary from data frame.
max_sal<- max(csv_data$salary)
print(max_sal)

#Getting the detais of the pweson who have maximum salary
details <- subset(csv_data,salary==max(salary))
print(details)
```

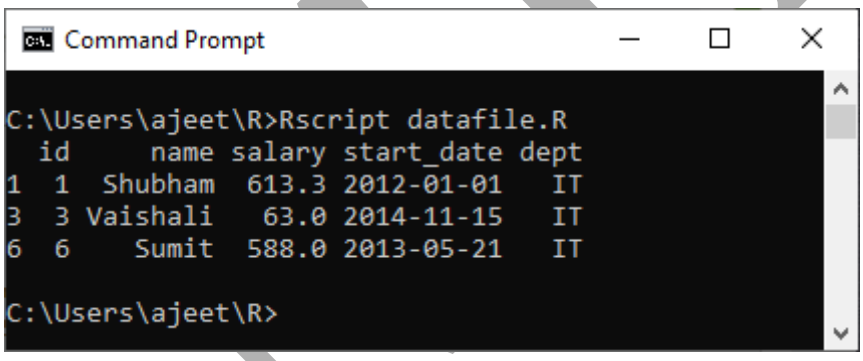
Output



Example: Getting the details of all the persons who are working in the IT department

1. # Creating a data frame.
2. csv_data<- read.csv("record.csv")
3. #Getting the detais of all the pweson who are working in IT department
4. details <- subset(csv_data,dept=="IT")
5. print(details)

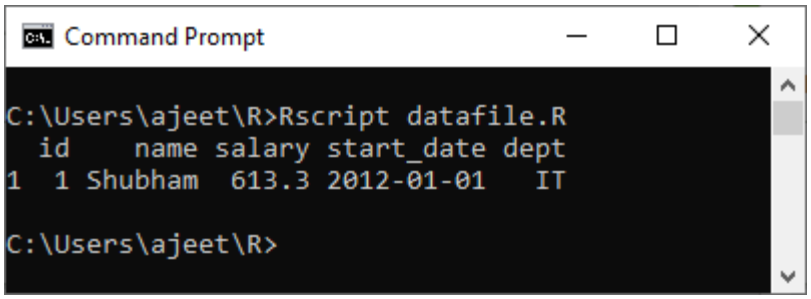
Output



Example: Getting the details of the persons whose salary is greater than 600 and working in the IT department.

1. # Creating a data frame.
2. csv_data<- read.csv("record.csv")
3. #Getting the detais of all the pweson who are working in IT department
4. details <- subset(csv_data,dept=="IT"&salary>600)
5. print(details)

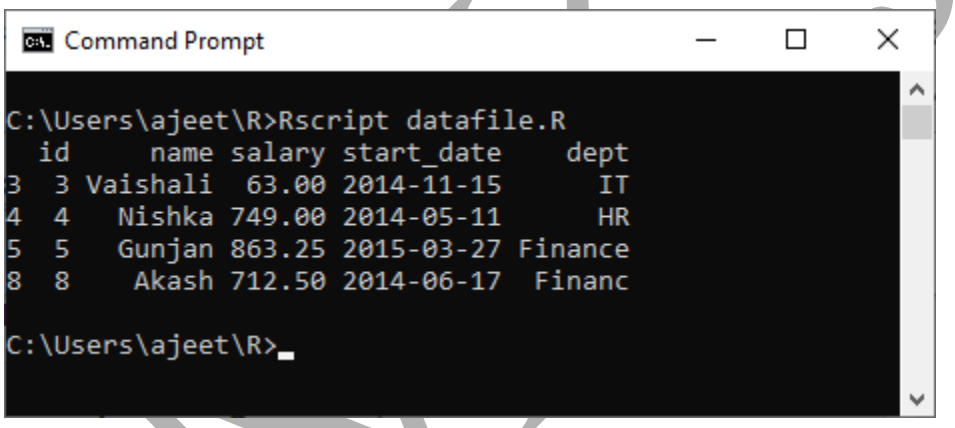
Output



Example: Getting details of those peoples who joined on or after 2014.

1. # Creating a data frame.
2. csv_data<- read.csv("record.csv")
3. #Getting details of those peoples who joined on or after 2014
4. details <- subset(csv_data,as.Date(start_date)>as.Date("2014-01-01"))
5. print(details)

Output



Writing into a CSV file

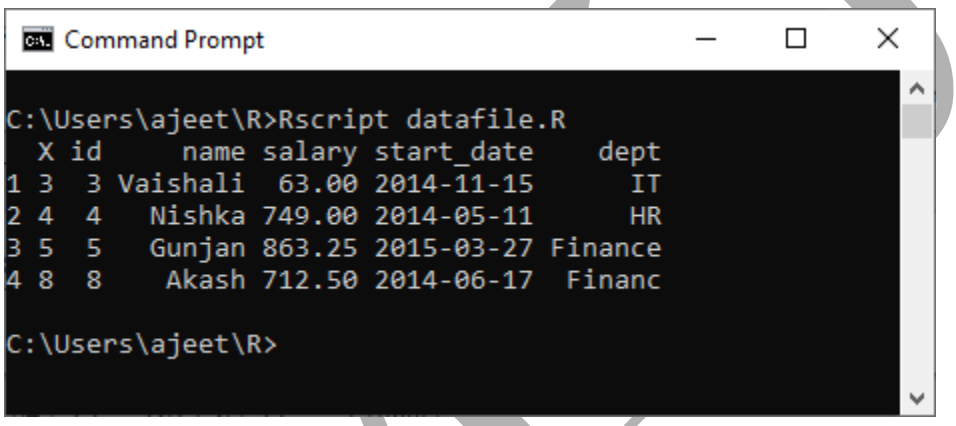
Like reading and analyzing, R also allows us to write into the .csv file. For this purpose, R provides a write.csv() function. This function creates a CSV file from an existing data frame. This function creates the file in the current working directory.

Let's see an example to understand how **write.csv()** function is used to create an output CSV file.

Example

```
1. csv_data<- read.csv("record.csv")
2. #Getting details of those peoples who joined on or after 2014
3. details <- subset(csv_data,as.Date(start_date)>as.Date("2014-01-01"))
4. # Writing filtered data into a new file.
5. write.csv(details,"output.csv")
6. new_details<- read.csv("output.csv")
7. print(new_details)
```

Output



4.XML File

Like HTML, XML is also a markup language which stands for Extensible Markup Language. It is developed by World Wide Web Consortium(W3C) to define the syntax for encoding documents which both humans and machine can read. This file contains markup tags. There is a difference between HTML and XML. In HTML, the markup tag describes the structure of the page, and in xml, it describes the meaning of the data contained in the file. In R, we can read the xml files by installing "XML" package into the R environment. This package will be installed with the help of the familiar command i.e., install.packages.

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

Creating XML File

We will create an xml file with the help of the given data. We will save the following data with the .xml file extension to create an xml file. XML tags describe the meaning of data, so that data contained in such tags can easily tell or explain about the data.

```
<records>
<employee_info>
<id>1</id>
<name>Shubham</name>
<salary>623</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

<employee_info>
<id>2</id>
<name>Nishka</name>
<salary>552</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

<employee_info>
<id>1</id>
<name>Gunjan</name>
<salary>669</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

<employee_info>
<id>1</id>
<name>Sumit</name>
<salary>825</salary>
<date>1/1/2012</date>
```

```
<dept>IT</dept>
</employee_info>

<employee_info>
<id>1</id>
<name>Arpita</name>
<salary>762</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

<employee_info>
<id>1</id>
<name>Vaishali</name>
<salary>882</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

<employee_info>
<id>1</id>
<name>Anisha</name>
<salary>783</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

<employee_info>
<id>1</id>
<name>Ginni</name>
<salary>964</salary>
<date>1/1/2012</date>
<dept>IT</dept>
</employee_info>

</records>
```

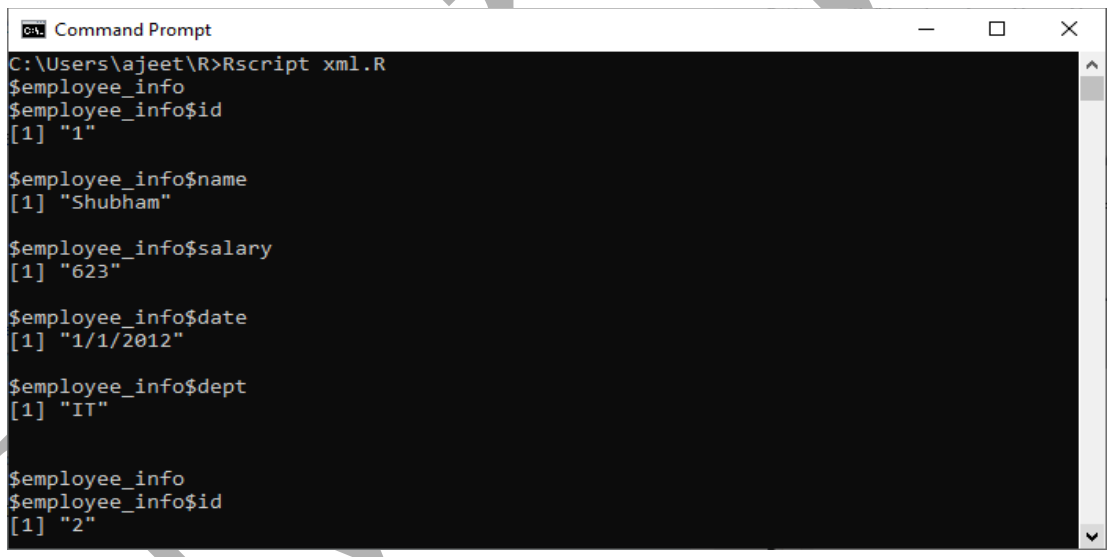
Reading XML File

In R, we can easily read an xml file with the help of xmlParse() function. This function is stored as a list in R. To use this function, we first need to load the xml package with the help of the library() function. Apart from the xml package, we also need to load one additional package named methods.

Example: Reading xml data in the form of a list.

```
# Loading the package required to read XML files.
library("XML")
# Also loading the other required package.
library("methods")
# Giving the input file name to the function.
result <- xmlParse(file = "xml_data.xml")
xml_data <- xmlToList(result)
print(xml_data)
```

Output



How to convert xml data into a data frame

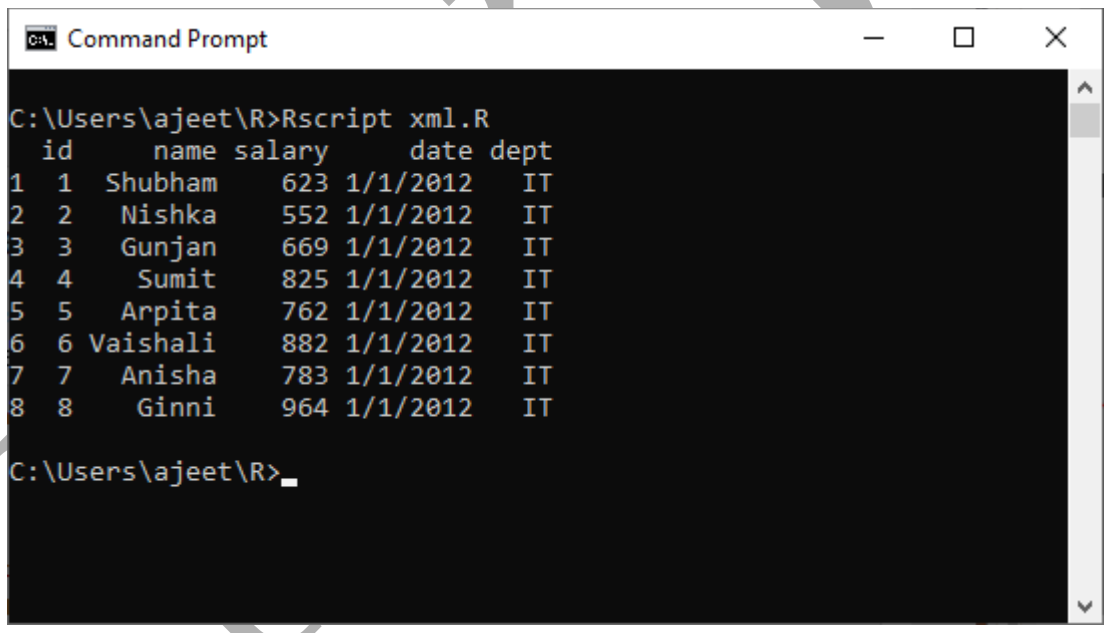
It's not easy to handle data effectively in large files. For this purpose, we read the data in the xml file as a data frame. Then this data frame is processed by the data analyst. R provide xmlToDataFrame() function to extract the information in the form of Data Frame.

Let's see an example to understand how this function is used and processed:

Example

```
1. # Loading the package required to read XML files.
2. library("XML")
3. # Also loading the other required package.
4. library("methods")
5. # Giving the input file name to the function xmlToDataFrame.
6. data_frame <- xmlToDataFrame("xml_data.xml")
7. #Printing the result
8. print(data_frame)
```

Output



Practice Questions:

1.Explain the working with XML in R programming?

2. How to convert xml data into a data frame

3. Define XML

4. What is meant by dataframe?

5.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")
```

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"

6.Database

In the relational database management system, the data is stored in a normalized format. Therefore, to complete statistical computing, we need very advanced and complex SQL queries. The large and huge data which is present in the form of tables require SQL queries to extract the data from it.

R can easily connect with many of the relational databases like MySql, SQL Server, Oracle, etc. When we extract the information from these databases, by default, the information is extracted in the form of data frame. Once, the data comes from the database to the R environment; it will become a normal R dataset. The data analyst can easily analyze or manipulate the data with the help of all the powerful packages and functions.

RMySQL Package

RMySQL package is one of the most important built-in package of R. This package provides native connectivity between the R and MySql database. In R, to work with MySql database, we first have to install the RMySQL package with the help of the familiar command, which is as follows:

```
install.packages("RMySQL")
```

When we run the above command in the R environment, it will start downloading the package RMySQL.

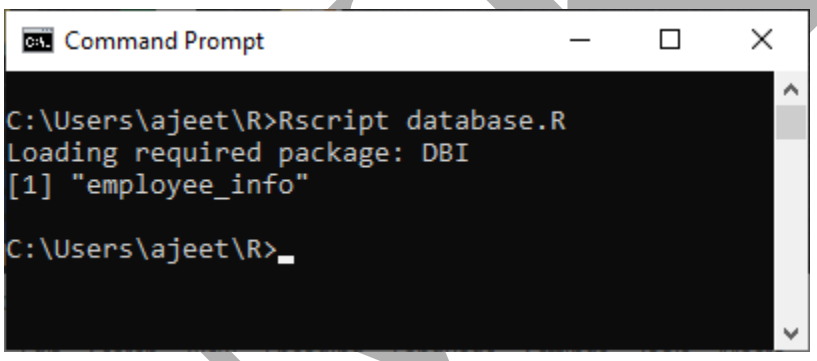
Create a connection between R and MySql

To work with MySql database, it is required to create a connection object between R and the database. For creating a connection, R provides **dbConnect()** function. This function takes the username, password, database name, and host name as input parameters. Let's see an example to understand how the **dbConnect()** function is used to connect with the database.

Example

```
#Loading RMySQL package into R
library("RMySQL")
# Creating a connection Object to MySQL database.
# Conneting with database named "employee" which we have created befoe with the helpof XA MPP server.
mysql_connect = dbConnect(MySQL(), user = 'root', password = '', dbname = 'employee',
  host = 'localhost')
# Listing the tables available in this database.
 dbListTables(mysql_connect)
```

Output



R MySQL Commands

In R, we can perform all the SQL commands like insert, delete, update, etc. For performing the query on the database, R provides the dbSendQuery() function. The query is executed in MySQL, and the result set is returned using the R fetch () function. Finally, it is stored in R as a data frame. Let's see the example of each and every SQL command to understand how dbSendQuery() and fetch() functions are used.

Create Table

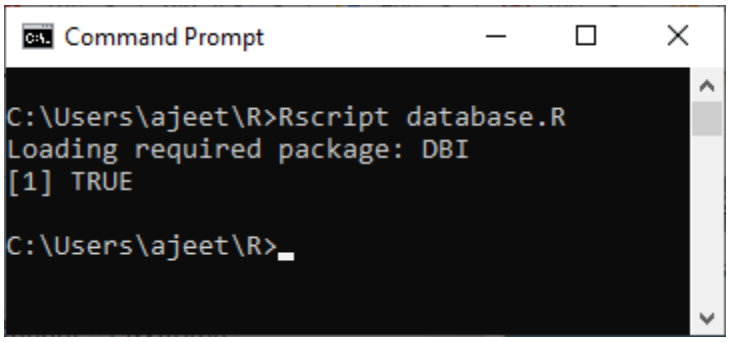
R provides an additional function to create a table into the database i.e., dbWriteTable(). This function creates a table in the database; if it does not exist else, it will overwrite the table. This function takes the data frame as an input.

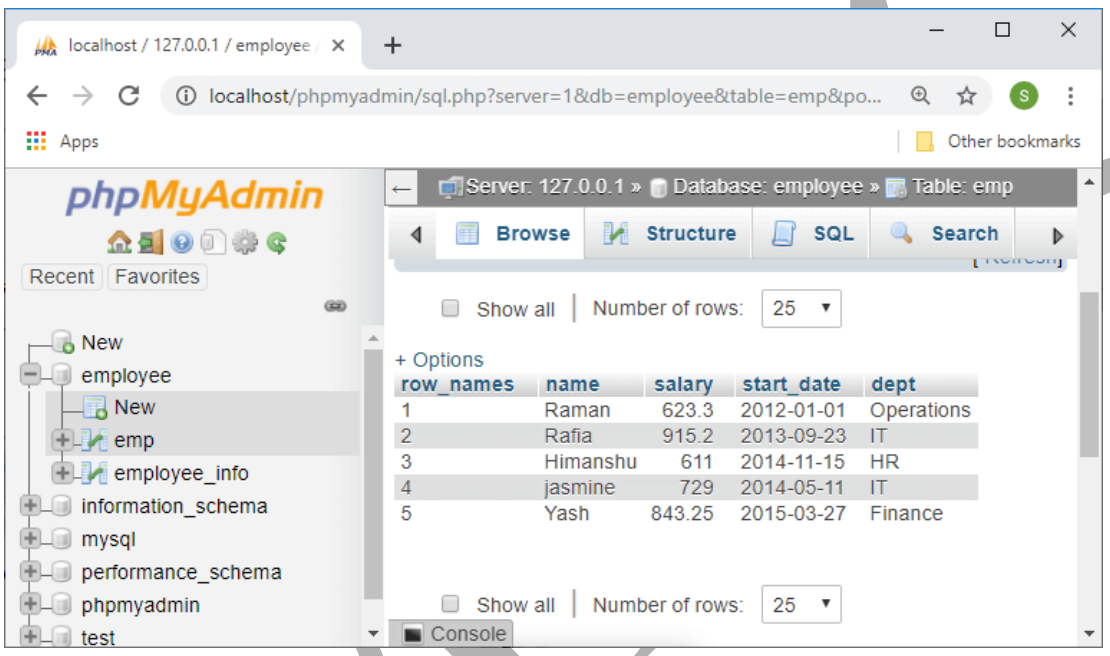
Example

```
#Loading RMySQL package into R
library("RMySQL")
# Creating a connection Object to MySQL database.
# Conneting with database named "employee" which we have created befoe with the helpof XA MPP server.
mysql_connect = dbConnect(MySQL(), user = 'root', password = '', dbname = 'employee',
   host = 'localhost')
#Creating data frame to create a table
emp.data<- data.frame(
name = c("Raman","Rafia","Himanshu","jasmine","Yash"),
salary = c(623.3,915.2,611.0,729.0,843.25),
start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11","2015-03-27")),
dept = c("Operations","IT","HR","IT","Finance"),
stringsAsFactors = FALSE  )
```

13. # All the rows of emp.data are taken inot MySql.

14. dbWriteTable(mysql_connect, "emp", emp.data[,], overwrite = TRUE)





Select

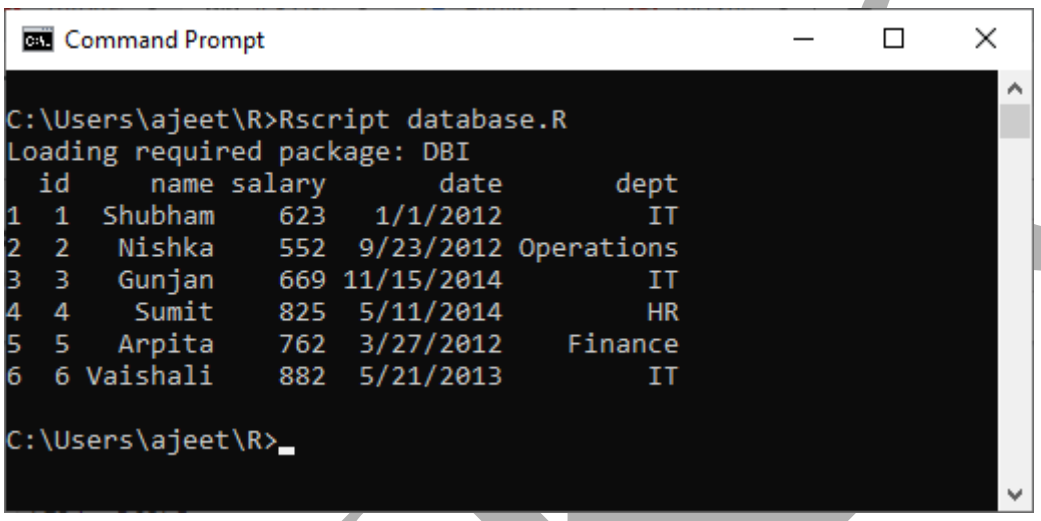
We can simply select the record from the table with the help of the fetch() and dbSendQuery() function. Let's see an example to understand how to select query works with these two functions.

Example

1. #Loading RMySQL package into R
2. library("RMySQL")
3. # Creating a connection Object to MySQL database.
4. # Conneting with database named "employee" which we have created befoe with the helpof XA MPP server.
5. mysql_connect = dbConnect(MySQL(), user = 'root', password = '', dbname = 'employee',

```
    host = 'localhost')
# selecting the record from employee_info table.
record = dbSendQuery(mysql_connect, "select * from employee_info")
# Storing the result in a R data frame object. n = 6 is used to fetch first 6 rows.
data_frame = fetch(record, n = 6)
print(data_frame)
```

Output



Insert command

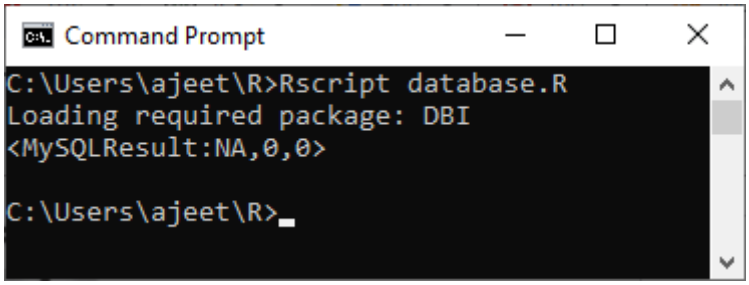
We can insert the data into tables with the help of the familiar method dbSendQuery() function.

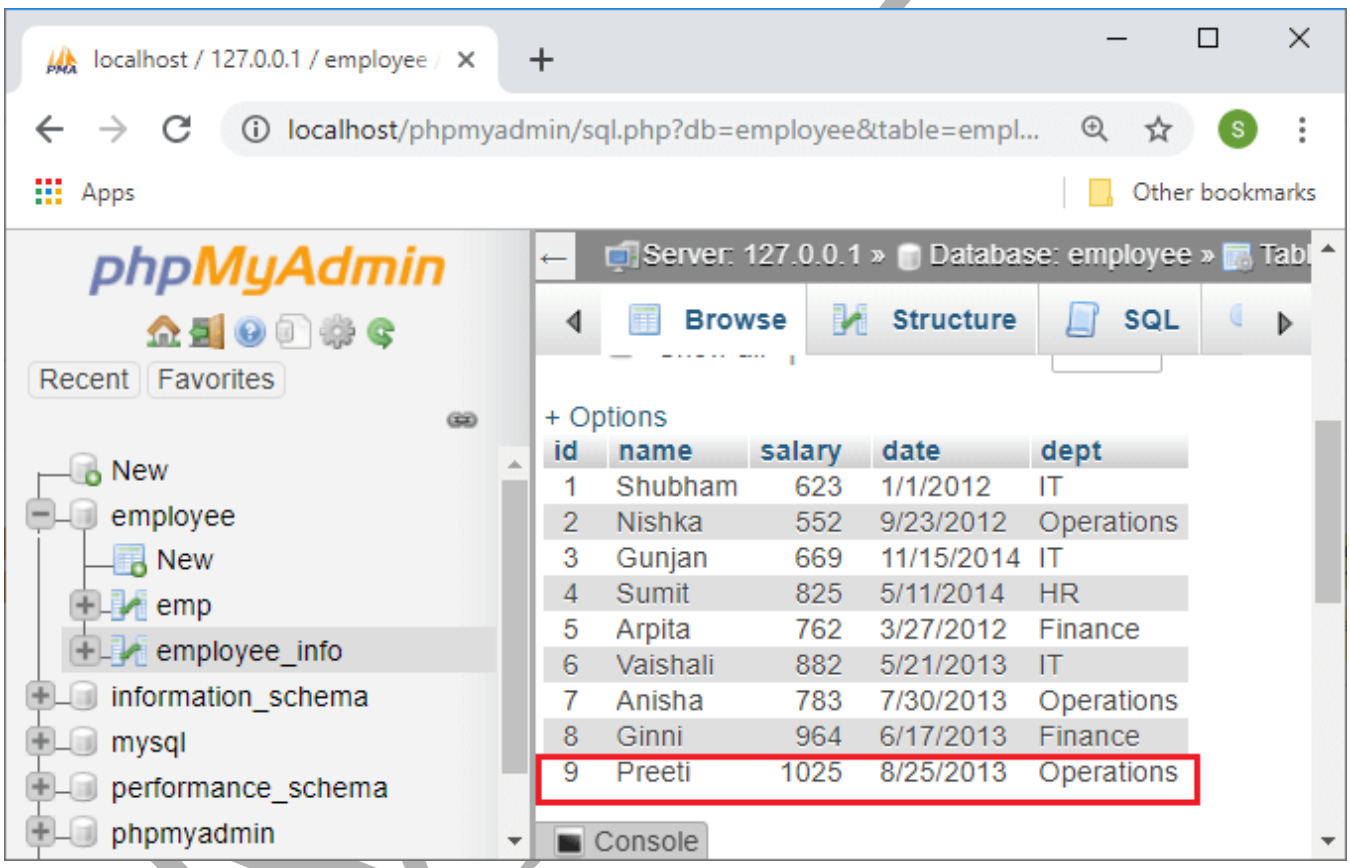
Example

```
#Loading RMySQL package into R
library("RMySQL")
# Creating a connection Object to MySQL database.
# Conneting with database named "employee" which we have created befoe with the helpof XA MPP server.
mysql_connect = dbConnect(MySQL(), user = 'root', password = '', dbname = 'employee',
    host = 'localhost')
# Inserting record into employee_info table.
```

8. dbSendQuery(mysql_connect, "insert into employee_info values(9,'Preeti',1025,'8/25/2013','Operations')")

Output





Update command

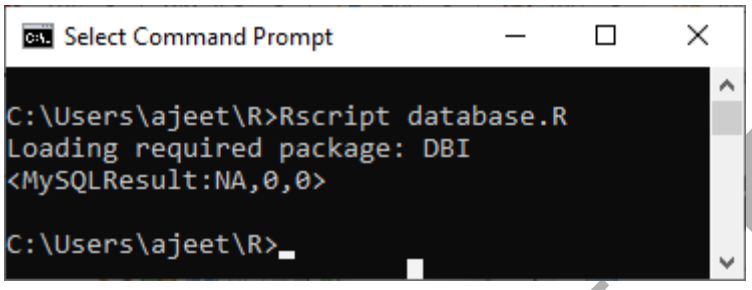
Updating a record in the table is much easier. For this purpose, we have to pass the update query to the dbSendQuery() function.

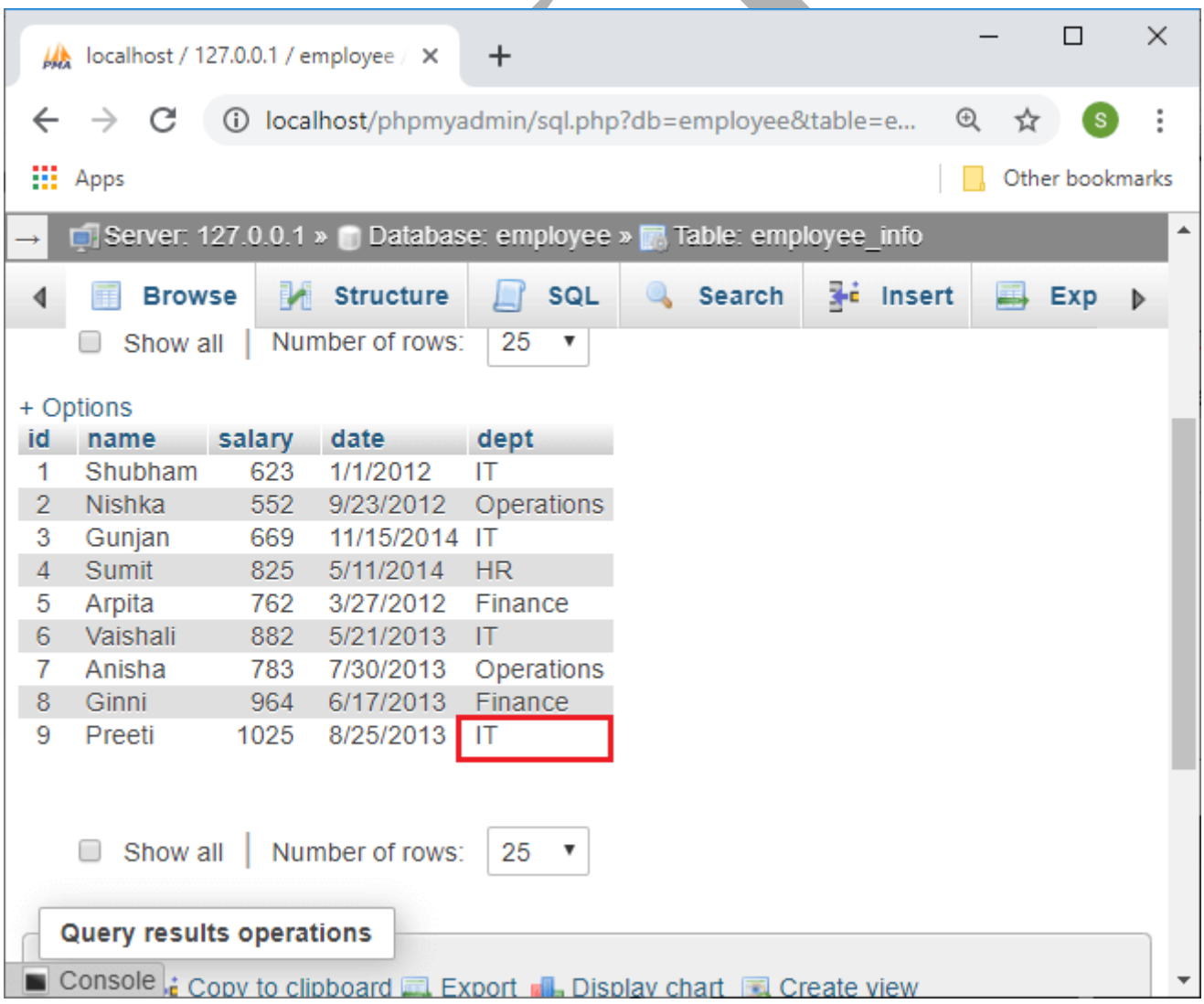
Example

1. #Loading RMySQL package into R
2. library("RMySQL")
3. # Creating a connection Object to MySQL database.

4. # Conneting with database named "employee" which we have created befoe with the helpof XA MPP server.
5. mysql_connect = dbConnect(MySQL(), user = 'root', password = '', dbname = 'employee',
6. host = 'localhost')
7. # Updating the record in employee_info table.
8. dbSendQuery(mysql_connect, "update employee_info set dept='IT' where id=9")

Output





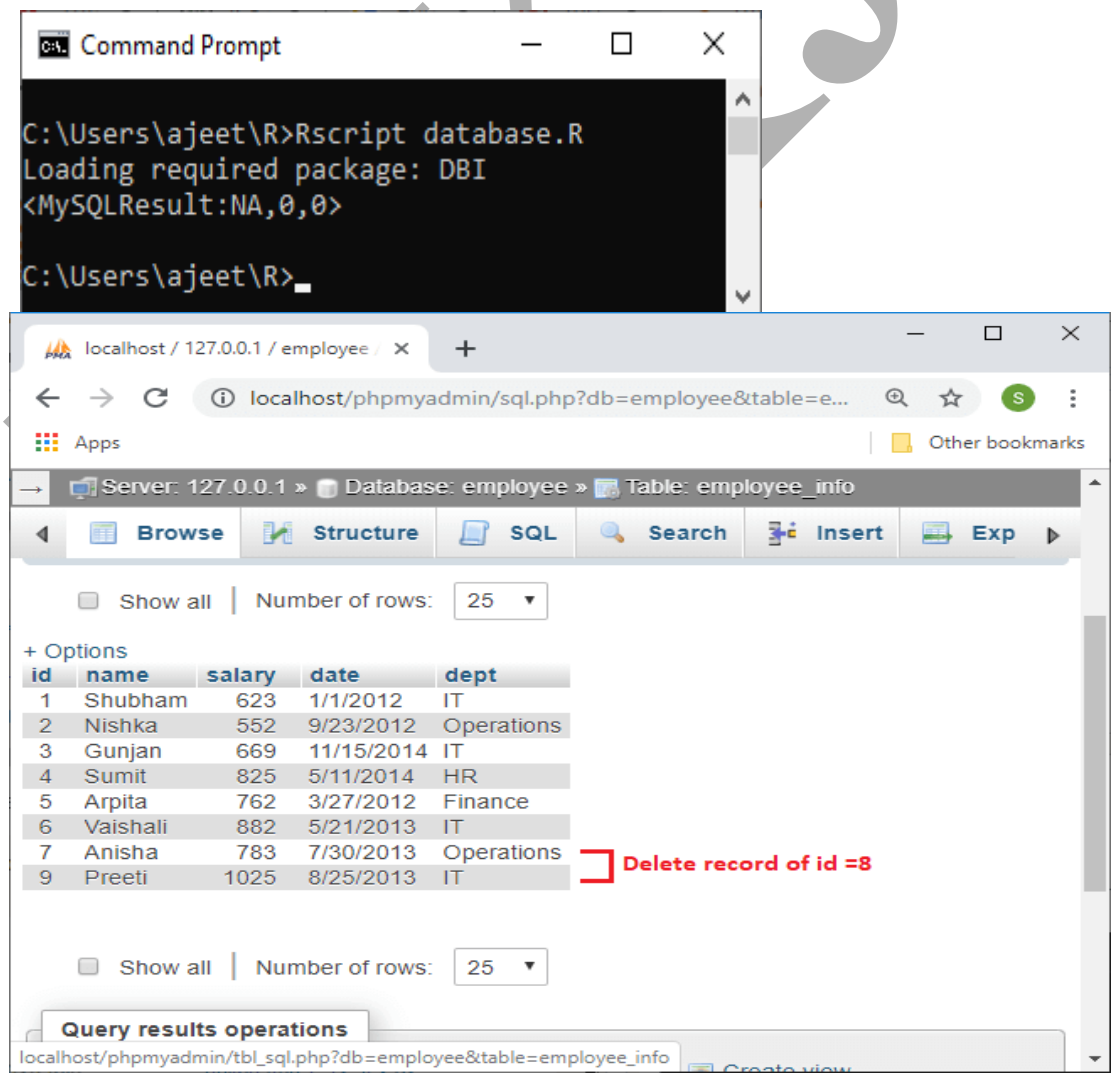
Delete command

Below is an example in which we delete a specific row from the table by passing the delete query in the dbSendQuery() function.

Example

```
#Loading RMySQL package into R
library("RMySQL")
# Creating a connection Object to MySQL database.
# Conneting with database named "employee" which we have created befoe with the helpof XA MPP server.
mysql_connect = dbConnect(MySQL(), user = 'root', password = '', dbname = 'employee',
   host = 'localhost')
# Deleting the specific record from employee_info table.
dbSendQuery(mysql_connect, "delete from employee_info where id=8")
```

Output



7.JSON File

JSON stands for JavaScript Object Notation. The JSON file contains the data as text in a human-readable format. Like other files, we can also read and write into the JSON files. For this purpose, R provides a package named rjson, which we have to install with the help of the familiar command install.packages.

Install rjson package

By running the following command into the R console, we will install the rjson package into our current working directory.

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

Creating a JSON file

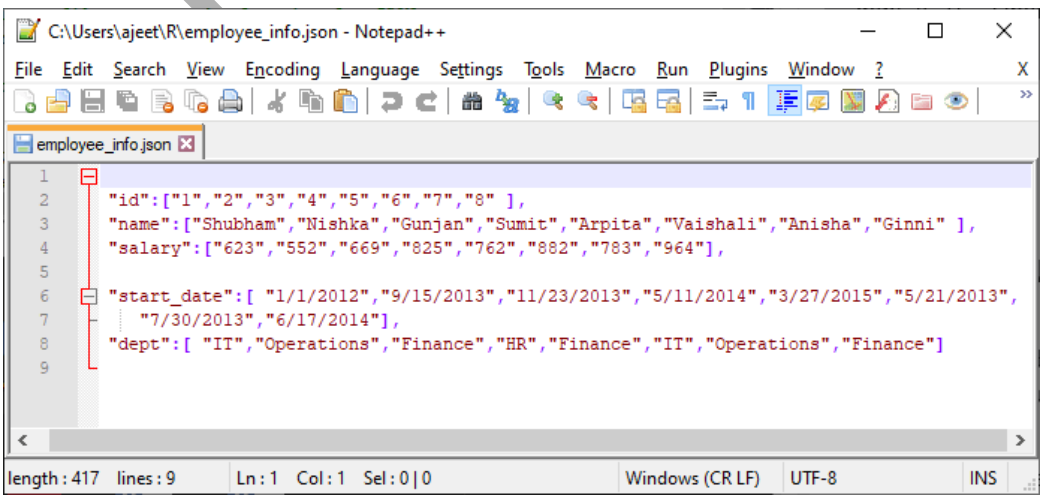
The extension of JSON file is .json. To create the JSON file, we will save the following data as employee_info.json. We can write the information of employees in any text editor with its appropriate rule of writing the JSON file. In JSON files, the information contains in between the curly braces({}).

Example: employee_info.json

```
{
  "id":["1","2","3","4","5","6","7","8" ],
  "name":["Shubham","Nishka","Gunjan","Sumit","Arpita","Vaishali","Anisha","Ginni" ],
  "salary":["623","552","669","825","762","882","783","964"],

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

Output



Read the JSON file

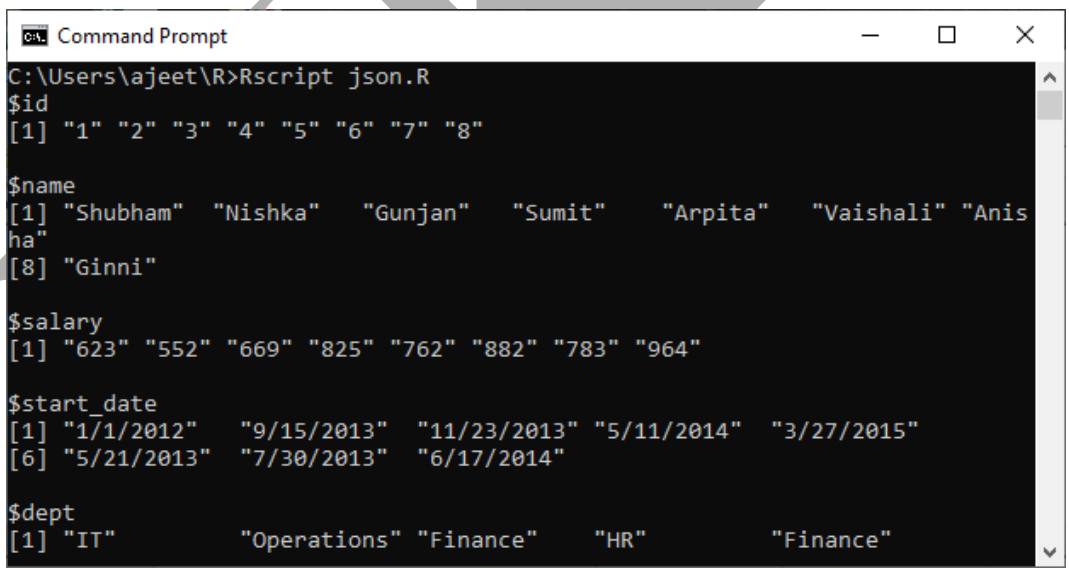
Reading the JSON file in R is a very easy and effective process. R provide from JSON() function to extract data from a JSON file. This function, by default, extracts the data in the form of a list. This function takes the JSON file and returns the records which are contained in it.

Let's see an example to understand how fromJSON() function is used to extract data and print the result in the form of a list. We will consider the employee_info.json file which we have created before.

Example

```
# Loading the package which is required to read JSON files.
library("rjson")
# Giving the input file name to the function fromJSON.
result <- fromJSON(file = "employee_info.json")
# Printing the result.
print(result)
```

Output



8.R Excel file

The xlsx is a file extension of a spreadsheet file format which was created by Microsoft to work with Microsoft Excel. In the present era, Microsoft Excel is a widely used spreadsheet program that sores data in the .xls or .xlsx format. R allows us to read data directly from these files by providing some excel specific packages. There are lots of packages such as XLConnect, xlsx, gdata, etc. We will use xlsx package, which not only allows us to read data from an excel file but also allow us to write data in it.

install xlsx Package

Our primary task is to install "xlsx" package with the help of install.package command. When we install the xlsx package, it will ask us to install some additional packages on which this package is dependent. For installing the additional packages, the same command is used with the required package name. There is the following syntax of install command:

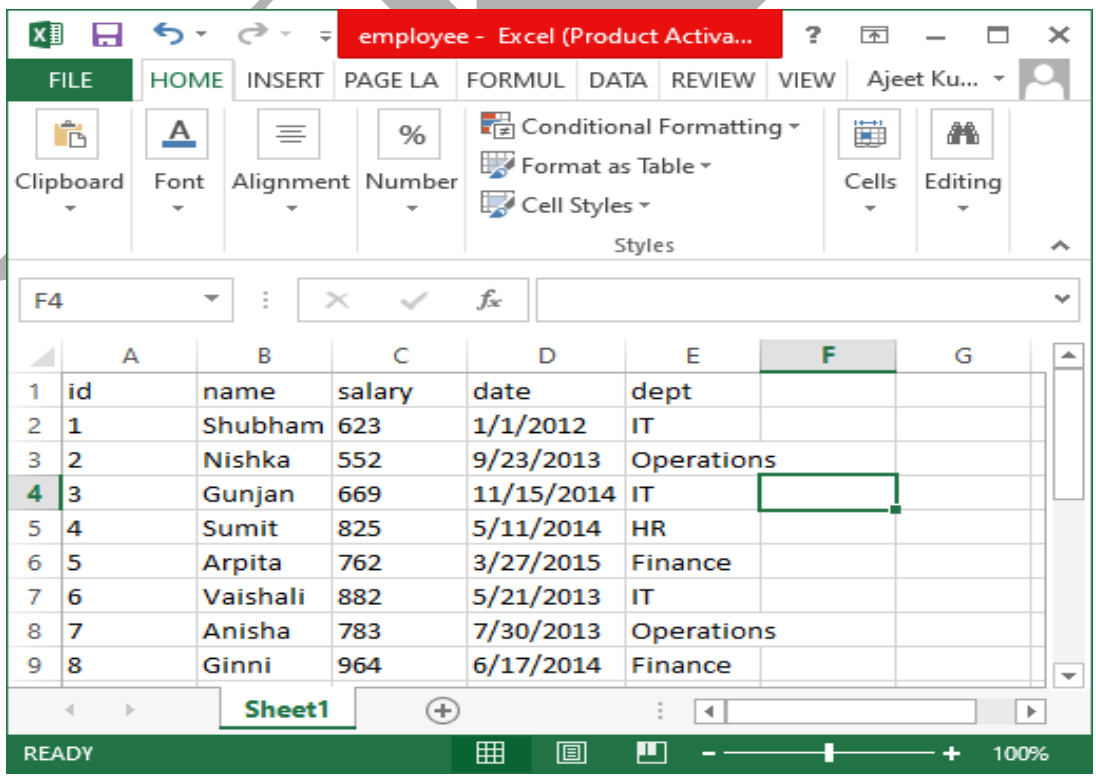
install.packages("package name")

Example

install.packages("xlsx")

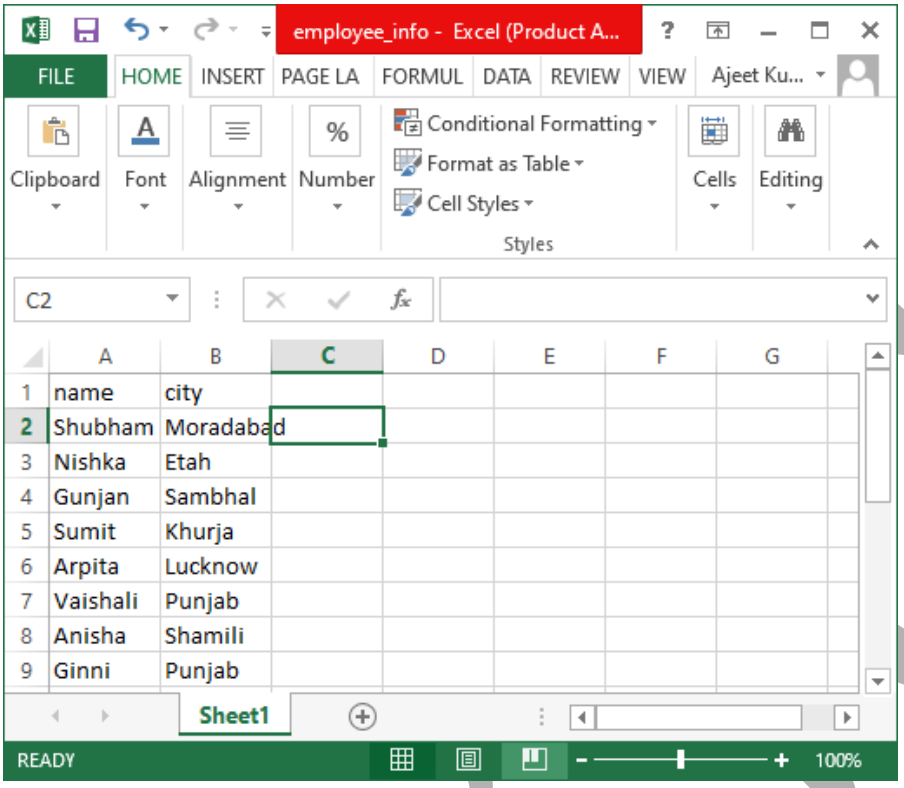
Creating an xlsx File

Once the xlsx package is loaded into our system, we will create an excel file with the following data and named it employee.



	A	B	C	D	E	F	G
1	id	name	salary	date	dept		
2	1	Shubham	623	1/1/2012	IT		
3	2	Nishka	552	9/23/2013	Operations		
4	3	Gunjan	669	11/15/2014	IT		
5	4	Sumit	825	5/11/2014	HR		
6	5	Arpita	762	3/27/2015	Finance		
7	6	Vaishali	882	5/21/2013	IT		
8	7	Anisha	783	7/30/2013	Operations		
9	8	Ginni	964	6/17/2014	Finance		

Apart from this, we will create another table with the following data and give it a name as employee_info.



Reading the Excel File

Like the CSV file, we can read data from an excel file. R provides read.xlsx() function, which takes two arguments as input, i.e., file name and index of the sheet. This function returns the excel data in the form of a data frame in the R environment. There is the following syntax of read.xlsx() function:

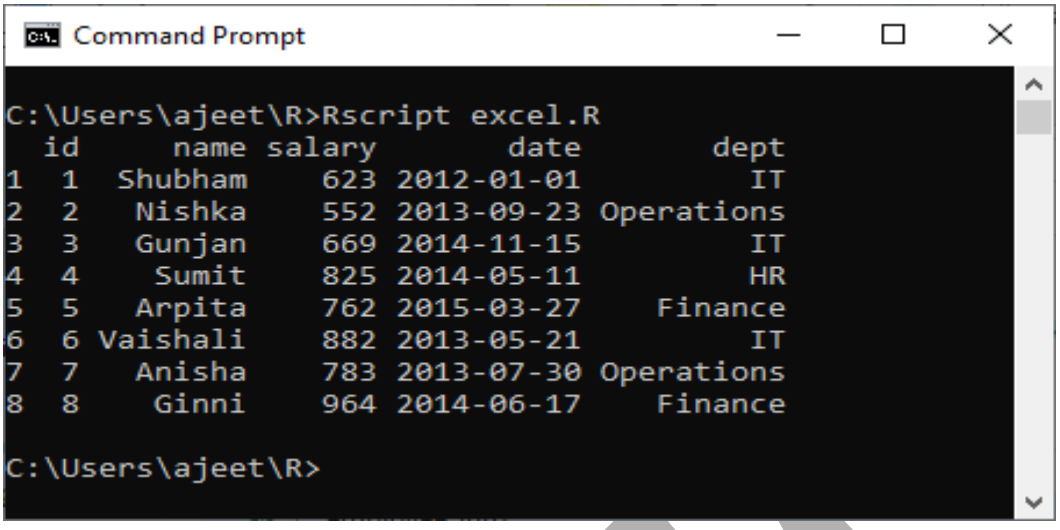
read.xlsx(file_name,sheet_index)

Let's see an example in which we read data from our employee.xlsx file.

Example

```
#Loading xlsx package
library("xlsx")
# Reading the first worksheet in the file employee.xlsx.
excel_data<- read.xlsx("employee.xlsx", sheetIndex = 1)
print(excel_data)
```

Output



9.Working with R Charts and Graphs:

9.1 Histograms

A histogram is a type of bar chart which shows the frequency of the number of values which are compared with a set of values ranges. The histogram is used for the distribution, whereas a bar chart is used for comparing different entities. In the histogram, each bar represents the height of the number of values present in the given range.

For creating a histogram, R provides hist() function, which takes a vector as an input and uses more parameters to add more functionality. There is the following syntax of hist() function:

hist(v,main,xlab,ylab,xlim,ylim,breaks,col,border)

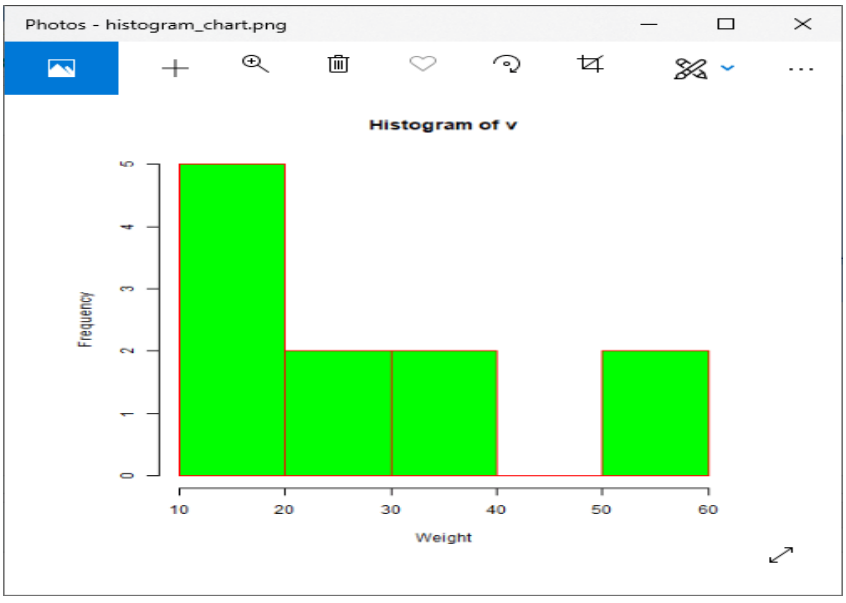
S.No	Parameter	Description
1.	v	It is a vector that contains numeric values.
2.	main	It indicates the title of the chart.
3.	col	It is used to set the color of the bars.
4.	border	It is used to set the border color of each bar.

5.	xlab	It is used to describe the x-axis.
6.	ylab	It is used to describe the y-axis.
7.	xlim	It is used to specify the range of values on the x-axis.
8.	ylim	It is used to specify the range of values on the y-axis.
9.	breaks	It is used to mention the width of each bar.

Example

```
1. # Creating data for the graph.
2. v <-  c(12,24,16,38,21,13,55,17,39,10,60)
3. # Giving a name to the chart file.
4. png(file = "histogram_chart.png")
5. # Creating the histogram.
6. hist(v,xlab = "Weight",ylab="Frequency",col = "green",border = "red")
7. # Saving the file.
8. dev.off()
```

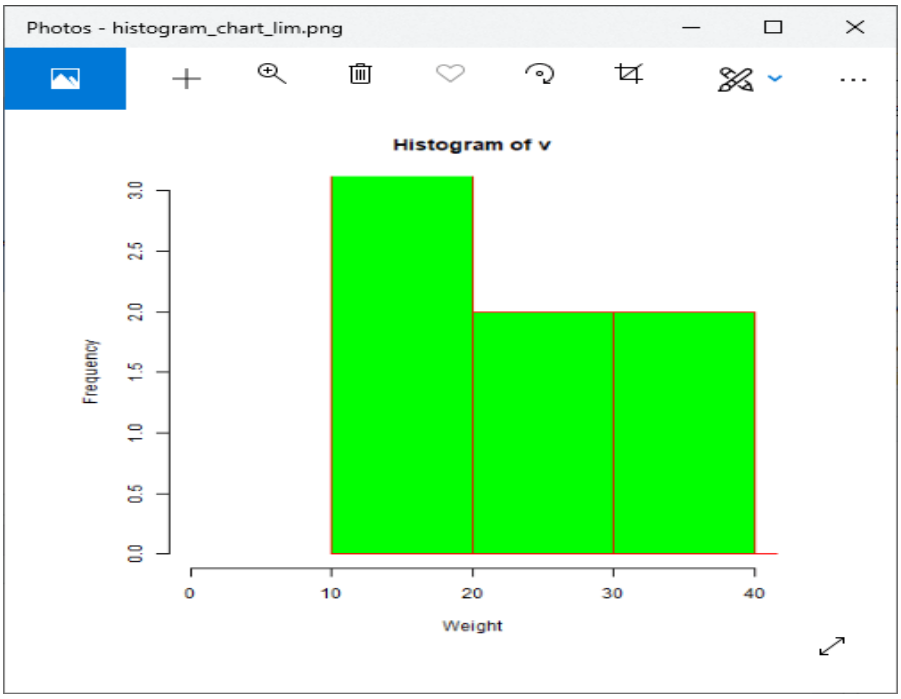
Output:



Example: Use of xlim & ylim parameter

```
1. # Creating data for the graph.
2. v <-  c(12,24,16,38,21,13,55,17,39,10,60)
3. # Giving a name to the chart file.
4. png(file = "histogram_chart_lim.png")
5. # Creating the histogram.
6. hist(v,xlab = "Weight",ylab="Frequency",col = "green",border = "red",xlim = c(0,40),
   ylim = c(0,3), breaks = 5)
7. # Saving the file.
8. dev.off()
```

Output:



R Bar Charts

A bar chart is a pictorial representation in which numerical values of variables are represented by length or height of lines or rectangles of equal width. A bar chart is used for summarizing a set of categorical data. In bar chart, the data is shown through rectangular bars having the length of the bar proportional to the value of the variable.

In R, we can create a bar chart to visualize the data in an efficient manner.

For this purpose, R provides the barplot() function, which has the following syntax:

barplot(h,x,y,main, names.arg,col)

S.No	Parameter	Description
1.	H	A vector or matrix which contains numeric values used in the bar chart.
2.	xlab	A label for the x-axis.
3.	ylab	A label for the y-axis.
4.	main	A title of the bar chart.
5.	names.arg	A vector of names that appear under each bar.
6.	col	It is used to give colors to the bars in the graph.

Example

```
# Creating the data for Bar chart
H<- c(12,35,54,3,41)
# Giving the chart file a name
png(file = "bar_chart.png")
# Plotting the bar chart
barplot(H)
# Saving the file
dev.off()
```

Labels, Title & Colors

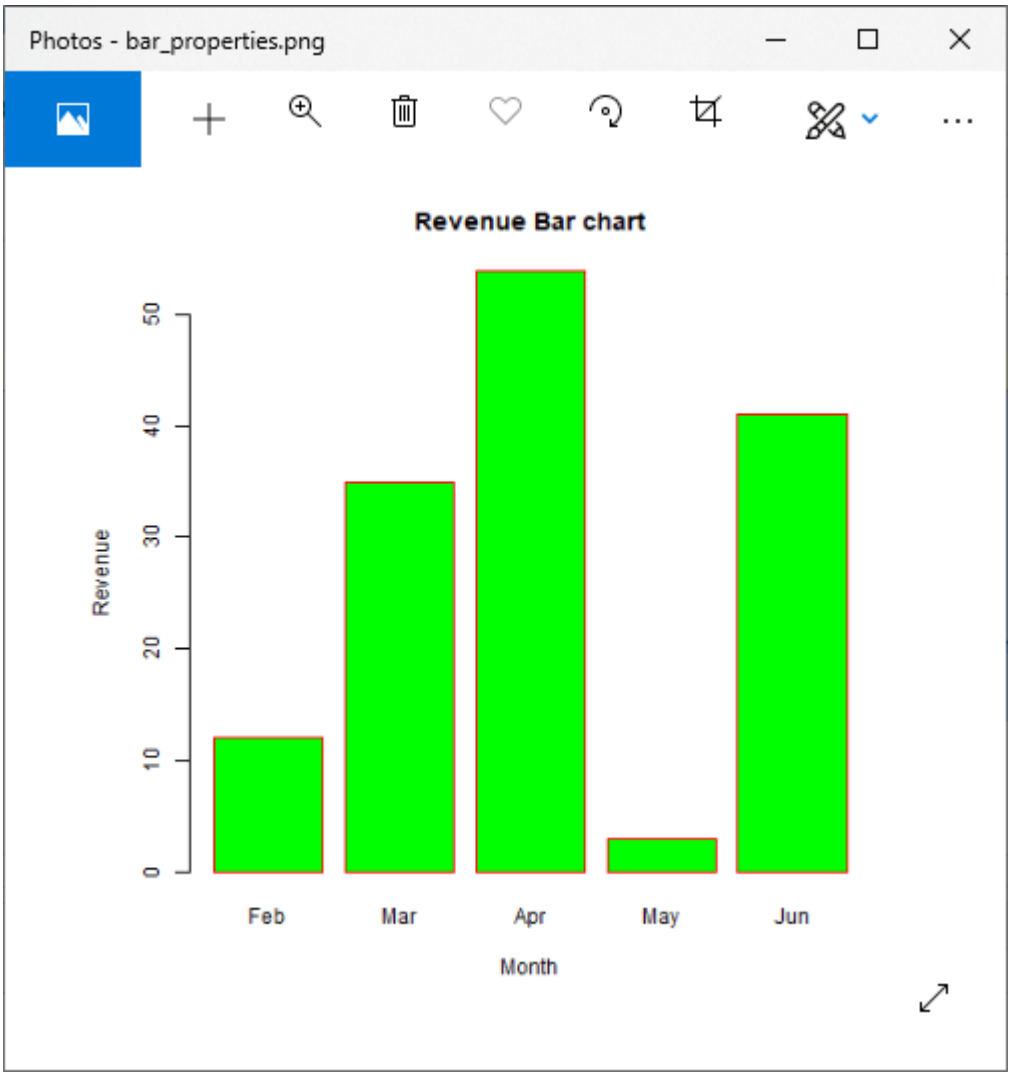
Like pie charts, we can also add more functionalities in the bar chart by-passing more arguments in the barplot() functions. We can add a title in our bar chart or can add colors to the bar by adding the main and col parameters, respectively. We can add another parameter i.e., args.name, which is a vector that has the same number of values, which are fed as the input vector to describe the meaning of each bar.

et's see an example to understand how labels, titles, and colors are added in our bar chart.

Example

```
# Creating the data for Bar chart
H <- c(12,35,54,3,41)
M<- c("Feb","Mar","Apr","May","Jun")
# Giving the chart file a name
png(file = "bar_properties.png")
# Plotting the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="Green",
 main="Revenue Bar chart",border="red")
# Saving the file
dev.off()
```

Output:



R Boxplot

Boxplots are a measure of how well data is distributed across a data set. This divides the data set into three quartiles. This graph represents the minimum, maximum, average, first quartile, and the third quartile in the data set. Boxplot is also useful in comparing the distribution of data in a data set by drawing a boxplot for each of them.

R provides a boxplot() function to create a boxplot. There is the following syntax of boxplot() function:

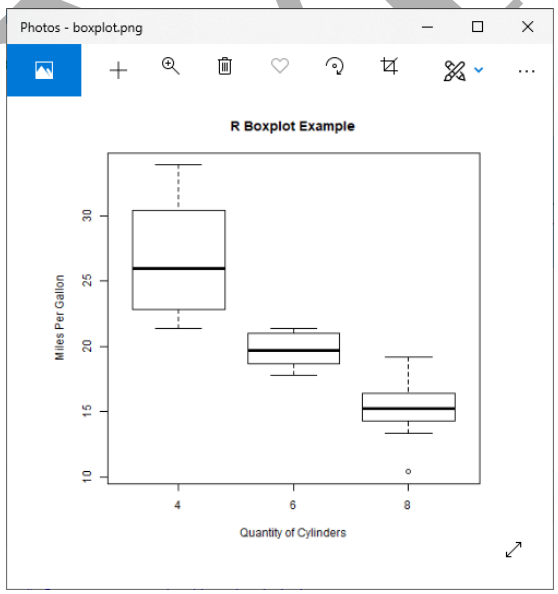
boxplot(x, data, notch, varwidth, names, main)

S.No	Parameter	Description
1.	x	It is a vector or a formula.
2.	data	It is the data frame.
3.	notch	It is a logical value set as true to draw a notch.
4.	varwidth	It is also a logical value set as true to draw the width of the box same as the sample size.
5.	names	It is the group of labels that will be printed under each boxplot.
6.	main	It is used to give a title to the graph.

Example

```
1. # Giving a name to the chart file.
2. png(file = "boxplot.png")
3. # Plotting the chart.
4. boxplot(mpg ~ cyl, data = mtcars, xlab = "Quantity of Cylinders",
5.         ylab = "Miles Per Gallon", main = "R Boxplot Example")
6. # Save the file.
7. dev.off()
```

Output:



R Line Graphs

A line graph is a pictorial representation of information which changes continuously over time. A line graph can also be referred to as a line chart. Within a line graph, there are points connecting the data to show the continuous change. The lines in a line graph can move up and down based on the data. We can use a line graph to compare different events, information, and situations.

A line chart is used to connect a series of points by drawing line segments between them. Line charts are used in identifying the trends in data. For line graph construction, R provides plot() function, which has the following syntax:

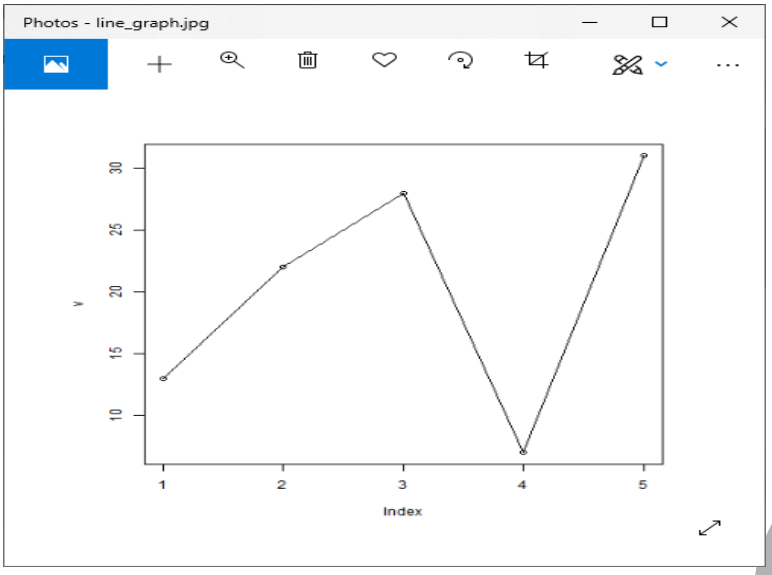
plot(v,type,col,xlab,ylab)

S.No	Parameter	Description
1.	v	It is a vector which contains the numeric values.
2.	type	This parameter takes the value ?l? to draw only the lines or ?p? to draw only the points and "o" to draw both lines and points.
3.	xlab	It is the label for the x-axis.
4.	ylab	It is the label for the y-axis.
5.	main	It is the title of the chart.
6.	col	It is used to give the color for both the points and lines

Example

```
1. # Creating the data for the chart.
2. v <- c(13,22,28,7,31)
3. # Giving a name to the chart file.
4. png(file = "line_graph.jpg")
5. # Plotting the bar chart.
6. plot(v,type = "o")
7. # Saving the file.
8. dev.off()
```

Output:



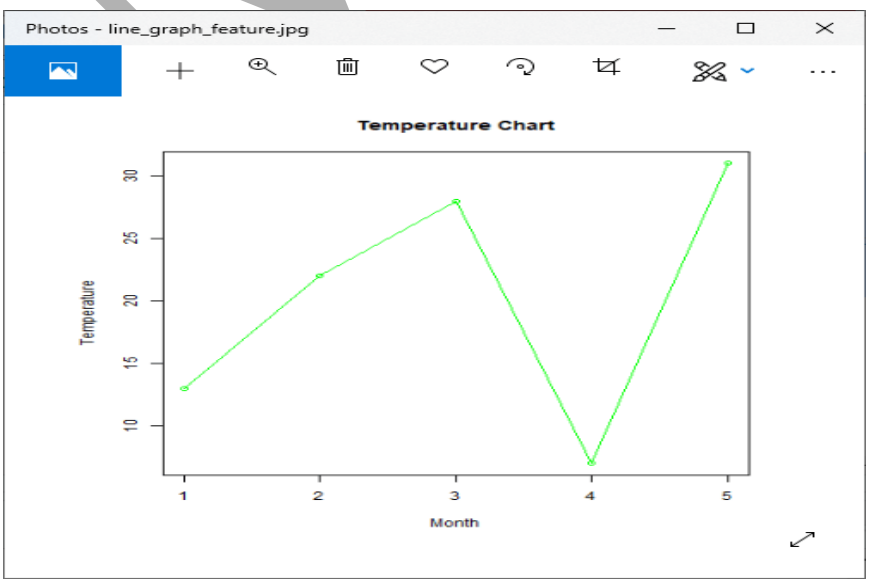
Line Chart Title, Color, and Labels

Like other graphs and charts, in line chart, we can add more features by adding more parameters. We can add the colors to the lines and points, add labels to the axis, and can give a title to the chart. Let's see an example to understand how these parameters are used in plot() function to create an attractive line graph.

Example

```
# Creating the data for the chart.
v <- c(13,22,28,7,31)
# Giving a name to the chart file.
png(file = "line_graph_feature.jpg")
# Plotting the bar chart.
plot(v,type = "o",col="green",xlab="Month",ylab="Temperature")
# Saving the file.
dev.off()
```

Output:



R Scatterplots

The scatter plots are used to compare variables. A comparison between variables is required when we need to define how much one variable is affected by another variable. In a scatterplot, the data is represented as a collection of points. Each point on the scatterplot defines the values of the two variables. One variable is selected for the vertical axis and other for the horizontal axis. In R, there are two ways of creating scatterplot, i.e., using plot() function and using the ggplot2 package's functions.

There is the following syntax for creating scatterplot in R:

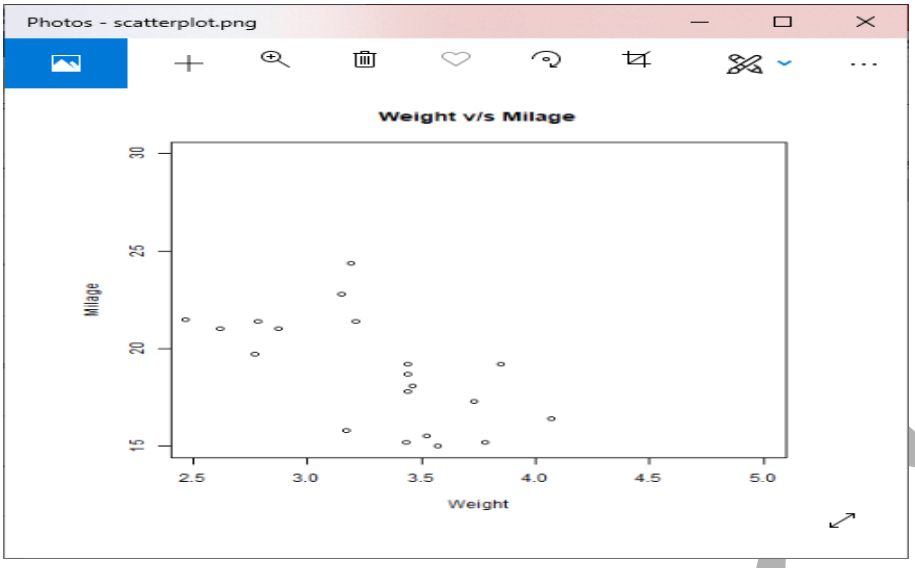
plot(x, y, main, xlab, ylab, xlim, ylim, axes)

S.No	Parameters	Description
1.	x	It is the dataset whose values are the horizontal coordinates.
2.	y	It is the dataset whose values are the vertical coordinates.
3.	main	It is the title of the graph.
4.	xlab	It is the label on the horizontal axis.
5.	ylab	It is the label on the vertical axis.
6.	xlim	It is the limits of the x values which is used for plotting.
7.	ylim	It is the limits of the values of y, which is used for plotting.
8.	axes	It indicates whether both axes should be drawn on the plot.

Example

```
1. #Fetching two columns from mtcars
2. data <-mtcars[,c('wt','mpg')]
3. # Giving a name to the chart file.
4. png(file = "scatterplot.png")
5. # Plotting the chart for cars with weight between 2.5 to 5 and mileage between 15 and 30.
6. plot(x = data$wt,y = data$mpg, xlab = "Weight", ylab = "Milage", xlim = c(2.5,5), ylim = c(15,30), main = "Weight v/sMilage")
7. # Saving the file.
8. dev.off()
```

Output:



Scatterplot using ggplot2

In R, there is another way for creating scatterplot i.e. with the help of ggplot2 package.

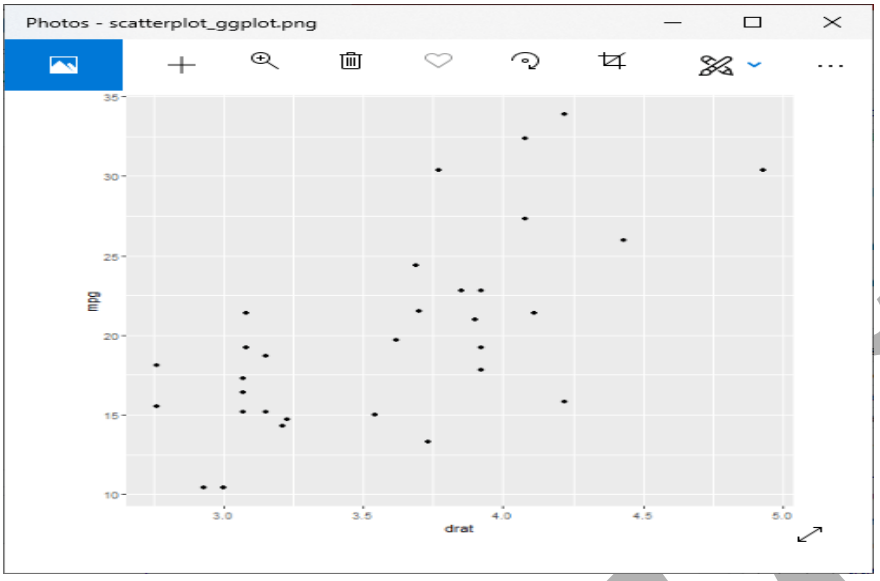
The ggplot2 package provides ggplot() and geom_point() function for creating a scatterplot. The ggplot() function takes a series of the input item. The first parameter is an input vector, and the second is the aes() function in which we add the x-axis and y-axis.

Let's start understanding how the ggplot2 package is used with the help of an example where we have used the familiar dataset "mtcars".

Example

```
#Loading ggplot2 package
library(ggplot2)
# Giving a name to the chart file.
png(file = "scatterplot_ggplot.png")
# Plotting the chart using ggplot() and geom_point() functions.
ggplot(mtcars, aes(x = drat, y = mpg)) +geom_point()
# Saving the file.
dev.off()
```

Output:



R Pie Charts

R programming language has several libraries for creating charts and graphs. A pie-chart is a representation of values in the form of slices of a circle with different colors. Slices are labeled with a description, and the numbers corresponding to each slice are also shown in the chart. However, pie charts are not recommended in the R documentation, and their characteristics are limited. The authors recommend a bar or dot plot on a pie chart because people are able to measure length more accurately than volume.

The Pie charts are created with the help of pie () function, which takes positive numbers as vector input. Additional parameters are used to control labels, colors, titles, etc.

There is the following syntax of the pie() function:

pie(X, Labels, Radius, Main, Col, Clockwise)

Here,

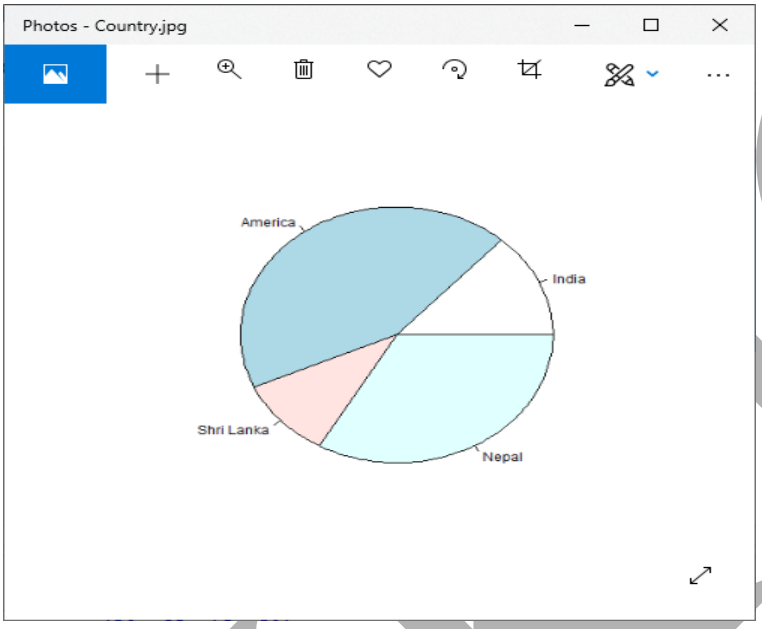
1. **X** is a vector that contains the numeric values used in the pie chart.
2. **Labels** are used to give the description to the slices.
3. **Radius** describes the radius of the pie chart.
4. **Main** describes the title of the chart.
5. **Col** defines the color palette.
6. **Clockwise** is a logical value that indicates the clockwise or anti-clockwise direction in which slices are drawn.

Example

```
1. # Creating data for the graph.
2. x <- c(20, 65, 15, 50)
```

```
labels <- c("India", "America", "Shri Lanka", "Nepal")
# Giving the chart file a name.
png(file = "Country.jpg")
# Plotting the chart.
pie(x,labels)
# Saving the file.
dev.off()
```

Output:



Title and color

A pie chart has several more features that we can use by adding more parameters to the pie() function. We can give a title to our pie chart by passing the main parameter. It tells the title of the pie chart to the pie() function. Apart from this, we can use a rainbow colour pallet while drawing the chart by passing the col parameter.

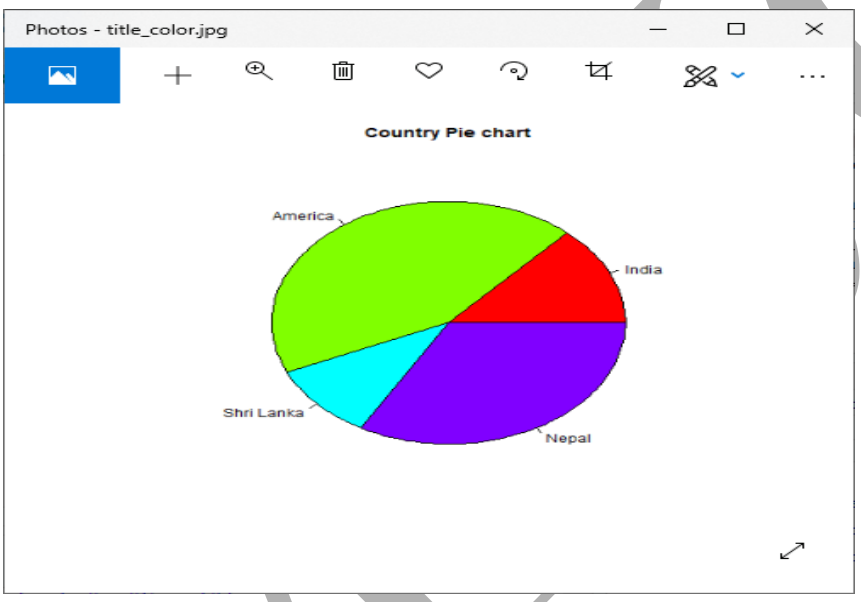
Note: The length of the pallet will be the same as the number of values that we have for the chart. So for that, we will use length() function.

Let's see an example to understand how these methods work in creating an attractive pie chart with title and color.

Example

```
1. # Creating data for the graph.
2. x <- c(20, 65, 15, 50)
3. labels <- c("India", "America", "Shri Lanka", "Nepal")
4. # Giving the chart file a name.
5. png(file = "title_color.jpg")
6. # Plotting the chart.
7. pie(x,labels,main="Country Pie chart",col=rainbow(length(x)))
8. # Saving the file.
9. dev.off()
```

Output:



Additional Links

1. https://egyankosh.ac.in/bitstream/123456789/87562/1/Unit-14.pdf
2. https://www.javatpoint.com/r-database