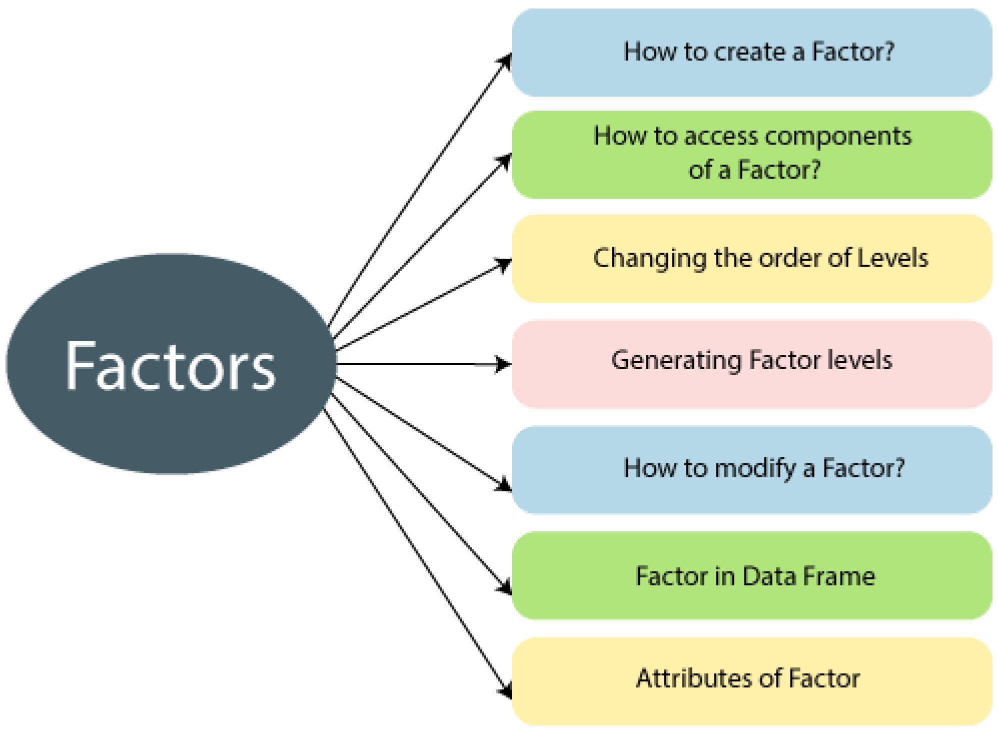
R factors

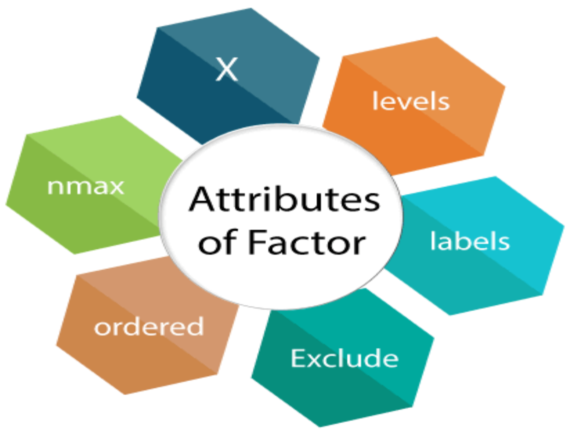
The factor is a data structure which is used for fields which take only predefined finite number of values. These are the variable which takes a limited number of different values. These are the data objects which are used to categorize the data and to store it on multiple levels. It can store both integers and strings values, and are useful in the column that has a limited number of unique values.



Factors have labels which are associated with the unique integers stored in it. It contains predefined set value known as levels and by default R always sorts levels in alphabetical order.

Attributes of a factor

There are the following attributes of a factor in R



How to create a factor?

In R, it is quite simple to create a factor. A factor is created in two steps

1. In the first step, we create a vector.
2. Next step is to convert the vector into a factor,

R provides factor() function to convert the vector into factor. There is the following syntax of factor() function

1. factor_data<- factor(vector)

Example

```
#Creating a vector as input.
Data<c("Shubham","Nishka","Arpita","Nishka","Shubham","Sumit","Nishka","Shubham","Sumit",
"Arpita","Sumit")
print(data)
print(is.factor(data))
# Applying the factor function.
factor_data<- factor(data)
print(factor_data)
print(is.factor(factor_data))
```

Accessing components of factor

Like vectors, we can access the components of factors. The process of accessing components of factor is much more similar to the vectors. We can access the element with the help of the indexing method or using logical vectors. Let's see an example in which we understand the different-different ways of accessing the components.

```
# Creating a vector as input.
data <-
 c("Shubham","Nishka","Arpita","Nishka","Shubham","Sumit","Nishka","Shubham","Sumit","Arpi
ta","Sumit")

# Applying the factor function.
factor_data<- factor(data)
#Printing all elements of factor
print(factor_data)

#Accessing 4th element of factor
print(factor_data[4])
```

```
#Accessing 5th and 7th element
print(factor_data[c(5,7)])
#Accessing all elemcent except 4th one
print(factor_data[-4])

#Accessing elements using logical vector
print(factor_data[c(TRUE,FALSE,FALSE,FALSE,TRUE,TRUE,TRUE,FALSE,FALSE,FALSE,TRUE)])
```

Modification of factor

Like data frames, R allows us to modify the factor. We can modify the value of a factor by simply re-assigning it. In R, we cannot choose values outside of its predefined levels means we cannot insert value if it's level is not present on it. For this purpose, we have to create a level of that value, and then we can add it to our factor.

Example

```
# Creating a vector as input.
data <- c("Shubham","Nishka","Arpita","Nishka","Shubham")

# Applying the factor function.
factor_data<- factor(data)
#Printing all elements of factor
print(factor_data)

#Change 4th element of factor with sumit
factor_data[4] <-"Arpita"
print(factor_data)

#change 4th element of factor with "Gunjan"
factor_data[4] <- "Gunjan"    # cannot assign values outside levels
print(factor_data)

#Adding the value to the level
levels(factor_data) <- c(levels(factor_data),"Gunjan")#Adding new level
factor_data[4] <- "Gunjan"
```

```
print(factor_data)
```

Factor in Data Frame

When we create a frame with a column of text data, R treats this text column as categorical data and creates factor on it.

Example

```
# Creating the vectors for data frame.
height <- c(132,162,152,166,139,147,122)
weight <- c(40,49,48,40,67,52,53)
gender <- c("male","male","female","female","male","female","male")

# Creating the data frame
input_data<- data.frame(height,weight,gender)
print(input_data)

# Testing if the gender column is a factor
print(is.factor(input_data$gender))

# Printing the gender column to see the levels.
print(input_data$gender
```

Changing order of the levels

In R, we can change the order of the levels in the factor with the help of the factor function.

Example

```
data <- c("Nishka","Gunjan","Shubham","Arpita","Arpita","Sumit","Gunjan","Shubham")
# Creaing the factors
factor_data<- factor(data)
print(factor_data)

# Apply the factor function with the required order of the level.
new_order_factor<- factor(factor_data,levels = c("Gunjan","Nishka","Arpita","Shubham","S
umit"))
print(new_order_factor)
```

Generating Factor Levels

R provides gl() function to generate factor levels. This function takes three arguments i.e., n, k, and labels. Here, n and k are the integers which indicate how many levels we want and how many times each level is required.

There is the following syntax of gl() function which is as follows

1. gl(n, k, labels)

1. n indicates the number of levels.
2. k indicates the number of replications.
3. labels is a vector of labels for the resulting factor levels.

Example

1. gen_factor<- gl(3,5,labels=c("BCA","MCA","B.Tech"))
2. gen_factor

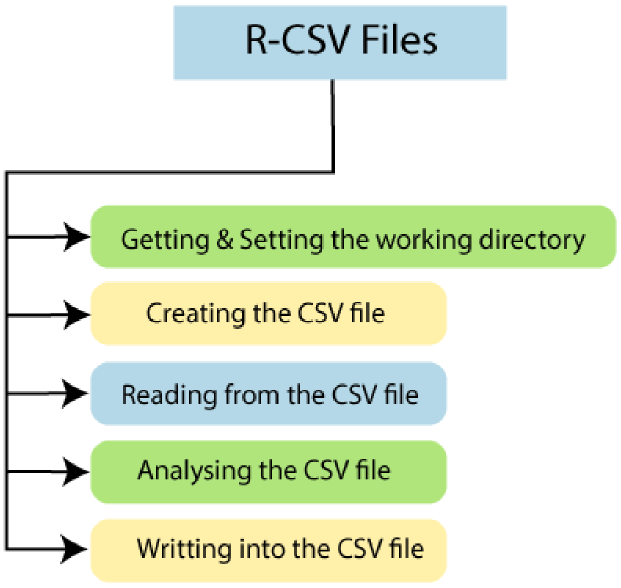
R CSV Files

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.

R allows us to read data from files which are stored outside the R environment. Let's start understanding how we can read and write data into CSV files. The file should be present in the current working directory so that R can read it. We can also set our directory and read file from there.


R-CSV Files
Getting & Setting the working directory
Creating the CSV file
Reading from the CSV file
Analysing the CSV file
Writting into the CSV file

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())
```

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,Financ
```

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.

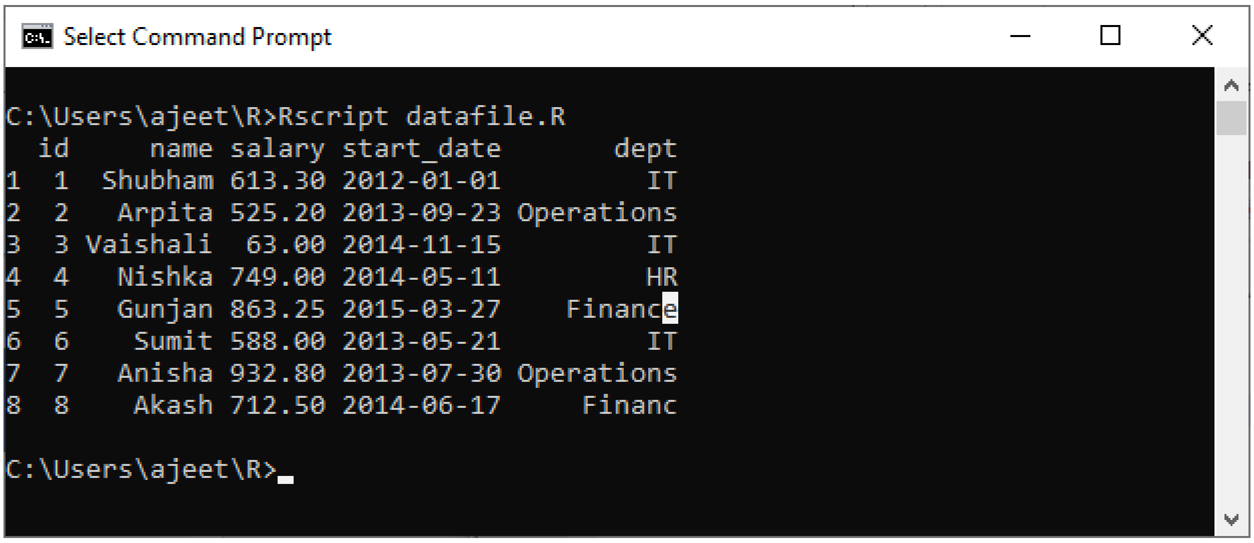
Let's use our record.csv file to read records from it using read.csv() function.

Example

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

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

Output



```
C:\Users\ajeet\R>Rscript datafile.R
  id      name salary start_date       dept
1  1   Shubham 613.30 2012-01-01         IT
2  2    Arpita 525.20 2013-09-23 Operations
3  3 Vaishali   63.00 2014-11-15         IT
4  4    Nishka 749.00 2014-05-11         HR
5  5    Gunjan 863.25 2015-03-27    Finance
6  6     Sumit 588.00 2013-05-21         IT
7  7    Anisha 932.80 2013-07-30 Operations
8  8     Akash 712.50 2014-06-17     Financ

C:\Users\ajeet\R>
```

Analyzing the CSV File

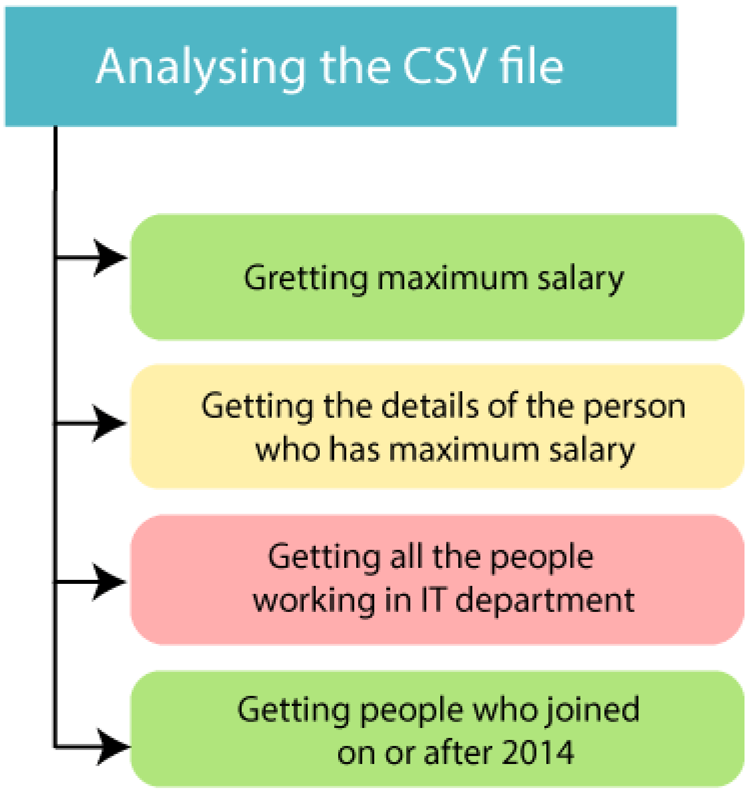
When we read data from the .csv file using **read.csv()** function, by default, it gives the output as a data frame. Before analyzing data, let's start checking the form of our output with the help of **is.data.frame()** function. After that, we will check the number of rows and number of columns with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

Example

```
csv_data<- read.csv("record.csv")
```

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

it is clear that our data is read in the form of the data frame. So we can apply all the functions of the data frame, which we have discussed in the earlier sections.


Analysing the CSV file
Gretting maximum salary
Getting the details of the person who has maximum salary
Getting all the people working in IT department
Getting people who joined on or after 2014

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)
```

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

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

#Getting the detais of all the pweson who are working in IT department
details <- subset(csv_data,dept=="IT")
print(details)
```

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

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

#Getting the detais of all the pweson who are working in IT department
details <- subset(csv_data,dept=="IT"&salary>600)
print(details)
```

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

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

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.

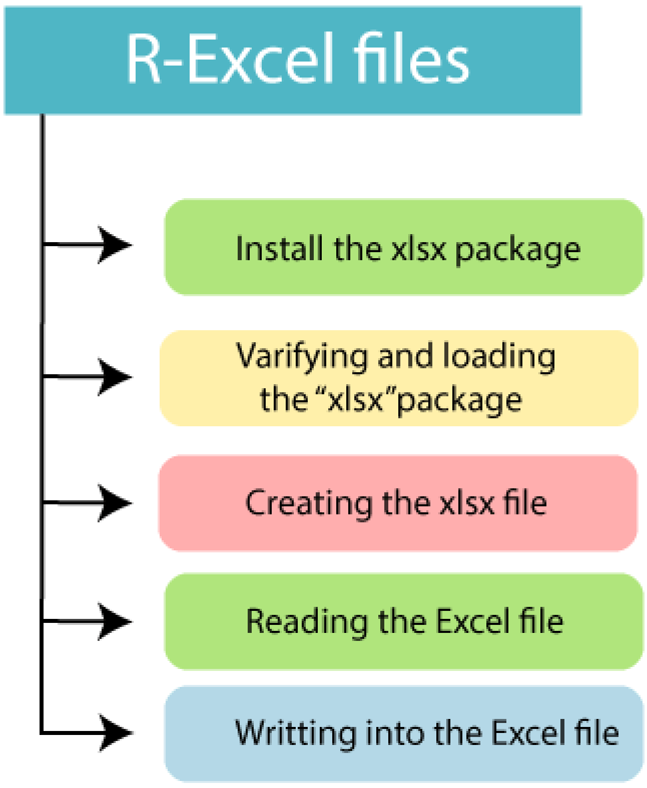
Let's see an example to understand how **write.csv()** function is used to create an output CSV file.

Example

```
csv_data<- read.csv("record.csv")

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

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:

1. install.packages("package name")

Example :

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

Verifying and Loading of "xlsx" Package

In R, grepl() and any() functions are used to verify the package. If the packages are installed, these functions will return True else return False. For verifying the package, both the functions are used together.

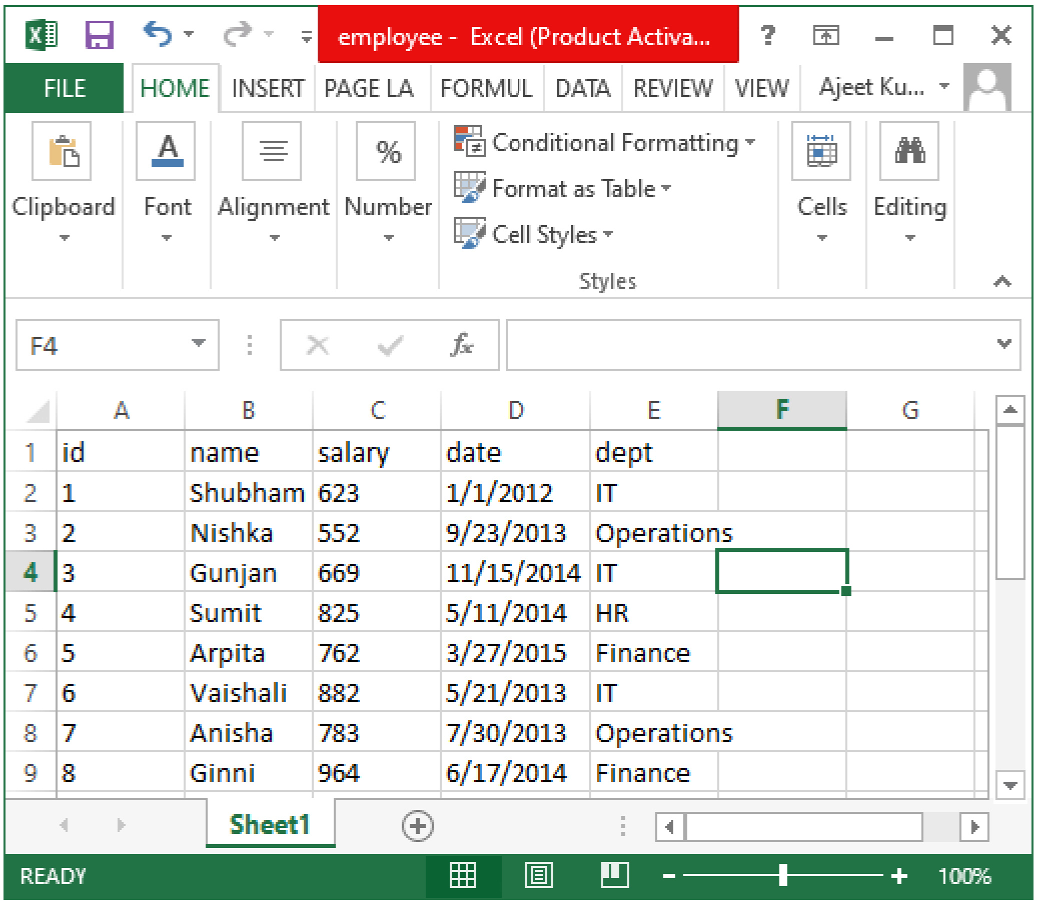
For loading purposes, we use the library() function with the appropriate package name. This function loads all the additional packages also.

Example

```
#Installing xlsx package
install.packages("xlsx")

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

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

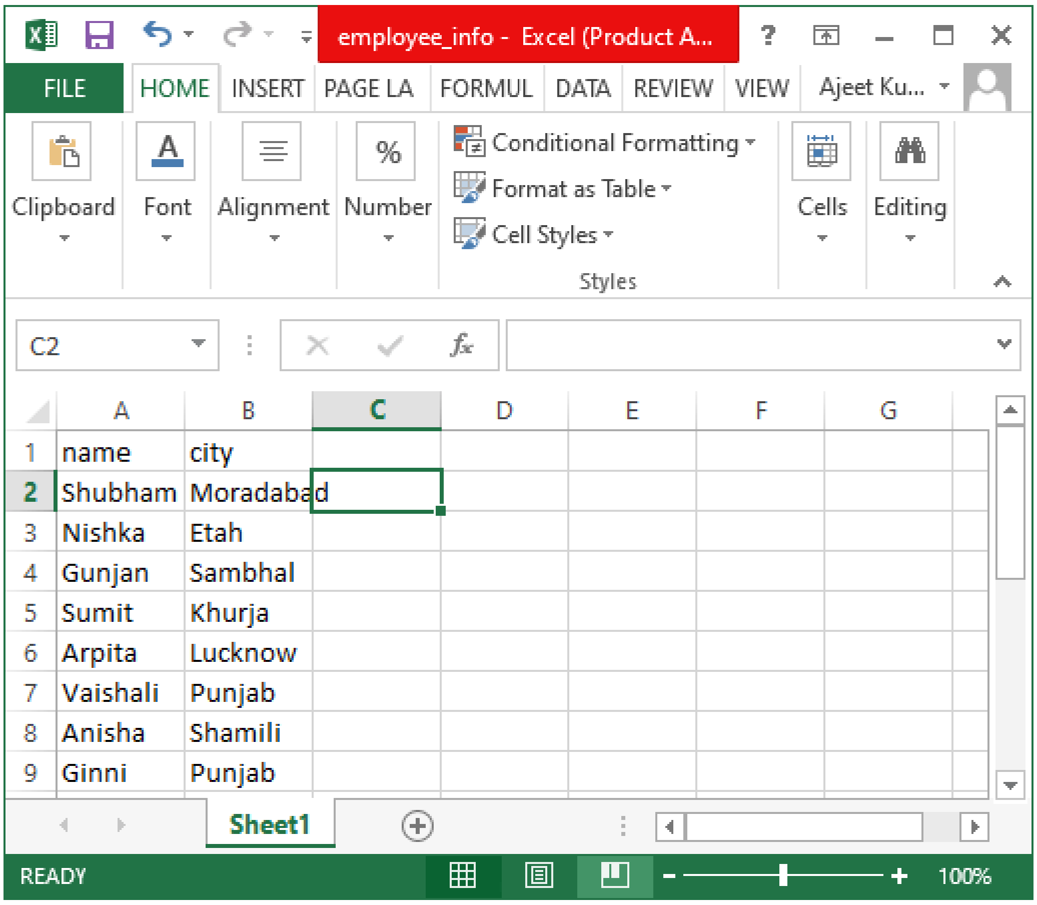
Creating an xlsx File

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.



	A	B	C	D	E	F	G
1	name	city					
2	Shubham	Moradabad					
3	Nishka	Etah					
4	Gunjan	Sambhal					
5	Sumit	Khurja					
6	Arpita	Lucknow					
7	Vaishali	Punjab					
8	Anisha	Shamili					
9	Ginni	Punjab					

Note: Both the files will be saved in the current working directory of the R workspace.

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:

1. 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)
```

Writing data into Excel File

In R, we can also write the data into our .xlsx file. R provides a write.xlsx() function to write data into the excel file. There is the following syntax of write.xlsx() function:

1. write.xlsx(data_frame,file_name,col.names,row.names,sheetnames,append)

Here,

- The data_frame is our data, which we want to insert into our excel file.
- The file_names is the name of that file in which we want to insert our data.
- The col.names and row.names are the logical values that are specifying whether the column names/row names of the data frame are to be written to the file.
- The append is a logical value, which indicates our data should be appended or not into an existing file.

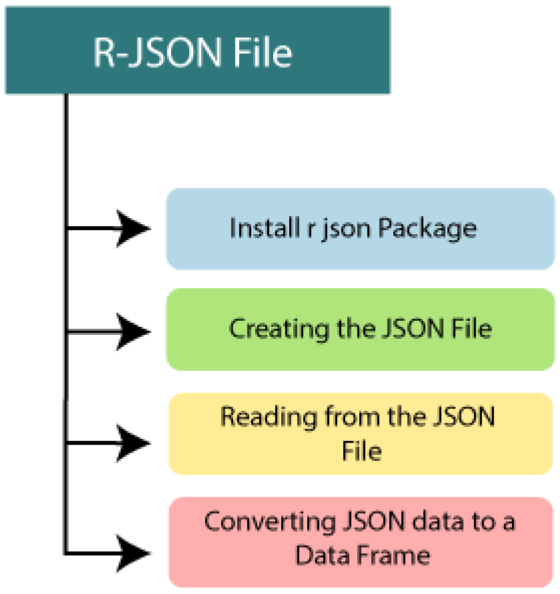
Let's see an example to understand how write.xlsx() function works with its parameters.

Example

```
#Loading xlsx package
library("xlsx")
#Creating data frame
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"),
)

# Writing the first data set in employee.xlsxRscript
write.xlsx(emp.data, file = "employee.xlsx", col.names=TRUE, row.names=TRUE,sheetName="Sh
eet2",append = TRUE)

# Reading the first worksheet in the file employee.xlsx.
excel_data<- read.xlsx("employee.xlsx", sheetIndex = 1)
print(excel_data)
# Reading the first worksheet in the file employee.xlsx.
excel_data<- read.xlsx("employee.xlsx", sheetIndex = 2)
print(excel_data)
```

R 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**


R-JSON File
Install r json Package
Creating the JSON File
Reading from the JSON File
Converting JSON data to a Data Frame

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"]
}
```

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)
```

Converting JSON data to a Data Frame

R provide, as.data.frame() function to convert the extracted data into data frame. For further analysis, data analysts use this function. Let's start an example to see how this function is used, and in our example, we will consider our employee_info.json file.

Example

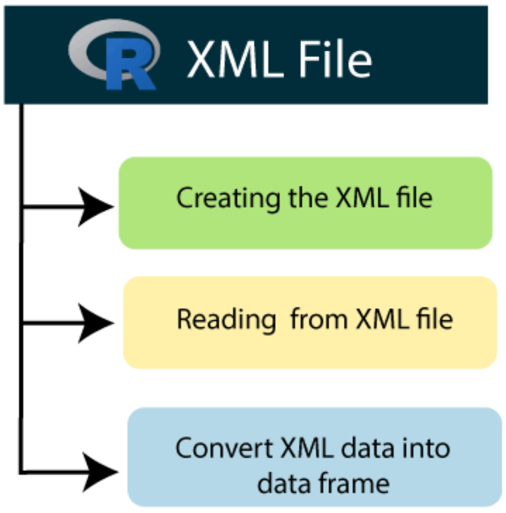
```
# Loading the package which is required to read JSON files.
library("rjson")
```

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

# Converting the JSON record to a data frame.
data_frame <- as.data.frame(result)

#Printing JSON data frame
print(data_frame)
```

R 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.

```
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.

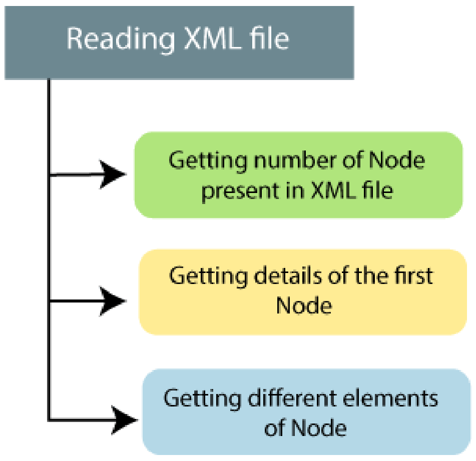
```
<records>
<employee_info>
<id>1</id>
<name>Shubham</name>
<salary>623</salary>
<date>1/1/2012</date>
```

```xml
<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.
Let's see an example to understand the working of xmlParse() function in which we read our xml_data.xml file.


Reading XML file
Getting number of Node present in XML file
Getting details of the first Node
Getting different elements of Node

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)
```

Example: Getting number of nodes present in xml file.

```
# 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")

#Converting the data into list
xml_data <- xmlToList(result)

#Printing the data
print(xml_data)

# Exracting the root node form the xml file.
root_node <- xmlRoot(result)

# Finding the number of nodes in the root.
root_size <- xmlSize(root_node)

# Printing the result.
print(root_size)
```

Example: Getting details of the first node in xml.

```
# 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")

# Exracting the root node form the xml file.
root_node <- xmlRoot(result)

# Printing the result.
print(root_node[1])
```

Example: Getting details of different elements of a node.

```
# 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")

# Exracting the root node form the xml file.
root_node <- xmlRoot(result)

# Getting the first element of the first node.
print(root_node[[1]][[1]])

# Getting the fourth element of the first node.
print(root_node[[1]][[4]])

# Getting the third element of the third node.
print(root_node[[3]][[3]])
```

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

```
# 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 xmlToDataFrame.
data_frame <- xmlToDataFrame("xml_data.xml")

#Printing the result
print(data_frame)
```

Data Reshaping in R

In R, Data Reshaping is about changing how the data is organized into rows and columns. In R, data processing is done by taking the input as a data frame. It is much easier to extract data from the rows and columns of a data frame, but there is a problem when we need a data frame in a format which is different from the format in which we received it. R provides many functions to merge, split, and change the rows to columns and vice-versa in a data frame.

Transpose a Matrix

R allows us to calculate the transpose of a matrix or a data frame by providing t() function. This t() function takes the matrix or data frame as an input and return the transpose of the input matrix or data frame. The syntax of t() function is as follows:

```
t(Matrix/data frame)
```

Example

```
a <- matrix(c(4:12),nrow=3,byrow=TRUE)

a

print("Matrix after transpose\n")

b <- t(a)

b
```

Joining rows and columns in Data Frame

R allows us to join multiple vectors to create a data frame. For this purpose R provides cbind() function. R also provides rbind() function, which allows us to merge two data frame. In some situation, we need to merge data frames to access the information which depends on both the data frame. There is the following syntax of cbind() function and rbind() function.

```
cbind(vector1, vector2,.......vectorN)

rbind(dataframe1, dataframe2,........dataframeN)
```

Example:

```
#Creating vector objects
Name <- c("Shubham Rastogi","Nishka Jain","Gunjan Garg","Sumit Chaudhary")
Address <- c("Moradabad","Etah","Sambhal","Khurja")
Marks <- c(255,355,455,655)

#Combining vectors into one data frame
info <- cbind(Name,Address,Marks)
```

```
#Printing data frame
print(info)

# Creating another data frame with similar columns
new.stuinfo <- data.frame(
   Name = c("Deepmala","Arun"),
   Address = c("Khurja","Moradabad"),
   Marks = c("755","855"),
   stringsAsFactors=FALSE
)

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

#Printing the data frame.
print(new.stuinfo)

# Combining rows form both the data frames.
all.info <- rbind(info,new.stuinfo)

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

# Printing the result.
print(all.info)
```

Merging Data Frame

R provides the merge() function to merge two data frames. In the merging process, there is a constraint i.e.; data frames must have the same column names.

Let's take an example in which we take the dataset about Diabetes in Pima Indian Women which is present in the "MASS" library. We will merge two datasets on the basis of the value of the blood pressure and body mass index. When selecting these two columns for merging, the records where values of these two variables match in both data sets are combined together to form a single data frame.

Example

MASS- Modern Applied Statistics with S

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

nrow(merging_pima)

R - Pie Charts

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

In R the pie chart is created using the pie() function which takes positive numbers as a vector input. The additional parameters are used to control labels, color, title etc.

Syntax

The basic syntax for creating a pie-chart using the R is −

pie(x, labels, radius, main, col, clockwise)

Following is the description of the parameters used −

- x is a vector containing the numeric values used in the pie chart.
- labels is used to give description to the slices.
- radius indicates the radius of the circle of the pie chart.(value between −1 and +1).
- main indicates the title of the chart.
- col indicates the color palette.
- clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise

Example:Part 1

```
# Creating data for the graph.
x <- c(20, 65, 15, 50)
labels <- c("India", "America", "Shri Lanka", "Nepal")  .
# Plotting the chart.
pie(x,labels)
```

Example:Part 2

```
# Creating data for the graph.
x <- c(20, 65, 15, 50)
lb1 <- c("India", "America", "Shri Lanka", "Nepal")
# Plotting the chart.
pie(x,labels=lb1,main=”Pie Chart of Cities)
```

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

```
# Creating data for the graph.
x <- c(20, 65, 15, 50)
labels <- c("India", "America", "Shri Lanka", "Nepal")
# Plotting the chart.
pie(x,labels,main="Country Pie chart",col=rainbow(length(x)))
```

Slice Percentage & Chart Legend

There are two additional properties of the pie chart, i.e., slice percentage and chart legend. We can show the data in the form of percentage as well as we can add legends to plots in R by using the legend() function. There is the following syntax of the legend() function.
legend(x,y=NULL,legend,fill,col,bg)
Here,

- x and y are the coordinates to be used to position the legend.
- legend is the text of legend
- fill is the color to use for filling the boxes beside the legend text.
- col defines the color of line and points besides the legend text.
- bg is the background color for the legend box.

General character expansion factor for default values of main.cex, labels.cex, and values.cex. Useful for adjustment of text for larger or smaller images.

Example

```
# Creating data for the graph.
x <- c(20, 65, 15, 50)
labels <- c("India", "America", "Shri Lanka", "Nepal")
pie_percent<- round(100*x/sum(x), 1)
```

```
# Plotting the chart.
pie(x, labels = pie_percent, main = "Country Pie Chart",col = rainbow(length(x)))
legend("topright", c("India", "America", "Shri Lanka", "Nepal"), cex = 0.8,
fill = rainbow(length(x)))
```

3 Dimensional Pie Chart

In R, we can also create a three-dimensional pie chart. For this purpose, R provides a plotrix package whose pie3D() function is used to create an attractive 3D pie chart. The parameters of pie3D() function remain same as pie() function. Let's see an example to understand how a 3D pie chart is created with the help of this function.

Example

```
# Getting the library.
library(plotrix)
# Creating data for the graph.
x <- c(20, 65, 15, 50,45)
labels <- c("India", "America", "Shri Lanka", "Nepal","Bhutan")
# Plot the chart.
pie3D(x,labelslabels = labels,explode = 0.1, main = "Country Pie Chart")
```

Example

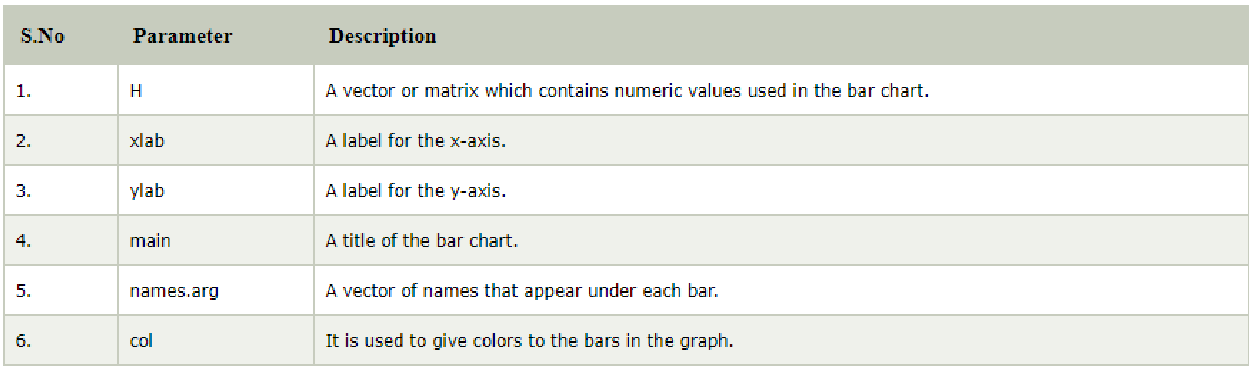
```
# Getting the library.
library(plotrix)
# Creating data for the graph.
x <- c(20, 65, 15, 50,45)
labels <- c("India", "America", "Shri Lanka", "Nepal","Bhutan")
pie_percent<- round(100*x/sum(x), 1)
# Plotting the chart.
pie3D(x, labels = pie_percent, main = "Country Pie Chart",col = rainbow(length(x)))
legend("topright", c("India", "America", "Shri Lanka", "Nepal","Bhutan"), cex = 0.8,
fill = rainbow(length(x)))
```

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)
# Plotting the bar chart
barplot(H)
```

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.

Let'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")

# Plotting the bar chart
barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="Green",
        main="Revenue Bar chart",border="red")
```

Group Bar Chart & Stacked Bar Chart

We can create bar charts with groups of bars and stacks using matrices as input values in each bar. One or more variables are represented as a matrix that is used to construct group bar charts and stacked bar charts.

Let's see an example to understand how these charts are created.

Example

```
library(RColorBrewer)
months <- c("Jan","Feb","Mar","Apr","May")
regions <- c("West","North","South")
# Creating the matrix of the values.
Values <matrix(c(21,32,33,14,95,46,67,78,39,11,22,23,94,15,16), nrow = 3, ncol = 5, byrow = TRU
E)
# Creating the bar chart
barplot(Values, main = "Total Revenue", names.arg = months, xlab = "Month", ylab = "Revenue", c
col =c("cadetblue3","deeppink2","goldenrod1"))
# Adding the legend to the chart
legend("topleft", regions, cex = 1.3, fill = c("cadetblue3","deeppink2","goldenrod1"))
```