Aim: Data Visualization using R

Theory :

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac. This programming language was named **R**, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of the Bell Labs Language **S**.

1.R - Overview :

The core of R is an interpreted computer language which allows branching and looping as well as modular programming using functions. R allows integration with the procedures written in the C, C++, .Net, Python or FORTRAN languages for efficiency.

R is freely available under the GNU General Public License, and pre-compiled binary versions are provided for various operating systems like Linux, Windows and Mac.

2.Features of R :

As stated earlier, R is a programming language and software environment for statistical analysis, graphics representation and reporting. The following are the important features of R −

- R is a well-developed, simple and effective programming language which includes conditionals, loops, user defined recursive functions and input and output facilities.
- R has an effective data handling and storage facility,
- R provides a suite of operators for calculations on arrays, lists, vectors and matrices.
- R provides a large, coherent and integrated collection of tools for data analysis.
- R provides graphical facilities for data analysis and display either directly at the computer or printing at the papers.

3.R - Environment Setup :

Ubuntu setup :

$ ***sudo apt-get install r-base***

Command to start R on terminal

$ ***R***

4.Basic Syntax :

you can start typing your program as follows −

```
> myString <- "Hello, World!"
> print ( myString)
[1] "Hello, World!"
```

R Script File :

Usually, you will do your programming by writing your programs in script files and then you execute those scripts at your command prompt with the help of R interpreter called **Rscript**.

```
# My first program in R Programming
myString <- "Hello, World!"

print ( myString)
```

Save the above code in a file test.R and execute it at Linux command prompt as given below. Even if you are using Windows or other system, syntax will remain same.

```
$ Rscript test.R
```

6.R - Data Types :

Generally, while doing programming in any programming language, you need to use various variables to store various information. Variables are nothing but reserved memory locations to store values. This means that, when you create a variable you reserve some space in memory.

You may like to store information of various data types like character, wide character, integer, floating point, double floating point, Boolean etc. Based on the data type of a variable, the operating system allocates memory and decides what can be stored in the reserved memory.

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are −

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

The simplest of these objects is the **vector object** and there are six data types of these atomic vectors, also termed as six classes of vectors. The other R-Objects are built upon the atomic vectors.

Vectors :

When you want to create vector with more than one element, you should use c() function which means to combine the elements into a vector.

```
# Create a vector.
apple <- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

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

```
[1] "red"    "green" "yellow"
[1] "character"
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.

list1 <- list(c(2,5,3),21.3,sin)

# Print the list.

print(list1)
```

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

```
[[1]]
[1] 2 5 3

[[2]]
[1] 21.3

[[3]]
function (x) .Primitive("sin")
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

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

```
  [,1] [,2] [,3]
[1,] "a"  "a"  "b"
[2,] "c"  "b"  "a"
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.

a <- array(c('green','yellow'),dim = c(3,3,2))

print(a)
```

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

```
, , 1


     [,1]     [,2]     [,3]

[1,] "green" "yellow" "green"

[2,] "yellow" "green" "yellow"

[3,] "green" "yellow" "green"


, , 2


     [,1]     [,2]     [,3]

[1,] "yellow" "green" "yellow"

[2,] "green" "yellow" "green"

[3,] "yellow" "green" "yellow"
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the **factor()** function. The **nlevels** functions gives the count of levels.

```
# Create a vector.

apple_colors <- c('green','green','yellow','red','red','red','green')


# Create a factor object.

factor_apple <- factor(apple_colors)


# Print the factor.

print(factor_apple)

print(nlevels(factor_apple))
```

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

```
[1] green  green  yellow red    red    red    green

Levels: green red yellow

[1] 3
```

Data Frames

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the **data.frame()** function.

```
# Create the data frame.

BMI <-           data.frame(

   gender = c("Male", "Male","Female"),

   height = c(152, 171.5, 165),

   weight = c(81,93, 78),

   Age = c(42,38,26)

)

print(BMI)
```

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

```
 gender height weight Age

1  Male  152.0      81 42

2  Male  171.5      93 38

3 Female 165.0      78  26
```

7. R - Functions :

A function is a set of statements organized together to perform a specific task. R has a large number of in-built functions and the user can create their own functions.

In R, a function is an object so the R interpreter is able to pass control to the function, along with arguments that may be necessary for the function to accomplish the actions.

The function in turn performs its task and returns control to the interpreter as well as any result which may be stored in other objects.

Function Definition

An R function is created by using the keyword function. The basic syntax of an R function definition is as follows –

```
function_name <- function(arg_1, arg_2, ...) {
   Function body
}
```

Function Components

The different parts of a function are −

- **Function Name** − This is the actual name of the function. It is stored in R environment as an object with this name.
- **Arguments** − An argument is a placeholder. When a function is invoked, you pass a value to the argument. Arguments are optional; that is, a function may contain no arguments. Also arguments can have default values.
- **Function Body** − The function body contains a collection of statements that defines what the function does.
- **Return Value** − The return value of a function is the last expression in the function body to be evaluated.

R has many **in-built** functions which can be directly called in the program without defining them first. We can also create and use our own functions referred as **user defined** functions.

Built-in Function

Simple examples of in-built functions are **seq()**, **mean()**, **max()**, **sum(x)** and **paste(...)** etc. They are directly called by user written programs.

 # Create a sequence of numbers from 32 to 44.

print(seq(32,44))

 # Find mean of numbers from 25 to 82.

print(mean(25:82))

 # Find sum of numbers frm 41 to 68.

print(sum(41:68))

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

[1] 32 33 34 35 36 37 38 39 40 41 42 43 44

[1] 53.5

[1] 1526

User-defined Function

We can create user-defined functions in R. They are specific to what a user wants and once created they can be used like the built-in functions. Below is an example of how a function is created and used.

```
# Create a function to print squares of numbers in sequence.

new.function <- function(a) {

  for(i in 1:a) {

    b <- i^2

    print(b)

  }}
```

Calling a Function :

```
# Create a function to print squares of numbers in sequence.
new.function <- function(a) {
  for(i in 1:a) {
    b <- i^2
    print(b)
  }
}

# Call the function new.function supplying 6 as an argument.
new.function(6)
```

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

```
[1] 1
[1] 4
[1] 9
[1] 16
[1] 25
[1] 36
```

Calling a Function without an Argument

```
# Create a function without an argument.
new.function <- function() {
  for(i in 1:5) {
    print(i^2)
  }
}

# Call the function without supplying an argument.
new.function()
```

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

```
[1] 1
[1] 4
[1] 9
[1] 16
```

```
[1] 25
```

Calling a Function with Argument Values (by position and by name)

The arguments to a function call can be supplied in the same sequence as defined in the function or they can be supplied in a different sequence but assigned to the names of the arguments.

```
# Create a function with arguments.
new.function <- function(a,b,c) {
   result <- a * b + c
   print(result)
}

# Call the function by position of arguments.
new.function(5,3,11)

# Call the function by names of the arguments.
new.function(a = 11, b = 5, c = 3)
```

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

```
[1] 26
[1] 58
```

Calling a Function with Default Argument

We can define the value of the arguments in the function definition and call the function without supplying any argument to get the default result. But we can also call such functions by supplying new values of the argument and get non default result.

```
# Create a function with arguments.
new.function <- function(a = 3, b = 6) {
   result <- a * b
   print(result)
}

# Call the function without giving any argument.
new.function()

# Call the function with giving new values of the argument.
new.function(9,5)
```

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

[1] 18
[1] 45

8.R - CSV Files :

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

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

Getting and Setting the Working Directory

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

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

# Set current working directory.
setwd("/home/amol/input")

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

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

[1] "/home/amol/Desktop/R"

[1] "/home/amol/input"

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

Input as CSV File

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

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

id,name,salary,start_date,dept

1,A,623.3,2012-01-01,IT

2,B,515.2,2013-09-23,Operations

3,C,611,2014-11-15,IT

4,D,729,2014-05-11,HR

5,E,843.25,2015-03-27,Finance

6,F,578,2018-05-21,IT

7,G,632.8,2013-07-30,Operations

8,H,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    A 623.30 2012-01-01            IT
2  2    B 515.20 2013-09-23 Operations
3  3    C 611.00 2014-11-15            IT
```

4 4 D 729.00 2014-05-11 HR

5 5 E 843.25 2015-03-27 Finance

6 6 F 578.00 2018-05-21 IT

7 7 G 632.80 2013-07-30 Operations

8 8 H 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  5       E 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       A  623.3 2012-01-01   IT

3  3       C  611.0 2014-11-15   IT

6  6       F  578.0 2018-05-21   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       A  623.3 2012-01-01   IT

3  3       C  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       C 611.00 2014-11-15     IT

4  4       D 729.00 2014-05-1
```

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)
```

Output:

X id name salary start_date dept

1 3 3 C 611.00 2014-11-15 IT

2 4 4 D 729.00 2014-05-11 HR

3 5 5 E 843.25 2015-03-27 Finance

4 6 6 F 578.00 2018-05-21 IT

5 8 8 H 722.50 2014-06-17 Finance

9.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":["A","B","C","D","E","F","G","H" ],
   "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] "A" "B" "C" "D" "E" "F" "G" "H"

$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"
[6] "5/21/2013" "7/30/2013" "6/17/2014"

$Dept
[1] "IT"          "Operations" "IT"       "HR"             "Finance"
[6] "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  StartDate    Dept
1  1    A  623.3   1/1/2012              IT
2  2    B  515.2  9/23/2013 Operations
3 3     C        611 11/15/2014         IT
4 4     D         729 5/11/2014         HR
5  5    E 843.25  3/27/2015   Finance
6  6    F         578  5/21/2013         IT
7  7    G  632.8  7/30/2013 Operations
8  8    H  722.5  6/17/2014   Finance
```

10.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)

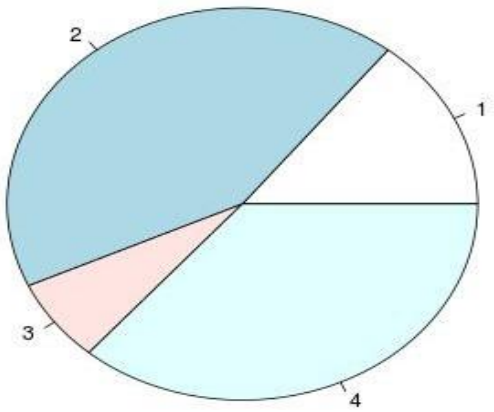
Following is the description of the parameters used −

- **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

A very simple pie-chart is created using just the input vector and labels. The below script will create and save the pie chart in the current R working directory.

```
# Create data for the graph.

x <- c(21, 62, 10, 53)

labels <- c("1", "2", "3", "4")

# Give the chart file a name.

png(file = "output.png")

# Plot the chart.

pie(x,labels)

# Save the file.

dev.off()
```

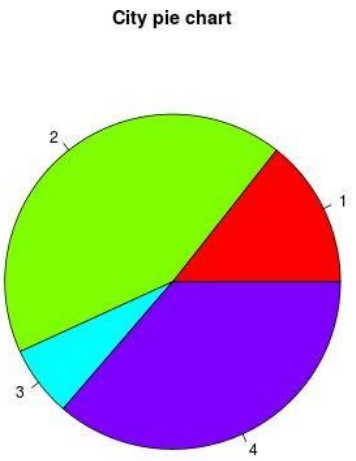


Pie Chart Title and Colors

We can expand the features of the chart by adding more parameters to the function. We will use parameter **main** to add a title to the chart and another parameter is **col** which will make use of rainbow colour pallet while drawing the chart. The length of the pallet should be same as the number of values we have for the chart. Hence we use length(x).

Example

```
# Create data for the graph.

x <- c(21, 62, 10, 53)

labels <- c("1", "2", "3", "4")

# Give the chart file a name.

png(file = "output.png")

# Plot the chart with title and rainbow color pallet.

pie(x, labels, main = "City pie chart", col = rainbow(length(x)))

# Save the file.

dev.off()
```



11.R - Bar Charts :

A bar chart represents data in rectangular bars with length of the bar proportional to the value of the variable. R uses the function barplot() to create bar charts. R can draw both vertical and Horizontal bars in the bar chart. In bar chart each of the bars can be given different colors.

Syntax

The basic syntax to create a bar-chart in R is −

barplot(H,xlab,ylab,main, names.arg,col)

Following is the description of the parameters used −

- H is a vector or matrix containing numeric values used in bar chart.
- xlab is the label for x axis.
- ylab is the label for y axis.
- main is the title of the bar chart.
- names.arg is a vector of names appearing under each bar.
- col is used to give colors to the bars in the graph.

Example

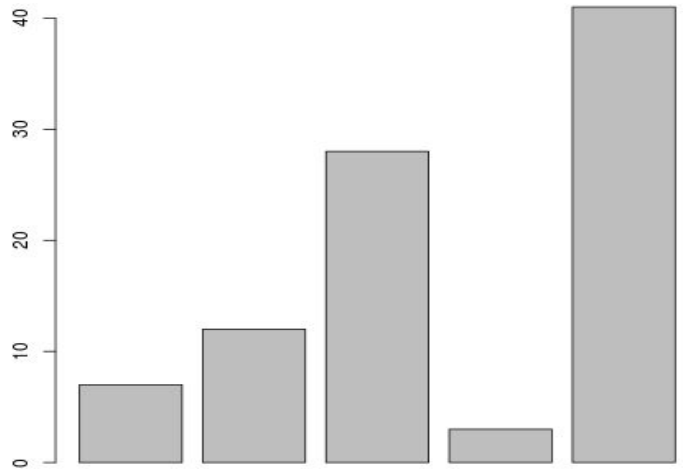
A simple bar chart is created using just the input vector and the name of each bar.

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart

H <- c(7,12,28,3,41)

# Give the chart file a name

png(file =  "barchart.png")

# Plot the bar chart

barplot(H)

# Save the file
```

dev.off()

When we execute above code, it produces following result −



Bar Chart Labels, Title and Colors

The features of the bar chart can be expanded by adding more parameters. The **main** parameter is used to add **title**. The **col** parameter is used to add colors to the bars. The **args.name** is a vector having same number of values as the input vector to describe the meaning of each bar.

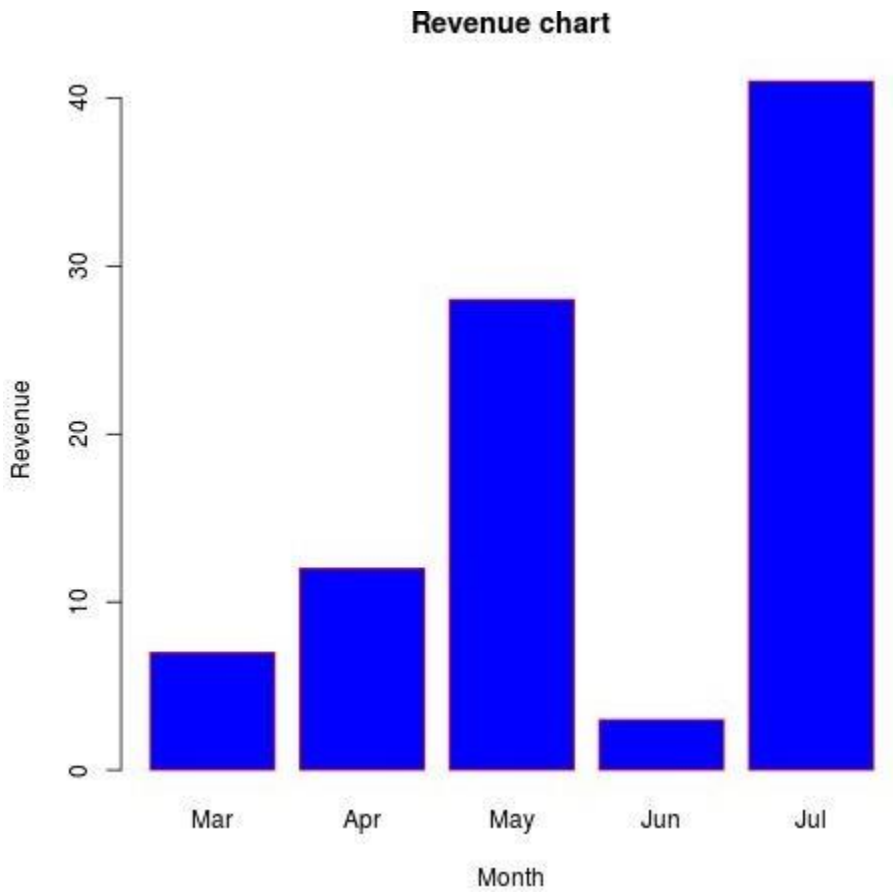
Example

The below script will create and save the bar chart in the current R working directory.

```
# Create the data for the chart

H <- c(7,12,28,3,41)

M <- c("Mar","Apr","May","Jun","Jul")

# Give the chart file a name

png(file = "barchart_months_revenue.png")

barplot(H,names.arg=M,xlab="Month",ylab="Revenue",col="blue",main="Revenue chart",border="red")
```

dev.off()

When we execute above code, it produces following result −



Coclusion: Thus we have visualized our data successfully using R Programming.