UNIT-II

R Data interfaces - CSV Files, XML files, Web Data- Data Preprocessing: Missing Values, Principle Component Analysis - Data Visualization – Charts & Graphs-Pie Chart, Bar Chart, Box plot, Histogram, Line graph, Scatter Plot.

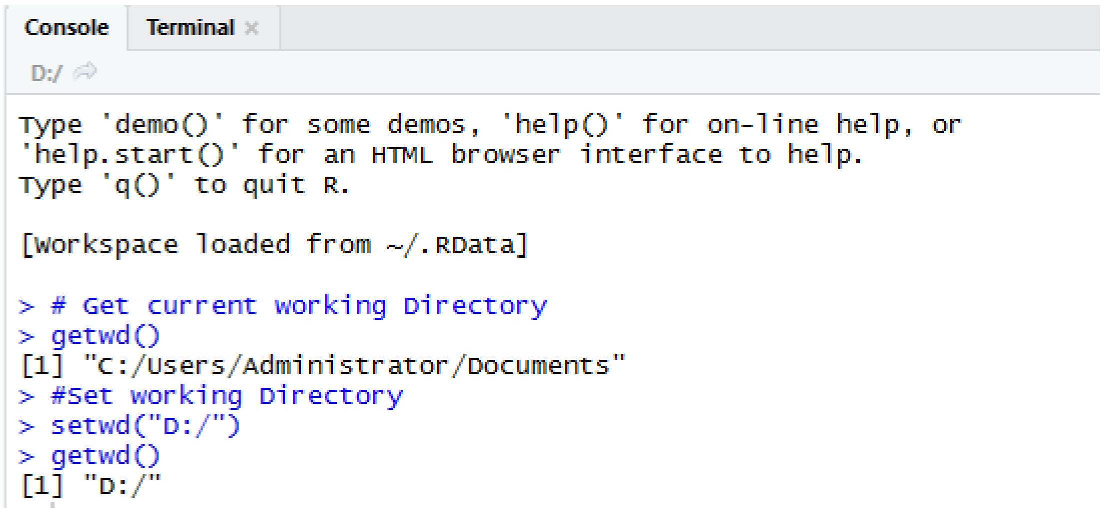
R Data Interfaces:

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.

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.

Example:

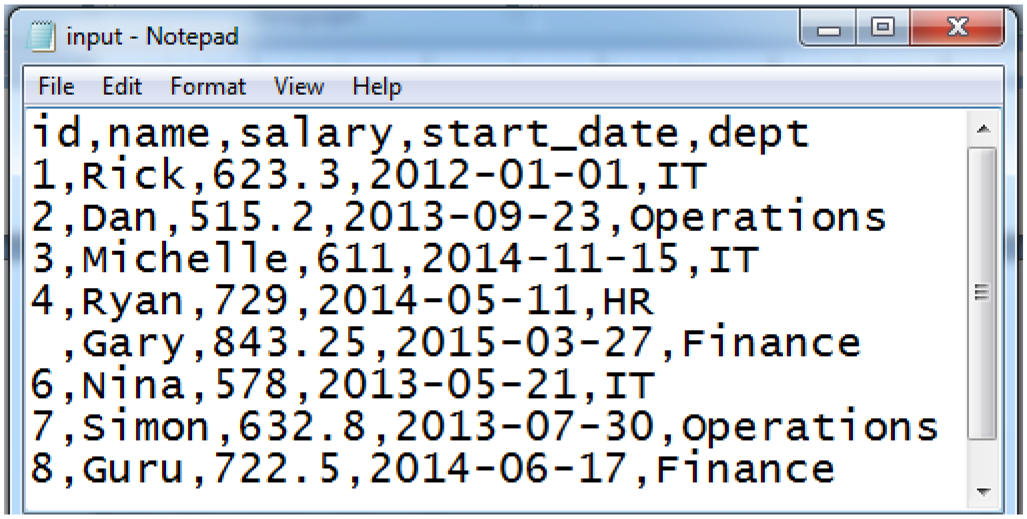


```
Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Workspace loaded from ~/.RData]

> # Get current working Directory
> getwd()
[1] "C:/Users/Administrator/Documents"
> #Set working Directory
> setwd("D:/")
> getwd()
[1] "D:/"
```

CSV FILES

Input as CSV files:

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

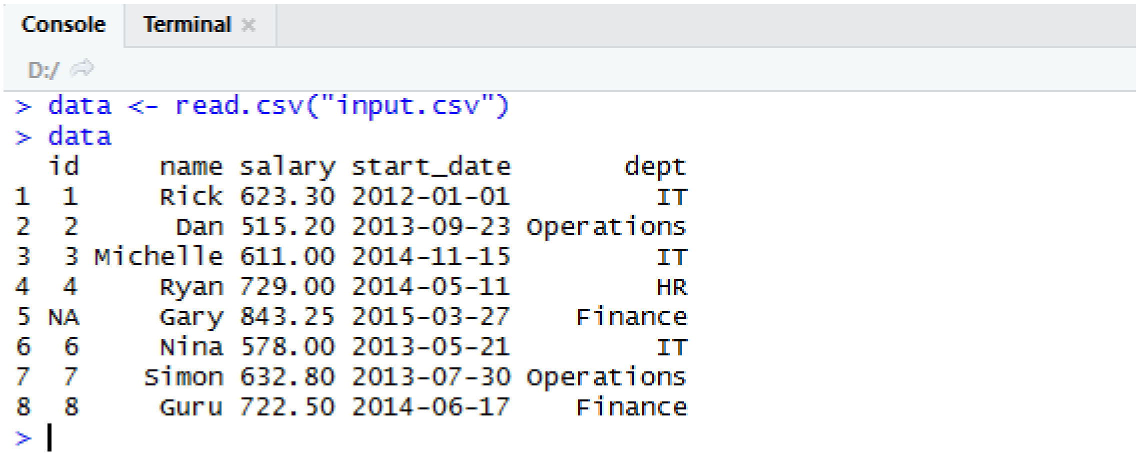


```
id,name,salary,start_date,dept
1,Rick,623.3,2012-01-01,IT
2,Dan,515.2,2013-09-23,Operations
3,Michelle,611,2014-11-15,IT
4,Ryan,729,2014-05-11,HR
 ,Gary,843.25,2015-03-27,Finance
6,Nina,578,2013-05-21,IT
7,Simon,632.8,2013-07-30,Operations
8,Guru,722.5,2014-06-17,Finance
```

Reading a CSV File

read.csv() function is used to read a CSV file available in your current working directory .

Example:

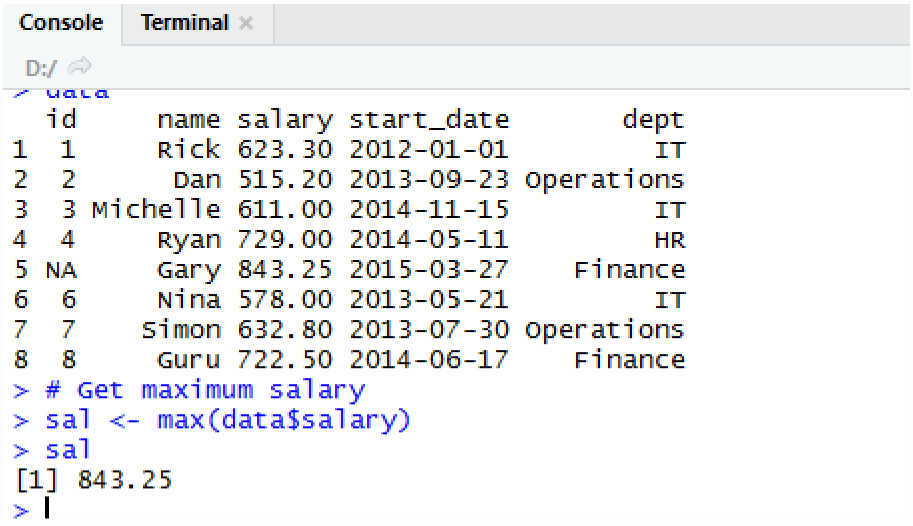


```
> data <- read.csv("input.csv")
> data
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> 
```

Analysing CSV file:

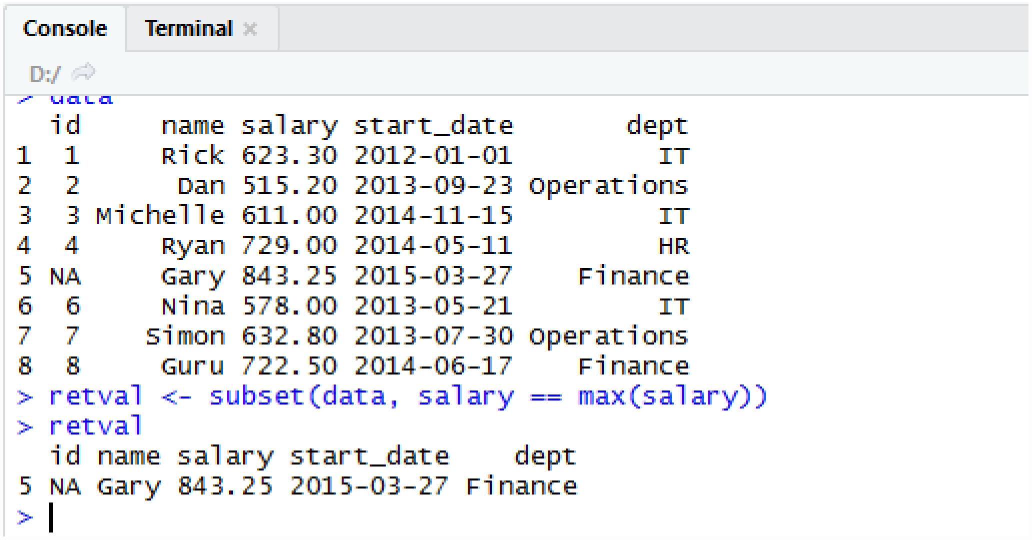
By default the read.csv() function gives the output as a data frame. Once we read data in a data frame, we can apply all the functions applicable to data frames.

1.Get maximum salary.



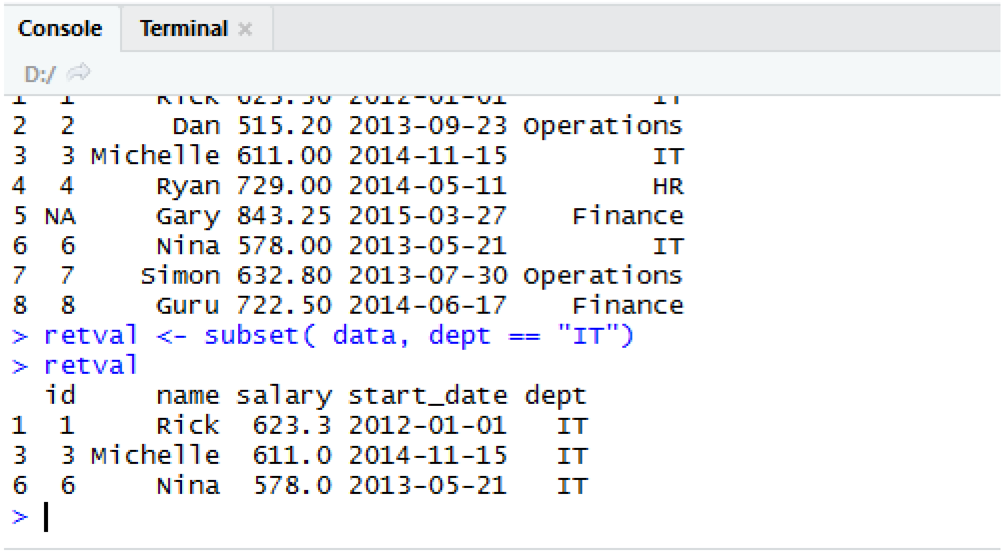
```
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> # Get maximum salary
> sal <- max(data$salary)
> sal
[1] 843.25
> 
```

2. Get the details of the person with max salary



```
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> retval <- subset(data, salary == max(salary))
> retval
  id name salary start_date    dept
5 NA Gary 843.25 2015-03-27 Finance
> 
```

3. Get all the people working in IT department

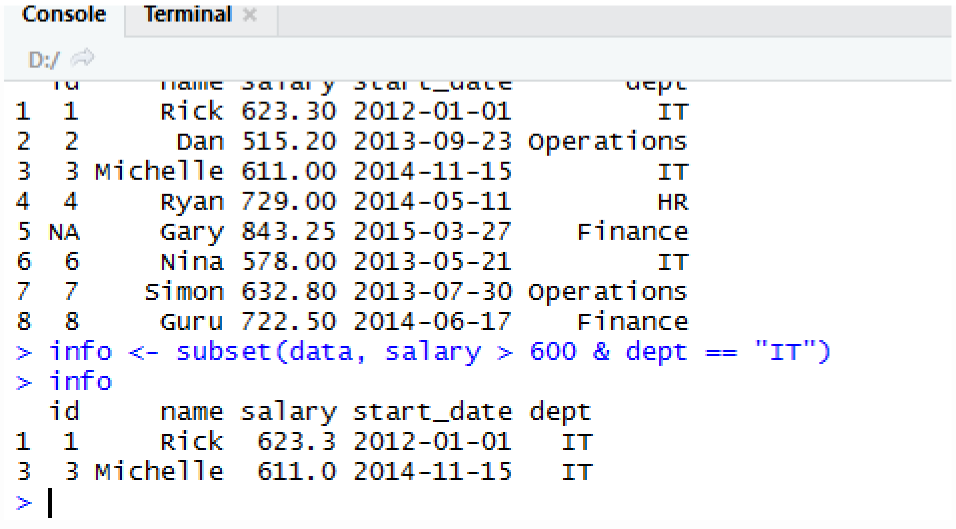


Console Terminal

D:/

```
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> retval <- subset( data, dept == "IT")
> retval
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
6  6     Nina  578.0 2013-05-21   IT
> 
```

4. Get the persons in IT department whose salary is greater than 600



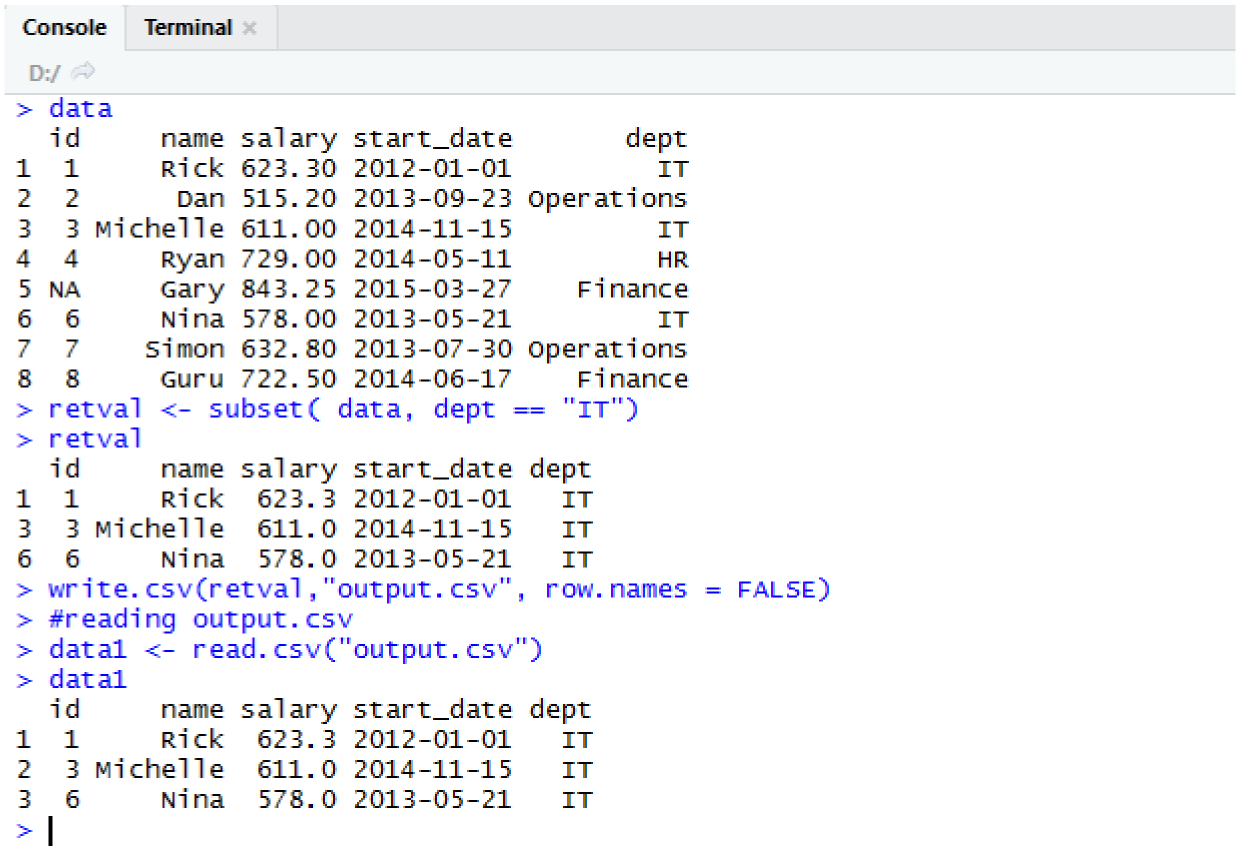
Console Terminal

D:/

```
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> info <- subset(data, salary > 600 & dept == "IT")
> info
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
> 
```

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.



```
> data
  id     name salary start_date       dept
1  1     Rick 623.30 2012-01-01         IT
2  2      Dan 515.20 2013-09-23 Operations
3  3 Michelle 611.00 2014-11-15         IT
4  4     Ryan 729.00 2014-05-11         HR
5 NA     Gary 843.25 2015-03-27    Finance
6  6     Nina 578.00 2013-05-21         IT
7  7    Simon 632.80 2013-07-30 Operations
8  8     Guru 722.50 2014-06-17    Finance
> retval <- subset( data, dept == "IT")
> retval
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
3  3 Michelle  611.0 2014-11-15   IT
6  6     Nina  578.0 2013-05-21   IT
> write.csv(retval,"output.csv", row.names = FALSE)
> #reading output.csv
> data1 <- read.csv("output.csv")
> data1
  id     name salary start_date dept
1  1     Rick  623.3 2012-01-01   IT
2  3 Michelle  611.0 2014-11-15   IT
3  6     Nina  578.0 2013-05-21   IT
> 
```

XML files

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

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

Install.packages(“XML”)

Input Data

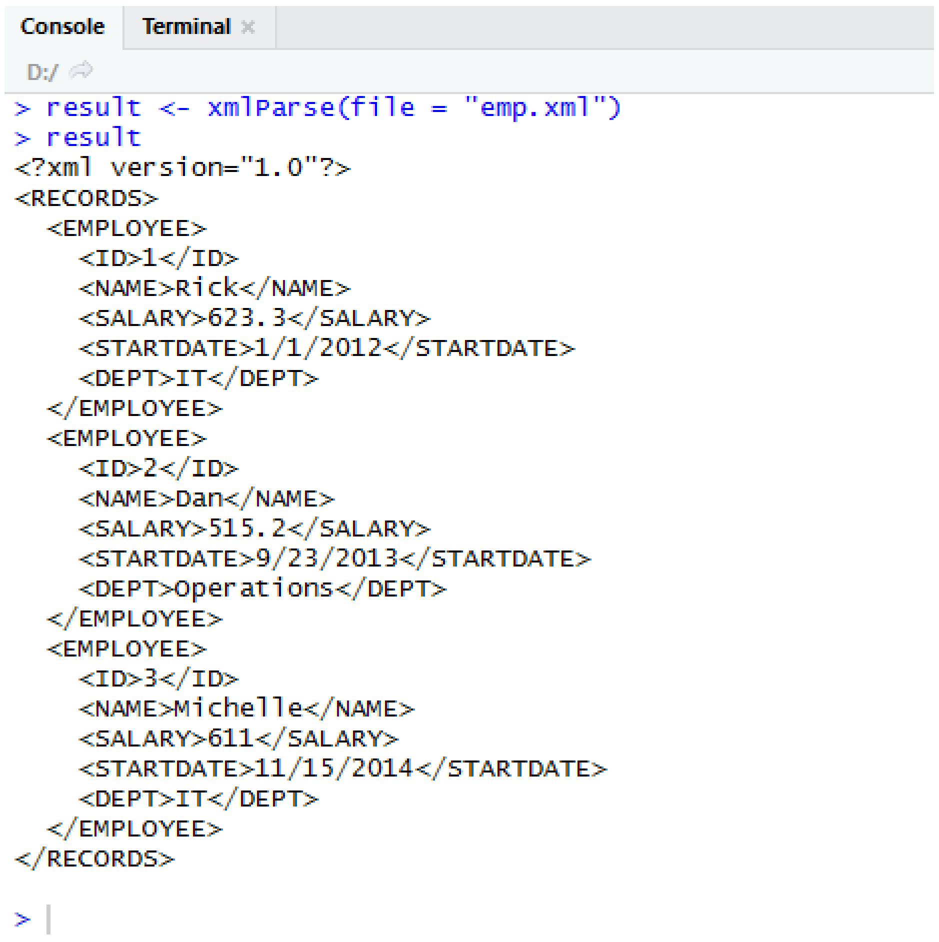
Create a XMl file by copying the below data into a text editor like notepad. Save the file with a **.xml** extension and choosing the file type as **all files(*.*)**.

emp.xml

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

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

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

</RECORDS>
```

Reading XML File

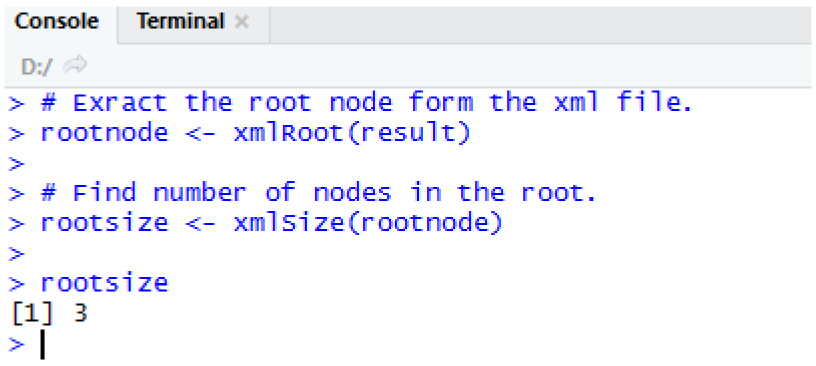
The xml file is read by R using the function **xmlParse()**. It is stored as a list in R.


Console Terminal

D:/

```
> result <- xmlParse(file = "emp.xml")
> result
<?xml version="1.0"?>
<RECORDS>
  <EMPLOYEE>
    <ID>1</ID>
    <NAME>Rick</NAME>
    <SALARY>623.3</SALARY>
    <STARTDATE>1/1/2012</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>2</ID>
    <NAME>Dan</NAME>
    <SALARY>515.2</SALARY>
    <STARTDATE>9/23/2013</STARTDATE>
    <DEPT>Operations</DEPT>
  </EMPLOYEE>
  <EMPLOYEE>
    <ID>3</ID>
    <NAME>Michelle</NAME>
    <SALARY>611</SALARY>
    <STARTDATE>11/15/2014</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
</RECORDS>

> |
```

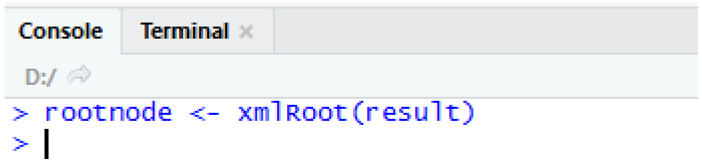
Get Number of Nodes Present in XML File


Console Terminal

D:/

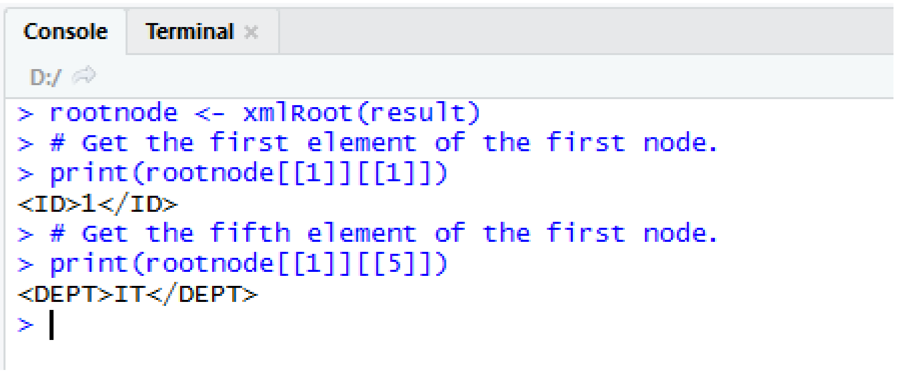
```
> # Exract the root node form the xml file.
> rootnode <- xmlRoot(result)
>
> # Find number of nodes in the root.
> rootsize <- xmlSize(rootnode)
>
> rootsize
[1] 3
> |
```

Details of the First Node



```
> rootnode <- xmlRoot(result)
>
```

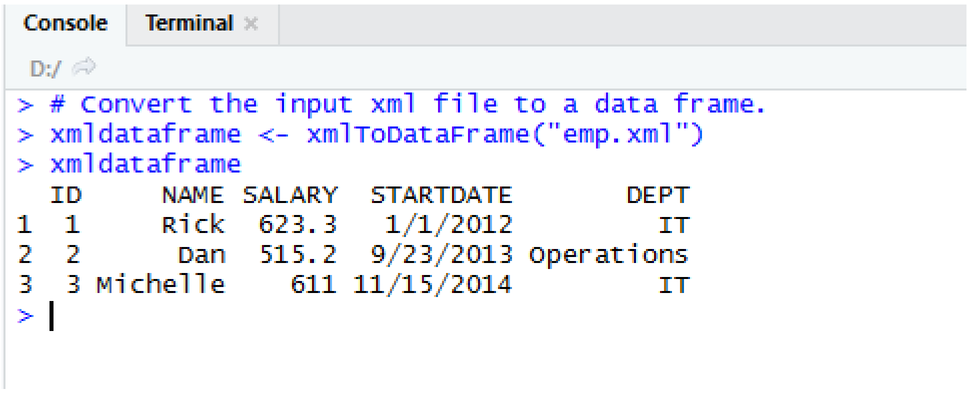
Get Different Elements of a Node



```
> rootnode <- xmlRoot(result)
> # Get the first element of the first node.
> print(rootnode[[1]][[1]])
<ID>1</ID>
> # Get the fifth element of the first node.
> print(rootnode[[1]][[5]])
<DEPT>IT</DEPT>
>
```

XML to Data Frame

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



```
> # Convert the input xml file to a data frame.
> xmldataframe <- xmlToDataFrame("emp.xml")
> xmldataframe
  ID     NAME SALARY  STARTDATE       DEPT
1  1     Rick  623.3   1/1/2012         IT
2  2      Dan  515.2  9/23/2013 Operations
3  3 Michelle    611 11/15/2014         IT
>
```

WEB DATA

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

Install R Packages

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

```
install.packages("RCurl")

install.packages("XML")

install.packages("stringr")

install.packages("plyr")
```

Input Data

We will visit the URL weather data and download the CSV files using R for the year 2015.

Example

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

```
# Read the URL.

url <- "http://www.geos.ed.ac.uk/~weather/jcmb_ws/"
```

```
# Gather the html links present in the webpage.

links <- getHTMLLinks(url)

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

filenames <- links[str_detect(links, "JCMB_2015")]

# Store the file names as a list.

filenames_list <- as.list(filenames)

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

downloadcsv <- function (mainurl,filename) {

   filedetails <- str_c(mainurl,filename)

   download.file(filedetails,filename)

}

# Now apply the lapply function and save the files into the current R working directory.

lapply(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"
```

R Charts and Graphs

R Programming language has numerous libraries to create charts and graphs.

Pie Chart

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

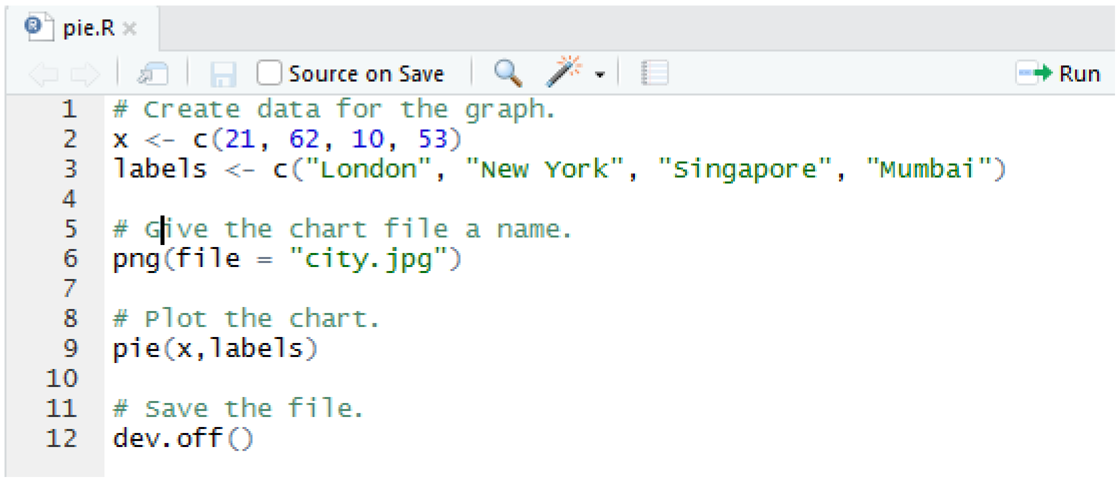
The basic syntax for creating a pie-chart using the R is −

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

Following is the description of the parameters used −

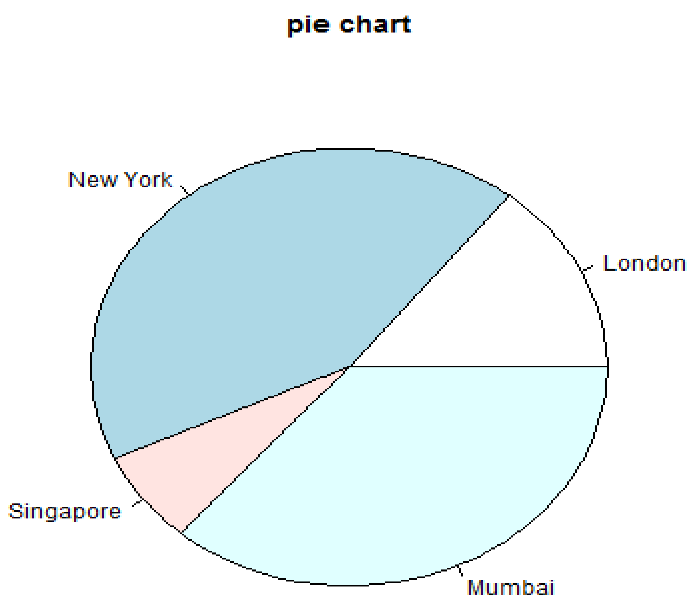
- **x** is a vector containing the numeric values used in the pie chart.
- **labels** is used to give description to the slices.
- **radius** indicates the radius of the circle of the pie chart.(value between −1 and +1).
- **main** indicates the title of the chart.
- **col** indicates the color palette.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

Example:



```r
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

# Give the chart file a name.
png(file = "city.jpg")

# Plot the chart.
pie(x,labels)

# Save the file.
dev.off()
```

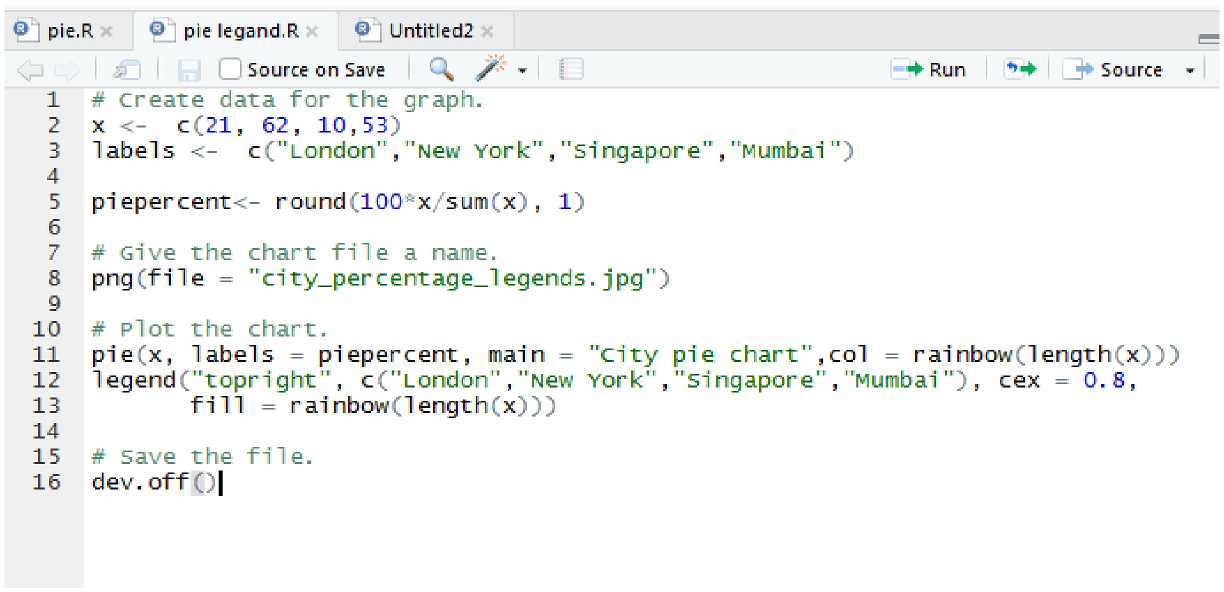
Output:



Slice Percentages and Chart Legend

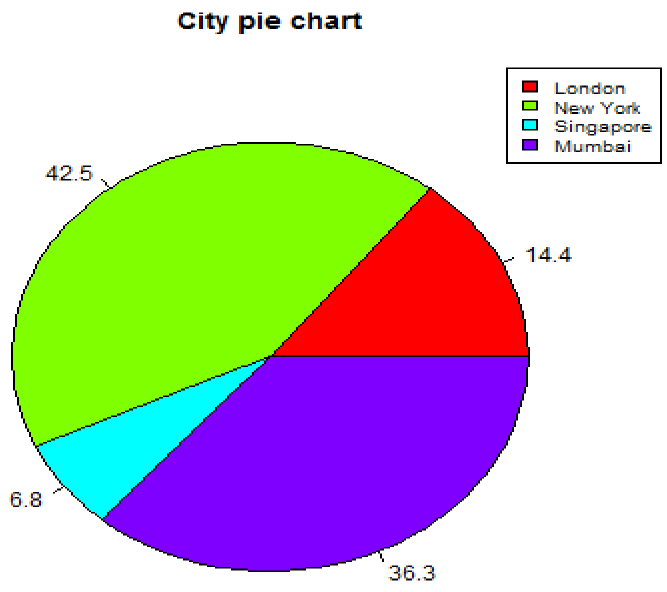
We can add slice percentage and a chart legend by creating additional chart variables.

Example:



```
# Create data for the graph.
x <-  c(21, 62, 10,53)
labels <-  c("London","New York","Singapore","Mumbai")

piepercent<- round(100*x/sum(x), 1)

# Give the chart file a name.
png(file = "city_percentage_legends.jpg")

# Plot the chart.
pie(x, labels = piepercent, main = "City pie chart",col = rainbow(length(x)))
legend("topright", c("London","New York","Singapore","Mumbai"), cex = 0.8,
       fill = rainbow(length(x)))

# Save the the file.
dev.off()
```

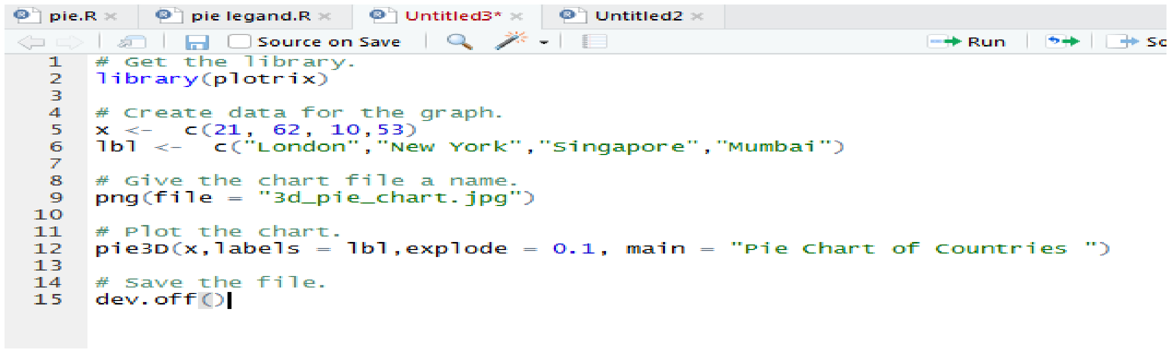
Output:



3D Pie Chart

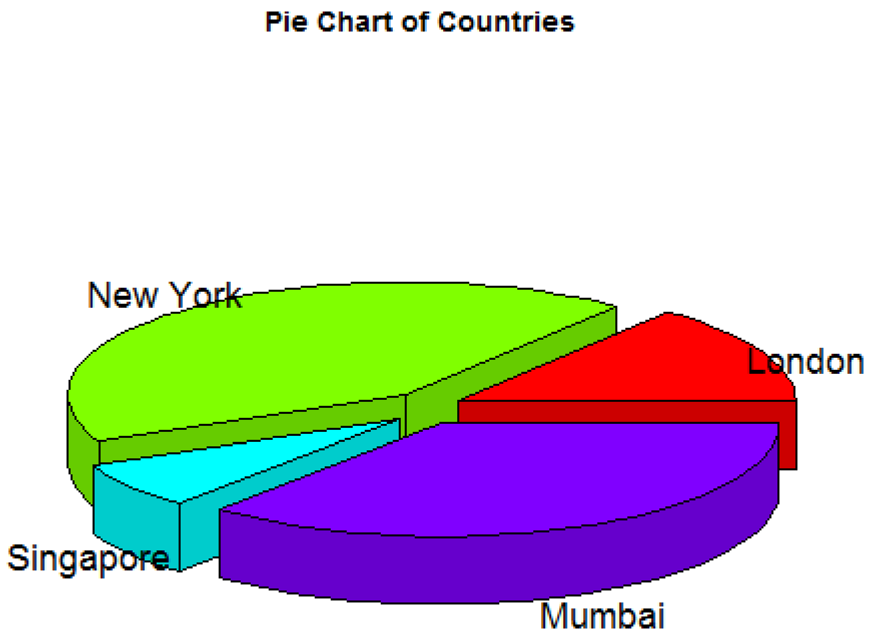
A pie chart with 3 dimensions can be drawn using additional packages. The package **plotrix** has a function called **pie3D()** that is used for this.

Example:



```
# Get the library.
library(plotrix)

# Create data for the graph.
x <-  c(21, 62, 10,53)
lbl <-  c("London","New York","Singapore","Mumbai")

# Give the chart file a name.
png(file = "3d_pie_chart.jpg")

# Plot the chart.
pie3D(x,labels = lbl,explode = 0.1, main = "Pie Chart of Countries ")

# Save the file.
dev.off()
```

Output:



BAR CHART

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 −

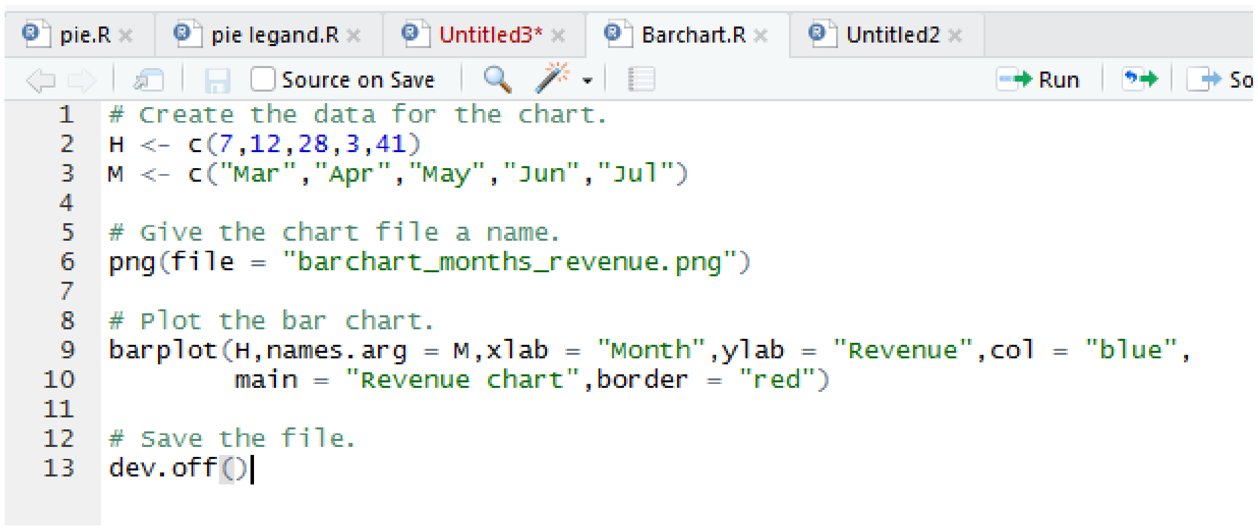
```
barplot(H, xlab, ylab, main, names.arg, col)
```

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.

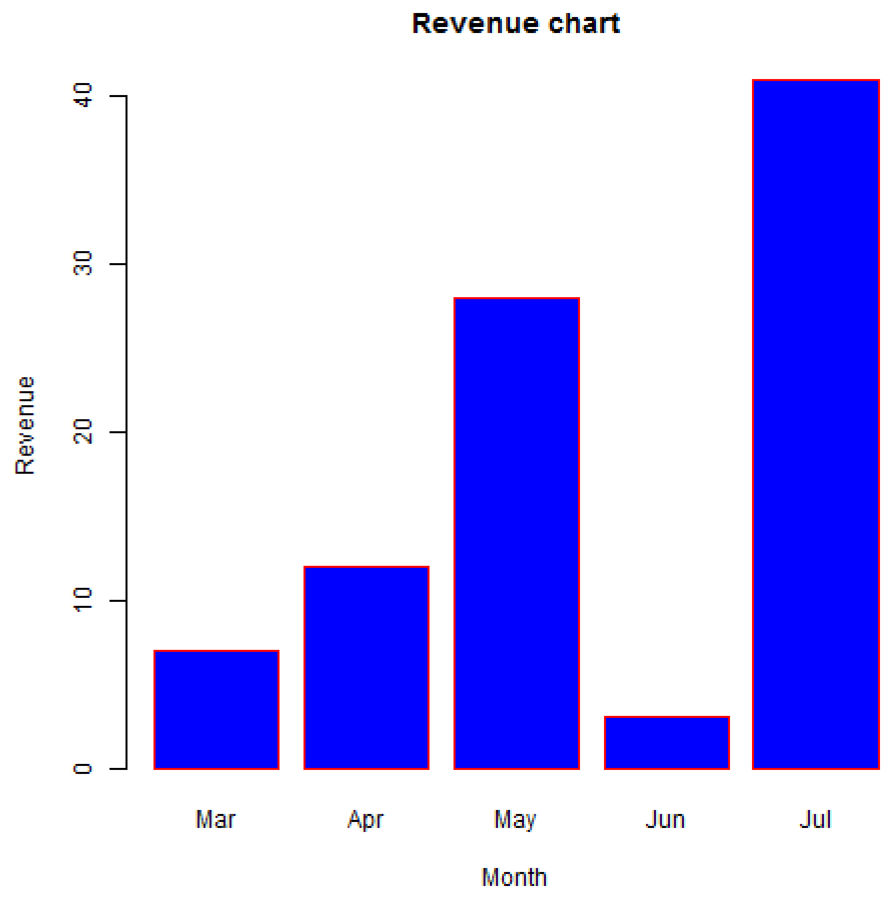
Example:

The following 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")

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

# Save the file.
dev.off()
```

Output:

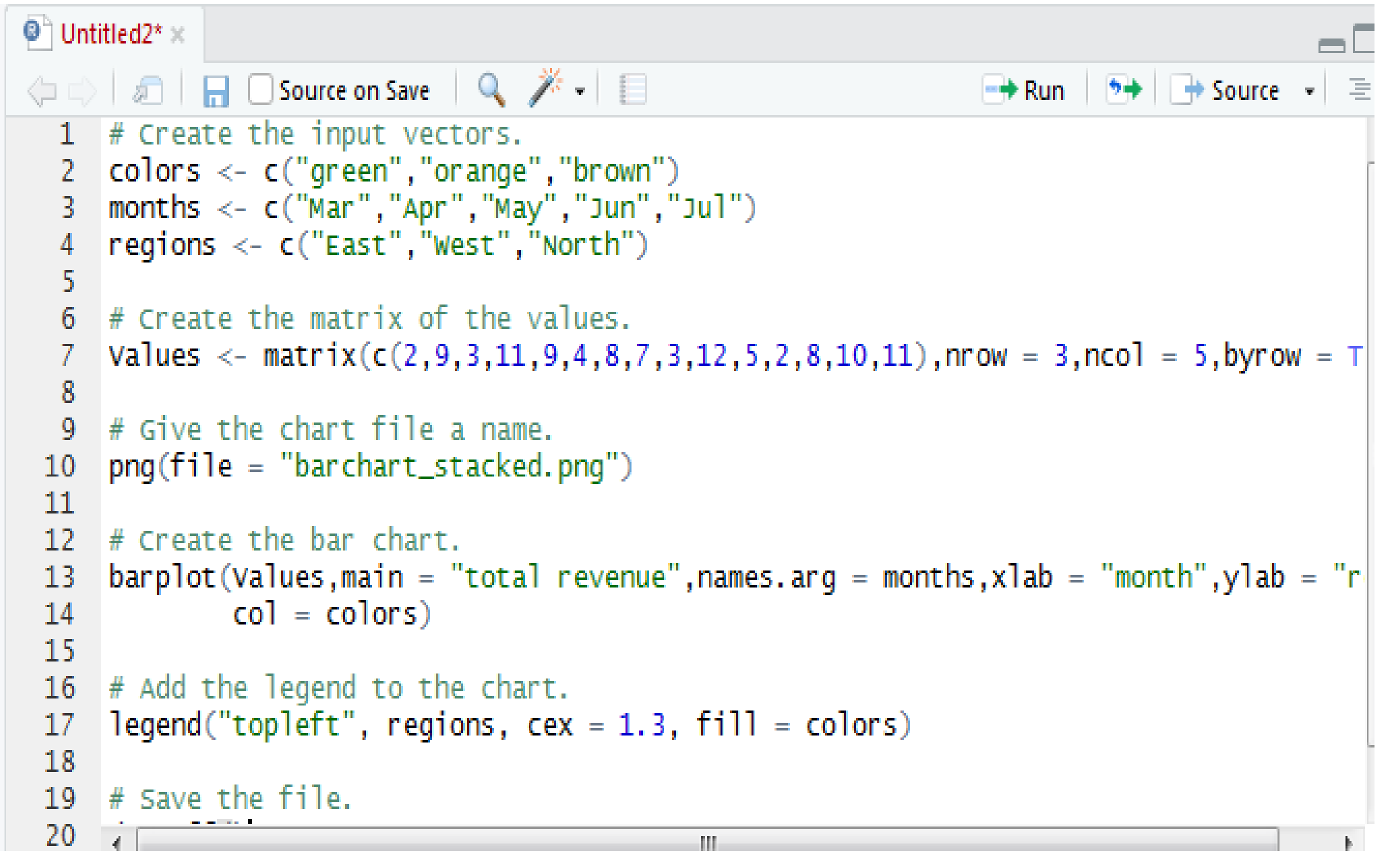


Group Bar Chart and Stacked Bar Chart

We can create bar chart with groups of bars and stacks in each bar by using a matrix as input values.

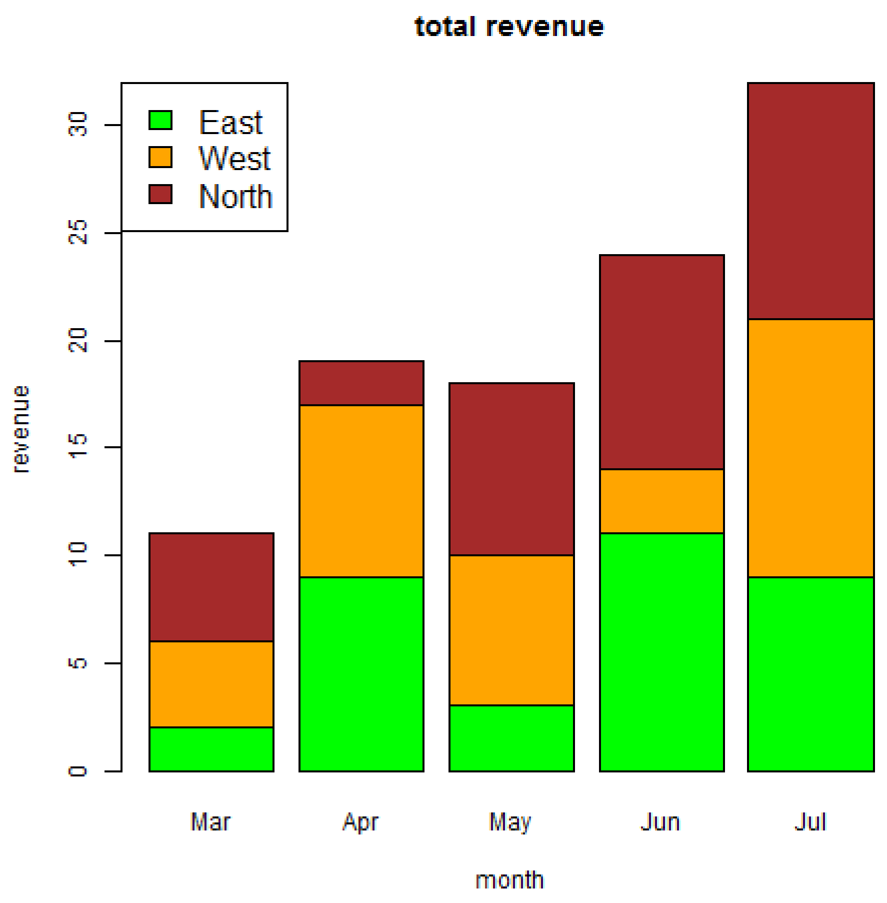
More than two variables are represented as a matrix which is used to create the group bar chart and stacked bar chart.

Example:



```
# Create the input vectors.
colors <- c("green","orange","brown")
months <- c("Mar","Apr","May","Jun","Jul")
regions <- c("East","West","North")

# Create the matrix of the values.
Values <- matrix(c(2,9,3,11,9,4,8,7,3,12,5,2,8,10,11),nrow = 3,ncol = 5,byrow = T

# Give the chart file a name.
png(file = "barchart_stacked.png")

# Create the bar chart.
barplot(Values,main = "total revenue",names.arg = months,xlab = "month",ylab = "r
        col = colors)

# Add the legend to the chart.
legend("topleft", regions, cex = 1.3, fill = colors)

# Save the file.
```

Output:



HISTOGRAM

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chat but the difference is it groups the values into continuous ranges. Each bar in histogram represents the height of the number of values present in that range.

R creates histogram using **hist()** function. This function takes a vector as an input and uses some more parameters to plot histograms.

Syntax

The basic syntax for creating a histogram using R is −

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

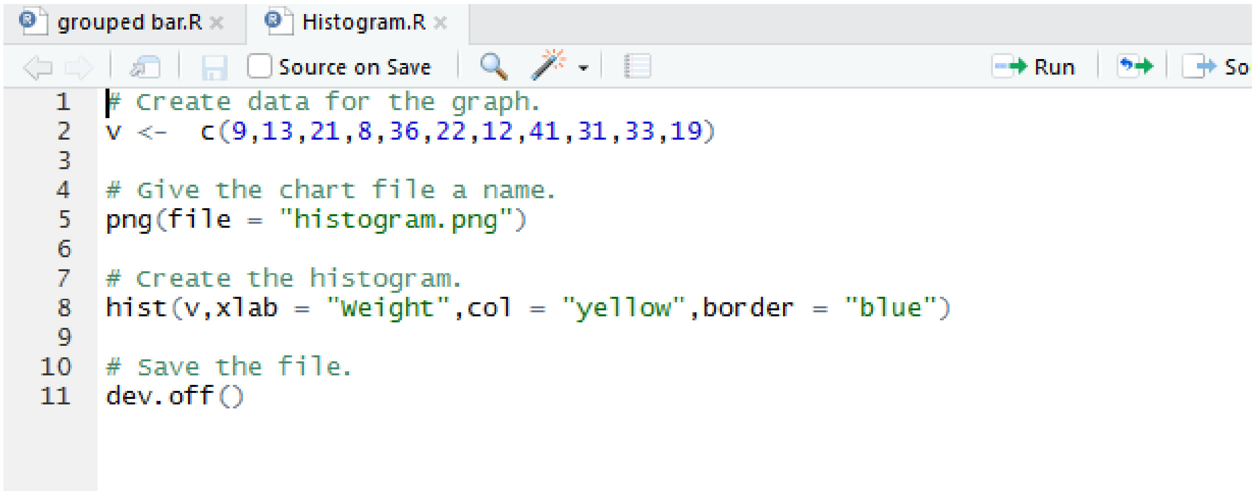
Following is the description of the parameters used −

- **v** is a vector containing numeric values used in histogram.
- **main** indicates title of the chart.
- **col** is used to set color of the bars.
- **border** is used to set border color of each bar.
- **xlab** is used to give description of x-axis.
- **xlim** is used to specify the range of values on the x-axis.
- **ylim** is used to specify the range of values on the y-axis.
- **breaks** is used to mention the width of each bar.

Example

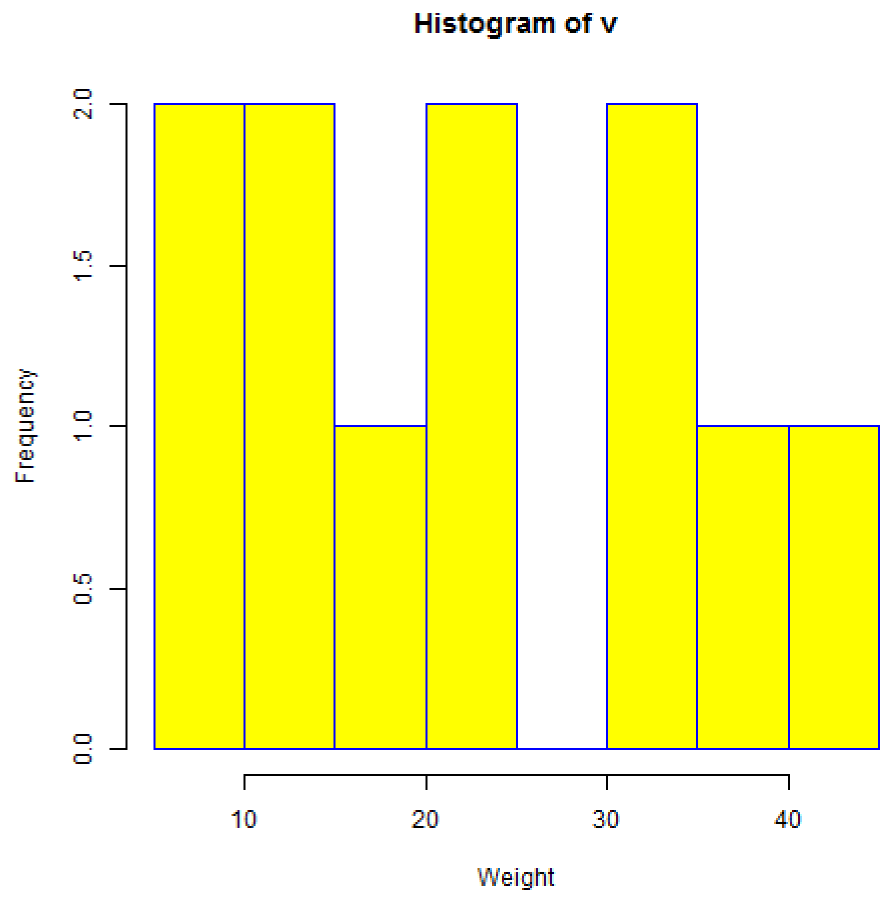
A simple histogram is created using input vector, label, col and border parameters.

The script given below will create and save the histogram in the current R working directory.



```
# Create data for the graph.
v <-  c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "yellow",border = "blue")

# Save the file.
dev.off()
```

Output:

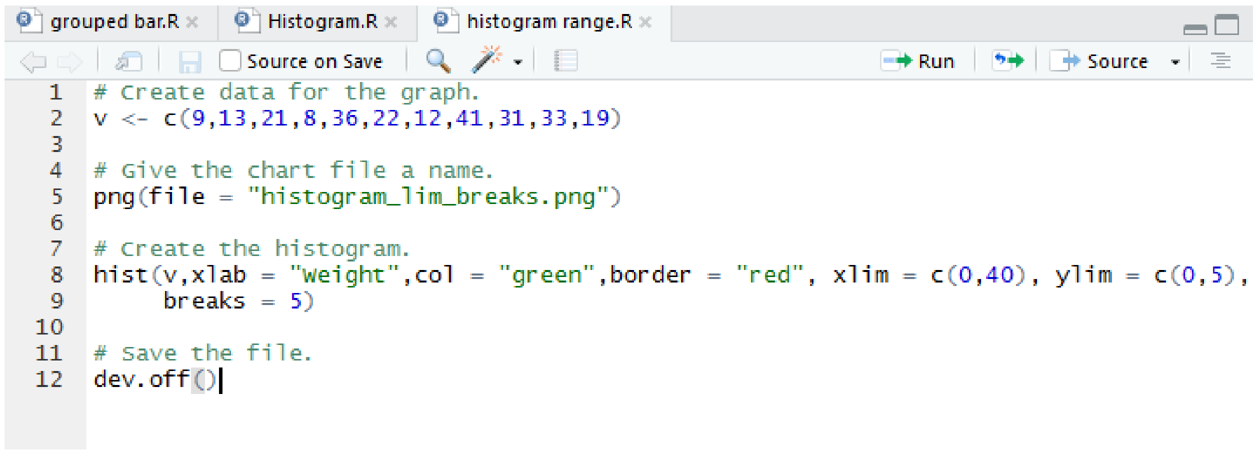


Range of X and Y values

To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.

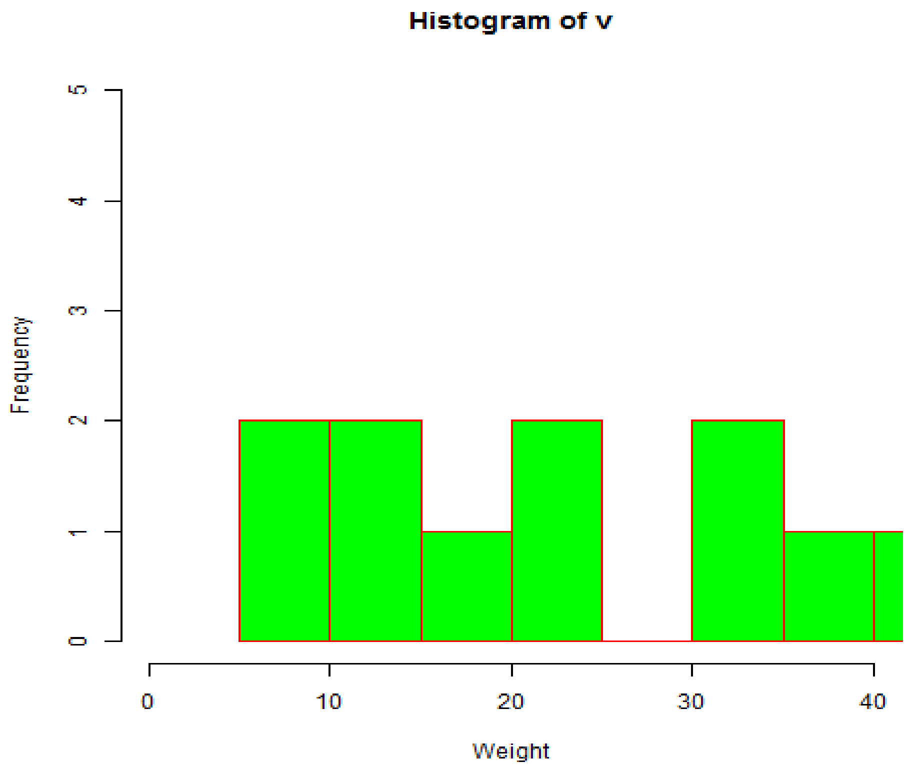
The width of each of the bar can be decided by using breaks.

Example:



```r
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# Give the chart file a name.
png(file = "histogram_lim_breaks.png")

# Create the histogram.
hist(v,xlab = "Weight",col = "green",border = "red", xlim = c(0,40), ylim = c(0,5),
     breaks = 5)

# Save the file.
dev.off()
```

Output:



LINE GRAPH

A line chart is a graph that connects a series of points by drawing line segments between them. These points are ordered in one of their coordinate (usually the x-coordinate) value. Line charts are usually used in identifying the trends in data.

The **plot()** function in R is used to create the line graph.

Syntax

The basic syntax to create a line chart in R is −

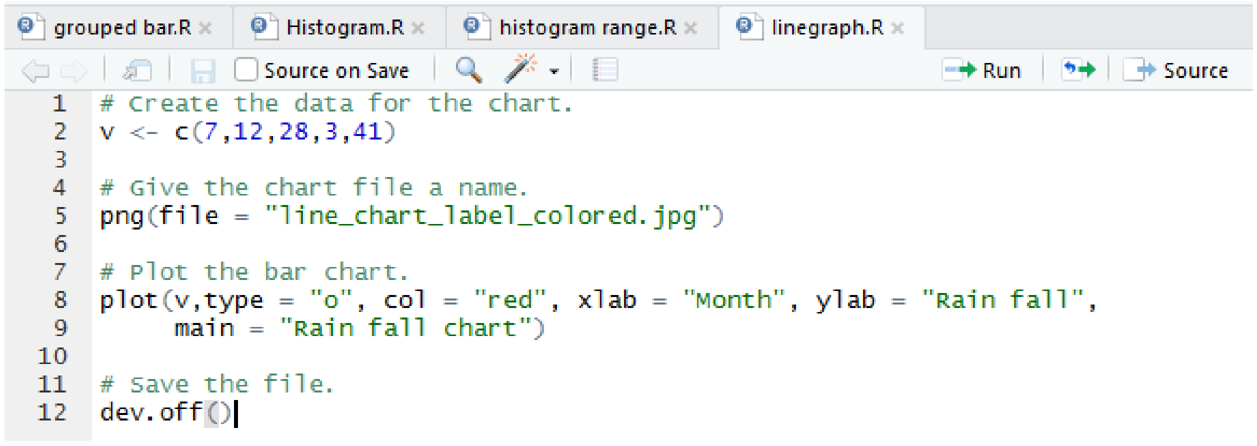
```
plot(v,type,col,xlab,ylab)
```

Following is the description of the parameters used −

- **v** is a vector containing the numeric values.
- **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- **xlab** is the label for x axis.
- **ylab** is the label for y axis.
- **main** is the Title of the chart.
- **col** is used to give colors to both the points and lines.

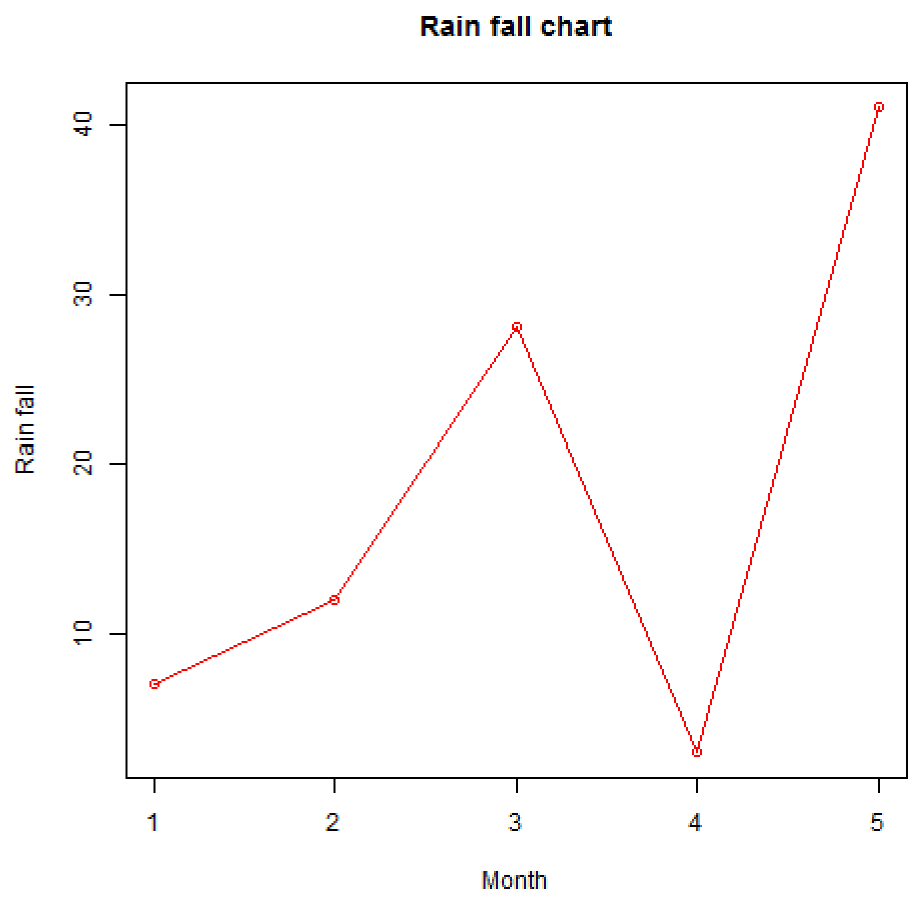
Example

A simple line chart is created using the input vector and the type parameter as "O". The below script will create and save a line chart in the current R working directory. The features of the line chart has been expanded by using additional parameters. We add color to the points and lines, give a title to the chart and add labels to the axes.



```
# Create the data for the chart.
v <- c(7,12,28,3,41)

# Give the chart file a name.
png(file = "line_chart_label_colored.jpg")

# Plot the bar chart.
plot(v,type = "o", col = "red", xlab = "Month", ylab = "Rain fall",
     main = "Rain fall chart")

# Save the file.
dev.off()
```

Output:

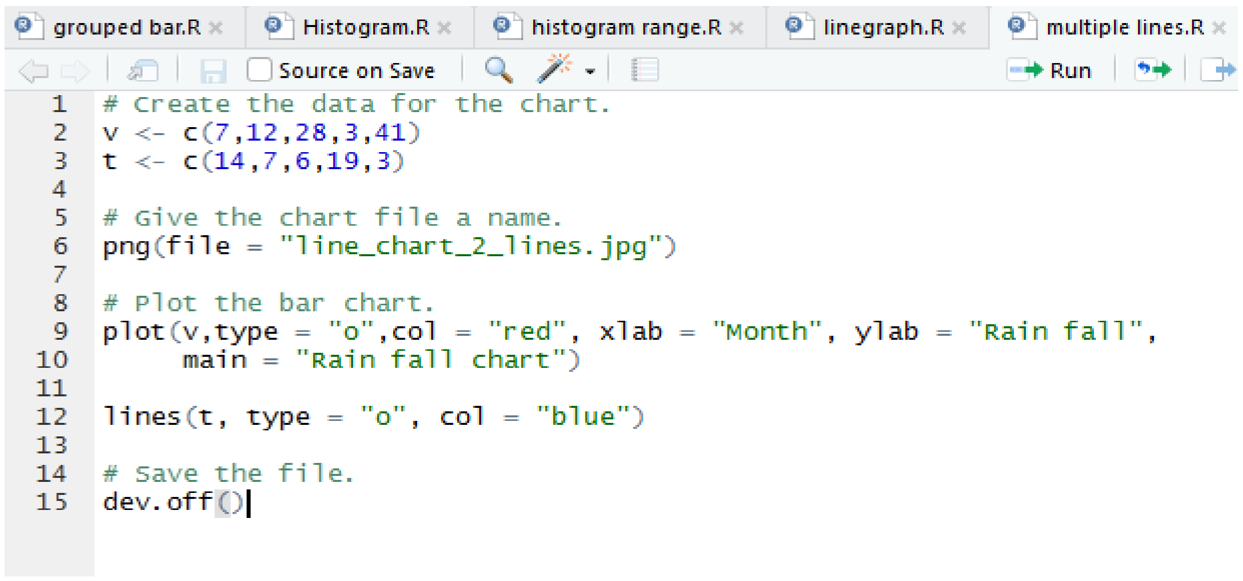


Multiple Lines in a Line Chart

More than one line can be drawn on the same chart by using the **lines()**function.

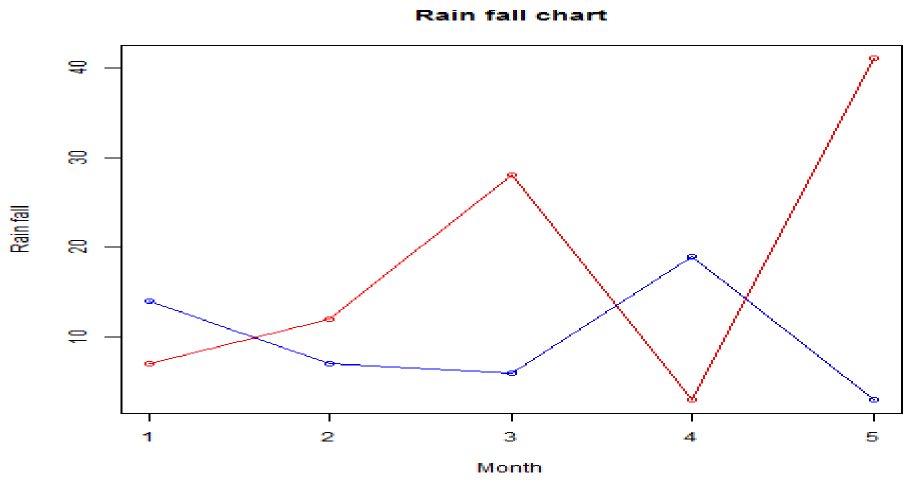
After the first line is plotted, the lines() function can use an additional vector as input to draw the second line in the chart,

Example:



```r
# Create the data for the chart.
v <- c(7,12,28,3,41)
t <- c(14,7,6,19,3)

# Give the chart file a name.
png(file = "line_chart_2_lines.jpg")

# Plot the bar chart.
plot(v,type = "o",col = "red", xlab = "Month", ylab = "Rain fall",
     main = "Rain fall chart")

lines(t, type = "o", col = "blue")

# Save the file.
dev.off()
```

Output:



Scattar Plots

Scatterplots show many points plotted in the Cartesian plane. Each point represents the values of two variables. One variable is chosen in the horizontal axis and another in the vertical axis.

The simple scatterplot is created using the **plot()** function.

Syntax

The basic syntax for creating scatterplot in R is –

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

Following is the description of the parameters used −

- **x** is the data set whose values are the horizontal coordinates.
- **y** is the data set whose values are the vertical coordinates.
- **main** is the tile of the graph.
- **xlab** is the label in the horizontal axis.
- **ylab** is the label in the vertical axis.
- **xlim** is the limits of the values of x used for plotting.
- **ylim** is the limits of the values of y used for plotting.
- **axes** indicates whether both axes should be drawn on the plot.

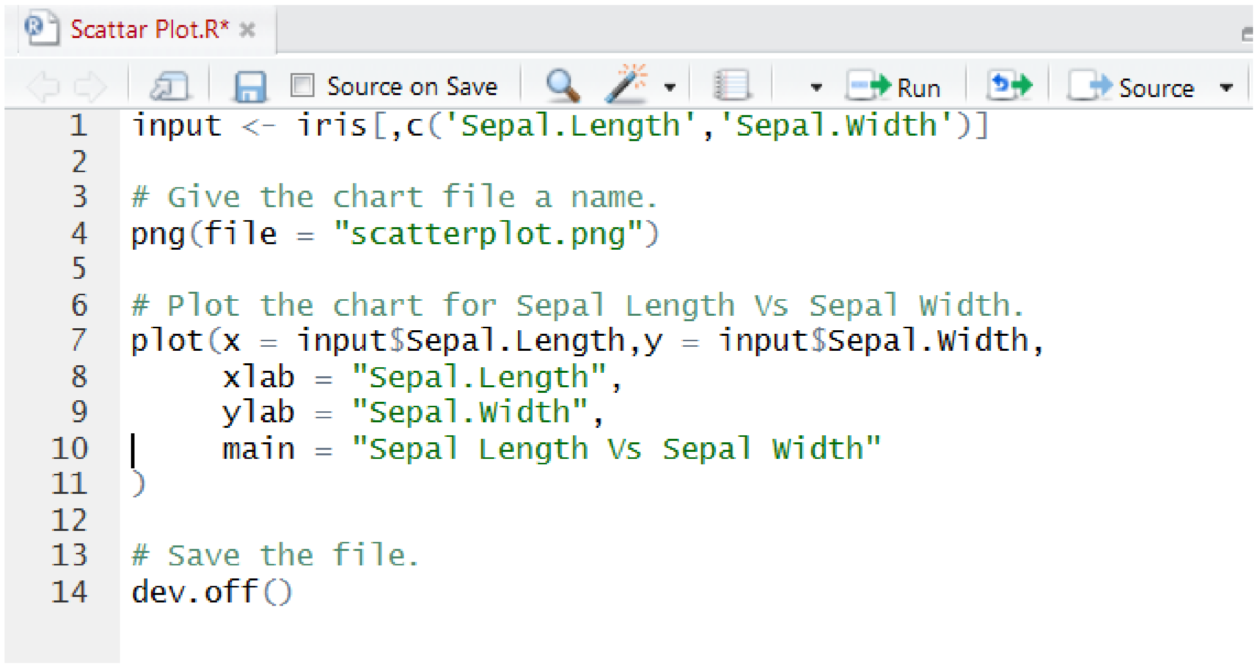
Example

We use the data set **"iris"** available in the R environment to create a basic scatter plot. Let's use the columns "Sepal length" and "Sepal Width" in iris.

Creating the Scatterplot

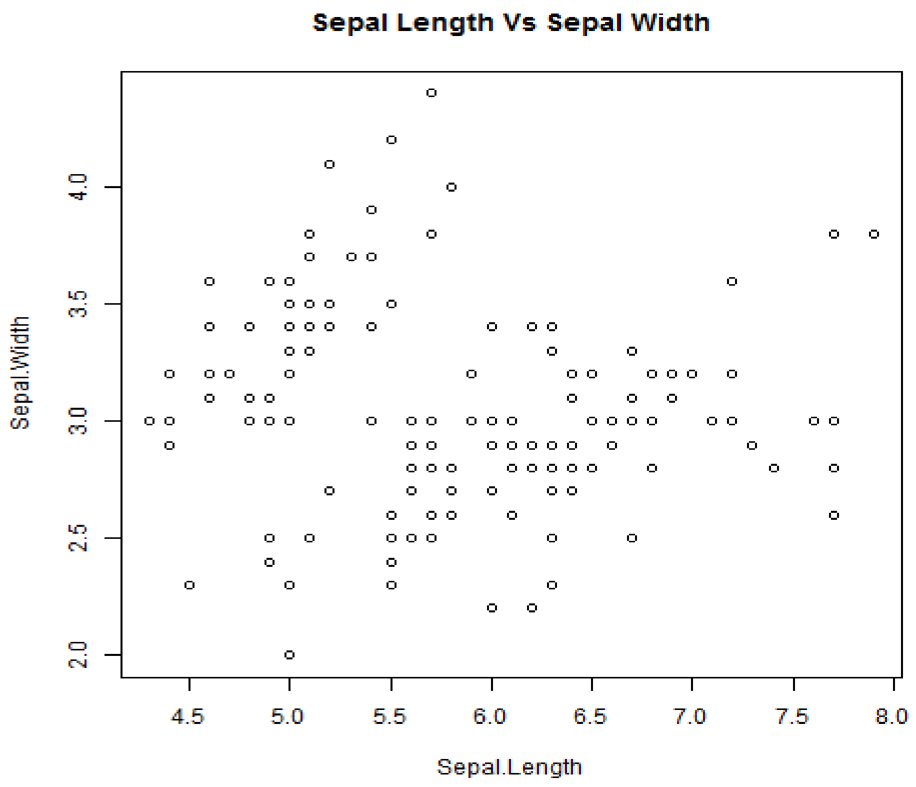
The below script will create a scatterplot graph for the relation between "Sepal length" and "Sepal Width" in iris dataset.

Example:



```
input <- iris[,c('Sepal.Length','Sepal.Width')]

# Give the chart file a name.
png(file = "scatterplot.png")

# Plot the chart for Sepal Length Vs Sepal Width.
plot(x = input$Sepal.Length,y = input$Sepal.Width,
     xlab = "Sepal.Length",
     ylab = "Sepal.Width",
     main = "Sepal Length Vs Sepal Width"
)

# Save the file.
dev.off()
```

Output:



Scatterplot Matrices

When we have more than two variables and we want to find the correlation between one variable versus the remaining ones we use scatterplot matrix. We use **pairs()** function to create matrices of scatterplots.

Syntax

The basic syntax for creating scatterplot matrices in R is −

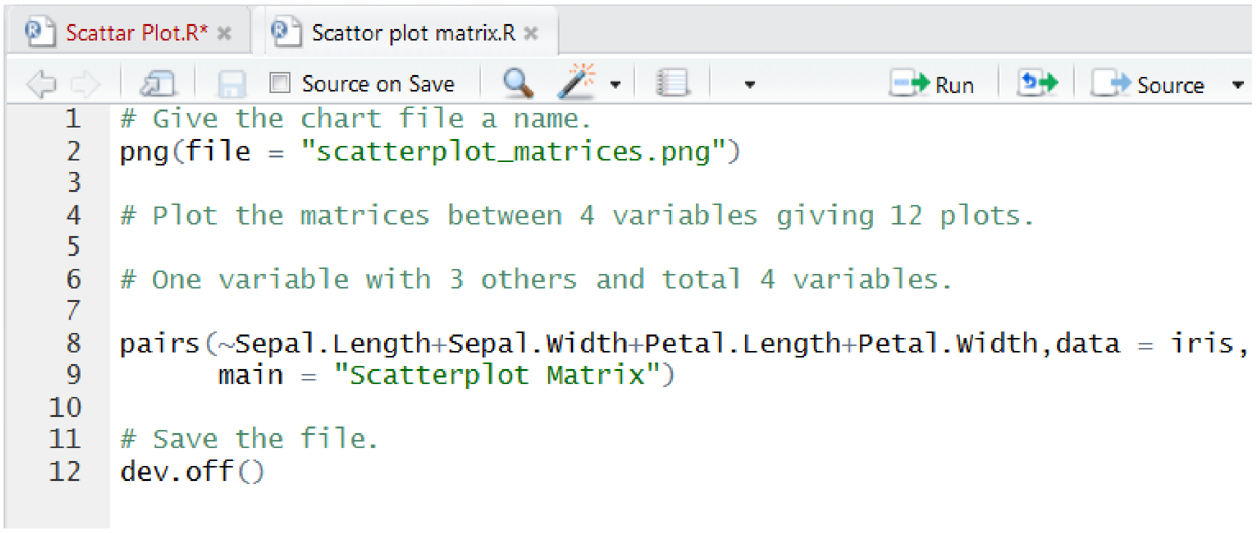
```
pairs(formula, data)
```

Following is the description of the parameters used −

- **formula** represents the series of variables used in pairs.
- **data** represents the data set from which the variables will be taken.

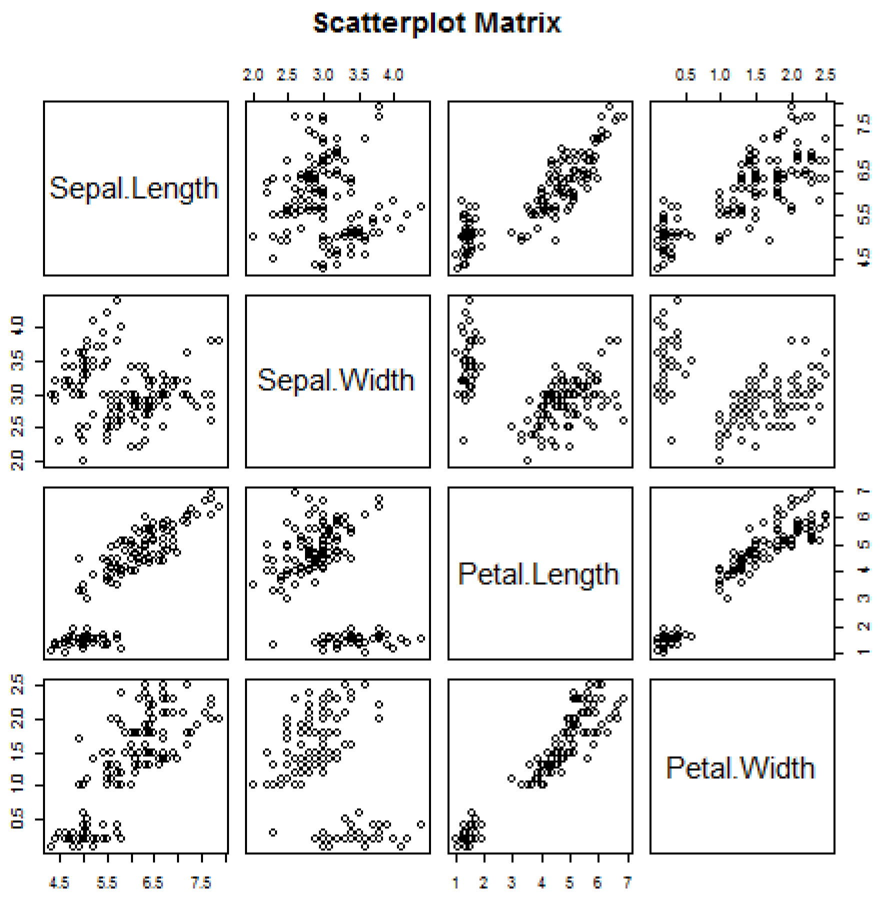
Example

Each variable is paired up with each of the remaining variable. A scatterplot is plotted for each pair.



```r
# Give the chart file a name.
png(file = "scatterplot_matrices.png")

# Plot the matrices between 4 variables giving 12 plots.

# One variable with 3 others and total 4 variables.

pairs(~Sepal.Length+Sepal.Width+Petal.Length+Petal.Width,data = iris,
      main = "Scatterplot Matrix")

# Save the file.
dev.off()
```

Output:



Box Plot

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

Boxplots are created in R by using the **boxplot()** function.

Syntax

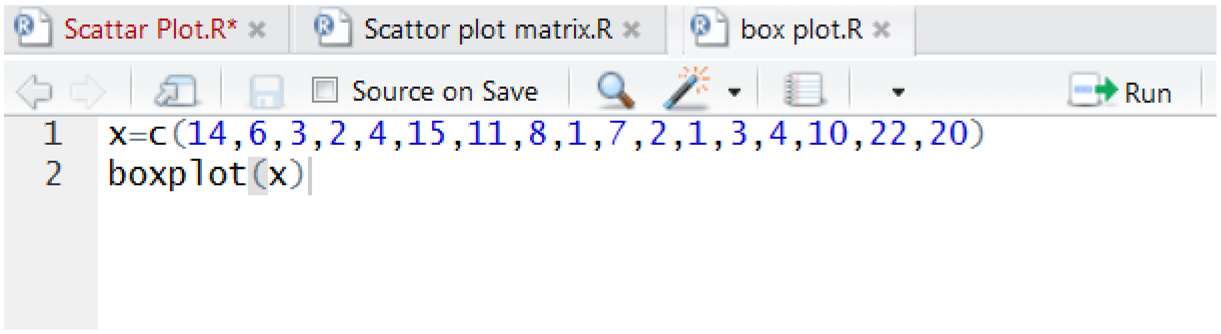
The basic syntax to create a boxplot in R is −

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

Following is the description of the parameters used −

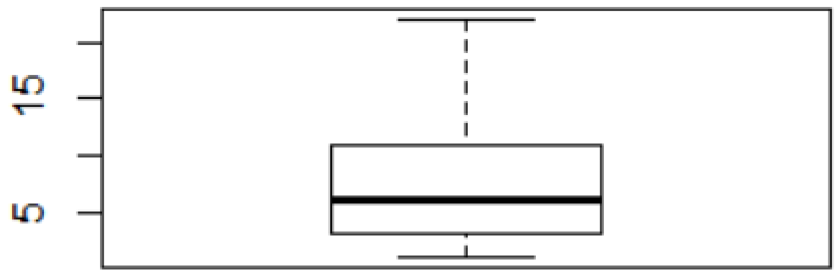
- **x** is a vector or a formula.
- **data** is the data frame.
- **notch** is a logical value. Set as TRUE to draw a notch.
- **varwidth** is a logical value. Set as true to draw width of the box proportionate to the sample size.
- **names** are the group labels which will be printed under each boxplot.
- **main** is used to give a title to the graph.

Example 1:



```
x=c(14,6,3,2,4,15,11,8,1,7,2,1,3,4,10,22,20)
boxplot(x)
```

Output:



Example 2:

boxplot(iris$Petal.Length,iris$Petal.Width)

Output: