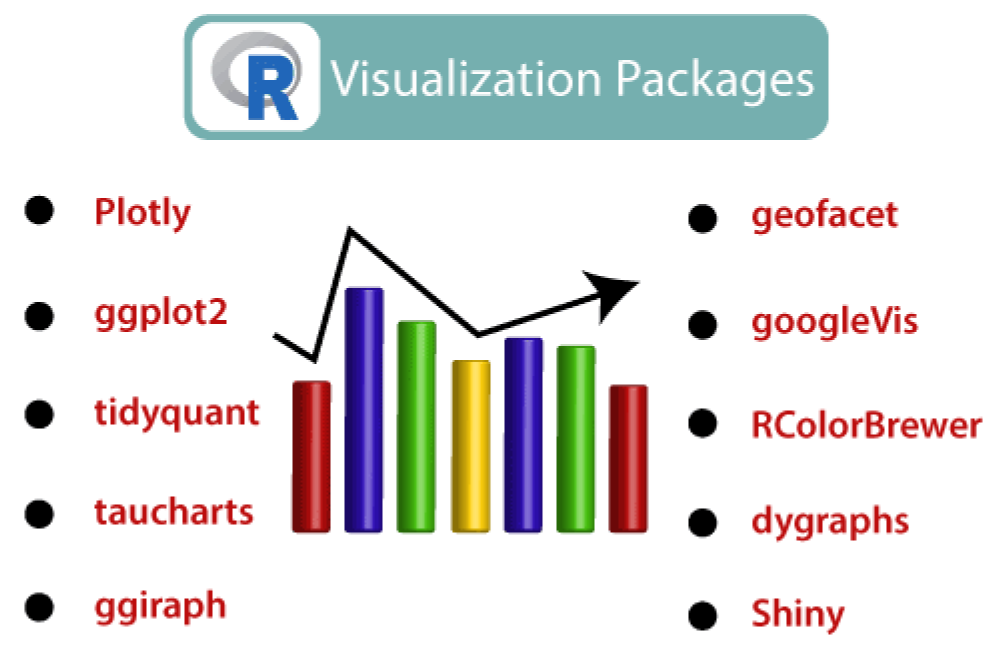
R Data Visualization

In R, we can create visually appealing data visualizations by writing few lines of code. For this purpose, we use the diverse functionalities of R. Data visualization is an efficient technique for gaining insight about data through a visual medium. With the help of visualization techniques, a human can easily obtain information about hidden patterns in data that might be neglected.

By using the data visualization technique, we can work with large datasets to efficiently obtain key insights about it.

R Visualization Packages

R provides a series of packages for data visualization. These packages are as follows:



1) plotly

The plotly package provides online interactive and quality graphs. This package extends upon the JavaScript library ?plotly.js.

2) ggplot2

R allows us to create graphics declaratively. R provides the **ggplot** package for this purpose. This package is famous for its elegant and quality graphs, which sets it apart from other visualization packages.

3) tidyquant

The **tidyquant** is a financial package that is used for carrying out quantitative financial analysis. This package adds under tidyverse universe as a financial package that is used for importing, analyzing, and visualizing the data.

4) taucharts

Data plays an important role in taucharts. The library provides a declarative interface for rapid mapping of data fields to visual properties.

5) ggiraph

It is a tool that allows us to create dynamic ggplot graphs. This package allows us to add tooltips, JavaScript actions, and animations to the graphics.

6) geofacets

This package provides geofaceting functionality for 'ggplot2'. Geofaceting arranges a sequence of plots for different geographical entities into a grid that preserves some of the geographical orientation.

7) googleVis

googleVis provides an interface between R and Google's charts tools. With the help of this package, we can create web pages with interactive charts based on R data frames.

8) RColorBrewer

This package provides color schemes for maps and other graphics, which are designed by Cynthia Brewer.

9) dygraphs

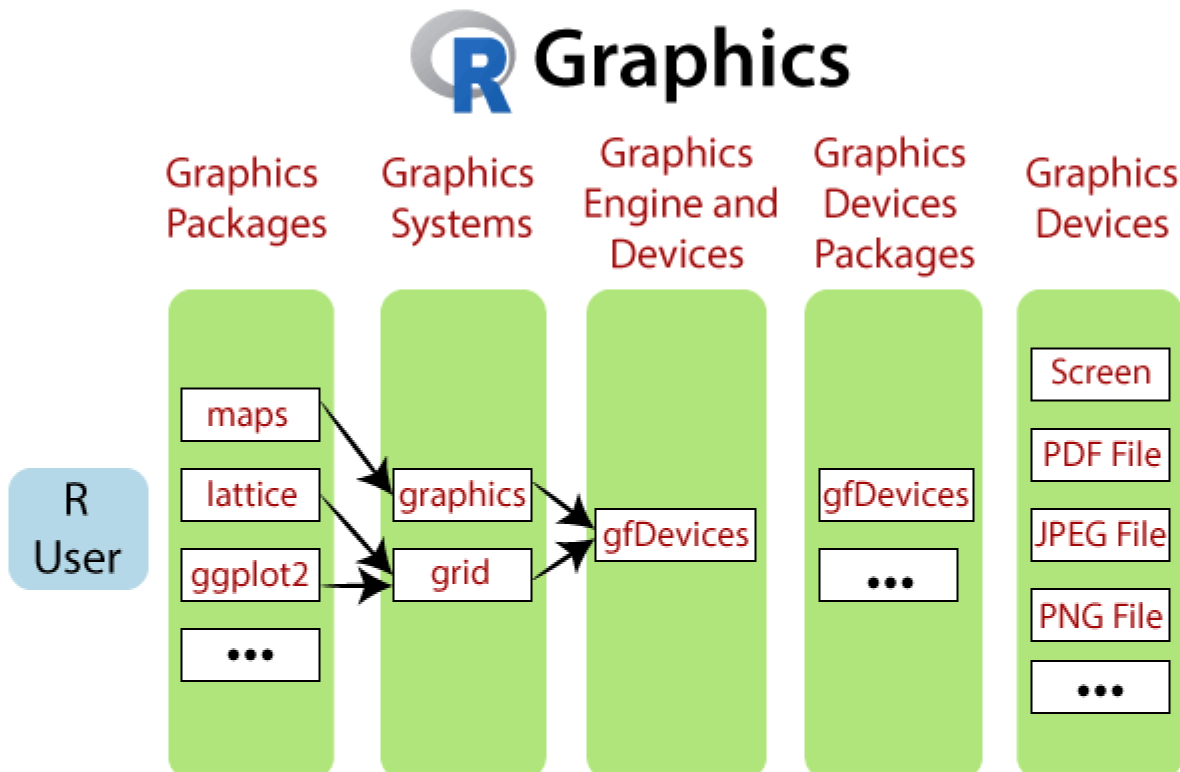
The dygraphs package is an R interface to the dygraphs JavaScript charting library. It provides rich features for charting time-series data in R.

10) shiny

R allows us to develop interactive and aesthetically pleasing web apps by providing a **shiny** package. This package provides various extensions with HTML widgets, CSS, and JavaScript.

R Graphics

Graphics play an important role in carrying out the important features of the data. Graphics are used to examine marginal distributions, relationships between variables, and summary of very large data. It is a very important complement for many statistical and computational techniques.



Standard Graphics

R standard graphics are available through package graphics, include several functions which provide statistical plots, like:

- Scatterplots
- Piecharts

- Boxplots
- Barplots etc.

We use the above graphs that are typically a single function call.

Graphics Devices

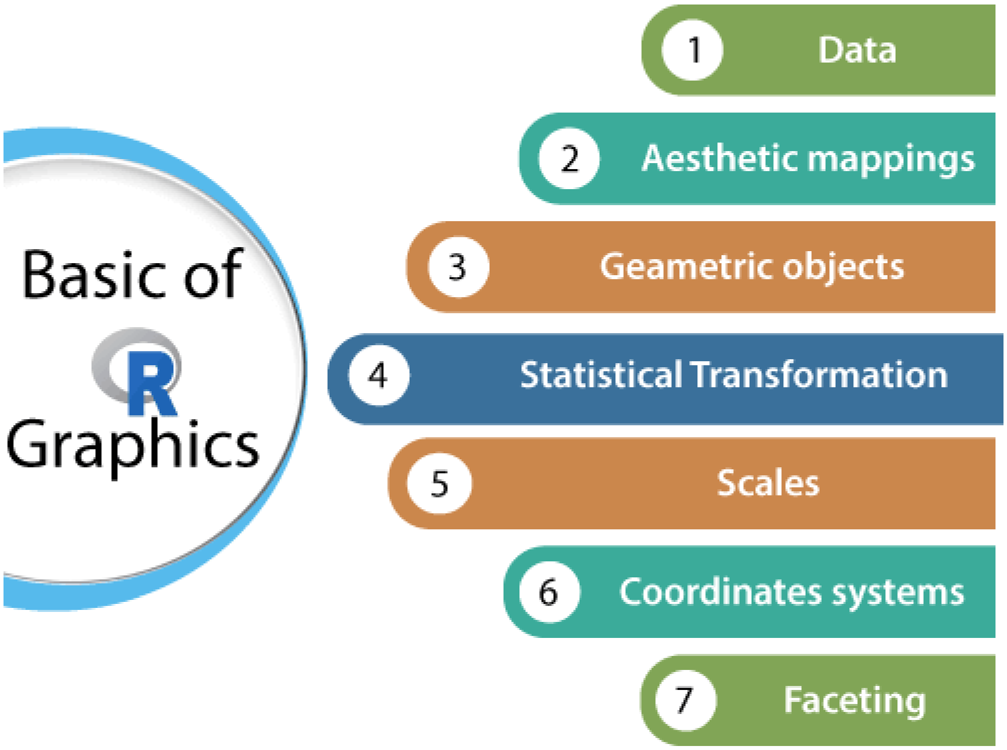
It is something where we can make a plot to appear. A graphics device is a window on your computer (screen device), a PDF file (file device), a Scalable Vector Graphics (SVG) file (file device), or a PNG or JPEG file (file device).

There are some of the following points which are essential to understand:

- The functions of graphics devices produce output, which depends on the active graphics device.
- A screen is the default and most frequently used device.
- R graphical devices such as the PDF device, the JPEG device, etc. are used.
- We just need to open the graphics output device which we want. Therefore, R takes care of producing the type of output which is required by the device.
- For producing a certain plot on the screen or as a GIF R graphics file, the R code should exactly be the same. We only need to open the target output device before.
- Several devices can be open at the same time, but there will be only one active device.

The basics of the grammar of graphics

There are some key elements of a statistical graphic. These elements are the basics of the grammar of graphics. Let's discuss each of the elements one by one to gain the basic knowledge of graphics.



1) Data

Data is the most crucial thing which is processed and generates an output.

2) Aesthetic Mappings

Aesthetic mappings are one of the most important elements of a statistical graphic. It controls the relation between graphics variables and data variables. In a scatter plot, it also helps to map the temperature variable of a data set into the X variable.

In graphics, it helps to map the species of a plant into the color of dots.

3) Geometric Objects

Geometric objects are used to express each observation by a point using the aesthetic mappings. It maps two variables in the data set into the x,y variables of the plot.

4) Statistical Transformations

Statistical transformations allow us to calculate the statistical analysis of the data in the plot.The statistical transformation uses the data and approximates it with the help of a regression line having x,y coordinates, and counts occurrences of certain values.

5) Scales

It is used to map the data values into values present in the coordinate system of the graphics device.

6) Coordinate system

The coordinate system plays an important role in the plotting of the data.

- Cartesian
- Plot

7) Faceting

Faceting is used to split the data into subgroups and draw sub-graphs for each group.

Advantages of Data Visualization in R

1. Understanding

It can be more attractive to look at the business. And, it is easier to understand through graphics and charts than a written document with text and numbers. Thus, it can attract a wider range of audiences. Also, it promotes the widespread use of business insights that come to make better decisions.

2. Efficiency

Its applications allow us to display a lot of information in a small space. Although, the decision-making process in business is inherently complex and multifunctional, displaying evaluation findings in a graph can allow companies to organize a lot of interrelated information in useful ways.

3. Location

Its app utilizing features such as Geographic Maps and GIS can be particularly relevant to wider business when the location is a very relevant factor. We will use maps to show business insights from various locations, also consider the seriousness of the issues, the reasons behind them, and working groups to address them.

Disadvantages of Data Visualization in R

1. Cost

R application development range a good amount of money. It may not be possible, especially for small companies, that many resources can be spent on purchasing them. To generate reports, many companies may employ professionals to create charts that can increase costs. Small enterprises are often operating in resource-limited settings, and are also receiving timely evaluation results that can often be of high importance.

2. Distraction

However, at times, data visualization apps create highly complex and fancy graphics-rich reports and charts, which may entice users to focus more on the form than the function. If we first add visual appeal, then the overall value of the graphic representation will be minimal. In resource-setting, it is required to understand how resources can be best used. And it is also not caught in the graphics trend without a clear purpose.

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:

```
1. pie(X, Labels, Radius, Main, Col, Clockwise)
```

Here,

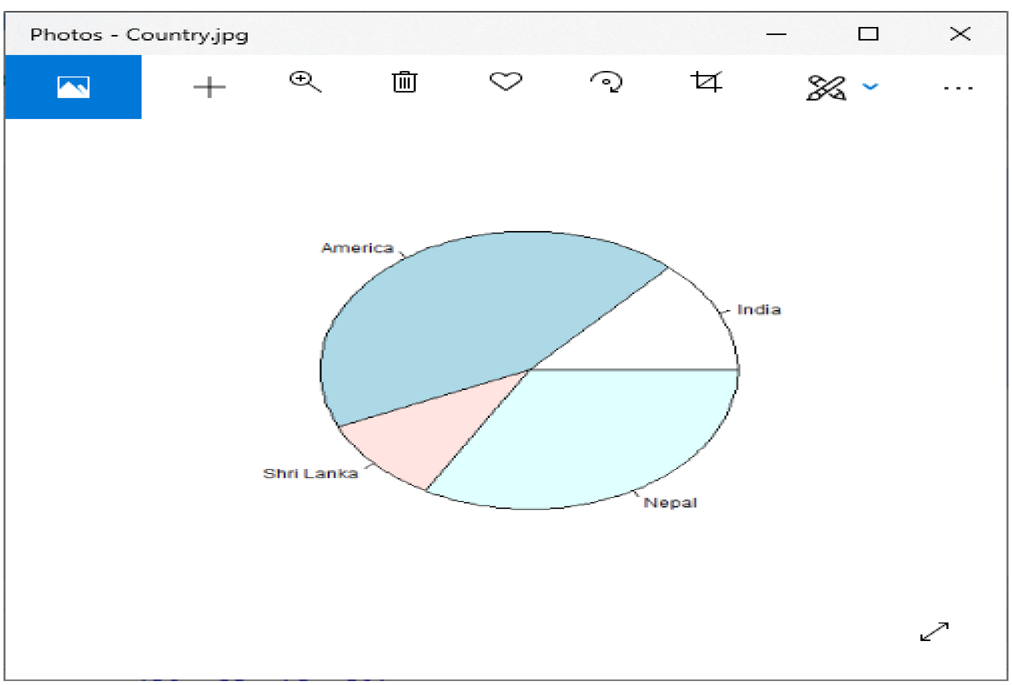
1. **X** is a vector that contains the numeric values used in the pie chart.

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)
3. labels <- c("India", "America", "Shri Lanka", "Nepal")
4. # Giving the chart file a name.
5. png(file = "Country.jpg")
6. # Plotting the chart.
7. pie(x,labels)
8. # Saving the file.
9. 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.

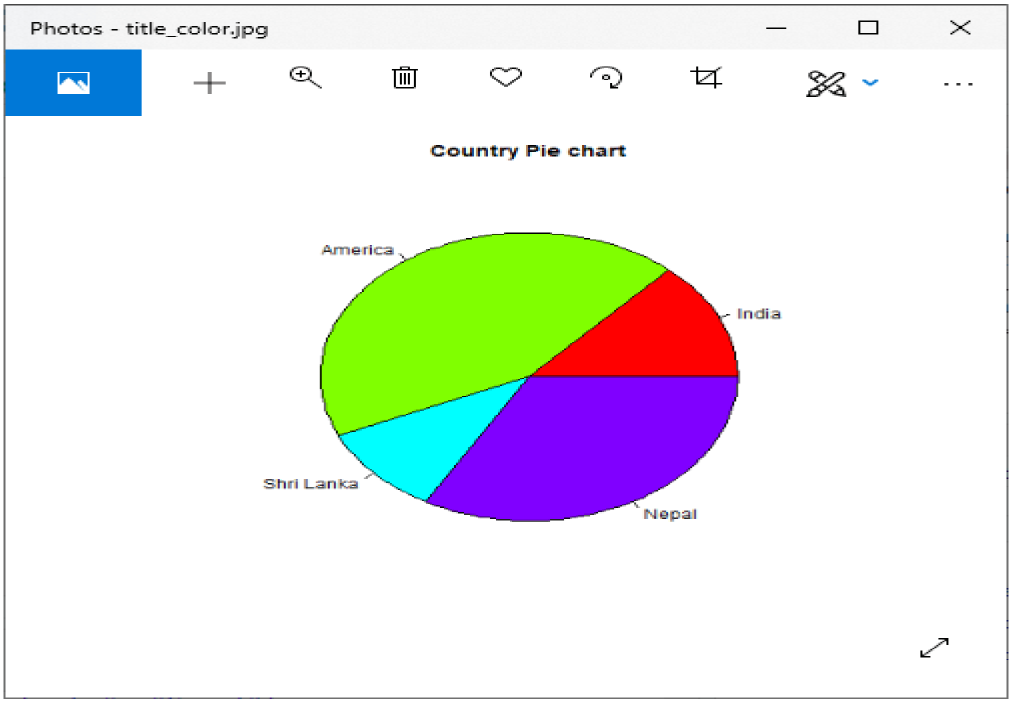
> Note: ***The length of the pallet will be the same as the number of values that we have for the chart. So for that, we will use length() function.***

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

Example

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

Output:



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.

```
1. legend(x,y=NULL,legend,fill,col,bg)
```

Here,

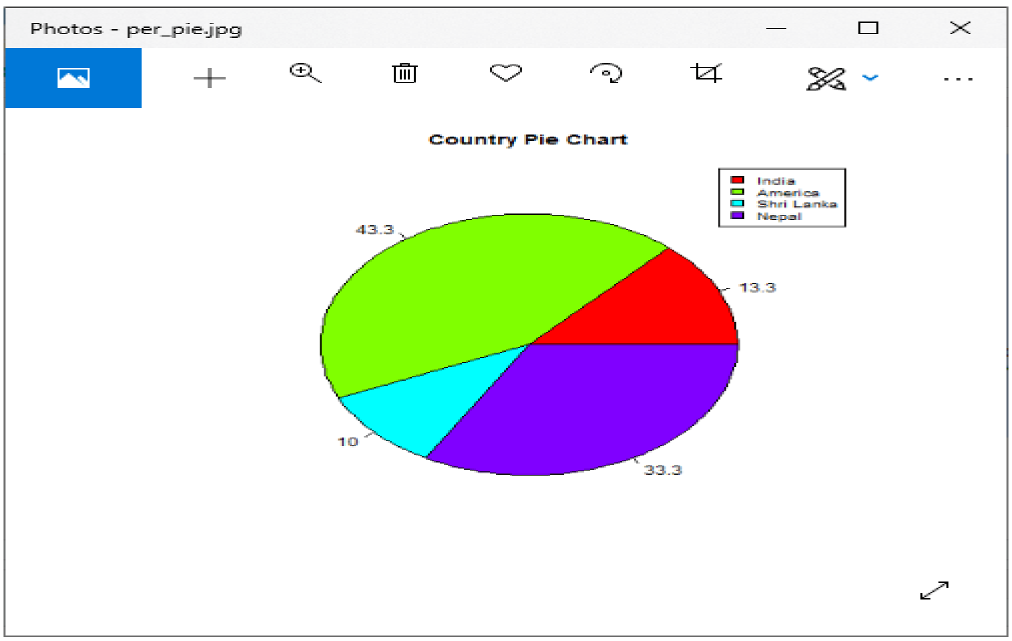
- x and y are the coordinates to be used to position the legend.
- legend is the text of legend
- fill is the color to use for filling the boxes beside the legend text.
- col defines the color of line and points besides the legend text.
- bg is the background color for the legend box.

Example

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

4. pie_percent<- round(100*x/sum(x), 1)
5. # Giving the chart file a name.
6. png(file = "per_pie.jpg")
7. # Plotting the chart.
8. pie(x, labels = pie_percent, main = "Country Pie Chart",col = rainbow(length(x)))
9. legend("topright", c("India", "America", "Shri Lanka", "Nepal"), cex = 0.8,
10. fill = rainbow(length(x)))
11. #Saving the file.
12. dev.off()

Output:



3 Dimensional Pie Chart

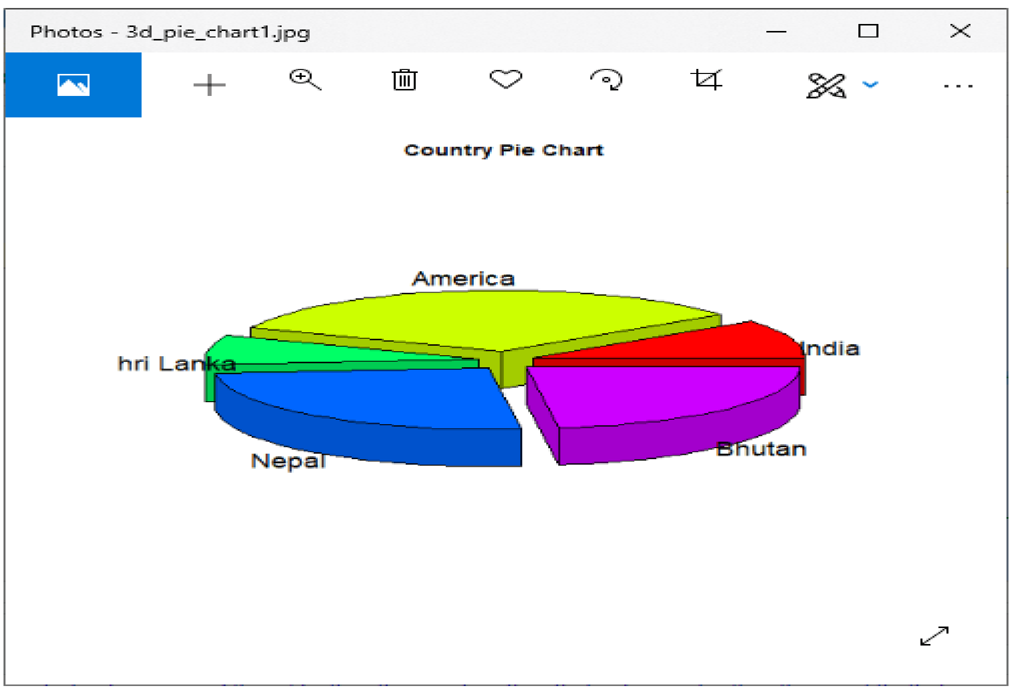
In R, we can also create a three-dimensional pie chart. For this purpose, R provides a plotrix package whose pie3D() function is used to create an attractive 3D pie chart. The parameters of pie3D() function remain same as pie() function. Let's see an example to understand how a 3D pie chart is created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

Example

1. # 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")
# Give the chart file a name.
png(file = "3d_pie_chart1.jpg")
# Plot the chart.
pie3D(x,labelslabels = labels,explode = 0.1, main = "Country Pie Chart")
# Save the file.
dev.off()
```

Output:

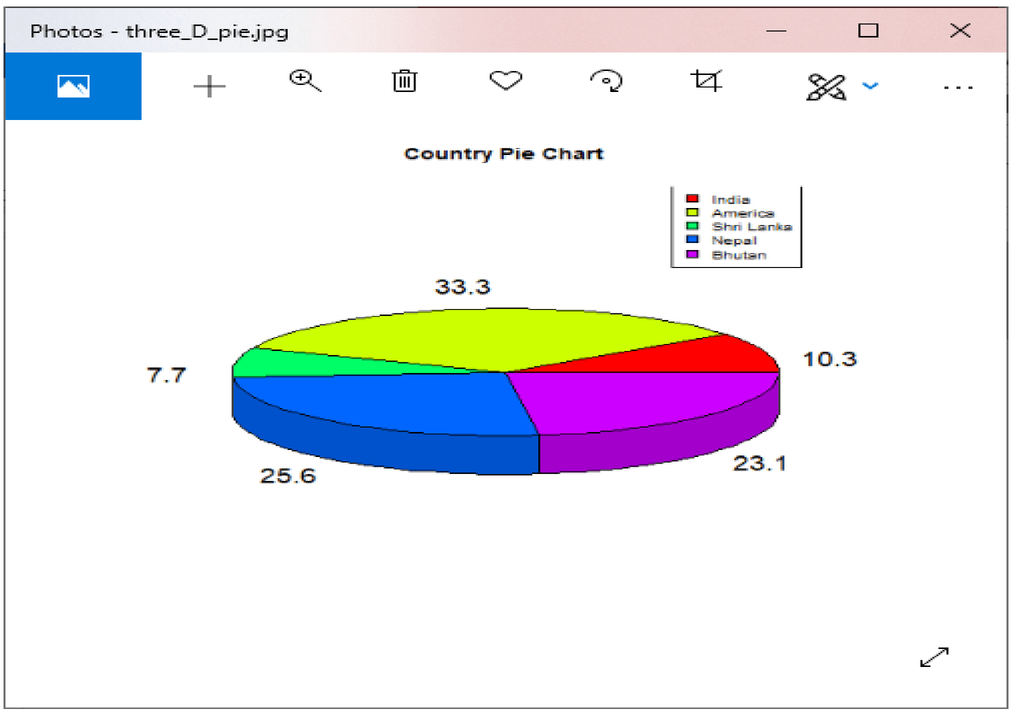


Example

```
# 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)
# Giving the chart file a name.
png(file = "three_D_pie.jpg")
# 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)))
#Saving the file.
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:

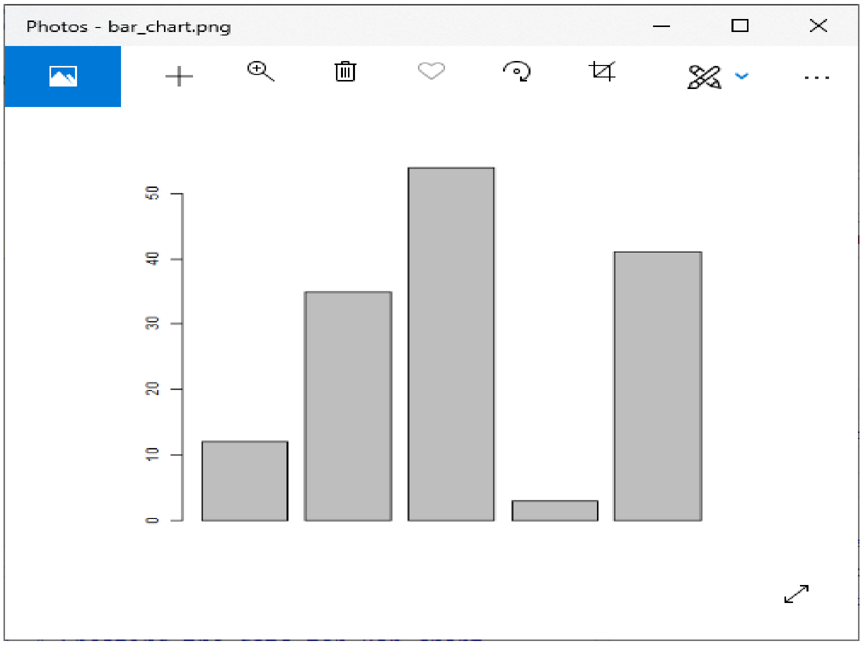
1. barplot(h,x,y,main, names.arg,col)

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

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

Output:



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

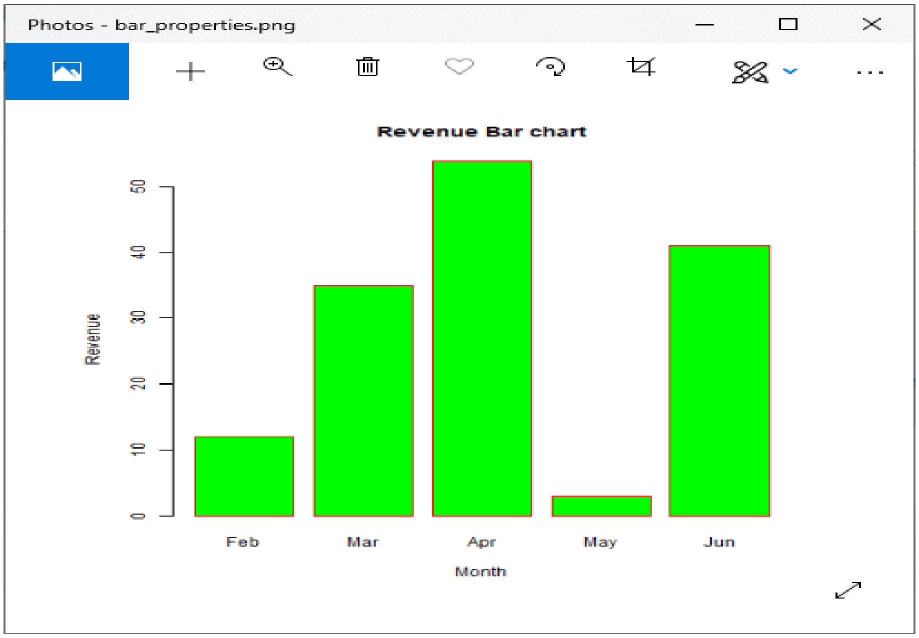
```
# Creating the data for Bar chart
H <- c(12,35,54,3,41)
M<- c("Feb","Mar","Apr","May","Jun")

# 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",
```

10.   main="Revenue Bar chart",border="red")
11. # Saving the file
12. dev.off()

Output:



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.

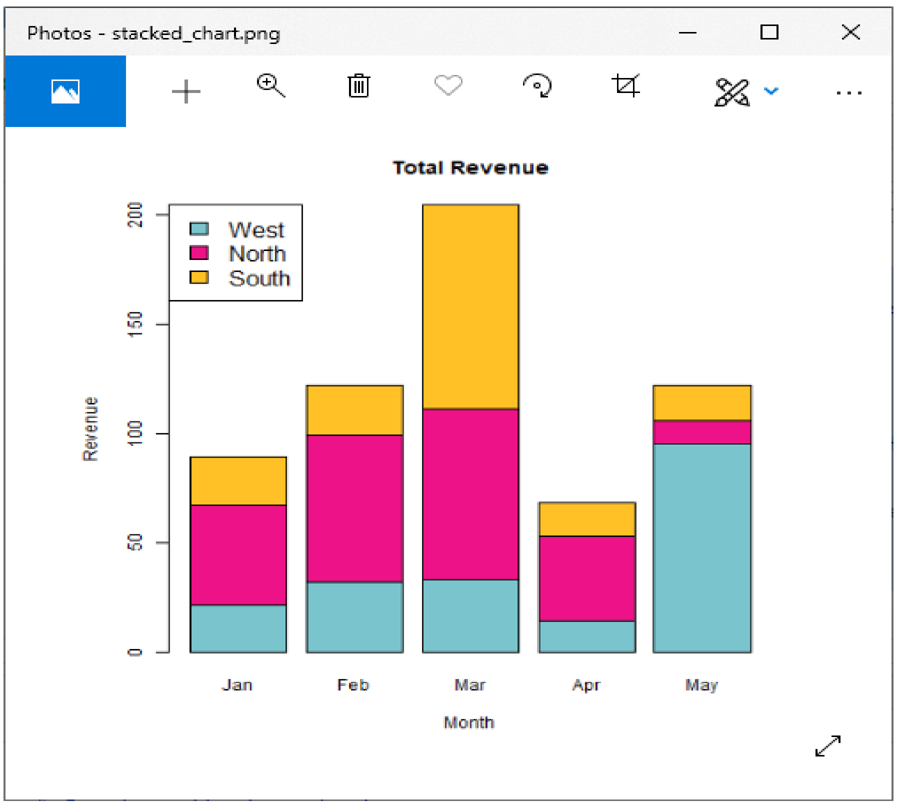
Let's see an example to understand how these charts are created.

Example

1. library(RColorBrewer)
2. months <- c("Jan","Feb","Mar","Apr","May")
3. regions <- c("West","North","South")
4. # Creating the matrix of the values.
5. Values <- matrix(c(21,32,33,14,95,46,67,78,39,11,22,23,94,15,16), nrow = 3, ncol = 5, byrow = TRUE)
6. # Giving the chart file a name

7. png(file = "stacked_chart.png")
8. # Creating the bar chart
9. barplot(Values, main = "Total Revenue", names.arg = months, xlab = "Month", ylab = "Revenue", ccol =c("cadetblue3","deeppink2","goldenrod1"))
10. # Adding the legend to the chart
11. legend("topleft", regions, cex = 1.3, fill = c("cadetblue3","deeppink2","goldenrod1"))
12.
13. # Saving the file
14. 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:

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

Here,

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.

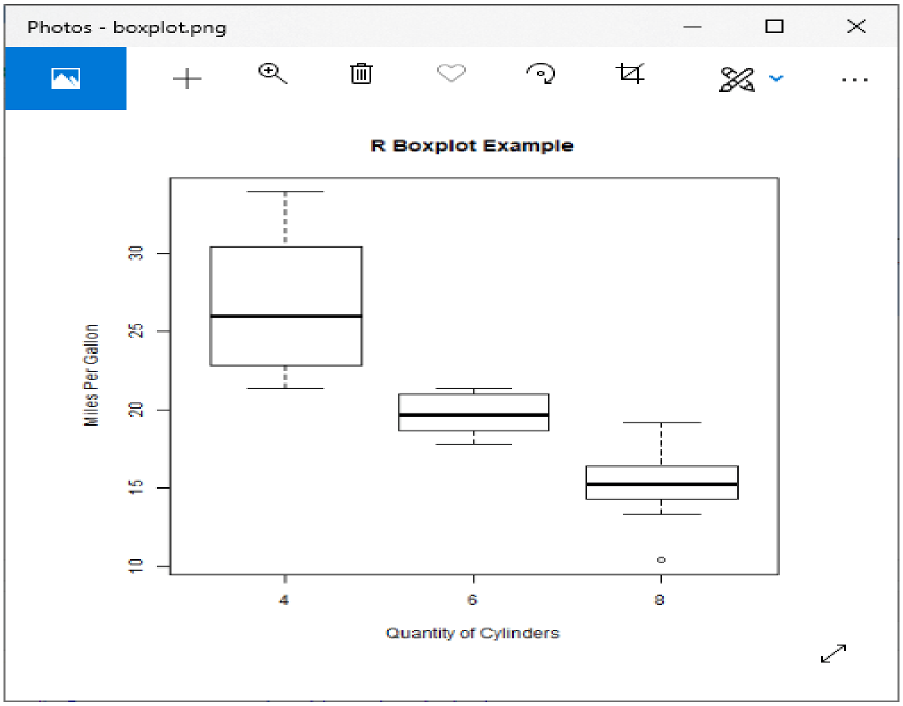
Let?s see an example to understand how we can create a boxplot in R. In the below example, we will use the "mtcars" dataset present in the R environment. We will use its two columns only, i.e., "mpg" and "cyl". The below example will create a boxplot graph for the relation between mpg and cyl, i.e., miles per gallon and number of cylinders, respectively.

Example

```
# Giving a name to the chart file.
png(file = "boxplot.png")
# Plotting the chart.
boxplot(mpg ~ cyl, data = mtcars, xlab = "Quantity of Cylinders",
        ylab = "Miles Per Gallon", main = "R Boxplot Example")

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

Output:



Boxplot using notch

In R, we can draw a boxplot using a notch. It helps us to find out how the medians of different data groups match with each other. Let's see an example to understand how a boxplot graph is created using notch for each of the groups.

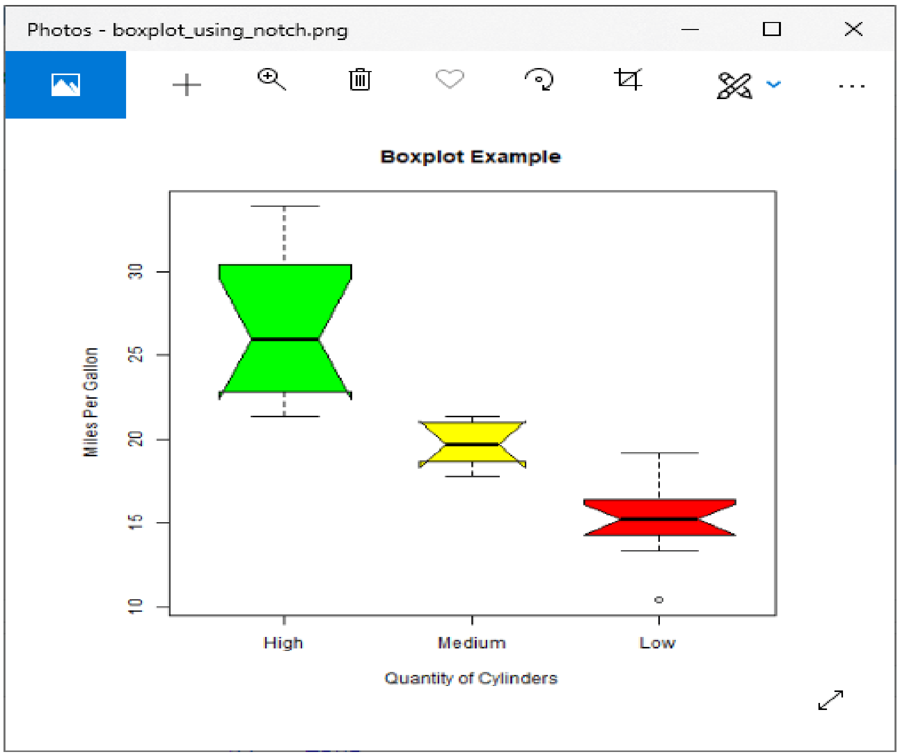
In our below example, we will use the same dataset ?mtcars."

Example

```
1. # Giving a name to our chart.
2. png(file = "boxplot_using_notch.png")
3. # Plotting the chart.
4. boxplot(mpg ~ cyl, data = mtcars,
5.         xlab = "Quantity of Cylinders",
6.         ylab = "Miles Per Gallon",
7.         main = "Boxplot Example",
8.         notch = TRUE,
9.         varwidth = TRUE,
```

```
    ccol = c("green","yellow","red"),
    names = c("High","Medium","Low")
)
# Saving the file.
dev.off()
```

Output:



Violin Plots

R provides an additional plotting scheme which is created with the combination of a **boxplot** and a **kernel density** plot. The violin plots are created with the help of vioplot() function present in the vioplot package.

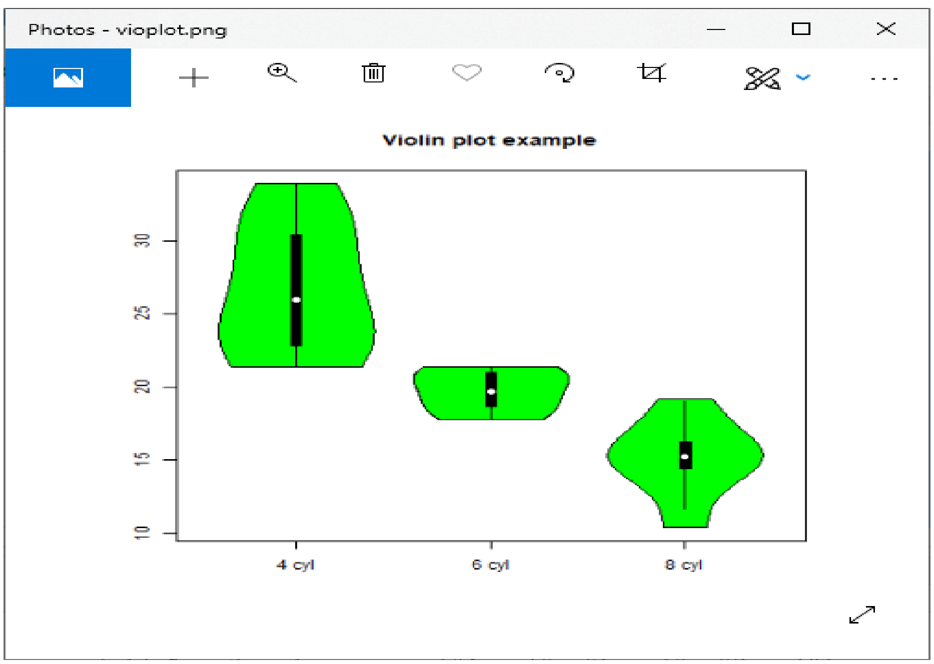
Let's see an example to understand the creation of the violin plot.

Example

```
# Loading the vioplot package
library(vioplot)
```

3. # Giving a name to our chart.
4. png(file = "vioplot.png")
5. #Creating data for vioplot function
6. x1 <- mtcars$mpg[mtcars$cyl==4]
7. x2 <- mtcars$mpg[mtcars$cyl==6]
8. x3 <- mtcars$mpg[mtcars$cyl==8]
9. #Creating vioplot function
10. vioplot(x1, x2, x3, names=c("4 cyl", "6 cyl", "8 cyl"),
11.   col="green")
12. #Setting title
13. title("Violin plot example")
14. # Saving the file.
15. dev.off()

Output:



Bagplot- 2-Dimensional Boxplot Extension

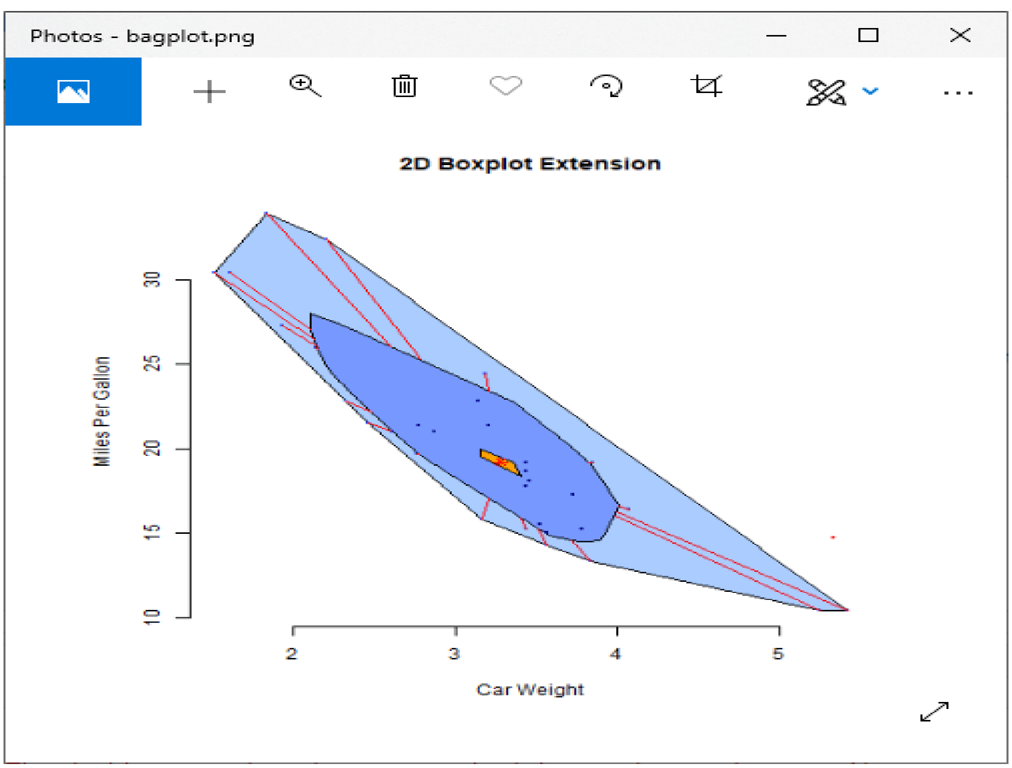
The bagplot(x, y) function in the **aplpack** package provides a biennial version of the univariate boxplot. The bag contains 50% of all points. The bivariate median is approximate. The fence separates itself from the outside points, and the outlays are displayed.

Let?s see an example to understand how we can create a two-dimensional boxplot extension in R.

Example

```
1. # Loading aplpack package
2. library(aplpack)
3. # Giving a name to our chart.
4. png(file = "bagplot.png")
5. #Creating bagplot function
6. attach(mtcars)
7. bagplot(wt,mpg, xlab="Car Weight", ylab="Miles Per Gallon",
8.   main="2D Boxplot Extension")
9. # Saving the file.
10. dev.off()
```

Output:



R Histogram

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:

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

Here,

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.

Let?s see an example in which we create a simple histogram with the help of required parameters like v, main, col, etc.

Example

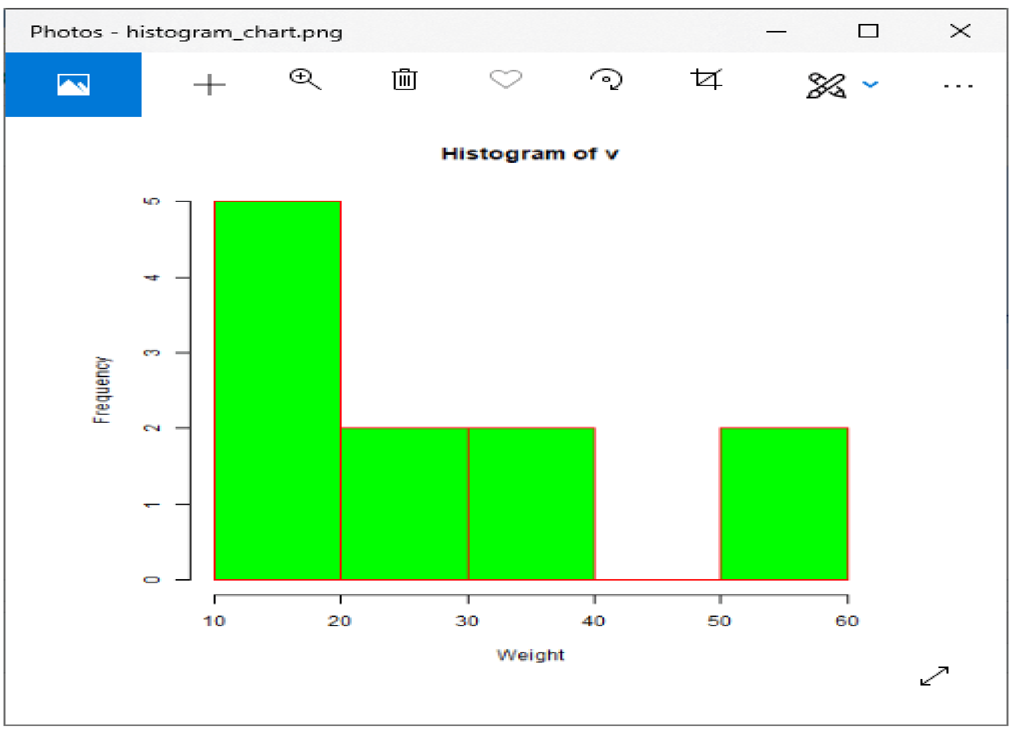
```
# Creating data for the graph.
v <- c(12,24,16,38,21,13,55,17,39,10,60)

# Giving a name to the chart file.
png(file = "histogram_chart.png")

# Creating the histogram.
hist(v,xlab = "Weight",ylab="Frequency",col = "green",border = "red")

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

Output:



Let?s see some more examples in which we have used different parameters of hist() function to add more functionality or to create a more attractive chart.

Example: Use of xlim & ylim parameter

```
# Creating data for the graph.
```

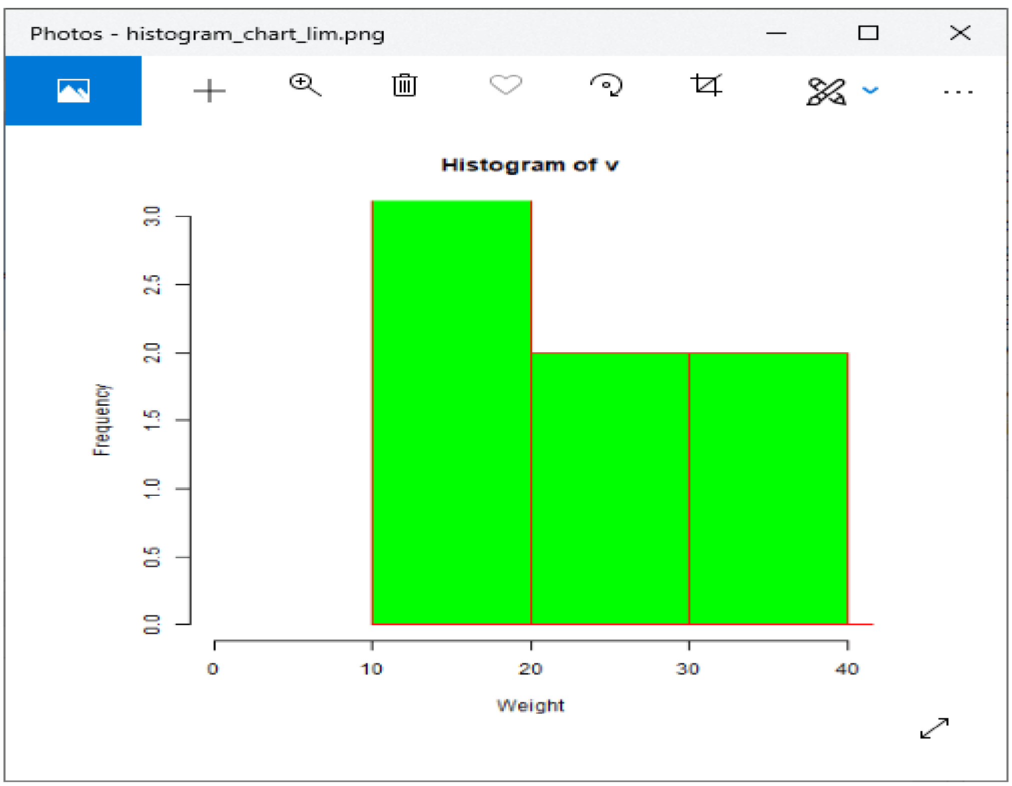
```
v <- c(12,24,16,38,21,13,55,17,39,10,60)

# Giving a name to the chart file.
png(file = "histogram_chart_lim.png")

# Creating the histogram.
hist(v,xlab = "Weight",ylab="Frequency",col = "green",border = "red",xlim = c(0,40), ylim
 = c(0,3), breaks = 5)

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

Output:



Example: Finding return value of hist()

```
# Creating data for the graph.
v <- c(12,24,16,38,21,13,55,17,39,10,60)

```

```
# Giving a name to the chart file.
png(file = "histogram_chart_lim.png")
# Creating the histogram.
m<-hist(v)
m
```

Output:

```
Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript histo.R
$breaks
[1] 10 20 30 40 50 60

$counts
[1] 5 2 2 0 2

$density
[1] 0.04545455 0.01818182 0.01818182 0.00000000 0.01818182

$mids
[1] 15 25 35 45 55

$xname
[1] "v"

$equidist
[1] TRUE

attr(,"class")
[1] "histogram"

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

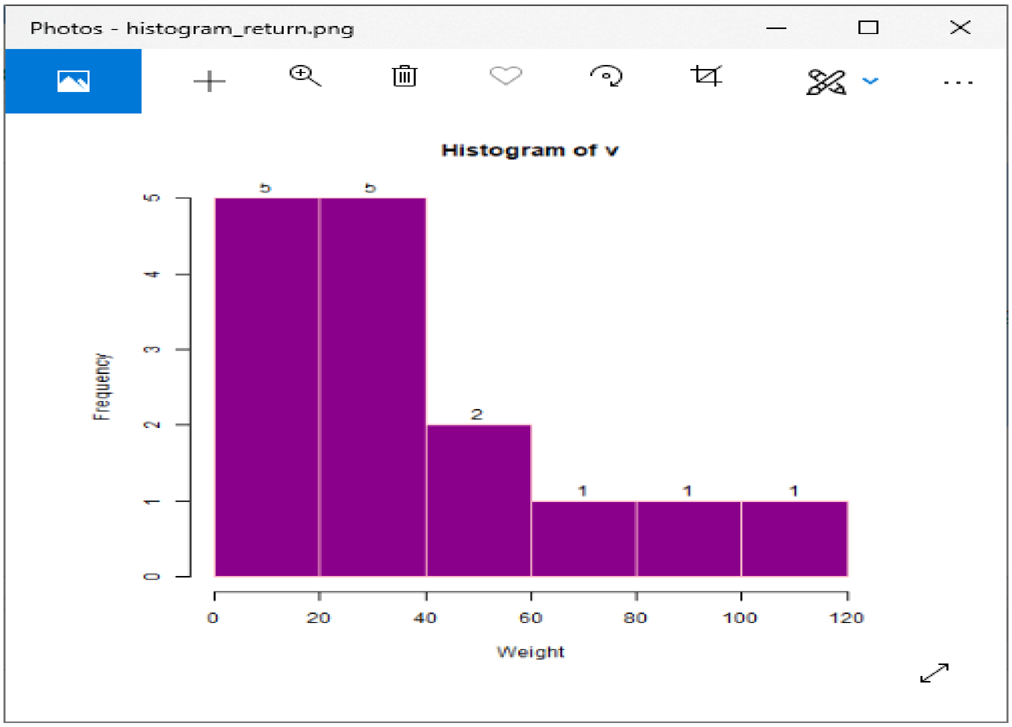
Example: Using histogram return values for labels using text()

```
# Creating data for the graph.
v <-  c(12,24,16,38,21,13,55,17,39,10,60,120,40,70,90)
# Giving a name to the chart file.
png(file = "histogram_return.png")

# Creating the histogram.
```

7. m<-
hist(v,xlab = "Weight",ylab="Frequency",col = "darkmagenta",border = "pink", breaks = 5)
8. #Setting labels
9. text(m$mids,m$counts,labels=m$counts, adj=c(0.5, -0.5))
10. # Saving the file.
11. dev.off()

Output:

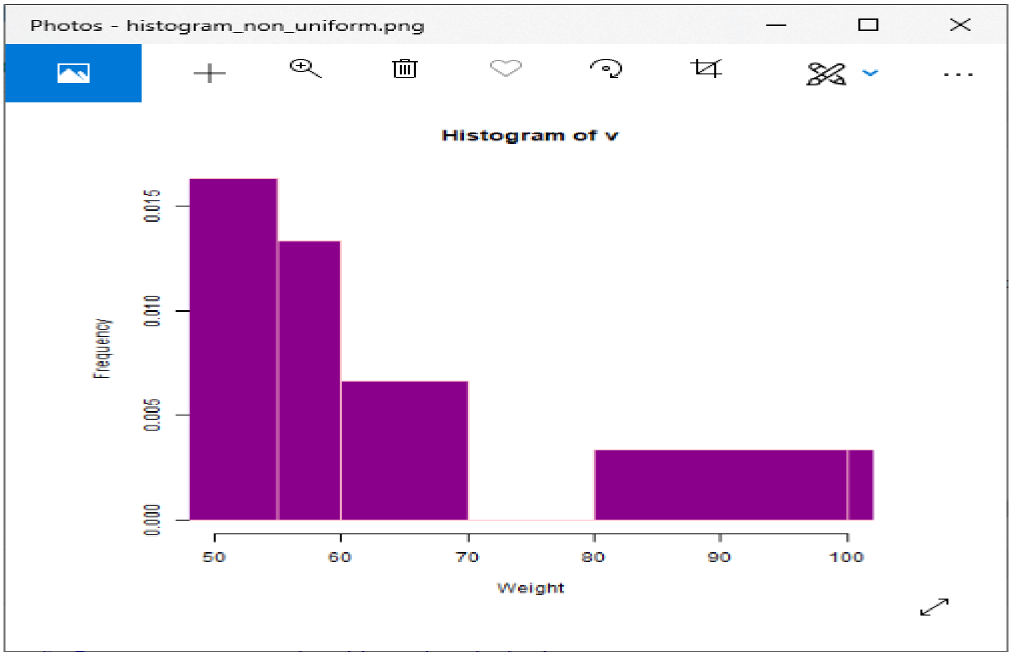


Example: Histogram using non-uniform width

1. # Creating data for the graph.
2. v <- c(12,24,16,38,21,13,55,17,39,10,60,120,40,70,90)
3. # Giving a name to the chart file.
4. png(file = "histogram_non_uniform.png")
5. # Creating the histogram.
6. hist(v,xlab = "Weight",ylab="Frequency",xlim=c(50,100),col = "darkmagenta",border = "pink", breaks=c(10,55,60,70,75,80,100,120))
7. # Saving the file.

8. 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:

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

Here,

S.No	Parameter	Description

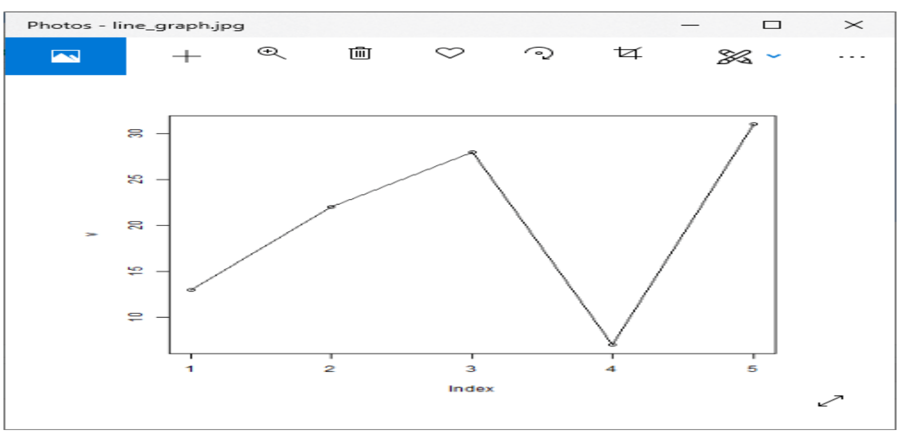
1.	v	It is a vector which contains the numeric values.
2.	type	This parameter takes the value ?I? to draw only the lines or ?p? to draw only the points and "o" to draw both lines and points.
3.	xlab	It is the label for the x-axis.
4.	ylab	It is the label for the y-axis.
5.	main	It is the title of the chart.
6.	col	It is used to give the color for both the points and lines

Let?s see a basic example to understand how plot() function is used to create the line graph:

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:



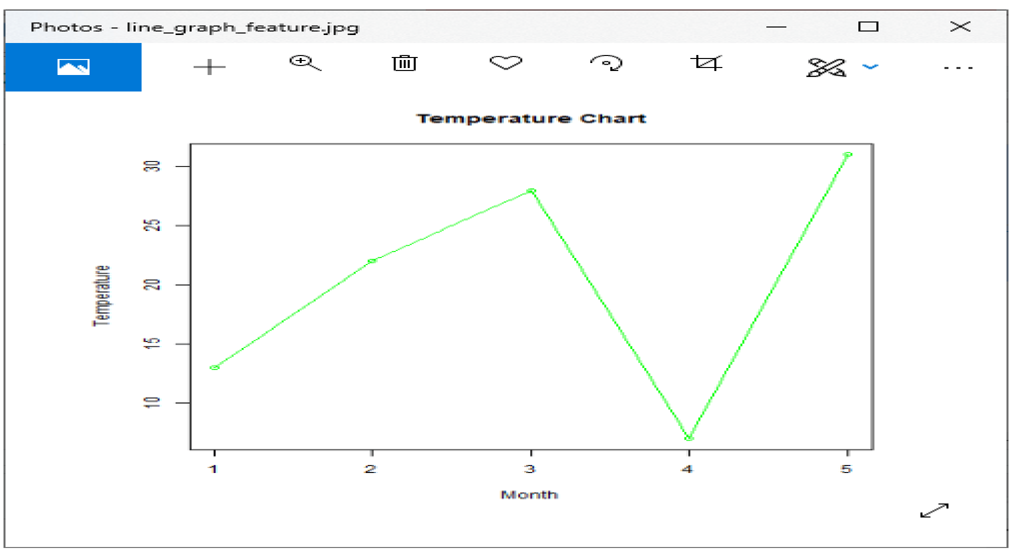
Line Chart Title, Color, and Labels

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:



Line Charts Containing Multiple Lines

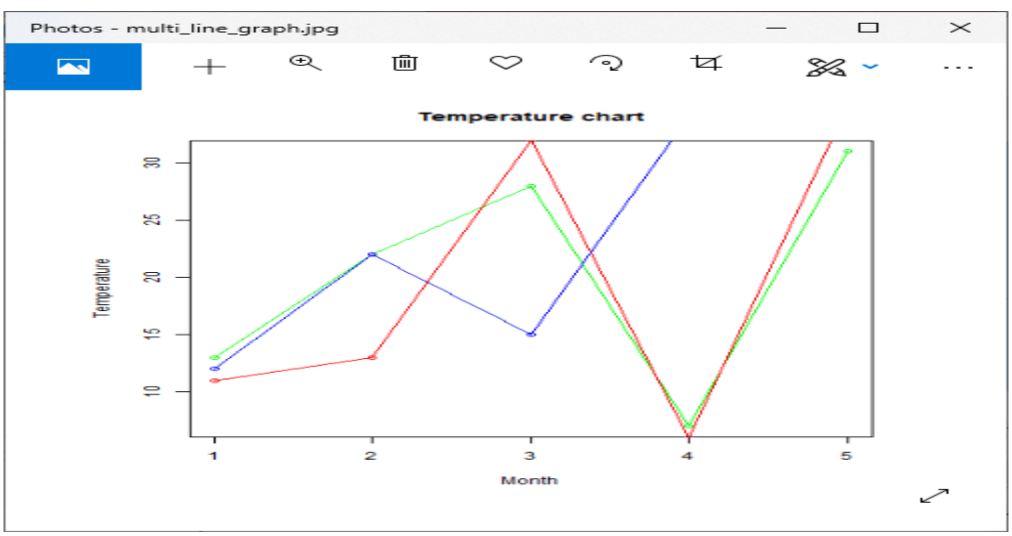
In our previous examples, we created line graphs containing only one line in each graph. R allows us to create a line graph containing multiple lines. R provides lines() function to create a line in the line graph.

The lines() function takes an additional input vector for creating a line. Let?s see an example to understand how this function is used:

Example

```
1. # Creating the data for the chart.
2. v <- c(13,22,28,7,31)
3. w <- c(11,13,32,6,35)
4. x <- c(12,22,15,34,35)
5. # Giving a name to the chart file.
6. png(file = "multi_line_graph.jpg")
7. # Plotting the bar chart.
8. plot(v,type = "o",col="green",xlab="Month",ylab="Temperature")
9. lines(w, type = "o", col = "red")
10. lines(x, type = "o", col = "blue")
11. # Saving the file.
12. dev.off()
```

Output:



Line Graph using ggplot2

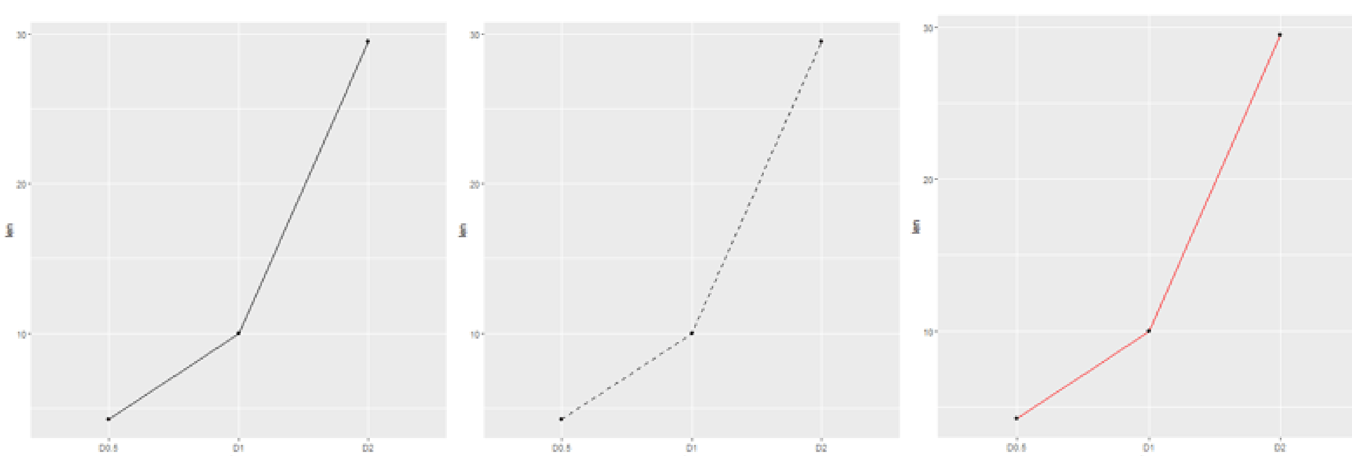
In R, there is another way to create a line graph i.e. the use of ggplot2 packages. The ggplot2 package provides geom_line(), geom_step() and geom_path() function to create line graph. To use these functions, we first have to install the ggplot2 package and then we load it into the current working library.

Let?s see an example to understand how ggplot2 is used to create a line graph. In the below example, we will use the predefined ToothGrowth dataset, which describes the effect of vitamin C on tooth growth in Guinea pigs.

Example

```
library(ggplot2)
#Creating data for the graph
data_frame<- data.frame(dose=c("D0.5", "D1", "D2"),
len=c(4.2, 10, 29.5))
head(data_frame)
png(file = "multi_line_graph2.jpg")
# Basic line plot with points
ggplot(data=data_frame, aes(x=dose, y=len, group=1)) +geom_line()+geom_point()
# Change the line type
ggplot(data=df, aes(x=dose, y=len, group=1)) +geom_line(linetype = "dashed")+geom_point()
# Change the color
ggplot(data=df, aes(x=dose, y=len, group=1)) +geom_line(color="red")+geom_point()
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:

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

Here,

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.

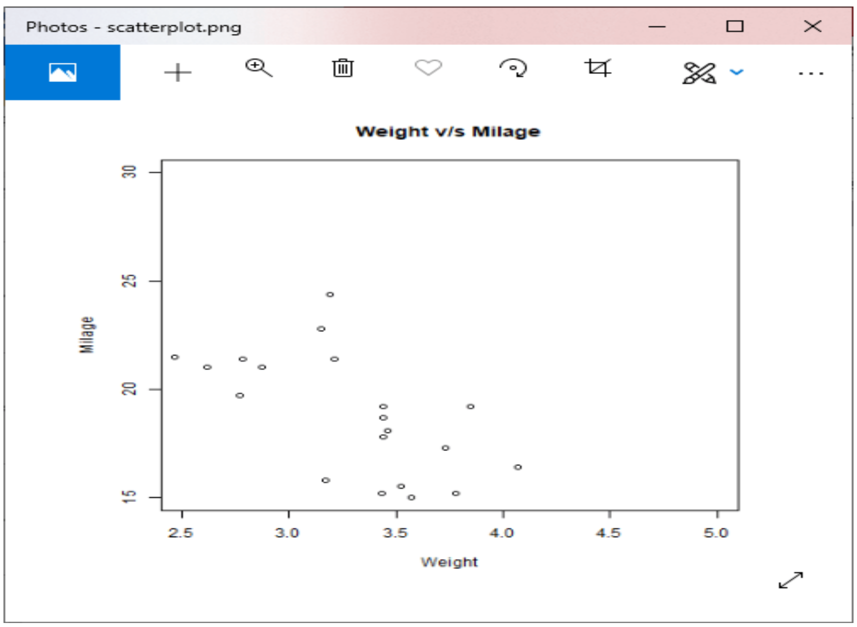
Let's see an example to understand how we can construct a scatterplot using the plot function. In our example, we will use the dataset "mtcars", which is the predefined dataset available in the R environment.

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.

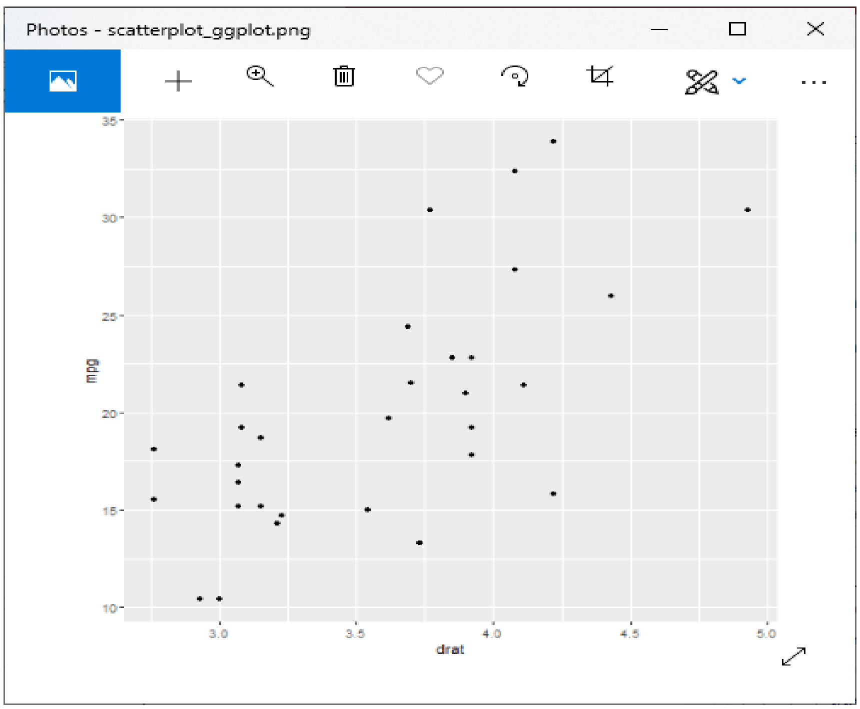
The ggplot2 package provides ggplot() and geom_point() function for creating a scatterplot. The ggplot() function takes a series of the input item. The first parameter is an input vector, and the second is the aes() function in which we add the x-axis and y-axis.

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

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

Output:



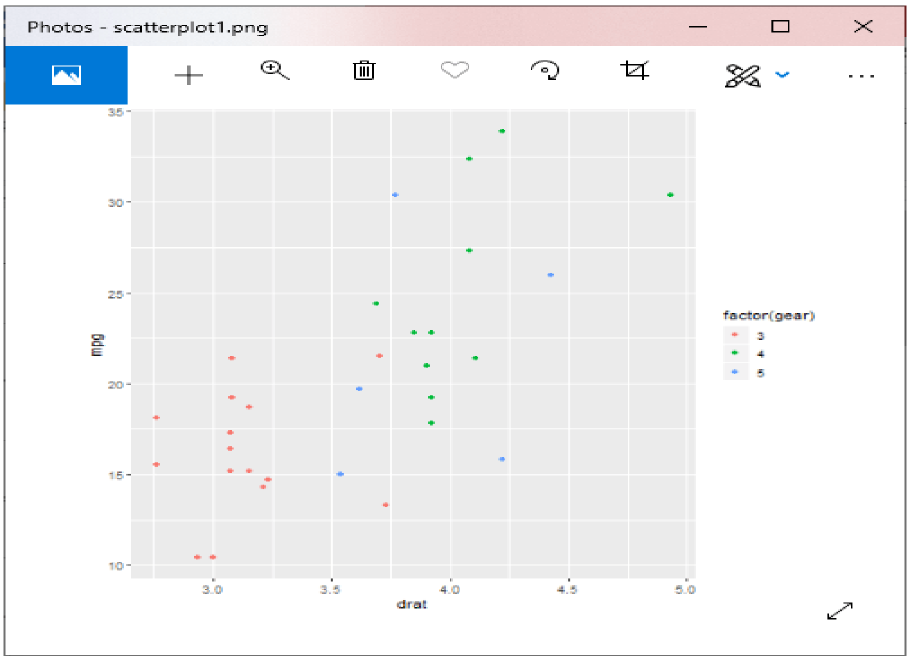
We can add more features and make a more attractive scatter plots also. Below are some examples in which different parameters are added.

Example 1: Scatterplot with groups

1. #Loading ggplot2 package
2. library(ggplot2)
3. # Giving a name to the chart file.

4. png(file = "scatterplot1.png")
5. # Plotting the chart using ggplot() and geom_point() functions.
6. #The aes() function inside the geom_point() function controls the color of the group.
7. ggplot(mtcars, aes(x = drat, y = mpg)) +
8. geom_point(aes(color=factor(gear)))
9. # Saving the file.
10. dev.off()

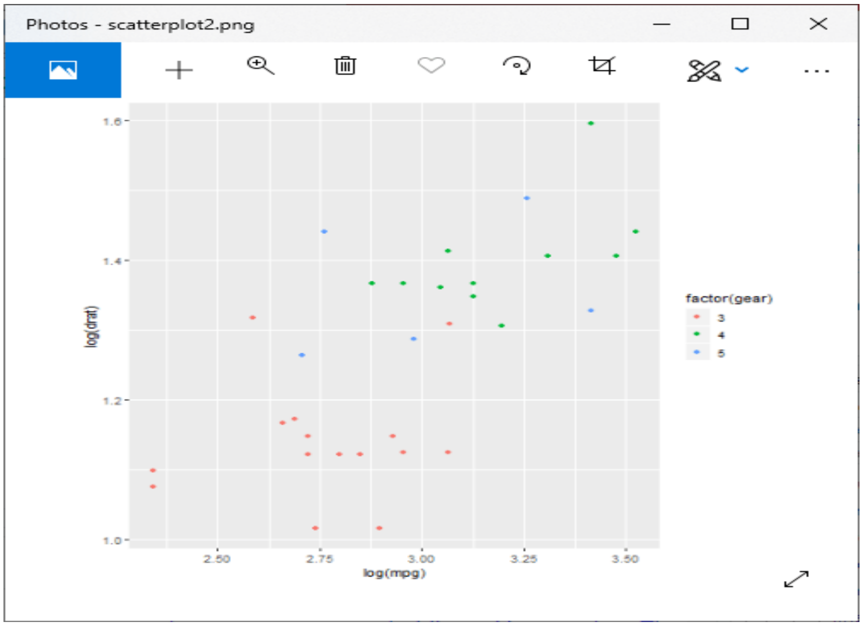
Output:



Example 2: Changes in axis

1. #Loading ggplot2 package
2. library(ggplot2)
3. # Giving a name to the chart file.
4. png(file = "scatterplot2.png")
5. # Plotting the chart using ggplot() and geom_point() functions.
6. #The aes() function inside the geom_point() function controls the color of the group.
7. ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color=factor(gear)))
8. # Saving the file.
9. dev.off()

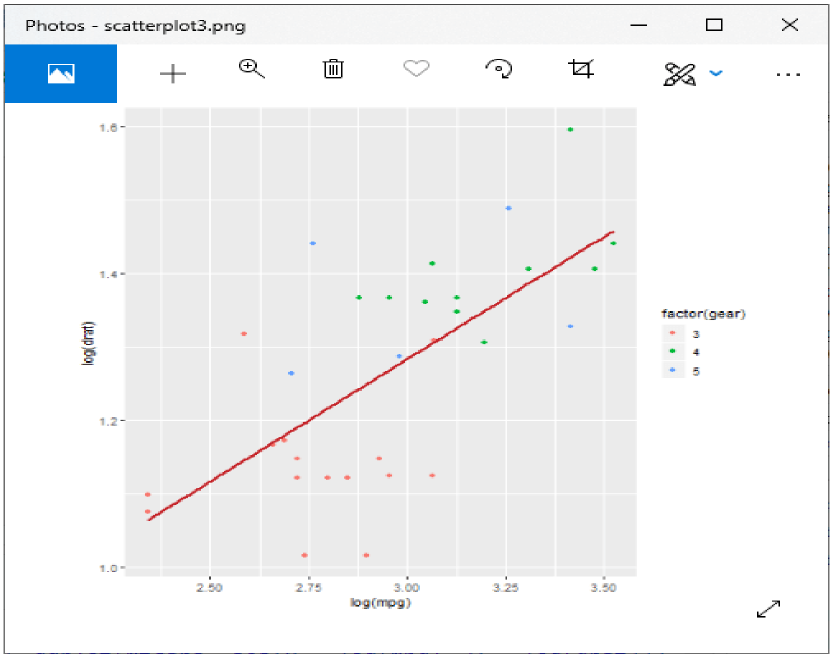
Output:



Example 3: Scatterplot with fitted values

```
#Loading ggplot2 package
library(ggplot2)
# Giving a name to the chart file.
png(file = "scatterplot3.png")
#Creating scatterplot with fitted values.
# An additional function stst_smooth is used for linear regression.
ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color = factor(gear))) + stat_smooth(method = "lm",col = "#C42126",se = FALSE,size = 1)
#in above example lm is used for linear regression and se stands for standard error.
# Saving the file.
dev.off()
```

Output:



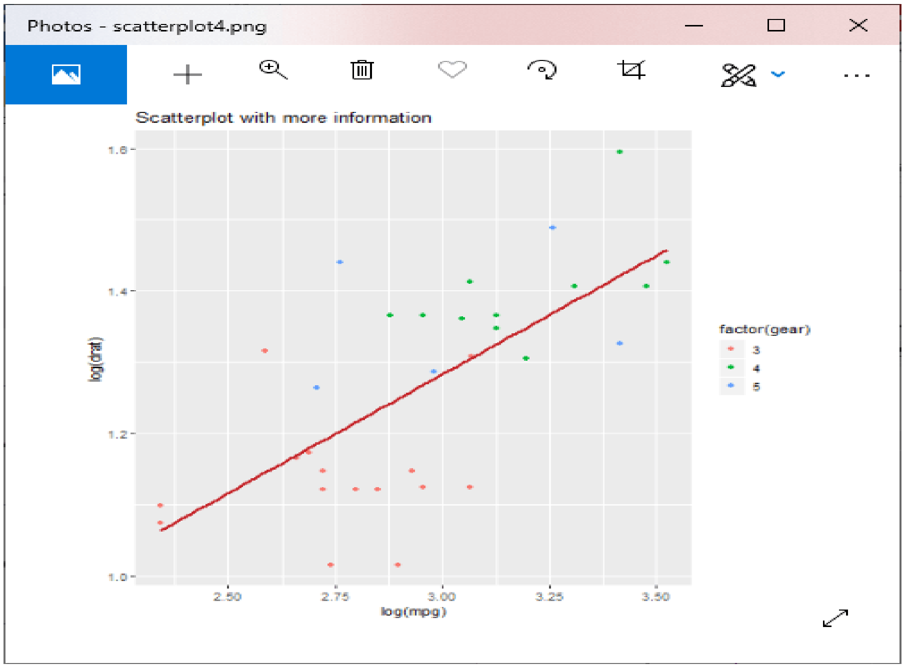
Adding information to the graph

Example 4: Adding title

1. #Loading ggplot2 package
2. library(ggplot2)
3. # Giving a name to the chart file.
4. png(file = "scatterplot4.png")
5. #Creating scatterplot with fitted values.
6. # An additional function stst_smooth is used for linear regression.
7. new_graph<-
   ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color = factor(gear)))
   +
8. stat_smooth(method = "lm",col = "#C42126",se = FALSE,size = 1)
9. #in above example lm is used for linear regression and se stands for standard error.
10. new_graph+
11. labs(
12. title = "Scatterplot with more information"
13.)
14. # Saving the file.

15. dev.off()

Output:

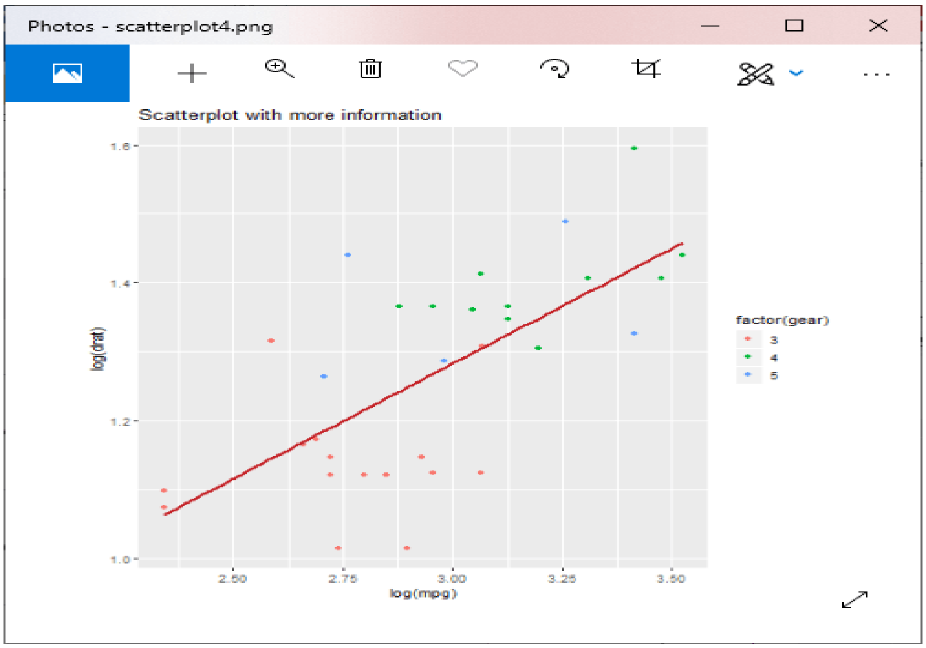


Example 5: Adding title with dynamic name

1. #Loading ggplot2 package
2. library(ggplot2)
3. # Giving a name to the chart file.
4. png(file = "scatterplot5.png")
5. #Creating scatterplot with fitted values.
6. # An additional function stst_smooth is used for linear regression.
7. new_graph<-
   ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color = factor(gear)))
   +
8. stat_smooth(method = "lm",col = "#C42126",se = FALSE,size = 1)
9. #in above example lm is used for linear regression and se stands for standard error.
10. #Finding mean of mpg
11. mean_mpg<- mean(mtcars$mpg)
12. #Adding title with dynamic name
13. new_graph + labs(

14. title = paste("Adding additiona information. Average mpg is", mean_mpg)
15.)
16. # Saving the file.
17. dev.off()

Output:

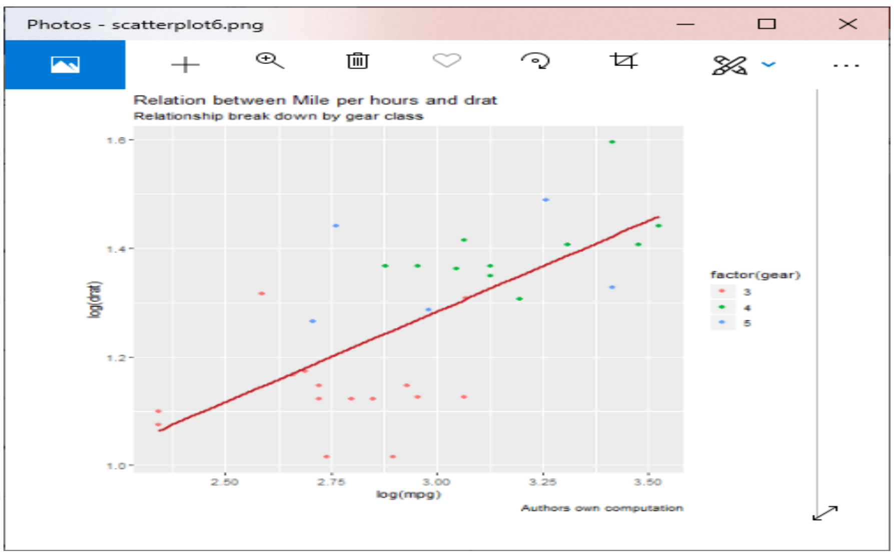


Example 6: Adding a sub-title

1. #Loading ggplot2 package
2. library(ggplot2)
3. # Giving a name to the chart file.
4. png(file = "scatterplot6.png")
5. #Creating scatterplot with fitted values.
6. # An additional function stst_smooth is used for linear regression.
7. new_graph<-
   ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color = factor(gear)))
   +
8. stat_smooth(method = "lm",col = "#C42126",se = FALSE,size = 1)
9. #in above example lm is used for linear regression and se stands for standard error.
10. #Adding title with dynamic name

```
new_graph + labs(
      title =
            "Relation between Mile per hours and drat",
      subtitle =
            "Relationship break down by gear class",
      caption = "Authors own computation"
)
# Saving the file.
dev.off()
```

Output:

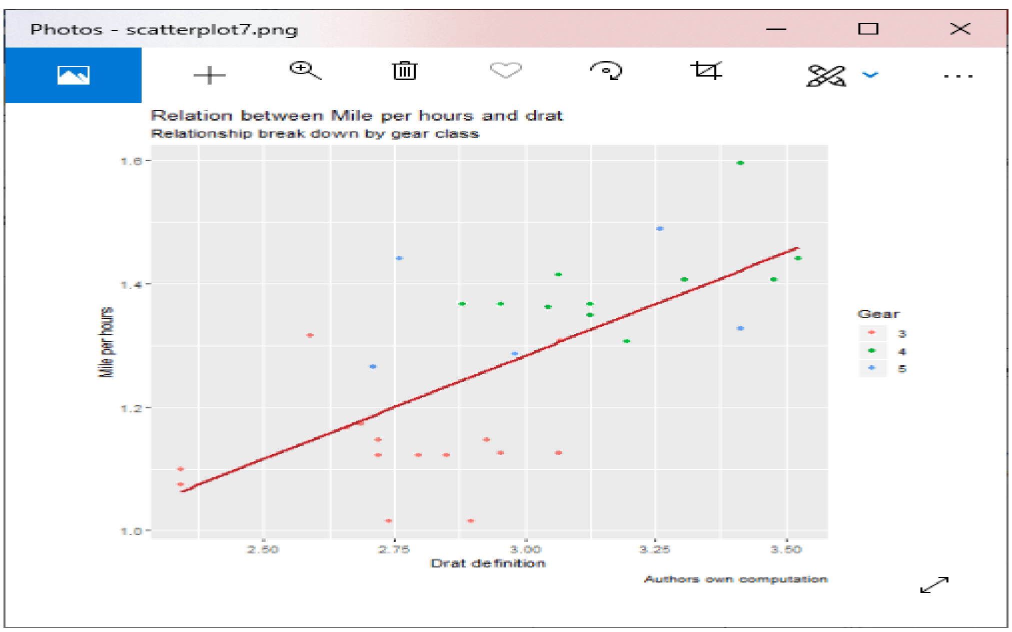


Example 7: Changing name of x-axis and y-axis

```
#Loading ggplot2 package
library(ggplot2
# Giving a name to the chart file.
png(file = "scatterplot7.png")
#Creating scatterplot with fitted values.
# An additional function stst_smooth is used for linear regression.
new_graph<-
ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color = factor(gear)))
+
```

```
stat_smooth(method = "lm",col = "#C42126",se = FALSE,size = 1)
#in above example lm is used for linear regression and se stands for standard error.
#Adding title with dynamic name
new_graph + labs(
        x = "Drat definition",
        y = "Mile per hours",
        color = "Gear",
        title = "Relation between Mile per hours and drat",
        subtitle = "Relationship break down by gear class",
        caption = "Authors own computation"
)
# Saving the file.
dev.off()
```

Output:

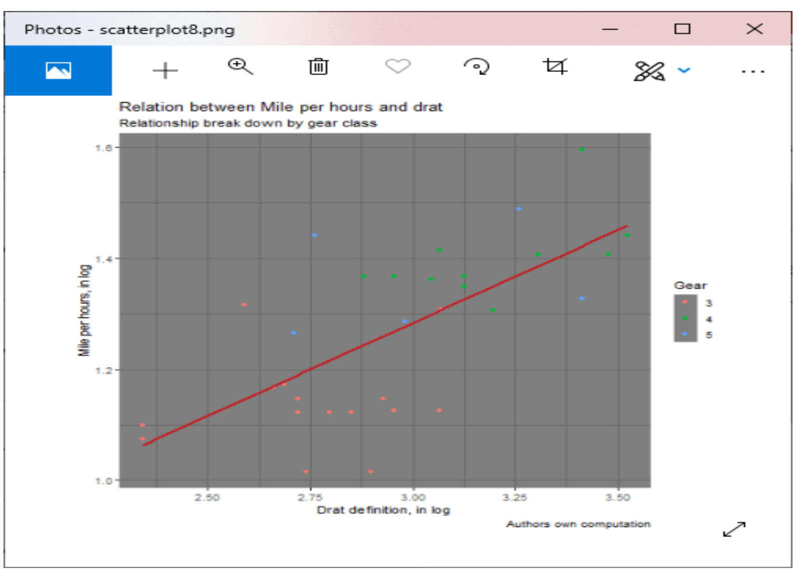


Example 8: Adding theme

```
#Loading ggplot2 package
library(ggplot2
# Giving a name to the chart file.
png(file = "scatterplot8.png")
```

```
#Creating scatterplot with fitted values.
# An additional function stst_smooth is used for linear regression.
new_graph<-
ggplot(mtcars, aes(x = log(mpg), y = log(drat))) +geom_point(aes(color = factor(gear)))
+
stat_smooth(method = "lm",col = "#C42126",se = FALSE,size = 1)
#in above example lm is used for linear regression and se stands for standard error.
#Adding title with dynamic name
new_graph+
theme_dark() +
        labs(
            x = "Drat definition, in log",
            y = "Mile per hours, in log",
            color = "Gear",
            title = "Relation between Mile per hours and drat",
            subtitle = "Relationship break down by gear class",
            caption = "Authors own computation"
        )
# Saving the file.
dev.off()
```

Output:



Linear Regression

Linear regression is used to predict the value of an outcome variable y on the basis of one or more input predictor variables x. In other words, linear regression is used to establish a linear relationship between the predictor and response variables.

In linear regression, predictor and response variables are related through an equation in which the exponent of both these variables is 1. Mathematically, a linear relationship denotes a straight line, when plotted as a graph.

There is the following general mathematical equation for linear regression:

1. y = ax + b

Here,

- y is a response variable.
- x is a predictor variable.
- a and b are constants that are called the coefficients.

Steps for establishing the Regression

The prediction of the weight of a person when his height is known, is a simple example of regression. To predict the weight, we need to have a relationship between the height and weight of a person.

There are the following steps to create the relationship:

1. In the first step, we carry out the experiment of gathering a sample of observed values of height and weight.
2. After that, we create a relationship model using the lm() function of R.
3. Next, we will find the coefficient with the help of the model and create the mathematical equation using this coefficient.
4. We will get the summary of the relationship model to understand the average error in prediction, known as residuals.
5. At last, we use the predict() function to predict the weight of the new person.

There is the following syntax of lm() function:

```
1. lm(formula,data)
```

Here,

S.No	Parameters	Description
1.	Formula	It is a symbol that presents the relationship between x and y.
2.	Data	It is a vector on which we will apply the formula.

Creating Relationship Model and Getting the Coefficients

Let's start performing the second and third steps, i.e., creating a relationship model and getting the coefficients. We will use the lm() function and pass the x and y input vectors and store the result in a variable named **relationship_model**.

Example

```
1. #Creating input vector for lm() function
2. x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
3. y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
4. # Applying the lm() function.
5. relationship_model<- lm(y~x)
6. #Printing the coefficient
7. print(relationship_model)
```

Output:

```
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
   47.50833      0.07276
```

Getting Summary of Relationship Model

We will use the summary() function to get a summary of the relationship model. Let's see an example to understand the use of the summary() function.

Example

```
1. #Creating input vector for lm() function
2. x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
3. y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
4.
5. # Applying the lm() function.
6. relationship_model<- lm(y~x)
7.
8. #Printing the coefficient
9. print(summary(relationship_model))
```

Output:

```
Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max 
-38.948  -7.390   1.869  15.933  34.087 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 47.50833   55.18118   0.861    0.414
x            0.07276    0.39342   0.185    0.858

Residual standard error: 25.96 on 8 degrees of freedom
Multiple R-squared:  0.004257, Adjusted R-squared:  -0.1202
F-statistic: 0.0342 on 1 and 8 DF,  p-value: 0.8579
```

The predict() Function

Now, we will predict the weight of new persons with the help of the predict() function. There is the following syntax of predict function:

```
1. predict(object, newdata)
```

Here,

S.No	Parameter	Description
1.	object	It is the formula that we have already created using the lm() function.
2.	Newdata	It is the vector that contains the new value for the predictor variable.

Example

```
#Creating input vector for lm() function
x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)

# Applying the lm() function.
relationship_model<- lm(y~x)

# Finding the weight of a person with height 170.
z <- data.frame(x = 160)
predict_result<-  predict(relationship_model,z)
print(predict_result)
```

Output:

```
1
59.14977
```

Plotting Regression

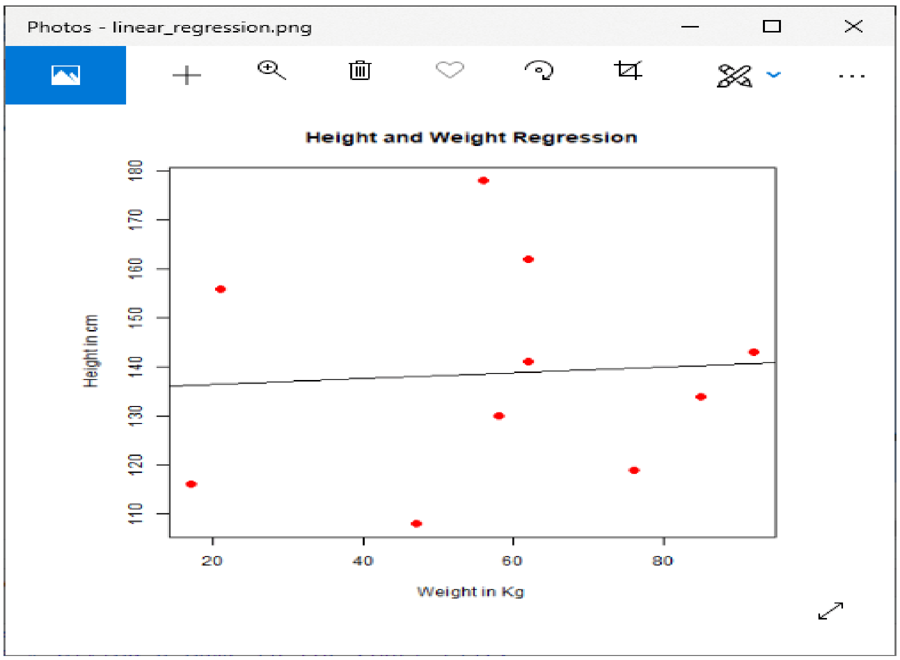
Now, we plot out prediction results with the help of the plot() function. This function takes parameter x and y as an input vector and many more arguments.

Example

```
#Creating input vector for lm() function
x <- c(141, 134, 178, 156, 108, 116, 119, 143, 162, 130)
y <- c(62, 85, 56, 21, 47, 17, 76, 92, 62, 58)
relationship_model<- lm(y~x)
# Giving a name to the chart file.
```

6. png(file = "linear_regression.png")
7. # Plotting the chart.
8. plot(y,x,col = "red",main = "Height and Weight Regression",abline(lm(x~y)),cex = 1.3,pch = 16,xlab = "Weight in Kg",ylab = "Height in cm")
9. # Saving the file.
10. dev.off()

Output:



R-Multiple Linear Regression

Multiple linear regression is the extension of the simple linear regression, which is used to predict the outcome variable (y) based on multiple distinct predictor variables (x). With the help of three predictor variables (x1, x2, x3), the prediction of y is expressed using the following equation:

$$y=b_0+b_1*x_1+b_2*x_2+b_3*x_3$$

The "b" values represent the regression weights. They measure the association between the outcome and the predictor variables. "

Or

Multiple linear regression is the extension of linear regression in the relationship between more than two variables. In simple linear regression, we have one predictor and one response variable. But in multiple regressions, we have more than one predictor variable and one response variable.

There is the following general mathematical equation for multiple regression -

$$y=b_0+b_1*x_1+b_2*x_2+b_3*x_3+\cdots b_n*x_n$$

Here,

- **y** is a response variable.
- **b0, b1, b2...bn** are the coefficients.
- **x1, x2, ...xn** are the predictor variables.

In R, we create the regression model with the help of the **lm()** function. The model will determine the value of the coefficients with the help of the input data. We can predict the value of the response variable for the set of predictor variables using these coefficients.

There is the following syntax of lm() function in multiple regression

```
1. lm(y ~ x1+x2+x3...., data)
```

Before proceeding further, we first create our data for multiple regression. We will use the "mtcars" dataset present in the R environment. The main task of the model is to create the relationship between the "mpg" as a response variable with "wt", "disp" and "hp" as predictor variables.

For this purpose, we will create a subset of these variables from the "mtcars" dataset.

```
1. data<-mtcars[,c("mpg","wt","disp","hp")]
2. print(head(input))
```

Output:

```
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

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.

[Previously saved workspace restored]

> data<-mtcars[,c("mpg","wt","disp","hp")]
> print(head(data))
                   mpg    wt disp  hp
Mazda RX4         21.0 2.620  160 110
Mazda RX4 Wag     21.0 2.875  160 110
Datsun 710        22.8 2.320  108  93
Hornet 4 Drive    21.4 3.215  258 110
Hornet Sportabout 18.7 3.440  360 175
Valiant           18.1 3.460  225 105
>
```

Creating Relationship Model and finding Coefficient

Now, we will use the data which we have created before to create the Relationship Model. We will use the lm() function, which takes two parameters i.e., formula and data. Let's start understanding how the lm() function is used to create the Relationship Model.

Example

```
1. #Creating input data.
2. input <- mtcars[,c("mpg","wt","disp","hp")]
3. # Creating the relationship model.
4. Model <- lm(mpg~wt+disp+hp, data = input)
5. # Showing the Model.
6. print(Model)
```

Output:

From the above output it is clear that our model is successfully setup. Now, our next step is to find the coefficient with the help of the model.

```
b0<- coef(Model)[1]
print(b0)
x_wt<- coef(Model)[2]
x_disp<- coef(Model)[3]
x_hp<- coef(Model)[4]
print(x_wt)
print(x_disp)
print(x_hp)
```

Output:

```
Rterm (64-bit)
> Model <- lm(mpg~wt+disp+hp, data = input)
> # Showing the Model.
> print(Model)

Call:
lm(formula = mpg ~ wt + disp + hp, data = input)

Coefficients:
(Intercept)           wt         disp           hp
  37.105505    -3.800891    -0.000937    -0.031157

> b0 <- coef(Model)[1]
> print(b0)
(Intercept)
   37.10551
> x_wt <- coef(Model)[2]
> x_disp <- coef(Model)[3]
> x_hp <- coef(Model)[4]
> print(x_wt)
       wt
-3.800891
> print(x_disp)
         disp
-0.0009370091
> print(x_hp)
         hp
-0.03115655
>
```

The equation for the Regression Model

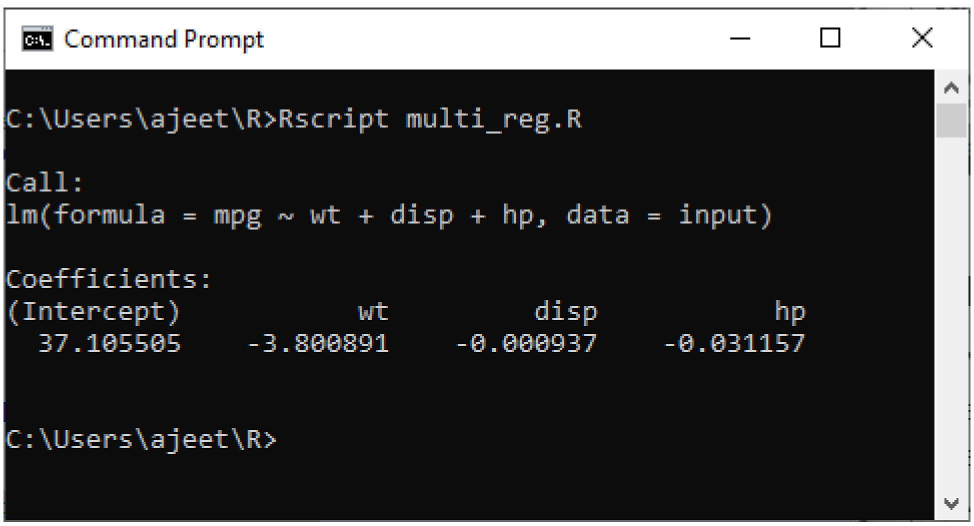
Now, we have coefficient values and intercept. Let's start creating a mathematical equation that we will apply for predicting new values. First, we will create an equation, and then we use the equation to predict the mileage when a new set of values for weight, displacement, and horsepower is provided.

Let's see an example in which we predict the mileage for a car with weight=2.51, disp=211 and hp=82.

Example

```
1. #Creating equation for predicting new values.
2. y=b0+x_wt*x1+x_disp*x2+x_hp*x3\
3. #Applying equation for prediction new values
4. y=b0+x_wt*2.51+x_disp*211+x_hp*82
```

Output:



R-Logistic Regression

In the logistic regression, a regression curve, y = f (x), is fitted. In the regression curve equation, y is a categorical variable. This Regression Model is used for predicting that y has given a set of predictors x. Therefore, predictors can be categorical, continuous, or a mixture of both.

The logistic regression is a classification algorithm that falls under nonlinear regression. This model is used to predict a given binary result (1/0, yes/no, true/false) as a set of independent variables. Furthermore, it helps to represent categorical/binary outcomes using dummy variables.

Logistic regression is a regression model in which the response variable has categorical values such as true/false or 0/1. Therefore, we can measure the probability of the binary response.

There is the following mathematical equation for the logistic regression:

$$y=1/(1+e^{\wedge}-(b_0+b_1 x_1+b_2 x_2+\cdots))$$

In the above equation, y is a response variable, x is the predictor variable, and b_0 and b_1, b_2,...b_n are the coefficients, which is numeric constants. We use the glm() function to create the regression model.

There is the following syntax of the glm() function.

1. glm(formula, data, family)

Here,

S.No	Parameter	Description
1.	formula	It is a symbol which represents the relationship b/w the variables.
2.	data	It is the dataset giving the values of the variables.
3.	family	An R object which specifies the details of the model, and its value is binomial for logistic regression.

Building Logistic Regression

The in-built data set "mtcars" describes various models of the car with their different engine specifications. In the "mtcars" data set, the transmission mode is described by the column "am", which is a binary value (0 or 1). We can construct a logistic regression model between column "am" and three other columns - hp, wt, and cyl.

Let's see an example to understand how the glm function is used to create logistic regression and how we can use the summary function to find a summary for the analysis.

In our example, we will use the dataset "BreastCancer" available in the R environment. To use it, we first need to install "mlbench" and "caret" packages.

Example

1. #Loading library
2. library(mlbench)
3. #Using BreastCancer dataset
4. data(BreastCancer, package = "mlbench")
5. breast_canc = BreastCancer[complete.cases(BreastCancer),]
6. #Displaying the information related to dataset with the str() function.
7. str(breast_canc)

Output:

Command Prompt

```
C:\Users\ajeet\R>Rscript logistic_regression.R
'data.frame':   683 obs. of  11 variables:
 $ Id             : chr  "1000025" "1002945" "1015425" "1016277" ...
 $ Cl.thickness   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 5 5 3 6 4 8 1 2 2 4 ...
 $ Cell.size      : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 4 1 8 1 10 1 1 1 2 ...
 $ Cell.shape     : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 4 1 8 1 10 1 2 1 1 ...
 $ Marg.adhesion  : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 5 1 1 3 8 1 1 1 1 ...
 $ Epith.c.size   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 2 7 2 3 2 7 2 2 2 2 ...
 $ Bare.nuclei    : Factor w/ 10 levels "1","2","3","4",..: 1 10 2 4 1 10 10 1 1 1 ...
 $ Bl.cromatin    : Factor w/ 10 levels "1","2","3","4",..: 3 3 3 3 3 9 3 3 1 2 ...
 $ Normal.nucleoli: Factor w/ 10 levels "1","2","3","4",..: 1 2 1 7 1 7 1 1 1 1 ...
 $ Mitoses        : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 5 1 ...
 $ Class          : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 1 1 1 1 ...

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

We now divide our data into training and test sets with training sets containing 70% data and test sets including the remaining percentages.

1. #Dividing dataset into training and test dataset.
2. set.seed(100)
3. #Creating partitioning.
4. Training_Ratio <- createDataPartition(b_canc$Class, p=0.7, list = F)

5. #Creating training data.
6. Training_Data <- b_canc[Training_Ratio,]
7. str(Training_Data)
8. #Creating test data.
9. Test_Data <- b_canc[-Training_Ratio,]
10. str(Test_Data)

Output:

```
Command Prompt

C:\Users\ajeet\R>Rscript logistic_regression.R
Loading required package: lattice
Loading required package: ggplot2
'data.frame':   479 obs. of  11 variables:
 $ Id             : chr  "1000025" "1002945" "1015425" "1016277" ...
 $ Cl.thickness   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 5 5 3 6 8 2 2 1 7 4 ...
 $ Cell.size      : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 4 1 8 10 1 1 1 4 1 ...
 $ Cell.shape     : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 4 1 8 10 2 1 1 6 1 ...
 $ Marg.adhesion  : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 5 1 1 8 1 1 1 4 1 ...
 $ Epith.c.size   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 2 7 2 3 7 2 2 2 6 2 ...
 $ Bare.nuclei    : Factor w/ 10 levels "1","2","3","4",..: 1 10 2 4 10 1 1 3 1 1 ...
 $ Bl.cromatin    : Factor w/ 10 levels "1","2","3","4",..: 3 3 3 3 9 3 2 3 4 3 ...
 $ Normal.nucleoli: Factor w/ 10 levels "1","2","3","4",..: 1 2 1 7 7 1 1 1 3 1 ...
 $ Mitoses        : Factor w/ 9 levels "1","2","3","4",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Class          : Factor w/ 2 levels "benign","malignant": 1 1 1 1 2 1 1 1 2 1 ...
'data.frame':   204 obs. of  11 variables:
 $ Id             : chr  "1017023" "1018099" "1033078" "1033078" ...
 $ Cl.thickness   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 4 1 2 4 1 5 8 4 1 3 ...
 $ Cell.size      : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 1 1 2 1 3 7 1 1 2 ...
 $ Cell.shape     : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 1 1 1 1 1 3 5 1 1 1 ...
 $ Marg.adhesion  : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 3 1 1 1 1 3 10 1 1 1 ...
 $ Epith.c.size   : Ord.factor w/ 10 levels "1"<"2"<"3"<"4"<..: 2 2 2 2 1 2 7 2 2 1 ...
 $ Bare.nuclei    : Factor w/ 10 levels "1","2","3","4",..: 1 10 1 1 1 3 9 1 1 1 ...
 $ Bl.cromatin    : Factor w/ 10 levels "1","2","3","4",..: 3 3 1 2 3 4 5 2 3 2 ...
 $ Normal.nucleoli: Factor w/ 10 levels "1","2","3","4",..: 1 1 1 1 1 4 5 1 1 1 ...
 $ Mitoses        : Factor w/ 9 levels "1","2","3","4",..: 1 1 5 1 1 1 4 1 1 1 ...
 $ Class          : Factor w/ 2 levels "benign","malignant": 1 1 1 1 1 2 2 1 1 1 ...

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

Now, we construct the logistic regression function with the help of glm() function. We pass the formula **Class~Cell.shape** as the first parameter and specifying the attribute family as "**binomial**" and use Training_data as the third parameter.

Example

1. #Creating Regression Model
2. glm(Class ~ Cell.shape, family="binomial", data = Training_Data)

Output:

```
C:\Users\ajeet\R>Rscript logistic_regression.R
Loading required package: lattice
Loading required package: ggplot2

Call:  glm(formula = Class ~ Cell.shape, family = "binomial", data = Training_Data)

Coefficients:
 (Intercept)  Cell.shape.L  Cell.shape.Q  Cell.shape.C  Cell.shape^4
      4.1470       20.7653        7.3156        5.4212       -1.3400
Cell.shape^5  Cell.shape^6  Cell.shape^7  Cell.shape^8  Cell.shape^9
     -4.2334       -4.9429       -3.2306       -1.8817       -0.9361

Degrees of Freedom: 478 Total (i.e. Null);  469 Residual
Null Deviance:      620.7
Residual Deviance: 189.4        AIC: 209.4

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

Now, use the summary function for analysis.

1. #Creating Regression Model
2. model<-glm(Class ~ Cell.shape, family="binomial", data = Training_Data)
3. #Using summary function
4. print(summary(model))

Output:

```
C:\Users\ajeet\R>Rscript logistic_regression.R
Loading required package: lattice
Loading required package: ggplot2

Call:
glm(formula = Class ~ Cell.shape, family = "binomial", data = Training_Data)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.5042  -0.1307  -0.1307   0.2982   3.0875

Coefficients:
              Estimate Std. Error z value Pr(>|z|)
(Intercept)     4.1470   390.1191   0.011    0.992
Cell.shape.L   20.7653  1536.3778   0.014    0.989
Cell.shape.Q    7.3156   844.2448   0.009    0.993
Cell.shape.C    5.4212   733.0098   0.007    0.994
Cell.shape^4   -1.3400  1586.6694  -0.001    0.999
Cell.shape^5   -4.2334  1900.3885  -0.002    0.998
Cell.shape^6   -4.9429  1616.8215  -0.003    0.998
Cell.shape^7   -3.2306  1037.6703  -0.003    0.998
Cell.shape^8   -1.8817   493.2900  -0.004    0.997
Cell.shape^9   -0.9361   153.7797  -0.006    0.995

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 620.69  on 478  degrees of freedom
Residual deviance: 189.36  on 469  degrees of freedom
AIC: 209.36

Number of Fisher Scoring iterations: 17
```

R Poisson Regression

The **Poisson Regression** model is used for modeling events where the outcomes are counts. Count data is a discrete data with non-negative integer values that count things, such as the number of people in line at the grocery store, or the number of times an event occurs during the given timeframe.

We can also define the **count data** as the rate data. So that it can express the number of times an event occurs within the timeframe as a raw count or as a rate. Poisson regression allows us to determine which explanatory variable (x values) influence a given response variable (y value, count, or a rate).

For example, poisson regression can be implemented by a grocery store to understand better, and predict the number of people in a row.

There is the following general mathematical equation for poisson regression:

Here,

S.No	Parameter	Description
1.	y	It is the response variable.
2.	a and b	These are the numeric coefficients.
3.	x	x is the predictor variable.

The poisson regression model is created with the help of the familiar function glm().

Let's see an example in which we create the poisson regression model using glm() function. In this example, we have considered an in-built dataset "wrapbreaks" that describe the tension(low, medium, or high), and the effect of wool type(A and B) on the number of wrap breaks per loom. We will consider wool "type" and "tension"as the predictor variables, and "breaks" is taken as the response variable.

Example

```
#Creting data for the poisson regression
reg_data<-warpbreaks
print(head(reg_data))
```

Output:

```
Select Command Prompt
Microsoft Windows [Version 10.0.18362.239]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ajeet>cd R

C:\Users\ajeet\R>Rscript poisson_regression.R
  breaks wool tension
1     26    A       L
2     30    A       L
3     54    A       L
4     25    A       L
5     70    A       L
6     52    A       L

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

Now, we will create the regression model with the help of the glm() function as:

1. #Creating Poisson Regression Model using glm() function
2. output_result <- glm(formula = breaks ~ wool+tension, data = warpbreaks,family = poisson)
3. output_result

Output:

```
Command Prompt

C:\Users\ajeet\R>Rscript poisson_regression.R

Call:  glm(formula = breaks ~ wool + tension, family = poisson,
 data = warpbreaks)

Coefficients:
(Intercept)        woolB     tensionM     tensionH
     3.6920      -0.2060      -0.3213      -0.5185

Degrees of Freedom: 53 Total (i.e. Null);  50 Residual
Null Deviance:      297.4
Residual Deviance: 210.4        AIC: 493.1

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

Now, let's use summary() function to find the summary of the model for data analysis.

1. #Using summary function
2. print(summary(output_result))

Output:

```
Command Prompt

C:\Users\ajeet\R>Rscript poisson_regression.R

Call:
glm(formula = breaks ~ wool + tension, family = poisson, data =
warpbreaks)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-3.6871  -1.6503  -0.4269   1.1902   4.2616

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.69196    0.04541  81.302  < 2e-16 ***
woolB       -0.20599    0.05157  -3.994 6.49e-05 ***
tensionM    -0.32132    0.06027  -5.332 9.73e-08 ***
tensionH    -0.51849    0.06396  -8.107 5.21e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 297.37  on 53  degrees of freedom
Residual deviance: 210.39  on 50  degrees of freedom
AIC: 493.06

Number of Fisher Scoring iterations: 4
```