

Unit-5

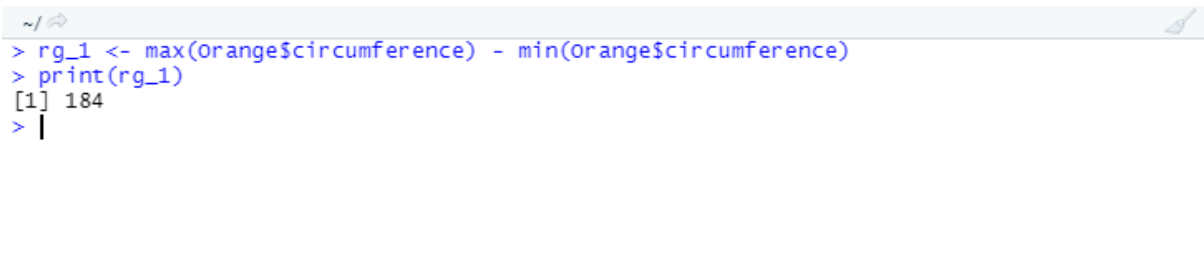
Descriptive Statistics:

When it comes to summarize, present and describe data in the simplest possible way, the descriptive statistics help. They are often called the first and important step in statistical analysis. Most of the time an actual analysis starts when an analyst digs the data and presents some descriptive statistics out of it for the user. The descriptive statistics allows us to understand the data with just an overview of the same. They are more or less the best alternatives when it comes to understanding the data and if done properly can already be a better start to the deep or advanced statistical analysis.

In this chapter, we are about to see how the descriptive statistics can be taken out under [R programming](#) with hands-on examples.

Data Range:

The range in statistics is nothing but the difference between the maximum and the minimum value. It gives you a better picture of the spread of the data.



```
> rg_1 <- max(Orange$circumference) - min(Orange$circumference)
> print(rg_1)
[1] 184
> |
```

Example code for finding out the range with output

Unfortunately, we don't have any dedicated function in R that computes the range for us. However, we are free to develop one of your own. See an example below:

```

> #Developing a function that computes the range
> range1 <- function(x){
+   range <- max(x) - min(x)
+   return(range)
+ }
>
> range1(orange$circumference)
[1] 184
>

```

Example code for creating a function that computes the range

Frequencies

This section describes the creation of frequency and contingency tables from categorical variables, along with tests of independence, measures of association, and methods for graphically displaying results.

Generating Frequency Tables

R provides many methods for creating frequency and contingency tables. Three are described below. In the following examples, assume that A, B, and C represent categorical variables.

table

You can generate frequency tables using the **table()** function, tables of proportions using the **prop.table()** function, and marginal frequencies using **margin.table()**.

```
# 2-Way Frequency Table
```

```
attach(mydata)
```

```
mytable <- table(A,B) # A will be rows, B will be columns
```

```
mytable # print table
```

```
margin.table(mytable, 1) # A frequencies (summed over B)
```

```
margin.table(mytable, 2) # B frequencies (summed over A)
```

```
prop.table(mytable) # cell percentages
```

```
prop.table(mytable, 1) # row percentages
```

```
prop.table(mytable, 2) # column percentages
```

table() can also generate multidimensional tables based on 3 or more categorical variables. In this case, use the **ftable()** function to print the results more attractively.

```
# 3-Way Frequency Table
```

```
mytable <- table(A, B, C)
```

```
ftable(mytable)
```

Table ignores missing values. To include **NA** as a category in counts, include the table option `exclude=NULL` if the variable is a vector. If the variable is a factor you have to create a new factor using `newfactor <- factor(oldfactor, exclude=NULL)`.

Mode, Mean and Median

Statistical analysis in R is performed by using many in-built functions. Most of these functions are part of the R base package. These functions take R vector as an input along with the arguments and give the result.

The functions we are discussing in this chapter are mean, median and mode.

Mean

It is calculated by taking the sum of the values and dividing with the number of values in a data series.

The function **mean()** is used to calculate this in R.

Syntax

The basic syntax for calculating mean in R is –

```
mean(x, trim = 0, na.rm = FALSE, ...)
```

Following is the description of the parameters used –

x is the input vector.

trim is used to drop some observations from both end of the sorted vector.

na.rm is used to remove the missing values from the input vector.

Example

```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.
result.mean <- mean(x)
print(result.mean)
```

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

```
[1] 8.22
```

Applying Trim Option

When trim parameter is supplied, the values in the vector get sorted and then the required numbers of observations are dropped from calculating the mean.

When trim = 0.3, 3 values from each end will be dropped from the calculations to find mean.

In this case the sorted vector is (-21, -5, 2, 3, 4.2, 7, 8, 12, 18, 54) and the values removed from the vector for calculating mean are (-21,-5,2) from left and (12,18,54) from right.

```
# Create a vector.
x <- c(12,7,3,4.2,18,2,54,-21,8,-5)

# Find Mean.
```

```
result.mean <- mean(x, trim = 0.3)
print(result.mean)
```

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

```
[1] 5.55
```

Applying NA Option

If there are missing values, then the mean function returns NA.

To drop the missing values from the calculation use `na.rm = TRUE`. which means remove the NA values.

```
# Create a vector.
x <- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5, NA)

# Find mean.
result.mean <- mean(x)
print(result.mean)

# Find mean dropping NA values.
result.mean <- mean(x, na.rm = TRUE)
print(result.mean)
```

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

```
[1] NA
[1] 8.22
```

Median

The middle most value in a data series is called the median.

The **median()** function is used in R to calculate this value.

Syntax

The basic syntax for calculating median in R is –

```
median(x, na.rm = FALSE)
```

Following is the description of the parameters used –

x is the input vector.

na.rm is used to remove the missing values from the input vector.

Example

```
# Create the vector.
x <- c(12, 7, 3, 4.2, 18, 2, 54, -21, 8, -5)
```

```
# Find the median.
median.result <- median(x)
print(median.result)
```

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

```
[1] 5.6
```

Mode

The mode is the value that has highest number of occurrences in a set of data. Unlike mean and median, mode can have both numeric and character data.

R does not have a standard in-built function to calculate mode. So we create a user function to calculate mode of a data set in R. This function takes the vector as input and gives the mode value as output.

Example

```
# Create the function.
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Create the vector with numbers.
v <- c(2,1,2,3,1,2,3,4,1,5,5,3,2,3)

# Calculate the mode using the user function.
result <- getmode(v)
print(result)

# Create the vector with characters.
charv <- c("o","it","the","it","it")

# Calculate the mode using the user function.
result <- getmode(charv)
print(result)
```

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

```
[1] 2
[1] "it"
```

Standard Deviation

You can calculate standard deviation in R using the `sd()` function. This standard deviation function is a part of standard R, and needs no extra packages to be calculated.

```
# set up standard deviation in R example
> test <- c(41,34,39,34,34,32,37,32,43,43,24,32)

# standard deviation R function
# sample standard deviation in r
> sd(test)
[1] 5.501377
```

As you can see, calculating standard deviation in R is as simple as that- the basic R function computes the standard deviation for you easily. Need to get the standard deviation for an entire data set? Use the `sapply()` function to map it across the relevant items. For this example, we're going to use the [ChickWeight dataset](#) in Base R. This will help us calculate the standard deviation of columns in R.

```
# standard deviation in R - dataset example
# using head to show the first handful of records
> head(ChickWeight)
  weight Time Chick Diet
1     42    0     1    1
2     51    2     1    1
3     59    4     1    1
4     64    6     1    1
5     76    8     1    1
6     93   10     1    1

# standard deviation in R - using sapply to map across columns
# how to calculate standard deviation in r data frame
> sapply(ChickWeight[,1:4], sd)
  weight      Time      Chick      Diet 
71.071960  6.758400 13.996847  1.162678
```

Learning how to calculate standard deviation in r is quite simple, but an invaluable skill for any programmer. These techniques can be used to calculate sample standard deviation in r, standard deviation of rows in r, and much more. None of the columns need to be

removed before computation proceeds, as each column's standard deviation is calculated.

Correlation

It is a statistical measure that defines the relationship between two variables that is how the two variables are linked with each other. It describes the effect of change in one variable on another variable.

If the two variables are increasing or decreasing in parallel then they have a positive correlation between them and if one of the variables is increasing and another one is decreasing then they have a negative correlation with each other. If the change of one variable has no effect on another variable then they have a zero correlation between them.

It is used to identify the degree of the linear relationship between two variables. It is represented by ρ and calculated as:-

$$\rho(x, y) = \text{cov}(x, y) / (\sigma_x \times \sigma_y)$$

Where

$\text{cov}(x, y)$ = covariance of x and y

σ_x = Standard deviation of x

σ_y = Standard deviation of y

$\rho(x, y)$ = correlation between x and y

The value of $\rho(x, y)$ varies between -1 to +1.

A positive value has a range from 0 to 1 where $\rho(x, y) = 1$ defines the strong positive correlation between the variables.

A negative value has a range from -1 to 0 where $\rho(x, y) = -1$ defines the strong negative correlation between the variables.

No correlation is defined if the value of $\rho(x, y) = 0$

Spotting Problems in Data with Visualization

Much of the statistical analysis is based on numerical techniques, such as *confidence intervals*, *hypothesis testing*, *regression analysis*, and so on. In many cases, these techniques are based on assumptions about the data being used. One way to determine if data confirm to these assumptions is the graphical data analysis with R, *as a graph can provide many insights into the properties of the plotted dataset*.

Graphs are useful for non-numerical data, such as colours, flavours, brand names, and more. When numerical measures are difficult or impossible to compute, graphs play an important role.

Statistical computing is done with the aim to produce high-quality graphics.

Various types of plots drawn in R programming are:

Plots with Single Variable – You can plot a graph for a single variable.

Plots with Two Variables – You can plot a graph with two variables.

Plots with Multiple Variables – You can plot a graph with multiple variables.

Special Plots – R has low and high-level graphics facilities.

First, you must complete the [Graphical Models Tutorial](#) before proceeding ahead

Plots with Single Variable

You may need to plot for a single variable in graphical data analysis with R programming. **For example** – A plot showing daily sales

values of a particular product over a period of time. You can also plot the time series for month by month sales.

The choice of plots is more restricted when you have just one variable to the plot. There are various plotting functions for single variables in R:

hist(y) – Histograms to show a frequency distribution.

plot(y) – We can obtain the values of y in a sequence with the help of the plot.

plot.ts(y) – Time-series plots.

pie(x) – Compositional plots like pie diagrams.

The types of plots available in R:

Histograms – Used to display the mode, spread, and symmetry of a set of data.

Index Plots – Here, the plot takes a single argument. This kind of plot is especially useful for error checking.

Time Series Plots – When a period of time is complete, the time series plot can be used to join the dots in an ordered set of y values.

Pie Charts – Useful to illustrate the proportional makeup of a sample in presentations.

A common mistake among beginners is getting confused between histograms and bar charts. Histograms have the response variable on the x-axis, and the y-axis shows the frequency of different values of the response. In contrast, a bar chart has the response variable on the y-axis and a categorical explanatory variable on the x-axis.

R –Pie Charts

R Programming language has numerous libraries to create charts and graphs. A pie-chart is a representation of values as slices of a circle

with different colors. The slices are labeled and the numbers corresponding to each slice is also represented in the chart.

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

Syntax

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

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

Following is the description of the parameters used –

x is a vector containing the numeric values used in the pie chart.

labels is used to give description to the slices.

radius indicates the radius of the circle of the pie chart.(value between -1 and +1).

main indicates the title of the chart.

col indicates the color palette.

clockwise is a logical value indicating if the slices are drawn clockwise or anti clockwise.

Example

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

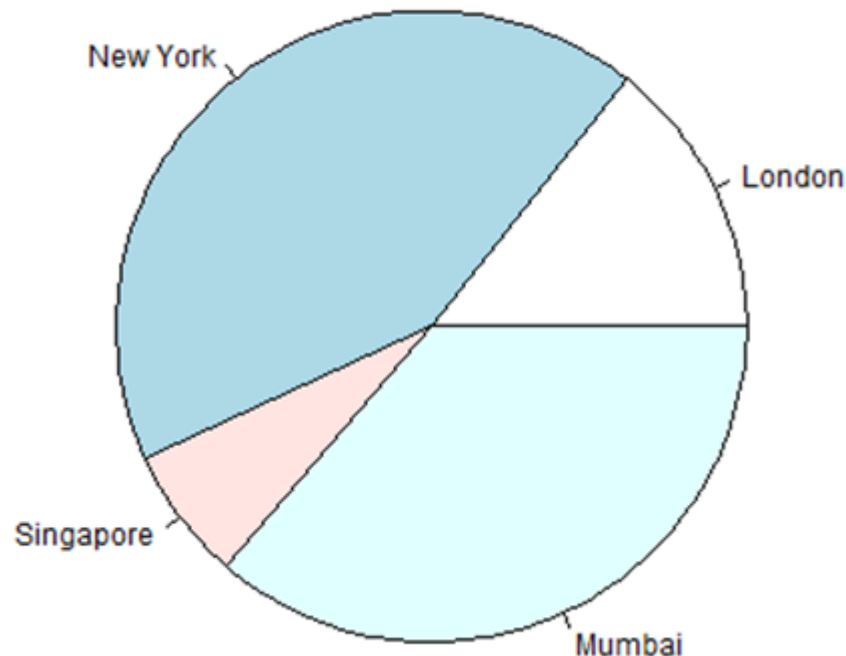
```
# Create data for the graph.
x <- c(21, 62, 10, 53)
labels <- c("London", "New York", "Singapore", "Mumbai")

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

# Plot the chart.
pie(x, labels)
```

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

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



Pie Chart Title and Colors

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

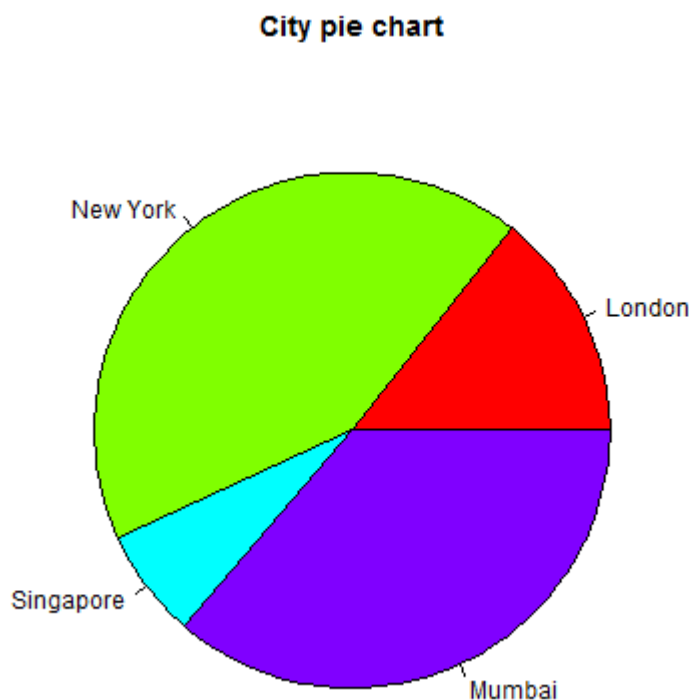
Example

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

```
# Create data for the graph.  
x <- c(21, 62, 10, 53)  
labels <- c("London", "New York", "Singapore", "Mumbai")  
  
# Give the chart file a name.  
png(file = "city_title_colours.jpg")
```

```
# Plot the chart with title and rainbow color pallet.  
pie(x, labels, main = "City pie chart", col = rainbow(length(x)))  
  
# Save the file.  
dev.off()
```

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



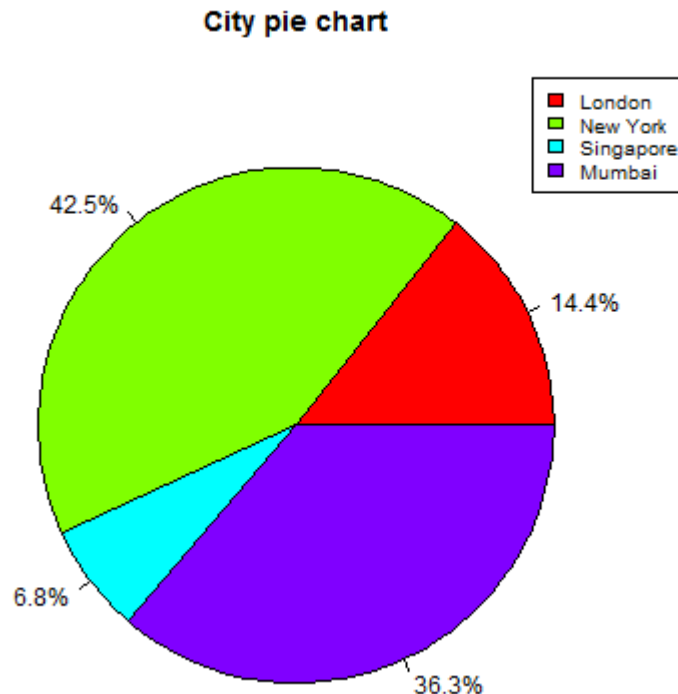
Slice Percentages and Chart Legend

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

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

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



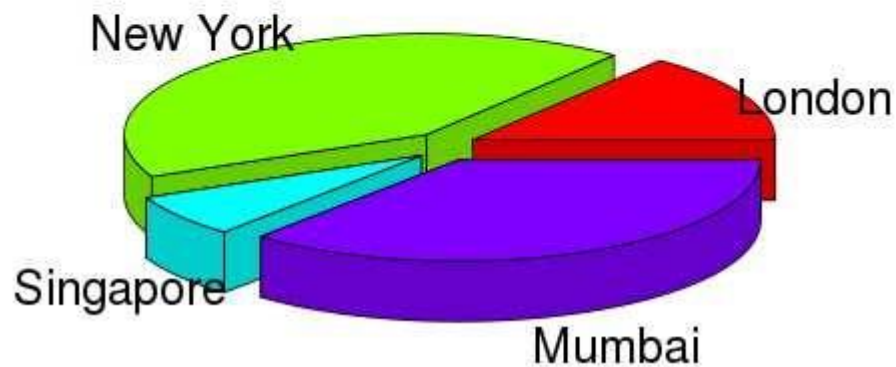
3D Pie Chart

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.

```
# 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()
```

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

Pie Chart of Countries



R Histograms

A histogram represents the frequencies of values of a variable bucketed into ranges. Histogram is similar to bar chart 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.

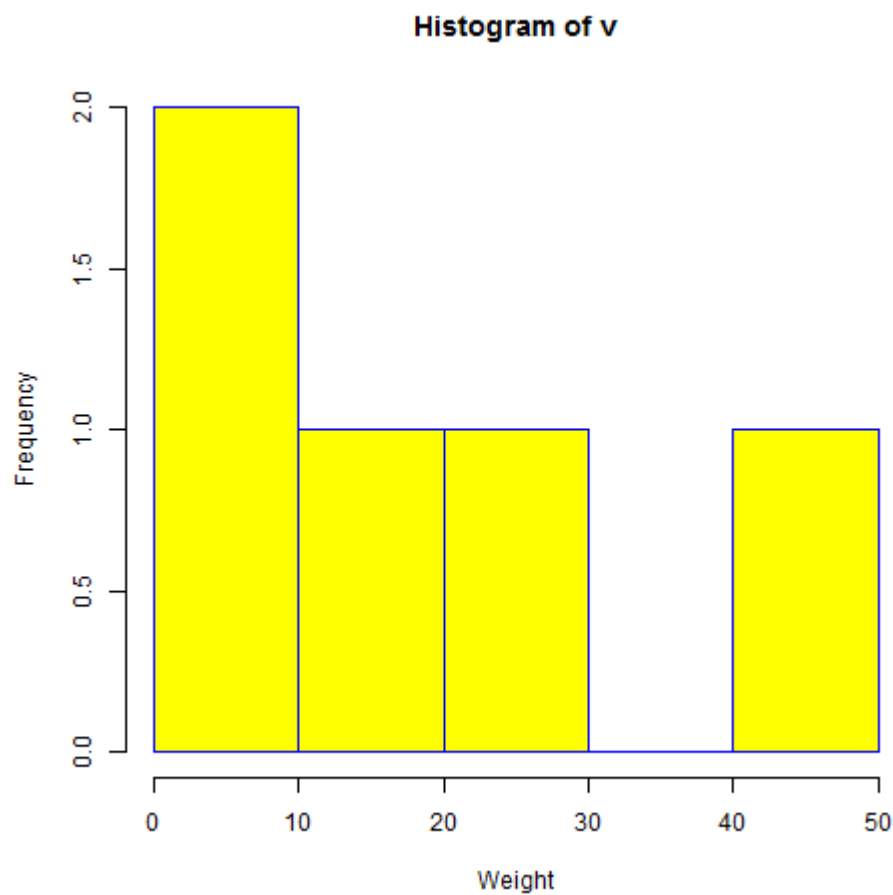
```
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# 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()
```

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



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`.

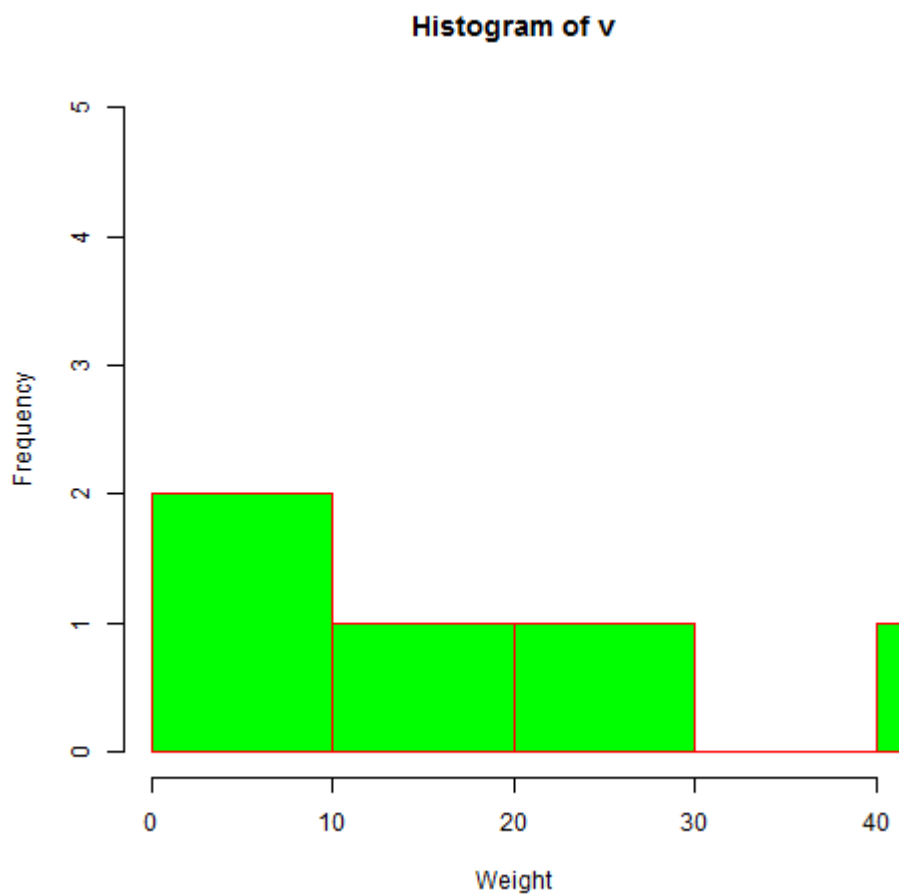
```
# Create data for the graph.
v <- c(9,13,21,8,36,22,12,41,31,33,19)

# 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()
```

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



R – Bar Charts

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

Syntax

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

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

Following is the description of the parameters used –

H is a vector or matrix containing numeric values used in bar chart.

xlab is the label for x axis.

ylab is the label for y axis.

main is the title of the bar chart.

names.arg is a vector of names appearing under each bar.

col is used to give colors to the bars in the graph.

Example

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

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

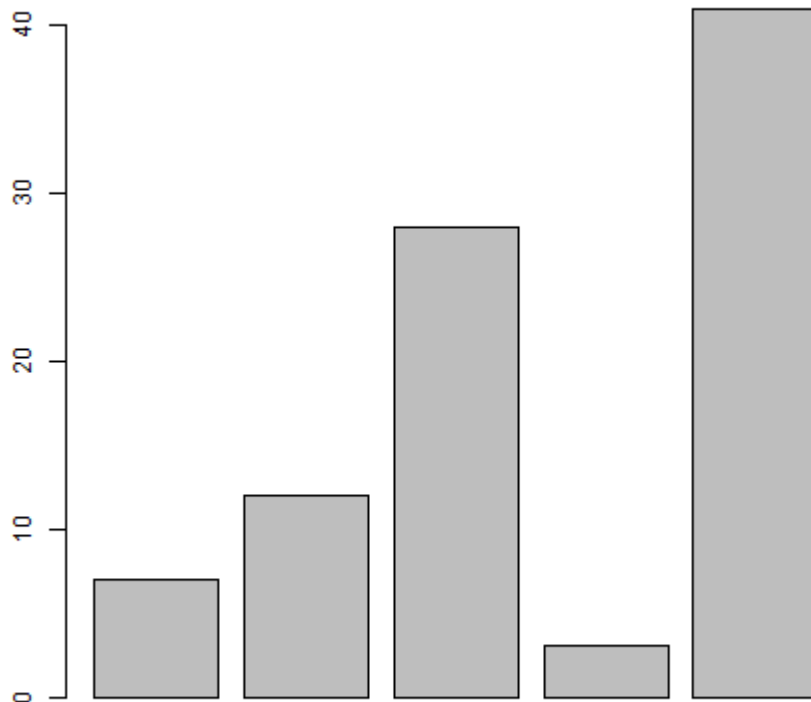
```
# Create the data for the chart
H <- c(7,12,28,3,41)

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

# Plot the bar chart
barplot(H)

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

When we execute above code, it produces following result –



Bar Chart Labels, Title and Colors

The features of the bar chart can be expanded by adding more parameters. The **main** parameter is used to add **title**.

The **col** parameter is used to add colors to the bars. The **args.name** is a vector having same number of values as the input vector to describe the meaning of each bar.

Example

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

```
# Create the data for the chart
H <- c(7,12,28,3,41)
M <- c("Mar","Apr","May","Jun","Jul")

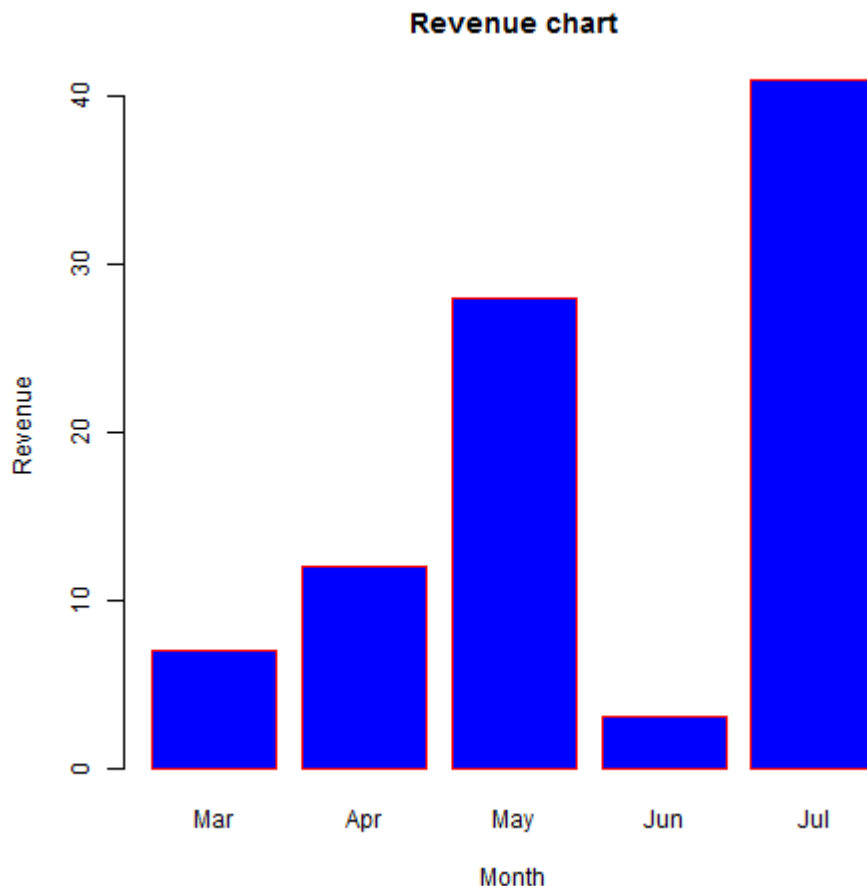
# Give the chart file a name
png(file = "barchart_months_revenue.png")

# 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()
```

When we execute above code, it produces following result –



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.

```
# 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 = TRUE)

# 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 = "revenue", col = colors)

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

# Save the file
dev.off()

```

