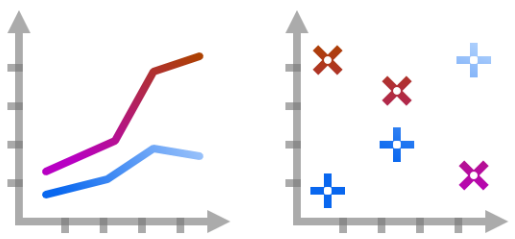
Graphics:-*Creating Graphs, The Workhorse of R Base Graphics, the plot() Function – Customizing Graphs, Saving Graphs to Files.*

Creating Graphs :- R is profoundly used for its substantial techniques for graphical interpretation of data of utmost importance of analysts**.** The primary styles are: dot plot, density plot (can be classified as histograms and kernel), line graphs, bar graphs (stacked, grouped and simple), pie charts (3D,simple and expounded), line graphs(3D,simple and expounded), box-plots(simple,notched and violin plots), bag-plots and scatter-plots (simple with fit lines, scatter-plot matrices, high-density plots and 3-D plots). The foundational function for creating graphs: plot(). This includes how to build a graph, from adding lines and points to attaching a legend.

The Workhorse of R Base Graphics: The plot() Function



The plot() function forms the foundation for much of R's base graphing operations, serving as the vehicle for producing many different kinds of graphs. plot() is a generic function, or a placeholder for a family of functions. The function that is actually called depends on the class of the object on which it is called. The basic syntax to create a line chart in R is −

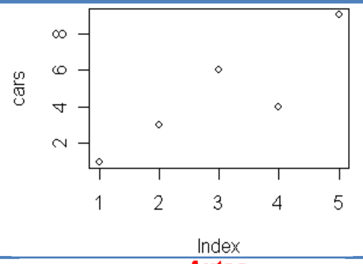
plot(v, type, col, xlab, ylab)

Following is the description of the parameters used −

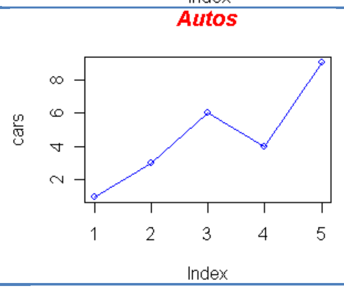
- ✓ **v** is a vector containing the numeric values.
- ✓ **type** takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.
- ✓ **xlab** is the label for x axis.
- ✓ **ylab** is the label for y axis.
- ✓ **main** is the Title of the chart.
- ✓ **col**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

Examples of plot function

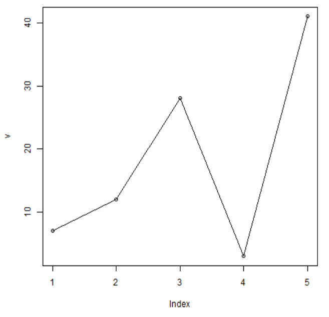
```
# Define the cars vector with 5 values
  cars <- c(1, 3, 6, 4, 9)
# Graph the cars vector with all defaults
   plot(cars)
  The default argument of type is points
```



```
# Define the cars vector with 5 values
cars <- c(1, 3, 6, 4, 9)
# Graph cars using blue points with lines
plot(cars, type="o", col="blue")
# Create a title with a red, bold/italic font
title(main="Autos",col.main="red", font.main=4)
```



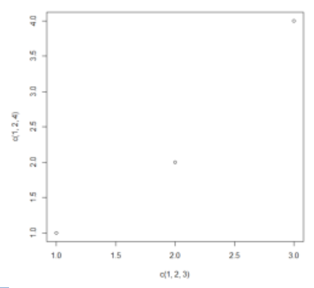
```
# Create the data for the chart.
v <- c(7,12,28,3,41)
# Give the chart file a name.
png(file = "line_chart.jpg")
# Plot the bar chart.
plot(v,type = "o")
# Save the file.
dev.off()
```



call plot() fucntion with an X vector and a Y vector, which are interpreted as a set of pairs in the (*x*,*y*) plane.

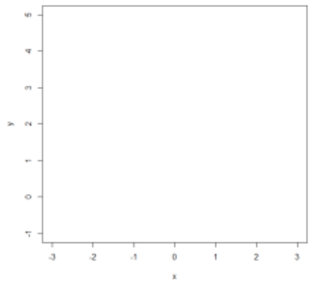
```
plot(c(1,2,3), c(1,2,4))
```

This will cause a window to pop up, plotting the points (1,1), (2,2), and (3,4), this is a very plain-Jane graph.



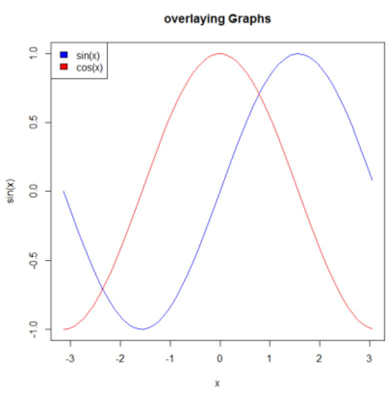
```
plot(c(-3,3), c(-1,5), type = "n", xlab="x", ylab="y")
```

#This draws axes labeled *x* and *y*. The horizontal (*x*) axis ranges from −3 to 3. The vertical (*y*) axis ranges from −1 to 5. The argument type="n" means that there is nothing in the graph itself.



Overlaying Plots:- If the plot( ) function is called many times, the current graph will be plotted in the same window and the previously existed graph will be replaced by the same. But in order to have a comparision between the results this plot is used. It is done by using the lines( ) and points( ) functions which add lines and points to the respective existing plot.

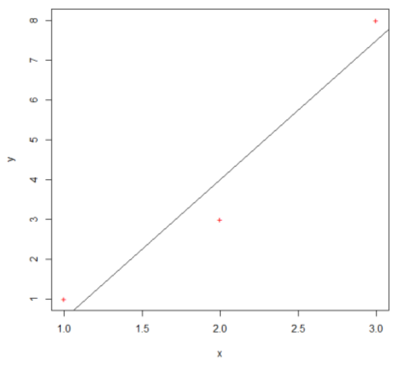
```
x <- seq(-pi,pi,0.1)
plot(x,sin(x),main="overlaying Graphs",type="l",col="blue")
lines(x,cos(x),col="red")
legend('topleft',c("sin(x)","cos(x)"),fill=c("blue","red"))
```



Abline function:- This function simply draws a straight line, with the function’s arguments treated as the intercept and slope of the line.

```
x <- c(1,2,3)
y <- c(1,3,8)
plot(x,y,col=”red”,pch=”+”)
lmout <- lm(y ~ x)
abline(lmout)
```

After the call to plot(), the graph will simply show the three points, along with the x- and y- axes with hash marks. The call to abline() then adds a line to the current graph. Now, which line is this?



As the result of the call to the linear-regression function lm( ) is a class instance containing the slope and intercept of the fitted line, as well as various other quantities that don’t concern us here. We’ve assigned that class instance to lmout. The slope and intercept will now be in lmout$coefficients.

- Some of the coloring functions in Graphs.

Function	Usage	Example
colors( )	Returns the built-in color names which R knows about.	> col <- colors() [234] > col [1] "gray81"

rgb( )	This function creates colors corresponding to the given intensities (between 0 and max) of the red, green and blue primaries. It returns hex code of the color	> rgb(1,0,1) [1] "#FF00FF" > rgb(33,64,123,max=255) [1] "#21407B" > rgb(0.3,0.7,0.5) [1] "#4CB280"
cm.colors( )	Create a vector of n contiguous colors.	> cm.colors(1) [1] "#80FFFFFF"
rainbow( )	Create a vector of n contiguous colors.	> rainbow(3) [1] "#FF0000FF" "#00FF00FF" "#0000FFFF"
heat.colors( )	Create a vector of n contiguous colors.	> heat.colors(1) [1] "#FF0000FF"
terrain.colors( )	Create a vector of n contiguous colors.	> terrain.colors(2) [1] "#00A600FF" "#F2F2F2FF"
par( )	par can be used to set or query graphical parameters. Parameters can be set by specifying them as arguments to par in tag = value form, or by passing them as a list of tagged values.	>par(mfrow=c(1,2)) # set the plotting area into a 1*2 array so 2 plots can be used https://www.youtube.com/watch?v=Z3V4Pbxeahg

Bar plot:- 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.

A bar graph is a chart that uses bars to show comparisons between categories of data. A bar graph will have two axes. One axis will describe the types of categories being compared, and the other will have numerical values that represent the values of the data. It does not matter which axis is which, but it will determine what bar graph is shown. If the descriptions are on the horizontal axis, the bars will be oriented vertically, and if the values are along the horizontal axis, the bars will be oriented horizontally.

Syntax:- *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.

Types of Bar Plot:-

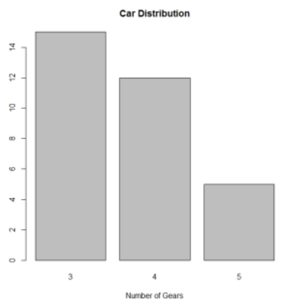
There are four types of bar diagrams, they are

1. Simple Bar plot
2. Multilple Bar plot
3. Sub-divided Bar plot or Component Bar plot
4. Deviation Bars

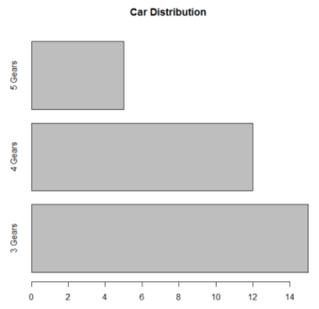
Examples on bar plot

Simple Bar plot:- To compare two or more independent variables. Each variable will relate to a fixed value. The values are positive and therefore, can be fixed to the horizontal value.

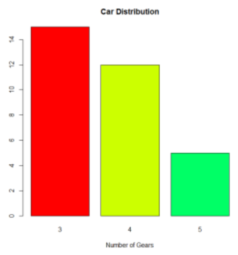
```
# Simple Bar Plot
counts <- table(mtcars$gear)
barplot(counts, main="Car Distribution",
xlab="Number of Gears")
```



```
# Simple Horizontal Bar Plot with Added Labels
counts <- table(mtcars$gear)
barplot(counts,
          main="Car Distribution",
         horiz=TRUE,
        names.arg=c("3 Gears", "4 Gears", "5 Gears"))
```



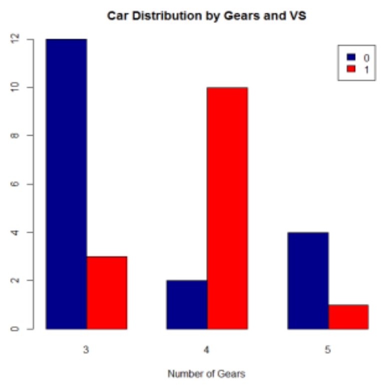
```
# Simple Bar Plot
counts <- table(mtcars$gear)
barplot(counts, main="Car Distribution",
       xlab="Number of Gears",col=rainbow(5))
```



# Grouped Bar Plot or multiple bar plot

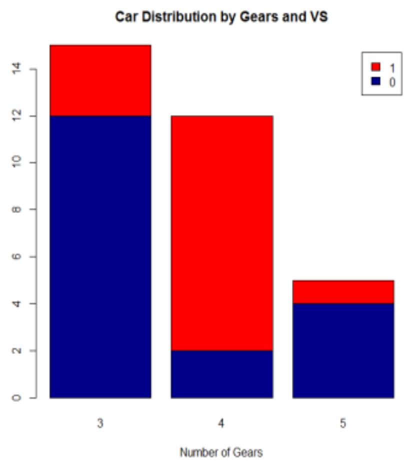
Multiple bar chart is an extension of simple bar chart. Grouped bars are used to represent related sets of data. For example, imports and exports of a country together are shown in multiple bar chart. Each bar in a group is shaded or coloured differently for the sake of distinction.

```
counts <- table(mtcars$vs, mtcars$gear)
barplot(counts, main="Car Distribution by Gears and VS",
xlab="Number of Gears", col=c("darkblue","red"),
       legend = rownames(counts), beside=TRUE)
```



Sub-divided Bar Diagram:- This chart consists of bars which are sub-divided into two or more parts. This type of diagram shows the variation in different components within each class as well as between different classes. Sub-divided bar plot is also known as component bar chart or staked chart.

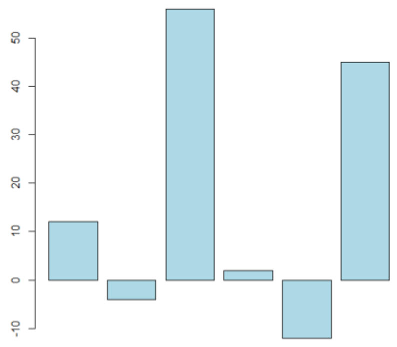
```
# Stacked Bar Plot with Colors and Legend
counts <- table(mtcars$vs, mtcars$gear)
barplot(counts, main="Car Distribution by Gears and VS",
xlab="Number of Gears", col=c("darkblue","red"),
       legend = rownames(counts))
```



# Deviation Bar Plot

A graph displays a deviation relationship when it features how one or more sets of quantitative values differ from a reference set of values. The graph does this by directly expressing the differences between two sets of values

Ex.:Deviation bars are used to represent net quantities - excess or deficit i.e. net profit, net loss, net exports or imports, swings in voting etc. Such bars have both positive and negative values. Positive values lie above the base line and negative values lie below it.



```
cars <- c(12,-4,56,2,-12,45)
barplot(cars,col="light blue")
```

Advantages

- *Show each data category in a frequency distribution*
- *Display relative numbers/proportions of multiple categories*
- *Summarize a large amount of data in a visual, easily intepretable form*
- *Make trends easier to highlight than tables do*
- *Estimates can be made quickly and accurately*
- *Permit visual guidance on accuracy and reasonableness of calculations*
- *Accessible to a wide audience*

Disadvantages

- *Often require additional explanation*
- *Fail to expose key assumptions, causes, impacts and patterns*
- *Can be easily manipulated to give false impressions*

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 pie chart, the circle is drawn with radii proportional to the square root of the quantities to be represented because the area of a circle is given by 2pr². The sectors are coloured and shaded differently. To construct a pie chart, we draw a circle with some suitable radius (square root of the total). The angles are calculated for each sector as follows:

$$\text{Angles for each sector} = \frac{\text{Component Part}}{\text{Total}} \times 360^{o}$$

Syntax:- **pie(x, labels, radius, main, col, clockwise)**

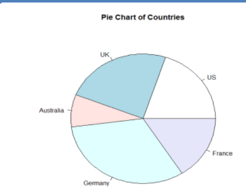
Following is the description of the parameters used

- **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.

Examples of pie chart

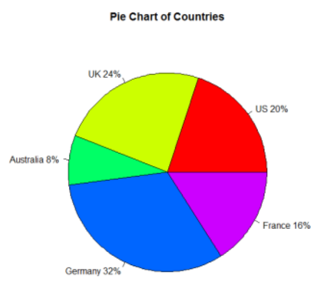
# Simple Pie Chart

```
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie(slices, labels = lbls, main="Pie Chart of Countries")
```



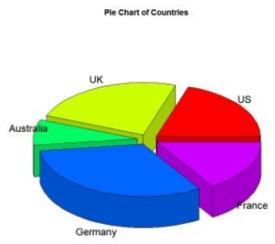
# Pie Chart with Percentages

```
slices <- c(10, 12, 4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pct <- round(slices/sum(slices)*100)
lbls <- paste(lbls, pct) # add percents to labels
lbls <- paste(lbls,"%",sep="") # ad % to labels
pie(slices,labels = lbls, col=rainbow(length(lbls)), main="Pie Chart of
Countries")
```



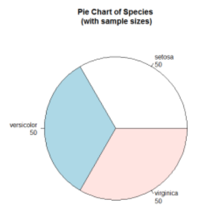
# 3D Exploded Pie Chart
The pie3D( ) function in the plotrix package provides 3D exploded pie charts.

```
library(plotrix)
slices <- c(10, 12, 4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
pie3D(slices,labels=lbls,explode=0.1, main="Pie Chart of Countries ")
```



# Pie Chart from data frame with Appended Sample Sizes

```
mytable <- table(iris$Species)
lbls <- paste(names(mytable), "\n", mytable, sep="")
pie(mytable, labels = lbls,
   main="Pie Chart of Species\n (with sample sizes)")
```



Advantages

- *Display relative proportions of multiple classes of data.*
- *Size of the circle can be made proportional to the total quantity it represents.*
- *Summarize a large data set in visual form.*
- *Be visually simpler than other types of graphs.*
- *Permit a visual check of the reasonableness or accuracy of calculations.*

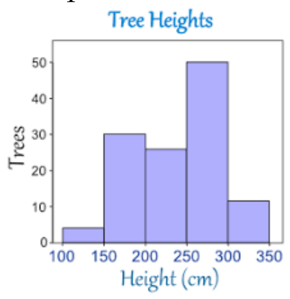
Disadvantages

- *Do not easily reveal exact values*
- *Many pie charts may be needed to show changes over time*
- *Fail to reveal key assumptions, causes, effects, or patterns*
- *Be easily manipulated to yield false impressions*

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:- **hist(v,main,xlab,xlim,ylim,breaks,col,border)**



Following is the description of the parameters used −

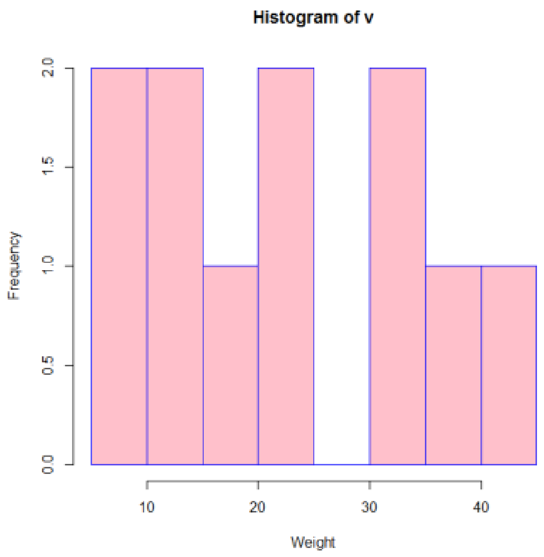
- v is a vector containing numeric values used in histogram.
- main indicates title of the chart.
- col is used to set color of the bars.
- border is used to set border color of each bar.
- xlab is used to give description of x-axis.
- xlim is used to specify the range of values on the x-axis.
- ylim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

- mids: center point of multiple cells.
- density: cell density

Examples of histogram

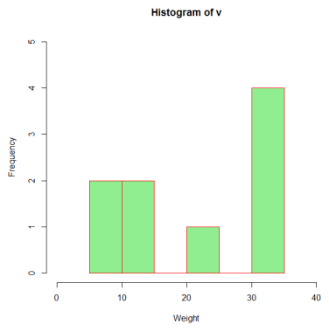
#Simple histogram

```
v <-  c(9,13,21,8,36,22,12,41,31,33,19)
h <- hist(v,xlab = "Weight",col = "pink",border = "blue")
> h
$breaks
[1]  5 10 15 20 25 30 35 40 45

$counts
[1] 2 2 1 2 0 2 1 1

$density
[1]     0.03636364     0.03636364     0.01818182     0.03636364
0.00000000 0.03636364 0.01818182
[8] 0.01818182

$mids
[1]  7.5 12.5 17.5 22.5 27.5 32.5 37.5 42.5

$xname
[1] "v"

$equidist
[1] TRUE

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



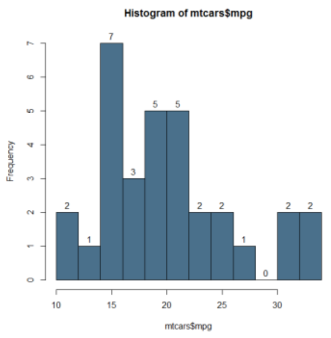
To specify the range of values allowed in X axis and Y axis, we can use the xlim and ylim parameters.The width of each of the bar can be decided by using breaks.

```
v <-  c(9,13,31,8,31,22,12,31,35)
hist(v,xlab = "Weight",col = "light green",
border = "red", xlim = c(0,40),  ylim = c(0,5),breaks = 5)
```



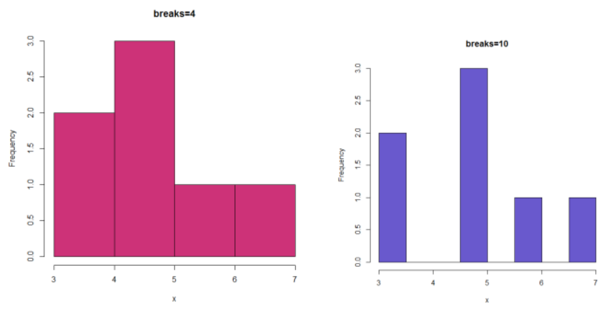
The following example utilizes the function text( ) which add text to a plot and return values to place the count above each cell.

```
h<-hist(mtcars$mpg, breaks=12, col="skyblue4")
text(h$mids,h$counts,labels = h$counts,adj=c(0.5,-0.5))
```



Break parameter tells the number of cells required in the histogram plot.

```
x<- c(5,3,5,7,3,6,5)
hist(x,breaks = 4,col="violetred3", ,main="breaks=4" )

hist(x,breaks = 10,col="slateblue3",main="breaks=10" )
```

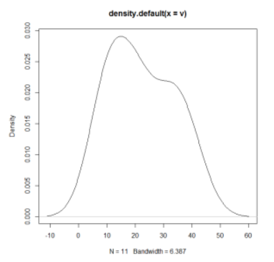


Kernel Density Plots:- Kernal density plots are usually a much more effective way to view the distribution of a variable. Create the plot using plot(density(x)) where x is a numeric vector.

Examples

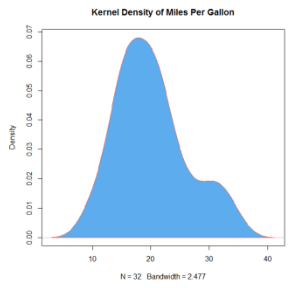
# Kernel Density Plot

```
v <-  c(9,13,21,8,36,22,12,41,31,33,19)
# returns the density data
d <- density(v)
# plots the results
plot(d)
```



# Filled Density Plot

```
d <- density(mtcars$mpg)
plot(d, main="Kernel Density of Miles Per Gallon")
polygon(d, col="steelblue2", border="tomato1")
```



Advantages

- *Visually strong.*
- *Can compare to normal curve.*
- *Usually vertical axis is a frequence count of item falling in to each category.*

Disadvantages

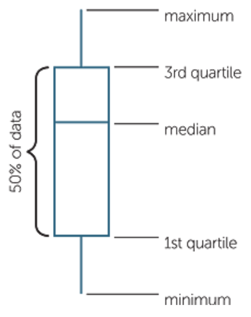
- *Cannot read exact values because data is grouped in categories.*
- *More difficult to compare two data sets.*
- *Use only with continuous data*

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:- **boxplot(x, data, notch, varwidth, names, main)**

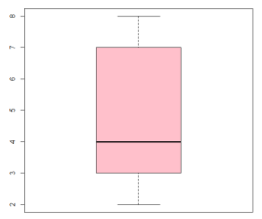
Following is the description of the parameters used −

- **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.

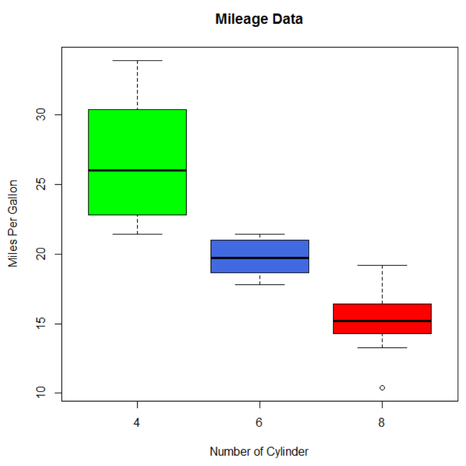


Examples

```
x <- c(7,3,2,4,8)
boxplot(x,col="pink")
```



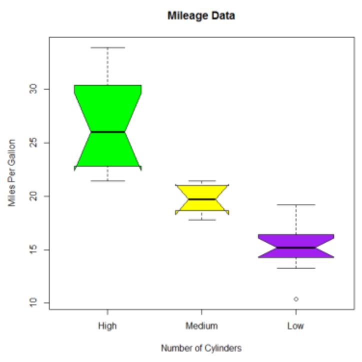
```
> input <- mtcars[,c('mpg','cyl')]
> print(head(input))
                    mpg cyl
Mazda RX4          21.0   6
Mazda RX4 Wag      21.0   6
Datsun 710         22.8   4
Hornet 4 Drive     21.4   6
Hornet Sportabout  18.7   8
Valiant            18.1   6

boxplot(mpg ~ cyl, data = mtcars,
           xlab ="Number of Cylinder",
           ylab = "Miles Per Gallon",
          main = "Mileage Data",
         col= c("green","royalblue","red"))
```



We can draw boxplot with notch to find out how the medians of different data groups match with each other.

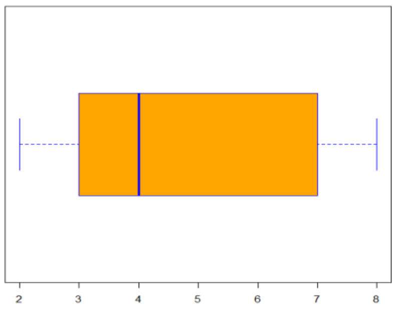
```
boxplot(mpg ~ cyl, data = mtcars,
      xlab = "Number of Cylinders",
      ylab = "Miles Per Gallon",
      main = "Mileage Data",
      notch = TRUE,
      varwidth = TRUE,
      col = c("green","yellow","purple"),
      names = c("High","Medium","Low"))
```

a~z , where the value of z determines the value of a.



We can draw boxplot horizontally by making horizontal as TRUE, the default value is FALSE.

```
x <- c(7,3,2,4,8)
boxplot(x,col="orange",horizontal = TRUE,border = "blue")
> b
$stats
     [,1]
[1,]    2
[2,]    3
[3,]    4
[4,]    7
[5,]    8

$n
[1] 5

$conf
         [,1]
[1,] 1.17361
[2,] 6.82639

$out
numeric(0)
$group
numeric(0)

$names
[1] "1"
```



- ***n:*** *It includes number of used to draw box-plot exculding the NA's.*
- ***conf:*** *It represents the lower and upper extremes of notch and the out-value of outliers.*
- ***group:*** *It represents same length vector as out whose elements indicate to which group the outlier belongs and*
- ***names:*** *names vector for a group*

Advantages:

- A box plot is a good way to summarize large amounts of data.
- It displays the range and distribution of data along a number line.

- Box plots provide some indication of the data's symmetry and skew-ness.
- Box plots show outliers.

Disadvantages

- Original data is not clearly shown in the box plot; also, mean and mode cannot be identified in a box plot.
- Exact values not retained.

Customizing Graphs:-

a) *Changing Character Sizes: (The cex Option)* The cex (for *character expand*) function allows to expand or shrink characters within a graph, which can be very useful. You can use it as a named parameter in various graphing functions. For instance, you may wish to draw the text "abc" at some point, say (2.5,4), in your graph but with a larger font,in order to call attention to this particular text.

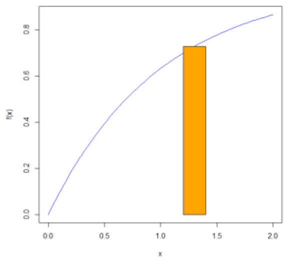
Example:- *text(2.5,4,"abc",cex = 1.5)*

This prints the same text as in our earlier example but with characters 1.5 times the normal size.

b) Changing the Range of Axes: The xlim and ylim Options

The ranges on the x- and y-axes of a plot can be broader or narrower than the default. This is especially useful while displaying several curves in the same graph. Axes can be modified by specifying the xlim and/or ylim parameters in a call to plot( ) or points( ). For example, ylim=c(0,90000) specifies a range on the y-axis of 0 to 90,000.

c) Adding a Polygon: The polygon() Function



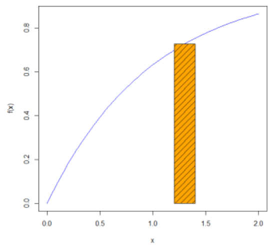
polygon() function is used to draw arbitrary polygonal objects. For example, the following code draws the graph of the function $f(x) = 1 - e^{-x}$ and then adds a rectangle that approximates the area under the curve from x = 1.2 to x = 1.4.

```
f <- function(x) return(1-exp(-x))
curve(f,0,2,col="blue")
polygon(c(1.2,1.4,1.4,1.2),c(0,0,f(1.3),f(1.3)),col="orange")
```

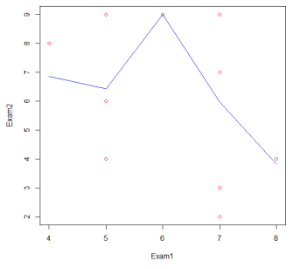
In the call to polygon() here, the first argument is the set of x- coordinates for the rectangle, and the second argument specifies the y-coordinates. The third argument specifies that the rectangle in this case should be shaded in solid gray.

As another example, we could use the density argument to fill the rectangle with striping. This call specifies 10 lines per inch:

```
polygon(c(1.2,1.4,1.4,1.2),c(0,0,f(1.3),f(1.3)),density=10)
```



d) Smoothing Points: The lowess() and loess() Functions



Just plotting a cloud of points, connected or not, may give you nothing but an uninformative mess. In many cases, it is better to smooth out the data by fitting a nonparametric regression estimator such as lowess(). Let's do that for our test score data. We'll plot the scores of exam 2 against those of exam 1:

```
testscore <- data.frame(c(4,6,8,5,5,8,7,7,7,7,5,5,5),c(8,9,4,6,4,4,2,7,3,9,9,6,4))
plot(testscore,xlab="Exam1",ylab="Exam2",col="red")
lines(lowess(testscore),col="blue")
```

Newer alternative to lowess() is loess(). The two functions are similar but have different defaults and other options.

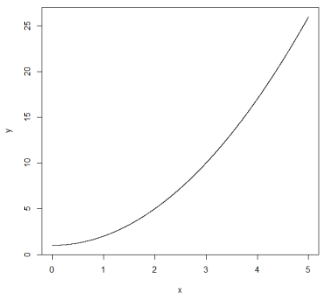
loess():-Fit a polynomial surface determined by one or more numerical predictors, using local fitting.

e) Graphing Explicit Functions

To plot the function g(t) = $(t^2 + 1)^{0.5}$ for t between 0 and 5.

You could use the following R code:

```
g <- function(t) { return (t^2+1)^0.5 }
x <- seq(0,5,length=10000)
y <- g(x)
plot(x,y,type="l")
```



But you could avoid some work by using the curve() function, which basically uses the same method:

```
curve((x^2+1)^0.5,0,5)
```

If you are adding this curve to an existing plot, use the add argument:

```
curve((x^2+1)^0.5,0,5,add=T)
```

The optional argument n has the default value 101, meaning that the function will be evaluated at 101 equally spaced points in the specified range of x.

Saving Graphs:-The R graphics display can consist of various graphics devices. The default device is the screen. Inorder to save a graph to a file, you must set up another device.
The graph can be saved in a variety of formats from the menu *File -> Save As.*
The graph can also be saved using one of the following functions.

Function	Output to
pdf("mygraph.pdf")	pdf file
win.metafile("mygraph.wmf")	windows metafile
png("mygraph.png")	png file
jpeg("mygraph.jpg")	jpeg file
bmp("mygraph.bmp")	bmp file
postscript("mygraph.ps")	postscript file

Let's go through the basics of R graphics devices first to introduce R graphics device concepts, and then discuss a second approach that is much more direct and convenient.

```
> pdf("d12.pdf")
```

This opens the file *d12.pdf*. We now have two devices open, as we can confirm:

```
> dev.list()
 X11 pdf
  2    3
```

The screen is named X11 when R runs on Linux. (It's named windows on Windows systems.) It is device number 2 here. Our PDF file is device number 3. Our active device is the PDF file:

```
> dev.cur()
pdf
3
```

All graphics output will now go to this file instead of to the screen. But what if we wish to save what's already on the screen?

Saving the Displayed Graph:-One way to save the graph currently displayed on the screen is to reestablish the screen as the current device and then copy it to the PDF device, which is 3 in our example, as follows:

```
> dev.set(2)
X11
2
> dev.copy(which=3)
pdf
3
```

But actually, it is best to set up a PDF device as shown earlier and then rerun whatever analyses led to the current screen. This is because the copy operation can result in distortions due to mismatches between screen devices and file devices.

Closing an R Graphics Device:-Note that the PDF file we create is not usable until we close it, which we do as follows:

```
> dev.set(3)
pdf
3
> dev.off()
X11
2
```

You can also close the device by exiting R, if you're finished working with it. But in future versions of R, this behavior may not exist, so it's probably better to proactively close.

Example:

```
# 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,col=c("green","pink","skyblue"))

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