**GRAPHICS**

In R, you can create a wide variety of plots and graphs to visualize data using built-in functions and libraries. \

R offers extensive capabilities for data visualization, making it a popular choice for data analysis and scientific research.

Here are some commonly used tools and packages for creating graphics in R:

Graphs in R language is a preferred feature which is used to create various types of graphs and charts for visualizations. R language supports a rich set of packages and functionalities to create the graphs using the input data set for data analytics. The most commonly used graphs in the R language are scattered plots, box plots, line graphs, pie charts, histograms, and bar charts. R graphs support both two dimensional and three-dimensional plots for exploratory data analysis.There are R function like plot(), barplot(), pie() are used to develop graphs in R language. R package like ggplot2 supports advance graphs functionalities.

### Types of Graphs in R

#### 1. Histogram

A histogram is a graphical tool that works on a single variable. Numerous variable values are grouped into bins, and a number of values termed as the frequency are calculated. This calculation is then used to plot frequency bars in the respective beans. The height of a bar is represented by frequency.

In R**,**we can employ the hist() function as shown below, to generate the histogram. A simple histogram of tree heights is shown below.

**Code:**

hist(trees$Height, breaks = 10, col = "orange", main = "Histogram of Tree heights", xlab = "Height Bin")

**Output:**

To understand the trend of frequency, we can add a density plot over the above histogram. This offers more insights into data distribution, skewness, kurtosis, etc. The following code does this, and the output is shown following the code.

**Code:**

hist(trees$Height, breaks = 10, col = "orange",  
+ main = "Histogram of Tree heights with Kernal Denisty plot",  
+ xlab = "Height Bin", prob = TRUE)

**Output:**

#### 2. Scatterplot

This plot is a simple chart type, but a very crucial one having tremendous significance. The chart gives the idea about a correlation amongst variables and is a handy tool in an exploratory analysis.

The following code generates a simple Scatterplot chart. We have added a trend line to it, to understand the trend, the data represents.

**Code:**

attach(trees)  
plot(Girth, Height, main = "Scatterplot of Girth vs Height", xlab = "Tree Girth", ylab = "Tree Height")  
abline(lm(Height ~ Girth), col = "blue", lwd = 2)

**Output:**

The chart created by the following code shows that there exists a good correlation between tree girth and tree volume.

**Code:**

plot(Girth, Volume, main = "Scatterplot of Girth vs Volume", xlab = "Tree Girth", ylab = "Tree Volume")  
abline(lm(Volume ~ Girth), col = "blue", lwd = 2)

**Output:**

##### Scatterplot Matrices

R allows us to compare multiple variables at a time because of it uses scatterplot matrices. Implementing the visualization is quite simple, and can be achieved using pairs() function as shown below.

**Code:**

pairs(trees, main = "Scatterplot matrix for trees dataset")

**Output:**

##### Scatterplot3d

They make visualization possible in three dimensions which can help to understand the relationship between multiple variables. So, to make scatterplots available in 3d, firstly scatterplot3d package must be installed. So, the following code generates a 3d graph as shown below the code.

**Code:**

library(scatterplot3d)  
attach(trees)  
scatterplot3d(Girth, Height, Volume, main = "3D Scatterplot of trees dataset")

We can add dropping-lines and colors, using the below code. Now, we can conveniently distinguish between different variables.

**Code:**

scatterplot3d(Girth, Height, Volume, pch = 20, highlight.3d = TRUE,  
+ type = "h", main = "3D Scatterplot of trees dataset")

**Output:**

#### 3. Boxplot

Boxplot is a way of visualizing data through boxes and whiskers. Firstly, variable values are sorted in ascending order and then the data is divided into quarters.

The box in the plot is the middle 50% of the data, known as IQR. The black line in the box represents the median.

**Code:**

boxplot(trees, col = c("yellow", "red", "cyan"), main = "Boxplot for trees dataset")

**Output:**

A variant of the boxplot, with notches, is as shown below.

**Code:**

boxplot(trees, col = "orange", notch = TRUE, main = "Boxplot for trees dataset")

**Output:**

#### 4. Line Chart

Line charts are useful when comparing multiple variables. They help us relationship between multiple variables in a single plot. In the following illustration, we will try to understand the trend of three tree features. So, as shown in the below code, initially, and the line chart for Girth is plotted using plot() function. Then line charts for Height and Volume are plotted on the same plot using lines() function.

The “ylim” parameter in plot() function has been, to accommodate all three line charts properly. Having legend is important here, as it helps understand which line represents which variable. In the legend “lty = 1:1” parameter means that we have the same line type for all variables, and “cex” represents the size of the points.

**Code:**

plot(Girth, type = "o", col = "red", ylab = "", ylim = c(0, 110),  
+ main = "Comparison amongst Girth, Height, and Volume of trees")  
lines(Height, type = "o", col = "blue")  
lines(Volume, type = "o", col = "green")  
legend(1, 110, legend = c("Girth", "Height", "Volume"),  
+ col = c("red", "blue", "green"), lty = 1:1, cex = 0.9)

**Output:**

#### 5. Dot plot

This visualization tool is useful if we want to compare multiple categories against a certain measure. For the below illustration, mtcars dataset has been used. The dotchart() function plots displacement for various car models as below.

**Code:**

attach(mtcars)  
dotchart(disp, labels = row.names(mtcars), cex = 0.75,  
+ main = "Displacement for various Car Models", xlab = "Displacement in Cubic Inches")

**Output:**

So, now we will sort the dataset on displacement values, and then plot them by different gears using dotchart() function.

**Code:**

m <- mtcars[order(mtcars$disp),] m$gear <- factor(m$gear)  
m$color[m$gear == 3] <- "darkgreen"  
m$color[m$gear == 4] <- "red"  
m$color[m$gear == 5] <- "blue"  
dotchart(m$disp, labels = row.names(m), groups = m$gear, color = m$color, cex = 0.75, pch = 20,  
+ main = "Displacement for Car Models", xlab = "Displacement in cubic inches")

**Output:**

We can add dropping-lines and colors, using the below code. Now, we can conveniently distinguish between different variables.

**Code:**

scatterplot3d(Girth, Height, Volume, pch = 20, highlight.3d = TRUE,  
+ type = "h", main = "3D Scatterplot of trees dataset")

**Output:**

#### 3. Boxplot

Boxplot is a way of visualizing data through boxes and whiskers. Firstly, variable values are sorted in ascending order and then the data is divided into quarters.

The box in the plot is the middle 50% of the data, known as IQR. The black line in the box represents the median.

**Code:**

boxplot(trees, col = c("yellow", "red", "cyan"), main = "Boxplot for trees dataset")

**Output:**

A variant of the boxplot, with notches, is as shown below.

**Code:**

boxplot(trees, col = "orange", notch = TRUE, main = "Boxplot for trees dataset")

**Output:**

#### 4. Line Chart

Line charts are useful when comparing multiple variables. They help us relationship between multiple variables in a single plot. In the following illustration, we will try to understand the trend of three tree features. So, as shown in the below code, initially, and the line chart for Girth is plotted using plot() function. Then line charts for Height and Volume are plotted on the same plot using lines() function.

The “ylim” parameter in plot() function has been, to accommodate all three line charts properly. Having legend is important here, as it helps understand which line represents which variable. In the legend “lty = 1:1” parameter means that we have the same line type for all variables, and “cex” represents the size of the points.

**Code:**

plot(Girth, type = "o", col = "red", ylab = "", ylim = c(0, 110),  
+ main = "Comparison amongst Girth, Height, and Volume of trees")  
lines(Height, type = "o", col = "blue")  
lines(Volume, type = "o", col = "green")  
legend(1, 110, legend = c("Girth", "Height", "Volume"),  
+ col = c("red", "blue", "green"), lty = 1:1, cex = 0.9)

**Output:**

#### 5. Dot plot

This visualization tool is useful if we want to compare multiple categories against a certain measure. For the below illustration, mtcars dataset has been used. The dotchart() function plots displacement for various car models as below.

**Code:**

attach(mtcars)  
dotchart(disp, labels = row.names(mtcars), cex = 0.75,  
+ main = "Displacement for various Car Models", xlab = "Displacement in Cubic Inches")

**Output:**

So, now we will sort the dataset on displacement values, and then plot them by different gears using dotchart() function.

**Code:**

m <- mtcars[order(mtcars$disp),] m$gear <- factor(m$gear)  
m$color[m$gear == 3] <- "darkgreen"  
m$color[m$gear == 4] <- "red"  
m$color[m$gear == 5] <- "blue"  
dotchart(m$disp, labels = row.names(m), groups = m$gear, color = m$color, cex = 0.75, pch = 20,  
+ main = "Displacement for Car Models", xlab = "Displacement in cubic inches")

**Output:**

# R Programming/Graphics

**R** includes at least three graphical systems, the standard **graphics** package, the **lattice** package for Trellis graphs and the *grammar-of-graphics* **ggplot2** package[[2]](https://en.wikibooks.org/wiki/R_Programming/Graphics" \l "cite_note-wickham-2). R has good graphical capabilities but there are some alternatives like [gnuplot](https://en.wikipedia.org/wiki/gnuplot" \o "w:gnuplot).

## Interactive Graphics

This section discuss some ways to draw graphics without using R scripts.

The **playwith** package provides a graphical user interface to customize the graphs, add a title, a grid, some text, etc and it exports the R code you need if you want to replicate the analysis If you want to know more, you can have a look at the screenshots on the website ([link](https://code.google.com/p/playwith/wiki/Screenshots)). See also the example on "R you Ready" .This package require GTK+ libraries.

library("playwith")

playwith(plot(x1))

There is also a graphical user interface **GrapheR** which makes it very easy to draw graphs for beginners[[4]](https://en.wikibooks.org/wiki/R_Programming/Graphics" \l "cite_note-4). This solution is cross-platform.

> library(GrapheR)

**latticist** ([link](https://code.google.com/p/latticist/)) is another similar project.

Note also that some graphical user interface such as RKward and R Commander makes it easy to draw graphs.

## Standard R graphs

In this section we present what you need to know if you want to customize your graphs in the default graph system.

* plot() is the main function for graphics. The arguments can be a single point such as 0 or c(.3,.7), a single vector, a pair of vectors or many other R objects.
* par() is another important function which defines the default settings for plots.
* There are many other plot functions which are specific to some tasks such as hist(), boxplot(), etc. Most of them take the same arguments as the plot() function.

> N <- 10^2

> x1 <- rnorm(N)

> x2 <- 1 + x1 + rnorm(N)

> plot(0)

> plot(0,1)

> plot(x1)

> plot(x1,x2) # scatter plot x1 on the horizontal axis and x2 on the vertical axis

> plot(x2 ~ x1) # the same but using a formula (x2 as a function of x1)

> methods(plot) # show all the available methods for plot (depending on the number of loaded packages).

### Titles, legends and annotations

#### Titles

main gives the main title, sub the subtitle. They can be passed as argument of the plot() function or using the title() function. xlab the name of the x axis and ylab the name of the y axis.

plot(x1,x2, main = "Main title", sub = "sub title" , ylab = "Y axis", xlab = "X axis")

plot(x1,x2 , ylab = "Y axis", xlab = "X axis")

title(main = "Main title", sub = "sub title" )

The size of the text can be modified using the parameters cex.main, cex.lab, cex.sub, cex.axis. Those parameters define a *scaling factor*, ie the value of the parameter multiply the size of the text. If you choose cex.main=2 the main title will be twice as big as usual.

#### Legend

legend(). The position can be "bottomleft", "bottomright", "topleft", "topright" or exact coordinates.

plot(x1, type = "l", col = 1, lty = 1)

lines(x2, col = 2, lty = 2)

legend("bottomleft", legend = c("x1","x2"), col = 1:2, lty = 1:2)

#### Text in the margin

mtext() puts some texts in the margin. The margin can be at the bottom (1), the left (2), the top (3) or the right (4).

plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 1) # the bottom

plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 2) # the left

plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 3) # the top

plot(x1, type = "l", col = 1, lty = 1) ; mtext("some text", side = 4) # the right margin

#### Text in the graph

text()

#### Mathematical annotations

We can add mathematical symbols using expression() and makes some substitution in a formula using substitute().

?plotmath # gives help for mathematical annotations

### Types

The type of a plot can be :

* n for none (nothing is printed),
* p for points,
* l for lines,
* b for both,
* o for both overlayed,
* h for histogram-like
* and s/S for steps.

|  |  |
| --- | --- |
| **R code** | **Output** |
| x1 <- rnorm(50)  png("plottype.png")  par(mfrow = c(2,2))  plot(x1, type = "p", main = "points", ylab = "", xlab = "")  plot(x1, type = "l", main = "lines", ylab = "", xlab = "")  plot(x1, type = "b", main = "both", ylab = "", xlab = "")  plot(x1, type = "o", main = "both overplot", ylab = "", xlab = "")  dev.off() | undefined |

**Axes**

The default output print the axes. We can remove them with axes=FALSE. We can also change them using the axis() function.

> plot(x1,x2,axes=FALSE)

>

> plot(x1,x2,axes=FALSE)

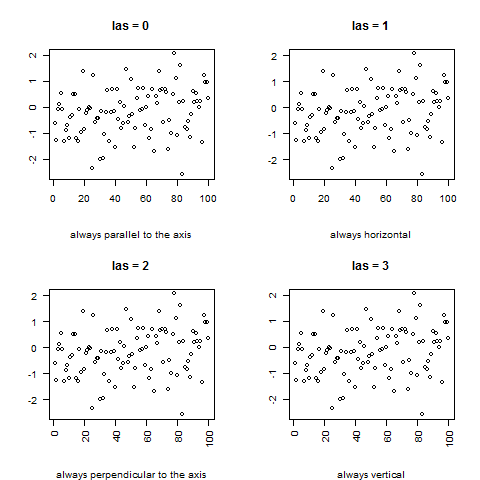
> axis(1,col="red",col.axis="blue",font.axis=3)

> axis(2,col="red",col.axis="blue",font.axis=2,las=2)

las specifies the style of axis labels. It can be 0, 1, 2 or 3.

* 0 : always parallel to the axis [default],
* 1 : always horizontal,
* 2 : always perpendicular to the axis,
* 3 : always vertical.

|  |  |
| --- | --- |
| **R code** | **Output** |
| x1 <- rnorm(100)  par(mfrow = c(2,2))  plot(x1, las = 0, main = "las = 0", sub = "always parallel to the axis", xlab = "", ylab = "")  plot(x1, las = 1, main = "las = 1", sub = "always horizontal", xlab = "", ylab = "")  plot(x1, las = 2, main = "las = 2", sub = "always perpendicular to the axis", xlab = "", ylab = "")  plot(x1, las = 3, main = "las = 3", sub = "always vertical", xlab = "", ylab = "") |  | |



It is also possible to add another y axis on the right by adding axis(4,).

**Margins**

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Book_important2.svg/40px-Book_important2.svg.png | **This section is a stub.** You can help Wikibooks by [expanding it](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit). |

Margins can be computed in inches or in lines. The default is par(mar = c(5,4,4,2)) which means that there are 5 lines at the bottom, 4 lines on the left, 4 lines in the top and 2 lines on the right. This can be modified using the par() function. If you want to specify margins in inches, use par(mai = c(bottom, left, top, right). If you want to modify margins in lines, use par(mar = c(bottom, left, top, right). See ?par to learn more about the topic.

**Colors**

The color of the points or lines can be changed using the col argument, fg for foreground colors (boxes and axes) and bg for background colors.

* show.col(object=NULL) (**Hmisc**) package plots the main R colors with their numeric code.
* The list of all colors in R ([pdf](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf))

colors() # list the r colors

show.col(object=NULL) # graphs the main R colors

plot(x1, col = "blue")

plot(x1, col = "red")

plot(x1, col = "red", col.axis = "dodgerblue", col.lab = "firebrick", col.main = "darkgreen", col.sub = "cyan4", main = "Testing colors", sub = "sub titles", ylab = "y axis", xlab = "x axis")

* We can also generate new colors using the rgb() function. The first argument is the intensity of red, the second, the intensity of green and the third, the intensity of blue. They vary between 0 and 1 by default but this can be modified with the option max = 255. col2rgb() returns the RGB code of R colors. col2hex() (**gplots**) gives the hexadecimal code. col2grey() and col2gray() (**TeachingDemos**) converts colors to grey scale.

> mycolor <- rgb(.2,.4,.6)

> plot(x1, col = mycolor)

> col2rgb("pink")

[,1]

red 255

green 192

blue 203

> library("gplots")

> col2hex("pink")

[1] "#FFC0CB"

**Points**

For points the symbols can be changed using the pch option which takes integer values between 0 and 25 or a single character. pch can also takes a vector as argument. In that case the first points will use the first element of the vector as symbol, and so on.

plot(x1, type = "p", pch = 0)

plot(x1, type = "p", pch = 10)

plot(x1, type = "p", pch = 25)

plot(x1, type = "p", pch = "a")

plot(x1, type = "p", pch = "\*")

plot(x1[1:26], type = "p", pch = 0:25)

plot(x1[1:26], type = "p", pch = letters)

The following code displays all the symbols on the same plot :

x <- rep(1,25)

plot(x, pch = 1:25, axes = F, xlab = "", ylab = "")

text(1:25,.95,labels = 1:25)

points() adds points to an existing plot.

> plot(x1, pch = 0) # plot x1

> points(x2, pch = 1, col = "red") # add x2 to the existing plot

**Lines**

We can change the line type with lty. The argument is a string ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash") or an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash). The line width can be changed with lwd. The default is lwd=1. lwd=2 means that the width is twice the normal width.

plot(x1, type = "l", lty = "blank")

plot(x1, type = "l", lty = "solid")

plot(x1, type = "l", lty = "dashed")

plot(x1, type = "l", lty = "dotted")

plot(x1, type = "l", lty = "dotdash")

plot(x1, type = "l", lty = "longdash")

plot(x1, type = "l", lty = "twodash")

lines() adds an additional lines on a graph.

plot(x1, type = "l", lty = "solid")

lines(x2, type = "l", lty = "dashed", col = "red")

abline() adds an horizontal line (h=), a vertical line (v=) or a linear function to the current plot (a= for the constant and b= for the slope). abline() can also plot the regression line.

> plot(x1, type = "l", lty = "solid")

> abline(h= -3, lty = "dashed", col = "gray")

> abline(v = 0, lty = "dashed", col = "gray")

> abline(a = -3 , b = .06, lty = "dotted", col = "red")

**Boxes**[

Each graph is framed by a box. bty specifies the box type.

plot(x1, bty = "o") # the default

plot(x1, bty = "n") # no box

plot(x1, bty = "l")

plot(x1, bty = "7")

plot(x1, bty = "u")

plot(x1, bty = "c")

plot(x1, bty = "]")

See also box() to add a box to an existing plot.

**Grid**[

grid() adds a grid to the current graph.

> plot(x1)

> grid()

Although grid has an optional argument nx for setting the number of grid lines, it is not possible to tell it explicitly where to place those lines (it will usually not place them at integer values). A more precise and manageable alternative is to use abline().

> abline(v=(seq(0,100,5)), col="lightgray", lty="dotted")

> abline(h=(seq(0,100,5)), col="lightgray", lty="dotted")

**Arrows and segments**[

**Polygons**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=18" \o "Edit section: Polygons) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=18)]

**Other figures**

We can also add a circle to a plot with the circle() function in the **calibrate** package.

**Background**

You can choose the background of your plot. For instance, you can change the background color with par(bg=).

par(bg="whitesmoke")

par(bg="transparent")

**Overlaying plots**[

matplot() can plot several plots at the same time.

N <- 100

x1 <- rnorm(N)

x2 <- rnorm(N) + x1 + 1

y <- 1 + x1 + x2 + rnorm(N)

mydat <- data.frame(y,x1,x2)

matplot(mydat[,1],mydat[,2:3], pch = 1:2)

**Multiple plots**

With par() we can display multiple figures on the same plot. mfrow = c(3,2) prints 6 figures on the same plot with 3 rows and 2 columns. mfcol = c(3,2) does the same but the order is not the same.

par(mfrow = c(3,2))

plot(x1, type = "n")

plot(x1, type = "p")

plot(x1, type = "l")

plot(x1, type = "h")

plot(x1, type = "s")

plot(x1, type = "S")

par(mfcol = c(3,2))

plot(x1, type = "n")

plot(x1, type = "p")

plot(x1, type = "l")

plot(x1, type = "h")

plot(x1, type = "s")

plot(x1, type = "S")

**Plotting a function**

* curve() plots a function. This can be added to an existing plot with the option add = TRUE.
* plot() can also plots functions.

curve(x^2, from = -1 , to = 1, main = "Quadratic function", ylab = "f(x)=x^2")

plot(rnorm(100))

curve((x/100)^2, add = TRUE, col = "red")

It is also possible to add another y axis on the right by adding axis(4,).

**Margins**

Margins can be computed in inches or in lines. The default is par(mar = c(5,4,4,2)) which means that there are 5 lines at the bottom, 4 lines on the left, 4 lines in the top and 2 lines on the right. This can be modified using the par() function. If you want to specify margins in inches, use par(mai = c(bottom, left, top, right). If you want to modify margins in lines, use par(mar = c(bottom, left, top, right). See ?par to learn more about the topic.

**Colors**

The color of the points or lines can be changed using the col argument, fg for foreground colors (boxes and axes) and bg for background colors.

* show.col(object=NULL) (**Hmisc**) package plots the main R colors with their numeric code.
* The list of all colors in R ([pdf](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf))

colors() # list the r colors

show.col(object=NULL) # graphs the main R colors

plot(x1, col = "blue")

plot(x1, col = "red")

plot(x1, col = "red", col.axis = "dodgerblue", col.lab = "firebrick", col.main = "darkgreen", col.sub = "cyan4", main = "Testing colors", sub = "sub titles", ylab = "y axis", xlab = "x axis")

* We can also generate new colors using the rgb() function. The first argument is the intensity of red, the second, the intensity of green and the third, the intensity of blue. They vary between 0 and 1 by default but this can be modified with the option max = 255. col2rgb() returns the RGB code of R colors. col2hex() (**gplots**) gives the hexadecimal code. col2grey() and col2gray() (**TeachingDemos**) converts colors to grey scale.

> mycolor <- rgb(.2,.4,.6)

> plot(x1, col = mycolor)

> col2rgb("pink")

[,1]

red 255

green 192

blue 203

> library("gplots")

> col2hex("pink")

[1] "#FFC0CB"

**Points**

For points the symbols can be changed using the pch option which takes integer values between 0 and 25 or a single character. pch can also takes a vector as argument. In that case the first points will use the first element of the vector as symbol, and so on.

plot(x1, type = "p", pch = 0)

plot(x1, type = "p", pch = 10)

plot(x1, type = "p", pch = 25)

plot(x1, type = "p", pch = "a")

plot(x1, type = "p", pch = "\*")

plot(x1[1:26], type = "p", pch = 0:25)

plot(x1[1:26], type = "p", pch = letters)

The following code displays all the symbols on the same plot :

x <- rep(1,25)

plot(x, pch = 1:25, axes = F, xlab = "", ylab = "")

text(1:25,.95,labels = 1:25)

points() adds points to an existing plot.

> plot(x1, pch = 0) # plot x1

> points(x2, pch = 1, col = "red") # add x2 to the existing plot

**Lines**

We can change the line type with lty. The argument is a string ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash") or an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash). The line width can be changed with lwd. The default is lwd=1. lwd=2 means that the width is twice the normal width.

plot(x1, type = "l", lty = "blank")

plot(x1, type = "l", lty = "solid")

plot(x1, type = "l", lty = "dashed")

plot(x1, type = "l", lty = "dotted")

plot(x1, type = "l", lty = "dotdash")

plot(x1, type = "l", lty = "longdash")

plot(x1, type = "l", lty = "twodash")

lines() adds an additional lines on a graph.

plot(x1, type = "l", lty = "solid")

lines(x2, type = "l", lty = "dashed", col = "red")

abline() adds an horizontal line (h=), a vertical line (v=) or a linear function to the current plot (a= for the constant and b= for the slope). abline() can also plot the regression line.

> plot(x1, type = "l", lty = "solid")

> abline(h= -3, lty = "dashed", col = "gray")

> abline(v = 0, lty = "dashed", col = "gray")

> abline(a = -3 , b = .06, lty = "dotted", col = "red")

**Boxes**

Each graph is framed by a box. bty specifies the box type.

plot(x1, bty = "o") # the default

plot(x1, bty = "n") # no box

plot(x1, bty = "l")

plot(x1, bty = "7")

plot(x1, bty = "u")

plot(x1, bty = "c")

plot(x1, bty = "]")

See also box() to add a box to an existing plot.

**Grid**

grid() adds a grid to the current graph.

> plot(x1)

> grid()

Although grid has an optional argument nx for setting the number of grid lines, it is not possible to tell it explicitly where to place those lines (it will usually not place them at integer values). A more precise and manageable alternative is to use abline().

> abline(v=(seq(0,100,5)), col="lightgray", lty="dotted")

> abline(h=(seq(0,100,5)), col="lightgray", lty="dotted")

**Arrows and segments**

**Polygons**

**Other figures**

We can also add a circle to a plot with the circle() function in the **calibrate** package.

**Background**

You can choose the background of your plot. For instance, you can change the background color with par(bg=).

par(bg="whitesmoke")

par(bg="transparent")

**Overlaying plots**

matplot() can plot several plots at the same time.

N <- 100

x1 <- rnorm(N)

x2 <- rnorm(N) + x1 + 1

y <- 1 + x1 + x2 + rnorm(N)

mydat <- data.frame(y,x1,x2)

matplot(mydat[,1],mydat[,2:3], pch = 1:2)

**Multiple plots**

With par() we can display multiple figures on the same plot. mfrow = c(3,2) prints 6 figures on the same plot with 3 rows and 2 columns. mfcol = c(3,2) does the same but the order is not the same.

par(mfrow = c(3,2))

plot(x1, type = "n")

plot(x1, type = "p")

plot(x1, type = "l")

plot(x1, type = "h")

plot(x1, type = "s")

plot(x1, type = "S")

par(mfcol = c(3,2))

plot(x1, type = "n")

plot(x1, type = "p")

plot(x1, type = "l")

plot(x1, type = "h")

plot(x1, type = "s")

plot(x1, type = "S")

**Plotting a function**

* curve() plots a function. This can be added to an existing plot with the option add = TRUE.
* plot() can also plots functions.

curve(x^2, from = -1 , to = 1, main = "Quadratic function", ylab = "f(x)=x^2")

plot(rnorm(100))

curve((x/100)^2, add = TRUE, col = "red")

It is also possible to add another y axis on the right by adding axis(4,).

**Margins**[

Margins can be computed in inches or in lines. The default is par(mar = c(5,4,4,2)) which means that there are 5 lines at the bottom, 4 lines on the left, 4 lines in the top and 2 lines on the right. This can be modified using the par() function. If you want to specify margins in inches, use par(mai = c(bottom, left, top, right). If you want to modify margins in lines, use par(mar = c(bottom, left, top, right). See ?par to learn more about the topic.

**Colors**

The color of the points or lines can be changed using the col argument, fg for foreground colors (boxes and axes) and bg for background colors.

* show.col(object=NULL) (**Hmisc**) package plots the main R colors with their numeric code.
* The list of all colors in R ([pdf](http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf))

colors() # list the r colors

show.col(object=NULL) # graphs the main R colors

plot(x1, col = "blue")

plot(x1, col = "red")

plot(x1, col = "red", col.axis = "dodgerblue", col.lab = "firebrick", col.main = "darkgreen", col.sub = "cyan4", main = "Testing colors", sub = "sub titles", ylab = "y axis", xlab = "x axis")

* We can also generate new colors using the rgb() function. The first argument is the intensity of red, the second, the intensity of green and the third, the intensity of blue. They vary between 0 and 1 by default but this can be modified with the option max = 255. col2rgb() returns the RGB code of R colors. col2hex() (**gplots**) gives the hexadecimal code. col2grey() and col2gray() (**TeachingDemos**) converts colors to grey scale.

> mycolor <- rgb(.2,.4,.6)

> plot(x1, col = mycolor)

> col2rgb("pink")

[,1]

red 255

green 192

blue 203

> library("gplots")

> col2hex("pink")

[1] "#FFC0CB"

**Points**

For points the symbols can be changed using the pch option which takes integer values between 0 and 25 or a single character. pch can also takes a vector as argument. In that case the first points will use the first element of the vector as symbol, and so on.

plot(x1, type = "p", pch = 0)

plot(x1, type = "p", pch = 10)

plot(x1, type = "p", pch = 25)

plot(x1, type = "p", pch = "a")

plot(x1, type = "p", pch = "\*")

plot(x1[1:26], type = "p", pch = 0:25)

plot(x1[1:26], type = "p", pch = letters)

The following code displays all the symbols on the same plot :

x <- rep(1,25)

plot(x, pch = 1:25, axes = F, xlab = "", ylab = "")

text(1:25,.95,labels = 1:25)

points() adds points to an existing plot.

> plot(x1, pch = 0) # plot x1

> points(x2, pch = 1, col = "red") # add x2 to the existing plot

**Lines**

We can change the line type with lty. The argument is a string ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", or "twodash") or an integer (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash). The line width can be changed with lwd. The default is lwd=1. lwd=2 means that the width is twice the normal width.

plot(x1, type = "l", lty = "blank")

plot(x1, type = "l", lty = "solid")

plot(x1, type = "l", lty = "dashed")

plot(x1, type = "l", lty = "dotted")

plot(x1, type = "l", lty = "dotdash")

plot(x1, type = "l", lty = "longdash")

plot(x1, type = "l", lty = "twodash")

lines() adds an additional lines on a graph.

plot(x1, type = "l", lty = "solid")

lines(x2, type = "l", lty = "dashed", col = "red")

abline() adds an horizontal line (h=), a vertical line (v=) or a linear function to the current plot (a= for the constant and b= for the slope). abline() can also plot the regression line.

> plot(x1, type = "l", lty = "solid")

> abline(h= -3, lty = "dashed", col = "gray")

> abline(v = 0, lty = "dashed", col = "gray")

> abline(a = -3 , b = .06, lty = "dotted", col = "red")

**Boxes**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=15" \o "Edit section: Boxes) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=15)]

Each graph is framed by a box. bty specifies the box type.

plot(x1, bty = "o") # the default

plot(x1, bty = "n") # no box

plot(x1, bty = "l")

plot(x1, bty = "7")

plot(x1, bty = "u")

plot(x1, bty = "c")

plot(x1, bty = "]")

See also box() to add a box to an existing plot.

**Grid**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=16" \o "Edit section: Grid) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=16)]

grid() adds a grid to the current graph.

> plot(x1)

> grid()

Although grid has an optional argument nx for setting the number of grid lines, it is not possible to tell it explicitly where to place those lines (it will usually not place them at integer values). A more precise and manageable alternative is to use abline().

> abline(v=(seq(0,100,5)), col="lightgray", lty="dotted")

> abline(h=(seq(0,100,5)), col="lightgray", lty="dotted")

**Arrows and segments**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=17" \o "Edit section: Arrows and segments) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=17)]

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Book_important2.svg/40px-Book_important2.svg.png | **This section is a stub.** You can help Wikibooks by [expanding it](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit). |

**Polygons**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=18" \o "Edit section: Polygons) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=18)]

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Book_important2.svg/40px-Book_important2.svg.png | **This section is a stub.** You can help Wikibooks by [expanding it](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit). |

**Other figures**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=19" \o "Edit section: Other figures) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=19)]

We can also add a circle to a plot with the circle() function in the **calibrate** package.

**Background**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=20" \o "Edit section: Background) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=20)]

You can choose the background of your plot. For instance, you can change the background color with par(bg=).

par(bg="whitesmoke")

par(bg="transparent")

**Overlaying plots**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=21" \o "Edit section: Overlaying plots) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=21)]

matplot() can plot several plots at the same time.

N <- 100

x1 <- rnorm(N)

x2 <- rnorm(N) + x1 + 1

y <- 1 + x1 + x2 + rnorm(N)

mydat <- data.frame(y,x1,x2)

matplot(mydat[,1],mydat[,2:3], pch = 1:2)

**Multiple plots**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=22" \o "Edit section: Multiple plots) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=22)]

With par() we can display multiple figures on the same plot. mfrow = c(3,2) prints 6 figures on the same plot with 3 rows and 2 columns. mfcol = c(3,2) does the same but the order is not the same.

par(mfrow = c(3,2))

plot(x1, type = "n")

plot(x1, type = "p")

plot(x1, type = "l")

plot(x1, type = "h")

plot(x1, type = "s")

plot(x1, type = "S")

par(mfcol = c(3,2))

plot(x1, type = "n")

plot(x1, type = "p")

plot(x1, type = "l")

plot(x1, type = "h")

plot(x1, type = "s")

plot(x1, type = "S")

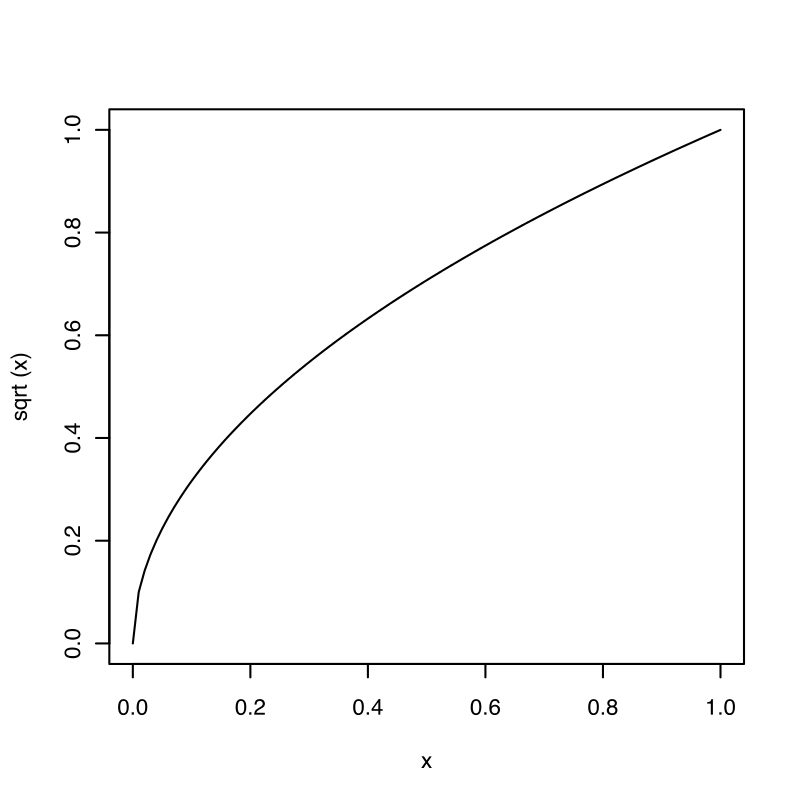
**Plotting a function**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=23" \o "Edit section: Plotting a function) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=23)]

* curve() plots a function. This can be added to an existing plot with the option add = TRUE.
* plot() can also plots functions.

curve(x^2, from = -1 , to = 1, main = "Quadratic function", ylab = "f(x)=x^2")

plot(rnorm(100))

curve((x/100)^2, add = TRUE, col = "red")



## xporting graphs[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=24) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=24)]

How can you export a graph ?

* First you can plot the graph and use the context menu (right click on Windows and Linux or control + click on Mac) to copy or save the graphs. The available options depend on your operating system. On Windows, you can also use copy the current graph to the clipboard as a Bitmap file (raster graphics) using CTRL + C or as a Windows Metafile (vector graphics) using CTRL + W. You can then paste it into another application.
* You can export a plot to **pdf**, **png**, **jpeg**, **bmp** or **tiff** by adding pdf("filename.pdf"), png("filename.png"), jpeg("filename.jpg"), bmp("filename.bmp") or tiff("filename.tiff") prior to the plotting, and dev.off() after the plotting.
* You can also use the savePlot() function to save existing graphs.
* Sweave also produce ps and pdf graphics (See the [Sweave section](https://en.wikibooks.org/wiki/R_Programming/Publication_Quality_Ouput" \l "Sweave" \o "R Programming/Publication Quality Ouput)).

It is better to use vectorial devices such as **pdf**, **ps** or **svg**.

How can you know the list of all available devices ?

* ?Devices
* Use the capabilities() function to see the list of available devices on your computer.

?Devices

> capabilities()

jpeg png tiff tcltk X11 aqua http/ftp sockets

TRUE TRUE TRUE TRUE FALSE FALSE TRUE TRUE

libxml fifo cledit iconv NLS profmem cairo

TRUE FALSE TRUE TRUE TRUE TRUE FALSE

png("r\_plot.png", width = 420, height = 340)

plot(x1, main = " Example")

dev.off()

pdf("r\_plot.pdf", width = 420, height = 340)

plot(x1, main = " Example")

dev.off()

postscript(file="graph1.ps",horizontal=F,pagecentre=F,paper="special",width=8.33,height=5.56)

plot(x1, main = "Example")

dev.off()

plot(x1, main = "Example")

savePlot("W:/Bureau/plot.pdf", type = "pdf")

savePlot("W:/Bureau/plot.png", type = "png")

We can also export to [SVG](https://en.wikipedia.org/wiki/Scalable_Vector_Graphics) using the svg() function.

svg("scatterplot.svg", width = 7, height = 7)

plot(x, y)

dev.off()

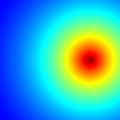
The **RSvgDevice** library which was used in earlier versions of R seems now outdated.

## Advanced topics[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=25" \o "Edit section: Advanced topics) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=25)]

### Animated plots[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=26" \o "Edit section: Animated plots) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=26)]

The **animation** package provides dynamic graphics capabilities. It is possible to export the animation in flash, mpeg or gif format. There are more example on the aniwiki website : <http://animation.yihui.name/>.

You can also create *motion charts* using the **googleVis** package[[5]](https://en.wikibooks.org/wiki/R_Programming/Graphics" \l "cite_note-5).



### Interactive Graphics[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=28" \o "Edit section: Interactive Graphics) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=28)]

The **iplots** package provides a way to have interactive data visualization in R[[6]](https://en.wikibooks.org/wiki/R_Programming/Graphics" \l "cite_note-6)**·**[[7]](https://en.wikibooks.org/wiki/R_Programming/Graphics#cite_note-7).

* [R GUI now offers interactive graphics – Deducer 0.4-2 connects with iplots](http://www.r-statistics.com/2010/10/r-gui-now-offers-interactive-graphics-deducer-0-4-2-connects-with-iplots/)

To create an interactive, animated plot viewable in a web browser, the [animint package](https://github.com/tdhock/animint) can be used. The main idea is to define an interactive animation as a list of ggplots with two new aesthetics:

* showSelected=variable means that only the subset of the data that corresponds to the selected value of variable will be shown.
* clickSelects=variable means that clicking a plot element will change the currently selected value of variable.

## Graphics gallery[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=29" \o "Edit section: Graphics gallery) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=29)]

In this section, we review all kind of statistical plots and review all alternatives to draw them using R. This include code for the standard graphics package, the **lattice** package and the **ggplot2** package. Also, we add some examples from the [commons repository](https://commons.wikimedia.org/wiki/Category:Created_with_R). We only add examples which are provided with the R code. You can click on any graph and find the R code.

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/4/4a/Commons-logo.svg/30px-Commons-logo.svg.png | [Wikimedia Commons](https://commons.wikimedia.org/wiki/) has media related to: ***[Category:Created with R](https://commons.wikimedia.org/wiki/Category:Created_with_R)*** |

### Line plot[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=30" \o "Edit section: Line plot) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=30)]

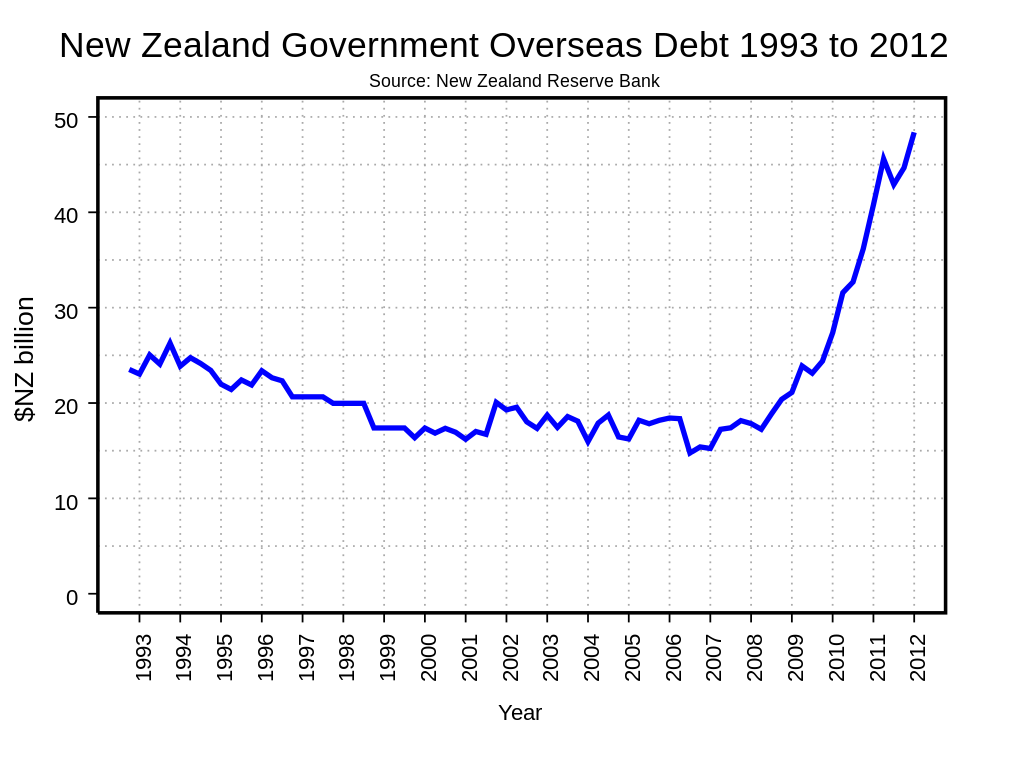
To draw a line plot, use the generic plot() function by setting type="l".

> x <- seq(0, 2\*pi, pi/10)

> plot(x, sin(x), type="l")

Then, you can add further lines on the same plot using the lines() function.

> lines(x, cos(x))



### Scatter plot[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=32" \o "Edit section: Scatter plot) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=32)]

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/Wikipedia-logo-v2.svg/40px-Wikipedia-logo-v2.svg.png | [Wikipedia](https://en.wikipedia.org/wiki/) has related information at [***Scatter plot***](https://en.wikipedia.org/wiki/Scatter_plot) |

* plot(x,y)
* plot(y ~ x)
* xyplot(y ~ x) (**lattice**)
* qplot(x,y) (**ggplot2**)

#### Log scale[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=33" \o "Edit section: Log scale) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=33)]

Sometimes it is useful to plot the log of a variable and to have a log scale on the axis. It is possible to plot the log of a variable using the log option in the plot() function.

* For a log log plot, use log = "xy"
* For a log in the x axis only, use log = "x"
* For a log in the x axis only, use log = "y"

plot(x, y , log = "xy")

#### Label points in a plot[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=34" \o "Edit section: Label points in a plot) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=34)]

* It is possible to add labels with the text() function.
* textxy() (**calibrate**) makes it easy to add labels.

N <- 10

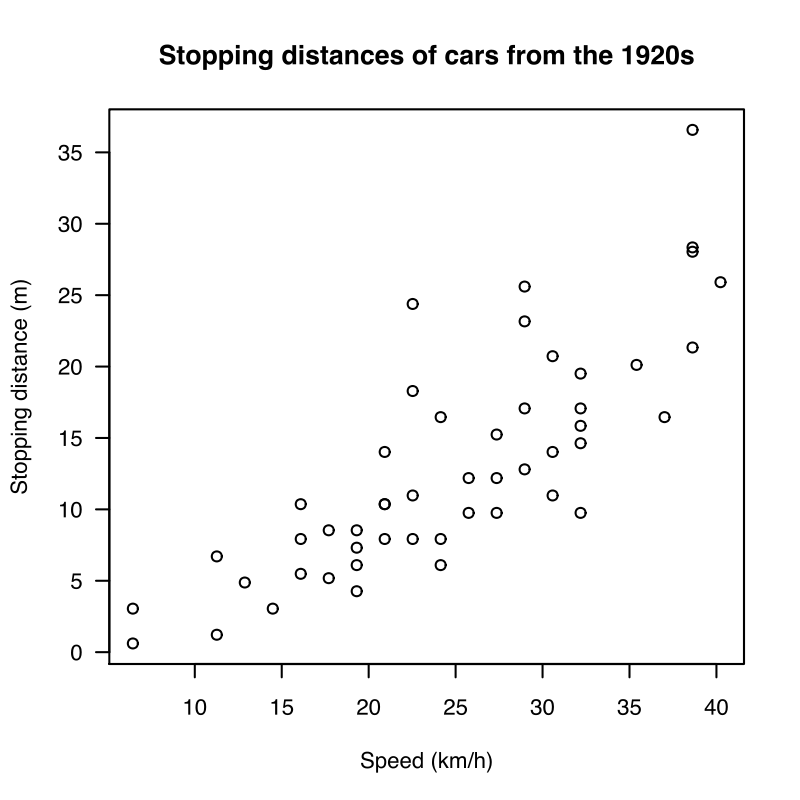
u <-rnorm(N)

x <- 1 + rnorm(N)

y <- 1 + x + u

plot(x, y)

textxy(x, y,labs = signif(x,3), cx=0.7)



**Histogram**[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=36" \o "Edit section: Histogram) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=36)]

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/Wikipedia-logo-v2.svg/40px-Wikipedia-logo-v2.svg.png | [Wikipedia](https://en.wikipedia.org/wiki/) has related information at [***histogram***](https://en.wikipedia.org/wiki/Histogram) |

* hist()
* histogram() (**lattice**)

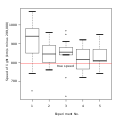
You can learn more about histograms in the [Non parametric methods](https://en.wikibooks.org/wiki/R_Programming/Nonparametric_Methods#Histogram) page.

### Box plot[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=38" \o "Edit section: Box plot) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=38)]

[Box plot](https://en.wikipedia.org/wiki/Box_plot) :

* boxplot()

#### Examples[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=39" \o "Edit section: Examples) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=39)]

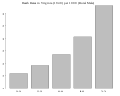
* [](https://en.wikibooks.org/wiki/File:Michelsonmorley-boxplot.svg)

### Bar charts[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=41" \o "Edit section: Bar charts) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=41)]

See [Bar charts](https://en.wikipedia.org/wiki/Bar_chart) on wikipedia.

* barplot() takes a table as argument and returns a bar chart.
* qlot() (**ggplot2**) with the option geom = "bar" takes a variable as argument and returns a bar chart[[8]](https://en.wikibooks.org/wiki/R_Programming/Graphics#cite_note-8).
* barchart() takes a variable as argument and returns a bar chart.

#### Examples[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=42" \o "Edit section: Examples) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=42)]

* [](https://en.wikibooks.org/wiki/File:Simple_barplot.svg)

### Dot plot[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=43" \o "Edit section: Dot plot) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=43)]

See also [Dot plot](https://en.wikipedia.org/wiki/Dot_plot_(statistics)) on Wikipedia.

* dotchart()

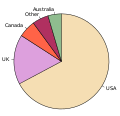
#### Examples

### Pie charts[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=45" \o "Edit section: Pie charts) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=45)]

|  |  |
| --- | --- |
| https://upload.wikimedia.org/wikipedia/commons/thumb/8/80/Wikipedia-logo-v2.svg/40px-Wikipedia-logo-v2.svg.png | [Wikipedia](https://en.wikipedia.org/wiki/) has related information at [***Pie chart***](https://en.wikipedia.org/wiki/Pie_chart) |

* pie()

#### Examples[[edit](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&veaction=edit&section=46) | [edit source](https://en.wikibooks.org/w/index.php?title=R_Programming/Graphics&action=edit&section=46)]

* [](https://en.wikibooks.org/wiki/File:English_dialects1997.svg)