SHC 798 R-code

Richard Lubega

2025-07-03

Introduction

R Markdown Preamble

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

See also the **R Markdown Cookbook**)

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

Sample Inbuilt Data Set

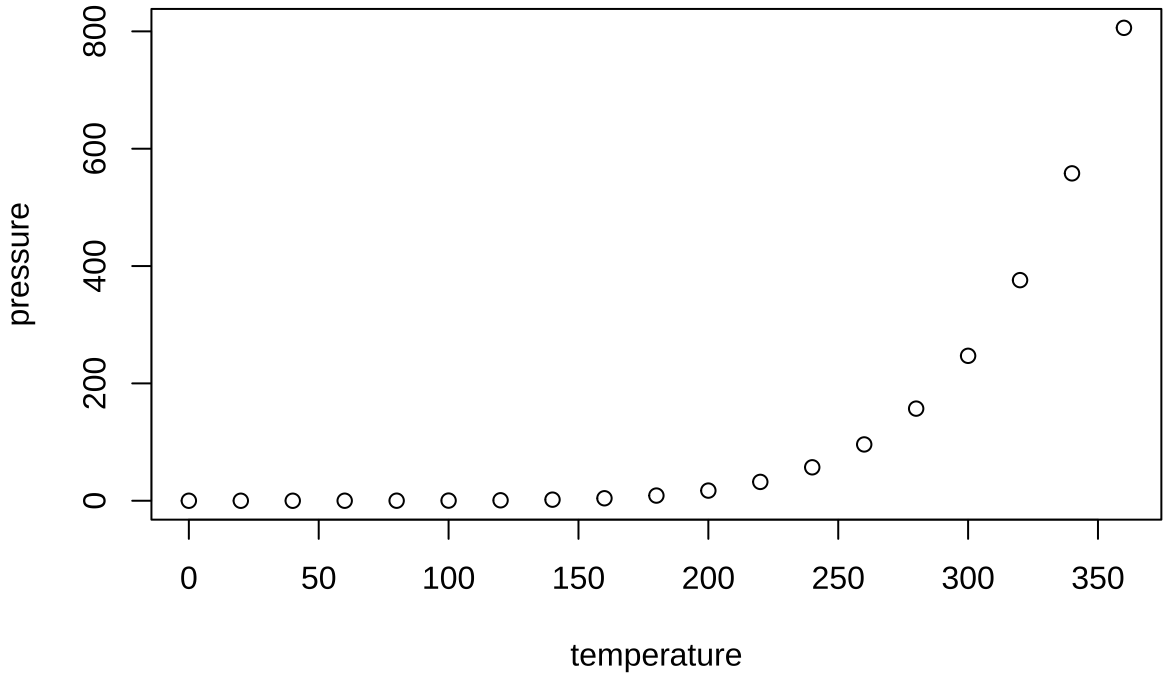
```r
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Including Plots - Example

Embed Plot

You can also embed plots, for example:

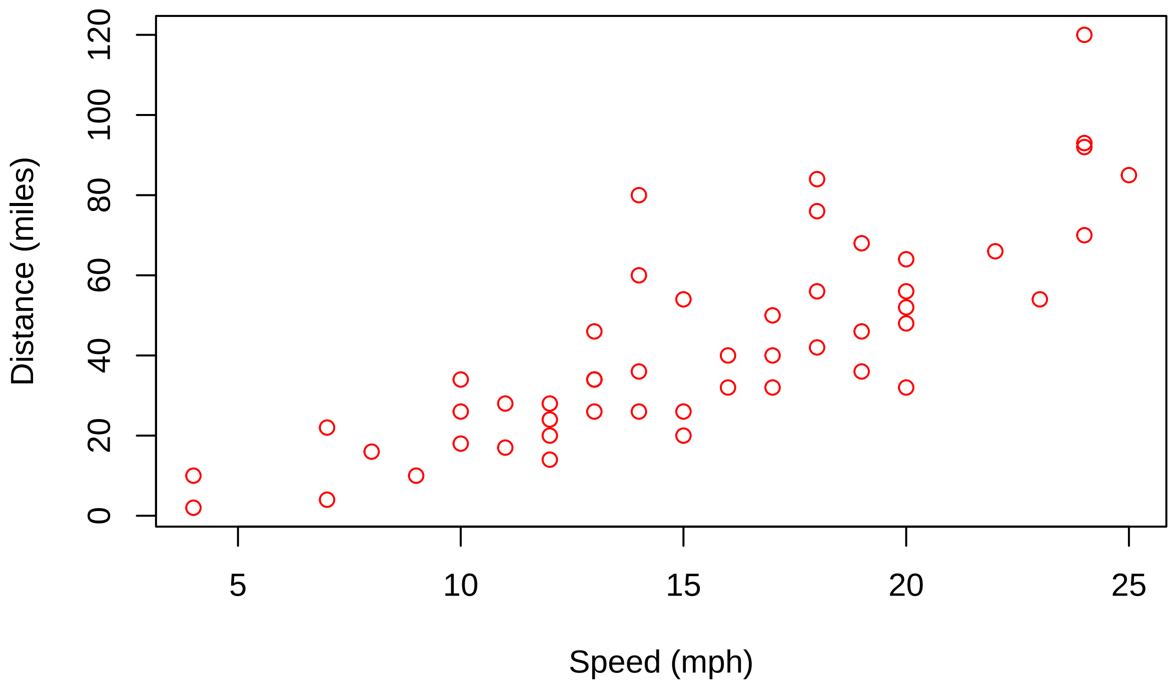


Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.

Another Plot

Let’s create another Test plot

```
plot(cars,  col = "red", xlab = "Speed (mph)", ylab = "Distance (miles)" )
```



A summary of the data frame is given below

```
pacman::p_load(knitr)
kable(summary(cars))
```

speed	dist
Min. : 4.0	Min. : 2.00
1st Qu.:12.0	1st Qu.: 26.00
Median :15.0	Median : 36.00
Mean :15.4	Mean : 42.98
3rd Qu.:19.0	3rd Qu.: 56.00
Max. :25.0	Max. :120.00

Start of My SHC 798 R-code

R basics

1. Why install **Rtools**?

2. R Shiny **Gallery**: There's Shiny for R and Shiny for Python

R Operations 1

What can be done?

```
# R as a calculator
sqrt(12)
```

```
## [1] 3.464102
```

```
x <- 3
y <- x^2
x+y
```

```
## [1] 12
```

```
# Creating Vectors
v1 <- c(1,5, 80)
v1
```

```
## [1]  1  5 80
```

```
v2 <- 2:11
v2
```

```
##  [1]  2  3  4  5  6  7  8  9 10 11
```

```
a <- c(1,6,10,22,7,13)
mean(a)
```

```
## [1] 9.833333
```

```
# Incomplete Statement
3*(4 +
     +2)
```

```
## [1] 18
```

```
#Assignments and Function Calls
t.a <- 3*(4+2)
t.b <- t.a + 2.5
mn <- mean(c(t.a,t.b))
mn
```

```
## [1] 19.25
```

R Help

Resources

1. Very useful and helpful **R Q&A Website**
2. Paradis 2005 - R for Beginners [Textbook]
3. R Reference Card [in the notes, 6 pages - Material from *R for Beginners*]
4. Base R Cheat Sheet [in the notes, 2 pages]
5. Venables et al 2017 - An introduction to R [Textbook]
6. **R for Data Science, r4ds** or the **r4ds-2e**
7. **W^{3}Schools** R Certification

R Operations 2

Importing Data sets

```
# Import Data: Website
url <- "https://stat.ethz.ch/Teaching/Datasets/WBL/sport.dat"
d.sport <- read.table(url, header = TRUE)
head(d.sport)
```

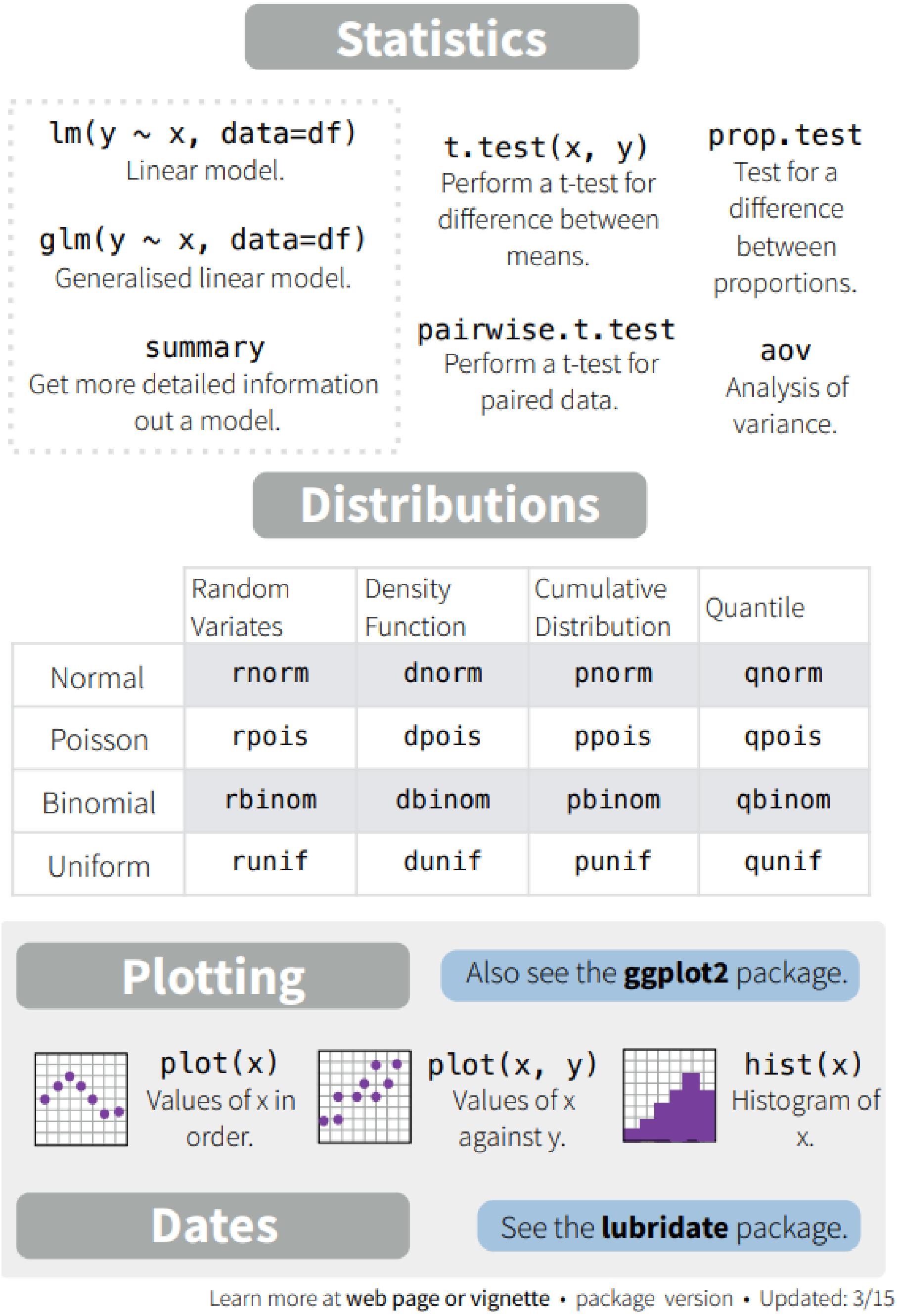
```
##            weit kugel hoch  disc stab speer punkte
## OBRIEN     7.57 15.66  207 48.78  500 66.90   8824
## BUSEMANN   8.07 13.60  204 45.04  480 66.86   8706
## DVORAK     7.60 15.82  198 46.28  470 70.16   8664
## FRITZ      7.77 15.31  204 49.84  510 65.70   8644
## HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
## NOOL       7.88 14.01  201 42.98  540 65.48   8543
```

```
# Setting the Working Directory
# Use:
getwd() #Prints the current working directory
```

```
## [1] "D:/2025 MEng Transportation/SHC 798 R-Proj"
```

```
# setwd("D:/2025 MEng Transportation/SHC 798 R-Proj") ~ Sets the working directory
# Alternatively, use "Session" → "Set Working Directory" → "Choose Directory…"
# Import Data: Files
  # - Different ways depending on the format (csv, txt, xlsx, etc.)
  # - Alternative: use the "Import Dataset" tool in RStudio (upper-right panel)
# Save data or write data to a file
  # - Text files
  # - Excel files: use CSV
```

R Objects & Indexing



Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

`aov`
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

Also see the **ggplot2** package.

`plot(x)`
Values of x in order.

`plot(x, y)`
Values of x against y.

`hist(x)`
Histogram of x.

Dates

See the **lubridate** package.

Learn more at **web page or vignette** • package version • Updated: 3/15

Figure 1: *Snippet from Resource 4*

```r
# R Objects: Data frames (Most essential)
str(d.sport)
```

```
## 'data.frame':    15 obs. of  7 variables:
##  $ weit  : num  7.57 8.07 7.6 7.77 7.48 7.88 7.64 7.61 7.27 7.49 ...
##  $ kugel : num  15.7 13.6 15.8 15.3 16.3 ...
##  $ hoch  : int  207 204 198 204 198 201 195 213 207 204 ...
##  $ disc  : num  48.8 45 46.3 49.8 49.6 ...
##  $ stab  : int  500 480 470 510 500 540 540 520 470 470 ...
##  $ speer : num  66.9 66.9 70.2 65.7 57.7 ...
##  $ punkte: int  8824 8706 8664 8644 8613 8543 8422 8318 8307 8300 ...
```

```r
# R Objects: Vectors
  # (e.g., a column from the data set d.sport)
kugel <- d.sport$kugel
str(kugel)
```

```
##  num [1:15] 15.7 13.6 15.8 15.3 16.3 ...
```

```r
participant <- rownames(d.sport)
str(participant)
```

```
##  chr [1:15] "OBRIEN" "BUSEMANN" "DVORAK" "FRITZ" "HAMALAINEN" "NOOL" ...
```

```r
# Select elements
participant[4]
```

```
## [1] "FRITZ"
```

```r
d.sport[c(3,6,4), c(1:3,7)]
```

```
##        weit kugel hoch punkte
## DVORAK 7.60 15.82  198   8664
## NOOL   7.88 14.01  201   8543
## FRITZ  7.77 15.31  204   8644
```

```r
d.sport["FRITZ", ]
```

```
##       weit kugel hoch  disc stab speer punkte
## FRITZ 7.77 15.31  204 49.84  510  65.7   8644
```

```r
# Accessing Parts of an Object
# To access only part of an object, use []
  # For vectors: myvector[x]
  # For two-dimensional objects, e.g. data frames or matrices: mydata.frame[x,y]
d.sport[ , ]
```

```
##            weit kugel hoch  disc stab speer punkte
## OBRIEN     7.57 15.66  207 48.78  500 66.90   8824
## BUSEMANN   8.07 13.60  204 45.04  480 66.86   8706
## DVORAK     7.60 15.82  198 46.28  470 70.16   8664
## FRITZ      7.77 15.31  204 49.84  510 65.70   8644
## HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
## NOOL       7.88 14.01  201 42.98  540 65.48   8543
## ZMELIK     7.64 13.53  195 43.44  540 67.20   8422
## GANIYEV    7.61 14.71  213 44.86  520 53.70   8318
## PENALVER   7.27 16.91  207 48.92  470 57.08   8307
## HUFFINS    7.49 15.57  204 48.72  470 60.62   8300
## PLAZIAT    7.82 14.85  204 45.34  490 52.18   8282
## MAGNUSSON  7.28 15.52  195 43.78  480 61.10   8274
## SMITH      7.47 16.97  195 49.54  500 64.34   8271
## MUELLER    7.25 14.69  195 45.90  510 66.10   8253
## CHMARA     7.75 14.51  210 42.60  490 54.84   8249
```

```
c(1,3,7)
```

```
## [1] 1 3 7
```

```
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
d.sport[1:10, ]
```

```
##            weit kugel hoch  disc stab speer punkte
## OBRIEN     7.57 15.66  207 48.78  500 66.90   8824
## BUSEMANN   8.07 13.60  204 45.04  480 66.86   8706
## DVORAK     7.60 15.82  198 46.28  470 70.16   8664
## FRITZ      7.77 15.31  204 49.84  510 65.70   8644
## HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
## NOOL       7.88 14.01  201 42.98  540 65.48   8543
## ZMELIK     7.64 13.53  195 43.44  540 67.20   8422
## GANIYEV    7.61 14.71  213 44.86  520 53.70   8318
## PENALVER   7.27 16.91  207 48.92  470 57.08   8307
## HUFFINS    7.49 15.57  204 48.72  470 60.62   8300
```

```
d.sport[-c(1, 3,7), ] #negative indices are excluded
```

```
##            weit kugel hoch  disc stab speer punkte
## BUSEMANN   8.07 13.60  204 45.04  480 66.86   8706
## FRITZ      7.77 15.31  204 49.84  510 65.70   8644
## HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
## NOOL       7.88 14.01  201 42.98  540 65.48   8543
## GANIYEV    7.61 14.71  213 44.86  520 53.70   8318
## PENALVER   7.27 16.91  207 48.92  470 57.08   8307
## HUFFINS    7.49 15.57  204 48.72  470 60.62   8300
## PLAZIAT    7.82 14.85  204 45.34  490 52.18   8282
## MAGNUSSON  7.28 15.52  195 43.78  480 61.10   8274
## SMITH      7.47 16.97  195 49.54  500 64.34   8271
## MUELLER    7.25 14.69  195 45.90  510 66.10   8253
## CHMARA     7.75 14.51  210 42.60  490 54.84   8249
```

```
d.sport[ , 2:3]
```

```
##            kugel hoch
## OBRIEN     15.66  207
## BUSEMANN   13.60  204
## DVORAK     15.82  198
## FRITZ      15.31  204
## HAMALAINEN 16.32  198
## NOOL       14.01  201
## ZMELIK     13.53  195
## GANIYEV    14.71  213
## PENALVER   16.91  207
## HUFFINS    15.57  204
## PLAZIAT    14.85  204
## MAGNUSSON  15.52  195
## SMITH      16.97  195
## MUELLER    14.69  195
## CHMARA     14.51  210
```

```
d.sport[c(1,3,6), 2:3]
```

```
##        kugel hoch
## OBRIEN 15.66  207
## DVORAK 15.82  198
## NOOL   14.01  201
```

```
# Function Calls
mean(kugel)
```

```
## [1] 15.19867
```

```
quantile(kugel)
```

```
##    0%   25%   50%   75%  100%
## 13.53 14.60 15.31 15.74 16.97
```

```
quantile(kugel,probs = c(.75, 0.9))
```

```
##    75%    90%
## 15.740 16.674
```

```
# Functions consist of mandatory and optional arguments:
  # mean(x, trim = 0, na.rm = FALSE, …)
  # x: mandatory argument
  # trim: optional argument, default is 0
  # na.rm: optional argument, default is FALSE

# The arguments of a function have a defined order and each argument has its own unique name
mean(x = kugel, na.rm = TRUE)
```

```
## [1] 15.19867
```

```r
mean(x = kugel, ,TRUE)
```

```
## [1] 15.19867
```

```r
# Useful Functions
nrow(d.sport)
```

```
## [1] 15
```

```r
ncol(d.sport)
```

```
## [1] 7
```

```r
dim(d.sport)
```

```
## [1] 15  7
```

```r
summary(d.sport)
```

```
##       weit           kugel            hoch            disc            stab    
##  Min.   :7.250   Min.   :13.53   Min.   :195.0   Min.   :42.60   Min.   :470  
##  1st Qu.:7.475   1st Qu.:14.60   1st Qu.:196.5   1st Qu.:44.32   1st Qu.:480  
##  Median :7.600   Median :15.31   Median :204.0   Median :45.90   Median :500  
##  Mean   :7.597   Mean   :15.20   Mean   :202.0   Mean   :46.38   Mean   :498  
##  3rd Qu.:7.760   3rd Qu.:15.74   3rd Qu.:205.5   3rd Qu.:48.85   3rd Qu.:510  
##  Max.   :8.070   Max.   :16.97   Max.   :213.0   Max.   :49.84   Max.   :540  
##      speer           punkte    
##  Min.   :52.18   Min.   :8249  
##  1st Qu.:57.37   1st Qu.:8278  
##  Median :64.34   Median :8318  
##  Mean   :61.99   Mean   :8445  
##  3rd Qu.:66.48   3rd Qu.:8628  
##  Max.   :70.16   Max.   :8824  
```

```r
# apply(d.sport)
head(d.sport)
```

```
##            weit kugel hoch  disc stab speer punkte
## OBRIEN     7.57 15.66  207 48.78  500 66.90   8824
## BUSEMANN   8.07 13.60  204 45.04  480 66.86   8706
## DVORAK     7.60 15.82  198 46.28  470 70.16   8664
## FRITZ      7.77 15.31  204 49.84  510 65.70   8644
## HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
## NOOL       7.88 14.01  201 42.98  540 65.48   8543
```

```
tail(d.sport)
```

```
##           weit kugel hoch  disc stab speer punkte
## HUFFINS   7.49 15.57  204 48.72  470 60.62   8300
## PLAZIAT   7.82 14.85  204 45.34  490 52.18   8282
## MAGNUSSON 7.28 15.52  195 43.78  480 61.10   8274
## SMITH     7.47 16.97  195 49.54  500 64.34   8271
## MUELLER   7.25 14.69  195 45.90  510 66.10   8253
## CHMARA    7.75 14.51  210 42.60  490 54.84   8249
```

R Packages

```
#install.packages("MASS")
#require(MASS) # for every R Session
#library(MASS) # or use this
```

Online resources: University of Pretoria | SHC 798 | Introduction to R 36

o List of all packages: http://cran.r-project.org/web/packages/

o By topic: http://cran.r-project.org/web/views/

o Ask Google / ChatGPT / GroK

Missing Values

```
d.sport.NA <- d.sport
d.sport.NA[2, "kugel"] <- NA
d.sport.NA[3, "hoch"] <- -999
# missing values are coded as NA (not available) and are treated in a special way, e.g. is.na():
is.na(d.sport.NA) #one logical value per element
```

```
##             weit kugel  hoch  disc  stab speer punkte
## OBRIEN     FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## BUSEMANN   FALSE  TRUE FALSE FALSE FALSE FALSE  FALSE
## DVORAK     FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## FRITZ      FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## HAMALAINEN FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## NOOL       FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## ZMELIK     FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## GANIYEV    FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## PENALVER   FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## HUFFINS    FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## PLAZIAT    FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## MAGNUSSON  FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## SMITH      FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## MUELLER    FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## CHMARA     FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
```

```r
sum(is.na(d.sport.NA)) # adds up the TRUE elements
```

```
## [1] 1
```

```r
which(is.na(d.sport.NA), arr.ind = TRUE) # where are the NA's
```

```
##          row col
## BUSEMANN   2   2
```

```r
# Specify missing values after reading in the data:
d.sport.NA[d.sport.NA == -999] <- NA

# Many functions have an argument to handle missing values, e.g. na.rm, na.omit:
sum(d.sport.NA$kugel)
```

```
## [1] NA
```

```r
sum(d.sport.NA$kugel, na.rm = TRUE)
```

```
## [1] 214.38
```

```r
na.omit(d.sport.NA)
```

```
##            weit kugel hoch  disc stab speer punkte
## OBRIEN     7.57 15.66  207 48.78  500 66.90   8824
## FRITZ      7.77 15.31  204 49.84  510 65.70   8644
## HAMALAINEN 7.48 16.32  198 49.62  500 57.66   8613
## NOOL       7.88 14.01  201 42.98  540 65.48   8543
## ZMELIK     7.64 13.53  195 43.44  540 67.20   8422
## GANIYEV    7.61 14.71  213 44.86  520 53.70   8318
## PENALVER   7.27 16.91  207 48.92  470 57.08   8307
## HUFFINS    7.49 15.57  204 48.72  470 60.62   8300
## PLAZIAT    7.82 14.85  204 45.34  490 52.18   8282
## MAGNUSSON  7.28 15.52  195 43.78  480 61.10   8274
## SMITH      7.47 16.97  195 49.54  500 64.34   8271
## MUELLER    7.25 14.69  195 45.90  510 66.10   8253
## CHMARA     7.75 14.51  210 42.60  490 54.84   8249
```

Basic Graphics

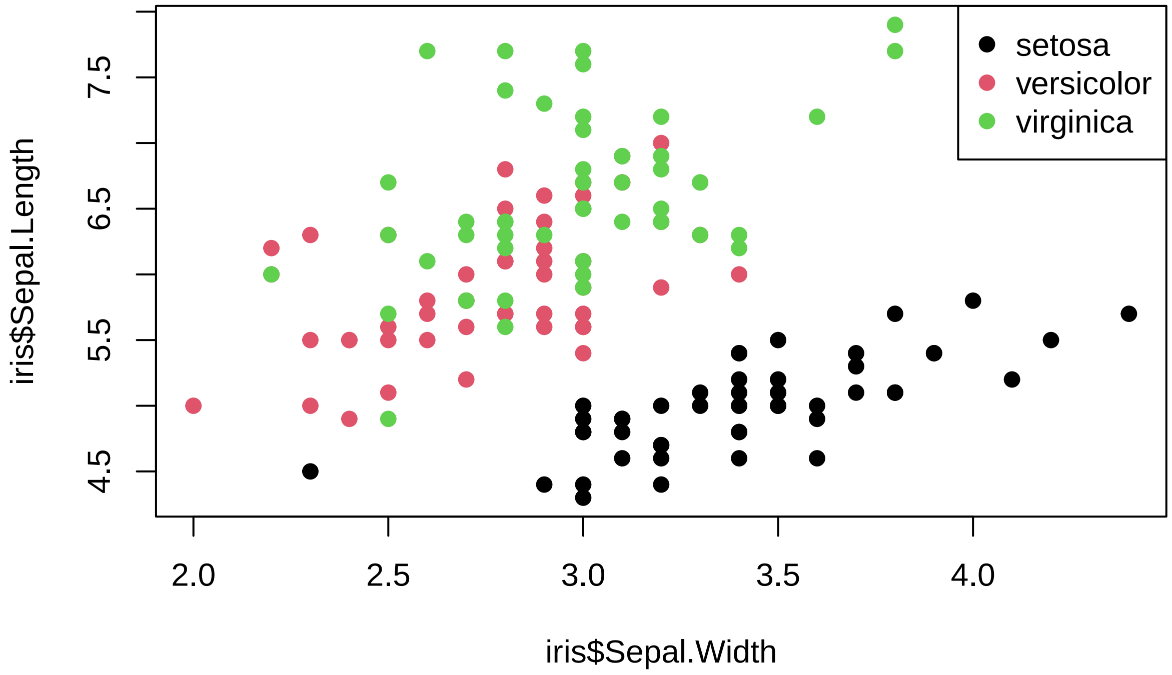
```r
# The Plot Function: only 1 mandatory argument i.e., x. The 2nd most important one is y.
  # many optional arguments [col,pch,main,cex, ...]
  # use function par (?par) to set or query graphical parameters

data(iris)
iris
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
## 10           4.9         3.1          1.5         0.1     setosa
## 11           5.4         3.7          1.5         0.2     setosa
## 12           4.8         3.4          1.6         0.2     setosa
## 13           4.8         3.0          1.4         0.1     setosa
## 14           4.3         3.0          1.1         0.1     setosa
## 15           5.8         4.0          1.2         0.2     setosa
## 16           5.7         4.4          1.5         0.4     setosa
## 17           5.4         3.9          1.3         0.4     setosa
## 18           5.1         3.5          1.4         0.3     setosa
## 19           5.7         3.8          1.7         0.3     setosa
## 20           5.1         3.8          1.5         0.3     setosa
## 21           5.4         3.4          1.7         0.2     setosa
## 22           5.1         3.7          1.5         0.4     setosa
## 23           4.6         3.6          1.0         0.2     setosa
## 24           5.1         3.3          1.7         0.5     setosa
## 25           4.8         3.4          1.9         0.2     setosa
## 26           5.0         3.0          1.6         0.2     setosa
## 27           5.0         3.4          1.6         0.4     setosa
## 28           5.2         3.5          1.5         0.2     setosa
## 29           5.2         3.4          1.4         0.2     setosa
## 30           4.7         3.2          1.6         0.2     setosa
## 31           4.8         3.1          1.6         0.2     setosa
## 32           5.4         3.4          1.5         0.4     setosa
## 33           5.2         4.1          1.5         0.1     setosa
## 34           5.5         4.2          1.4         0.2     setosa
## 35           4.9         3.1          1.5         0.2     setosa
## 36           5.0         3.2          1.2         0.2     setosa
## 37           5.5         3.5          1.3         0.2     setosa
## 38           4.9         3.6          1.4         0.1     setosa
## 39           4.4         3.0          1.3         0.2     setosa
## 40           5.1         3.4          1.5         0.2     setosa
## 41           5.0         3.5          1.3         0.3     setosa
## 42           4.5         2.3          1.3         0.3     setosa
## 43           4.4         3.2          1.3         0.2     setosa
## 44           5.0         3.5          1.6         0.6     setosa
## 45           5.1         3.8          1.9         0.4     setosa
## 46           4.8         3.0          1.4         0.3     setosa
## 47           5.1         3.8          1.6         0.2     setosa
## 48           4.6         3.2          1.4         0.2     setosa
## 49           5.3         3.7          1.5         0.2     setosa
## 50           5.0         3.3          1.4         0.2     setosa
## 51           7.0         3.2          4.7         1.4 versicolor
## 52           6.4         3.2          4.5         1.5 versicolor
## 53           6.9         3.1          4.9         1.5 versicolor
## 54           5.5         2.3          4.0         1.3 versicolor
## 55           6.5         2.8          4.6         1.5 versicolor
## 56           5.7         2.8          4.5         1.3 versicolor
## 57           6.3         3.3          4.7         1.6 versicolor
## 58           4.9         2.4          3.3         1.0 versicolor
## 59           6.6         2.9          4.6         1.3 versicolor
## 60           5.2         2.7          3.9         1.4 versicolor
## 61           5.0         2.0          3.5         1.0 versicolor
## 62           5.9         3.0          4.2         1.5 versicolor
## 63           6.0         2.2          4.0         1.0 versicolor
## 64           6.1         2.9          4.7         1.4 versicolor
## 65           5.6         2.9          3.6         1.3 versicolor
## 66           6.7         3.1          4.4         1.4 versicolor
## 67           5.6         3.0          4.5         1.5 versicolor
## 68           5.8         2.7          4.1         1.0 versicolor
## 69           6.2         2.2          4.5         1.5 versicolor
## 70           5.6         2.5          3.9         1.1 versicolor
## 71           5.9         3.2          4.8         1.8 versicolor
## 72           6.1         2.8          4.0         1.3 versicolor
## 73           6.3         2.5          4.9         1.5 versicolor
## 74           6.1         2.8          4.7         1.2 versicolor
## 75           6.4         2.9          4.3         1.3 versicolor
## 76           6.6         3.0          4.4         1.4 versicolor
## 77           6.8         2.8          4.8         1.4 versicolor
## 78           6.7         3.0          5.0         1.7 versicolor
## 79           6.0         2.9          4.5         1.5 versicolor
## 80           5.7         2.6          3.5         1.0 versicolor
## 81           5.5         2.4          3.8         1.1 versicolor
## 82           5.5         2.4          3.7         1.0 versicolor
## 83           5.8         2.7          3.9         1.2 versicolor
## 84           6.0         2.7          5.1         1.6 versicolor
## 85           5.4         3.0          4.5         1.5 versicolor
## 86           6.0         3.4          4.5         1.6 versicolor
## 87           6.7         3.1          4.7         1.5 versicolor
## 88           6.3         2.3          4.4         1.3 versicolor
## 89           5.6         3.0          4.1         1.3 versicolor
## 90           5.5         2.5          4.0         1.3 versicolor
## 91           5.5         2.6          4.4         1.2 versicolor
## 92           6.1         3.0          4.6         1.4 versicolor
## 93           5.8         2.6          4.0         1.2 versicolor
## 94           5.0         2.3          3.3         1.0 versicolor
## 95           5.6         2.7          4.2         1.3 versicolor
## 96           5.7         3.0          4.2         1.2 versicolor
## 97           5.7         2.9          4.2         1.3 versicolor
## 98           6.2         2.9          4.3         1.3 versicolor
## 99           5.1         2.5          3.0         1.1 versicolor
## 100          5.7         2.8          4.1         1.3 versicolor
## 101          6.3         3.3          6.0         2.5  virginica
## 102          5.8         2.7          5.1         1.9  virginica
## 103          7.1         3.0          5.9         2.1  virginica
## 104          6.3         2.9          5.6         1.8  virginica
## 105          6.5         3.0          5.8         2.2  virginica
## 106          7.6         3.0          6.6         2.1  virginica
## 107          4.9         2.5          4.5         1.7  virginica
## 108          7.3         2.9          6.3         1.8  virginica
## 109          6.7         2.5          5.8         1.8  virginica
## 110          7.2         3.6          6.1         2.5  virginica
## 111          6.5         3.2          5.1         2.0  virginica
## 112          6.4         2.7          5.3         1.9  virginica
## 113          6.8         3.0          5.5         2.1  virginica
## 114          5.7         2.5          5.0         2.0  virginica
## 115          5.8         2.8          5.1         2.4  virginica
## 116          6.4         3.2          5.3         2.3  virginica
## 117          6.5         3.0          5.5         1.8  virginica
## 118          7.7         3.8          6.7         2.2  virginica
## 119          7.7         2.6          6.9         2.3  virginica
## 120          6.0         2.2          5.0         1.5  virginica
## 121          6.9         3.2          5.7         2.3  virginica
## 122          5.6         2.8          4.9         2.0  virginica
## 123          7.7         2.8          6.7         2.0  virginica
## 124          6.3         2.7          4.9         1.8  virginica
## 125          6.7         3.3          5.7         2.1  virginica
## 126          7.2         3.2          6.0         1.8  virginica
## 127          6.2         2.8          4.8         1.8  virginica
## 128          6.1         3.0          4.9         1.8  virginica
## 129          6.4         2.8          5.6         2.1  virginica
## 130          7.2         3.0          5.8         1.6  virginica
## 131          7.4         2.8          6.1         1.9  virginica
## 132          7.9         3.8          6.4         2.0  virginica
## 133          6.4         2.8          5.6         2.2  virginica
## 134          6.3         2.8          5.1         1.5  virginica
## 135          6.1         2.6          5.6         1.4  virginica
## 136          7.7         3.0          6.1         2.3  virginica
## 137          6.3         3.4          5.6         2.4  virginica
## 138          6.4         3.1          5.5         1.8  virginica
## 139          6.0         3.0          4.8         1.8  virginica
## 140          6.9         3.1          5.4         2.1  virginica
## 141          6.7         3.1          5.6         2.4  virginica
## 142          6.9         3.1          5.1         2.3  virginica
## 143          5.8         2.7          5.1         1.9  virginica
## 144          6.8         3.2          5.9         2.3  virginica
## 145          6.7         3.3          5.7         2.5  virginica
## 146          6.7         3.0          5.2         2.3  virginica
## 147          6.3         2.5          5.0         1.9  virginica
## 148          6.5         3.0          5.2         2.0  virginica
## 149          6.2         3.4          5.4         2.3  virginica
## 150          5.9         3.0          5.1         1.8  virginica
```

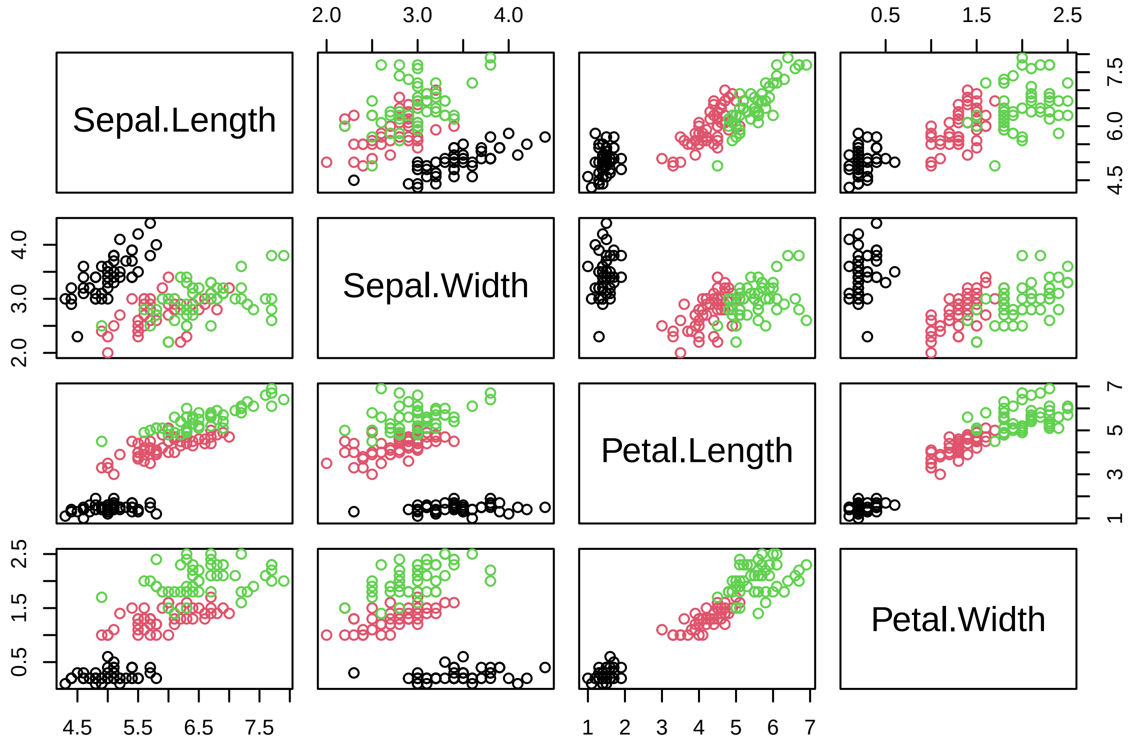
```r
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

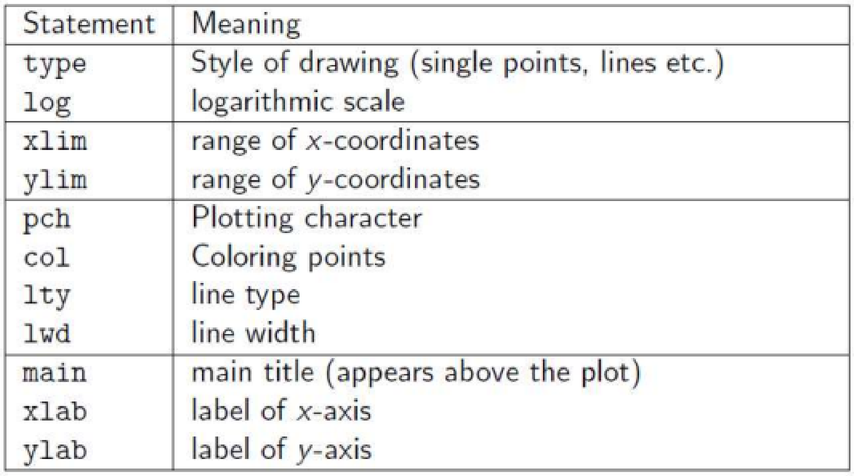
```r
# A factor represents categorical values with different "levels"
# High-level plotting function: opens a plot
plot(x = iris$Sepal.Width, y =iris$Sepal.Length, col = iris[ , "Species"], pch = 19)

#Low-level plotting functions: add to an existing plot
legend("topright", legend = levels(iris[ , "Species"]), pch = 19, col = 1:3) # adds legend
```



```r
# In an Rmd file such as this one, you need to call both the plot() & legend() functions at the same ti

pairs(iris[ , -5], col = iris[ , 5])
```



Arguments of plot



Statement	Meaning
type	Style of drawing (single points, lines etc.)
log	logarithmic scale
xlim	range of x-coordinates
ylim	range of y-coordinates
pch	Plotting character
col	Coloring points
lty	line type
lwd	line width
main	main title (appears above the plot)
xlab	label of x-axis
ylab	label of y-axis

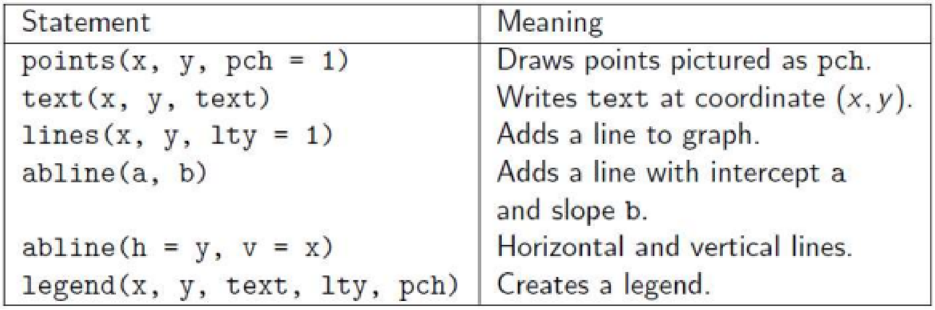
Figure 2: **Plot Arguments**

Three categories of R graphics functions:

- High-level plotting functions such as plot() to generate a new graphics display.
- Low-level plotting functions such as legend() to add further graphical elements to an existing plot.

- Interactive functions such as identify() to amend or collect information interactively from a plot.

Low-level plotting functions



Statement	Meaning
`points(x, y, pch = 1)`	Draws points pictured as pch.
`text(x, y, text)`	Writes text at coordinate (x, y).
`lines(x, y, lty = 1)`	Adds a line to graph.
`abline(a, b)`	Adds a line with intercept a and slope b.
`abline(h = y, v = x)`	Horizontal and vertical lines.
`legend(x, y, text, lty, pch)`	Creates a legend.

Figure 3: Low-level plotting functions

Useful plot functions

- plot, pairs, interaction.plot
- boxplot, hist
- plot3d

Graphical Output

```
pdf(file = "iris_plot.pdf") # open the graphics device
plot(Sepal.Length ~ Sepal.Width, data = iris)
# add anything else you want in your plots
dev.off() #close the graphic device
```

```
## pdf
##   2
```

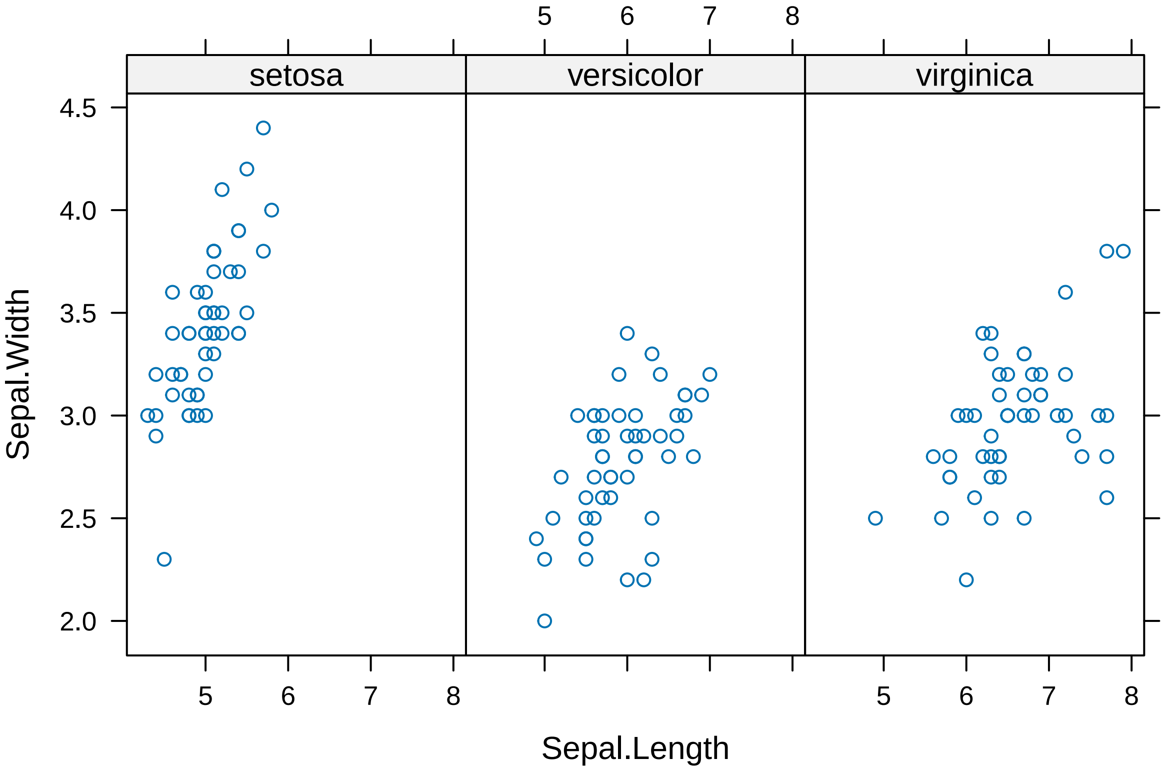
Several plots in one graphical window: splits the graphical window into 3 rows and 2 columns.

```
par(mfrow=c(3,2))
```

Other Graphics: in lattice

The lattice package functions: good for repeating graphs for various groups. See the **Lattice Graphs in R** (at http://www.statmethods.net/advgraphs/trellis.html) for more information.

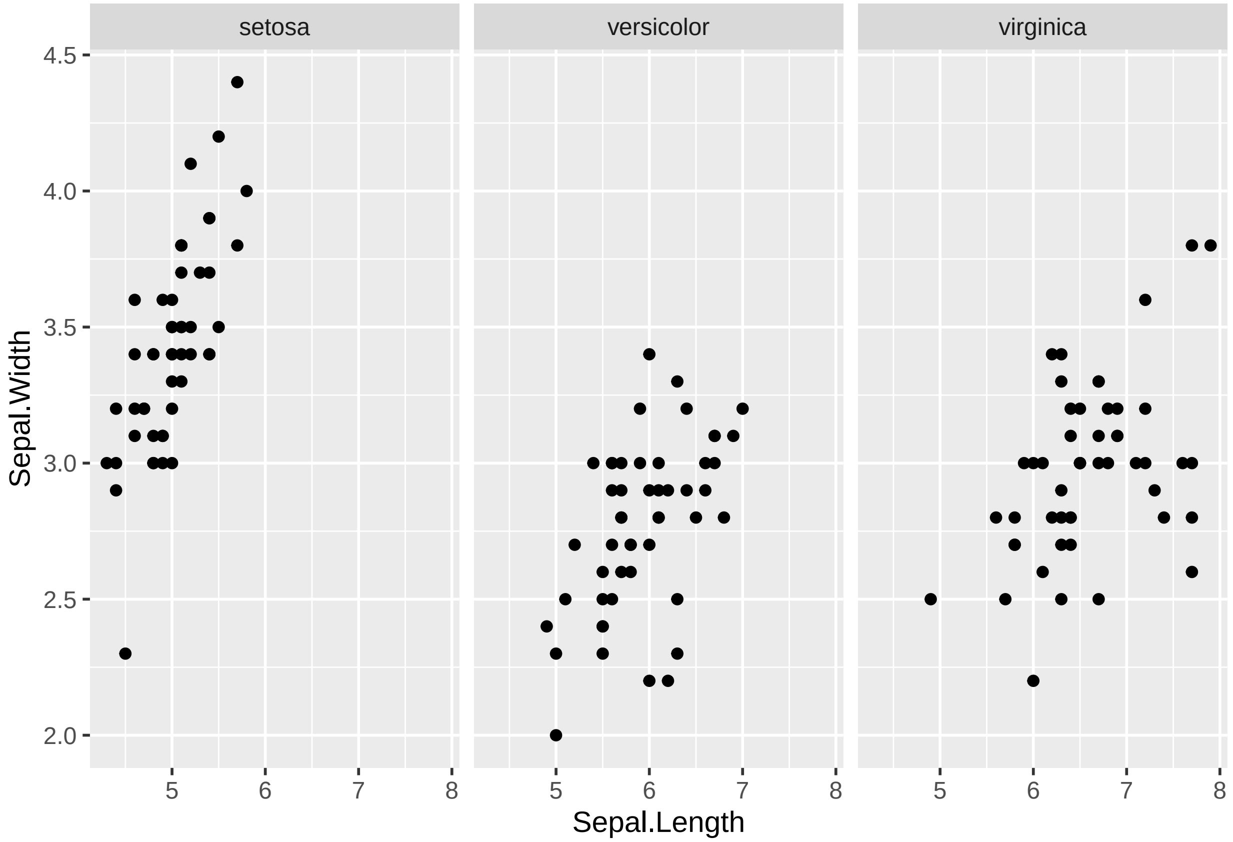
```
pacman::p_load(lattice)
xyplot(Sepal.Width ~ Sepal.Length | Species, data = iris)
```



Other Graphics: in ggplot2

ggplot2 package: very flexible, based on grammar of graphics.

```
pacman::p_load(ggplot2)
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) + facet_grid(rows = ~ Species) + geom_point
```



Hypothesis Testing

Approach: Hypothesis testing in 6 steps:

1. Declare *model* by which data were generated (e.g. population is normally distributed, large sample size and   not known).
2. Define null hypothesis, $\mathbf{H_0}$ and alternative hypothesis, $\mathbf{H_A}$; where $\mathbf{H_0}$ is the statement being tested in a test of (statistical) significance and $\mathbf{H_A}$ is the statement that is hoped or expected to be true instead of the null hypothesis
3. Choose the *level of significance*, .
4. Determine *critical values* for the *level of significance   and *degrees of freedom*, df = (n-1)
5. Define and calculate **test statistic**, e.g. one-sample test:
6. **Compare** the test statistic to the **critical values** and make **decision** to **reject** or **fail to reject** H_0 .

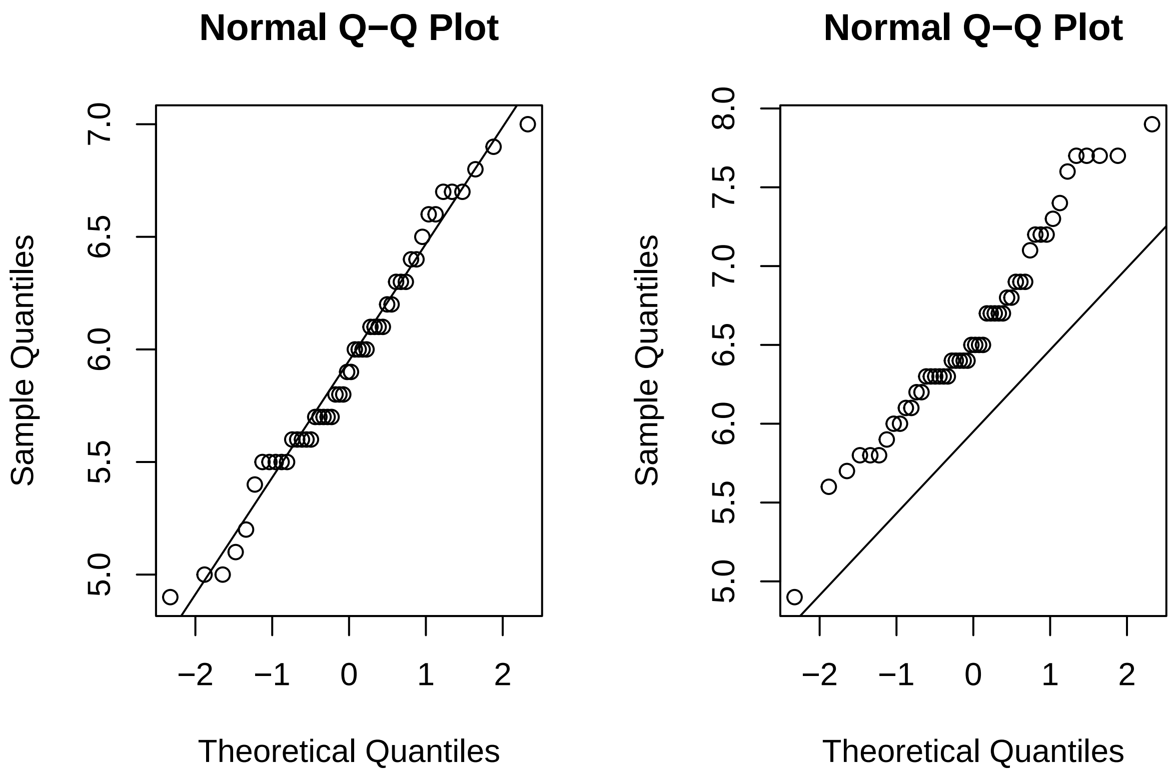
Hypothesis Tests – An Example

```
# Is the sepal length of versicolor different to that of virginica?
#Let's use a *t-Test and a *Wilcoxon rank-sum test.
testdata <- iris[iris$Species != "setosa", c("Sepal.Length", "Species")]
testdata$Species <- droplevels(testdata$Species)
str(testdata) # prepare and check the data
```

```
## 'data.frame':    100 obs. of  2 variables:
```

```
##  $ Sepal.Length: num  7 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2 ...
##  $ Species     : Factor w/ 2 levels "versicolor","virginica": 1 1 1 1 1 1 1 1 1 1 ...
```

```
# Check normality assumption of t-Test using QQ-Plot:
versi.id <- testdata$Species == "versicolor"
par(mfrow=c(1,2))
qqnorm(testdata$Sepal.Length[versi.id]); qqline(testdata$Sepal.Length[versi.id])
qqnorm(testdata$Sepal.Length[!versi.id]); qqline(testdata$Sepal.Length[versi.id])
```

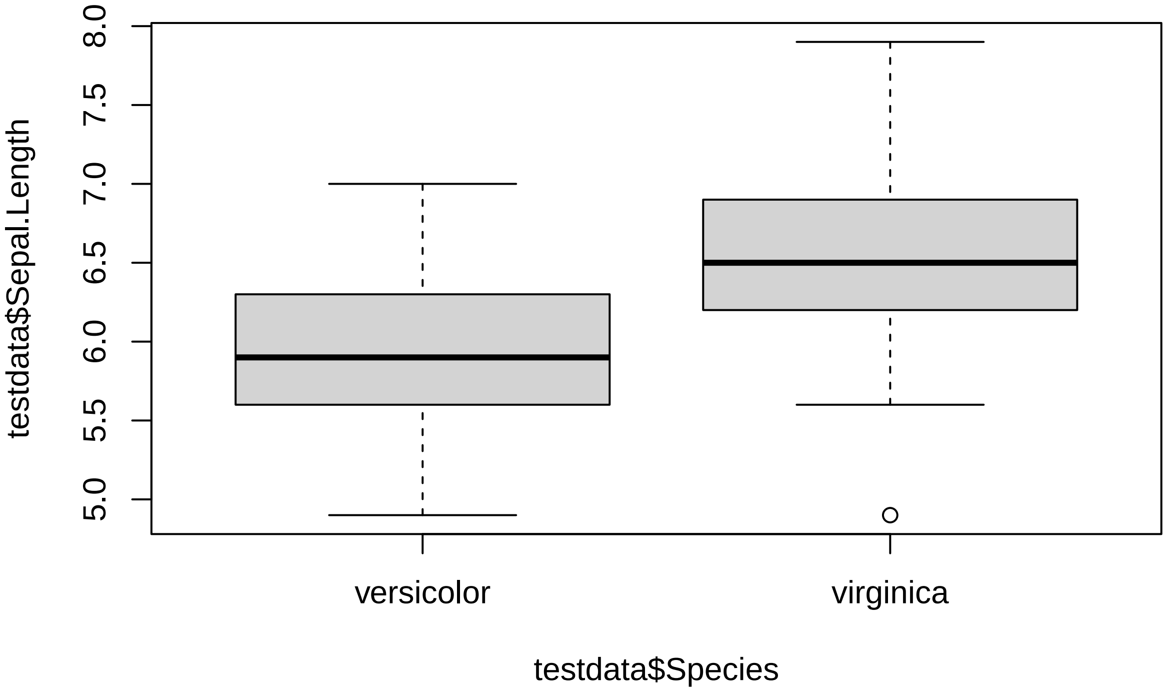


```
# Two-sample t-test:
versi.id <- testdata$Species == "versicolor"
t.test(x =testdata$Sepal.Length[versi.id], y = testdata$Sepal.Length[!versi.id], var.equal = TRUE)
```

```
##
##  Two Sample t-test
##
## data:  testdata$Sepal.Length[versi.id] and testdata$Sepal.Length[!versi.id]
## t = -5.6292, df = 98, p-value = 1.725e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8818516 -0.4221484
## sample estimates:
## mean of x mean of y
##     5.936     6.588
```

```
# T-test rejects the null hypothesis at 5% significance level.
# Do not forget to visually check the normality assumptions (QQ-plot)
```

```
# Check assumption of \*Wilcoxon Rank Test: same distribution, just a location shift.
boxplot(testdata$Sepal.Length ~ testdata$Species)
```



```
# Now, perform the test:
versi.id <- testdata$Species == "versicolor"
wilcox.test(x =testdata$Sepal.Length[versi.id], y = testdata$Sepal.Length[!versi.id], var.equal = TRUE)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  testdata$Sepal.Length[versi.id] and testdata$Sepal.Length[!versi.id]
## W = 526, p-value = 5.869e-07
## alternative hypothesis: true location shift is not equal to 0
```

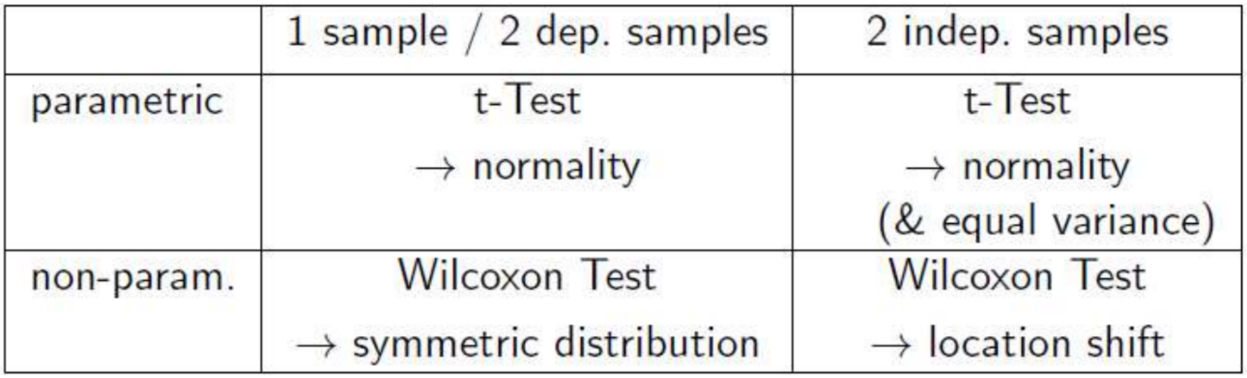
```
# Wilcoxon Rank Test also rejects the null hypothesis at 5% significance level.
```

The **Wilcoxon Rank Test** is the *preferred* test for a two-sample statistical test.

Hypothesis Tests - Summary

How to proceed:

- Formulate the null & alternative hypotheses
- Choose the appropriate test
- Collate data, i.e., do an experiment
- Look at data: plot(), pairs(), hist(), boxplot()
- Validate assumptions of test (e.g., T-test, Wilcoxon test)
- Carry out the test and interpret result



	1 sample / 2 dep. samples	2 indep. samples
parametric	t-Test → normality	t-Test → normality (& equal variance)
non-param.	Wilcoxon Test → symmetric distribution	Wilcoxon Test → location shift

Figure 4: Hypothesis Tests

Hypothesis Tests – **Chi-squared test of independence**

- Hypothesis: H_0: Independence of education and marriage status
- H_A: Dependence of education and marriage status

```r
url <- "https://stat.ethz.ch/Teaching/Datasets/edu.txt"
d.edu <- read.table(url, header = TRUE)

# Cross-tables in R
# Count number of cases with same value:
table(d.edu[, "Married"])
```

```
##
## Married more Married once
##          205         1231
```

```r
# Cross-table
table(d.edu[, "Education"], d.edu[, "Married"])
```

```
##
##              Married more Married once
##   College              61          550
##   No College          144          681
```

```r
# Now we perform a Chi-squared test
chisq.test(d.edu[, "Education"], d.edu[, "Married"])
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  d.edu[, "Education"] and d.edu[, "Married"]
## X-squared = 15.405, df = 1, p-value = 8.675e-05
```

```
# Result: Reject H0, i.e. education and marriage are dependent.
```

Correlation

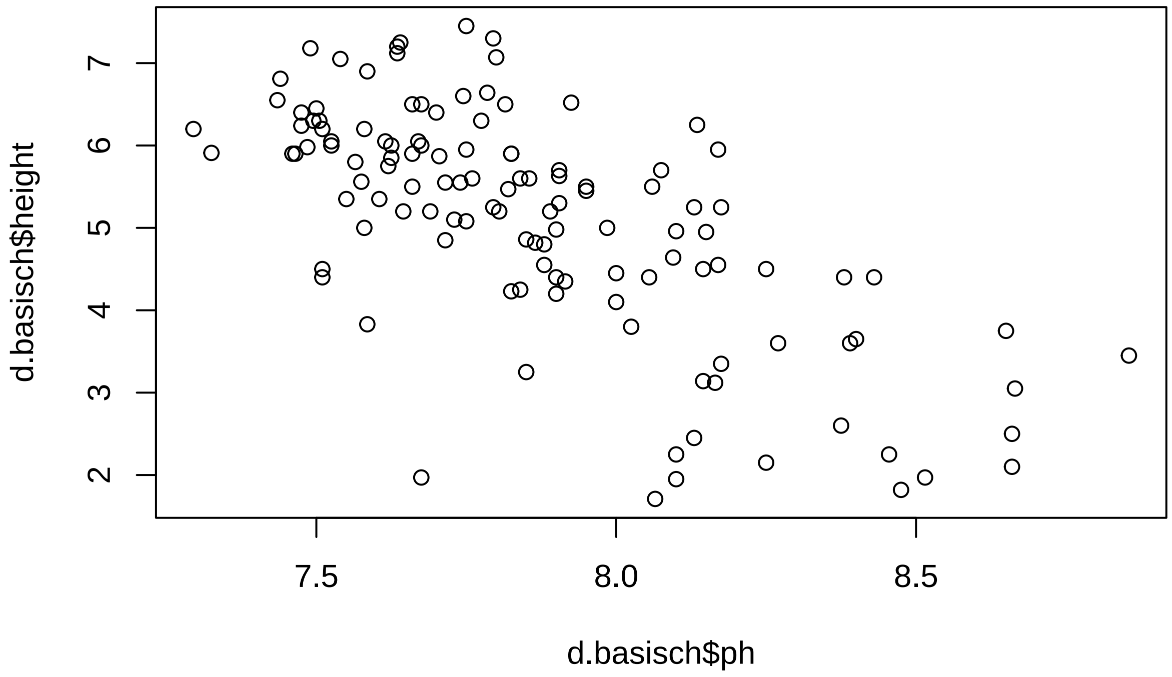
```
# Correlation
url1 <- "https://stat.ethz.ch/Teaching/Datasets/basischOhneNA.dat"
d.basisch <- read.table(url1, header = TRUE)
str(d.basisch)
```

```
## 'data.frame':    123 obs. of  4 variables:
##  $ ph     : num  7.33 7.69 7.9 8.14 7.62 ...
##  $ l.sar  : num  0.0969 0.4393 1 1.316 0.0607 ...
##  $ height: num  5.91 5.2 4.4 4.5 6.05 6 5.35 5.55 4.95 5.2 ...
##  $ h.quad: num  34.9 27 19.4 20.2 36.6 ...
```

```
# Calculate the (Pearson) correlation of ph and height:
cor(d.basisch$ph, d.basisch$height)
```

```
## [1] -0.6925717
```

```
# Corresponding plot:
plot(d.basisch$ph, d.basisch$height)
```



All plots show 2 variables with a correlation of 0.7

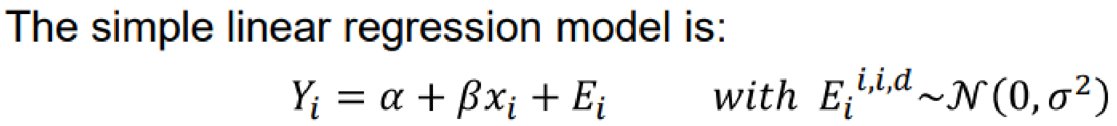
- one looks good
- another does not
- outlier(s) influence the result
- ALWAYS FIRST LOOK AT PLOTS

Regression

Simple Linear Regression (SLR)

From the d.basisch() data,

1. **Response variable**: *height* or h.quad: Height of trees or squared height, respectively.
2. **Possible explanatory variables**: *ph*: pH-values of soil and l.sar: log(sodium absorption ratio)



The simple linear regression model is:

$$Y_i = \alpha + \beta x_i + E_i \qquad with\ \ {E_i}^{i,i,d} \sim \mathcal{N}(0, \sigma^2)$$

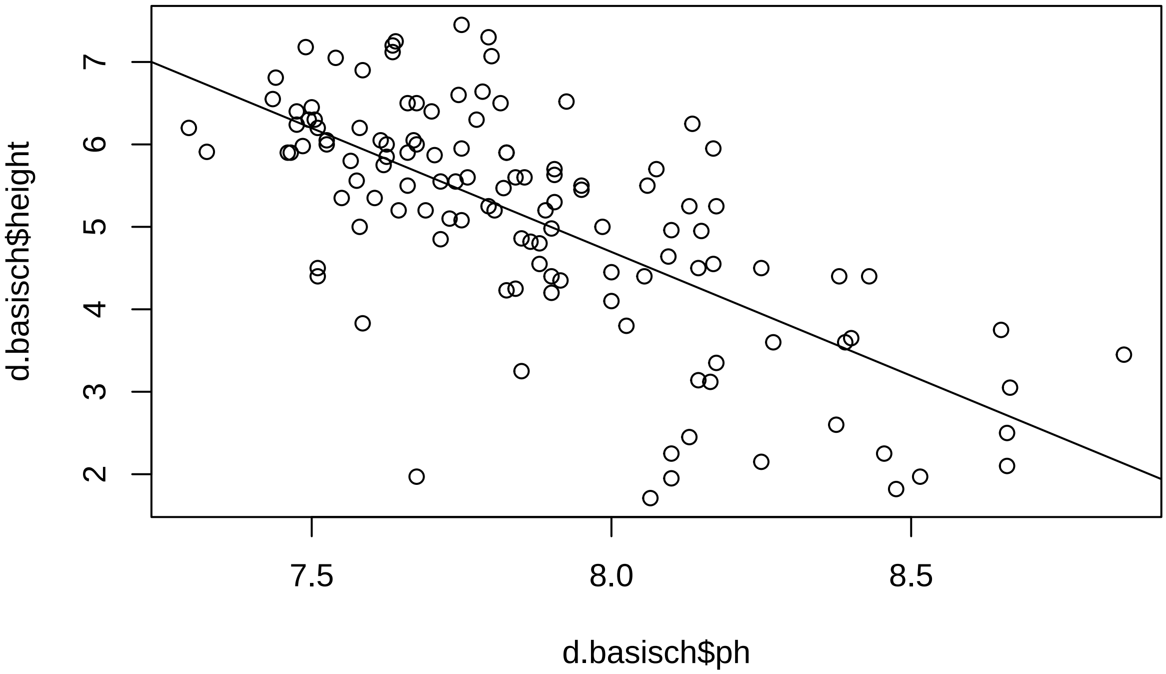
Figure 5: The Simple Linear Regression Model

Let us pick the variable ph as the explanatory variable.

```
# Fit to data using lm:
fit <- lm(formula = height ~ ph, data = d.basisch)
summary(fit)
```

```
##
## Call:
## lm(formula = height ~ ph, data = d.basisch)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7020 -0.5471  0.0874  0.6663  2.0033
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  28.7227     2.2395   12.82   <2e-16 ***
## ph           -3.0034     0.2844  -10.56   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.008 on 121 degrees of freedom
## Multiple R-squared:  0.4797, Adjusted R-squared:  0.4754
## F-statistic: 111.5 on 1 and 121 DF,  p-value: < 2.2e-16
```

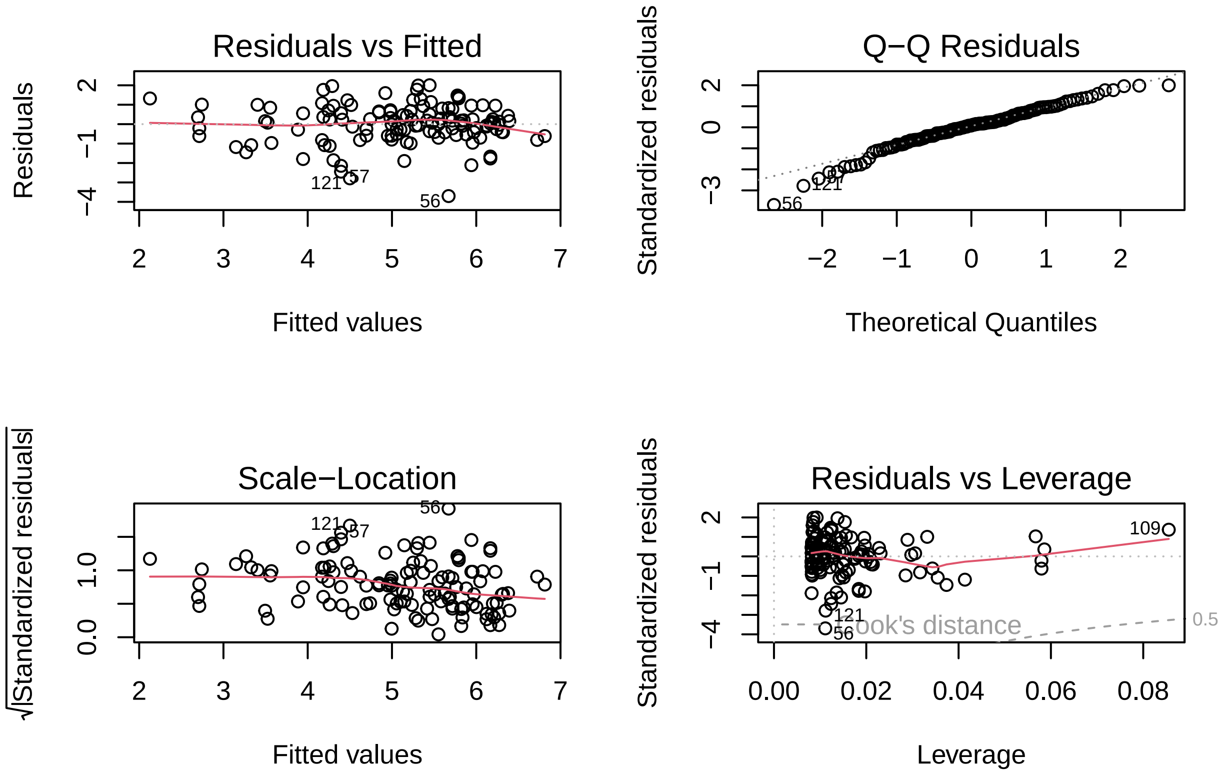
```
# Estimated equation: height = 28.7 - 3.0pH

# Drawing line into scatterplot:
plot(d.basisch$ph, d.basisch$height) + abline(fit)
```



```
## integer(0)
```

SLR - Residual Analysis Diagnostics plots are straightforward:

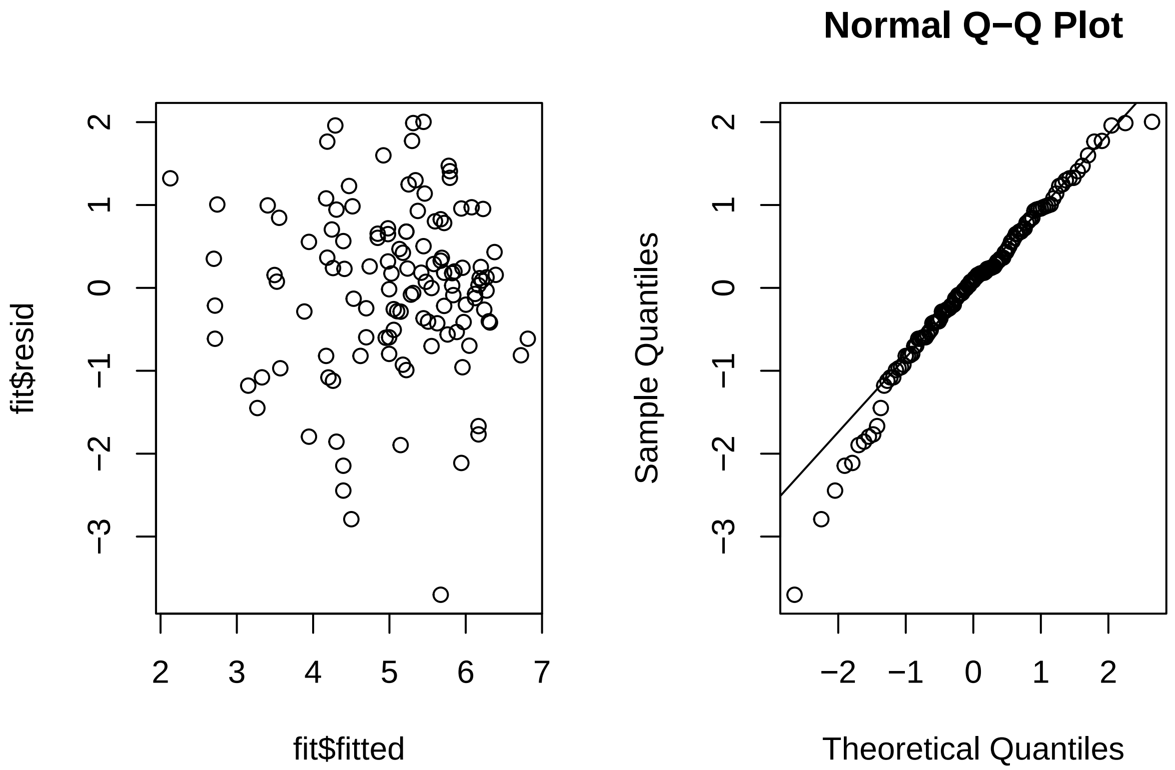
```
par(mfrow = c(2,2))
plot(fit)
```



1. Tukey-Anscombe plot (is the variance of the errors Ei constant? Is the regression function correct?)
2. Q-Q plot (are the errors Ei normally distributed?)
3. Scale location plot (similar to Tukey-Anscombe plot)
4. Leverage plot (what points have a strong influence on the fit?)

Residual plots by hand:

```r
par(mfrow = c(1,2))
plot(fit$fitted,fit$resid)

#Tukey-Anscombe
qqnorm(fit$resid) #Quantile-Quantile Plot
qqline(fit$resid) # adds the diagonal line
```



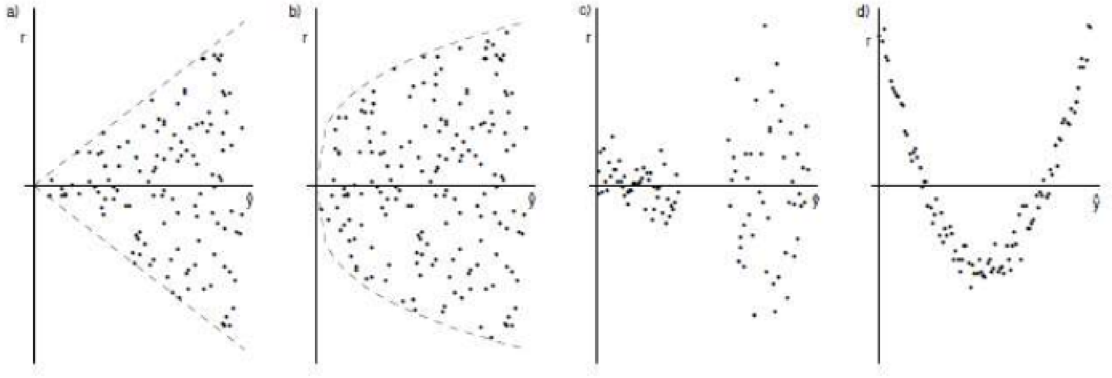


Figure 6: Some bad Tukey-Anscombe plots

Multiple Linear Regression

Expand the simple linear model to more than one explanatory variable.

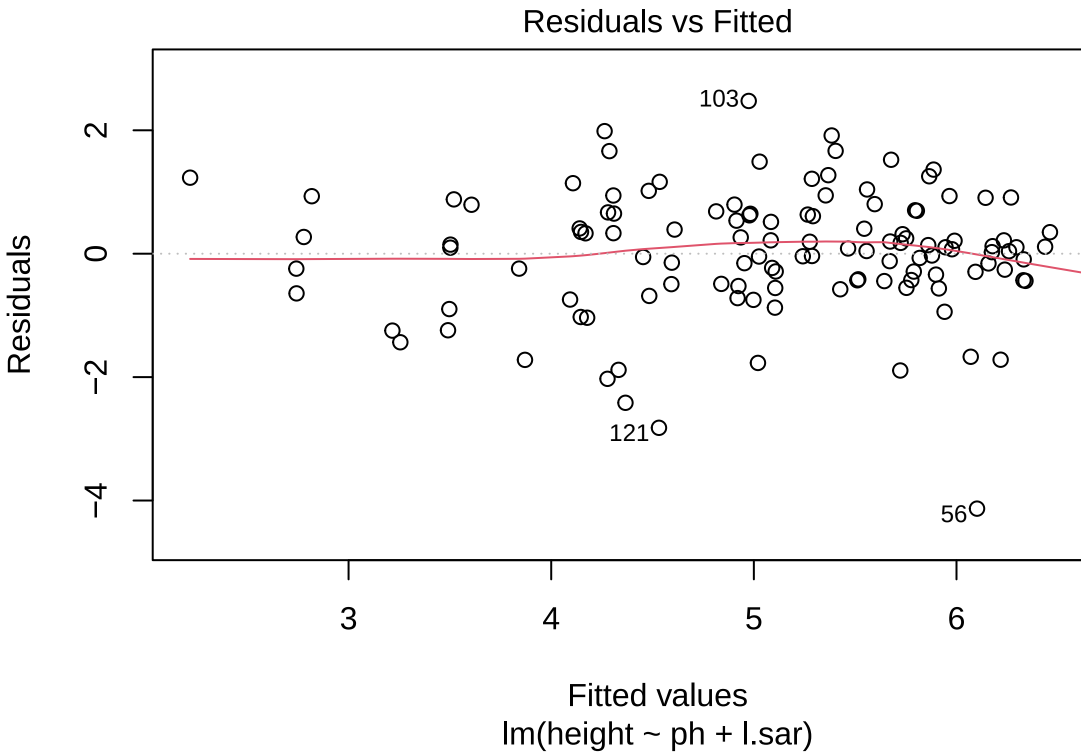
$$Y_i = \alpha + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + E_i \qquad with\ \ E_i{}^{i,i,d} \sim \mathcal{N}(0, \sigma^2)$$

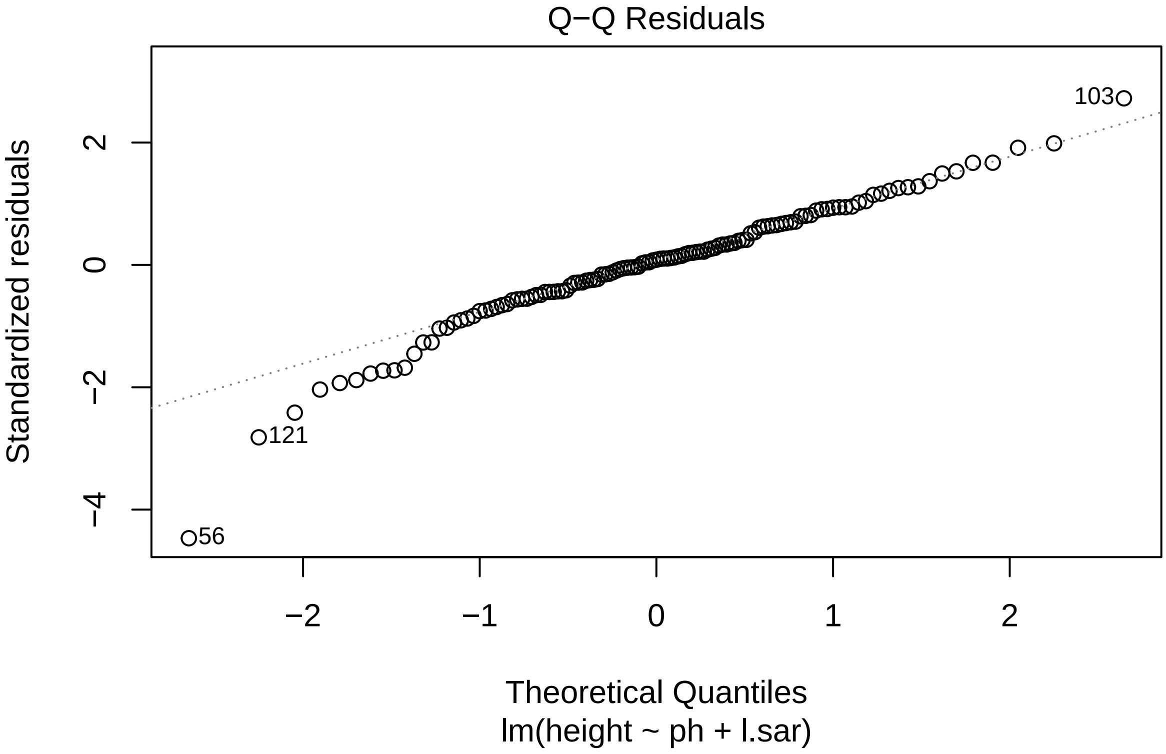
Figure 7: The Multiple Linear Regression Model

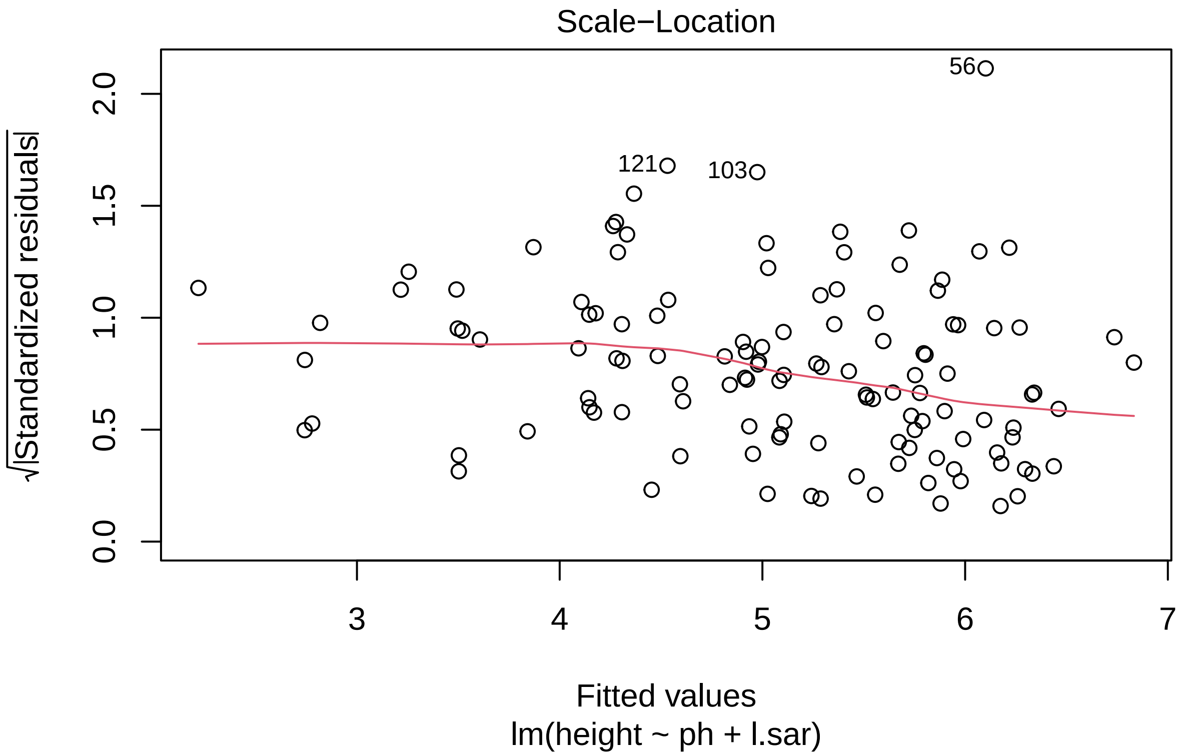
```
# Fit the model with lm
fitm <- lm(height ~ ph + l.sar, data = d.basisch)
summary(fitm)
```

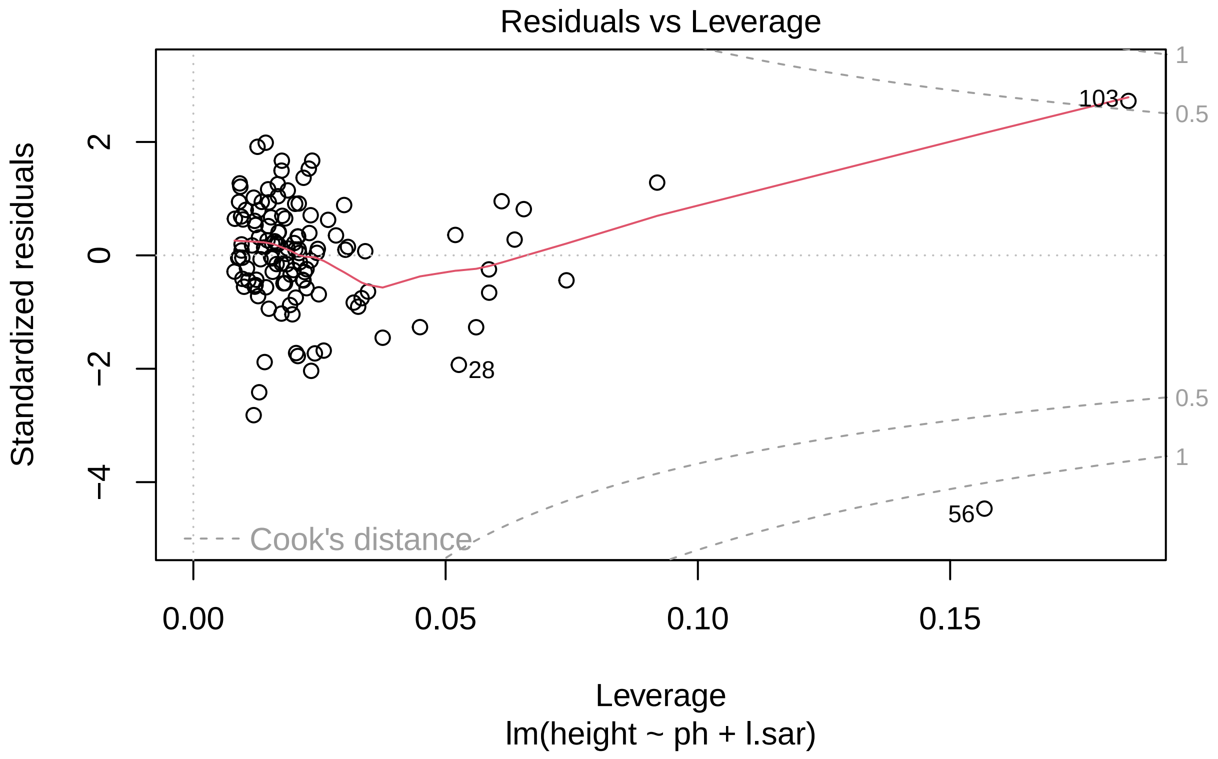
```
##
## Call:
## lm(formula = height ~ ph + l.sar, data = d.basisch)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.1314 -0.4911  0.0849  0.6488  2.4754
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  26.9466     2.7445   9.818  < 2e-16 ***
## ph           -2.7558     0.3603  -7.649  5.6e-12 ***
## l.sar        -0.2519     0.2255  -1.117    0.266
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.007 on 120 degrees of freedom
## Multiple R-squared:  0.485,  Adjusted R-squared:  0.4764
## F-statistic: 56.51 on 2 and 120 DF,  p-value: < 2.2e-16
```

```
# Look at the same plots as for simple linear regression
plot(fitm)
```

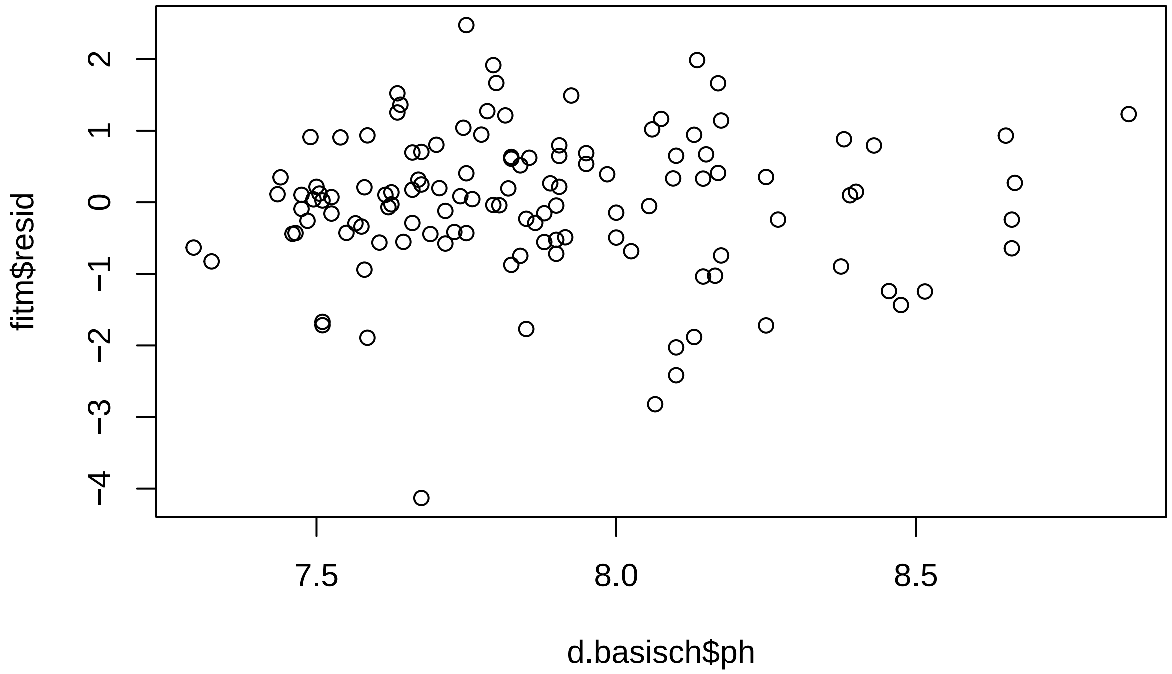

Residuals vs Fitted
Residuals
2
0
−2
−4
103
121
56
3
4
5
6
Fitted values
lm(height ~ ph + l.sar)


Q−Q Residuals
Standardized residuals
2
0
−2
−4
103
121
56
−2
−1
0
1
2
Theoretical Quantiles
lm(height ~ ph + l.sar)

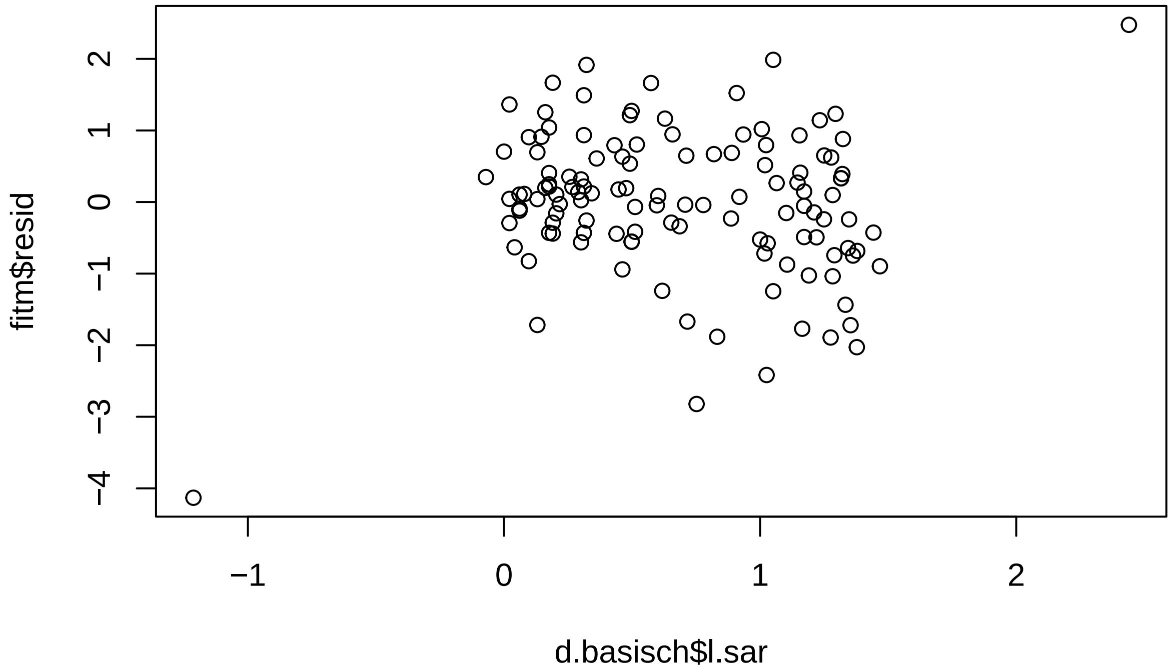

Scale−Location
√|Standardized residuals|
0.0
0.5
1.0
1.5
2.0
3
4
5
6
7
Fitted values
lm(height ~ ph + l.sar)
56
121
103



```
# It may help to plot the explanatory variables against the residuals.
plot(d.basisch$ph, fitm$resid)
```



```
plot(d.basisch$l.sar, fitm$resid)
```



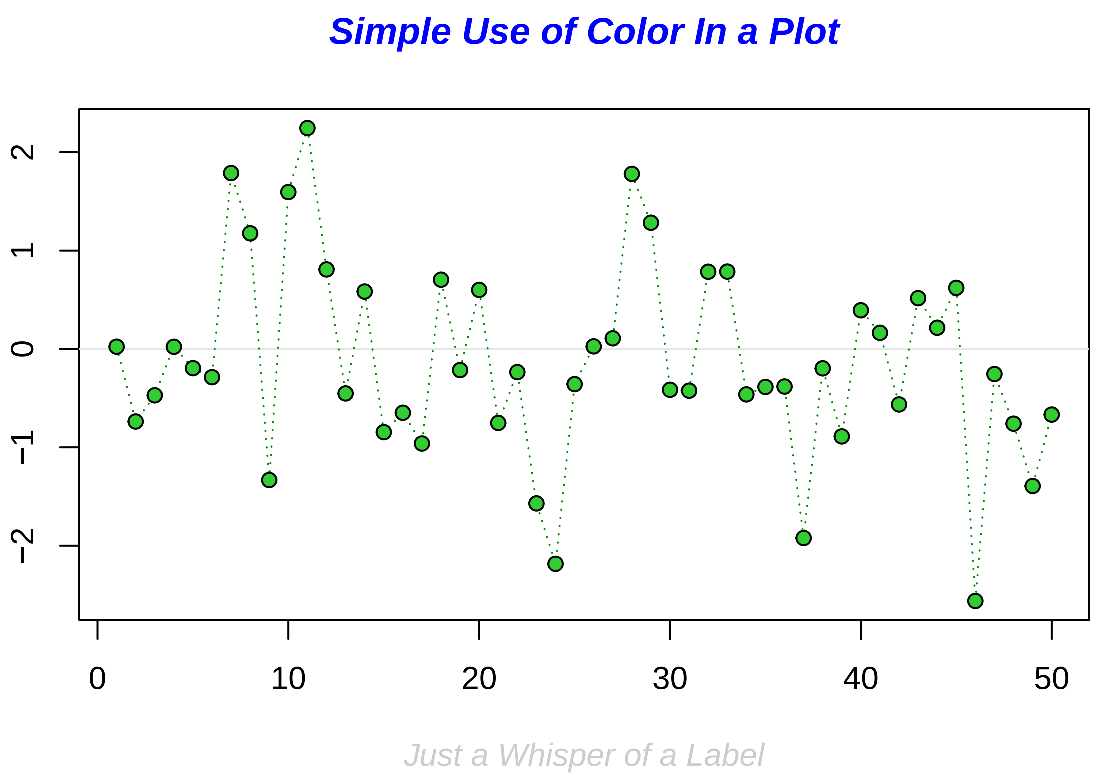
Other

Graphics - Demo

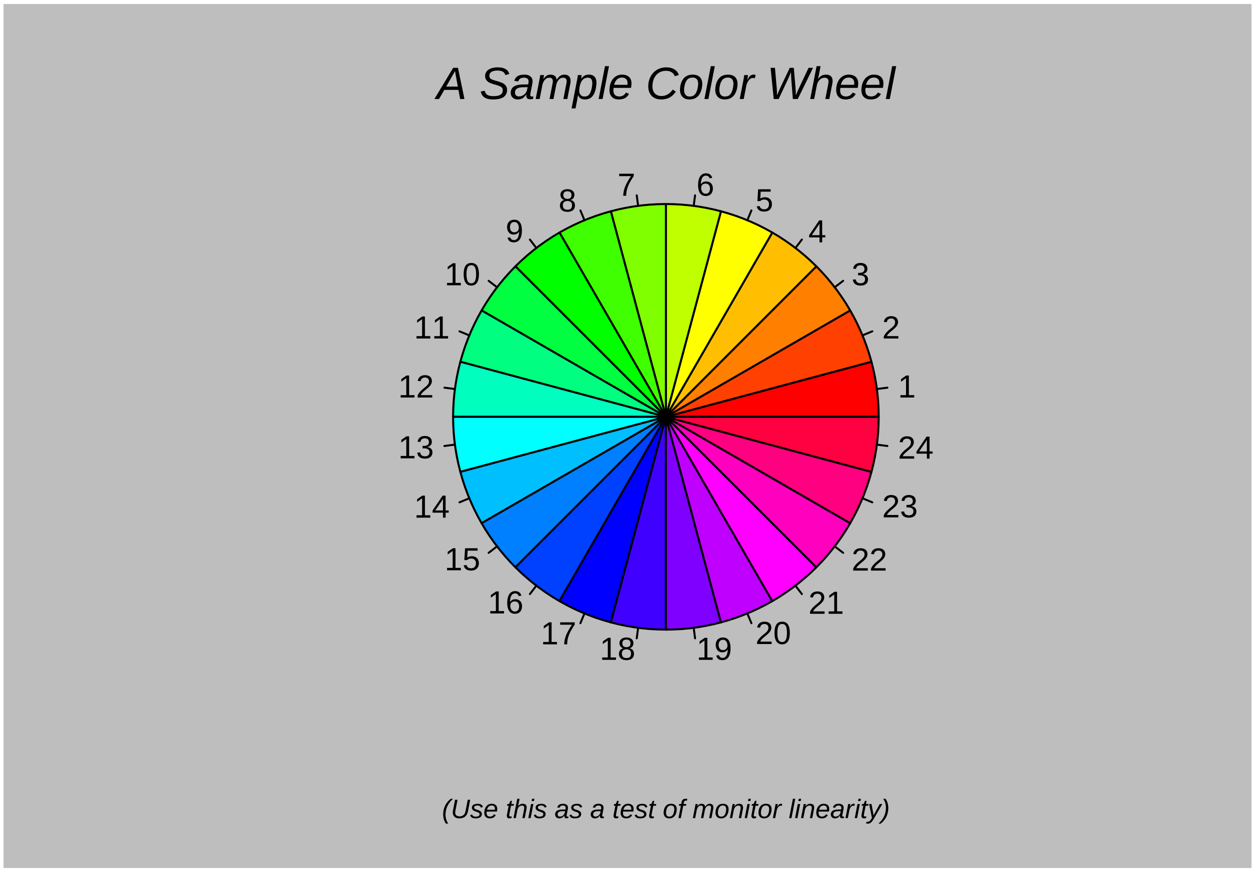
```
demo(graphics)
```

```
##
##
##  demo(graphics)
##  ---- ~~~~~~~~
##
## > #  Copyright (C) 1997-2009 The R Core Team
## >
## > require(datasets)
##
## > require(grDevices); require(graphics)
##
## > ## Here is some code which illustrates some of the differences between
## > ## R and S graphics capabilities.  Note that colors are generally specified
## > ## by a character string name (taken from the X11 rgb.txt file) and that line
## > ## textures are given similarly.  The parameter "bg" sets the background
## > ## parameter for the plot and there is also an "fg" parameter which sets
```

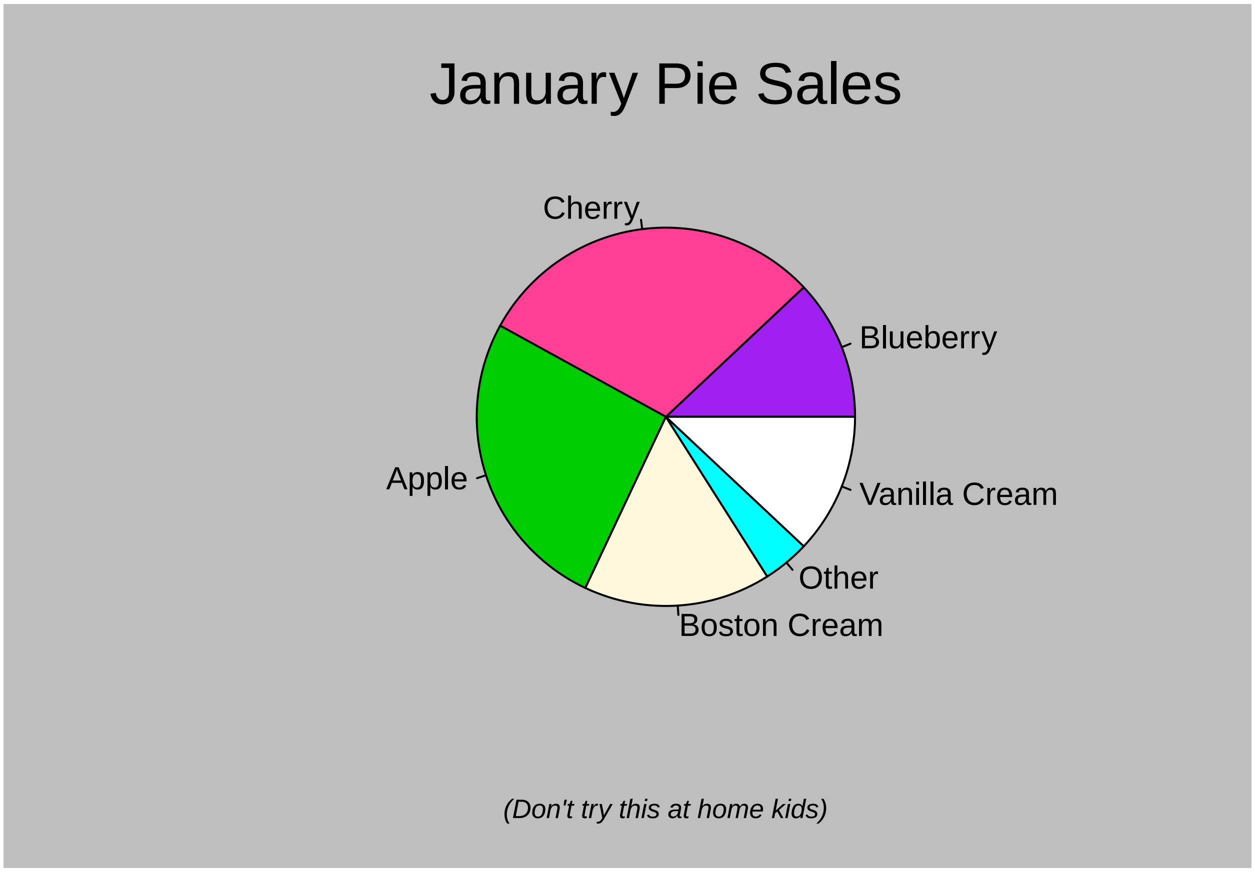
```
## > ## the foreground color.
## >
## >
## > x <- stats::rnorm(50)
##
## > opar <- par(bg = "white")
##
## > plot(x, ann = FALSE, type = "n")
```



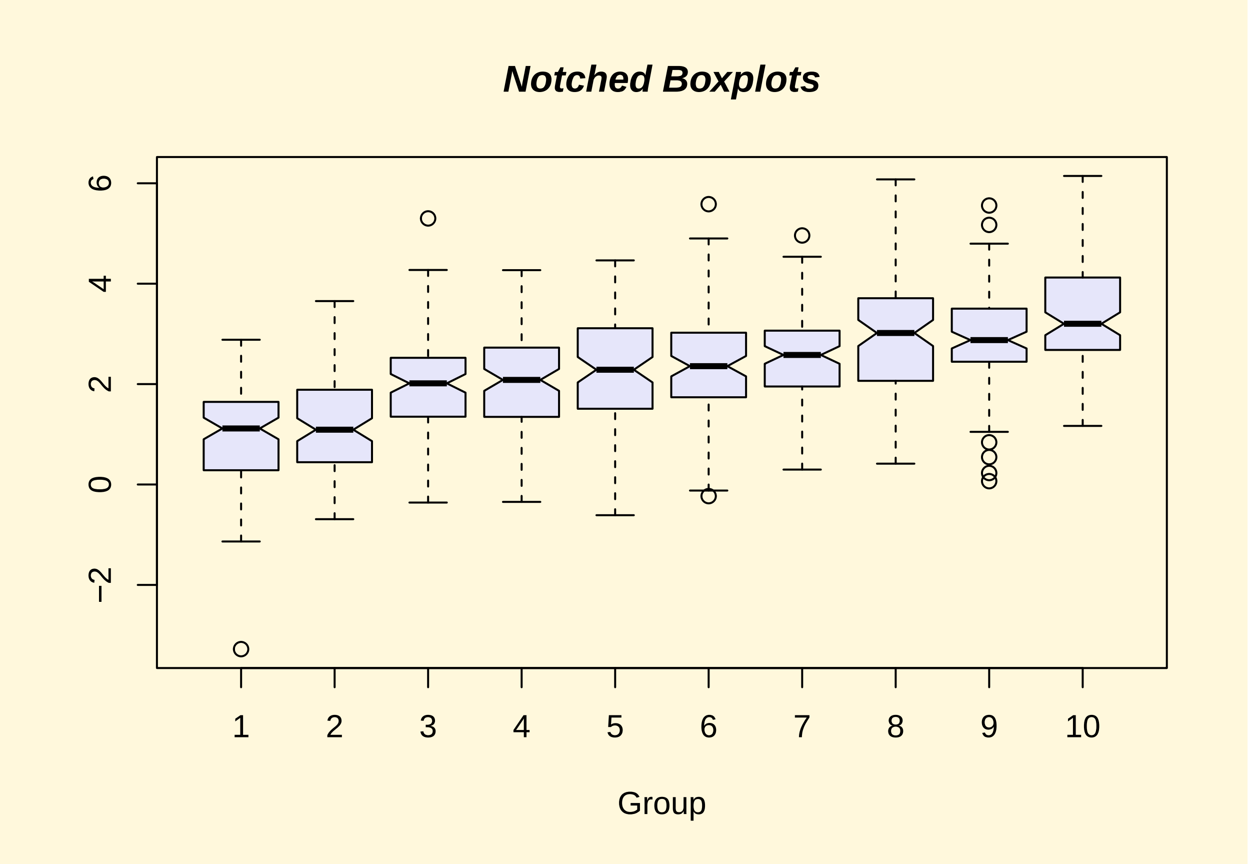
```
##
## > abline(h = 0, col = gray(.90))
##
## > lines(x, col = "green4", lty = "dotted")
##
## > points(x, bg = "limegreen", pch = 21)
##
## > title(main = "Simple Use of Color In a Plot",
## +       xlab = "Just a Whisper of a Label",
## +       col.main = "blue", col.lab = gray(.8),
## +       cex.main = 1.2, cex.lab = 1.0, font.main = 4, font.lab = 3)
##
## > ## A little color wheel.    This code just plots equally spaced hues in
## > ## a pie chart.    If you have a cheap SVGA monitor (like me) you will
## > ## probably find that numerically equispaced does not mean visually
## > ## equispaced.  On my display at home, these colors tend to cluster at
## > ## the RGB primaries.  On the other hand on the SGI Indy at work the
## > ## effect is near perfect.
## >
## > par(bg = "gray")
##
## > pie(rep(1,24), col = rainbow(24), radius = 0.9)
```



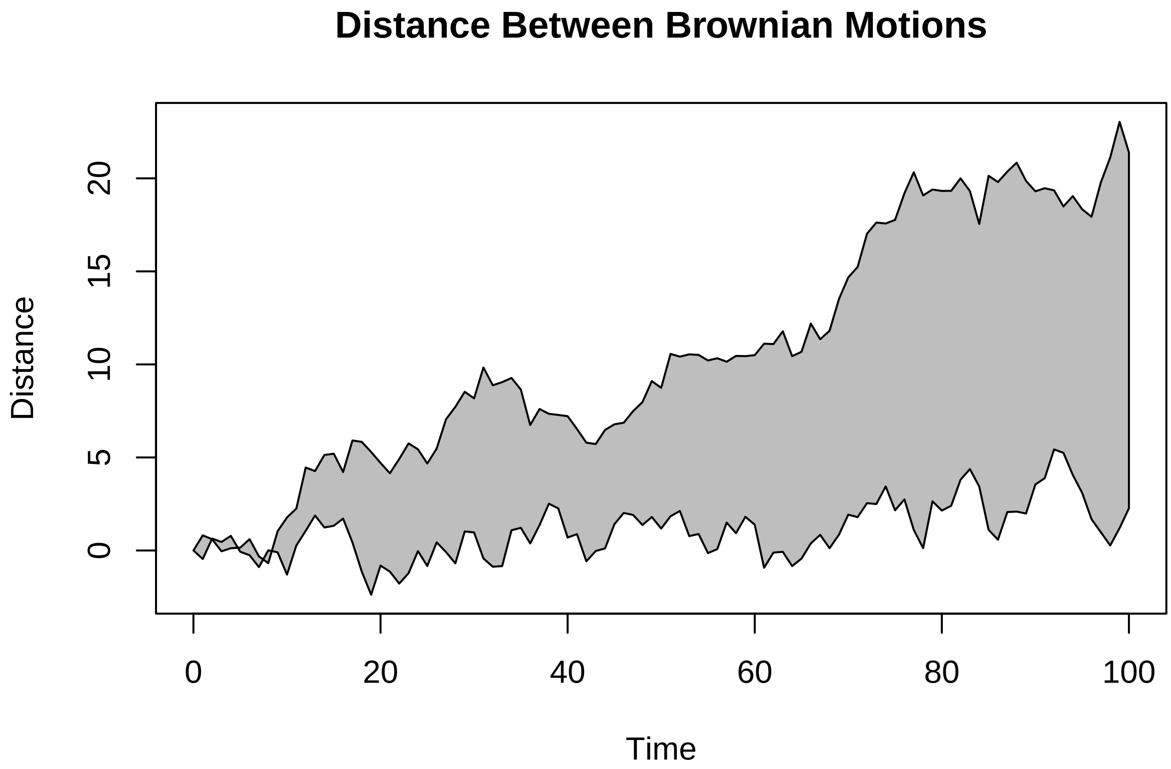
```
##
## > title(main = "A Sample Color Wheel", cex.main = 1.4, font.main = 3)
##
## > title(xlab = "(Use this as a test of monitor linearity)",
## +       cex.lab = 0.8, font.lab = 3)
##
## > ## We have already confessed to having these.  This is just showing off X11
## > ## color names (and the example (from the postscript manual) is pretty "cute".
## >
## > pie.sales <- c(0.12, 0.3, 0.26, 0.16, 0.04, 0.12)
##
## > names(pie.sales) <- c("Blueberry", "Cherry",
## +               "Apple", "Boston Cream", "Other", "Vanilla Cream")
##
## > pie(pie.sales,
## +     col = c("purple","violetred1","green3","cornsilk","cyan","white"))
```



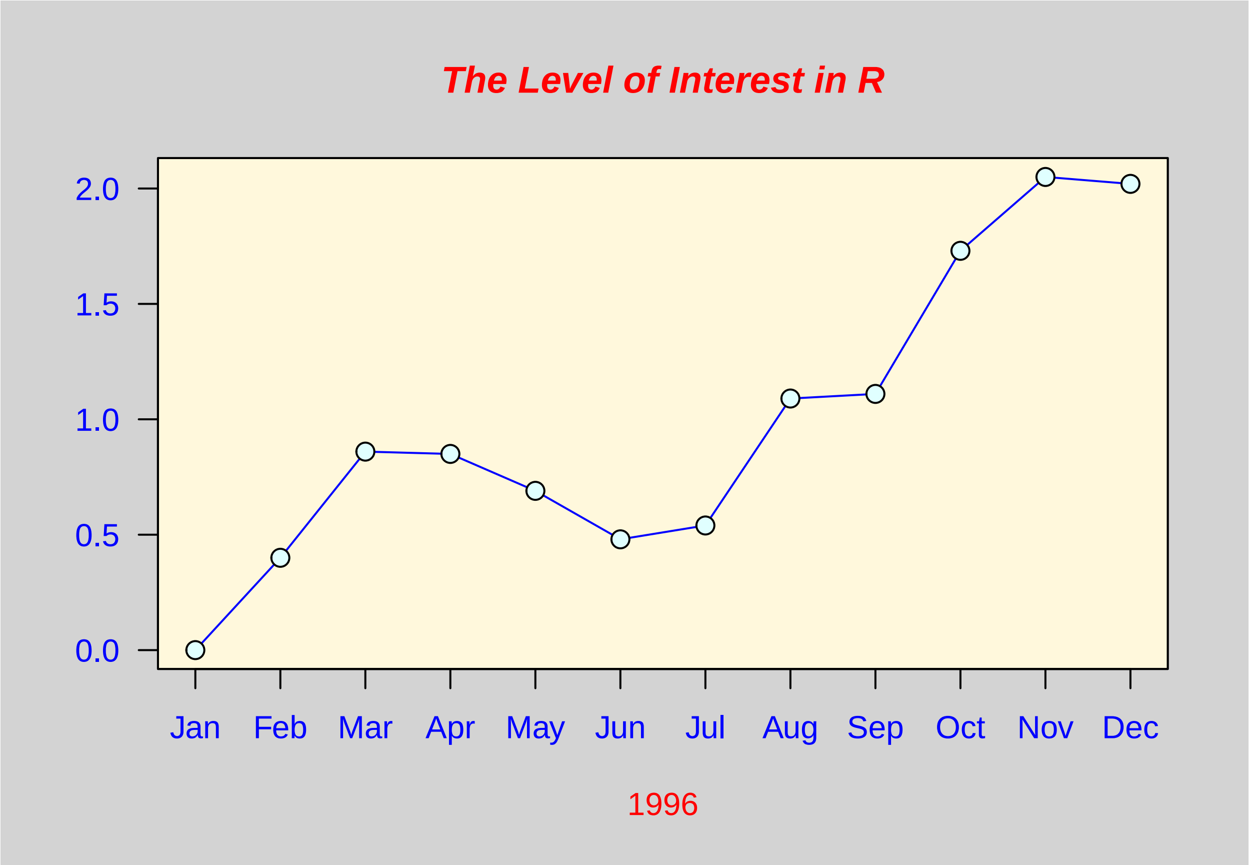
```
##
## > title(main = "January Pie Sales", cex.main = 1.8, font.main = 1)
##
## > title(xlab = "(Don't try this at home kids)", cex.lab = 0.8, font.lab = 3)
##
## > ## Boxplots:  I couldn't resist the capability for filling the "box".
## > ## The use of color seems like a useful addition, it focuses attention
## > ## on the central bulk of the data.
## >
## > par(bg="cornsilk")
##
## > n <- 10
##
## > g <- gl(n, 100, n*100)
##
## > x <- rnorm(n*100) + sqrt(as.numeric(g))
##
## > boxplot(split(x,g), col="lavender", notch=TRUE)
```



```
##
## > title(main="Notched Boxplots", xlab="Group", font.main=4, font.lab=1)
##
## > ## An example showing how to fill between curves.
## >
## > par(bg="white")
##
## > n <- 100
##
## > x <- c(0,cumsum(rnorm(n)))
##
## > y <- c(0,cumsum(rnorm(n)))
##
## > xx <- c(0:n, n:0)
##
## > yy <- c(x, rev(y))
##
## > plot(xx, yy, type="n", xlab="Time", ylab="Distance")
```

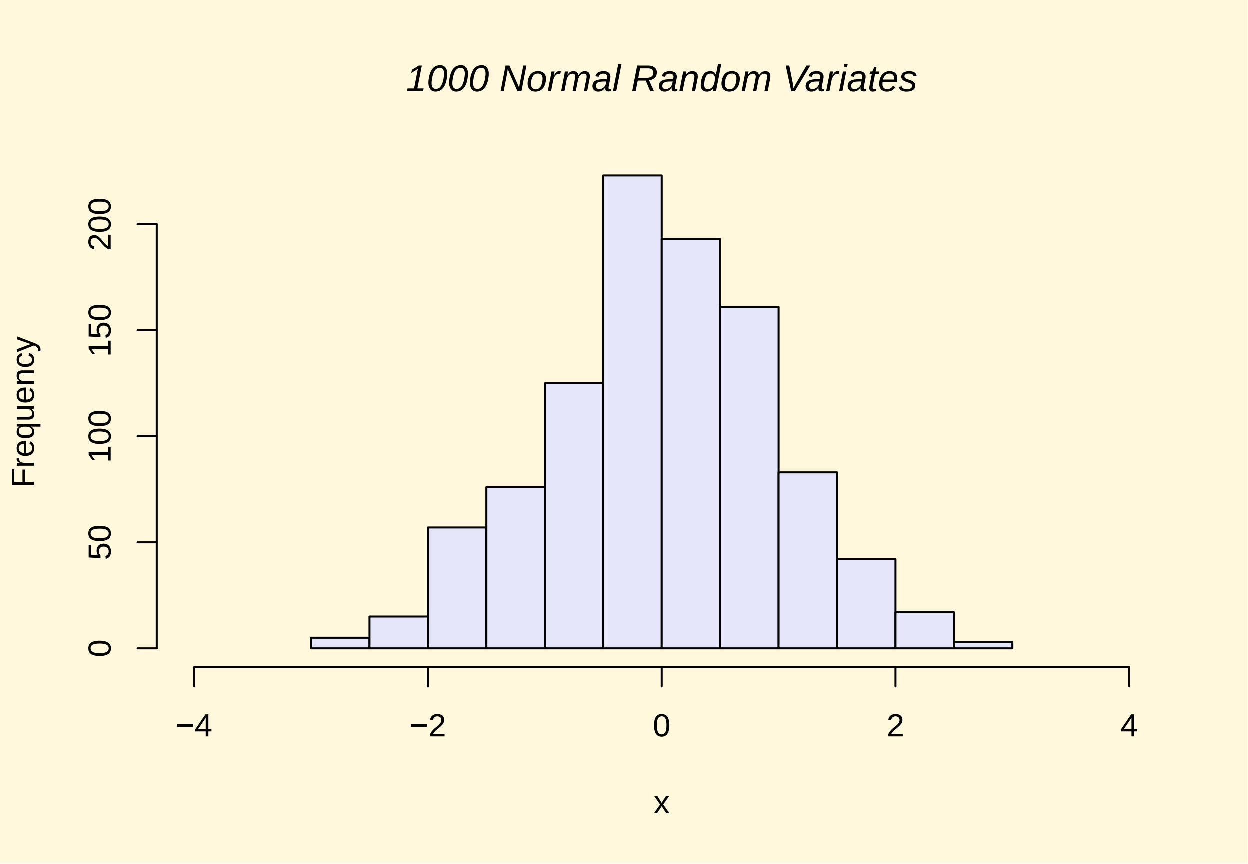


```
##
## > polygon(xx, yy, col="gray")
##
## > title("Distance Between Brownian Motions")
##
## > ## Colored plot margins, axis labels and titles.    You do need to be
## > ## careful with these kinds of effects.    It's easy to go completely
## > ## over the top and you can end up with your lunch all over the keyboard.
## > ## On the other hand, my market research clients love it.
## >
## > x <- c(0.00, 0.40, 0.86, 0.85, 0.69, 0.48, 0.54, 1.09, 1.11, 1.73, 2.05, 2.02)
##
## > par(bg="lightgray")
##
## > plot(x, type="n", axes=FALSE, ann=FALSE)
```

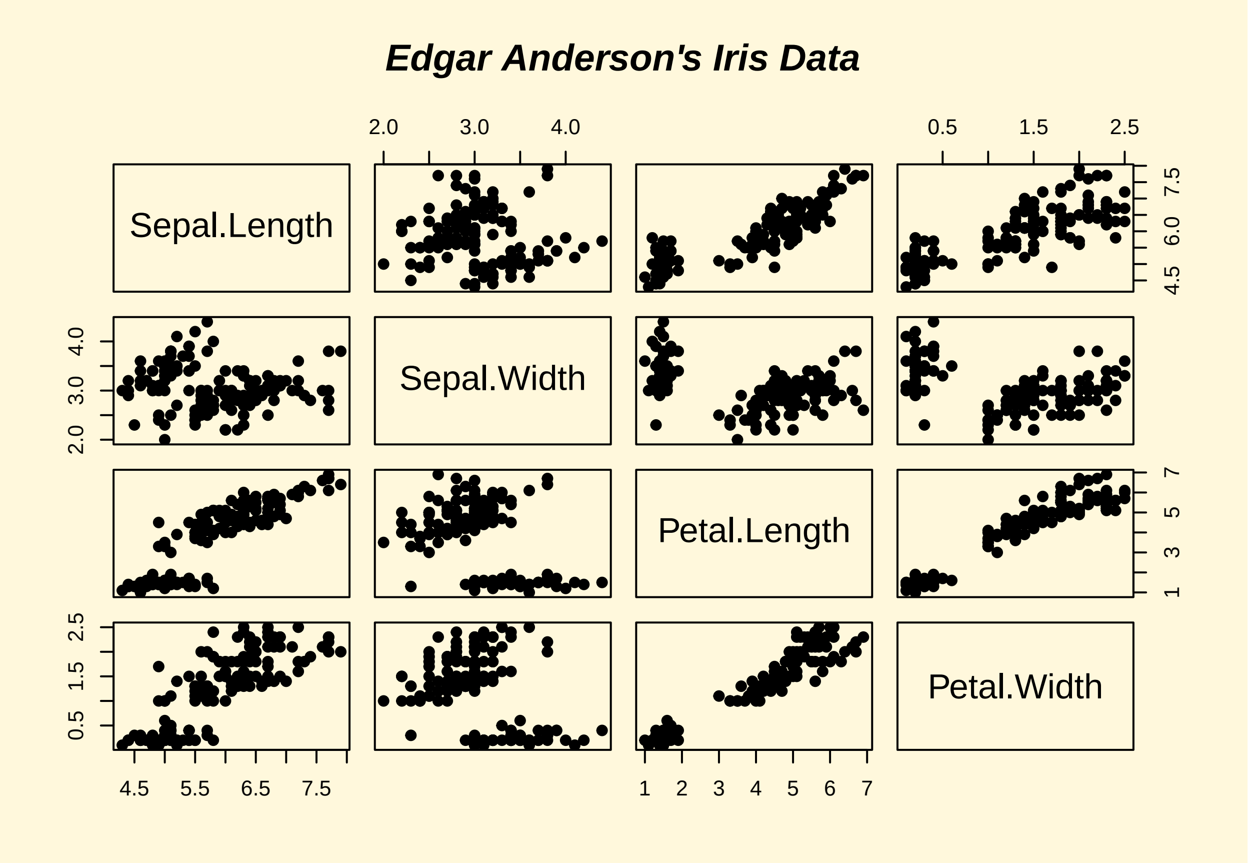


```
##
## > usr <- par("usr")
##
## > rect(usr[1], usr[3], usr[2], usr[4], col="cornsilk", border="black")
##
## > lines(x, col="blue")
##
## > points(x, pch=21, bg="lightcyan", cex=1.25)
##
## > axis(2, col.axis="blue", las=1)
##
## > axis(1, at=1:12, lab=month.abb, col.axis="blue")
##
## > box()
##
## > title(main= "The Level of Interest in R", font.main=4, col.main="red")
##
## > title(xlab= "1996", col.lab="red")
##
## > ## A filled histogram, showing how to change the font used for the
## > ## main title without changing the other annotation.
## >
## > par(bg="cornsilk")
##
## > x <- rnorm(1000)
##
```

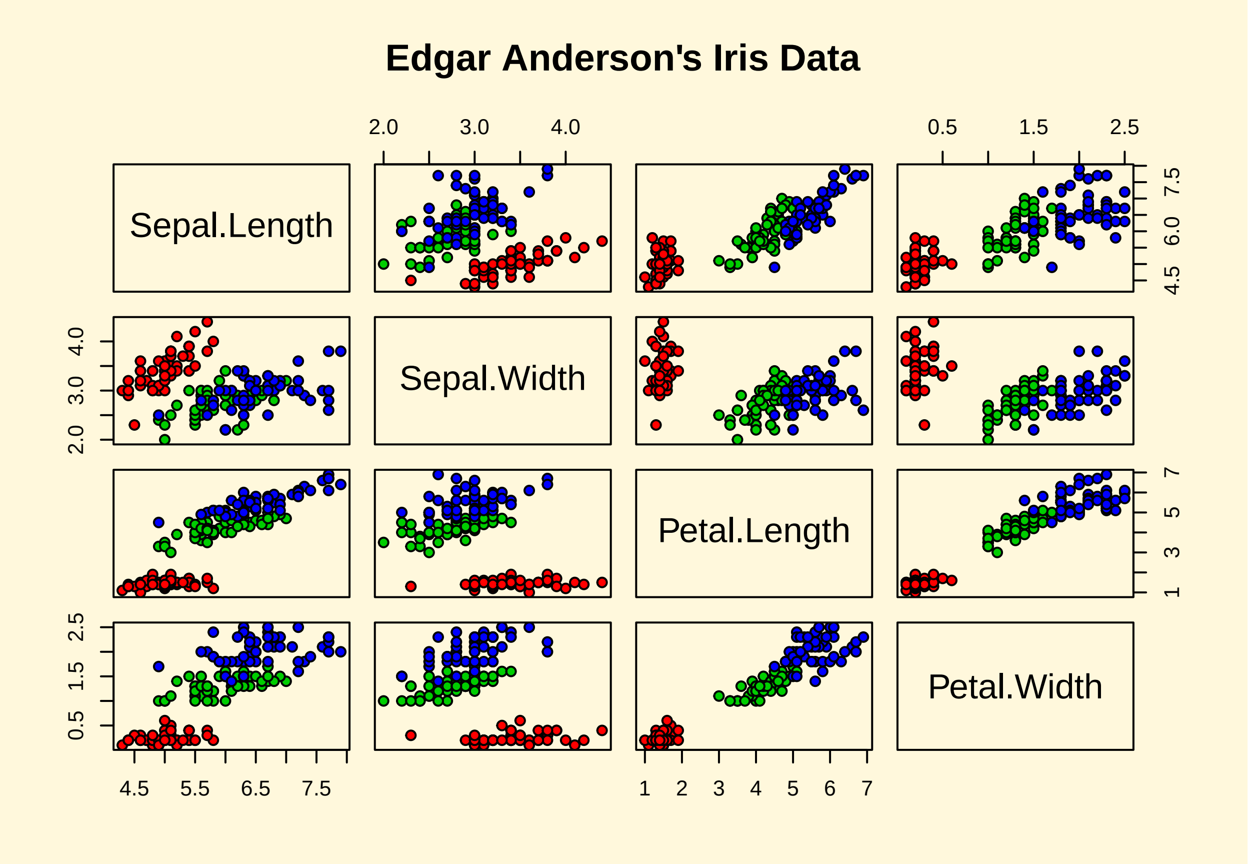
```
## > hist(x, xlim=range(-4, 4, x), col="lavender", main="")
```



```
##
## > title(main="1000 Normal Random Variates", font.main=3)
##
## > ## A scatterplot matrix
## > ## The good old Iris data (yet again)
## >
## > pairs(iris[1:4], main="Edgar Anderson's Iris Data", font.main=4, pch=19)
```

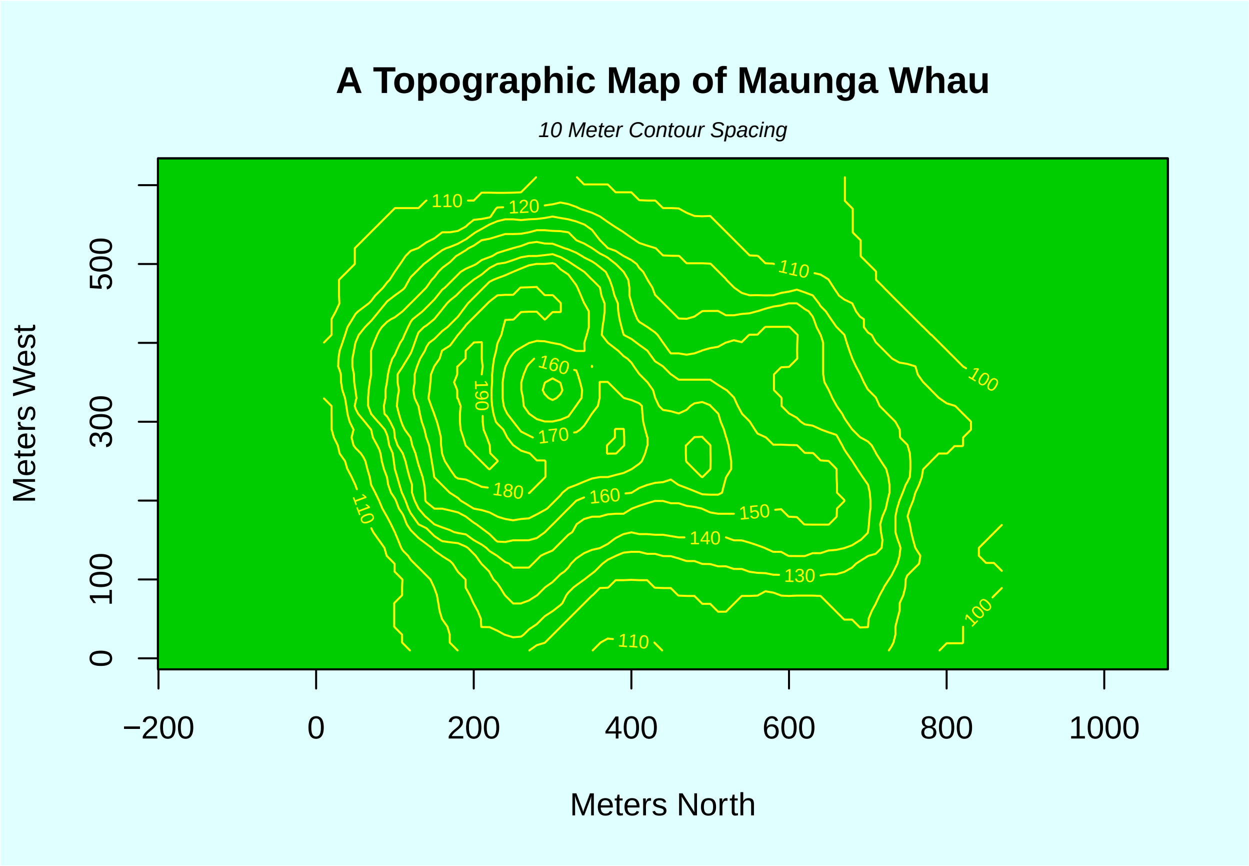


```
##
## > pairs(iris[1:4], main="Edgar Anderson's Iris Data", pch=21,
## +       bg = c("red", "green3", "blue")[unclass(iris$Species)])
```



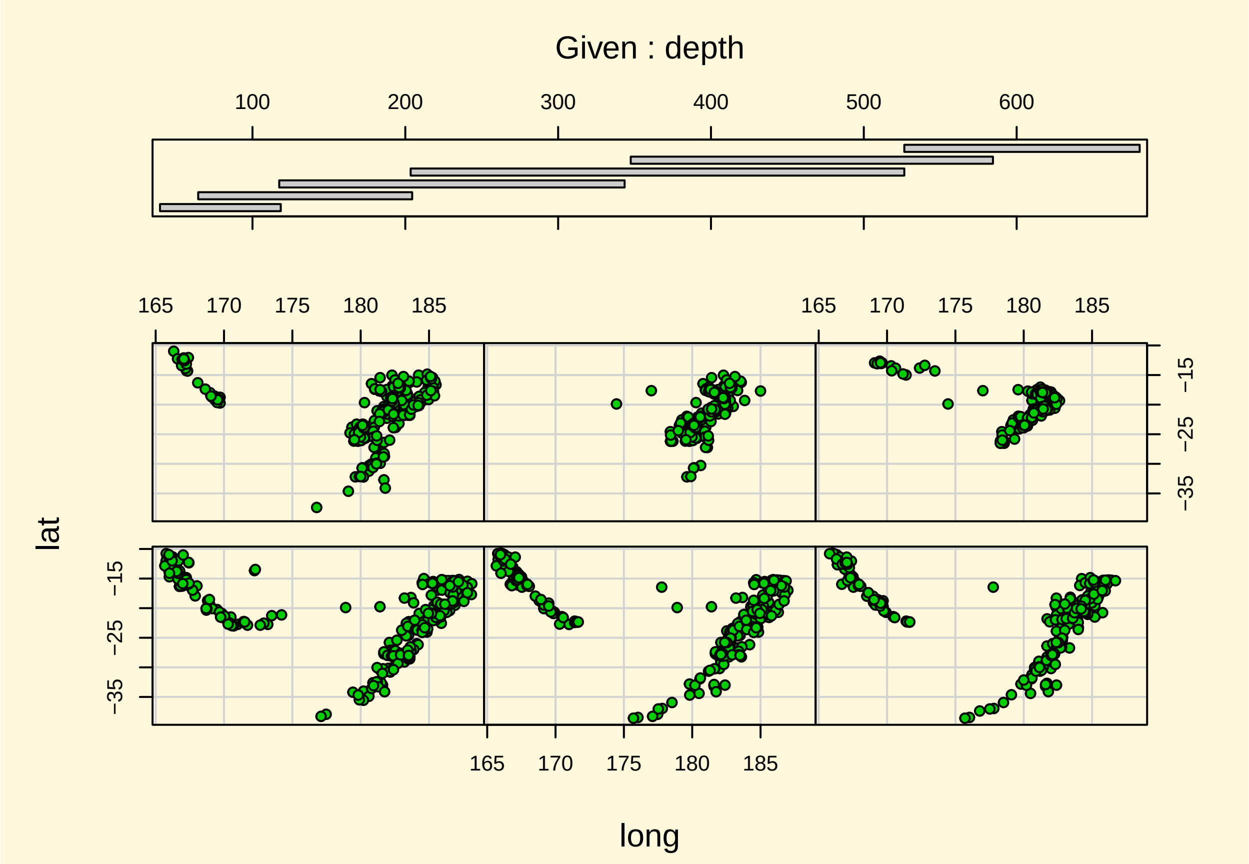
```
##
## > ## Contour plotting
## > ## This produces a topographic map of one of Auckland's many volcanic "peaks".
## >
## > x <- 10*1:nrow(volcano)
##
## > y <- 10*1:ncol(volcano)
##
## > lev <- pretty(range(volcano), 10)
##
## > par(bg = "lightcyan")
##
## > pin <- par("pin")
##
## > xdelta <- diff(range(x))
##
## > ydelta <- diff(range(y))
##
## > xscale <- pin[1]/xdelta
##
## > yscale <- pin[2]/ydelta
##
## > scale <- min(xscale, yscale)
##
## > xadd <- 0.5*(pin[1]/scale - xdelta)
##
```

```
## > yadd <- 0.5*(pin[2]/scale - ydelta)
##
## > plot(numeric(0), numeric(0),
## +      xlim = range(x)+c(-1,1)*xadd, ylim = range(y)+c(-1,1)*yadd,
## +      type = "n", ann = FALSE)
```



```
##
## > usr <- par("usr")
##
## > rect(usr[1], usr[3], usr[2], usr[4], col="green3")
##
## > contour(x, y, volcano, levels = lev, col="yellow", lty="solid", add=TRUE)
##
## > box()
##
## > title("A Topographic Map of Maunga Whau", font= 4)
##
## > title(xlab = "Meters North", ylab = "Meters West", font= 3)
##
## > mtext("10 Meter Contour Spacing", side=3, line=0.35, outer=FALSE,
## +       at = mean(par("usr")[1:2]), cex=0.7, font=3)
##
## > ## Conditioning plots
## >
## > par(bg="cornsilk")
```

```
##
## > coplot(lat ~ long | depth, data = quakes, pch = 21, bg = "green3")
```

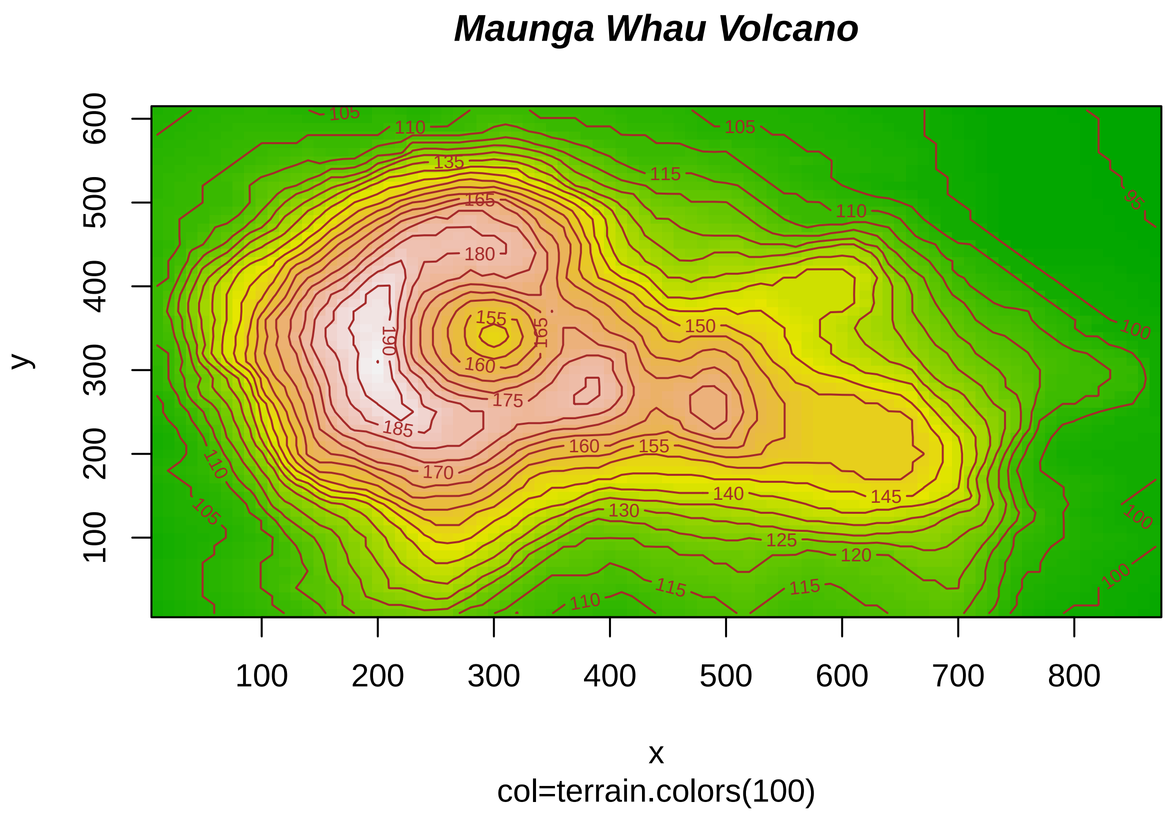


```
##
## > par(opar)
```

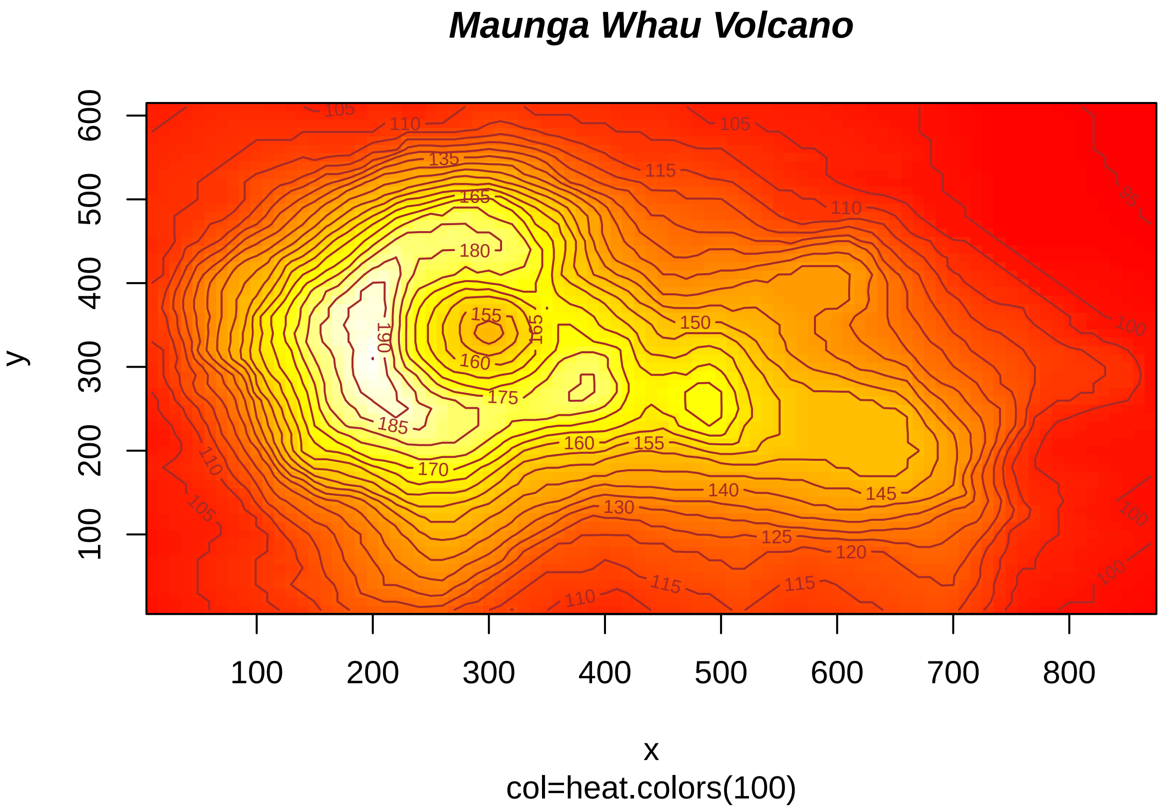
```
demo(image)
```

```
##
##
##  demo(image)
##  ---- ~~~~~
##
## > #  Copyright (C) 1997-2009 The R Core Team
## >
## > require(datasets)
##
## > require(grDevices); require(graphics)
##
## > x <- 10*(1:nrow(volcano)); x.at <- seq(100, 800, by=100)
##
## > y <- 10*(1:ncol(volcano)); y.at <- seq(100, 600, by=100)
##
## >                     # Using Terrain Colors
```

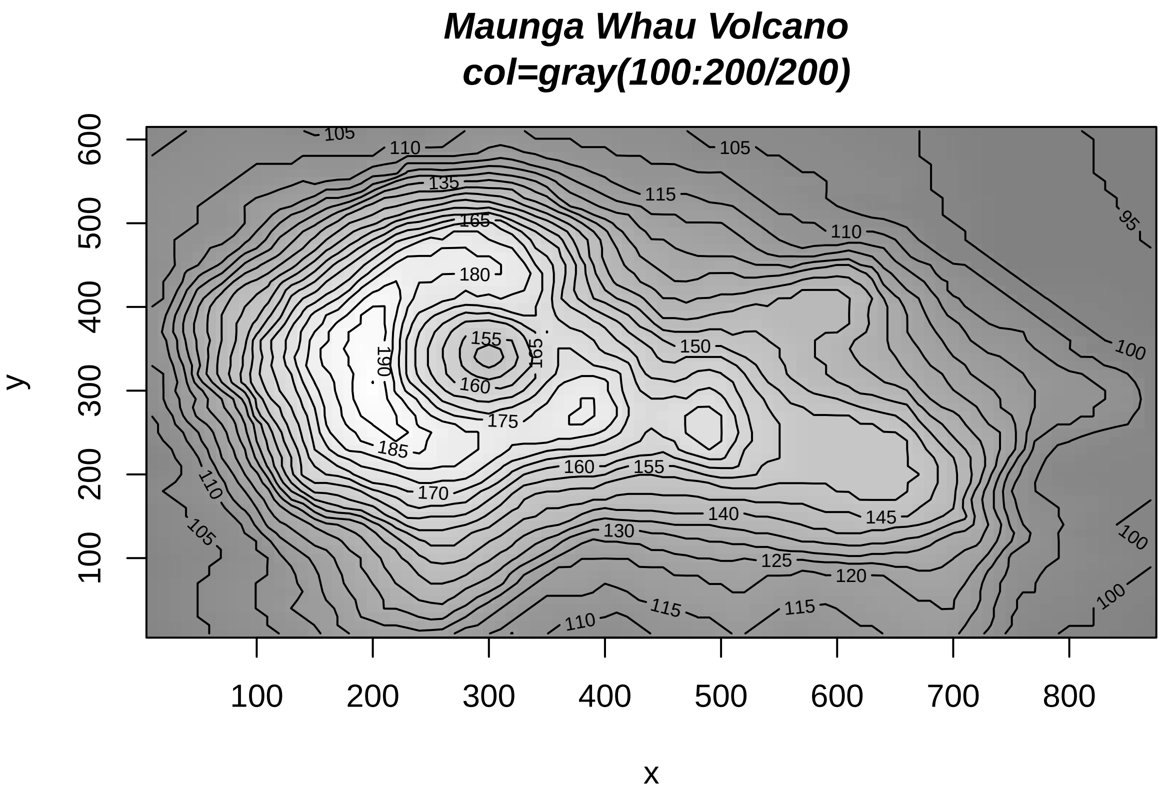
```
## >
## > image(x, y, volcano, col=terrain.colors(100),axes=FALSE)
```



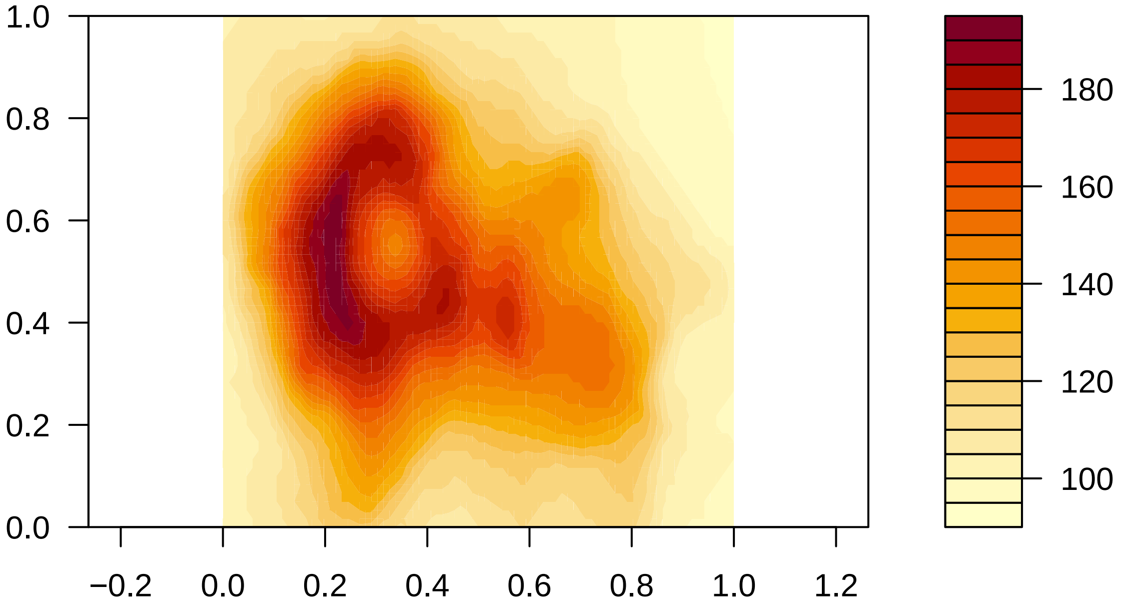
```
##
## > contour(x, y, volcano, levels=seq(90, 200, by=5), add=TRUE, col="brown")
##
## > axis(1, at=x.at)
##
## > axis(2, at=y.at)
##
## > box()
##
## > title(main="Maunga Whau Volcano", sub = "col=terrain.colors(100)", font.main=4)
##
## >                       # Using Heat Colors
## >
## > image(x, y, volcano, col=heat.colors(100), axes=FALSE)
```



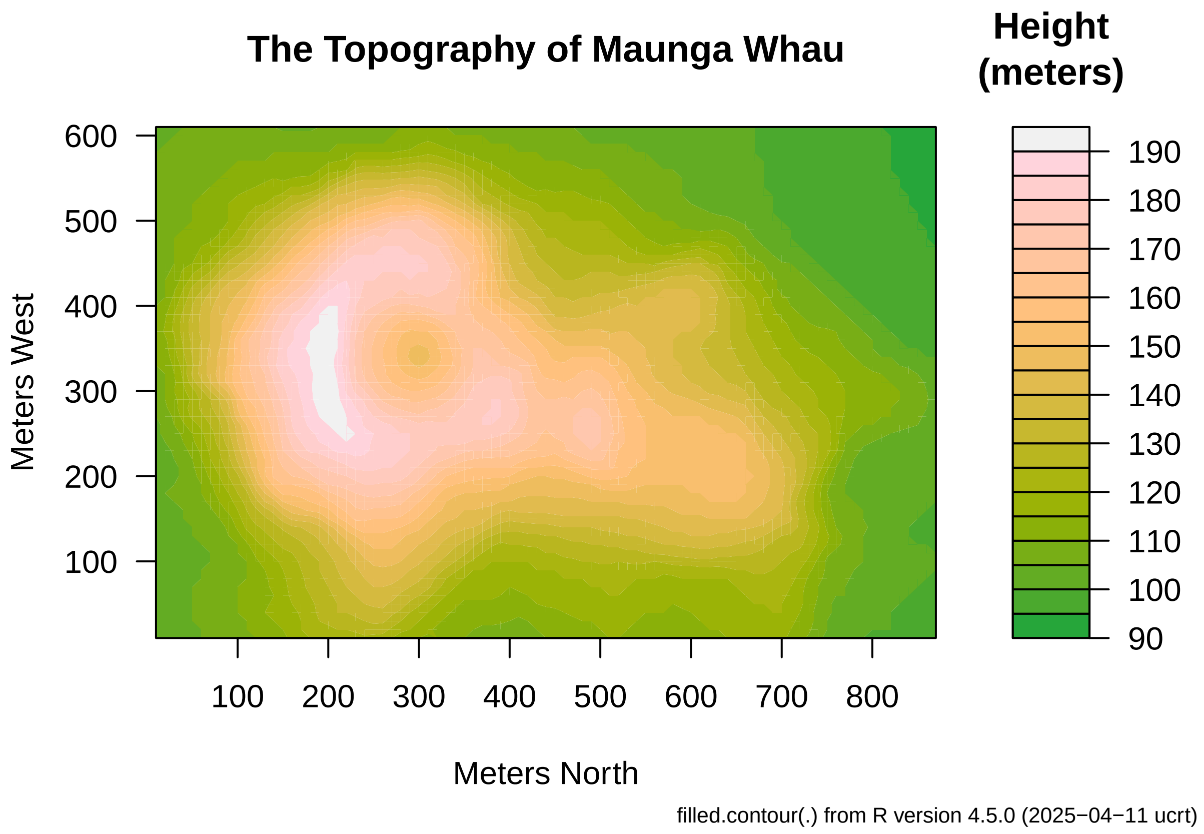
```
##
## > contour(x, y, volcano, levels=seq(90, 200, by=5), add=TRUE, col="brown")
##
## > axis(1, at=x.at)
##
## > axis(2, at=y.at)
##
## > box()
##
## > title(main="Maunga Whau Volcano", sub = "col=heat.colors(100)", font.main=4)
##
## >                     # Using Gray Scale
## >
## > image(x, y, volcano, col=gray(100:200/200), axes=FALSE)
```



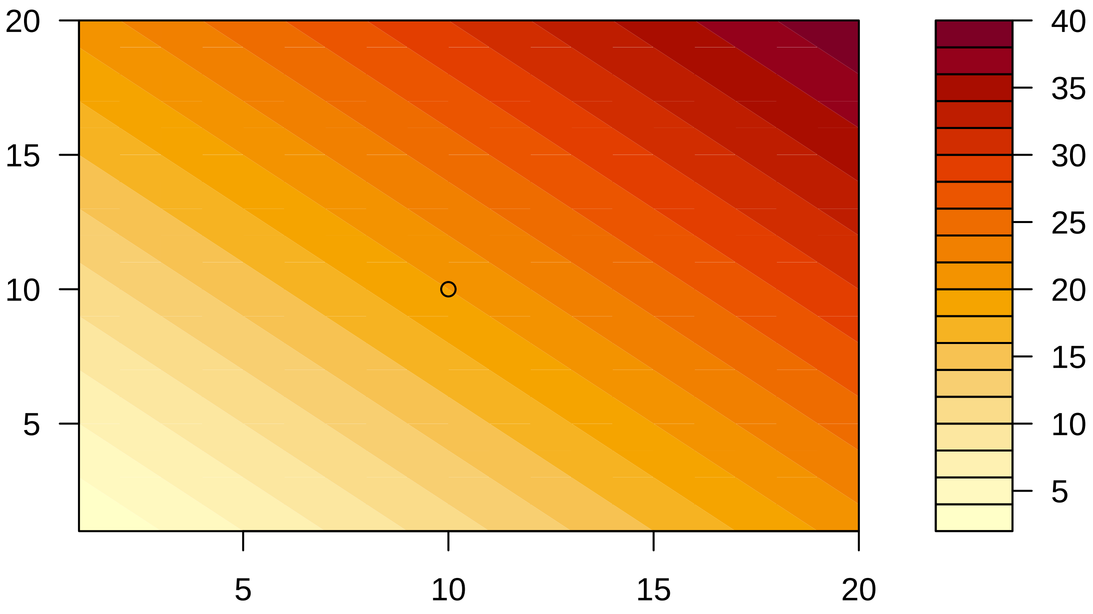
```
##
## > contour(x, y, volcano, levels=seq(90, 200, by=5), add=TRUE, col="black")
##
## > axis(1, at=x.at)
##
## > axis(2, at=y.at)
##
## > box()
##
## > title(main="Maunga Whau Volcano \n col=gray(100:200/200)", font.main=4)
##
## > ## Filled Contours are even nicer sometimes :
## > example(filled.contour)
##
## flld.c> require("grDevices") # for colours
##
## flld.c> filled.contour(volcano, asp = 1) # simple
```



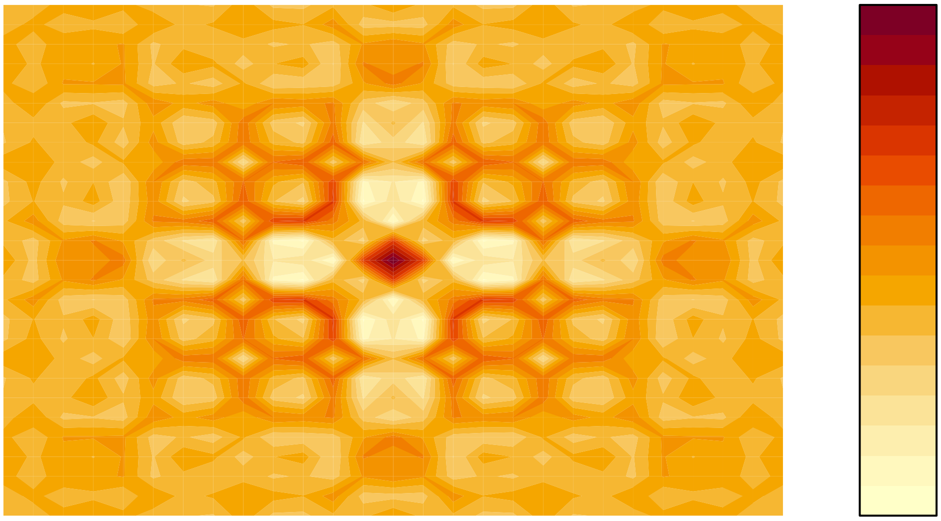
```
##
## flld.c> x <- 10*1:nrow(volcano)
##
## flld.c> y <- 10*1:ncol(volcano)
##
## flld.c> filled.contour(x, y, volcano,
## flld.c+     color.palette = function(n) hcl.colors(n, "terrain"),
## flld.c+     plot.title = title(main = "The Topography of Maunga Whau",
## flld.c+     xlab = "Meters North", ylab = "Meters West"),
## flld.c+     plot.axes = { axis(1, seq(100, 800, by = 100))
## flld.c+                   axis(2, seq(100, 600, by = 100)) },
## flld.c+     key.title = title(main = "Height\n(meters)"),
## flld.c+     key.axes = axis(4, seq(90, 190, by = 10)))  # maybe also asp = 1
```



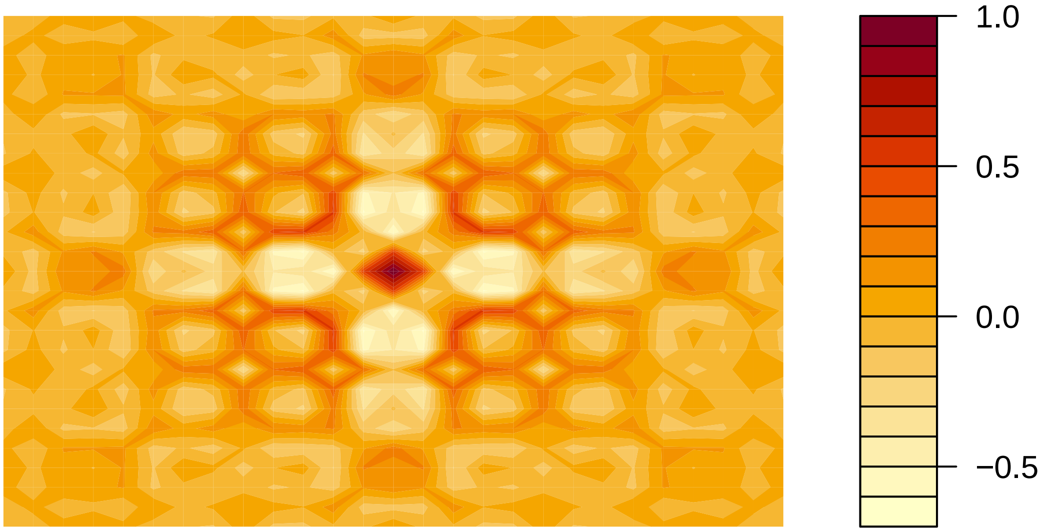
```
##
## flld.c> mtext(paste("filled.contour(.) from", R.version.string),
## flld.c+       side = 1, line = 4, adj = 1, cex = .66)
##
## flld.c> # Annotating a filled contour plot
## flld.c> a <- expand.grid(1:20, 1:20)
##
## flld.c> b <- matrix(a[,1] + a[,2], 20)
##
## flld.c> filled.contour(x = 1:20, y = 1:20, z = b,
## flld.c+                plot.axes = { axis(1); axis(2); points(10, 10) })
```



```
##
## flld.c> ## Persian Rug Art:
## flld.c> x <- y <- seq(-4*pi, 4*pi, length.out = 27)
##
## flld.c> r <- sqrt(outer(x^2, y^2, `+`))
##
## flld.c> ## "minimal"
## flld.c> filled.contour(cos(r^2)*exp(-r/(2*pi)), axes = FALSE, key.border=NA)
```



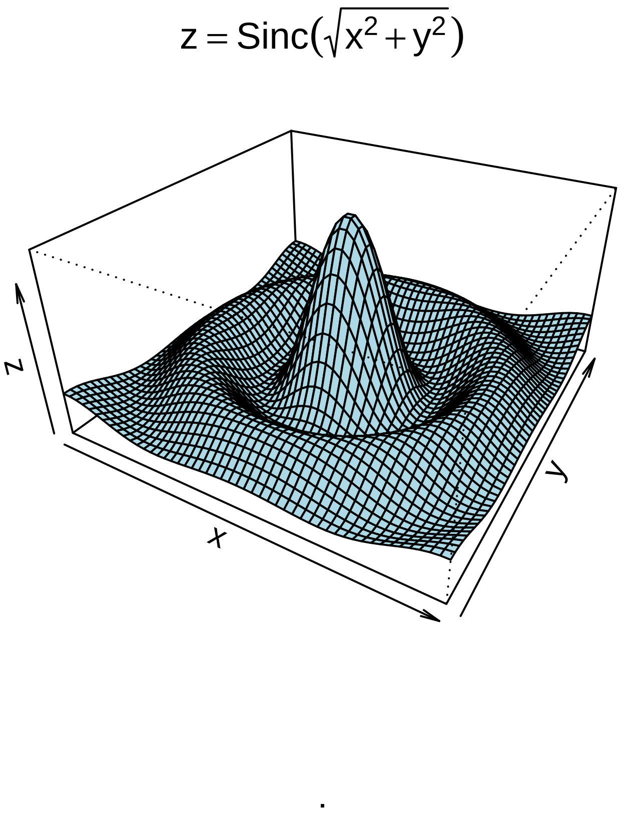
```
##
## flld.c> ## rather, the key *should* be labeled (but axes still not):
## flld.c> filled.contour(cos(r^2)*exp(-r/(2*pi)), frame.plot = FALSE,
## flld.c+                plot.axes = {})
```



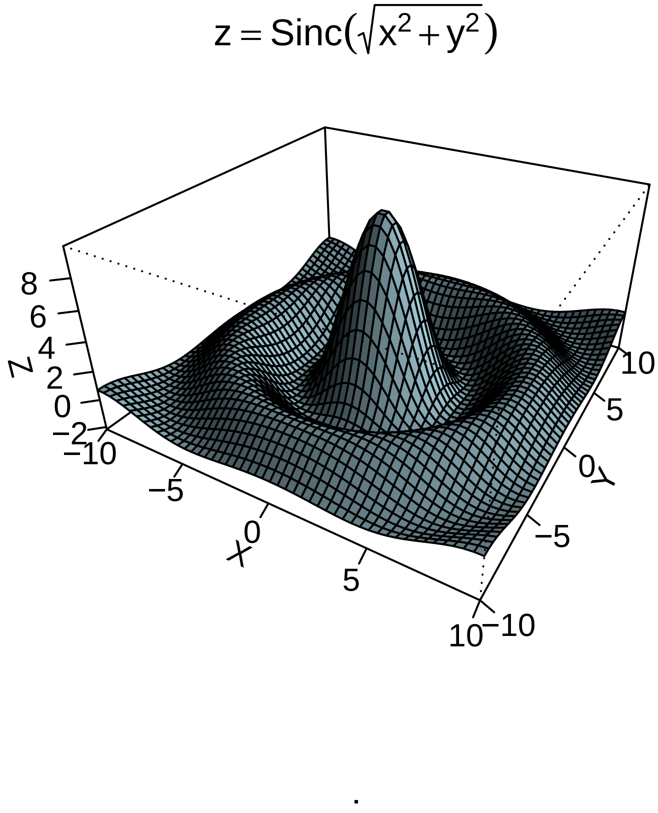
```
demo(persp)
```

```
##
##
##  demo(persp)
##  ---- ~~~~~
##
## > ### Demos for  persp()  plots   -- things not in  example(persp)
## > ### -------------------------
## >
## > require(datasets)
##
## > require(grDevices); require(graphics)
##
## > ## (1) The Obligatory Mathematical surface.
## > ##     Rotated sinc function.
## >
## > x <- seq(-10, 10, length.out = 50)
##
## > y <- x
##
## > rotsinc <- function(x,y)
## + {
## +     sinc <- function(x) { y <- sin(x)/x ; y[is.na(y)] <- 1; y }
## +     10 * sinc( sqrt(x^2+y^2) )
```

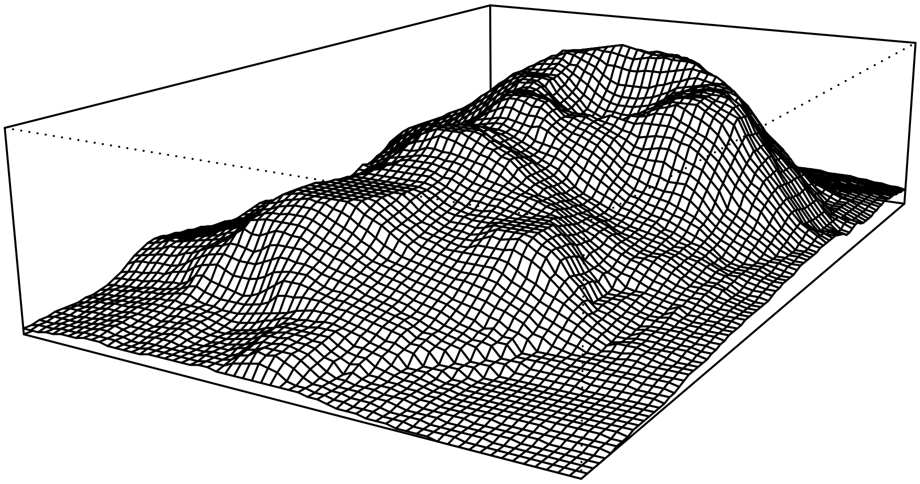
```
## + }
##
## > sinc.exp <- expression(z == Sinc(sqrt(x^2 + y^2)))
##
## > z <- outer(x, y, rotsinc)
##
## > oldpar <- par(bg = "white")
##
## > persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue")
```



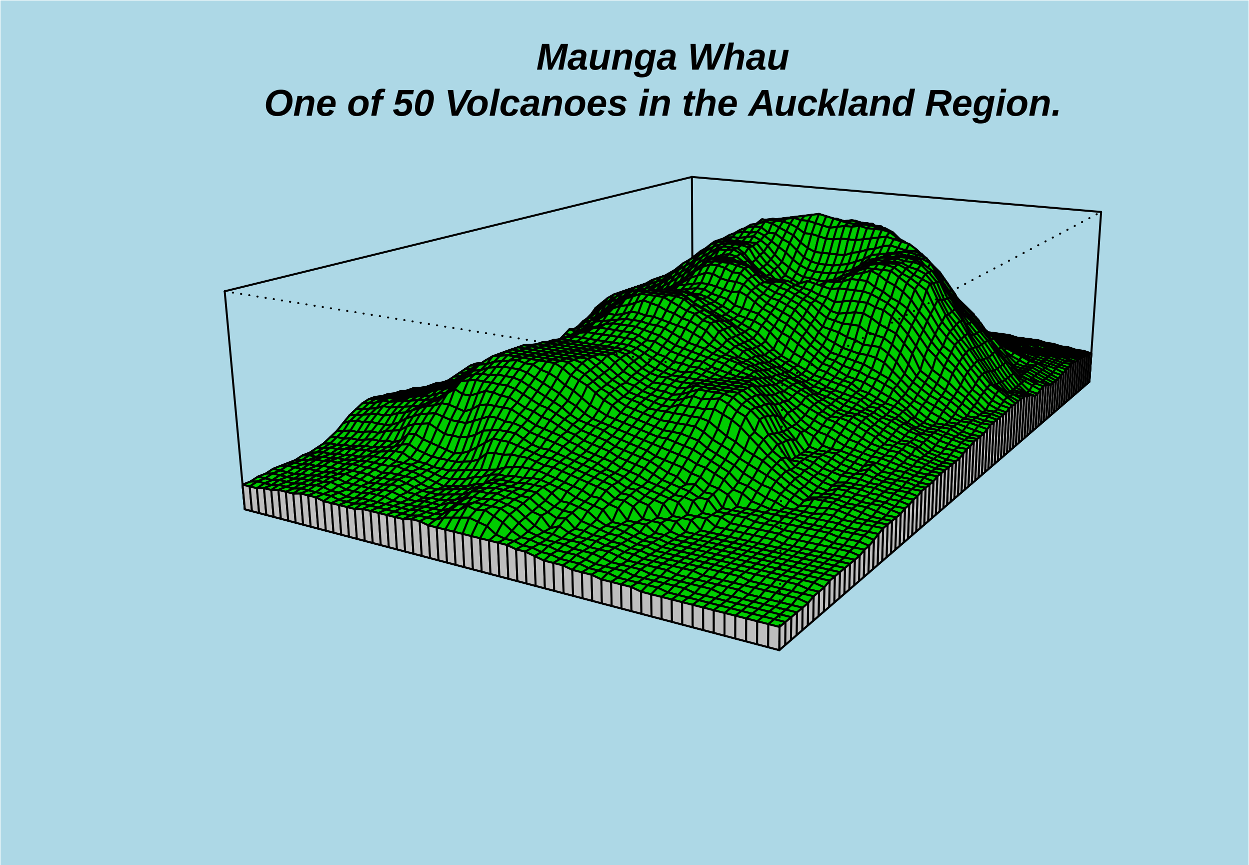
```
##
## > title(sub=".")## work around persp+plotmath bug
##
## > title(main = sinc.exp)
##
## > persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "lightblue",
## +       ltheta = 120, shade = 0.75, ticktype = "detailed",
## +       xlab = "X", ylab = "Y", zlab = "Z")
```



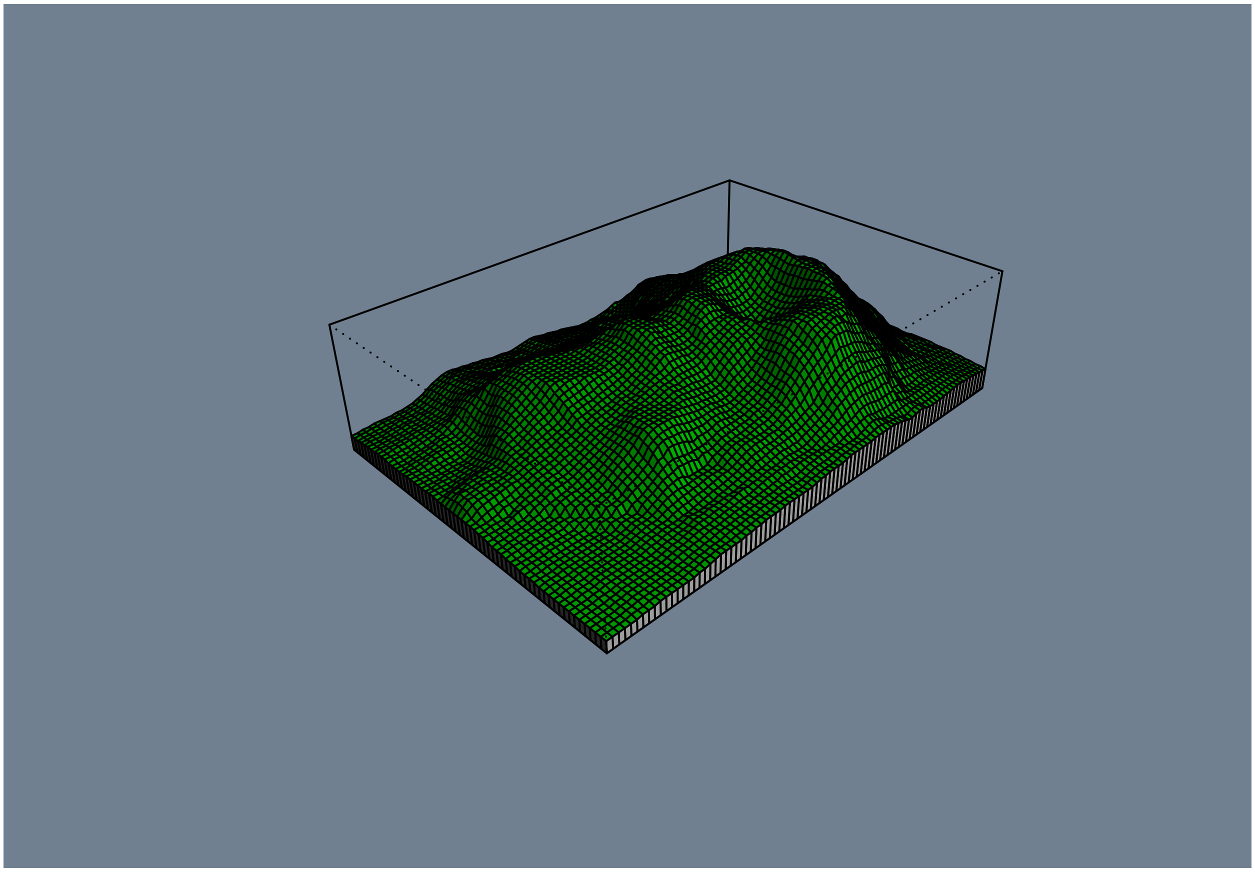
```
##
## > title(sub=".")## work around persp+plotmath bug
##
## > title(main = sinc.exp)
##
## > ## (2) Visualizing a simple DEM model
## >
## > z <- 2 * volcano        # Exaggerate the relief
##
## > x <- 10 * (1:nrow(z))   # 10 meter spacing (S to N)
##
## > y <- 10 * (1:ncol(z))   # 10 meter spacing (E to W)
##
## > persp(x, y, z, theta = 120, phi = 15, scale = FALSE, axes = FALSE)
```



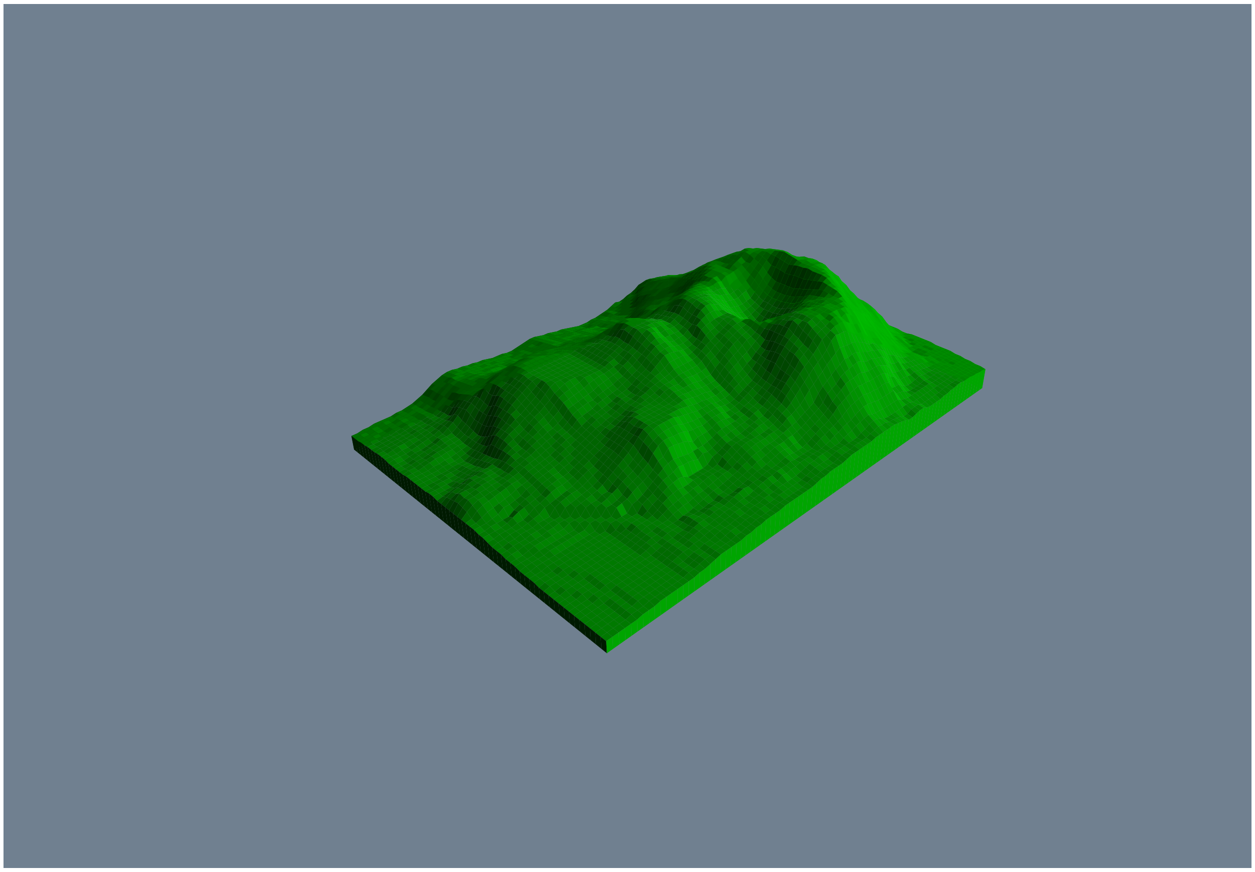
```
##
## > ## (3) Now something more complex
## > ##     We border the surface, to make it more "slice like"
## > ##     and color the top and sides of the surface differently.
## >
## > z0 <- min(z) - 20
##
## > z <- rbind(z0, cbind(z0, z, z0), z0)
##
## > x <- c(min(x) - 1e-10, x, max(x) + 1e-10)
##
## > y <- c(min(y) - 1e-10, y, max(y) + 1e-10)
##
## > fill <- matrix("green3", nrow = nrow(z)-1, ncol = ncol(z)-1)
##
## > fill[ , i2 <- c(1,ncol(fill))] <- "gray"
##
## > fill[i1 <- c(1,nrow(fill)) , ] <- "gray"
##
## > par(bg = "lightblue")
##
## > persp(x, y, z, theta = 120, phi = 15, col = fill, scale = FALSE, axes = FALSE)
```



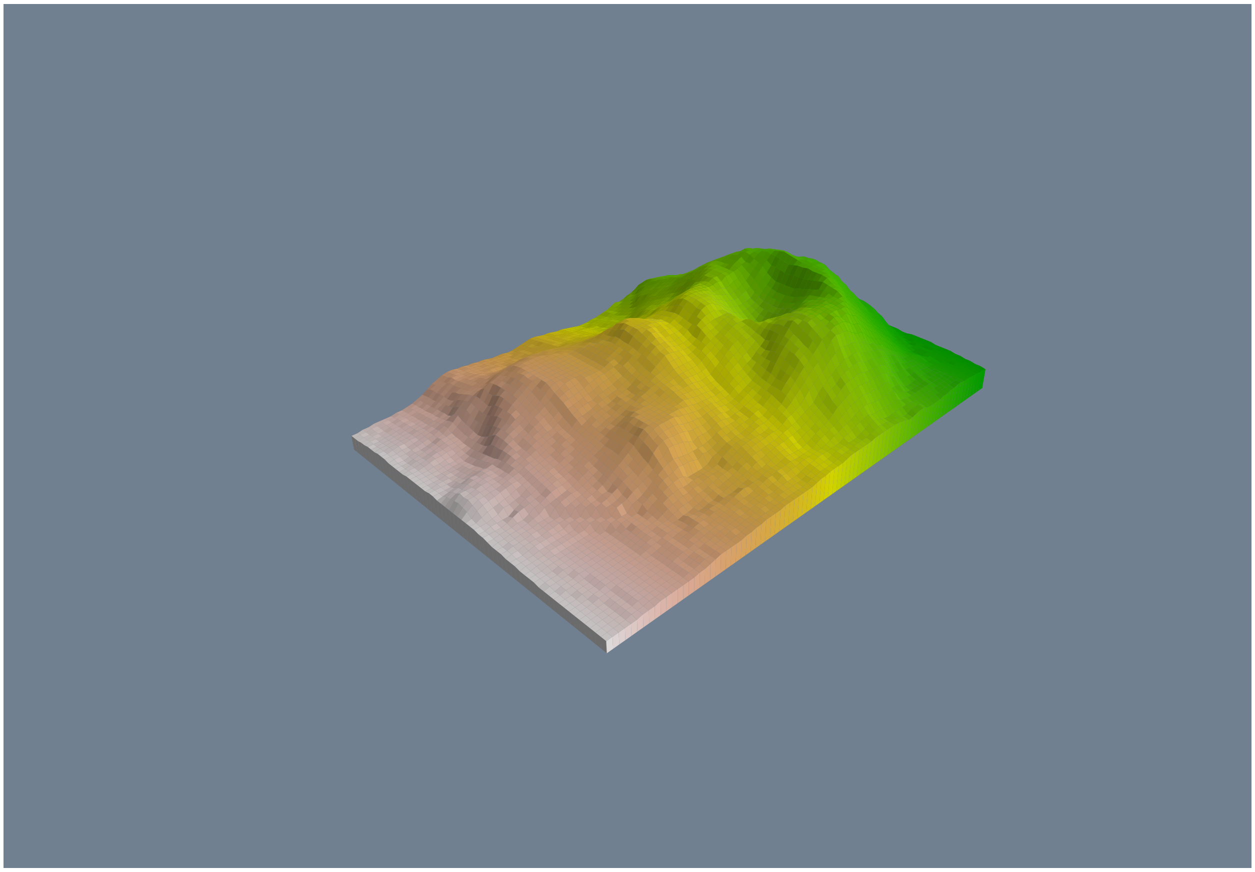
```
##
## > title(main = "Maunga Whau\nOne of 50 Volcanoes in the Auckland Region.",
## +       font.main = 4)
##
## > par(bg = "slategray")
##
## > persp(x, y, z, theta = 135, phi = 30, col = fill, scale = FALSE,
## +       ltheta = -120, lphi = 15, shade = 0.65, axes = FALSE)
```



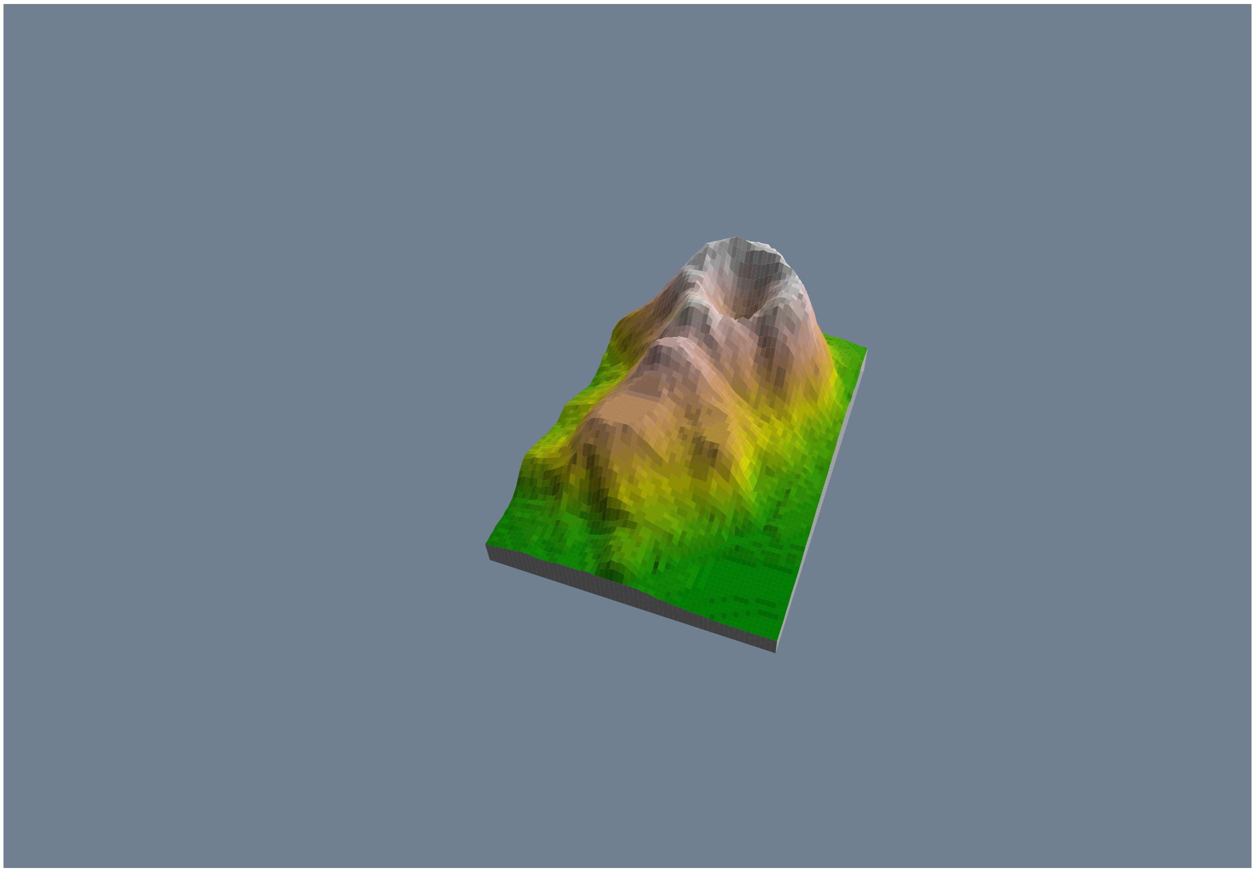
```
##
## > ## Don't draw the grid lines :  border = NA
## > persp(x, y, z, theta = 135, phi = 30, col = "green3", scale = FALSE,
## +       ltheta = -120, shade = 0.75, border = NA, box = FALSE)
```



```
##
## > ## `color gradient in the soil' :
## > fcol <- fill ; fcol[] <- terrain.colors(nrow(fcol))
##
## > persp(x, y, z, theta = 135, phi = 30, col = fcol, scale = FALSE,
## +       ltheta = -120, shade = 0.3, border = NA, box = FALSE)
```



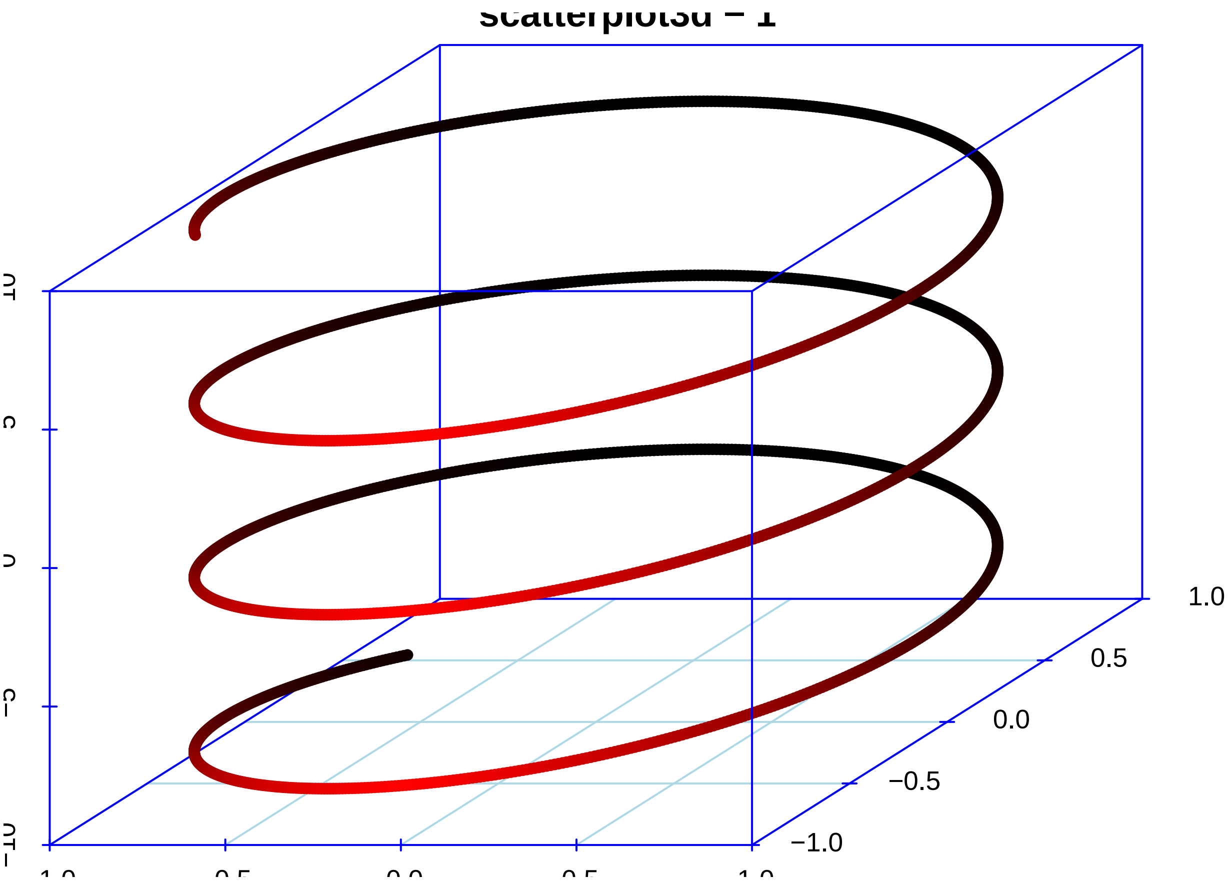
```
##
## > ## `image like' colors on top :
## > fcol <- fill
##
## > zi <- volcano[ -1,-1] + volcano[ -1,-61] +
## +            volcano[-87,-1] + volcano[-87,-61]  ## / 4
##
## > fcol[-i1,-i2] <-
## +     terrain.colors(20)[cut(zi,
## +                            stats::quantile(zi, seq(0,1, length.out = 21)),
## +                            include.lowest = TRUE)]
##
## > persp(x, y, 2*z, theta = 110, phi = 40, col = fcol, scale = FALSE,
## +       ltheta = -120, shade = 0.4, border = NA, box = FALSE)
```



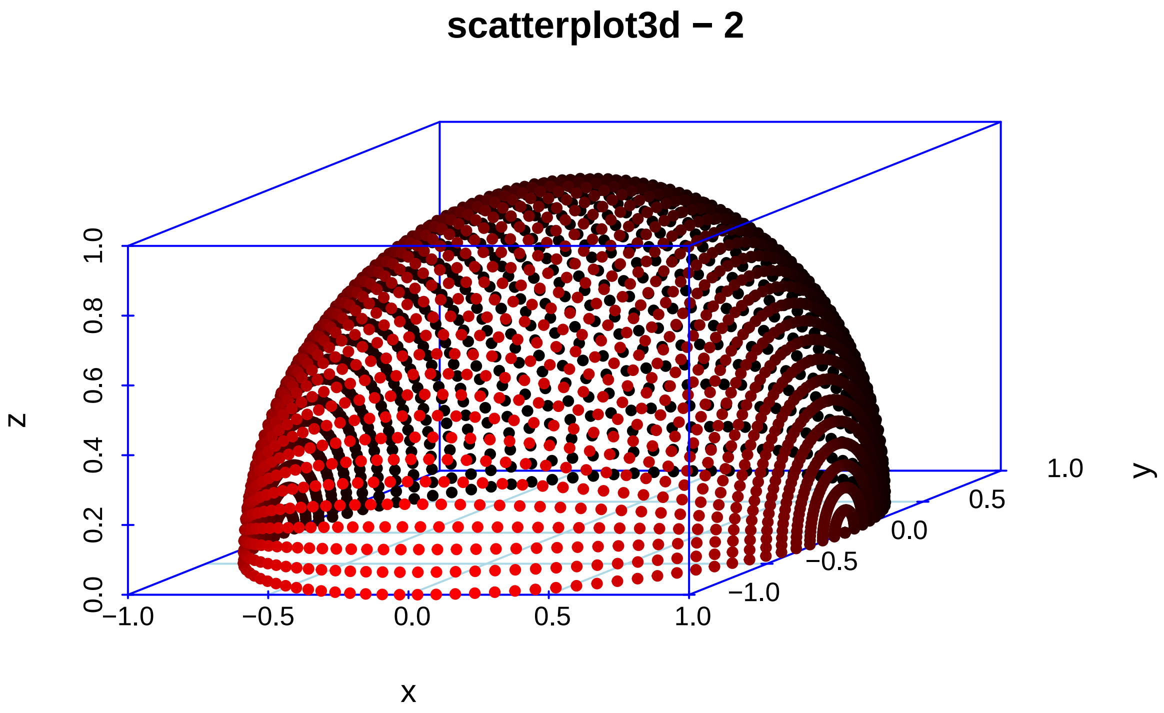
```
##
## > ## reset par():
## > par(oldpar)
```

```
pacman::p_load(scatterplot3d); example(scatterplot3d)
```

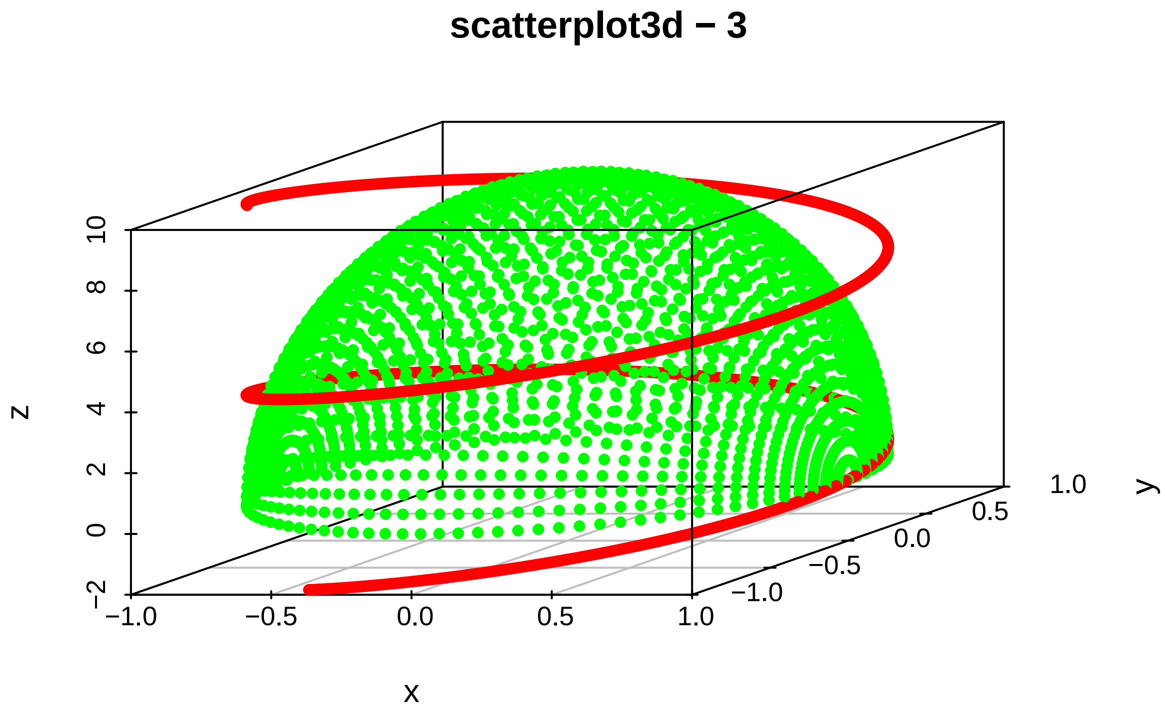
```
##
## scttr3>   ## On some devices not all colors can be displayed.
## scttr3>   ## Try the postscript device or use highlight.3d = FALSE.
## scttr3>
## scttr3>   ## example 1
## scttr3>   z <- seq(-10, 10, 0.01)
##
## scttr3>   x <- cos(z)
##
## scttr3>   y <- sin(z)
##
## scttr3>   scatterplot3d(x, y, z, highlight.3d=TRUE, col.axis="blue",
## scttr3+       col.grid="lightblue", main="scatterplot3d - 1", pch=20, mar=c(0,0,0,0))
```



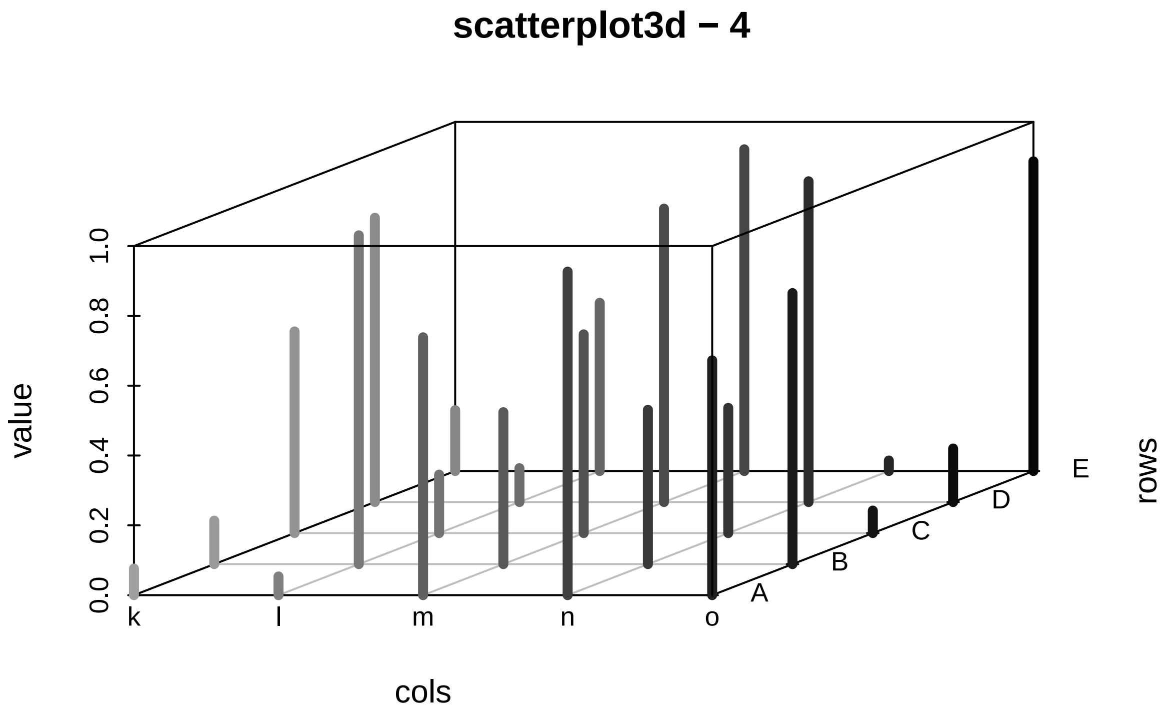
```
##
## scttr3>   ## example 2
## scttr3>   temp <- seq(-pi, 0, length = 50)
##
## scttr3>   x <- c(rep(1, 50) %*% t(cos(temp)))
##
## scttr3>   y <- c(cos(temp) %*% t(sin(temp)))
##
## scttr3>   z <- c(sin(temp) %*% t(sin(temp)))
##
## scttr3>   scatterplot3d(x, y, z, highlight.3d=TRUE,
## scttr3+       col.axis="blue", col.grid="lightblue",
## scttr3+       main="scatterplot3d - 2", pch=20)
```



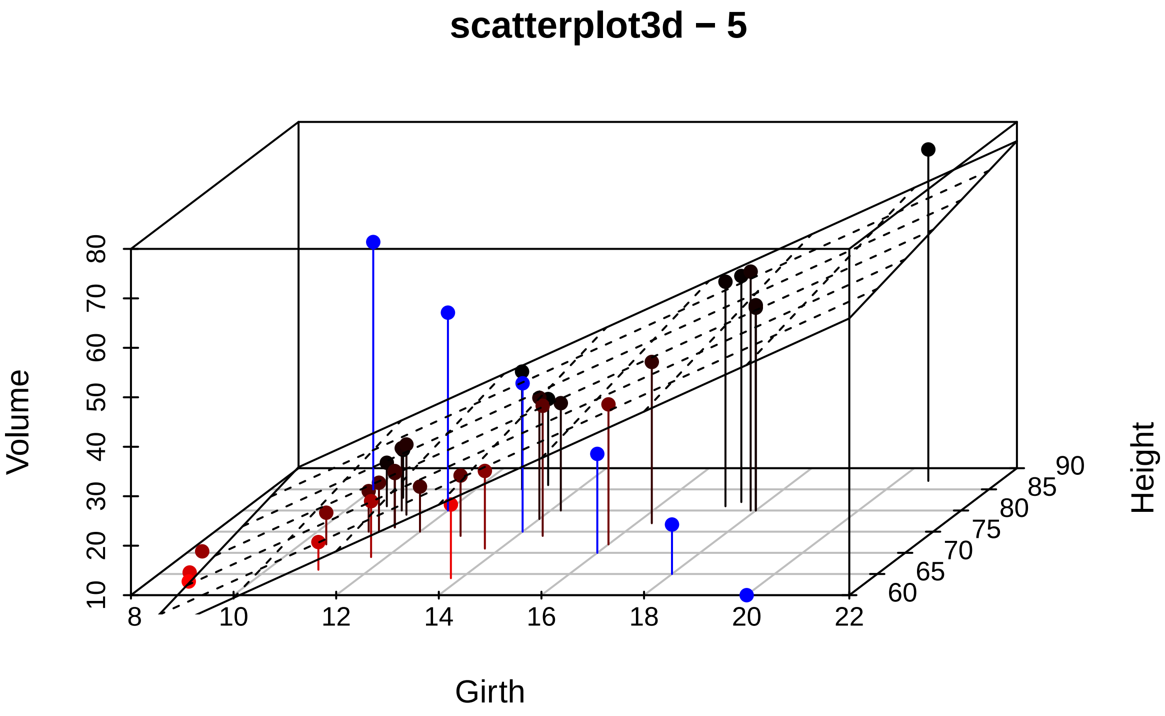
```
##
## scttr3>   ## example 3
## scttr3>   temp <- seq(-pi, 0, length = 50)
##
## scttr3>   x <- c(rep(1, 50) %*% t(cos(temp)))
##
## scttr3>   y <- c(cos(temp) %*% t(sin(temp)))
##
## scttr3>   z <- 10 * c(sin(temp) %*% t(sin(temp)))
##
## scttr3>   color <- rep("green", length(x))
##
## scttr3>   temp <- seq(-10, 10, 0.01)
##
## scttr3>   x <- c(x, cos(temp))
##
## scttr3>   y <- c(y, sin(temp))
##
## scttr3>   z <- c(z, temp)
##
## scttr3>   color <- c(color, rep("red", length(temp)))
##
## scttr3>   scatterplot3d(x, y, z, color, pch=20, zlim=c(-2, 10),
## scttr3+       main="scatterplot3d - 3")
```



```
##
## scttr3>   ## example 4
## scttr3>   my.mat <- matrix(runif(25), nrow=5)
##
## scttr3>   dimnames(my.mat) <- list(LETTERS[1:5], letters[11:15])
##
## scttr3>   my.mat # the matrix we want to plot ...
##            k          l         m          n          o
## A 0.07616657 0.05362723 0.7385162 0.92693264 0.67262017
## B 0.12457756 0.94191012 0.4352650 0.44215389 0.77640473
## C 0.57776599 0.16751451 0.5693004 0.35883170 0.06451534
## D 0.81445001 0.09710834 0.8404474 0.91897886 0.15338307
## E 0.17439519 0.48181829 0.9216480 0.03012741 0.88729232
##
## scttr3>   s3d.dat <- data.frame(cols=as.vector(col(my.mat)),
## scttr3+       rows=as.vector(row(my.mat)),
## scttr3+       value=as.vector(my.mat))
##
## scttr3>   scatterplot3d(s3d.dat, type="h", lwd=5, pch=" ",
## scttr3+       x.ticklabs=colnames(my.mat), y.ticklabs=rownames(my.mat),
## scttr3+       color=grey(25:1/40), main="scatterplot3d - 4")
```

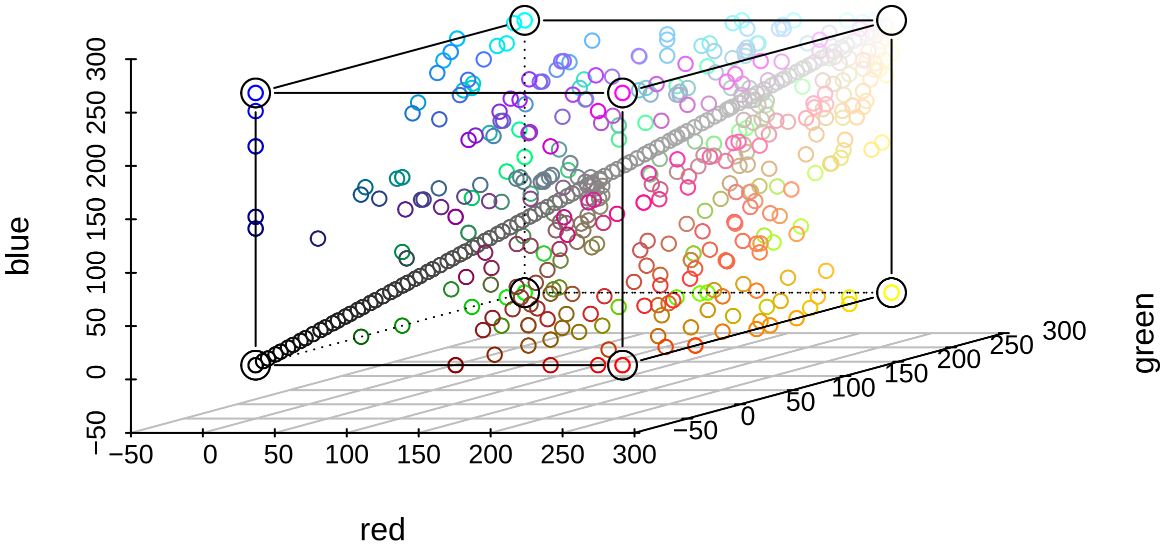


```
##
## scttr3>   ## example 5
## scttr3>   data(trees)
##
## scttr3>   s3d <- scatterplot3d(trees, type="h", highlight.3d=TRUE,
## scttr3+       angle=55, scale.y=0.7, pch=16, main="scatterplot3d - 5")
```

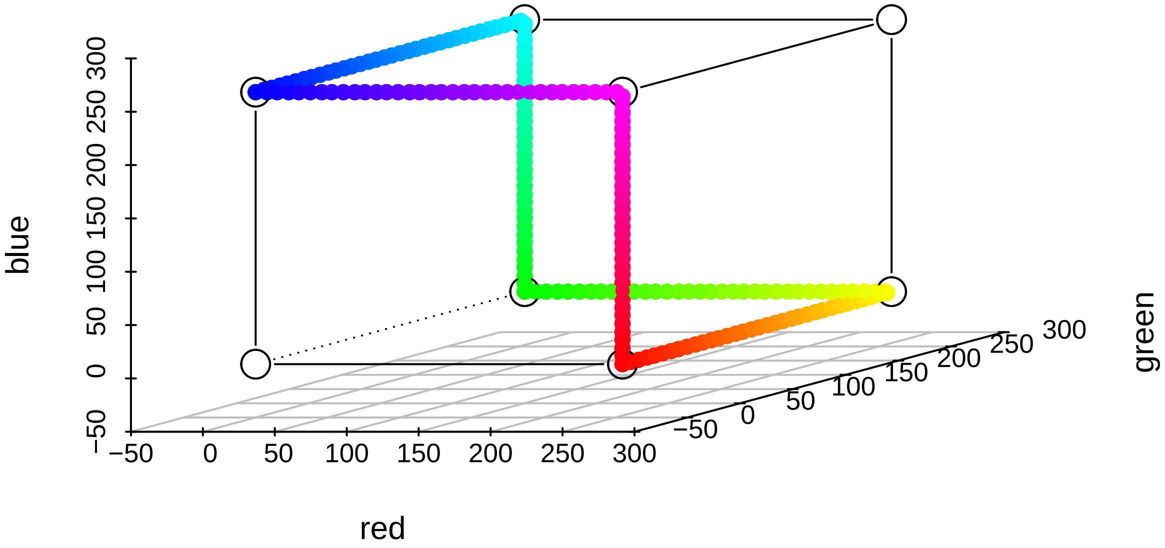


```
##
## scttr3>   # Now adding some points to the "scatterplot3d"
## scttr3>   s3d$points3d(seq(10,20,2), seq(85,60,-5), seq(60,10,-10),
## scttr3+       col="blue", type="h", pch=16)
##
## scttr3>   # Now adding a regression plane to the "scatterplot3d"
## scttr3>   attach(trees)
##
## scttr3>   my.lm <- lm(Volume ~ Girth + Height)
##
## scttr3>   s3d$plane3d(my.lm, lty.box = "solid")
##
## scttr3>   ## example 6; by Martin Maechler
## scttr3>   cubedraw <- function(res3d, min = 0, max = 255, cex = 2, text. = FALSE)
## scttr3+   {
## scttr3+     ## Purpose: Draw nice cube with corners
## scttr3+     cube01 <- rbind(c(0,0,1), 0, c(1,0,0), c(1,1,0), 1, c(0,1,1), # < 6 outer
## scttr3+                     c(1,0,1), c(0,1,0)) # <- "inner": fore- & back-ground
## scttr3+     cub <- min + (max-min)* cube01
## scttr3+     ## visibile corners + lines:
## scttr3+     res3d$points3d(cub[c(1:6,1,7,3,7,5) ,], cex = cex, type = 'b', lty = 1)
## scttr3+     ## hidden corner + lines
## scttr3+     res3d$points3d(cub[c(2,8,4,8,6),     ], cex = cex, type = 'b', lty = 3)
## scttr3+     if(text.)## debug
## scttr3+         text(res3d$xyz.convert(cub), labels=1:nrow(cub), col='tomato', cex=2)
## scttr3+   }
```

```
##
## scttr3>   ## 6 a) The named colors in R, i.e. colors()
## scttr3>   cc <- colors()
##
## scttr3>   crgb <- t(col2rgb(cc))
##
## scttr3>   par(xpd = TRUE)
##
## scttr3>   rr <- scatterplot3d(crgb, color = cc, box = FALSE, angle = 24,
## scttr3+       xlim = c(-50, 300), ylim = c(-50, 300), zlim = c(-50, 300))
```



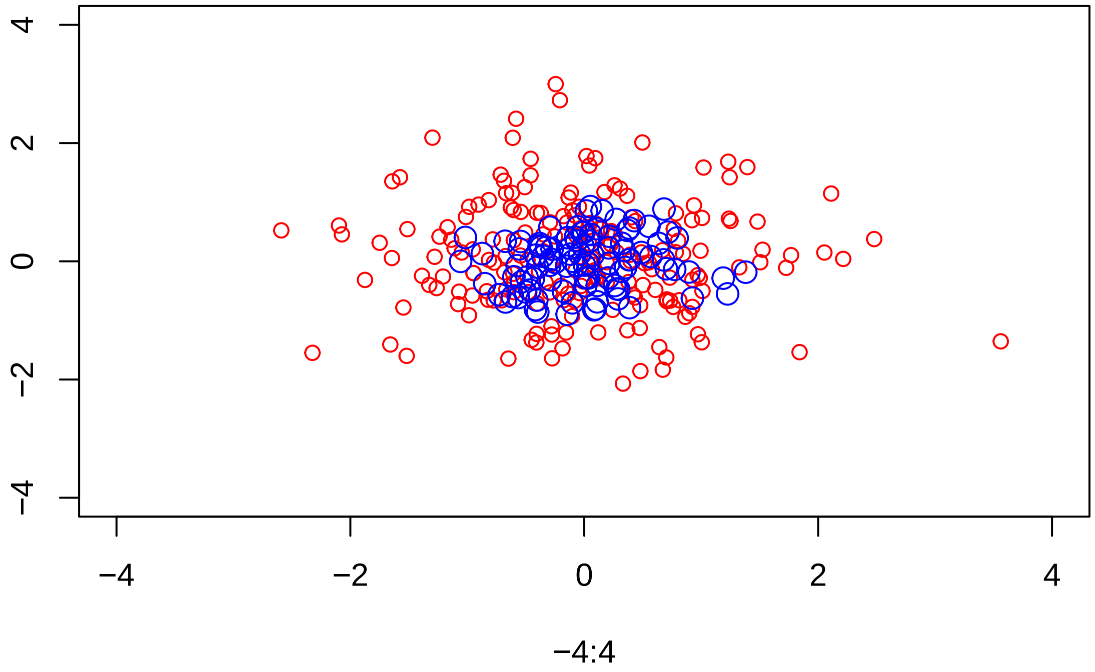
```
##
## scttr3>   cubedraw(rr)
##
## scttr3>   ## 6 b) The rainbow colors from rainbow(201)
## scttr3>   rbc <- rainbow(201)
##
## scttr3>   Rrb <- t(col2rgb(rbc))
##
## scttr3>   rR <- scatterplot3d(Rrb, color = rbc, box = FALSE, angle = 24,
## scttr3+       xlim = c(-50, 300), ylim = c(-50, 300), zlim = c(-50, 300))
```



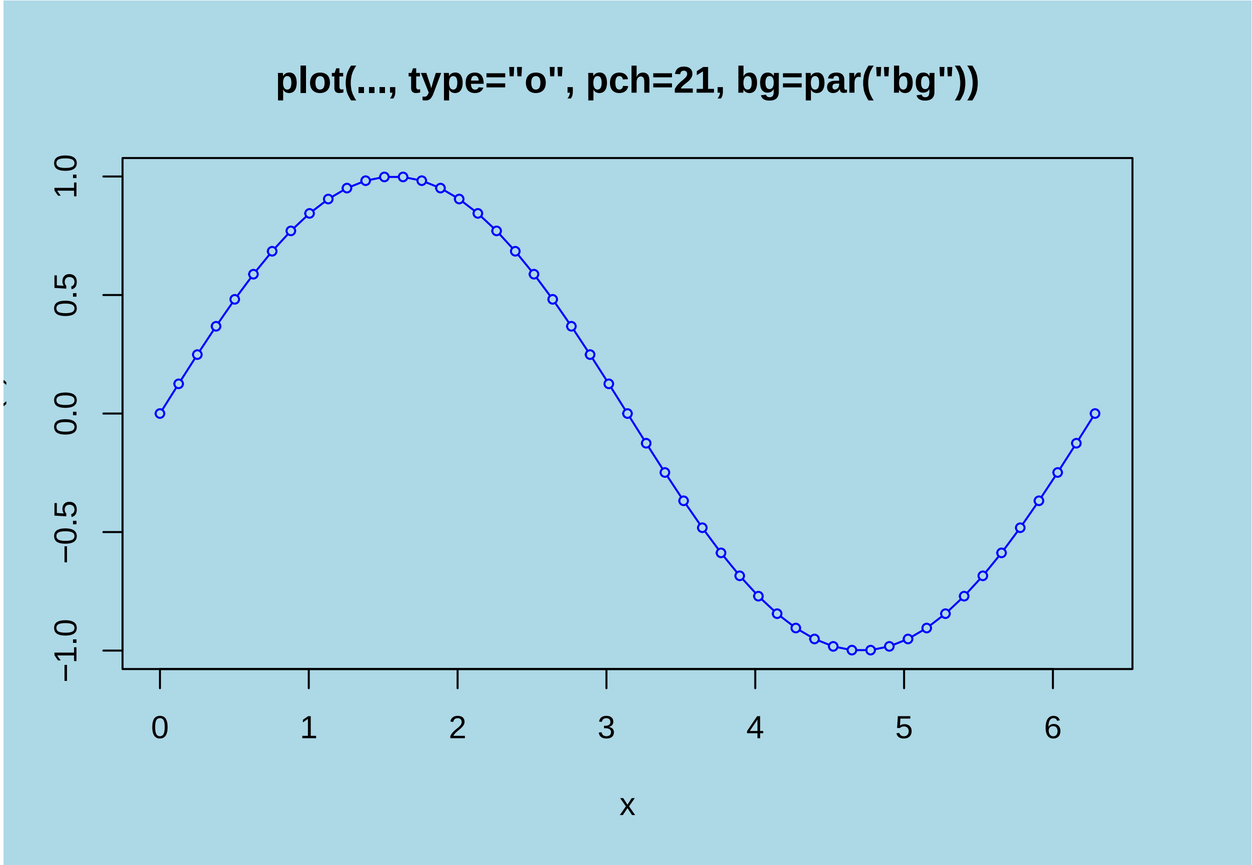
```
##
## scttr3>   cubedraw(rR)
##
## scttr3>   rR$points3d(Rrb, col = rbc, pch = 16)
```

```
example(points)
```

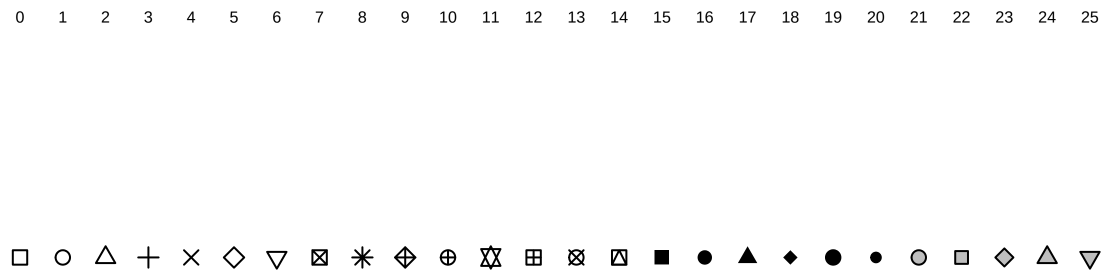
```
##
## points> require(stats) # for rnorm
##
## points> plot(-4:4, -4:4, type = "n")  # setting up coord. system
```



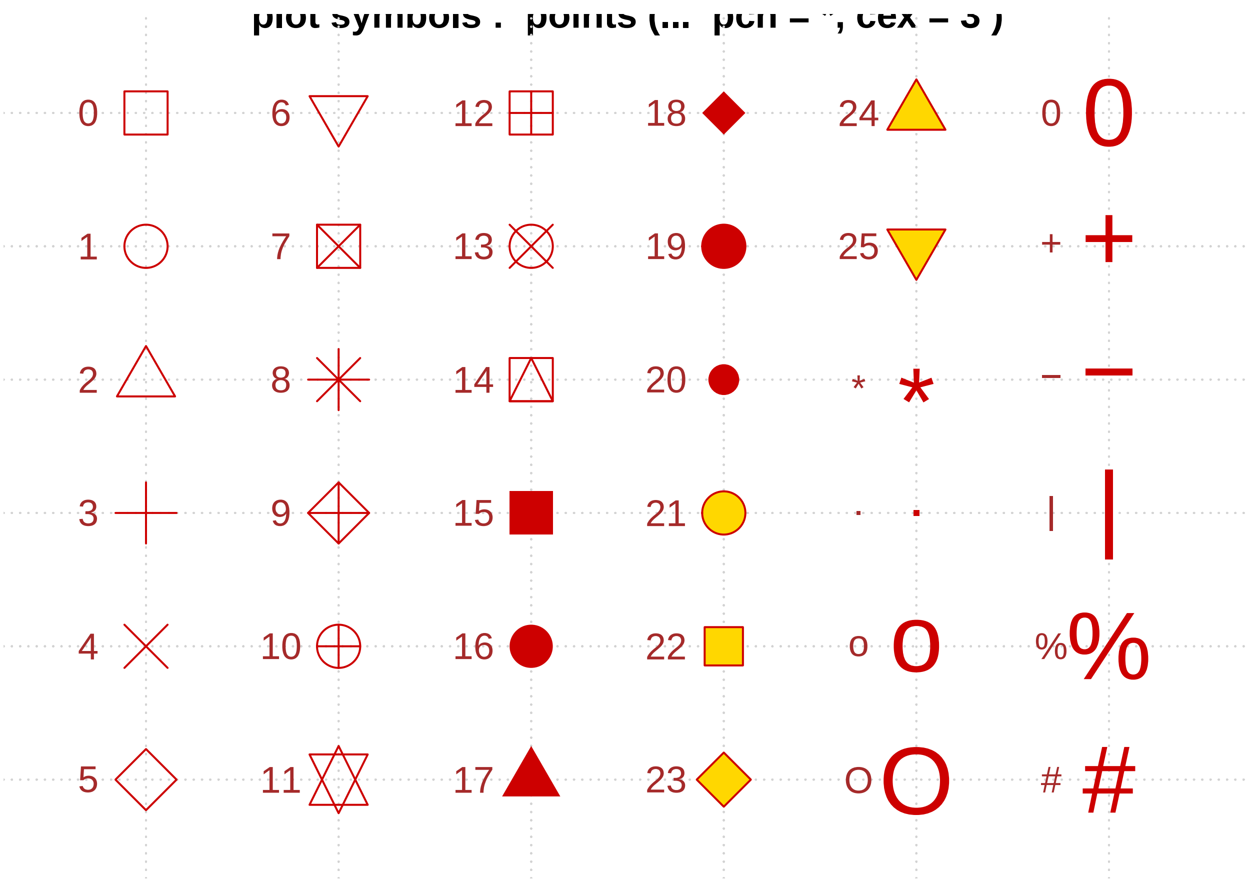
```
##
## points> points(rnorm(200), rnorm(200), col = "red")
##
## points> points(rnorm(100)/2, rnorm(100)/2, col = "blue", cex = 1.5)
##
## points> op <- par(bg = "light blue")
##
## points> x <- seq(0, 2*pi, length.out = 51)
##
## points> ## something "between type='b' and type='o'":
## points> plot(x, sin(x), type = "o", pch = 21, bg = par("bg"), col = "blue", cex = .6,
## points+  main = 'plot(..., type="o", pch=21, bg=par("bg"))')
```



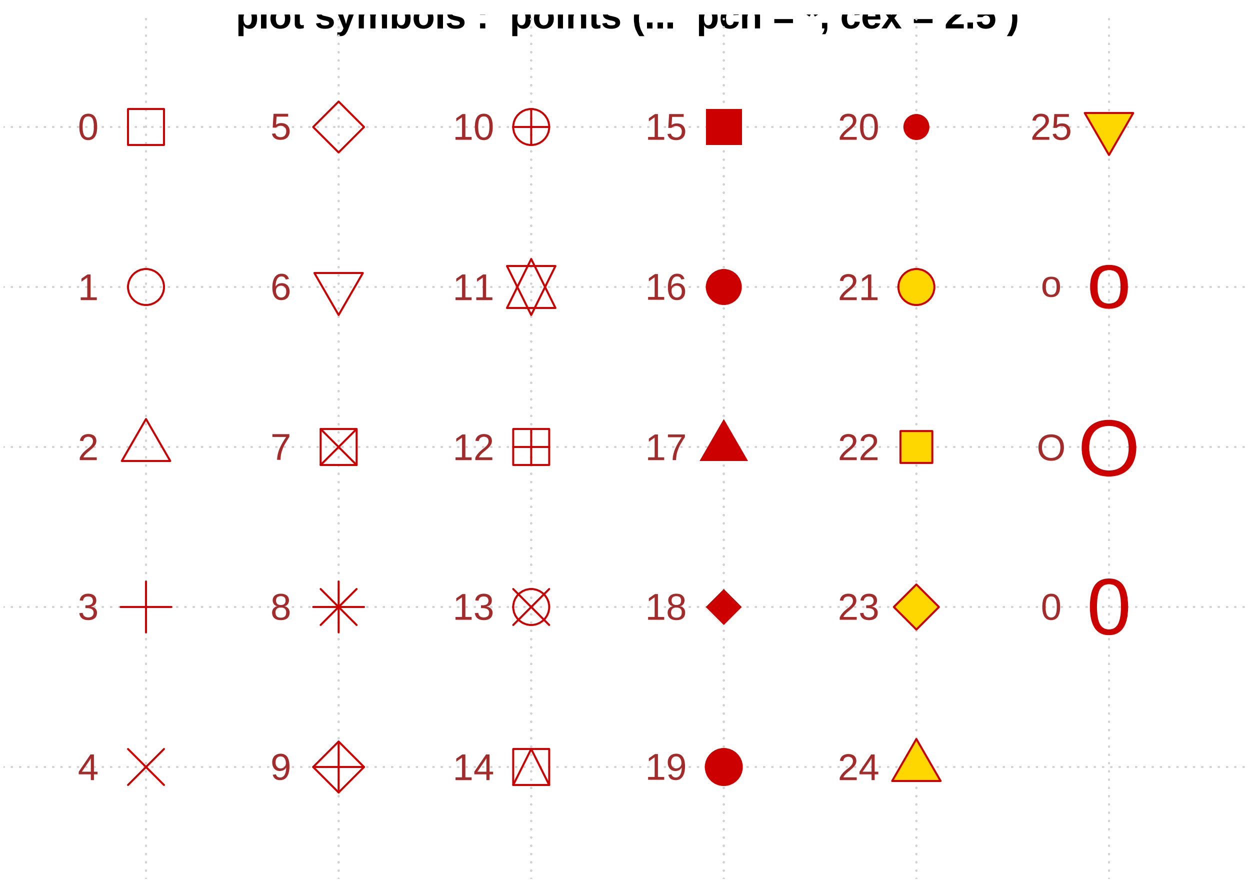
```
##
## points> par(op)
##
## points> ## Illustration of pch = 0:25 (as in the figure shown above in PDF/HTML help)
## points> ## Not run: png("pch.png", height = 0.7, width = 7, res = 100, units = "in")
## points> par(mar = rep(0,4))
##
## points> plot(c(-1, 26), 0:1, type = "n", axes = FALSE)
```



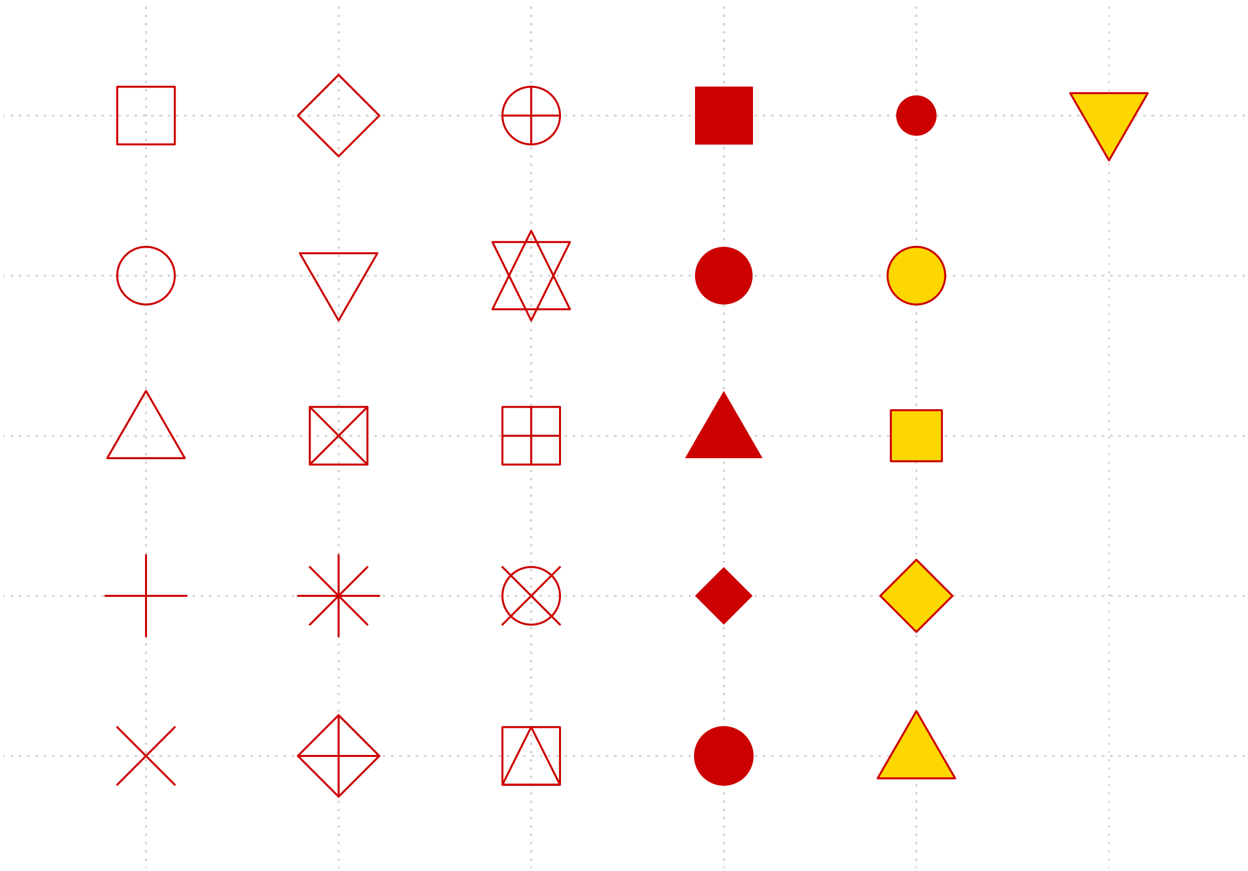
```
##
## points> text(0:25, 0.6, 0:25, cex = 0.5)
##
## points> points(0:25, rep(0.3, 26), pch = 0:25, bg = "grey")
##
## points> ##-------- Showing all the extra & some char graphics symbols ---------
## points> pchShow <-
## points+   function(extras = c("*",".", "o","O","0","+","-","|","%","#"),
## points+            cex = 3, ## good for both .Device=="postscript" and "x11"
## points+            col = "red3", bg = "gold", coltext = "brown", cextext = 1.2,
## points+            main = paste("plot symbols :  points (...  pch = *, cex =",
## points+                         cex,")"))
## points+   {
## points+     nex <- length(extras)
## points+     np  <- 26 + nex
## points+     ipch <- 0:(np-1)
## points+     k <- floor(sqrt(np))
## points+     dd <- c(-1,1)/2
## points+     rx <- dd + range(ix <- ipch %/% k)
## points+     ry <- dd + range(iy <- 3 + (k-1)- ipch %% k)
## points+     pch <- as.list(ipch) # list with integers & strings
## points+     if(nex > 0) pch[26+ 1:nex] <- as.list(extras)
## points+     plot(rx, ry, type = "n", axes  =  FALSE, xlab = "", ylab = "", main = main)
## points+     abline(v = ix, h = iy, col = "lightgray", lty = "dotted")
## points+     for(i in 1:np) {
## points+       pc <- pch[[i]]
## points+       ## 'col' symbols with a 'bg'-colored interior (where available) :
## points+       points(ix[i], iy[i], pch = pc, col = col, bg = bg, cex = cex)
## points+       if(cextext > 0)
## points+           text(ix[i] - 0.3, iy[i], pc, col = coltext, cex = cextext)
## points+     }
## points+   }
##
## points> pchShow()
```



```
##
## points> pchShow(c("o","O","0"), cex = 2.5)
```



```
##
## points> pchShow(NULL, cex = 4, cextext = 0, main = NULL)
```



```
##
## points> ## No test:
## points> ##D ## ------------ test code for various pch specifications -------------
## points> ##D # Try this in various font families (including Hershey)
## points> ##D # and locales.  Use sign = -1 asserts we want Latin-1.
## points> ##D # Standard cases in a MBCS locale will not plot the top half.
## points> ##D TestChars <- function(sign = 1, font = 1, ...)
## points> ##D {
## points> ##D    MB <- l10n_info()$MBCS
## points> ##D    r <- if(font == 5) { sign <- 1; c(32:126, 160:254)
## points> ##D        } else if(MB) 32:126 else 32:255
## points> ##D    if (sign == -1) r <- c(32:126, 160:255)
## points> ##D    par(pty = "s")
## points> ##D    plot(c(-1,16), c(-1,16), type = "n", xlab = "", ylab = "",
## points> ##D         xaxs = "i", yaxs = "i",
## points> ##D         main = sprintf("sign = %d, font = %d", sign, font))
## points> ##D    grid(17, 17, lty = 1) ; mtext(paste("MBCS:", MB))
## points> ##D    for(i in r) try(points(i%%16, i%/%16, pch = sign*i, font = font,...))
## points> ##D }
## points> ##D TestChars()
## points> ##D try(TestChars(sign = -1))
## points> ##D TestChars(font = 5)  # Euro might be at 160 (0+10*16).
## points> ##D                      # macOS has apple at 240 (0+15*16).
## points> ##D try(TestChars(-1, font = 2))  # bold
## points> ## End(No test)
## points>
## points>
```