Introduction to R and Rstudio

Anders K. Krabberød (modified from Ramiro Logares)

October 2021

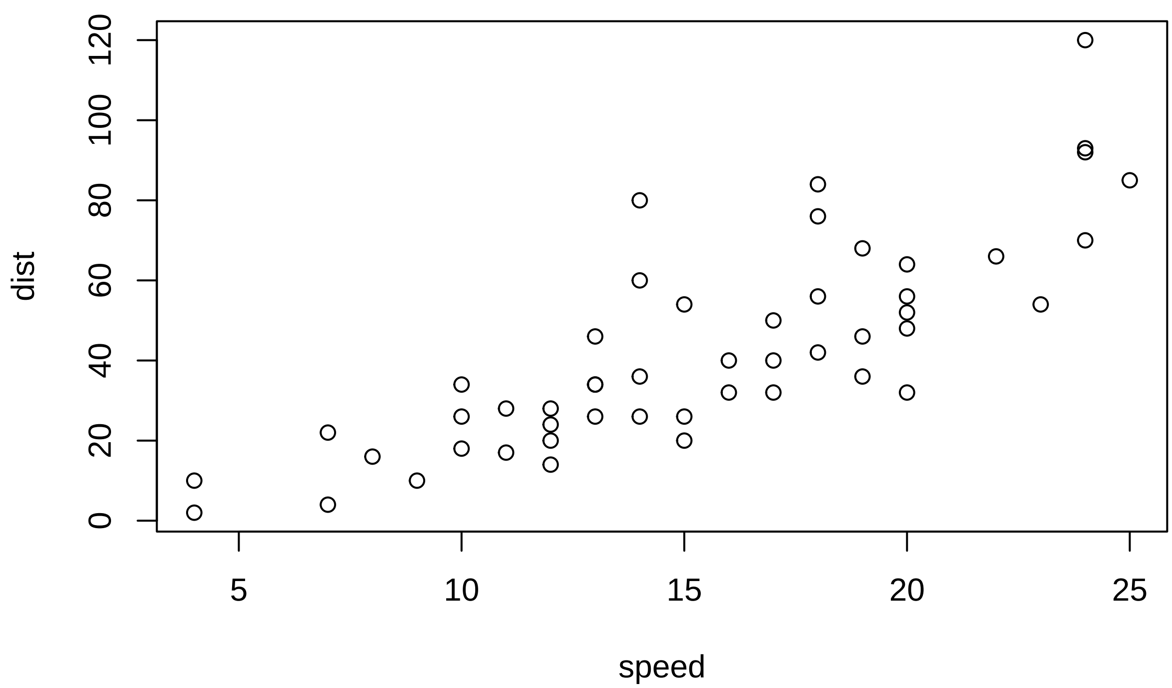
This is an R Markdown Notebook. Markdown is a lightweight markup language for creating formatted text using a plain-text editor. For more information see the book R Markdown: The Definitive Guide.

When you execute code within the notebook, the results appear beneath the code.

Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor inside it and pressing *Cmd+Shift+Enter*.

Example:

```
plot(cars)
```



Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Cmd+Option+I* (macOS) or *Ctrl+Alt+I* (Windows).

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Cmd+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

Basic operations

```r
6+6 # sums two numbers. The answer will appear underneath:
```

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

```r
6-6
```

```
## [1] 0
```

```r
6*6
```

```
## [1] 36
```

```r
6/6
```

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

```r
log(6)
```

```
## [1] 1.791759
```

```r
log2(6)
```

```
## [1] 2.584963
```

```r
log10(6)
```

```
## [1] 0.7781513
```

```r
log(6,7)
```

```
## [1] 0.9207822
```

```r
log(6,3)
```

```
## [1] 1.63093
```

```r
6^6
```

```
## [1] 46656
```

```r
sin(pi/2)
```

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

```r
cos(pi/2)
```

```
## [1] 6.123234e-17
```

```r
# ETC
```

The hashtag `#` can be used inside chunks for commenting.

Define a variable with `<-` You can also use an equal sign, but this is not recommended. The "arrow" makes the code easier to read.

I can write **things**!

```r
mysum <- 6+6 #sum two integers and assign it to a variable
mysum = 6+6 #same as above
mysum #check variable content by executing "mysum"
```

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

Check the list of variables defined so far

```
ls()
```

```
## [1] "mysum"
```

The same variables are also listed in the *Environment* panel of RStudio

Remove a variable:

```
rm(mysum)
rm(list=ls()) # NB will remove all variables!
```

Objects in R:

R consists of a number of different data objects. Some of the most imporant are vectors, lists, matrices, and data frames **Vectors** are one are one dimensional and contain values of the same type (logical, integer, character, numeric etc.). **Lists** are data objects of R that contain various types of elements including strings, numbers, vectors, and a nested list inside it. It can also consist of matrices or functions as elements. **Matrices** two-dimensional layout data with elements of the same data type. They usually contain numeric values in order to perform mathematical operations.
Data frame is a 2-dimensional data structure wherein each column consists of the value of one variable and each row consists of a value set from each column. Each column can be of separate types.

Data types:

Numeric: numbers with decimals

```
mynumber<-66.6
print(mynumber)
```

```
## [1] 66.6
```

```
mynumber
```

```
## [1] 66.6
```

```
class(mynumber) # check the type
```

```
## [1] "numeric"
```

```
#Integer: numbers with no decimals
mynumber.int<-as.integer(mynumber)
class(mynumber.int)
```

```
## [1] "integer"
```

```
mynumber.int
```

```
## [1] 66
```

Character: can be a letter or a combination of letters enclosed by quotes

```
mychar<-"bioinfo course"
mychar
```

```
## [1] "bioinfo course"
```

```
str(mychar)
```

```
##  chr "bioinfo course"
```

```
class(mychar)
```

```
## [1] "character"
```

Logical: a variable that can be TRUE or FALSE (boolean)

```
im.true<-TRUE
im.true<-FALSE
im.true
```

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

```
class(im.true)
```

```
## [1] "logical"
```

Factor: used to refer to a qualitative relationship. to generate a factor, we'll use a vector defined with the function `c()`

```
myfactor<-factor(c("good", "bad", "ugly", "good","good","bad", "ugly", "stupid"))
myfactor
```

```
## [1] good   bad    ugly   good   good   bad    ugly   stupid
## Levels: bad good stupid ugly
```

```
class(myfactor)
```

```
## [1] "factor"
```

```
levels(myfactor) # this can be used to check the levels of a factor
```

```
## [1] "bad"    "good"   "stupid" "ugly"
```

```
nlevels(myfactor)
```

```
## [1] 4
```

```
class(levels(myfactor))
```

```
## [1] "character"
```

```
?nlevels
```

Lists It can contain elements of various data types (e.g.vectors,functions,matrices,another list) Example with three different data types in one list

```
list1<-c(1:15) # integer vector
list2<-factor(1:5) # factor vector
list3<-letters[1:5]
grouped.lists<-list(list1,list2,list3)
grouped.lists
```

```
## [[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
##
## [[2]]
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5
##
## [[3]]
## [1] "a" "b" "c" "d" "e"
```

```
str(grouped.lists)
```

```
## List of 3
##  $ : int [1:15] 1 2 3 4 5 6 7 8 9 10 ...
##  $ : Factor w/ 5 levels "1","2","3","4",..: 1 2 3 4 5
##  $ : chr [1:5] "a" "b" "c" "d" ...
```

Accessing elements of a list

```
grouped.lists[[2]] # accessing the first vector
```

```
## [1] 1 2 3 4 5
## Levels: 1 2 3 4 5
```

```
grouped.lists[[3]][5] # accessing the 5th element from the third vector
```

```
## [1] "e"
```

Ungroup the list

```
ungrouped.list<-unlist(grouped.lists)
class(ungrouped.list)  # NB: the list becomes a character datatype. WHY?
```

```
## [1] "character"
```

```
length(ungrouped.list)
```

```
## [1] 25
```

```
length(grouped.lists)
```

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

Vectors Objects that are used to store values or other information of the same data type They are created with the function "c()" that will generate a 1D array

```
species<-c(123,434,655,877,986) # we create a numeric vector
class(species)
```

```
## [1] "numeric"
```

```
length(species) # number of elements in the vector
```

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

```
species[1] # accessing the fifth element in the vector
```

```
## [1] 123
```

```
species[1:3]
```

```
## [1] 123 434 655
```

```
sum(species) # sum the values in the vector
```

```
## [1] 3075
```

```
species.names<-c("dog","lion","human","pig","cow") # we create a character vector
class(species.names)
```

```
## [1] "character"
```

Create a sequence of numbers:

```r
seq.num<-c(1:100)
seq.num
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
##  [19]  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
##  [37]  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
##  [55]  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
##  [73]  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
##  [91]  91  92  93  94  95  96  97  98  99 100
```

```r
sum(seq.num)
```

```
## [1] 5050
```

```r
seq.num.by2 <- seq(1,100, 2)
```

we can access the 5th to the 10th element using the :

```r
seq.num.by2[5:10]
```

```
## [1]  9 11 13 15 17 19
```

Matrix
Like a vector, a matrix stores information of the same data type, but it has 2 dimensions.

syntax for generating a matrix:
`mymatrix <- matrix(vector, nrow=r, ncol=c, byrow=FALSE, dimnames=list(char_vector_rownames, char_vector_colnames))`

`byrow=F` indicates that the matrix should be filled by columns

```r
mymatrix <- matrix(seq(1:100), nrow=10, ncol=10, byrow=FALSE, dimnames=list(c(3:12), letters[1:10]))
mymatrix
```

```
##     a  b  c  d  e  f  g  h  i   j
## 3   1 11 21 31 41 51 61 71 81  91
## 4   2 12 22 32 42 52 62 72 82  92
## 5   3 13 23 33 43 53 63 73 83  93
## 6   4 14 24 34 44 54 64 74 84  94
## 7   5 15 25 35 45 55 65 75 85  95
## 8   6 16 26 36 46 56 66 76 86  96
## 9   7 17 27 37 47 57 67 77 87  97
## 10  8 18 28 38 48 58 68 78 88  98
## 11  9 19 29 39 49 59 69 79 89  99
## 12 10 20 30 40 50 60 70 80 90 100
```

A matrix with random numbers:

```r
mymatrix.rand <- matrix(sample (seq(1:100),100), nrow=10, ncol=10, byrow=FALSE, dimnames=list(c(1:10), l
mymatrix.rand
```

```
##     a  b  c  d  e   f  g  h  i  j
## 1  39 57 96 94 78  79 14 16 36 35
## 2  59 47 75 68 11 100 84 21 19 72
## 3  76 74 70 89 66  45 23 63 32 43
## 4  44 92 34  2 10  17 22 48 24 27
## 5  56 53 46 20 97  54 51 71 69  1
## 6   5 73 80 90 86   4  6 82 52  3
## 7  37 15 33 99 65  77 31  7 38  8
```

```
## 8  93 41 28 67 81  12 98 55 85 60
## 9  91 58  9 95 18  30 61 64 42 87
## 10 40 26 29 83 50  13 88 49 25 62
```

```
mymatrix[3:6,c(4,7)] # We select what sections of the matrix we want to look at
```

```
##    d  g
## 5 33 63
## 6 34 64
## 7 35 65
## 8 36 66
```

```
mymatrix[3:6,c("b","c")]
```

```
##    b  c
## 5 13 23
## 6 14 24
## 7 15 25
## 8 16 26
```

Dataframes are more general than a matrix and can contain different data types.
Variables or features are in columns, while observations are in rows.
=>NB: this is one of the most common objects in metabarcoding analyses<= Generated with the `data.frame()` function

```
my.data.frame<-data.frame(
  Name=c("Game of Thrones","MrRobot","WestWorld"),
  Budget=c(344,59,122),
  Seasons=c(8,4,3),
  Audience=c(300,14,80),
  Actors=c(221,56, 90)
)
print(my.data.frame)
```

```
##              Name Budget Seasons Audience Actors
## 1 Game of Thrones    344       8      300    221
## 2         MrRobot     59       4       14     56
## 3       WestWorld    122       3       80     90
```

```
row.names(my.data.frame)<-my.data.frame[,1] # Assign to the row names the names in the first column
my.data.frame<-my.data.frame[,-1] # Remove the fisrt column
my.data.frame # By clicking this object in the "Environment" panel on the right, you'll see a window wi
```

```
##                 Budget Seasons Audience Actors
## Game of Thrones    344       8      300    221
## MrRobot             59       4       14     56
## WestWorld          122       3       80     90
```

```
class(my.data.frame)
```

```
## [1] "data.frame"
```

```
ncol(my.data.frame) # Number of columns
```

```
## [1] 4
```

```
nrow(my.data.frame) # Number of rows
```

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

```r
colnames(my.data.frame) # Column names
```

```
## [1] "Budget"   "Seasons"  "Audience" "Actors"
```

```r
rownames(my.data.frame) # Name of rows
```

```
## [1] "Game of Thrones" "MrRobot"         "WestWorld"
```

```r
colSums(my.data.frame) # Sum values in columns
```

```
##   Budget  Seasons Audience   Actors
##      525       15      394      367
```

```r
rowSums(my.data.frame) # We sum the values, even if they make no sense in the example
```

```
## Game of Thrones         MrRobot       WestWorld
##             873             133             295
```

```r
rbind(my.data.frame,my.data.frame) #  appends dataframes one below the other (column names identical)
```

```
##                  Budget Seasons Audience Actors
## Game of Thrones     344       8      300    221
## MrRobot              59       4       14     56
## WestWorld           122       3       80     90
## Game of Thrones1    344       8      300    221
## MrRobot1             59       4       14     56
## WestWorld1          122       3       80     90
```

```r
cbind(my.data.frame,my.data.frame) # appends dataframes one next to the other (row names identical)
```

```
##                 Budget Seasons Audience Actors Budget Seasons Audience Actors
## Game of Thrones    344       8      300    221    344       8      300    221
## MrRobot             59       4       14     56     59       4       14     56
## WestWorld          122       3       80     90    122       3       80     90
```

```r
head(my.data.frame, 2) # Useful to have a look to the beginning of the dataframe (specially useful in b
```

```
##                 Budget Seasons Audience Actors
## Game of Thrones    344       8      300    221
## MrRobot             59       4       14     56
```

```r
my.data.frame[1:2,2:4] # Useful to look at specific sections of the dataframe
```

```
##                 Seasons Audience Actors
## Game of Thrones       8      300    221
## MrRobot               4       14     56
```

Let's generate a dataframe with different data types

```r
my.data.frame.2<-data.frame(
  Name=c("Game of Thrones","MrRobot","WestWorld", "Chernobyl"),
  Rating=c("Excellent","Very Good","Excellent", "Very Good"),
  Audience.Restriction=c(TRUE,FALSE,TRUE, FALSE)
)
my.data.frame.2
```

```
##              Name    Rating Audience.Restriction
## 1 Game of Thrones Excellent                 TRUE
## 2         MrRobot Very Good                FALSE
## 3       WestWorld Excellent                 TRUE
```

```
## 4       Chernobyl Very Good                FALSE
```

Rename the row names

```
row.names(my.data.frame.2)<-my.data.frame.2[,1]
my.data.frame.2
```

```
##                            Name    Rating Audience.Restriction
## Game of Thrones Game of Thrones Excellent                 TRUE
## MrRobot                 MrRobot Very Good                FALSE
## WestWorld             WestWorld Excellent                 TRUE
## Chernobyl             Chernobyl Very Good                FALSE
```

```
my.data.frame.2<-my.data.frame.2[,-1] # Remove redundant column 1
my.data.frame.2
```

```
##                    Rating Audience.Restriction
## Game of Thrones Excellent                 TRUE
## MrRobot         Very Good                FALSE
## WestWorld       Excellent                 TRUE
## Chernobyl       Very Good                FALSE
```

```
str(my.data.frame.2) # Let's look at the data types within this dataframe
```

```
## 'data.frame':    4 obs. of  2 variables:
##  $ Rating              : chr  "Excellent" "Very Good" "Excellent" "Very Good"
##  $ Audience.Restriction: logi  TRUE FALSE TRUE FALSE
```

Variables in this case are characters and logical (TRUE/FALSE)

#Merge two dataframes based in a pattern # We will use the series names to merge these dataframes as this is what they have in common

```
data.frame.large<-merge(my.data.frame, my.data.frame.2, by="row.names") # "by" indicates the column use
```

#Useful commands to work with tables or dataframes

```
getwd() # get working directory
```

```
## [1] "/Users/anderkkr/Dropbox/Projects/00_Master_projects/21_undervisning/2021/UNIS_AB332_prep"
```

You can change the working directory:

```
#setwd("path/to/my/directory") # set working directory
```

```
str(data.frame.large)
```

```
## 'data.frame':    3 obs. of  7 variables:
##  $ Row.names           : 'AsIs' chr  "Game of Thrones" "MrRobot" "WestWorld"
##  $ Budget              : num  344 59 122
##  $ Seasons             : num  8 4 3
##  $ Audience            : num  300 14 80
##  $ Actors              : num  221 56 90
##  $ Rating              : chr  "Excellent" "Very Good" "Excellent"
##  $ Audience.Restriction: logi  TRUE FALSE TRUE
```

```
dim(data.frame.large) # Table dimensions
```

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

```r
nrow(data.frame.large) # Number of rows
```

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

```r
ncol(data.frame.large) # Number of columns
```

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

```r
colnames(data.frame.large) # Name of columns
```

```
## [1] "Row.names"             "Budget"                "Seasons"
## [4] "Audience"              "Actors"                "Rating"
## [7] "Audience.Restriction"
```

```r
rownames(data.frame.large) # Name of rows
```

```
## [1] "1" "2" "3"
```

```r
#colSums(data.frame.large[]) # Sum of numeric values in columns
#rowSums(data.frame.large) # Sum of numeric values in rows
head(data.frame.large) # See table header
```

```
##          Row.names Budget Seasons Audience Actors    Rating Audience.Restriction
## 1 Game of Thrones    344       8      300    221 Excellent                 TRUE
## 2         MrRobot     59       4       14     56 Very Good                FALSE
## 3       WestWorld    122       3       80     90 Excellent                 TRUE
```

```r
t(data.frame.large) # Transpose table
```

```
##                      [,1]              [,2]        [,3]
## Row.names            "Game of Thrones" "MrRobot"   "WestWorld"
## Budget               "344"             " 59"       "122"
## Seasons              "8"               "4"         "3"
## Audience             "300"             " 14"       " 80"
## Actors               "221"             " 56"       " 90"
## Rating               "Excellent"       "Very Good" "Excellent"
## Audience.Restriction "TRUE"            "FALSE"     "TRUE"
```

Table subsetting Format: `my.table[row, column]`

Replace *my.table* with a data frame in the following and see if you understand the different opperations:

```r
my.table[1,2] #Get value from row 1, column 2
my.table[1,]  #Get values from row 1 across all columns
my.table$column.name<-NULL #Remove column
my.table[-5,-2] # Remove row 5 and column 2
my.table[-(5:10),] # Remove rows 5 to 10, keep all columns
my.table[,-(which(colSums(my.table)==0))] # Remove columns that sum 0
```

Installing packages

R has a large repository of packages for different applications

```r
install.packages("vegan") # Installs the community ecology package Vegan with hundreds of functions
library("vegan") # load Vegan
```

#Plotting

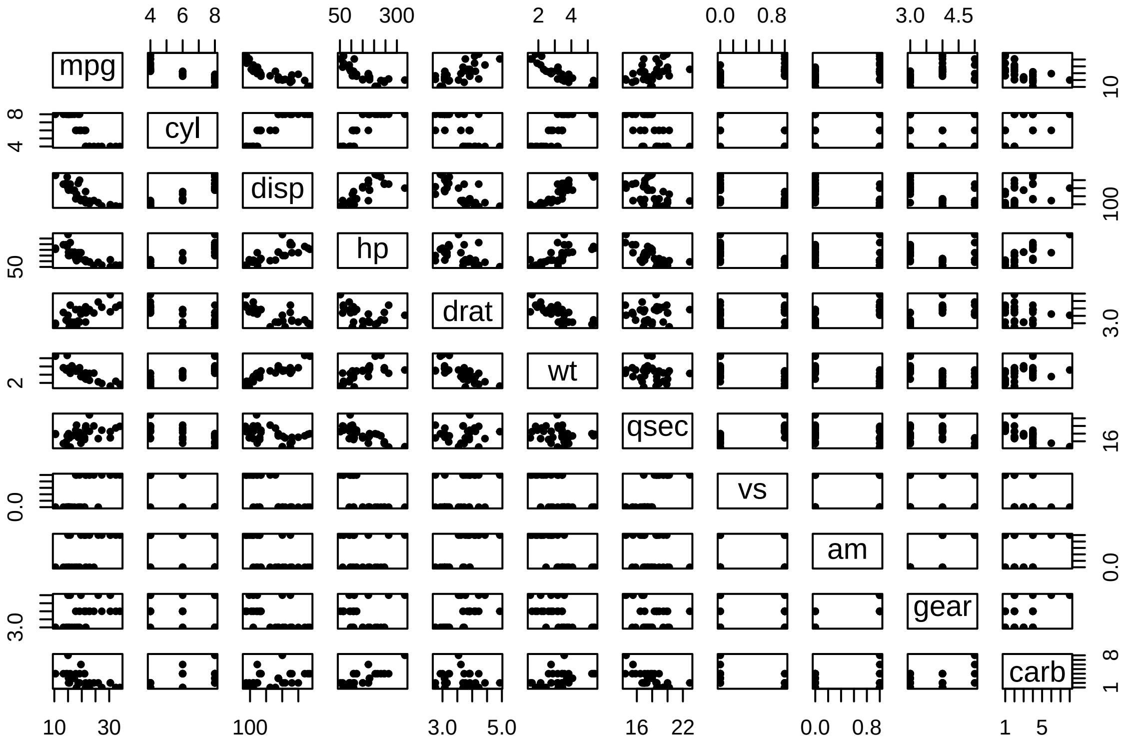
```r
data("mtcars") # We load a dataset that comes with R
mtcars
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

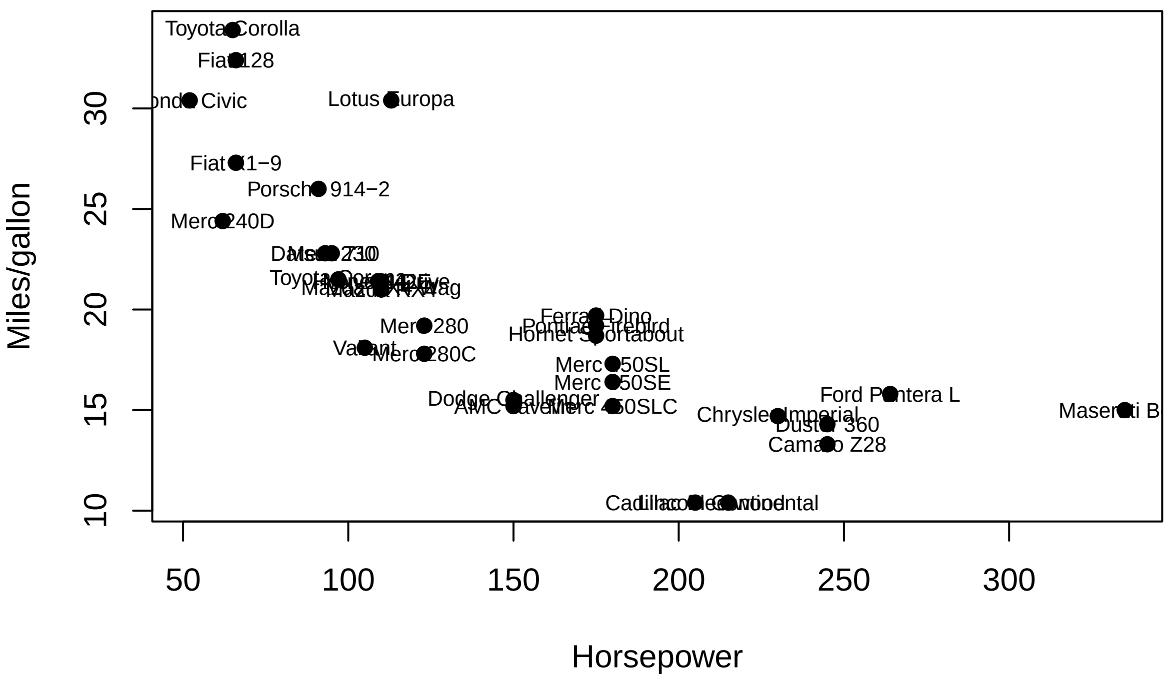
The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models).

```
# [, 1] mpg Miles/(US) gallon
# [, 2] cyl Number of cylinders
# [, 3] disp    Displacement (cu.in.)
# [, 4] hp  Gross horsepower
# [, 5] drat    Rear axle ratio
# [, 6] wt  Weight (1000 lbs)
# [, 7] qsec    1/4 mile time
# [, 8] vs  Engine (0 = V-shaped, 1 = straight)
# [, 9] am  Transmission (0 = automatic, 1 = manual)
# [,10] gear    Number of forward gears
# [,11] carb    Number of carburetors
```

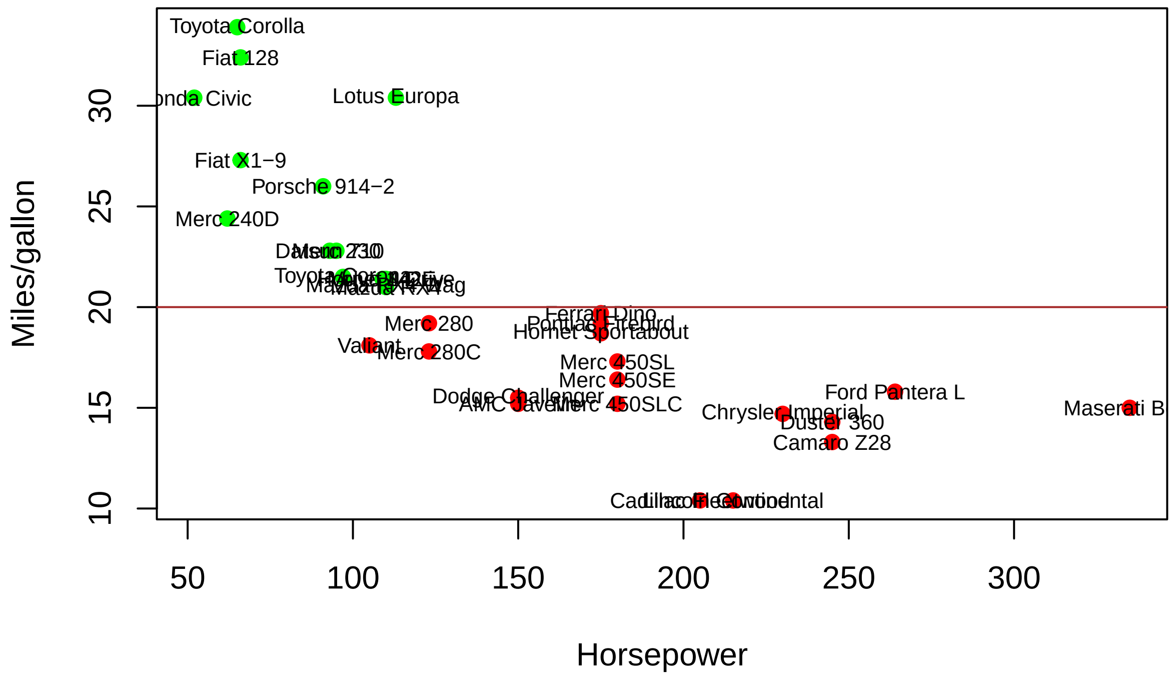
```
#boxplot(mtcars) # make a boxplot of variables across car models

plot(mtcars, pch=19, cex=0.6) # make x-y plots for all variables
```



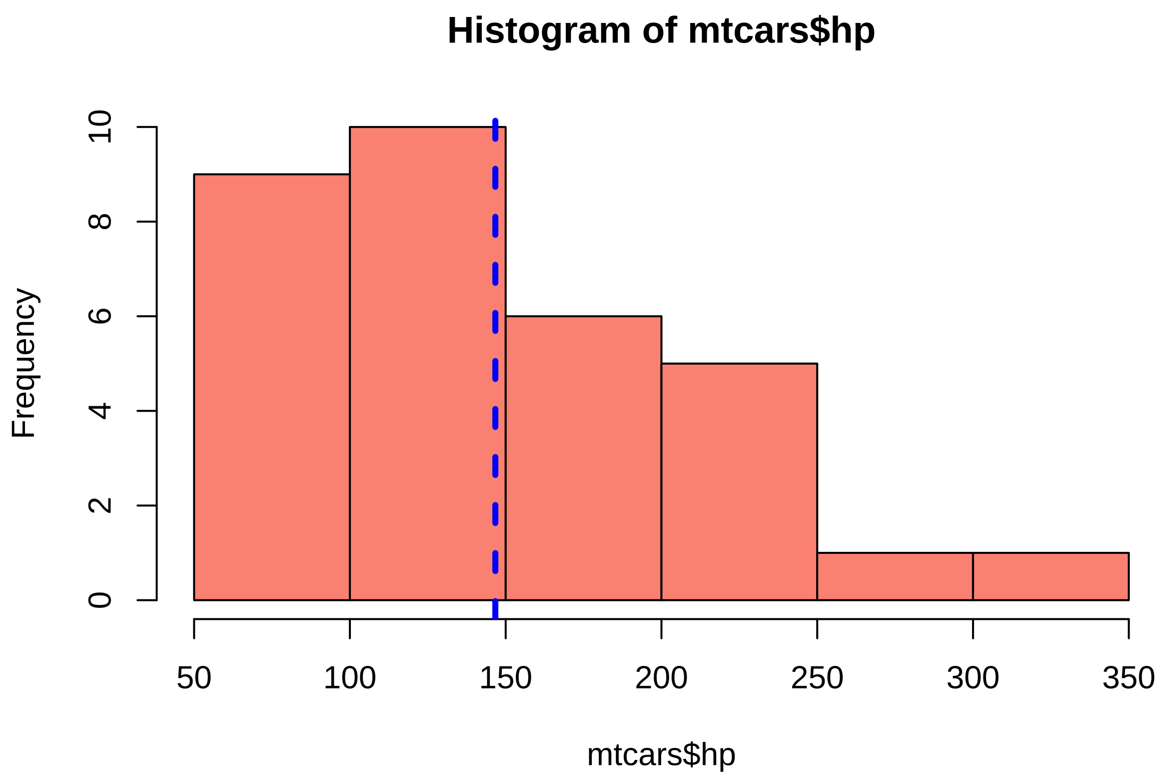

```
plot(mtcars$hp, mtcars$mpg, xlab="Horsepower", ylab="Miles/gallon", pch=19) # we plot horsepower vs. mi
text(mtcars$hp, mtcars$mpg, row.names(mtcars), cex=0.7) # we add the car model
```



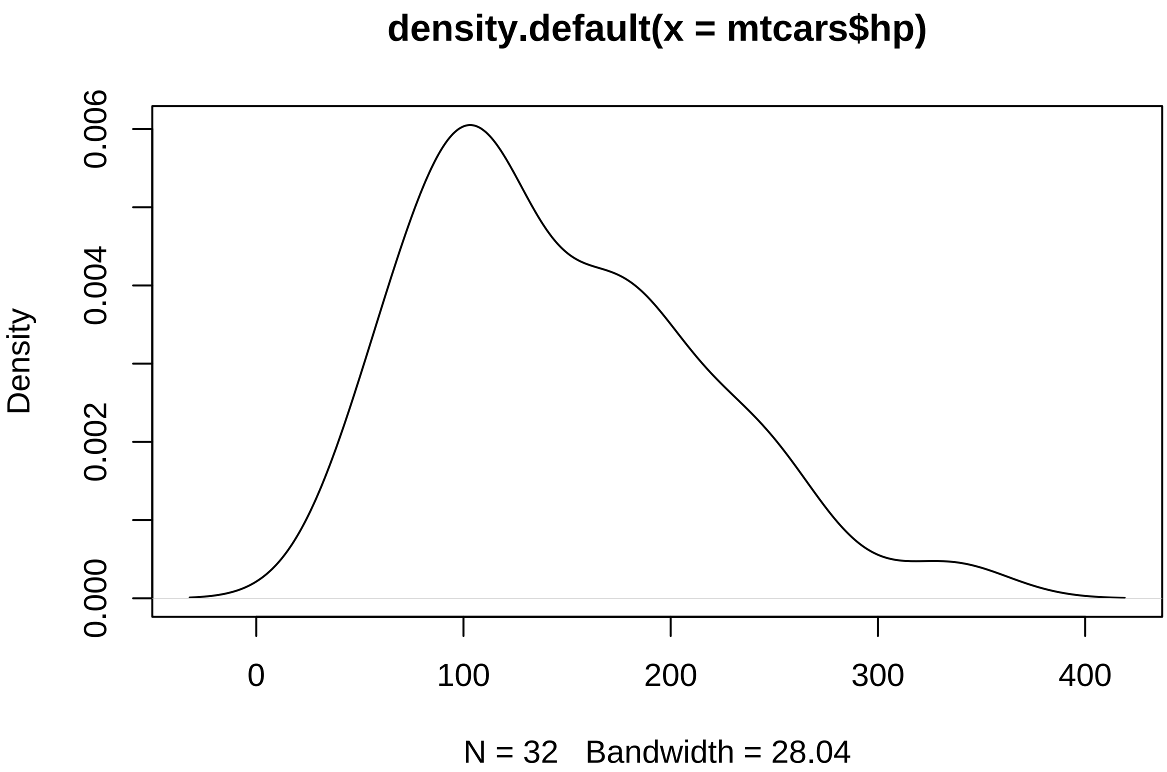

```
plot(mtcars$hp, mtcars$mpg, xlab="Horsepower", ylab="Miles/gallon", pch=19, col=ifelse(mtcars$mpg<20,"re
text(mtcars$hp, mtcars$mpg, row.names(mtcars), cex=0.7) # we add the car model
abline(h=20, col="brown") # add a line to the plot
```




```
hist(mtcars$hp, col="salmon")
abline(v=mean(mtcars$hp), col="blue", lwd=3, lty=2)
```



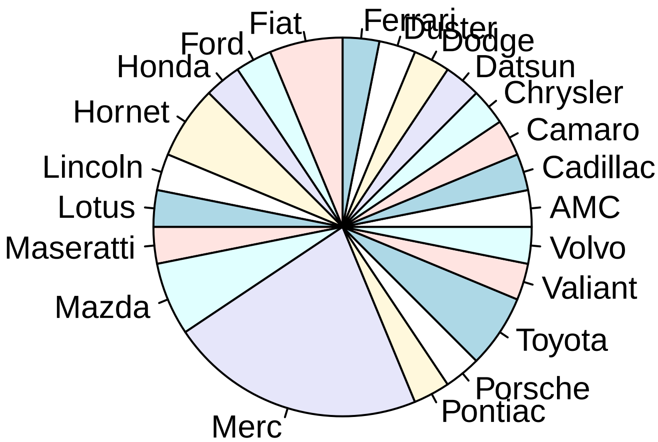

```
plot(density(mtcars$hp))
```




```
brands<-c("Mazda", "Mazda","Datsun", "Hornet", "Hornet", "Valiant", "Duster", "Merc", "Merc", "Merc", "I
          "Honda", "Toyota", "Toyota","Dodge","AMC","Camaro","Pontiac", "Fiat","Porsche", "Lotus", "Ford
mtcars$brand<-brands # we add an extra column with brands
mtcars[1:5,] # let's check
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb  brand
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4  Mazda
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4  Mazda
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1 Datsun
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1 Hornet
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2 Hornet
```

```
pie(table(mtcars$brand)) # we make a piechart of the brands
```



For plotting the package ggplot2 is very becomming increasingly popular: It is already included in the package **tidyverse**, but can also be installed by itself:

```r
#install.packages("ggplot2")
#library(ggplot2)
```

Some tips and hits

Use "?" before any function to get the help page.

```r
?sum
```

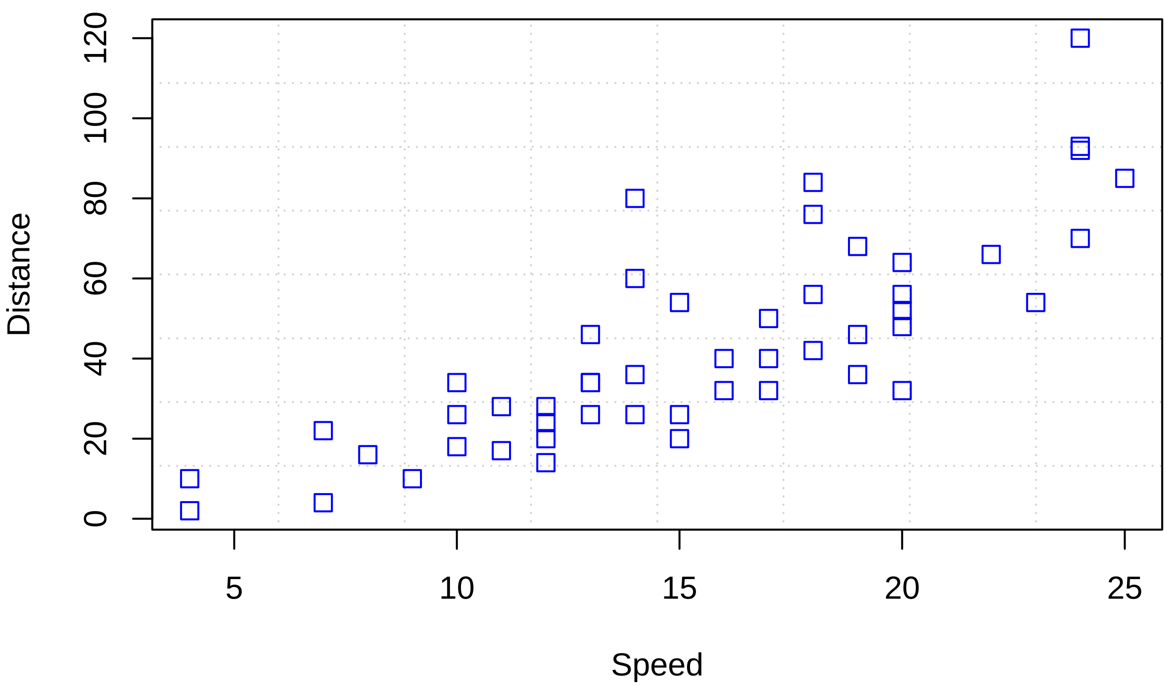
If you don't know where a function is from use "??":

```r
??rarefy
```

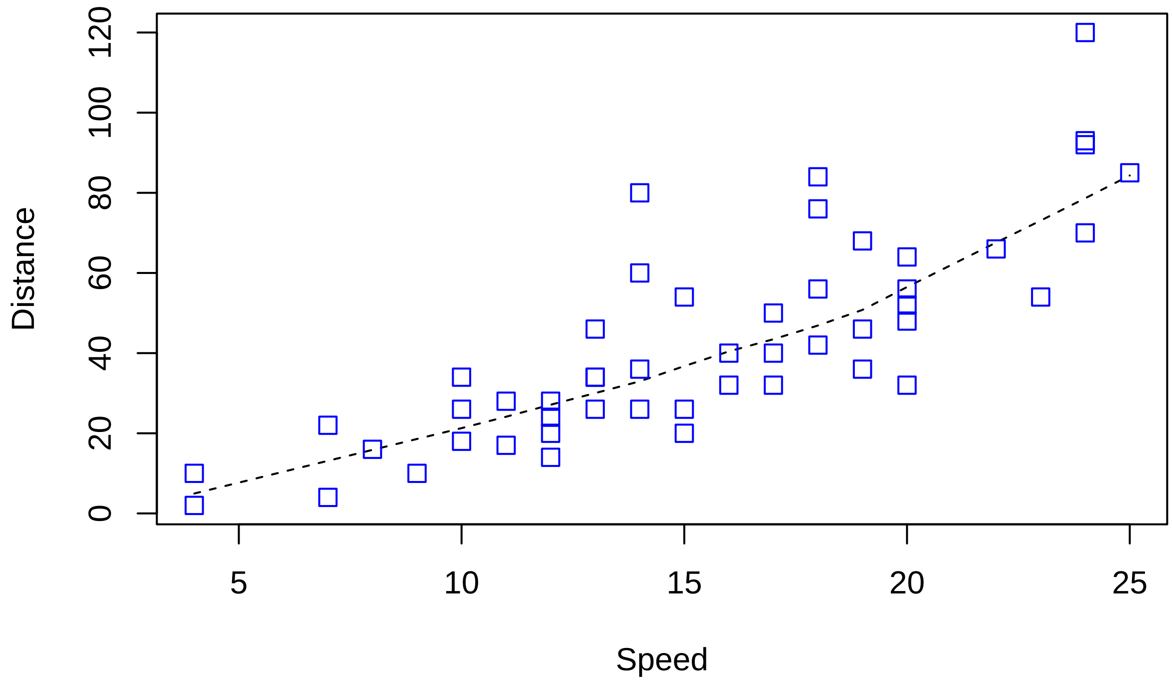
You can ask for an example for a function or package with `example()`. This will print both the code and the result (not all packages provides examples, though)

```r
example(plot)
```

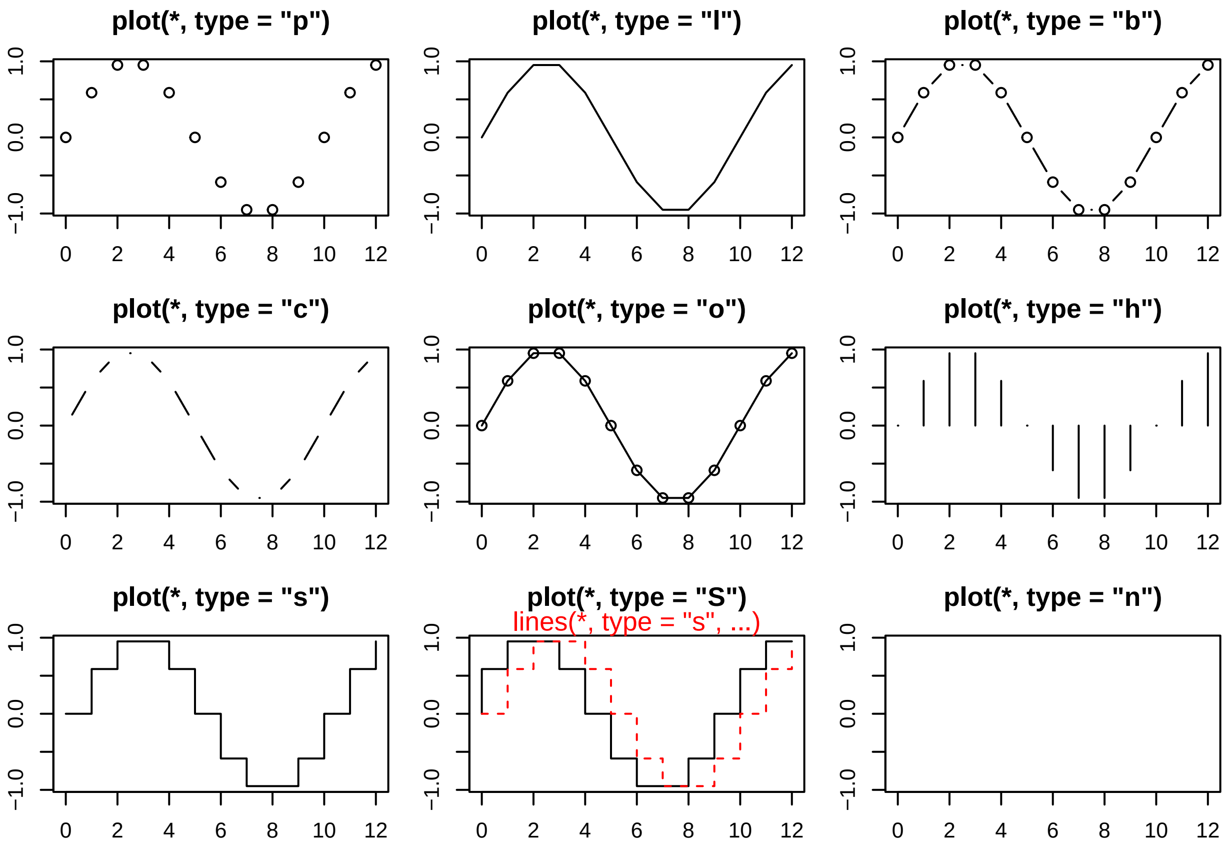
```
##
## plot> Speed <- cars$speed
##
## plot> Distance <- cars$dist
##
## plot> plot(Speed, Distance, panel.first = grid(8, 8),
## plot+      pch = 0, cex = 1.2, col = "blue")
```



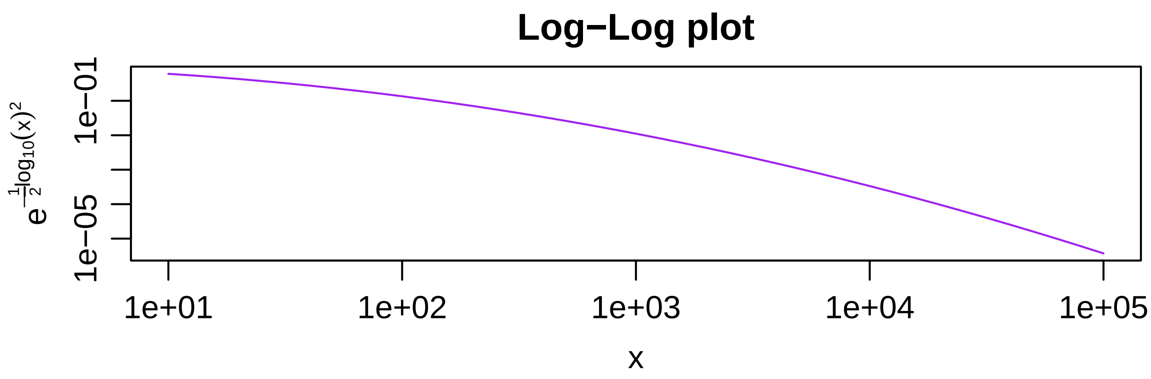

```
##
## plot> plot(Speed, Distance,
## plot+      panel.first = lines(stats::lowess(Speed, Distance), lty = "dashed"),
## plot+      pch = 0, cex = 1.2, col = "blue")
```

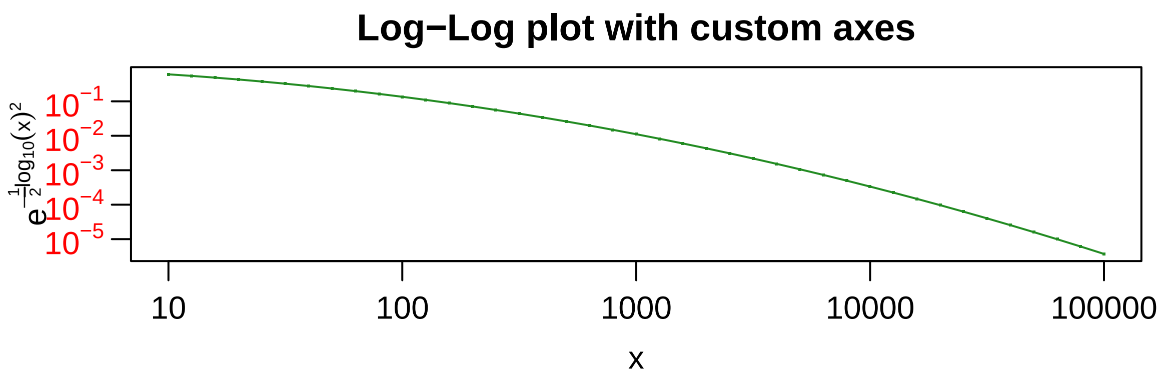



```
##
## plot> ## Show the different plot types
## plot> x <- 0:12
##
## plot> y <- sin(pi/5 * x)
##
## plot> op <- par(mfrow = c(3,3), mar = .1+ c(2,2,3,1))
##
## plot> for (tp in c("p","l","b",  "c","o","h",  "s","S","n")) {
## plot+    plot(y ~ x, type = tp, main = paste0("plot(*, type = \"", tp, "\")"))
## plot+    if(tp == "S") {
## plot+       lines(x, y, type = "s", col = "red", lty = 2)
## plot+       mtext("lines(*, type = \"s\", ...)", col = "red", cex = 0.8)
## plot+    }
## plot+ }
```




```
##
## plot> par(op)
##
## plot> ##--- Log-Log Plot  with  custom axes
## plot> lx <- seq(1, 5, length.out = 41)
##
## plot> yl <- expression(e^{-frac(1,2) * {log[10](x)}^2})
##
## plot> y <- exp(-.5*lx^2)
##
## plot> op <- par(mfrow = c(2,1), mar = par("mar")-c(1,0,2,0), mgp = c(2, .7, 0))
##
## plot> plot(10^lx, y, log = "xy", type = "l", col = "purple",
## plot+      main = "Log-Log plot", ylab = yl, xlab = "x")

##
## plot> plot(10^lx, y, log = "xy", type = "o", pch = ".", col = "forestgreen",
## plot+      main = "Log-Log plot with custom axes", ylab = yl, xlab = "x",
## plot+      axes = FALSE, frame.plot = TRUE)
```






```
##
## plot> my.at <- 10^(1:5)
##
## plot> axis(1, at = my.at, labels = formatC(my.at, format = "fg"))
##
## plot> e.y <- -5:-1 ; at.y <- 10^e.y
##
## plot> axis(2, at = at.y, col.axis = "red", las = 1,
## plot+      labels = as.expression(lapply(e.y, function(E) bquote(10^.(E)))))
##
## plot> par(op)
```

End