Marek Gągolewski
Konstancja Bobecka-Wesołowska
Przemysław Grzegorzewski

Computer Statistics with R

2. Exploratory Data Analysis (Descriptive Statistics)

Faculty of Mathematics and Information Science
Warsaw University of Technology
[Last update: December 9, 2012]

Contents

Info

These tutorials are likely to contain bugs and typos. In case you find any don't hesitate to *contact us*! Thanks in advance!

2.1. Preliminaries

The aim of *exploratory data analysis* is to provide a general insight into a given data set. Such an analysis can be performed to obtain some kind of summary of the data e.g. via graphical displays. The methods described in this chapter are used for preliminary data exploration. They are quite simple and often require no statistical assumptions.

The measurable aspects of objects of concern can be represented by one or many *variables*. Generally, there are two types of variables.

Note

A *qualitative (categorical) variable* represents a property which can be measured on a nominal scale, i.e. it may have one of a fixed set of discrete values called *levels*, *categories* or *classes*. For example, the variable "gender" has (most often) two levels: "male" and "female". Therefore the object "John Smith" will probably characterized by "gender"=="male".

A *quantitative variable* depicts an object's property which can be expressed by a numerical value, e.g. the height of a research participant, his rank in an IQ test etc.

The variable type determines its statistical description methods. Below we list the most commonly used.

1. Qualitative variables:
   (a) Tables: frequency table (`table()`), percentage table (`prop.table()`).
   (b) Graphical methods: pie chart (`pie()`), bar plot (`barplot()`).
2. Quantitative variables:
   (a) Numerical characteristics (sample statistics):
      – Measures of location: mean (`mean()`), trimmed mean (`mean(..., trim=...)`), winsorized mean, median (`median()`) and other quantiles (`min()`, `max()`, `quantile()`), mode.
      – Measures of dispersion: variance (`var()`), standard deviation (`sd()`), interquartile range (`IQR()`), range (`diff(range())`).
      – Measures of distribution shape: skewness, kurtosis.
   (b) Graphical methods: box-and-whisker plot (`boxplot()`), histogram (`hist()`), stem-and-leaf display (`stem()`).

2.2. Analysis of qualitative data

Ex. 2.1. A group of 3rd year Computer Science students wants to elect their class leader. There were four candidates: John, Kate, Mary, and Peter. The 26 votes were given as follows:

> Mary, Mary, Mary, John, Kate, Mary, Peter, Peter, John, John, Peter, Kate, Peter, John, John, Mary, Mary, Mary, Kate, John, John, Mary, John, Mary, Mary, John.

The person who wins the most votes will become the leader for the next academic year. Your task is to help a friend of yours who prepares a newspaper article. Perform an exploratory analysis of the data.

Solution.
First we input the data into R. We have 26 voting cards, the results may be expressed as a character vector.

```
votes <- c("Mary", "Mary", "Mary", "John", "Kate", "Mary", "Peter",
           "Peter", "John", "John", "Peter", "Kate", "Peter", "John",
           "John", "Mary", "Mary", "Mary", "Kate", "John", "John", "Mary",
           "John", "Mary", "Mary", "John");
votes <- factor(votes); # convert to factor - categorization
length(votes);
## [1] 26
```

These are qualitative data. The variable of interest has 4 levels. Each level corresponds to a different candidate.

Frequency table

Vote counting may be done via the `table()` function.

```
(votesTab <- table(votes)); # frequency table
## votes
##  John  Kate  Mary Peter
##     9     3    10     4
```

Who is the winner? Of course, it's Mary. She got 10 votes.

Proportions table

The results may also be displayed as a *proportions table*.

```
prop.table(votesTab)
## votes
##   John   Kate   Mary  Peter
## 0.3462 0.1154 0.3846 0.1538
```

Kate got only 11.5% of the votes. Note that `prop.table()` has been invoked on a frequency table, not on a character vector.

We can split the frequency table into two vectors:

```
cands <- names(votesTab); # only category names
NumVotes <- as.vector(votesTab); # only vote counts
print(cands);
## [1] "John"  "Kate"  "Mary"  "Peter"
print(NumVotes);
## [1]  9  3 10  4
cands[2]; NumVotes[2]; # results for 2nd person
## [1] 3
```

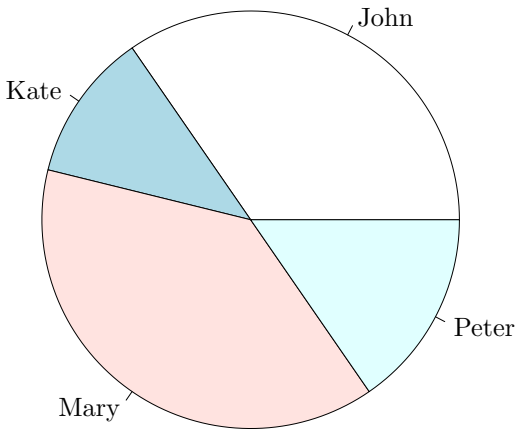
See how we may reconstruct the `votes` vector (up to a permutation of its elements) using `cands` and `NumVotes`.

```
rep(cands, NumVotes)
##  [1] "John"  "John"  "John"  "John"  "John"  "John"  "John"  "John"  "John"
## [10] "Kate"  "Kate"  "Kate"  "Mary"  "Mary"  "Mary"  "Mary"  "Mary"  "Mary"
## [19] "Mary"  "Mary"  "Mary"  "Mary"  "Peter" "Peter" "Peter" "Peter"
```

Pie chart

In many applications a graphical representation of data is more user-friendly. Among the most popular is the *pie chart*.

```
pie(votesTab)
```



or, equivalently:

```
pie(NumVotes, labels = cands)
```

Task

Try also some different settings:

```
pie(votesTab, col=c("red", "blue", "yellow", "green"));
pie(votesTab, col=heat.colors(4));
pie(votesTab, border=NULL, radius=1, main="Election results",
   labels=c("J", "K", "M", "P"));
pie(votesTab, labels=paste(cands, " - ", NumVotes));
```

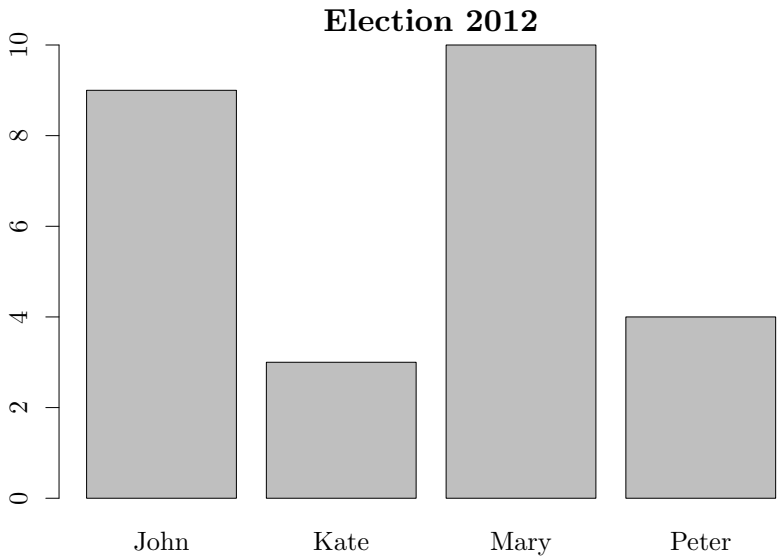
More details on graphical parameters may be found in Sec. 2.6.

Bar plot

A *bar plot* is a different type of chart:

```
barplot(votesTab, main = "Election 2012")
```

Some say that bar plots may be read more efficiently. People have problems with comparing areas of objects. It is much easier to differentiate their lengths.



Task

Consider also:

```
barplot(NumVotes, names=cands);
barplot(prop.table(votesTab), names=as.vector(prop.table(votesTab)),
   horiz=T, legend=names(votesTab), col=rainbow(4));
```

Info

No wonder we say that a picture is worth a thousand words!

□

Ex. 2.2. The file `cars.csv` stores information on the most popular car makes in the 1970's.

1. The variable `mpg` describes cars' fuel efficiency in miles per US gallon. Create a variable `fc`, expressing fuel consumption in liters per 100 kilometers.
2. Categorize the newly-created variable as follows:

Category code	Fuel consumption [l/100 km]
small	less than 7
medium	no less than 7, less than 10
large	no less than 10

3. Create a bar plot for the fuel consumption.

Solution.
The database is stored in a CSV file. Here we have its preview:

```
mpg;cylinders;horsepower;accel;year;weight;origin;make;model;price;carmakers
43,1;4;48;21,5;78;1985;2;Volkswagen;Rabbit Dl ;2400;America
```

```
36,1;4;66;14,4;78;1800;1;Ford      ;Fiesta    ;1900;Europe
32,8;4;52;19,4;78;1985;3;Mazda     ;GLC Deluxe;2200;Japan
39,4;4;70;18,6;78;2070;3;Datsun    ;B210 GX   ;2725;
36,1;4;60;16,4;78;1800;3;Honda     ;Civic CVCC;2250;
19,9;8;110;15,5;78;3365;1;Oldsmobile;Cutlass   ;3300;
19,4;8;140;13,2;78;3735;1;Dodge     ;Diplomat  ;3125;
20,2;8;139;12,8;78;3570;1;Mercury   ;Monarch   ;2850;
19,2;6;105;19,2;78;3535;1;Pontiac   ;Phoenix   ;2800;
...
```

The first line consists of column (variable) names (`header=T`). The successive lines store information on different cars (one car per row). The columns are separated with a semi-colon (`sep=";"`). Additionally, a comma serves for a decimal separator (`dec=","`). Let us use the `read.table()` function to import the file.

```
cars <- read.table("../datasets/cars.csv", head = T, sep = ";", dec = ",")
```

First we should check if the file was imported correctly:

```
class(cars); # is it really a data frame?
```

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

```
head(cars)   # display a few initial rows
```

```
##    mpg cylinders horsepower accel year weight origin       make      model
## 1 43.1         4         48  21.5   78   1985      2 Volkswagen Rabbit Dl
## 2 36.1         4         66  14.4   78   1800      1 Ford       Fiesta
## 3 32.8         4         52  19.4   78   1985      3 Mazda      GLC Deluxe
## 4 39.4         4         70  18.6   78   2070      3 Datsun     B210 GX
## 5 36.1         4         60  16.4   78   1800      3 Honda      Civic CVCC
## 6 19.9         8        110  15.5   78   3365      1 Oldsmobile Cutlass
##   price carmakers
## 1  2400   America
## 2  1900   Europe
## 3  2200   Japan
## 4  2725
## 5  2250
## 6  3300
```

Note that the two last columns were printed below the rest. This is because the screen width was to small to display them together.

In this exercise the `mpg` variable is of our interest. It is of quantitative type. However, we are going to categorize its values, that is represent it as qualitative data.

```
cars$mpg
```

```
##   [1] 43.1 36.1 32.8 39.4 36.1 19.9 19.4 20.2 19.2 20.5 20.2 25.1 20.5 19.4
##  [15] 20.6 20.8 18.6 18.1 19.2 17.7 18.1 17.5 30.0 27.5 27.2 30.9 21.1 23.2
##  [29] 23.8 23.9 20.3 17.0 21.6 16.2 31.5 29.5 21.5 19.8 22.3 20.2 20.6 17.0
##  [43] 17.6 16.5 18.2 16.9 15.5 19.2 18.5 31.9 34.1 35.7 27.4 25.4 23.0 27.2
##  [57] 23.9 34.2 34.5 31.8 37.3 28.4 28.8 26.8 33.5 41.5 38.1 32.1 37.2 28.0
##  [71] 26.4 24.3 19.1 34.3 29.8 31.3 37.0 32.2 46.6 27.9 40.8 44.3 43.4 36.4
##  [85] 30.4 44.6 40.9 33.8 29.8 32.7 23.7 35.0 23.6 32.4 27.2 26.6 25.8 23.5
##  [99] 30.0 39.1 39.0 35.1 32.3 37.0 37.7 34.1 34.7 34.4 29.9 33.0 34.5 33.7
## [113] 32.4 32.9 31.6 28.1   NA 30.7 25.4 24.2 22.4 26.6 20.2 17.6 28.0 27.0
## [127] 34.0 31.0 29.0 27.0 24.0 23.0 36.0 37.0 31.0 38.0 36.0 36.0 36.0 34.0
## [141] 38.0 32.0 38.0 25.0 38.0 26.0 22.0 32.0 36.0 27.0 27.0 44.0 32.0 28.0
## [155] 31.0
```

```
length(cars$mpg)
```

```
## [1] 155
```

Among the 155 records there is one missing observation. We are going to remove it, as this information is not useful in this case.

Conversion of units

It can be easily seen that to convert miles per US gallon to liters per 100 km we should transform the data as follows.

$$\texttt{fc} = \frac{1}{\texttt{mpg}} \frac{3.785 \cdot 100}{1.609}, \tag{2.1}$$

because 1 mile = 1.609 km and 1 gallon = 3.785 l. The resulting vector will be stored as `fc`:

```
# remove missing observations:
mpg <- na.omit(cars$mpg);            # or:
mpg <- as.vector(na.omit(cars$mpg)); # or:
mpg <- cars$mpg[!is.na(cars$mpg)];
# convert:
fc <- 3.785*100/(mpg*1.609);
print(fc, digits=3);
```

```
##   [1]  5.46  6.52  7.17  5.97  6.52 11.82 12.13 11.65 12.25 11.48 11.65
##  [12]  9.37 11.48 12.13 11.42 11.31 12.65 13.00 12.25 13.29 13.00 13.44
##  [23]  7.84  8.55  8.65  7.61 11.15 10.14  9.88  9.84 11.59 13.84 10.89
##  [34] 14.52  7.47  7.97 10.94 11.88 10.55 11.65 11.42 13.84 13.37 14.26
##  [45] 12.93 13.92 15.18 12.25 12.72  7.37  6.90  6.59  8.59  9.26 10.23
##  [56]  8.65  9.84  6.88  6.82  7.40  6.31  8.28  8.17  8.78  7.02  5.67
##  [67]  6.17  7.33  6.32  8.40  8.91  9.68 12.32  6.86  7.89  7.52  6.36
##  [78]  7.31  5.05  8.43  5.77  5.31  5.42  6.46  7.74  5.27  5.75  6.96
##  [89]  7.89  7.19  9.93  6.72  9.97  7.26  8.65  8.84  9.12 10.01  7.84
## [100]  6.02  6.03  6.70  7.28  6.36  6.24  6.90  6.78  6.84  7.87  7.13
## [111]  6.82  6.98  7.26  7.15  7.44  8.37  7.66  9.26  9.72 10.50  8.84
## [122] 11.65 13.37  8.40  8.71  6.92  7.59  8.11  8.71  9.80 10.23  6.53
## [133]  6.36  7.59  6.19  6.53  6.53  6.53  6.92  6.19  7.35  6.19  9.41
## [144]  6.19  9.05 10.69  7.35  6.53  8.71  8.71  5.35  7.35  8.40  7.59
```

Categorization of a quantitative variable

We are now ready to split the quantitative variable into 3 disjoint classes. The results will be stored in a factor-type vector named `fuelcons`. Its length should be equal to the length of `fc`. Each element `fuelcons[i]` will describe a category for `fc[i]`, `i` $= 1, ..., 154$.

```
fuelcons <- character(length(fc)); # an empty character vector
fuelcons[fc<7] <- "small";
fuelcons[fc>=7 & fc<10] <- "medium";
fuelcons[fc>=10] <- "large";
fuelcons <- factor(fuelcons); # convert to factor
print(fuelcons);
```

```
##   [1] small  small  medium small  small  large  large  large  large  large
##  [11] large  medium large  large  large  large  large  large  large  large
##  [21] large  large  medium medium medium medium large  large  medium medium
##  [31] large  large  large  large  medium medium large  large  large  large
##  [41] large  large  large  large  large  large  large  large  large  medium
##  [51] small  small  medium medium large  medium medium small  small  medium
##  [61] small  medium medium medium medium small  small  medium small  medium
##  [71] medium medium large  small  medium medium small  medium small  medium
##  [81] small  small  small  small  medium small  small  small  medium medium
##  [91] medium small  medium medium medium medium medium large  medium small
## [101] small  small  medium small  small  small  small  small  medium medium
## [111] small  small  medium medium medium medium medium medium medium large
## [121] medium large  large  medium medium small  medium medium medium medium
## [131] large  small  small  medium small  small  small  small  small  small
## [141] medium small  medium small  medium large  medium small  medium medium
## [151] small  medium medium medium
```

```
## Levels: large medium small
```

It is worth noting that the expression:

```
fuelcons[fc < 7] <- "small"
```

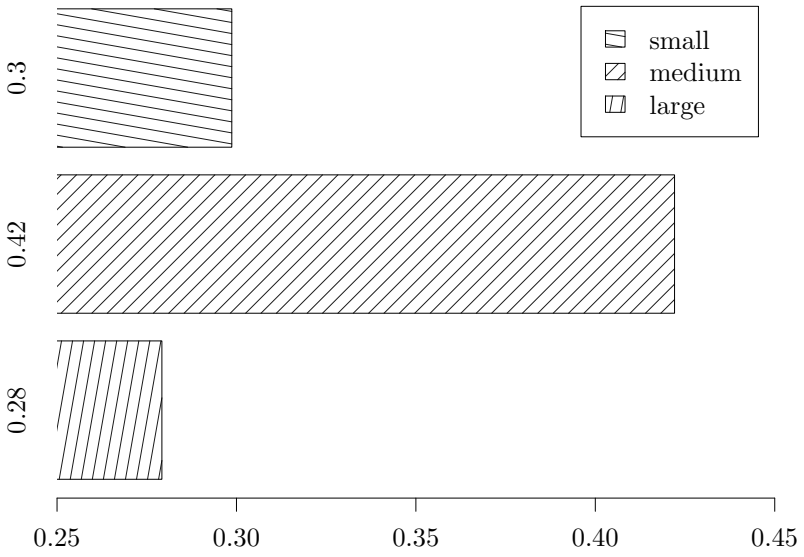
means "get all elements in `fuelcons` which represent the cars with fuel consumption less than 7 l/100 km. Then, assign them the category `"small"`".

We are now ready to draw a bar plot for the fuel consumption.

```
fcTab <- table(fuelcons); print(fcTab);
```

```
## fuelcons
##  large medium  small
##     43     65     46
```

```
barplot(prop.table(fcTab), names=round(as.vector(prop.table(fcTab)), 2),
   horiz=T, legend=names(fcTab), xlim=c(0.25, 0.45), xpd=FALSE,
   col="gray10", density=15, angle=c(80,45,-10));
```



Details

The same task may also be done in a different way, using the `cut()` function. It categorizes numeric variables according to the intervals of form $(b_1, b_2], (b_2, b_3], \ldots, (b_{n-1}, b_n]$ (closed on the right, (`right=TRUE`) or $[b_1, b_2), [b_2, b_3), \ldots, [b_{n-1}, b_n)$ (closed on the left, `right=FALSE`) for some $b_1 < b_2 < \cdots < b_n$.

```
cut(fc, c(-Inf, 7, 10, Inf), right = FALSE)
##   [1] [-Inf,7)  [-Inf,7)  [7,10)    [-Inf,7)  [-Inf,7)  [10, Inf) [10, Inf)
##   [8] [10, Inf) [10, Inf) [10, Inf) [10, Inf) [7,10)    [10, Inf) [10, Inf)
##  [15] [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf)
##  [22] [10, Inf) [7,10)    [7,10)    [7,10)    [7,10)    [10, Inf) [10, Inf)
##  [29] [7,10)    [7,10)    [10, Inf) [10, Inf) [10, Inf) [10, Inf) [7,10)
##  [36] [7,10)    [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf)
##  [43] [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf) [10, Inf)
##  [50] [7,10)    [-Inf,7)  [-Inf,7)  [7,10)    [7,10)    [10, Inf) [7,10)
##  [57] [7,10)    [-Inf,7)  [-Inf,7)  [7,10)    [-Inf,7)  [7,10)    [7,10)
##  [64] [7,10)    [7,10)    [-Inf,7)  [-Inf,7)  [7,10)    [-Inf,7)  [7,10)
##  [71] [7,10)    [7,10)    [10, Inf) [-Inf,7)  [7,10)    [7,10)    [-Inf,7)
##  [78] [7,10)    [-Inf,7)  [7,10)    [-Inf,7)  [-Inf,7)  [-Inf,7)  [-Inf,7)
##  [85] [7,10)    [-Inf,7)  [-Inf,7)  [-Inf,7)  [7,10)    [7,10)    [7,10)
##  [92] [-Inf,7)  [7,10)    [7,10)    [7,10)    [7,10)    [7,10)    [10, Inf)
##  [99] [7,10)    [-Inf,7)  [-Inf,7)  [-Inf,7)  [7,10)    [-Inf,7)  [-Inf,7)
## [106] [-Inf,7)  [-Inf,7)  [-Inf,7)  [7,10)    [7,10)    [-Inf,7)  [-Inf,7)
## [113] [7,10)    [7,10)    [7,10)    [7,10)    [7,10)    [7,10)    [7,10)
## [120] [10, Inf) [7,10)    [10, Inf) [10, Inf) [7,10)    [7,10)    [-Inf,7)
## [127] [7,10)    [7,10)    [7,10)    [7,10)    [10, Inf) [-Inf,7)  [-Inf,7)
## [134] [7,10)    [-Inf,7)  [-Inf,7)  [-Inf,7)  [-Inf,7)  [-Inf,7)  [-Inf,7)
## [141] [7,10)    [-Inf,7)  [7,10)    [-Inf,7)  [7,10)    [10, Inf) [7,10)
## [148] [-Inf,7)  [7,10)    [7,10)    [-Inf,7)  [7,10)    [7,10)    [7,10)
## Levels: [-Inf,7) [7,10) [10, Inf)
```

⊡

2.3. Analysis of quantitative data

Ex. 2.3. Make a preliminary analysis of fuel consumption (in l/100 km) of vehicles from the `cars.csv` file.

Solution.
This time we consider the `fc` vector as-is, that is as a quantitative variable.

Numerical characteristics

Let us begin with calculating some numerical characteristics of the fuel consumption. We assume that you know the definitions of the following measures.

1. Measures of location:

```
mean(fc)  # arithmetic mean

## [1] 8.767

median(fc)  # median

## [1] 8.14

min(fc)  # minimum

## [1] 5.048
```

```
max(fc)  # maximum

## [1] 15.18

range(fc)  # min. & max. as one vector

## [1]  5.048 15.177

quantile(fc, c(0.1, 0.25, 0.5, 0.75, 0.9))  # some quantiles

##    10%    25%    50%    75%    90%
##  6.191  6.863  8.140 10.433 12.297

summary(fc)  # a convenient function, many-in-one

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.05    6.86    8.14    8.77   10.40   15.20

mean(fc, trim = 0.1)  # trimmed mean (10% each side)

## [1] 8.558
```

2. Measures of dispersion:

```
var(fc)  # variance

## [1] 5.895

sd(fc)  # standard deviation

## [1] 2.428

IQR(fc)  # interquartile range

## [1] 3.57

diff(range(fc))  # range

## [1] 10.13

sd(fc)/mean(fc)  # coefficient of variation

## [1] 0.277
```

3. Measures of distribution shape:

```
library("e1071")  # an auxiliary library is required

## Loading required package:  class
```

```
skewness(fc)  # skewness

## [1] 0.6802

kurtosis(fc)  # kurtosis

## [1] -0.5847
```

Info

It is worth noting that the `skewness()` and `kurtosis()` functions implement biased estimators of the underlying parameters.

Note

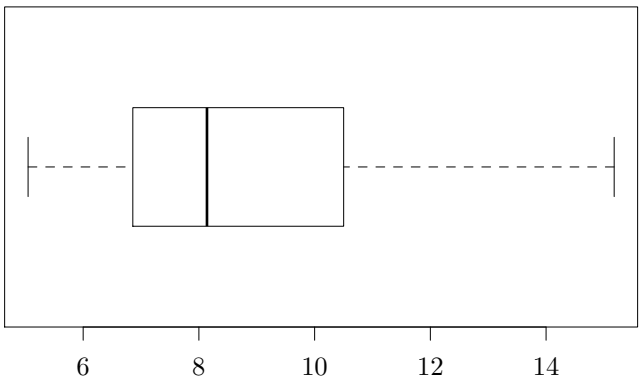
If the library `e1071` has not been pre-installed in your R distribution, you should download it by calling:

```
install.packages("e1071")
```

Box-and-whisker plot

A *box-and-whisker* plot is a graphical method to represent basic sample statistics and to identify potential outliers. We say that an observation is an *outlier* if its value is less than $Q_1 - 1.5\,\mathrm{IQR}$ or greater than $Q_3 + 1.5\,\mathrm{IQR}$, where Q_1 — the value of the first sample quartile, Q_3 — the third quartile, and $\mathrm{IQR} = Q_3 - Q_1$.

```
boxplot(fc, horizontal = T)  # horizontal box plot
```



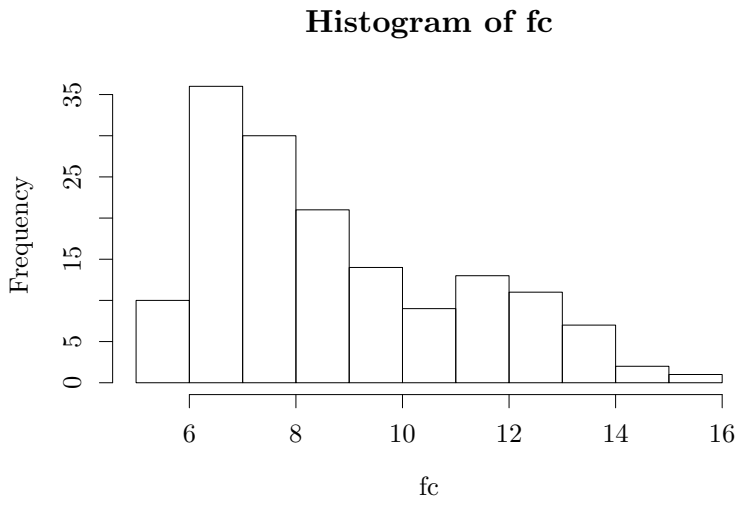
There are no outliers in the analyzed set. The empirical distribution is right-skewed.

Histogram

A *histogram* estimates the shape of an underlying probability distribution (density) function. The observations are grouped first into disjoint classes (intervals in the real line

of the same length). Next, the number of values falling in each interval is counted, and the results are drawn on a figure similar to the bar plot.

```
hist(fc)
```



Again, we observe that the data are right skewed. The histogram for `fc` consists of 11 classes.

Details

R selects the number of classes using the Sturges formula by default. Other "heuristic" formulas may be chosen by setting the parameter `breaks=`*Identifier*. The table below lists the available identifiers.

Identifier	Name	Expression
`"Sturges"`	Sturges' formula	$k = \lceil \log_2 n + 1 \rceil$ $(n \geq 30)$,
`"Scott"`	Scott's formula for the normal distribution	$h = 3.5s/n^{1/3}$,
`"FD"`	Freedman-Diaconis' formula	$h = 2\ \mathrm{IQR}/n^{1/3}$,

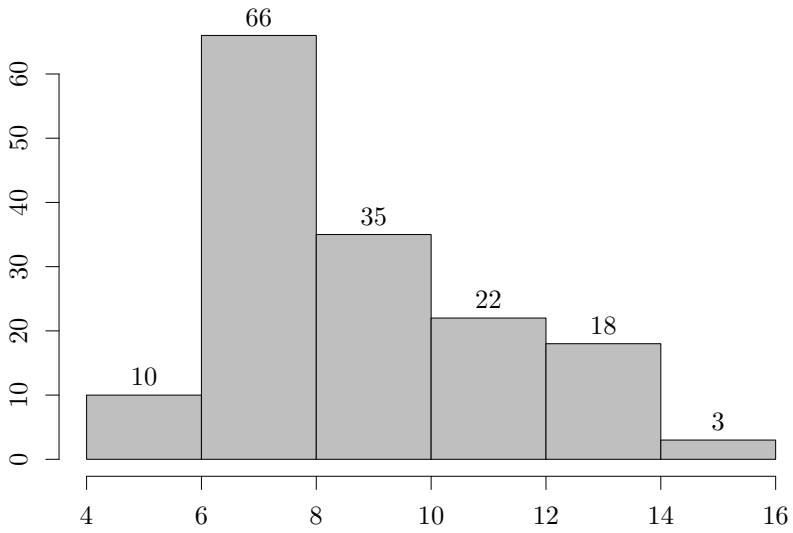
where k — number of classes, h — interval width, n — number of observations, s — sample standard deviation, IQR — interquartile range.

We may also *suggest* (R just knows better what we really need) the desired number of classes.

```
# the histogram will be stored as an object
par(xpd=TRUE) # allow drawing on figure margins
h <- hist(fc, breaks=5, # suggested number of classes=5
              labels=T, # add a label above each bar
  col="gray", main=NA)
h # print its properties
```

```
## $breaks
## [1]  4  6  8 10 12 14 16
##
## $counts
## [1] 10 66 35 22 18  3
##
## $intensities
## [1] 0.03247 0.21429 0.11364 0.07143 0.05844 0.00974
##
```

```
## $density
## [1] 0.03247 0.21429 0.11364 0.07143 0.05844 0.00974
##
## $mids
## [1]  5  7  9 11 13 15
##
## $xname
## [1] "fc"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

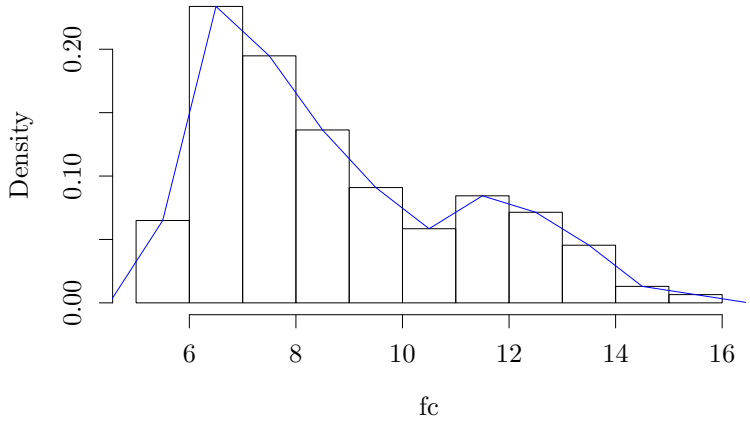


It is easily seen that the number of classes often significantly affects the shape of the histogram. However, there is no golden rule valid for all samples. We often have to plot ("test") several figures and choose the most informative and æsthetic histogram.

Let us draw a *density histogram* (`prob=TRUE`, so the total bar area will be equal to 1) together with a polyline joining the middles of each bar.

```
h <- hist(fc, prob=T, main=NA)

intWidth <- h$breaks[2]-h$breaks[1] # we assume equal lengths
numClass <- length(h$mids) # mids - mid points in each interval
polylineX <- c(h$mids[1]-intWidth, h$mids, h$mids[numClass]+intWidth)
# we need 2 additional segments with and end in y=0
polylineY <- c(0, h$density, 0)
lines(polylineX, polylineY, col="blue") # draw the polyline
```



Note

It can be shown that the area under the polyline is equal to 1. Therefore, it may be used as an estimator of the probability density function. There are also different methods to find the underlying function, e.g. kernel density estimators, briefly described in Sec. 2.5.

Details

A *stem-and-leaf display* is similar to a histogram. It is easy to be drawn on a sheet of paper (or in text-only mode), and therefore was more popular in the past.

```
stem(fc)
```

```
##
##   The decimal point is at the |
##
##     5 | 033345788
##     6 | 000222222334445555555678888999999
##     7 | 000122233333444444566667788999
##     8 | 0123444446666677778889
##     9 | 0133447788899
##    10 | 0012255799
##    11 | 1344556666689
##    12 | 113333679
##    13 | 003444889
##    14 | 35
##    15 | 2
```

We see that the approximate observation values are 5.0, 5.3, 5.3, 5.3, 5.4 etc. Moreover, we have a picture of the data distribution's shape.

Let us change the scale and see how it affects the form of the chart.

```
stem(fc, scale = 2)
```

```
##
##   The decimal point is at the |
##
##    5 | 03334
##    5 | 5788
##    6 | 00022222233444
##    6 | 555555556778888999999
##    7 | 000122233333444444
##    7 | 5566667788999
##    8 | 012344444
##    8 | 6666677778889
##    9 | 013344
##    9 | 7788899
##   10 | 00122
##   10 | 55799
##   11 | 1344
##   11 | 556666689
##   12 | 113333
##   12 | 679
##   13 | 003444
##   13 | 889
##   14 | 3
##   14 | 5
##   15 | 2
```

⊡

Ex. 2.4. Consider the `cars.csv` file once again. Perform a preliminary analysis of fuel consumption (in l/100 km) individually for American, European and Japanese cars. Restrict the data set to cars that cost less than 10000$.

Solution.
Let us create the vector `fc` once again, without removing the missing observations this time. We are going to split it into 3 subvectors, each representing the fuel consumption of cars from different regions.

Vector split

```
fc <- 3.785 * 100/(cars$mpg * 1.609)
cars$origin
```

```
##   [1] 2 1 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 1 3 1 1 3 2 2 2 2 2
##  [36] 3 1 1 1 1 1 1 1 1 1 1 1 1 1 2 3 1 1 2 1 2 1 1 1 3 2 1 1 1 1 2 3 1 3 1
##  [71] 1 1 1 2 3 3 3 3 3 1 3 2 2 2 2 3 2 3 2 3 3 2 1 3 1 1 1 1 1 3 1 3 3 3 3
## [106] 3 1 1 1 2 2 3 3 3 3 2 2 2 3 3 1 1 1 1 1 1 1 1 1 1 1 2 3 3 1 1 3 3 3
## [141] 3 3 3 1 1 1 1 3 1 1 1 2 1 1 1
```

The meaning of category identifiers in `origin` is stored in the variable `carmakers` (artificially added by its creator).

```
head(cars$carmakers)
```

```
## [1] America Europe  Japan
## Levels:         America Europe  Japan
```

We may convert `origin` to a quantitative (factor) type (now it is a numeric vector) and give each category a descriptive label.

```
prod <- factor(cars$origin)
levels(prod) <- c("America", "Europe", "Japan")
head(prod)
```

```
## [1] Europe  America Japan   Japan   Japan   America
## Levels: America Europe Japan
```

```
table(prod)
```

```
## prod
## America  Europe   Japan
##      85      26      44
```

Now we create 3 new vectors, each corresponding to the fuel consumption measurements of cars from different regions worth < 10000\$:

```
fcA <- fc[prod == "America" & cars$price < 10000]
fcE <- fc[prod == "Europe" & cars$price < 10000]
fcJ <- fc[prod == "Japan" & cars$price < 10000]
```

Note that restoring `NA`s was required to split `fc` into subsets; the vectors `fc`, `origin` and `price` must be of the same length. Now (yes, not before the last operation!) they may be removed.

```
fcA <- fcA[!is.na(fcA)]  # we don't need missing values from now...
fcE <- fcE[!is.na(fcE)]  #
fcJ <- fcJ[!is.na(fcJ)]  #

length(fcA) + length(fcE) + length(fcJ)  # how many observations?
```

```
## [1] 152
```

```
sum(cars$price < 10000 & !is.na(fc))  # check
```

```
## [1] 152
```

To determine the values of basic sample statistics, we shall call appropriate functions individually for each subvector.

```
summary(fcA)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    6.03    8.11    9.68    9.85   11.60   15.20
```

```
summary(fcE)
```

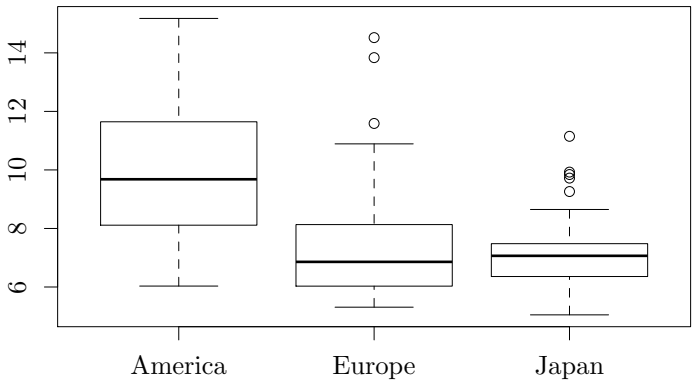
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.31    6.03    6.86    7.74    8.13   14.50
```

```
summary(fcJ)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    5.05    6.36    7.07    7.21    7.46   11.10
```

It may be interesting to compare the results graphically. Three vectors may be passed at the same time to the box-and-whisker plotting routine.

```
boxplot(fcA, fcE, fcJ, names = c("America", "Europe", "Japan"))
```



We conclude that, on the average, the fuel consumption of American cars is greater than of the other vehicles (so this statement conforms to a popular opinion). Moreover, this group is the most dispersed. The European and Japanese samples contain outliers.

Details

Unfortunately, the `histogram()` function does not allow to illustrate all the data at the same time. However, to compare the empirical distributions, the plots may be drawn in one figure (additionally we are interested in results for the whole sample). It can be done by calling `par(mfrow=c(4,1))`; R will "prepare" some space for 4 plots. Each call to a graphical function will place the new chart in the next cell of a "virtual alignment table", one below another. What is more, we should ensure the ranges of each x axis are the same.

```
par(mfrow = c(4, 1))  # 4-in-1
rngeall <- range(fc) + c(-1, 1)  # from min. to max. observation in the whole sample
rngeall
```

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

```
rngeall <- range(na.omit(fc)) + c(-1, 1)  # ah, missing data!
rngeall
```

```
## [1]  4.048 16.177
```

```
# 4 histograms:
hist(fc, ylab = "", xlab = "All cars", xlim = rngeall, main = "Fuel consumption")
```

```
## Error:  figure margins too large
```

```
hist(fcA, ylab = "", xlab = "American cars", xlim = rngeall, main = NA)
```

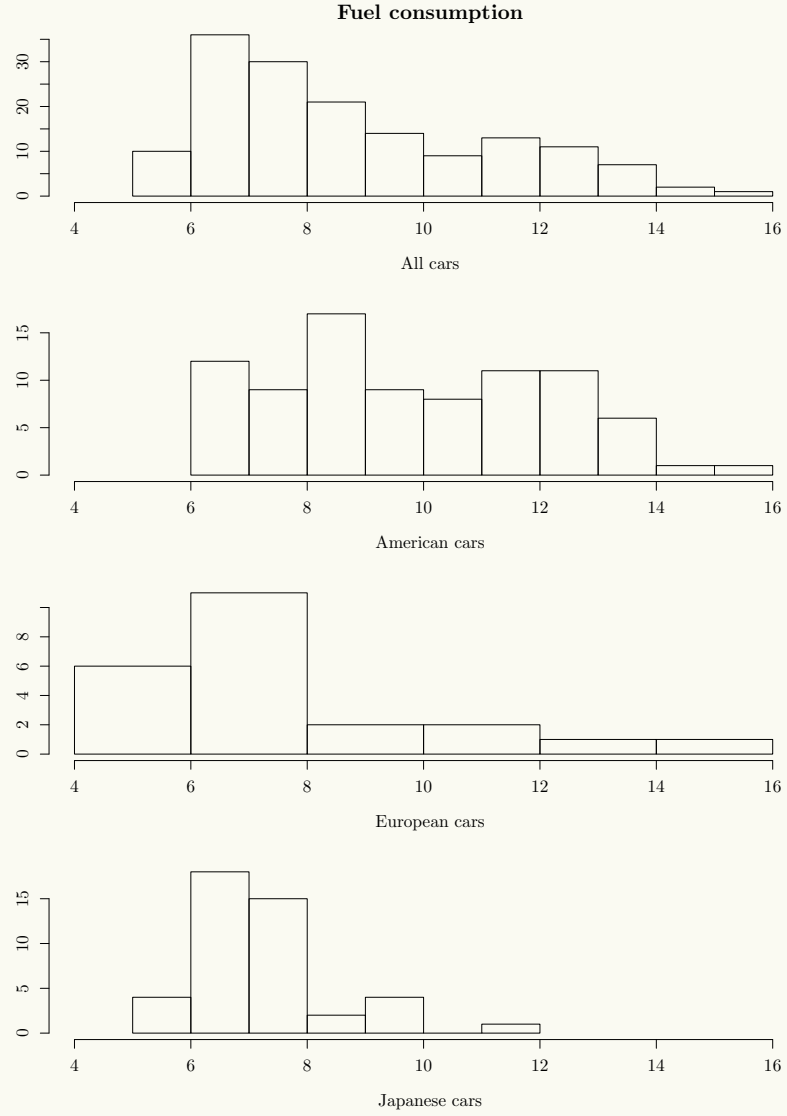
```
## Error:  figure margins too large
```

```
hist(fcE, ylab = "", xlab = "European cars", xlim = rngeall, main = NA)
```

```
## Error:  figure margins too large
```

```
hist(fcJ, ylab = "", xlab = "Japanese cars", xlim = rngeall, main = NA)
```

```
## Error:  figure margins too large
```



The interpretation of the results is left to the reader.

□

2.4. Time series plots

Ex. 2.5. Here are shares records (in PLN) of some company on 20 consecutive days:

23.30, 24.50, 25.30, 25.30, 24.30, 24.80, 25.20, 24.50, 24.60, 24.10,
24.30, 26.10, 23.10, 25.50, 22.60, 24.60, 24.30, 25.40, 25.20, 26.80.

Create a plot of share prices as a function of time.

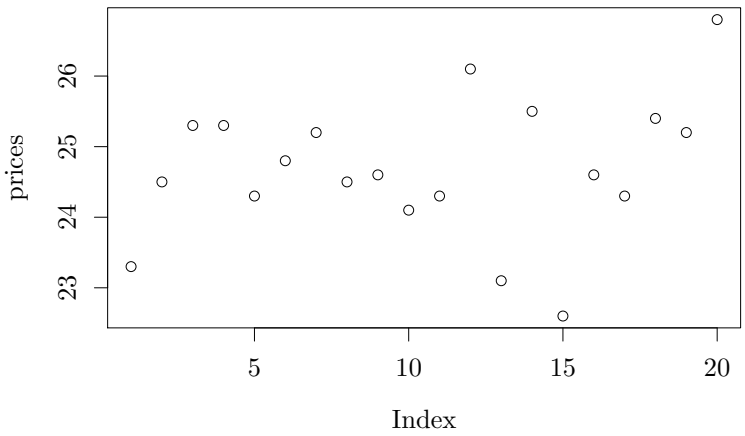
Solution.
First we enter the data:

```
prices <- c(23.3, 24.5, 25.3, 25.3, 24.3, 24.8, 25.2, 24.5, 24.6, 24.1, 24.3,
    26.1, 23.1, 25.5, 22.6, 24.6, 24.3, 25.4, 25.2, 26.8)
```

Time series plot

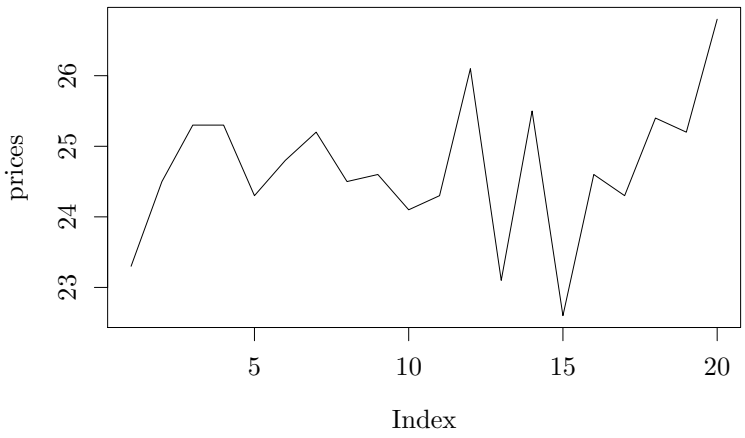
We plot the time series using the `plot()` function.

```
plot(prices)  # equivalent to:
plot(1:20, prices)
```



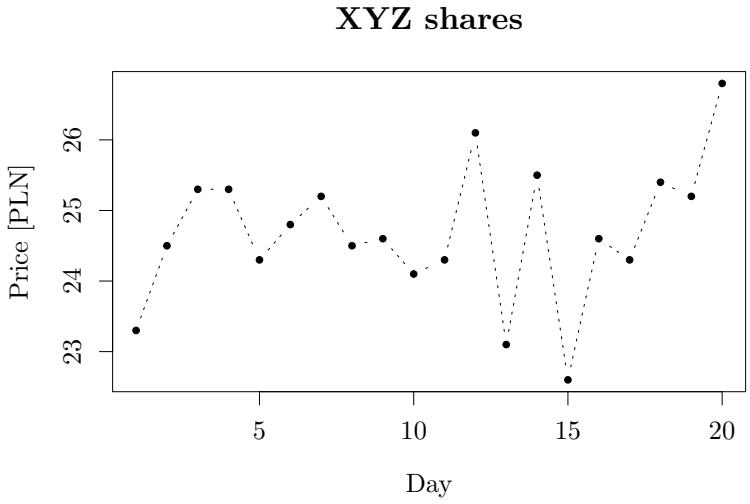
Unfortunately, the plot is quite unreadable. Let us try to join the symbols with lines.

```
plot(prices, type = "l")
```



The piecewise linear function suggests a continuous (and linear!) change of price between the time points. To emphasize a discrete nature of the data, we use the `type="b"` (*both* lines and points) parameter.

```
plot(prices, type="b", pch=20, lty=3,
main="XYZ shares", xlab="Day", ylab="Price [PLN]")
```



Details

The parameters `pch` and `lty` change default graphical settings for drawing symbols and lines. More information may be found in Sec. 2.6.

□

2.5. Kernel density estimators ⋆

We have noticed above that a probability histogram can be used to estimate the density function of the data. However, they often have weak statistical properties. Kernel density estimators are often used in such case.

Definition 1. The *Rosenblatt-Parzen kernel density estimator* for a given sample $\mathbf{X} = (X_1, \ldots, X_n)$ is a function

$$f_{\mathbf{X}}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right), \tag{2.2}$$

where $h > 0$ is a fixed *coefficient of smoothness* (bandwidth) and K — a continuous probability distribution function, called *kernel*, such that

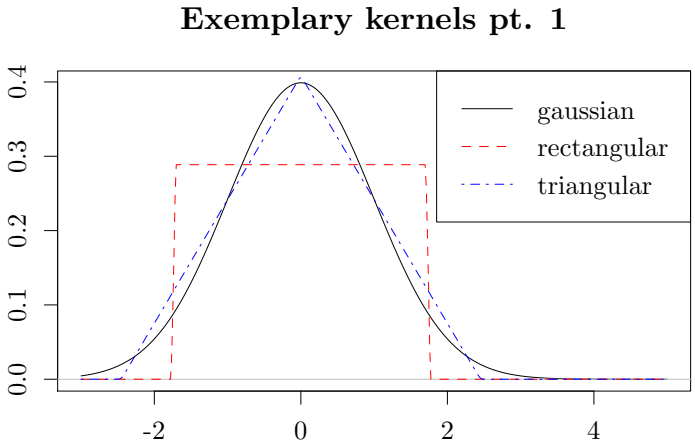
1. $K(x) = K(-x)$ (symmetry), and
2. K has maximum at 0.

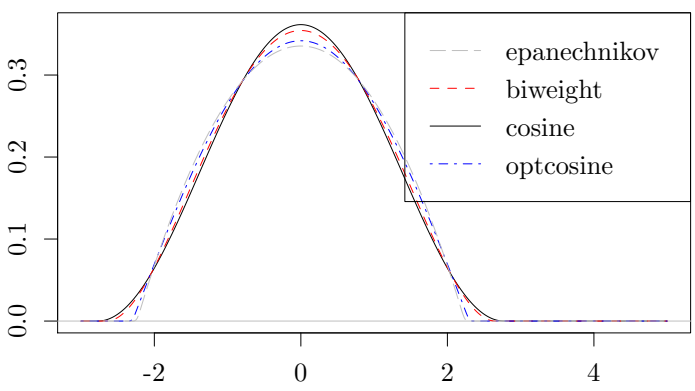
The reader is referred e.g. to [2] for more details on this very interesting topic.

To obtain a kernel density estimator in R we use the `density()` function. The kernel is defined via the `kernel` parameter. Its possible values are: `"gaussian"` (default), `"rectangular"`, `"triangular"`, `"epanechnikov"`, `"biweight"`, `"cosine"`, `"optcosine"` (or `g`, `r`, ... for brevity, respectively).

Available kernels are depicted in the figures below.

```
x <- 0.0;
kde_g <- density(x, bw=1, kernel="g", from=-3, to=5)
kde_r <- density(x, bw=1, kernel="r", from=-3, to=5)
kde_t <- density(x, bw=1, kernel="t", from=-3, to=5)
# etc. ...
plot(kde_g,  col=1, lty=1, main="Exemplary kernels pt. 1",
   xlab=NA, ylab=NA)
lines(kde_r, col=2, lty=2)
lines(kde_t, col=4, lty=4)
# etc. ...
legend("topright", c("gaussian", "rectangular", "triangular"),
   col=c(1,2,4), lty=c(1,2,4))
```

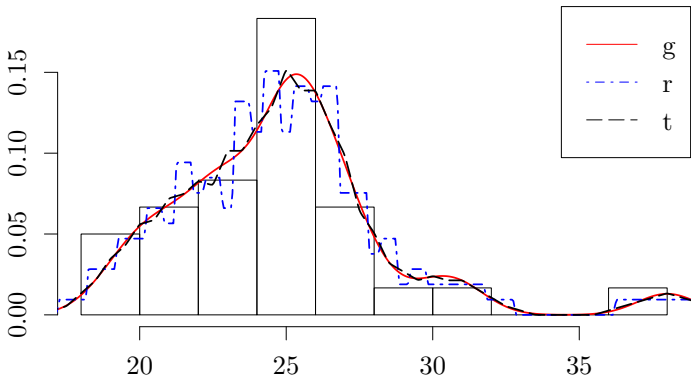




Consider some kernel density estimators for the following data.

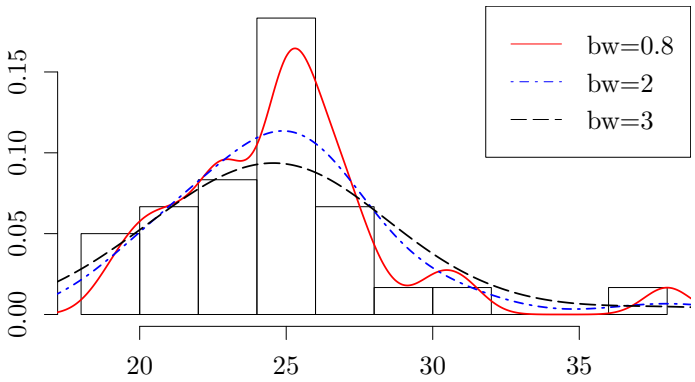
```
x <- c(26, 22, 26, 20, 25, 21, 20, 28, 27, 26, 38, 23, 30, 21, 25, 26, 23, 25,
    27, 27, 25, 22, 23, 31, 19, 25, 25, 23, 25, 24)
```

```
hist(x, prob = T, breaks = 10, main = NA, xlab = NA, ylab = NA)
lines(density(x, kernel = "g"), col = 2, lty = 1, lwd = 2)
lines(density(x, kernel = "r"), col = 4, lty = 4, lwd = 2)
lines(density(x, kernel = "t"), col = 1, lty = 5, lwd = 2)
legend("topright", c("g", "r", "t"), col = c(2, 4, 1), lty = c(1, 4, 5))
```



Moreover, let us examine the effect of bandwidth change.

```
hist(x, prob = T, breaks = 10, main = NA, xlab = NA, ylab = NA)
lines(density(x, bw = 0.8), col = 2, lty = 1, lwd = 2)
lines(density(x, bw = 2), col = 4, lty = 4, lwd = 2)
lines(density(x, bw = 3), col = 1, lty = 5, lwd = 2)
legend("topright", c("bw=0.8", "bw=2", "bw=3"), col = c(2, 4, 1), lty = c(1,
    4, 5))
```



2.6. Commonly used graphical parameters ⋆

We often need to change the appearance of plots produced by R, for example when we are dissatisfied with default line styles, colors, or labels. In this section we will skim through the methods of visual elements adjustment.

Task

The most important graphical parameters are extensively described in the manual, see `?plot.default` or `?par`, and [1; 3; 4].

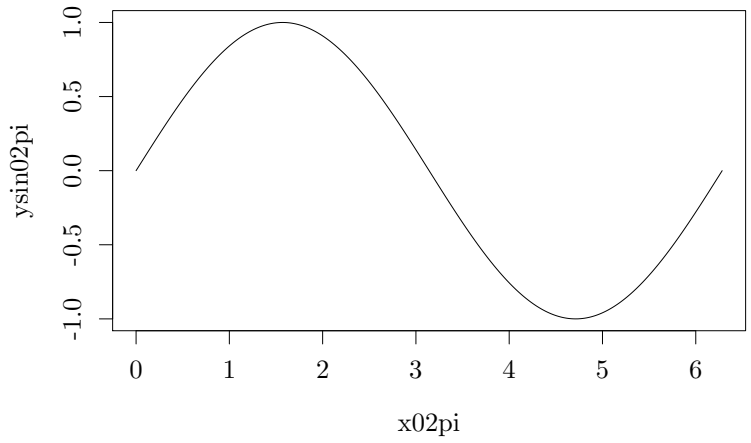
2.6.1. Plot description

Here are some parameters that set the textual description of a plot.

Parameter	Description
`main`	Plot title
`sub`	Plot subtitle
`xlab`	Label under the x axis
`ylab`	Label near the y axis

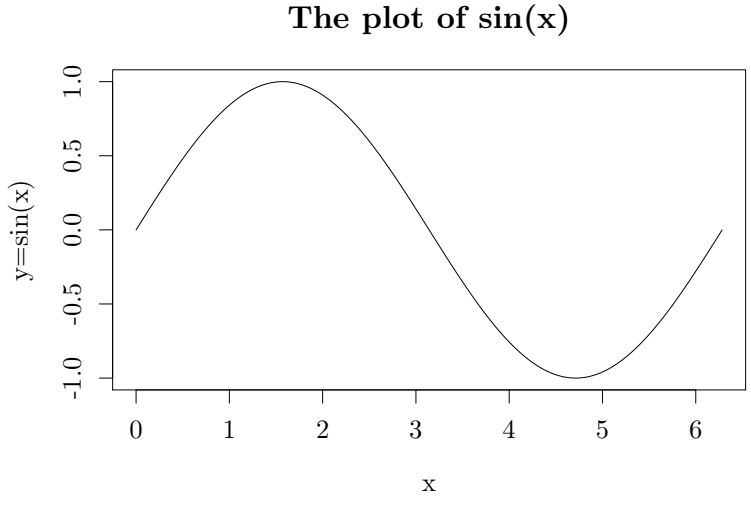
Consider the following examples. Notice how R assigns default labels.

```
x02pi <- seq(0, 2 * pi, length = 1000)
ysin02pi <- sin(x02pi)
plot(x02pi, ysin02pi, type = "l")
```



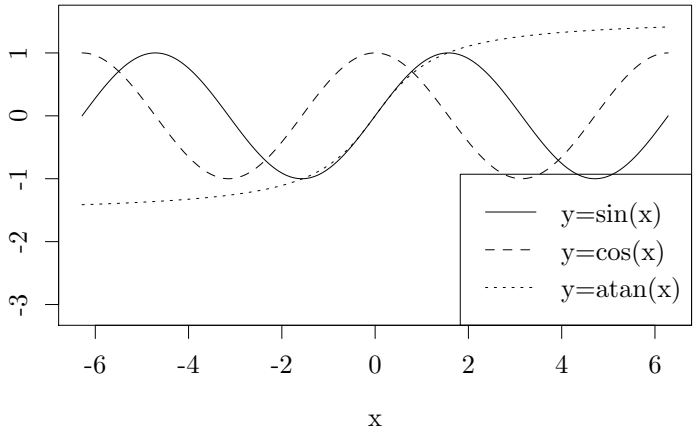
And after some tweaks:

```
plot(x02pi, ysin02pi, type="l",
   main="The plot of sin(x)",
   xlab="x",
   ylab="y=sin(x)")
```



A legend may be added by the `legend()` function.

```
x <- seq(-2*pi, 2*pi, length=1000)
ysin  <- sin(x)
ycos  <- cos(x)
yatan <- atan(x)
plot (x, ysin,  lty=1, type="l", main=NA, xlab="x", ylab=NA, ylim=c(-pi, pi*0.5))
lines(x, ycos,  lty=2)
lines(x, yatan, lty=3)
legend("bottomright", c("y=sin(x)", "y=cos(x)", "y=atan(x)"),
   lty=c(1,2,3))
```



2.6.2. Colors

Many graphical functions allow us to produce colored figures. For example, for a greater readability we may sometimes want to create plots of multiple functions, each with a different color.

Individual colors may be accessed via descriptive identifiers (see Sec. 2.6.2.1). Moreover, if there are many elements to be drawn, color palettes may be used (Sec. 2.6.2.2).

Task

Refer to `?col2rgb`, `?rgb`, `?hsv`, `?hcl`, `?colors`.

2.6.2.1. Color names

Each color value may be determined by a character string of the form `"#rrggbbaa"` (red, green, blue, and alpha ("transparency") component as an 8bit hexadecimal value). However, most often we will use predefined color names. They are available via the `colors()` function.

```
colornames <- colors()
length(colornames)
## [1] 657
colornames[1:20]  # the first 20 names
```

```
##  [1] "white"          "aliceblue"      "antiquewhite"   "antiquewhite1"
##  [5] "antiquewhite2"  "antiquewhite3"  "antiquewhite4"  "aquamarine"
##  [9] "aquamarine1"    "aquamarine2"    "aquamarine3"    "aquamarine4"
## [13] "azure"          "azure1"         "azure2"         "azure3"
## [17] "azure4"         "beige"          "bisque"         "bisque1"
```

2.6.2.2. Color palettes

There are also some predefined color palettes available in R, e.g. `rainbow`, `heat.colors`, `topo.colors`, `terrain.colors`, `cm.colors`. Each such function takes the number of different colors to be generated as a parameter.

```
rainbow(3)
```

```
## [1] "#FF0000FF" "#00FF00FF" "#0000FFFF"
```

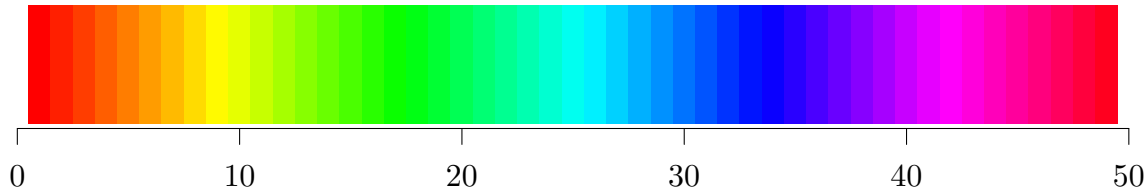
Let us take look at some of the available palettes. The following function will be used to generate their previews.

```
palette <- function(pal)
{
  n <- length(pal); # the number of colors in each pallete
  i <- 1:n;
  d <- 1.0/6.0;

  par(mar=c(2,0,0,0), oma=c(0,0,0,0)); # no margin
  plot(NA, NA, yaxt="n", frame.plot=F, ylab=NA, main=NA, xlab=NA,
    ylim=c(0.0, 1.0), xlim=c(1, n));            # init
  rect(i-0.5, 0, i+0.5, 1, col=pal, border=NA); # draw n rectangles
}
```

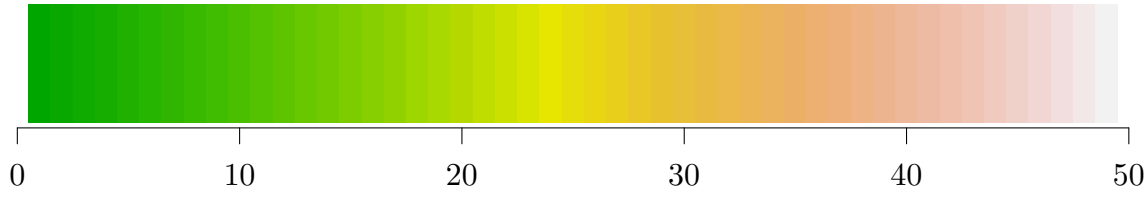
Below we show the previews of 5 palettes consisting of 49 colors.

```
palette(rainbow(49))
```

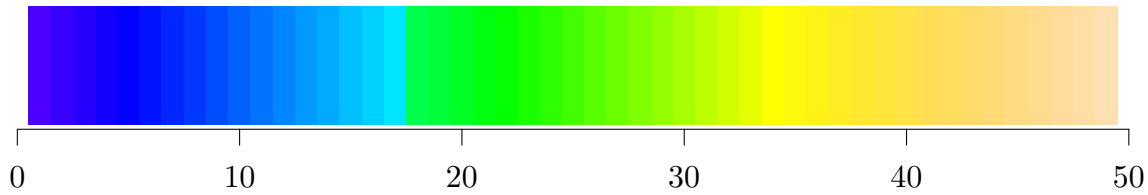



```
palette(heat.colors(49))
```

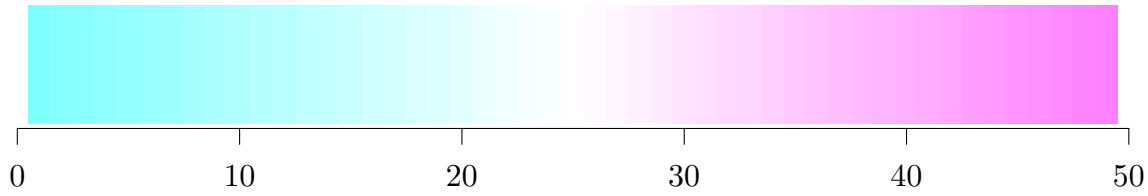

```
palette(terrain.colors(49))
```



```
palette(topo.colors(49))
```




```
palette(cm.colors(49))
```

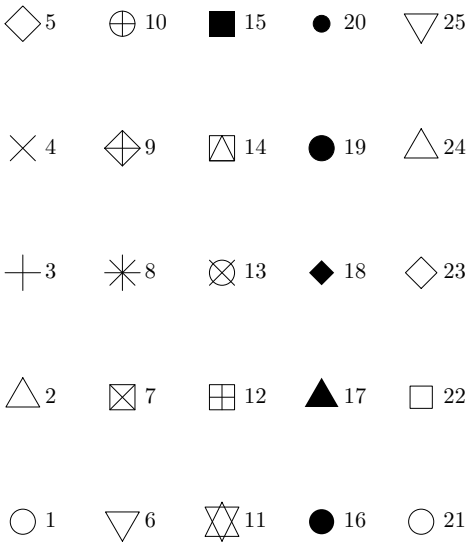


2.6.3. Symbols

The `pch` parameter may be used to change the default *plotting character*. Symbol scaling is performed with the `cex` parameter.

The list of available plotting symbols may be generated as follows.

```
plot(rep(1:5, each = 5), rep(1:5, 5), pch = 1:25, cex = 3, type = "p", main = NA,
    xlab = NA, ylab = NA, frame.plot = F, xaxt = "n", yaxt = "n", xlim = c(1,
        6))
text(rep(1:5, each = 5), rep(1:5, 5), 1:25, pos = 4, offset = 1)
```



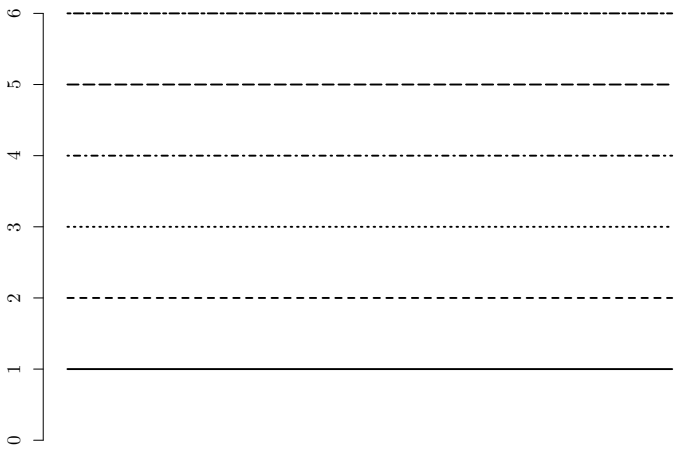
2.6.4. Line types

The `lty` parameter (*line type*) has 7 values: `0`, `1`,..., `6`) or, equivalently, `"blank"` (plot nothing), `"solid"` (default), `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"` and `"twodash"`.

The `lwd` parameter changes the *line width*.

```
plot(NA, NA,          # init
  main=NA, xlab=NA, ylab=NA, frame.plot=F, xaxt="n",
  ylim=c(0,6), xlim=c(0.95, 1.05));
```

```
lines(c(0.95, 1.05), c(0, 0), lty=0, lwd=3);
lines(c(0.95, 1.05), c(1, 1), lty=1, lwd=3);
lines(c(0.95, 1.05), c(2, 2), lty=2, lwd=3);
lines(c(0.95, 1.05), c(3, 3), lty=3, lwd=3);
lines(c(0.95, 1.05), c(4, 4), lty=4, lwd=3);
lines(c(0.95, 1.05), c(5, 5), lty=5, lwd=3);
lines(c(0.95, 1.05), c(6, 6), lty=6, lwd=3);
```



Bibliography

[1] M.J. Crawley. *The R Book*. Wiley, 2007.

[2] J. Koronacki and J. Ćwik. *Statystyczne systemy uczące się*. WNT, 2005.

[3] H.V. Mittal. *R Graphs Cookbook*. Packt Publishing, 2011.

[4] P. Murrell. *R Graphics*. Chapman & Hall/CRC, 2006.