Practical 2 – part 2

Review the following text, open (if needed, install) the *R* and try the commands from the text (generate some data, try to find some descriptive statistics and make some charts).

1. GENERAL INTRODUCTION TO *R*

Foreword

At the very beginning of computerized data analysis era people started to create small dedicated programs, each of which was able to solve one or a few data analysis problems. As the number of these small programs increased they were collected into statistical packages --- in such a way statistical packages like SAS or SPSS were created. In such package solving some particular task is easy and efficient but to complete an entire data analysis project is still rather tricky – because the integration between these separate programs in the collection is weak and each small program was designed keeping in mind only one task at time.

Some statisticians (Richard A. Becker, John M. Chambers and others) got very frustrated with such software and they published in 1984 an article describing how ideal statistical software should look like. It took some time before there appeared useful computer programs which tried to fulfil their dream. At present there are two of them: S-plus and R. They are rather similar – if one knows how to use R then he/she can also use S-plus. However, there exist minor differences: R is free, S-plus costs quite a lot of money; S-plus handles large datasets in more efficient way etc.

As a surprise to some computer users R does not have small buttons to which to click to get your analysis done. Instead one has to give commands and write a program to do the data analysis. Why? There are a few reasons. First, you can always show your program to others interested in your work and they can understand how did you do your analysis, check the analysis for errors or simply repeat everything exactly as you did it. In science it is important that others can repeat your analysis and can see how you come to your conclusions. At second, it simply is a faster way to get the analysis done. The first time you do the analysis it can be quite time-consuming (especially if you do not yet know the commands), however during a typical real-life data-analysis project one usually has to reanalyze the data ten or more times (additional observation became available during the work, data-entry errors will be discovered as work progresses, referees require additional analysis or transformations etc). It is not easy to repeat a thousand-click analysis for ten times without making mistakes especially if the time between the reanalyses can be measured in months – you just might not remember one checkbox you had to select to get correct answers. But if you have a program, you just modify one command and run it again to get all the result desired.

Installing R

R is freeware program available for Windows, Linux, MacOS X and for other platforms. You can download R for Windows by going into the R-homepage (http:\\www.r-project.org), select CRAN (under Download) -> select a mirror site near you -> Windows -> base -> R-2.8.1-win32.exe. After downloading the executable you should run it to install R to your computer.

In certain platforms (Windows XP, for example) the cruel administrator of the computer might have denied the users the right to install new programs to the computer. If this is the case you might encounter a few error messages at the end of the installation process – R is not able to put the icons into the start menu/desktop. Do not worry – you should still be able to run the program: just search the computer for the file Rgui.exe and run it – R should work without problems. Nevertheless, it would be polite to ask someone in charge in the computer class to install the new software.

The standard installation requires approximately 50-60 Mb of disk space, together with some additional add-ins and contributed packages it can easily take 600Mb or more on your hard disk.

References

If you want additional materials or books about *R*, here are a few recommendations for beginners:

1. *An Introduction to R* (Available from http:\\www.r-project.org -> *Manuals* -> *An Introduction to R*). A useful starting point to *R* could be to try out the sample session (given in Appendix A).

2. John Maindonald. *Using R for Data Analysis and Graphics - Introduction, Codes and Commentary.* Available from http://cran.r-project.org/doc/contrib/usingR.pdf.

3. Peter Dalgaard. *Introductory Statistics with R*. Springer, 2002. ISBN 0-387-95475-9.

Remark: you can find book 2 and other great online texts/books on *R*-homepage under the selection *Documentation / Other -> contributed documentation* (http://cran.r-project.org/other-docs.html).

There are also books in other languages.

Eesti keeles on Internetist leitav hulk vähem või rohkem põhjalikke õpetusi, millest esialgu võiks ehk silmad üle lasta ja huvi korral ühtteist kirjeldatust proovida Krista Fischeri ja Märt Mölsi konspektidest (järgnev ingliskeelne juhend hõlmad suuresti neis konspektides õpetatut).

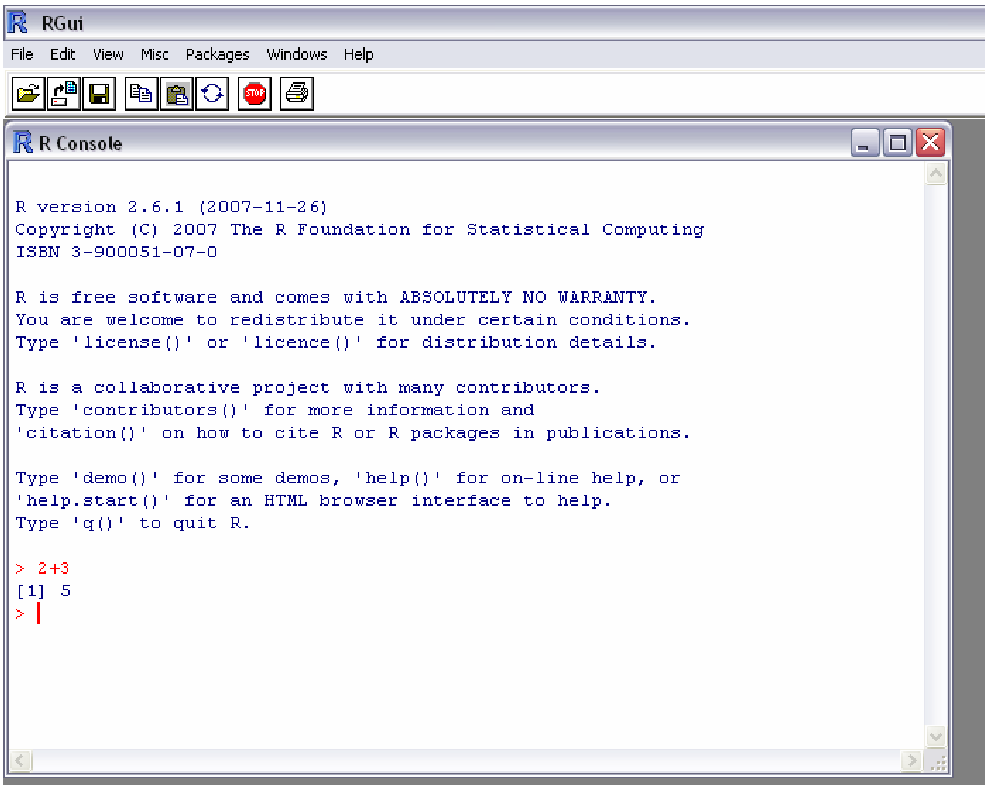
1. Märt Möls (TÜ, matemaatiline statistika), „Sissejuhatus R-i“ – http://www.ms.ut.ee/mart/biomeetria2007/Praks1_SissejuhatusR.pdf.

2. Allan Sims ja Andres Kiviste (EMÜ, metsandus), „Statistiline analüüs R keskkonnas metsanduslike näidetega“, 2006, 72 lk – http://www.eau.ee/~allsi/Rkonspekt.pdf

3. Ott Toomet (TÜ, majandus) – http://www.obs.ee/~siim/Rintro.pdf

1.1 Getting started

After starting *R* you should see the command prompt „>“, showing that *R* is waiting for your commands. You can simply type `2+3` after the command prompt and press the *Enter* key. You should get the answer to your question:



```
R version 2.6.1 (2007-11-26)
Copyright (C) 2007 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 2+3
[1] 5
> |
```

Therefore it is possible to conclude: *R* can be used for doing simple calculations. You can try the following calculations, for example (try all of them out by yourself as well as the examples that follow through the text):

> `2**8` - calculate 28, equivalently you can enter 2^8 to get the same answer;

> `sin(0.5*pi)` - standard functions like sin, cos, log, exp etc are available;

> `(2+3)*4` - use brackets, if needed;

> `sqrt(9)` - square root of 9;

> `sqrt(1:10)` - ....

Notice that one has to use a dot (.) to separate decimal places in numbers, not comma (,).

If needed, intermediate results could be saved, and used later as needed. Try it out!

```
> x=2
> y=sqrt(2)
```

To see the saved value, just type the name of the object:

```
> x
[1] 2
> y
[1] 1.414214
```

And you can use them in later calculations:

```
> (4+x)/2
> y**2
> z = (0:10)+ x
```

If you want to recall the names you have used to save the results, just type `ls()`:

```
> ls()
[1] "z" "x" "y"
```

If you want to remove one of them, type `rm`(<name of unwanted object>):

```
> rm(y)
> ls()
[1] "z" "x"
```

If you want to delete them all, use the command `rm(list=ls())`:

```
> rm(list=ls())
> ls()
character(0)
```

Please notice, that *R* is case sensitive, e.g. object „`x`“ is different from object „`X`“:

```
> x=2
> X=3
> x+X
[1] 5
> SQRT(4)
Error: could not find function "SQRT"
> sqrt(4)
[1] 2
```

Remark: In addition to „=“-sign one can use „<-“ to assign a value to the object:

```
> x<-32
> x+2
[1] 34
```

Weird values in *R*: you may encounter or use values like `Inf`, `-Inf`, `NA`, `NaN` during your calculations:

```
> 1/0
[1] Inf
> -1/0
[1] -Inf
> Inf+2
[1] Inf
> log(0)
[1] -Inf
> mean(c(2,3,NA))
[1] NA
> 0/0
[1] NaN
```

As you might have quessed, `Inf` is an infinitely big number, `NA` – missing (Not Available) value, `NaN` – Not A Number;

Sometimes *R* uses scientific notation:

```
> 1/10000
[1] 1e-04
> 10**6
[1] 1e+06
```

Finding help

You can ask for additional information about a function by using `?` or `help()`:

```
> ?mean
> help("if")
```

If you are unsure about the name of a command, but you still remember part of the name, you might try the function `apropos()`:

```
> apropos("cos")
> apropos("test")
```

If you do not know the name of the function, then you might try to select from the menu

Help -> HTML Help

Common mistakes

For every bracket that starts „`(`“ there must be a corresponding closing bracket „`)`“, for every starting apostroph (`"`) there must be a corresponding closing one. In *R* one command can be splitted into multiple rows. However, sometimes we do unintentionally – for example sometimes we forget to close the apostrophe in preceding line. In this case the prompt changes to + sign and a perfectly correct command may be added to the one with a mistake. *R* interprets them together and finds, that the combined command has mistakes, even though the last command you gave was correct.

To cancel last command (which had mistakes in it) you can just press *Escape*-key and normal prompt > should reappear.

One can recall last command issued by pressing the up arrow.

Writing programs

Often it is wise to write commands in a separate window – one can use notepad or the integrated Script editor. You can then run your program or selected commands by either making a copy-and-paste from the text editor (notepad), or by selecting some rows in Script editor and pressing *Ctrl+R*. You can document the work you are doing in such way. One can and should add comments to the program. Every line beginning with the symbol # is ignored by the *R* and hence can contain the comments suitable for humans.

Documenting and saving your work

- You can save all the objects you have created in *R* by choosing from the menu
  File-> Save Workspace.
- Of course one can restore all the objects creted during a previous session by loading the *Workspace* one desires.
- All commands issued during a session can be saved to a text file by selecting
  File -> Save History.
- The file you created via „*Save History*“ can be opened and edited using notepad, for example.
- *File -> Save to File*
  saves all the commands together with the results given to a text file (everything you see on the window named „*R Console*“ will be saved).

1.2 Acquiring Data

To do Data Analysis one needs data. For meaningful results the dataset should be entered or imported correctly into the program one intends to use. In this section we will investigate some possibilities to get the data to *R*.

Entering Small amounts of Data within the program

If you need to enter only a few values (at the moment), it might be good idea to do it within your program. A few examples will follow:

```
> explevel=c(987, 604, 802, 340, 560, 660)
> explevel
> mean(explevel)
> summary(explevel)

> genotype=c("AA","Aa","AA","aa","Aa","Aa")
> table(genotype)
> barplot(table(genotype))

> mydata=data.frame(genotype, explevel)
> mydata
```

Eventhough one can enter as large datasets as one wishes in such a way, it usually is viable only in for tiny amounts of data to be eneterd in such a way.

Browsing and modifying the dataset

To print the dataset named *mydata* one has to simply ask for it:

```
> mydata
```

or, for just a few (3) observations, one can issue a following command:

```
> mydata[1:3, ]
```

To see the dataset in an *Excel*-like environment one may issue a command

```
> edit(mydata)
```

Eventhough you seem to be able to modify the data values after giving the command `edit(mydata)`, you actually cannot change the actual dataset. If you want to modify the underlying dataset you could issue the following command:

```
> mydata2=edit(mydata)
```

After issuing such a command all modifications you do in the edit window will be saved to a new dataset called *mydata2*. The old, unmodified, dataset *mydata* remains unchanged.

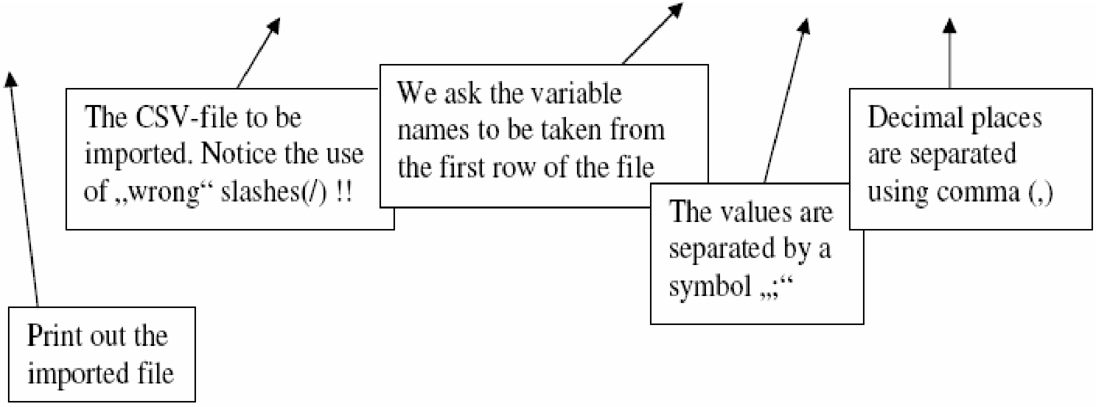
Importing data from Excel / from text (CSV) file

Often data is entered in a spreadsheet-style program like *Excel* or *Calc* (*OpenOffice*). There are many different ways to import data from *Excel*, but maybe the easiest and most reliable way is to save the spreadsheet as a csv-file – choose from menu *Save As*, and change the *save as type* to CSV (comma delimited).

Unfortunately *Excel* may create very different CSV-files depending on the computer parameters (CSV-files created tend to depend on country they are created). Therefore, in the first time we make such a conversion, we should take a quick look at the file created by *Excel*. What separates the variables or values from each other? Which symbol is used to separate decimal places in a number?

To read an CSV-file to *R* we may type the following command:

```
imported=read.csv(“C:/myfiles/FromExcel.CSV“,header=TRUE,sep=“;“,dec=“,“)
imported
```



The CSV-file to be imported. Notice the use of „wrong“ slashes(/) !!

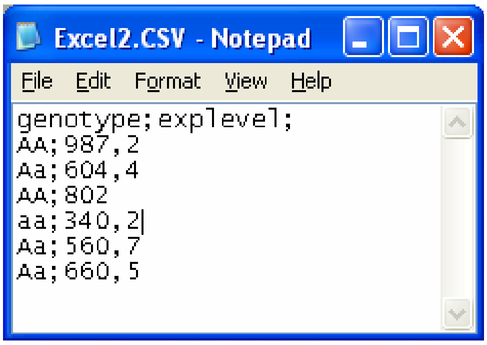
We ask the variable names to be taken from the first row of the file

The values are separated by a symbol „;“

Decimal places are separated using comma (,)

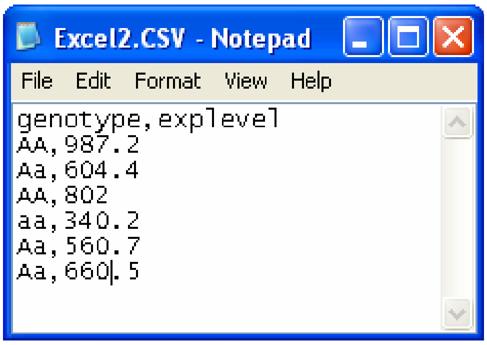
Print out the imported file

Lets see a few examples of CSV-files (as opened in text editor) and the corresponding `read.csv` commands



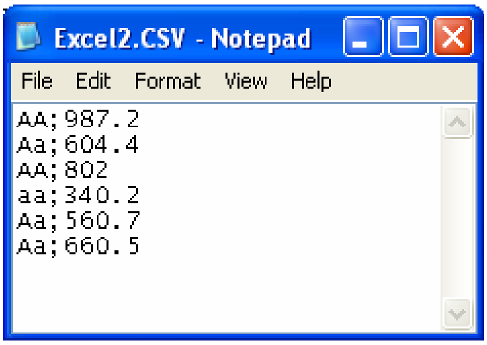
```
genotype;explevel;
AA;987,2
Aa;604,4
AA;802
aa;340,2
Aa;560,7
Aa;660,5
```

```
dataset=read.csv(“C:/temp/Excel2.CSV“,
header=TRUE, sep=“;“, dec=“,“)
```



```
genotype,explevel
AA,987.2
Aa,604.4
AA,802
aa,340.2
Aa,560.7
Aa,660.5
```

```
dataset=read.csv(“C:/temp/Excel2.CSV“,
header=TRUE, sep=“,“, dec=“.“)
```



```
AA;987.2
Aa;604.4
AA;802
aa;340.2
Aa;560.7
Aa;660.5
```

```
dataset=read.csv("C:/temp/Excel2.CSV",
header=FALSE, sep=";", dec=".")
```

For more detailed questions look at the section *R Data Import/Export* in your *HTML Help* file. Topics covered in the help file include: importing data from other statistical packages like *SPSS*; picking up data from Relational databases like *Oracle* and *MySQL*; how to read *dbf*-files etc.

Generating data

Sometimes the data needed can be easily generated. For example one can generate consecutive numbers (like observation number) using the following command:

```
> 1:25
> ObsNr=1:45
> ObsNr
```

Trickier sequences can be generated using functions like `seq` and `rep`:

```
> seq(0,10, by=2)
> seq(0,100, length=8)
> rep(1, 5)
> rep(c(1,10), 5)
> rep(c(1,10), c(3,5))
```

Sometimes it is useful to generate just random data. Use functions like `runif` and `rnorm` for this:

```
> x=rnorm(100)
> y=1+2*x+rnorm(100)
> plot(x,y)
```

How to add the variables you generate to an existing dataset? The already familiar function `data.frame` can be used. If you still have the previously generated dataset `mydata`, you can use the following commands to add observation number to the dataset:

```
> Obs=1:6
> mydata=data.frame(mydata, Obs)
> mydata
```

Saving and loading *R* datasets

For saving a (specific) dataset one can use the `save` command. Instead of (/) one can also use (\\) when giving the file path:

```
> save(genedata, file="C:\\temp\\genes")
```

To retrieve the dataset use the `load`-function:

```
> load("C:\\temp\\genes")
```

1.3 Referring, searching and subsetting data

At the beginning we might want to create a small dataset called *genedata*:

```
> genedata=data.frame(phenotype=c(120,134,140,146,200),
+ gene=c("A","A","B","B","B"))
> genedata
phenotype gene
1 120 A
2 134 A
3 140 B
4 146 B
5 200 B
```

Now few examples:

`genedata[1:3, ]` – first 3 observations

`genedata[3, 1]` – observation 3, the value of 2. variable

`genedata$gene` – all genotype values

`genedata$gene[2:3]` – genotype values for observations 2 to 3

It can be laborious to write the dataset name in front of the variable name. However, it is possible to say to *R*, that from now on we work with a particular dataset. Then we can refer to the variables by using only the name of the variable. This can be done by using the function `attach()`:

```
> genedata$phenotype
[1] 120 134 140 146 200
> phenotype
Error: object "phenotype" not found
> attach(genedata)
> phenotype
[1] 120 134 140 146 200
```

How to select observations corresponding to the genotype „A“?

```
> genedata[gene == "A",]
phenotype gene
1 120 A
2 134 A
```

How to select only phenotype data corresponding to genotype “A“?

```
> phenotype[gene == "A"]
[1] 120 134
```

How to select genotypes corresponding to below-average phenotypes?

```
> gene[phenotype<mean(phenotype)]
[1] A A B B
Levels: A B
```

One can use the selected observations as a starting point for further analysis. For example the mean or average phenotype values for genotype „A“ can be calculated:

```
> mean(phenotype[gene=="A"])
[1] 127
```

Sometimes the values in a variable have to be transformed before the analysis. Sometimes it is useful to change units, sometimes to make data more „normal“, sometimes because other reasons. How one can make such a transformation? We shall see a few options.

`lnfeno=log(phenotype)` – make a new variable called *lnfeno* containing logarithmic phenotype values. Notice that this new variable is not a part of *genedata* dataset. You may add it to the dataset by using the function `data.frame()`.

`genedata$logfeno=log(phenotype)` – a new variable *logfeno* is added to the *genedata* dataset. You have to reattach the dataset to use the short name for the variable:

```
> genedata$logfeno=log(phenotype)
> genedata
phenotype gene logfeno
1 120 A 4.787492
2 134 A 4.897840
3 140 B 4.941642
4 146 B 4.983607
5 200 B 5.298317
```

```
> genedata$logfeno
[1] 4.787492 4.897840 4.941642 4.983607 5.298317
> logfeno
Error: object "logfeno" not found
> detach(genedata)
> attach(genedata)
> logfeno
[1] 4.787492 4.897840 4.941642 4.983607 5.298317
```

A few useful commands:

`dim(genedata)` – finds the dimensions (number of observations and variables) of a dataset

`length(fenotype)` – finds the number of observations in a vector (variable)

`names(genedata)` – variable names

`summary(genedata)` – brief summary for each variable

Pay Attention!

1. If we pick subsets from a dataset (data frame), we always have to have a comma (,) within the square brackets:

```
dataset[<selected observations>, <selected variables>]
```

If we subset from a data vector (variable), then we do not have to specify the variable any more and comma is not needed:

```
variable[<observations to select>]
```

2. Square brackets are used only for selecting observations from a dataset/variable. They must immediately follow the dataset/variable name.

3. To check equality double equal signs (== instead of =) have to be used! Single (=) is used to assign a new value to an object (and the old value will be deleted)!

2 DESCRIPTIVE STATISTICS WITH *R*

Before starting with basic concepts of data analysis, one should be aware of different types of data and ways to organize data in computer files.

2.1 Some basic terms

Population – an aggregate of subjects (creatures, things, cases and so on). For a given study, a *target population* has to be specified: on which subjects we'll generalize or use the results?

Sample – collection of subjects *in the study*. In general, a sample should be representative for the target population.

Observation – a study unit or *subject* or an individual. Often a human being, also an animal, plant or anything else.

Variable – quality or quantity, measured or recorded for each subject in the sample (age, sex, height, weight, smoking level, etc.).

Dataset – a set of values of all variables of interest for all individuals in the study. The numeric results obtained from the dataset will be used to draw conclusions about the target population.

2.2 Organization of data

A dataset is mostly organized (and stored as a computer file) in a form of a data matrix.

A data matrix representing sex (1-male; 0-female), age, no. of children, weight (kg), and height (cm) of 7 individuals:

NO	SEX	AGE	NO OF CHILDREN	WEIGHT	HEIGHT
1.	0	57	1	65	158
2.	1	70	3	100	175
3.	0	45	0	71	162
4.	0	38	2	58	164
5.	0	25	1	81	170
6.	1	50	4	68	172
7.	1	61	0	85	179

Each row of such a matrix represents one observation. All rows have the same length: the same data has been recorded for all individuals.

Each column represents one *variable*.

For instance, WEIGHT is the name of a variable, representing the body weight (in kg) of an individual.

2.3 Types of data

- **Numeric data**

  – **Discrete data** – the variable can take only integer values (0, 1, 2 etc.)

  examples: number of children, number of friends

  – **Continuous data** – any real-numbered values (often within a certain range) are possible

  examples: body weight, age

- **Qualitative (non-numeric, categorical) data**

  – **Nominal data**: unordered categories

  examples: blood group, eye color

  – **Ordinal or ordered data**: ordered categories

  examples: smoking level, attitudes (good-moderate-bad)

Numeric coding of nominal or ordered data does not make the data numeric!

2.4 Summarizing/presenting data

Mean

The sample **mean** is the arithmetic average of the data.

It can be calculated, by summing all of the data values and dividing the sum by the total sample size.

Example:

Data: 1 3 5 2 9

Mean: (1 + 3 + 5 + 2 + 9)/5 = 20/5 = 4

Mathematically: for a variable X, mean is often denoted as $\overline{x}$ and calculated as:

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \ldots + x_n}{n},$$

where x_1, x_2, . . . , x_n denote observations of a variable and n is the number of observations in the sample.

***R*:**

```
> x = c(1,3,5,2,9)
> mean(x)
> [1] 4
```

If there are missing values:

```
> x = c(1,3,5,2,9,NA,7,10)
> mean(x)
[1] NA
> mean(x, na.rm=T)
[1] 5.285714
```

Median

Sometimes it is of interest to sort values of a variable in ascending or descending order. The order number of an observation in such a row is called as *rank*.

Median is the middle point of ordered data – either the middle observation (if the number of observations is odd) or the average of the two middle observations (if the number of observations is even).

Example:

Body heights of 11 individuals (in cm):

155, 160, 171, 182, 162, 153, 190, 167, 168, 165, 191.

Ordered data:

153 155 160 162 165 167 168 170 171 182 191

median: 167

***R*:**

```
> x = c(155, 160, 171, 182, 162, 153, 190, 167, 168, 165, 191)
> median(x)
[1] 167
> x = c(155, 160, 171, 182, 162, 153, 190, 167, 168, 165, 191, 175)
> # added 1 observation
> median(x)
[1] 167.5
```

The advantage, but sometimes also the disadvantage of the median is, that it is not affected by extreme values in the data. It does not matter, how small or how big are the values that are larger or smaller than the median.

Neither the mean nor the median provides sufficient information about the data: one should also know about variability.

Standard deviation (*SD*, *s*) is a quantity that reflects the variability of the sample. One could interpret *SD* as the approximate mean distance from the mean.

More precisely, *SD* is defined as the square root of the variance (s^2) (sum of squared differences from the mean divided by sample size minus 1; the latter called as the sample variance, s^2),

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})^2 ,$$

$$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2} \,.$$

Similarly to the mean, standard deviation is sensitive to the extremes in the data.

***R*:**

```
> x=c(1,4,5,7,8,11)
> mean(x)
[1] 6
> median(x)
[1] 6
> var(x) # variance
[1] 12
> sd(x) # standard deviation
[1] 3.464102
> x=c(1,4,5,7,8,110) # change the last observation from 11 to 110
> mean(x)
[1] 22.5
> median(x)
[1] 6
> var(x)
[1] 1843.5
> sd(x)
[1] 42.936
```

A more robust approach is to divide the distribution of the (ordered) data into four, and find the points below which are 25%, 50% and 75% of the distribution.

These are known as **quartiles** (the median is the second quartile).

Example: a sample:

6 9 9 10 9 10 3 12 7 6 6 4 8 8 3 8 6 4 11 11

The ordered sample

3 3 4 4 6 6 6 6 7 8 8 8 9 9 9 9 10 11 11 12

The ordered sample divided to 4 parts:

3 3 4 4 6 | 6 6 6 7 8 | 8 8 9 9 9 | 9 10 11 11 12

Quartiles: the cutpoints: 6, 8 (the median) and 9.

***R*:**

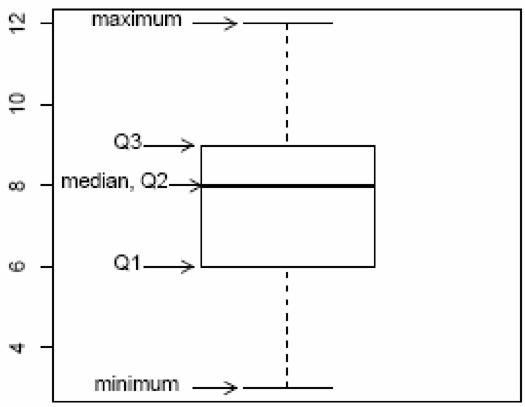
In *R*, you can use function `quantile` to get median and quartiles, or you can also use the function `summary`, to get also the mean:

```
> z=c(3, 3, 4, 4, 6, 6, 6, 6, 7, 8, 8, 8, 9, 9, 9, 9, 10, 11, 11, 12)
> summary(z)
Min. 1st Qu. Median Mean 3rd Qu. Max.
3.00   6.00   8.00  7.45   9.00  12.00
```

The variation of the data can be summarized in the **interquartile range** (*IQR*), the distance between the first and third quartile (here: $IQR = 9 - 6 = 3$).

In general *p*th **percentile** is the *p*%-cutpoint of ordered data (from smallest to largest). Sometimes in official statistics, deciles are used – 10th, 20th, etc percentiles.

A **boxplot** is a graphical representation of median and quartiles:

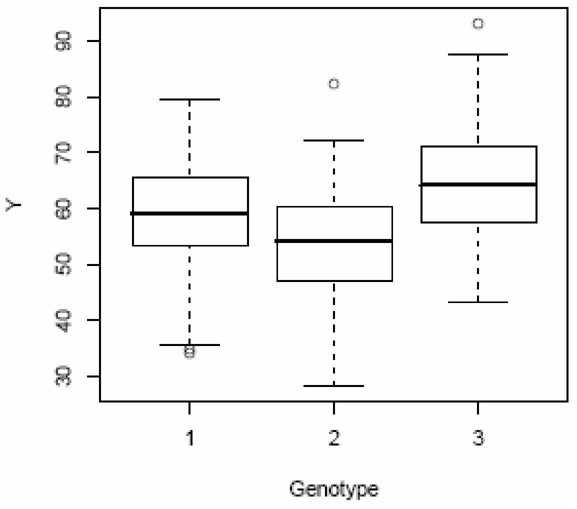


***R*:**

```
> boxplot(x)
```

Boxplot gives an overview of the **distribution** of the data. It is often used to compare data across different groups.

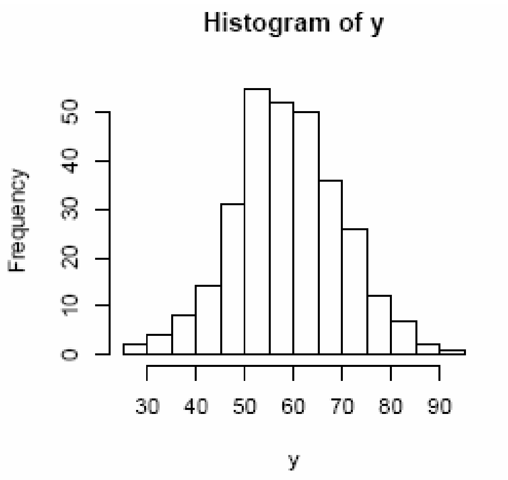
```
> boxplot(Y~g)  # g is a categorical variable with values 1, 2 and 3
(genotype)
```



Another way to look at the distribution, is to plot a histogram of the data. To obtain a histogram, the scale of the variable is divided to consecutive intervals of equal length and the number of observations in each interval is counted.

***R*:**

```
> hist(Y)
```



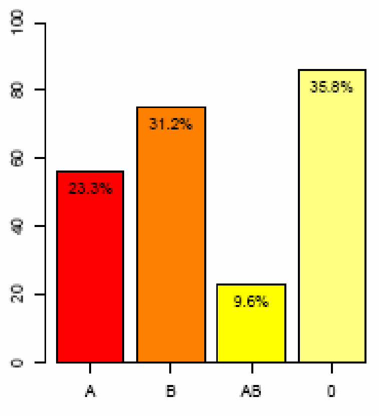
For nominal data, mean and standard deviation do not make much sense; neither do median and percentiles. To see the distribution of the data, look at the frequency table.

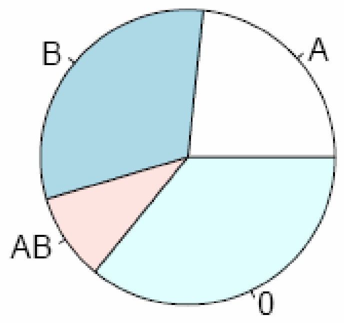
Example:

Blood groups of 240 individuals:

Blood group	n	%
A	56	23.3%
B	75	31.2%
AB	23	9.6%
0	86	35.8%

. . . and its graphical representation – bar chart or pie chart:





Note: pie charts are misleading and should be avoided!

***R*:**

```
> eyecol=c(1,2,1,2,2,2,3,3,1,4,2,2,2,3,1,4,3,2,1,1,1) # reading in data
> table(eyecol) # a simple frequency table
eyecol
1 2 3 4
8 7 4 2

> eyecol=factor(eyecol, labels=c("blue","grey","brown","green"))
> # created a factor variable
> table(eyecol) # a more meaningful table
eyecol
blue grey brown green
8 7 4 2

> prop.table(table(eyecol)) # relative frequencies
eyecol
blue grey brown green
0.3809524 0.3333333 0.1904762 0.0952381

> round(100*prop.table(table(eyecol)),1)
> # percentages, rounded to 1 decimal place
eyecol
blue grey brown green
38.1 33.3 19.0 9.5

> barplot(table(eyecol)) # a simple bar chart
> barplot(table(eyecol),col=c("blue","grey","brown","green"),main="Eye
color")
> # a nicer one
```

Some notes on statistical graphics:

- A graph should communicate useful information more efficiently than any other (numeric) summaries
- A good graph should have a good information to ink ratio – avoid fancy details, that do not add information, but make the graph more complicated (larger, more colorful).
- Pay attention to the scale of the graph!

2.5 *R*: Descriptive statistics by groups

Suppose you would like to compare means or other descriptive statistics in different subgroups of your sample. In *R*, you can use the function `tapply` for that. This function takes 3 arguments: the numeric variable, a categorical grouping variable and the function to apply.

To understand, how it works, try the following examples:

```
weight = c(56, 67, 65, 78, 49, 87, 55, 63, 70, 72, 79, 52, 60, 78, 90)
sex = c(1,1,1,2,1,2,1,1,1,2,1,1,1,2,2)
tapply(weight,sex,mean)
tapply(weight,sex,summary)
```