Biological Data Analysis – part I
(Distance learning)

The purpose of the course “Biological Data Analysis” is to introduce the basic methods needed for the analysis of datasets encountered in the field of bioinformatics. Nowadays, data analysis is mostly done with the help of computers.

We have selected the R software as it is one of the most popular statistical software packages in bioinformatics, having excellent capabilities for unlimited number of analysis tasks and - the teachers themselves like it and use it every day. Luckily R is freeware, so one can just download it and start using it. However, it is also a tool one has to learn before making it work. The main aim of distance learning part of this course is to teach you the basics of R, so we could concentrate more on learning and understanding data analysis methods during the face-to-face meeting.

The materials for distance learning have been divided into 6 parts; each part ends with a short list of exercises. Solving these exercises is highly recommended but not absolutely mandatory; however you have to complete the final mini-analysis project given at the end of the course materials. The dataset and instructions for the mini-project will be e-mailed to you before the end of the week (by Nov 25th).

The first encounter with R can be tricky; hence do not hesitate to ask for help, if needed. There exists a list for course participants to discuss R and matters related to the course – if you send an e-mail to the address BDA@lists.ut.ee your e-mail will be forwarded to all participants of the course. We will try to provide some assistance as soon as possible. You can also contact teachers directly.

Wishing you a successful start of the course!

lecturers:

Märt Möls,	Mart.Mols@ut.ee
Krista Fischer,	Krista.Fischer@ut.ee
Lars Snipen	

assistants:

Anne Selart,	anne.selart@ut.ee
Egils Stalidzans,	
Anu Roos.	anu.roos@ut.ee

Part I
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 (95 and later) -> base -> R-2.4.0-win32.exe. After downloading the executable you should run it to install R to your computer.

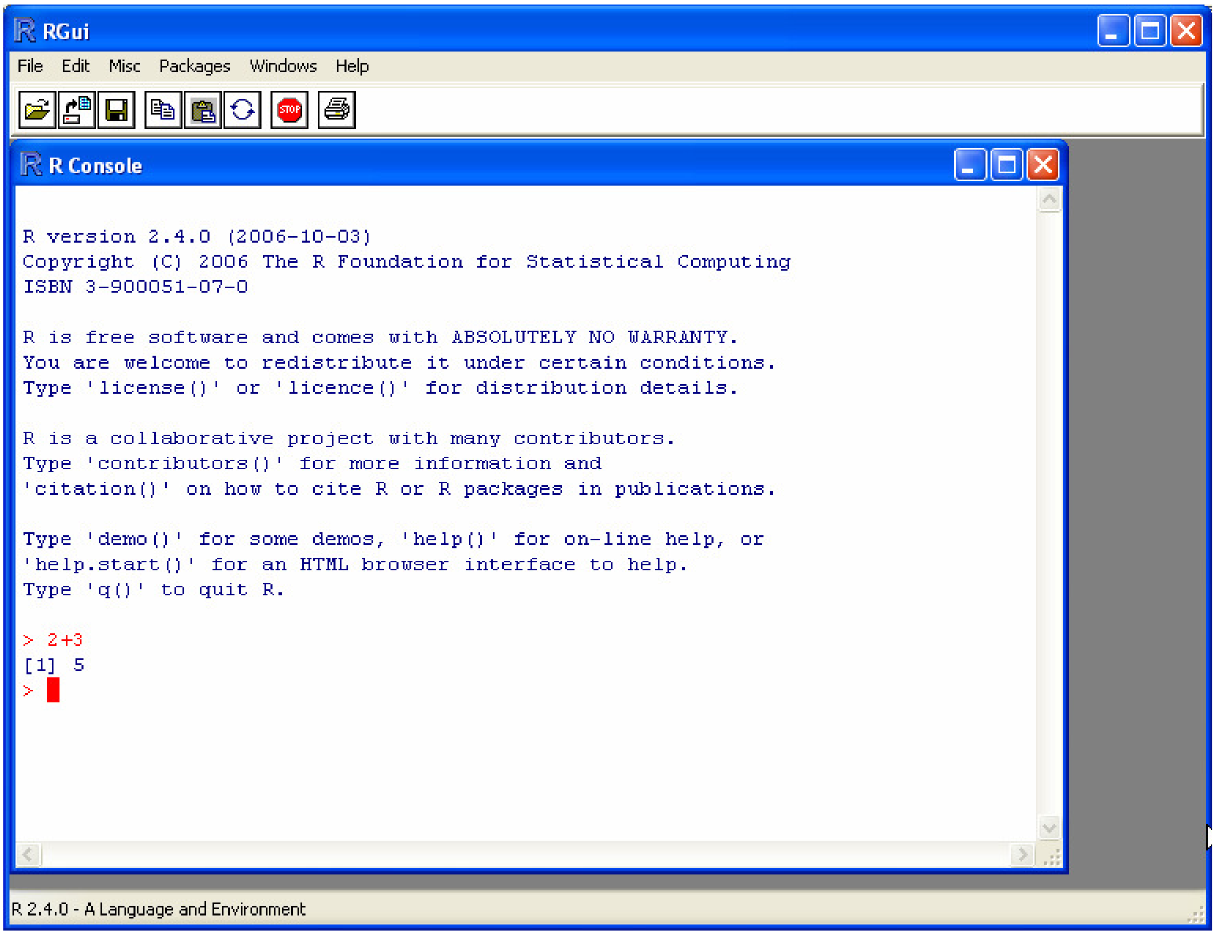
In certain platforms (Windows XP, for example) the cruel administrator of the computer might have denied the users the right to install new programs to the computer. If this is the case you might encounter a few error messages at the end of the installation process – R is not able to put the icons into 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 recuires approximately 50-60 Mb of disk space, together with some aditional add-ins and contributed packages it can easily take 600Mb or more on your harddisk.

References
If you want additional materials or books about R, here are a few reccommendations 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, Examples and Commentary*
   Available from http://cran.r-project.org/doc/contrib/usingR-2.pdf. Remark: you can find this book and other great online texts/books on R-homepage under the selection (Documentation) Other -> contributed documents.
3. Peter Dalgaard. *Introductory Statistics with R*. Springer, 2002. ISBN 0-387-95475-9.

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:


RGui
File Edit Misc Packages Windows Help
R Console
R 2.4.0 - A Language and Environment

```
R version 2.4.0 (2006-10-03)
Copyright (C) 2006 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 throught the text):

`> 2**8`	- calculate 2^8, 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 infinitly 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
For more options and information.

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 apostroph in preceeding line. In this case the prompt changes to + sign and a perfectly correct command may be added to the one with a mistake. R interpretates 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 selectig 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).
If you encounter a problem and ask for help via e-mail it is strongly reccommended that you include the file produced by „Save to File" with your e-mail. If you do so, tutors will see the command issued before encountering the trouble and we could spot more easily the mistake.

Exercises I

1. Install R to your computer.
2. Calculate the value of following expressions using R:

   a) $\frac{1}{1+0{,}00022}-(1-0{,}00022)$ = ?

   b) $1{,}79451+1{,}79451^{1{,}79451}+\sin(1{,}79451)$ = ?
3. Find out what does the function *pie()*, and try out at least one example of it.

You can find the answers to selected exercises at the end of the document.

Part II
Acuiring 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)
```

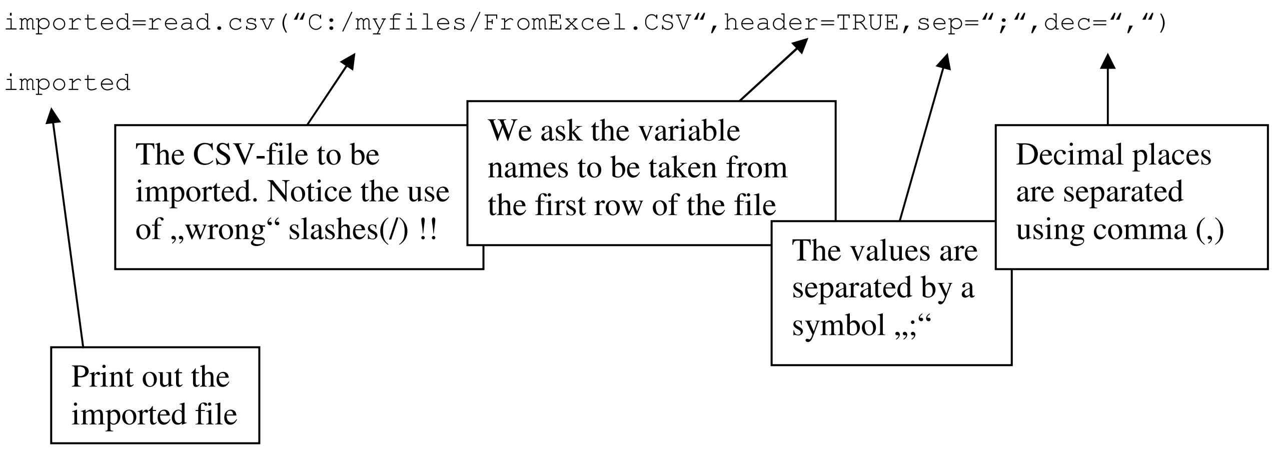
After issuing such a command all modifications you do in the edit window will be saved to a new dataset called *mydata2*. The old, unmodified, dataset *mydata* remains unchanged.

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(/) !!

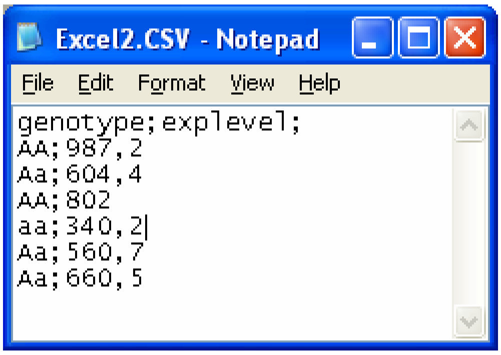
We ask the variable names to be taken from the first row of the file

The values are separated by a symbol „;“

Decimal places are separated using comma (,)

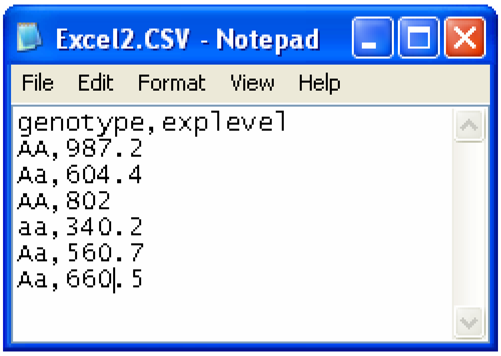
Print out the imported file

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



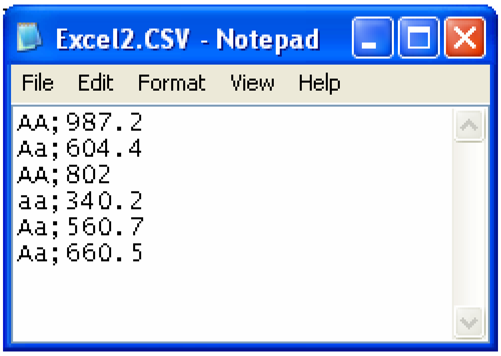
```
genotype;explevel;
AA;987,2
Aa;604,4
AA;802
aa;340,2
Aa;560,7
Aa;660,5
```

```
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")
```

Exercises II

1. In the experiment first 10 observations correspond to intensity level 10; next 100 observations correspond to intensity level 5 and last 40 observations correspond to intensity level 2. Generate a variable named Intensity, in which the intensity levels for all 150 observations would be recorded.

2. Command

```
> data(CO2)
```

will create an example dataset called CO2. How do you print first ten observations from that dataset?

Part III
Referring, searching and subsetting data

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

```
> genedata=data.frame(fenotype=c(120,134,140,146,200),
+    gene=c("A","A","B","B","B"))
> genedata
  fenotype 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 labourous to write the dataset name infront 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$fenotype
[1] 120 134 140 146 200
> fenotype
Error: object "fenotype" not found
> attach(genedata)
> fenotype
[1] 120 134 140 146 200
```

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

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

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

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

How to select genotypes corresponding to below-average fenotypes?

```
> gene[fenotype<mean(fenotype)]
[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 fenotype values for genotype „A“ can be calculated:

```
> mean(fenotype[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(fenotype)` - make a new variable called *lnfeno* containing logarithmed *fenotype* 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(fenotype)` - 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$logfeno=log(fenotype)
> genedata
  fenotype gene  logfeno
1      120    A 4.787492
2      134    A 4.897840
3      140    B 4.941642
4      146    B 4.983607
5      200    B 5.298317
> genedata$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)!

Exercises III

1. How to print only fenotype values which are bigger than 145?
2. Create a sample dataset CO2 by issuing command *data(CO2).* two types of Treatment have been used – „chilled“ and „nonchilled“. What is the average value of variable *conc* for chilled objects? (For more information about the dataset type *?CO2*)
3. Find the observation numbers for objects with genotype „A“.
   Hint: You may first create a variable holding observation numbers, before proceeding!

Solutions to Exercises

Exercises I

2. a.

```
> 1/(1+0.00022)-(1-0.00022)
[1] 4.838935e-08
```

Hence the answer is 0,00000004838935

2. b.

```
> a=1.79451
> a+a**a+sin(a)
 [1] 5.625263
```

3.

Function *pie()* draws a pie-chart, one possible example could look like following:
pie(c(20,30,25), labels=c("Biology","Biological Data\n Analysis", "Bioinformatics"))

Exercises II

1.

```
> rep(c(10,5,2),c(10,100,40))
```

2.

```
> CO2[1:10,]
```

Exercises III

1.

```
> fenotype[fenotype>145]
[1] 146 200
```

2.

```
> data(CO2)
> attach(CO2)
> mean(conc[Treatment=="chilled"])
[1] 435
```

3.

```
> Obs=1:5
> Obs[gene=="A"]
[1] 1 2

Alternative solution:
> (1:5)[gene=="A"]
[1] 1 2
```