

Biostatistics and experimental design

Lecturer: Prof. dr. Ann Vanreusel, dr. Freija Hauquier

Practical Exercises: Freija Hauquier, Lara Macheriotou, Elise Toussaint

Contents

1. Excel.....	3
1.1. Basics.....	3
1.2. Exercise: Data analysis of hyperbenthos data.....	5
2. Introduction to R.....	6
2.1. Installing R.....	6
2.2. R-interface.....	6
2.3 R as a calculator.....	7
2.4 Objects, vectors, matrices and data frames.....	8
2.4.1. Objects.....	8
2.4.2. Vectors.....	8
2.4.3 Matrices and lists.....	11
2.4.4. Data frames.....	12
2.5 Creating graphs in R.....	17
2.5.1 The x-y plot.....	17
2.5.2 Bar graphs.....	20
2.5.3 The package “Lattice”.....	22
2.6. Scripts.....	22
2.7 Close R.....	23
2.8 Exercises.....	23
3 Descriptive statistics.....	25
4 Formulas for statistical analysis.....	26
5 ANOVA.....	27
5.1 Assumptions for ANOVA.....	27
5.1.1 Normality of the data.....	28
5.1.2 Homogeneity of variances.....	31
5.2 t-test.....	31
5.3 F-test.....	31
5.4 ANOVA.....	31
5.4.1 1-way ANOVA.....	32
5.4.2 2-way ANOVA.....	32
5.5 Non-parametric tests.....	33
<i>Name=function(arg1,arg2,...) expr1.....</i>	<i>30</i>

5.6 Post-hoc tests	33
6 Correlation and regression	35
6.1 Correlation.....	35
6.1.1 Test for normality	35
6.1.2 Parametric correlation: Pearson product moment	35
6.1.3 Non-parametric correlation: Spearman rank.....	35
6.2 Regression	37
6.2.1 Simple regression	37
6.2.2 Multiple regression.....	38
6.4 Tasks	43
6.4.1 Glucose experiment.....	43
6.4.2 Analysis of the hyperbenthos data in 3 European estuaries	43

1. Excel

1.1. Basics

Excel is a spreadsheet program that allows you to order, to process and to save numerical and text data in rows (1,2,3, ...), columns (A, B, C, ...) and sheets (sheet 1, sheet 2, ...). Each unit (number, formula, text) is placed in a cell which is indicated by a row number and a column letter, and if more than 1 sheet, by a sheet label. In this exercise we will learn the basic principles of Excel.

- Start by filling in the following dataset in the first worksheet. Put the species in rows and the stations in columns. Species names and station labels are filled in respectively on the first row and first column.

	station1	station2	station3	station4	station5	station6	station7	station8	station9
species1	0	0	1	1	6	60	5	1	0
species2	0	0	1	20	4	40	100	1	1
species3	0	0	1	1	2	20	1	0	0
species4	0	0	1	1	6	60	0	0	0
species5	1	2	1	1	4	40	0	0	0
species6	0	0	1	1	8	80	0	0	0
species7	1	2	1	1	1	10	2	0	0
species8	120	2	1	1	8	80	10	1	0

- Calculate at bottom of the data matrix the total amount of organisms per station (Σ function in toolbar or type: **=SUM(B2:B9)**)
- **FORMULAS** can be filled in the cell where one wants to see the result. A formula is always preceded by =. Each new operation within the formula, is delineated by brackets. In this way rather complex calculations can be done in one movement. Some basic operations are: sum, - subtraction, / division, * multiplication, ^ exponent.
- A formula can be copied from 1 cell to all other possible adjacent cells by holding the square in the right corner at the bottom of the cell (a 'handler'). When the cursor changes into a small cross you can drag the formula to the other cells you wish. You can also simply **copy and paste** the formula. Take into account that the formulas that you are using are relative in the sense that each time you drag the content of the cell, the rows and columns are adapted e.g. *=sum (a1:a20) becomes =sum(b1:b20) when this formula is dragged from cell a21 to cell b21*
- If this effect is not wanted, you can place a dollar sign (\$) in front of the row or column code. In that case the row and/or columns are fixed such that one can work with a particular cell content when dragging. Try this out.
- **Sort** species from most abundant (dominant) over all stations to least abundant. Make sure that the correct species names stay in place. Therefore you have to calculate in column L the total density per species over all stations. Highlight all filled cells with exceptions of the station labels. Select Data – Sort-Sort by column L, Descending.
- Calculate per species the following variables (use **fx FUNCTION** in the toolbar. Clicking **ALL** shows all possible functions that can be calculated directly in Excel). Below are some commonly used functions:
 - The mean density (AVERAGE)
 - The standard deviation (STDEV) and the variance (VAR)
 - The number of stations where a species is present (COUNTIF >0)
 - The number of times a species is NOT present (FREQUENCY, bin array = 0).

- In order to calculate a **DATA DISTRIBUTION** (histogram), the “Data Analysis Toolpack” should be installed. This toolpack is available if you see the subtab “Data Analysis” under the “Data” tab. If this is not the case, you need to load the “Data Analysis Toolpack”: <https://support.office.com/en-us/article/Load-the-Analysis-ToolPak-6a63e598-cd6d-42e3-9317-6b40ba1a66b4>
In order to calculate a data distribution (e.g. the number of values between 1 and 100, between 101-200, ...), enter the values 0, 20, 40, 60, 80, 100 and 120 somewhere in the data sheet. Select “DATA – DATA ANALYSIS – HISTOGRAM”. Use “input range” to select the data that should be analysed, “bin range” is used to select the class borders. Using “Output range”, you can select the area in the data sheet where the output will be generated. A graph output can be generated, including a cumulative data distribution.
- Many other types of graphs can be constructed. Try out some graphs by first highlighting the data that you want to show in the graph, and next by choosing the selected graph. Consider carefully for each graph, what is exactly visualized and if the graph is appropriate for your data. How for instance would you visualize the relation between the abundance of species 1 and species 2 (from the data matrix we could already see that both species are related in appearance). *Take into account that continuous data can be connected, but this is not allowed for independent data points.*
- Insert a sheet, and rename each **sheet** by double clicking the worksheet label. Name the original worksheet for instance ‘raw data’. Copy and paste the first row and the first column to the 2nd worksheet. Transfer the original data matrix to percentages (in relation to the total number of organisms per station), so that you can see which species are dominant in each station. Calculate again the sum per station (has to be 1 or 100). Change **formulas** to **real values** (copy-paste special – values).
- Copy the complete data matrix (total densities included) from sheet1 to a third worksheet and **transform** the raw data in **log (base 10)** values. If there are zero values in the data, do not forget to add 1 to the raw data. Show the values up to three decimals. Transfer the formula to real values. Transpose **row and columns** (copy-paste special – transpose). Never put the transposed matrix on top of the original matrix. Rename the work sheet (e.g. ‘log data’).
- Calculate on the first work sheet the Shannon-Wiener diversity index for each sample/station:
 - $H' = - \sum p (\ln p)$ with p = density per species/total density
 - Show the obtained results in a graph in order to illustrate the change in diversity over the different stations. Compare with number of species (=COUNTIF >0) and the dominance per station.

Some useful tips

- **Freeze panes** (Click Window in menu bar): allows you to freeze row and/or column labels when you go through a large data matrix. In case of the example data matrix: when you highlight cell b2, and next you click freeze panes, you can scroll down to column Z or row 50 and still see the respectively station and species names
- **The undo function** allows you to cancel the last (up to more than 10, depending on their complexity) operations.
- Although Windows offers a typical mouse-driven environment, it can sometimes be useful to use shortcuts. Some are mentioned in the pull-down menus of the menu bar (all shortcuts are shown in the help-menu). Some often used combinations are:
 - Ctrl-C or Ctrl-Insert (Copy)
 - Ctrl-V or Shift-Insert (Paste)
 - End-arrows (jumps to the last filled in cell (if a filled cell is highlighted), or to the next filled cell if an empty cell is highlighted.
 - Shift-End+arrows (idem as above but highlight all cells)
 - Ctrl-Home (goes to Cel A1)
 - Ctrl-PgUp and PgDown (changes worksheets)

1.2. Exercise: Data analysis of hyperbenthos data

The **hyperbenthos** includes animals that live just above the surface in marine and brackish water ecosystems. The hyperbenthos can be sampled quantitatively (number of animals per surface area) with a hyperbenthic sledge. The hyperbenthos includes mysids, amphipods, isopods, cumaceans, pycnogonids and chaetognaths (permanent hyperbenthos) as well as (post)larval stages of shrimp, crabs and fish (temporary hyperbenthos). On Minerva you will find the file "HYPERBENTHOS.xls".

	A	B	C	D	E	F	G	H	I
1		w31	w30a	w30b	w30c	w28	w27	w25a	w25
2	Sapieleg	1	11	0	0	0	0	0	0
3	Gastspjn	11	49	133	117	26	1	7	52
4	Schlispir	21	54	92	12	7	0	2	10
5	Schlievry	109	11	264	47	135	1	3	70
6	Mesoslab	81	1598	178	383	367	134	456	148
7	Neominte	0	0	0	0	0	0	0	0
8	Praufftex	0	0	0	0	1	0	0	4
9	Euryptic	0	0	0	0	0	0	0	1
10	Idotine	4	81	9	13	4	3	4	6
11	SyniSpec	0	0	0	0	0	0	0	0
12	Sphanugi	0	0	0	0	0	0	0	0
13	Sphaserr	0	0	0	0	0	0	0	0
14	CymoSpec	0	0	0	0	0	0	0	0
15	DaphSpec	0	0	0	0	0	0	0	0
16	Caprine	0	4	0	1	0	0	0	0
17	CapriSpec	0	0	0	0	0	0	0	0
18	Parityji	0	1	2	0	0	0	0	0
19	Gammcrin	3	103	6	7	7	0	0	4
20	Gammisail	0	0	21	0	0	0	0	0
21	Gammzadd	0	0	0	0	0	0	0	0
22	Gammjueub	0	0	0	0	0	0	0	0
23	Gammlicu	0	0	0	0	0	0	0	0
24	Melipalm	0	0	0	0	0	0	0	0
25	AtMscam	4	30	0	3	5	1	0	0

	A	B	C	D	E	F	G	H
1	stations	Saliniteit	1/Secchi diepte	Temperatuur				
2	w31	31	0.01	20				
3	w30a	30	0.01	20				
4	w30b	29	0.01	20				
5	w30c	29	0.01	20				
6	w28	28	0.01	20				
7	w27	27	0.01	20				
8	w25a	26	0.01	20				
9	w25b	25	0.01	20				
10	w21	21	0.01	20				
11	w19	19	0.01	21				
12	w17	17	0.01	21				
13	w12	12	0.03	22				
14	w10	10	0.02	23				
15	w8	8	0.02	23				
16	g26a	26	0.00	21				
17	g26b	26	0.00	21				
18	g24	24	0.00	22				
19	g20	20	0.01	22				
20	g18	18	0.03	23				
21	g14	14	0.03	23				

This file includes two worksheets: one with the environmental data (salinity, temperature and secchi-depth), and a second sheet including species abundances per station.

In total, 41 stations were sampled in August 1991: 14 in the Westerschelde, 15 in the Gironde, and 12 in the Eems. In each of these estuaries sampling was done along a salinity gradient from polyhaline to oligohaline environment. The labels of the samples reflect the estuary (first letter) and the salinity. In case multiple samples were taken in the same salinity region, these are indicated by small letter (a, b, c, ...). For example sample W30c was taken in the Westerschelde in a salinity region of 30 psu.

The aim of this exercise is to explore this dataset with Excel. Solve the following exercises and post your results and explanation in a Word file on Minerva before the next practical.

1. Examine how the total densities of hyperbenthos change over the salinity gradient for each estuary. Visualise the results in a graph. Compare the results between the estuaries and discuss the results.
2. What are the 10 dominant taxa in the samples of the Westerschelde at 25 PSU? What are their abundances? Make a graph showing the cumulative frequency distribution for these 10 taxa.
3. Compare the biodiversity between samples of the Westerschelde and the Eems at 0-10 psu and at 20-30 psu. Compare both salinity zones between the two estuaries in terms of mean biodiversity (e.g. Shannon-Wiener index) and variance (e.g. standard deviation). Make a graph and discuss the results.
4. Make a frequency distribution graph of density data of the Westerschelde over the geometric classes (0, 1-75, 76-150, 151-300, 301-600, 601-1200, etc.).
5. Examine whether the densities of the mysid *Mesopodopsis slabberi* (Mesoslab) is related to one of the environmental variables. Make graphs and discuss the results.

Post your results and discussion in a Word file on Minerva before the next practical. Use the Minerva dropbox and send to Freija Hauquier.

2. Introduction to R

There is a wide array of statistical software packages (SAS, Statistica, S-Plus, R, MATLAB...). These packages are slightly different, but they do offer the most important statistical routines. For this course, we choose to use **R**. The main reason for this choice is that R is free open-source software. There is a large group of R-users developing packages that are implemented in the R software environment. Hence, a lot of statistical techniques can be applied through one or more of these packages and this allows the user to perform a wide range of statistics (regression, anova, discriminant analysis, ordination, ...) in a single software package. Since programming in R is relatively straightforward, advanced statistics can be applied within R.

A possible disadvantage of R is the lack of a GUI (Graphical User Interface). R users need to type commands in the command line. The command syntax can be found in the associated help files. GUIs for R are available on the internet, but we will not use these during this course. Using commands is not too difficult once you get acquainted with R.

An advantage of using R is the fact that a large community of R users is very active on internet fora. Whenever you face a problem, there is a good chance that someone has faced the same problem, solved it, and posted the solution on the internet. There is also a large range of books dealing with several aspects of R. Some of these books can be downloaded for free (from computers within the UGent network). Check the Use-R books from the Springer publishers (<http://www.springer.com/series/6991>) for a good starting point. The current notes are based on the book "**Biostatistical Design and Analysis using R. A practical Guide**" by Murray Logan.

This chapter is meant as an introduction to work with R, the possibilities of the package and some important functions needed for data exploration. You will also learn how to do mathematics that are usually done in MS Excel. Using several examples, you will get familiar with working in R, but we do advice you to practice a lot, maybe even with a self-made dataset. Later on, the commands that are introduced here will be considered as general knowledge.

2.1. Installing R

When R is already installed on your computer, you can skip this part. If not, you will need to install it via: <http://www.r-project.org/>

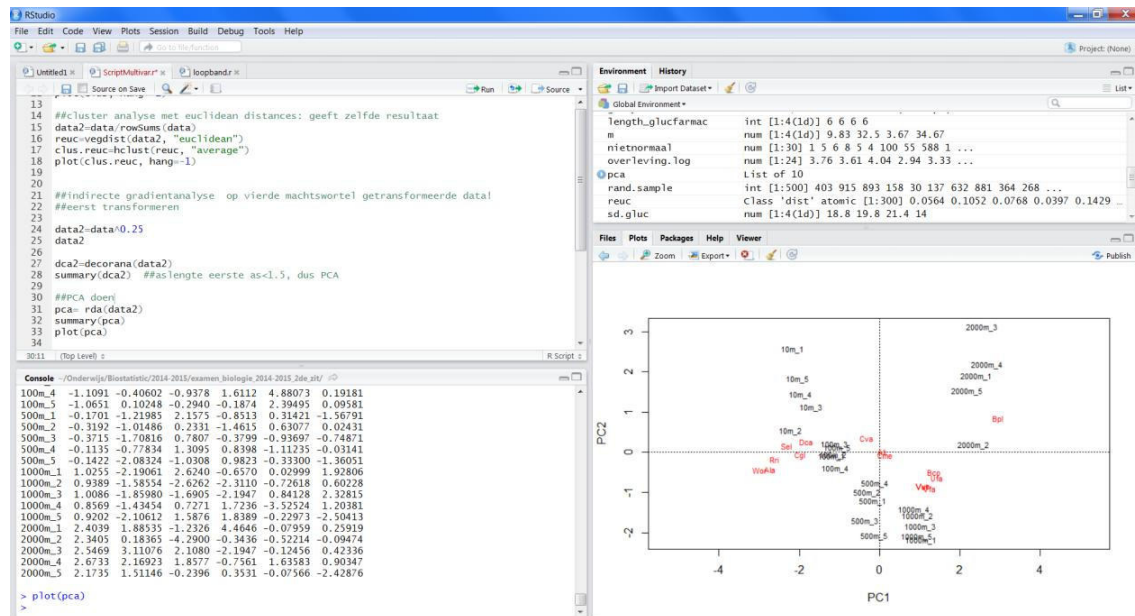
2.2. R-interface

To start R from a Windows environment, click the blue "R" icon on your desktop, or use start => programs. You see a screen with some text, and a ">" on the last line. This window is the **R console**. The last line is the **command line**, where commands will be entered. A command is executed by typing it at the command line and pressing "**enter**". Now it should be clear: R-GUI is not a GUI!

Typing commands at the command line can lead to typing errors, or very unstructured scripts (the collection of commands needed to perform a task). Therefore, we use a script-editor to enter the commands. R has a build-in script editor, that can be activated by *File-New Script* or **Ctrl+n**. This command opens a new window that can be arranged next to the R console. Commands can be typed in the editor, and sent to the console via **Ctrl+r**, or using the Run icon. The command will be executed by R.

Recently, R Studio became a better alternative. R Studio integrate the R console in RStudio, and offers the possibility for direct inspection of created plots. You can download R Studio from <http://www.rstudio.com/> (Mac and Linux users will first have to install R before installing R studio). A screenshot of RStudio is below. The upper right window is the text editor. Here you can type text, edit text, copy and paste...In short, you can use it as word processor to write code. However, code cannot be executed in the editor, you need to send it to the console, where it will be executed. If the result is a graph, you can see it in the lower right window (check "Plots" tab), when the results is not a graph, it will be in the console. The upper right window provides

information about the items you are working with (Workspace tab), or provide an overview of what you have been doing (History tab).



2.3 R as a calculator

R can be used as a regular calculator. Type the following commands:

```
> 45 + 23
> 3 * 6
> 3^2 + 2 * (1 - 0.2)
> sqrt(9)
> exp(2)
> sin(0.3)^2 + cos(0.3)^2
> 625^0.25
> log10(1000)
> log(1000, 10)           #same as log10(1000)
> log(1000)              #natural logarithm
> exp(2)                 #natural exponential function e^x
> log(exp(3))
```

Transforming numerical data:

```
> sqrt(x)                # square root
> sqrt(x+0.5)            # modified square root transformation
> log(x)                 # the natural log of x
> log10(x)               # log base 10 of x
> exp(x)                 # exponential of x
> abs(x)                 # absolute value of x
> asin(sqrt(x))          # arcsine square root (used for proportions)
```

Be careful! R is case-sensitive!

2.4 Objects, vectors, matrices and data frames

2.4.1. Objects

Everything within R is an object. A number is an object, a variable is an object, output is an object...Objects need to be given a name in R. Names can consist of virtually any sequence of letters and/or numbers, provided that

- names start with a letter (i.e. "A1" is a valid name, "1A" is not a valid name)
- names cannot contain following symbols: space, , - + * / # % & [] { } () ~

It is not a good idea to use names that are also commands in R. This can cause confusion, both for R and for the user. As names have to be typed, using long names can lead to typing errors, typing effort and loss of time (i.e. shorten "temperature" as "temp"). Keep in mind that you need to remember the meaning of a name. When you need to transform the variable temperature, a good name could be log.temp or LogTemp. In most examples and text books, the "." sign is used.

An example:

```
>VAR1 = 2+3          #the expression (2+3) is allocated to the object VAR1
>VAR1               #this command calls the value of the object with the name VAR1
[1] 5               R evaluated the expression, and returns the output
```

Remark: expressions can be allocated to objects using "=" (this course) or using "<-" (many text books or examples on the WWW)

Objects can be used for calculations:

```
>VAR1-1             # subtract 1 from the value of object VAR1
>[1] 4             output
```

2.4.2. Vectors

A vector is a collection of objects of the same type. All objects within a vector are numerical, text...Vectors can be created using the *concatenate (c)* function.

```
> VAR2=c(2, 4, 6)   # creates a vector containing the numerical objects 2, 4 en 6
```

When vectors consist of numerical objects, vectors can be used in calculations:

```
> VAR2*2           # multiplies all objects of the vector VAR2 by 2
[1] 4 8 12
```

Vectors are the basic unit for data storage in R. Vectors can be seen as columns with a length equaling the number of rows in the column. The different types of vectors that can be used in R are listed in the following table:

Vector class	Example
Integer (whole numbers)	<pre>> 2:6 # vector of integers from 2 to 6 [1] 2 3 4 5 6 > c(1,3,9,12) # vector of integers [1] 1 3 9 12</pre>

Numeric (Real numbers)	> c(8.4, 2.1) [1] 8.4 2.1	# vector van real numbers
Character (letters)	> c('A', 'ABC') [1] "A" "ABC"	# vector of letters
Logical (TRUE or FALSE)	> c(2:4)==3 [1] FALSE TRUE FALSE	# evaluate the expression # the printed logical vector

Biological data mainly consist of numbers (i.e. temperature data) or text (list of stations in which temperature was measured). Therefore, this course will mainly deal with integer, numeric and character vectors.

2.4.2.1. Creating vectors

Vectors can be created fast using “clever commands”. In practice, these commands are not going to be used frequently in this course. We start with creating a vector consisting of all integer numbers from 10 to 18 (including 18):

```
> VAR1=10:18
> VAR1
[1] 10 11 12 13 14 15 16 17 18
```

Note: the previously made vector VAR1 is now replaced by the newly created vector VAR1

Numerical sequences can be obtained through the **seq()** function:

```
> VAR2=seq(from=2, to=16,by=4)    creates a sequence from 2 to 16, with interval 4
> VAR2
[1] 2 6 10 14
```

Or shorter:

```
VAR2=seq(2, 16,4)
```

Using **seq()**, the length of vector can be defined when the function is fully written:

```
> VAR2=seq(from=2, to=16, length=5)
> VAR2
[1] 2.0 5.5 9.0 12.5 16.0
```

This creates a vector with 5 numbers (numeric objects), spread evenly over the interval 2-16.

When sequences need to be repeated, the **rep()** function can be used:

```
> VAR=rep(4,5)                creates a vector containing the number “4” 5 times
> VAR
[1] 4 4 4 4 4

> VAR1=rep("station", 5)    creates a vector containing 5 times the text “station”
> VAR1
[1] "station" "station" "station" "station" "station"
```

Note: text is entered between the “ ” signs. R does not recognise text not enclosed in “ ”.

```
> VAR2=rep(c("yes", "no"), 5)    Repeat“yes” “no” 5 times in this order
> VAR2
[1] "yes" "no" "yes" "no" "yes" "no" "yes" "no" "yes" "no"
```

Note: in this case, one function (the *c*-function) is nested within the *rep* function. The inner function is executed first (i.e. creating the “yes” “no” sequence), and is repeated 5 times.

```
> VAR3=rep(c(2,3),c(4,5))          repeat 2 four times, and 3 five times> VAR3
[1] 2 2 2 2 3 3 3 3 3
```

Biological datasets are not only characterized by numbers, text (i.e. station names) needs to be stored in vectors as well. This is done in character vectors. As mentioned before: R only recognizes text when it is between “”.

Suppose an investigator noted the presence of plant species in 10 quadrants, and measured temperature as well.

De vector containing the station names can be made as:

```
Stations= c("Q1", "Q2", "Q3", "Q4", "Q5", "Q6", "Q7", "Q8", "Q9", "Q10")
> Stations
[1] "Q1" "Q2" "Q3" "Q4" "Q5" "Q6" "Q7" "Q8" "Q9" "Q10"
```

This is time consuming, and it is far more elegant to use the **paste** function. This function combines the separate parts (i.e. Q and 1) of the object name in one single name (Q1). The paste function consist of three parts: (1) the first part of the combination; (2) the second part of the combination and (3) the separator. If we need to combine Q with a number between 1 and 10 and without a space in between the Q and the number, we use

```
> Stations=paste("Q", 1:10, sep= "")
> Stations
[1] "Q 1" "Q 2" "Q 3" "Q 4" "Q 5" "Q 6" "Q 7" "Q 8" "Q 9" "Q 10"
```

By combining functions, complicated vectors can be made fast. Next example shows how a vector is created, which consists of 5 stations (A till E – using the **LETTERS** function) which are sampled twice. Replicates should be after each other in the vector, and station name and replicate number should be separated by a “_” sign:

```
> New.Stations=paste(rep(LETTERS[1:5], each=2), 1:2, sep="_")
> New.Stations
[1] "A_1" "A_2" "B_1" "B_2" "C_1" "C_2" "D_1" "D_2" "E_1" "E_2"
```

2.4.2.2. Special vectors

Data in biological datasets are often coupled, and the correct link needs to be specified to allow later analyses. We go back to the example of the researcher measuring temperature in 10 quadrants. Half of these quadrants were shaded; the other half was not shaded. Temperature is stored in the vector **temp**:

```
temp=c(10.2, 11, 13, 12.3, 14, 22, 23, 21.5, 23.6, 22.1)
```

The information about the quadrants is stored in the vector **Stations** (see above)

```
> Stations
[1] "Q 1 " "Q 2 " "Q 3 " "Q 4 " "Q 5 " "Q 6 " "Q 7 " "Q 8 " "Q 9 "
"Q 10 "
```

The first five quadrants were not in the shade, the last five were. We store this information in the vector **shadow**

```
> shadow=rep(c("no", "yes"), each=5)
> shadow
[1] "no" "no" "no" "no" "no" "yes" "yes" "yes" "yes" "yes"
```

When calling the vector **temp** we have no idea about the link between temperature and the quadrant in which the temperature was measured. We can solve this, by linking the elements of the vector **temp** to the codes of the quadrants using the function **names()**:

```
> names(temp)=Stations
> temp
  Q 1   Q 2   Q 3   Q 4   Q 5   Q 6   Q 7   Q 8   Q 9   Q 10
10.2  11.0  13.0  12.3  14.0  22.0  23.0  21.5  23.6  22.1
```

Now, every observation is coupled with its corresponding quadrant.

We could test statistically whether the temperature is significantly different between quadrants that are shaded, and the other quadrants. To do this, we need to “explain” to R that the text “yes” and “no” are actually **two factors**. This is done using the function **factor()**

```
> shadow=factor(shadow)
> shadow
 [1] no no no no no yes  yes  yes  yes  yes
Levels: yes no
```

The content of the vector **shadow** is no longer text (the output is no longer between “”). R treats the vector as a numeric vector with values 1 and 2 (2 levels), where 1 codes for “no” and 2 codes for “yes”. This allows the user to perform statistical tests. Other, shorter methods to create factors include:

```
shadow=factor(rep(c("no","yes"), each=5))
or
shadow=gl(2,5,10, c("no","yes"))
```

The input for the function **gl()** consists of the number of levels (here: 2), the amount of replicates (here: 5), and the factor labels (the concatenation of “no” and “yes”).

All these different kinds of vectors can be merged in a single data frame that will be used in the final analyses.

2.4.3 Matrices and lists

Vectors only have 1 dimension: length. For certain purposes, it can be useful to transform vectors to matrices with dimension length=length of the columns, and dimension width=number of columns.

We transform the vector “temp” to a matrix with 5 rows (“nrow=5”) using:

```
> matrix(temp,nrow=5)
      [,1] [,2]
[1,] 10.2 22.0
[2,] 11.0 23.0
[3,] 13.0 21.5
[4,] 12.3 23.6
[5,] 14.0 22.1
```

By default, matrices are filled by columns. If you need to fill a matrix by rows, you need to add “byrow=T” to the expression.

Matrices can be created based on different vectors, when these vectors have the same length. Suppose the X and Y coordinates of 5 stations are stored in 2 dedicated vectors:

```
> X=c(16.92, 24.03, 7.61, 15.49, 11.77)
> Y=c(8.37, 12.93, 16.65, 12.2, 123.12)
> XY=cbind(X,Y)           cbind: merge as columns
> XY
```

```

      X      Y
[1,] 16.92  8.37
[2,] 24.03 12.93
[3,]  7.61 16.65
[4,] 15.49 12.20
[5,] 11.77 123.12

```

```

> XY=rbind(X,Y)                rbind: merge as rows
> XY
      [,1] [,2] [,3] [,4] [,5]
X 16.92 24.03  7.61 15.49 11.77
Y  8.37 12.93 16.65 12.20 123.12

```

Lists are used to store objects of *different* length. As an example: we obtain data at 5 stations, we obtain 2 replicate measurements per station. As such we get 10 sample names (A1-A2-...-E1-E2), 10 values for the observations, but only 5 geographical X coordinates and 5 geographical Y coordinates (1 X and 1 Y value for each station). This results in 2 vectors of length 10, and 2 vectors of length 5. These vectors can be combined using the **list()** function.

2.4.4. Data frames

For the analyses of biological data, data are mainly stored in data frames. Data frames are used to store vectors of **the same length**. The vectors can be numeric or character vectors.

2.4.4.1 Creating data frames

Data frames can be created based on existing vectors. In this example, we use the vectors that were created before. The vectors are merged in the data frame with the name **dataset**, using the command **data.frame()**:

```

> Stations=paste("Q", 1:10, sep= "")
> shadow=gl(2,5,10, c("no","yes"))
> temp=c(10.2,11,13,12.3,14,22,23,21.5,23.6,22.1)
> dataset=data.frame(Stations, shadow, temp)
> dataset
  Stations shadow temp
1      Q1      no 10.2
2      Q2      no 11.0
3      Q3      no 13.0
4      Q4      no 12.3
5      Q5      no 14.0
6      Q6      yes 22.0
7      Q7      yes 23.0
8      Q8      yes 21.5
9      Q9      yes 23.6
10     Q10      yes 22.1

```

We see that vectors are treated as columns in a data frame. The order in which the vectors are incorporated in **data frame ()** is important. **Dataset=data.frame(shadow, temp, Stations)** will generate a different data frame. As we are now working with the data frame, it is important to remove the original vectors from the workspace to avoid confusion (both for the user and R). This can be done using **rm()**.

```

> rm(Stations, shadow, temp)

```

Vectors within a data frame can be called by **data frame\$vector**. We can call the vector **temp** from the data frame **dataset** with the command

```
> dataset$temp
[1] 10.2 11.0 13.0 12.3 14.0 22.0 23.0 21.5 23.6 22.1
```

However, this takes a lot of typing. Using the command **attach()**, we can call vectors by their names.

```
> attach(dataset)
> Stations
[1] "Q1" "Q2" "Q3" "Q4" "Q5" "Q6" "Q7" "Q8" "Q9" "Q10"
```

Note: this approach is often discouraged in textbooks, because it can cause confusion with the original vector names. Therefore, it is very important to remove the original vectors.

2.4.4.2. Reading data frames

In many cases, data frames are not created within R. Datasets can be opened by R as '.txt' or '.csv' files. These files are not default created by e.g. excel. However, using "save as – text (Tab delimited)", .txt files are saved easily in excel and other software packages. The first row needs to contain the vector names, data should be arranged below. In the first column, rownames can be added, but cell A1 needs to be empty. **R only recognises "." as a decimal sign!** Fields containing "," will be recognised as text. For this course, files are mainly provided as .txt files, or they should be created in excel.

R searches for data in its **working directory**. Using **getwd()** you can check the current working directory. In most cases, the file you need to work with is not in the actual working directory. You can change the working directory with **setwd("C:/...)** and you need to specify the path to the folder where the file is stored. Alternatively, you can use 'File-Change Dir' from the R console to browse to the correct directory.

You can specify the working directory in several ways in RStudio. All of them can be found under "Session – Working Directory). You can browse to the directory ("Choose Directory"), or set the active directory in the "Files" pane as working directory. Alternatively, you can set the working directory as the location where you saved the script ("to source file location).

On Minerva/Zephyr, you can find the datafile "gluc.txt". This file contains data from an experimental treatment of mice. Mice were treated with adrenaline or saline. Half of the mice within each treatment were infected with pertussis (whooping cough). The concentration of glucose in the blood, and survival of the mice was measured.

Download this file, and point R to the correct working directory. To open the file in R, you need the command **read.table()**. In the argument, we add the name of the file, and we add the expression "**header=TRUE**" to show that the first row contains the column names.

```
> gluc=read.table("gluc.txt", header=TRUE)
> gluc
  pharmac  bact  glucose  survival  food
1     sal    p    163        43     11
2     sal    p    157        37     23
3     sal    p    177        57     24
4     sal    p    139        19     12
5     sal    p    148        28     34
6     sal    p    144        24     32
7     sal    n    330         5     41
8     sal    n    302         2     23
9     sal    n    283         1     14
10    sal    n    273         9     12
11    sal    n    307         3     34
12    sal    n    279         2     32
13    adr    p     94        78     31
14    adr    p    109        50     42
15    adr    p    146        30     12
16    adr    p    141        25     23
17    adr    p    124        10     21
```

```

18  adr  p    114      2    24
19  adr  n    221      6    43
20  adr  n    200      9    23
21  adr  n    233     10    33
22  adr  n    180     12    11
23  adr  n    198     16    12
24  adr  n    213      6    12

```

```
> attach(gluc)
```

We now opened the data frame, we inspected it (Gluc) and we made sure we can call the vectors by their name (attach(Gluc)).

2.4.4.3 Selecting parts of data frames

Sometimes, calculations should only be done on parts of the dataset. We need to select these data (in R terminology: indexing) using the command **data frame[row, column]**. We mention the rows and/or columns to be selected between [].

```
> gluc[2, ]
  pharmac bact glucose survival food
2    sal    p    157      37    23
```

We selected the second row. Selecting the third column is achieved by

```
> gluc[, 3]
 [1] 163 157 177 139 148 144 330 302 283 273 307 279  94 109 146 141 124
 [16] 114 221 200 233 180 198 213
```

Selecting the values from the second and fourth row, from the first and third column:

```
> gluc[c(2,4), c(1,3)]
  pharmac glucose
2    sal    157
4    sal    139
```

We will now select the data for which the glucose concentration exceeds 200

```
> gluc[glucose>200,]
  pharmac bact glucose survival food
7    sal    n    330      5    41
8    sal    n    302      2    23
9    sal    n    283      1    14
10   sal    n    273      9    12
11   sal    n    307      3    34
12   sal    n    279      2    32
19   adr    n    221      6    43
21   adr    n    233     10    33
24   adr    n    213      6    12
```

In some cases, the use of the **subset()** command provides a more elegant solution to a selection problem. The argument for this function consists of three parts : **subset data frame, subset=,select=**), only the first part is obligatory. With “subset”, rows are selected, “select” is used for column selection. To select the columns “pharmac”, “bact” and “survival” from the data frame gluc, and to store this selected information in a new data frame gluc1, we use:

```
> gluc1=subset(gluc, select=c("pharmac","bact","survival"))
> gluc1
  pharmac bact survival
1    sal    p          43
```

2	sal	p	37
3	sal	p	57
4	sal	p	19
5	sal	p	28
6	sal	p	24
7	sal	n	5
8	sal	n	2
9	sal	n	1
10	sal	n	9
11	sal	n	3
12	sal	n	2
13	adr	p	78
14	adr	p	50
15	adr	p	30
16	adr	p	25
17	adr	p	10
18	adr	p	2
19	adr	n	6
20	adr	n	9
21	adr	n	10
22	adr	n	12
23	adr	n	16
24	adr	n	6

Here, we did not use the subset argument to select rows as we wanted all rows to be included in the new data frame.

To select all columns (hence, we do not use the 'select' argument) with the values for those mice treated with saline, we use:

```
> gluc.saline=subset(gluc, pharmac=="sal")
> gluc.saline
  pharmac bact glucose survival food
1     sal    p   163         43    11
2     sal    p   157         37    23
3     sal    p   177         57    24
4     sal    p   139         19    12
5     sal    p   148         28    34
6     sal    p   144         24    32
7     sal    n   330          5    41
8     sal    n   302          2    23
9     sal    n   283          1    14
10    sal    n   273          9    12
11    sal    n   307          3    34
12    sal    n   279          2    32
```

Subsets with only saline and pertussis:

```
gluc.sal.pert=subset(gluc, pharmac=="sal" & bact=="p")
gluc.sal.pert
```

Rows and columns are selected using:

```
> gluc.s2=subset(gluc, pharmac=="sal", select=c("bact", "survival"))
> gluc.s2
```


2.4.4.4 Calculations with data frames

Very often, transformation of data is required before an actual analysis can be performed. Then we need to do calculations on parts of the data frame, and we need the result of this calculation to be added to the data frame. As an example: we need to log-transform the data in the vector survival, and the new data should be added to the data frame:

```
> log.survival=log(survival)
> gluc=data.frame(gluc,log.survival)
> gluc
  pharmac bact glucose survival food log.survival log.survival.1
1     sal   p   163      43     11     3.7612001     3.7612001
2     sal   p   157      37     23     3.6109179     3.6109179
3     sal   p   177      57     24     4.0430513     4.0430513
```

Or shorter:

```
>gluc$log.survival=log(survival)
```

It is frequently convenient to get information about the distribution of the data. This can be obtained using the command **summary()**. Between (), you can specify a data frame or a vector.

```
> summary(gluc)
```

The commands **mean()** and **sd()** are used to calculate the mean and the standard deviation, respectively. This can only be done with numeric vectors. In order to prevent error messages as a result of the sd() function, the vector needs to be specified.

```
> mean(glucose)
[1] 194.7917
> sd(glucose)
[1] 69.48818
```

Actually, it doesn't make sense to calculate the average glucose concentration in the blood, as the mice were subjected to different treatments. If we are interested in the mean concentrations per group of treatments, we could make subsets and calculate the mean and standard deviation for each subset. However, this is time consuming and this can be avoided using the **tapply()** function. This function allows the execution of a function for each level of a factor. There are three arguments associated with tapply(): the vector for which we need to apply a function, the factor and the function that needs to be executed (**tapply(vector, factor, function)**). If we are interested in the average glucose concentration in the blood of the mice subjected to the different infection treatments, we use

```
> tapply(glucose, bact, mean)
      n      p
251.5833 138.0000
```

If we need to calculate the mean concentration for each infection-farmac combination, we apply

```
> tapply(glucose,bact:pharmac,mean)
  n:adr  n:sal  p:adr  p:sal
207.5000 295.6667 121.3333 154.6667
```

2.4.4.5 Calculation of the standard error

R does not have a standard function to calculate the standard error of a mean. The standard error is calculated as the standard deviation/square root (number of observations). This takes a few steps in R. We use the function **length** to retrieve information about the number of observations.

```

> sd.gluc=tapply(glucose, bact:farmac, sd)
> number.observations=tapply(glucose, bact:farmac, length)
> se=sd.gluc/sqrt(number.observations)
> se
  n:adr  n:sal  p:adr  p:sal
7.671375 8.754681 8.073276 5.707695

```

2.5 Creating graphs in R

R is a strong graphical programme, because graphs are highly adaptable to the user's aim. Several packages allow creating very specific graphs and a lot of websites gather scripts for specific graphs.

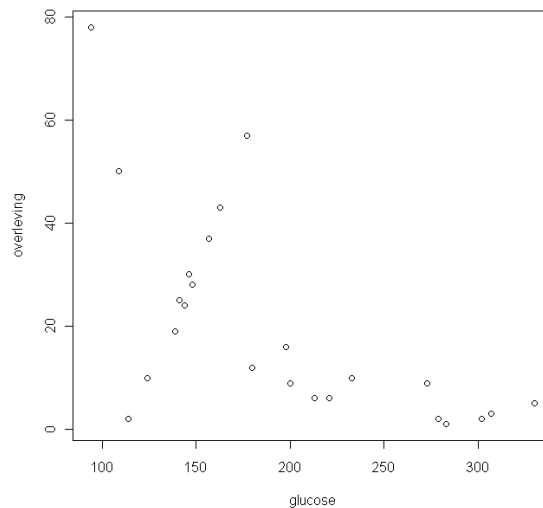
Advanced users usually use the `ggplot2` package to create graphs. There is a lot of information on `ggplot2` available on ggplot2.org and in the "Cookbook for R" (www.cookbook-r.com/).

2.5.1 The x-y plot

A frequently used function is `plot()`, which can be further defined by adding arguments between the brackets. First, we have to define what has to be plotted: `plot(x,y)` will plot the values of vector `x` on the x-axis and the values of vector `y` on the y-axis. The same result is obtained with `plot(y~x)` (read as: plot `y` as a function of `x`). We will plot survival as a function of the glucose concentration as:

```
> plot(survival~glucose)
```

The output does not appear in the R console, but in a new window (R Graphics). Whenever a new plot is created, the previous one is overwritten. If a plot has to be saved, you can add a new R graphics window by writing the command `windows()`. The plot looks like this:



The x- and y axes get a standard title, which corresponds with the name of the vectors that are plot. With the arguments `xlab = "..."` and `ylab = "..."` you can add a better title to the axes.

```
> plot(survival~glucose, xlab="glucose concentration(mg/l)", ylab="survival (days)")
```

The character size can be adapted with `cex` (character expansion). Titles of axes can only be adapted using `cex.lab`. Try next command and look at the table below for more details

```
> plot(survival~glucose, xlab="glucose concentration (mg/l)",
ylab="survival (days)", cex.lab=1.5)
```

Table 5.3 Character expansion parameters.

Parameter	Applies to
cex	All subsequent characters
cex.axis	Axes tick labels
cex.lab	Axes titles
cex.main	Main plot title
cex.sub	Plot sub-titles

A general title can be added with **main = "..."**. Add the general title "SURVIVAL" by:

```
> plot(survival~glucose, xlab="glucose concentration (mg/l)",
ylab="survival (days)", cex.lab=1.5, main=" SURVIVAL ")
```

The scale of both x – and y axes was defined by R. It can be adjusted with **xlim=c(minimum,maximum)** and **ylim=c(minimum, maximum)**. We will set the scale of the y-axis between 0 and 120 by:

```
> plot(survival~glucose, xlab="glucose concentration (mg/l)",
ylab="survival (days)", cex.lab=1.5, main="SURVIVAL", ylim=c(0,120))
```

The symbols in the plot can be changed by **pch=number**, in which the number codes for a symbol. The most frequently used symbols and their codes are mentioned in the table below:

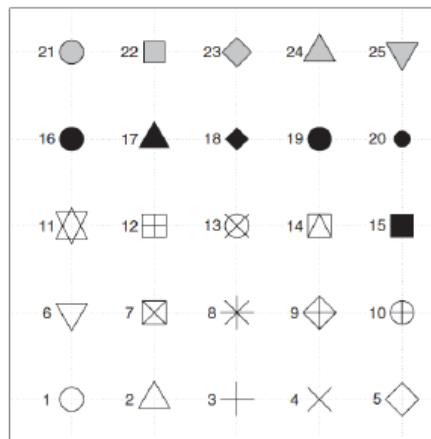
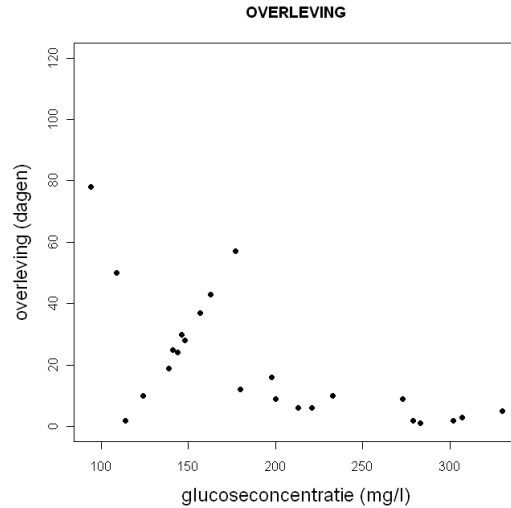


Fig 5.2 Basic pch plotting symbols.

We will change the plot symbols to black circles (code 19) by:

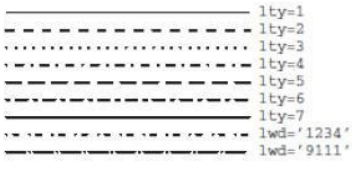
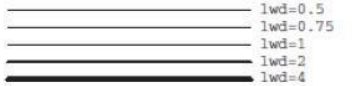


```
> plot(survival~glucose, xlab="glucose concentration (mg/l)", ylab="survival
(days)", cex.lab=1.5, main="SURVIVAL", ylim=c(0,120), pch=19)
```

We get this figure:



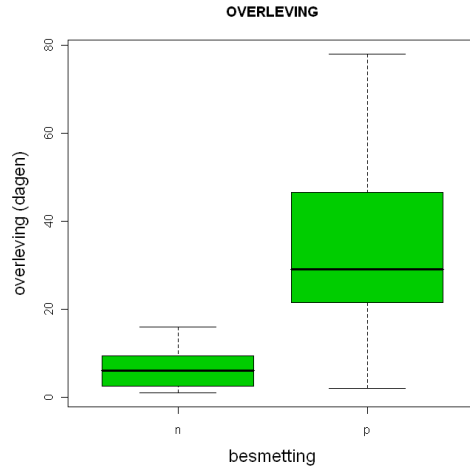
You can choose not to work with data points, but with lines (**type = "l"**), or with a combination (**type = "b"**). The line type can be adjusted with **lty = number**, and the thickness of the line is indicated by **lwd = number**. The table below summarizes the most frequently used options:

Table 5.4 Line characteristics.

Parameter	Description	Examples
<code>lty</code>	The type of line. Specified as either a single integer in the range of 1 to 6 (for predefined line types) or as a string of 2 or 4 numbers that define the relative lengths of dashes and spaces within a repeated sequence.	 <code>lty=1</code> <code>lty=2</code> <code>lty=3</code> <code>lty=4</code> <code>lty=5</code> <code>lty=6</code> <code>lty=7</code> <code>lwd='1234'</code> <code>lwd='9111'</code>
<code>lwd</code>	The thickness of a line as a multiple of the default thickness (which is device specific)	 <code>lwd=0.5</code> <code>lwd=0.75</code> <code>lwd=1</code> <code>lwd=2</code> <code>lwd=4</code>
<code>lend</code>	The line end style (square, butt or round)	 <code>lend=2</code> <code>lend=1</code> <code>lend=0</code>
<code>ljoin</code>	The style of the join between lines	 <code>ljoin=0</code> <code>ljoin=1</code> <code>ljoin=2</code>

When a factor is plotted on the x axis, R will generate a boxplot by default. With **col=number**, colours in the graph can be adjusted (this applies for all types of plots):

```
> plot(survival~bact, xlab="bacteria", ylab="survival (days)", cex.lab=1.5,
main="SURVIVAL", col=3)
```



2.5.2 Bar graphs

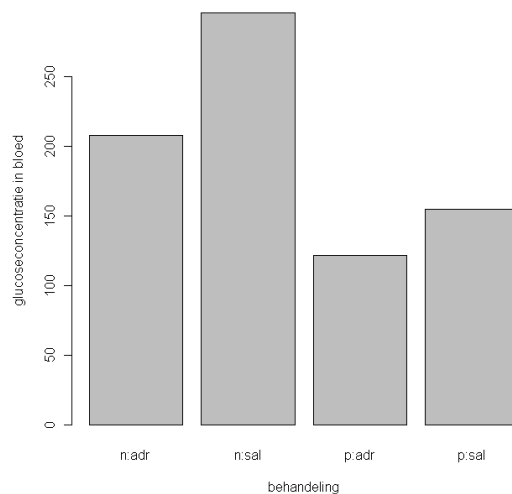
Bar graphs are frequently used in biology to plot a mean value and its standard error. R does not have a standard function to plot error bars, so we will have to create the code ourselves. To do this, we use the function **arrows**. In what follows, a bar graph will be created, then error bars are added and the final touch is given to the graph.

First, we calculate the standard deviation (`sd.gluc`), the number of observations (`observations`), and the mean (`means.gluc`). The standard error (`se.gluc`) is calculated for each combination of observations (using the `tapply` command).

```
> sd.gluc=tapply(glucose, bact:farmac, sd)
> observations=tapply(glucose, bact:farmac, length)
> se.gluc=sd.gluc/sqrt(observations)
> se.gluc
> means.gluc=tapply(glucose, bact:farmac, mean)
```

The boxplot is created by:

```
> b=barplot(means.gluc, xlab="treatment", ylab="glucose concentration in blood")
```



Plot the error bars on the graph by:

```
> arrows(b, means.gluc+se.gluc,b,means.gluc-se.gluc, angle=90, code=3)
```

The graph is still not good-looking. We can change some settings by adding arguments to the command `barplot()`. Using `?barplot`, we can ask for information about the function `barplot`. By `windows()` we create a blank new graphic window.

```
> windows ()
```

The bars exceed the scale of the y-axis. We can adjust this by calculating the minimum value of the mean-se and the maximum value of the mean + se and scale the y-axis in between those values. We add "1" to these values, to avoid that the error bars touch the limits of the graph.

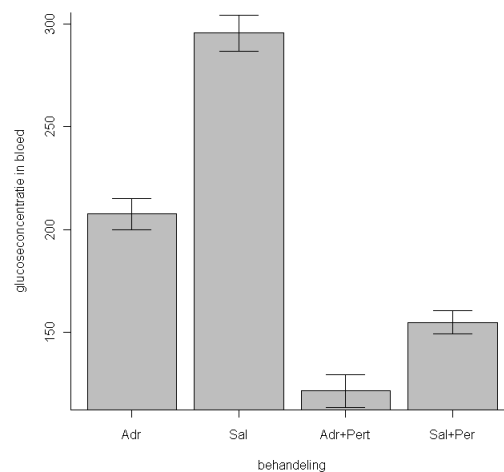
```
> max=max(means.gluc+se.gluc+1)
> min=min(means.gluc-se.gluc-1)
> b=barplot(means.gluc, ylim=c(min,max),xlab="treatment", ylab="glucose
concentration in blood")
```

Still, this does not look very nice: the bars exceed the limit of the y-axis and the x-axis is plotted in the bars. This can be avoided by writing `xpd=FALSE` in the argument.

```
> b=barplot(means.gluc, ylim=c(min,max),xlab="treatment", ylab="glucose
concentration in blood", xpd=FALSE)
```

Or better:

```
> b=barplot(means.gluc, ylim=c(0,max),xlab="treatment", ylab="glucose
concentration in blood", xpd=FALSE)
```



On the x-axis, we get as labels the names generated by R based on the original data frame. Because `n:adr`, `n:sal` etc. are not very informative, we can replace these with labels we choose ourselves. When we do not want the labels to be plotted automatically, we can use the argument `axisnames = FALSE` (or shorter: `axisnames=F`).

```
> b=barplot(means.gluc, ylim=c(min,max),xlab="treatment", ylab="glucose
concentration in blood", xpd=FALSE, axisnames=F)
```

A new x-axis is plotted with the command `axis()`. Since we want to create the x-axis, we use the argument "1", we point out where we want to plot the names ("at = b" or: take the coordinates of vector b) and we supply a vector with the names).

```
> axis(1, at=b, labels=c("Adr", "Sal", "Adr+Pert", "Sal+Per"))
```

We now have an incomplete x-axis that can be adjusted by adding the lower part of a box to the graph (**box(bty="l")** adds an incomplete box in the shape of an “L”).

We are satisfied with the lay-out of this graph, so we can add the error bars.

```
> arrows(b, means.gluc+se.gluc,b,means.gluc-se.gluc, angle=90, code=3)
```

2.5.3 The package “Lattice”

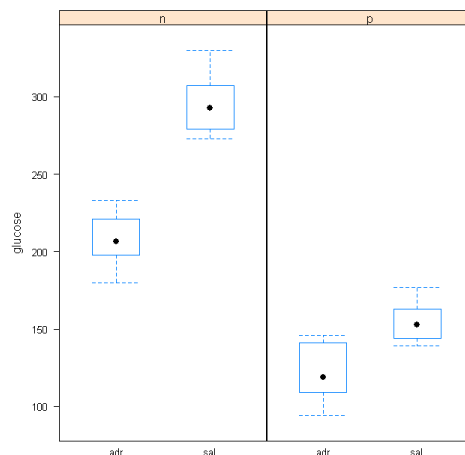
While installing R on your computer, not all possibilities within R are installed, since that would occupy too much space on your hard drive. You can load packages using **library()**, with between brackets the name of the package that you would like to use. Lattice is a package that allows you to create graphs that cannot be created with standard packages. Lattice is very much adapted to illustrate experimental data with a limited amount of commands. Install the package lattice by

```
> library (lattice)
```

Now we would like to create a graph in which we plot the glucose concentration for each bacterial and pharmaceutical treatment.

```
> bwplot(glucose~farmac | bact)
```

The vector following the “|” sign is the conditioning variable: for each level of this conditioning variable (in this case bact and thus “n” and “p”) a graph will be created, in which the glucose concentration for each level of “farmac” will be plotted.



Within “lattice” also other plots can be called. Try ?lattice for more information. Try also:

```
> xyplot(glucose~farmac | bact, pch=16, col=2)
```

2.6. Scripts

Command lines that are written in the R console are lost upon closure of R. In R editor, Tinn-R or R studio, you can save all commands as a script. A script has the advantage that you can add comments to your command lines, preceded by #. R will ignore the text written following the “#” sign. You can run entire scripts in R without

receiving any error message. As explained above, it can take a while before you have finished the layout of your graph. At first sight, it seems to take a lot of time to create graphs in R. However, once finished a script, you can immediately re-use it with another dataset. Creating a graph for a thesis or a paper takes – per graph – less time in another programme, but when you need a lot of (complicated) graphs, R scripts can save a lot of your time. An example of a script (to create a bargraph with error bars) can be found below:

```
> #script: creating bargraphs with error bars
> gluc=read.table("gluc.txt", header=T)
> gluc
> attach(gluc)
> #calculate se
> sd.gluc=tapply(glucose, bact:farmac, sd)
> observations=tapply(glucose, bact:farmac, length)
> se.gluc=sd.gluc/sqrt(observations)
> se.gluc
> means.gluc=tapply(glucose, bact:farmac,mean)
> means.gluc
> #calculate the minimum and maximum for the y-axis
> max=max(means.gluc+se.gluc+1)
> min=min(means.gluc-se.gluc-1)
> #create barplot: xpd= FALSE necessary to keep bars within y-axis limits,
axisnames=F necessary to plot a nice x-axis
> b=barplot(means.gluc, ylim=c(min,max), xpd=FALSE, xlab="treatment",
ylab="glucose concentration in blood", axisnames=F)
> #plot error bars on graph
> arrows(b, means.gluc+se.gluc,b,means.gluc-se.gluc, angle=90, code=3)
> # apply correct axis
> axis(1, at=b, labels=c("Adr","Sal","Adr+Pert","Sal+Per"))
> #add lower line
> box(bty="l")
```

2.7 Close R

When closing R, it is advisable to remove all objects from the working memory of the computer. You can call for a list of all objects by:

```
> ls()
```

Next, you put these objects in a list (**list=ls()**), that can be removed by the **rm()** command.

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

2.8 Exercises

1. Imagine you have collected macrobenthic samples along a continental slope. Starting from a depth of 100 m and ending at a depth of 1500 m, you have taken two samples in every station. The samples were taken at 100 m, 200 m ,300m, , 1500 m. All macrobenthic individuals were counted and you have obtained the following dataset:

depth	macrobenthos
100	2512
100	2412
200	2654
200	2102
300	2033
300	2000
400	1896
400	1923

500	1896
500	1523
600	1547
600	1546
700	1236
700	1248
800	1249
800	1023
900	1011
900	999
1000	915
1000	955
1100	947
1100	850
1200	848
1200	857
1300	800
1300	798
1400	745
1400	512
1500	517
1500	499

- A) Make 2 vectors, containing this data and connect them in a data frame. Do this in R, not in Excel.
- B) Make a graph in which mean macrobenthos density \pm standard error in function of depth is shown. Give both axes a meaningful name!

Remark: by changing a vector to a factor vector, R will automatically call the levels alphabetically, and not in the order in which the levels were original placed. Exp.: a vector "V=c("low","medium","high") will be shown in a graph as "high, low,medium". You can specify the order by: factor=factor(V,levels=c("low","medium","high")).

3 Descriptive statistics

Since it is usually not possible to work with data on entire populations, scientists use sample data. Every element from the population has an equal chance to be included in the sample. Based on sample data, statistics are used to make statements about the entire population. By means of statistical tests, it can be assessed whether relations are random, or significantly different from random.

Based on representative sample data, a null hypothesis (H_0) is formulated. This H_0 states that there is no relation between variables, or that groups of samples are drawn from the same population. By calculating a test statistic (e.g. F (ANOVA), t (Student's t -test)), and comparing the value of this test statistic with a relevant theoretical distribution of this test statistic, we assess whether the calculation value can be obtained by chance. If the chance of obtaining the calculated value is too small (e.g. $<5\%$ or $p < 0.05$), we reject the null hypothesis, and we accept a significant relationship between variables or a significant difference between groups of samples.

In general, null hypotheses are rejected at a p -value < 0.05 , or in other words: with a confidence of at least 95% that the null hypothesis is false.

When the investigator knows the direction of a relation, or difference, one-tailed tests are applied. The chance of obtaining a significant test statistic is only assessed at one side of the distribution of this statistic. However, in most cases, this direction cannot be predicted, and two-tailed tests are used in which equal chances on differences on the positive and negative sides of the theoretical distribution are assumed.

4 Formulas for statistical analysis

The first step in conducting statistical analyses is formulating a testable hypothesis. Following steps can be used as a guidance:

1. Define the dependent variable(s). These are the variables of interest, for which you want to test the distribution. In ecological research, dependent variables are often densities, diversity indices, biomass values...
2. Define the independent variables. Generally, one wants to describe the independent variable (e.g. densities) as a function of (a number of) independent variables (e.g. environmental variables: salinity, temperature, presence of food sources, presence of disturbances, sediment variables...)
3. Define whether interactions between independent variables should be taken into account (e.g. as in a 2 Way ANOVA)
4. Write down the formula. Using R (but valid for the bulk of statistical software programmes), we use a structure that can be generalised as:

$>Y\sim f(A,B,\dots)$

At the left hand side of the ‘~’, we write the dependent variables, at the right hand side we write the independent variables and their possible interactions. If interactions are not considered (e.g. in a linear regression), the formula looks like:

$>Y\sim A+B+C$

When interactions are considered important as well, this can be written as:

$>Y\sim A+B+A:B$

or, in short:

$>Y\sim A*B$

When only the interaction effect is considered, the formula becomes

$>Y\sim A:B$

When B is nested in A (e.g. B only makes sense as part of treatment A, then the independent effect of B is not considered, and only the interaction with A is investigated:

$>Y\sim A+A:B$

or, in short:

$>Y\sim A/B$

Both dependent and independent variables can be continuous, or discrete (e.g. factors: “Disturbed”, “Undisturbed”). This is important for deciding on the correct statistical approach (regression analysis, ANOVA...). When the dependent variable is multidimensional, we use multivariate techniques (ordination, MDS). However, the structure of the formula does not change.

5 ANOVA

Analysis of variance or ANOVA is used in order to test if there is any difference for a dependent variable between groups. We use the dataset *Gluc* as example. We want to identify if there is any difference in survival between mice with an adrenaline-injection and mice with a saline-injection. Further we want to know if there is an effect of a pertussis-infection on survival and if the pertussis influences the effect of the adrenaline-injection.

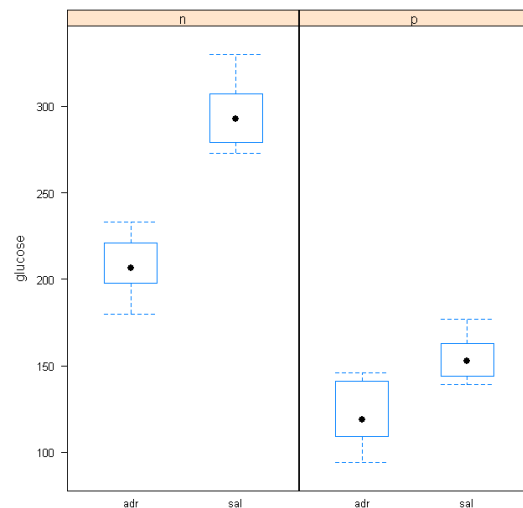
5.1 Assumptions for ANOVA

Three assumptions have to be fulfilled before we can apply ANOVA (1). all subsets of the data needs to be normally distributed, and (2). the variances are homogeneous.

If one of the assumptions is not fulfilled you have to apply a transformation. If that still does not help, you have to use a non-parametric alternative test.

Before testing the assumptions, it is useful to perform an exploratory data analysis. From the graphs produced in sections 2.5.2 and 2.5.3, it seems like the survival of mice is influenced by pertussis, possibly in combination with the adrenaline treatment. One of the graphs was made using:

```
>library(lattice)
>bwplot(glucose~farmac | bact)
```



In addition, we can make a graph to investigate the relation between the means and the variances. When a positive relation between means and variances (in other words: a higher mean value is associated with a higher variance) is detected, a transformation is often needed to meet the assumptions.

Means can be calculated using the command *mean()*, variances are obtained using *var()*. Since we need to calculate means and variances for each possible combination of treatments, we could create subsets, calculate means and variances, and assign these values to two vectors. This is however time consuming. A more elegant solution is provided by the function *tapply()*.

The means and variances for the variable *survival* and the grouping variables *farmac* and *bact* can be calculated using:

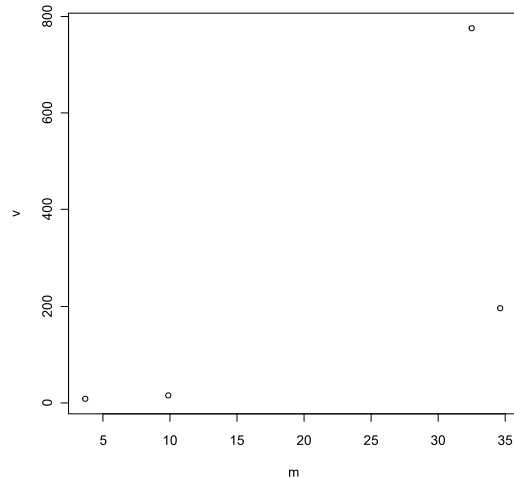
```
> m = tapply(Gluc$survival, Gluc$farmac:Gluc$bact, mean)
> v = tapply(Gluc$survival, Gluc$farmac:Gluc$bact, var)
> m
```

```
adr:n      adr:p      sal:n      sal:p
9.833333   32.500000   3.666667   34.666667
```

```
> v
adr:n      adr:p      sal:n      sal:p
14.566667  775.100000  8.666667  195.466667
```

The correlation between means and variables is obtained through a graph that can be produced using

```
> plot(v ~ m)
```



The variance does not increase when the mean increase: the highest mean value is not associated with the highest variance. To confirm the absence of a correlation, we use

```
> cor.test(m, v)
```

```
Pearson's product-moment correlation
```

```
data: m and v
t = 1.4237, df = 2, p-value = 0.2905
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.7909135  0.9932778
sample estimates:
      cor
0.7094608
```

We obtain a non-significant correlation ($p=0.29$) with a value of 0.7 (more information about correlation tests: see further).

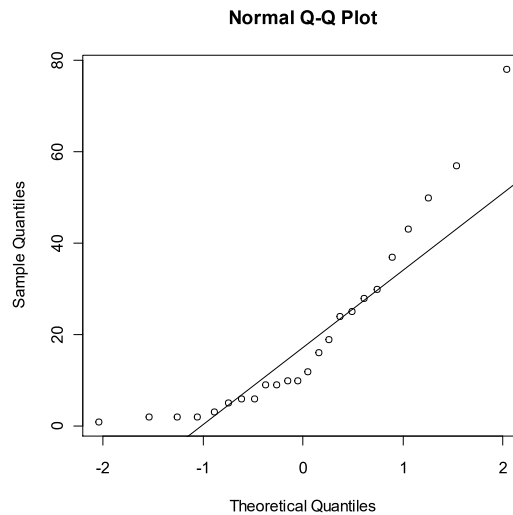
NOTE: it makes no sense to produce this graph, and calculate the correlation, when only two groups are investigated. Means and variances are assumed to be independent in that case.

5.1.1 Normality of the data

Visually

With a QQ-norm graph (normal probability plot) we plot the data against a normal distribution with the same mean and variance. If all the data are on one line they are normally distributed. The stronger the data deviate from a straight line, the lesser they approach a normal distribution.

```
> qqnorm(Gluc$survival)
> qqline(Gluc$survival)
```



This method is not quantitative but often recommended.

Shapiro-Wilk normality test

A quantitative test providing a p value is the normality test: a $p < 0.05$ indicates that the data are not normal distributed. (Null hypothesis: the data originate from a normal distribution)

```
> shapiro.test(Gluc$survival)
```

```
Shapiro-Wilk normality test
data:  Gluc$survival
W = 0.844, p-value = 0.001696
```

We have to test normality for all subsets (groups) of the ANOVA. Make subsets like shown in 2.13, produce the qqplots, and apply the normality tests:

```
> qqnorm(Gluc.a$survival)
> qqline(Gluc.a$survival)

> qqnorm(Gluc.s$survival)
> qqline(Gluc.s$survival)

> shapiro.test(Gluc.a$survival)
> shapiro.test(Gluc.s$survival)
```

In case of a two-way ANOVA there are 4 groups to test for in this example:

```
> shapiro.test(Gluc.ap$survival)
> shapiro.test(Gluc.an$survival)
> shapiro.test(Gluc.sp$survival)
> shapiro.test(Gluc.sn$survival)
```

An alternative method is to apply the function `tapply()` (see also `help(tapply)`):

```
> attach(Gluc)
> tapply(survival, farmac, shapiro.test)
> tapply(survival, bact, shapiro.test)
> tapply(survival, farmac:bact, shapiro.test)
```

This function calculates a function (*shapiro.test()*) of a variable (*survival* for each group identified by a grouping variable (*farmac,...*)

If none of the test results is significant, the data meet the assumption for normality.

The *tapply()* function cannot be applied in combination with *qqnorm* and *qqplot*. We can produce plots for each subset, however this is time consuming. R allows the user to produce customised functions. The general syntax of a function is:

```
Name=function(arg1,arg2,...) expr1
```

The functions has a name (Name), and executes an (user-defined) expression for the arguments considered (arg1, arg2...defined by the user). The next example describes a function producing qqplots for all combinations of groups:

```
> qqnorm.and.line <- function(x, ...) {
  qqnorm(x, ...)
  qqline(x)
}
```

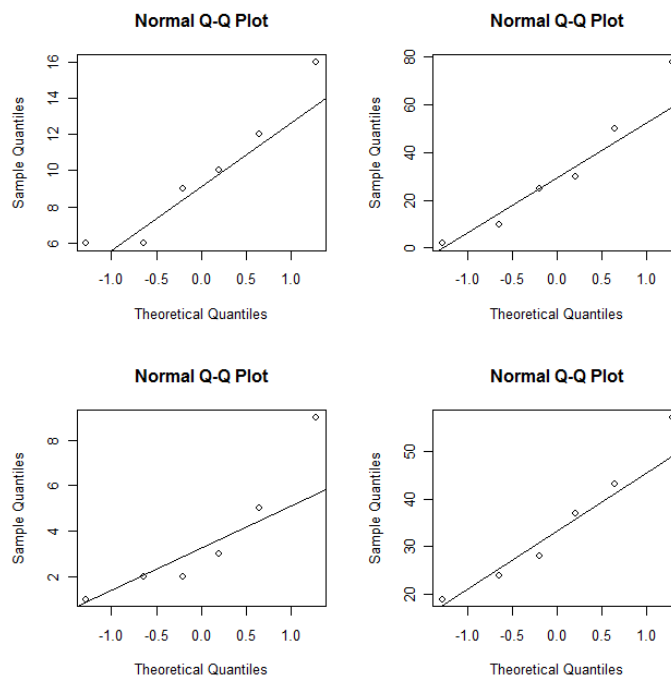
The name of the function is 'qqnorm.and.line', the expression can be found between {}. The function needs to be called by its name, and "x" should be replaced by the relevant argument. The command lines below will result in 4 qqplots, since we nest the function *qqnorm.and.line* within *tapply()*. In order to see the 4 plots in a single window, we arrange the plots in 2 rows and 2 columns, using the *par()* function.

```
>par(mfrow = c(2, 2))
>tapply(gluc$survival, gluc$farmac:gluc$bact, qqnorm.and.line)
```

NOTE: if we use *attach()*, this can be written as:

```
>par(mfrow = c(2, 2))
>tapply(survival,farmac:bact, qqnorm.and.line)
```

The result looks like:



In every plot, the data are close to the line, and we conclude that the data meet the assumption of normality.

5.1.2 Homogeneity of variances

The Levene test is used for testing if the variances of a variable (survival) are homogeneous for all groups (adrenaline and saline). Install and load the package "car" (see section 2.15).

```
> library(car)
> leveneTest(Gluc$survival, Gluc$farmac)

Levene's Test for Homogeneity of Variance
      Df F value Pr(>F)
group 1  0.0519  0.8218
      22
```

If $p > 0.05$, we can conclude that variances are homogeneous (= nul hypothesis). In order to test the homogeneity of the variances in the subgroups of a 2-way anova, we combine the effect of *farmac* and *bact*:

```
> fb = Gluc$farmac:Gluc$bact
> levene.test(Gluc$survival, fb)

Levene's Test for Homogeneity of Variance
      Df F value Pr(>F)
group 3  4.4997  0.01438 *
      20
```

--- Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

5.2 t-test

In order to test if the means of **two** groups of data differ or not, we can apply a Student's t-test. This test is valid when the data are normally distributed, or when both samples are sufficiently large (at least 30 cases).

```
> t.test(survival ~ farmac, data = Gluc, var.equal = TRUE)
```

In this case the second variable, *farmac* has to be a factor with two levels. We assume here that the variances in both groups are equal.

Note: When *attach(Gluc)* is used you do not need to indicate *data=Gluc*

5.3 F-test

```
> var.test(survival ~ farmac, data = Gluc, equal = TRUE)
```

5.4 ANOVA

Calculating an ANOVA in R consists of two parts: first we do a minimization (=*to calculate mean and variances*) with the function *aov()*. In a second step the statistical significance and sum of squares are calculated and placed in a table with the function *anova()* or the function *Anova()* from the package *car*.

The default function *anova()* in R will perform a type I (= sequential) sum of squares (SSI). This type is dependent on the order that the independent variables are entered. In most case we want the type III (= marginal) sum of squares (SSIII), which corrects for side effects, while SSI is only used for pure nested designs.

Therefore we use the function *Anova()* from the package *car* (Capital! !), which allows to calculate type II and type III SS. In the case of a balanced design we obtain the same results with type I, II or III. In the case of unbalanced designs, you have to be aware the use the correct SS. In that case use *Anova()* or *drop1()* for a type III SS

5.4.1 1-way ANOVA

In a one-way ANOVA, we use one 1 continuous dependent variable (survival), and 1 discrete independent variable (farmac). We test the hypothesis that survival is not affected by the pharmaceutical treatment.

```
> x = aov(survival ~ farmac, data = Gluc)
> anova(x)
```

```
Analysis of Variance Table
Response: survival
          Df Sum Sq Mean Sq F value Pr(>F)
farmac    1   24.0    24.0  0.0562 0.8148
Residuals 22 9393.3   427.0
```

```
> library(car)
> Anova(x, type = "III")
```

```
Anova Table (Type III tests)
Response: survival
          Sum Sq    Df F value    Pr(>F)
(Intercept) 5376.3    1  12.5918 0.001802 **
Farmac       24.0     1   0.0562 0.814783
```

```
Residuals    9393.3    22
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The function *aov()* applies the anova. The function *Anova()* gives a table with an overview of the results of the anova: degrees of freedom , sum of squares, mean square, F-test, p-value. In the case of a one-way ANOVA with two groups (*sal* and *adr*) the same results as in a t-test will be obtained.

The most important information to retrieve in these tests is the p- value $Pr(> F)$. The p value provides the probability dat the nul hypothesis can be accepted (no differences between groups). In most cases a 95% confidence interval is used and is the nul hypothesis rejected if $p < 0.05$. Here we can conclude that the farmaceutical treatment has no effect on the survival of mice.

Task

Test the effect of the pertussis-treatment on the survival of mice. Formulate the correct hypothesis, make a graph and give an interpretation of the test.

5.4.2 2-way ANOVA

For a two-way ANOVA the independent variable (e.g. *farmac*) is replaced by two independent variables *farmac*bact*. "*" returns the variances within and between groups while "+" only looks within groups and ":" looks between groups.

We test the hypothesis that survival of the mice is not affectd by the combination of both treatments (farmac and bact)

```
> x = aov(survival ~ farmac * bact, data = Gluc)
> Anova(x, type="III")
```

Analysis of Variance Table

Response: survival

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
farmac	1	24.0	24.0	0.0966	0.7591659
bact	1	4320.2	4320.2	17.3885	0.0004728 ***
farmac:bact	1	104.2	104.2	0.4193	0.5246671
Residuals	20	4969.0	248.4		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

First, we investigate the interaction effect. As $p > 0.05$, we accept the hypothesis that the combination of treatments does not affect survival of the mice. As this interaction term is not significant, we can explore the one-way terms (e.g. *farmac* and *bact* independently).

We conclude that the pertussis infection has a significant effect on the survival of mice (“ H_0 : infection with pertussis does not effect the survival of mice” is rejected). The pharmaceutical treatment has no effect on the survival (“ H_0 : pharmaceutical treatment has no effect on the survival” is accepted). The full interpretation of this test is that mice infected with pertussis do not survive better or worse with an adrenaline treatment. We can make a graph to find out whether the effect of pertussis is positive or negative.

5.5 Non-parametric tests

In the case the assumptions for ANOVA are not fulfilled (even not after transformation) you can go for a non-parametric alternative. These tests require fewer assumptions, but they are also less powerful.

Two sample Wilcoxon test (alternative for a t-test) and k-sample Kruskal-Wallis Rank test (alternative for 1-way ANOVA):

```
> wilcox.test(survival ~ farmac, data = Gluc)
```

```
Wilcoxon rank sum test with continuity correction
data: survival by farmac
W = 81.5, p-value = 0.6028
alternative hypothesis: true location shift is not equal to 0
```

```
> kruskal.test(survival ~ fb, data = Gluc)
```

```
Kruskal-Wallis rank sum test
data: survival by fb
Kruskal-Wallis chi-squared = 14.3988, df = 3, p-value = 0.002410
```

The results have to be interpreted in a similar way as the t-test and ANOVA: p is the probability to have a false H_0 (no differences between groups).

When a dataset with two independent grouping variables cannot be analysed with a 2-way ANOVA because the assumptions are not met, the use of a non-parametric alternative (PERMANOVA) can be a valid alternative. PERMANOVA is based on distances in multivariate space, and can be done in R using the command *adonis()* from the *vegan* package. However, PERMANOVA is out of the scope of the current course.

5.6 Post-hoc tests

When the F ratio of the ANOVA is higher than the tabulated vF value we can conclude that groups differ. However when there are more than two groups to compare, we do not know yet which groups differ. It is not recommended to perform multiple t tests because of the increasing risk of making type I errors. Therefore there are alternative tests, such as the Tukey test.

```
> x = aov(survival ~ farmac * bact, data = Gluc)
```

```

> TukeyHSD(x)

Tukey multiple comparisons of means
 95% family-wise confidence level

Fit: aov(formula = survival ~ farmac * bact, data = Gluc)

$farmac
      diff          lwr          upr      p adj
sal-adr   -2  -15.42303  11.42303  0.7591659

$bact
      diff          lwr          upr      p adj
p-n 26.83333  13.41030  40.25636  0.0004728

$'farmac:bact'
      diff          lwr          upr      p adj
sal:n-adr:n -6.166667 -31.6380108  19.30468  0.9043471
adr:p-adr:n  22.666667  -2.8046775  48.13801  0.0921197
sal:p-adr:n  24.833333  -0.6380108  50.30468  0.0576493
adr:p-sal:n  28.833333   3.3619892  54.30468  0.0230019
sal:p-sal:n  31.000000   5.5286558  56.47134  0.0136906
sal:p-adr:p   2.166667 -23.3046775  27.63801  0.9950944

```

This test compares all groups pairwise. The result is a table with the differences between means (*diff*), and the upper and lower boundaries of the 95% confidence interval. When 0 lies within the 95% confidence interval (lower and upper boundary differ in sign), then the groups are not significantly different. Remark: the Tukey test cannot be applied for a non-parametric test!

5.7 Exercises ANOVA

For each exercise:

1. Formulate Hypothesis
2. Create meaningful and correct graph
3. Perform analyses according to the hypothesis.
 - a. Test assumptions
 - b. Perform analysis and correct post-hoc when needed
 - c. Provide interpretation

Chicks

6 groups of chicks were fed on 6 different diets for several weeks. Is there a significant difference in weight due to the diets? Which diets give the best result?

```
> data(chickwts)
```

Plant growth

Plant growth is measured in 3 groups: two treatments and a control. Do the treatments have an effect on the growth?

```
> data(PlantGrowth)
```

Poison

Test the effects of three different poisons and four different treatments on the survival of a group organisms

```
> data(poisons, package = "boot")
```

6 Correlation and regression

6.1 Correlation

An important step in data analysis is to determine whether there are (linear) relationships between variables. In a first step, the relationship between variables can be investigated in scatterplot.

```
> pairs(Gluc)
```

The standard method for calculating a correlation between two variables is to determine the Pearson product moment correlation. For this method, both variables should be normally distributed.

6.1.1 Test for normality

The normality of each variable is examined in the same way as for ANOVA (see 5.1.1.). When both variables are normally distributed (possibly after transformation), Pearson correlation is used; in other cases, the Spearman rank correlation should be calculated.

6.1.2 Parametric correlation: Pearson product moment

The correlation coefficient r indicates the strength of the linear relationship between two variables. The coefficient of determination r^2 indicates the proportion of explained variance. When $r = +1$ or -1 , a perfect linear model is obtained; when $r=0$ there is no linear model. To get an explained variance of more than 50%, it is necessary to get an absolute R greater than 0.7 .

```
> cor(Gluc)
> cor(Gluc[3:5])
> cor(Gluc$survival, Gluc$glucose)
```

the function `cor.test()` provides more output:

```
> cor.test(Gluc$survival, Gluc$glucose)

Pearson's product-moment correlation

data: Gluc$survival and Gluc$glucose
t = -3.8258, df = 22, p-value = 0.0009214
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
-0.8250908 -0.3069374
sample estimates:
      cor
-0.6320725
```

We obtain a correlation coefficient of -0.63. As $p < 0.05$, we obtain a reliable correlation.

NOTE: in case of small datasets, we can obtain less reliable correlation coefficients.

6.1.3 Non-parametric correlation: Spearman rank

The Spearman rank correlation is a widely applied non-parametric method. It will not investigate the actual values but the relative order (ranks) of the data, and how that order differs between the two variables. Being a non-parametric test, it is less sensitive, and thus it will be less likely you obtain a p value < 0.05 . The

interpretation of r and r^2 is similar as for the Pearson product moment test. The "method" must be specified explicitly in the formula. `cor.test ()` gives more output

```
> cor(Gluc$survival, Gluc$glucose, method = "spearman")  
> cor.test(Gluc$survival, Gluc$glucose, method = "spearman")
```

6.1.4 General remark correlation

Outliers: a single outlier can strongly influence correlation. Therefore it is best to combine correlation analysis with visual inspection of the data.

Correlation does not imply **causation**: A correlation between two variables does not mean there is a causal relationship. The variation in two variables can both result from a third, non-measured variable.

6.2 Regression

6.2.1 Simple regression

In simple linear regression, a linear relationship between a single independent variable X and a dependent variable Y is assumed. The best fitting line through a cloud of data points is calculated. This is the line for which the sum of the squared vertical distances (or squared residuals) from all points to the regression line, is minimal. In other words, the residuals are minimized. After the regression is performed, two assumptions need to be tested:

1. no outliers
2. normal distribution of the residuals

Again, a transformation can help when the data do not meet the assumptions. If the residuals still do not meet the assumptions after transformation, other types of regression can be used but these fall outside the scope of this course.

We test the relationship between the survival of the mice (dependent variable) and the glucose concentration in their blood (independent variable). In a first step, the regression is performed, and the results are stored in the vector "x". In a second step, the content of this vector is called.

```
> x = lm(survival ~ glucose, data = Gluc)
> summary(x)

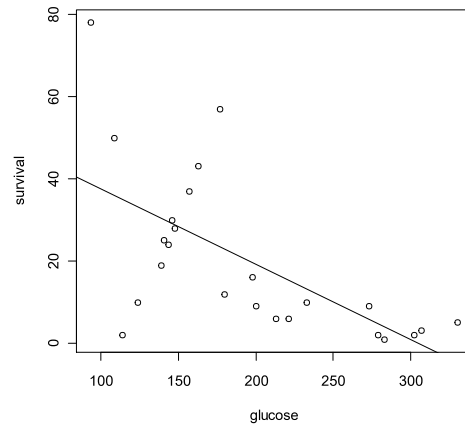
Call:
lm(formula = survival ~ glucose, data = Gluc)
Residuals:
    Min       1Q   Median       3Q      Max
-33.037  -9.559  -2.799   5.045  39.282

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  56.01974     9.92628   5.644 1.13e-05 ***
glucose      -0.18406     0.04811  -3.826 0.000921 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 16.03 on 22 degrees of freedom
Multiple R-Squared:  0.3995, Adjusted R-squared:  0.3722
F-statistic: 14.64 on 1 and 22 DF, p-value: 0.0009214
```

In the first part of the output, the actual regression equation is printed. The estimates of the parameters a and b from the regression equation are shown in the first column of the table (under "Estimate"). The intercept α (also sometimes called "constant") shows how much the regression line is shifted horizontally. If this value is close to zero, the regression line goes through the origin. The parameter b reflects the slope of the regression line. In the third ("t value") and fourth ("Pr >|t|") column, we see the value of the t-test and the corresponding p-value. This test shows whether both the a and b value are significantly different from zero. The t-test for b is in the case of the simple regression similar as the overall F test (same results). The null hypothesis of the t-test is that a or b are zero. Furthermore, also a multiple r -squared is shown here. This is a coefficient of determination R^2 , and indicates how much variation of the dependent variable Y is explained by the regression. The F-test will indicate whether the overall regression model is significant. When $p < 0.05$, the null hypothesis is rejected and the regression is significant. We can plot the regression line in two steps:

```
> plot(survival ~ glucose, data = Gluc) #plot the dependent variable
(survival) in function of the independent variable (glucose)
> abline(x) #draw the regression line
```



An extensive plot function exists for the linear regression object. You'll get some useful graphs for the regression analysis

```
> plot(x)
```

Note: the formula is implicitly assumes that the intercept of the y-axis is free to vary. If you want a regression through the origin, then this formula can be adapted by adding "-1" :

```
> x = lm(survival ~ glucose - 1, data = Gluc)
```

In this example, however, this makes no sense. Can you see why?

6.2.2 Multiple regression

A multiple regression is applied when several independent variables are used to explain the variation in a particular variable. One must be careful that the number of observations (cases) should always be 10 to 20 times higher than the number of independent variables.

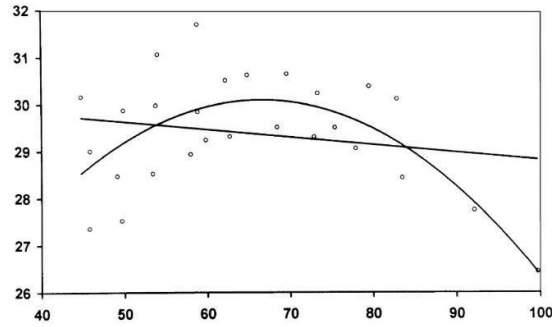
When preparing a multiple regression it is also important to consider the relationship between the independent variables. Therefore, we test if the independent variables are not multicollinear, i.e. correlated with each other. When this is the case, the independent variables are reflecting similar features and it is no longer possible to assess the influence of each independent variable. This results in a very large standard deviation of the regression coefficients, threatening the the validity of the model.

Multicollinearity can be checked by calculating

- **Correlation** between independent variables. If there are strong correlations between independent variables ($|r| > 0.9$), it is recommended to remove one of the highly correlated independent variables from the model.
- Another method to test for multicollinearity is calculating the **variance inflation factor (VIF)** (= $1/\text{tolerance value}$). As a rule of thumb, a VIF < 10 (or tolerance value > 0.1) indicates no multicollinearity.

```
> x=lm(survival~glucose+food, data = Gluc)
> library(car)
> vif(x)
```

Next to the measured values of independent variables (eg salinity, temperature), we can also add quadratic terms to the regression equation (even in simple regression). This can lead to a better regression line, because nonlinear relationships (e.g. see scatterplot below) are included in the model.



Furthermore, we can insert, next to the variables themselves, an interaction term in the regression equation. This term will quantify how e.g. the relationship between the density and temperature, is influenced by the depth. If consider the squares of each variable, we obtain the following regression equation (e.g. investigating the relationship between densities of marine organisms, and temperature of the water and water depth).
 $densiteit = \alpha + \beta_1 * temperature + \beta_2 * depth + \beta_3 * temperature^2 + \beta_4 * depth^2 + \beta_5 * temperature * depth + \epsilon$

This is the most complex model we can make with two independent variables depth and temperature (diepte en temperatuur) (= "full model"). In general this can be formulated as :

$$z_i = \alpha + \beta_1 x_i + \beta_2 y_i + \beta_3 x_i^2 + \beta_4 y_i^2 + \beta_5 x_i y_i + \epsilon$$

Between this full model and the simplest model (only intercept α) is a series of regression equations. We will now have to examine the optimal regression equation. A more complex equation will give better predictions, but there are more β 's we need to estimate and so we need more observations. We will therefore need to find the optimal regression equation for a given set of independent variables. This can be automated (stepwise method) or manually. Generally, the variables have to significantly explain the dependent variable and may not be multicollinear with other variables.

Below are two methods to arrive at a final regression model:

6.2.2.1 Manual method

A multiple regression command has the same structure as for a simple regression. The partial regression coefficients (β_i) are a measure of the contribution of the various independent variables to the regression. At-test examines whether the contribution of all selected independent variables is also significant. If a particular variable is not significant (partial t-test), then this variable contributes not significantly to the variation of Y, and the variable should be removed from the equation (the regression is redone without this variable).

We start from drie models:

1. We start with investigating the relationship between survival of the mice, and the independent variables food and glucose concentration. No interactions between glucose and food is assumed, so this is a linear model without interaction terms ($z = \alpha + \beta_1 x + \beta_2 y$).

```
> x=lm(survival~glucose+food, data = Gluc)
> summary(x)
```

```
Call:
lm(formula = survival ~ glucose + food, data = Gluc)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
```


-33.372 -7.456 -1.957 5.909 36.861

Coefficients:

```
              Estimate Std. Error t value Pr(>|t|)
(Intercept) 50.03895    11.99191   4.173 0.000430 ***
glucose     -0.18864     0.04859  -3.882 0.000861 ***
food         0.28492     0.31742   0.898 0.379574
```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 16.1 on 21 degrees of freedom

Multiple R-squared: 0.4217, Adjusted R-squared: 0.3666

F-statistic: 7.657 on 2 and 21 DF, p-value: 0.003181

- 1.1. Check if the regression is significant (global p)
- 1.2. If not, remove term with lowest significance value (highest p value) and redo regression
- 1.3. Check if the effect on R^2 and the p values
- 1.4. Repeat until you have a good model (all p values < 0,05)

2. Model with interaction terms

```
> y=lm(survival~food*glucose, data = Gluc)
> summary(y)
```

- 2.1. Go through the same steps as above until you have a good model. *Note:* if a certain variable is removed, than you also need to remove all interactions with that variable.

3. Full model with interactions and quadratic terms

3.1. Add quadratic terms

```
> food2=Gluc$food^2
> Gluc=data.frame(Gluc, food2)
> ...
```

- 3.2. Go through the same steps as above. *Note:* If the quadratic term of a variable is included, than the variable itself also needs to be included.

```
> z=lm(survival~food*glucose+food2+glucose2, data=Gluc)
> summary(z)
> ...
```

What if there are multiple good models?

- The higher R^2 , the better the model
- Simple models (less variables) are preferred
- Models (e.g. simple vs complex model) can be compared using ANOVA
`anova(model1, model2)`

6.2.2.2 Automated (stepwise) method

In a stepwise method, independent variables (and possibly their squares) are added or removed step by step to or from the model, based on an F-test. We feed the most complex model (full model). The method will search among the most complex and the most simple model (only an intercept) for the optimal model based on the Akaike Information Criterion (AIC). The AIC is used to choose the optimum regression model. It looks for a balance between the highest predictive power of a regression model and the number of parameters (terms) used in that model. Consequently, if two regression models give the same output, the most simple model will be elected. In a forward selection strategy (forward), in each step the independent variable with the highest F value is added. At the same time the influence of the variables already included in the model is taken into account. The variables are therefore added in order of their marginal, relative influence on the dependent variable. When the F-value of a certain independent variable is less than the minimum "F-to-enter" (FIN), the

variable is not included in the model. In the backward selection strategy (backward) for each step the variable with the lowest F value is each time removed from the model. If this value is greater than the "F-to-remove", the variable is not removed from the model. The "Both" method is a combination of forward and backward selection. Define the most complex model:

```
> x = lm(survival ~glucose*food + food2 + glucose2,data=Gluc)
```

Run as step wise analysis:

```
> y = step(x, direction = "both")
```

Start: AIC= 140.7

```
survival ~ glucose * food + food2 + glucose2
```

	Df	Sum of Sq	RSS	AIC
- food 2	1	12.3	5130.4	138.8
- glucose:food	1	165.9	5284.0	139.5
- glucose2	1	185.5	5303.7	139.6
<none>			5118.1	140.7

Step: AIC= 138.76

```
survival ~ glucose + food + glucose2 + glucose:food
```

	Df	Sum of Sq	RSS	AIC
- glucose:food	1	161.3	5291.7	137.5
- glucose2	1	178.9	5309.3	137.6
<none>			5130.4	138.8
+ food	1	12.3	5118.1	140.7

Step: AIC= 137.5

```
survival ~ glucose + food + glucose2
```

	Df	Sum of Sq	RSS	AIC
- food	1	63.6	5355.4	135.8
- glucose2	1	154.3	5446.0	136.2
- glucose	1	419.2	5710.9	137.3
<none>			5291.7	137.5
+ glucose:food	1	161.3	5130.4	138.8
+ food	1	7.7	5284.0	139.5

Step: AIC= 135.79

```
survival ~ glucose + glucose2
```

	Df	Sum of Sq	RSS	AIC
- glucose2	1	299.6	5655.0	135.1
<none>			5355.4	135.8
- glucose	1	666.7	6022.1	136.6
+ food	1	70.0	5285.4	137.5
+ food	1	63.6	5291.7	137.5

Step: AIC= 135.09

```
survival ~ glucose
```

	Df	Sum of Sq	RSS	AIC
<none>			5655.0	135.1
+ glucose2	1	299.6	5355.4	135.8
+ survival	1	211.5	5443.5	136.2
+ survival	1	208.9	5446.0	136.2
- glucose	1	3762.4	9417.3	145.3

Summary of the final model after stepwise analysis:

```
> summary(y)
```

Call:

```
lm(formula = survival ~ glucose, data = Gluc)
```

Residuals:

```
      Min       1Q   Median       3Q      Max
-33.037  -9.559   -2.799   5.045  39.282
```

Coefficients:

```
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  56.01974     9.92628   5.644 1.13e-05 ***
glucose      -0.18406     0.04811  -3.826 0.000921 ***
---
```

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 16.03 on 22 degrees of freedom

Multiple R-Squared: 0.3995, Adjusted R-squared: 0.3722

F-statistic: 14.64 on 1 and 22 DF, p-value: 0.0009214

6.2.2.2 Multiple regression: final notes

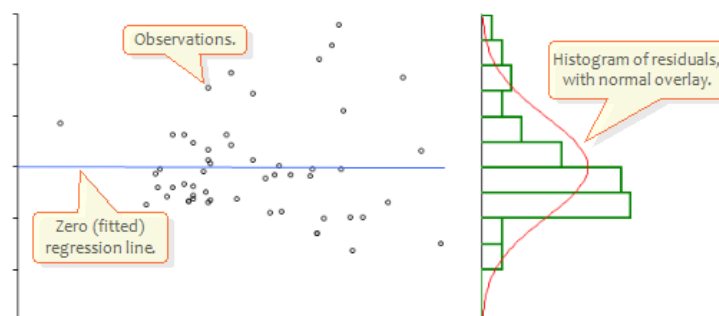
In a multiple regression pay attention to the following :

- Are there more cases than independent variables? (10 to 20 times more)
- Is the total regression significant? (F test)
- What percentage of the variation is explained by the combination of independent variables (R^2 adjusted = R^2 adjusted for the number of independent variables)
- Is there overlap (multicollinearity) between the selected variables?
- Is their contribution significant? (Partial regression t-test)

6.3 Testing Assumptions: residual analysis

After performing a simple or multiple regression, residuals should always be tested for normality and the absence of outliers. These must satisfy two conditions:

The following figure shows how the residuals should be normally distributed around the regression line.



We define outliers as residuals $resid()$ of which the absolute value $abs()$ is greater than three times the standard deviation $sd()$. We can remove the data for which the residuals are indeed > 3 times the standard deviation. Removing outliers improves the distribution function towards a normal distribution.

```
> x = lm(survival ~ glucose, data = Gluc)
> e = resid(x)
> e[abs(e) > 3 * sd(e)]
```

The last command selects the values from the vector e , that are larger than the 3 times the standard deviation. When outliers are detected, they can be removed, and the remaining residual values are assigned to a new vector $e1$.

```
> e1 = e[abs(e) < 3 * sd(e)]
```

Test normal distribution of the residual (with all outliers removed):

```
> shapiro.test(e1)
```

If the residuals (without outliers) are normally distributed, **remove the outliers from the data** and run the regression again, without the outliers.

The first command creates a new vector $Gluc1$, containing the data for which the residuals meet the assumption, the second command performs the regression analysis.

```
> Gluc1 = subset(Gluc, abs(e) < 3 * sd(e))
> lm(survival ~ glucose, data = Gluc1)
```

If the residuals are not normally distributed, we perform a transformation of the data. If they are still not normally distributed, we need to do a regression based on a different distribution of the residuals (not in this course).

6.4 Tasks

6.4.1 Glucose experiment

Look for a link between glucose and survival and interpret the results. Consider the usefulness of a linear regression analysis and argue. Consider carefully what you choose as the dependent and what as independent variable?

6.4.2 Analysis of the hyperbenthos data in 3 European estuaries

We consider the hyperbenthos density in three European estuaries from the first exercise (HyperRegressie.txt).

1. Is there a link between total hyperbenthos density and salinity, temperature or secchi?
2. If they relate, can the variation in hyperbenthos density be explained and described by a known function of one of the environmental variables, or a combination of some of these? Give a significant regression equation.