Introduction to

Part I

by
Knut Helge Jensen

Today's main tasks

1) Learn how to import data

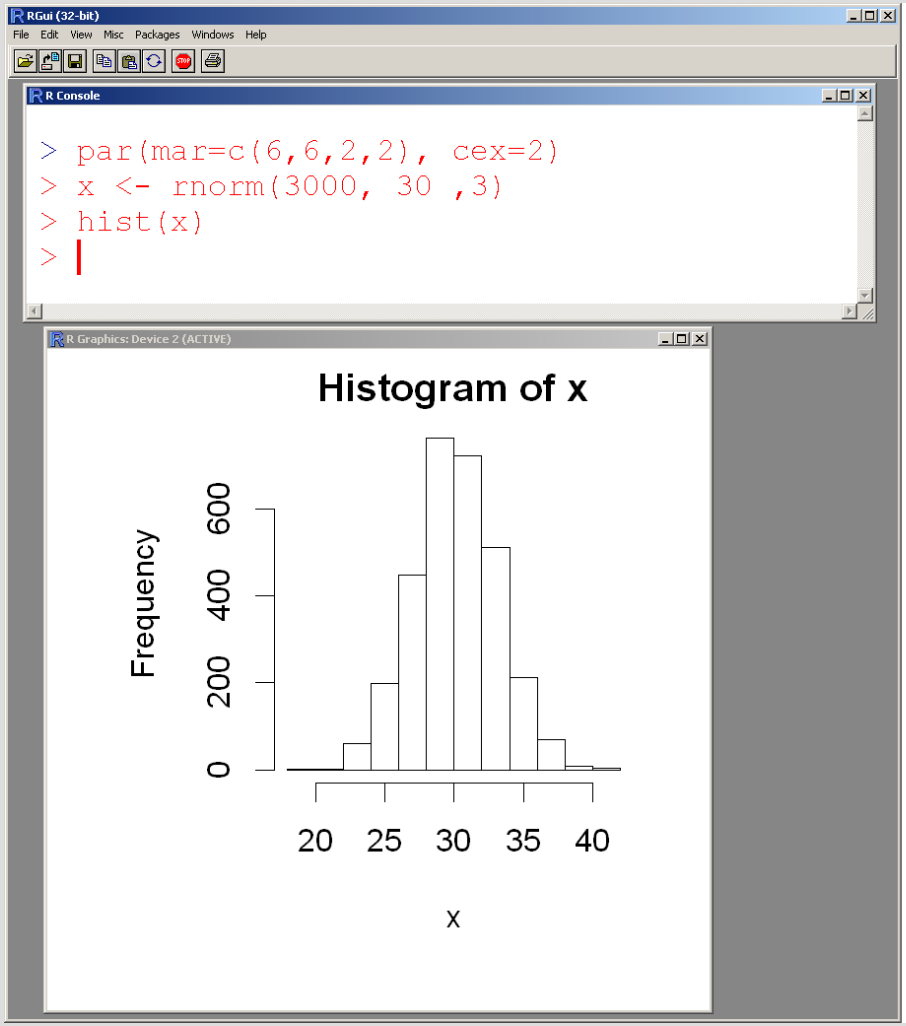
2) Introduction to plotting with ggplot2

What is  ?

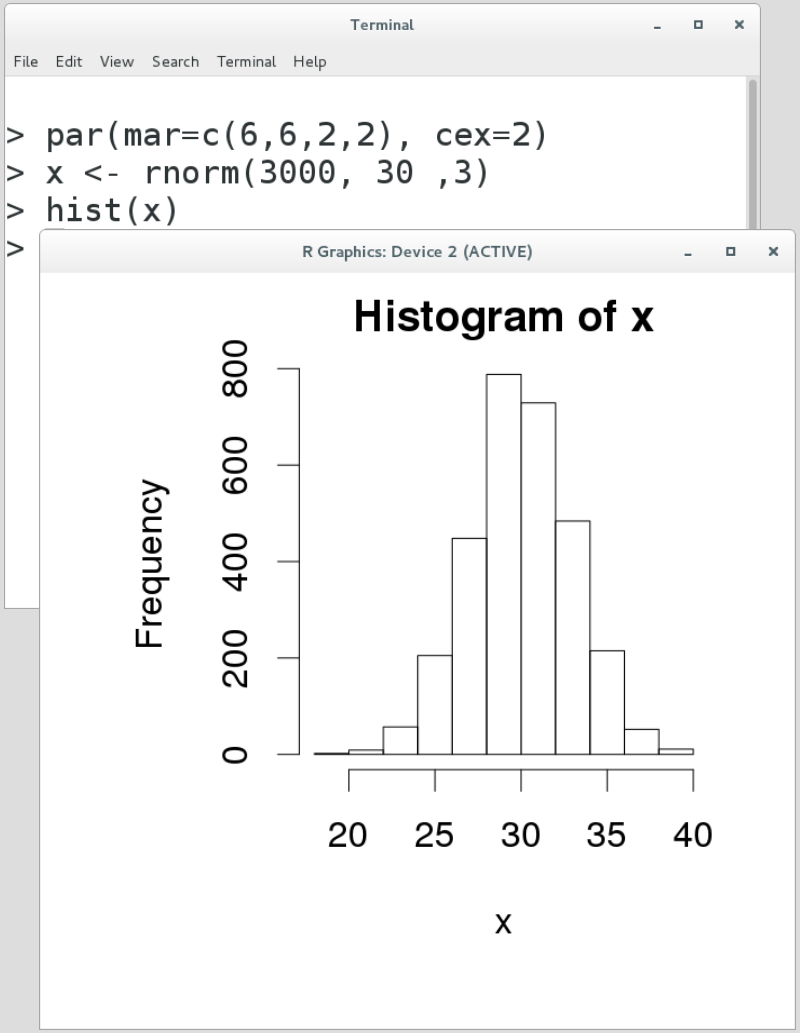
- It is a powerful statistical and graphical tool.
- It is an object oriented programming language.
  - Language organized around objects rather than actions.
  - Objects have data and functions.
- It has easy data import facilities via text files, clipboard, etc.
- It is open source software.
- It can be downloaded for free from (http://www.r-project.org/).
- It is available for most platforms (Windows, Mac, Linux, FreeBSD, etc.).

How R looks

Windows:



Linux:



Power of 

- Can work with multiple datasets and data structures at the same time
  Lists, data frames, vectors, matrices, arrays
- Datasets can be of any dimension
- Many people writing new functions, so capabilities of R are constantly expanding
- One can modify and build functions to suit personal needs
- Almost every analytical tool needed is available
- Easy to communicate about statistical analyses through exchange of code (R syntax) over email etc.

Disadvantages of 

- Steep learning curve.
  - Command driven (not "point and click").
  - Trade-off between flexibility and simplicity.

- Easy to make mistakes (and not know they were made).

- Learning R is very much "trial & error" at first.
  - Most of early mistakes are "typing errors":
    - Forget a comma or have wrong bracket matching.
    - Misspell a function or object: "Mean" instead of "mean".

A little  -teaser

You want to create a model for stopping distance (m) of a car depending on its speed (km/h). What is the programming needed to perform the following six tasks?

1) Import some external data for the relationship between the stopping distance and speed.
2) Make a **publication ready** plot of the relationship.
3) Make a linear regression model for the relationship.
4) Print summary statistics for the model (parameter estimates etc.).
5) Add a line to the plot that represents the model from task 3).
6) Add a 95% confidence interval for the line.

Answer

1)
```
demo.df <-
read.table("http://folk.uib.no/nzlkj/bio300b/stop.txt",
header=T)
```

2)
```
library(ggplot2)
p1 <- ggplot(demo.df, aes(Speed, Stop.distance))
p1 <- p1 + geom_point()
p1 <- p1 + theme_gray(base_size=24)
p1 <- p1 + labs(x="Speed (km/h)", y="Stopping distance (m)")
p1
```

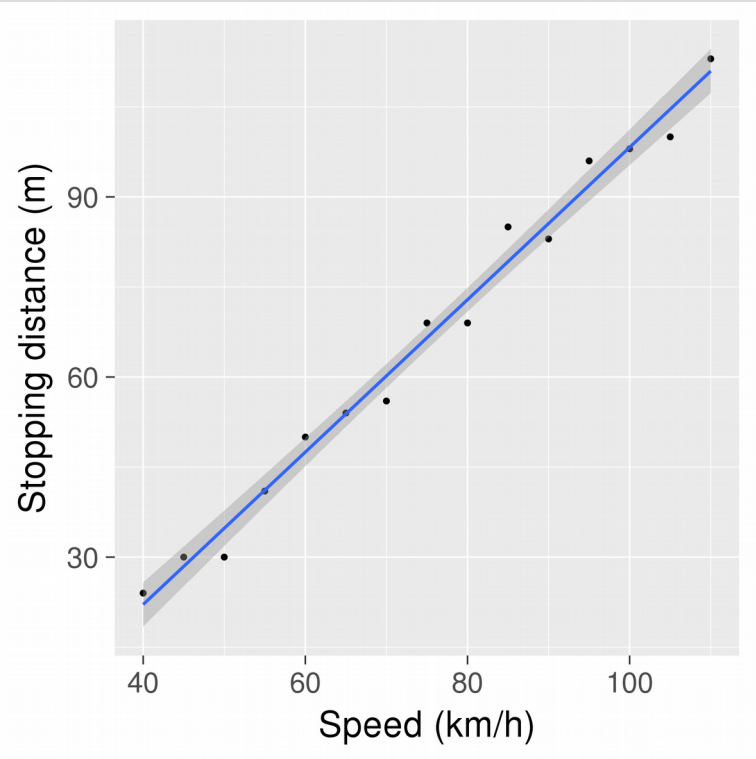
3)
```
fit1.lm <- lm(Stop.distance~Speed, data=demo.df)
```

4)
```
summary(fit1.lm)
```

5)
```
p1 <- p1 + geom_smooth(method = "lm", formula = y~x)
```
&
```
p1
```
6)

Resulting statistical output and plot

```
Call:
lm(formula = Stop.distance ~ Speed, data = demo.df)

Residuals:
    Min      1Q  Median      3Q     Max
-4.8012 -3.2262  0.1595  2.2542  5.7738

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -28.66310    3.23288  -8.866 7.13e-07 ***
Speed         1.26929    0.04142  30.643 1.65e-13 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.'

Residual standard error: 3.466 on 13 degrees of freedom
Multiple R-squared:  0.9863,  Adjusted R-squared:  0.9853
F-statistic:   939 on 1 and 13 DF,  p-value: 1.654e-13
```



How to start in 

Basics

Defining & accessing objects

Objects in R

R is an object oriented language

```
x <- 36.8
```

Here *x* is the name you have given the object, and the object contains a single number: 36.8.

The arrow (**<-**) is called the assignment symbol. Thus, in the above syntax you tell R to put 36.8 into an object called x.

If you after this for example write

```
x <- 12.5
```

it will not add 12.5 to your object but replace it with 12.5.

An object may of course contain much more than a single character or integer

A simple example of how to create a vector:

```
testdata <- c(30, 31, 32, 33, 34, 35, 36, 37,
              38, 39, 40)
```

The letter **c** is a frequently used R function. It combines values. Thus, by the above syntax you put numbers from 30 to 40 into an object called testdata.

Tip! Since the data is continuously increasing in equal steps, an easier way to write the above syntax is: **`testdata <- c(30:40)`**

When an object like **`testdata`** is created, it is easy to do calculations with it.

Let us assume that testdata is something that represents seconds. You want this variable to be measured in minutes instead. Thus, you need to divide each number with 60.

You do this simply by the following syntax:

```
testdata <- testdata/60
```

Note! Giving the same name here will overwrite the original variable. If you would like to keep the original variable, you could just write a different name on the left side of the assignment symbol.

How to make a screen print of an object you have created

<name-of-object> followed by Enter 

From the above object:

```
testdata
 [1] 0.5000000 0.5166667 0.5333333 0.5500000 0.5666667 0.5833333 0.6000000
 [8] 0.6166667 0.6333333 0.6500000 0.6666667
```

How to list objects

```
ls()
[1] "x"  "testdata"
```

How to remove objects

```
rm(<name-of-object>)
```

Example:

```
rm(testdata)
```

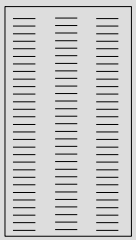
Note! If you need to remove all objects you have created:

```
rm(list=c(ls()))
```

Data structures in R

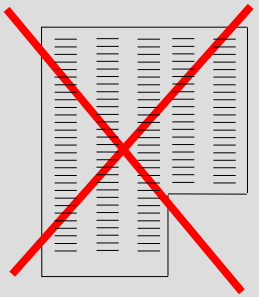
Vector – a series of numbers or characters. If the vector consists of both numbers and characters both will be read as characters. An exception is NA which is the default code for a missing value. **Note!** In this course, just think about vectors as variables.

Matrix – a series of vectors in a system of rows and columns. A matrix may differ in number of rows and columns but each column must have equal number of rows. A matrix can either contain characters or numbers, not both.



or

but not



...data structures continued

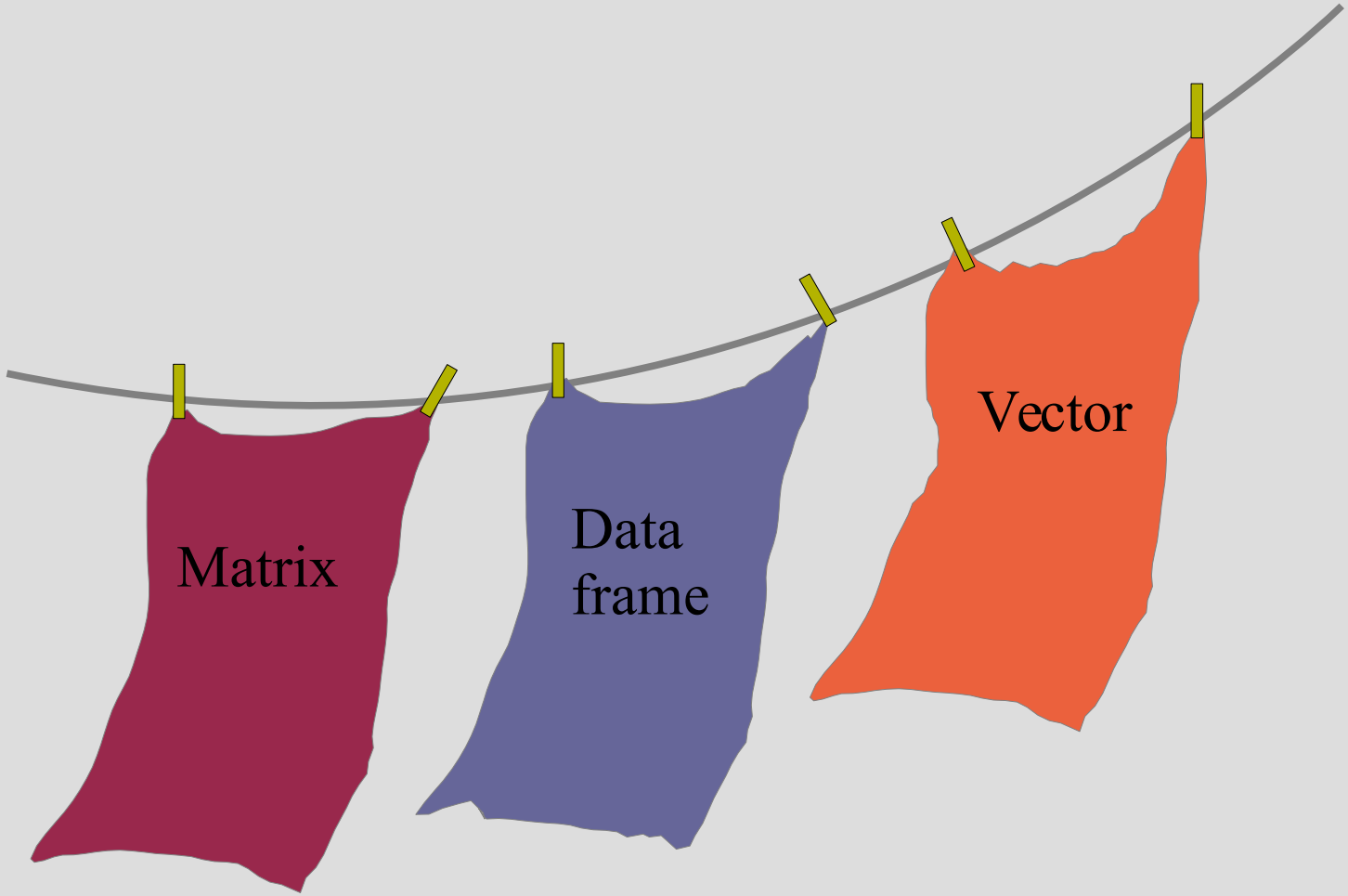
Array – a matrix of many dimensions, a matrix is a two dimensional array.

Data.frame – data frames are fundamental to the use of the R modelling and graphics functions. A data frame is a generalization of a matrix, in which different columns may have different modes (numerical or categorical):

response	predictor
1.23	A
2.45	A
1.11	A
2.35	A
5.78	B
4.32	B
3.31	B
6.98	B

...data structures continued

List – a list is the widest operative unit. You may combine vectors, matrices and data frames in the same list.



Functions in R

Examples of frequently used functions in R.

Function	Purpose
`read.table`	read a file in table format and creates a data frame from it
`attach`	attach a data frame to the search path
`names`	get or set the names of an object
`str`	compactly display the internal *str*ucture of an R object
`ls`	list objects
`rm`	remove object(s)
`c`	combine values into a vector or list
`data.frame`	create data frames
`cbind`	combine R objects by columns
`rbind`	combine R objects by rows
`plot`	plotting of R objects
`ggplot`	main function for easy making of advanced plots

Function	Purpose
`length`	get or set the length of vectors (including lists) and factors etc.
`mean`	calculate the arithmetic mean
`median`	calculate the median
`sqrt`	calculate square root
`t.test`	perform one and two sample t-tests on vectors of data
`lm`	to fit linear models
`lme`	to fit linear mixed effect models
`glm`	to fit generalised linear models
`anova`	compare models or create variance or deviance tables of models
`summary`	produce result summaries of various models
`predict`	make predictions from models
`resid`	extract residuals from a model

The use of R functions is a fast and flexible way to perform both simple and advanced tasks.

How to get help

Help from within R:

You often need to know more about a function before using it.
Example: Syntax help for the *lm* function (linear models):

```
?lm
```

You often need help to find a function.
Example: You have forgotten the name of the function for standard deviation:

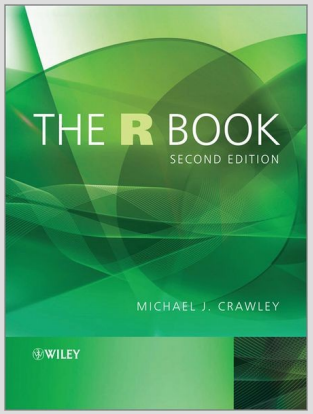
```
??"standard deviation"
```

Help from outside R:

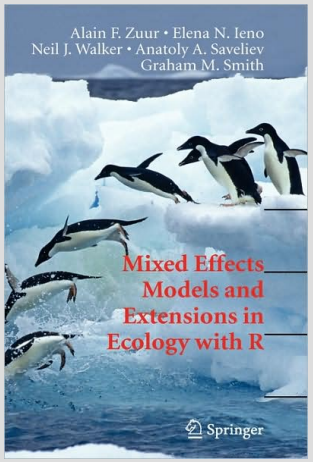
Search engines
A search engine like Google is your friend when working with R. Any question, someone else has most likely had it first.
Example of an internet search: r-help + ggplot

R-bloggers: http://www.r-bloggers.com

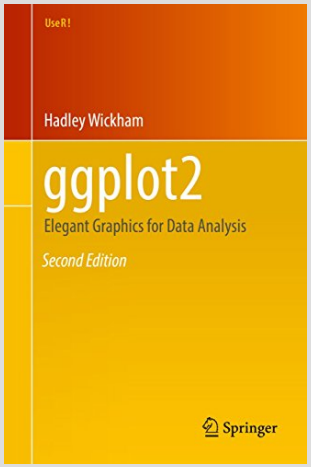
Recommended books about R



ISBN 978-0-470-97392-9



ISBN: 978-0-387-87457-9



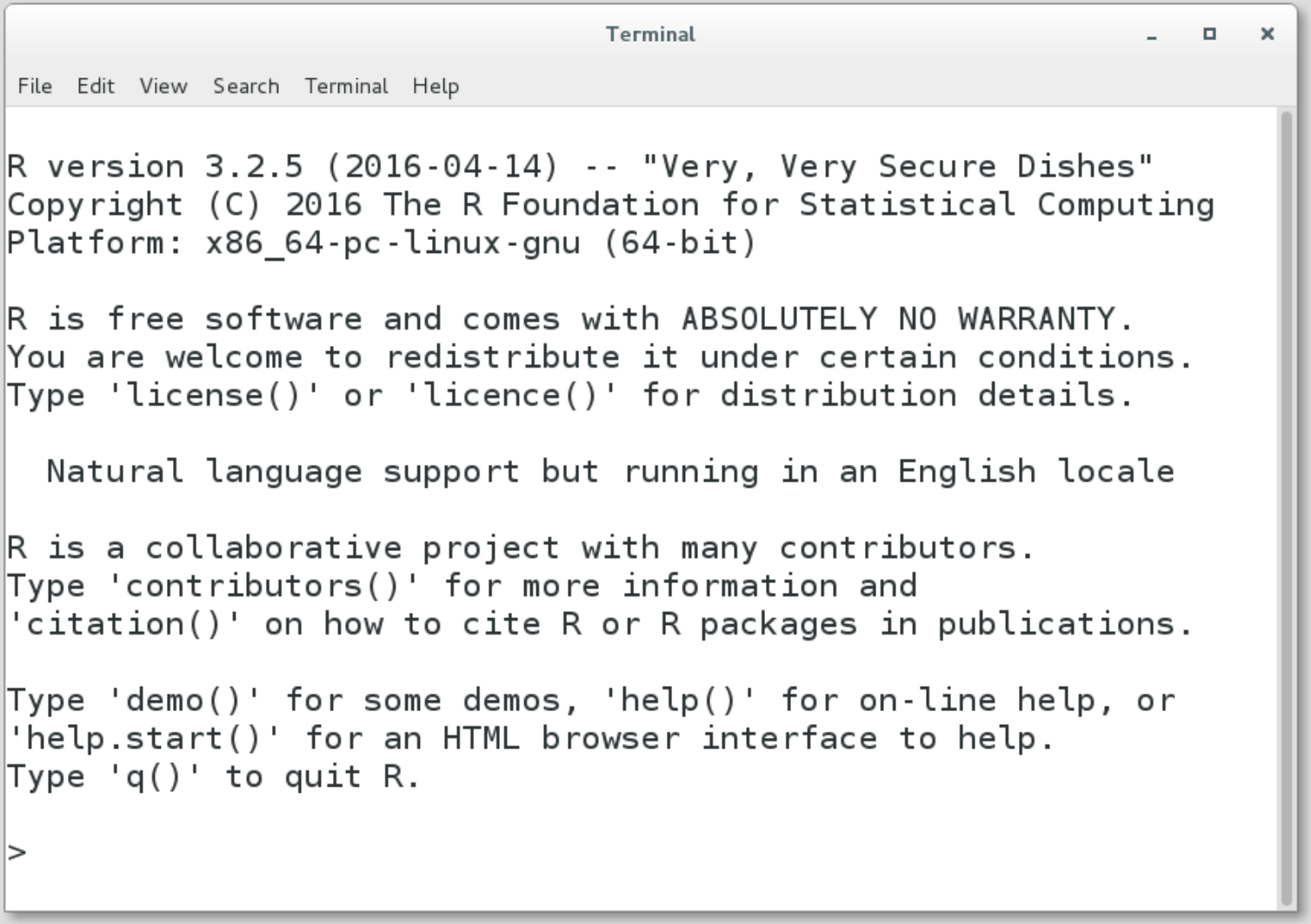
ISBN: 978-3-319-24275-0

Note1! Before you buy books, remember to check if you are allowed free download through your institution.

Note2! You also find a lot of free documentation at R's website: http://www.r-project.org/

The R command window

Nearly everything you do in R is through syntax commands



Do not write your commands directly into the command window of R - **use a text editor!!!**

Open | syntax.example.r | ~/work/kurs/2017/bio300b/day1 | Save

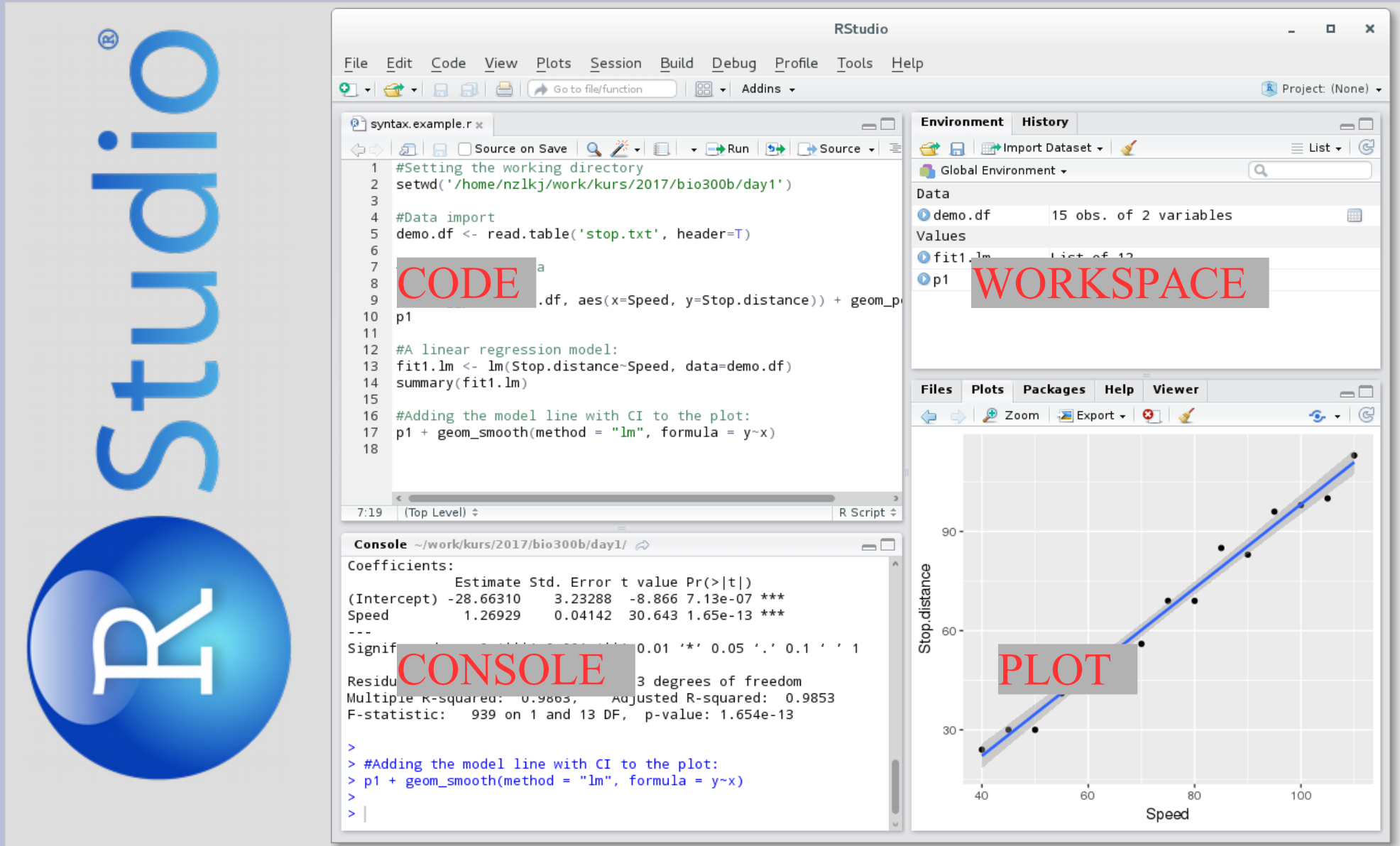
```
1 #Setting the working directory
2 setwd('/home/nzlkj/work/kurs/2017/bio300b/day1')
3
4 #Data import
5 demo.df <- read.table('stop.txt', header=T)
6
7 #Plotting the data
8 library(ggplot2)
9 p1 <- ggplot(demo.df, aes(x=Speed, y=Stop.distance)) + geom_point()
10 p1
11
12 #A linear regression model:
13 fit1.lm <- lm(Stop.distance~Speed, data=demo.df)
14 summary(fit1.lm)
15
16 #Adding the model line with CI to the plot:
17 p1 + geom_smooth(method = "lm", formula = y~x)
```

R | Tab Width: 8 | Ln 16, Col 28 | INS

Examples of text editors for the three most common desktop operative systems

Linux	Mac	Windows
Gedit	TextEdit	Notepad
Kate	Brackets	Notepad++
Geany	TextWrangler	Tinn-R
Notepadqq	TextMate	Crimson Editor
Emacs	Emacs	Emacs
Vi improved	Vi improved	Vi improved
RStudio	RStudio	RStudio

A recommended text editor that integrates the things you do into a single environment: https://www.rstudio.com/



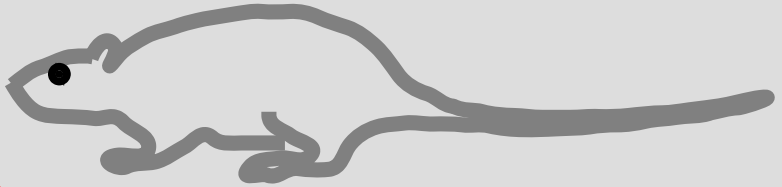
Preparing data for R

(or any other statistical package)

Think in terms of variables

Example: Body mass (g) of *Rattus norvegicus* depending on sex.

Sex	Body.mass
female	240.62
female	235.55
female	249.91
female	258.42
female	272.92
male	361.29
male	353.17
male	347.15
male	353.11
male	347.75

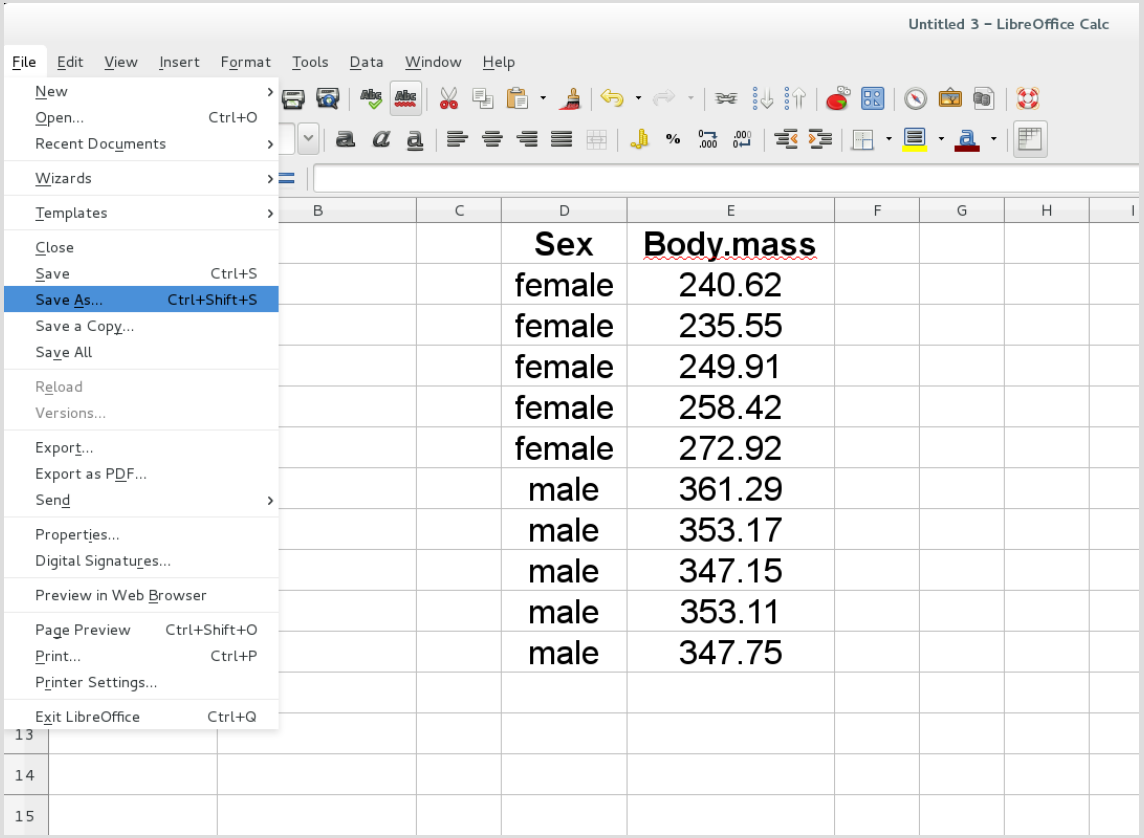


Females	Males
240.62	361.29
235.55	353.17
249.91	347.15
258.42	353.11
272.92	347.75

From spreadsheet to text file format

When you have a spreadsheet **only** containing the data including names of each variable,

go to "Save as" on the "File" menu and choose a plain text file format like for instance csv or txt.



Setting a working directory within R

It is recommended to set a working directory for each project you do in R. This will make it easier to collect your current work within the same directory. You do this by the **setwd** function in R or just by "point & click" if you are using RStudio (Session -> Set Working Directory -> Choose Directory).

Example (Note that the directory must exist before setting it in R):

```
setwd("/home/nzlkj/rintro/day1")
```

You may check what is your current working directory by typing

```
getwd()
```

How to import data

(and create what R calls a **data frame** which is the same as a dataset)

Use the *read.table* function:

From a text file:

```
rats.df <- read.table("rats.txt", header=T)
```

This assumes that you have a dataset file called rats.txt within your current working directory.

From the clipboard:

Windows, Linux, and FreeBSD users:

```
rats.df <- read.table("clipboard", header=T)
```

MacOS users:

```
rats.df <- read.table(pipe("pbpaste"), header=T)
```

Adapting *read.table* according to the format of the text file you want to import

The most important settings to know about:

```
rats.df <- read.table("textfile.txt", header=T, dec=".",
sep="", na.strings="NA")
```

Explanations

`header=T` Does the dataset contain names of variables (T) or not (F).

`dec="."` The decimal symbol in your dataset.

`sep=""` The separator between variables
Space: **`sep=""`** Tabulator: **`sep="\t"`**
Comma: **`sep=","`** Semicolon: **`sep=";"`** etc.

`na.strings` Definition of how missing values are coded.

It is also possible to add data directly into the R syntax instead of importing via an external text file

```
rats.df <- read.table(header=T, text="
Sex Body.mass
female 240.62
female 235.55
female 249.91
female 258.42
female 272.92
male 361.29
male 353.17
male 347.15
male 353.11
male 347.75
")
```

Note! Only use space (not tabulator!) as separator between variables when using this method. Additionally, the decimal symbol must be period (.).

How to list names of variables in your data frame

names(*name-of-data-frame*)

From the above example:

```
names(rats.df)
[1] "Sex"         "Body.mass"
```

How to check the structure of your data frame

str(*name-of-data-frame*)

From the above example:

str(rats.df)

```
'data.frame': 10 obs. of  2 variables:
$ Sex       : Factor w/ 2 levels "female","male": ...
$ Body.mass : num  241 236 250 258 273 ...
```

How to view all variables

View all variables but only the six first or last rows, respectively:

head(*name-of-data-frame*)
tail(*name-of-data-frame*)

From the above example:
head(rats.df)
tail(rats.df)

```
      Sex Body.mass
1  female    240.62
2  female    235.55
3  female    249.91
4  female    258.42
5  female    272.92
6    male    361.29
```

View the whole dataset in a separate window looking like a spreadsheet:

View(rats.df)

Rules for object names

- R is case sensitive! Body.mass and body.mass is NOT equivalent.
- Avoid open space: Body.mass NOT Body mass
- Avoid giving the same name to a **data frame object** as you have given to one of your variables within a data frame object:

~~`Body.mass <- read.table("rats.csv"...`~~

- Only use letters from the English alphabet.
- Avoid names reserved for functions.
- Do not use names starting with numbers.

The content of a dataset file called rats.csv

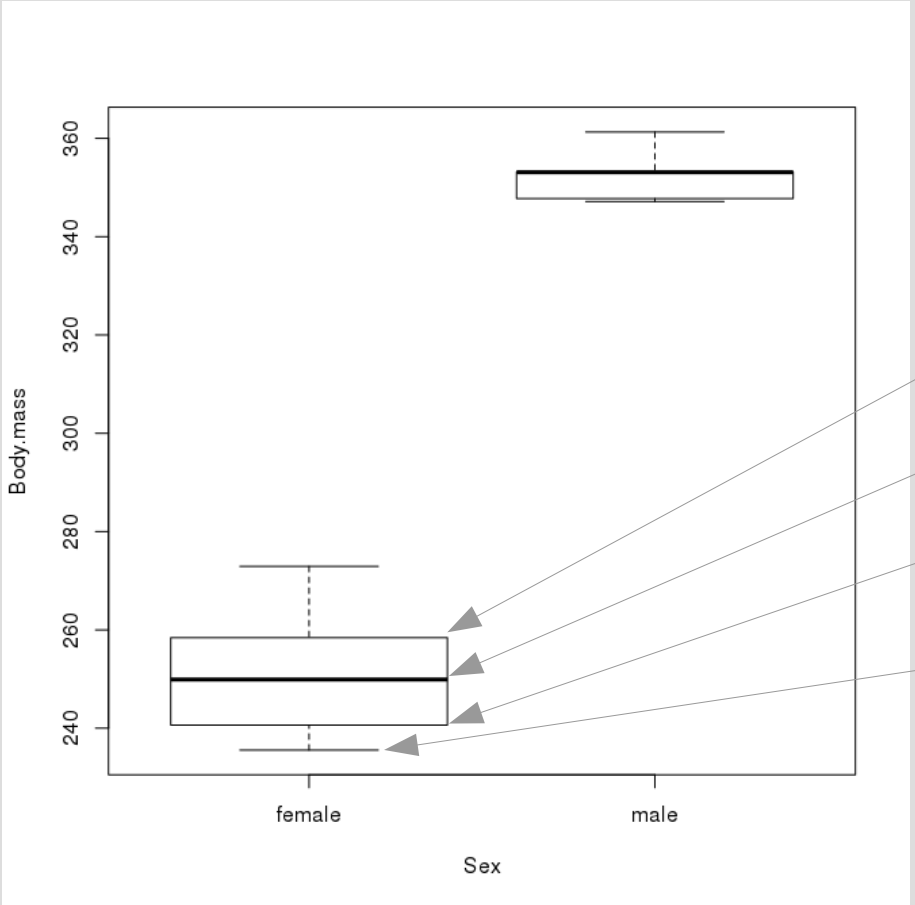
Sex	Body.mass
female	240.62
female	235.55
female	249.91
female	258.42
female	272.92
male	361.29
male	353.17
male	347.15
male	353.11
male	347.75

& graphics

A short introduction to graphics with the standard plotting module of R

As an example with the **`rats.df`** dataset, the following is R syntax for a simple plot with categorical x-axis:

```
plot(Body.mass~Sex, data=rats.df)
```



75th percentile (Q3)

50th percentile (median)

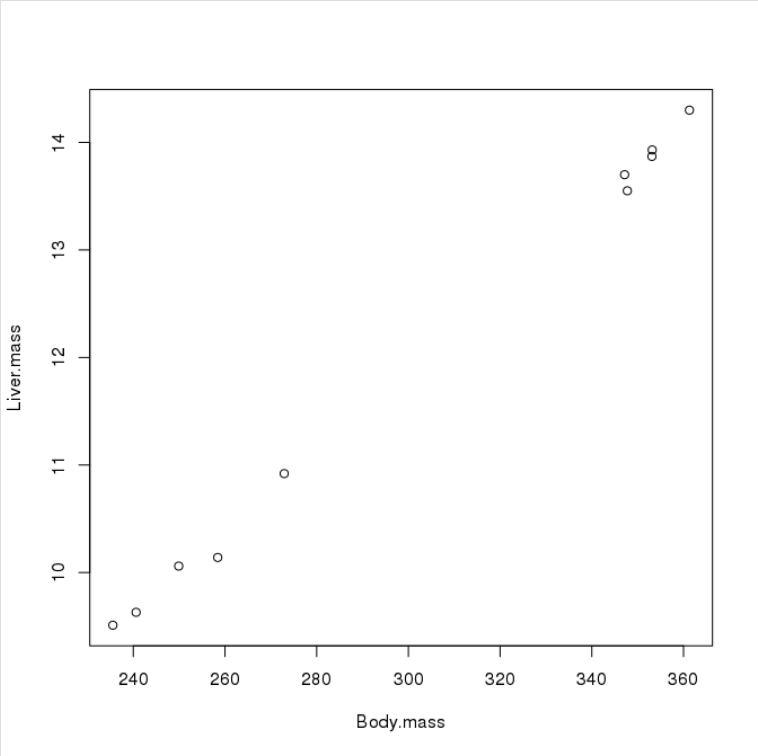
25th percentile (Q1)

Error bars: Minimum and maximum values except for eventual outliers. Outliers marked as solid circles and defined as 1.5 times the length of the box away from the box.

…the standard plotting module continued

Assume that the rats.df dataset also contained a third variable called Liver.mass representing liver mass of each individual rat. If you were to plot Liver.mass against Body.mass, the syntax would follow the same principle: `plot(y~x, data=df)`:

```
plot(Liver.mass~Body.mass, data=rats.df)
```



Thus, the standard plotting module of R automatically creates a box plot when x is categorical and a scatter plot when x is numerical.

…the standard plotting module continued

The standard plotting module of R is very flexible in what kind of figures you can create.

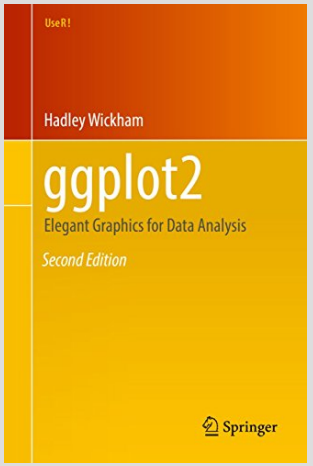
To have an idea about the flexibility, write:

`demo(graphics)` in R and follow the instructions.

The standard plotting module can produce almost any type of plot. Still, we will focus on ggplot2 for the rest of this course because it is less syntax demanding when you want to produce complex figures.

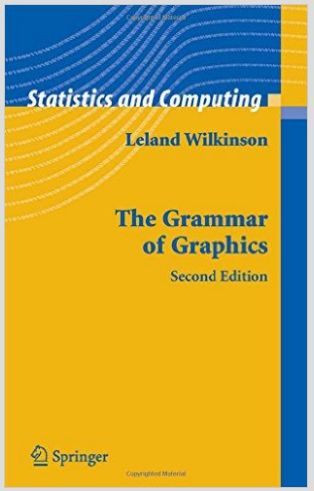
& graphics by using ggplot2

Written by Hadley Wickham



ISBN: 978-3-319-24275-0

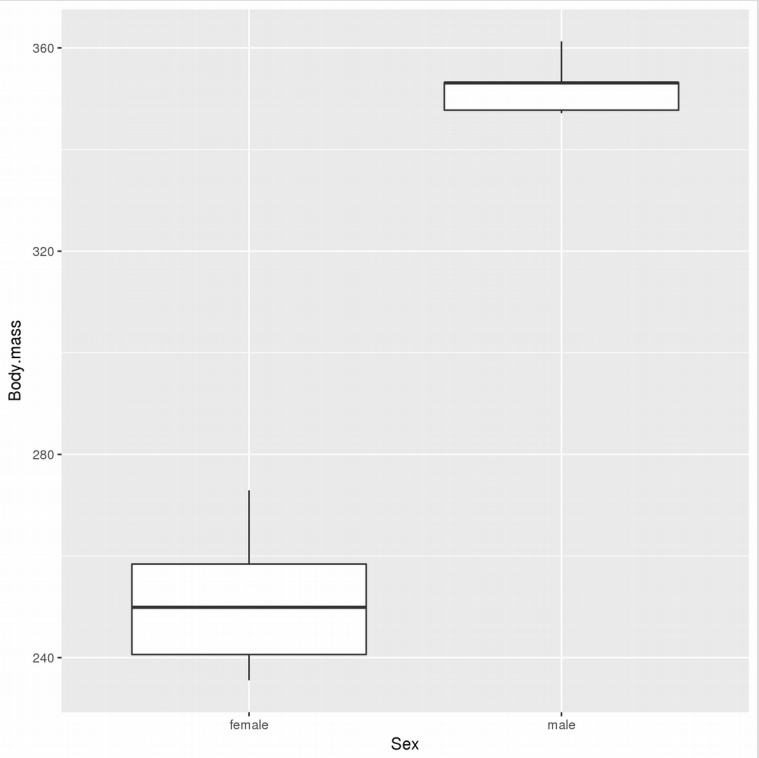
Implementation of The Grammar of Graphics, Written by Leland Wilkinson



ISBN: 978-0387245447

The same plot as the first we created with the rats.df dataset, i.e. Sex as categorical predictor and Body.mass as response variable

```
library(ggplot2)
p1 <- ggplot(data=rats.df, aes(x=Sex, y=Body.mass))
p1 <- p1 + geom_boxplot()
p1
```



Explaining the syntax

```
p1 <- ggplot(data=rats.df, aes(x=Sex, y=Body.mass))
p1 <- p1 + geom_boxplot()
p1
```

ggplot is the function for starting a plot. Else, you see that the structure of ggplot syntax is to add components step by step (**p1 + ...**).

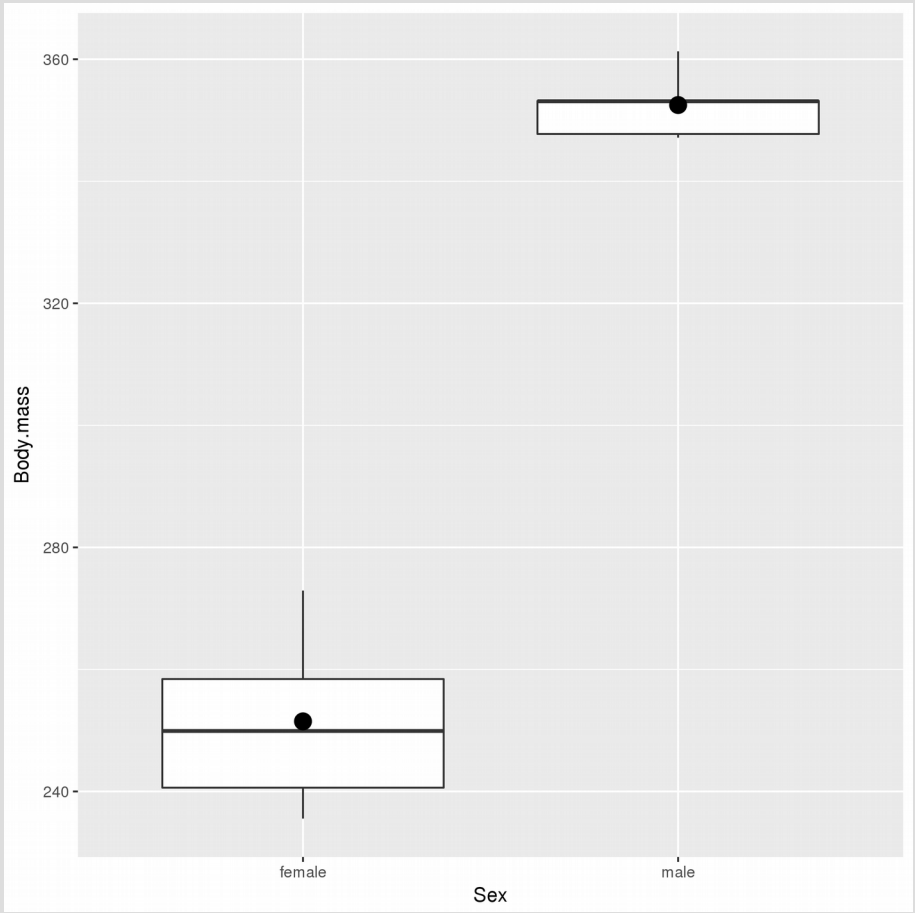
rats.df A ggplot always start with the name of the dataset you want to use for plotting, here **rats.df**. You may drop typing data= and just type **rats.df**

aes is the aesthetics: How to map the columns in the dataset onto the visual properties of the plot. In this case the variables Sex and Body.mass are placed on the x- and y-axis, respectively. You may drop x= and y= and just type:
aes(Sex, Body.mass)

geom_boxplot() defines what to draw. Possible to use more than one type of geometry. Two other examples: geom_point(), geom_line()

How to add mean values to the boxplot

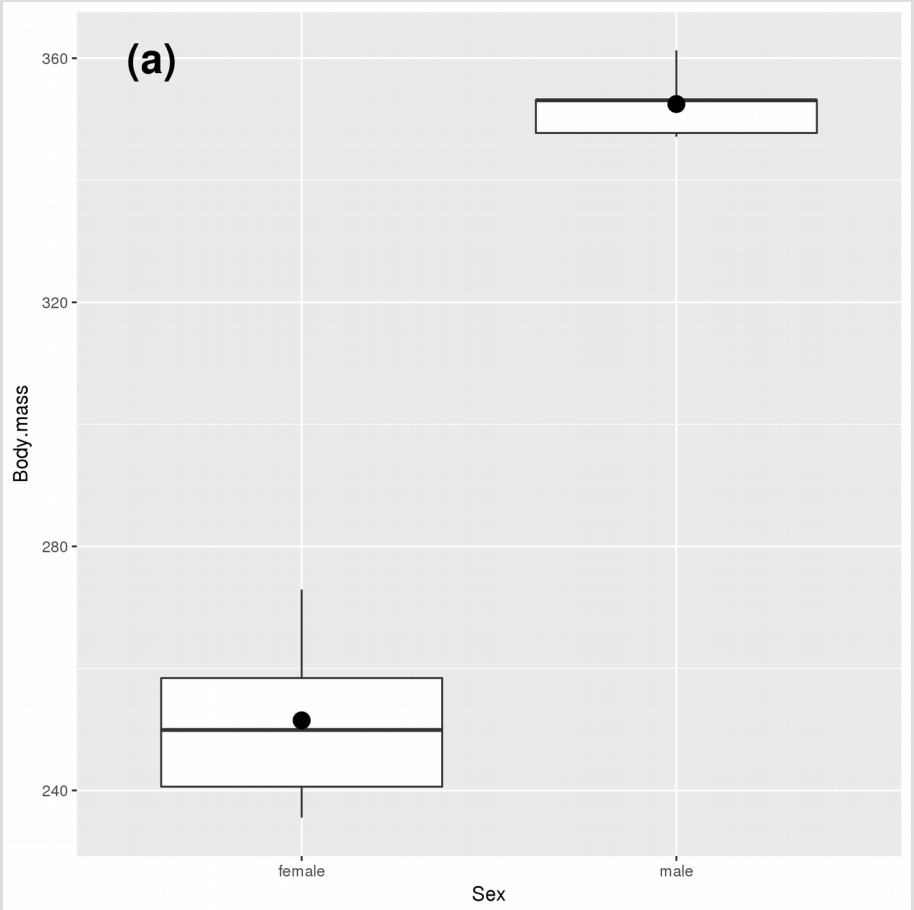
```
p1 <- p1 + stat_summary(fun.y="mean",
geom="point", colour="black", size=4)
p1
```



Type `?stat_summary` for information about syntax.

How to add annotations

```
p1 <- p1 + annotate("text", x=0.6, y=360,
label="(a)", fontface=2, colour="black", size=8)
p1
```



Type `?annotate` for information about syntax.

Plots with two categorical predictors

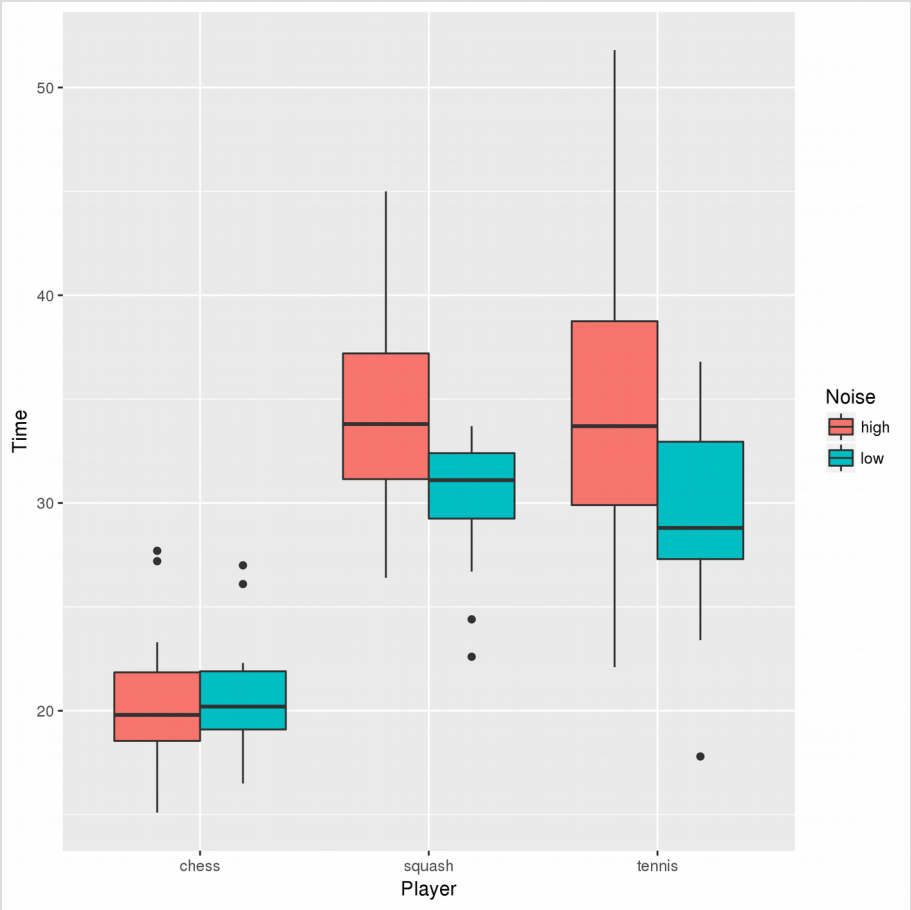
Case: Time spent (seconds) on solving a mathematical problem tested among professional tennis, chess and squash players and with low and high noise level. The latter to look at ability to concentrate under disturbance.

Noise	Player	Time
Low	tennis	27
Low	tennis	27.6
High	tennis	32.4
High	tennis	31.1
Low	chess	20.7
Low	chess	26.1
High	chess	16.8
High	chess	19.3
Low	squash	31.1
Low	squash	32.4
High	squash	41.3
High	squash	33.8

```
#Importing the dataset to R:
solving.time.df <-
read.table("http://folk.uib.no/nzlkj/data/solving.time.txt",
header=T, sep=",")
```

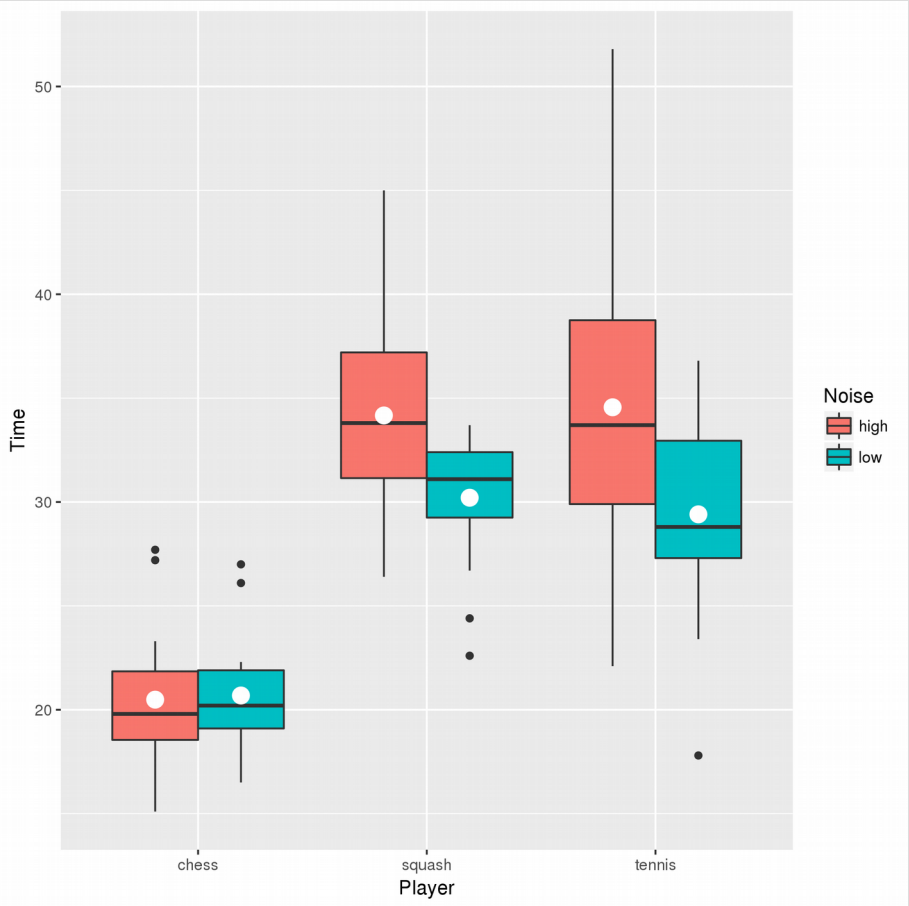
…two categorical predictors continued

```
p2 <- ggplot(solving.time.df, aes(Player, Time, fill=Noise))
p2 <- p2 + geom_boxplot()
p2
```



How to add mean values when you have two categorical predictors:

```
p2 <- p2 + stat_summary(fun.y="mean", geom="point",
colour="white", size=4,
position=position_dodge(width=0.75), show.legend=F)
p2
```



The setting **`width=0.75`*** is the total width of each pair of adjacent boxes. This information will place the points in the middle of each box.

The setting **`show.legend=F`** is to avoid a legend for these points to appear in the plot.

*Value independent of how many adjacent boxes you have.

Plots with one categorical and one numerical predictor

Time spent (seconds) on solving a mathematical problem tested among professional tennis, chess and squash players and with continuously increasing noise level (0-100 dB). The latter to look at ability to concentrate under disturbance.

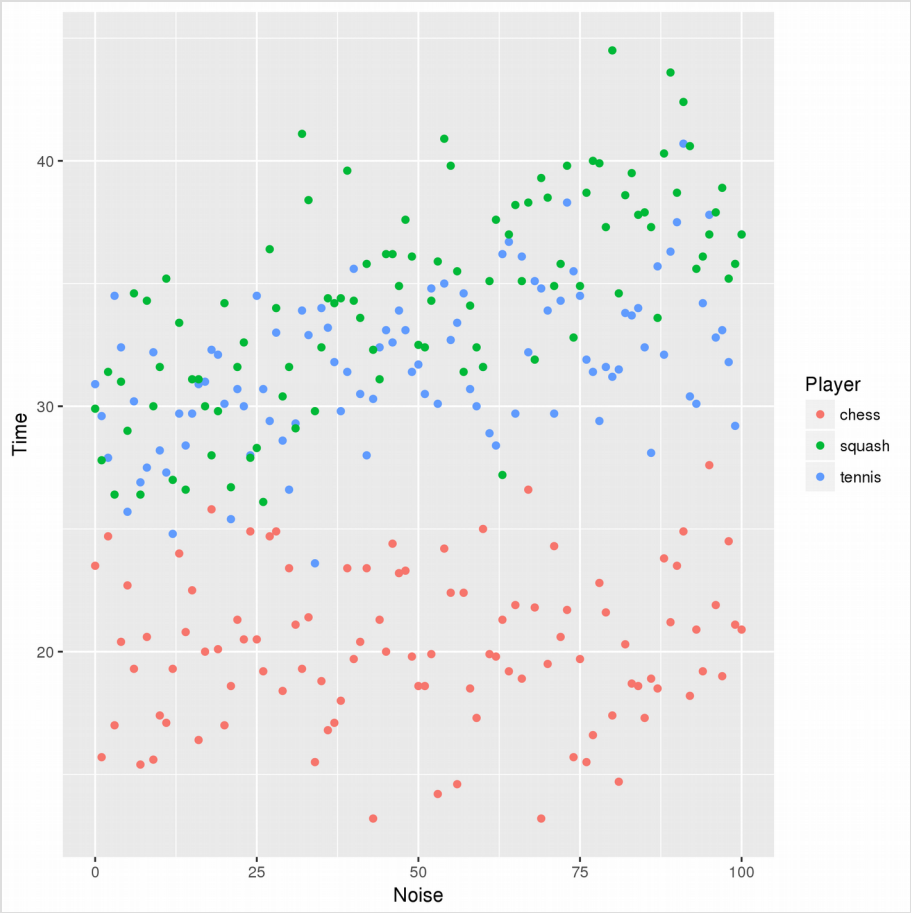
Noise	Player	Time
0	tennis	30.9
1	tennis	29.6
2	tennis	27.9
etc.	tennis	etc.
0	chess	23.5
1	chess	15.7
2	chess	24.7
etc.	chess	etc.
0	squash	29.9
1	squash	27.8
2	squash	31.4
etc.	squash	etc.

```
#Importing the dataset to R:
solving.time2.df <-
read.table("http://folk.uib.no/nzlkj/data/solving.time2.txt",
header=T, sep=",")
```

one categorical and one numerical predictor continued...

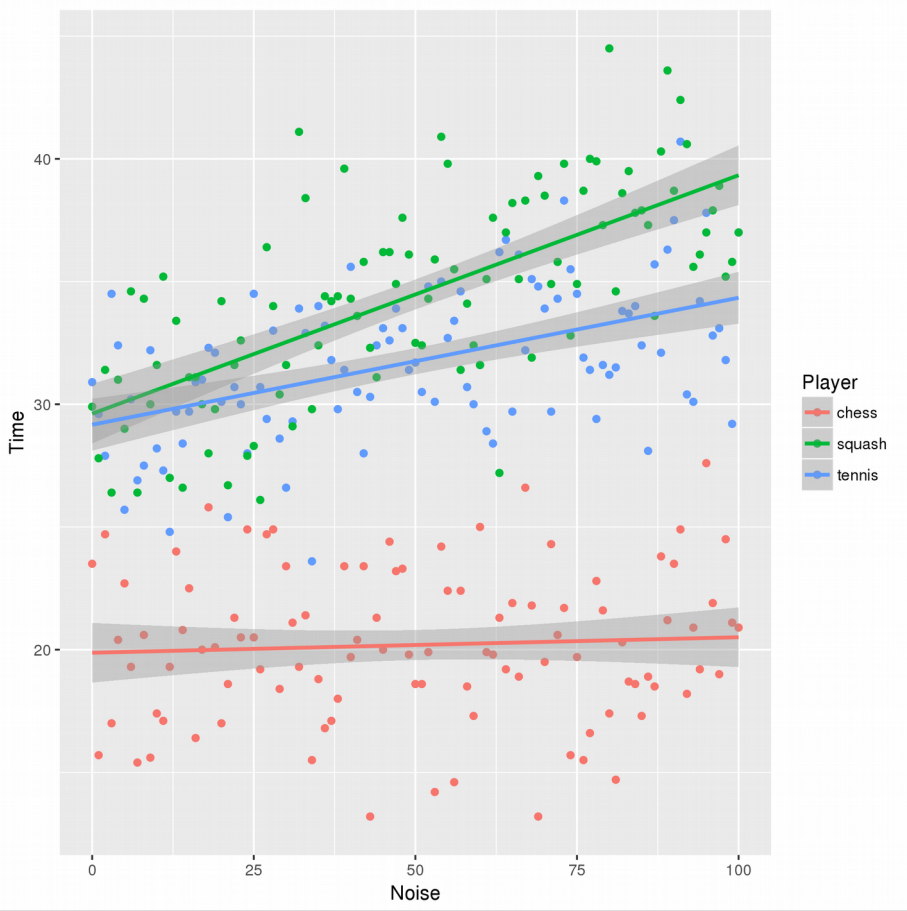
```
p3 <- ggplot(solving.time2.df, aes(Noise, Time, colour=Player))
p3 <- p3 + geom_point()
p3
```

Note! You may replace *colour* with *shape* if you want to have different symbols instead of different colours for each level of Player. It is also possible to include both.



How to add regression lines with 95% confidence intervals

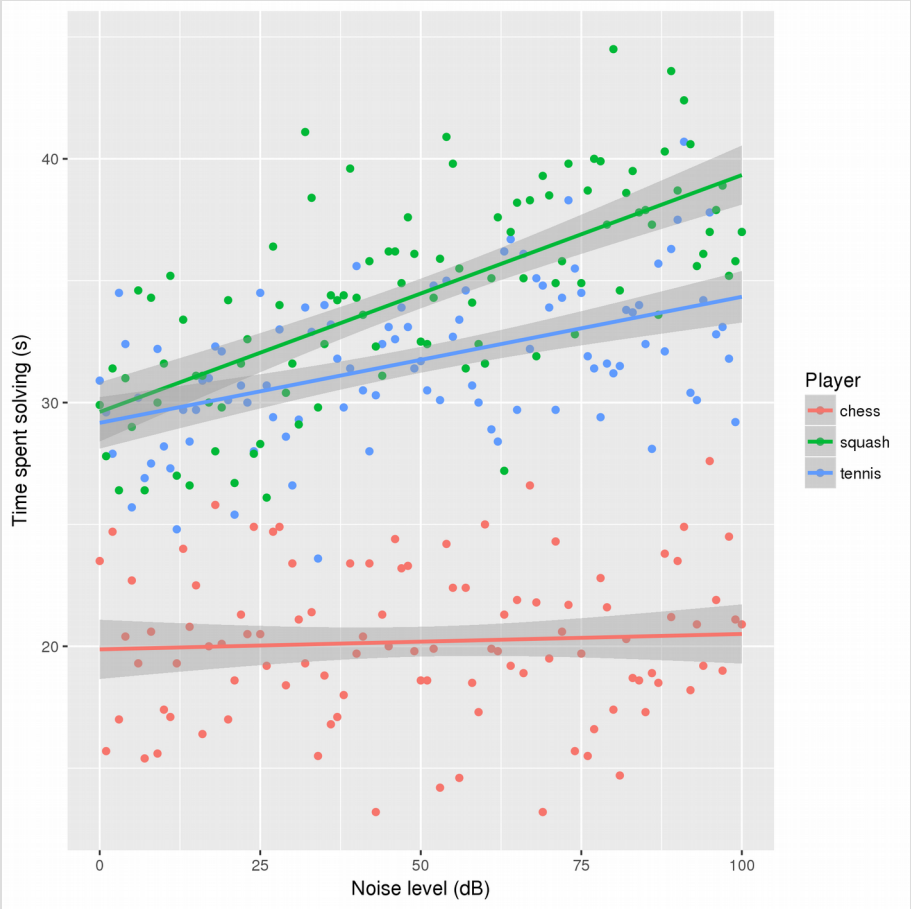
```
p3 <- p3 + geom_smooth(method = "lm", formula =
y~x, aes(colour=Player))
p3
```



Note! If you do not want the intervals around the regression lines, add the statement **`se=F`** within geom_smooth.

How to change axis titles

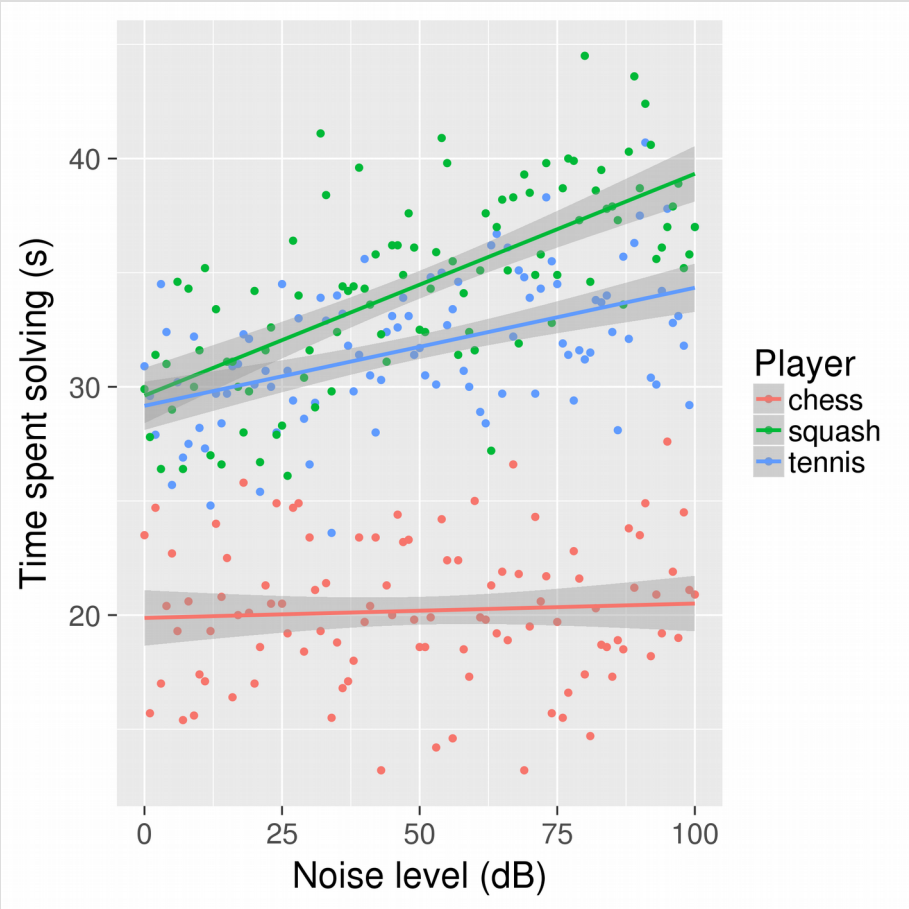
```
p3 <- p3 + labs(x="Noise level (dB)",
y="Time spent solving (s)")
p3
```



Type **`?labs`** for how to add plot title, subtitle, caption etc.

How to change font size

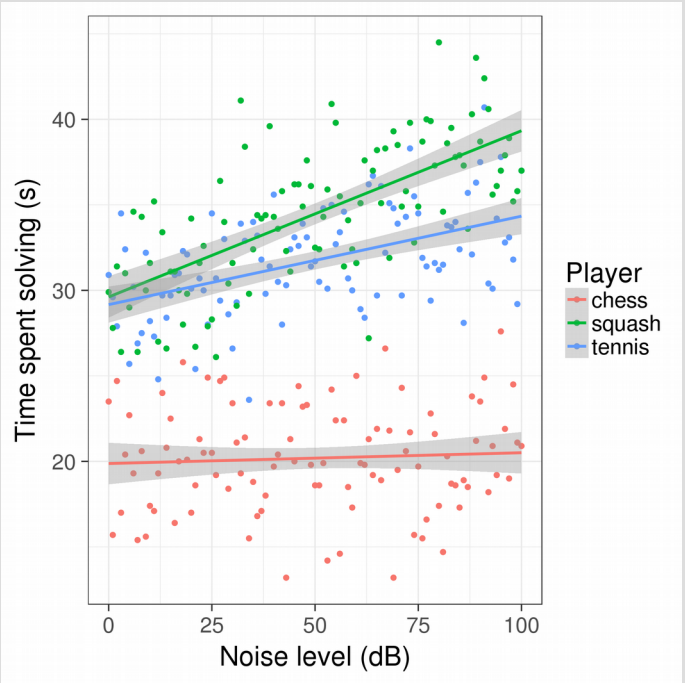
```
p3 <- p3 + theme_grey(base_size=20)
p3
```



How to change the look of plots with themes

Don't like default plot? Change it with themes

```
p3 <- p3 +
theme_bw(base_size=20)
p3
```



Different themes where **`theme_grey()`** is default:

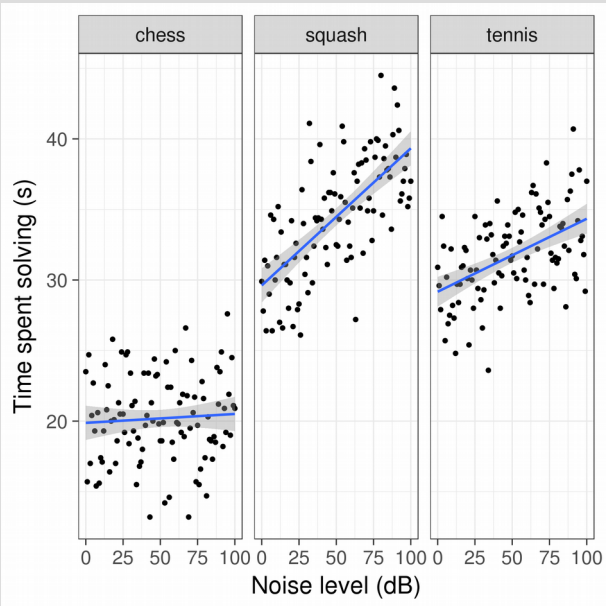
`theme_grey(), theme_gray(), theme_bw(), theme_linedraw(), theme_light(), theme_dark(), theme_minimal(), theme_classic(), theme_void()`

How to split a plot into partitions based on levels of your categorical predictor(s).

```
facet_wrap() #By one variable
facet_grid() #By two variables
```

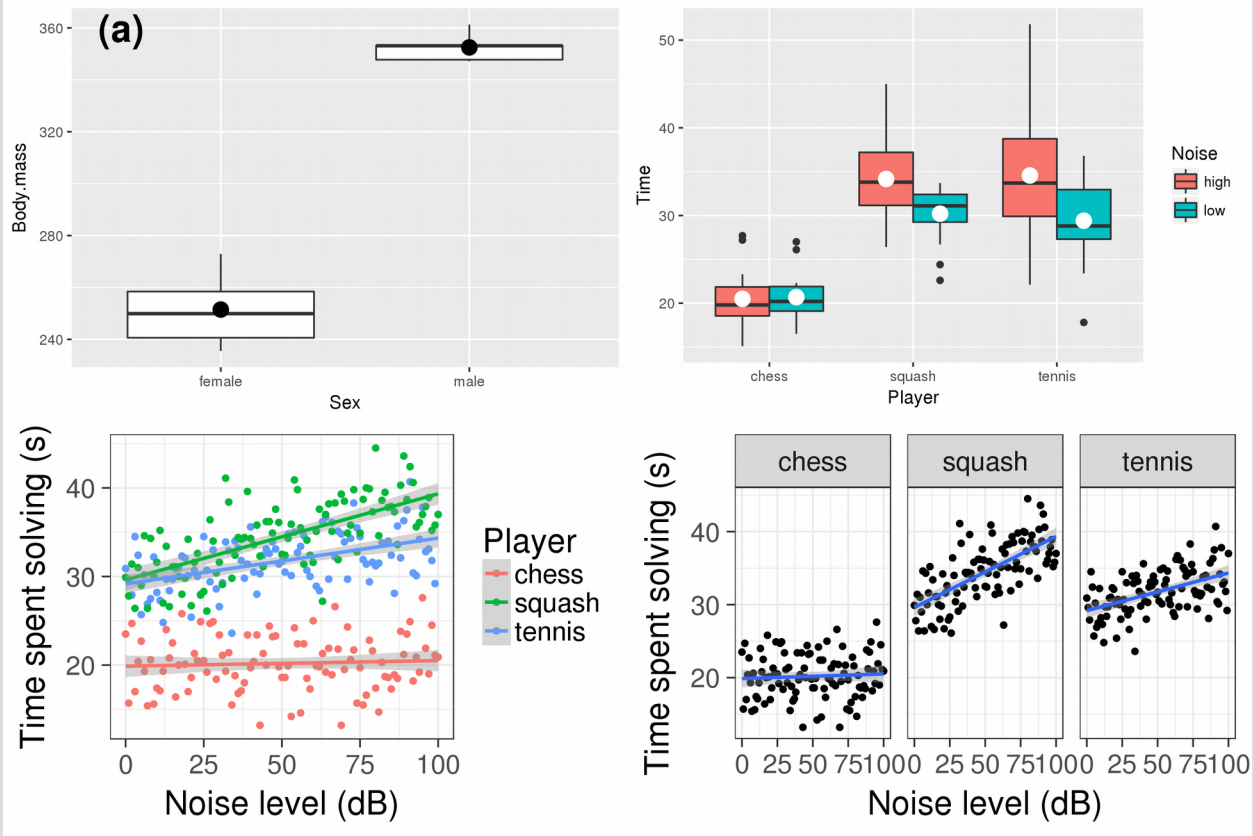
A new version of figure **p3** where facets are used:

```
p3b <- ggplot(solving.time2.df, aes(Noise, Time))
p3b <- p3b + geom_point()
p3b <- p3b + facet_wrap(~Player)
p3b <- p3b + theme_bw(base_size=20)
p3b <- p3b + geom_smooth(method = "lm",
formula = y~x)
p3b <- p3b + labs(x="Noise level (dB)",
y="Time spent solving (s)")

p3b
```



How to create plots with multiple plots per page, i.e. putting your figures together on a single page

```
library(gridExtra)#Needs to be installed
library(grid)
p4 <- grid.arrange(p1,p2,p3,p3b, ncol=2, nrow=2)
grid.draw(p4)
```



How to save plots

```
ggsave("myplot.png")
```

Works out what format to use from file extension name.

Type **?ggsave** for arguments to change size, resolution etc.

When saving a plot like **p4** with multiple plots per page:

```
ggsave("myplot.png", p4)
```

You need to specify the name you gave to your multiple plots object.

Creating maps with ggmap

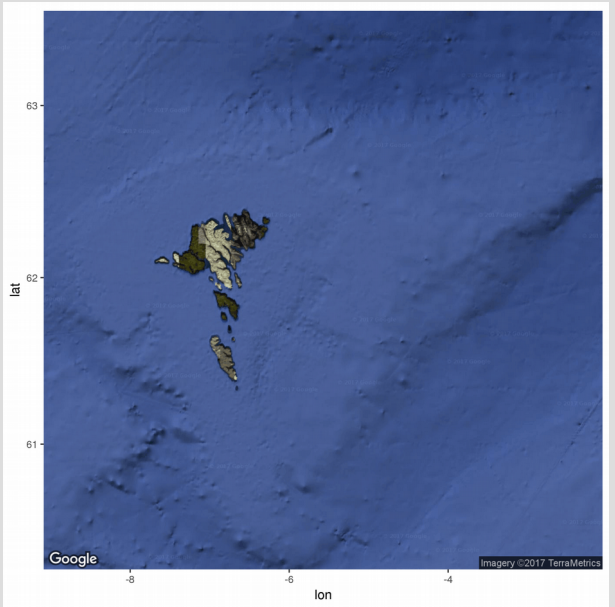
Case: You have been on a survey around Faroe Islands (Færøyene) and sampled krill (*Meganyctiphanes norvegica* and *Thysanoessa longicaudata*). You want to create a map over the sampling stations describing the amount of krill catched at each station.

```
#Importing dataset:
faroe.krill.df <-
read.table("http://folk.uib.no/nzlkj/data/faroe.krill.txt",
sep=",", header=T)
head(faroe.krill.df)
```

Station	Latitude	Longitude	Krill.m2
1	60.91728	-8.61076	1033.5
2	61.26099	-8.56577	630.15
3	61.59187	-8.54018	2370.96
etc	etc	etc	etc

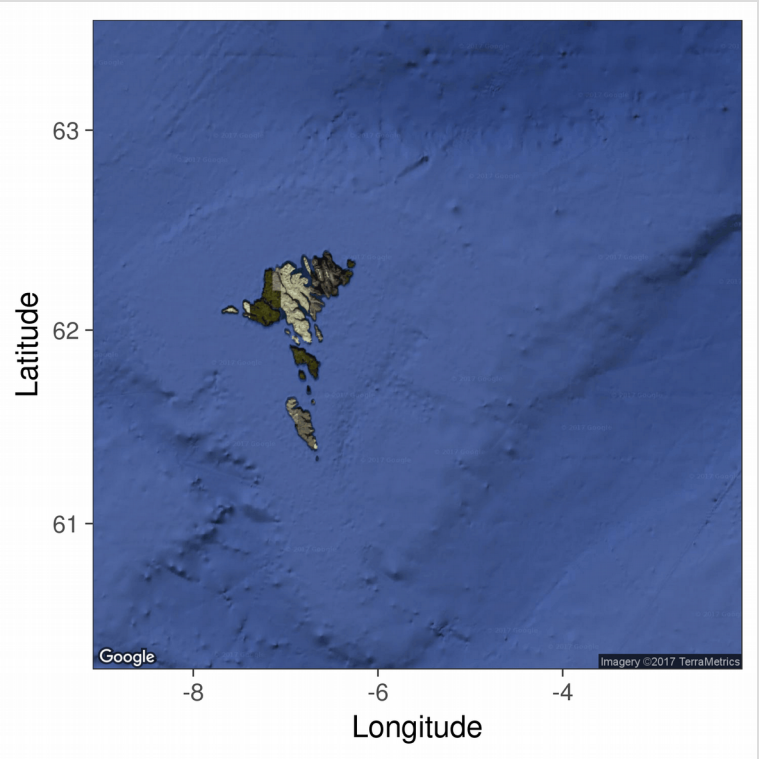
…maps with ggmap continued

```
#Loading required library (must be installed):
library(ggmap)

#Defining location and map type:
faroe <- get_map(location = c(lon = -5.578105,
lat = 61.928955), zoom = 7, maptype =
"satellite")

#Creating basic map:
map1 <- ggmap(faroe)
map1
```



…maps with ggmap continued

```
#Increase text size and add axis titles:
map1 <- map1 + theme_bw(base_size=20)
map1 <- map1 + labs(x="Longitude", y="Latitude")
map1
```



…maps with ggmap continued

```
#Add sampling stations showing amount
#of krill at each of them:
map1 <- map1 + geom_point(data=faroe.krill.df,
aes(Longitude, Latitude, size=Krill.m2),
shape=19, col="red")
map1
```