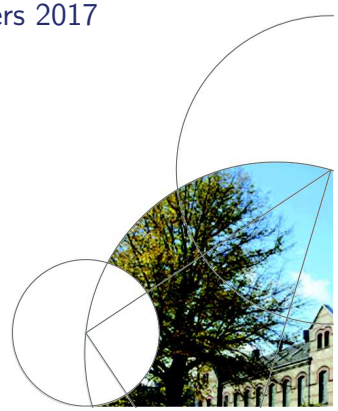
Faculty of Health Sciences

Introduction to R and R studio

Basic statistics for experimental researchers 2017

Julie Forman

Department of Biostatistics, University of Copenhagen



Why R?

R has a huge selection of both simple and advanced statistical procedures, and good graphical features.

R is open source - developed by a dedicated international network of statisticians, mostly in universities.

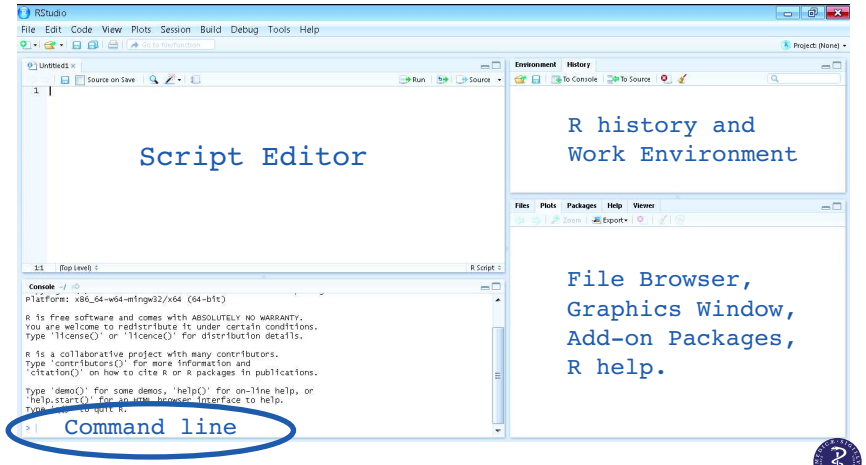
R is well supported. You can get help at SUND's statistical consulting service at the department of Biostatistics and at the many online help forums.

R scripts make **reproducible research** easy.

R was the preferred statistical software of the 2015/16-students.

R studio

The user friendly interface that makes working with R much easier.



Let's go for a demonstartion!

Learning R

Nobody learns programming from reading a book or listening to a lecture. **You learn it by doing it**.

- First try to run the R-scripts from the lectures line by line.
- Can you understand the output? And the code?
- Next try to do a bit of programming on your own by copy pasting from the scripts and making suitable adaptions.
- You need to use the R-help to find out what exactly the R-functions you are using can do for you.
- . . . or maybe ask your friend, statistics teacher, or Google.
- Actually this is all you need to do statistical analyses in R.

These notes are meant as a short reference guide covering the most basic R syntax, descriptive statistics, and tips for the work flow.

Outline

The command line: R as a pocket calculator

R can do all simple calculations +, −, ∗, /, powers, logarithms etc.

```
> 2+3
[1] 5
> log(1)
[1] 0
> (1.27^2+2.04^2)/2
[1] 2.88725
```

You can save your results by assigning them to a named R-object

```
> v1 <- 1.27^2
> v2 <- 2.04^2
> (v1+v2)/2
[1] 2.88725
```

Note: R is case sensitive; `V1` is not the same as `v1`.

You can use the `rm`-function to remove redundant objects.

```
> rm(v1, v2)
```

Vectors and matrices

You store data series and tables in R as vectors and matrices.

```
> myvector <- c(18.1, 7.5, 11.1, 12,4, 8.6)
> myvector
[1] 18.1  7.5 11.1 12.0  4.0  8.6

> mytable <- matrix(c(14, 6, 3, 12, 9, 1), 2, 3)
> mytable
     [,1] [,2] [,3]
[1,]   14    3    9
[2,]    6   12    1
```

You can also generate regular sequences of numbers with `seq`. This is useful if you want to draw a curve.

```
> xs <- seq(0, 2, by=0.01)
> plot(xs, exp(xs), type='l')
```

Note that arithmetic functions are evaluated *elementwise*.

Extracting data from vectors and matrices

Use the brackets $[\cdot, \cdot]$ to pick out entries in a vector.

```
> myvector[2]
[1] 7.5
> myvector[myvector>12]
[1] 18.1
> myvector[1] <- 8.1
> myvector
[1]  8.1  7.5 11.1 12.0  4.0  8.6
```

Likewise entries, rows, and columns in a matrix are identified by their indices.

```
mymatrix[2,3]            # entry in second row, third column
mymatrix[1, ]            # the first row
mymatrix[ ,c(1,3)]       # the first and thrid column
```

Using R-functions

R has a wide range of functions that can do all from simple calculations to generating figures and making statistical analyses.

```
sqrt(17)

summary(mydata)

hist(mydata$volume, probability=TRUE)

t.test(volume~type, data=mydata)
```

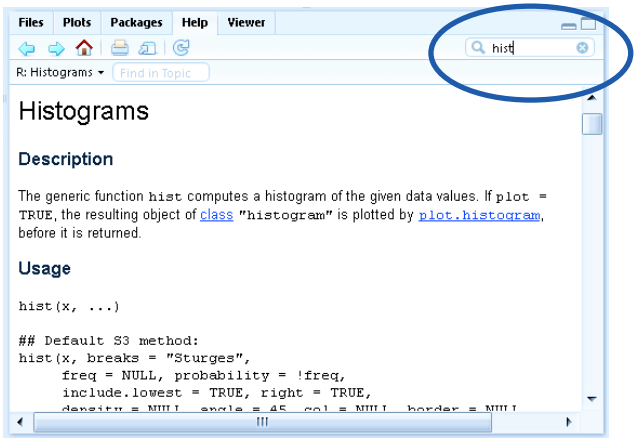
Note: R uses *named arguments* so you don't have to remember in which order to state the arguments to a function.

For instance the histogram will be the same if you write

```
hist(probability=TRUE, x=mydata$volume)
```

Using R help

R has built in help where you can look up a function, read which arguments it takes, and see examples of how to use it.



Common errors

No programming without bugs. Here we list the most typical ones.

- Misspelled variable name, function name, argument name etc.

  Note: R is case sensitive so `X` and `x` is not the same.

- Missing commas between the arguments of a function.

- Misplaced or mismatched paranthesis or quotation marks.

  Note: If R doesn't return output but instead changes the prompt > to +, it means that the command you have submitted is incomplete. You can choose to enter e.g. the missing) or you can press ctrl+c to abort the running R-process and start over.

Outline

Importing data from a spread sheet (in Windows)

If you keep your data in an excell file you can import it in R.

1. Save data in Excell as a csv-file (semicolon separated).
2. Read the data into R with the `read.csv2`-function.

It is importhant to tell R exactly where the datafile is located.

- You can cllick your way to where the data is:
  ```
  > mydata <- read.csv2(file.choose(), header=TRUE)
  ```
- or you can specify the path to where the data is, e.g. by
  ```
  setwd("C:/Documents/teaching/basic/Rdemo")
  mydata <- read.csv2("mydatafile.csv", header=TRUE)
  ```

Note: The argument `header=TRUE` tells R that the first line in the spreadsheet contains the names of the variables. Variable names should not contain any spacing or special characters such as the danish letters æ, ø and å as this will cause trouble in R.

Extracting data from a dataframe

Use $-notation to access the individual variables in your dataframe.

```
mydata$mouseid
table(mydata$treatment)
```

The brackets $[\cdot,\cdot]$ can be used to pick out single entries, whole rows or columns, or a subset of variables. either specifying these by their indices or by their names

```
mydata[1,4]                                  # is the same
mydata[1,'treatment']                        # as this

mydata[1, ]                                  # is the same
mydata[mydata$mouseid==21&mydata$day==1, ]    # as this

mydata[ ,4]                                  # is the same
mydata[ ,'treatment']                        # as this

mydata[ ,c('treatment','volume')]
```


Adding new variables to a dataframe

Use the `transform`-furction to add new variables to a dataframe

Transform a quantitative variable:

```
newdata <- transform(mydata, logvolume=log(volume))
```

Categorize a quantative variable with `cut`:

```
newdata <- transform(mydata,
   size=cut(volume, c(0,200,500,2000), levels=c('s','m','l')) )
```

Join groups of a categorical variable with `factor`:

```
newdata <- transform(mydata,
   type = factor(treatment, labels=c('actv','ctrl','actv')) )
```


Subsetting

Use the `subset`-function to pick out a subset of your data for further analysis.

```
day1 <- subset(mydata, day==1)
largesize <- subset(mydata, volume>1000)
missing <- subset(mydata, is.na(volume))
```

You can also use <, >=, and <= to identify subsets of your data.

It is possible to combine several conditions or negate one;
In R & means "and", | means "or", and ! means "not".

```
notmissing <- subset(mydata, !is.na(volume))
nogrowth <- subset(mydata, day>14 & volume<100)
```

Outline

Summary statistics

The following R-functions computes summary statistics.

R-function	Usage
mean	mean(mydata$volume)
sd	sd(mydata$volume)
min	min(mydata$volume)
max	max(mydata$volume)
range	range(mydata$volume)
median	median(mydata$volume)
quantile	quantile(mydata$volume, prob=c(0.25,0.75))
length	lenght(mydata$volume)

Use the `table`-function **to tabulate categorical data**. Store the table and use `prop.table` to get the proportions in each group.

```
> mytab <- table(mydata$treatment)
> prop.table(mytab)
```

Group-wise summary statistics

The `aggregate`-function can be use to compute summary statistics for several treatment groups (and several outcomes).

Note: This only works with brackets and not with $-notation.

```
> aggregate(day1["volume"], day1["treatment"], mean)
  treatment   volume
1     chemo 157.0375
2     contr 201.6400
3     radio 172.0750
>
> aggregate(day1["volume"], day1["treatment"], quantile,
+ prob=c(0.25, 0.50, 0.75))
  treatment volume.25% volume.50% volume.75%
1     chemo     54.800    107.550    155.625
2     contr    100.300    209.100    249.575
3     radio     81.800    148.150    190.700
```

Making plots in R

To visualize the distribution of a single variable use:

R-function	Usage
hist	hist(mydata$volume)
boxplot	boxplot(mydata$volume)
stripchart	stripchart(mydata$volume)
barplot	barplot(table(mydata$treatment)))

To visualize the association between two variables use:

R-function	Usage
plot	plot(mydata$day, mydata$volume) # ATT
boxplot	boxplot(mydata$volume~mydata$treatment)
stripchart	stripchart(mydata$volume~mydata$treatment)
barplot	barplot(table(mydata$dead,mydata$day))

ATT: If you apply the function `plot` to one or two variables or even an entire dataframe, then R chooses a plot that matches the data.
In the above case you get a scatterplot of two numerical variables.

Changing the look of your graphics

The plotting functions in R take additional graphical arguments that will help you make exactly the plot you want for presentation.

Argument	Effect	Example
type	connect point in a scatterplot	type='l'
lty	change the line type	lty=2
lwd	change the line width	lwd=2
pch	change the plotting symbol	pch=16
col	change the color	col='blue'
xlim	change the range on the x-axis	xlim=c(0,100)
xlab	change the label on the x-axis	xlab='Duration (days)'
	ylim and ylab have similar effects.	
main	change the headline	main='Growth curves'
cex	mangnify the plotting symbol	cex=1.5
	cex.axis, cex.lab, and cex.main has similar effects.	
mar	change the width of the margins	mar=c(5,6,1,1)

Note: You can also reset Rs defaults with the `par`-function.

Adding elements to a plot

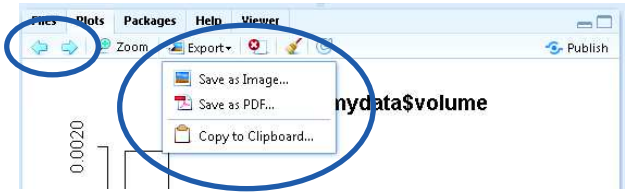
If the following functions are called right after a plotting function then additional elements are overlaid on the plot.

R-function	Effect
points	add one or more points to the plot
lines	similar but the points are joined with lines
abline	adds a straight line to the plot
text	adds text at the specified coordinates
legend	adds a legend to the plot
title	adds a title to the plot

The `axis`-function allows you to costommize your own axis. Note that before you use it you will have to specify the argument `xaxt=FALSE` or `yaxt=FALSE` to your initial plot to leave out Rs default axis.

Saving figures

Choose Export from the menu in the graphics window.



- Use the arrows to browse through the figures you've created.

You can also use R code to export a figure to say a png-file:

```
png('myfigure.png', height=480, width=640)
hist(mydata$volume, prob=TRUE, xlab="Volume", main="")
dev.off()
```

Note: It is important to remenber `dev.off()` to finish the creation of the figure.

Outline

Writing R-scripts

We strongly recommend that you save the commands for your statistical analyses in R scripts so that you can extend or re-run them later on.

To make a new script choose `File` → `New File` → `R Script` from the menu. Don't forget to save the script while you are working on it.

It is a good idea to add many comments in the script. These will help you understand the code if you haven't worked with it for a while or if you want to copy paste from it when you are doing an analysis for a new project.

- Write your comments after # in the script.

An example

```
# Load the data from "tumorvols.rda" to do exercise 1.
load(file.choose())
# The data "tumordat" should appear in "Envinronment"
summary(tumordat)

# Output histogram to a file.
png("histogram.png", height=480, width=640)
par(cex.lab=2, cex.axis=2)
hist(tumordat$volume, prob=TRUE, xlab="Volume", main="")
dev.off()

# Volumes are skew - use log-transform.
tumordat <- transform(tumordat, logvolume=log(volume))

# Confidence interval for the mean log-volume.
t.test(tumordat$logvolume)$conf.int
```

Scripting and other short cuts

You can execute your R program from the script editor.

- Use ctrl+enter (cmd+enter on Mac) to run the script one line at a time or to run a highligthed section of the script.

When you are working at the command line you don't have to retype the commands that you have alreaddy used.

- Use arrows $\vee$ and $\wedge$ to search trough your previous commands.

You can also look up the commands in the R history window.

- Double click on a line in the history to execute it.
- Use shift+double click on a line to copy it to the script editor.

Managing your working directory

The working directory is the directory on your computer where R collects and saves data by default.

You can change the working directory by choosing `Session` → `Set Working Directory` → `Choose Directory` from the menu or by using the `setwd`-function.

```
setwd("C:/Documents/teaching/basic/Rdemo")
```

When you end a session in R studio and choose save, your R history and work environment get stored in two files `.Rhistory` and `.R` in your working directory. Next time you open R studio these files are loaded and everything will appear just as you left them.