Using R for Statistics in Medical Research

Elrozy Andrinopoulou
e.andrinopoulou@erasmusmc.nl
Sten Willemsen
s.willemsen@erasmusmc.nl

25 February - 1 March 2019

Department of Biostatistics, Erasmus University Medical Center

What is this Course About

- Statistics have flourished in the recent years mainly due to the possibility of doing complex analysis using computers
- The most valuable tool of a modern quantitative researcher is his/her personal computer
  - Many statistical software exist to do simple and specialized analysis
- Analysts must not only learn how to use the software but also the ideas behind it

What is this Course About (cont'd)

- Therefore, the aim of this course is threefold:
- Aim A: learn the software
  - learn how to use the statistical programming language **R**
- Aim B: apply basic statistical methods
  - learn basic statistical tests, regression analysis and data visualizations which you will later need for the more advanced courses such as **Repeated Measurements (CE08), Bayesian Statistics (CE09)**, etc.
- Aim C: tools
  - learn some interesting tools for reporting data analyses in a reproducible manner and building simple interactive web applications

Agenda

- **Part A**
  - Introduction
    - What does R look like ?
    - What is R ?
    - Why R ?
    - Where do I get R ?
    - How does R work ?
    - How to get help in R ?

Agenda (cont'd)

- **Part B**
  - Using R
  - In Practice Examples
  - Basics in R
  - Common R Objects
  - Importing Data and Saving your Work
  - Data Transformation
  - Data Exploration
  - Indexing
  - Functions

Agenda (cont'd)

- **Part B**
  - Loops and Control Flow
  - The apply family
  - Merging data sets
  - Graphs
  - Statistical Tests
  - Regression Models

Agenda (cont'd)

- **Part C**
  - Markdown
  - Git
  - Shiny

Schedule

- February 25: 10h00 - 13h00, 14h00 - 17h00
- February 26: 10h00 - 13h00, 14h00 - 17h00
- February 27: 10h00 - 13h00, 14h00 - 17h00
- February 28: 9h30 - 13h00, 14h00 - 16h00

Exams

- Date: March 1, 14h15 - 17h00
- Format: Assignment
- Open-book

Structure & Material

- Lectures: slides interchanged with live R sessions
- In-between the lectures $\Rightarrow$ Interactive Tutorials, Practice Sessions and Shiny apps
  - you will be asked to perform small and big tasks
  - solutions of the practicals available beforehand
- Material
  - slides
  - R code with the output & R code in soft format
  - **more than what we are going to cover!**

Structure & Material (cont’d)

- You are welcome to try along
- You are welcome to interrupt and ask questions

References (cont'd)

- More books that use R (or S) can be found at:
  http://www.r-project.org/doc/bib/R-books.html, or
  http://www.r-project.org/doc/bib/R-jabref.html

References (cont'd)

- R ships with a number of helpful manuals (illustrated later)
- Other manuals and helpful material are available on-line via CRAN:
  `http://cran.r-project.org/other-docs.html`
  - 'Simple R' by John Verzani (`http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf`)
  - 'R reference card' by Tom Short (`http://cran.r-project.org/doc/contrib/Short-refcard.pdf`)

Part A

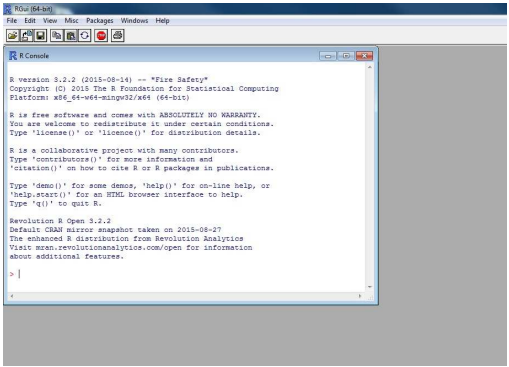
Introduction

A little bit of history

- A little bit of history
  - it was initiated in 1992 by Ross Ihaka and Robert Gentleman at University of Auckland, New Zealand
  - in 1997 the R Core Team was established with renowned members of the statistical computing community
  - nowadays, the R Core Team has grown and consists of about 20 members, experts in computing

Introduction (cont'd)

- What does R look like ?



Introduction (cont'd)

- What is R ?
  - is a software environment for statistical computing and graphics.
  - Unlike SPSS, R is purely command driven

Introduction (cont'd)

- Why R ?
  - R is a free software environment for statistical computing and graphics.
  - it compiles and runs on LINUX, Windows and MacOS
  - R has extensive and powerful graphics & data manipulation capabilities
  - it can easily interface with low-level programming languages, e.g., C/C++ or Fortran
  - it can be easily extended via R packages
  - the source code is available
  - users are allowed to modify and redistribute the code

Introduction (cont'd)

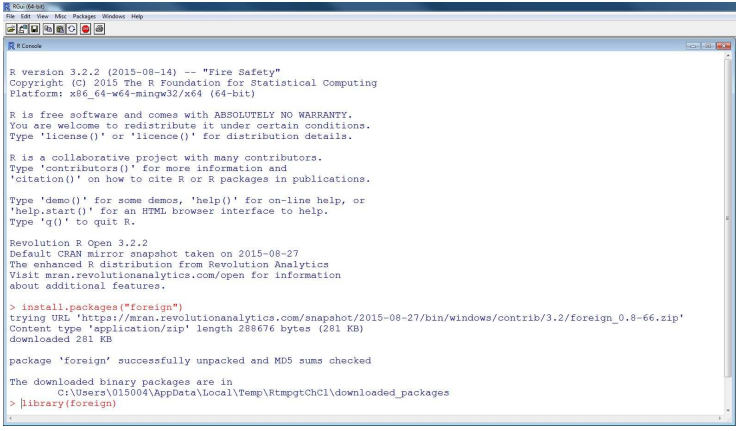
- Where do I get R ?
  - `http://cran.r-project.org`
  - choose your platform, e.g., Windows, Linux
  - e.g., for Windows: `Windows` → `base` → `Download R 3.5.2 for Windows`
  - Install . . .

Introduction (cont'd)

- How does R work ?
  - Packages built for specific tasks
  - Download R packages from the CRAN web site $\Rightarrow$ within R
    * Packages
    * Install package(s) . . .
    * make your choice(s)
    * load the package using `library()` (note: install does not mean load)

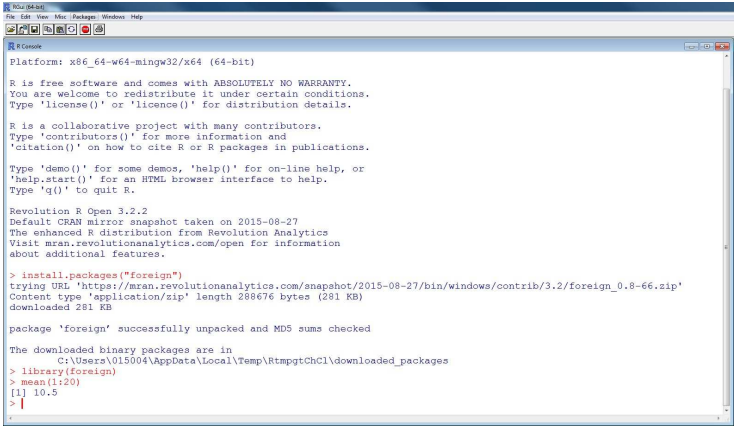
Introduction (cont'd)

How does R work ?



Introduction (cont'd)

How does R work ?



Introduction (cont'd)

How to get help in R

- Within R
  - `help.search("topic")` or `??"topic"` (depends on the installed packages)
  - `RSiteSearch("topic")` (requires internet connection)
  - `help()` or `?` invoke the on-line help file for the specified function
  - checking the FAQ

Introduction (cont'd)

How to get help in R

- On the internet
  - R-help (`https://stat.ethz.ch/mailman/listinfo/r-help` – mailing list)
  - R-seek (`http://www.rseek.org` – Google-like searched engine)
  - R-wiki (`http://rwiki.sciviews.org/doku.php`)
  - CRAN Task Views (`http://cran.r-project.org/web/views/` – categorization of packages)

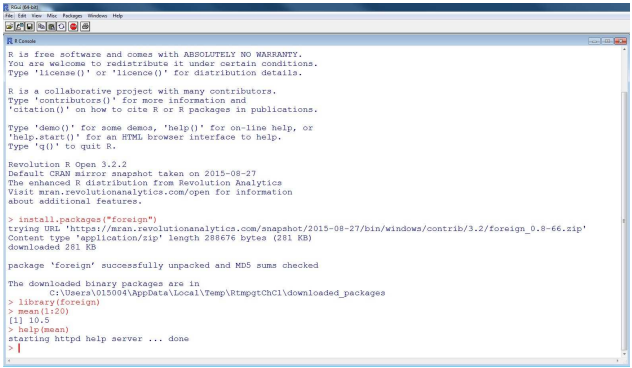
Introduction (cont'd)

How to get help in R

- On the internet
  - Crantastic (http://crantastic.org/ – categorization of packages + reviews)
  - Equalis (http://www.equalis.com/forums/ – R forum)
  - R4stats (http://www.r4stats.com/ – examples of basic R programs)
  - R related Blogs (http://www.r-bloggers.com/ – many useful illustrations of R and R packages)

Introduction (cont'd)

How to get help in R



Introduction (cont'd)

How to get help in R

mean {base}

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An R object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Introduction (cont'd)

Disadvantages of R

- appears intimidating to the first-time user
- output is not so nice looking (but there are some alternatives)
- exporting output is more difficult

Introduction (cont'd)

Disadvantages of R

- cannot easily handle very big data sets (depends on the installed RAM)
- a lot of things are available but it is sometimes hard to find your way
- the quality of the available packages is greatly varying

Part B

1. Using R

- R is a command-based procedural language
  - write and execute commands
  - use and define functions
- You may write the commands in the R console (Windows) or in a shell (Linux)

You will become more familiar with the syntax as you use it

1. Using R (cont'd)

- Strongly advisable to use a suitable text editor – Some available options:
  - RWinEdt (for Windows; you also need WinEdt installed)
  - Tinn-R (for Windows; http://sciviews.org/Tinn-R/)
  - Rkward (for Linux)
  - Emacs (w. ESS, all platforms)
  - Visual Studio (for Windows)
  - Rstudio (all major platforms; http://www.rstudio.org/)
  - for more check http://www.sciviews.org/_rgui/projects/Editors.html

1. Using R (cont'd)

- For this course: Rstudio (http://www.rstudio.org/)
  - free
  - works fine in Windows, MacOS and Linux
  - helpful with errors

1. Using R (cont'd)

- Can I use R without Rstudio?
- Can I use Rstudio without R?

1. Using R (cont'd)

- R and Rstudio

1. Using R (cont'd)

- R

1. Using R (cont'd)

- Rstudio

2. Practical Examples

id	years	status	drug	age	…	serBilir
1	1.095170	dead	D-penicil	58.76684		14.5
2	14.152338	alive	D-penicil	56.44782		1.1
3	2.770781	dead	D-penicil	70.07447		1.4
4	5.270507	dead	D-penicil	54.74209		1.8
5	4.120578	transplanted	placebo	38.10645		3.4
6	6.853028	dead	placebo	66.26054		0.8
⋮						

2. Practical Examples (cont)

- What is a dataset/vector?
- Common questions
  - What is the average age?
  - What is the percentage of drug?
  - What is the average age for males?
  - What is the average age for drug?
  - What is the percentage of alive people?
  - …

2. Practical Examples (cont)

- What is a dataset/vector?

id	years	status	drug	age	…	serBilir
1	**1.095170**	dead	D-penicil	58.76684		14.5
2	**14.152338**	alive	D-penicil	56.44782		1.1
3	**2.770781**	dead	D-penicil	70.07447		1.4
4	**5.270507**	dead	D-penicil	54.74209		1.8
5	**4.120578**	transplanted	placebo	38.10645		3.4
6	**6.853028**	dead	placebo	66.26054		0.8
⋮						

2. Practical Examples (cont)

- Common questions
  - What is the average age?
  - What is the percentage of drug?
  - What is the average age for males?
  - What is the average age for drug?
  - What is the percentage of alive people?
  - …

- **Let’s go to the interactive tutorial!**

3. Basics in R

- Elementary commands: **expressions** and **assignments**
- An **expression** given as command is evaluated printed and discarded
- An **assignment** evaluates an expression and passes the value to a variable – the result is not automatically printed

3. Basics in R (cont'd)

- Expression is given as a command,

```
> 10
[1] 10
```

- However, it cannot be viewed again unless the command is rerun.
- In order to store information, the expression should assign the command

```
> x <- 10
> x
[1] 10
```

3. Basics in R (cont'd)

Using R as a calculator

- Basic arithmetics
  - `+`, `-`, `*`, `/`, `^`
- Complicated arithmetics
  - `%*%`, `t()`, `%/%`

3. Basics in R (cont'd)

- R is case sensitive, e.g.,
  - `"sex"` is different than `"Sex"`
- Commands are separated by a semi-colon or by a newline
- Comments can be put anywhere, starting with a hashmark (#): everything to the end of the line is a comment
- Assign a value to an object by <- or =

3. Basics in R (cont'd)

- Missing values
  - are coded as `NA` (i.e., not available) `is.na()`
- Infinity
  - is coded as `Inf` (plus infinity) or `-Inf` (minus infinity) `is.finite()`
- Not a number
  - is coded as `NaN` (Not a Number)

    Example: `0/0`

    `[1] NaN`

4. Common R Objects

id	years	status	drug	age	…	serBilir	age40higher
1	1.09517	dead	D-penicil	58.7668		14.5	TRUE
2	14.15234	alive	D-penicil	56.4478		1.1	TRUE
3	2.77078	dead	D-penicil	70.0745		1.4	TRUE
4	5.27051	dead	D-penicil	54.7421		1.8	TRUE
5	4.12058	transplanted	placebo	38.1065		3.4	FALSE
6	6.853028	dead	placebo	66.2605		0.8	TRUE
⋮							

There are different kinds of variables.

It depends on the type of variable what we can do with it

4. Common R Objects (cont'd)

- in R Everything (data, results, …) is an object
- A variable is an object with a name so we can refer to it.
- In order to list the created variables use `objects()` or `ls()`

```
> objects()
# Now remove all objects
> rm(list=ls(all=TRUE))
> objects()
character(0)
```

- To investigate a specific object **pbc2.id** use: `str(pbc2.id)`

4. Common R Objects (cont'd)

The simplest data structures are:

Main types of `mode`s

- `numeric` : quantitative data
- `integer` : whole numbers
- `character` : qualitative data
- `logical` : TRUE or FALSE
- `raw` : binary data - not discussed in this course
- `complex` : complex numbers $ai + b$ - not discussed

4. Common R Objects (cont'd)

- To find out what type of vector you have you can use the mode function
- There are also other (non-atomic modes). We will discuss the most important ones (`list` and `function` later.)

4. Common R Objects (cont'd)

Vectors

```
> #Creating a vector consisting of 1,2, and 3
> vector1 <- c(1,2,3)
> vector1
[1] 1 2 3
> vector1 <- 1:3
> vector1
[1] 1 2 3

> #Creating a non-numeric vector
> non.num.vector <- c("A","B","C")
> non.num.vector
[1] "A" "B" "C"
```

4. Common R Objects (cont'd)

Vectors

```
> # Running basic arithmetic operations
> vector1 + vector1
[1] 2 4 6
> # Initialize vector of size 3 and mode 'numeric'
> vector('numeric', 3)
```

4. Common R Objects (cont'd)

- Every object in R has a `class`
- This dictates how (some) generic methods (e.g., `print`, `summary`, etc.) deal with this object

Some common R `classes`:

	Elements same type?	Elements can be different
1d	`vector`	`list`
2d	`matrix`	`data.frame`
3d (or more)	`array`	

4. Common R Objects (cont'd)

- **Matrices** have the same type of elements
- **Data.frames** and **lists** do not need to have the same type of elements
- **Lists** may have elements of different length

4. Common R Objects (cont'd)

Matrices and Arrays

```
> matrix(vector2, 2, 2)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> matrix(vector2, 2, 2, byrow = TRUE)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> # create an array (output not shown)
> array(1:12,dim = c(3,2,4))  # 3 x 2 x 4 array (cube)
```

4. Common R Objects (cont'd)

Lists

```
> mylist = list(names = c("Jack","Mary"),
child.ages = c(4,7,9,10,11))
> mylist
$names
[1] "Jack" "Mary"
$child.ages
[1]  4  7  9 10 11
> vector('list', 3) # initializes list (NB: not a vector)
```

4. Common R Objects (cont'd)

Differences between **Matrices**, **Data.frames** and **lists**

Matrix

years	age
1.095170	58.76684
14.152338	56.44782
2.770781	70.07447
5.270507	54.74209
4.120578	38.10645
6.853028	66.26054

4. Common R Objects (cont'd)

Differences between **Matrices**, **Data.frames** and **lists**

Data frame

patient	years	age	sex	age40higher
1	1.095170	58.76684	female	TRUE
2	14.152338	56.44782	female	TRUE
3	2.770781	70.074473	male	TRUE
4	5.270507	54.74209	female	TRUE
5	4.120578	38.10645	female	FALSE
6	6.853028	66.26054	female	TRUE

4. Common R Objects (cont'd)

Differences between **Matrices**, **Data.frames** and **lists**

List

years	age	format	No patients per gender
1.095170	58.76684	short format	females 88
14.152338	56.44782		males 12
2.770781	70.07447		
5.270507	54.74209		
4.120578	38.10645		
6.853028	66.26054		

5. Importing Data and Saving your Work

- `read.table()` and its variants
  - note: use forward slashes or double backward slashes in the file names, e.g.,
    `"C:/Documents and Settings/User/Data/file.txt"` or
    `"C:\\Documents and Settings\\User\\Data\\file.txt"`
- Specialized functions for importing data from other statistical packages
  - packages `foreign`, `Hmisc` & `readxl`
  - `read.spss()`, `read.csv()`, `read.dta()`, `sas.get()`, `read_excel()`, etc.

5. Importing Data and Saving your Work

Examples

```
library(foreign)
d1 <- read.spss("C:\\Documents and Settings\\User\\Data\\file.sav",
to.data.frame=TRUE)

d2 <- read.csv("C:\\Documents and Settings\\User\\Data\\file.csv")
```

5. Importing Data and Saving your Work (cont´d)

- `ls()` lists R objects in a session & `rm()` removes objects
- `save()`
  - can be used to save a list of R objects
  - a binary file with all the objects available in your last R session
  - platform independent
- You can load your saved R objects using `load()`
  - be careful about overwriting

5. Importing Data and Saving your Work (cont´d)

- **To a TEXT file:**

```
write.csv(mydata, "c:/mydata.txt")
write.table(mydata, "c:/mydata.txt", sep="\t")
```

5. Importing Data and saving your work (cont´d)

- **To an Excel Spreadsheet:**

```
library(xlsx)
write.xlsx(mydata, "c:/mydata.xlsx")
```

6. Data Transformation

- Round continuous variables
- `numeric` variables to `factor`s
- Compute new variables
  - transform variables
- Matrices of wide $\Leftrightarrow$ long format
- Missing values and outliers

6. Data Transformation (cont'd)

id	years	status	drug	age	…	serBilir
1	1.10	dead	D-penicil	58.77		14.5
2	14.15	alive	D-penicil	56.45		1.1
3	2.77	dead	D-penicil	70.07		1.4
4	5.27	dead	D-penicil	54.74		1.8
5	4.12	transplanted	placebo	38.11		3.4
6	6.85	dead	placebo	66.26		0.8
⋮						

6. Data Transformation (cont'd)

id	years	status	drug	age	…	serBilir
1	1.10	dead	D-penicil	58.77		14.5
1	1.10	dead	D-penicil	58.77		21.3
2	14.15	alive	D-penicil	56.45		1.1
2	14.15	alive	D-penicil	56.45		0.8
2	14.15	alive	D-penicil	56.45		1.0
2	14.15	alive	D-penicil	56.45		1.9
⋮						

7. Data Exploration

- Common questions
  - What is the mean and sd for age?
  - What is the mean and sd for follow-up years?
  - What is the median and interquartile range for age?
  - What is the percentage of drug and placebo?
  - What is the mean and sd for age in males?
  - What is the mean and sd for follow-up years in females?
  - What is the mean and sd for baseline serum bilirubin?

7. Data Exploration (cont'd)

- Common questions: **Hints**
  - Check functions: `length(...)`, `mean(...)`, `min(...)`, `sd(...)`, `tapply(...)`, `median(...)`, `IQR(...)`, `quantile(...)`, `split`, `by` and `tapply`

- **Let’s go to the interactive tutorial!**

8. Indexing

- When transforming and analyzing data we often need to select specific observation or variables.
- **Examples**: Select …
  - …the 3rd element for vector age
  - …the 3rd column of a matrix
  - …the sex of the 10th patient
  - …the baseline details of the 5th patient
  - …the serum cholesterol for all males
  - …the age for male patients or patients that have serum bilirubin more than 3
  - …first measurement per patient

8. Indexing (cont'd)

- When transforming and analyzing data we often need to select specific observation or variables.
- This can be done using square bracket (`[,]`) notation and indices.
- Three basic types
  - position indexing
  - logical indexing
  - name indexing

8. Indexing (cont'd)

Position indexing with vectors

```
x <- c(5, 4, 7, 12, 2)
```

- Use positive value to select an element. `x[2]` is `4`
- Multiple elements can be selected by using a (longer) vector of indices. `x[c(2, 4])` is `c(4, 12)`
- Indices do not have to be in ascending order and may be duplicated. `x[c(2, 2, 1)]` is `c(4, 4, 5)`
- Use a negative index to exclude elements. `x[c(-2, -4)]` is `c(5, 7, 2)`
- Positive and negative indices cannot be combined

8. Indexing (cont'd)

Logical indexing with vectors

```
x <- c(5, 4, 7, 12, 2)
```

- Use a logical index of the same length to select elements where the value is `TRUE`

```
> i <- c(FALSE, TRUE, FALSE, TRUE, FALSE)
> x[i]
[1] 4 12
```

8. Indexing (cont'd)

Logical indexing with vectors

```
x <- c(5, 4, 7, 12, 2)
```

- Logical indexing is often used in combination with **conditions**

```
> x[x > 5]
[1] 4 7 12
```

8. Indexing (cont'd)

Name/ character indexing with vectors

```
x <- c(foo=5, bar=4, one=7, two=12, three=2)
```

- When a vector has been **named** the element names can be used as indices

```
> x[c('foo', 'one')]
foo one
5   7
```

- See the `names` function
- NB: A factor variable will not be converted to character when used as an index

8. Indexing (cont'd)

Matrices and arrays

- Indexing matrices as similar to indexing vectors
- We now use a double index `x[i, j]`
  - The first position denotes the **rows**
  - The first position denotes the **columns**
- When we leave a position blank all elements are selected
  - e.g `M[c(2,3), ]` selects all columns of 2nd and 3rd row
- When a single row or column is selected the matrix is converted to a vector unless `drop=FALSE` is used
- Indexing an array works the same way

8. Indexing (cont'd)

Lists

- Lists can be subsetted in the same way as vectors using `[` and `]`, this always returns a list
- **Double square brackets** `[[` and `]]` can be used to extract a single element (what is in the list at this position)
- `$` provides a convinient notation to extract an element by name

8. Indexing (cont'd)

Data.frames

- When you use a single index the data.frame acts like a **list of variables**
- When you using a double index, indexing works like a matrix
  - The first position is used to select observations
  - The second position is used to select variables

- **Let’s go to the interactive tutorial!**

9. Functions

- What are functions & how to define them
  - each package has several functions
  - functions $\Rightarrow$ a group of (organized) R commands
- Why define functions?
  - R is a procedural language
  - Organize your code: easier to reuse, maintain and distribute to others
  - Formal definition of arguments
  - It protects from dangerous use of variable names

9. Functions (cont'd)

- How to use functions in packages
  - study the on-line help file `(?mean)`, especially sections
    - Arguments
    - Value
    - Examples

9. Functions (cont'd)

Function parameters

- Most functions use parameters (formal arguments)
- When you use them the parameters take the value of the arguments you provide in the function call
- You can specify the parameters by **name** `mean(x=x, na.rm=TRUE)` or **position** `mean(x, 0, TRUE)`
  - First the parameter name is used then the arguments are matched by position
- Often parameters have a **default value**

9. Functions (cont'd)

- Some great and useful R functions for datasets
  - What is the number of males and females? `table()`
  - What is the mean weight and height? `mean()`,
  - How many patients are included in the dataset? `length()`
  - Divide the dataset into groups. `split()`

9. Functions (cont'd)

- Some great and useful R functions for vectors in general
  - Create a sequence. `seq()`
  - Create a vector with repeated values. `rep()`
  - Obtain all possible combinations. `expand.grid()`
  - Combine vectors by columns or rows. `cbind()`, `rbind()`

9. Functions (cont'd)

You can also create your own functions!

- Easily use the code again for different data
- Distribute to others

9. Functions (cont'd)

Define function:

```
myfunction <- function(arg1, opt2=2, ...){

  functionbody

 return(...)
}
```

Use the function:

```
myfunction(1, 2)  myfunction(1, opt2=2)  myfunction(1)
```

9. Functions (cont'd)

- When your function gives you an error it can be usefull to use the `debug` function to find it
- If your function works correctly but you want to improve it's speed you can use the `microbenchmark` package.

10. Loops and Control Flow

- Loops:
  - Repeat a statistical test or a computation
    - `for()`: loop over a sequence of values, e.g., iterations
    - `while()`: loop as long as a prespecified condition is satisfied
    - `replicate()`: replicate a number of times
- Control flow:
  - Perform a statistical test or a computation in specific cases
    - `if-else`: standard control flow
    - `ifelse()`: conditional element selection (vectorized version)
    - `switch()`: select one of a list of alternatives

10. Loops and Control Flow (cont'd)

```
> for (i in 1:10){ print(i) }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

10. Loops and Control Flow (cont'd)

```
> for (i in 1:10){
if (i < 5) {
print(i)
}
}
[1] 1
[1] 2
[1] 3
[1] 4
```

11. The apply Family

- Manipulate slices of data from matrices, arrays, lists and dataframes in a repetitive way avoiding explicit use of loop constructs
  - An aggregating function, like for example the mean, or the sum
  - Other transforming or subsetting functions
  - Other vectorized functions, which return more complex structures like lists, vectors, matrices and arrays

11. The apply Family (cont'd)

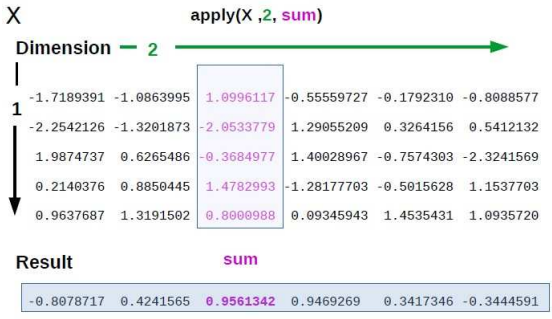
`apply(), lapply() , sapply(), tapply(), mapply()`

But how and when should we use these?

11. The apply Family (cont'd)

How To Use apply() in R

- operates on arrays/matrices



11. The apply Family (cont'd)

How To Use lapply() in R

- You want to apply a given function to every element of a list and obtain a list as result
- The difference with apply():
  - It can be used for other objects like dataframes, lists or vectors
  - The output returned is a list

11. The apply Family (cont'd)

How To Use sapply() in R

- sapply() is similar to lapply(), but it tries to simplify the output

11. The apply Family (cont'd)

How To Use tapply() in R

- Apply a function to subsets of a vector and the subsets are defined by some other vector, usually a factor

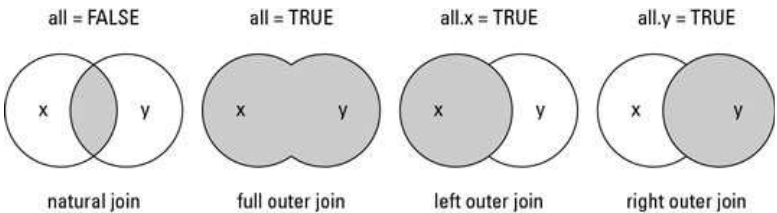
11. The apply Family (cont'd)

How To Use mapply() in R

- Multivariate apply
- Its purpose is to be able to vectorize arguments to a function that is not usually accepting vectors as arguments
- mapply() applies a function to multiple list or multiple vector arguments.

12. Merging Data Sets

- **EXAMPLE 1**: (basic)
- **EXAMPLE 2**: in one dataset a subcategory is missing from a variable
- **EXAMPLE 3**: merge many rows to one
- **EXAMPLE 4**: common IDs have different variable name in the 2 data sets
- **EXAMPLE 5**: different data set variables but same variable name



- **Let’s go to the interactive tutorial!**

13 Graphs

- R has very powerful graphics capabilities
- Some good references are
  - Murrel, P. (2005) *R Graphics*. Boca Raton: Chapman & Hall/CRC.
  - Sarkar, D. (2008) *Lattice Multivariate Data Visualization with R*. New York: Springer-Verlag.

13 Graphs (cont'd)

- Traditional graphics system
  - package `graphics`
- Trellis graphics system
  - package `lattice` (which is based on package `grid`)
- Grammar of Graphics implementation (i.e., Wilkinson, L. (1999) *The Grammar of Graphics*. New York: Springer-Verlag)
  - packages `ggplot` & `ggplot2`

13. Graphs

- Important plotting functions
  - `plot()`: scatter plot (and others)
  - `barplot()`: bar plots
  - `boxplot()`: box-and-whisker plots
  - `dotchart()`: dot plots
  - `hist()`: histograms
  - `pie()`: pie charts
  - `qqnorm()`, `qqline()`, `qqplot()`: distribution plots
  - `pairs()`: for multivariate data

13. Graphs (cont'd)

- Controlling the appearance of plots
  - position of the plots
  - graphical parameters

14. Statistical Tests

id	years	status	drug	age	…	serBilir
1	1.095170	dead	D-penicil	58.76684		14.5
2	14.152338	alive	D-penicil	56.44782		1.1
3	2.770781	dead	D-penicil	70.07447		1.4
4	5.270507	dead	D-penicil	54.74209		1.8
5	4.120578	transplanted	placebo	38.10645		3.4
6	6.853028	dead	placebo	66.26054		0.8
⋮						

14. Statistical Tests (cont'd)

- Is there a difference in serum bilirubin values between drug and placebo?
  - T-test
    - null hypothesis $H_0 : \mu_1 = \mu_2$ (the population means in the two groups are equal)
    - assumption: data is approximately normally distributed or large enough sample
- Is there a difference in age between the status groups?
  - Anova
    - extension of tests for testing two means, now testing two or more groups
    - null hypothesis $H_0 : \mu_1 = \mu_2 = \mu_3 = \ldots$

14. Statistical Tests (cont'd)

- Is there a difference in serum bilirubin values between drug and placebo?
  - Wilcoxon test - Nonparametric alternative
    - Based on the sum of the ranks of the values in both groups
    - H_0: the median difference in the population equals zero
- Is there a difference in age between the status groups?
  - Kruskal-Wallis test - Nonparametric alternative
    - Applied if equality of variances does not hold
    - It is an extension of the Wilcoxon rank sum test to 3 or more groups
    - H_0: each group has the same distribution of values in the population

14. Statistical Tests (cont'd)

- Are the variables status and drug independent?
  - Chi-square test
    - H_0: 2 categorical variables are independent
    - Chi-square should not be calculated if the expected value in any category is $<$ 5
  - Fisher's exact test
    - Nonparametric alternative

14. Statistical Tests (cont'd)

- Is there a linear association between age and serum bilirubin?
  - Correlations - **Pearson**, **Spearman**
    - $r = 0$, no linear correlation
    - $r > 0$, positive linear correlation
    - $r < 0$, negative linear correlation
    - $r = 1$, perfect positive linear correlation
    - $r = -1$, perfect negative linear correlation

14. Statistical Tests (cont'd)

- In some cases, extreme values may be obtained
- Most of the times we exclude the outliers

15. Regression Models

- What it the association of **age** with **serum bilirubin** accounting for **sex** and **drug**?
  - Linear regression models
    - response variable y continuous
    - covariates $x_1, x_2, \ldots$
    - H_0: β is zero
    - assumption: errors are approximately normally distributed
    - model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \cdots + \beta_k x_k + \epsilon, \epsilon \sim N(0, \sigma^2)$
  - In R using `lm()`
    - different options for the `formula` argument
    - summarizing the fitted models: hypothesis testing, plots, etc.

15. Regression Models (cont'd)

- What it the association of **age** with **drug** accounting for **sex**?
  - Logistic regression models
    - response variable y dichotomous
    - covariates $x_1, x_2, \ldots$
    - H_0: β is zero
    - model: $\ln(\frac{y}{1-y}) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \cdots + \beta_k x_k$
  - In R using `glm()`
    - different options for the `family` argument

15. Regression Models (cont'd)

- Using formulas in R
  - serBilir = age + sex + drug $\Rightarrow$ `serBilir ~ age + sex + drug`
  - serBilir = age + sex + drug + age*sex $\Rightarrow$

    `serBilir ~ age + sex + drug + age:sex` $\Rightarrow$

    `serBilir ~ drug + age*sex` (the same)
  - serBilir = age + age^2 $\Rightarrow$ `serBilir ~ age + I(age^2)`
  - drug = age + sex $\Rightarrow$ `drug ~ age + sex`

15. Regression Models (cont'd)

- Formulas are used to define statistical regression models in R but also in other functions to denote a relationship between a response variable and covariates
- A lot of other statistical modeling techniques are available via R packages
- The basic structure is:

```
model.function(formula, data, ...)
```

- **Let’s go to the interactive tutorial!**

Review of Key Points

- The `str()` function can be used to compactly display the structure of an R object
- Data manipulations
  - use indexing
  - some useful functions `split()`, `ave()`, `table()`, `duplicated()`, …

Review of Key Points (cont'd)

- Functions
  - split big calculations in smaller steps
  - make your code easier to read and maintain
- Graphs
  - for standard plots use `plot()`, ...
  - for conditioning plots `xyplot()`, ...

Review of Key Points (cont'd)

- Formulas are used to define statistical regression models in R but also in other functions to denote a relationship between a response variable and covariates
- A lot of other statistical modeling techniques are available via R packages
- The basic structure is:

```
model.function(formula, data, ...)
```

Review of Key Points (cont'd)

- `ls()` lists R objects in a session & `rm()` removes objects
- `save()`
  - can be used to save a list of R objects
  - a binary file with all the objects available in your last R session
  - platform independent
- You can load your saved R objects using `load()`
  - be careful about overwriting

Review of Key Points (cont'd)

- The `str()` function can be used to compactly display the structure of an R object
- Data manipulations
  - use indexing
  - some useful functions `split()`, `ave()`, `table()`, `duplicated()`, …

Review of Key Points (cont'd)

- Functions
  - split big calculations in smaller steps
  - make your code easier to read and maintain
- Graphs
  - for standard plots use `plot()`, ...
  - for conditioning plots `xyplot()`, ...

Review of Key Points (cont'd)

R can do everything …we just need to figure out how …

Part C

1. Markdown

- R Markdown is a format for writing reproducible, dynamic reports with R
- Use it to embed R code and results into slideshows, pdfs, html documents, Word files and more

1. Markdown (cont'd)

- In **Rstudio**
  - File $\rightarrow$ New File $\rightarrow$ R Markdown...
  - Insert title and author
  - Select format

1. Markdown (cont'd)

- **Writing-part**

  Header

- Input

```
# Methods
## Data collection
## Statistical analysis
# Results
```

- Output

Methods

Data collection

Statistical analysis

Results

1. Markdown (cont'd)

- **Writing-part**

  Emphasis

- Input

```
*Methods*
_Methods_
**Methods**
__Methods__
```

- Output

Methods

Methods

Methods

Methods

1. Markdown (cont'd)

- **Writing-part**

  Bullets

- Input

```
* Method 1
- Method 2
+ Method 3
```

- Output

  - Method 1
  - Method 2
  - Method 3

1. Markdown (cont'd)

- **Writing-part**

  Bullets

- Input

```
* Method 1
- Method 2
+ Method 3
```

- Output

  - Method 1
  - Method 2
    - Method 3

1. Markdown (cont'd)

- **Writing-part**

  Links

- Input

```
[link](http://example.com)
```

- Output

  link

1. Markdown (cont'd)

- **Writing-part**

  Images

- Input

```
![Rsymbol](https://
upload.wikimedia.org/
wikipedia/commons/1/12/
R_logo_2000.png)
```

- Output

  Guess???

1. Markdown (cont'd)

- **Writing-part**

  Images

- Input

```
![Rsymbol](https://
upload.wikimedia.org/
wikipedia/commons/1/12/
R_logo_2000.png)
{ width=50% }
```

- Output

  Guess???

1. Markdown (cont'd)

- **Writing-part**

  Horizontal rules

- Input

```
---
```

- Output

---

1. Markdown (cont'd)

- **Writing-part**

  Escaping

- Input

```
\*Method\*
```

- Output

  Method

Escaping Markdown characters with a back-slash allows you to use any characters which might be getting accidentally converted into HTML.

1. Markdown (cont'd)

- **Writing-part**

  Code / Highlights

- Input

  ```
  `Methods`
  ```

- Output

  Methods

Check the difference between pdf, word and html.

1. Markdown (cont'd)

- **R-part (Global options chunks)**
  - eval: If FALSE, knitr will not run the code in the code chunk.
  - include: If FALSE, knitr will run the chunk but not include the chunk in the final document.

1. Markdown (cont'd)

- **R-part (Global options chunks)**
  - **echo**: If `FALSE`, knitr will not display the code in the code chunk above it's results in the final document.
  - **results**: If `'hide'`, knitr will not display the codes results in the final document. If `'hold'`, knitr will delay displaying all output until the end of the chunk. If `'asis'`, knitr will pass through results without reformatting them (useful if results return raw HTML, etc.)
  - **error**: If `FALSE`, knitr will not display any error messages generated by the code.
  - **message**: If `FALSE`, knitr will not display any messages.
  - **warning**: If `FALSE`, knitr will not display any warning messages.

1. Markdown (cont'd)

- **R-part (Global options chunks)**
  - **cache**: If `TRUE`, knitr will cache the results to reuse in future knits. Knitr will reuse the results until the code chunk is altered.
  - **cache.comments**: If `FALSE`, knitr will not rerun the chunk if only a code comment has changed.

1. Markdown (cont'd)

- **R-part (Global options chunks)**
  - **fig.cap**: A character string to be used as a figure caption in LaTex.
  - **fig.height**, **fig.width**: The width and height to use in R for plots created by the chunk (in inches).

1. Markdown (cont'd)

- **R-part (Global options chunks)**
  - … https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

1. Markdown (cont'd)

- **spinr**
  - Sometimes you:
    - Want to create pdf / html /doc files from R
    - Do not want to make a .Rmd file
  - With spinr we can do this!
  - Spin converts .R to .Rmd and then creates the desired output from there.

1. Markdown (cont'd)

- **spinr** syntax
  - When you want a line to be interpreted as markdown prepend it with `#'`
  - When you want start a new code chunk (with different options) use `#+`

2. Git

What is Git and why use it?

- Git is software for source control (version control)
- This means Git keeps track of the different versions of your code
- Helps you answer questions as:
  - Hi Sten. Which file should I look at to see the changes you made last week?
  - What has changed between November and now?
  - Can you send me the version that was submitted to Statistical? Modelling

Git (cont'd)

- Maybe you already use some versioning scheme (using the filenames)

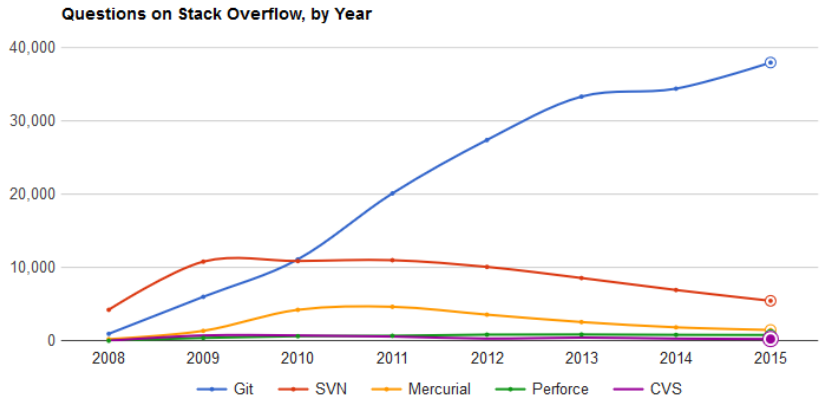
```
analysis.sps
analysis1.sps
analysisFinal.sps
analysisFinal_long.sps
analysisFinal_revision.sps
analysisFinal_revisionnew.sps
analysis2.sps
```

2. Git (cont'd)

- 'Version control systems' (VCSs) are pieces of software that help you manage different versions of programs (documents etc …)
  - Easy to go back to older versions
  - Assist in maintaining multiple versions in parallel (e.g. windows / linux)
  - simplifies working in teams
- There are different 'version control systems' (VCSs)
  - CVS, Subversion, Bazaar, Mercurial, Perforce, Git
- We can distinguish between:
  - systems with a **central repository** (cvs, subversion) and
  - systems where **all repositories are treated as equal** (Mercurial, Git)

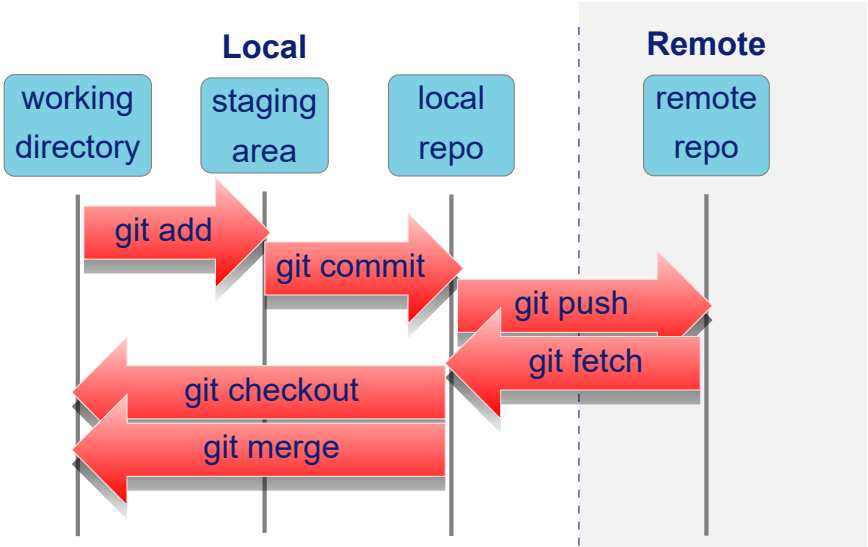
2. Git (cont'd)

- It seems that now git is more or less the standard



2. Git (cont'd)

Git workflow



2. Git (cont'd)

- Git primarily works via the command line
- When you install git for windows an environment is created that includes some of the basic tools that are also available on linux/unix machine including a version bash
- There are a number of GUIs (GitKraken seems ok)
- Git is also supported from a number of other software packages (like RStudio)

2. Git (cont'd)

- Git in RStudio
  - To use Git in RStudio
  - We need to create a new project:
    Files $\rightarrow$ New project ... $\rightarrow$ Version control
  - Or add to existing project
    Tools $\rightarrow$ Version Control $\rightarrow$ Project Setup
  - Click the dropdown box Version control system and select Git
  - There is now a git tab in the upper righthand pannel from which we can do the most common operations

3. Shiny

- Shiny is an R package that makes it easy to build **interactive web apps** straight from R
- Shiny combines the computational power of R with the interactivity of the modern web
- Shiny apps are easy to write - No web development skills are required

3. Shiny (cont'd)

- In **Rstudio**
  - File $\rightarrow$ New File $\rightarrow$ Shiny Web App...
  - Insert application name

3. Shiny (cont'd)

- **Example I**

```
> library(shiny)
> runExample("01_hello")
```

3. Shiny (cont'd)

- **Shiny structure**
  - **a user interface object (ui)**
  - **a server function**
  - **a call to the shinyApp function**

3. Shiny (cont'd)

- **Shiny structure**
  - **a user interface object (ui)**: layout and appearance of your app
  - **a server function**: instructions that your computer needs to build your app
  - **a call to the shinyApp function**: creates Shiny app objects

3. Shiny (cont'd)

Shiny structure

```
> library(shiny)

# Define UI ----
> ui <- fluidPage(
)

# Define server logic ----
> server <- function(input, output) {
}

# Run the app ----
> shinyApp(ui = ui, server = server)
```