Aalto University
School of Science

Basics of using the R software

Experimental and Statistical Methods in Biological Sciences I

Heini Saarimäki, BECS

18/09/2014

Demo sessions

- Demo sessions (Thu 12.15 – 15.00) x 5
  - Demos, example code, and exercises

- How to pass this part of the course
  - Either attend the session OR
  - Hand in your answers to the demo session exercises **before** the session (exercises available in Noppa on Monday before the session)
    - i.e. DL on Thursday at 12pm

Homework

- Homework assignments x 5
  - Online after each demo session
  - Based on real data, publications attached

- How to pass this part of the course
  - Submit 1) commented .r script, and 2) answers in a text document, by email to Dima before the next lecture
  - dmitry.smirnov@aalto.fi
  - i.e. DL always on following Monday at 9am

Help!

- Reception hours
  - Tuesdays 1-2pm
    room F326, BECS, Rakentajanaukio 2C
  - In case you need help with R / homework assignments

- Or by email

  heini.heikkila@aalto.fi
  dmitry.smirnov@aalto.fi

- Remember to bring / attach your .r script!

Course demo content

1. Basics of using the R software
2. Preparing data for statistical analysis
3. Descriptive statistics
4. Plotting data
5. Comparing means with t-tests
6. Analysis of variance
7. Correlation and linear regression
8. Testing categorical associations

Course demo content

1. **Basics of using the R software**
2. **Preparing data for statistical analysis**
3. Descriptive statistics
4. Plotting data
5. Comparing means with t-tests
6. Analysis of variance
7. Correlation and linear regression
8. Testing categorical associations

Structure of demo sessions

- Short slide show introducing the basic contents
  - Example .r code in Noppa to serve as reference for exercises
- Exercise sheet
  - Answers demonstrated by TAs
  - .r code available in Noppa after the demo session
- If you cannot attend the demo session, you can make up for you absence by providing answers to the exercise sheet in before hand

Outline for today

- Part I. Basics of using the R software
  - Slides + demonstration
  - Exercises
  - Break

- Part II. Preparing data for statistical analyses
  - Demonstration
  - Exercises

Outline for today: Part I.

0. What is R?
1. Getting started
2. Data types
3. Data structures
4. Subscripting
5. Operations on your data

Outline for today: Part I.

0. **What is R?**
1. Getting started
2. Data types
3. Data structures
4. Subscripting
5. Operations on your data

0. What is R?

- Free open source software environment, based on the S language developed by Bell Labs
- With R you can write functions, do calculations, apply most available statistical techniques, or create simple or complicated graphs
- User-contributed packages expand the functionality of R
- A large user group supports it
- The use of R requires programming (*)

0. What is R? - Readings and info on R

- Free online tutorials:
  - www.statmethods.net/about/books.html
  - http://ipsur.org/index.html
  - www.personality-project.org/R/r.guide.html
  - …
- Books:
  - Crawley, M. J. (2005) Statistics: An introduction using R.
  - Crawley, M. J. (2007) The R book.
  - Spector, P. (2008) Data manipulation with R.
  - Zuur, A. F. et al. (2009) A beginner's guide to R.

0. What is R? Readings and info on R

- Potentially useful sidekicks:
  - R-Commander GUI: http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/
  - Tinn-R: http://sourceforge.net/projects/tinn-r/
  - R studio: http://www.rstudio.com

- Online resources:
  - R search engine: www.rseek.org
  - Blog aggregator: www.r-bloggers.com
  - Reference cards: http://decvheatscheet.com/tag/r/?page=1
  - R graph gallery: http://gallery.r-enthusiasts.com

0. What is R? How to install and run R

- Go to www.r-project.org
- Downloads: CRAN
- Set your mirror (e.g., in Sweden)
- Go to section ”Precompiled binary distributions…”
  - Select your operation system
  - Select ”base”
  - Select the installation and follow installation instructions

0. What is R? How to run R in Aalto network

- Create a network folder for your R demo materials (Z:)
- Windows:
  - R is installed
  - Starting R opens also Tinn-R; if you don't want to use it, simply close the window
- Linux:
  - Start by typing R on the command line
  - Or open RStudio

0. What is R? R interface

- Console window
  - Executes functions and commands
  - Displays outputs related to those operations
- Graphics window
  - Displays visual information (plots)
- Editor window
  - Resembles a basic word processor
  - Is called when a text file is opened in R
  - Allows to save code, document it, and rerun it at a later stage

Outline for today: Part I.

0. What is R?
1. **Getting started**
    - Command-line syntax
    - Variable assignment
    - Functions and packages
2. Data types
3. Data structures
4. Subscripting
5. Operations on your data

1.Getting started: Command line syntax

```
> # This is a comment
> # The > is the prompt
> 3*(2+2)   # This is an example
[1] 12
> # The [1] is a counter (not part of the data)
```

1. Getting started: Variables and assignments

- Variables are names that refer to temporarily stored values
- Create variables by assigning a value to a name
- Assignment operators: `<-` or `=`
- Names can contain letters, numbers, '_' and '.', but cannot begin with a number – remember also case-sensitivity

1. Getting started: Variables and assignments - example

```
x = 3       # x becomes 3, i.e. assigns value 3 to x
x            # prints its own current value
[1] 3

x = x+1 # x becomes x+1, its previous value is lost
y = x+1 # y becomes x+1, x is unchanged
```

1. Getting started: Functions, arguments, and returned values

- Data structures store your data
- Functions process it
- Think of functions as operators: like + and – but more specialized

  return value = function(argument)

- A function is written once but may be called many times, each time with different arguments
- A function returns a value, the result of the operation

1. Getting started: Functions, arguments, and returned values

- You call a function by typing its name followed by brackets containing the arguments
- You assign the returned value to a variable for further processing

```
> x = 9           # prepare function input

> y = sqrt(x)     # call function sqrt() with argument, returned value
                    is assigned to y
```

1. Getting started: Finding functions online

- No need to write your own functions, others probably have done that already
- Some useful web sites:
  - R web site: http://www.r-project.org
  - Downloads: http://cran.r-project.org
  - Function finder: http://www.rseek.org
  - Task view: http://cran.r-project.org/scr/contrib/Views/
  - Help archive: http://tolstoy.newcastle.edu.au/R/help

1. Getting started: Finding functions in R

- Most functions have a sensible name
  - mean(), sd(), var(), t.test(), …

- Advanced search for function names:

```
> help.search(”read”)    # search all
> apropos(”^read”)       # names starting with ”read”
> apropos(\\.test$)      # names ending with ”.test”
```

1. Getting started: Function help

```
> help.start()      # manuals and reference (every function has a
                      help page)
> help(t.test)      # help page for function ”t.test”
> ?t.test           # … same as above
```

- ”usage” - a synopsis of the function’s arguments
- ”arguments” - describes the values you can pass as arguments
- ”value” - describes the value returned by the function
- ”examples” – learn by running examples

1. Getting started: Packages and libraries

- A package is a collection of previously programmed functions, often including functions for specific tasks.
- While some packages are included in the base installation, most have to downloaded and installed

- **install**: add package to base version of R

```
> install.packages(arm)     # download and install package ”arm”
```

- **load**: make the functions in the package ready to use for your R session

```
> library(”foreign”)        # load package named ”foreign”
```

1. Getting started: Useful package related commands

```
> sessionInfo()              # packages loaded into memory
> library()                  # list packages installed on your computer
> library(help="foreign")    # list the functions in "foreign"
> update.packages()          # update one or all packages
```

1. Getting started: Starting and organizing your session

```
> getwd()                  # get working directory
> setwd('/u/heikkih3/Documents/R_course')
                           # set path to working directory
> dir()                    # list files in working directory
> ls()                     # list objects in workspace
> rm(list(=ls())           # remove all objects in workspace, i.e.,
                             clear workspace
> graphics.off()           # close all figures
```

Outline for today: Part I.

0. What is R?
1. Getting started
2. **Data types**
    - Numeric
    - Character
    - Logical
3. Data structures
4. Subscripting
5. Operations on your data

2. Data types

Numeric

```
[1] 1  2  3  4  5
```

Character

```
[1] ”apple”  ”orange”  ”apple”  ”orange”  ”orange”
```

Logical

```
[1] TRUE  FALSE  TRUE  TRUE  FALSE
```

2. Data types
Conditional operators

```
x == y      # x equal to y?
x != y      # x not equal to y?
x < y       # x less than y?
x <= y      # x less than or equal to y?
x > y       # x greater than y?
x >= y      # x greater than or equal to y?
```

2. Data types

Logical values are returned by conditional expressions:

```
> 3>2                          # is 3 greater than 2?
[1] TRUE

> ”apple” > ”orange”           # characters are compared alphabetically
[1] FALSE
> ”Apple” > ”apple”            # … and are case-sensitive
[1] TRUE
```

Outline for today: Part I.

0. What is R?
1. Getting started
2. Data types
3. **Data structures**
    - Vector
    - Array/matrix
    - Data frame
    - List
4. Subscripting
5. Operations on your data

3. Data structures: Vector

- **Vector** is an ordered row or column of cells
- Each cell contains a single value
- Cell values must all be the same data type

E.g.

```
[1] 1  2  3  4  5
```

3. Data structures: Vector

Functions that make vectors:

```
> c(1,3,4.5)                          # combine
> c(”apple”, ”apple”, ”orange”
> 1:5                                 # :
> 5.5:-5.5
> seq(0, 2*pi, by=0.1)                # sequence of numbers
> seq(0, 2*pi, length=64)
> rep(1:3, times=3)                   # repeat a vector
> rep(1:3, times=c(3,5,2))
> rep(1:3, each=3)
```

3. Data structures: Arrays/Matrices

- While vectors are one-dimensional, **arrays** are a multidimensional extension of vectors
- The most commonly used array in R is the **matrix**, a 2-dimensional array (according to R)

- A matrix is a 2-dimensional layout of cells
- The cell values must all be the same data type

3. Data structures: Matrices

Functions that make matrices:

```
> x = 1:12                          # create an example vector
> matrix(x, nrow=3, ncol=4)         # matrix arranges by columns by default
> matrix(x, nrow=3, ncol=4, byrow=TRUE) # arranging by rows
> cbind(x,x)                        # bind vectors or matrices by columns
> rbind(x,x)                        # … or by rows
```

3. Data structures: Data frames

- While we could store our data in a matrix, with rows representing observations and columns representing variables…
- … a **data frame** is like matrix, just better (easy access to elements) and more flexible (can store different data types)
- In R, data frames are the preferred method for working with ”observations and variables” –type datasets

  → This is what we will mostly use

3. Data structures: Data frames

Make a data frame from column vectors

```
> data.frame()
```

Read text data from a file into a data frame

```
> read.table(”ageweight.txt”, header=T, sep=”,”)
```

Read text data from an online source

```
> read.table(’http://www.url.fi/data.txt’, header=T, sep=”,”)
```

Read .csv file

```
> read.csv()
```

Export data as a table

```
> write.table()
```

Save and load R objects:

```
> save()
> load()
```

3. Data structures: Lists

- A **list** is a collection of objects
- The components of a list can be different types of objects and can be of different lengths
- Lists are used to pass structures arguments to functions, and to return multi-valued objects from functions

```
> mylist = list(fruit=c(”apple”, ”orange”), price=c(1.2, 1.4, 0.8))
```

3. Data structures: Basic diagnostic functions

```
> class(x)      # get the class (data type) of x
> str(x)        # display structure of object x
> dim(x)        # retrieve dimension (rows, columns) of x
> nrow(x)       # number of rows present in x
> ncol(x)       # number of columns present in x
> names(x)      # get/set column names of x
> head(x)       # returns the first part of x
> tail(x)       # returns the last part of x
```

Outline for today: Part I.

0. What is R?
1. Getting started
2. Data types
3. Data structures
4. **Subscripting**
    - Numeric
    - Character
    - Conditional (logical)
5. Operations on your data

4. Subscripting

- For objects that contain more than one element, subscripting is used to access some or all of those elements
- Subcsripting operations are very fast and efficient, and are often the most powerful tool for accessing and manipulating data in R
- Types of subscripts in R: numeric, character, logical

4. Subscripting: Numeric subscripting of vectors

- Cells in the vector are ordered
- The first cell/element is addressed by subscript/index number 1 (not 0)
- Syntax uses square brackets
- `x[i]`
  - `x` is the vector we want to subscript
  - Subscript `i` can be a single number or a vector of subscripts
- Cells are returned in the order given in the subscript

4. Subscripting: Numeric subscripting of vectors

```
> x = 1:12          # create sequence of number from 1 to 12
> x[1]              # first cell
> x[length(x)]      # last cell
> x[1:6]            # first 6 cells
> x[6:1]            # first 6 cells (reverse order)
> x[c(3,5,1)]       # cells 3, 5, 1 in that order
```

4. Subscripting: Numeric subscripting of data frames

- Matrix and data frame cells have pairs of subscripts: `x[i,j]`
  - `i` is the row subscript, `j` is the column subscript
  - Rule of thumb: rows first, then columns.

4. Subscripting: Numeric subscripting of data frames

- Matrix and data frame cells have pairs of subscripts: `x[i,j]`
  - `i` is the row subscript, `j` is the column subscript
  - Rule of thumb: rows first, then columns.
- An empty index (e.g., `x[i,]`) is shorter form of complete index
- A single subscript (e.g, `x[j]`) returns data frame columns by number
- Double-brackets (e.g., `x[[j]]`) return single columns in their own data type
- Negative subscripts return all cells NOT subscripted (e.g., `x[-i,]`), often useful when removing data!

4. Subscripting: Numeric subscripting of data frames

```
> data = read.table(”ageweight.txt”, header=T, sep=”,”)
> data[2:3, 3:2]          # rows 2:3 and columns 3:2
> data[2:3,]              # rows 2:3 and all columns
> data[-(2:3),]           # remove rows 2:3
> data[,2:3]              # columns 2:3 and all rows
> data[2:3]               # columns 2:3
> data[[2]]               # single column as a vector
> data[2:3,2] = -9        # set specific cells to -9
```

4. Subscripting: Character subscripts

- Columns of a data frame, and components of a list, can be addressed by name.
- The syntax uses a $ sign.

```
> names(data)   # get the column names
> data$AGE      # get the column named AGE
```

- A character vector addresses columns by name:

```
> data[, c("AGE", "WEIGHT")]        # same as data[,2:3]
```

4. Subscripting: Conditional subscripting

- Uses logical subscript vectors to get or set a subset of data based on a condition
- Logical index vectors specify a pattern of cells to be addressed: cells corresponding to cells of the logical vector that are `TRUE`

```
> data$SEX == 1                                    # logical vector
(conditional)

> data[data$SEX==1,]                             # get rows where sex==1

> data[data$SEX==1, "AGE"]                # get AGE where sex==1
```

- Logical vectors may be combined using operators `&` (AND) and `|` (OR) to form composite conditions:

```
> data[data$SEX==1 & data$AGE>50,]  # get rows where sex==1
            AND age > 50
```

Outline for today: Part I.

0. What is R?
1. Getting started
2. Data types
3. Data structures
4. Subscripting
5. **Operations on your data**
    - Vector arithmetics
    - Vector functions
    - Descriptive functions

5. Operations on your data: Vector arithmetics

- The arithmetic operators are: `+ - * / ^ %%`
- Unary operators (`-x` and `x^2`) apply element-wise
- Binary operators apply pair-wise
  - If the vectors are different lengths the shorter is ”recycled” to match the length of the longer, concatenating it with itself until it is at least as long, and trimming excess off the end
- Scalar values are single-element vectors, recycled as necessary

5. Operations: Vector arithmetics

```
> x = c(0,2,4,6,8,10)
> y = x(1,3,5,7,9,11)
> -y
> y^2
> y + x
> y^2                          # scalar 2 is recycled to match length(y)
```

5. Operations: Vectorized functions

- Arithmetic functions take vector arguments and return vector values
- The arithmetic operation is applied element-wise to the vector argument
- Example functions:

round	round to given number of decimal places
ceiling	round values up
floor	round values down
abs	absolute (unsigned) value
sqrt	square root
exp	exponential
log, log10, log2	log to base e, 10, and 2
sin, cos, tan	trigonometric functions
asin, acos, atan	inverse (arc) trigonometric functions
scale	scaling and centering

5. Operations: Vectorized functions

```
> x = seq(0,1,length=10)
> round(x,2)                    # round to 2 decimal places
> sqrt(x)
> log(x+1)
> scale(x)                      # standardize as z-scores
```

5. Operations: Descriptive functions

- Descriptive functions return a single valued summary of a vector
- Example functions:

length	number of elements in a vector
sum	sum of the values in a vector
min, max, range	minimum, maximum, and range
mean	mean of the values in a vector
median	median of the values in a vector
sd, var	standard deviation and variance
cov, cor	covariance and Pearson correlation

5. Operations: Descriptive functions

```
> x = seq(0,1,length=10)
> length(x)                    # length (number of elements)
> mean(x)                      # mean
> sd(x)                        # standard deviation
```

5. Operations: Descriptive functions

- Descriptive functions *cov* and *cor* return either a single value or a matrix, depending upon the arguments
- If the arguments are two vectors, a single value is returned
- If the argument is a matrix, a matrix is returned
- The *i,j*'th element of the matrix is the covariance (using *cov*) or correlation (using *cor*) between the i'th and j'th column vectors

5. Operations: Descriptive functions

```
> x = rnorm(100)          # ?rnorm
> y = rnorm(100)
> cov(x,y)                # scalar covariance
> cor(x,y)                # Pearson correlation coefficient
> cov(cbind(x,y))         # covariance matrix
> cor(cbind(x,y))         # correlation matrix
```

Time for exercises and a break!

- Need more to read?

  www.statsmethods.net/index.html

Aalto University
School of Science

Preparing data for statistical analysis

Experimental and Statistical Methods in Biological Sciences I

Heini Saarimäki, BECS

18/09/2014

Part II: Preparing data for statistical analysis

- A few things need to be taken care of before starting the statistical analysis
  - Check for correct data format
  - Check for outliers and missing data

- Today we focus on how to modify your data in general, later we go into details in preparing your data to some more specific analyses
  - E.g., testing and correcting for normality and other assumptions

Outline for part II

1. Example data
2. Recap
3. Factors
4. Outliers and missing data

Example data: AgeWeight dataset

```
# load in data:

> AgeWeight <- read.csv('http://becs.aalto.fi/~heikkih3/age_weight.
csv', header=TRUE)

# examine a bit:

> head(AgeWeight)
```

Recap 1/2

We have already encountered several types of data:

```
# numeric
> num.v <- c(1.2, 2.2, 3.0)
# character
> chr.v <- c('cat', 'dog', 'mouse')
```

We have seen simple operations on that data:

```
> num.v/2
```

Recap 2/2

We have seen data collected in data frames:

```
> my.df <- data.frame(x=rnorm(10), y=1:10)
```

And we have learned about subscripting and functions:

```
> my.df[,2]
> my.df[2,]
> mean(my.df$x)
```

Factors

- But what if our data represents categories?
- If the categories are numerically related, use a number.
- If they are completely unrelated (arbitrary), use a character.
- BUT if the categories are related but not numerically, use a factor.

- Examples of factors:
  - male and female populations;
  - easy, moderate, and difficult stimuli;
  - ’yes’, ’maybe’, ’no’ etc. answers

Factors

Examine our data:

```
> head(AgeWeight)
```

Here, sex (coded as male=1 in variable MALE) is an independent variable with a categorical response.

You can create a set of suitable labels:

```
> AgeWeight$SEX <- ”m”
```

```
> AgeWeight[which(AgeWeight$MALE==0),'SEX'] <- 'f'
```

But `AgeWeight$SEX` is still just a set of characters. You need to tell R that it’s a factor:

```
> AgeWeight$SEX <- factor(AgeWeight$SEX)
```

Factors

- Factors can also be represented by numbers. Compare the two variables below:

```
> a.v. <- rep(c(1:5), 2)
> a.v

> b.v. <- factor(rep(c(1:5), 2)
> b.v
```

- Note that factor are just labels of whatever type.

Factors: reading .csv files

- .csv files only include characters and numbers
- R has to guess which are factors
- Rule: numbers are numbers, characters are factors.
- You have to tell R if numbers are factors (or if characters are strings).

summary()

- So far, we've looked at variables by typing their names.
- The *summary()* function tells you uselful things about the variables.

```
> x <- c(1:10)
> summary(x)

> y <- c('cat', 'dog', 'mouse', 'horse')
> summary(y)

> z <- factor(c('cat', 'dog', 'dog', 'horse'))
> summary(z)
```

Objects

- Everything in R is actually an ’object’ of a given class.
- *summary()* and other functions take the class into account.

```
> class(x)
> class(y)
> class(z)

> class(class)
```

Objects

- When you type the name of an object, you are implicitly using the function *print()*
- *print()* won't necessarily tell you everything that's 'in' the object, it will just tell you useful information
- If you really want to see what's 'in' the object, use *str()*

```
> z
> print(z)
> str(z)
```

Objects

- When you use a function, such as *mean()*, it returns an object:

```
> mean(x)
> print(mean(x))        # equivalents
> class(mean(x))
> m <- mean(x)          # this stores the result
> class(m)
```

- Complicated functions return complicated objects!

```
> o.lm <- lm(x ~ y, my.df)  # stores your model for linear regression
> summary(o.lm)             # explore results for linear regression
```

- The 'R' way: create objects, then explore them:

```
> cooks.distance(o.lm)  # more results
> plot(o.lm, which=1:2) # plot these results
```

Outliers in the data

- While getting familiar with your data, you should make sure that the data makes sense
  - *summary(), describe()*
  - plotting

→ will help you to identify outliers and missing values

- One cause for anomalities in the data are outliers, possibly caused by
  - Typing errors
  - Funny coding of missing values (e.g., 999)

Missing values

- Often, we don’t have a complete record of the data
- R uses the special code NA to represent a missing value
  - ’not available’

```
> my.df[7,2] <- NA
> my.df
```

Missing values: Effects of NA

- Simple R functions are picky about data with missing values
  - This is a good thing!
- You have to explicitely tell R to ignore missing data

```
> mean(my.df$x)                    # get an error
> mean(my.df$x, na.rm=TRUE)
```

- Some functions are a bit more complex (see *?cor*); some ignore NAs by default