Introduction to R

Mark van der Loo

UFPEL, October 2019

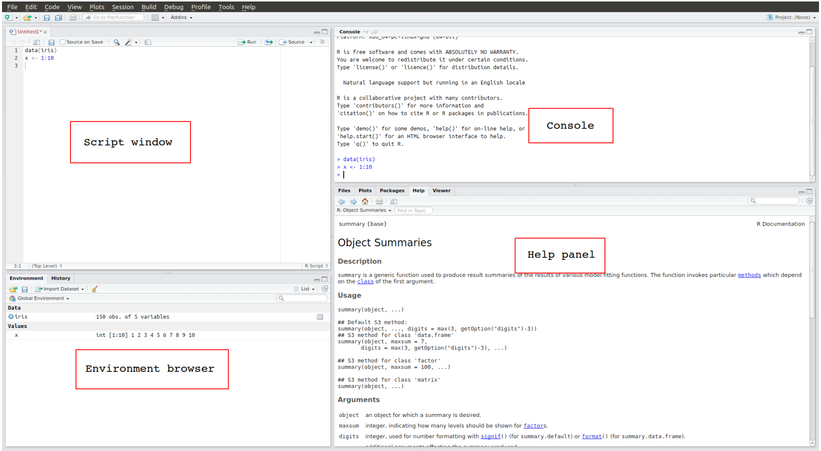
Contents, today's goal

- R, RStudio
- Working with command-line and scripts
- Basic data types (vector, data.frame)
- Some plts plots
- Basic data processing

R and RStudio

R and the R community

RStudio



```
data(iris)
x <- 1:10
```

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> data(iris)
> x <- 1:10
>
```

Data

iris	150 obs. of 5 variables

Values

x	int [1:10] 1 2 3 4 5 6 7 8 9 10

summary {base}　R Documentation

Object Summaries

Description

summary is a generic function used to produce result summaries of the results of various model fitting functions. The function invokes particular methods which depend on the class of the first argument.

Usage

```
summary(object, ...)

## Default S3 method:
summary(object, ..., digits = max(3, getOption("digits")-3))
## S3 method for class 'data.frame'
summary(object, maxsum = 7,
       digits = max(3, getOption("digits")-3), ...)

## S3 method for class 'factor'
summary(object, maxsum = 100, ...)

## S3 method for class 'matrix'
summary(object, ...)
```

Arguments

object	an object for which a summary is desired.
maxsum	integer, indicating how many levels should be shown for factors.
digits	integer, used for number formatting with signif() (for summary.default) or format() (for summary.data.frame).

Console

- Connects to the ‘R interpreter’
- You can type commands there or copy them from the script window
- Resultats are printed to the console again.

```
1 + 1
```

```
[1] 2
```

Script window

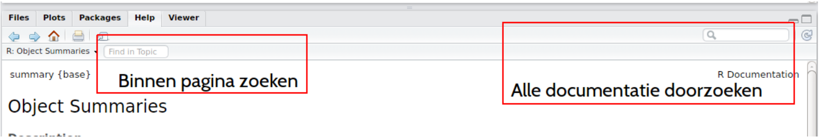
- Here you can open and edit several types of text files, e.g.
  - `.R` (R scripts)
  - `.Rmd` to create reports that include your results
  - `C/C++` for programming with `C` or `C++`
- Use `CTRL-ENTER` to send the currently selected command to the R interpreter.
- The script window is the single most important place in RStudio! **WRITE ALL YOUR CODE IN SCRIPTS**.

Environment browser

- Gives an interactive overview of all data loaded into R
  - data sets, results of modeling; anything really.
- You can get the same overview by typing `ls()` in the command-line

Help panel

- Help pages for each R function



Binnen pagina zoeken

Alle documentatie doorzoeken

summary {base}

R Documentation

Object Summaries

- Open a help page for a function: `?<function>` or search: `??<search term>`.

Note

The help pages are pretty dense and technical. They are aimed to be technical documentation, but don't be intimidated! There is lots of help online.

Getting help

- Q-and-A site `stackoverflow.com`
  - Easily found via Google.
  - `n00b`-friendly
- R-help mailinglist `r-project.org/mail.html`
  - You may get answers from the R-core developers.
  - DO READ THE POSTING GUIDE

Tip of the day

Error message? Cut-and-paste it in Google.

Literature

- Working with R:
  - R in a Nutshell (J. Addler) *O'Reily*
  - **R for data science** (H. Wickham and G. Grolemund) *O'Reilly*
- Programming, package development:
  - The Art of R Programming (N. Matloff) *No Starch Press*
  - Testing R code (R. Cotton) *O'Reilly*
  - R Packages (H. Wickham) *O'Reilly*
  - Advanced R (H. Wichham) *CRC Press*
- Applications:
  - *Use R!* series: `www.springer.com/series/6991`
  - *The R Series* `crcpress.com/go/the-r-series`
  - ...
- See also `r-project.org/doc/bib/R-books.html`

Basic data types and the R command-line

Some tips

Repeat commands

Use arrow keys ↑, ↓ to cycle through previous commands

Keyboard shortcuts (in script window)

`CTRL+ENTER`	Execute current command
`CTRL-SHIFT-S`	Execute current script

Auto-complete

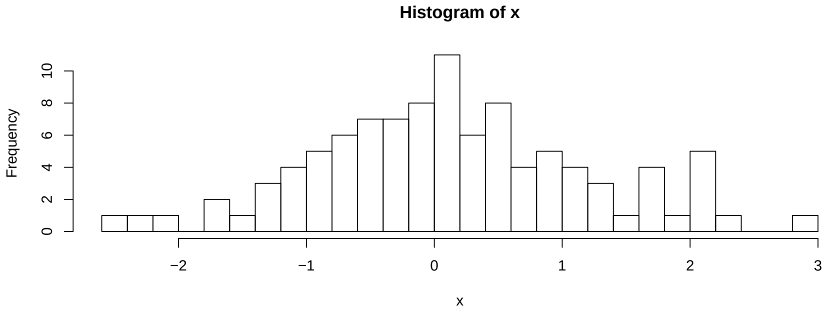
Use `tab` to complete names of objects, columns in `data.frames` and file names (between quotes).

Vectors

The basic unit in R is a *vector*: a sequence of values of the same type (like a column of data in SAS or SPSS –but not Excel!).

Example

```r
# Sample 100 numbers from the normal distribution
# Store under the name 'x'
x <- rnorm(100)
# plot a histogram of x
hist(x, breaks=20)
```



Example (cont'd) statistical summaries.

```r
summary(x) # overview
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max. 
-2.57753 -0.54664  0.06918  0.12463  0.78946  2.85021 
```

```r
sd(x)      # standard deviation
```

```
[1] 1.070897
```

```r
head(x,3)  # first three values
```

```
[1] -0.30295450 -0.16087374  0.09127928
```

Example (cont'd) metadata

```r
length(x)
```

```
[1] 100
```

```r
class(x)
```

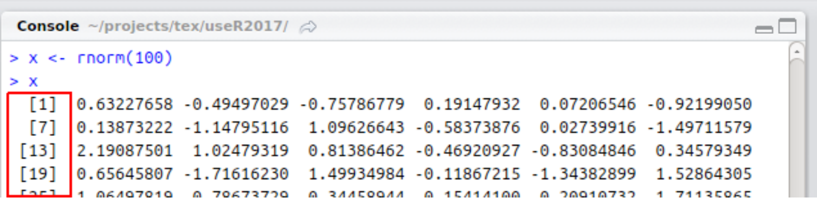
```
[1] "numeric"
```

```r
y <- c(joe=1, bill=7, averett=3)
names(y)
```

```
[1] "joe"     "bill"    "averett"
```

Some observations

- You can create and name vectors under (almost) any name. Use <- to store something under a given name.
- You do calculations with *functions*, like `sd`, `min`, `mean`
- When a vector is printed, the first column in the terminal shows the position.



```
> x <- rnorm(100)
> x
  [1] 0.63227658 -0.49497029 -0.75786779  0.19147932  0.07206546 -0.92199050
  [7] 0.13873222 -1.14795116  1.09626643 -0.58373876  0.02739916 -1.49711579
 [13] 2.19087501  1.02479319  0.81386462 -0.46920927 -0.83084846  0.34579349
 [19] 0.65645807 -1.71616230  1.49934984 -0.11867215 -1.34382899  1.52864305
 [25] 1.06497819  0.78673729  0.34458944  0.15414100  0.20910732  1.71135865
```

Creating vectors

`c(...)`	Assign value by value (`x <- c(1,6,2)`)
`seq(from, to, [by])`	Create a sequence (`x <- seq(1,10,2)`)
`seq_len(length.out)`	Create a sequence `1,2,...,length.out`
`:` (dubbele punt)	`a:b` gives `a,a+1,...,b`
`rnorm(n,[mean],[sd])`	Sample from normal distribution
`runif(n,[min],[max])`	Sample from uniform distribution

Opmerkingen

- Argumenten in square brackets are optional.
- `seq()` also works for time/data sequences

Summarizing vectors

`mean,median`	mean, median
`sum`	Sum
`min,max`	Minimum, maximum
`sd`	Standaard deviation
`fivenum`	Tukey's five-number statistics
`summary`	Sammary (works for all types)
`hist`	Histogram
`boxplot`	Boxplot
`length`	Nr of elements in a vector
`class`	Type of data
`names`	Labels

Remeber that

R is case sensitive

```r
x <- 10
X <- 11
ls()
```

```
[1] "x" "X" "y"
```

Variabelen can be overwritten

```r
x <- 10
x <- "fiets"
x
```

```
[1] "fiets"
```

Computing with vectors

Addition etc works element-by-element.

```r
x <- c(1,3,2,6)
y <- c(2,5,7,3)
x + y # add
```

```
[1] 3 8 9 9
```

```r
x * y # multiply
```

```
[1]  2 15 14 18
```

```r
x ^ y # x to the power of y
```

```
[1]   1 243 128 216
```

Computing with vectors (cont'd): Recycling

For vectors of unequal length, the shortes is repeated

```
x
```

```
[1] 1 3 2 6
```

```
2 * x  # here is '2' a vector of length 1
```

```
[1]  2  6  4 12
```

```
x + 3
```

```
[1] 4 6 5 9
```

Transformations

All the usual math functions are available

```
x <- c(0,1,4,9, 12)
sqrt(x) # squqre root of x
```

```
[1] 0.000000 1.000000 2.000000 3.000000 3.464102
```

Examples

```
exp, log, log10
sqrt
sin, cos, tan, sinh, cosh, tanh
```

Data types

`numeric`	Numbers (integer or real)
`integer`	Integers
`logical`	Boolean (`TRUE`,`FALSE`)
`character`	Text
`factor`	Categorial (nominal) data
`POSIXct`	Date/time

Opmerkingen

- R converts automatically from integer to numeric
- There are a few more types (complex, raw) not shown here

Missing values

- Missing values are represented with NA.
- Almost any calculation involving NA will result in NA

```
x <- c(1,4,2,NA,6)
c( mean1 = mean(x), mean2 = mean(x, na.rm=TRUE) )
```

```
mean1 mean2
   NA  3.25
```

- Skip NAwith na.rm=TRUE

RStudio project | data import | data frames

Contents

- Create an RStudio project
- Scripts
- Reading csv files
- Introducing `dplyr`

Reading text files with readr

Reading

`read.csv`	Comma for columns, dot for decimals
`read.csv2`	Semicolin for colums, comma for decimals
`read.table`	Any 'rectangular' text data.

Wegschrijven

`write.csv`	Kommascheiding, punt is decimaalteken
`write.csv2`	Puntkommascheiding, komma is decimaalteken
`write.table`	Alle rechthoekige bestanden in tekstformaat.

```r
dat <- read.csv("myfile.csv")
write.csv2(dat, "yourfile.csv", row.names=FALSE)
```

File names in R

- Alwasys in quotes.
- It can also be a `url`.
- Always use forward slash as directory separator:

```r
dat <- read.csv("C:/users/joe/documents/foo.csv")
```

Tip oif the day

Always work in an RStudio project. It makes it much easier to locate files.

Data frames

Een `data.frame` is a bunch of vectors of the same length.

```r
# this dataset is built into R for examples.
head(InsectSprays,3)
```

```
  count spray
1    10     A
2     7     A
3    20     A
```

Summarizing data frames

```
summary(InsectSprays)
```

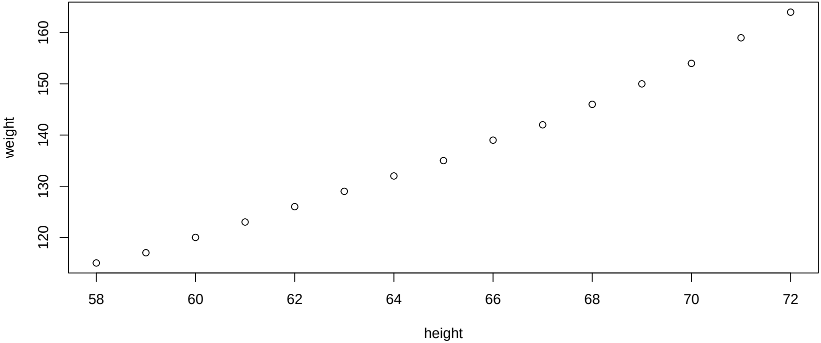
```
     count           spray
 Min.   : 0.00   A:12
 1st Qu.: 3.00   B:12
 Median : 7.00   C:12
 Mean   : 9.50   D:12
 3rd Qu.:14.25   E:12
 Max.   :26.00   F:12
```

Some handy functions

Functie	**description**
`summary`	Statististical summary
`str`	Technical summary
`colMeans`, `rowMeans`	mean per column, row
`colSums`, `rowSums`	sum per column, row
`names`	column names
`ncol` `nrow`	nr of columns, rows
`dim`	vector with `nrow, ncol`

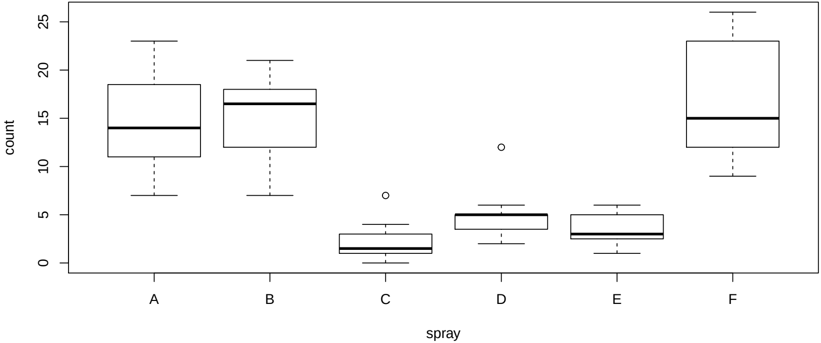
Plotting (1)

```
plot(weight ~ height, data=women)
```



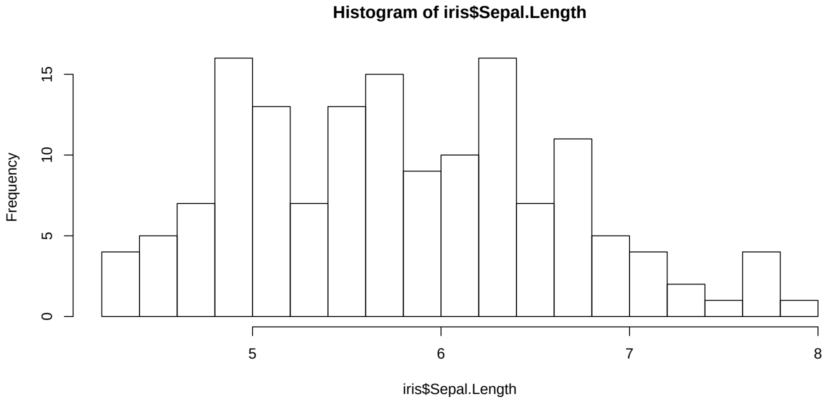
Plotting (2)

```
plot(count ~ spray, data=InsectSprays)
```



Plotting (3)

```r
# met '$' selecteer je een kolom
hist(iris$Sepal.Length, breaks=20)
```



Introdiction to data manipulation with dplyr

```r
library(dplyr)
```

Verbs for common operations

`filter`	Rijen selecteren
`select`	Kolommen selecteren
`rename`	Kolommen hernoemen
`distinct`	Unieke rijen selecteren
`arrange`	Sorteren
`transmute`	Nieuwe kolommen berekenen
`mutate`	Nieuwe kolommen toevoegen

dplyr::filter

Select rows.

```
filter(.data, ...)
```

Here, `.data` is a `data.frame` (or tibble) and `...` are conditions.

```
filter(iris, Sepal.Length > 7)
filter(iris, Sepal.Length > 7, Species=="virginica")
filter(iris, Sepal.Length > mean(Sepal.Length))
```

Comparison operators

Expression	TRUE when
`x == y`	`x` equals `y`
`x <= y`	`x` does not exceed `y`
`x < y`	`x` strictly smaller than `y`
`x > y`	`x` strictly larger than `y`
`x >= y`	`x` larger than or equal to `y`
`x != y`	`x` unequal to `y`
`x %in% y`	`x` appears in `y`

Example: %in%

```
x <- c("noot", "boom", "roos", "vis", "aap")
y <- c("aap", "noot", "mies")
x %in% y
```

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

Logical operators

Operator	Betekenis
`&`	AND
`|`	OR (en/of)
`!`	NOT
`all(x)`	are all entries in `x` `TRUE`?
`any(x)`	is at least entry in `x` `TRUE`?

dplyr::select

Select columns

```
select(.data, ...)
```

Use `...` to select columns:

```
select(iris, Sepal.Width, Petal.Width)
```

Or give the selected columns new names:

```
select(iris, bladlengte=Petal.Length
       , soort=Species)
```

dplyr::rename

Rename columns

```
rename(.data, ...)
```

Specify as `<new name> = <old name>`.

```
rename(iris, species = Species)
rename(iris, leaf_size = Sepal.Width, species=Species)
```

dplyr::distinct

Keep only unique rows

```
distinct(.data, ..., .keep_all=FALSE)
```

With ... you specify what columns determine wheter a record is unique. In case of duplicates, the first record is kept. The `keep_all` option determines whether to keep all columns or just the ones specified in ....

```
distinct(iris, Species, keep_all=TRUE)
```

dplyr::arrange

Sorteer de rijen.

```
arrange(.data, ...)
```

Use `...` to specify sorting variables. Each next variable is a tie-breaker for the previos ones. Use `desc` to sort descending in stead of increasing.

```
arrange(iris, Sepal.Length, Petal.Width)
arrange(iris, Sepal.Length, desc(Petal.Width))
```

dplyr::mutate

Add columns

```
mutate(.data, ...)
```

Use `...` to specify a sequence of expressions that define the new columns.

```
mutate(women
  , lengthM    = height * 2.54/100
  , weightKg   = weight/2.046
  , bmi        = weightKg/(lengthM^2))
```

Expressions are alsways in the form `<new name> = <expression>`.

dplyr::transmute

Compute new columns

```
transmute(.data, ...)
```

Same as `mutate`, except only the new columns are returned.

```
transmute(women, ratio=height/weight)
```