Introduction to R

Mark van der Loo

https://github.com/markvanderloo/UFPEL2019

Contents, today's goal

- What is R, RStudio
- Working with command-line and scripts
- Data frames
- plots, summaries, basic data processing
- read/write csv files

R and RStudio

R and the R community

Good to remember

R and R packages

- R is the program doing all the calculations. It is developed by the *R Core team*, consisting of 20 scholars.
- Users can publish *R packages* that add new functionality.

RStudio

RStudio makes it much easier to work with R. It is a separate software, developed by RStudio Inc.

Citing R, citing packages

```
# to cite R, type
citation()
# to cite R package 'validate', type:
citation("validate")
```

Please download *and unzip*

Go to: github.com/markvanderloo/UFPEL2019

Contents

Lectures 2, 3, and 4 will have a hands-on component so make sure you bring a laptop with the necessary software installed (described below).

Lecture	Content	Materials
1	Structuring data and analyses	
2	Reproducibility and introduction to R	r_intro_ufpel2019.zip
3	Data cleaning 1 raw data, data validation	
4	Data cleaning 2 fixing errors, missing data	

RStudio

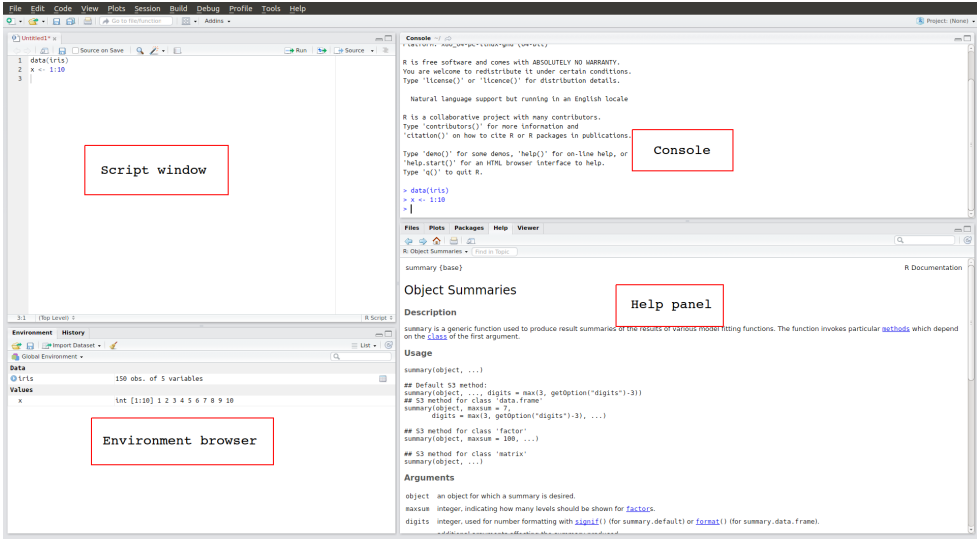


Figure 1:

https://github.com/markvanderloo/UFPEL2019

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

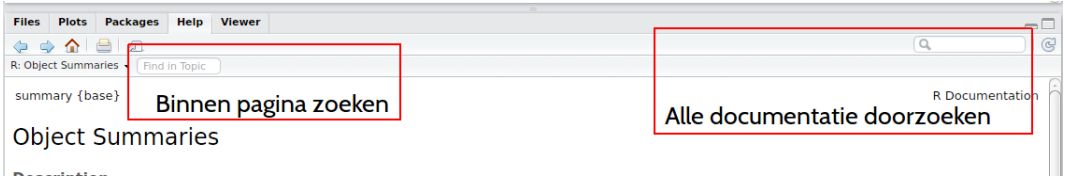


Figure 2: img

- 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

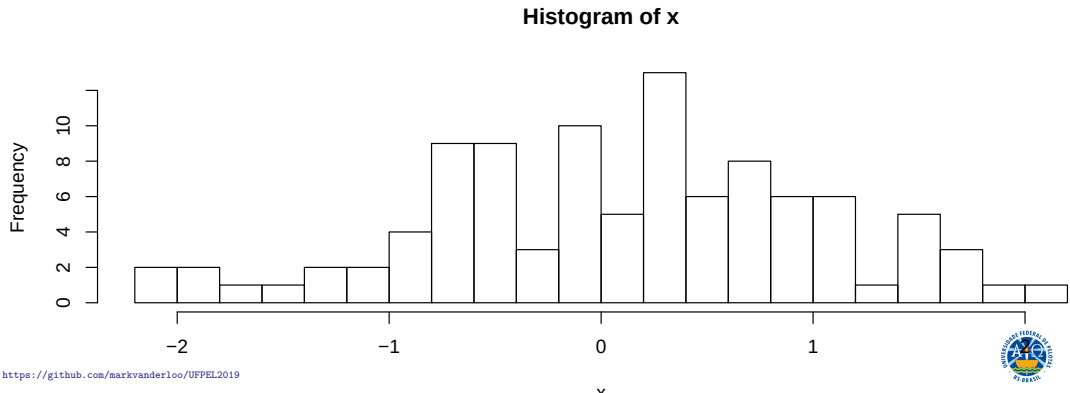
Use `tab` to complete names of objects, columns in `data.frames` and file names (between quotes).

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

```
# 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.

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

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-2.0919 -0.5674  0.1837  0.1008  0.6585  2.1498
```

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

```
[1] 0.9202403
```

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

```
[1] -0.06215967  0.02584545  0.24075953
```

Example (cont'd) metadata

```
length(x)
```

```
[1] 100
```

```
class(x)
```

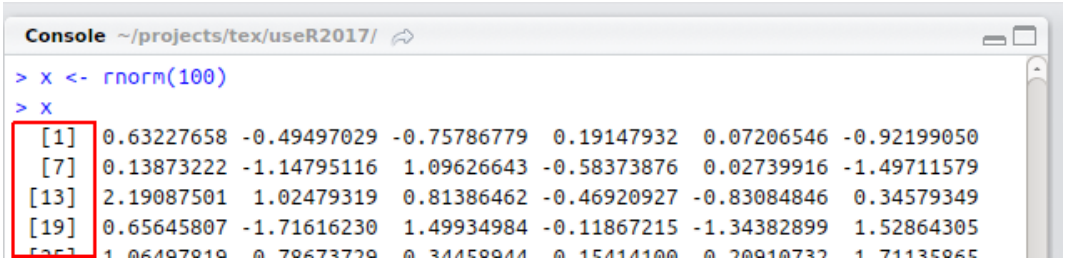
```
[1] "numeric"
```

```
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.



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

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

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

Variabelen can be overwritten

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

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

Computing with vectors

Addition etc works element-by-element.

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

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

```
x * y # multiply
```

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

```
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 `NA`with `na.rm=TRUE`

RStudio project | data import | data frames

Contents

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

Reading text files

Reading

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

Writing

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

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

File names in R

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

```
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

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

```
# 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)
```

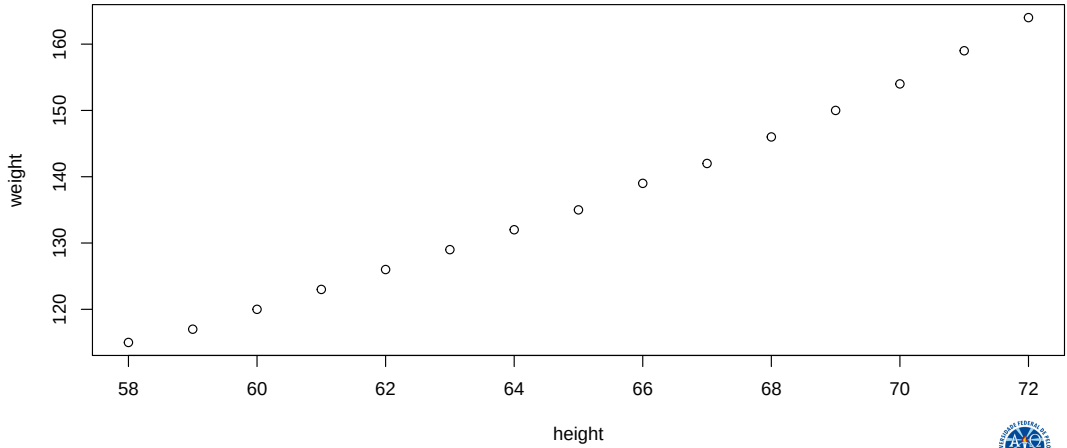
```
     count       spray
 Min.   : 0.00   A:12
 1st Qu.: 3.00   B:12
 Median : 7.00   C:12
 Mean   : 9.50   D:12
 3rd Qu.:14.25   E:12
 Max.   :26.00   F:12
```

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`

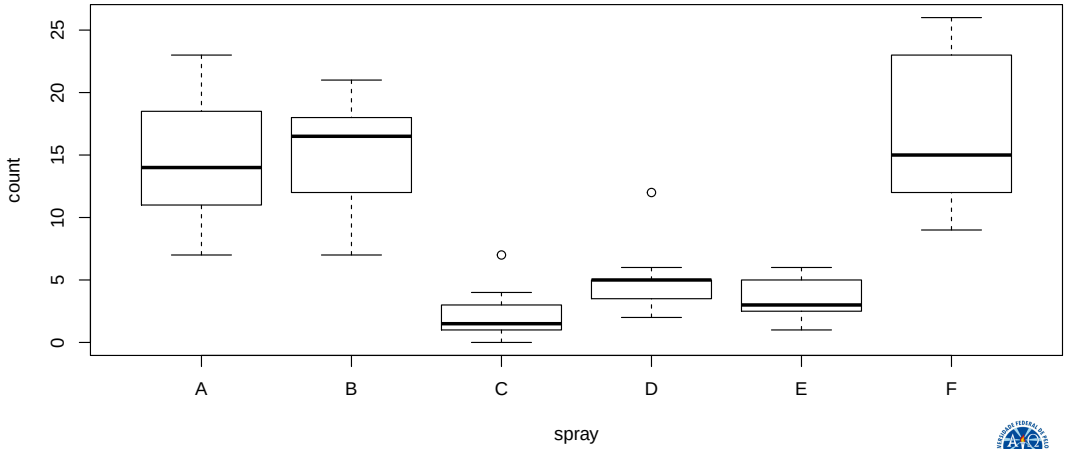
Plotting (1)

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



Plotting (2)

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



Plotting (3)

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

Introduction to data manipulation with `dplyr`[1]

```
library(dplyr)
```

Verbs for common operations

`filter`	Select rows
`select`	Select columns
`rename`	Rename columns
`distinct`	Keep unique rows
`arrange`	Sort
`transmute`	Compute new columns
`mutate`	Add new columns (or overwrite old ones)

[1] Wickham et al (2019). dplyr: A Grammar of Data Manipulation. https://CRAN.R-project.org/package=dplyr

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)
```