An introduction to 

Reading and writing data

Noémie Becker & Benedikt Holtmann

Winter Semester 16/17

Course outline

- Review – Data types and structures
- Reading data
  - How should data look like
  - Importing data into R
  - Checking and cleaning data
  - Common problems
- Writing data

Creating objects

General form:

variable <- value

Examples:

```
x <- 3                    # The variable ‘x’ is assigned the value ‘3’

y <- x^2 +3

MyFunction <- sqrt
```

Data types

Data type	Description	Example
logical	True or False	TRUE, FALSE
numeric	real numbers or decimal	2.3, pi, sqrt(2)
integer	whole numbers	-5L, 0L, 7L
character	character or string	“male”, “female”
complex	complex numbers	2.1+3i, 1+0i

Vectors

To create vectors, you can use the functions: **c()**, **seq()**, and **rep()**

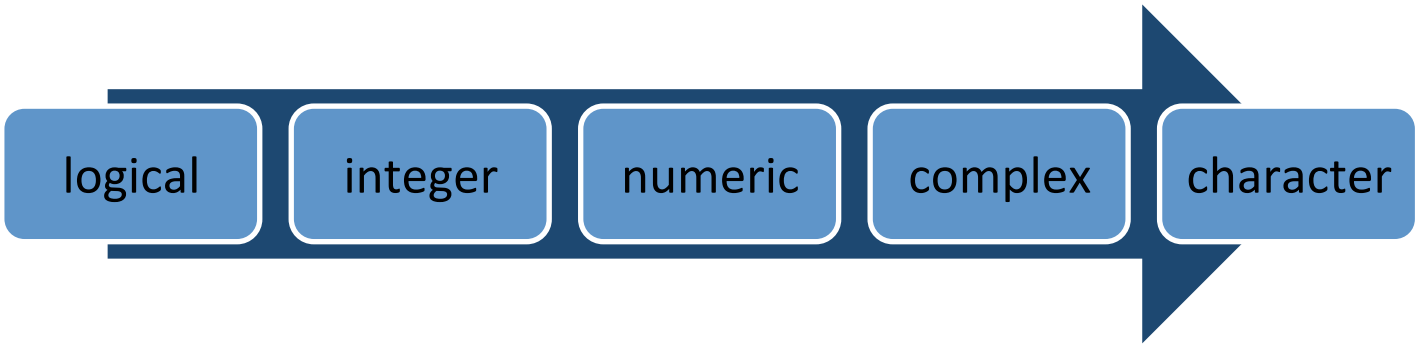
Examples:

```
c(2,7,8,12,3,25)
c(2:5, 3:6)
seq(1, 8)
seq(from=4, to=10, by=2)
seq(4, 10, 2)
rep(1, 4)
rep(4:5, 3)
rep(1:4, each = 2)
```

Coercion of vectors

You also can coerce vectors using the functions:

as.logical(x)
as.integer(x)
as.numeric(x)
as.complex(x)
as.character(x)
as.factor(x)



Vectors

To create vectors, you can use the functions: **c()**, **seq()**, and **rep()**

Examples:

```
c(2,7,8,12,3,25)
c(2:5, 3:6)
seq(1, 8)
seq(from=4, to=10, by=2)
seq(4, 10, 2)
rep(1, 4)
rep(4:5, 3)
rep(1:4, each = 2)
```

Vectors

To create vectors, you can use the functions: **c()**, **seq()**, and **rep()**

Examples:

```
c(2,7,8,12,3,25)
c(2:5, 3:6)
seq(1, 8)
seq(from=4, to=10, by=2)
seq(4, 10, 2)
rep(1, 4)
rep(4:5, 3)
rep(1:4, each = 2)
```

Matrices

To create matrices, you can use the functions: **matrix()** , **dim()**, and **cbind()** or **rbind()**

Examples:

```
m<- matrix(data=c(10:22), nrow = 2, ncol = 4)
m<- matrix(10:22, 2, 4)  # is the same

y <- 1:6
dim(y) <- c(3, 2)

cbind(1:3, 5:7)

rbind(1:3, 5:7, 10:12)
```

Data frames

To create data drames, you can use the functions: **data.frame()**

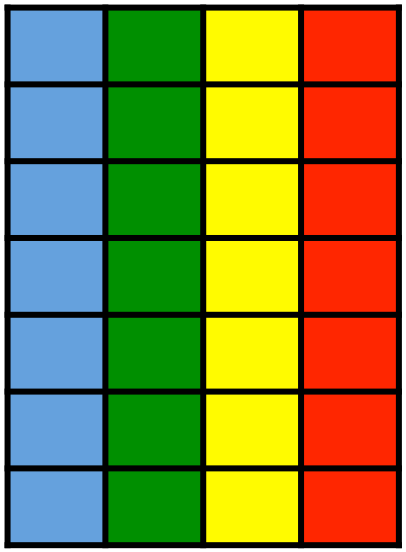
Examples:

```
df <- data.frame(ID = 1:3, Sex = c("F", "F", "M"), Mass = c(17, 18, 18))

df_2 <- data.frame(x = 1:3, y = c("a", "b", "c"))
```

Data frames

- A data frame is a very important data type in R
- Data frames are displayed in a tabular layout
- A data frame contains scientific observations and measurements, which are used for statistics



Accessing data structures

To access single or multiple elements of vectors, matrices and data frames you can use **x[ ]**

Examples:

```
x[7]
x[c(1,4,9,12)]
x[-2:-7]                    # excludes elements 2 to 7

z[2, 3]                     # row by column
z[2, ]                      # 2nd row
z[ , 3]                     # 3rd column

z[1:2, 1:2]
```

Accessing data structures

You can also access elements by name, using **x[ ]** or **$**

Examples:

```
df[c("Sex", "Mass")]
```

```
df$Mass
```

Workflow for reading and writing data frames

1) Import your data

2) Check, clean and prepare your data (can be up to 80% of your project)

3) Conduct your analyses

4) Export your results

5) Clean R environment and close session

How should data look like?

- Columns should contain variables
- Rows should contain observations, measurements, cases, etc.
- Use first row for the names of the variables
- Enter **NA** (in capitals) into cells representing missing values
- You should avoid names (or fields or values) that contain spaces
- Store data as **.csv** or **.txt** files as those can be easily read into R

How should data look like?

Bird_ID	Sex	Mass	Wing
Bird_1	F	17.45	75.0
Bird_2	F	18.20	75.0
Bird_3	M	18.45	78.25
Bird_4	F	17.36	NA
Bird_5	M	18.90	84.0
Bird_6	M	19.16	81.83

Import data

Import data using **read.table()** and **read.csv()** functions

Examples:

```
myData<- read.table(file = "datafile.txt")

myData<- read.csv(file = "datafile.csv")


# Creates a data frame named myData
```

Import data

Import data using **read.table()** and **read.csv()** functions

Examples:

```
myData<- read.csv(file = "datafile.csv")

# Error in file(file, "rt") : cannot open the connection
# In addition: Warning message:
# In file(file, "rt") :
#   cannot open file 'datafile.csv': No such file or directory
```

Important: Set your working directory (**setwd()**) first, so that R uses the right folder to look for your data file!

Import data

Check ?read.table or ?read.csv

read.csv(**file, header = FALSE, sep = " "**, quote = "\"",
dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
row.names, col.names, as.is = !stringsAsFactors,
na.strings = "NA", colClasses = NA, nrows = -1,
skip = 0, check.names = TRUE, fill = !blank.lines.skip,
strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#",
allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAsFactors(),
fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

Import data

Reduce errors when loading a data file

- The **header = TRUE** argument tells R that the first row of your file contains the variable names
- The **sep = ”,"** argument tells R that fields are separated by comma
- The **strip.white = TRUE** argument removes white space before or after factors that has been mistakenly insert during data entry (e.g. “small” vs. “small_” become both “small”)
- The **na.strings = " "** argument replaces empty cells by NA (missing data in R)

Import data

Reduce errors when loading a data file

```
myData<- read.csv(file = "datafile.csv”,
                  header = TRUE,
                  sep = ”,",
                  strip.white = TRUE,
                  na.strings = " ")
```

Import data

R allows to us a URL in place of a filename in R:

```
myData<- read.csv(file = "http://environmentalcomputing.net/wp-
                          content/uploads/2016/05/Snail_feeding.csv",
                          header = TRUE,
                          sep = ", ",
                          strip.white = TRUE,
                          na.strings = " ")
```

Import data

Missing and special values

NA = not available

Inf and -Inf = positive and negative infinity

NaN = Not a Number

NULL = argument in functions that means that no value was assigned to the argument

Import data

Missing and special values

Important command: is.na()

```
v <- c(1, 3, NA, 5)
is.na(v)
[1] FALSE FALSE TRUE FALSE
```

Ignore missing data: na.rm=TRUE

```
mean(v, na.rm=TRUE)
[1] 3
```

Import data from other programs

File format	function	library
ERSI ArcGIS	read.shapefile()	shapefiles
Matlap	readMat()	R.matlap
minitab	read.mtp()	foreign
SAS (permanent data)	read.ssd()	foreign
SAS (XPORT format)	read.xport()	foreign
SPSS	read.spss()	foreign
Stata	read.dta()	foreign
Systat	read.systat()	foreign

Import objects

R objects can be imported with the load( ) function:

Usually model outputs such as 'YourModel**.Rdata**'

Example:

```
load("~/Desktop/YourModel.Rdata")
```

Checking and cleaning data

An example on marine snails provided by

Environmental Computing

www.environmentalcomputing.net

Checking and cleaning data

Set directory:
setwd("~/Desktop")

Import the sample data:

```
Snail_data<- read.csv(file = "Snail_feeding.csv",
                      header = T,
                      strip.white = T,
                      na.strings = " ")
```

Checking and cleaning data

Use the **str()** command to check the status and data type of each variable:

str(Snail_data)

```
'data.frame':   769 obs. of  12 variables:
$ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
$ Sex      : Factor w/ 4 levels "female","male",..: 2 2 4 2 2 2 2 2 2 2 ...
$ Size     : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
$ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
$ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
$ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
$ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
$ X        : logi  NA NA NA NA NA NA ...
$ X.1     : logi  NA NA NA NA NA NA ...
$ X.2     : logi  NA NA NA NA NA NA ...
```

Checking and cleaning data

Use the **str()** command to check the status and data type of each variable:

str(Snail_data)

```
'data.frame':   769 obs. of  12 variables:
$ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
$ Sex      : Factor w/ 4 levels "female","male",..: 2 2 4 2 2 2 2 2 2 2 ...
$ Size     : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
$ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
$ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
$ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
$ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
$ X       : logi  NA NA NA NA NA NA ...
$ X.1     : logi  NA NA NA NA NA NA ...
$ X.2     : logi  NA NA NA NA NA NA ...
```

Checking and cleaning data

To get rid of the extra columns we can just choose the columns we need by using **Snail_data[m, n]**

Checking and cleaning data

To get rid of the extra columns we can just choose the columns we need by using **Snail_data[m, n]**

```
Snail_data <- Snail_data[ , 1:7]          # takes columns 1 to 7

str(Snail_data)
```

```
'data.frame': 769 obs. of  7 variables:
$ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
$ Sex     : Factor w/ 4 levels "female","male",..: 2 2 4 2 2 2 2 2 2 2 ...
$ Size    : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
$ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
$ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
$ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
$ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
```

Checking and cleaning data


```
'data.frame':   769 obs. of  7 variables:
 $ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sex     : Factor w/ 4 levels "female","male",..: 2 2 4 2 2 2 2 2 2 2 ...
 $ Size    : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
 $ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
 $ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
 $ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
```

Checking and cleaning data

```
'data.frame':   769 obs. of  7 variables:
 $ Snail.ID: int  1 1 1 1 1 1 1 1 1 1 ...
 $ Sex     : Factor w/ 4 levels "female","male",..: 2 2 4 2 2 2 2 2 2 2 ...
 $ Size    : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 2 ...
 $ Feeding : logi  FALSE FALSE FALSE FALSE FALSE TRUE ...
 $ Distance: num  0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.6 0.85 0.59 ...
 $ Depth   : num  1.66 1.26 1.43 1.46 1.21 1.56 1.62 162 1.96 1.93 ...
 $ Temp    : int  21 21 18 19 21 21 20 20 19 19 ...
```

The variable Sex has 4 levels, when it should have only two (“female” and “male”)

Checking and cleaning data

You can check the levels of a factor or character with the **unique()** or **levels()**

```
unique(Snail_data$Sex)
```

```
[1] male   males  Male   female
Levels: female male Male males
```

Checking and cleaning data

To turn “males” or “Male” into the correct “male”, you can use the the **[ ]-Operator** together with the **which()** function:

```
Snail_data$Sex[which(Snail_data$Sex == "males")] <- "male”

Snail_data$Sex[which(Snail_data$Sex == "Male")] <- "male”
```

Checking and cleaning data

To turn “males” or “Male” into the correct “male”, you can use the the **[ ]-Operator** together with the **which()** function:

```
Snail_data$Sex[which(Snail_data$Sex == "males")] <- "male"

Snail_data$Sex[which(Snail_data$Sex == "Male")] <- "male"
```

Or both together:

```
Snail_data$Sex[which(Snail_data$Sex == "males" |
                     Snail_data$Sex == "Male")] <- "male"
```

Checking and cleaning data

Check if it worked using **unique()**

```
unique(Snail_data$Sex)
```

```
[1] male     female
Levels: female male Male males
```

Checking and cleaning data

Check if it worked using **unique()**

```
unique(Snail_data$Sex)
```

```
[1] male     female
Levels: female male Male males
```

You can remove the extra levels using **factor()**

```
Snail_data$Sex <- factor(Snail_data$Sex)
```

```
unique(Snail_data$Sex)
# [1] male   female
# Levels: female male
```

Checking and cleaning data

The **summary()** function provides summary statistics for each variable:

summary(Snail_data**)**

```
   Snail.ID         Sex          Size          Feeding         Distance      
Min.   : 1.00   female:384   large:383   Mode :logical   Min.   :0.0000  
1st Qu.: 4.00   male  :385   small:385   FALSE:503       1st Qu.:0.2800  
Median : 8.50                NA's :  1   TRUE :266       Median :0.5100  
Mean   : 8.49                            NA's :0         Mean   :0.5125  
3rd Qu.:12.00                                            3rd Qu.:0.7500  
Max.   :16.00                                            Max.   :1.0000 ...
...
...
Continues
```

Checking and cleaning data

The **summary()** function provides summary statistics for each variable:

```
summary(Snail_data)
```

…

…

Continued

```
     Depth             Temp
Min.   :  1.000   Min.   :18.00
1st Qu.:  1.260   1st Qu.:19.00
Median :  1.510   Median :19.00
Mean   :  1.716   Mean   :19.49
3rd Qu.:  1.760   3rd Qu.:20.00
Max.   :162.000   Max.   :21.00
NA's   :6
```

Checking and cleaning data

The **summary()** function provides summary statistics for each variable:

summary(Snail_data**)**

…

…

Continued

Depth	Temp
Min. : 1.000	Min. :18.00
1st Qu.: 1.260	1st Qu.:19.00
Median : 1.510	Median :19.00
Mean : 1.716	Mean :19.49
3rd Qu.: 1.760	3rd Qu.:20.00
Max. :162.000	Max. :21.00
NA's :6	

Checking and cleaning data

To find depths greater than 2m you can use the **[ ]-Operator** together with the **which()** function:

```
Snail_data[which(Snail_data$Depth > 2), ]
```

```
  Snail.ID  Sex   Size   Feeding  Distance  Depth  Temp
8  1        male  small  TRUE       0.6     162    20
```

Checking and cleaning data

To find depths greater than 2m you can use the **[ ]-Operator** together with the **which()** function:

```
Snail_data[which(Snail_data$Depth > 2), ]
```

```
  Snail.ID  Sex   Size   Feeding  Distance  Depth  Temp
8    1       male  small  TRUE       0.6     162    20
```

Replace value

```
Snail_data[8, 6] <- 1.62
```

Checking and cleaning data

Finding and removing duplicates using **duplicated()**

Example:

duplicated(Snail_data)

Checking and cleaning data

Finding and removing duplicates using **duplicated()**

```
duplicated(Snail_data)

[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE …
[15] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE …
[29] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE …
[43] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE …
…                              # Cut to reduce space
```

duplicated() shows that the 17th row is a duplicate of an earlier row

Checking and cleaning data

To remove duplicate rows you can use the **[ ]-Operator** together with the **duplicated()** function:

```
Snail_data<- Snail_data[!duplicated(Snail_data), ]
```

Or use **unique()**

```
Snail_data<- unique(Snail_data)
```

Check if it worked

```
duplicated(Snail_data)
```

Checking and cleaning data

Finding and removing duplicates using **duplicated()**

Faster when you incorporate **which()**

```
Snail_data[which(duplicated(Snail_data)), ]
```

	Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp	X	X.1	X.2
17	1	male	small	FALSE	0.87	1.95	18	NA	NA	NA

Checking and cleaning data

Two other operations that might be useful to get an overview of your data are **sort()** and **order()**

Checking and cleaning data

Two other operations that might be useful to get an overview of your data are **sort()** and **order()**

Sorting single vectors

```
sort(Snail_data$Depth)

[1] 1.00 1.00 1.00 1.00 1.00 1.01 1.01 1.01 1.01 1.01 1.02 1.02 ...
[18] 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.03 1.04 1.04 ...
[35] 1.05 1.05 1.05 1.06 1.06 1.06 1.06 1.06 1.06 1.06 1.06 1.07 ...
[52] 1.07 1.07 1.07 1.08 1.08 1.08 1.09 1.09 1.09 1.09 1.09 1.09 ...
...                                  # Cut to reduce space
```

Checking and cleaning data

Two other operations that might be useful to get an overview of your data are **sort()** and **order()**

Sorting data frames

```
Snail_data[order(Snail_data$Depth, Snail_data$Temp), ]
```

	Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp
494	11	female	small	FALSE	0.76000	1.00	18
607	13	female	large	FALSE	0.45000	1.00	18
86	2	male	small	FALSE	0.09000	1.00	19
239	5	male	large	TRUE	0.03000	1.00	20
511	11	female	small	TRUE	0.62000	1.00	21
...			# Cut to reduce space				

Checking and cleaning data

Two other operations that might be useful to prepare your data are **sort()** and **order()**

Sorting data frames in decreasing order

```
Snail_data[order(Snail_data$Depth, Snail_data$Temp,
decreasing=TRUE),  ]
```

	Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp
762	16	female	large	FALSE	0.92000	2.00	21
412	9	female	small	TRUE	0.48000	2.00	19
37	1	male	small	FALSE	0.67000	2.00	18
155	4	male	small	FALSE	0.38000	2.00	18
434	11	female	small	FALSE	0.49000	2.00	18

... # Cut to reduce space

Checking and cleaning data

Large data frames

Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp
762	16	female	large	FALSE	0.92000	2.00	21
412	9	female	small	TRUE	0.48000	2.00	19
37	1	male	small	FALSE	0.67000	2.00	18
155	4	male	small	FALSE	0.38000	2.00	18
434	11	female	small	FALSE	0.49000	2.00	18

... **# Cut to reduce space**

[reached getOption("max.print") -- omitted 626 rows]

Checking and cleaning data

To get an overview of an object use or **head()** or **tail()**

Examples:

head(Snail_data, n = 4) # returns first 4 rows of Snail_data

	Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp
1	1	male	small	FALSE	0.17	1.66	21
2	1	male	small	FALSE	0.87	1.26	21
3	1	male	small	FALSE	0.22	1.43	18
4	1	male	small	FALSE	0.13	1.46	19

Checking and cleaning data

To get an overview of an object use or **head()** or **tail()**

Examples:

tail(Snail_data, n = 4) # returns last 4 rows of Snail_data

	Snail.ID	Sex	Size	Feeding	Distance	Depth	Temp
766	16	female	large	TRUE	0.65	1.71	20
767	16	female	large	TRUE	0.46	1.27	19
768	16	female	large	FALSE	0.36	1.28	21
769	16	female	large	FALSE	0.42	1.82	19

Checking and cleaning data

To get an overview of an object use or **head()** or **tail()**

Examples:

head() and **order()** combined

```
head(Snail_data[order(Snail_data$Depth),], n=10)

# returns first 10 rows of Snail_data with increasing depth
```

Export data

To export data use the **write.table()** or **write.csv()** functions

Check **?read.table** or **?read.csv**

```
write.table(x, file = " ", append = FALSE, quote = TRUE, sep = " ",
            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
            col.names = TRUE, qmethod = c("escape", "double"),
            fileEncoding = "")
```

Export data

To export data use the **write.table()** or **write.csv()** functions

Example:

```
write.csv(Snail_data,                          # object you want export
          file = "Snail_data_checked.csv",     # file name saved
          row.names = FALSE)                   # exclude row names
```

Export objects

To export R objects, such as model outputs, use the function **save()**

Example:

```
save(My_t-test, file = "T-test_master_thesis.Rdata")
```

Export data

At the end use **rm()** to clean the R environment

rm(list=ls()) # will remove all objects from the memory

Why do this in R?

- You can follow which changes are made
- Set up a script already when only part of the data is available
- It is quick to run the script again on the full data set

Which R functions did we learn?

read.table() read.csv()	Import data saved as text file (.txt) or as comma separated format (.csv) into R
is.na()	indicates which elements are missing (NA)
load()	reloads datasets written with the function save
str()	provides an overview of an object
summary()	returns basic statistical summary for variables
duplicated()	indicates which rows are duplicates
unique()	removes duplicate elements
which()	allows to select specific elements from a data frame
sort()	sorts values in a vector or a factor
order()	sorts data by a set of variables at the same time
head()	returns the first records of an object
tail()	returns the last records of an object

Which R functions did we learn?

write.table() write.csv()	save a data frame as text file (.txt) or as comma separated format (.csv) into R
save()	writes an external representation of R objects
rm() rm(list=ls())	removes specific or all objects from working environment