
```
x <- 0:5
y <- sum(x[-6]+2)
```

What it the value of `y` ?

Answer: 20

An introduction to R
WS 2019/2020

Reading and writing data

Dr. Noémie Becker
Dr. Eliza Argyridou

Special thanks to:
Prof. Dr. Martin Hutzenthaler, Dr. Sonja Grath and Dr. Benedikt Holtmann for significant contributions to course development, lecture notes and exercises

What you should know after day 4

Part I: Reading data

- How should data look like
- Importing data into R
- Checking and cleaning data
- Common problems

Part II: Writing data

Work flow for reading and writing data frames

1) Import your data
2) Check, clean and prepare your data
3) Conduct your analyses
4) Export your results
5) Clean R environment and close session

What you should know after day 4

Part I: Reading data

- How should data look like
- Importing data into R
- Checking and cleaning data
- Common problems

Part II: Writing data

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

Example

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

IMPORTANT:
All values of the same variable MUST go in the same column!

Example: Data of expression study
3 groups/treatments: Control, Tropics, Temperate
4 measurements per treatment

Control	Tropical	Temperate
6.1	6.3	7.1
5.9	6.2	8.2
5.8	5.8	7.3
5.4	6.3	6.9

NOT a data frame!

Same data as data frame

Response	Treatment
6.1	control
5.9	control
5.8	control
5.4	control
6.3	tropical
6.2	tropical
5.8	tropical
6.3	tropical
7.1	temperate
8.2	temperate
7.3	temperate
6.9	temperate

What you should know after day 4

Part I: Reading data

- How should data look like
- Importing data into R
- Checking and cleaning data
- Common problems

Part II: Writing data

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

Example:

```
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! And check for typos!

Useful arguments

You can reduce possible 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 inserted during data entry (e.g. “small” vs. “small ” become both “small”)
- The **na.strings = " "** argument replaces empty cells by NA (missing data in R)

Useful arguments

Check these arguments carefully when you load your data

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


Missing and special values

NA	= not available
Inf and -Inf	= positive and negative infinity
NaN	= Not a Number
NULL	= argument in functions meaning that no value was assigned to the argument

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)
mean(v, na.rm = TRUE)
```


Import objects

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

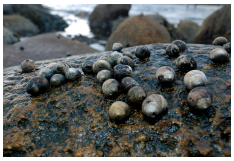
Usually model outputs such as ‘YourModel**.Rdata**’

Example:

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


Real world example on marine snails

`Snail_feeding.csv`



www.environmentalcomputing.net

Environmental Computing

We have integrated some problems with the data set (typos, duplicates, weird values...) – on purpose!

You now should recognize these problems and take care of them.

→ We provide a script with a possible solution on the webpage (`Day4_script.R`)

Our tasks

- Download the data `Snail_feeding.csv`
- Import the data into R
- Get an overview of the data
- Identify possible problems and solve these problems
- Export the corrected data into the file `Snail_data_checked.csv`
- Clean the R session

Importing the data

Download the file `Snail_feeding.csv` from the course page.

Set directory, for example:

```
setwd("~/Desktop/Day_4")
```

Import the sample data into a variable `Snail_data`:

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

Note: Remember that you can chose the name for your variable yourself if not stated otherwise – '`Snail_data`' is just an option.

Get an overview of your data

After you read in your data, you can briefly check it with some useful commands:

summary()	provides summary statistics for each variable
names()	returns the column names
str()	gives overall structure of your data
head()	returns the first lines (default: 6) of the file and the header
tail()	returns the last lines of the file and the header


```
summary(Snail_data)
names(Snail_data)
str(Snail_data)
head(Snail_data)
tail(Snail_data)
head(Snail_data, n = 10)
```


Our problems

Problem 1: What are the last 3 columns?
Solution: We want to remove them

Problem 2: Why does the variable 'Sex' has 4 levels?
Solution: We would expect 'male' and 'female' – we want to correct the levels.

Problem 3: Possible problem with variable 'Depth' - Max: 162, Mean: 1.7?
Solution: We want to find the entry with the maximum depth and want to change the value to 1.62 (162 was a typo)

Problem 4: Do we have any duplicated rows?
Solution: If yes, we want to remove these rows.

Note: The most complicated problem is problem 3 – we will solve it last.

What you should know after day 4

Part I: Reading data

- How should data look like
- Importing data into R
- Checking and cleaning data
- Common problems

Part II: Writing data

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  10 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 ...
```

Problem 1:
What are the last 3 columns?
→ We want to ignore them

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]**

```
# we are interested in columns 1:7
Snail_data <- Snail_data[ , 1:7]
# get an overview of your data
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 ...
```

Problem 2:
Why do we have 4 levels in the column 'Sex'? 'male', 'female' and what?
→ We want to check that and keep 'male' and 'female'

Checking and cleaning data

Something seems to be weird with the column 'Sex' …

```
unique(Snail_data$Sex)
```
Or
```
levels(Snail_data$Sex)
```

To turn “males” or “Male” into the correct “male”, you can use 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 with **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          Depth             Temp
Min.   : 1.00   female:384   large:383   Mode :logical   Min.   :0.0000   Min.   :  1.000   Min.   :18.00
1st Qu.: 4.00   male  :385   small:385   FALSE:503       1st Qu.:0.2800   1st Qu.:  1.260   1st Qu.:19.00
Median : 8.00                NA's :  1   TRUE :266       Median :0.5100   Median :  1.510   Median :19.00
Mean   : 8.49                            NA's :0         Mean   :0.5125   Mean   :  1.716   Mean   :19.49
3rd Qu.:12.00                                            3rd Qu.:0.7500   3rd Qu.:  1.760   3rd Qu.:20.00
Max.   :16.00                                            Max.   :1.0000   Max.   :162.000   Max.   :21.00
                                                                                            NA's   :6
```


Finding and removing duplicates

Problem 4:
We want to check if our data contain any duplicated rows – in this particular case, we want to remove duplicates (in your own data you should know best what to do and what to expect)

Function: **duplicated()**

Example:
```
duplicated(Snail_data)
```

… truly helpful?

Finding and removing duplicates

Function: **duplicated()**

Example:
```
duplicated(Snail_data)
```

… truly helpful?

```
sum(duplicated(Snail_data))
```

… Ah! Better! Think: Why does it actually work with **sum()**?

You probably want to know WHICH row is duplicated: **which()**

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

Finding and removing duplicates

To remove duplicated 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
sum(duplicated(Snail_data))
```


CAUTION

R does not re-set the row number automatically!
You have to do this 'by hand'.

What do I mean? Remember: The duplicated row we just removed, was row 17.

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

Let us have a look at the first 20 rows of our data.

```
head(Snail_data, n = 20)
```

```
16       1 male small   FALSE     0.87   1.95   18
18       1 male small   FALSE     0.32   1.66   21
19       1 male small   FALSE     0.93   1.95   19
```

```
# Re-number the rows
row.names(Snail_data) <- 1:nrow(Snail_data)
```


Operations on vectors

You assess elements of a vector with the [ ]-operator:

```
x <- c(12, 15, 13, 17, 11)
x[4]
x[3:5]
x[-2]
x[-(3:5)]
```

Example vector x:

```
x <- 1:5
x[3:5]
x[-2]
x > 3
```

Some useful functions

`class(x)`	returns class of vector x
`length(x)`	returns the total number of elements
`x[length(x)]`	returns last value of vector x
`rev(x)`	returns reversed vector
`sort(x)`	returns sorted vector
`unique(x)`	returns vector without multiple elements

More operations on vectors

```
x <- c(12, 15, 13, 17, 11)
x[x>12] <- 0
x[x==0] <- 2
sum(x==2)
[1] 3
x==2
[1] FALSE TRUE TRUE TRUE FALSE
as.integer(x==2)
[1] 0 1 1 1 0
```


```
x <- 1:10
y <- c(1:5, 1:5)
# compare:
x == y
x = y
```


More operations on vectors

```
v <- c(13,15,11,12,19,11,17,19)
length(v)     # returns the length of v
rev(v)        # returns the reversed vector
sort(v)       # returns the sorted vector
unique(v)     # returns vector without multiple elements

some_values <- (v > 13)
which(some_values)    # indices where 'some_values' is
                      # TRUE
which.max(v)          # index of (first) maximum
which.min(v)          # index of (first) minimum
```

Brainteaser: How can you get the indices for ALL minima?

More operations on vectors

```
v <- c(13,15,11,12,19,11,17,19)
length(v)     # returns the length of v
rev(v)        # returns the reversed vector
sort(v)       # returns the sorted vector
unique(v)     # returns vector without multiple elements

some_values <- (v > 13)
which(some_values)    # indices where 'some_values' is
                      # TRUE
which.max(v)          # index of (first) maximum
which.min(v)          # index of (first) minimum
```

Brainteaser: How can you get the indices for ALL minima?

```
all_minima <- (v == min(v))
which(all_minima)
```

The real world again …

```
summary(Snail_data)
```

```
   Snail.ID        Sex        Size     Feeding          Distance         Depth             Temp
Min.   : 1.00   female:384   large:383   Mode :logical   Min.   :0.0000   Min.   :  1.000   Min.   :18.00
1st Qu.: 4.00   male  :385   small:385   FALSE:503       1st Qu.:0.2800   1st Qu.:  1.260   1st Qu.:19.00
Median : 8.00                NA's :  1   TRUE :266       Median :0.5100   Median :  1.510   Median :19.00
Mean   : 8.49                            NA's :0         Mean   :0.5125   Mean   :  1.716   Mean   :19.49
3rd Qu.:12.00                                            3rd Qu.:0.7500   3rd Qu.:  1.760   3rd Qu.:20.00
Max.   :16.00                                            Max.   :1.0000   Max.   :162.000   Max.   :21.00
                                                                                            NA's   :6
```

?

Problem 3:
Possible problem with variable 'Depth' - Max: 162, Mean: 1.7?
The lecturer tells us, it was a typo and the value should be 1.62.

The real world again …

To find depths greater than 2 meter 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

which.max(Snail_data$Depth)
```

Replace value:

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

summary(Snail_data)
```


Sorting 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)
```

Sorting data frames

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

Sorting data frames in decreasing order

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

Example:
head() and **order()** combined

```
# returns first 10 rows of Snail_data with
# increasing depth
head(Snail_data[order(Snail_data$Depth),], n=10)
```


What you should know after day 4

Part I: Reading data

- How should data look like
- Importing data into R
- Checking and cleaning data
- Common problems → we just covered most of them!!

Part II: Writing data

What you should know after day 4

Part I: Reading data

- How should data look like
- Importing data into R
- Checking and cleaning data
- Common problems

Part II: Writing data

Exporting data

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

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

Example:

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

Exporting objects

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

Example:

```
save(My_t_test, file = "T_test_master_thesis.Rdata")
```


Cleaning up the environment

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

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

Take-home message

Typical call:

```
read.table("filename.txt", header = TRUE)
read.csv("filename.csv", header = TRUE)

write.table(dataframe, file = "filename.txt")
write.csv(dataframe, file = "filename.csv")
```

Command	header	sep	dec	fill
read.table()	FALSE	""	"."	FALSE
read.csv()	TRUE	","	"."	TRUE
read.csv2()	TRUE	";"	","	TRUE
read.delim()	TRUE	"\t"	"."	TRUE
read.delim2()	TRUE	"\t"	","	TRUE

Why do all 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 (and again ...) on the full data set