An introduction to R

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:**

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

y <- x^2 + 3

MyFunction <- sqrt
```
### Data types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical</td>
<td>True or False</td>
<td>TRUE, FALSE</td>
</tr>
<tr>
<td>numeric</td>
<td>real numbers or decimal</td>
<td>2.3, pi, sqrt(2)</td>
</tr>
<tr>
<td>integer</td>
<td>whole numbers</td>
<td>-5L, 0L, 7L</td>
</tr>
<tr>
<td>character</td>
<td>character or string</td>
<td>“male”, “female”</td>
</tr>
<tr>
<td>complex</td>
<td>complex numbers</td>
<td>2.1+3i, 1+0i</td>
</tr>
</tbody>
</table>
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: \texttt{c()}, \texttt{seq()}, and \texttt{rep()}

Examples:

\texttt{c(2,7,8,12,3,25)}
\texttt{c(2:5, 3:6)}
\texttt{seq(1, 8)}
\texttt{seq(from=4, to=10, by=2)}
\texttt{seq(4, 10, 2)}
\texttt{rep(1, 4)}
\texttt{rep(4:5, 3)}
\texttt{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:

```r
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 frames, you can use the functions: `data.frame()`

Examples:

```r
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 \texttt{x[ ]} or $\texttt{\$}$.

**Examples:**

\begin{verbatim}
  df[c("Sex", "Mass")]

  df$Mass
\end{verbatim}
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?**

<table>
<thead>
<tr>
<th>Bird_ID</th>
<th>Sex</th>
<th>Mass</th>
<th>Wing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bird_1</td>
<td>F</td>
<td>17.45</td>
<td>75.0</td>
</tr>
<tr>
<td>Bird_2</td>
<td>F</td>
<td>18.20</td>
<td>75.0</td>
</tr>
<tr>
<td>Bird_3</td>
<td>M</td>
<td>18.45</td>
<td>78.25</td>
</tr>
<tr>
<td>Bird_4</td>
<td>F</td>
<td>17.36</td>
<td>NA</td>
</tr>
<tr>
<td>Bird_5</td>
<td>M</td>
<td>18.90</td>
<td>84.0</td>
</tr>
<tr>
<td>Bird_6</td>
<td>M</td>
<td>19.16</td>
<td>81.83</td>
</tr>
</tbody>
</table>
Import data

Import data using \texttt{read.table()} and \texttt{read.csv()} functions

\textbf{Examples:}

\begin{verbatim}
myData <- read.table(file = "datafile.txt")

myData <- read.csv(file = "datafile.csv")
\end{verbatim}

\texttt{# 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`

```r
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 = " ")
R allows to us a URL in place of a filename in R:

```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()`

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

Ignore missing data: `na.rm=TRUE`

```r
mean(v, na.rm=TRUE)
[1] 3
```
### Import data from other programs

<table>
<thead>
<tr>
<th>File format</th>
<th>function</th>
<th>library</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERSI ArcGIS</td>
<td>read.shapefile()</td>
<td>shapefiles</td>
</tr>
<tr>
<td>Matlab</td>
<td>readMat()</td>
<td>R.matlab</td>
</tr>
<tr>
<td>minitab</td>
<td>read.mtp()</td>
<td>foreign</td>
</tr>
<tr>
<td>SAS (permanent data)</td>
<td>read.ssd()</td>
<td>foreign</td>
</tr>
<tr>
<td>SAS (XPORT format)</td>
<td>read.xport()</td>
<td>foreign</td>
</tr>
<tr>
<td>SPSS</td>
<td>read.spss()</td>
<td>foreign</td>
</tr>
<tr>
<td>Stata</td>
<td>read.dta()</td>
<td>foreign</td>
</tr>
<tr>
<td>Systat</td>
<td>read.systat()</td>
<td>foreign</td>
</tr>
</tbody>
</table>
**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
Reading and writing data

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

```r
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 FALSE TRUE ...  
$ Distance: num 0.17 0.87 0.22 0.13 0.36 0.84 0.69 0.85 0.59 ...  
$ Depth : num 1.66 1.26 1.43 1.46 1.21 1.56 1.62 1.96 1.31 1.93 ...  
$ Temp : int 21 21 18 19 21 22 20 20 19 19  ...  
$ X : logi NA NA NA NA NA NA NA NA NA NA ...  
$ X.1 : logi NA NA NA NA NA NA NA NA NA NA ...  
$ X.2 : logi NA NA NA NA NA NA NA NA NA NA ... 
```
Reading and writing data

Checking and cleaning data

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

```r
str(Snail_data)
```

'data.frame': 769 obs. of 12 variables:
$ Snail.ID: int 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 ...
$ Size : Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 2 ...
$ Feeding : logi FALSE 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 1.62 1.96 1.93 ... 
$ Temp : int 21 21 18 19 21 21 20 20 19 19 ... 
$ X : logi NA NA NA NA NA NA NA ... 
$ X.1 : logi NA NA NA NA NA NA NA ... 
$ X.2 : logi NA 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 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 ...
```
Reading and writing data

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 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 ...
Size: Factor w/ 2 levels "large","small": 2 2 2 2 2 2 2 2 ...
Feeding: logi FALSE FALSE FALSE 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 1.62 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()` functions.

```r
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:

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

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

[1] male  female
Levels: female male Male males
Checking and cleaning data

Check if it worked using `unique()`

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

[1] male  female
Levels: female male Male males

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

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

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

# [1] male  female
# Levels: female male
The `summary()` function provides summary statistics for each variable:

```r
summary(Snail_data)
```

<table>
<thead>
<tr>
<th>Variable</th>
<th>Min.</th>
<th>1st Qu.</th>
<th>Median</th>
<th>Mean</th>
<th>3rd Qu.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Snail.ID</td>
<td>1.00</td>
<td>4.00</td>
<td>8.50</td>
<td>8.49</td>
<td>12.00</td>
<td>16.00</td>
</tr>
<tr>
<td>Sex</td>
<td>female:384</td>
<td>male :385</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>large:383</td>
<td>small:385</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>NA's: 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Feeding</td>
<td>Mode :logical</td>
<td>FALSE:503</td>
<td>TRUE :266</td>
<td>NA's :0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distance</td>
<td>Min. :0.0000</td>
<td>1st Qu.:0.2800</td>
<td>Median :0.5100</td>
<td>Mean :0.5125</td>
<td>3rd Qu.:0.7500</td>
<td>Max. :1.0000</td>
</tr>
</tbody>
</table>

...  
...  
Continues
The `summary()` function provides summary statistics for each variable:

```r
summary(Snail_data)
```

Continued

<table>
<thead>
<tr>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min. : 1.000</td>
<td>Min. :18.00</td>
</tr>
<tr>
<td>1st Qu.: 1.260</td>
<td>1st Qu.:19.00</td>
</tr>
<tr>
<td>Median : 1.510</td>
<td>Median :19.00</td>
</tr>
<tr>
<td>Mean : 1.716</td>
<td>Mean :19.49</td>
</tr>
<tr>
<td>3rd Qu.: 1.760</td>
<td>3rd Qu.:20.00</td>
</tr>
<tr>
<td>Max. :162.000</td>
<td>Max. :21.00</td>
</tr>
<tr>
<td>NA's :6</td>
<td></td>
</tr>
</tbody>
</table>
The `summary()` function provides summary statistics for each variable:

```r
summary(Snail_data)
```

```
...  
...  
Continued

<table>
<thead>
<tr>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Min.  : 1.000</td>
<td>Min. : 18.00</td>
</tr>
<tr>
<td>1st Qu.: 1.260</td>
<td>1st Qu.: 19.00</td>
</tr>
<tr>
<td>Median : 1.510</td>
<td>Median : 19.00</td>
</tr>
<tr>
<td>Mean   : 1.716</td>
<td>Mean   : 19.49</td>
</tr>
<tr>
<td>3rd Qu.: 1.760</td>
<td>3rd Qu.: 20.00</td>
</tr>
<tr>
<td>Max.   : 162.000</td>
<td>Max. : 21.00</td>
</tr>
<tr>
<td>NA's   : 6</td>
<td></td>
</tr>
</tbody>
</table>
Checking and cleaning data

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

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

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>TRUE</td>
<td>0.6</td>
<td>162</td>
</tr>
</tbody>
</table>
Reading and writing data

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), ]
```

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>TRUE</td>
<td>0.6</td>
<td>162</td>
</tr>
</tbody>
</table>

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 FALSE FALSE FALSE FALSE FALSE FALSE ...
[15] FALSE FALSE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE ...
[29] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE ...
[43] FALSE FALSE FALSE FALSE FALSE FALSE 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:

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

Or use `unique()`

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

Check if it worked
```r
duplicated(Snail_data)
```
Reading and writing data

Checking and cleaning data

Finding and removing duplicates using \texttt{duplicated()}

Faster when you incorporate \texttt{which()}

\begin{verbatim}
Snail_data[which(duplicated(Snail_data)), ]
\end{verbatim}

\begin{verbatim}
    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
\end{verbatim}
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  1.01  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
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), ]
```

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>494</td>
<td>11</td>
<td>female</td>
<td>small</td>
<td>FALSE</td>
<td>0.76000</td>
<td>1.00</td>
</tr>
<tr>
<td>607</td>
<td>13</td>
<td>female</td>
<td>large</td>
<td>FALSE</td>
<td>0.45000</td>
<td>1.00</td>
</tr>
<tr>
<td>86</td>
<td>2</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.09000</td>
<td>1.00</td>
</tr>
<tr>
<td>239</td>
<td>5</td>
<td>male</td>
<td>large</td>
<td>TRUE</td>
<td>0.03000</td>
<td>1.00</td>
</tr>
<tr>
<td>511</td>
<td>11</td>
<td>female</td>
<td>small</td>
<td>TRUE</td>
<td>0.62000</td>
<td>1.00</td>
</tr>
</tbody>
</table>
```

... # Cut to reduce space
Two other operations that might be useful to prepare your data are `sort()` and `order()`

**Sorting data frames in decreasing order**

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

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>762</td>
<td>16</td>
<td>female</td>
<td>large</td>
<td>FALSE</td>
<td>0.92000</td>
<td>2.00</td>
</tr>
<tr>
<td>412</td>
<td>9</td>
<td>female</td>
<td>small</td>
<td>TRUE</td>
<td>0.48000</td>
<td>2.00</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.67000</td>
<td>2.00</td>
</tr>
<tr>
<td>155</td>
<td>4</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.38000</td>
<td>2.00</td>
</tr>
<tr>
<td>434</td>
<td>11</td>
<td>female</td>
<td>small</td>
<td>FALSE</td>
<td>0.49000</td>
<td>2.00</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

# Cut to reduce space
## Checking and cleaning data

### Large data frames

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>762</td>
<td>16</td>
<td>female</td>
<td>large</td>
<td>FALSE</td>
<td>0.92000</td>
<td>2.00</td>
</tr>
<tr>
<td>412</td>
<td>9</td>
<td>female</td>
<td>small</td>
<td>TRUE</td>
<td>0.48000</td>
<td>2.00</td>
</tr>
<tr>
<td>37</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.67000</td>
<td>2.00</td>
</tr>
<tr>
<td>155</td>
<td>4</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.38000</td>
<td>2.00</td>
</tr>
<tr>
<td>434</td>
<td>11</td>
<td>female</td>
<td>small</td>
<td>FALSE</td>
<td>0.49000</td>
<td>2.00</td>
</tr>
</tbody>
</table>

... # Cut to reduce space

[ reached getOption("max.print") -- omitted 626 rows ]
Reading and writing data

Checking and cleaning data

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

**Examples:**

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

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.17</td>
<td>1.66</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.87</td>
<td>1.26</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.22</td>
<td>1.43</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>male</td>
<td>small</td>
<td>FALSE</td>
<td>0.13</td>
<td>1.46</td>
</tr>
</tbody>
</table>
Reading and writing data

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

<table>
<thead>
<tr>
<th>Snail.ID</th>
<th>Sex</th>
<th>Size</th>
<th>Feeding</th>
<th>Distance</th>
<th>Depth</th>
<th>Temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>766</td>
<td>16</td>
<td>female</td>
<td>large</td>
<td>TRUE</td>
<td>0.65</td>
<td>1.71</td>
</tr>
<tr>
<td>767</td>
<td>16</td>
<td>female</td>
<td>large</td>
<td>TRUE</td>
<td>0.46</td>
<td>1.27</td>
</tr>
<tr>
<td>768</td>
<td>16</td>
<td>female</td>
<td>large</td>
<td>FALSE</td>
<td>0.36</td>
<td>1.28</td>
</tr>
<tr>
<td>769</td>
<td>16</td>
<td>female</td>
<td>large</td>
<td>FALSE</td>
<td>0.42</td>
<td>1.82</td>
</tr>
</tbody>
</table>
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
Reading and writing data

Export data

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

Check `?read.table` or `?read.csv`

```r
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:

```r
write.csv(Snail_data, 
          file = "Snail_data_checked.csv", 
          row.names = FALSE)  # object you want export
                            # file name saved
                            # 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")
```
Reading and writing data

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?

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write.table()</td>
<td>save a data frame as text file (.txt) or as comma separated format (.csv) into R</td>
</tr>
<tr>
<td>write.csv()</td>
<td></td>
</tr>
<tr>
<td>save()</td>
<td>writes an external representation of R objects</td>
</tr>
<tr>
<td>rm()</td>
<td>removes specific or all objects from working environment</td>
</tr>
<tr>
<td>rm(list=ls())</td>
<td></td>
</tr>
</tbody>
</table>