

Introduction to R: Session 01

Holger Sennhenn-Reulen^a, Nordwestdeutsche Forstliche Versuchsanstalt (NW-FVA)

May 16, 2024 (Version 0.6)

Contents

1	(Int)R(o)	3
1.1	What is R?	3
1.2	Why useR?	4
1.3	Literature	6
1.4	R and RStudio	6
1.5	R is a language	6
1.6	R is an environment for statistical computing	6
1.7	A short history of R	6
2	Basic vector types	7
3	Basic mathematical commands	7
3.1	Exercises	8
4	Symbols and values	9
4.1	Exercises	9
5	Mathematical functions	10
5.1	Exercises	10
6	Functions basics	11
7	Documentation	12
7.1	Exercises	12
8	Objects	13
8.1	Objekt names	13
8.2	Save and load	13
8.3	Litter service	14
9	Dataobject classes	15
9.1	Vector	15
9.2	Calculations	15
9.3	Matrix	16
9.4	List	16
9.5	Synthesis and indexing	17
9.6	Dataframe	17
9.7	Functions on dataframes	18
10	Functions for character strings	19
	nchar	19
	tolower und toupper	19
	gsub	19

^aPrivate webpage: uncertaintree.github.io

substring	19
strsplit	20
paste	20
11 Factors	21
11.1 Exercises	22

All contents are licensed under CC BY-NC-ND 4.0 ([Link](#)).

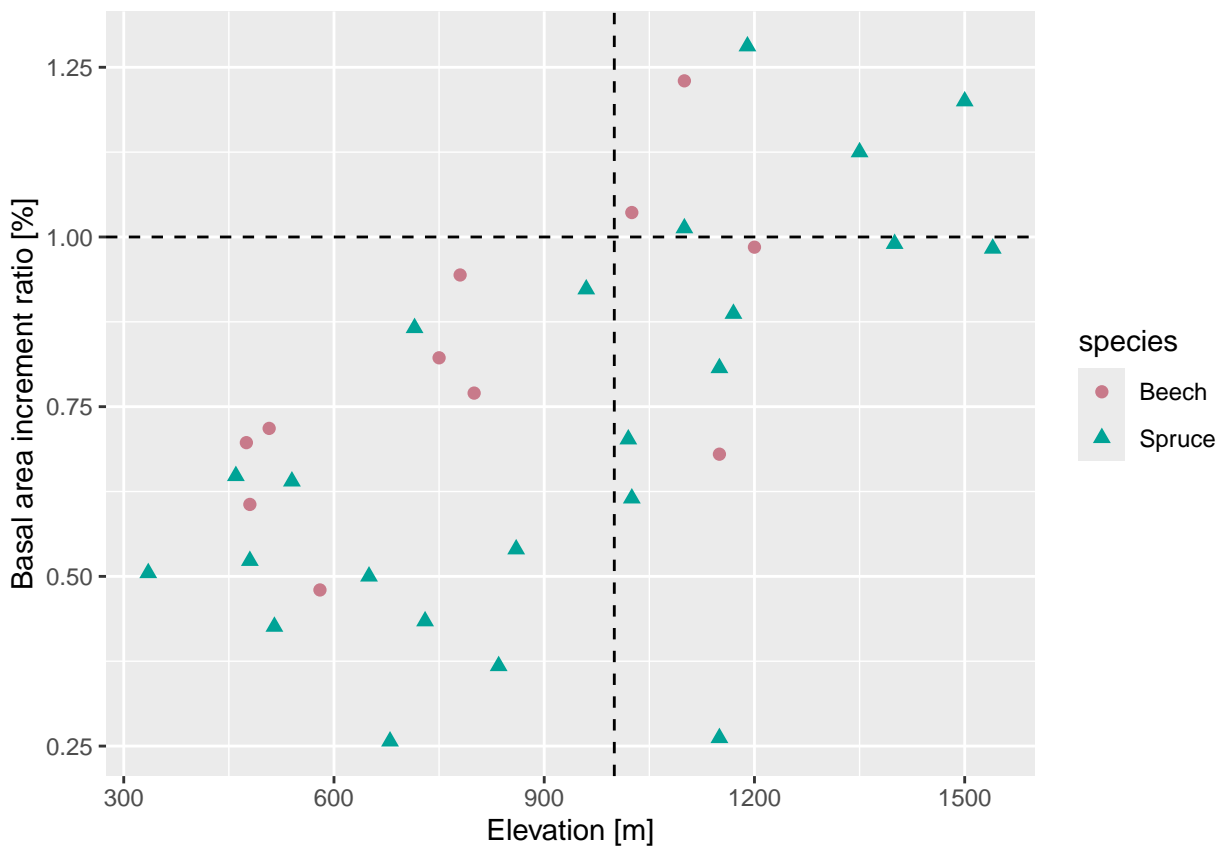
1 (Int)R(o)

1.1 What is R?

'A Language for Data Analysis and Graphics'

A first example: **Scatter plot, data-management and descriptive analysis**

```
## A scatter-plot ...
library("ggplot2")
library("colorspace")
ggplot(data = drought,
       aes(x = elev, y = bair, color = species, shape = species)) +
  geom_point(size = 2) +
  geom_vline(xintercept = 1000, linetype = "dashed") +
  geom_hline(yintercept = 1, linetype = "dashed") +
  scale_color_discrete_qualitative(pal = "Dark") +
  labs(x = "Elevation [m]", y = "Basal area increment ratio [%]")
```



```
## ... , some datamanagement, ...
range(drought$elev)

## [1] 335 1540

(tmp <- range(drought$elev %/% 250))

## [1] 1 6

br <- seq(tmp[1] * 250, (tmp[2] + 1) * 250, by = 250)
(drought$elev_cut <- cut(drought$elev, breaks = br))
```

```
## [1] (250,500]          (250,500]          (250,500]          (500,750]
## [5] (500,750]          (500,750]          (500,750]          (500,750]
## [9] (500,750]          (750,1e+03]        (750,1e+03]        (750,1e+03]
## [13] (1e+03,1.25e+03]    (1e+03,1.25e+03]    (1e+03,1.25e+03]    (1e+03,1.25e+03]
## [17] (1e+03,1.25e+03]    (1e+03,1.25e+03]    (1e+03,1.25e+03]    (1.25e+03,1.5e+03]
## [21] (1.25e+03,1.5e+03]  (1.25e+03,1.5e+03]  (1.5e+03,1.75e+03]  (250,500]
## [25] (250,500]          (500,750]          (500,750]          (500,750]
## [29] (750,1e+03]        (750,1e+03]        (1e+03,1.25e+03]    (1e+03,1.25e+03]
## [33] (1e+03,1.25e+03]    (1e+03,1.25e+03]
## 6 Levels: (250,500] (500,750] (750,1e+03] ... (1.5e+03,1.75e+03]

## ... and then some descriptive statistics:
library("plyr")
ddply(.data = drought, .variables = c("species", "elev_cut"),
      summarize, .drop = F,
      n = length(bair),
      mean_bair = mean(bair))

##   species      elev_cut n mean_bair
## 1   Beech   (250,500] 2 0.6515000
## 2   Beech   (500,750] 3 0.6733333
## 3   Beech   (750,1e+03] 2 0.8570000
## 4   Beech (1e+03,1.25e+03] 4 0.9827500
## 5   Beech (1.25e+03,1.5e+03] 0      NaN
## 6   Beech (1.5e+03,1.75e+03] 0      NaN
## 7  Spruce   (250,500] 3 0.5586667
## 8  Spruce   (500,750] 6 0.5205000
## 9  Spruce   (750,1e+03] 3 0.6103333
## 10 Spruce (1e+03,1.25e+03] 7 0.7952857
## 11 Spruce (1.25e+03,1.5e+03] 3 1.1050000
## 12 Spruce (1.5e+03,1.75e+03] 1 0.9830000
```

1.2 Why useR?

Because of packages such as mgcv! ... and because we can organize our whole working-with-data-process reproducibly (markdown!) in one place!

(Note: the following figure is much too overloaded!)

```
## library("mgcv")
m <- mgcv::gam(bud_burst_days_since_may1st ~ te(year), data = frost)
nd <- data.frame(year = seq(min(frost$year), max(frost$year), by = .5))
tmp <- mgcv::gam.mh(m, thin = 5, ns = 2000, rw.scale = 2)$bs
coda::effectiveSize(coda::as.mcmc(tmp))
tmp <- predict(m, newdata = nd, type = "lpmatrix") %*% t(tmp)
tmp <- apply(X = tmp, MAR = 1, FUN = quantile,
             probs = seq(.01, .99, by = .01))
tmp <- data.frame(q = as.numeric(tmp),
                  year = rep(nd$year, each = nrow(tmp)),
                  p = rep(seq(.01, .99, by = .01), ncol(tmp)))
library("ggrepel")
ggplot(data = tmp, aes(x = year, y = q)) +
  geom_line(aes(group = p, color = p)) +
  scale_color_continuous_sequential(pal = "Light Grays") +
  geom_point(data = frost, aes(x = year, y = bud_burst_days_since_may1st)) +
  geom_text_repel(data = frost, aes(x = year, y = bud_burst_days_since_may1st,
                                   label = end_1st_dev_stage - bud_burst),
                 min.segment.length = 0, box.padding = .5) +
  labs(x = "Year", y = "Bud burst [Days since May, 1st]",
```

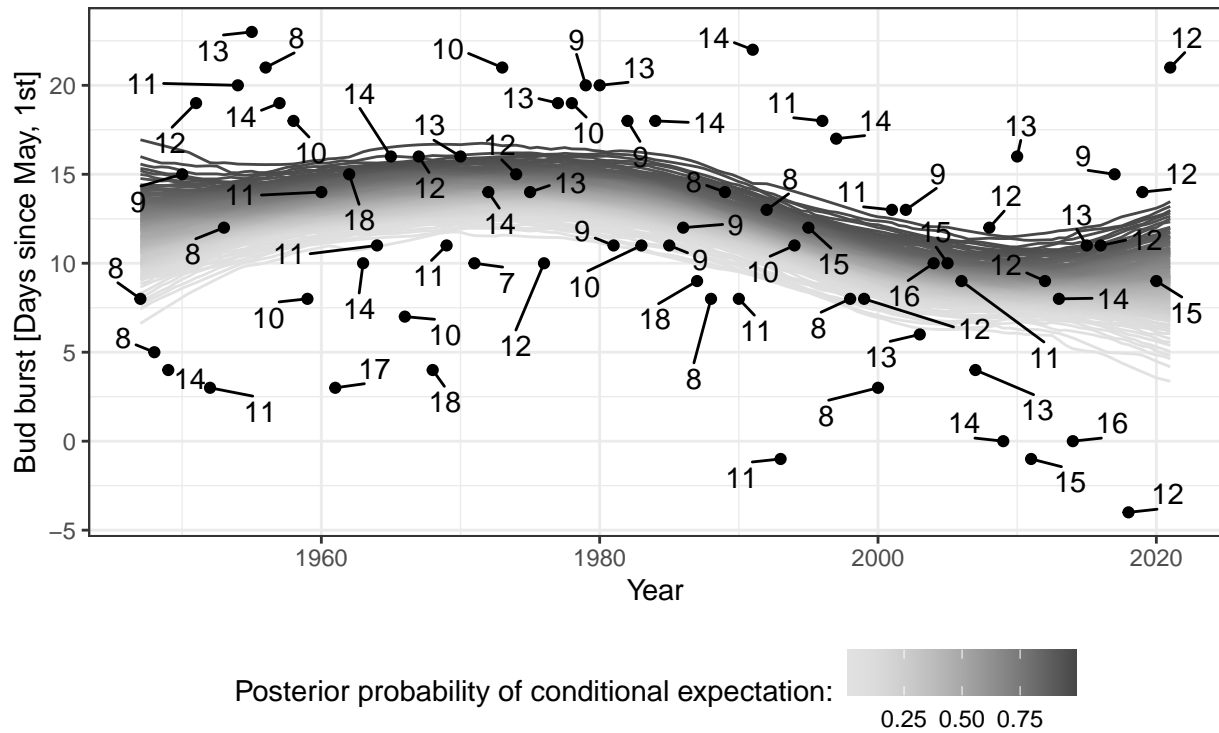
```

title = "At least one frost event during 1st dev. stage",
subtitle = "Labels give time [days] between bud burst and end of 1st development stage",
color = "Posterior probability of conditional expectation:" +
theme_bw() +
theme(legend.position = "bottom")

```

At least one frost event during 1st dev. stage

Labels give time [days] between bud burst and end of 1st development stage



1.3 Literature

1.3.1 Books...

- Everitt, Hothorn (2006): *A Handbook of Statistical Analyses using R*. Chapman and Hall.
- Ligges (2008): *Programmieren mit R*. Springer Verlag.
- Venables, Smith (2002): *An introduction to R*. Network Theory Verlag.
- Verzani (2005): *Using R for Introductory Statistics*. Chapman and Hall.
- Wickham (2016): *Advanced R*.

1.3.2 ... and the wide web:

- education.rstudio.com
- cran.r-project.org manual R-intro
- cran.r-project.org manual R-lang

1.4 R and RStudio

R:

- <https://r-project.org>
- Freely available
- Supported by all major operating system: Windows, Linux/Unix, Mac OS, ...

RStudio and others editors for working with R:

- **RStudio** for Windows, Mac OS and Linux:
- RStudio website
- RStudio cheat sheet
- Previously, I presented a list of alternatives to RStudio, but in 2024, that doesn't seem suitable anymore!?

1.5 R is a language

- 'To understand computations in R, two slogans are helpful: Everything that exists is an object. Everything that happens is a function call.' (John M. Chambers)
- Interpreted language: **R** interprets and evaluates your function calls without a compilation step.
- Simple syntax with clear similarities to mathematical notation.

1.6 R is an environment for statistical computing

- Rich toolbox for doing (graphical) statistics with an ever growing list of ever improving 'add-on' packages.
- R is freely available for anyone.
- R code is the product: it is transparent and allows you to reproduce any single step of your analysis.

1.7 A short history of R

- 1976: Development of programming language S at Bell Laboratories
- 1988: Software S-PLUS released (for purchase)
- 1992: Ross Ihaka and Robert Gentleman start project R
- 1995: R available for free under GPL
- 1998: Comprehensive R archive network (CRAN) is founded
- 2000: First 'complete' version R-1.0.0 released
- 2004: First conference on R (useR!) takes place
- 2021: Current version is **R-4.4.0** (May 2024)

2 Basic vector types

In R, there is no such thing as a single number, since in **R**, a single number is just a vector of length 1:

```
length(3)
```

```
## [1] 1
```

R works on six basic vector types, of which four are used in our daily work as empirical researchers:

- `logical` (`TRUE` and `FALSE`),
- `integer` (numbers we know as ..., -1, 0, 1, 2, ...; explicitly created in **R** with suffix `L`),
- `numeric` (extending integers to 'any' value such as 1/3), and
- `character` (`'R'`, `'Beech'`, ...).

```
class(TRUE)
```

```
## [1] "logical"
```

```
class(-1L)
```

```
## [1] "integer"
```

```
class(1/3)
```

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

```
class("R")
```

```
## [1] "character"
```

Again, the difference between integer and numeric (*real number*) is not important for our applied work, so just think of `integer` and `numeric` as classes that represent numerical values in a computer. The following section introduces some manipulations we can do with them, `logical` and `character` will be treated later.

3 Basic mathematical commands

Mathematical operation	Command
Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	^
Rest of integer division (Modulo)	%%
Integer division	%/%
Brackets	()

3.1 Exercises

Run the following lines.

Note: First and third line define objects `a` and `b` by using an indexing command on data-object `drought`. You don't need to fully understand what this does at this point of the course, just think that objects `a` and `b` each represent a single number with some applied meaning (Basal area increment ratio at 1st, and 2nd plot, respectively).

```
a <- drought$bair[1] ## Basal area increment ratio at 1st plot
a ## ... growth is about a half in 2003 in comparison to growth in 2002
b <- drought$bair[2] ## Basal area increment ratio at 2nd plot
b ## ... growth is about two thirds in 2003 in comparison to growth in 2002
a - b ## Whats the difference of bair?
a / b ## Whats the ratio of bair?
b %/% a ## How often is 'b' contained in 'a'?
b %% a ## If 'b' is contained once in 'a', what's remaining?
(a + b)/2 ## Arithmetic mean of 'a' and 'b'
```

Did we just calculate an arithmetic mean 'by hand'?

```
mean(c(a, b))
```

```
## [1] 0.5765
```


4 Symbols and values

Objective	Call
Decimal sign	.
List and separate objects, arguments, ...	,
Several R -calls in one line (not recommended)	;
'from-to' operator for integer sequences	:
Comments and help	#, ?
Number π	pi
A concept called infinity	Inf
Base 10 exponential notation (eg. $10^{-3} = 0.001$)	1e-3
Integer value	L
Empty object	NULL
Missing value (<i>not available</i>)	NA
Non-defined value (<i>not a number</i>)	NaN
Comparison	>, >=, ==, !=, <=, <
Boolean	TRUE, T, FALSE, F
Negation (<i>not</i>)	!
Conjunction (<i>and</i>)	&
Disjunction (<i>or</i>)	

Remember, everything in **R** is an object or a function, so we even can't use , without a function call such as:

```
c(a, b) ## first two bair values in a vector
```

```
## [1] 0.505 0.648
```

4.1 Exercises

Run the following lines and ask yourself *What is the applied question behind each line?*.

```
pi ## Okay, there is no applied meaning here, it's just a number, an important one, though :)
a / b == pi
a > b
a < b
a == a & b != 2/3
a > b | b < 2/3
a / 0
0 / 0 ## Let:  $x = 0 / 0$ . So:  $0 * x = 0$ , which is true for any number  $x$ .
```

5 Mathematical functions

Mathematical function	Call
Exponential function with basis e	<code>exp()</code>
Natural logarithm	<code>log()</code>
Square root	<code>sqrt()</code>
Absolute value	<code>abs()</code>
Trigonometric functions	<code>sin()</code> , <code>cos()</code> , <code>tan()</code>
Sum and product	<code>sum()</code> , <code>prod()</code>
Round (up and down)	<code>round()</code> (<code>floor()</code> , <code>ceiling()</code>)
Maximum and minimum value	<code>max()</code> , <code>min()</code>
Factorial $n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$	<code>factorial()</code>
Binomial coefficient $\binom{n}{k}$	<code>choose()</code>

... for further functions, see `?Special` and `?groupGeneric`.

Also note: There are only countable many numbers that a computer is able to represent. Therefore, computers can also only represent some real numbers, and as a consequence, 'rounding' is more complicated than we might think! Here are some explanations why R results in 0 2 2 4 4 6 6 instead of 0 1 2 3 4 5 6 when it does `round(seq(0.5, 6.5))`. Many thanks to Jan Schick for pointing me towards this 'rounding' standard in R, as well as providing me this nice example. Session 05 will also briefly touch upon R and computers and the 'density' of real numbers.

5.1 Exercises

Run the following lines.

```
a <- drought$bair[1]; a; b <- drought$bair[2]; b ## Basal are increment ratio values of first two st
exp(x = a)
sqrt(x = a)
a - b
abs(x = a - b)
sum(a, b)
max(a, b)
```

Computers construct real values!

```
sin(x = pi)
```

```
## [1] 1.224647e-16
```

```
round(sin(pi), digits = 5)
```

```
## [1] 0
```

```
cos(x = pi)
```

```
## [1] -1
```

6 Functions basics

(We will have a more detailed focus on functions in the third session.)

Basic properties of functions in R:

- Function call of function `f` using brackets: `f()`
- Documentation (*'help-page'*) using preceded question mark (`?f`), or calling `help(f)`
- Search for documentation of `f` by: `help.search('f')`
- Syntax:

```
f(arg1 = value1, arg2 = value2, ...) ## '...' is for optional further arguments.
```

- Arguments often have default objects, eg. for `pch` in `plot()`:

```
plot(x = a, y = b)
```

```
plot(x = a, y = b, pch = 1) ## same!
```

- Providing argument names is not optional in any case:

```
sin(pi/2)
```

```
## [1] 1
```

```
sin(x = pi/2) ## Same
```

```
## [1] 1
```

```
plot(pi, pi, 1) ## Error!
```

```
## Error in plot.xy(xy, type, ...): unzulässiger Plottyp '1'
```

7 Documentation

Structure:

- **Description:** In brief, what is this function about?
- **Usage:** How to call the function? What are mandatory arguments?
- **Arguments:** Explaining each argument
- **Details:** Some text about implementation, scope, ...
- **Value:** Explaining the resulting object
- **Authors** and **References** and **See also** and ...
- **Examples:** Always at the bottom, always scroll down there, you will never be disappointed!

7.1 Exercises

Go to the documentation for `lm`.

- What is this function about?
- Copy-paste and run the example.

8 Objects

Basic Conventions:

- Objects store values, results or algorithms, ie. functions.
- Assignment of contents by `<-` or `=` (or very seldomly `->`).
- Type of contents is described by object classes.

```
(a <- drought$bair[1]); (a = drought$bair[1])
```

```
## [1] 0.505
```

```
## [1] 0.505
```

```
log(b <- drought$bair[2]); b
```

```
## [1] -0.4338646
```

```
## [1] 0.648
```

```
(a <- b)
```

```
## [1] 0.648
```

8.1 Objekt names

There are few hard technical restrictions on how to name objects in **R**, and a few soft rules that make life much simpler:

- Object names are required to begin with a lower or upper case letter, no numbers or any other signs are allowed.
- For any trailing character, numbers and some signs (restrict yourself to underscore `_` and `.`) are allowed
- Don't use `?`, `$`, `%`, `^`, `&`, `*`, `(`, `)`, `-`, `#`, `?`, `,`, `<`, `>`, `/`, `|`, `\`, `[`, `]`, `{`, and `@`
- **As short as possible, as long as needed!**
- This trade-off is simple, but (hopefully) still very useful: long object names mean more typing (RStudio weakens this point by auto-complete), but leave less room for questions on content (*What was a again?*).
- So try to give 'telling names', ie. contents (and units!) should be clearly visible from the object's name: `first_bair_measurement` is better than `a`, `elev_m` is better than `elev`.
- Ask yourself: Will I be able to immediately recall the content of an object by the object name, even after two weeks no work on this project?

We get an error message:

```
3values <- c(1, 2, 4)
```

```
## Error: <text>:1:2: unerwartetes Symbol
## 1: 3values
##      ^
```

```
three_values <- c(1, 2, 4)
```

8.2 Save and load

```
save(frost, file = "frost_data.RData")
load(file = "frost_data.RData")
```

Where will my files be stored? See section *Preparatory Work in R* in Session 04.

8.3 Litter service

- Names of objects in current session: `ls()`
- Litter service using `rm()`
- `dput()` comes as a helper

```
ls()
```

```
## [1] "a"          "b"          "br"         "df"         "drought"
## [6] "frost"      "m"          "nd"         "three_values" "tmp"
```

```
dput(ls())
```

```
## c("a", "b", "br", "df", "drought", "frost", "m", "nd", "three_values",
## "tmp")
```

```
dput(ls()[!(ls() %in% c("drought", "frost", "df"))])
```

```
## c("a", "b", "br", "m", "nd", "three_values", "tmp")
```

```
rm("a", "b", "br", "m", "nd", "three_values", "tmp")
ls()
```

```
## [1] "df"          "drought" "frost"
```

9 Dataobject classes

9.1 Vector

A vector is a one-dimensional combination of length 1 vectors of the same basic vector type.

9.1.1 Basics

Objective	Call
Combination of elements	<code>c(A, B)</code>
Indexing	<code>c(A, B)[1]</code>
Length of vector	<code>length(x)</code>
Integer sequence	<code>A:B</code>
Flexible sequence	<code>seq(A, B, length = N), seq(A, B, by = K)</code>
Repeat	<code>rep(A, times = N), rep(c(A, B), each = K)</code>
Sort elements	<code>sort(c(A, B), decreasing = FALSE)</code>
Order of elements	<code>order(c(A, B))</code>

9.1.2 Exercises

Copy-paste and run each line: describe what it does and what it results in.

```
a <- drought$elev[1]; b <- drought$elev[2]
(x <- c(a, b))
c(x, a)
c(x, x)
x[2] <- drought$elev[3]; x
length(x)
a:b
seq(a, b, by = .5)
seq(a, b, by = 5)
seq(a, b, length = 10) ## Sequenz
rep(x, times = 3)
rep(x, each = 3)
rep(x, length = 7)
sort(x, decreasing = TRUE)
order(c(x, 100))
a <- as.character(drought$species[1]); b <- as.character(drought$species[2])
(x <- c(a, b))
c(x, "Beech")
c(x, 2)
```

9.2 Calculations

- Basic calculations operate independently on all elements
- Function calls work (usually) with vectorized arguments
- Summary of the content: `summary()`
- Frequency table: `table()` (see Session 04 for much more detail)

9.2.1 Exercises

Copy-paste and run each line: describe what it does and what it results in.

```
x <- drought$bair
x^2
sin(x)
length(x)
summary(x)
boxplot(x)
stripchart(x, add = T, pch = 16, method = "jitter", vertical = T)

species <- drought$species
xtabs(~ species)
xtabs(~ species + (drought$elev > 1000))
```

9.3 Matrix

A vector is a two-dimensional combination of single-element objects of the same class.

9.3.1 Basics

P×P matrix with content content:

```
A <- matrix(nrow = P, ncol = Q, data = content)
```

Fill matrix column- (default) or line-wise using argument byrow = TRUE:

- Indexing of one element: `A[1, 1]`
- Indexing of first row (result is vector): `A[1,]`
- Indexing of several rows (result is matrix): `A[1:3,]`
- Indexing of first column (result is vector): `A[, 1]`
- Indexing of several columns (result is matrix): `A[, 1:3]`
- Indexing of several rows and columns (result is matrix): `A[1:3, 1:3]`

9.3.2 Exercises

```
a <- drought$elev
b <- drought$bair
length(a) == length(b)
A <- matrix(nrow = length(a), ncol = 2, data = c(a, b))
all(A[, 1] == a)
A[1:3, ]
A[1:3, 2]
A[c(1:3, 29), ]
B <- matrix(nrow = 3, ncol = 3, data = 1:9); B
diag(B)
B %*% B
B %*% c(1, 1, 1)
rowSums(B)
c(1, 1, 1) %*% B
colSums(B)
```

9.4 List

A list is a general 'container' object.

9.4.1 Basics

- Lists may contain elements storing objects of varying classes, lengths, (character, numeric, integer, factor, ...)
- Construct a list using `list(...)`
- Indexing: `x[[1]]` or `x[['name']]` or `x$name`
- `str` gives the structure
- For lists of 'consistent' classes (or generic functions):

```
lapply(X = mylist, FUN = myfun)
```

9.4.2 Exercises

```
beech <- list(species = "Fagus",
             n = sum(drought$species == "Beech"),
             data = subset(drought, species == "Beech"))
beech
beech[[1]]
beech[['species']]
beech$species
str(beech)
lapply(beech, FUN = dim)
lapply(beech, FUN = summary)
```

9.5 Synthesis and indexing

- We may provide names for elements of vectors, matrices and lists:
- `dimnames(A)`, `rownames(A)`, `colnames(A)` for matrices
- Select a named element by using the `$` sign
- Select elements by `[[]]` (for lists), `[]` (for vectors), and `[,]` (for matrices).

9.5.1 Exercises

```
c(one = 1, two = 2, four = 4)
x <- c(1, 2, 4)
x
names(x) <- c("one", "two", "four")
x
names(beech)
A <- matrix(nrow = 10, ncol = 2, data = c(drought$elev[1:10], drought$bair[1:10]))
colnames(A)
colnames(A) <- c("elev", "bair")
colnames(A)
rownames(A)
names(A)
dimnames(A)
```

9.6 Dataframe

Based on their object properties, dataframes can be classified between matrices and lists, whereby the columns (the so-called 'variables') of the dataframe correspond to the elements of a list:

- Dataframes are more rigid than lists because all columns must have the same length,
- Dataframes are more flexible than matrices, since all columns can contain different contents (numbers, strings, ...).
- Dataframes are generated with `data.frame()`
- Indexing:

- Rows and columns can be indexed in the same way as the elements of a matrix,
- Columns can be indexed in the same way as the elements of a list (we have already used this by `drought$elev`).

9.6.1 Exercises

```
weather <- data.frame(day = c("Monday", "Tuesday", "Wednesday"),
                      daily_mean_temperature_C = c(12, 14, 11),
                      precipitation_sum_mm = c(5, 9, 25),
                      site = "Goettingen")

weather
weather[1:2, ]
weather$day[1:2]
weather[["daily_mean_temperature_C"]][3]
weather[1:2, c("daily_mean_temperature_C", "precipitation_sum_mm")]
weather[1:2, c("day", "precipitation_sum_mm")]
```

9.7 Functions on dataframes

- `summary`: Summary of each of the variables in the dataframe
- `str`: overview of the structure of the dataframe
- `dim`: Dimension of the dataframe (number of rows and columns)
- The first `N` lines of a dataframe `df` are extracted with `head(df, n = N)`, and
- the last `N` lines of a dataframe `df` are extracted with `tail (d, n = N)`.

9.7.1 Exercises

```
summary(weather)
str(weather)
head(weather, n = 2)
dim(weather)
```

10 Functions for character strings

If we want to change character strings (such as variable names of a dataframe), the following functions can be helpful:

nchar

Returns the number of characters of a string:

```
nchar(names(weather))
```

```
## [1] 3 24 20 4
```

tolower und toupper

Replaces uppercase with lowercase letters (and vice versa).

```
toupper(c("Beech", "BEech", "beech"))
```

```
## [1] "BEECH" "BEECH" "BEECH"
```

```
tolower(c("Beech", "BEech", "beech"))
```

```
## [1] "beech" "beech" "beech"
```

gsub

Replace a pattern:

```
names(weather)
```

```
## [1] "day" "daily_mean_temperature_C"
## [3] "precipitation_sum_mm" "site"
```

```
names(weather) <- gsub(names(weather), pattern = "_mm", replacement = "__mm", fixed = T)
names(weather) <- gsub(names(weather), pattern = "_C", replacement = "__C", fixed = T)
names(weather)
```

```
## [1] "day" "daily_mean_temperature__C"
## [3] "precipitation_sum__mm" "site"
```

substring

Substrings from first to last.

```
substring(names(weather), first = 1, last = 5)
```

```
## [1] "day" "daily" "preci" "site"
```

strsplit

Splits strings according to a certain pattern (strsplit always returns a list).

```
names(weather) <- gsub(names(weather), pattern = "__", replacement = "_", fixed = T)
(tmp <- strsplit(names(weather), split = "_", fixed = T))
```

```
## [[1]]
## [1] "day"
##
## [[2]]
## [1] "daily"          "mean"          "temperature" "C"
##
## [[3]]
## [1] "precipitation" "sum"          "mm"
##
## [[4]]
## [1] "site"
```

paste

Merges strings (arguments sep or collapse).

```
sapply(tmp, FUN = function(x){paste(x, collapse = "_")})
```

```
## [1] "day"          "daily_mean_temperature_C"
## [3] "precipitation_sum_mm"  "site"
```

11 Factors

For the analysis of qualitative characteristics, it makes sense to represent variables with character strings as **factors**.

A factor comes with three ingredients:

- data vector `x`
- levels
- labels

We can generate a factor without specification of levels and labels: In this case, factor levels (`levels`) are generated (in alphabetical order), numbered internally (see this numeric expression using `as.numeric`), and represented externally by the original values of the character string.

```
(tmp <- factor(x = c("Beech", "Spruce", "Beech")))
```

```
## [1] Beech Spruce Beech
## Levels: Beech Spruce
```

```
attributes(tmp)
```

```
## $levels
## [1] "Beech" "Spruce"
##
## $class
## [1] "factor"
```

```
levels(tmp)
```

```
## [1] "Beech" "Spruce"
```

```
as.numeric(tmp)
```

```
## [1] 1 2 1
```

```
as.character(tmp)
```

```
## [1] "Beech" "Spruce" "Beech"
```

We can provide levels, to which the numerical representation will refer to:

```
(tmp <- factor(x = c("Beech", "Spruce", "Beech"),
               levels = c("Spruce", "Beech")))
```

```
## [1] Beech Spruce Beech
## Levels: Spruce Beech
```

```
attributes(tmp)
```

```
## $levels
## [1] "Spruce" "Beech"
##
## $class
## [1] "factor"
```

```
levels(tmp)
```

```
## [1] "Spruce" "Beech"
```

```
as.numeric(tmp)
```

```
## [1] 2 1 2
```

```
as.character(tmp)
```

```
## [1] "Beech" "Spruce" "Beech"
```

Using labels, the values represented externally are the result of the mapping from levels to labels

```
(tmp <- factor(x = c("Beech", "Spruce", "Beech"),
               levels = c("Beech", "Spruce"),
               labels = c("Fagus sylvatica", "Picea abies")))
```

```
## [1] Fagus sylvatica Picea abies      Fagus sylvatica
```

```
## Levels: Fagus sylvatica Picea abies
```

```
attributes(tmp)
```

```
## $levels
```

```
## [1] "Fagus sylvatica" "Picea abies"
```

```
##
```

```
## $class
```

```
## [1] "factor"
```

```
levels(tmp)
```

```
## [1] "Fagus sylvatica" "Picea abies"
```

```
as.numeric(tmp)
```

```
## [1] 1 2 1
```

```
as.character(tmp)
```

```
## [1] "Fagus sylvatica" "Picea abies"      "Fagus sylvatica"
```

11.1 Exercises

```
tmp <- factor(x = c("Beech", "Spruce", "Beech", "Oak"),
              levels = c("Beech", "Spruce"),
              labels = c("Fagus sylvatica", "Picea abies"))
attributes(tmp)
levels(tmp)
as.numeric(tmp)
as.character(tmp)
tmp <- factor(x = c("Beech", "Spruce", "Beech", "Oak"),
              levels = c("Beech", "Oak", "Spruce"),
              labels = c("Deciduous", "Deciduous", "Conifer"))
attributes(tmp)
levels(tmp)
as.numeric(tmp)
as.character(tmp)
```