Introduction to R: Session 01

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

Oktober 27, 2021 (Version 0.2)

Contents

[a] Private webpage: uncertaintree.github.io

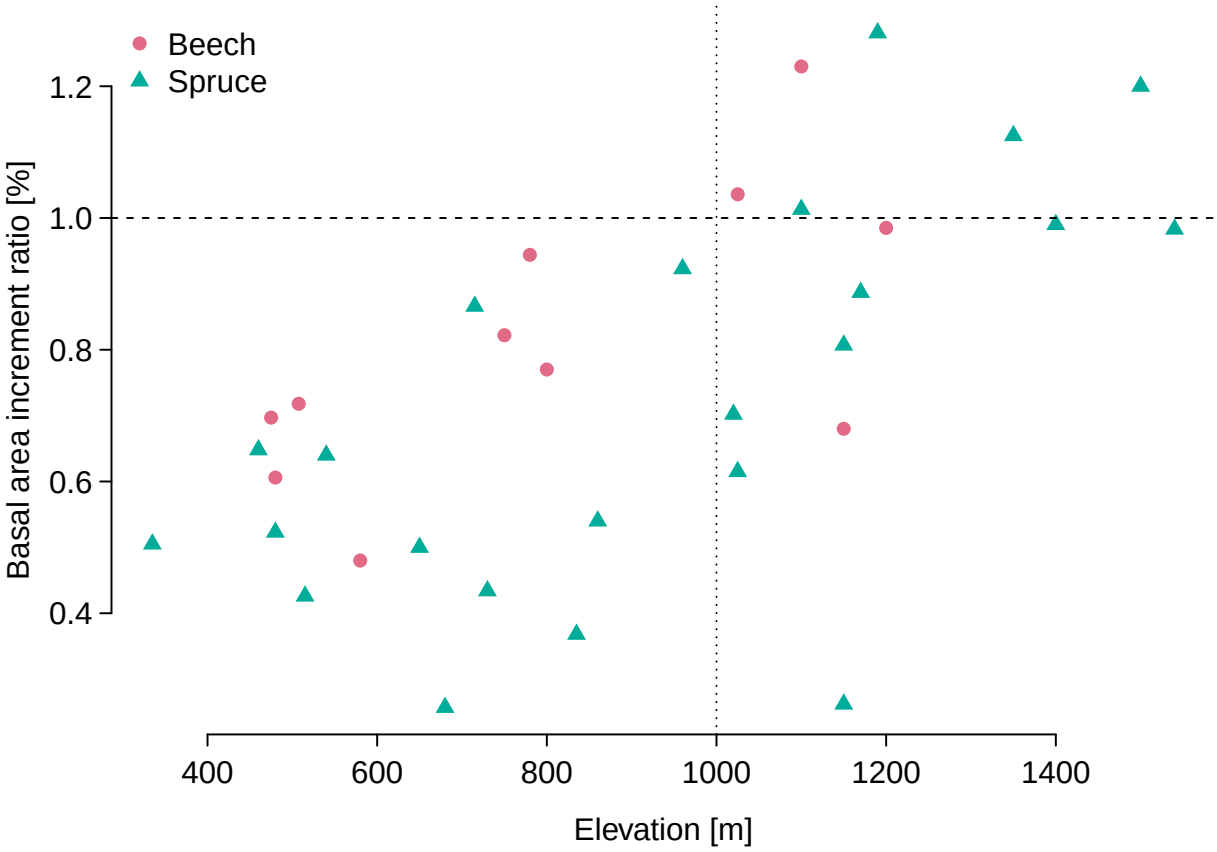
1 (Int)R(o)

1.1 What is R?

'A Language for Data Analysis and Graphics'

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

```
## A scatter-plot:
paint <- colorspace::qualitative_hcl(n = 2)
par(mar = c(3, 3, .1, .1), mgp = c(2, .5, 0), tcl = -.3)
plot(drought$elev, drought$bair, las = 1, bty = "n",
     xlab = "Elevation [m]", ylab = "Basal area increment ratio [%]",
     pch = c(16, 17)[1 + (drought$species == "Spruce")],
     col = paint[1 + (drought$species == "Spruce")])
abline(h = 1, lty = 2)
abline(v = 1000, lty = 3)
legend("topleft", pch = c(16, 17), col = paint, legend = c("Beech", "Spruce"),
       bg = "white", bty = "n")
```




```
## ... some data-management:
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 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
```

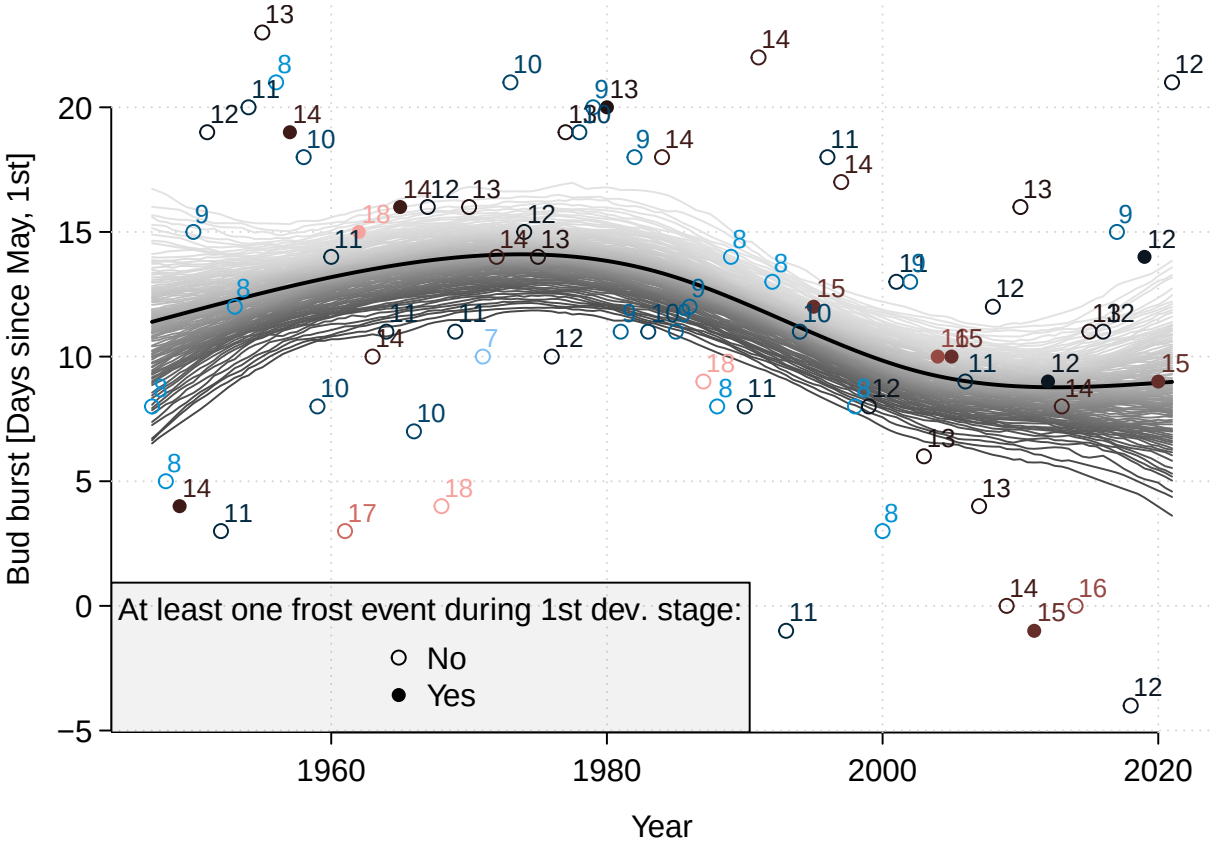
1.2 Why useR?

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!)

```
par(mar = c(3, 3, .1, .1), mgp = c(2, .5, 0), tcl = -.3)
plot(frost$year, frost$bud_burst_days_since_may1st, type = "n",
     xlab = "Year", ylab = "Bud burst [Days since May, 1st]",
     las = 1, bty = "n")
grid()
legend("bottomleft", pch = c(1, 16), legend = c("No", "Yes"),
       title = "At least one frost event during 1st dev. stage:",
       bg = rgb(.5, .5, .5, alpha = .1))
## 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))
br <- mgcv::gam.mh(m, thin = 5, ns = 2000, rw.scale = 2)$bs
coda::effectiveSize(coda::as.mcmc(br))
Mu <- predict(m, newdata = nd, type = "lpmatrix") %*% t(br)
Mu_q <- apply(X = Mu, MAR = 1 , FUN = quantile,
              probs = seq(.01, .99, by = .01))
paint <- colorspace::sequential_hcl(n = nrow(Mu_q), pal = "Light Grays")
for (i in 1:nrow(Mu_q)) {
  lines(nd$year, Mu_q[i, ], col = paint[i])
}
lines(nd$year, predict(m, newdata = nd), lwd = 2)
tmp <- as.numeric(frost$end_1st_dev_stage - frost$bud_burst)
tmp <- tmp - min(tmp) + 1
paint <- colorspace::diverging_hcl(n = max(tmp), pal = "Berlin")
points(frost$year, frost$bud_burst_days_since_may1st, col = paint[tmp],
       pch = c(1, 16)[1 + (frost$n_frost > .5)])
```

```
text(frost$year, frost$bud_burst - frost$may1st,
     labels = frost$end_1st_dev_stage - frost$bud_burst,
     adj = c(-.1, -.5), cex = .8, col = paint[tmp])
```



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 2021, 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-4.1.2* (November 2021)

2 Basic mathematical commands

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

2.1 Exercise

Run the following lines.

```
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
```

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

```
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
```

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

```
a - b ## Whats the difference of bair?
```

```
## [1] -0.143
```

```
a / b ## Whats the ratio of bair?
```

```
## [1] 0.779321
```

```
b %/% a ## How often is 'b' contained in 'a'?
```

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

```
b %% a ## If 'b' is contained once in 'a', what's remaining?
```

```
## [1] 0.143
```

```
(a + b)/2 ## Arithmetic mean of 'a' and 'b'
```

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

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

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

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

3 Symbols and values

Objective	Call
Decimal sign	`.`
List and seperate 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 (*not available*)	`NA`
Non-defined value (*not a number*)	`NaN`
Comparison	`>`, `>=`, `==`, `!=`, `<=`, `<`
Boolean	`TRUE`, `T`, `FALSE`, `F`
Negation (*not*)	`!`
Conjunction (*and*)	`&`
Disjunction (*or*)	`|`

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

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

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

3.1 Exercise

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.
```

4 Mathematical functions

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

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

4.1 Exercise

Run the following lines.

```
a <- drought$bair[1]; a; b <- drought$bair[2]; b ## Hoehen erster und zweiter Baum
exp(x = a)
sqrt(x = a)
a - b
abs(x = a - b)
```

(Computers construct real values)[https://en.wikipedia.org/wiki/Floating-point_arithmetic]:

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

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

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

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

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

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

```
sum(a, b)
max(a, b)
```

5 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 = wert1, arg2 = wert2, ...) ## '...' 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!
```

- Giving objects to arguments 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, ...): ungültiger Plottyp
```

6 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!

6.1 Exercises

Go to the documentation for `lm`.

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

7 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.

```
names(drought)
```

```
## [1] "bair"     "elev"     "species"  "elev_cut"
```

```
#rm(spati2)
#(a <- d$h[1]); (a = d$h[1])
#log(b <- d$h[2]); b
#(a <- b)
```

7.1 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 requared to begin with a lower or upper case letter, no numbers or any other signs are allowed.
- As a second or 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 remember the contents from the name after two weeks without working 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)
```

7.2 Save and load

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

7.3 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"        "i"            "m"            "Mu"           "Mu_q"
## [11] "nd"           "paint"        "three_values" "tmp"
```

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

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

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

```
## c("br", "df", "i", "m", "Mu", "Mu_q", "nd", "paint", "three_values",
## "tmp")
```

```
rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", "Mu",
   "Mu_q", "nd", "paint", "quantile_sequence", "species", "three_values", "tmp")
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'bair' nicht gefunden
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'd' nicht gefunden
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'd_breaks_cut' nicht gefunden
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'dd' nicht gefunden
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'elev' nicht gefunden
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'quantile_sequence' nicht gefunden
```

```
## Warning in rm("bair", "br", "d", "d_breaks_cut", "dd", "elev", "i", "m", :
## Objekt 'species' nicht gefunden
```

```
ls()
```

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

8 Dataobject classes

8.1 Vector

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

8.1.1 Basics

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

8.1.2 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))
```

```
## [1] 335 460
```

```
c(x, a)
```

```
## [1] 335 460 335
```

```
c(x, x)
```

```
## [1] 335 460 335 460
```

```
x[2] <- drought$elev[3]; x
```

```
## [1] 335 480
```

```
length(x)
```

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

```
a:b
```

```
##   [1] 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352
##  [19] 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370
##  [37] 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388
##  [55] 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406
##  [73] 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424
##  [91] 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442
## [109] 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460
```

```
seq(a, b, by = .5)
```

```
##   [1] 335.0 335.5 336.0 336.5 337.0 337.5 338.0 338.5 339.0 339.5 340.0 340.5
##  [13] 341.0 341.5 342.0 342.5 343.0 343.5 344.0 344.5 345.0 345.5 346.0 346.5
##  [25] 347.0 347.5 348.0 348.5 349.0 349.5 350.0 350.5 351.0 351.5 352.0 352.5
##  [37] 353.0 353.5 354.0 354.5 355.0 355.5 356.0 356.5 357.0 357.5 358.0 358.5
##  [49] 359.0 359.5 360.0 360.5 361.0 361.5 362.0 362.5 363.0 363.5 364.0 364.5
##  [61] 365.0 365.5 366.0 366.5 367.0 367.5 368.0 368.5 369.0 369.5 370.0 370.5
##  [73] 371.0 371.5 372.0 372.5 373.0 373.5 374.0 374.5 375.0 375.5 376.0 376.5
##  [85] 377.0 377.5 378.0 378.5 379.0 379.5 380.0 380.5 381.0 381.5 382.0 382.5
```

```
##  [97] 383.0 383.5 384.0 384.5 385.0 385.5 386.0 386.5 387.0 387.5 388.0 388.5
## [109] 389.0 389.5 390.0 390.5 391.0 391.5 392.0 392.5 393.0 393.5 394.0 394.5
## [121] 395.0 395.5 396.0 396.5 397.0 397.5 398.0 398.5 399.0 399.5 400.0 400.5
## [133] 401.0 401.5 402.0 402.5 403.0 403.5 404.0 404.5 405.0 405.5 406.0 406.5
## [145] 407.0 407.5 408.0 408.5 409.0 409.5 410.0 410.5 411.0 411.5 412.0 412.5
## [157] 413.0 413.5 414.0 414.5 415.0 415.5 416.0 416.5 417.0 417.5 418.0 418.5
## [169] 419.0 419.5 420.0 420.5 421.0 421.5 422.0 422.5 423.0 423.5 424.0 424.5
## [181] 425.0 425.5 426.0 426.5 427.0 427.5 428.0 428.5 429.0 429.5 430.0 430.5
## [193] 431.0 431.5 432.0 432.5 433.0 433.5 434.0 434.5 435.0 435.5 436.0 436.5
## [205] 437.0 437.5 438.0 438.5 439.0 439.5 440.0 440.5 441.0 441.5 442.0 442.5
## [217] 443.0 443.5 444.0 444.5 445.0 445.5 446.0 446.5 447.0 447.5 448.0 448.5
## [229] 449.0 449.5 450.0 450.5 451.0 451.5 452.0 452.5 453.0 453.5 454.0 454.5
## [241] 455.0 455.5 456.0 456.5 457.0 457.5 458.0 458.5 459.0 459.5 460.0
```

```
seq(a, b, by = 5)
```

```
##  [1] 335 340 345 350 355 360 365 370 375 380 385 390 395 400 405 410 415 420 425
## [20] 430 435 440 445 450 455 460
```

```
seq(a, b, length = 10) ## Sequenz
```

```
##  [1] 335.0000 348.8889 362.7778 376.6667 390.5556 404.4444 418.3333 432.2222
##  [9] 446.1111 460.0000
```

```
rep(x, times = 3)
```

```
## [1] 335 480 335 480 335 480
```

```
rep(x, each = 3)
```

```
## [1] 335 335 335 480 480 480
```

```
rep(x, length = 7)
```

```
## [1] 335 480 335 480 335 480 335
```

```
sort(x, decreasing = TRUE)
```

```
## [1] 480 335
```

```
order(c(x, 100))
```

```
## [1] 3 1 2
```

```
a <- as.character(drought$species[1]); b <- as.character(drought$species[2])
(x <- c(a, b))
```

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

```
c(x, "Beech")
```

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

```
c(x, 2)
```

```
## [1] "Spruce" "Spruce" "2"
```

8.2 Calculations

- Grundlegende Berechnungen operieren unabhängig auf allen Elementen
- Funktionsaufrufe arbeiten (in der Regel) mit vektorisierten Argumenten
- Summary des Inhalts: `summary()`
- Häufigkeitstabelle: `table()`

8.2.1 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))
```

8.3 Matrix

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

8.3.1 Basics

P×P matrix with content `content`:

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

Fill matrix column- (default) or linewise using argument by`row = 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]`

8.3.2 Exercises

```
a <- drought$elev
b <- drought$bair
length(a) == length(b)
```

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

```
A <- matrix(nrow = length(a), ncol = 2, data = c(a, b))
all(A[, 1] == a)
```

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

```
A[1:3, ]
```

```
##      [,1]  [,2]
## [1,]  335 0.505
## [2,]  460 0.648
## [3,]  480 0.523
```

```
A[1:3, 2]
```

```
## [1] 0.505 0.648 0.523
```

```
A[c(1:3, 29), ]
```

```
##      [,1]  [,2]
## [1,]  335 0.505
## [2,]  460 0.648
## [3,]  480 0.523
## [4,]  780 0.944
```

```
B <- matrix(nrow = 3, ncol = 3, data = 1:9); B
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
diag(B)
```

```
## [1] 1 5 9
```

```
B %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   30   66  102
## [2,]   36   81  126
## [3,]   42   96  150
```

```
B %*% c(1, 1, 1)
```

```
##      [,1]
## [1,]   12
## [2,]   15
## [3,]   18
```

```
rowSums(B)
```

```
## [1] 12 15 18
```

```
c(1, 1, 1) %*% B
```

```
##      [,1] [,2] [,3]
## [1,]    6   15   24
```

```
colSums(B)
```

```
## [1]  6 15 24
```

8.4 List

A list is a general 'container' object.

8.4.1 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)
```

8.4.2 Exercises

```
beech <- list(species = "Fagus",
              n = sum(drought$species == "Beech"),
              data = subset(drought, species == "Beech"))
beech
```

```
## $species
## [1] "Fagus"
##
## $n
## [1] 11
##
```

```
## $data
##     bair   elev species         elev_cut
## 24 0.697  475.0   Beech        (250,500]
## 25 0.606  480.0   Beech        (250,500]
## 26 0.718  507.5   Beech        (500,750]
## 27 0.480  580.0   Beech        (500,750]
## 28 0.822  750.0   Beech        (500,750]
## 29 0.944  780.0   Beech      (750,1e+03]
## 30 0.770  800.0   Beech      (750,1e+03]
## 31 1.036 1025.0   Beech (1e+03,1.25e+03]
## 32 1.230 1100.0   Beech (1e+03,1.25e+03]
## 33 0.680 1150.0   Beech (1e+03,1.25e+03]
## 34 0.985 1200.0   Beech (1e+03,1.25e+03]
```

```
beech[[1]]
```

```
## [1] "Fagus"
```

```
beech[['species']]
```

```
## [1] "Fagus"
```

```
beech$species
```

```
## [1] "Fagus"
```

```
str(beech)
```

```
## List of 3
##  $ species: chr "Fagus"
##  $ n      : int 11
##  $ data   :'data.frame': 11 obs. of  4 variables:
##   ..$ bair    : num [1:11] 0.697 0.606 0.718 0.48 0.822 ...
##   ..$ elev    : num [1:11] 475 480 508 580 750 ...
##   ..$ species : Factor w/ 2 levels "Beech","Spruce": 1 1 1 1 1 1 1 1 1 1 ...
##   ..$ elev_cut: Factor w/ 6 levels "(250,500]","(500,750]",..: 1 1 2 2 2 3 3 4 4 4 ...
```

```
lapply(beech, FUN = dim)
```

```
## $species
## NULL
##
## $n
## NULL
##
## $data
## [1] 11  4
```

```
lapply(beech, FUN = summary)
```

```
## $species
##    Length     Class      Mode
##         1 character character
##
## $n
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      11      11      11      11      11      11
##
## $data
##       bair             elev          species                  elev_cut
##  Min.   :0.4800   Min.   : 475.0   Beech :11   (250,500]         :2
##  1st Qu.:0.6885   1st Qu.: 543.8   Spruce: 0   (500,750]         :3
##  Median :0.7700   Median : 780.0               (750,1e+03]       :2
##  Mean   :0.8153   Mean   : 804.3               (1e+03,1.25e+03]  :4
##  3rd Qu.:0.9645   3rd Qu.:1062.5               (1.25e+03,1.5e+03]:0
```

```
##  Max.   :1.2300   Max.   :1200.0                (1.5e+03,1.75e+03]:0
```

8.5 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).

8.5.1 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)
```

8.6 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`).

8.6.1 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
```

```
##         day daily_mean_temperature_C precipitation_sum_mm       site
## 1    Monday                       12                    5 Goettingen
## 2   Tuesday                       14                    9 Goettingen
## 3 Wednesday                       11                   25 Goettingen
```

```
weather[1:2, ]
```

```
##       day daily_mean_temperature_C precipitation_sum_mm       site
## 1  Monday                       12                    5 Goettingen
## 2 Tuesday                       14                    9 Goettingen
```

```
weather$day[1:2]
```

```
## [1] Monday  Tuesday
## Levels: Monday Tuesday Wednesday
```

```
weather[["daily_mean_temperature_C"]][3]
```

```
## [1] 11
```

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

```
##   daily_mean_temperature_C precipitation_sum_mm
## 1                       12                    5
## 2                       14                    9
```

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

```
##       day precipitation_sum_mm
## 1  Monday                    5
## 2 Tuesday                    9
```

8.7 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)`.

8.7.1 Exercises

```
summary(weather)
```

```
##         day    daily_mean_temperature_C precipitation_sum_mm         site
##  Monday   :1   Min.   :11.00            Min.   : 5           Goettingen:3
##  Tuesday  :1   1st Qu.:11.50            1st Qu.: 7
##  Wednesday:1   Median :12.00            Median : 9
##                Mean   :12.33            Mean   :13
##                3rd Qu.:13.00            3rd Qu.:17
##                Max.   :14.00            Max.   :25
```

```
str(weather)
```

```
## 'data.frame':    3 obs. of  4 variables:
##  $ day                     : Factor w/ 3 levels "Monday","Tuesday",..: 1 2 3
##  $ daily_mean_temperature_C: num  12 14 11
##  $ precipitation_sum_mm    : num  5 9 25
##  $ site                    : Factor w/ 1 level "Goettingen": 1 1 1
```

```
head(weather, n = 2)
```

```
##       day daily_mean_temperature_C precipitation_sum_mm       site
## 1  Monday                       12                    5 Goettingen
## 2 Tuesday                       14                    9 Goettingen
```

```
dim(weather)
```

```
## [1] 3 4
```

9 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"
```

```
## [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 orcollapse).

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

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

10 Working with 'real' dataframes

R is a powerful tool not only for analysis, but also for managing of data.

10.1 Reproducible data management

- Avoid any steps in spreadsheet-software as MS Excel that are done 'by hand' and are non-repdroducible.
- Generate an *R*-Script `01_datamanagement.R` that loads your data and does all the steps and stops with an object `df` that is used for any further steps.
- Run this file using `source(01_datamanagement.R)`.

10.2 Preparatory steps prior R

Data management in *R* begins with loading the data into *R*. But before we think about the technical way in which we read our data into *R*, we should first ensure that they are 'prepared' for this *R* import. If we neglect a few basic rules in this preparatory work, we actually only hold back problems (for which we then only have to find unnecessarily 'complicated' solutions in *R*).

- The first line of the dataframe is reserved for the variable names: If we set `header = TRUE` (*R* functions`read.csv ()`or`read.table ()`) the first line in the dataframe becomes for the variable names recycled.
- The first column (s) should be used to define the observation unit: Tree ID (, section, section ID, ...). It is preferable to keep the information here separately:
- It is easier to combine several variables (in *R*) than to split one variable into several variables.

10.2.1 Variable names

- Avoid variable names, values, or cells with spaces. If ignored – and not adequately handled during data import – each word is treated as a (value of) a separate variable. This leads to errors that are always related to the fact that the number of elements varies between the lines (matrix 'behavior' of the dataframe).
- If several words are to be put together, ideally use underscores, eg. `Count_per_ha` (Alternatively, you can also use dots – `Count.per.ha` –, or start every single word with a capital letter – `CountPerHa`).
- **Decide on a rule and try to stick to it!**
- Functions like `tolower`,`gsub ()`and`stringsplit` are useful helpers to quickly carry out repairs in *R*.
- Make sure that all missing values – especially empty cells – are marked with *NA*.
- Delete all free text comments (this only leads to extra columns or an inflation of *NA*s).

10.3 Import dataframes

10.3.1 Preparatory Work in *R*

Before reading in a dataframe, you have to tell *R* where it can be found.

- To achieve this, it can be helpful to find out which working directory *R* is currently accessing:

```
getwd()
```

- If different from the storage location of the dataframe, we have to change this working directory:

```
setwd("<Pfad des Ordners in welchem der Datensatz gespeichert ist>")
```

By executing this function call, *R* now knows in which working directory we want to work.

** For RStudio users **

It is very helpful to save the current `.R` code file and the dataframe in the same directory: At the beginning of a working session, you can then click on *Session* → *Set Working Directory* → *To Source File Location* (this also facilitates collaboration).

Spreadsheet software (such as MS Excel) allows dataframes to be saved in many different file formats:

- The most popular non-standard formats (i.e. not `.xls` or`.xlsx`) to save a dataframe are `.csv` (for *comma-separated value*) and`.txt` (tab-separated text file).
- Depending on this choice, a line's contents are either separated by commas or tabs (referred to as 'separation argument').
- **Attention:** Under operating systems with German language setting, commas are the default setting for the argument to represent decimal numbers. Here it is necessary to change the default values for the arguments `sep` and`dec` to `sep ="; "` and `dec =", "` (required in `read.table()`, `read.csv()` and `read.csv2()`; see also the next sections).

10.3.2 `read.table()`.

If the data was saved in the above-mentioned tab-separated text format `.txt`, the function `read.table()` is a simple way of importing data:

```
d <- read.table("<FileName>.txt", header = TRUE)
```

- By previously defining the working directory (`setwd ()`), the dataser can be imported directly by specifying the file name.
- The value of the `header` argument can be used to determine whether the first line of the dataframe contains the variable names (`TRUE` is the default value here).
- The separation argument `sep` is set to the value `""`, since the `read.table` command is defined for tab-separated text formats.

To specify a different separation argument, the value of the argument `sep` must be changed:

```
df <- read.table("<FileName>.txt", header = FALSE, sep = ";")
```

10.3.3 `read.csv()` and `read.csv2()`

Die Funktionen `read.csv()` und `read.csv2()` können genutzt werden um Datenätze im `.csv` (Comma Separated Values) Format einzulesen.

- `.csv` dataframes can be easily imported into *R* and are therefore a preferred format.
- `read.csv()` and `read.csv2()` use different separation arguments: for `read.csv()` the comma, for `read.csv2()` the semicolon (open your dataframe in a text editor in order to quickly find out your separation argument).
- `read.csv()` and `read.csv2()` are special cases of `read.table()`: This means that the arguments for `read.table()` can alternately also be used for `read.csv()` and `read.csv2()`.
- `read.csv()` and `read.csv2()` are very similar to `read.table()` and differ from `read.table()` in only three aspects:
  - The separation argument `sep`,
  - he argument `header` is always set to`TRUE`, and
  - the argument `fill` is also `TRUE`, which means that if lines have unequal lengths, `read.csv()` and `read.csv2()` will always fill the short lines with empty cells.

```
d <- read.csv("data.csv", header = T, sep = ";", dec = ",", stringsAsFactor = F)
names(d)
```

11 Factors

For the analysis of qualitative characteristics, it makes sense to represent variables with character strings as factors. In doing so, factor levels (`levels`) are generated, which are numbered internally, but represented externally by the original values of the character string.

It is generally advisable to import a dataframe with the argument `stringsAsFactor = FALSE` (if partial dataframes are to be created at a later point in time, loading with `stringsAsFactor = FALSE` is helpful in order not to have any 'ghosts' later; see the next but one section).

Practical procedure:

- Import the data with `stringsAsFactor = FALSE`
- Check the structure of the dataframe with the help of `str()`: Surprisingly, have variables been imported as strings (class `chr`)? If so, check the characteristics of these variables for incorrect values with `unique()` or `xtabs ()`.
- Also check with `unique()` or `xtabs()` the characteristics of the variables that were planned to be read in as `chr`: Typing errors can easily lurk here as well.
- Use `df$var <- as.factor(df$var)` to finally assign the `factor` class to variable `var`.

12 attach, subset and merge

Never say never but **never use** `attach()`.

. . . use a short name for the dataframe object (e.g. `df`) to keep the paperwork to a minimum!

The function `merge(data_set1, data_set2, by, ...)` can be used to link two dataframes using a key variable:

```
library("plyr")
dd <- ddply(drought, c("species"), summarize,
            mean_bair = mean(bair),
            sd_bair = sd(bair),
            mean_elev = mean(elev),
            sd_elev = sd(elev))
head(merge(drought, dd, by = c("species")))
```

```
##   species  bair  elev    elev_cut mean_bair   sd_bair mean_elev  sd_elev
## 1   Beech 0.697 475.0   (250,500] 0.8152727 0.2159037  804.3182 277.0922
## 2   Beech 0.606 480.0   (250,500] 0.8152727 0.2159037  804.3182 277.0922
## 3   Beech 0.718 507.5   (500,750] 0.8152727 0.2159037  804.3182 277.0922
## 4   Beech 0.480 580.0   (500,750] 0.8152727 0.2159037  804.3182 277.0922
## 5   Beech 0.822 750.0   (500,750] 0.8152727 0.2159037  804.3182 277.0922
## 6   Beech 0.944 780.0 (750,1e+03] 0.8152727 0.2159037  804.3182 277.0922
```

`subset` can be used to split a dataframe according to the result of a logical comparison:

```
sdf <- subset(drought, elev_cut == "(500,750]" & species == "Beech")
sdf
```

```
##     bair  elev species  elev_cut
## 26 0.718 507.5   Beech (500,750]
## 27 0.480 580.0   Beech (500,750]
## 28 0.822 750.0   Beech (500,750]
```