Coursera

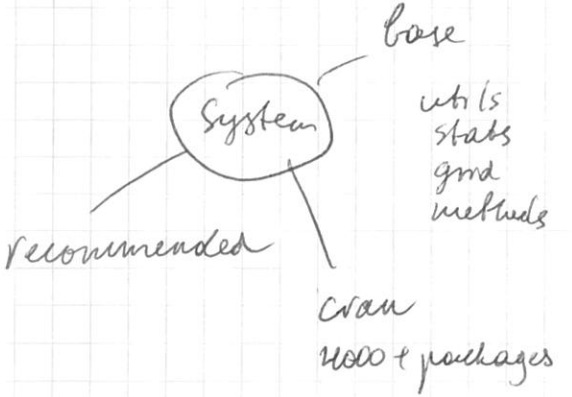
Computing for Data Analysis

Installing - just follow the wizard

About R

Not ideal for everything

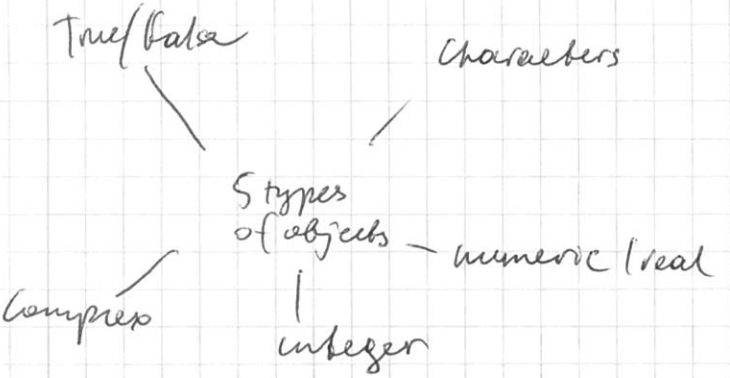
- needs to load a whole object into memory → impossible to work with huge objects



Books: Software for DA
Use R!

W1

Data types



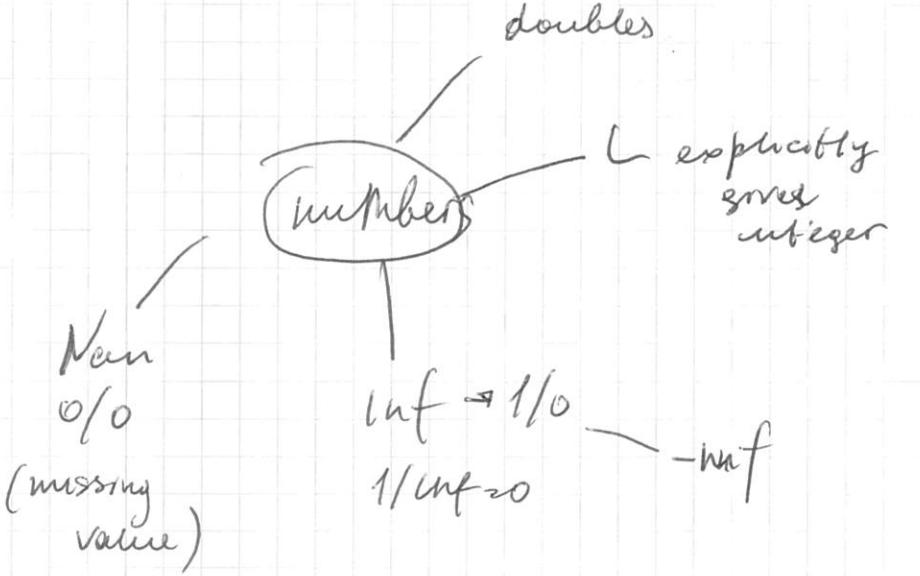
the most basic object - is a vector

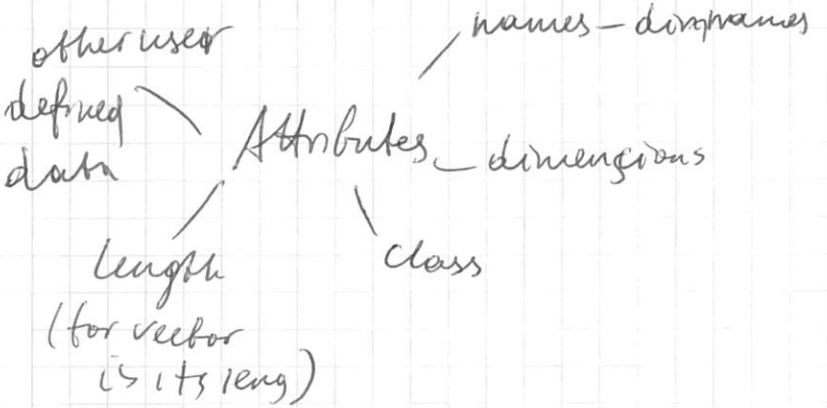
↓

everything is of the same class

but there's a list - vector that can contain different types

vector() creates an empty vector





attributes() – general function

At the prompt ~~the~~ we type expressions

> x <- 1 assignment

> print(x)

[1] 1 <- Numeric vector with one expression

> x – just prints

> msg <- "hello"

- comment

When a complete expression is typed, it's evaluated and the result of evaluation is returned

> 5

[1] 5

↑ first element

it's vector

```
x <- 1:20
```

sequence

```
> print x
 [1]  1 ... 15
[16] 16 .. 20
```

C. other function for creating vectors

(stands for concatinate)

```
x <- c(0.5, 0.6)      Numeric
  <- c(TRUE, FALSE)
     c(T, F)          logical
     c("a", "b")      character
     ~~c~~ 0 9:29     integer
     c(1+0i, 2+4i)    complex
```

```
x <- vector("numeric", length = 10)
```

y ← c(1.7, "a") ← character

↑ different types

no error!

R will try to COERCE the elements

(tries to find „least common denom.")

1.7 → "1.7"

c(T, 2) => [1, 2]

c("a", TRUE) => ["a", "TRUE"]

can be explicitly coerce

as.* function x <- 0:6

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

=> gets converted

if no conversion is possible,
NA is returned

Matrices

vectors with dimension attribute

↓

(nrow, ncol)

m ← matrix(nrow=2, ncol=3) ← initialized with NA

dim(h) ⇒ 2 3

attributes(m)

$dim - 2, 3

M. constructed column-wise

m ← matrix (1:6, nrow=2, ncol=3)

"from upper left" con

1 3 5
2 4 6

another way

```
m ← 1:20
```

changing dimension attribute

```
?. dim(m) ← c(2,5)
```

doesn't work!...

```
column-binding [row-binding
cbind()        rbind()
```

```
x ← 1:3   y ← 10:12
```

```
cbind(x, y)
```

⇓

```
1  10
2  11
3  12
```

```
rbind(x, y)
```

⇓

```
1  2  3
10 11 12
```

Lists

vector with different data

x ← list(1, "a", T, 1+4i)

[[1]] ← double-bracketed
[1] 1
...

Factor

categorical data vector

can be

ordered unordered

factor — integer vector with each integer having a label

ifelse() glm()

self-describing ⇒ better than just numbers

x ← factor(c("yes", "yes", "no", "yes", "no"))

table(x) ⇒ frequency

no 2 | yes 3

levels: sorted alphabetically

unclass(x) ← strips out the class

2 2 1 2 1 2 1

1 = yes, 2 = no

attr(, "levels") ⇒ "no" "yes"

x ← factor(c(...), levels = c("yes", "no"))

explicit ← (not always good) ← auto

levels: ~~no~~ yes no (note the order!)

Missing Values

NaN – not a number

NA – missing value, any types

is.na() – if NA

is.nan

```
x <- c(1, 2, NA, 10, 3)
```

is.na(x)

returns logical vector

FALSE TRUE

NaN is NA

is.na ↓ true

is.na(NA) => True

is.nan(NA) => false

is.nan(NaN) => True

is.na(NaN) => True

Data Frames

for storing tabular data

Special type of list
with every el of the same len

special attributes:
row.names

read.table()
read.csv()

data.matrix converts to a matrix
↓
all coerce!

x ← data.frame(foo = 1:4
bar = c(T, T, F, F))

nrow(x) returns number of rows
ncol(x) number of cols

Names

good for describing data

```
x <- 1:3
names(x) => NULL
names(x) <- c("foo", "bar", "n")
```

x => will be printed with the names

names(x) returns the vector

list can also have names

```
x <- list(a = 1, b = 2, c = 3)
```

matrices can also

```
m <- matrix(1:4, nrow = 2, ncol = 2)
dimnames(m) <- list(c("a", "b"),
                    c("c", "d"))
```

m =>

	c	d
a	1	3
b	2	4

Summary of Data Types:

- atomic classes
  numeric, logical, char, int, complex
- vector, list
- factors
- missing values
- data frames
- names

Subsetting

extracting subsets

- `[` brackets — returns the same type (vector → vector, list → list)
- `[[` double brackets — element of a list / data frame, only a single element
- `$` extraction from list of data frame, but using a name

starts from 1

```
x <- c("a", "b", "c", "c", "d", "a")
x[1] => "a"
x[1:4] => a b c c
x[x > "a"] b c c d

u <- x > "a"
F T T T T F
x[u] b c c d
(only trues)
```

(1 🏷)

x <- matrix (1:6 , 2, 3)

[i, j]

x [1, 2] => 3

x [2, 1] => 2

1	3	5
2	4	6

missing

x [1,] ← first row

x [, 2] ← second col

3, 4

returns a VECTOR!

x [1, 2, drop = F]

will return a matrix of 1 el

preserves dimensions

x [1, , drop = F]

returns a matrix

List

```
x ← list(foo = 1:4, bar = 0.6)
x[1]
   ⇓
$foo                (returns list)
[1] 1 2 3 4
```

x[[1]] => [1] 1 2 3 4 (vector)

x$bar <=> x[["bar"]] => 0.6 (element)

```
x["bar"]
   ⇓
$bar                (returns list)
[1] 0.6
```

x ← list(foo, bar, baz)

x[c(1, 3)]

name ← "bar"

x[name] ; x$name ("name" doesn't exist)

⇓ bar ; ⇓ null!

x[[c(1, 3)]]

↓

$ list 3th element

||

x[[1]][[3]] ⇐ the same

partial matching - for command line

x ← list(aaa... = 1:5)

x$a ← partially matches!

x["a"] ⇒ NULL

↑

doesn't work!

x[["a", exact = F]]

↓

⇒ the same as

removing NAs

```
bad <- is.na(x)
x[!bad]
```

↑ inversion!

```
x <- with nans
y <- with nans

good <- complete.cases(x, y)
x[good], y[good]
```

~~new~~

```
r <- matrix
good <- complete.cases(r)
r[good, ][1:6,]
```

Vectorised operators

no loops!

```
x <- 1:4   y <- 6:9

x + y
x > 2
x >= 2   <= each with each
y <= 8
x * y
x / y
```

```
x <- matrix(1:4, 2, 2)
y <- matrix(rep(10, 4), 2, 2)

x * y  # element-wise mult!
x / y  - element-wise
x %*% y  - matrix multiplication
```

Read/Write data

read.table read.csv
readLines – from text file (char vector)
source – R code
dget – R object
load
unserialize

↓ analogues

write.table
writeLines
dump
dput
save
serialize

read.table

args:

- file
- header - logical (var. names in the 1st row)
- sep - separator (" ", "
- colClass - char vector - types w. this
  (not required)
- n rows
- comment.char - character to be ignored after it
- skip - n of lines to skip
- stringsAsFactors

  should character vars be coded as factors?

data <- read.table("foo.txt")

↑ only file is required others set to default

default values!

- comment, R figures out types

for small data

read.csv - similar,
def. separator is comma

For larger sets

read.table - ~~read~~ RTFM!

make a rought calculation
if RAM is enought

another opt:
comment.char = "" - faster
colClasses - faster, if explicitly defined
Much

How to figure out the classes?

```
initial <- read.table("db.txt", nrows=100)
classes <- sapply(initial, class)   # reads the classes
head(initial)   # just shows a head of data
tabAll <- read.table("db.txt"
                     colClasses=classes)
```

nrows - doesn't make R faster, but better for memory

str(.Platform)

Rought Calc for RAM

15 mln rows × 120 cols (Numeric) ≈ 8 Bytes

15 000 000 × 120 × 8 Bytes = 1,34 GB

(then multiply by 2)

Textual formats

dump and dput include the metadata

source dget

↑

it's editable, good for ~~SVN~~ Version Control!

(not very space efficient)

```
y <- data.frame(a = 1, b = "a")
dput(y) => to the console
(some R code)
dput(y, file = "y.R")
new.y <- dget("y.R")
```

dump can be used for multiple objects

```
dump(c("x", "y"), file = "data.R")
```

("x", "y") — names of objects

```
rm(x, y)  # removes vars
source("data.R")
```

Interfaces to outer world

file - connects to a file

gzfile (gzip)

bzfile (bzip2)

url - to a web page

File Connections

str(file)

description - name of the file

open - code indicator

"r"	"w"	"a"	"rb"	"wb"	"ab"
read	write (and create a new file)	append	binary		

```
con <- file("foo.txt", "r")
data <- read.csv(con)
close(con)
```

⟺ `read.csv("data.txt")`

```
con <- gzfile("w.gz")
x <- readLines(con, 10)
                    ↑
                first 10 lines
```

~~writeLines~~

```
con <- url("http://...", "r")
x <- readLines(con)
```

Working directory and Editing R files

```
getwd()          ← should be there
read.csv("file")
```

File → change dir...

dir() ⇒ ~~ls~~

ls - list of functions

File → new script...

```
myfunc <- function() {
    x <- rnorm(100)
    mean(x)
}
```

Save as ~ mycode.R

```
source("mycode.R")
```

STR

structure of an R object
for diagnosis

str(str) ← brief summary of function, data

summary(x) min, max
~~str~~ on vector - mean, etc

~~PA1~~ PA 1

7. mean of Ozone layer without NA values

```
ozon ← data$Ozon
mean(ozon[!is.na(ozon)])
```

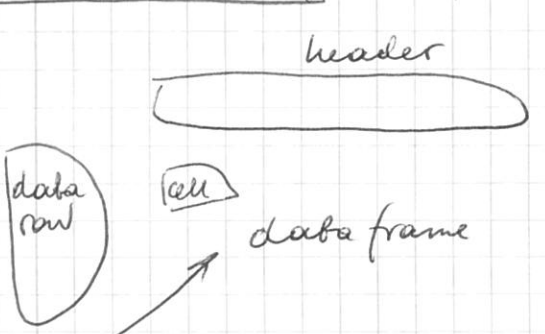
~~8~~ The mean of Temp when Month = 6.

```
mean(data[data$Month == 6, ]
     $Temp)
```

8. Extract the subset of rows of the data frame where Ozon values are above 31 and Temp > 90
What is the mean of Solar.R in this subset?

```
good ← data$Ozone > 31 & data$Temp > 90
good[is.na(good)] ← False
mean(data[good,]$Solar.R)
```

Data Frames (Again)



df[1] - returns whole first column
["name"] name col

df[c("name", "name2")]

↓
returns 2 cols

df[24,] - 24th row

L <= df$name == 0
df[L,]
df[L,]$name