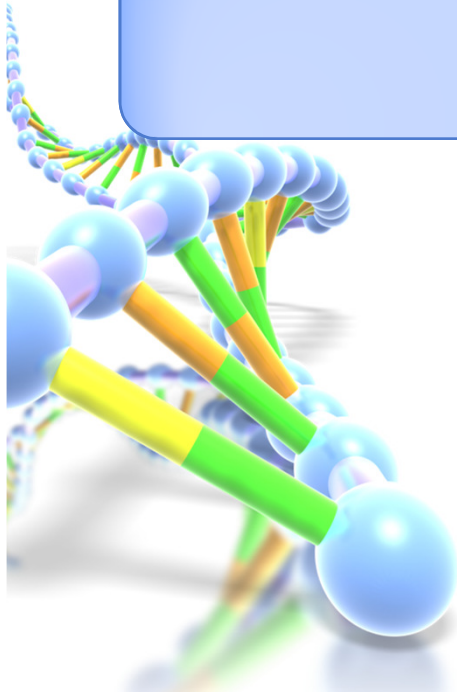


Les 2 - Introductie R (2)

**Emile Apol
Patrick Deelen**

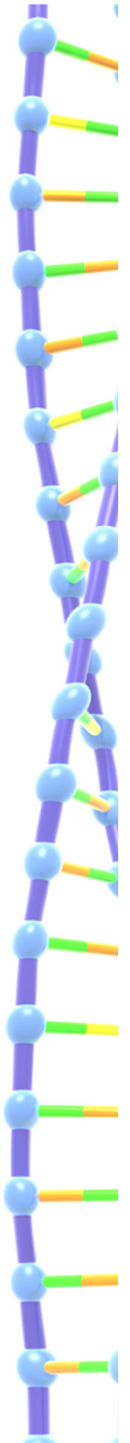


Hanze University Groningen
APPLIED SCIENCES

*Institute for
Life Science & Technology*

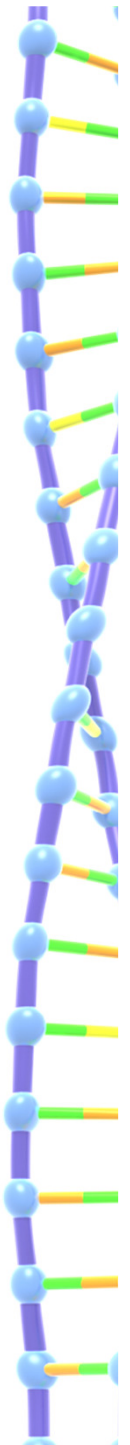
LES 2

- Sort/order
- R datatypen
 - Data frame
 - List
 - Factor
- Input/Output
- Apply
- Handige functies



REPLACE

- Met de functie **replace** kan je elementen van een vector vervangen
 - Gebruik hierbij een assignment (`<-`)
- Voorbeeld:
`a <- c(1, 3, 7, 5, 8, 11, 13, 4)`
- Vervang elk element in `a > 5` door 25:
`a[a > 5] <- 25` geeft: 1, 3, 25, 5, 25, 25, 25, 4
- Met behulp van **replace**:
`b <- replace(a, a > 5, 25)`
`a <- replace(a, a > 5, 25)`



SORT EN ORDER

- Met de functie **sort** kan je een vector sorteren
 - Alleen gebruiken als het echt nodig is
 - Annotaties of andere stukken code kunnen uitgaan van een bepaalde volgorde

- Voorbeeld:

```
a <- c(1, 3, 7, 5, 8, 11, 13, 4)
```

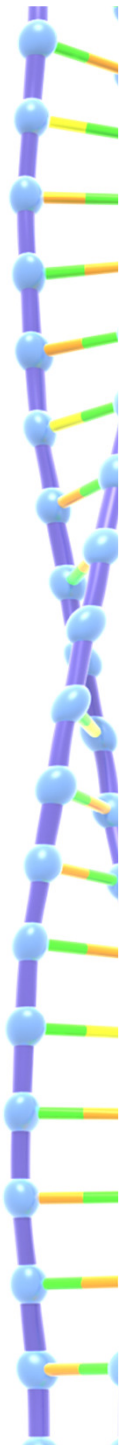
- Sorteert:

```
sort(a)          geeft:      1, 3, 4, 5, 7, 8, 11, 13
```

- Index van gesorteerde elementen van a:

```
volg <- order(a)  geeft:      1, 2, 8, 4, 3, 5, 7, 8
```

```
a[volg]           geeft:      1, 3, 4, 5, 7, 8, 11, 134
```



SORT EN ORDER

- Voorbeeld:

a <- c(1, 3, 7, 5, 8, 11, 13, 4)

- Sorteert/order:

sort(a) geeft: 1, 3, 4, 5, 7, 8, 11, 13

volg <- order(a) geeft: 1, 2, 8, 4, 3, 5, 7, 8

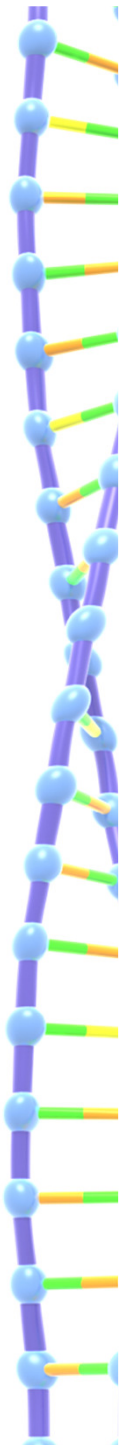
a[volg] geeft: 1, 3, 4, 5, 7, 8, 11, 13

want:

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
a	1	3	7	5	8	11	13	4
sort(a)	1	3	4	5	7	8	11	13
order(a)	1	2	8	4	3	5	6	7

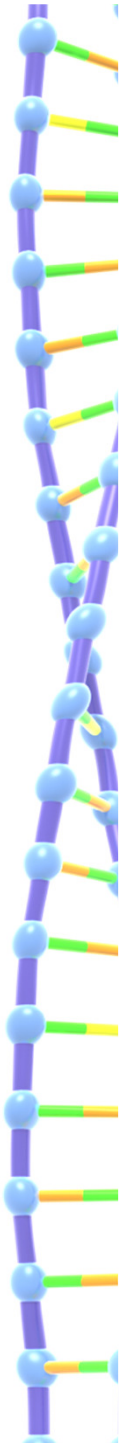
ORDER

- De functie **order** geeft de gesorteerde volgorde van elementen
 - `y <- c("C", "D", "A")`
 - `order(y)` #geeft: 3 1 2
 - `y[order(y)]` #geeft: "A" "C" "D"
 - `y[c(3, 1, 2)]` #geeft: "A" "C" "D"
 - `y[c(3, 1)]` #geeft: "A" "C"
 - `y[order(y)[1:2]]` #geeft: "A" "C"
- Vector **b** sorteren op volgorde van elementen vector **a**:
 - `b[order(a)]`



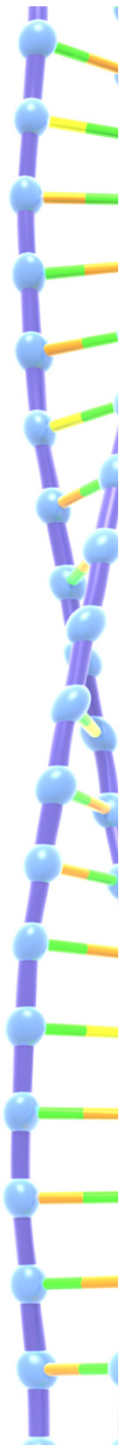
DATA FRAME

- Lijkt op een matrix
- Kolommen mogen verschillende modus hebben.
 - `geneNames <- c("P53", "BRCA1", "VAMP1")`
 - `sig <- c(TRUE, TRUE, FALSE)`
 - `meanExp <- c(4.5, 7.3, 5.4)`
 - `Genes <- data.frame(geneNames, sig, meanExp)`
- Of:
 - `genes <- data.frame(
 "name" = geneNames,
 "significant" = sig,
 "meanExp" = meanExp
)`



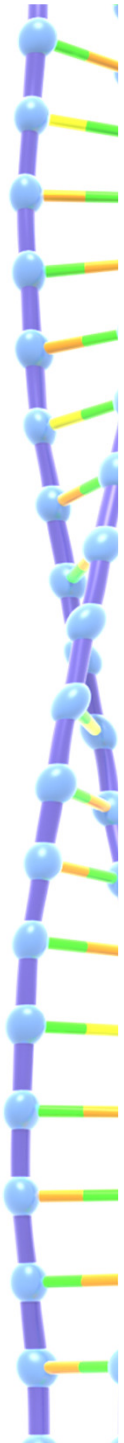
DATA FRAME

- rownames en colnames werken hetzelfde als bij een matrix
 - `genes[2, 1]` `#rij 2, element 1`
 - `genes[, 1:2]` `#kolom 1 en 2`
 - `genes[1:2]` `#kolom 1 en 2 (!)`
 - `genes[1:2,]` `#rij 1 en 2`
 - `genes[c("name", "meanExp")]`
 `#kolom "name" en "meanExp"`
 - `genes$name` `#kolom "name"`
- Info over dataframe:
 - `str(genes)`



DATA FRAME

- `genes$name` #kolom "name"
- Door middel van het "attachen" van het dataframe **genes** heb je *direct* toegang tot de objecten **name**, **significance** en **meanExp**:
`attach(genes)`
`name`
`significance`
`meanExp`
- Analyse klaar: "detachen" van dataframe:
`detach(genes)`
- Verwijderen van objecten uit Workspace met **rm**:
 - `rm(sig, geneNames)`



SORTEREN OP MEERDERE LEVELS

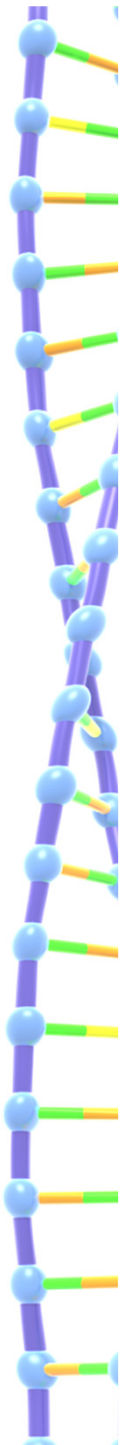
- Je kunt met **order** ook op **meerdere levels sorteren**, bijv. eerst op naam en dan binnen naam op leeftijd:
 - `myData <- data.frame(name, age)`
 - `myData[order(name, age),]`

sorteer de rijen, niet de kolommen!

	name	age
1	David	30
2	David	29
3	Emile	43
4	Anna	19



	name	age
1	Anna	19
2	David	29
3	David	30
4	Emile	43



"SCOPE" VAN VARIABELEN IN R

- R heeft een **search path** voor objecten/variabelen:

- **search()**

```
> search()
[1] ".GlobalEnv"      "package:VGAM"      "package:stats4"    "package:splines"
[5] "package:mvtnorm" "genExpr.2"         "genExpr.1"         "tools:rstudio"
[9] "package:stats"   "package:graphics" "package:grDevices" "package:utils"
[13] "package:datasets" "package:methods"  "AutoLoads"         "package:base"
```

- R zoekt *vanaf begin van deze lijst* totdat object gevonden is.

- Volgorde:

- 1^e positie:

altijd .GlobalEnv

- laatste positie:

altijd package:base

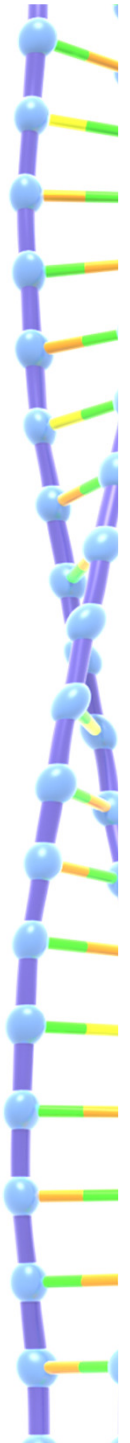
- 2^e positie:

altijd laden nieuw package /
attachen dataframe

zelf gedefinieerde
variabelen

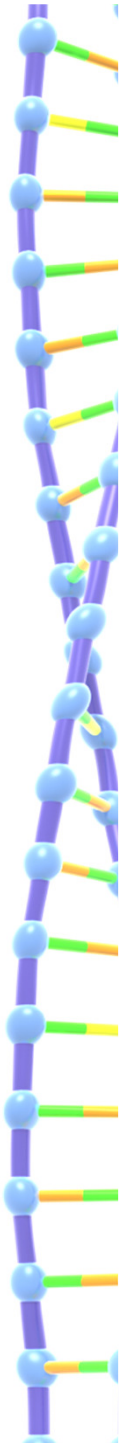
INPUT/OUTPUT

- Info/veranderen van “working directory”:
 - `getwd()` `# print working dir`
 - `setwd(path)` `# set working dir`
 - `dir()` `# list content dir`
- met als conventies voor `path` (Windows/Linux):
 - Windows `"E:\\emile\\datasets"`
 - Linux `"~/datasets"`
 - of `"/home/emile/datasets"`



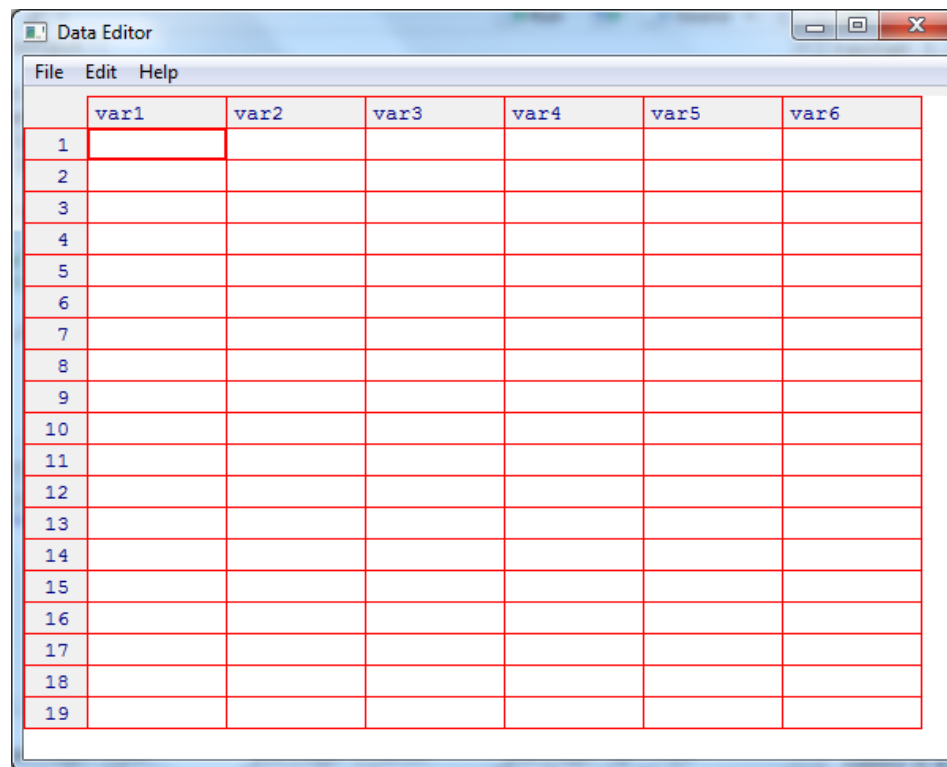
INPUT/OUTPUT

- Inlezen van data uit ascii *.txt file in een dataframe:
 - `myData <- read.table("data.txt", header = T)`
 - kolommen worden genoemd zoals headers
- Nieuwe kolomnamen geven:
 - `myData <- read.table("data.txt", header = T, "column1" = nr, "column2" = x, "column3" = y)`
- Informatie over `myData`:
 - `summary(myData)`
 - `str(myData)`



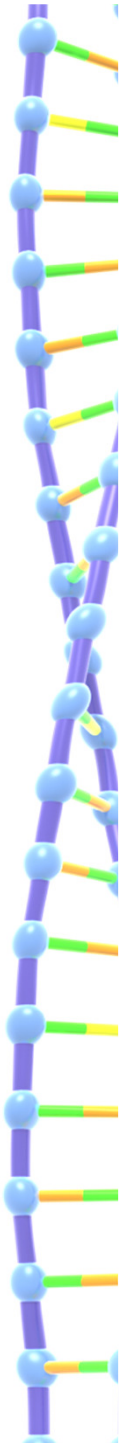
INPUT/OUTPUT

- Je kan ook een “spreadsheet” openen om data in te voeren:
 - `myData <- data.frame() # leeg frame`
 - `edit(myData)`



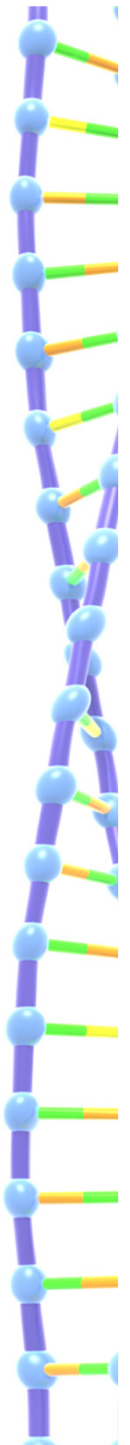
The screenshot shows the 'Data Editor' window in R. It has a menu bar with 'File', 'Edit', and 'Help'. Below the menu bar is a table with 6 columns labeled 'var1', 'var2', 'var3', 'var4', 'var5', and 'var6'. The rows are numbered 1 through 19. The first cell in the first row (var1, row 1) is highlighted with a red border.

	var1	var2	var3	var4	var5	var6
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						



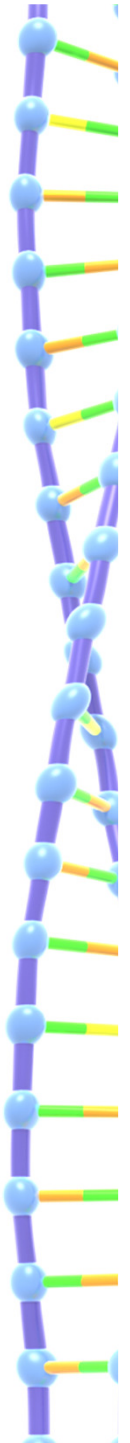
INPUT/OUTPUT

- Schrijven van dataframe/matrix/vector naar file:
 - `write.table(myData, file="file.dat")`
- Standaard een coma-separated file, met kolom- en rijnamen, tenzij als optie(s)
 - `col.names = F`
 - `row.names = F`
 - `sep = "\t"` of `sep = ";"`
tab separated of ; - separated
- Schrijven van (grote) datamatrices (package MASS)
 - `write.matrix()`



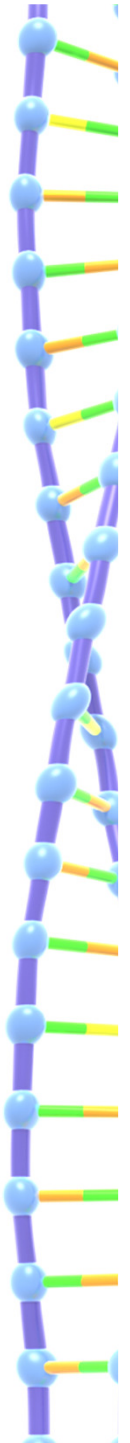
LIST

- Een list is een geordende verzameling van componenten
- Een list kan alle mogelijke objecten bevatten
 - Dus ook 1 of meerdere listen
- Voorbeeld:
 - `geneName <- "P53"`
 - `annotations <- c("Onco gene", "Cell Cycle", "Apoptosis")`
 - `gene <- list("name" = geneName, "annotation" = annotations)`



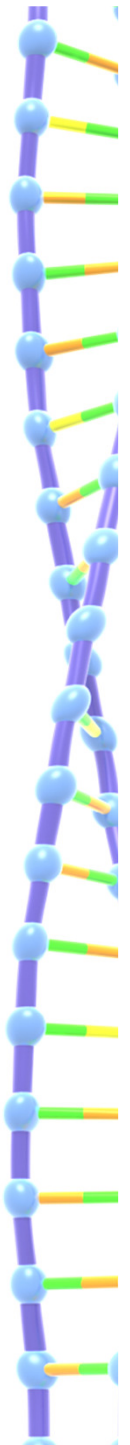
LIST

- `gene[[2]]` #Annotatie vector
- `gene[[2]][2]` #Cell Cycle
- `gene$name`
- `gene[["name"]]`



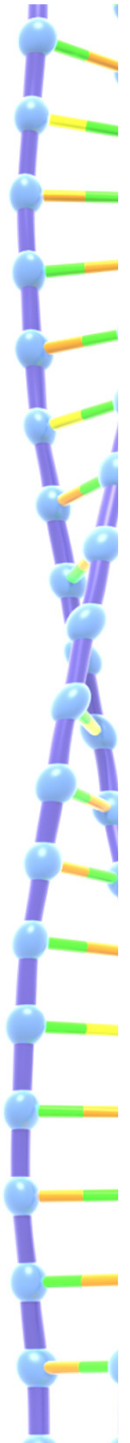
VERSCHIL [] EN [[]]

- Enkele blokhaken zijn om een subset te selecteren
 - Bijv. element 1 t/m 10 van een vector
- Dubbele blokhaken selecteren een element in een list
- `gene[1]` `#list met lengte 1 die gen naam bevat`
- `gene[[1]]` `#gen naam`
- `gene[1]$name` `#gen naam`
- `gene$name` `#gen naam`



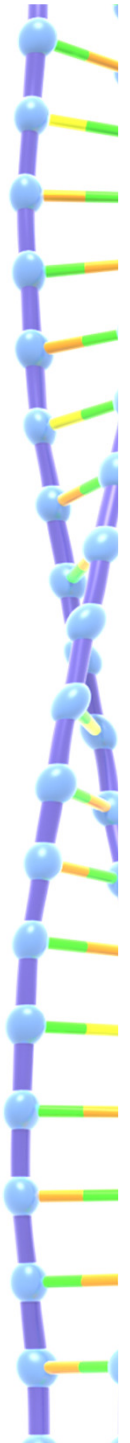
IF / FOR

- `if (TRUE) {
 print("TRUE")
} else {
 print("FALSE")
}`
- `for(i in 1:3) {
 print(i)
}`



FUNCTION

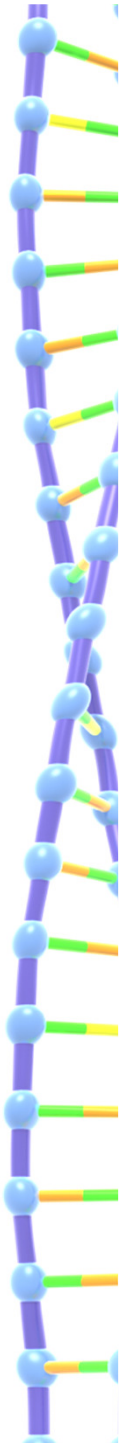
- `mean2 <- function(x) {
 return(mean(x))
}`
- `mean2(3:5) # geeft dan 4`
- function wordt gebruikt om een functie te maken
- return wordt gebruikt om een waarde terug te geven
 - Denk hierbij wel om de haken ()
- Let op: Als je in RStudio een functie aanpast moet je hem opnieuw uitvoeren



APPLY

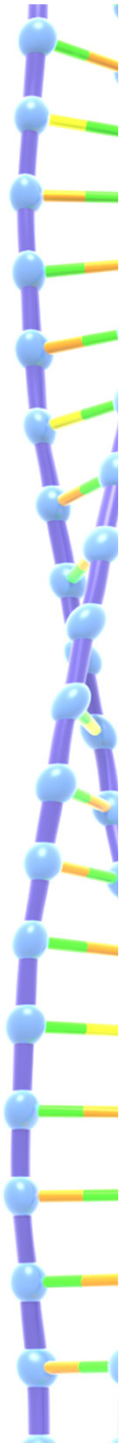
- apply is erg handig om berekening over een matrix uit te voeren
- sapply kan je gebruiken om door een list of vector heen te lopen
- Is vaak beter leesbaar en sneller
- ```
a <- matrix(
 rnorm(500, mean = 5, sd = 1),
 nrow = 5, ncol = 100
)
```
- ```
apply(a, 1, mean)
```

 - 4.899979 5.005887 5.099075 4.950709 5.087162



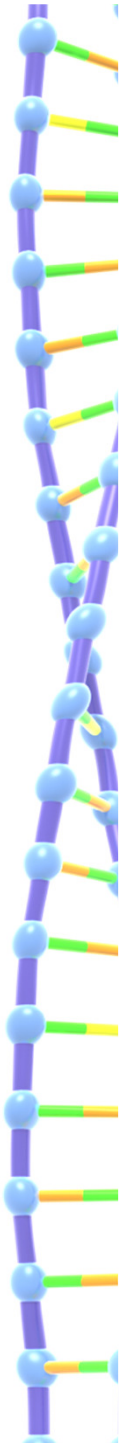
APPLY

- `apply(a, 1, mean)`
- Het tweede argument geeft aan of je over rijen (1) of over de kolommen (2) wil rekenen.
- Met `c(1, 2)` wordt je functie op ieder element uitgevoerd



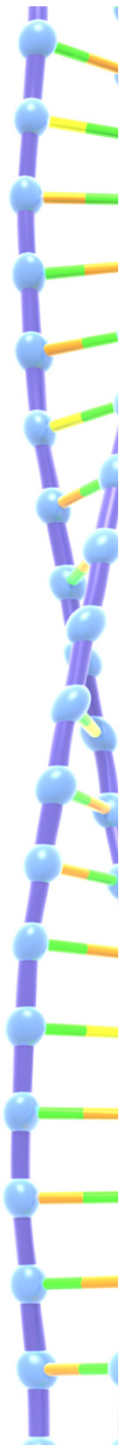
APPLY

- `apply(a, 1, function(x) {
 return(mean(x))
})`
- Bovenstaande is mogelijk maar is niet goed leesbaar.
Beter zo:
- `mean2 <- function(x) {
 return(mean(x))
}`
- `apply(a, 1, mean2)`



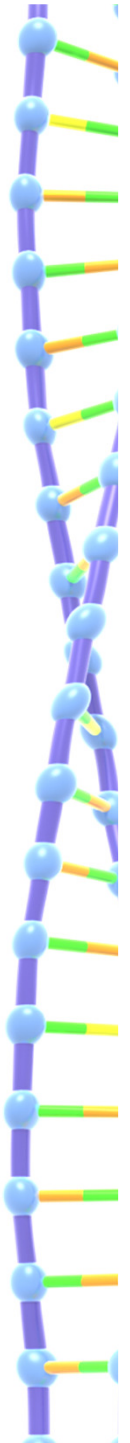
NOG EEN PAAR HANDIGE FUNCTIES

paste	Strings concatenatie
cbind	Kolommen van matrix aan elkaar plakken
rbind	Rijen van matrix aan elkaar plakken
names	Namen in list of van vector elementen
rownames	Regel namen in matrix of data frame
colnames	Kolom namen in matrix of data frame
head	Eerste deel van een variabele
length	Lengte van vector of list
str	Structuur van een object
class	Klasse van een variabele
dim	Dimensies van een array
t	Matrix transponeren



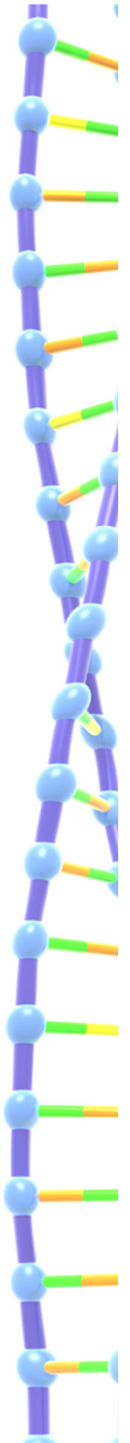
KEYWORDS/RESERVED NAMES

- De lijst met reserved names binnen R:
 - **FALSE, Inf, NA, NaN, NULL, TRUE**
 - **break, else, for, function, if, in, next, repeat, while**
- De volgende namen kun je (beter) niet gebruiken als naam voor je variabele/object:
 - **c, q, s, t, C, D, F, I, T, diff, mean, pi, range, rank, var**



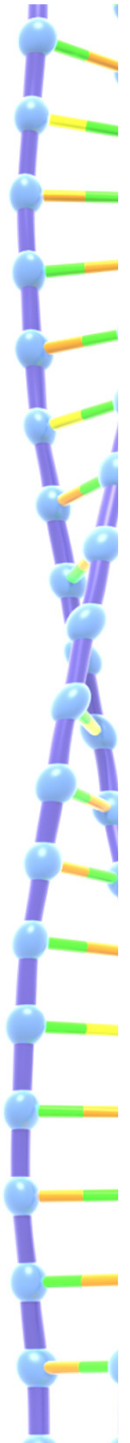
MEETSCHALEN

- Variabelen kunnen worden ingedeeld in 1 van de 4 onderstaande meetschalen
 - Nominaal
 - Ordinaal
 - Interval
 - Ratio
- R heeft functies voor verschillende meetschalen



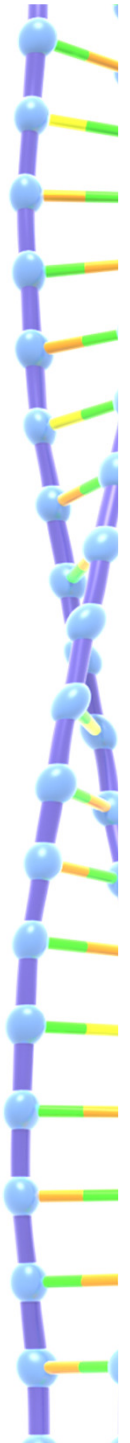
NOMINALE SCHAAL

- Kan gebruikt worden voor labels (nomen = “naam”)
- Bijvoorbeeld subtypen kanker in micro array experiment
- Alleen modus kan hierop toegepast worden
 - Meest voorkomende waarde



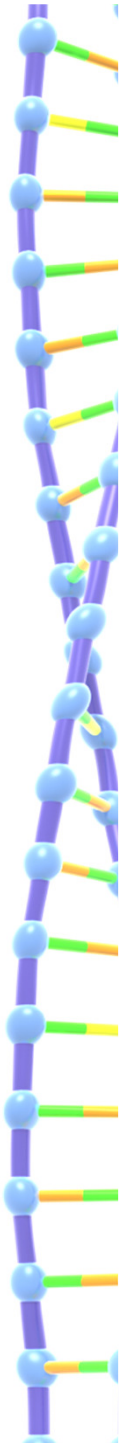
ORDINALE SCHAAL

- Ordinale variabelen beschrijven alleen een volgorde, niet hoe groot het verschil is
- Bijvoorbeeld Beste, Goed, Redelijk, Slecht
- Je kan geen waarde geven aan het verschil tussen Goed en Beste.
- Naast de modus kan je ook de mediaan uitrekenen
 - Mediaan: middelste waarde in een gesorteerde lijst



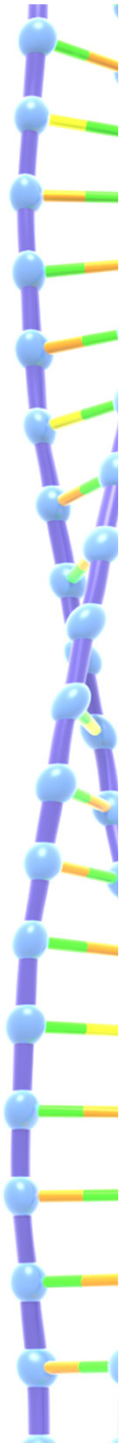
INTERVAL SCHAAL

- De verschillende waarden hebben betekenis
- Er is geen echt nulpunt
- Bijvoorbeeld °C
- Verschil tussen 90°C en 100°C is gelijk aan verschil tussen 10°C en 20°C.
- Je kan optellen en aftrekken
- Je kan ook gemiddelde en SD uitrekenen.
- Je kan niet zeggen dat 20°C twee keer zo groot is als 10°C



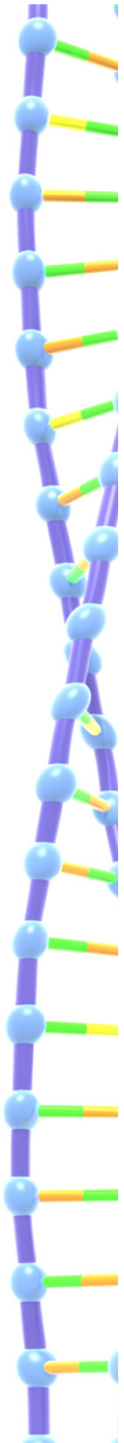
RATIO SCHAAL

- Er is wel een echt nulpunt
- Bijvoorbeeld graden Kelvin
 - Heeft echt nulpunt
 - 20K is 2 keer 10K
- Afstanden, energie, microarray meting



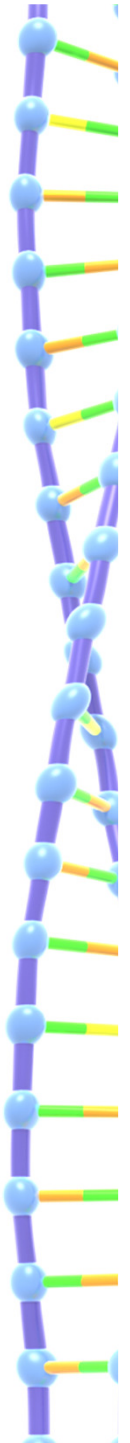
MEETSCHALEN - OVERZICHT

Niveau	Kenmerkend	Volgorde	Verschillen	Nulpunt
Nominaal	•			
Ordinaal	•	•		
Interval	•	•	•	
Ratio	•	•	•	•



FACTOR

- Factors in R kunnen gebruikt worden om variabelen in nominale of ordinale schaal op te slaan
- R kan er dan rekening mee houden dat ze in deze schalen staan
- Functies in R:
 - **factor()**
 - **as.factor()**



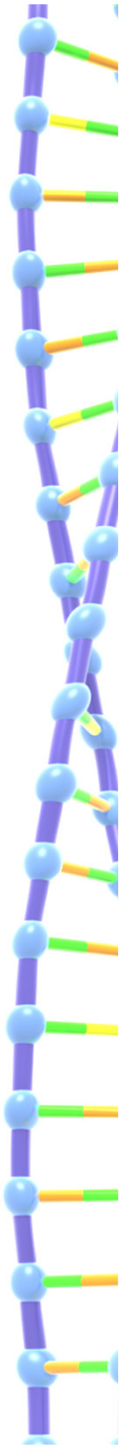
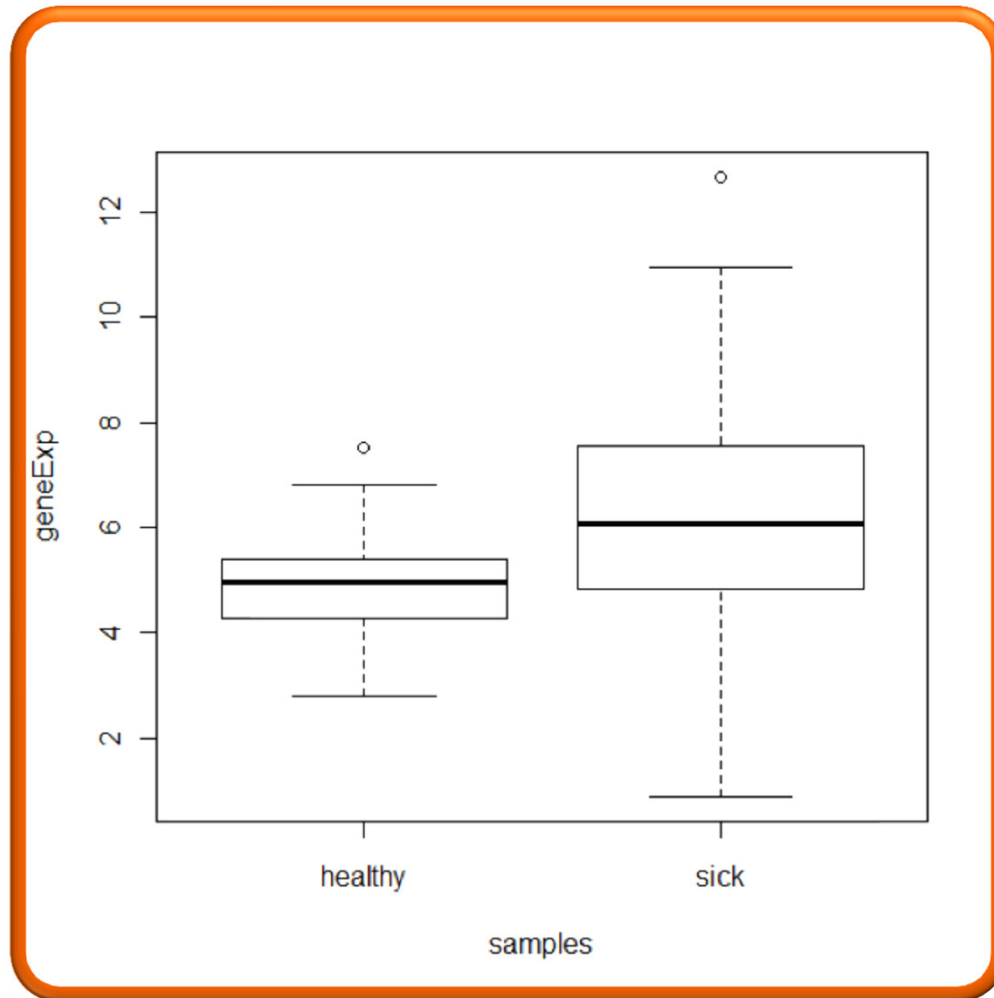
FACTOR

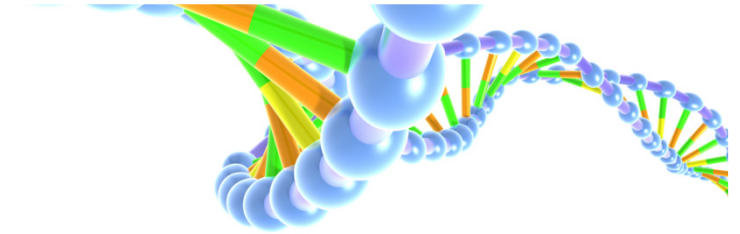
- `geneExp <- c(
 rnorm(100, mean = 5, sd = 1),
 rnorm(100, mean = 6, sd = 2)
)`
- `samples <- factor(c(
 rep("healthy", 100),
 rep("sick", 100)
))`
- `plot(geneExp ~ samples)`
- `plot(samples, geneExp)`

random getallen
 $N(\mu, \sigma)$

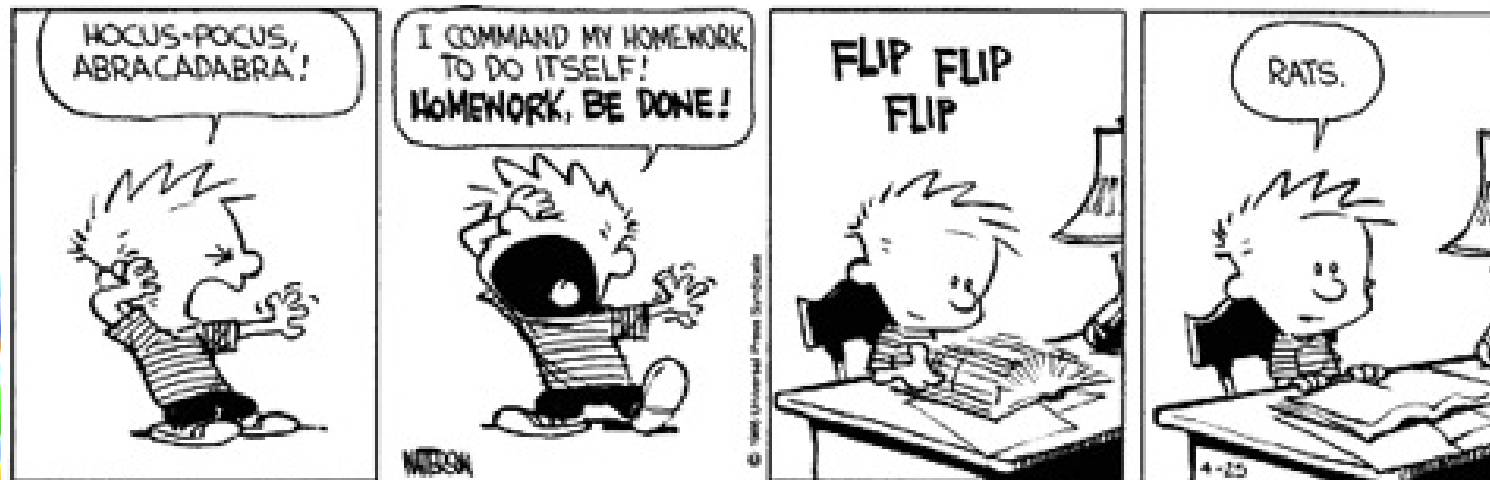
~ betekent "als
functie van"

PLOT





Jullie kunnen nu de opdrachten van les 2 maken



Hanze University Groningen
APPLIED SCIENCES

*Institute for
Life Science & Technology*