MAX PLANCK INSTITUTE FOR **BIOLOGY OF AGEING**

Introduction to R

bioinformatics@age.mpg.de

Outline

1) Introduction to R Studio
2) Overview over basic data types
3) Overview over basic functions
4) How to get help
5) Read in data
6) Basic table functions
7) Basic table manipulation
8) Loops and if queries

END OF DAY 1

R Studio

RStudio

https://rstudio.age.mpg.de

File Edit Code View Plots Session Build Debug Profile Tools Help

first_steps.R | M

Source on Save | Run | Source

```
1  # this is a script
```

1:19 (Top Level) R Script

Console ~/

```
R version 3.2.4 (2016-03-10) -- "Very Secure Dishes"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> plot(seq(1,10), seq(10,1))
> M = matrix(1,5,6)
> View(M)
> View(M)
> View(M)
>
```

Environment | History

Import Dataset | List

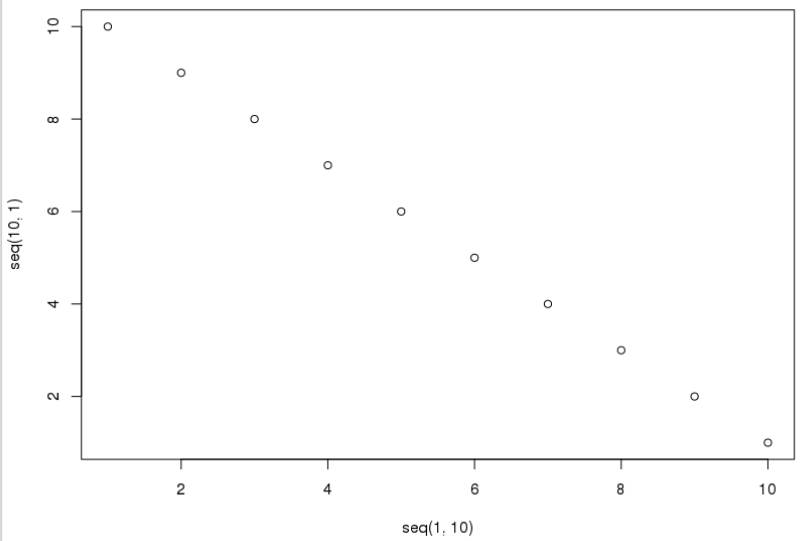
Global Environment

Data

M	num [1:5, 1:6] 1 1 1 1 1 1 1 1 1 1 ...

Files | Plots | Packages | Help | Viewer

Zoom | Export | Publish

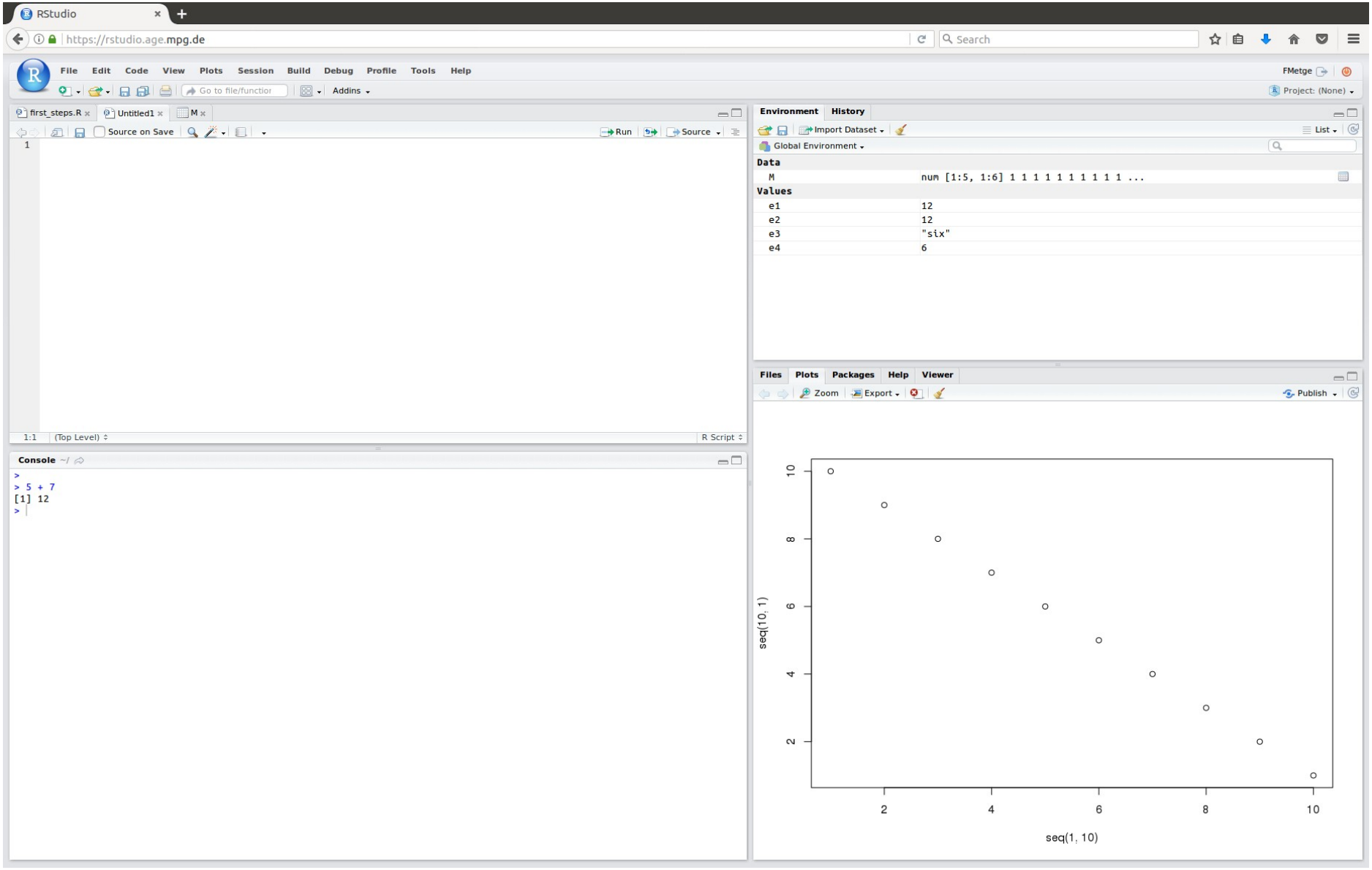


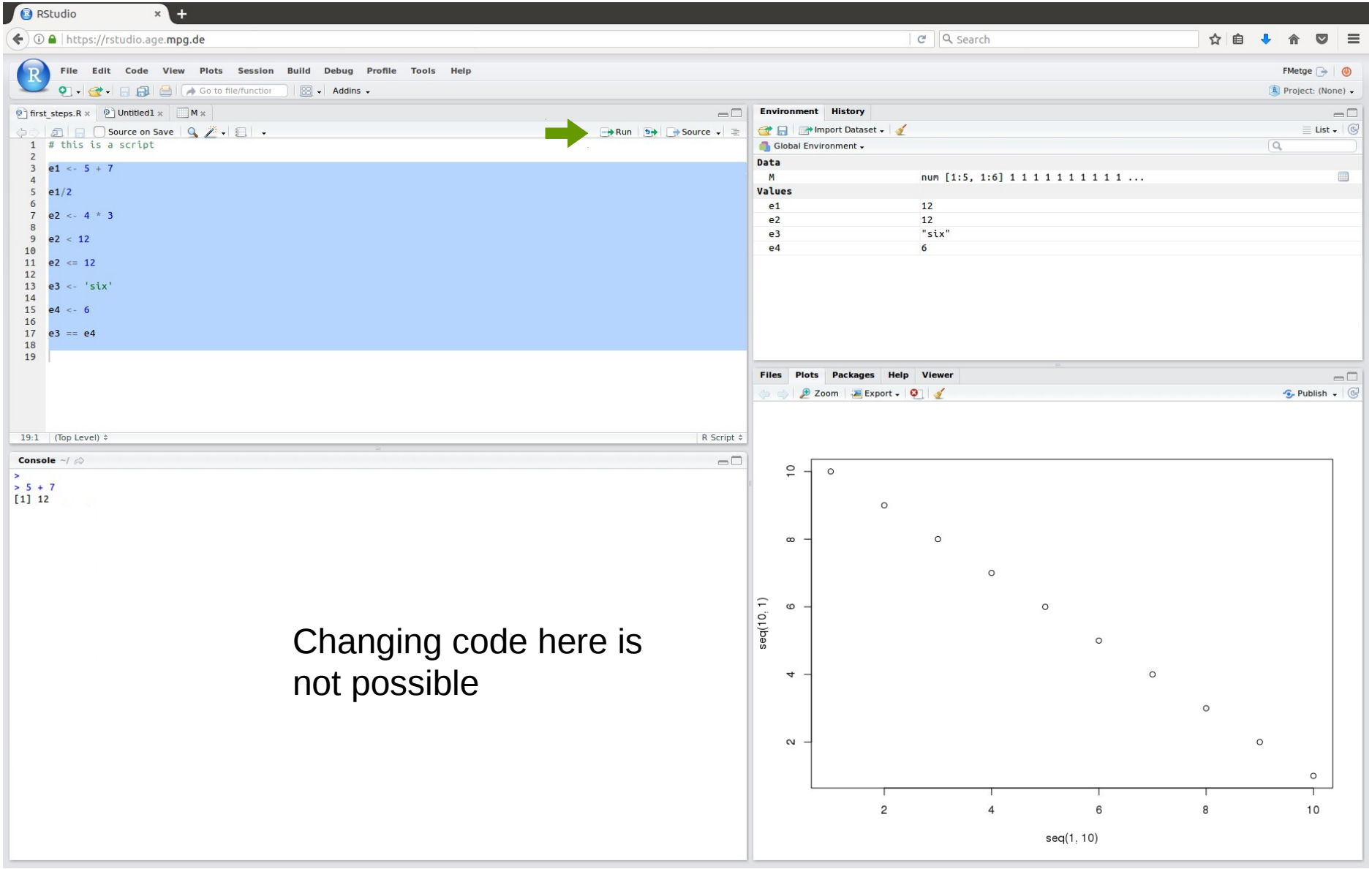
Reinforce what you have learned in the online tutorial

Basic calculations:

- Add 5 and 7
- Divide the results by 2
- Multiply 4 and 3
- Test if the result is smaller than 12

- Create one variable with six as character and one with 6 as number
- Compare if both variables are the same


RStudio
https://rstudio.age.mpg.de
Search
File Edit Code View Plots Session Build Debug Profile Tools Help
FMetge
Go to file/functior
Addins
Project: (None)
first_steps.R
Untitled1
M
Source on Save
Run
Source
1
1:1 (Top Level)
R Script
Console ~/
>
> 5 + 7
[1] 12
>
Environment History
Import Dataset
List
Global Environment
Data
M num [1:5, 1:6] 1 1 1 1 1 1 1 1 1 1 ...
Values
e1 12
e2 12
e3 "six"
e4 6
Files Plots Packages Help Viewer
Zoom
Export
Publish
seq(10, 1)
seq(1, 10)
2
4
6
8
10


RStudio
https://rstudio.age.mpg.de
Search
File Edit Code View Plots Session Build Debug Profile Tools Help
FMetge
Project: (None)
Addins
first_steps.R
Untitled1
M
Source on Save
Run
Source
1 # this is a script
2
3 e1 <- 5 + 7
4
5 e1/2
6
7 e2 <- 4 * 3
8
9 e2 < 12
10
11 e2 <= 12
12
13 e3 <- 'six'
14
15 e4 <- 6
16
17 e3 == e4
18
19
19:1 (Top Level)
R Script
Environment
History
Import Dataset
List
Global Environment
Data
M num [1:5, 1:6] 1 1 1 1 1 1 1 1 1 1 ...
Values
e1 12
e2 12
e3 "six"
e4 6
Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
Console ~/
>
> 5 + 7
[1] 12
Changing code here is not possible
seq(10, 1)
seq(1, 10)
2
4
6
8
10

RStudio

https://rstudio.age.mpg.de

File Edit Code View Plots Session Build Debug Profile Tools Help

FMetge

Addins

Project: (None)

first_steps.R | Untitled1 | M

Source on Save

Run Source

```
1  # this is a script
2
3  e1 <- 5 + 7
4
5  e1/2
6
7  e2 <- 4 * 3
8
9  # e2 < 12
10
11 e2 <= 14
12
13 e3 <- 'six'
14
15 e4 <- 6
16
17 e3 == e4
18
19
```

18:1 (Top Level) R Script

Environment History

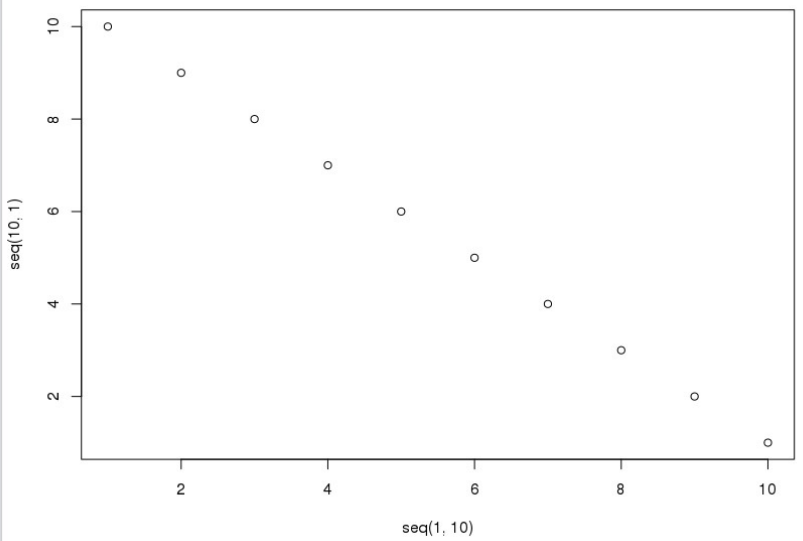
Import Dataset List

Global Environment

Data	
M	num [1:5, 1:6] 1 1 1 1 1 1 1 1 1 1 ...
Values	
e1	12
e2	12
e3	"six"
e4	6

Files Plots Packages Help Viewer

Zoom Export Publish



Console ~/

```
>
> 5 + 7
[1] 12
> e1 <- 5 + 7
>
> e1/2
[1] 6
>
> e2 <- 4 * 3
>
> e2 < 12
[1] FALSE
>
> e2 <= 12
[1] TRUE
>
> e3 <- 'six'
>
> e4 <- 6
>
> e3 == e4
[1] FALSE
>
```

RStudio

https://rstudio.age.mpg.de

File Edit Code View Plots Session Build Debug Profile Tools Help

FMetge

Addins

Project: (None)

first_steps.R | Untitled1 | M

Source on Save

Run Source

```
1  # this is a script
2
3  e1 <- 5 + 7
4
5  e1/2
6
7  e2 <- 4 * 3
8
9  # e2 < 12
10
11 e2 <= 14
12
13 e3 <- 'six'
14
15 e4 <- 6
16
17 e3 == e4
18
19
```

19:1 (Top Level) R Script

Environment History

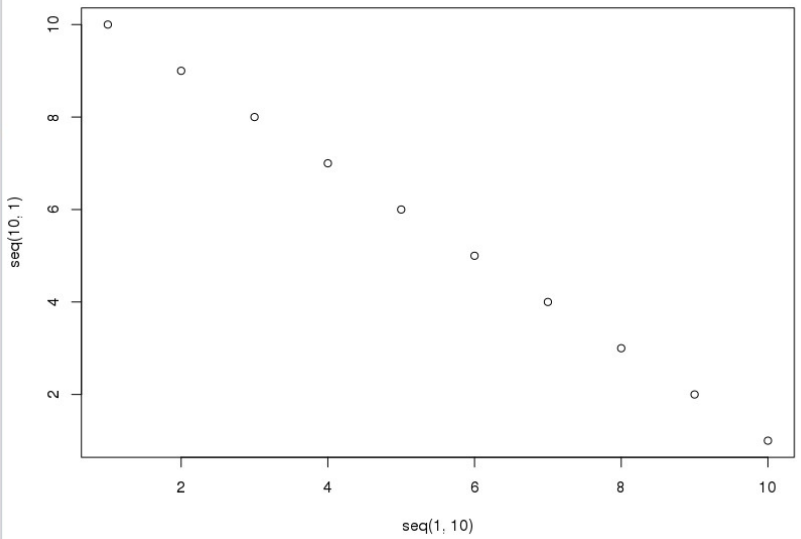
Import Dataset List

Global Environment

Data	
M	num [1:5, 1:6] 1 1 1 1 1 1 1 1 1 1 ...
Values	
e1	12
e2	12
e3	"six"
e4	6

Files Plots Packages Help Viewer

Zoom Export Publish



Console ~/

```
> e2 <- 4 * 3
>
> e2 < 12
[1] FALSE
>
> e2 <= 12
[1] TRUE
>
> e3 <- 'six'
>
> e4 <- 6
>
> e3 == e4
[1] FALSE
```

In the future, code will displayed this way

```
# this is code
> 3+7
[10]
```

Basic data types

- Numeric

```
> x <- 10.5 # or x <- 10
```

- Integer

```
> x <- as.integer(2)
```

- Complex

```
> x <- 1 + 2i
> sqrt(-1) # NaN
> sqrt(as.complex(-1)) # 0+1i
```

- Logical

```
> x <- TRUE; FALSE; T; F
```

- Character

```
> apple <- “Apple”
> x <- “6”
```

Useful functions

- Find out type of a variable

```
> class(x)
> typeof(x)
```

- Test if variable is of a certain type

```
> is.numeric(x)
> is.logical(x)
```

- Force a variable to be in a certain type

```
> as.numeric(x)
> as.character(x)
> as.integer(5.7)       # 5
> as.integer(“apple”)   # NA
> as.integer(TRUE)      # 1
```

More complex data structures (Vector)

A vector is a sequence of elements of the same basic type

```
> # numerical vector
> a <- c(2, 3, 4)

> # logical vector
> b <- c(TRUE, FALSE, TRUE)

> # character vector
> c <- c(“apple”, “six”)

> # mixed vector?
> d <- c(“six”, a, 5)

> length(d)
```

More complex data structures (Matrix)

A matrix is a collection of elements of the same data type arranged in a two-dimensional rectangular layout.

```
> # 3 columns, fill matrix column wise (default)
> A <- matrix(c(2,3,4,5,6,7), ncol = 3)

> # 2 rows, fill matrix row wise
> A <- matrix(c(2,3,4,5,6,7), nrow = 2, byrow = T)

> # alternative matrix generation
> B <- matrix(1, 2, 3)
> C <- matrix(NA, 3, 2)

> # transpose a matrix
> t(A)

> # concatenate two matrices row or column wise
> D <- cbind(A, B) # rbind(A, B)
```

More complex data structures (List)

A list is a generic vector containing objects of the same or different data types

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd")
> b = c(TRUE, FALSE, TRUE)

> # list without names
> x = list(n, s, b)

> # same list with names
> xn = list(Numbers = n, Strings = s, Boolean = b)

> # access list elements
> x[2] # vs. x[[2]]
> xn$Strings # vs. xn["Strings"] or xn[["Strings"]]
```

More complex data structures (Data Frame)

A data frame is list of vectors of equal length organized in rows and columns

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)

> # data frame witout names
> df = data.frame(n, s, b)

> # data frame with column names
> names(df) <- c("Numbers", "Strings", "Boolean")
> dfn = data.frame(Numbers = n, Strings = s, Boolean = b)

> # access elements in a data frame
> dfn$Numbers # whole first column
> dfn[,1]     # whole first column
> dfn[2,3]    # second element of the third column
```

5min Break

Basic functions

- Sum

```
> M = matrix(c(5,6,7,8), 2)
> sum(M)       # 26
> rowSums(M)   # 12 14
> colSums(M)   # 11 15
```

- Mean

```
> mean(M)      # 6.5
> rowMeans(M) # 6 7
> colMeans(M) # 5.5 7.5
```

- Variation and standard deviation

```
> var(as.numeric(M))    # 1.67
> sd(M)                 # 1.29
```

Basic functions

- Median

```
> mean(c(5,6,7,89))     # 26.75
> median(c(5,6,7,89))   # 6.5
```

- Quantiles

```
> quantile(c(5,6,7,89), probs
   = c(.25, .5, .75))
#  25%   50%   75%
# 5.75  6.50 27.50
```

Basic functions

- Sequence

```
> seq(from = 1, to = 10, by = 1)
[1] 1 2 3 4 5 6 7 8 9 10
```

- Repeat

```
> rep(x = NA, times = 3)
[1] NA NA NA
> rep(c(0,1), 5)
[1] 0 1 0 1 0 1 0 1 0 1
```

- Random Numbers

```
> rnorm(n = 5, mean = 5, sd = 1)
[1] 4.62 5.46 6.58 4.60 5.42
> runif(n = 5, min = 1, max = 5)
[1] 1.46 1.07 3.55 4.96 3.28
```

Distribution	R function
Normal	rnorm(n, mean, sd)
Uniform	runif(n, min, max)
Binomial	rbinom(n, size, prob)
Beta	rbeta(n, shape1, shape2, ncp)
Exponential	rexp(n, rate)
Poisson	rpois(n, lambda)
Chi^2	rchisq(n, df, ncp)
Student's t	rt(n, df, ncp)
...	

How to get help (inside R)

- If you have a rough idea wha

```
> ??rowmean
```

- If you know what the functio

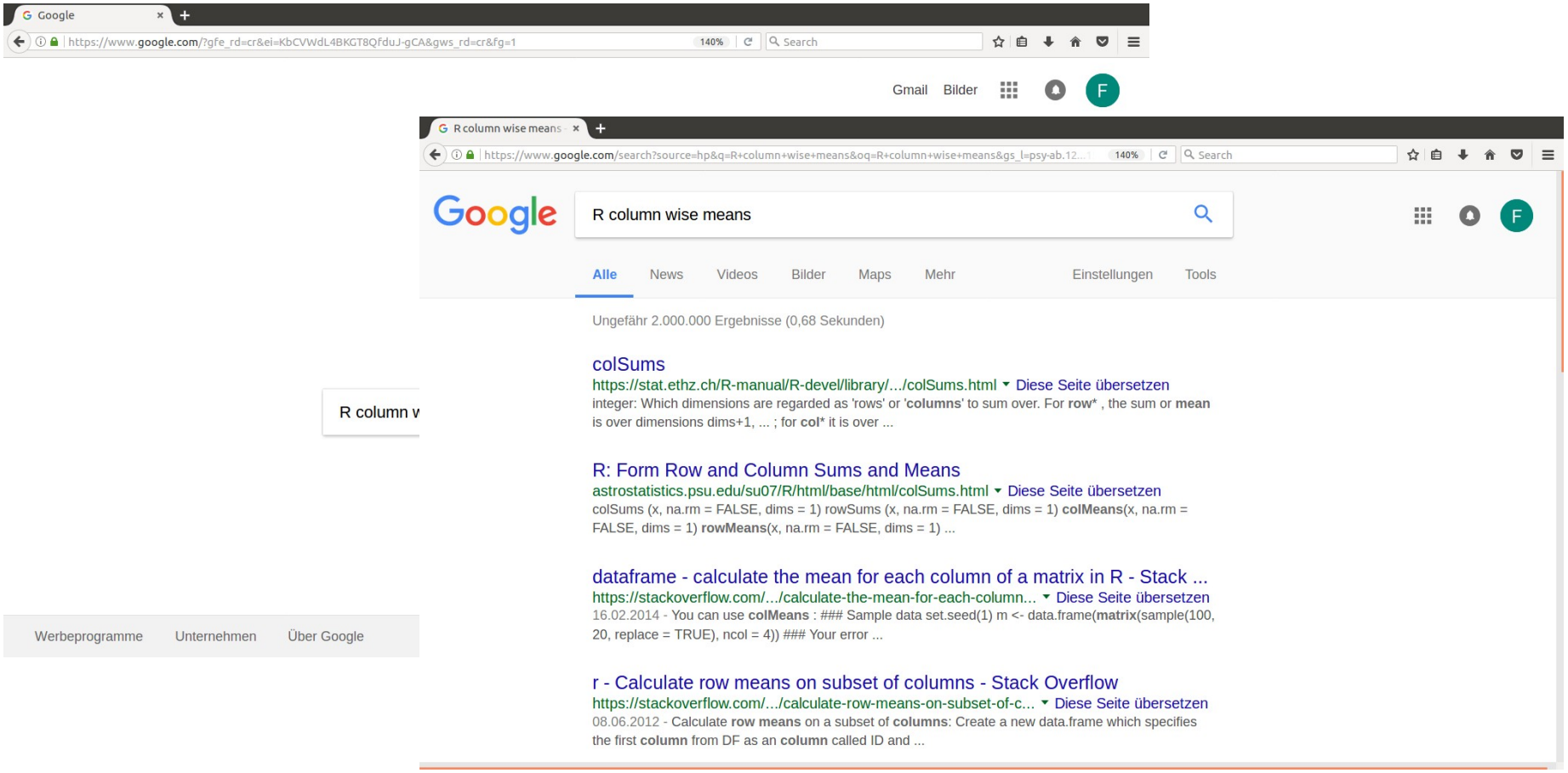
```
> ?seq
```

All R help functions are struc

Description	A short over
Usage	How to call meaningful
Arguments	A detailed d
Details	A more or le
Value	The value w
References/See Also	Citation and
Examples	Different ex

```
seq                     package:base                     R Documentation

Sequence Generation

Description:

     Generate regular sequences.  'seq' is a standard generic with a
     default method.  'seq.int' is a primitive which can be much faster
Help files with alias or concept or title matching 'rowmean' using
fuzzy matching:


base::colSums           Form Row and Column Sums and Means
  Aliases: rowMeans, .rowMeans
Matrix::colSums         Form Row and Column Sums and Means
  Aliases: rowMeans, rowMeans,diagonalMatrix-method,
    rowMeans,CsparseMatrix-method, rowMeans,TsparseMatrix-method,
    rowMeans,RsparseMatrix-method, rowMeans,dgCMatrix-method,
    rowMeans,igCMatrix-method, rowMeans,lgCMatrix-method,
    rowMeans,ngCMatrix-method, rowMeans,denseMatrix-method,
    rowMeans,ddenseMatrix-method
Matrix::dgeMatrix-class
                        Class "dgeMatrix" of Dense Numeric (S4 Class)
                        Matrices
  Aliases: rowMeans,dgeMatrix-method
Matrix::indMatrix-class
                        Index Matrices
  Aliases: rowMeans,indMatrix-method


Type '?PKG::FOO' to inspect entries 'PKG::FOO', or 'TYPE?PKG::FOO' for
entries like 'PKG::FOO-TYPE'.

~
~
~
~
(END)
     'NaN' or 'NA').

     The interpretation of the unnamed arguments of 'seq' and 'seq.int'
     is _not_ standard, and it is recommended always to name the
     arguments when programming.

     'seq' is generic, and only the default method is described here.
:
```

How to get help (outside R)



stackoverflow.com

5min Break

Read in data

Table

```
> D = read.table(“mouse.tab”, as.is = T, header = F )
```

CSV

```
> D = read.csv(“mouse.csv”, as.is = T, header = T)
> # default separator “,”, decimals as “.”
```

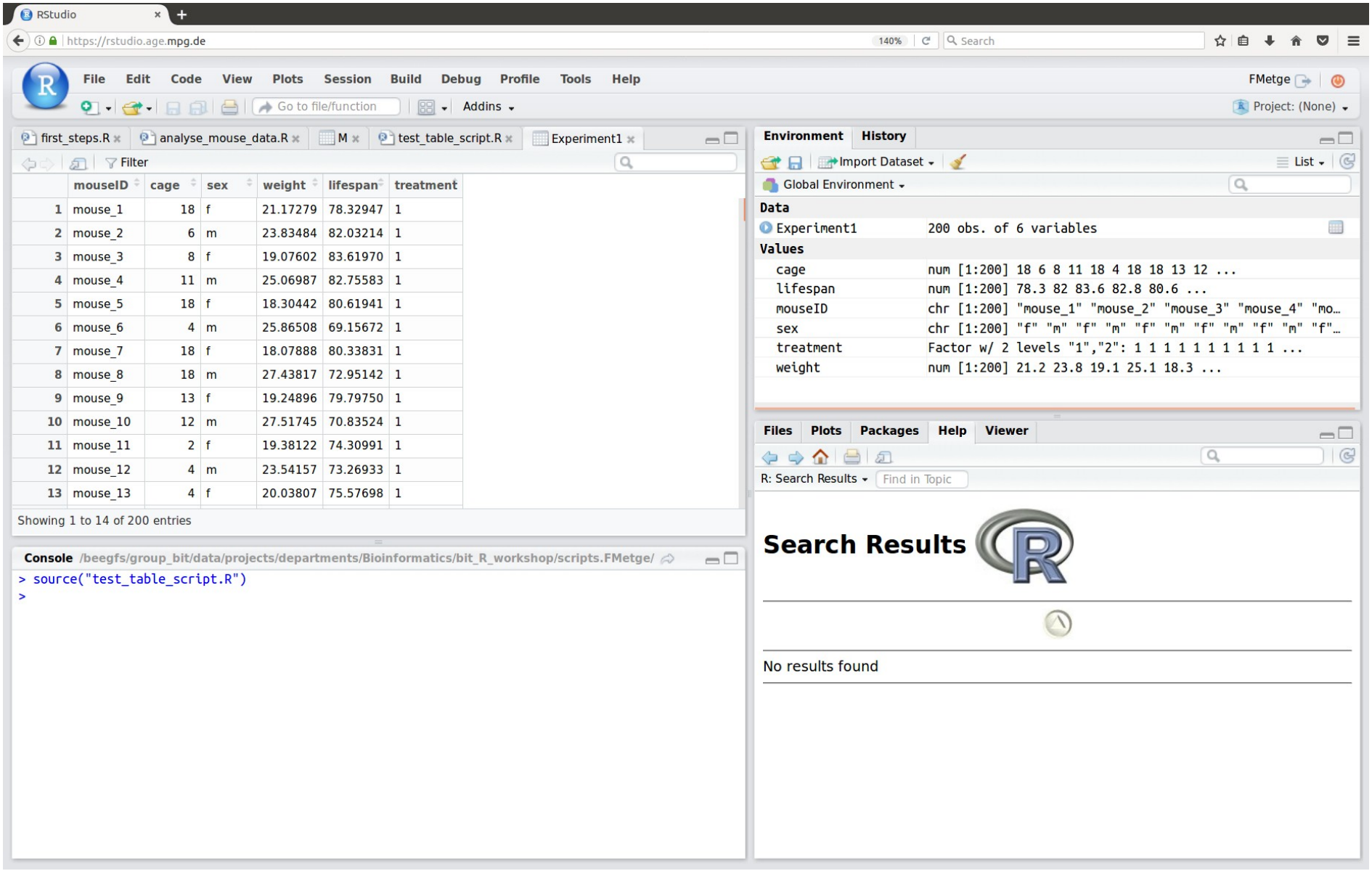
EXCEL

- Not in the standard library, but various packages are available

```
> library(xlsx)
> D = read.xlsx(“mouse.xlsx”, sheetName = “first”)
```

script

```
> source(“test_table_script.R”)
> ls()
```


RStudio
https://rstudio.age.mpg.de
140%
Search
File Edit Code View Plots Session Build Debug Profile Tools Help
FMetge
Go to file/function
Addins
Project: (None)
first_steps.R
analyse_mouse_data.R
M
test_table_script.R
Experiment1
Filter
mouseID cage sex weight lifespan treatment
1 mouse_1 18 f 21.17279 78.32947 1
2 mouse_2 6 m 23.83484 82.03214 1
3 mouse_3 8 f 19.07602 83.61970 1
4 mouse_4 11 m 25.06987 82.75583 1
5 mouse_5 18 f 18.30442 80.61941 1
6 mouse_6 4 m 25.86508 69.15672 1
7 mouse_7 18 f 18.07888 80.33831 1
8 mouse_8 18 m 27.43817 72.95142 1
9 mouse_9 13 f 19.24896 79.79750 1
10 mouse_10 12 m 27.51745 70.83524 1
11 mouse_11 2 f 19.38122 74.30991 1
12 mouse_12 4 m 23.54157 73.26933 1
13 mouse_13 4 f 20.03807 75.57698 1
Showing 1 to 14 of 200 entries
Console /beegfs/group_bit/data/projects/departments/Bioinformatics/bit_R_workshop/scripts.FMetge/
> source("test_table_script.R")
>
Environment History
Import Dataset
List
Global Environment
Data
Experiment1 200 obs. of 6 variables
Values
cage num [1:200] 18 6 8 11 18 4 18 18 13 12 ...
lifespan num [1:200] 78.3 82 83.6 82.8 80.6 ...
mouseID chr [1:200] "mouse_1" "mouse_2" "mouse_3" "mouse_4" "mo…
sex chr [1:200] "f" "m" "f" "m" "f" "m" "f" "m" "f" "m" "f"…
treatment Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
weight num [1:200] 21.2 23.8 19.1 25.1 18.3 ...
Files Plots Packages Help Viewer
R: Search Results
Find in Topic
Search Results
No results found

Basic table functions

Summary

```
> summary(Experiment1)
       mouseID         cage       sex         weight          lifespan     treatment
 mouse_1  :  1   Min.   : 1.00   f:100   Min.   :17.48   Min.   :60.46   1:100
 mouse_10 :  1   1st Qu.: 6.00   m:100   1st Qu.:21.41   1st Qu.:71.59   2:100
 mouse_100:  1   Median :10.00           Median :24.50   Median :76.17
 mouse_101:  1   Mean   :10.36           Mean   :24.53   Mean   :75.38
 mouse_102:  1   3rd Qu.:15.00           3rd Qu.:27.06   3rd Qu.:79.40
 mouse_103:  1   Max.   :19.00           Max.   :34.01   Max.   :93.09
 (Other)  :194
```

Table

```
> table(Experiment1$cage)

 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 4 11  7 11 10  8 13 14 10 16  8 11 13 13 13  8 11 13  6
```

Basic table functions

Correlation

```
> cor(Experiment1$cage, Experiment1$weight,
        method = 'spearman')
[1] 0.01028674
> cor(Experiment1$lifespan, Experiment1$weight)
[1] -0.6827497
```

Wilcoxon Test

```
> wilcox.test(Experiment1$lifespan ~ Experiment1$treatment)

    Wilcoxon rank sum test with continuity correction

data:  Experiment1$lifespan by Experiment1$treatment
W = 7296, p-value = 2.037e-08
alternative hypothesis: true location shift is not equal to 0
```

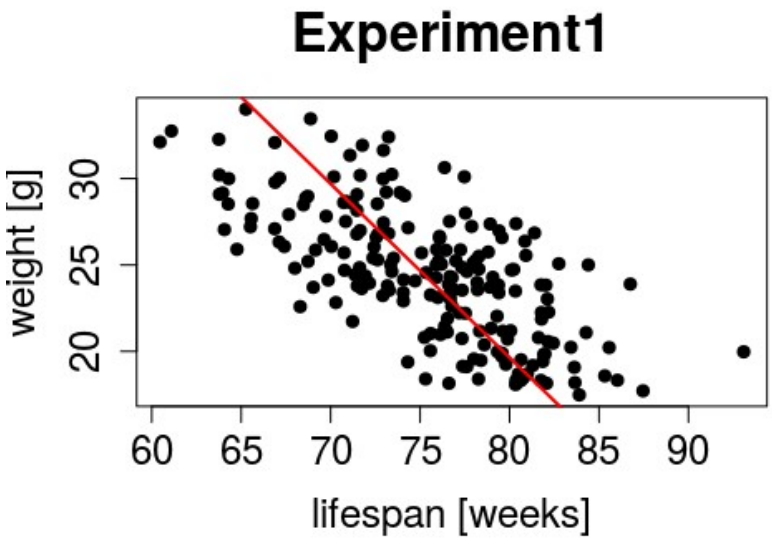
Basic table functions

Because both groups are normally distributed we can use a t-test

```
> t.test(Experiment1$lifespan ~ Experiment1$treatment)

    Welch Two Sample t-test

data:  Experiment1$lifespan by Experiment1$treatment
t = 6.0342, df = 193.78, p-value = 7.972e-09
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 2.968538 5.851315
sample estimates:
mean in group 1 mean in group 2
       77.58123        73.17130
```

Basic Table functions

```
> plot(Experiment1$lifespan,
       Experiment1$weight)

> L = lm(Experiment1$lifespan
       ~ Experiment1$weight)

> L

Call:
lm(formula = Experiment1$lifespan
       ~ Experiment1$weight)

Coefficients:
       (Intercept)  Experiment1$weight
           100.028              -1.005

> abline(L)
```



Basic table manipulations

You have a second treatment you want to add to the bottom of the existing data frame (rbind)

```
> Experiment <- rbind(Experiment1, Treatment3)
```

You have the body size of each mouse and you want to add it to the left of the exisiting data frame (cbind)

```
> ExperimentS <- cbind(Experiment, size)
```

Basic table manipulations

Merge

```
> ExperimentP <- merge(ExperimentS, Exp2,
    by.x = 'mouseID', by.y = 'mouseID_pupps', all = T)
```

Subset

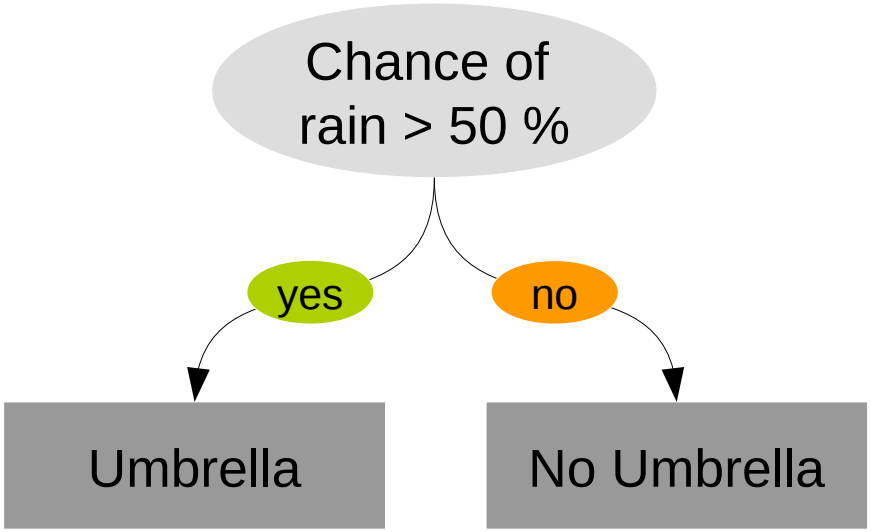
```
> Exp_cage10 <- subset(ExperimentP, cage == 10)
> Exp_cage10_11 <- subset(ExperimentP,
    cage %in% c(10, 11))[,1:8]
```

Names

```
> names(Exp_cage10_11)
[1] "mouseID" "cage" "sex.x" "weight" "lifespan" "treatment" "size" "num_pupps"

> names(Exp_cage10_11)[c(3, 9)] <- "sex"
```

5min Break

Conditional procedures




```
> chance_of_rain = 0.3

> if(chance_of_rain > 0.5){
    print(“take umbrella”)
} else {
    print(“you don’t need an
           umbrella”)
}

[1] “you don’t need an
      umbrella”
```

if(condition = True){

- do something

} else {

- do something different

}

if / else if / else

```
> x = 5
> y = NA
> z = 'six'

if(x == 5){
  print("x is equal to 5")
}

if(x < 5){
  print('x is smaller than 5')
}else{
  print('x is equal or larger than 5')
}
```

if / else if / else

```
> x = 5
> y = NA
> z = 'six'

> if(x < 5){
+  print("x is smaller than 5")
+ }else if(x == 5){
+   print("x is equal to 5")
+ }else{
+   print("x is larger than 5")
+ }
```

if / else if / else

```
> x = 5
> y = NA
> z = 'six'


> if(is.na(y)){
+  print("y is not available")
+ }

> if(!is.character(z)){
+  print("z is not a character")
+ } else {
+  print("z is a character")
+ }
```

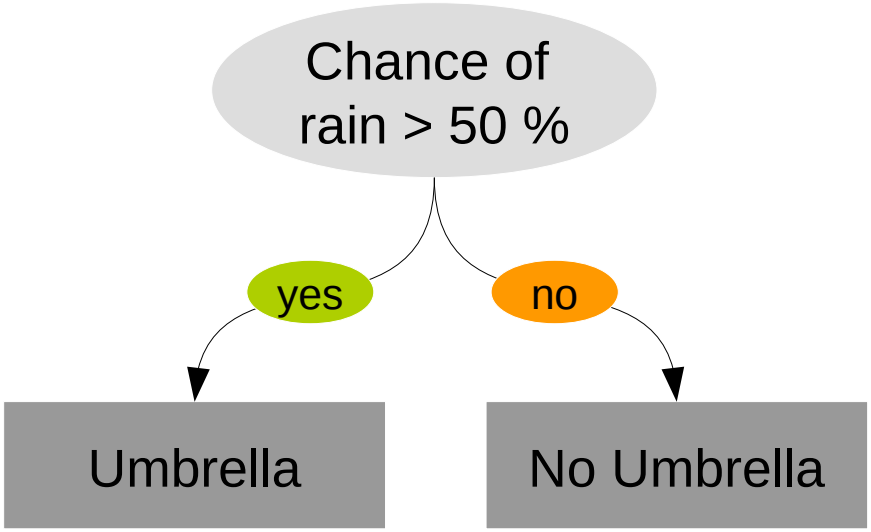
if / else if / else

```
> a = c(1,2,3)
> b = c(1,2,3)
> c = c(1,3)


> if(a == b){
+   print(paste("a and b agree on", sum(a == b),
+   "elements", sep = ' '))
+ }
[1]    "a and b agree on 3 elements" # (warning)

> if(a == c){
+   print(paste("a and c agree on", sum(a == c),
+   "elements", sep = ' '))
+ }
[1]    ERROR
```

if / else if / else

```
> a = c(1,2,3)
> b = c(1,2,3)
> c = c(1,3)

> if(length(a) == length(c) & a == c){
+  print(paste("a and c agree on", sum(a == c),
+  "elements", sep = ' '))
+} else {
+  print("a and c cannot be compared, they differ in length")
+}

> if(length(a) == length(c) && a == c){
+  print(paste("a and c agree on", sum(a == c),
"elements", sep = ' '))
+ } else {
+  print("a and c cannot be compared, they differ in length")
+}
```

Loops


Chance of rain > 50 %
yes
no
Umbrella
No Umbrella


Köln
Mittwoch, 15:00
Überwiegend sonnig
28 °C | °F
Niederschlag: 0%
Luftfeuchte: 39%
Wind: 16 km/h
Temperatur
Niederschlag
Wind
0% 0% 15% 23% 5% 6% 3%
16:00 19:00 22:00 01:00 04:00 07:00 10:00 13:00
Mi. 28° 15°
Do. 24° 12°
Fr. 26° 17°
Sa. 27° 16°
So. 24° 16°
Mo. 25° 16°
Di. 25° 14°
Mi. 22° 14°

Loops

Loops are useful if you want to repeat the same command over and over again

Different type of loops

While loop

- Repeat while condition is true

For loop

- Iterate over index
- No conditional statement

Most procedures can be written as for or while loops. For loops are almost always the saver choice

Day	1	2	3	4	5	6	7	8
Chance of rain [%]	0	0	15	23	5	6	3	0

For-loop

```
> for(i in something){
+ do something
+ }
```

Use if you have a fixed number of iterations

Will almost always finish

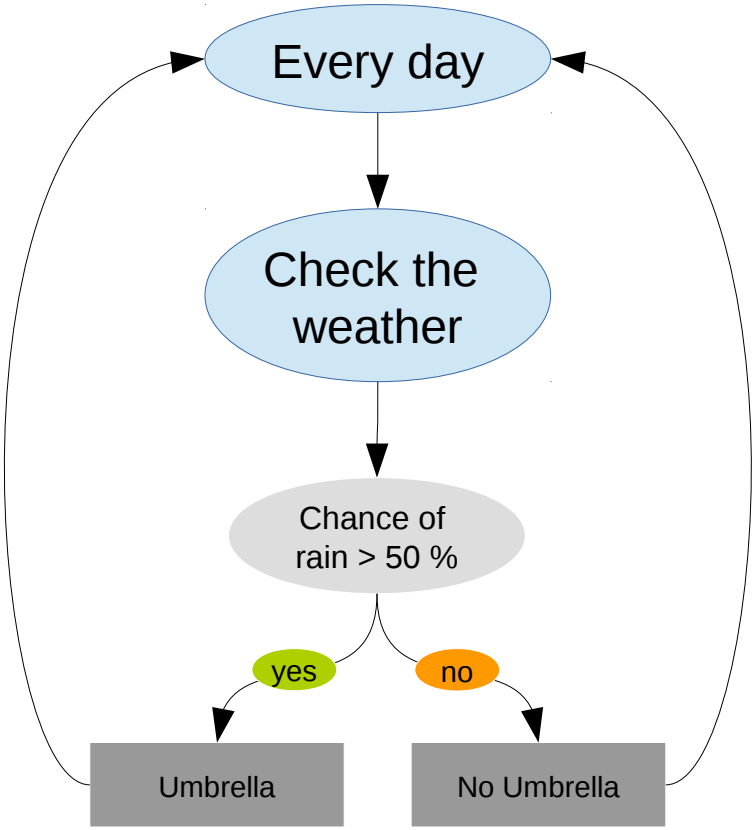
While-loop

```
> while(condition = TRUE){
+ do something
+ }
```

Use if you do not know how often you need to repeat something

Could run infinitely

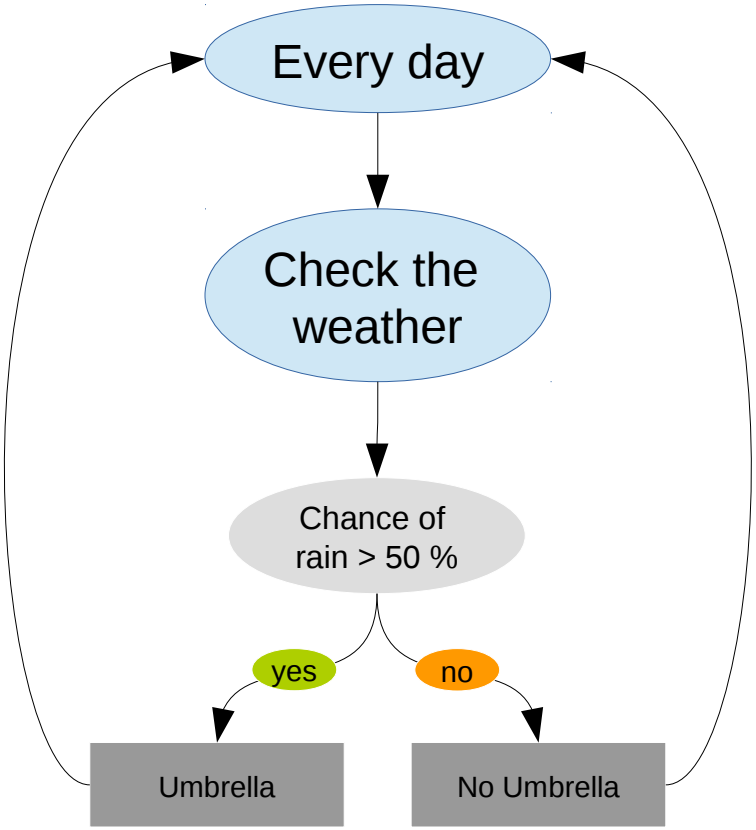
While - loops



day	1	2	3	4	5	6	7	8
check_ weather	0	0	15	23	5	6	3	0

```
> day = 1
> while(day < 9){
+ chance_of_rain =
+          check_weather(day)
+
+ if(chance_of_rain > 10){
+   print(“take umbrella”)
+ } else {
+   print(“you don’t need an
+          umbrella”)
+ }
+ day = day + 1
+ }

[1] “you don’t need an
       umbrella”
…
[8] “you don’t need an
       umbrella”
```

For - loops



day	1	2	3	4	5	6	7	8
check_ weather	0	0	15	23	5	6	3	0

```
>
> for(day in 1:8){
+ chance_of_rain =
+          check_weather(day)
+
+ if(chance_of_rain > 10){
+   print(“take umbrella”)
+ } else {
+   print(“you don’t need an
+          umbrella”)
+ }
+
+ }

[1] “you don’t need an
       umbrella”

…
[8] “you don’t need an
       umbrella”
```

More examples

for

```
> for(x in 2:4){
+ print(c(x, x * x))
+ }

[1] 2 4
[1] 3 9
[1]  4 16
```

while

```
> x = 1
> while(x < 4){
+ x <- x + 1
+ print(c(x,x * x))
+ }

[1] 2 4
[1] 3 9
[1]  4 16

> x = 1
> while(x < 4){
+ print(c(x,x * x))
+ x <- x + 1
+ }
```

Exit a loop

```
> for(x in 2:4){
+   if(x == 3){
+       break
+   } else {
+   print(x)
+   }
+ }

[1] 2
```

```
> x = 1
> while(x < 5){
+   x = x + 1
+   if(x == 3){
+       break
+   } else {
+   print(x)
+   }
+ }

[1] 2
```

Skip one iteration in a loop

```
> for(x in 2:4){
+   if(x == 3){
+       next
+   } else {
+   print(x)
+   }
+ }

[1] 2
[1] 4
[1] 5
```

```
> x = 1
> while(x < 5){
+   x = x + 1
+   if(x == 3){
+       next
+   } else {
+   print(x)
+   }
+ }

[1] 2
[1] 4
[1] 5
```

Example

You have several items you want to pack

Each Item has a weight

You are only allowed to pack 10kg
because you are flying Ryanair

Problem:

Add Items to your bag until you reached
the maximum weight

```
> items = c('shoes', 'shirt',
+   'pants', 'underwear',
+   'book', 'tooth brush',
+   'pillow', 'head phones',
+   'hair dryer')

> sizes = c(2.4, 2, 3.1, 1.5,
      1.1, 0.3, 0.8, 0.5, 1)

> bag = 0
```

Example

```
> bag = 0
> things_packed = character(0)
> for(i in 1:length(items)){
+   if(bag + sizes[i] > 10){
+     break
+   }
+   bag = bag + sizes[i]
+   things_packed =
+   c(things_packed, items[i])
+   print(c(bag, things_packed))
+ }

[1] "2.4" "shoes"
[1] "4.4" "shoes" "shirt"
[1] "7.5" "shoes" "shirt" "pants"
[1] "9" "shoes" "shirt" "pants" "underwear"
```

```
> i = 1
> bag = 0
> things_packed = character(0)
> while( bag + sizes[i] < 10){
+   bag = bag + sizes[i]
+   things_packed =
+   c(things_packed, items[i])
+   i = i + 1
+   print(c(bag, things_packed))
+ }

[1] "2.4" "shoes"
[1] "4.4" "shoes" "shirt"
[1] "7.5" "shoes" "shirt" "pants"
[1] "9" "shoes" "shirt" "pants" "underwear"
```

Example improved

```
> bag = 0
> things_packed = character(0)
> for(i in 1:length(items)){
+   if(bag + sizes[i] > 10){
+     next
+   }
+   bag = bag + sizes[i]
+   things_packed =
+   c(things_packed, items[i])
+   print(c(bag, things_packed))
+ }


[1] "2.4" "shoes"
[1] "4.4" "shoes" "shirt"
[1] "7.5" "shoes" "shirt" "pants"
[1] "9" "shoes" "shirt" "pants" "underwear"
[1] "9.3" "shoes" "shirt" "pants" "underwear" "tooth brush"
[1] "9.8" "shoes" "shirt" "pants" "underwear" "tooth brush"
       "head phones"
```

```
> i = 1
> bag = 0
> things_packed = character(0)
> while( bag + sizes[i] < 10){
+   if(bag + sizes[i] > 10){
+     next
+   }
+   bag = bag + sizes[i]
+   things_packed =
+   c(things_packed, items[i])
+   i = i + 1
+   print(c(bag, things_packed))
+ }

[1] "2.4" "shoes"
[1] "4.4" "shoes" "shirt"
[1] "7.5" "shoes" "shirt" "pants"
[1] "9" "shoes" "shirt" "pants" "underwear"

# infinite - BAD !!!!!
```

Example while loop corrected

```
> i = 1
> bag = 0
> things_packed = character(0)
> while( bag + sizes[i] < 10){
+   if(bag + sizes[i] > 10){
+     i = i + 1
+     next
+   }
+   bag = bag + sizes[i]
+   things_packed =
+   c(things_packed, items[i])
+   i = i + 1
+   print(c(bag, things_packed))
+ }


[1] "2.4" "shoes"
[1] "4.4" "shoes" "shirt"
[1] "7.5" "shoes" "shirt" "pants"
[1] "9" "shoes" "shirt" "pants" "underwear"
[1] "9.3" "shoes" "shirt" "pants" "underwear" "tooth brush"
[1] "9.8" "shoes" "shirt" "pants" "underwear" "tooth brush"
      "head phones"

# exits on an ERROR (ran out of the array)
```

```
> i = 1
> bag = 0
> things_packed = character(0)
> while( bag < 10 &
+           i <= length(sizes)){
+   if(bag + sizes[i] > 10){
+     i = i + 1
+     next
+   }
+   bag = bag + sizes[i]
+   things_packed =
+   c(things_packed, items[i])
+   i = i + 1
+   print(c(bag, things_packed))
+ }


[1] "2.4" "shoes"
[1] "4.4" "shoes" "shirt"
[1] "7.5" "shoes" "shirt" "pants"
[1] "9" "shoes" "shirt" "pants" "underwear"
[1] "9.3" "shoes" "shirt" "pants" "underwear" "tooth brush"
[1] "9.8" "shoes" "shirt" "pants" "underwear" "tooth brush"
      "head phones"
```

Loops for real application

gene	R1	R2	R3	R4	R5	R6	Diff?
AA	10	10	6	9	7	11	
BB	11	12	11	11	13	9	
CC	10	12	9	11	9	10	
DD	11	6	12	28	20	17	
EE	12	5	9	10	10	11	

You want to know which genes are differentially expressed

Perform the T-test **for** each gene

```
> gene[1,9] = t.test(gene[1,2:4], gene[1,5:7])$p.value
> gene[2,9] = t.test(gene[2,2:4], gene[2,5:7])$p.value
> gene[3,9] = t.test(gene[3,2:4], gene[3,5:7])$p.value
> gene[4,9] = t.test(gene[4,2:4], gene[4,5:7])$p.value
…
> gene[8,9] = t.test(gene[8,2:4], gene[8,5:7])$p.value
> gene[9,9] = t.test(gene[9,2:4], gene[9,5:7])$p.value
```

Loops for real application

```
> gene<- read.csv("gene_exp1.csv")
> pvals = numeric(0)
> for(i in 1:9){
+   pvals[i] <- t.test(gene[i,2:4], gene[i,5:7])$p.value
+ }
> gene$Diff <- pvals < 0.05
> gene

  gene R1 R2 R3 R4 R5 R6  Diff
1   AA 10 10  6  9  7 11 FALSE
2   BB 11 12 11 11 13  9 FALSE
3   CC 10 12  9 11  9 10 FALSE
4   DD 11  6 12 28 20 17  TRUE
5   EE 12  5  9 10 10 11 FALSE
6   FF 11  8 12  1  3  4  TRUE
7   GG 23 12 11 11  9 11 FALSE
8   HH  6  7  8  8 11  8 FALSE
9   II 11  8 13 21 18 20  TRUE
```

Exercise

Load the table generated in “create_gene_expression_table.R” using source()

Return all genes which are significant differentially expressed

Use the t.test()$p.value and a for loop

Use Experiment_DEG, group 1 > Rep1 – 4, group2 > 5 - 8

END OF DAY 1

MAX PLANCK INSTITUTE FOR **BIOLOGY OF AGEING**

Introduction to R

bioinformatics@age.mpg.de

Outline

1) Loops (Solution to yesterdays exercise)
2) Apply function
3) Write your own function
4) Basic plotting
5) Installing R packages (CRAN, Bioconductor)
6) Step by step DEG analysis for home

END OF DAY 2

Loops for real application

```
> gene<- read.csv("gene_exp1.csv")
> pvals = numeric(0)
> for(i in 1:9){
+   pvals[i] <- t.test(gene[i,2:4], gene[i,5:7])$p.value
+ }
> gene$Diff <- pvals < 0.05
> gene

  gene R1 R2 R3 R4 R5 R6  Diff
1   AA 10 10  6  9  7 11 FALSE
2   BB 11 12 11 11 13  9 FALSE
3   CC 10 12  9 11  9 10 FALSE
4   DD 11  6 12 28 20 17  TRUE
5   EE 12  5  9 10 10 11 FALSE
6   FF 11  8 12  1  3  4  TRUE
7   GG 23 12 11 11  9 11 FALSE
8   HH  6  7  8  8 11  8 FALSE
9   II 11  8 13 21 18 20  TRUE
```

Exercise

Load the table generated in “create_gene_expression_table.R” using source()

Return the number of genes which are significant differentially expressed

Use the t.test()$p.value and a for loop

Calculate the t-test for each gene using a for loop

```
>  head(Experiment_DEG)
     gene_id   Rep1   Rep2   Rep3   Rep4   Rep5   Rep6   Rep7   Rep8
  1  gene_1 105.39  94.43 116.86 141.04  93.34 115.71 113.25  82.68
  2  gene_2  87.40 142.94  51.73  94.68  70.89 115.54 102.72 104.11
  3  gene_3 117.37  79.42 121.76  79.98 108.93 112.08  76.63  87.15
  4  gene_4 134.54  77.28 126.44  91.36  78.83 104.22 105.84 115.79
  5  gene_5 100.48 143.68 105.27 104.19 116.44  86.16 107.04 110.42
  6  gene_6 107.36 118.16 108.50  84.17 109.03 103.13 111.16  69.55
```

Calculate the t-test for each gene using a for loop

```
> source('create_gene_expression_table.R')
>
> # calculate the t-test for each gene
>
> # 1. using a for loop
>
> pvals = numeric(10000)
>
> for(i in 1:10000){
+  pvals[i] = as.numeric(t.test(Experiment_DEG[i, 2:5],
+         Experiment_DEG[i, 6:9])$p.value)
+ }
>
> length(subset(Experiment_DEG, pvals < 0.05)$gene_id)

[1] 1424
```

Takes a rather long time → apply function

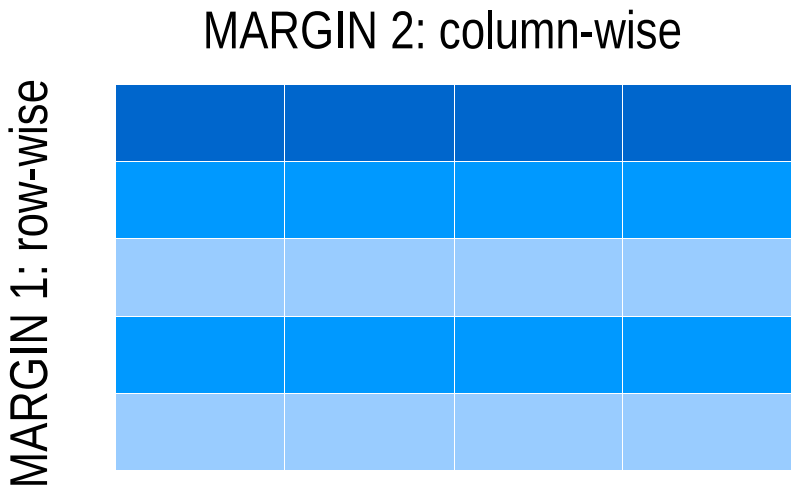
Calculate the t-test for each gene using apply

```
> # 2. using apply
>
> pvals_a = apply(Experiment_DEG[,2:9], 1,
+   function(x) {as.numeric(t.test(x[1:4], x[5:8])$p.value)})
>
> length(subset(Experiment_DEG, pvals_a< 0.05)$gene_id)

[1] 1424
```

Apply: use instead of a loop (if you loop over matrix, data.frame)

apply(X, MARGIN, FUN…)

FUN: standard function, or your own function



Apply

Obtain time spent running for/apply using systems.time()

```
> system.time(
+  for(i in 1:10000){
+   pvals[i] = as.numeric(t.test(Experiment_DEG[i, 2:5],
+       Experiment_DEG[i, 6:9])$p.value)
+   }
+ )

[1]   user  system elapsed
    11.040   0.000  11.726
```

```
> system.time(
+   apply(Experiment_DEG[,2:9], 1,
+       function(x) {as.numeric(t.test(x[1:4], x[5:8])$p.value)})
+ )

[1]   user  system elapsed
     3.331   0.000   3.152
```

→ much faster than for loop

More apply examples

```
>  head(Experiment_DEG)
     gene_id   Rep1   Rep2   Rep3   Rep4   Rep5   Rep6   Rep7   Rep8
1  gene_1 105.39  94.43 116.86 141.04  93.34 115.71 113.25  82.68
2  gene_2  87.40 142.94  51.73  94.68  70.89 115.54 102.72 104.11
3  gene_3 117.37  79.42 121.76  79.98 108.93 112.08  76.63  87.15
```

Apply instead of looping over a matrix

```
> # row wise variance
> apply(Experiment_DEG[,2:9], 1, var)

> # column wise variance
> apply(Experiment_DEG[,2:9], 2, var)
```

Three ways to calculate the row wise mean

```
> for(i in 1:10000){
+   mean(Experiment_DEG[i,2:9])})        # elapsed: 2.381 seconds
> apply(Experiment_DEG[,2:9], 1, mean)  # elapsed: 0.097 seconds

> # bonus, built in functions
> rowMeans(Experiment_DEG[,2:9])        # elapsed: 0.001 seconds
```

Versions of apply

```
> ??apply
```

base::apply	**Apply Functions Over Array Margins**
base::by	Apply a Function to a Data Frame Split by Factors
base::eapply	Apply a Function Over Values in an Environment
base::lapply	**Apply a Function over a List or Vector**
base::mapply	Apply a Function to Multiple List or Vector Arguments
base::rapply	Recursively Apply a Function to a List
base::.subset	Internal Objects in Package 'base'
base::tapply	**Apply a Function Over a Ragged Array**

Versions of apply

lapply(X, FUN, ...)

```
> l <- list(a = 1:10, b = 11:20)
> lapply(l, mean)   # returns a list
    $a
    [1] 5.5
    $b
    [1] 15.5
```

sapply(X, FUN, ...)

```
> l <- list(a = 1:10, b = 11:20)
> sapply(l, mean)   # returns a vector
      a    b
    5.5 15.5
```

Versions of apply

tapply(X, INDEX, FUN ...)

```
> attach(iris); head(iris)
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
 1          5.1         3.5          1.4         0.2  setosa
 2          4.9         3.0          1.4         0.2  setosa
 3          4.7         3.2          1.3         0.2  versicolor
 4          4.6         3.1          1.5         0.2  versicolor
 5          5.0         3.6          1.4         0.2  virginica
 6          5.4         3.9          1.7         0.4  virginica
> tapply(iris$Petal.Length, Species, mean)
    setosa versicolor  virginica
     1.462      4.260      5.552

> by(iris$Petal.Length, Species, mean)
Species: setosa
[1] 1.462
------------------------------------------
Species: versicolor
[1] 4.26
------------------------------------------
Species: virginica
[1] 5.552
```

5min Break

Write your own function

When to write a function

- If you repeat the same procedure with minor changes or on different data

How is the basic structure

```
> myFunction <- function(arg1, arg2, ...){
+   Do something with arg1 and arg2
+   return(result)
+ }
```

Two ways to write a function

- Think first and design the function as you go along
- Wrap function around already written code

Use apply with your own function

```
> pvals_a = apply(Experiment_DEG[,2:9], 1,
+   function(x) {as.numeric(t.test(x[1:4], x[5:8])$p.value)})
```

Write your own function (design your function)

Imagine that you have several experiments and always want to perform the same commands (summary statistics, row-wise t.test, filter for significance)

```
analyze_experiment <- function(){

}
```

```
analyze_experiment <- function(D){

    return(results)
}
```

```
analyze_experiment <- function(D){
    S = summary(D)

    return(results)
}
```

```
analyze_experiment <- function(D,                    ){

  S = summary(D)
  ColMean = colMeans(D[,2:length(D)], na.rm = T)
  ColVar = apply(D[,2:length(D)], 2, var)









  return(results)
}
```

Perform row-wise t.test and add it to the existing table, filter table by significant genes

```
analyze_experiment <- function(D,                    ){

  S = summary(D)
  ColMean = colMeans(D[,2:length(D)], na.rm = T)
  ColVar = apply(D[,2:length(D)], 2, var)


  D$pvalues = apply(D, 1, function(x)
    {as.numeric(t.test(as.nunmeric(x[group == 1]),
        as.numeric(x[group == 2]))$p.value)})


  significant_D = subset(D, pvalues <= alpha)









  return(results)
}
```

Create a list with all results in order to return them later on

```
analyze_experiment <- function(D,                ){

  S = summary(D)
  ColMean = colMeans(D[,2:length(D)], na.rm = T)
  ColVar = apply(D[,2:length(D)], 2, var)


  D$pvalues = apply(D, 1, function(x)
    {as.numeric(t.test(as.nunmeric(x[group == 1]),
        as.numeric(x[group == 2]))$p.value)})


  significant_D = subset(D, pvalues <= alpha)



  results = list(Summary = S, column.wise.mean = ColMean,
        column.wise.variance = ColVar, table.pvalues = D,
        significant = significant_D)

  return(results)
}
```

Check all variables you need

```
analyze_experiment <- function(D, group, alpha){

  S = summary(D)
  ColMean = colMeans(D[,2:length(D)], na.rm = T)
  ColVar = apply(D[,2:length(D)], 2, var)


  D$pvalues = apply(D, 1, function(x)
    {as.numeric(t.test(as.nunmeric(x[group == 1]),
        as.numeric(x[group == 2]))$p.value)})


  significant_D = subset(D, pvalues <= alpha)



  results = list(Summary = S, column.wise.mean = ColMean,
        column.wise.variance = ColVar, table.pvalues = D,
        significant = significant_D)

  return(results)
}
```

Comment your code

```
analyze_experiment <- function(D, group, alpha){
  # summary stats
  S = summary(D)
  ColMean = colMeans(D[,2:length(D)], na.rm = T)
  ColVar = apply(D[,2:length(D)], 2, var)

  # row-wise t.test
  D$pvalues = apply(D, 1, function(x)
    {as.numeric(t.test(as.nunmeric(x[group == 1]),
        as.numeric(x[group == 2]))$p.value)})

  # filter for significance
  significant_D = subset(D, pvalues <= alpha)

  # gather results

  results = list(Summary = S, column.wise.mean = ColMean,
        column.wise.variance = ColVar, table.pvalues = D,
        significant = significant_D)
  # return results
  return(results)
}
```

Run your own function

Copy paste, source or press the “Run” button in R studio

```
> ls()
```

Our function is available to us in this session, now lets use it

```
> head(Experiment_DEG)
     gene_id   Rep1   Rep2   Rep3   Rep4   Rep5   Rep6   Rep7   Rep8
1  gene_1 105.39  94.43 116.86 141.04  93.34 115.71 113.25  82.68
2  gene_2  87.40 142.94  51.73  94.68  70.89 115.54 102.72 104.11
3  gene_3 117.37  79.42 121.76  79.98 108.93 112.08  76.63  87.15
```

Run function on Experiment_DEG, specifying 1-4 as group 1 and 5-8 as group 2

```
> AD = analyze_experiment(Experiment_DEG, c(0,1,1,1,1,2,2,2,2), 0.05)
```

Lets look at the results

```
> typeof(AD)
[1] "list"
> names(AD)
[1] "Summary" "column.wise.mean" "column.wise.variance" "table.pvalues" "significant"
```

Wrap a function around your already written code

```
D1$pvalues = apply(D1, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:5]), as.numeric(x[6:9]))$p.value)})
D1_sig05 = subset(D1, pvalues <= 0.05)



D2$pvalues = apply(D2, 1, function(x) {as.numeric(t.test(as.numeric
        (x[1:3]), as.numeric(x[4:6]))$p.value)})
D2_sig01 = subset(D2, pvalues <= 0.01)

…

DY$pvalues = apply(DY, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:10]), as.numeric(x[11:16]))$p.value)})
DY_sig005 = subset(DY, pvalues <= 0.005)


DZ$pvalues = apply(DZ, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:4]), as.numeric(x[5:7]))$p.value)})
DZ_sig03 = subset(DZ, pvalues <= 0.03)
```

Identify differences

```
D1$pvalues = apply(D1, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:5]), as.numeric(x[6:9]))$p.value)})
D1_sig05 = subset(D1, pvalues <= 0.05)



D2$pvalues = apply(D2, 1, function(x) {as.numeric(t.test(as.numeric
        (x[1:3]), as.numeric(x[4:6]))$p.value)})
D2_sig01 = subset(D2, pvalues <= 0.01)

…

DY$pvalues = apply(DY, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:10]), as.numeric(x[11:16]))$p.value)})
DY_sig005 = subset(DY, pvalues <= 0.005)


DZ$pvalues = apply(DZ, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:4]), as.numeric(x[5:7]))$p.value)})
DZ_sig03 = subset(DZ, pvalues <= 0.03)
```

Wrap function around

```
analyze_experiment2 <- function(){
  D1$pvalues = apply(D1, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:5]), as.numeric(x[6:9]))$p.value)})
  D1_sig05 = subset(D1, pvalues <= 0.05)

}

D2$pvalues = apply(D2, 1, function(x) {as.numeric(t.test(as.numeric
        (x[1:3]), as.numeric(x[4:6]))$p.value)})
D2_sig01 = subset(D2, pvalues <= 0.01)

…

DY$pvalues = apply(DY, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:10]), as.numeric(x[11:16]))$p.value)})
DY_sig005 = subset(DY, pvalues <= 0.005)


DZ$pvalues = apply(DZ, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:4]), as.numeric(x[5:7]))$p.value)})
DZ_sig03 = subset(DZ, pvalues <= 0.03)
```

Use variables where differences were

```
analyze_experiment2 <- function(D, group, alpha ){
  D$pvalues = apply(D, 1, function(x) {as.numeric(t.test(as.numeric
        (x[group == 1]), as.numeric(x[group == 2]))$p.value)})
  D_sig = subset(D, pvalues <= alpha)

}

D2$pvalues = apply(D2, 1, function(x) {as.numeric(t.test(as.numeric
        (x[1:3]), as.numeric(x[4:6]))$p.value)})
D2_sig01 = subset(D2, pvalues <= 0.01)

…

DY$pvalues = apply(DY, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:10]), as.numeric(x[11:16]))$p.value)})
DY_sig005 = subset(DY, pvalues <= 0.005)


DZ$pvalues = apply(DZ, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:4]), as.numeric(x[5:7]))$p.value)})
DZ_sig03 = subset(DZ, pvalues <= 0.03)
```

Return results

```
analyze_experiment2 <- function(D, group, alpha ){
  D$pvalues = apply(D, 1, function(x) {as.numeric(t.test(as.numeric
        (x[group == 1]), as.numeric(x[group == 2]))$p.value)})
  D_sig = subset(D, pvalues <= alpha)
  return(list(table.pvalues = D, significant = D_sig))
}

D2$pvalues = apply(D2, 1, function(x) {as.numeric(t.test(as.numeric
        (x[1:3]), as.numeric(x[4:6]))$p.value)})
D2_sig01 = subset(D2, pvalues <= 0.01)

…

DY$pvalues = apply(DY, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:10]), as.numeric(x[11:16]))$p.value)})
DY_sig005 = subset(DY, pvalues <= 0.005)


DZ$pvalues = apply(DZ, 1, function(x) {as.numeric(t.test(as.numeric
        (x[2:4]), as.numeric(x[5:7]))$p.value)})
DZ_sig03 = subset(DZ, pvalues <= 0.03)
```

Clean up

```
analyze_experiment2 <- function(D, group, alpha ){
  D$pvalues = apply(D, 1, function(x) {as.numeric(t.test(as.numeric
        (x[group == 1]), as.numeric(x[group == 2]))$p.value)})
  D_sig = subset(D, pvalues <= alpha)
  return(list(table.pvalues = D, significant = D_sig))
}
```

Run

```
analyze_experiment2 <- function(D, group, alpha ){
  D$pvalues = apply(D, 1, function(x) {as.numeric(t.test(as.numeric
        (x[group == 1]), as.numeric(x[group == 2]))$p.value)})
  D_sig = subset(D, pvalues <= alpha)
  return(list(table.pvalues = D, significant = D_sig))
}

D1_res = analyze_experiment2(D1, c(0,1,1,1,1,2,2,2,2),0.05)
D2_res = analyze_experiment2(D2, c(1,1,1,2,2,2),0.01)
...
DY_res = analyze_experiment2(DY, c(0,rep(1,9), rep(2, 6)),0.005)
DZ_res = analyze_experiment2(DZ, c(0,1,1,1,2,2,2),0.03)
```

Want to add something?

```
analyze_experiment2 <- function(D, group, alpha ){
  S = summary(D)
  CM = colMeans(D[,group %in% c(1,2)], na.rm = T)
  CV = apply(D[,group %in% c(1,2)], 2, var)


  D$pvalues = apply(D, 1, function(x) {as.numeric(t.test(as.numeric
        (x[group == 1]), as.numeric(x[group == 2]))$p.value)})
  D_sig = subset(D, pvalues <= alpha)
  return(list(Summary = S, column.wise.mean = CM,
        column.wise.variance = CV, table.pvalues = D,
        significant = D_sig))
}

D1_res = analyze_experiment2(D1, c(0,1,1,1,1,2,2,2,2),0.05)
D2_res = analyze_experiment2(D2, c(1,1,1,2,2,2),0.01)
...
DY_res = analyze_experiment2(DY, c(0,rep(1,9), rep(2, 6)),0.005)
DZ_res = analyze_experiment2(DZ, c(0,1,1,1,2,2,2),0.03)
```

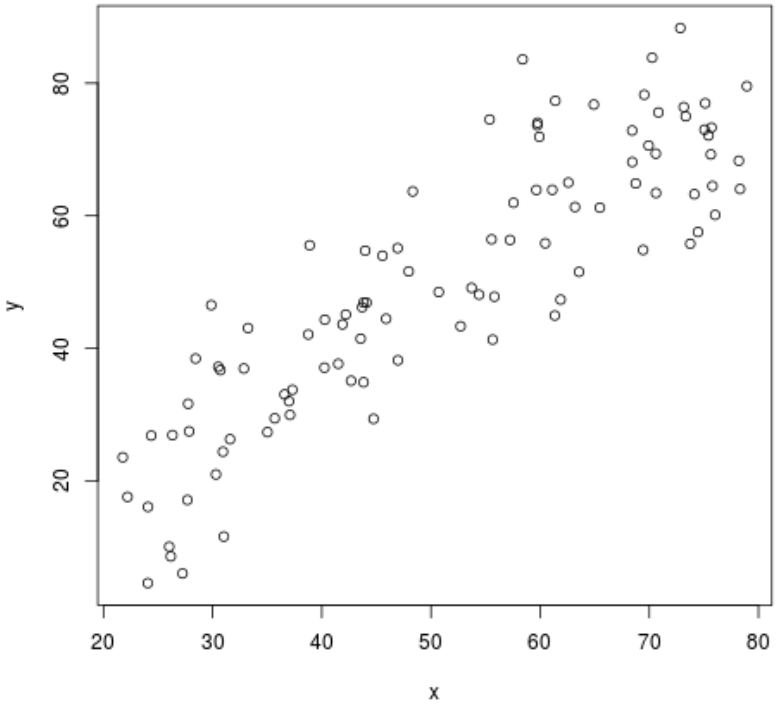
Think about other examples, when could it be useful to write a function

5min Break

Basic plotting

Simple Plot

```
> x = runif(100, 20, 80)
> y = rnorm(100, x, 10)

> plot(x, y)
```



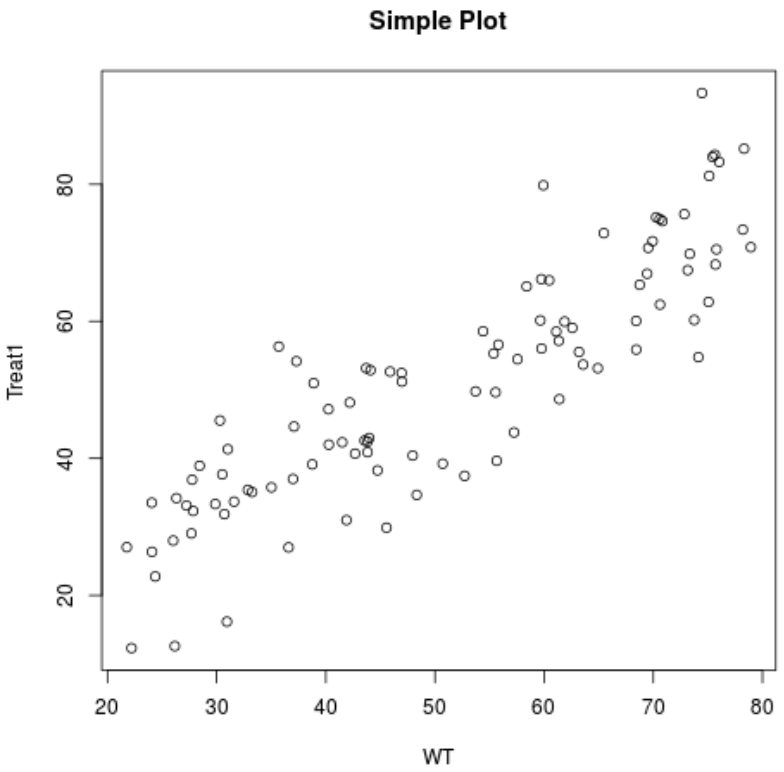
Scatterplot

Simple Plot

```
> x = runif(100, 20, 80)
> y = rnorm(100, x, 10)

> plot(x, y)
```

Add title and axis labels

```
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1')
```



Scatterplot

Simple Plot

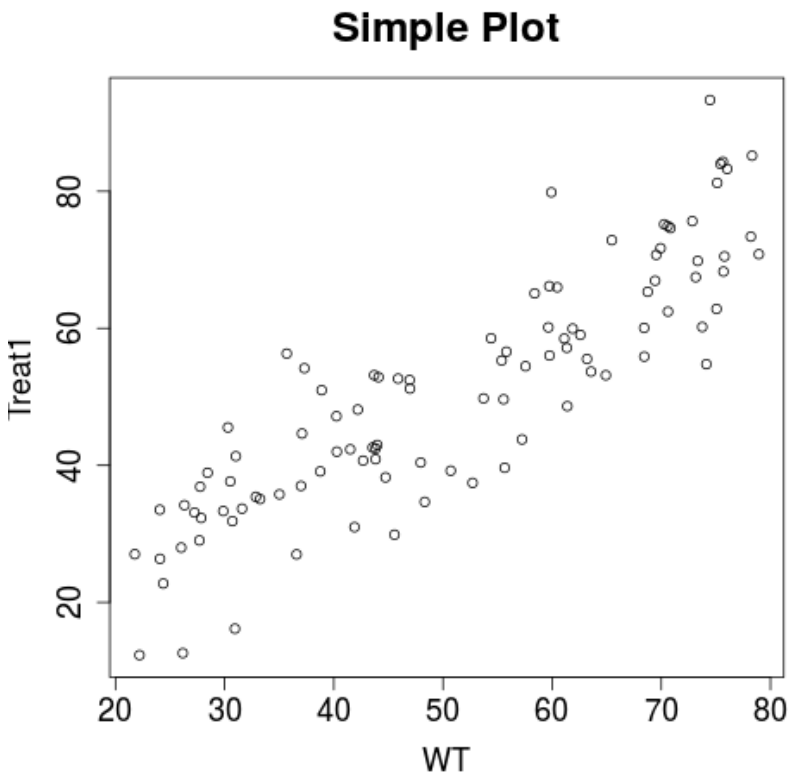
```
> x = runif(100, 20, 80)
> y = rnorm(100, x, 10)

> plot(x, y)
```

Add title and axis labels

```
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1')
```

Make text bigger

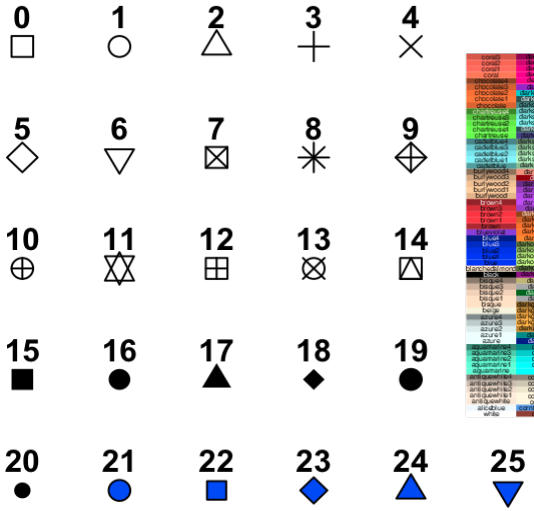
```
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1',
    cex.main = 2, cex.axis = 1.5,
    cex.lab = 1.5)
```

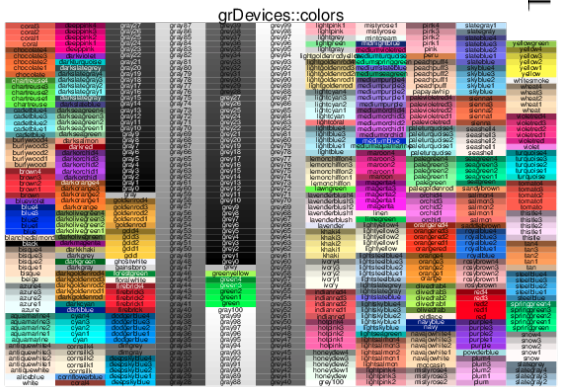


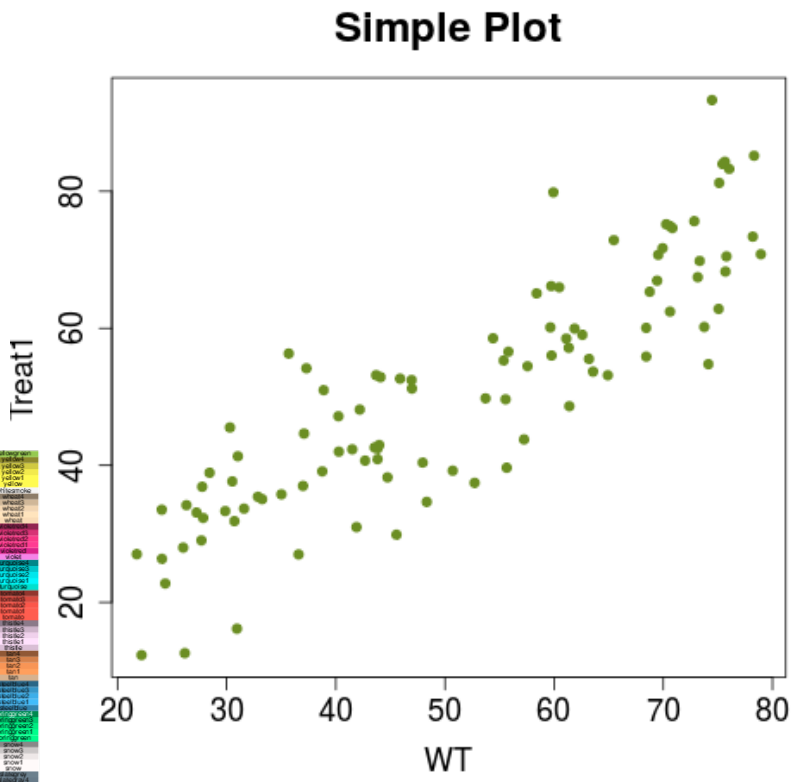
Scatterplot

Change color and shape

```
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1',
    cex.main = 2, cex.axis = 1.5,
    cex.lab = 1.5, pch = 17,
    col = 'olivedrab')
```



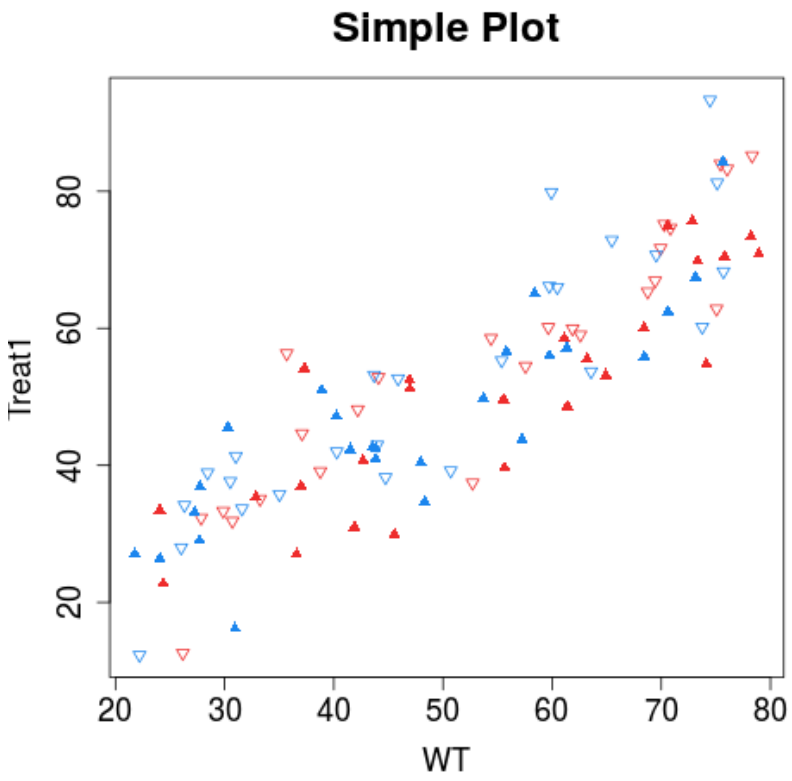




Scatterplot

Change color and shape using group variables

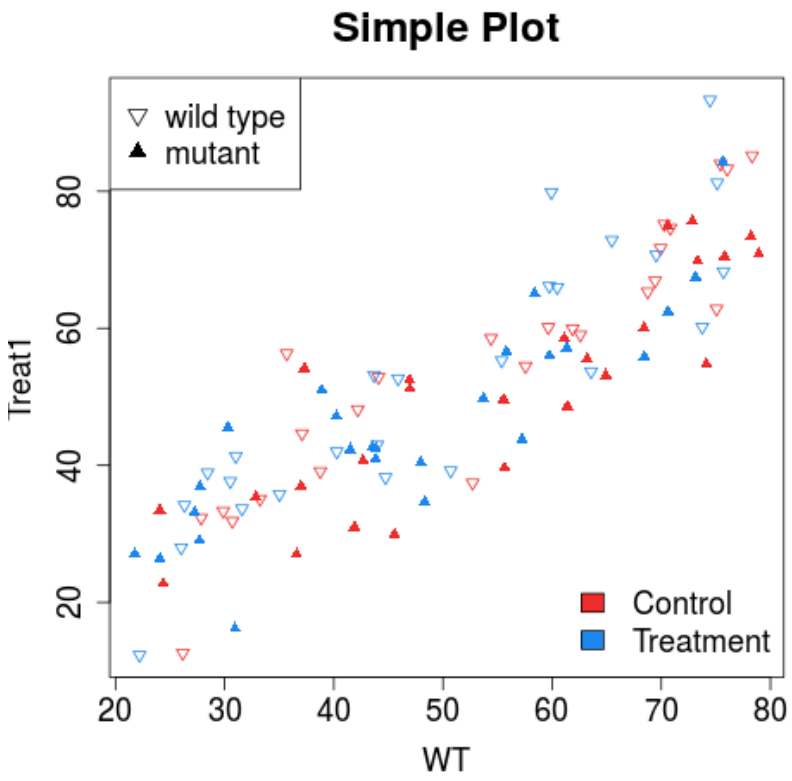
```
> x = runif(100, 20, 80)
> y = rnorm(100, x, 10)
> group_col = c(rep('firebrick2',
    50),rep('dodgerblue2', 50))
> group_pch = rep(c(6,17), 50)
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1',
    cex.main = 2, cex.axis = 1.5,
    cex.lab = 1.5,
    pch = group_pch,
    col = group_col)
```



Scatterplot

Add legend

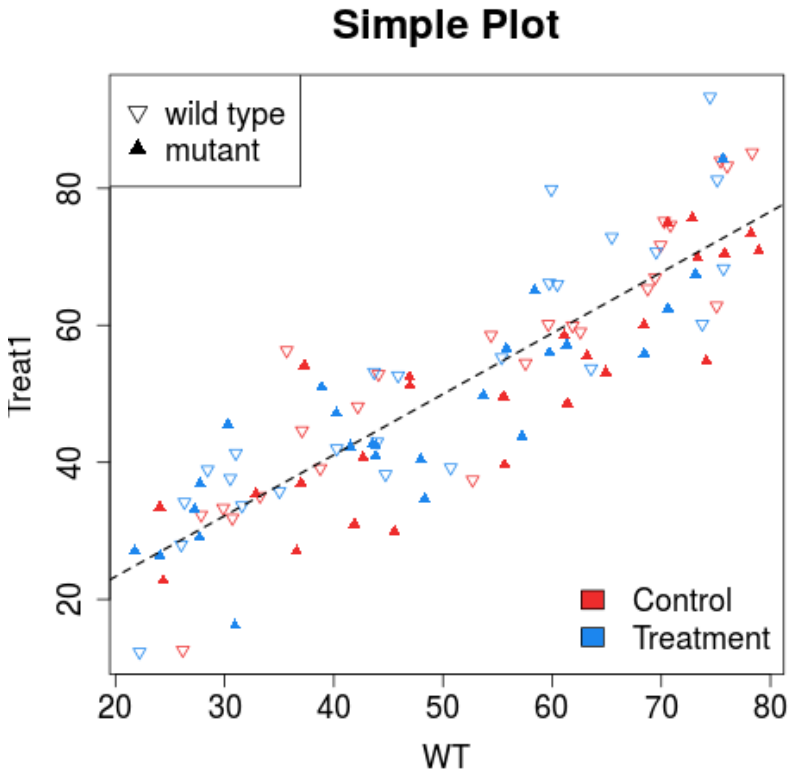
```
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1',
    cex.main = 2, cex.axis = 1.5,
    cex.lab = 1.5,
    pch = group_pch,
    col = group_col)
> legend('topleft', c('wild
    type', 'mutant'),
    pch = c(6, 17), cex = 1.5)
> legend('bottomright',
    c('Control', 'Treatment'),
    fill = c('firebrick2',
    'dodgerblue2'),
    cex = 1.5, bty = 'n'))
```



Scatterplot

Add regression line

```
> plot(x, y, main = 'Simple Plot',
    xlab = 'WT', ylab = 'Treat1',
    cex.main = 2, cex.axis = 1.5,
    cex.lab = 1.5,
    pch = group_pch,
    col = group_col)
> legend('topleft', c('wild
    type', 'mutant'),
    pch = c(6, 17), cex = 1.5)
> legend('bottomright',
    c('Control', 'Treatment'),
    fill = c('firebrick2',
    'dodgerblue2'),
    cex = 1.5, bty = 'n'))
> abline(lm(y ~ x), lty = 2,
    lwd = 1.5)
```

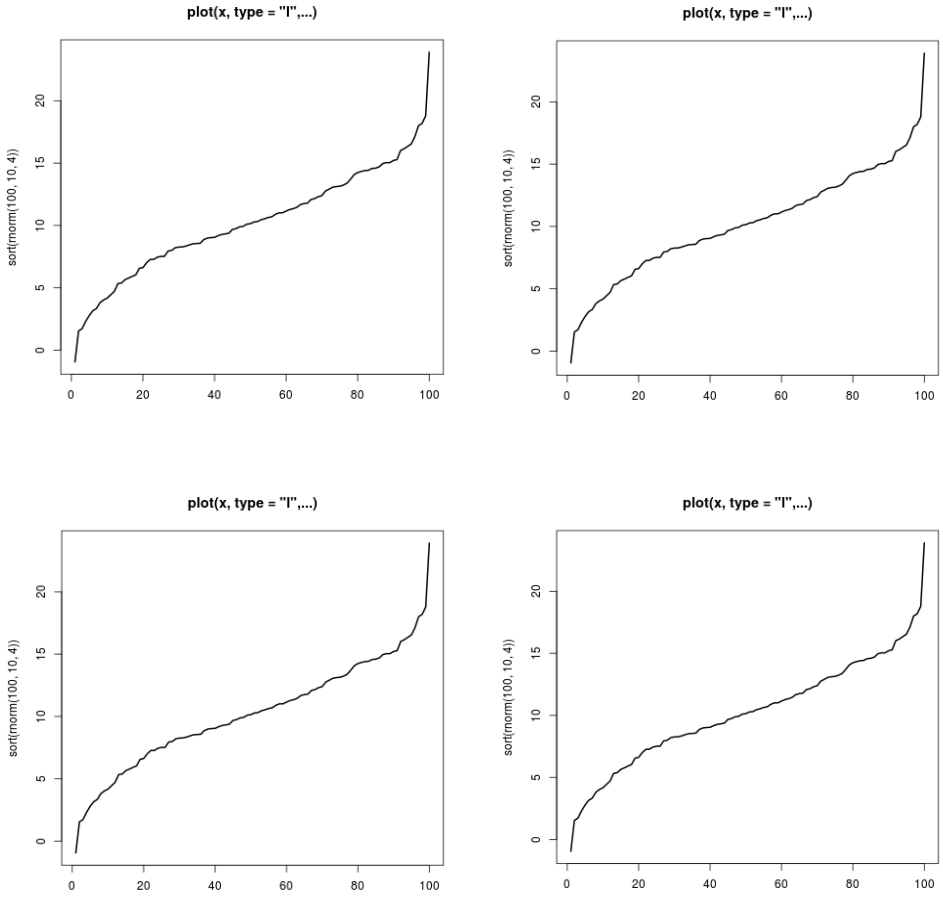


Other plots

Lineplot

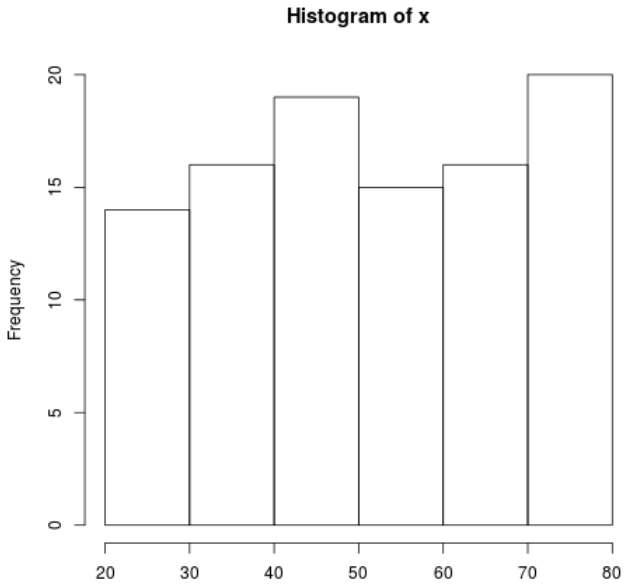
plot(x, y, type = …)

- "p" for points,
- "l" for lines,
- "b" for both,
- "c" for the lines part alone of "b",
- "o" for both ‘overplotted’,
- "h" for ‘histogram’ like vertical lines,
- "s" for stair steps,
- "S" for other steps, see ‘Details’ below,
- "n" for no plotting.

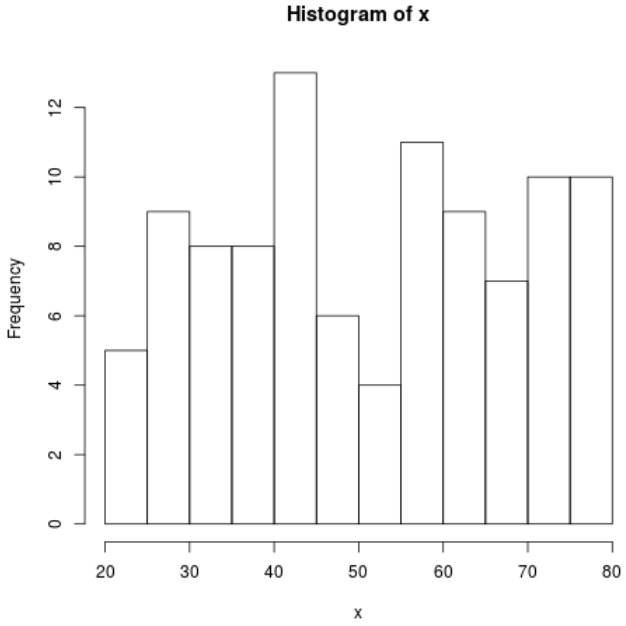


Histogram

```
> hist(x)
```




```
> hist(x, breaks = 10)
```

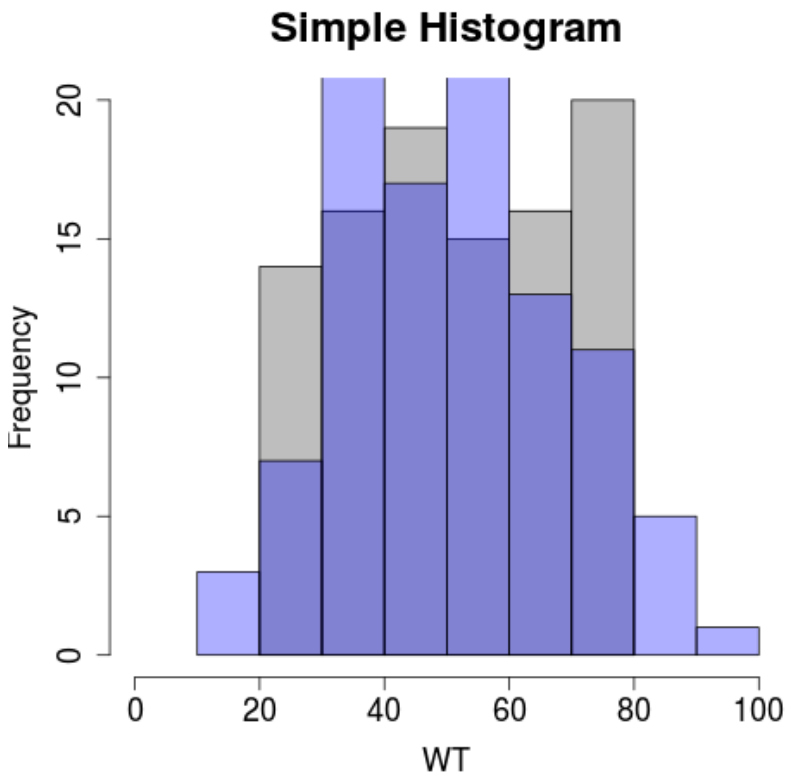


Histogram

```
> hist(x)
```

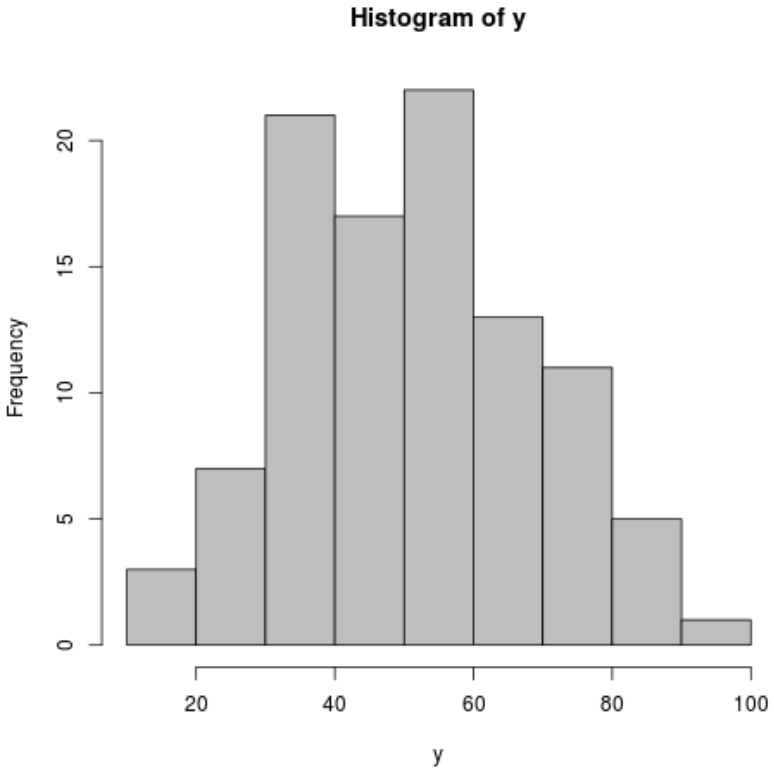
```
> hist(x, breaks = 10)
```

```
> hist(x, xlab = 'WT',
    main = 'Simple Histogram',
    cex.main = 2, cex.axis = 1.5,
    cex.lab = 1.5, col = 'gray',
    xlim = c(0, 100))
> hist(y,col = '#0000FF50',
    add = T)
```



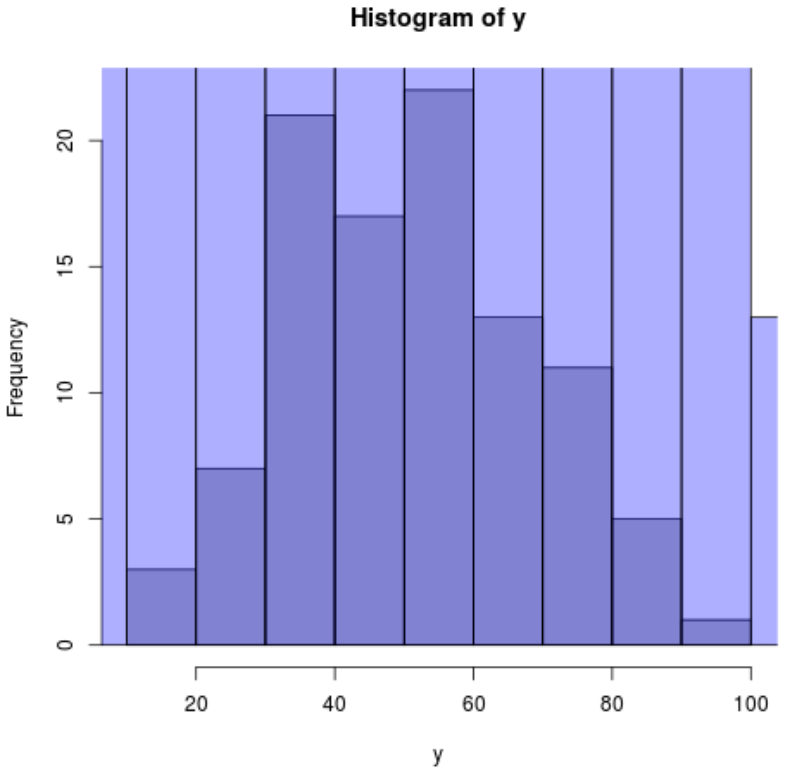
Histogram

```
> hist(y, col = 'gray')
```



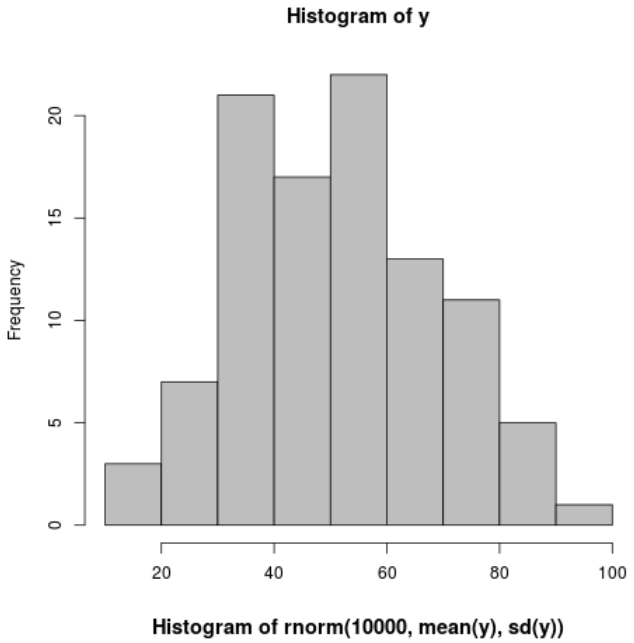
Histogram

```
> hist(y, col = 'gray')
> hist(rnorm(10000, mean(y),
    sd(y)), col = '#0000FF50',
    add = T)
```



Histogram

```
> hist(y, col = 'gray')
> hist(rnorm(10000, mean(y),
    sd(y)), col = '#0000FF50',
    add = T)
```




```
> hist(rnorm(10000, mean(y),
    sd(y)), col = '#0000FF50')
```

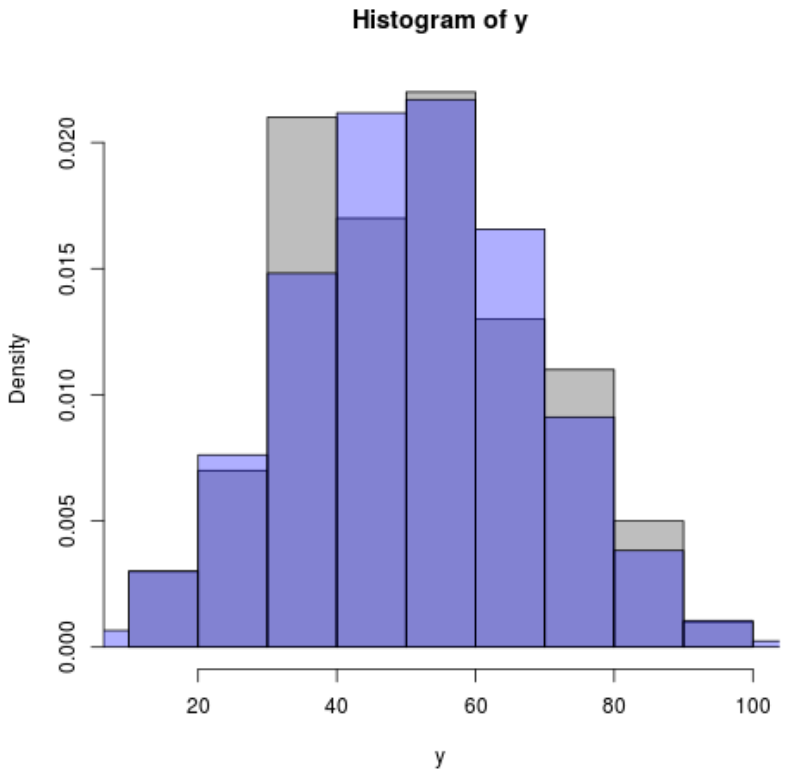
```
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

Histogram

```
> hist(y, col = 'gray')
> hist(rnorm(10000, mean(y),
    sd(y)), col = '#0000FF50',
    add = T)
```

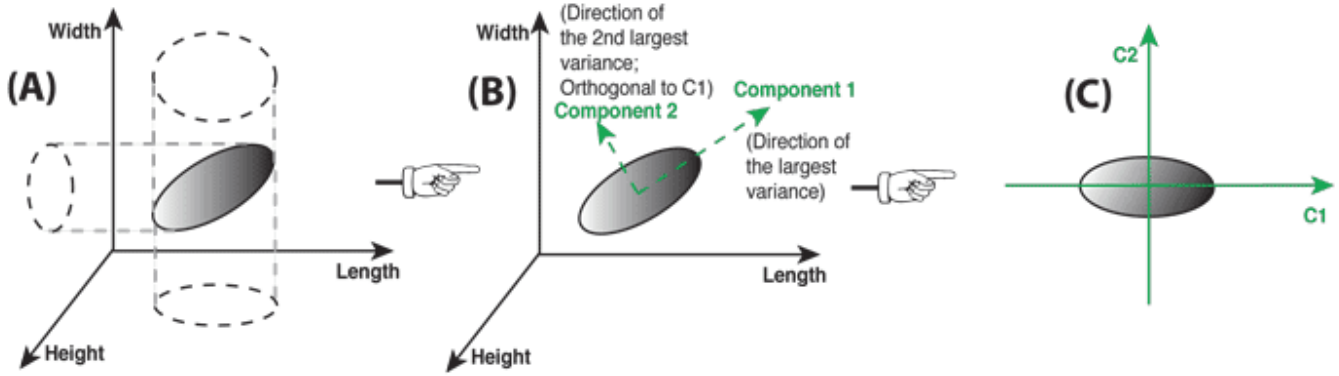
```
> hist(rnorm(10000, mean(y),
    sd(y)), col = '#0000FF50')
```

```
> hist(y, col = 'gray', freq = F)
> hist(rnorm(10000, mean(y),
    sd(y)), col = '#0000FF50',
    add = T, freq = F)
```



Principal Component Analysis (PCA)

- Orthogonal transformation
- Correlated variables → linearly uncorrelated variables (principal components, PC)
- Dimensionality reduction, # PC <= min(# observation, # variables)
- First PC has the largest possible variance
- Different algorithms to obtain PCs
- PCA is mostly used as a tool in:
    - exploratory data analysis
    - visualize genetic distance and relatedness between populations.



http://4.bp.blogspot.com/-pleL0HvLUgU/UYqpNFdd8EI/AAAAAAAAAHA/uf11u9lcq5g/s1600/PCA_1.png

PCA in R

Load example data

```
> wine <- read.csv("wine.csv")
> head(wine)
```

```
> head(wine)
  Cvs Alcohol Malic.acid  Ash Alcalinity.of.ash Magnesium Total.phenols Flavanoids Nonflavanoid.phenols Proanthocyanins
1   1   14.23       1.71 2.43              15.6       127          2.80       3.06                 0.28            2.29
2   1   13.20       1.78 2.14              11.2       100          2.65       2.76                 0.26            1.28
3   1   13.16       2.36 2.67              18.6       101          2.80       3.24                 0.30            2.81
4   1   14.37       1.95 2.50              16.8       113          3.85       3.49                 0.24            2.18
5   1   13.24       2.59 2.87              21.0       118          2.80       2.69                 0.39            1.82
6   1   14.20       1.76 2.45              15.2       112          3.27       3.39                 0.34            1.97
  Color.intensity  Hue OD280.OD315.of.diluted.wines Proline
1            5.64 1.04                         3.92    1065
2            4.38 1.05                         3.40    1050
3            5.68 1.03                         3.17    1185
4            7.80 0.86                         3.45    1480
5            4.32 1.04                         2.93     735
6            6.75 1.05                         2.85    1450
```

PCA in R

Load example data

```
> wine <- read.table("wine.csv")
> head(wine)
```

First column should be factors

```
> wineClasses <- factor(wine$Cvs)
```

Compute principal components

```
> winePCA <- prcomp(scale(wine[,-1]))
```

Look at results

```
> summary(winePCA)
> names(winePCA)
```

```
> summary(winePCA)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9   PC10    PC11    PC12    PC13
Standard deviation     2.169 1.5802 1.2025 0.95863 0.92370 0.80103 0.74231 0.59034 0.53748 0.5009 0.47517 0.41082 0.32152
Proportion of Variance 0.362 0.1921 0.1112 0.07069 0.06563 0.04936 0.04239 0.02681 0.02222 0.0193 0.01737 0.01298 0.00795
Cumulative Proportion  0.362 0.5541 0.6653 0.73599 0.80162 0.85098 0.89337 0.92018 0.94240 0.9617 0.97907 0.99205 1.00000
> names(winePCA)
[1] "sdev"     "rotation" "center"   "scale"    "x"
```

PCA in R

Load example data

```
> wine <- read.table("wine.csv")
> head(wine)
```

First column should be factors

```
> wineClasses <- factor(wine$Cvs)
```

Compute principal components

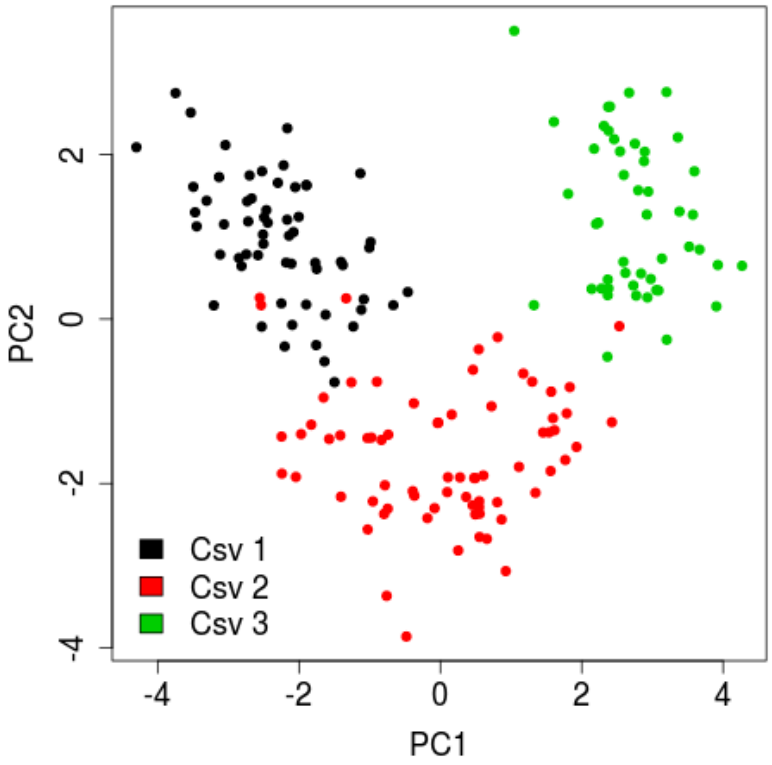
```
> winePCA <- prcomp(scale(wine[,-1]))
```

Look at results

```
> summary(winePCA)
> names(winePCA)
```

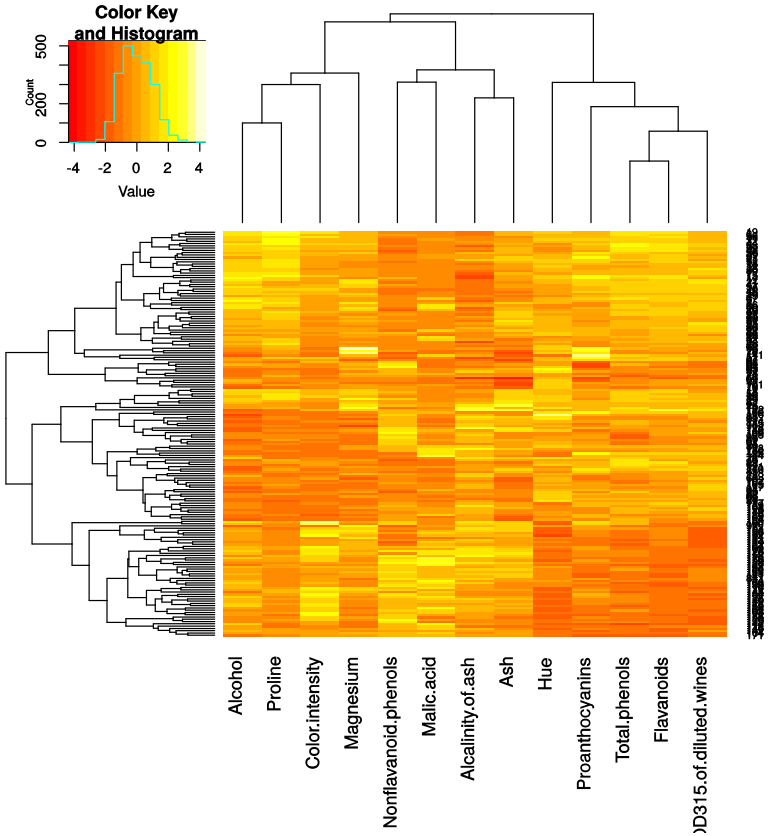
Plot first vs second component

```
> plot(winePCA$x[,1:2],
    col = wineClasses, pch = 19)
```



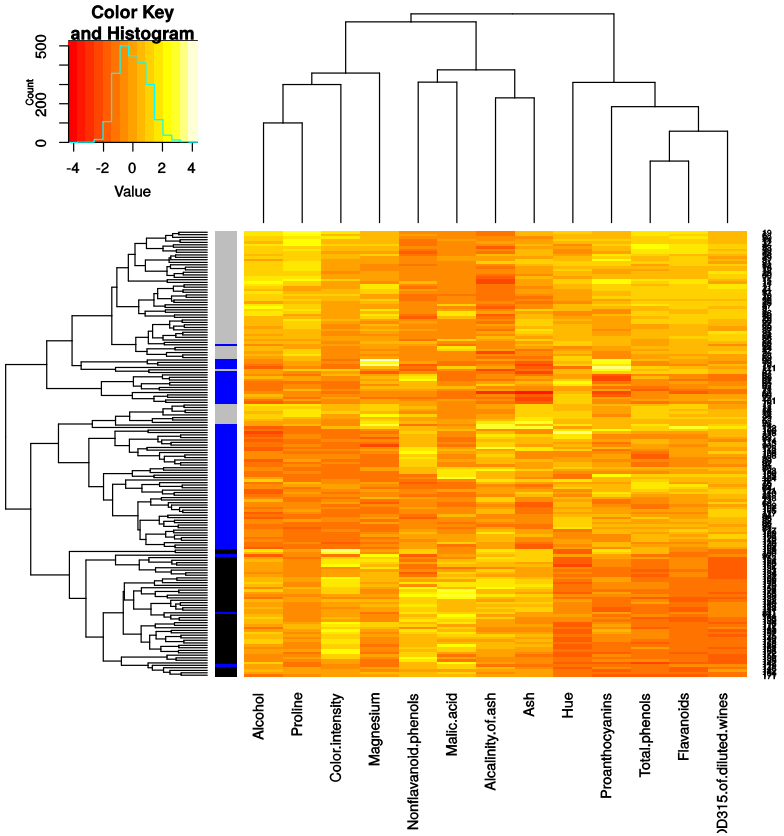
Heatmap (gplots)

```
> library(gplos)
> ?heatmap.2
> heatmap.2(scale(wine[,-1]),
    mar = c(10,5), trace = 'none')
```



Heatmap (gplots)

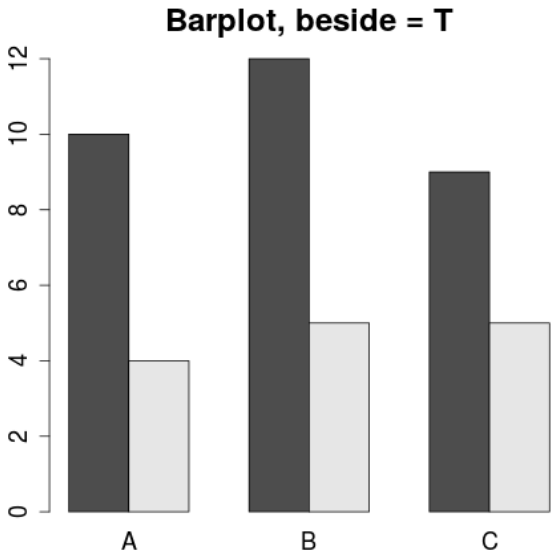
```
> library(gplos)
> ?heatmap.2
> heatmap.2(scale(wine[,-1]),
    mar = c(10,5), trace = 'none')

> rowCol = rep(NA, 178)
> rowCol[wine$Cvs == 1] <- 'gray'
> rowCol[wine$Cvs == 2] <- 'blue'
> rowCol[wine$Cvs == 3] <- 'black'

> heatmap.2(scale(wine[,-1]),
    mar = c(10,5), trace = 'none',
    RowSideColors = rowCol)
```



Other plots

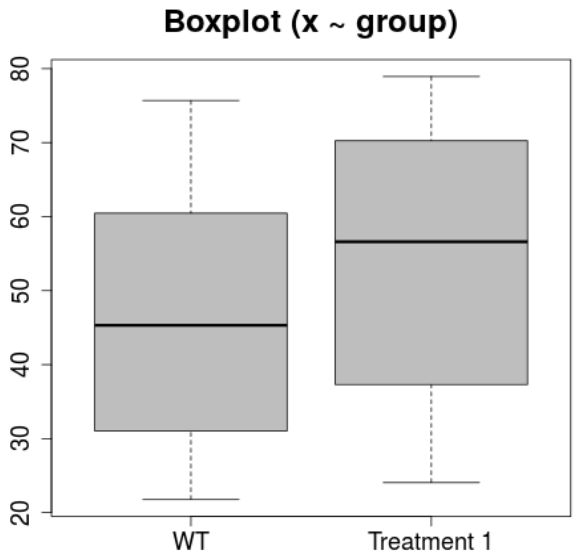
Barplot

```
barplot(height, width = 1, space = NULL, names.arg =
NULL, legend.text = NULL, beside = FALSE, horiz =
FALSE, density = NULL, angle = 45,col = NULL, border
= par("fg"), main = NULL, sub = NULL, xlab = NULL,
ylab = NULL, xlim = NULL, ylim = NULL, xpd = TRUE,
log = "", axes = TRUE, axisnames = TRUE, cex.axis =
par("cex.axis"), cex.names = par("cex.axis"), inside
= TRUE, plot = TRUE, axis.lty = 0, offset = 0, add =
FALSE, args.legend = NULL, ...)
```



Boxplot

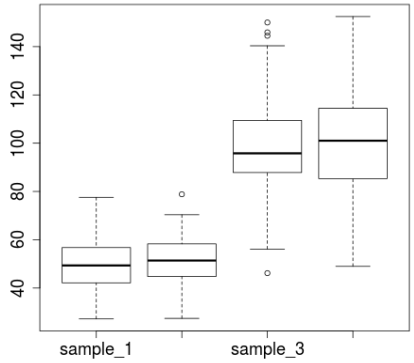
```
boxplot(x (~ group), ..., range = 1.5, width =
NULL, varwidth = FALSE, notch = FALSE, outline
= TRUE, names, plot = TRUE, border = par("fg"),
col = NULL, log = "", pars = list(boxwex = 0.8,
staplewex = 0.5, outwex = 0.5), horizontal =
FALSE, add = FALSE, at = NULL)
```

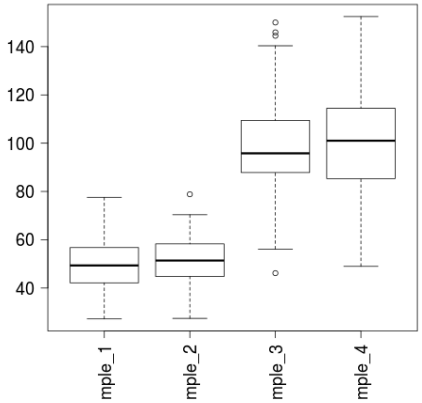


Plot properties (par)

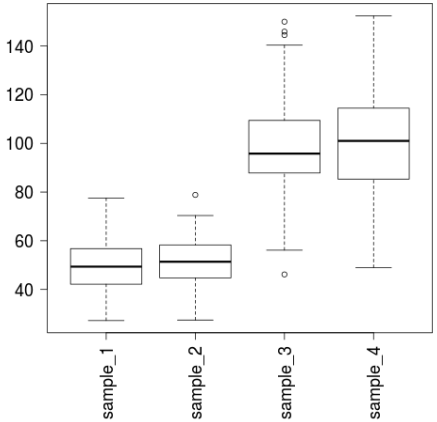
Margins

```
par(mar = c(bottom, left, top, right))
```



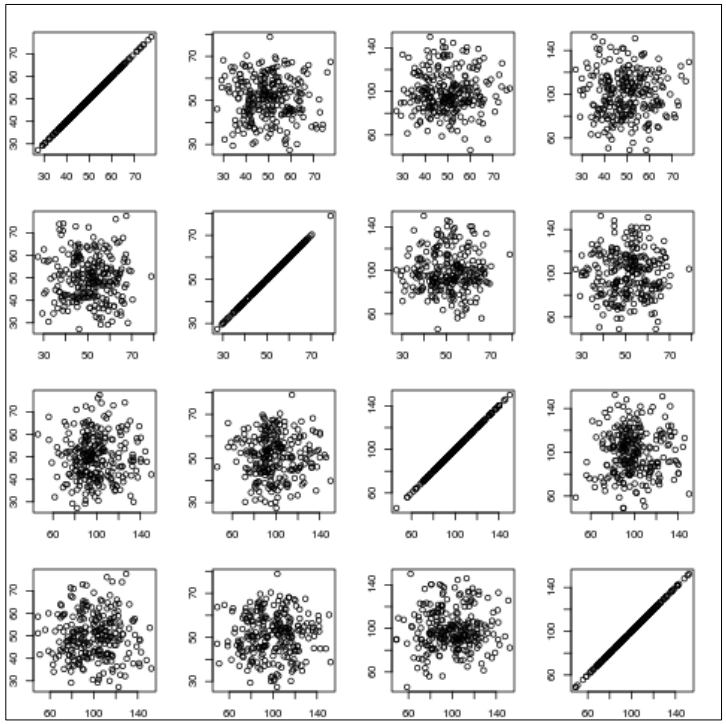



```
> par(mar =
  c(8, 4, 1, 1))
> boxplot(x ~ g,
  las = 2)
```



Mfrow

```
par(mfrow = c(nrow, ncol))
```

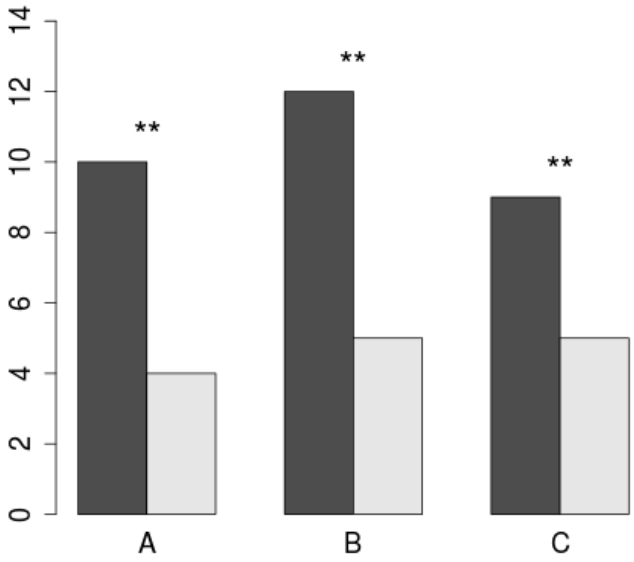


```
> par(mfrow = c(4,4)
> par(mar = c(2,2,2,2))
```

Add text to plot

Within Plot

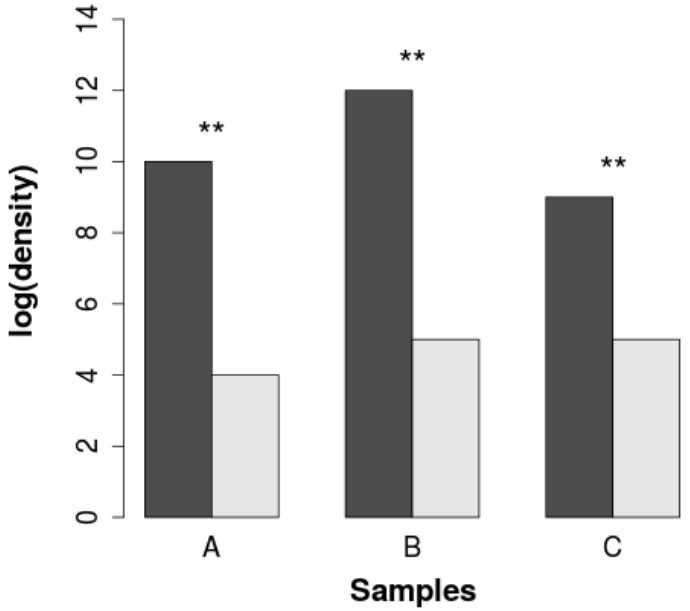
```
text(x, y, labels, adj, pos, cex, ... )
```




```
> bp = barplot(M, ...)
> text(x = colMeans(bp),
     y = c(11, 13, 10), '**')
```

Within Margin

```
mtext(text, side = 3, line = 0, ... )
```




```
> par(mar = c(5, 6, 4, 1))
> mtext('log(density)', 2, 4)
> mtext('Samples', 1, 3)
```

5min Break

Install packages

Is it the right package I am looking for

Install package (cran, bioconductor)

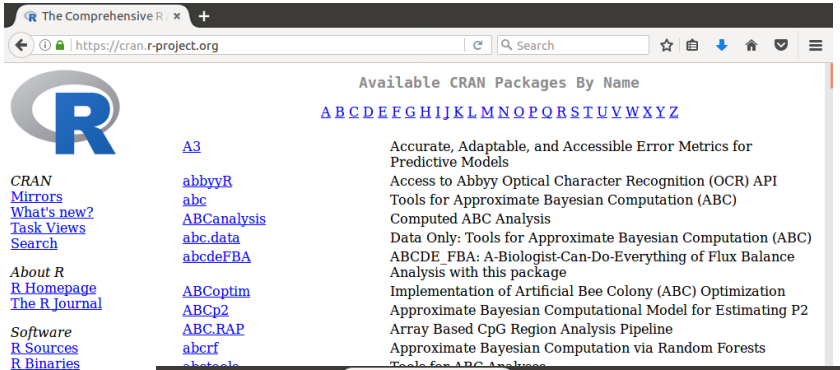
What functions does the package have

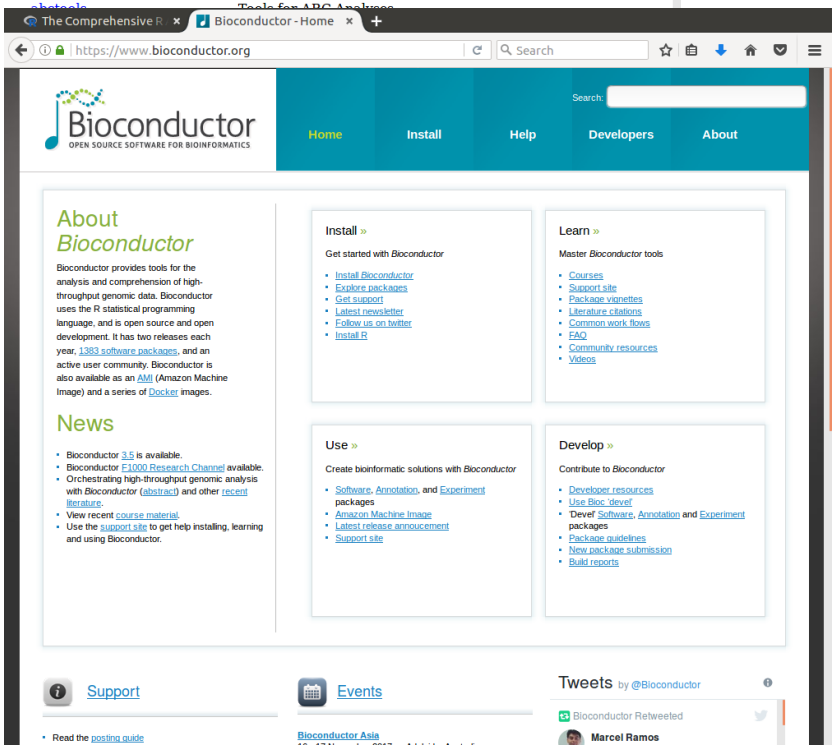
Examples

Help

What to do if you are stuck

https://cran.r-project.org/





https://www.bioconductor.org/

Install packages

Cran

```
> ?install.packages
# install.packages(pkgs, lib,
    repos = ,method,
    available = , destdir = ,
    dependencies = NA, ...)

> install.packages('gplots')
> library(gplots)
> heatmap.2(...)
```

Bioconductor

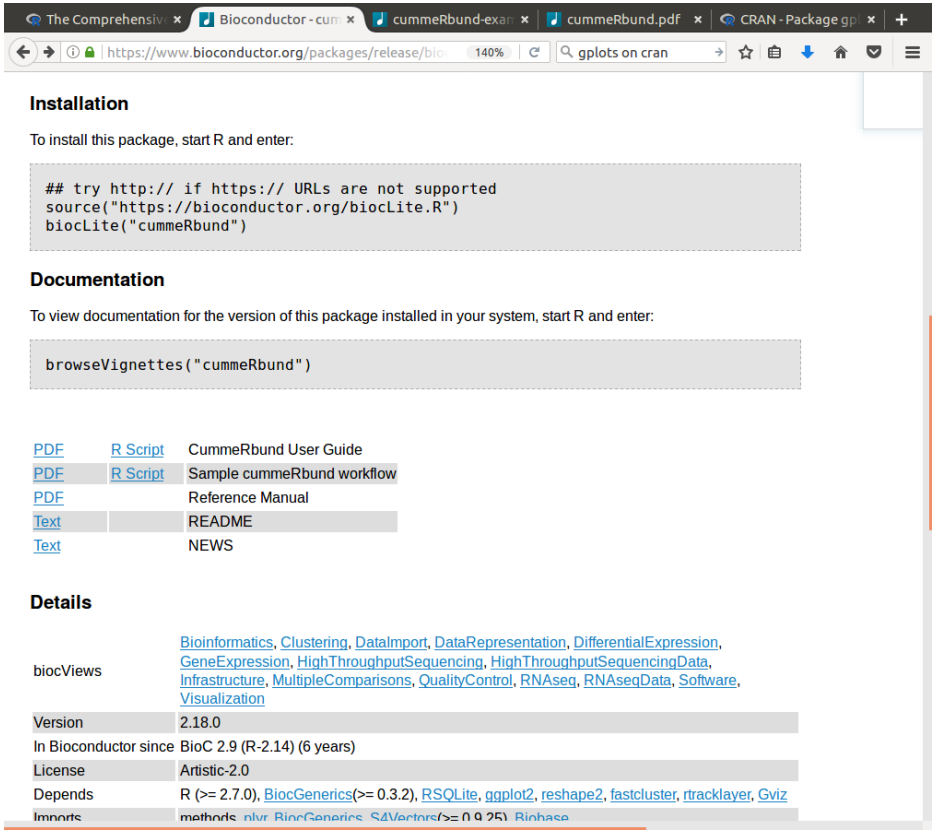
```
> source("https://
    bioconductor.org/biocLite.R")
> ? biocLite
# biocLite(pkgs =, ask=TRUE,
  suppressUpdates=FALSE, ...)

> biocLite("cummeRbund")
> cuff <- readCufflinks(...)
```

Repository CRAN

Date/Publication 2016-03-30 21:04:18

R topics documented:

What functions are in the packages?



Installation

To install this package, start R and enter:

```
## try http:// if https:// URLs are not supported
source("https://bioconductor.org/biocLite.R")
biocLite("cummeRbund")
```

Documentation

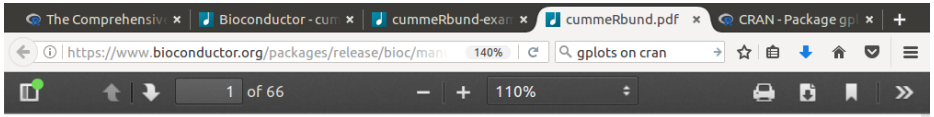
To view documentation for the version of this package installed in your system, start R and enter:

```
browseVignettes("cummeRbund")
```

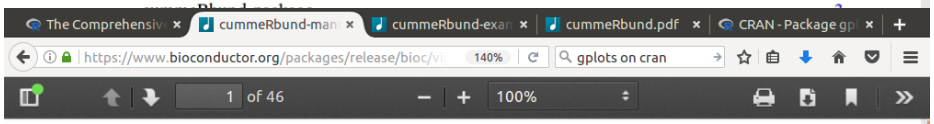
PDF	R Script	CummeRbund User Guide
PDF	R Script	Sample cummeRbund workflow
PDF		Reference Manual
Text		README
Text		NEWS

Details

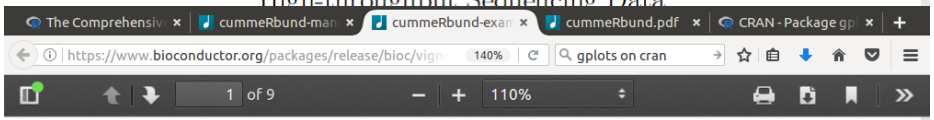
biocViews	Bioinformatics, Clustering, DataImport, DataRepresentation, DifferentialExpression, GeneExpression, HighThroughputSequencing, HighThroughputSequencingData, Infrastructure, MultipleComparisons, QualityControl, RNAseq, RNAseqData, Software, Visualization
Version	2.18.0
In Bioconductor since	BioC 2.9 (R-2.14) (6 years)
License	Artistic-2.0
Depends	R (>= 2.7.0), BiocGenerics(>= 0.3.2), RSQLite, ggplot2, reshape2, fastcluster, rtracklayer, Gviz
Imports	methods, plyr, BiocGenerics, S4Vectors(>= 0.9.25), Biobase



R topics documented:



CummeRbund: Visualization and Exploration of Cufflinks High-throughput Sequencing Data



CummeRbund workflow

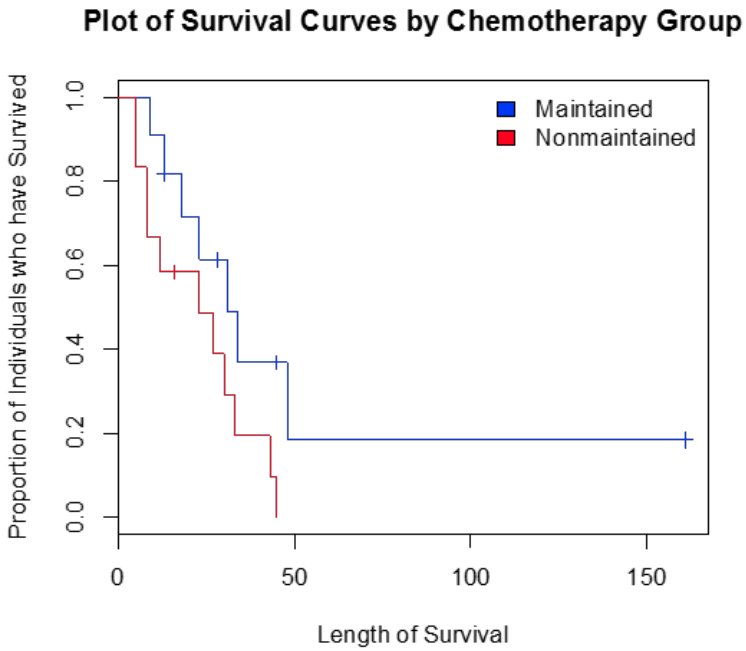
Loyal A. Goff

This document is a work in progress and will continually be updated as new features or analyses are integrated into the cummeRbund pipeline. This guide is being released as is, with the understanding that existing gaps will be completed in due time. Please bear with us as we work to expand this resource.

Contents

Survival package for lifespan analysis

```
> install.packages('survival')
> library(survival)
>
```



CRAN - Package survival

https://cran.r-project.org/web/p

survival: Survival Analysis

Contains the core survival analysis routines, including definition of Surv objects, Kaplan-Meier and Aalen-Johansen (multi-state) curves, Cox models, and parametric accelerated failure time models.

Version:	2.41-3
Priority:	recommended
Depends:	R (≥ 2.13.0)
Imports:	graphics, Matrix, methods, splines, stats, utils
Published:	2017-04-04
Author:	Terry M Therneau [aut, cre], Thomas Lumley [ctb, trl] (original S->R port and maintainer until 2009)
Maintainer:	Terry M Therneau <therneau.terry at mayo.edu>
License:	LGPL-2 | LGPL-2.1 | LGPL-3 [expanded from: LGPL (≥ 2)]
Copyright:	see file COPYRIGHTS
URL:	https://github.com/therneau/survival
NeedsCompilation:	yes
Citation:	survival citation info
Materials:	NEWS
In views:	ClinicalTrials, Econometrics, SocialSciences, Survival
CRAN checks:	survival results

Downloads:

Reference manual:	survival.pdf
Vignettes:	Adjusted Survival Curves Multi-state models and competing risks Multi-state example Splines, plots, and interactions Cox models and "type 3" Tests Roundoff error and tied times Using Time Dependent Covariates Validation
Package source:	survival_2.41-3.tar.gz

http://sphweb.bumc.bu.edu/otlt/MPH-Modules/BS/R/R7_LogisticRegression-Survival/R7_LogisticRegression-Survival_print.html

Do it yourself, a guide to a differential gene expression analysis

Experiment: Synaptic Plasticity
Organism: rat
Tissue: hippocampal neurons
Condition: control, picrotoxin
Replicates: 3 per condition

What had been done before:
Quality control
(Trimming): Flexbar
Alignment: STAR, Bowtie, HISAT, Tophat
Counting gene features

Data: rat_hippocampal_neurons.gene_counts.tsv

Do it yourself, a guide to a differential gene expression analysis

What to do next:

PCA

DEG

Furthermore:

Visualization

KEGG Pathway, GO-Terms

Data: rat_hippocampal_neurons.gene_counts.tsv

END OF DAY 2