Place the following commands in the correct order. What is the function `twice()` doing?

```
a) if (is.numeric(x)==FALSE){
b) return(2*x)
c) }
d) twice <- function(x){
e) }
f) stop("Argument must be numeric")
```

Answer: d, a, f, c, b, e (c and e can be swapped of course).
The function is returning the value of its argument times 2

An introduction to R
WS 2019/2020

Basic Statistics in R

Dr. Noémie Becker
Dr. Eliza Argyridou

Special thanks to:
Dr. Sonja Grath for addition to slides

What you should know after day 9

sapply() and tapply()
A final word on functions

Some distributions in R
- Example: Normal distribution
- Random numbers in R

Statistical tests in R
Example: t-test

sapply() and tapply()

You can use **sapply()** to apply the same function to each element of an object.

Example (from lecture 8):

```
gcContent("AACGTGGCTA")
[1] 0.5

sapply(c("AACGTGGCTA", "AATATATTAT"), gcContent)
AACGTGGCTA AATATATTAT
       0.5        0.0
```

You can use **tapply()** for data frames.
Example (see also exercise sheet 7):
We want to have the mean weight increase for each treatment.

```
tapply(heartbeats$wghtincr, heartbeats$treatment, mean)
        0         1
-31.72727  37.50000
```


What you should know after day 9

sapply() and tapply()
A final word on functions

Some distributions in R
- Example: Normal distribution
- Random numbers in R

Statistical tests in R
Example: t-test

Some distributions in R

Distribution	R name
beta	beta
binomial	binom
Cauchy	cauchy
chi-square	chisq
exponential	exp
F	F
gamma	gamma
geometric	geom
hypergeometric	hyper
log-normal	lnorm
logistic	logis
multinomial	multinom
multivariate	mvnorm
normal	norm
Poisson	pois
Student's t	t
uniform	unif
distribution of the Wilcoxon rank statistic	wilcox
...	...

Some distributions in R

For each distribution:
dxxx: density of the xxx distribution
pxxx: distribution function of the xxx distribution ('p' for probability)
qxxx: quantile function of the xxx distribution
rxxx: random number generator for the xxx distribution

Example: Normal distribution

```
dnorm(x, mean = μ, sd = ρ)
```

Standard normal distribution:
mean 0, standard deviation 1

Default (from usage):

```
dnorm(x, mean = 0, sd = 1)
dnorm(x)
```

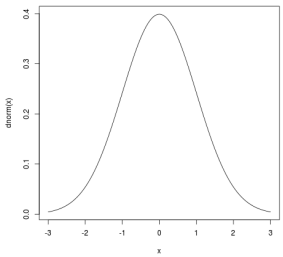

Example: Normal distribution

Recall:

```
plot()
```

plot(*fun*)
→ If *fun* is a function, then plot(*fun*, from = a, to = b) plots *fun* in the range [a, b]

```
plot(dnorm, from = -3, to = 3)
```

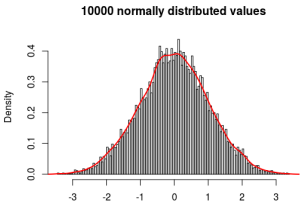


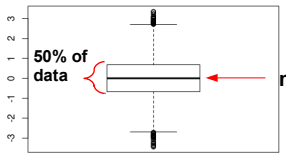
Box- and whisker plot (boxplot)

Get 10000 normally distributed values:

```
rnorm(10000)
```

```
boxplot(rnorm(10000))
```





Box- and whisker plot (boxplot)

Get 10000 normally distributed values:

```
rnorm(10000)
```

```
boxplot(rnorm(10000))
```

Change the mean to -1

```
rnorm(10000, mean=-1)
```


Random numbers

Actually: Random numbers are not really random...

→ Computers produce **pseudo**-random numbers

Reasons:

1. A normally distributed variable has a continuum of potential values – but computers only can represent a finite number of values
2. Results should be reproducible

Properties of pseudo-random numbers:

- Almost no regularities in the generated sequence
- Random sequence is reproducible
- Random sequence is generated quickly

Example: Normal distribution

Task: Get 5 normally distributed values

```
rnorm(5)
# example output
[1] -1.03966898  1.78222896 -2.31106908  0.87860458  0.03580672
```

What happens when I run `rnorm(5)` again?

```
rnorm(5)
# we get a different output! - for example:
[1]  1.0128287  0.4322652  2.0908192 -1.1999258  1.5896382
```

How can we get the same values over and over again?

```
set.seed(123)
rnorm(5)
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
set.seed(123)
rnorm(5)
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
```

The value in set.seed() can be any integer. If you use the same integer again, you will get the same "random" sequence again.

What you should know after day 9

sapply() and tapply()
A final word on functions

Some distributions in R
- Example: Normal distribution
- Random numbers in R

Statistical tests in R
Example: t-test

Statistical tests – a simple example

Task:
You want to show that a treatment is effective.

Information you have:
- You collected data for 2 groups of patients with and without treatment.
- 80% of the patients with treatment recovered, whereas only 30% of the patients without treatment recovered.

A pessimist now might say that this difference is just by chance.

What could you do to convince the pessimist that the treatment has an effect?

→ **You assume that the pessimist is right and you show that under this hypothesis the data would be very unlikely.**

In statistical words

The pessimist (by chance) thinking is the null hypothesis H_0.

What you want to show is the alternative hypothesis H_1.

→ Show that the observation and everything 'more extreme' is sufficiently unlikely under the null hypothesis. Many scientists have agreed that it suffices that this probability is at most 5%.

→ This outcome refutes the pessimist, in statistical language:
We reject the null hypothesis on a significance level of 5%.

p value = P(observation and everything more 'extreme' / H_0 is true)
→ If the *p* value is greater than 5%, you say that you cannot reject the null hypothesis.

Example: *t*-test

There is a huge variety of statistical tests that you can perform in R. As an example, we will cover the ***t*-test** in this lecture.

Main idea:
We have measured a variable in one ore two samples and we want to test:

- One-sample:
  test if the mean is equal to a certain value (often 0).
- Two-samples:
  test if the two means are equal.

In case of two samples there are two possibilities:

- The same individuals are measured twice (e.g., before and after a treatment):
  paired t-test
- Independent samples (e.g., two different groups of patients):
  unpaired t-test

Function `t.test()` in R

```
?t.test
t.test(x, y = NULL,
       alternative = c("two.sided", "less", "greater"),
       mu = 0, paired = FALSE, var.equal = FALSE,
       conf.level = 0.95, ...)
```

We will have two examples:

1)Two-samples: unpaired t-test
heartbeat.txt
group with treatment vs. group without treatment

2)Two-samples: paired t-test
data: shoes, package = MASS
The same persons have worn shoes of two different types (A and B) and we consider a measure of shoe damage.

Example 1

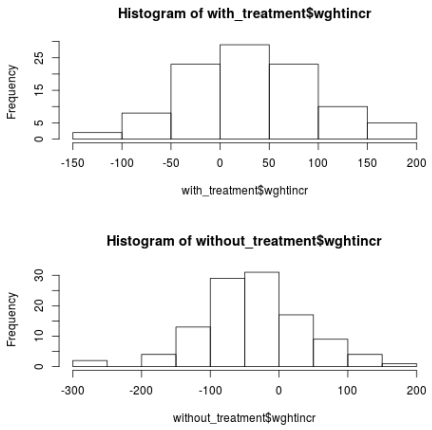
Data: `heartbeats.txt`

We want to compare the group that listened to heartbeats (`heartbeats$treatment == 1`, with treatment) to the group that did not listen to heartbeats (`heartbeats$treatment == 0`, without treatment)

The two groups contain different individuals.
→ Unpaired *t*-test

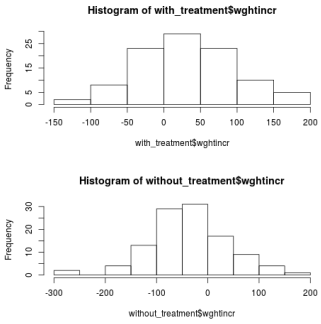
What do we expect?

Let us plot histograms to get a feeling for our data.



Version 1 – Histograms with default values

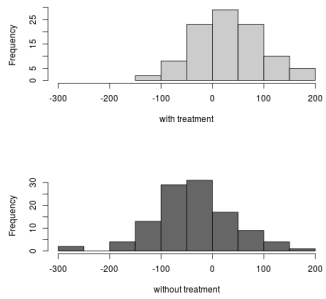
```
# Import the data
heartbeats <-
read.table("data/heartbeats.txt",
header = TRUE)

# Subset the data
with_treatment <-
subset(heartbeats,
heartbeats$treatment == 1)
without_treatment <-
subset(heartbeats,
heartbeats$treatment == 0)

# Plot an histogram for
weightincrease for both groups
# Version 1 - hist() with default
# values
# set the layout of the plot
png(file = "heartbeats_1.png")
par(mfrow = c(2, 1))
# histograms with default values
hist(with_treatment$wghtincr)
hist(without_treatment$wghtincr)
dev.off()
```



What do you suggest?

Version 2 – Histograms with adjusted x-axes

```
# set the layout of the plot
png(file = "heartbeats_2.png")
par(mfrow = c(2, 1))
# histograms with adjusted x-
# axes
hist(with_treatment$wghtincr,
     col = "grey80",
     main = "",
     xlab = "with treatment",
     xlim = c(-300, 200)
     )
hist(without_treatment$wghtincr,
     main = "",
     col = "grey40",
     xlab = "without treatment",
     xlim = c(-300, 200)
     )
dev.off()
```



Unpaired t-test

```
mytest1 <- t.test(without_treatment$wghtincr,
with_treatment$wghtincr)

mytest1

   Welch Two Sample t-test

data:  without_treatment$wghtincr and with_treatment$wghtincr
t = -6.9307, df = 206.3, p-value = 5.254e-11
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -88.91976 -49.53478
sample estimates:
mean of x mean of y
-31.72727  37.50000
```

p value. Access to it directly with `mytest1$p.value`

Degrees of freedom. Access to it directly with `mytest1$parameter`

Value of the *T* statistic. Access to it directly with `mytest1$statistic`

Mean of the two vectors. Acces to it directly with `mytest1$estimate`

Conclusion: We can reject the null hypothesis that the two groups had the same average weight increase.

Example 2

Data: `shoes` from the package `MASS`

From the help page (??shoes):
shoes {MASS}
Shoe wear data of Box, Hunter and Hunter
Description
A list of two vectors, giving the wear of shoes of materials A and B for one foot each of ten boys.
Usage
shoes
Source
G. E. P. Box, W. G. Hunter and J. S. Hunter (1978) Statistics for Experimenters. Wiley, p. 100
References
Venables, W. N. and Ripley, B. D. (2002) Modern Applied Statistics with S. Fourth edition. Springer.

The two groups contain the same individuals.
→ Paired *t*-test

Paired *t*-test

```
data(shoes, package = "MASS")
str(shoes)
List of 2
 $ A: num [1:10] 13.2 8.2 10.9 14.3 10.7 6.6 9.5 10.8 8.8 13.3
 $ B: num [1:10] 14 8.8 11.2 14.2 11.8 6.4 9.8 11.3 9.3 13.6

t.test(shoes$A, shoes$B, paired = TRUE)

   Paired t-test

data:  shoes$A and shoes$B
t = -3.3489, df = 9, p-value = 0.008539
alternative hypothesis: true difference in means is not equal
to 0
95 percent confidence interval:
 -0.6869539 -0.1330461
sample estimates:
mean of the differences -0.41
```

Conclusion: We can reject the null hypothesis that the two types of shoes have the same wear.

Take-home message

- The classic distributions can be found in R and you can plot them, draw random samples from them etc… (e.g. `dnorm`, `pnorm`, `qnorm`, `rnorm`)
- Most statistic tests can be found in R with `name.test()`
- Mind the options
- More in the STATISTIC lectures.

General information on the exam

Room: B00.019
Time: 10-11:30 (exam will last 90 minutes)

In total 100 points, you pass with 50%

7 parts

Getting started	12 points	Day 2
Data types and structures	12 points	Day 3
Reading and writing data	12 points	Day 4
Manipulating data	12 points	Day 5
Visualization	12 points	Day 6
Algorithmics (general and in R)	30 points	Days 1, 7, 8
Statistics	10 points	Day 9