Intro to R

Marc A.T. Teunis

2018-01-25

Using R in Life Sciences - January 2018

## R is case sensitive

*Important note: R is case-sensitive, meaning that it will matter whether you use capital or lower case, keep this in mind if you get a cryptic warning or error!*

## Help getting started with R

Writing R code can be challenging and espacially in the beginning you will get mind-boggling errors. R is extremely picky about capitals and every opening (, [ or { must be matched by a closing ), ] or }.

Mind that . is the decimal placeholder!

If you’re stuck, try the help. You can get help about any R function by running ?function\_name and ?package\_name or ??package\_name, skip down to the examples and look for code that matches what you’re trying to do.

Another great tool is Google: trying googling the error message, as it’s likely someone else has had the same problem, and has gotten help online.

## Statistics and R

* This course does not cover statistical applications of R.
* Allbeit that we will look at an example of a linear regression later on.
* Nowadays R is a versatile language and can be used for various applications, but it was originally designed as a language for performing statistical analyses.
* This is why very many statistical applcations are available for R.
* To start learning about statisics and R, I can highly recommend the book “Discovering Statistics Using R” by Dr. Andy Field: <https://uk.sagepub.com/en-gb/eur/discovering-statistics-using-r/book236067%20>

## For more on learning R in the context of statistics: start e.g. with:

<http://www.statsteachr.org/> or

<https://www.coursera.org/learn/statistical-inference/home/welcome> or

<https://www.youtube.com/watch?v=ACWuV16tdhY&index=21&list=PLqzoL9-eJTNBDdKgJgJzaQcY6OXmsXAHU> and

<https://www.youtube.com/watch?v=kvmSAXhX9Hs&index=29&list=PLqzoL9-eJTNBDdKgJgJzaQcY6OXmsXAHU>

and the rest of Mike Marin’s lectures, that are also a really great way to start learning R.

## Packages

## List of available CRAN packages

<http://cran.r-project.org/web/packages/available_packages_by_date.html>

For the course we use tidyverse a lot

# install.packages("tidyverse")  
library(tidyverse)

## ── Attaching packages ────────────────────────────────────────── tidyverse 1.2.1 ──

## ✔ ggplot2 2.2.1 ✔ purrr 0.2.4  
## ✔ tibble 1.3.4 ✔ dplyr 0.7.4  
## ✔ tidyr 0.7.2 ✔ stringr 1.2.0  
## ✔ readr 1.1.1 ✔ forcats 0.2.0

## ── Conflicts ───────────────────────────────────────────── tidyverse\_conflicts() ──  
## ✖ dplyr::filter() masks stats::filter()  
## ✖ dplyr::lag() masks stats::lag()

The tidyverse (in itself an R-package) is actually a collection of packages! See: <http://tidyr.tidyverse.org/> for more info

## Biological Packages

For Biological application go to <http://www.bioconductor.org>

Installing Bioconductor packages is different from CRAN packages example: {affy}

Each Bioconductor package has a landing page:

<https://www.bioconductor.org/packages/release/bioc/html/affy.html>

## installing `{affy}` package  
## try http:// if https:// URLs are not supported  
source("https://bioconductor.org/biocLite.R")  
biocLite("affy")  
  
## loading affy package and the vignettes pages of this package  
library(affy)  
browseVignettes("affy")

## Vignettes

Vignettes are long explanations and demos of a package. Commonly, a vignette contains examples and a workflow that shows how the package can be used and which (research) questions can be adressed with the functions and datasets in the package. It usually is a good place to start for examples. It also shows the so-called dependence of a package: it explains which other packages you would need and how the data should look to be able to work with the package the vignette belongs to.

## Getting Help for R functions and packages

install.packages("dplyr")  
library(dplyr)  
library(ggplot2)  
??dplyr  
??ggplot2  
?mean  
??mean # goes to the page with functions related with '.mean.'  
apropos("mean") # search on more options of or alternatives for a certain function

## Examples and demos on functions and packages

example(mean) # to see a worked example  
demo(graphics) # demonstration of R functions

## Functions; naming arguments

# ?rnorm  
  
## named arguments  
set.seed(1234)  
q <- rnorm(n = 10000, mean = 20, sd = 2)  
  
## no naming  
set.seed(1234)  
qq <- rnorm(10000, 20, 2)  
all(q == qq)

## [1] TRUE

**Write code for humans, be explicit!**

# Data objects

## Vectors

R is an object oriented language: meaning you can create and work with (manipulate/index/access) objects. Vectors are R’s elementary objects and come in different flavours:

* Nummeric vector: contains only numbers: decimal separator in R is “.” (decimal point) and not “,” (decimal comma) as is common in the English language.
* Character vector: contains only “words”, but words can also be numbers: “23” or other items “100%” or “$2,000.00”
* An integer vector: an nummeric series: 1, 2, 3 is an integer of length 3.
* Logical: logical vectors contain only two values: “TRUE” and/or “FALSE”
* Mixed: Vectors do not need to be of one type. They can be mixed. They can only be of one class, so this operation will induce ***coercion***.

## Numeric vectors

c(2,8,5) # combines its arguments to form a vector

## [1] 2 8 5

nv\_1 <- c(2,8,5) # assignment statement  
nv\_1 # view the content of the object "nv\_1"

## [1] 2 8 5

## Class tells you about the classification

nv\_2 <- c(8.4,5.6,10.1,13.1,2.5,7.8,15.2,3.8,20.9)  
nv\_2

## [1] 8.4 5.6 10.1 13.1 2.5 7.8 15.2 3.8 20.9

class(nv\_2)

## [1] "numeric"

# Number notations and rounding

## Scientific notations

big\_numbers <- rnorm(10, mean = 10000000, sd = 2)  
big\_numbers %>% formatC(format = "e", digits = 2)

## [1] "1.00e+07" "1.00e+07" "1.00e+07" "1.00e+07" "1.00e+07" "1.00e+07"  
## [7] "1.00e+07" "1.00e+07" "1.00e+07" "1.00e+07"

## Rounding numbers

small\_numbers <- runif(10, min = 0.001, max = 0.1) %>% print()

## [1] 0.019267222 0.053825642 0.082573099 0.094016565 0.083529459  
## [6] 0.031792997 0.050231930 0.062286384 0.032758184 0.005497985

small\_numbers %>% round(digits = 2)

## [1] 0.02 0.05 0.08 0.09 0.08 0.03 0.05 0.06 0.03 0.01

## Character vectors

cv <- c("this is", "an", "example of", "1", "character", "vector", "with length:", "length(cv)")  
cv

## [1] "this is" "an" "example of" "1"   
## [5] "character" "vector" "with length:" "length(cv)"

length(cv)

## [1] 8

class(cv)

## [1] "character"

## Integers

int <- as.integer(1:5)  
int

## [1] 1 2 3 4 5

length(int)

## [1] 5

class(int)

## [1] "integer"

## Logical vectors

lv <- c(TRUE, FALSE, TRUE, TRUE)  
lv

## [1] TRUE FALSE TRUE TRUE

class(lv)

## [1] "logical"

## logical vectors can also be converted to numeric vectors  
nlv <- as.numeric(lv)  
nlv

## [1] 1 0 1 1

## note that coercion of a logical to a numeric vector changes the "TRUE" value to 1 and the "FALSE" value to 0

## Manipulating vectors

You can add/subtract/devide or use other arithmetic functions on numeric vectors

a <- c(1,3,5,7,9)  
b <- c(2,4,6,8,10)  
z1 <- a - b  
z1

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

z2 <- b - a  
z2

## [1] 1 1 1 1 1

z3 <- a / b  
z3

## [1] 0.5000000 0.7500000 0.8333333 0.8750000 0.9000000

## Apply functions to a vector

z4 <- sum(a)  
z4

## [1] 25

z5 <- max(a) - max(b)  
z5

## [1] -1

## Series in base-R

numbers\_times <- rep(1:4, times = 4) %>% print()

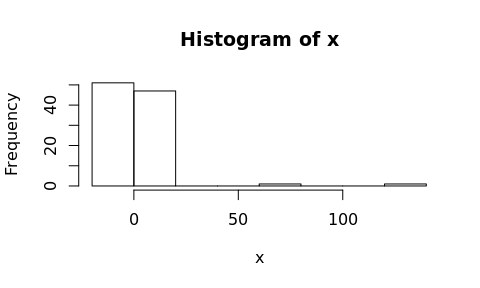
## [1] 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4

numbers\_each <- rep(1:4, each = 4) %>% print()

## [1] 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4

## Series in {tidyverse}

library(modelr)  
x <- rcauchy(100)  
hist(x)



## More {tidyverse} series

seq\_range(x, n = 10)

## [1] -19.611097 -3.441416 12.728264 28.897945 45.067626 61.237306  
## [7] 77.406987 93.576668 109.746349 125.916029

seq\_range(x, n = 10, trim = 0.1)

## [1] -4.6139456 -3.1951848 -1.7764241 -0.3576633 1.0610975 2.4798583  
## [7] 3.8986191 5.3173799 6.7361407 8.1549014

seq\_range(x, by = 1, trim = 0.1)

## [1] -4.6139456 -3.6139456 -2.6139456 -1.6139456 -0.6139456 0.3860544  
## [7] 1.3860544 2.3860544 3.3860544 4.3860544 5.3860544 6.3860544  
## [13] 7.3860544

## Make pretty {tidyverse} sequences

y <- runif(100)  
seq\_range(y, n = 10)

## [1] 0.002563953 0.113102416 0.223640879 0.334179342 0.444717804  
## [6] 0.555256267 0.665794730 0.776333193 0.886871656 0.997410118

seq\_range(y, n = 10, pretty = TRUE)

## [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

seq\_range(y, n = 10, expand = 0.5, pretty = TRUE)

## [1] -0.4 -0.2 0.0 0.2 0.4 0.6 0.8 1.0 1.2 1.4

## Indexing with […]

You can get the individual items of a vector by using the index []

x <- c(8, 5, 10, 13, 2, 7, 15, 3, 20, 8)  
x # create vector with 10 variables

## [1] 8 5 10 13 2 7 15 3 20 8

length(x)

## [1] 10

mode(x) # information on data mode (numeric, character, logic)

## [1] "numeric"

## Using the index

x

## [1] 8 5 10 13 2 7 15 3 20 8

x[3] ## creating a subset by indexing:

## [1] 10

x[c(3, 4, 7)] # apply a simple function

## [1] 10 13 15

mean(x[c(2,5)]) # example of a function

## [1] 3.5

## Using the vector index [] some more

i <- 5  
x[c(i, i+2)] # (i = 5 & i = 7), i-th element

## [1] 2 15

x[-2] # all but the ith (second) element

## [1] 8 10 13 2 7 15 3 20 8

x[3:5] # element 3 to 5

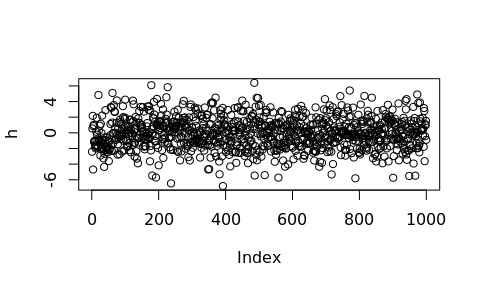
## [1] 10 13 2

x[x > 9] # all greater than some value

## [1] 10 13 15 20

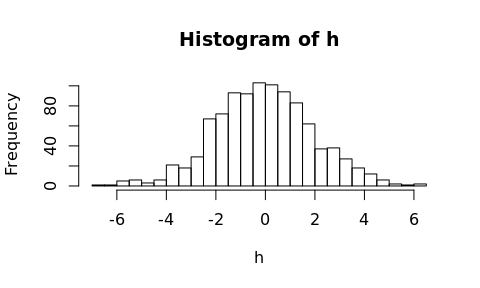
## Filtering elements from a vector by logical-indexing

set.seed(1234)  
h <- rnorm(n = 1000, mean = 0, sd = 2)  
plot(h)



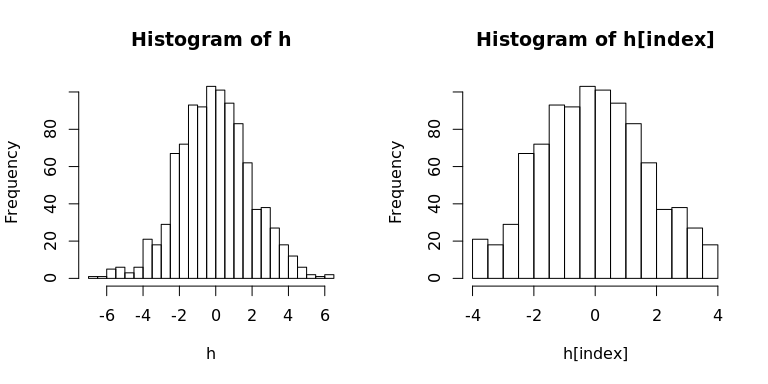
## Histogram

hist(h, breaks = 20)



## Truncate the distribution at -4 and 4

index <- h > -4 & h < 4  
par(mfrow = c(1,2))  
hist(h, breaks = 20)  
hist(h[index], breaks = 20)



## Concatenate or c() command

The c() function can be used to **concatenate**, **combine** or **create** objects in R

pp <- c(x, 0, x)  
pp

## [1] 8 5 10 13 2 7 15 3 20 8 0 8 5 10 13 2 7 15 3 20 8

sum(c(pp, c(1:5), c(1,2, c(1:10)))) # sum of elements in the 'concatenated' vector

## [1] 255

## Plotting series/vectors

c <- c(1:7,9,11) # dependent variable (response)  
d <- c(1:9) # independent variable (terms)  
  
# performing a linear correlation (response ~ terms)  
m1 <- lm(c ~ d)  
summary(m1)

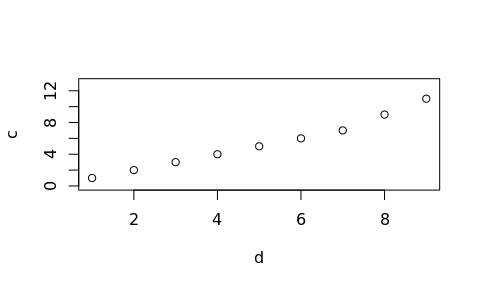
##   
## Call:  
## lm(formula = c ~ d)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -0.70000 -0.33333 0.03333 0.21667 0.93333   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) -0.58333 0.38670 -1.508 0.175   
## d 1.18333 0.06872 17.220 5.47e-07 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 0.5323 on 7 degrees of freedom  
## Multiple R-squared: 0.9769, Adjusted R-squared: 0.9736   
## F-statistic: 296.5 on 1 and 7 DF, p-value: 5.47e-07

m1$coefficients

## (Intercept) d   
## -0.5833333 1.1833333

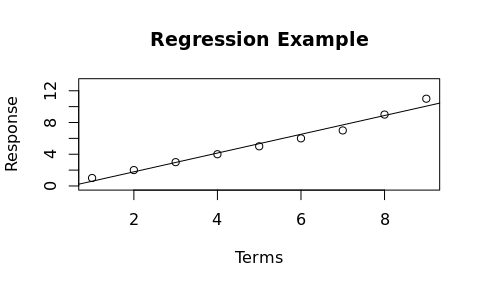
## Make a plot

plot(d, c, ylim=c(0,13))



## Plotting the model

plot(d, c, ylim=c(0,13), abline(m1),   
 main = "Regression Example",  
 xlab = "Terms", ylab = "Response") #abline plots the correlation model in the graph



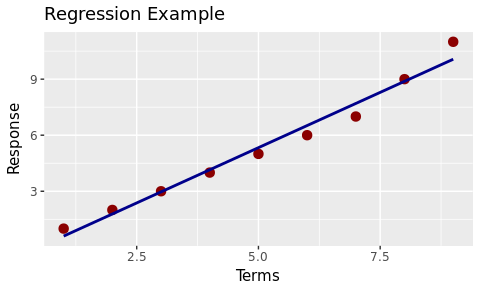
## ggplot2 version

During the course we will mostly use the ggplot2 plotting system! ggplot2 works with dataframes, so we first need to create one

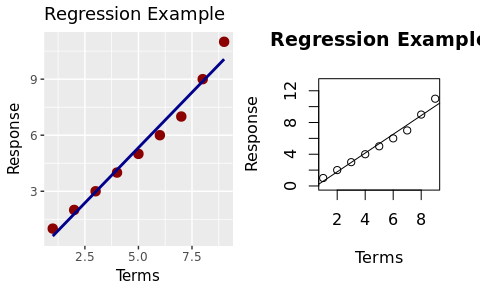
tbl\_df <- tibble(x = d, y = c)

## ggplot code

ggplot(data = tbl\_df, aes(x = x, y = y)) +  
 geom\_point(colour = "darkred", size = 3) +  
 geom\_smooth(method = "lm", se = FALSE, formula = y ~ x, colour = "darkblue") +  
 ggtitle("Regression Example") + xlab("Terms") + ylab("Response")



## Combine the two in one slide



## Handeling character vectors

What happens if you use arithmetic functions on character vectors and numeric vectors?

w <- c("1", "2")  
u <- c("a", "b", "c")  
z6 <- w-y   
z7 <- u+w

This does not work because the vectors are of different type. We can use the paste command for this

w <- c("1", "2")  
u <- c("a", "b", "c")  
z8 <- paste(u,w, sep = "\_")  
z8 ## because w is shorter than u, it get's recycled over u

## [1] "a\_1" "b\_2" "c\_1"

## Let’s clean up the workspace

rm(list=ls())  
root <- find\_root\_file(criterion = is\_rstudio\_project)  
## Note: never use this in code that is meant for others!!!

*The above is an effective way to clear all the items in the Global Environment, but is is not very friendly to use this in code you share with others: can you think of why?*

## Data Structures: Lists

When using R for data analysis you will most likely work with data in a matrix, an array, a list or even more likely: a dataframe.

A matrix is a table with only numeric values. An array consists of multiple matices. A list is collection of R objects of different data type. A dataframe is a table with variable names in the first row and observations in the consecutive rows. The columns in a dataframe represnet different variables.

The dataframe and the list are the most widely used datastructures when considering experimental Biological data.

## Create a List

lst <-list(first\_names = c(male = "Fred", female = "Mary"), no.children = 3, child\_ages=c(4,7,9),  
child\_names = c("Suzy", "Marvin", "Jane"), address = c("Pandmonium Alley 114, Chaosville"),  
marital\_status = TRUE)  
length(lst) # number of elements in the list

## [1] 6

head(lst, 2)

## $first\_names  
## male female   
## "Fred" "Mary"   
##   
## $no.children  
## [1] 3

## Accessing items in a list

lst[[1]] # 1st element of List

## male female   
## "Fred" "Mary"

lst[[3]][2] # second item of third element

## [1] 7

names(lst) # named elements in this list

## [1] "first\_names" "no.children" "child\_ages" "child\_names"   
## [5] "address" "marital\_status"

lst$child\_names # pull "named" elements from a list using `$` operator

## [1] "Suzy" "Marvin" "Jane"

## str() gives you the structure of an object

str(lst) # display structure of lst

## List of 6  
## $ first\_names : Named chr [1:2] "Fred" "Mary"  
## ..- attr(\*, "names")= chr [1:2] "male" "female"  
## $ no.children : num 3  
## $ child\_ages : num [1:3] 4 7 9  
## $ child\_names : chr [1:3] "Suzy" "Marvin" "Jane"  
## $ address : chr "Pandmonium Alley 114, Chaosville"  
## $ marital\_status: logi TRUE

The new RStudio Interface also enables interactive exploration of R-objects (demo)

## Selecting single elements in a list

To select a single element from a variable in a list

lst$child\_ages[3]

## [1] 9

lst[[6]][2] # returns the value of the second element for your variable

## [1] NA

## Looping over lists

purrr::map(lst, is.na)

## $first\_names  
## male female   
## FALSE FALSE   
##   
## $no.children  
## [1] FALSE  
##   
## $child\_ages  
## [1] FALSE FALSE FALSE  
##   
## $child\_names  
## [1] FALSE FALSE FALSE  
##   
## $address  
## [1] FALSE  
##   
## $marital\_status  
## [1] FALSE

## The Dataframe

The dataframe is the most widely used data structure in the context of experimental biology and chemistry. **Remember “Tidy data!”**

## Creating a data frame from individual vectors

people\_df <- data.frame(age=c(24, 27, 19, 34),   
 sex=c("F","F","M", "M"),   
 weight=c(64, 55, 80, 70),  
 names = c("Christa", "Suzan",   
 "Matt", "John"))  
head(people\_df)

## age sex weight names  
## 1 24 F 64 Christa  
## 2 27 F 55 Suzan  
## 3 19 M 80 Matt  
## 4 34 M 70 John

## Or the ‘tidyverse’ way

people\_tbl <- tibble::tibble(age=c(24, 27, 19, 34),   
 sex=c("F","F","M", "M"),   
 weight=c(64, 55, 80, 70),  
 names = c("Christa", "Suzan",   
 "Matt", "John"))  
people\_tbl

## # A tibble: 4 x 4  
## age sex weight names  
## <dbl> <chr> <dbl> <chr>  
## 1 24 F 64 Christa  
## 2 27 F 55 Suzan  
## 3 19 M 80 Matt  
## 4 34 M 70 John

## Viewing the contents of a dataframe

summary(people\_df)

## age sex weight names   
## Min. :19.00 F:2 Min. :55.00 Christa:1   
## 1st Qu.:22.75 M:2 1st Qu.:61.75 John :1   
## Median :25.50 Median :67.00 Matt :1   
## Mean :26.00 Mean :67.25 Suzan :1   
## 3rd Qu.:28.75 3rd Qu.:72.50   
## Max. :34.00 Max. :80.00

table(people\_df)

## , , weight = 55, names = Christa  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 64, names = Christa  
##   
## sex  
## age F M  
## 19 0 0  
## 24 1 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 70, names = Christa  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 80, names = Christa  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 55, names = John  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 64, names = John  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 70, names = John  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 1  
##   
## , , weight = 80, names = John  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 55, names = Matt  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 64, names = Matt  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 70, names = Matt  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 80, names = Matt  
##   
## sex  
## age F M  
## 19 0 1  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 55, names = Suzan  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 1 0  
## 34 0 0  
##   
## , , weight = 64, names = Suzan  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 70, names = Suzan  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0  
##   
## , , weight = 80, names = Suzan  
##   
## sex  
## age F M  
## 19 0 0  
## 24 0 0  
## 27 0 0  
## 34 0 0

head(people\_df) # gives the content of the data frame

## age sex weight names  
## 1 24 F 64 Christa  
## 2 27 F 55 Suzan  
## 3 19 M 80 Matt  
## 4 34 M 70 John

names(people\_df)

## [1] "age" "sex" "weight" "names"

str(people\_df)

## 'data.frame': 4 obs. of 4 variables:  
## $ age : num 24 27 19 34  
## $ sex : Factor w/ 2 levels "F","M": 1 1 2 2  
## $ weight: num 64 55 80 70  
## $ names : Factor w/ 4 levels "Christa","John",..: 1 4 3 2

people\_df$age # gives the content of the variable "age" from the data frame ""

## [1] 24 27 19 34

## Using index on dataframes

Using the index “[]” on a dataframe is a bit tricky. The dataframe always consists of rows and columns. Indexing a dataframe goes like:

dataframe[row number(s), column number(s)]

people\_df$age[1] # first element of this vector

## [1] 24

people\_df[,2] # content of 2nd variable (column) which is a character vector -> factor

## [1] F F M M  
## Levels: F M

people\_df[1,] # content of the 1st row

## age sex weight names  
## 1 24 F 64 Christa

# multiple indices  
people\_df[2:3, c(1,3)] # remember to use c

## age weight  
## 2 27 55  
## 3 19 80

## Import data into R

library(tidyverse)  
path\_to\_gender\_age\_data <- file.path(root, "data", "gender.txt")  
gender\_age <- read\_delim(path\_to\_gender\_age\_data,  
 delim = "/")

## Parsed with column specification:  
## cols(  
## gender = col\_character(),  
## age = col\_integer()  
## )

## Inspecting the dataframe

# getting the first few rows   
head(gender\_age)

## # A tibble: 6 x 2  
## gender age  
## <chr> <int>  
## 1 F 12  
## 2 F 23  
## 3 F 34  
## 4 M 12  
## 5 M 15  
## 6 M 17

# getting information on the variable, the dimensions  
str(gender\_age)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 17 obs. of 2 variables:  
## $ gender: chr "F" "F" "F" "M" ...  
## $ age : int 12 23 34 12 15 17 18 14 56 67 ...  
## - attr(\*, "spec")=List of 2  
## ..$ cols :List of 2  
## .. ..$ gender: list()  
## .. .. ..- attr(\*, "class")= chr "collector\_character" "collector"  
## .. ..$ age : list()  
## .. .. ..- attr(\*, "class")= chr "collector\_integer" "collector"  
## ..$ default: list()  
## .. ..- attr(\*, "class")= chr "collector\_guess" "collector"  
## ..- attr(\*, "class")= chr "col\_spec"

# selecting a variable  
gender\_age$gender

## [1] "F" "F" "F" "M" "M" "M" "F" "F" "M" "F" "M" "M" "M" "F" "F" "F" "M"

gender\_age$age

## [1] 12 23 34 12 15 17 18 14 56 67 89 34 45 12 3 5 4

gender\_age[1:6,2]

## # A tibble: 6 x 1  
## age  
## <int>  
## 1 12  
## 2 23  
## 3 34  
## 4 12  
## 5 15  
## 6 17

## read\_csv

CSV is a format of a data file that uses commas or semicolons as seprators for the columns.

library(readr)  
skin <- read\_csv(paste0(root, "/data/skincolumns.csv"))

## Parsed with column specification:  
## cols(  
## `Genotype A` = col\_double(),  
## `Genotype B` = col\_double()  
## )

str(skin)

## Classes 'tbl\_df', 'tbl' and 'data.frame': 15 obs. of 2 variables:  
## $ Genotype A: num 54.9 48.8 50.8 45 28.7 ...  
## $ Genotype B: num 30.5 24.8 24.2 27.9 18.8 ...  
## - attr(\*, "spec")=List of 2  
## ..$ cols :List of 2  
## .. ..$ Genotype A: list()  
## .. .. ..- attr(\*, "class")= chr "collector\_double" "collector"  
## .. ..$ Genotype B: list()  
## .. .. ..- attr(\*, "class")= chr "collector\_double" "collector"  
## ..$ default: list()  
## .. ..- attr(\*, "class")= chr "collector\_guess" "collector"  
## ..- attr(\*, "class")= chr "col\_spec"

## Inspecting the skin dataframe

head(skin) # content of the data frame

## # A tibble: 6 x 2  
## `Genotype A` `Genotype B`  
## <dbl> <dbl>  
## 1 54.89321 30.48837  
## 2 48.77277 24.75629  
## 3 50.75909 24.20443  
## 4 44.96176 27.94116  
## 5 28.74590 18.76385  
## 6 35.31823 25.51043

dim(skin)

## [1] 15 2

attributes(skin)

## $names  
## [1] "Genotype A" "Genotype B"  
##   
## $class  
## [1] "tbl\_df" "tbl" "data.frame"  
##   
## $row.names  
## [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
##   
## $spec  
## cols(  
## `Genotype A` = col\_double(),  
## `Genotype B` = col\_double()  
## )

summary(skin)

## Genotype A Genotype B   
## Min. :28.75 Min. :18.76   
## 1st Qu.:42.89 1st Qu.:23.95   
## Median :47.98 Median :25.16   
## Mean :48.22 Mean :25.73   
## 3rd Qu.:53.04 3rd Qu.:27.40   
## Max. :69.13 Max. :31.59   
## NA's :1

## ?read\_csv # help on the function  
  
## dataset contains an NA, some functions do not work with NAs:  
mean(skin$`Genotype A`)

## [1] 48.22066

mean(skin$`Genotype B`)

## [1] NA

# to remove the NA  
skin\_noNA <- na.omit(skin)  
mean(skin\_noNA$`Genotype B`)

## [1] 25.72858

## Smoking example with multiple variables

Create data frame with 4 variables: gender, smoke, age, weigth

## Create individual vectors

set.seed(6000) # set it at the same number, then we will all produce the same output  
gender<-c(sample(c(1,2), size=1000, replace=TRUE))  
smoke<-c(sample(c(1,2), size=1000, replace=TRUE))  
age<-c(sample(c(1,2,3), size=1000, replace=TRUE))  
weight<-round(rnorm(1000, mean=65, sd=10),1)  
smoking <- data.frame(cbind(gender, smoke, age, weight))  
  
head(smoking, 5)

## gender smoke age weight  
## 1 2 1 3 65.1  
## 2 2 1 1 66.9  
## 3 1 2 3 46.9  
## 4 1 1 3 70.5  
## 5 1 2 1 66.9

smoking[1:5,]

## gender smoke age weight  
## 1 2 1 3 65.1  
## 2 2 1 1 66.9  
## 3 1 2 3 46.9  
## 4 1 1 3 70.5  
## 5 1 2 1 66.9

## Replace numbers by descriptive labels

# replace numbers by characters  
smoking$genderf=factor(smoking$gender, labels=c("female","male"))  
table(smoking$genderf,smoking$gender)

##   
## 1 2  
## female 497 0  
## male 0 503

smoking$smokef=factor(smoking$smoke, labels=c("Y","N"))  
table(smoking$smokef,smoking$smoke)

##   
## 1 2  
## Y 521 0  
## N 0 479

smoking$agecf=factor(smoking$age, labels=c("A: <15","B: 15-30","C: >30"))  
table(smoking$agecf,smoking$age)

##   
## 1 2 3  
## A: <15 332 0 0  
## B: 15-30 0 355 0  
## C: >30 0 0 313

summary(smoking)

## gender smoke age weight   
## Min. :1.000 Min. :1.000 Min. :1.000 Min. :28.50   
## 1st Qu.:1.000 1st Qu.:1.000 1st Qu.:1.000 1st Qu.:58.40   
## Median :2.000 Median :1.000 Median :2.000 Median :65.60   
## Mean :1.503 Mean :1.479 Mean :1.981 Mean :65.47   
## 3rd Qu.:2.000 3rd Qu.:2.000 3rd Qu.:3.000 3rd Qu.:72.50   
## Max. :2.000 Max. :2.000 Max. :3.000 Max. :98.60   
## genderf smokef agecf   
## female:497 Y:521 A: <15 :332   
## male :503 N:479 B: 15-30:355   
## C: >30 :313   
##   
##   
##

## Subsetting data frame; the easier {dplyr} way

library(tidyverse)  
names(smoking)

## [1] "gender" "smoke" "age" "weight" "genderf" "smokef" "agecf"

smoking\_male <- smoking %>% filter(genderf == "male")  
smoking\_age\_gender <- smoking %>% select(genderf, agecf)

## Or the base-R way

library(tidyverse)  
names(smoking)

## [1] "gender" "smoke" "age" "weight" "genderf" "smokef" "agecf"

# create a logical index  
smoking\_male\_index <- smoking[, "genderf"] == "male"   
smoking\_male <- smoking[smoking\_male\_index, ]  
# slecting only age and gender  
smoking\_age\_gender <- smoking\_male[, c(1,3)]

## Summarizing data

For summarizing data there are many ways in R (as is the case with many operations). I find the dplyr way the most intuitive. Using dplyr has the advantage that you can easily built on existing function and examples, because the code is faily easy to understand.

## The pipe or %>% operator

* One thing that makes dplyr great is that it works with the %>% symbol. Which is in programming jargon also called the “pipe” symbol.
* The pipe symbol takes the previous result of an operation and put it in the next.
* The pipe symbol can replace the use of annoying round brackets: or “(parentheses)”

## Example

If we want to calculate the sum, of the square root of the mean, of two nummeric vectors (each of length = 1000) we could do:

library(dplyr)  
set.seed(12345)   
zzz <- sqrt(mean(x <- rnorm(10000, mean = 34, sd = 2))) +  
sqrt(mean(y <- rnorm(10000, mean = 23, sd = 4)))  
zzz

## [1] 10.6257

The above is almost impossible to read because of all the round brackets, note the closing brackets: 3 of them!!

## let’s try the dplyr way with the %>% (pipe) operator

set.seed(12345)  
qqq <- rnorm(10000, mean = 34, sd = 2) %>%   
 mean() %>%   
 sqrt() + rnorm(10000, mean = 23, sd = 4) %>%   
 mean() %>%   
 sqrt()  
qqq

## [1] 10.6257

zzz == qqq

## [1] TRUE

The dplyr way is a lot better is it not? Do you get what the above (dplyr-way) lines do?

## The dplyr way explained

1. the result of the fist vector (1000 random numbers, with mean of 34 and a sd of 2 is piped into the function mean, which calculates the mean of these 1000 numbers (which will be around 34)
2. From that the square root is taken by piping the mean in the function sqrt()
3. The result of the firt vector is added to the calculation of the second vector (which is done the same way as the first).
4. The result is qqq
5. Is qqq equal to zzz, which we calculated the “old-fashioned” mathematical way? YES!

## Let’s apply the above to our smoking dataset

* Usually, if we want to make a graph we need some sort of summarizing variable that indicates wat is going on, e.g. a group mean and standard deviation.
* From the smoking dataset we will genrate a new dataframe with average weight, grouped by gender, age and grouped by smoking or non smoking. We will use dplyr verse to do so.
* I will also show you how to sort the data, let’s say to decreasing weight, by using the dplyr function ‘arrange’.

## Summarty of the smoking data

## it is handy to have the names of the dataset variables at hand when writing pipes  
  
names(smoking)

## [1] "gender" "smoke" "age" "weight" "genderf" "smokef" "agecf"

str(smoking)

## 'data.frame': 1000 obs. of 7 variables:  
## $ gender : num 2 2 1 1 1 2 1 1 1 2 ...  
## $ smoke : num 1 1 2 1 2 2 1 2 2 1 ...  
## $ age : num 3 1 3 3 1 3 1 3 3 1 ...  
## $ weight : num 65.1 66.9 46.9 70.5 66.9 93 67.1 74.5 77.6 61.6 ...  
## $ genderf: Factor w/ 2 levels "female","male": 2 2 1 1 1 2 1 1 1 2 ...  
## $ smokef : Factor w/ 2 levels "Y","N": 1 1 2 1 2 2 1 2 2 1 ...  
## $ agecf : Factor w/ 3 levels "A: <15","B: 15-30",..: 3 1 3 3 1 3 1 3 3 1 ...

smoking\_weight <- smoking %>% select(genderf, smokef, weight, agecf) %>%  
 group\_by(genderf, smokef, agecf) %>% summarise(mean\_weight = mean(weight)) %>%  
 arrange(desc(mean\_weight))  
  
head(smoking\_weight)

## # A tibble: 6 x 4  
## # Groups: genderf, smokef [4]  
## genderf smokef agecf mean\_weight  
## <fctr> <fctr> <fctr> <dbl>  
## 1 female N C: >30 67.07625  
## 2 female Y A: <15 66.93647  
## 3 male Y B: 15-30 66.77500  
## 4 female Y C: >30 66.35316  
## 5 male Y C: >30 65.51786  
## 6 male N B: 15-30 65.32222

## Looking in detail at the pipe

Let’s look at what is going on above.

The smoking\_weight dataset contains newly created, more descriptive denominators, we will use those.

1. The first step is to inspect the dataframe to see if the grouping variables are set to “factor”
2. The second step is to select the proper variables to work with
3. Than the data is grouped by the genderf, agecf and by the smokef variables
4. Summarise creates a new variable mean\_weight that is what it says: the mean weight for the groups defined.
5. Finally, the data is sorted, according the newly created mean\_weight variable, in decending order “(dec(mean\_weight))”. If we want ascending order we can just use arrange(mean\_weight). Arrange has the default to order the variable in ascending order.