

R & RStudio

a concise introduction

Dr Umberto Noè

The University of Edinburgh

Introduction

Welcome

Thank you for attending the short course **R & RStudio: a concise introduction!**

About me

- Dr Umberto Noè
- Senior Teaching Coordinator (Statistics) in the Department of Psychology
- Background in Statistics and Machine Learning

What this course is:

- an introduction to the fundamentals of the R programming language
- an overview of the RStudio interactive development environment (IDE)

Who this course is for:

- someone that never used R

or

- someone that used R a long time ago and needs a refresher

Prerequisites

There is no assumed prior knowledge of R programming

You are only required to have:

- A basic understanding of descriptive statistics
- A basic understanding of data visualisation

Course overview

- **Day 1: Tuesday 6th December 2022**

- Where: Lister Learning and Teaching Centre, room 1.16
- Seminar 1: 9:00 - 11:00
- Installing R: 11:00 - 12:00
- Lunch break: 12:00 - 13:00
- Lab 1: 13:00 - 15:00

- **Day 2: Wednesday 7th December 2022**

- Where: Lister Learning and Teaching Centre, room 1.16
- Seminar 2.A: 9:00 - 11:00
- Lab 2: 11:00 - 13:00
- Lunch break: 13:00 - 14:00
- Seminar 2.B: 14:00 - 15:00

Seminar 1 (Day 1): R building blocks

- Introduction to R and RStudio
 - What are R and RStudio?
 - Installing R and RStudio
 - The RStudio panes
 - Packages (CRAN vs GitHub)
 - Executing R code in RStudio (Console, Script files, Quarto files)
 - Comments
- R as a calculator
- Getting help
- Variables and their class (Numeric, Character, Logical)
- Basic functions
- Fundamental containers (Vectors, Matrices, Lists)

Seminar 2.A (Day 2): Everything data

- Grammar of data
 - Cases and variables
 - rownames, colnames
- Data frames
- A small detour
 - Factors
 - Missing values
- Subsetting data frames
 - Numeric vs Logical indexing
- Transforming variables
- More R functions

Seminar 2.B (Day 2): Advanced topics

- Control statements
- Projects
- Paths
- Quarto

Seminar 1

What are R and RStudio?

R: Engine



RStudio: Dashboard



Installing R and RStudio

- Installation instructions vary with the operating system.
- Detailed guidance is provided in the following page:

<https://uoepsy.github.io/files/install-update-r.html>

The RStudio panes

Live demo

Packages

Live demo

- CRAN: `palmerpenguins`
- GitHub: `taylor`

Executing R code in RStudio

Live demo:

- Console
- Script file
- Quarto file

Providing help: comments

- Comment your code
- Comments are ignored by R
- Provide a helpful comment for a block of code, not every line of code

```
# This is a comment  
### This is also a comment
```

```
# This is a  
# multiline comment
```

R as a calculator

What will R print?

```
1 + 2
```

```
5 - 3
```

```
2 * 3
```

```
1/2
```

```
2^3
```

```
9^(1/2)
```

```
9^0.5
```

```
1 + 3 * 10
```

```
(1 + 3) * 10
```

```
sqrt(4)
```

R as a calculator

```
1 + 2
```

```
[1] 3
```

```
5 - 3
```

```
[1] 2
```

```
2 * 3
```

```
[1] 6
```

```
1/2
```

```
[1] 0.5
```

```
2^3
```

```
[1] 8
```

```
9^(1/2)
```

```
[1] 3
```

```
9^0.5
```

```
[1] 3
```

```
1 + 3 * 10
```

```
[1] 31
```

```
(1 + 3) * 10
```

```
[1] 40
```

```
sqrt(4)
```

```
[1] 2
```

Getting help: help pages

- Using the console

```
?sqrt  
help(sqrt)
```

- The F1 trick
- The Help pane
- The internet
 - Google and Stack Overflow
 - RStudio cheatsheets

Variables

What will R print?

```
x <- 3
```

```
x
```

```
y <- 2 * x
```

```
y
```

```
z <- x * y
```

```
z
```

```
a <- Y * 3
```

```
A
```

Variables - Careful!

`<-` is called the **assignment operator**

```
x <- 3
```

```
x
```

```
[1] 3
```

```
y <- 2 * x
```

```
y
```

```
[1] 6
```

```
z <- x * y
```

```
z
```

```
[1] 18
```

```
a <- Y * 3
```

```
A
```

```
# Error! Names are case sensitive!
```

Variables

= also works!

```
x = 3  
x
```

```
[1] 3
```

```
y = 2 * x  
y
```

```
[1] 6
```

```
z = x * y  
z
```

```
[1] 18
```


Variables

What will R print?

```
x <- 3  
y <- 2 * x  
z <- x * y
```

Variables

Nothing, as the names were not called.

```
x <- 3  
y <- 2 * x  
z <- x * y
```

Basics of functions

What is a function?

- You have already seen two examples of a function: `sqrt(4)` and `help(sqrt)`.

```
# sqrt() square root function
num <- sqrt(4)
num
```

```
[1] 2
```

```
# help() - a function that works on functions
help(sqrt)
```

```
# c() is the combine function
nums <- c(2, 5, 9)
nums
```

```
[1] 2 5 9
```

What is a function?

- A function is a transformation that takes something as an input, does some computation with the input, and returns an output.
 - `input`: is a value
 - `function_name`: is the name of the computation to do to the input
 - `output`: is the result of the computation

```
output <- function_name(input)
```

What is a function?

- Functions can also take more than one input

```
output <- function_name(input1, input2, input3)
```

What is a function? Example

What will R print?

```
res1 <- sum(1, 2, 3)
```

```
res1
```

```
res2 <- rep(10, times = 6)
```

```
res2
```

```
res3 <- sum(res2)
```

```
res3
```

What is a function? Example

```
res1 <- sum(1, 2, 3)  
res1
```

```
[1] 6
```

```
res2 <- rep(10, times = 6)  
res2
```

```
[1] 10 10 10 10 10 10
```

```
res3 <- sum(res2)  
res3
```

```
[1] 60
```


What is a function? Example

- Let's define a function that does what `*` does

- Recall that $a * b$ means $\underbrace{a + a + \dots + a}_{b \text{ times}}$

```
multiply <- function(input1, input2) {  
  numbers <- rep(input1, times = input2)  
  result <- sum(numbers)  
  return(result)  
}
```

- Always test!

```
10 * 5
```

```
[1] 50
```

```
multiply(10, 5)
```

```
[1] 50
```

Variables and their class

Numeric variables

What will R print?

```
x <- 3
x

y <- 2 * x
y

z <- x * y
z
```

```
data.class(x)
data.class(y)
data.class(z)
```

Numeric variables

```
x <- 3
```

```
x
```

```
[1] 3
```

```
y <- 2 * x
```

```
y
```

```
[1] 6
```

```
z <- x * y
```

```
z
```

```
[1] 18
```

```
data.class(x)
```

```
[1] "numeric"
```

```
data.class(y)
```

```
[1] "numeric"
```

```
data.class(z)
```

```
[1] "numeric"
```

Character variables

What will R print?

```
first_name <- "Umberto"  
first_name
```

```
department <- 'Psychology'  
department
```

```
nchar(department)  
length(department)
```

```
data.class(first_name)  
data.class(department)
```

Character variables

```
first_name <- "Umberto"  
first_name
```

```
[1] "Umberto"
```

```
department <- 'Psychology'  
department
```

```
[1] "Psychology"
```

```
nchar(department)
```

```
[1] 10
```

```
length(department)
```

```
[1] 1
```

```
data.class(first_name)
```

```
[1] "character"
```

```
data.class(department)
```

```
[1] "character"
```

Character variables

- What if you have an apostrophe withing a string of characters?

```
# Use double quotes
```

```
"Let's"
```

```
[1] "Let's"
```

```
# Or the escape symbol \'
```

```
'Let\'s'
```

```
[1] "Let's"
```

Logical variables

Logical variables are often obtained as the result of checking a condition.

What will R print?

```
TRUE  
FALSE  
!TRUE  
!FALSE  
2 == 3  
2 == 2
```

```
2 != 3  
2 != 2  
2 > 3 # also >= available  
2 < 3 # also <= available  
!(2 > 3)  
!(2 < 3)
```


Logical variables

```
TRUE
```

```
[1] TRUE
```

```
FALSE
```

```
[1] FALSE
```

```
!TRUE
```

```
[1] FALSE
```

```
!FALSE
```

```
[1] TRUE
```

```
2 == 3
```

```
[1] FALSE
```

```
2 == 2
```

```
[1] TRUE
```

```
2 != 3
```

```
[1] TRUE
```

```
2 != 2
```

```
[1] FALSE
```

```
2 > 3 # also >= available
```

```
[1] FALSE
```

```
2 < 3 # also <= available
```

```
[1] TRUE
```

```
!(2 > 3)
```

```
[1] TRUE
```

```
!(2 < 3)
```

```
[1] FALSE
```

Logical variables

```
data.class(TRUE)
```

```
[1] "logical"
```

```
data.class(FALSE)
```

```
[1] "logical"
```

Containers

In this section we will discuss different basic containers, designed to hold collections of values.

- A vector is a collection of values all of the same type
- A matrix is a tabular arrangement (in rows and columns) of values all of the same type
- A list is the most general container, and can also hold data of mismatching types as well as other containers.

A vector is a collection of values all of the same type.

```
# Create a vector with the combine function
```

```
c(1, 3, 5)
```

```
c(10, 11, 12, 13, 14, 15)
```

```
# sequence from 1 to 10
```

```
seq(10, 15)
```

```
seq(10, 15, by = 1)
```

Vectors

A vector is a collection of values all of the same type.

```
# Create a vector with the combine function  
c(1, 3, 5)
```

```
[1] 1 3 5
```

```
c(10, 11, 12, 13, 14, 15)
```

```
[1] 10 11 12 13 14 15
```

```
# sequence from 1 to 10  
seq(10, 15)
```

```
[1] 10 11 12 13 14 15
```

```
seq(10, 15, by = 1)
```

```
[1] 10 11 12 13 14 15
```

Vectors

```
# from:to is a short for seq(from, to, by = 1)
```

```
10:15
```

```
# you can combine vectors
```

```
c(1:4, 10)
```

```
# in steps of 2
```

```
seq(1, 10, by = 2)
```

```
# five equally spaced numbers
```

```
seq(0, 10, length.out = 5)
```

Vectors

```
# from:to is a short for seq(from, to, by = 1)
10:15
```

```
[1] 10 11 12 13 14 15
```

```
# you can combine vectors
c(1:4, 10)
```

```
[1] 1 2 3 4 10
```

```
# in steps of 2
seq(1, 10, by = 2)
```

```
[1] 1 3 5 7 9
```

```
# five equally spaced numbers
seq(0, 10, length.out = 5)
```

```
[1] 0.0 2.5 5.0 7.5 10.0
```


- An important function

```
years <- c(2007, 2008, 2009, 2010, 2011)
council <- c("Glasgow", "Fife", "Aberdeen")
length(years)
```

```
[1] 5
```

```
length(council)
```

```
[1] 3
```

- If you mix numbers and characters, everything will become a character!

```
data <- c(2, 6, "", 10)
data
```

```
[1] "2"  "6"  ""   "10"
```

Matrices

A matrix is a rectangular arrangement of values in rows and columns.

```
M <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2)
M
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

```
M <- matrix(c("a","b","c","d","e","f"),
            nrow = 2, ncol = 3, byrow = TRUE)
M
```

	[,1]	[,2]	[,3]
[1,]	"a"	"b"	"c"
[2,]	"d"	"e"	"f"

Lists

- General containers that can hold data of mismatching types, different lengths, and even other containers.

```
L <- list(  
  a = 1:10,  
  b = matrix(c("a","b","c","d","e","f"), nrow = 2)  
)
```

```
L
```

```
$a
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

```
$b
```

	[,1]	[,2]	[,3]
[1,]	"a"	"c"	"e"
[2,]	"b"	"d"	"f"

- General syntax:

```
list(item_name1 = item_value1,  
      item_name2 = item_value2, ...)
```

Indexing

Indexing

Indexing is the practice of retrieving a specific entry from a data container.

Indexing

```
# vector  
v <- c(11, 22, 33, 44, 55, 66)  
v
```

```
[1] 11 22 33 44 55 66
```

```
# matrix  
M <- matrix(c(11, 12, 13, 21, 22, 23, 31, 32, 33),  
            nrow = 3, ncol = 3)  
M
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

Indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")  
L
```

```
$a  
[1] 101 102 103
```

```
$b  
      [,1] [,2] [,3]  
[1,]   11   21   31  
[2,]   12   22   32  
[3,]   13   23   33
```

```
$c  
[1] "hello"
```


Vector: numerical indexing

What will R print?

```
v <- c(11, 22, 33, 44, 55, 66)
v
```

```
[1] 11 22 33 44 55 66
```

```
# vector[entries]
v[2]
v[c(1, 3)]
v[c(1, 2, 3)]
v[1:3]
```

Vector: numerical indexing

```
v <- c(11, 22, 33, 44, 55, 66)
```

```
# vector[entries]  
v[2]
```

```
[1] 22
```

```
v[c(1, 3)]
```

```
[1] 11 33
```

```
v[c(1, 2, 3)]
```

```
[1] 11 22 33
```

```
v[1:3]
```

```
[1] 11 22 33
```

Vector: logical indexing

What will R print?

```
v <- c(11, 22, 33, 44, 55, 66)
v
```

```
[1] 11 22 33 44 55 66
```

```
# vector[mask]
v[c(TRUE, TRUE, FALSE, FALSE, FALSE, FALSE)]
v[v < 30]
```

Vector: logical indexing

```
v <- c(11, 22, 33, 44, 55, 66)
```

```
# vector[mask]
```

```
v[c(TRUE, TRUE, FALSE, FALSE, FALSE, FALSE)]
```

```
[1] 11 22
```

```
v[v < 30]
```

```
[1] 11 22
```

Matrix: numerical indexing

What will R print?

```
# matrix
M <- matrix(c(11, 12, 13, 21, 22, 23, 31, 32, 33),
            nrow = 3, ncol = 3)
```

M

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
# matrix[rows, cols]
M[2,3]
M[c(1,2), 3]
M[c(1, 2, 3), 3]
```

```
M[, 3]
M[2, ]
M[c(1, 2), ] # or M[1:2, ]
```

Matrix: numerical indexing

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
# matrix[rows, cols]
```

```
M[2,3]
```

```
[1] 32
```

```
M[c(1,2), 3]
```

```
[1] 31 32
```

```
M[c(1, 2, 3), 3]
```

```
[1] 31 32 33
```

```
M[, 3]
```

```
[1] 31 32 33
```

```
M[2, ]
```

```
[1] 12 22 32
```

```
M[c(1, 2), ] # or M[1:2, ]
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32

Matrix: logical indexing

What will R print?

```
# matrix
M <- matrix(c(11, 12, 13, 21, 22, 23, 31, 32, 33),
            nrow = 3, ncol = 3)
M
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
M[c(TRUE, TRUE, FALSE), ]
M[c(TRUE, TRUE, FALSE),
  c(FALSE, FALSE, TRUE)]
```

```
M[, c(TRUE, TRUE, FALSE)]
M[c(TRUE, TRUE, FALSE), 2]
```

Matrix: logical indexing

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
M[c(TRUE, TRUE, FALSE), ]
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32

```
M[c(TRUE, TRUE, FALSE),  
  c(FALSE, FALSE, TRUE)]
```

```
[1] 31 32
```

```
M[, c(TRUE, TRUE, FALSE)]
```

	[,1]	[,2]
[1,]	11	21
[2,]	12	22
[3,]	13	23

```
M[c(TRUE, TRUE, FALSE), 2]
```

```
[1] 21 22
```


List: numerical indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")  
L
```

\$a

```
[1] 101 102 103
```

\$b

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

\$c

```
[1] "hello"
```

List: numerical indexing

What will R print?

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
L[1]
```

```
L[2]
```

```
L[[1]]
```

```
L[[2]]
```

List: numerical indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
L[1]
```

```
$a
```

```
[1] 101 102 103
```

```
L[2]
```

```
$b
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
L[[1]]
```

```
[1] 101 102 103
```

```
L[[2]]
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

List: named indexing

What will R print?

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
L["a"]  
L[["a"]]
```

```
L$a  
L$b
```

List: named indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
L["a"]
```

```
$a
```

```
[1] 101 102 103
```

```
L[["a"]]
```

```
[1] 101 102 103
```

```
L$a
```

```
[1] 101 102 103
```

```
L$b
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

List: mixed indexing

What will R print?

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
L$a  
L$a[2]
```

```
L[[2]]  
L[[2]][, 2]
```

List: mixed indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
L$a
```

```
[1] 101 102 103
```

```
L$a[2]
```

```
[1] 102
```

```
L[[2]]
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
L[[2]][, 2]
```

```
[1] 21 22 23
```

List: logical indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
# list[mask]  
L[c(TRUE, FALSE, TRUE)]
```


List: logical indexing

```
# list  
L <- list(a = 101:103, b = M, c = "hello")
```

```
# list[mask]  
L[c(TRUE, FALSE, TRUE)]
```

```
$a  
[1] 101 102 103
```

```
$c  
[1] "hello"
```

Useful functions

which

```
v <- c(11, 22, 33, 44, 55, 66)
v
```

```
[1] 11 22 33 44 55 66
```

```
v < 30
```

```
[1] TRUE TRUE FALSE FALSE FALSE FALSE
```

```
which(v < 30)
```

```
[1] 1 2
```

```
v[v < 30] # logical indexing
```

```
[1] 11 22
```

```
v[which(v < 30)] # numerical indexing
```

```
[1] 11 22
```

summary, mean, range

```
v <- c(11, 22, 33, 44, 55, 66)
v
```

```
[1] 11 22 33 44 55 66
```

```
summary(v)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
11.00	24.75	38.50	38.50	52.25	66.00

```
mean(v)
```

```
[1] 38.5
```

```
range(v)
```

```
[1] 11 66
```

lapply

```
L <- list(a = c(11, 22, 33),  
          b = "hello",  
          c = 21:30)
```

```
lapply(L, length)
```

```
$a  
[1] 3
```

```
$b  
[1] 1
```

```
$c  
[1] 10
```

vector names

```
pop <- c(4005, 4015, 4075)  
pop
```

```
[1] 4005 4015 4075
```

```
names(pop) <- c("2000", "2001", "2002")  
pop
```

```
2000 2001 2002  
4005 4015 4075
```

vector names

```
pop[pop > 4050]
```

2002

4075

```
pop[1]
```

2000

4005

```
pop["2000"]
```

2000

4005

list names

```
L <- list(a = 101:103, b = M, c = "hello")
```

```
L
```

```
$a
```

```
[1] 101 102 103
```

```
$b
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
$c
```

```
[1] "hello"
```


list names

```
names(L)
```

```
[1] "a" "b" "c"
```

```
names(L) <- c("A", "B", "C")
```

```
L
```

```
$A
```

```
[1] 101 102 103
```

```
$B
```

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
$C
```

```
[1] "hello"
```

matrix rownames / colnames

```
M <- matrix(c(11, 12, 13, 21, 22, 23, 31, 32, 33),  
            nrow = 3, ncol = 3)
```

M

	[,1]	[,2]	[,3]
[1,]	11	21	31
[2,]	12	22	32
[3,]	13	23	33

```
rownames(M) <- c("Child", "Teenager", "Adult")  
colnames(M) <- c("Left", "Right", "Ambidextrous")
```

M

	Left	Right	Ambidextrous
Child	11	21	31
Teenager	12	22	32
Adult	13	23	33

matrix rownames / colnames

```
M
```

	Left	Right	Ambidextrous
Child	11	21	31
Teenager	12	22	32
Adult	13	23	33

```
M["Child", "Right"]
```

```
[1] 21
```

```
M[c("Child", "Teenager"), "Right"]
```

Child	Teenager
21	22

rbind

```
rbind(  
  c(1, 3, 5),  
  c(11, 33, 55)  
)
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	11	33	55

```
M1 <- matrix(1:6, 2, 3)  
M2 <- matrix(11:16, 2, 3)  
rbind(M1, M2)
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6
[3,]	11	13	15
[4,]	12	14	16

cbind

```
cbind(  
  c(1, 3, 5),  
  c(11, 33, 55)  
)
```

	[,1]	[,2]
[1,]	1	11
[2,]	3	33
[3,]	5	55

```
M1 <- matrix(1:6, 2, 3)  
M2 <- matrix(11:16, 2, 3)  
cbind(M1, M2)
```

	[,1]	[,2]	[,3]	[,4]	[,5]	[,6]
[1,]	1	3	5	11	13	15
[2,]	2	4	6	12	14	16

rbind/cbind with names

- If you r/cbind vectors with names you will get:

```
v1 <- c(1, 3, 5)
names(v1) <- c("2010", "2011", "2012")
v2 <- c(11, 33, 55)
names(v2) <- c("2020", "2021", "2022")
```

`rbind(v1, v2)`

	2010	2011	2012
v1	1	3	5
v2	11	33	55

`cbind(v1, v2)`

	v1	v2
2010	1	11
2011	3	33
2012	5	55

- Change any names you don't like with `rownames() <-` or `colnames() <-`.

Seminar 2.A

Grammar of data

- Data are organised in:
 - units or cases (rows)
 - variables (columns)

	name	start	end	party
1	Eisenhower	1953-01-20	1961-01-20	Republican
2	Kennedy	1961-01-20	1963-11-22	Democratic
3	Johnson	1963-11-22	1969-01-20	Democratic
4	Nixon	1969-01-20	1974-08-09	Republican
5	Ford	1974-08-09	1977-01-20	Republican
6	Carter	1977-01-20	1981-01-20	Democratic
7	Reagan	1981-01-20	1989-01-20	Republican
8	Bush	1989-01-20	1993-01-20	Republican
9	Clinton	1993-01-20	2001-01-20	Democratic
10	Bush	2001-01-20	2009-01-20	Republican
11	Obama	2009-01-20	2017-01-20	Democratic
12	Trump	2017-01-20	2021-01-20	Republican

Dataframes

- Dataframes are data containers in which:
 - all columns must have the same length, i.e. number of cases
 - the columns (variables) can be of different type (numeric, character, ...)
- Internally, dataframes are lists in which the items all have the same length.

Dataframes

```
DF <- data.frame(  
  id = 1:3,  
  name = c("a", "b", "c")  
)  
DF
```

	id	name
1	1	a
2	2	b
3	3	c

```
L <- list(  
  id = 1:3,  
  name = c("a", "b", "c")  
)  
L
```

```
$id  
[1] 1 2 3  
  
$name  
[1] "a" "b" "c"
```

Indexing dataframes

```
DF <- data.frame(  
  id = 1:3,  
  name = c("a", "b", "c")  
)  
DF
```

	id	name
1	1	a
2	2	b
3	3	c

- Just like lists:

```
DF$name
```

```
[1] "a" "b" "c"
```

```
DF$name[2]
```

```
[1] "b"
```

- In addition:

```
DF[2, "name"]
```

```
[1] "b"
```

```
DF[2, 2]
```

```
[1] "b"
```

Palmer penguins data

- We will use data containing size measurements for adult foraging penguins near Palmer Station, Antarctica

```
penguins <- read.csv("data/palmer_penguins.csv")
```

```
dim(penguins)
```

```
[1] 344    8
```

```
nrow(penguins) # dim(penguins)[1]
```

```
[1] 344
```

```
ncol(penguins) # dim(penguins)[2]
```

```
[1] 8
```

```
names(penguins)
```

```
[1] "species"           "island"             "bill_length_mm"
[4] "bill_depth_mm"     "flipper_length_mm"  "body_mass_g"
[7] "sex"               "year"
```

Palmer penguins data

- Top six rows of the data:

```
head(penguins)    # To show 10 rows: head(penguins, n = 10)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
1	Adelie	Torgersen	39.1	18.7	181	
2	Adelie	Torgersen	39.5	17.4	186	
3	Adelie	Torgersen	40.3	18.0	195	
4	Adelie	Torgersen	NA	NA	NA	
5	Adelie	Torgersen	36.7	19.3	193	
6	Adelie	Torgersen	39.3	20.6	190	

	sex	year
1	male	2007
2	female	2007
3	female	2007
4	<NA>	2007
5	female	2007
6	male	2007

Palmer penguins data

- Dataframe structure

```
str(penguins)
```

```
'data.frame':   344 obs. of  8 variables:
 $ species      : chr  "Adelie" "Adelie" "Adelie" "Adelie" ...
 $ island       : chr  "Torgersen" "Torgersen" "Torgersen" "Torge
 $ bill_length_mm : num  39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1
 $ bill_depth_mm : num  18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 2
 $ flipper_length_mm: int  181 186 195 NA 193 190 181 195 193 190 ...
 $ body_mass_g   : int  3750 3800 3250 NA 3450 3650 3625 4675 3475
 $ sex           : chr  "male" "female" "female" NA ...
 $ year          : int  2007 2007 2007 2007 2007 2007 2007 2007 20
```

A small detour: factors and NAs

- Factor: tells R that a variable is **categorical**
- Missing value: NA, short for **not available**

Factors

- To make a variable a factor:

```
penguins$species <- factor(penguins$species)
penguins$island <- factor(penguins$island)
penguins$sex <- factor(penguins$sex)
```

```
str(penguins)
```

```
'data.frame':   344 obs. of  8 variables:
 $ species      : Factor w/ 3 levels "Adelie","Chinstrap",...: 1 1 1 1 1 1 1 1 1 1
 $ island       : Factor w/ 3 levels "Biscoe","Dream",...: 3 3 3 3 3 3 3 3 3 3
 $ bill_length_mm: num   39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 34.1
 $ bill_depth_mm: num   18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 18.1
 $ flipper_length_mm: int   181 186 195 NA 193 190 181 195 193 190 ...
 $ body_mass_g   : int   3750 3800 3250 NA 3450 3650 3625 4675 3475 3475
 $ sex          : Factor w/ 2 levels "female","male": 2 1 1 NA 1 1 1 1 1 1
 $ year         : int    2007 2007 2007 2007 2007 2007 2007 2007 2007 2007
```


Factors

- Alternatively, you can use:

```
penguins <- transform(penguins,  
                      species = factor(species),  
                      island = factor(island),  
                      sex = factor(sex))
```

```
str(penguins)
```

```
'data.frame':   344 obs. of  8 variables:  
 $ species      : Factor w/ 3 levels "Adelie","Chinstrap",...: 1 1 1 1 1 1 1 1 1 1 ...  
 $ island       : Factor w/ 3 levels "Biscoe","Dream",...: 3 3 3 3 3 3 3 3 3 3 ...  
 $ bill_length_mm: num   39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 31.1 ...  
 $ bill_depth_mm: num   18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 18.7 ...  
 $ flipper_length_mm: int   181 186 195 NA 193 190 181 195 193 190 ...  
 $ body_mass_g   : int   3750 3800 3250 NA 3450 3650 3625 4675 3475 3325 ...  
 $ sex           : Factor w/ 2 levels "female","male": 2 1 1 NA 1 1 1 1 1 1 ...  
 $ year          : int    2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

Factors

To assign new labels to the levels (possible categories):

```
penguins$sex <- factor(penguins$sex,  
                      levels = c("male", "female"),  
                      labels = c("M", "F"))
```

or

```
penguins <- transform(  
  penguins,  
  sex = factor(sex,  
               levels = c("male", "female"),  
               labels = c("M", "F"))  
)
```

Factors

```
str(penguins)
```

```
'data.frame':   344 obs. of  8 variables:
 $ species      : Factor w/ 3 levels "Adelie","Chinstrap",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ island       : Factor w/ 3 levels "Biscoe","Dream",...: 3 3 3 3 3 3 3 3 3 3 ...
 $ bill_length_mm : num  39.1 39.5 40.3 NA 36.7 39.3 38.9 39.2 34.1 35.6 ...
 $ bill_depth_mm  : num  18.7 17.4 18 NA 19.3 20.6 17.8 19.6 18.1 20.6 ...
 $ flipper_length_mm: int  181 186 195 NA 193 190 181 195 193 190 ...
 $ body_mass_g    : int  3750 3800 3250 NA 3450 3650 3625 4675 3475 4050 ...
 $ sex           : Factor w/ 2 levels "M","F": 1 2 2 NA 2 1 2 1 NA 2 ...
 $ year          : int   2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
```

Factors

```
head(penguins)
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
1	Adelie	Torgersen	39.1	18.7	181	
2	Adelie	Torgersen	39.5	17.4	186	
3	Adelie	Torgersen	40.3	18.0	195	
4	Adelie	Torgersen	NA	NA	NA	
5	Adelie	Torgersen	36.7	19.3	193	
6	Adelie	Torgersen	39.3	20.6	190	

	sex	year
1	M	2007
2	F	2007
3	F	2007
4	<NA>	2007
5	F	2007
6	M	2007

Missing values

- Finding NAs

```
summary(penguins)
```

species	island	bill_length_mm	bill_depth_mm
Adelie :152	Biscoe :168	Min. :32.10	Min. :13.10
Chinstrap: 68	Dream :124	1st Qu.:39.23	1st Qu.:15.60
Gentoo :124	Torgersen: 52	Median :44.45	Median :17.30
		Mean :43.92	Mean :17.15
		3rd Qu.:48.50	3rd Qu.:18.70
		Max. :59.60	Max. :21.50
		NA's :2	NA's :2

flipper_length_mm	body_mass_g	sex	year
Min. :172.0	Min. :2700	M :168	Min. :2007
1st Qu.:190.0	1st Qu.:3550	F :165	1st Qu.:2007
Median :197.0	Median :4050	NA's: 11	Median :2008
Mean :200.9	Mean :4202		Mean :2008
3rd Qu.:213.0	3rd Qu.:4750		3rd Qu.:2009
Max. :231.0	Max. :6300		Max. :2009
NA's :2	NA's :2		

Missing values

- Missing values are a concern as they affect any subsequent computations:

```
11 + NA + 3
```

```
[1] NA
```

```
iq <- c(101, NA, 105)  
mean(iq)
```

```
[1] NA
```

```
sd(iq)
```

```
[1] NA
```

Handling missing values

- A possible solution:

```
mean(iq, na.rm = TRUE)
```

```
[1] 103
```

```
sd(iq, na.rm = TRUE)
```

```
[1] 2.828427
```

Handling missing values

```
dim(penguins)
```

```
[1] 344  8
```

- Remove rows having missing values in **any** variable:

```
penguins_clean <- na.omit(penguins)
dim(penguins_clean)
```

```
[1] 333  8
```

- Remove rows having missing values in variables **bill_length_mm**, **sex**:

```
id_clean <- complete.cases(penguins$bill_depth_mm,
                           penguins$sex) # TRUE or FALSEs
penguins_clean <- penguins[id_clean, ]
dim(penguins_clean)
```

```
[1] 333  8
```


Shortcut

- Subset the penguins data to only keep the rows for which we have complete cases on bill_depth_mm and sex

```
penguins_clean <- subset(penguins,  
                          complete.cases(bill_depth_mm, sex))  
dim(penguins_clean)
```

```
[1] 333    8
```

Subset function

- Keep rows where `rows_mask` is `TRUE`, and keep columns in select:

```
subset(<dataframe>, <rows_mask>, select = <columns to keep>)
```

- Keep rows where `rows_mask` is `TRUE`, and drop columns with a minus in select:

```
subset(<dataframe>, <rows_mask>, select = -<columns to drop>)
```

Indexing data frames: numeric

```
penguins_clean[1:2, ]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
1	Adelie	Torgersen	39.1	18.7	181	3750
2	Adelie	Torgersen	39.5	17.4	186	3800

	sex	year
1	M	2007
2	F	2007

```
penguins_clean[c(1, 10), ]
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
1	Adelie	Torgersen	39.1	18.7	181	3750
15	Adelie	Torgersen	34.6	21.1	198	3250

	sex	year
1	M	2007
15	M	2007

Indexing data frames: numeric

```
penguins_clean[1:3, 1]
```

```
[1] Adelie Adelie Adelie  
Levels: Adelie Chinstrap Gentoo
```

```
penguins_clean[1:3, c(1, 2, 3)]
```

	species	island	bill_length_mm
1	Adelie	Torgersen	39.1
2	Adelie	Torgersen	39.5
3	Adelie	Torgersen	40.3

```
penguins_clean[1:3, 1:3]
```

	species	island	bill_length_mm
1	Adelie	Torgersen	39.1
2	Adelie	Torgersen	39.5
3	Adelie	Torgersen	40.3

Caution

- If you select only one column, the result is not a data frame anymore, but a vector:

```
head(penguins[, 3])
```

```
[1] 39.1 39.5 40.3    NA 36.7 39.3
```

- `drop = FALSE` ensures the result is still a dataframe

```
head(penguins[, 3, drop = FALSE])
```

	bill_length_mm
1	39.1
2	39.5
3	40.3
4	NA
5	36.7
6	39.3

Indexing data frames: logical

- Logical subsetting is typically used to subset the data to only keep the rows satisfying a specific condition.
- For example, to keep only the rows where there are no missing values for `bill_length_mm`:

```
id <- complete.cases(penguins$bill_length_mm)
head(id) # first 6 values
```

```
[1] TRUE TRUE TRUE FALSE TRUE TRUE
```

```
penguins_clean2 <- penguins[id, ]
```

- or

```
penguins_clean2 <- subset(penguins,
                           complete.cases(bill_length_mm))
```

Key points

- After you read the data, always check for:
 - Variables that should be factors
 - If there are any variables with missing values

Transforming variables

- We have seen an example of transforming variables, when we changed some variables to be factors:

```
data$variable <- factor(data$variable)
```

- or

```
data <- transform(data,  
                  factor(variable))
```

- in general, `factor(...)` can be replaced by a generic computation on a variable.

Transforming variables

We can use the same logic to transform variables, i.e. changing existing ones or creating new ones.

- To create a new variable, use a new variable name

```
penguins$bill_depth_cm <- penguins$bill_depth_mm / 100  
  
# or  
  
penguins <- transform(penguins,  
                      bill_depth_cm = bill_depth_mm / 100)
```

Transforming variables

- To change an existing variable, overwrite the result into the same name

```
penguins$body_mass_g <- penguins$body_mass_g / 1000  
  
# or  
  
penguins <- transform(penguins,  
                      body_mass_g = body_mass_g / 1000)
```

- **NOTE:** You should not do as above, as the label is now misleading. You should have created a new variables called `body_mass_kg`

More R functions

- Create a function that takes as input the dataframe and species name, and outputs a vector of average bill length for each year.

More R functions

```
get_averages <- function(data, species) {  
  
  data <- data[data$species == species,  
               c("year", "bill_length_mm")]  
  
  data_2007 <- data[data$year == 2007, ]  
  data_2008 <- data[data$year == 2008, ]  
  data_2009 <- data[data$year == 2009, ]  
  
  M_2007 <- mean(data_2007$bill_length_mm, na.rm = TRUE)  
  M_2008 <- mean(data_2008$bill_length_mm, na.rm = TRUE)  
  M_2009 <- mean(data_2009$bill_length_mm, na.rm = TRUE)  
  
  all_means <- c(M_2007, M_2008, M_2009)  
  names(all_means) <- c("2007", "2008", "2009")  
  
  return(all_means)  
}
```

More R functions

```
all_means <- rbind(  
  get_averages(penguins, "Adelie"),  
  get_averages(penguins, "Chinstrap"),  
  get_averages(penguins, "Gentoo")  
)  
rownames(all_means) <- c("Adelie", "Chinstrap", "Gentoo")  
all_means
```

	2007	2008	2009
Adelie	38.82449	38.56000	38.98269
Chinstrap	48.72308	48.70000	49.05417
Gentoo	47.01471	46.93696	48.50000

Pretty printing

- Use the kable function from the knitr package.

```
library(knitr)
```

```
kable(all_means, row.names = TRUE, digits = 2)
```

	2007	2008	2009
Adelie	38.82	38.56	38.98
Chinstrap	48.72	48.70	49.05
Gentoo	47.01	46.94	48.50

- It has options to:
 - show/hide row and column names (row.names = TRUE/FALSE, col.names = TRUE/FALSE)
 - round the results to a specific number of digits (e.g., digits = 2)

Shortcut - tapply

```
tapply(numeric_variable, groups, function)
```

```
all_means2 <- tapply(penguins$bill_length_mm,  
                     list(penguins$species, penguins$year),  
                     function(x) mean(x, na.rm = TRUE))
```

```
all_means2
```

	2007	2008	2009
Adelie	38.82449	38.56000	38.98269
Chinstrap	48.72308	48.70000	49.05417
Gentoo	47.01471	46.93696	48.50000

Shortcut - tapply

```
knitr::kable(all_means2, row.names = TRUE, digits = 2)
```

	2007	2008	2009
Adelie	38.82	38.56	38.98
Chinstrap	48.72	48.70	49.05
Gentoo	47.01	46.94	48.50

Seminar 2.B

Control statements

- Control statements allow you to control the execution of the code until a certain condition is met.
- For: run code for a range of cases
- While: run code while a condition is true. Stop when it is false.
- If: run code if a condition is true

For

```
for (i in 1:10) {  
  print(c(i, i^2))  
}
```

```
[1] 1 1  
[1] 2 4  
[1] 3 9  
[1] 4 16  
[1] 5 25  
[1] 6 36  
[1] 7 49  
[1] 8 64  
[1] 9 81  
[1] 10 100
```

```
counter <- matrix(NA, 10, 2)
colnames(counter) <- c("index", "squared_index")

for (i in 1:10) {
  counter[i, ] <- c(i, i^2)
}

counter
```

	index	squared_index
[1,]	1	1
[2,]	2	4
[3,]	3	9
[4,]	4	16
[5,]	5	25
[6,]	6	36
[7,]	7	49
[8,]	8	64
[9,]	9	81
[10,]	10	100

```
for (species in c("Adelie", "Chinstrap", "Gentoo")) {  
  avgs <- get_averages(penguins, species)  
  print(avgs)  
}
```

2007	2008	2009
38.82449	38.56000	38.98269
2007	2008	2009
48.72308	48.70000	49.05417
2007	2008	2009
47.01471	46.93696	48.50000

For

```
M <- matrix(NA, nrow = 3, ncol = 3)
species <- c("Adelie", "Chinstrap", "Gentoo")

for (i in 1:length(species)) {
  M[i, ] <- get_averages(penguins, species[i])
}
```

M

	[,1]	[,2]	[,3]
[1,]	38.82449	38.56000	38.98269
[2,]	48.72308	48.70000	49.05417
[3,]	47.01471	46.93696	48.50000

While

```
i <- 1
while (i < 10) {
  print(i)
  i <- i + 1
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
```

```
x <- -5

if (x < 0) {
  print("Negative")
} else {
  print("Positive")
}
```

```
[1] "Negative"
```

```
x <- 8

if (x < 0) {
  print("Negative")
} else {
  print("Positive")
}
```

```
[1] "Positive"
```


- Live demo
 - Paths tell R where the file is located on the PC
 - `setwd()` and `getwd()`

Projects

- Live demo
 - How to organise your work

- Live demo
 - Computational documents