STS ST07
Programmation en R

Rebecca Dauwe

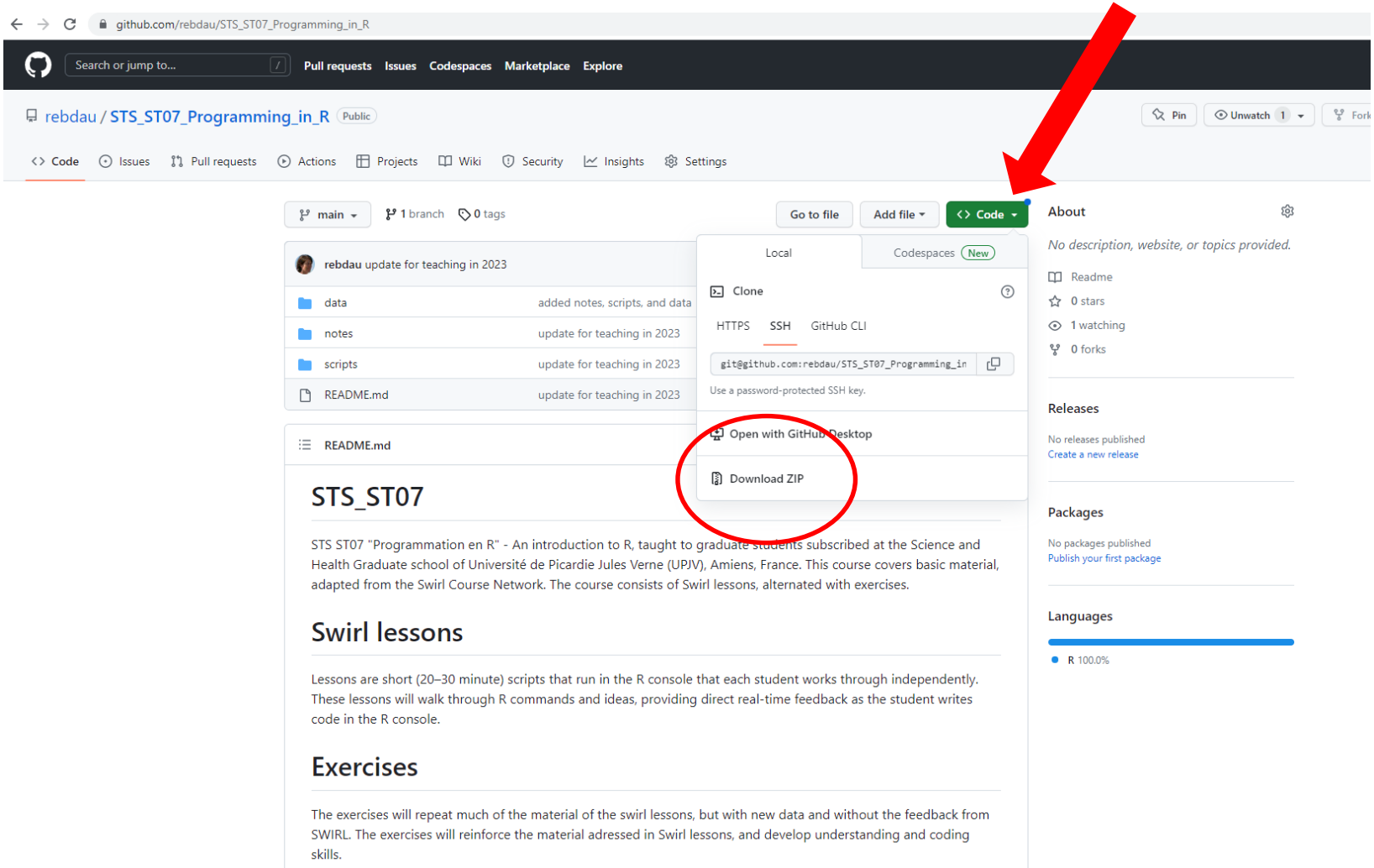
2024

app.wooclap.com/RCODING

Download course

https://github.com/rebdau/STS_ST07_Programming_in_R

Green button



I. Getting started

Starting

- Start R
- Add packages
  - A small handful of `packages` is loaded into memory upon startup
  - Huge number of « add-on `packages` »: additional statistical and graphical tools

→reasonable amount of functionality is available, and when we want more, we can get it

Packages

1) Install the package in your library
2) Load the package to use it

```
> library(package)
```

Let’s install the swirl package

- `swirl` is an `R package` that we will use for learning R interactively**:**
- You work directly in R console = real working environment
- Each swirl lesson takes about 15 min.
  - •multiple choice and text-based questions
  - •questions that require you to enter actual R code
  - •Immediate evaluation of code and immediate feedback

```
> install.packages("swirl")
```

II. First swirl lesson:

Basic Building Blocks

1. Install SWIRL package

This is done already; you only need to do this once.

```
> install.packages("swirl")
```

2. Install the lessons

Lessons for STS ST07 are on my GitHub page, in the repository “STS_ST07"

You only need to do this once (for now, until we find errors, etc)

```
# load the swirl package

> library(swirl)
# access github and download the course

# this is a function from the swirl package!

# This command downloads the first set of lessons to your computer

install_course_github("rebdau", "STS_ST07_swirl")
```

Follow the instructions in
STS ST07 Introduction to swirl.pdf

3. Start swirl

You will need to do this every time you start R or want to continue an old lesson or start a new lesson.

```
# load the swirl package into your current R session
library(swirl)
# this is what you will get:
# library(swirl) | Hi! Type swirl() when you are ready to begin
# swirl is a function, so you need '()'
swirl()
# this is what you will get:
# | Welcome to swirl! Please sign in. If you've been here before,
use the same
# | name as you did then. If you are new, call yourself something
unique.

What shall I call you?
```

4. Choose a name

Enter your name.
I will have to identify you, so please use **firstname lastname**
This name will also allow you to continue lessons if you stop them in the middle.

5. Choose a course

```
| Please choose a course, or type 0 to exit swirl.
1: STS_ST07_swirl
2: Take me to the swirl course repository! Selection:
```

We will be working through the lessons in the ‘STS_ST07’ course.
Type: ‘1’

6. Choose a lesson

```
| Please choose a lesson, or type 0 to return to course menu.
1: Basic Building Blocks
2. ...
```

Choose the first lesson: Basic Building Blocks
Type: ‘1’

7. Do the lesson!

Hit ‘Enter’ to advance when presented with ‘…’
The screen also shows you how far through the lesson you are (0%).

8. Completing the lesson

You will need to be connected to the internet to submit your lesson

When you are done, the last question will ask if you want to submit your answers to me to verify that your completed the lesson.

This will bring up a new web page, a Google form.

Scroll down, and click ‘submit’.

This will send an excrypted response to the Google form so that I can verify you completed the lesson.

Some useful commands for swirl

bye()
Exit swirl

play()
Leave swirl temporarily and gain access to the console again

nxt()
Return to swirl after playing

main()
Return to the main menu

info()
Display a list of these special commands

These commands are also listed in STS ST07 Introduction to swirl.pdf

==========

ORDER OF LESSONS AND EXERCISES

==========

1: Basic Building Blocks
2: Exercise 1
3: Workspace Directories and Files
4: Exercise 2
5: Data classes
6: Logic
7: Sequences of Numbers
8: Subsetting Vectors
9: Matrices and Data Frames
10: Exercise 3
11:Exercise 4
12: Lists
13: Lists example
14: lapply and sapply
15: vapply and tapply
16: Functions
17: Exercise 5

1. Something about **functions** and **objects**
2. Good practice: some coding **conventions**
3. History
4. Getting help

Something about **functions** and **objects**

```
> install.packages("swirl")
--- Please select a CRAN ...

> args(install.packages)
function (pkgs, lib, repos = getOption("repos"), contriburl = contrib.url(repos,
    type), method, available = NULL, destdir = NULL, dependencies = NA,
    type = getOption("pkgType"), configure.args = getOption("configure.args"),
    configure.vars = getOption("configure.vars"), clean = FALSE,
    Ncpus = getOption("Ncpus", 1L), verbose = getOption("verbose"),
    libs_only = FALSE, INSTALL_opts, quiet = FALSE, keep_outputs = FALSE,
    ...)
```

`install.packages()` is a `function`
`"swirl"` is an `argument`

`args()` is also a `function`, used to ask the arguments of a function.

An argument must have double quotes (`""`) if it is something not yet known by R.
An argument does not have quotes if it is something known by R:

- A function that is part of a loaded package (`install.packages`)
- An object stored in the workspace

```
> mypack <- "swirl"
> mypack
[1] "swirl"
> install.packages(mypack)
--- Please select a CRAN mirror for use in this session ---
trying URL
'https://ftp.belnet.be/mirror/CRAN/bin/windows/contrib/4.1/swirl_2.4.5.zip'
…
```

`mypack <- "swirl"` creates the object `mypack` in the workspace.
By typing `mypack` I ask what this object is .
The object `mypack` is just the word `"swirl"`

Now R knows `mypack`
I can now use the name of this `object` (without quotes!) instead of typing `"swirl"`
as an argument

Conventions

1. The equals sign = does work for assignment, but it is also used for other things, for example in passing arguments.

```
> a = 1:10
> a
[1]  1  2  3  4  5  6  7  8  9 10
> b <- 11: 20
> b
[1] 11 12 13 14 15 16 17 18 19 20
```

```
install_course_github(github_username = "rebdau", course_name = "STS_ST07")
```

The arrow <- is only used for assignment. **Please use it for assignment**.

Conventions

2. Use **spaces** liberally between arguments, between objects, between arithmetic operators…
3. Call your objects useful names. Don't call your model model, or your dataframe data.
4. You can terminate your lines with semi-colons ; , but don't.
5. Write comments!

    In R, # is the comment symbol. All what comes after # is ignored.

Example of bad coding:

```
>constant=3.28;x=constant*x;
```

Example of good coding:

```
#Calculate the height in meter
> feet_per_meter <- 3.28 #one meter corresponds to 3.28 feet
> heights_m <- heights_ft / feet_per_meter
```

History

- Up and down arrows: to correct previous commands
- **`history()`**: to create scripts

```
> history(50)
> savehistory(file="History.txt") # History saved as a text document
> loadhistory(file="History.txt") # Text document loaded into History
```

Getting help

- Help on a function

```
> help(mean) # Works! mean() is an R command
> help(lm) # Works! lm() is an R command
> ?mean  #the same as help(mean)
```

Use the example code at the bottom of help pages!

- Help on a specific type of data treatment

```
>help(regression) # Fails! regression() is not an R
  command
>help.search("regression")  #not so interesting…
> ??regression# the same as  help.search("regression")
>
> RSiteSearch("regression") #interesting!!
```

Getting help

- Manuals on the R site (R language and statistics):
  http://cran.r-project.org/
  >documentation>Contributed

- Asking the R community:
  - using the email list R-help (mainly programming questions)
    http://www.r-project.org/posting-guide.html
  - Bioconductor (bio-informatics / biostatistics)
    https://support.bioconductor.org/
    https://support.bioconductor.org/t/tutorials/

- **Most often, Google will help you best!**

III. Workspace and Files

Workspace

- All the `objects` that we make, are stored within a `workspace`.
- We can save, load, share, or archive these workspaces.
- We can list the contents of our workspace using **`ls()`**.
- We can clear items from our workspace using **`rm()`**.
- The files to which the **`save.image()`** command writes the objects in our workspace are readable by R. They are considerably compressed when compared with, say, comma delimited (.csv) files

```
> ls()
[1] "mypack"
> rm(mypack)
> ls()
character(0)
```

Working Directory

- `working directory` = the location to and from which R writes and reads by default.

```
> getwd() #what is the working directory?
[1] "C:/Users/rebdau/Documents"
> setwd("..") #go one directory upstream
> getwd() #what is the working directory now?
[1] "C:/Users/rebdau"
#you can also go to a downstream directory:
> setwd("./Documents") # ./ represents here the present working directory.
> getwd()
[1] "C:/Users/rebdau/Documents"
```

Working Directory

- Directory paths have backslashes (\) in Windows, but forward slashes (/) in everything else.
- Make robust code that works on all operating systems: use the `file.path()` function:

```
> mainDir <- "C:/Users"
> subDir <- "rebdau"
> subsubDir <- "Documents"
> mypath <- file.path(mainDir, subDir, subsubDir)
> mypath
[1] "C:/Users/rebdau/Documents"
> setwd(mypath)
> getwd()
[1] "C:/Users/rebdau/Documents"
```

[example]

```
> setwd("C:/Users/rebdau/Documents") # Set this to be the working directory
> setwd(mypath)
```

- `setwd()` is a function
- `"C:/Users/rebdau"` is an argument that is a string, unknown to R
- `mypath` is an argument that is an object, available in the workspace (-> no quotes)

- `# Set this to be the working directory` is a comment: all what comes after # is not taken into account.

Working Directory

- By default, R reads and writes into the `working directory`
- To read or write in another location, you need to mention it explicitely.

swirl lesson:

Workspace_Directories_and_Files

```
> swirl()

| Welcome to swirl! Please sign in. If you've been here before, use
the same
| name as you did then. If you are new, call yourself something
unique.

What shall I call you? firstname lastname

| Please choose a course, or type 0 to exit swirl.

1: fes720 Basic
2: STS ST07
3: Take me to the swirl course repository!

Selection: 2

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks
2: README.md
3: Workspace Directories and Files

Selection: 3
```

After-swirl

1. Something about **function arguments**
2. Saving the workspace or not?
3. Good practice

Function arguments

Can be matched positionally or by name

Positional matching:

- R assigns the first value to the first argument, the second value to second argument, etc.

Matching by name:

- Order doesn't matter

You can mix positional matching with matching by name.

Example:
sd()
[standard deviation]

```
> mydata <- c(1:20,NA)
> mydata
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 NA
>
> args(sd)
function (x, na.rm = FALSE) #two arguments
> ## Positional match first argument, default for 'na.rm'
> sd(mydata)
[1] NA
> ## Positional matching for both arguments
> sd(mydata,TRUE)
[1] 5.91608
> ## Specify both arguments by name (order mixed up)
> sd(na.rm = TRUE, x = mydata)
[1] 5.91608
> ## Mix of positional and matching by name.
> ## don't need to give the whole name of the arguments!!!
> sd(n = TRUE, mydata)
[1] 5.91608
```

Stopping

- Use Ctrl-C or Esc to stop processing.
- Quit R by typing the command q().
- When you do so, R will ask you if you want to save your `workspace`.
- If you choose to save the `workspace` then a compressed image of the `objects`, called `.RData`, will be saved into your working directory.
- To access these `objects` again in a future session, use the `load()` function.

```
>load(".RData")
```

good practice when programming or doing analysis in R:

- keep the commands in a script
  - can be saved and reused

- Use an appropriate editor
  - more efficient
  - less error prone

Important editor features:

- syntax highlighting: functions, strings, comments,… are written in distinct colors
- auto-completion for R code
- code folding: lines of code can be automatically grouped and temporarily hidden from view.
- directly evaluate lines of code in R

Notepad++ with NppToR

- set language to “R” → syntax highlighting
- NppToR Hotkeys:
  - F8 Evaluate a line of code or selection.
  - Ctrl+F8 Evaluate the entire current file.
  - Shift+F8 Evaluate the file to the point of the cursor
- nice introduction to NppToR in the R Journal for June 2010 (page 62), written by the developer, Andrew Redd: NppToR_AndrewRedd.pdf

IV. Importing and exporting data

Importing data

Import

- **list.files()**
- **read.csv ()**

```
#absolute path
> ufc <- read.csv(file="C://path/to/workshop/data/ufc.csv")
#relative path
> ufc <- read.csv(file="../data/ufc.csv")
```

Check data

- **dim()**
- **names()**
- **str()**
- **head()**

Exporting data

```
#absolute path
> write.csv(ufc,
file="C://path/to/filenameworkshop/output/file.csv")
#relative path
> write.csv(ufc, file="../output/file.csv")

#saving graphics in a pdf file
> pdf("../graphics/fileName.pdf") # Opens a pdf device
> plot(1:10,1:10) # ... or can do something more sophisticated
> dev.off() # Closes the pdf device and saves the file
```

Being organized

Set up the directory structure on your hard drive that you will use for the exercises in this workshop:

- Create a single directory, called, for example, **rprogramming**
- Within this directory, create the following subdirectories:
  - **data**
  - **graphics**
  - **notes**
  - **scripts**

  (We won't necessarily use all of them.)

R permits relative **directory labelling**

➔**from the script directory**

- the data will be in `../data`
- the graphics will be in `../data`
- the images will be in `../data`

Exercise 1

- Using the menu bar, choose **rprogramming** as your working directory

- Using the command line, set the working directory to **rprogramming/scripts**
- Which files are in rprogramming/data?
- Load ufc.csv from an absolute address, and name it ufc
- Load Lflavum.csv from a relative address and name it Lflavum
- How many variables do these datasets have?
- What are the dimensions of these datasets?
- Save these same datasets in the data subdirectory, under different names (ufc_ex1.csv and Lflavum_ex1.csv), and check in excel! (correct the arguments of write.csv() if necessary.)
- Create, in the graphics subdirectory a pdf named “1to10.pdf”, in which you execute the following command:

```
plot(1:10,1:10)
```

- Save your commands in a commented script in the scripts directory

V. Classes of data

Atomic objects:
can't be broken down any further

- `Numeric`
- `Integer`
- `Character`
- `Factor`
- `Logical`

swirl lesson:

Data_classes

Update the STS_ST07 swirl lessons

```
> library(swirl)

| Hi! I see that you have some variables saved in your workspace. To keep things
running smoothly, I recommend you clean up before starting
| swirl.

| Type ls() to see a list of the variables in your workspace. Then, type
rm(list=ls()) to clear your workspace.

| Type swirl() when you are ready to begin.

> uninstall_course("STS_ST07")
Course uninstalled successfully!
> install_course_github("rebdau","STS_ST07")

  |
  |
|   0%
  |

|=======================================================================
==================================================| 100%

Downloading: 3.5 kB     >
```

Load Data_classes

```
 > swirl()

| Welcome to swirl! ...

What shall I call you? firstname lastname

| Would you like to continue with one of these lessons?

1: R Programming Basic Building Blocks
2: STS ST07 Workspace Directories and Files
3: No. Let me start something new.

Selection: 3

| Please choose a course, or type 0 to exit swirl.

1: STS ST07
2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks
2: Data classes
3: Logic
4: README.md
5: Sequences of Numbers
6: Workspace Directories and Files

Selection: 2
```

Useful commands:

```
class(object)      # what class is it?
is.className()     # check the class
as.className()     # change the class
str()   # gives you the class of the variables in a data.frame
```

Numeric

`is.numeric()`

`as.numeric()`

- Mathematical operations:

  +

  -

  *

  /

  ^

  ==

  …

Character

- = string
- collection of one or more alphanumerics, denoted by double quotes.

```
is.character()
as.character()
paste()
substr()
sub()
```

factor

- Categorical variable
- can take only a limited number of values = `levels`

```
as.factor()
factor() #short version of as.factor()

is.factor()
table()
```

logical

- =Boolean
- A special kind of **factor**, with **two levels**: **True and False**.
- True and False levels are interchangeable with the numbers 1 and 0 (respectively).
- The output of several useful functions are logical.

```
> is.character("john")
[1] TRUE
```

- Can construct logical statements using the operators

  and (&)

  or (|)

  not (!)

- **`which()`** #gives indices of TRUE in a vector

Missing data (NA)

- Not a unique class, can be mixed in with all other kinds of data.
- Treatment not uniform in all functions: sometimes you have to tell the function to ignore them, and sometimes you don't.

```
is.na()
complete.cases()
```

swirl lesson:

Sequences of Numbers

Load Sequences of Numbers

```
 > swirl()

| Welcome to swirl!

What shall I call you? firstname lastname

| Would you like to continue with one of these lessons?

1: R Programming Basic Building Blocks
2: STS ST07 Workspace Directories and Files
3: No. Let me start something new.

Selection: 3

| Please choose a course, or type 0 to exit swirl.

1: STS ST07
2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks
2: Data classes
3: Logic
4: README.md
5: Sequences of Numbers
6: Workspace Directories and Files

Selection: 5
```

swirl lesson:

Logic

Load Logic

```
 > swirl()

| Welcome to swirl! ...

What shall I call you? firstname lastname

| Would you like to continue with one of these lessons?

1: R Programming Basic Building Blocks
2: STS ST07 Workspace Directories and Files
3: No. Let me start something new.

Selection: 3

| Please choose a course, or type 0 to exit swirl.

1: STS ST07
2: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.

1: Basic Building Blocks
2: Data classes
3: Logic
4: README.md
5: Sequences of Numbers
6: Workspace Directories and Files

Selection: 3
```

Containers of data

- `vector`
- `dataframe`
- `matrix`
- `list`

➔Different mechanisms that we have for the collective storage and manipulation of data

Vector

- one-dimensional collection of atomic objects
- Vectors can contain numbers, characters, factors, or logicals.
- But all the objects in a vector must be of the **same class**
- All the objects that we created earlier were vectors, although some were of length 1.
- Vectors are manipulated using **subscripts [ ]**.
  - A list of positional numbers (**indices**)
    - Can be (much) longer than the `vector`!
  - A list of FALSE, TRUE, with the same length as the `vector`

```
> a <- c("a", "b", "c")
> a[c(1,1,1,1,1,1,1,1,1,1)]
 [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
> a[rep(1,10)]
 [1] "a" "a" "a" "a" "a" "a" "a" "a" "a" "a"
> a[c(TRUE,FALSE,TRUE)]
[1] "a" "c"
```

Vector

- **order()**

```
> a <- c(50,30,80)
> order(a)
[1] 2 1 3
> o <- order(a)
> a[o]
[1] 30 50 80
```

- Arithmetic operations: member by member!
- recycling

```
> a <- 1:10
> a
 [1]  1  2  3  4  5  6  7  8  9 10
> a * 2 #member by member
 [1]  2  4  6  8 10 12 14 16 18 20
> a * c(1,2) #recycling
 [1]  1  4  3  8  5 12  7 16  9 20
```

Dataframe

- = `list` of `vectors` with the **same length**
- **2 dimensions**
  - `rows` : observations
  - `columns` : variables
- `columns` : variables of different classes (`characer`, `numeric`, `logical`, `factor`)
- `subscripts [,]`
  - blank: the whole dimensin is assumed
  - Negative numbers: `rows` or `columns` that will be omitted
- Each `column`, or variable, has a unique name
  - `dataframe$variable`
  - `dataframe[,c("variable1", "variable2", "variable3")]`

Dataframe

```
read.csv()  #→ table imported as a dataframe
is.data.frame()    # Check if it's a dataframe
as.data.frame()    #change to a dataframe
#SUPER HANDY FUNCTION to stick columns together:
data.frame()       #create a dataframe...
sapply()    #apply a function on each column of a
dataframe
```

Matrix

- = `vector` with more dimensions
- Thus: all objects in a matrix must be of the same class
- `subscripts [,]`
  - blank: the whole dimensin is assumed
  - Negative numbers: rows or columns that will be omitted

`cbind()`

`rbind()`

List

- Contains objects that can be of different lengths and of different classes.
  - A `dataframe` is a list of `vectors`
  - The output of a function is often a `list`
- Access to the elements of a list (works thus also for dataframe):

  `[[]]`

  `$`

  `[]` →the element will be a list
- **`lapply()`** #apply a function on each element of a `list`

- Every object followed by () is a function, being called
- Every object followed by [] is subsetted

Data fusion

- Fuse datasets with different dimensions

```
%in%
match()
merge()
```

Select data based on a categorical variable

- `tapply()`

  Execute a function on a single numeric variable, for each category determined by a categorical variable (`factor`)

Select data based on a categorical variable

- **`aggregate()`**

  Execute a function on **more than one numeric variable**, for each category determined by a categorical variable (`factor`)

aggregate()

```
aggregate(
x=list(diameter=ufc$dbh, height=ufc$height),
by=list(species=ufc$species),
FUN=mean, na.rm=TRUE)
```

Result:

A `dataframe`:

First column:

- Column name = categorical variable (`species`)
- Contains the labels of the levels of the categorical variable

Other columns:

- Column names = names of the data in the function (`diameter`, `height`)
- Contain the output of the function for each level