FIRST CONTACT WITH R

DIGITAL SOCIETY SCHOOL

DATA-DRIVEN TRANSFORMATION

EMMA BEAUXIS-AUSSALET

e.m.a.l.beauxis@hva.nl

CONTE NT

BASICS OF PROGRAMMING VARIABLE

A **variable** is like a **box** in which we ca
store data

4

```
my_variable <- "my data"
```

VARIABLE

A **variable** is like a **box** in which we ca
store data

name the variable as you please

but it must start with a letter

```
my_variable <- "my data"
```

DATA TYPES

Variables can store different kinds of data

Text

```
my_text_variable <- "Hello!"
```

Number

Boolean

```
my_number_variable
<- 2019
```

Table

```
c("B","beta"))
```

```
my_table[ ,3] <-
c("ALPHA","BRAVO")
my_table[2,] <-
c("B","beta","bravo")
my_table[2,3] <-
"BRAVO"
success <- TRUE
failure <- FALSE
failure <- FALSE
```

6

```
my_table <- rbind(c("A","alpha"),


my_table <- cbind(my_table,
c("alpha","bravo"))
```

IF - THEN - ELSE

Computations can be **executed** only under certain **conditions**

```
if(age < 18) {

print("Warning!")

} else {

print("Welcome")

}
```

7

IF - THEN - ELSE

Computations can be **executed** only under certain **conditions**

```
if(age < 18) {
print("Warning!")
} else {
print("Welcome")
}
```

test the condition done is passed i

done is failed if test

8

IF - THEN - ELSE

```
success <- TRUE
if(success) {
print("This is executed")

} if(!success){
```

```
print("Not happening!") }
```

The **condition** is

boolean

```
if(grade == 0){
print("Terrible Fail!")
} else if(grade < 5){
print("Fail")
} else if(grade < 8){
print("Pass")
} else {
print("Awesome!") }
```

Conditions can be **tes**

one after the other

9

IF - THEN - ELSE

Conditions can be **tested together**

```
if(grade > 4 & grade < 5) {
```

All conditions

```
print("Almost there!") }

if(grade < 2 | grade > 8) {
```

At least one condit

```
print("That's extreme!") }

   if( (dish == "Soup" & wine == "None")| (dish == "Fish"
                    != "Red") |(dish == "Duck" & wine == "F
```

Conditions can be

```
print("I want to eat this.") }
```

complex and **nested**

```
if( (dish == "Fish" & wine != "Red") &
(wine == "Rosé" &
(hour > 23 | temperature > 25) ) ) {
print("This is an exception.")
}
```


FUNCTION

A **function** is a **series of operations** t
can be **repeated**

```
my_function <- function(grade){

if(grade < 5) {
print("Fail") } else {
print("Pass") } }
```

```
my_function(0)
```

```
11
my_function(8)
```

FUNCTION

A **function** is a **series of operations** t
can be **repeated**

```
my_function <- function(grade){
if(grade < 5) {
print("Fail") } else {
```

```
print("Pass") } }
my_function(0)
12
```

Results depend on the input variable

```
my_function(8)
```

FUNCTION

```
give_grade <- function(Q1, Q2){
grade <- 0 if(Q1 == "France") {
grade <- grade + 1 } if(Q2 == “Louis XIV") { grade <- gra
} print(grade) }
```

Functions can take seve**variables** as **input**

Functions can **return** a single **result**

```
test_Q1 <- function(Q1){ if(Q1 == "France") {
} } return(TRUE)
```

Input variables can have **default values**

```
grade_Q1 <- function(Q1, grade = 0){
```

LOOP

Operations can be **repeated** a certain **number of times**

```
for( i in 1:10 ){
print(i)
}
```

```
for( my_letter in my_table$letter ){
print(my_letter)
```

```
}
```

14

LOOP

i Loops can **repeat**

operations until certain conditions change

While Loop

```
<- 5
while( i > 0 ){
print( my_table$letter[i] )
i <- i - 1
}
```

Loops can be interrupted

```
for( my_letter in my_table$letter ){
print(my_letter)
```

```
if( my_letter == "B") {
break

} } for( i in 1:10 ){
if( i %in% c(2,4,6,8,10) ) {
                                                    next } p
```

15

LOOP

i Loops can **repeat**

operations until certain **conditions change**

While Loop

```
<- 5
while( i > 0 ){
print( my_table$letter[i] )
i <- i - 1
}
```

Loops can be **interrupted**

```
for( i in 1:10 ){
if( i %in% c(2,4,6,8,10) ) {
                                                    next } p
```

```
for( my_letter in my_table$letter ){
print(my_letter)
if( my_letter == "B") {
break } }
```

Terminate the loop

16

Skip next item

to the

EXERCISE

1. Separate into 2 groups: with and witl
programming experience

2. Make teams (2-3 max) that mix learr
from both groups

3. Download the exercises: link

4. Pitch what was tricky or helpful

18

PROGRAMMING WITH R

R STUDIO

R is a **programming language**

R Studio is an **environment** for programming, debugging, and **executi**

First install R https://cran.r-project.org/

Then install R Studio

https://www.rstudio.com/products/rstudio/download/

R O

STUDI

R Studio has **3 main panels**

Find help

20

Using the **console** on the **right side** is recommended

Write code

Execute code

Select to execute the lines ... ...pres

Command of code + Enter

...See the results here !

21

LIBRARIES

A **library** is a collection of useful **pre-m**
functions

```
install.packages("tidyverse")
```

```
library(tidyverse)
```

22

LIBRARIES

A **library** is a collection of useful **pre-m**

functions

Download a library

```
install.packages("tidyverse")
```

```
library(tidyverse)
```

Load a library

23

...needs no internet ...to do each time

you open R Studio

...needs internet ...to do only once for the whole comp

DATA TYPES

Collections store several elements.

Operations can be applied to all elements.

Series of Numbers

```
my_numbers <- c(3, 5, 7, 11, 13, 17)

my_numbers[7] <- 19

my_numbers <- 1:10
```

Series of Booleans

```
my_numbers <- my_numbers / 2
```


Series of Texts

```
my_texts <- c("Hi", "Salut", "Ciao")

my_texts[4] <- "Hallo"

my_texts <- paste(my_texts, "Mary")
```

```
my_booleans <- my_texts == "Hi Mary"
```

DATA TYPES

Collections can be searched and order

Check the presence of an

```
my_boolean
```

element

```
<- "Hi Mary" %in% my_texts
```

Extract unique elements

```
if(! "Hi Mary" %in% my_texts ){
print("'Hi Mary' is missing.")
}25
```

Order elements

```
sort(my_texts)
sort(1:10, decreasing = TRUE)
my_data <- c("Red","Red","Blue",
"Red","Black","Black")
unique(my_data)
```

DATA TYPES

Data types are easily modified **Series**

Texts

Number to Text

```
my_texts <-
as.character(1:10)

my_texts <-
paste("Item", 1:10)
```

Text to Number Series of Booleans

```
my_number <-
as.numeric("1")

26
my_texts <- c("Hi",
"Salut", "Ciao")

my_texts <-
paste(my_texts,
"Mary")
```

```
my_booleans <-                Mary"
my_texts == "Hi
my_texts[4] <- "Hallo"
```

DATA TYPES

Dates and timestamps are special data types.

Date Date & Time

```
timestamp <- as.POSIXct(
"2019-03-07 12:00:00")
now <- Sys.time()
next_minute <- Sys.time() + 60
```

https://www.r-bloggers.com/date-formats-in-r/
http://www.noamross.net/blog/2014/2/10/using-times-and-dates-in-r---presentat
code.html

```
date <- as.Date("2019-03-07")
today <- Sys.Date()
tomorrow <- today + 1
format(today, "Year is %Y")
as.numeric(today - tomorrow)
```

DATA TYPES

Variables can store complex data structures

List

```
my_list <-
list(my_table,
my_texts,
my_numbers, "etc")
```

```
my_list[[4]] <-
"anything"
```

```
my_list[[1]][,3] <-
c("Alpha","Bravo")
```


Tibble

```
my_tibble <-
tibble(1:2, c("a","b"),
list(my_texts,
my_numbers),
list(my_table,
my_list))
```

```
my_tibble[ ,2] <-
c("A","B")
```

```
my_tibble[1,3][[1]][[1]]
[5] <- "Hola"
```

Data Frame

```
"greek","code")

my_table$code <-
c("alpha","bravo")
```

```
my_data_frame <-
data.frame(my_table)
colnames(my_table) <- c("letter",
```

PIPE OPERATOR

The pipe (**%>%**) makes our code **eas**
to read

```
second_function( first_function(data) )
```

```
data %>% first_function() %>%

second_function()
```

29

...Same as...

PIPE OPERATOR

The pipe (**%>%**) makes our code **eas**
to read

```
second_function( first_function(data) )
```

```
data %>% first_function() %>%

second_function()
```

This is executed first...

...But we read it last !

PIPE TOR OPERA

Functions take the **piped variable** as their **first input** variable

```
my_function(my_data
, my_variable)

my_data %>%
my_function(my_vari
able)
```

```
grade %>%
round(digits = 1)
```

```
grade %>%
round(digits =
my_data)
```

The pipe can be used **anywhere**

```
if( grade %>%
my_function() > 5 ) {

print("Pass")

}
```

```
my_function(



)
```

```
grade %>% round() %>%


my_variable = time %>% round()
```

31

Start TIPS your scripts with nice settings Cheatsheet

https://www.rstudio.com/resources/cheatsheets/
https://github.com/DigitalSocietySchool/R_FirstContact/tree/master/Cheatsheet

Tutorial

https://r4ds.had.co.nz/
https://www.datacamp.com/tracks/tidyverse-fundamentals
https://www.tidyverse.org/learn/

32

```
# Set the working environment if(!require('rstudioapi') ){
```

```
install.packages('rstudioapi') } library(rstudioapi) setwd
dirname(
getSourceEditorContext()$path )) # Disable scientific
notation options(scipen = 999)
# Disable text encoding as 'factors' options(stringsAsFa
FALSE)
```

https://www.datacamp.com/community/tutorials/r-data-import-tutoria

- TEXT FILE .....

CSV FILE ..... F

33

READING & WRITIN DATA

Format the content of tables TEXT FILE

Write

```
write("Next Line",
"my_text.txt",
append = TRUE)
for( i in
1:nrow(my_table) ){

paste(
my_table$name[i],
"has grade",
my_table$grade[i] )
%>% write(
"class_grade.txt",
}append = TRUE )
```

Read

```
read_file("my_text.txt"
)
write("My text",
"my_text.txt")
```

Use the raw

console

output

```
my_table %>%
print() %>%
capture.output() %>%
write("my_text.txt")
```


CSV FILE

Write as plain text **Write** tables as they

Read

```
read.csv("my_data.csv")
write( "A, alpha\n
B, beta",
"my_data.csv")

write("C, gamma",
"my_data.csv",
append = TRUE)
```

Use other separator

```
read.csv( "my_data.tsv",
```

```
write.csv( my_table,
"my_data.csv",
row.names = FALSE)
```

```
sep = "\t" )
```

EXCEL FILE

Main Libraries

https://www.datacamp.com/community/tutorials/r-tutorial-

Excel files have **different formats** read-e

into-r

depending on the version of Excel

library(readxl)

library(gdata)

The **R libraries** for reading Excel files

not work with all formats library(xlsRea

library(XLConnect)

The right library can be found library(xlsx)

with **trials and errors**

36

XML DATA

Main Libraries

https://www.datacamp.com/community/tutorials/r-data-import-tutorial#xml

XML data has a **standard format**

https://github.com/r-lib/xml2/blob/master/README.md

that also applies to HTML data

```
library(xml2)
read_xml("<client>
```

The **R libraries** for reading XML files

```
<name> Emma </name>
```

may work slightly differently

```
</client>")
library(XML)
```

The first step to begin with `xmlTreeParse('<`

```
name="Emma">')
```

is to **follow a tutorial** (https://www.w3schools.com/xml/)

37

HTML

DATA

Access

Webpages

Browse Links

```
library(RCurl)

getURL("https://digitalsocietyschool.org")
```

https://www.datacamp.com/community/tutorials/r-web- scraping-rvest

```
"https://digitalsocietyschool.org" %>%
read_html() %>%
html_nodes( "a" ) %>% html_attr( "href" )
```

Read HTML Content Read HTML Tab

```
library(rvest)

"https://digitalsocietyschool.org" %>%
read_html()
read_html(
"http://w3schools.com/html/html_tables.asp"
) %>% html_table()
```

https://dplyr.tidyverse.org/

- OVERVIEW DATA .....
- ORDER DATA .....
- EXTRACT DATA .....
- ADD DATA .....
- MERGE DATA .....

39

MANIPULATING DATA

```
library(tidyverse)
```

OVERVIEW DATA

Summary

https://dplyr.tidyverse.org/reference/summarise.html

```
# Get mean & sum of each column
```

Simple functio

for

```
data %>% colMeans()
data %>% colSums()
```

quickly checking the da

```
# Get summary statistics of each column dat
summary() Over
# Get specific statistics of any column data %>% sum
min(my_column), # Get row names & first
max(my_column) ) data %>% glimpse()
# Get statistics for groups of row # Get first & last rows
data %>% group_by( other_column ) %>% data %>% h
summarise( min(my_column), data %>% tail()
max(my_column) )
```


ORDER DATA

Order by column value

```
# From lowest to
highest column value
```

```
data %>% arrange(
my_column )

# From highest to
lowest column value
data %>% arrange(
desc(my_column) )

# Order by one
column then another
data %>% arrange(
my_column,
other_column )
```

41

Group by column value

```
# Group rows with the
same column value
data %>% group_by(
my_column )

data %>% group_by(
my_column,
other_column )

# Best value per
group data %>%
group_by(
my_column ) %>%
top_n( 1,
another_column )
```

EXTRACT

DATA

Extract Row

https://dplyr.tidyverse.org/reference/select.html

https://dplyr.tidyverse.org/reference/filter.html

```
# Get rows with certain column values
data %>% filter( my_column > 5 )

data %>% filter( my_column > 5 & my_column < 8 )
```

Extract Column

```
data %>% filter( my_column > 5, other_column == 0 )

# Get certain columns
data %>% select( my_column, other_column )
# Get rows with the 10 highest values
data %>% select( starts_with("my_") )
data %>% top_n( 10, my_column )
# Random sample 10 rows data %>% sample_n( 10 )
```

```
data %>% select(
contains("other") )
# Get rows with the
10 lowest values data
%>% top_n( -10,
my_column )
```

42

ADD DATA

https://dplyr.tidyverse.org/reference/mutate.html

Add Column

```
# Add a new column
data %>%
mutate( new_column
= my_numbers )

# Make a column
from another column
data %>%
mutate( bonus =
my_column + 2 )
```

43

Add Row

```
data %>%
```

bind_rows(
my_new_rows)

MERG E DATA

Add All Columns

Equivalent

left_join(

my_data, other_data,

by=c('id_column'='other_id_column')

)

right_join(

other_data, my_data,
by =
c('other_column'='my_column')

)

44

MERGE DATA

Match on several columns **Keep Only Matching Rows**

```
inner_join(

my_data, other_data,
by =
c('my_column'='other
_column')
)
```

45

```
inner_join(

my_data, other_data,
```

```
by =
c('my_column'='other
_column',
                    'my_extra'='other_ext
                    ra',

                    )
```

- VISUAL DIMENSIONS .....
- MULTIPLE DIMENSIONS .....
- NETWORK & FLOW .....

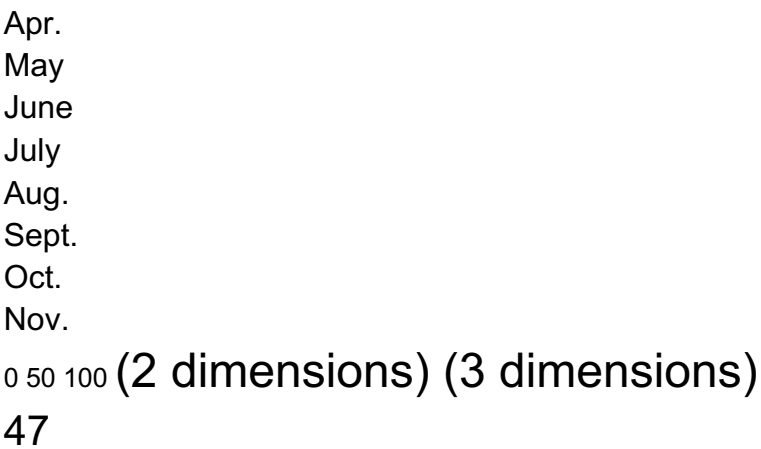
BASICS OF VISUALIZATION

VISUAL DIMENSIO

Visual features can be used to represe
different variables

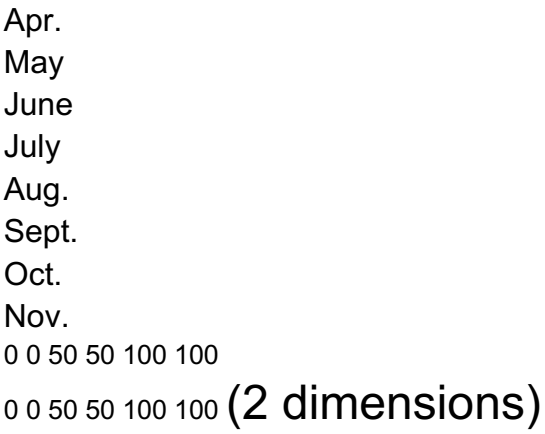
Length

Variables are the data dimensions enc into visual dimensions



Color

Position



Apr.
May
June
July
Aug.
Sept.
Oct.

Nov.(3 dimensions)

Size

Transparency

Apr.
May
June
July
Aug.
Sept.
Oct.

Nov.0 0 50 50 100 100 (3 dimensions)

VISUAL DIMENSIO

Visual features offer many options to represent the same data...

Area

100

...but some visual features are hard

compare with human eyes

100
50

50

50 25Pie Chart Donut Chart 0002017 201820102014

2018

2010

2014

2018

Length

Area

48

Position

MULTIPLE DIMENSIONS

To add a data dimension, add a visual dimension

1 Dimension

...but the graph becomes complex

2 Dimensions

3 Dimensions

4 Dimensions

July
Aug.
Sept.
Oct.
Nov.
0 50 100
Apr.
May
June
July
Aug.
Sept.
Oct.
Nov.
0 50 100
49

5 Dimensions

MULTIPLE DIMENSIONS Multiple Views

2014 2015 2016 2017 **2018**
Apr.
Apr.
May
May
June
June
July
July

Aug.
Aug.
Sept.
Sept.
Oct.
Oct.
Nov.
Nov.
0 50 100
0 50 100

50

Heatmap

Scatterplot Matrix

MULTIPLE DIMENSIONS

Radar Cha

51

Cyclic Graph

Radial Graph

Glyph

NETWORK & FLOW

Sankey Diagram Chord Diagram

52

NETWORK & FLOW

Force-
Directed
Graph
Tree

53

Radial Layout
Radial Layout

VARIANCE

Boxplot
Ridge Lines

54

Violin Plot
Violin Plot

VISUALIZATION WITH R

https://ggplot2.tidyverse.org/
https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf

```
library(ggplot2)
```

SAVING GRAPHS

```
data %>%
ggplot( aes( x = my_column, y = other_column ) ) +
geom_point( aes( text = paste("Tip Box:", another_colu
+
geom_smooth( aes( colour = last_column, fill = last_col
+
facet_wrap( ~ last_column )

ggsave( "my_visualization.pdf", width=18, height=18)
```

```
ggsave( "my_image.png", width=10, height=10, unit="cr
```