

# 00-IntRoduction

Ottavia M. Epifania, Ph.D

Lezione di Dottorato @Università Cattolica del Sacro Cuore (MI)

13 Giugno 2024

# Table of contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming

# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming

- R is an open source software for statistical computing, graphics, and so much more
- RStudio is the perfect IDE for R → allows for a better, easier use of R
- R runs on Windows, MacOS, Unix

# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming

```
setwd("C:/Users/huawei/OneDrive/Documenti/GitHub/RcouRse")
```



## R projects for the win

Dealing with working directories is a pain in the neck you should try to avoid at all costs

Besides, after months under review, the manuscript has finally come back and now you have to revise it. But where are the data...? and the scripts with the analyses...?

R projects for the win!

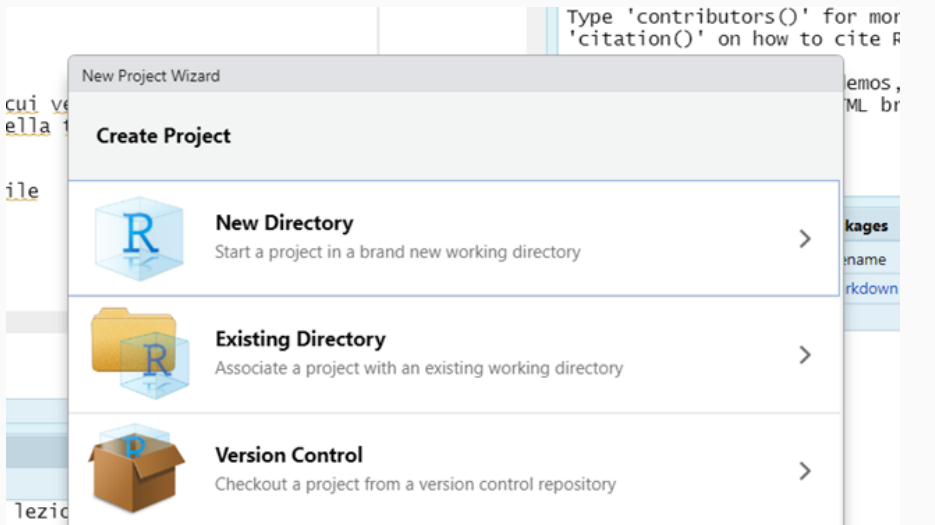
An R project automatically create an “Island” with its sub islands

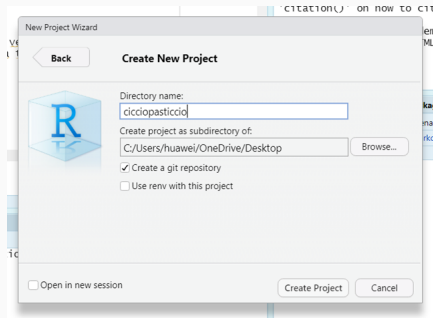
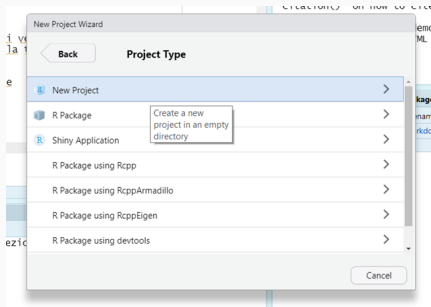
They are pretty convenient for at least two reasons:

- ① Allow for havin multiple R instances open at the same time → You can work on multiple projects at the same time
- ② All the files you need are there



File → New project:





# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics**
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming

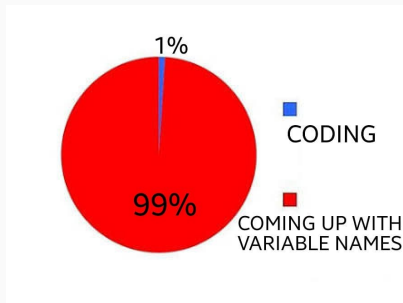


To assign a value to an object, there are two operators:

- ```
② X <- log(2^2)
```

**Careful!** R is case sensitive: x and X are two different objects!!!

# Variable names



Valid variable names are letters, numbers, dots, underscores (e.g., `variable_name`)

Variable names cannot start with numbers

Again, R is case sensitive

# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help**
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming

R community is the best feature of R

Just copy & paste any error message or warning in Google or ask Google  
 “how to **[something]** in r”

Ask R to help you! Type ? in your console followed by the name of the  
 function:

?mean()

Will show you the help page of the mean() function



# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy**
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming

# Organize your files

---

R projects are the best way to organize your files (and your workflow)

They allow you to have all your files in a folder organized in sub folders

You don't have to worry about the wording directories because it's all there!

By creating a new project, you can also initialize a Shiny app

- R project “basic”
- R package
- Shiny project

OME, UniPD (Lezioni UniCatt)

If it feels like you're losing it, just clean it up:

```
rm(list=ls()) # remove everything from the environment
```

```
save.image("my-computations.RData")
```

```
load("my-Computations.RData")
```

# When to save the environment

---

The computations are slow and you need them to be always and easily accessible

The best practice is to save the script and document it in an RMarkdown file → Reproducibility!

# Your turn!



- Create an R project for this course in your “documents” folder (choose a nice name :)
- Create a new R script (shift + ctrl + n)
- Calculator Using the script:
  - $\sqrt{(15)} * 14 - \frac{22}{4}$  [48.72177]
  - $\frac{\sqrt{7-\pi}}{3(45-34)}$  [0.05952372]
- Save the script
- Assign the results of the first equation to a variable named `my_results`

# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R**
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming



## Functions and arguments (pt. I)

Almost everything in R is done with functions, consisting of:

- a name: `mean`
- a pair of brackets: `()`
- some arguments: `na.rm = TRUE`

```
mean(1:5, trim = 0, na.rm = TRUE)
```

[1] 3

Arguments may be set to default values; what they are is documented in `?mean()`

## Functions and arguments (pt. II)

## Arguments can be passed

- without name (in the defined order)
- with name (in arbitrary order) → *keyword matching*

```
mean(x, trim = 0.3, na.rm = TRUE)
```

No arguments? No problems, just brackets:

```
ls(), dir(), getwd()
```

Want to see the code of a function? Just type its name in the console without brackets:

chisq.test

---

c()

Concatenates several objects together to combine them into a unique object →

```
x = c(1, 2, 3) # create a vector as a concatenation of "1", "2", "3"
x
```

[1] 1 2 3

```
X = 1:3 # create the same identical vector
X
```

[1] 1 2 3

$$\mathbf{x} = \mathbf{X}$$

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

# Vectors

Vectors are created by combining together different objects

Vectors are created by using the `c()` function.

All elements inside the `c()` function **must** be separated by a comma

Different types of objects  $\rightarrow$  types of vectors:

- `int`: numeric integers
- `num`: numbers
- `logi`: logical
- `chr`: characters
- `factor`: factor with different levels

## int and num

---

int: refers to integer: -3, -2, -1, 0, 1, 2, 3

```
months = c(5, 6, 8, 10, 12, 16)
```

```
[1] 5 6 8 10 12 16
```

num: refers to all numbers from  $-\infty$  to  $\infty$ : 1.0840991, 0.8431089, 0.494389, -0.7730161, 2.9038161, 0.9088839

```
weight = seq(3, 11, by = 1.5)
```

```
[1] 3.0 4.5 6.0 7.5 9.0 10.5
```





# Create vectors

---

Concatenate elements with `c()`: `vec = c(1, 2, 3, 4, 5)`

Sequences:

```
-5:5 # vector of 11 numbers from -5 to 5
```

```
[1] -5 -4 -3 -2 -1 0 1 2 3 4 5
```

```
seq(-2.5, 2.5, by = 0.5) # sequence in steps of 0.5
```

```
[1] -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5
```

Repeating elements:

```
rep(1:3, 4)
```

```
[1] 1 2 3 1 2 3 1 2 3 1 2 3
```



## Create vectors II

```
rep(c("condA", "condB"), each = 3)
```

```
[1] "condA" "condA" "condA" "condB" "condB" "condB"
```

```
rep(c("on", "off"), c(3, 2))
```

```
[1] "on" "on" "on" "off" "off"
```

```
paste0("item", 1:4)
```

```
[1] "item1" "item2" "item3" "item4"
```

# Don't mix them up unless you truly want to

---

`int + num → num`

`int/num + logi → int/num`

`int/num + factor → int/num`

`int/num + chr → chr`

`chr + logi → chr`



$\sqrt{a}$ 

The same operation can be applied to each element of the vector:

```
(a - mean(a))^2 # squared deviation
```

```
[1] 12.25  6.25  2.25  0.25  0.25  2.25  6.25 12.25
```

# Your turn!



- Create a new script & save it in your R project!
- Assign the following values to a variable named `my_var` (you have to *concatenate* the values)  
`23, 24, 25, 27, 28, 29, 30`
- Compute the mean using the function `mean()`
- Compute the mean of vector using the functions `sum()` and `length()`
- Find the minimum (`min()`) and maximum (`max()`) value of the vector

# Index the elements in a vector

---

```
names = c("Pasquale", "Egidio", "Giulia", "Livio", "Andrea")
```

|          |        |        |       |        |
|----------|--------|--------|-------|--------|
| Pasquale | Egidio | Giulia | Livio | Andrea |
|----------|--------|--------|-------|--------|

1

2

3

4

5

# Index the elements in a vector

---

```
names = c("Pasquale", "Egidio", "Giulia", "Livio", "Andrea")
```

|          |        |        |       |        |
|----------|--------|--------|-------|--------|
| Pasquale | Egidio | Giulia | Livio | Andrea |
| 1        | 2      | 3      | 4     | 5      |

```
vector_name[index]
```





# Index the elements in a vector II

---

|          |        |        |       |        |
|----------|--------|--------|-------|--------|
| Pasquale | Egidio | Giulia | Livio | Andrea |
|----------|--------|--------|-------|--------|

1

2

3

4

5

`names[1] →`

# Index the elements in a vector II

---

|          |        |        |       |        |
|----------|--------|--------|-------|--------|
| Pasquale | Egidio | Giulia | Livio | Andrea |
|----------|--------|--------|-------|--------|

1

2

3

4

5

`names[1]` → Pasquale

`names[3]` →

## Index the elements in a vector II

|          |        |        |       |        |
|----------|--------|--------|-------|--------|
| Pasquale | Egidio | Giulia | Livio | Andrea |
|----------|--------|--------|-------|--------|

1

2

3

4

5

`names[1] → Pasquale`

`names[3] → Giulia`

`names[seq(2, 5, by = 2)] →`

## Index the elements in a vector II

|          |        |        |       |        |
|----------|--------|--------|-------|--------|
| Pasquale | Egidio | Giulia | Livio | Andrea |
|----------|--------|--------|-------|--------|

1

2

3

4

5

`names[1] → Pasquale`

`names[3] → Giulia`

`names[seq(2, 5, by = 2)] → Egidio, Livio`

# Index the elements in a vector: Examples

```
weight
```

```
[1] 3.0 4.5 6.0 7.5 9.0 10.5
```

```
weight[2]          # second element of weight
```

```
[1] 4.5
```

```
(weight[6] = 15.2) # replace the sixth element of weight
```

```
[1] 15.2
```

```
weight[seq(1, 6, by = 2)] # elements 1, 3, 5
```

```
[1] 3 6 9
```

```
weight[2:6]        # elements from 2 to 6 (included)
```

```
[1] 4.5 6.0 7.5 9.0 15.2
```

```
weight[-2]         # remove the second element
```

```
[1] 3.0 6.0 7.5 9.0 15.2
```

# Index with logic

---

```
weight
```

```
[1] 3.0 4.5 6.0 7.5 9.0 15.2
```

## Index with logic

```
weight
```

```
[1] 3.0 4.5 6.0 7.5 9.0 15.2
```

Which are the values  $> 7$ ?

```
weight > 7
```

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

## Index with logic

weight

```
[1] 3.0 4.5 6.0 7.5 9.0 15.2
```

Which are the values  $> 7$ ?

```
weight > 7
```

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

“Filter” the vector with logic

```
weight[weight > 7] # only weights > 7
```

```
[1] 7.5 9.0 15.2
```

```
weight[weight >= 4.5 & weight < 8] # values between 4.5 and 8
```

[1] 4.5 6.0 7.5



# Your turn!

---



- Considering `my_var`
  - Third element
  - Extract all the odd elements and assign them to a new variable `my_vector1`
  - Extract all elements  $> 25$  from `my_vector1`

# Matrices and arrays

---

```
matrix(data, nrow, ncol, byrow = TRUE)
```

# Matrices and arrays

```
matrix(data, nrow, ncol, byrow = TRUE)
```

Create a  $3 \times 4$  matrix:

```
A = matrix(1:12, nrow=3, ncol = 4, byrow = TRUE)
```

Label and transpose:

```
rownames(A) = paste("a", 1:nrow(A), sep = t(A))
colnames(A) = paste("b", 1:ncol(A), sep = "-")
```

|     | a_1 | a_2 | a_3 |
|-----|-----|-----|-----|
| b_1 | 1   | 5   | 9   |
| b_2 | 2   | 6   | 10  |
| b_3 | 3   | 7   | 11  |
| b_4 | 4   | 8   | 12  |

# Matrices and arrays

Matrix can be created by concatenating columns or rows:

```
cbind(a1 = 1:4, a2 = 5:8, a3 = 9:12) # column bind
```

|      | a1 | a2 | a3 |
|------|----|----|----|
| [1,] | 1  | 5  | 9  |
| [2,] | 2  | 6  | 10 |
| [3,] | 3  | 7  | 11 |
| [4,] | 4  | 8  | 12 |

```
rbind(a1 = 1:4, a2 = 5.8, a3 = 9:12) # row bind
```

|    | [,1] | [,2] | [,3] | [,4] |
|----|------|------|------|------|
| a1 | 1.0  | 2.0  | 3.0  | 4.0  |
| a2 | 5.8  | 5.8  | 5.8  | 5.8  |
| a3 | 9.0  | 10.0 | 11.0 | 12.0 |



# Index elements in matrices

---

|      | [,1] | [,2] | [,3] |
|------|------|------|------|
| [1,] | 1, 1 | 1, 2 | 1, 3 |
| [2,] | 2, 1 | 2, 2 | 2, 3 |
| [3,] | 3, 1 | 3, 2 | 3, 3 |

```
matrix_name[row, column]
```

# Index elements in matrices I

---

```
A[2, 3] # cell in row 2 column 3
```

```
[1] 7
```

```
A[2, ] # second row
```

```
b_1 b_2 b_3 b_4  
  5   6   7   8
```

```
A[, 3] # third column
```

```
a_1 a_2 a_3  
  3   7  11
```

# Index elements in arrays

```
array_name[row, col, tab]
```

```
my_array[2, 1, 3] # cell in 2nd row 1st col of 3rd tab
```

```
my_array[, , 3] # 3rd tab
```

```
my_array[1, ,2] # 1st row in tab 2
```



# Your turn!

---



- Create a  $3 \times 3$  matrix with the 3-times table up to 24
- Assign the matrix to the variable `my_mat`
- Name the row names as “row” and the column names as “column”
- Traspose `my_mat` and assign it to the variable `my_t`
- Index from `my_t`:
  - The first row
  - The second column
  - The cell `[3,3]`

# Lists

---

Can store different objects (e.g., vectors, data frames, other lists):

```
my_list = list(w = weight, m = months, s = ses1, a = A)
```

The components of the list can be indexed with \$ or [[]] and the name (or position) of the component:

Index months:

```
my_list[["m"]] # my_list$m
```

```
[1] 5 6 8 10 12 16
```

Index weight:

```
my_list[[1]] # my_list$weight or my_list[["w"]]
```

```
[1] 3.0 4.5 6.0 7.5 9.0 15.2
```

# Your turn!



- Create a a list with the following elements
  - `my_mat`
  - `my_mat1`
  - The elements `> 25` of `my_var`
  - Assign the list to the variable `my_list`
- Date un nome ad ogni elemento all'interno della lista

# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames**
- 8 Data input and output
- 9 Basics of Programming

Data frames are lists that consist of vectors and factors of equal length.  
The rows in a data frame refer to one unit:

```
id = paste0("sbj", 1:6)
babies = data.frame(id, months, weight)
```

```
babies
```

|   | id   | months | weight |
|---|------|--------|--------|
| 1 | sbj1 | 5      | 3.0    |
| 2 | sbj2 | 6      | 4.5    |
| 3 | sbj3 | 8      | 6.0    |
| 4 | sbj4 | 10     | 7.5    |
| 5 | sbj5 | 12     | 9.0    |
| 6 | sbj6 | 16     | 15.2   |

## Working with data frames

---

Index elements in a data frame:

```
babies$months # column months of babies
```

```
[1]  5  6  8 10 12 16
```

```
babies$months[2] # second element of column months
```

```
[1] 6
```

```
babies[, "id"] # column id
```

```
[1] "sbj1" "sbj2" "sbj3" "sbj4" "sbj5" "sbj6"
```

```
babies[2, ] # second row of babies (obs on baby 2)
```

```
      id months weight  
2 sbj2      6    4.5
```

```
babies[babies$weight > 7, ] # all obs above 7 kg
babies[babies$id %in% c("sbj1", "sbj6"), ] # obs of sbj1
   # and sbj7
```

## Working with data frames II

```
dim(babies) # show the dimensions of the data frame
```

```
[1] 6 3
```

```
names(babies) # variable names (= colnames(babies))
```

```
[1] "id"      "months" "weight"
```

```
View(babies) # open data viewer
```

```
plot(babies) # parwise plot
```

You can use these commands also on other R objects



## Working with data frames III

```
str(babies) # show details on babies
```

```
'data.frame':  6 obs. of  3 variables:
 $ id      : chr  "sbj1" "sbj2" "sbj3" "sbj4" ...
 $ months: num  5 6 8 10 12 16
 $ weight: num  3 4.5 6 7.5 9 15.2
```

```
summary(babies) # descriptive statistics
```

| id               | months       | weight         |
|------------------|--------------|----------------|
| Length:6         | Min. : 5.0   | Min. : 3.000   |
| Class :character | 1st Qu.: 6.5 | 1st Qu.: 4.875 |
| Mode :character  | Median : 9.0 | Median : 6.750 |
|                  | Mean : 9.5   | Mean : 7.533   |
|                  | 3rd Qu.:11.5 | 3rd Qu.: 8.625 |
|                  | Max. :16.0   | Max. :15.200   |

# Sorting

---

order():

```
babies[order(babies$weight), ] # sort by increasing weight
```

|   | id   | months | weight |
|---|------|--------|--------|
| 1 | sbj1 | 5      | 3.0    |
| 2 | sbj2 | 6      | 4.5    |
| 3 | sbj3 | 8      | 6.0    |
| 4 | sbj4 | 10     | 7.5    |
| 5 | sbj5 | 12     | 9.0    |
| 6 | sbj6 | 16     | 15.2   |

```
babies[order(babies$weight,      # sort by decreasing weight  
             decreasing = T), ]
```

Multiple arguments in order:

```
babies[order(babies$weight, babies$months, decreasing = TRUE), ]
```

```
# Single response variable, single grouping variable
aggregate(y ~ x, data = data, FUN, ...)
```

```
# Multiple response variables, multiple grouping variables
aggregate(cbind(y1, y2) ~ x1 + x2, data = data, FUN, ...)
```

## Aggregating: Example

ToothGrowth # Vitamin C and tooth growth (Guinea Pigs)

```
len supp dose
1 4.2 VC 0.5
2 11.5 VC 0.5
3 7.3 VC 0.5
....
```

```
aggregate(len ~ supp + dose, data = ToothGrowth, mean)
```

```
supp dose len
1 OJ 0.5 13.23
2 VC 0.5 7.98
3 OJ 1.0 22.70
4 VC 1.0 16.77
5 OJ 2.0 26.06
6 VC 2.0 26.14
```

## Reshaping: Long to wide

Data can be organized in wide format (i.e., one line for each statistical unit) or in long format (i.e., one line for each observation).

Indometh # Long format

|     | Subject | time | conc |
|-----|---------|------|------|
| 1   | 1       | 0.25 | 1.50 |
| 2   | 1       | 0.50 | 0.94 |
| 3   | 1       | 0.75 | 0.78 |
| 4   | 1       | 1.00 | 0.48 |
| 5   | 1       | 1.25 | 0.37 |
| 6   | 1       | 2.00 | 0.19 |
| ... |         |      |      |

```
# From long to wide
df.w <- reshape(Indometh, v.names = "conc", timevar = "time",
  idvar = "Subject", direction = "wide")
```

|       | Subject | conc.0.25 | conc.0.5 | conc.0.75 | conc.1 | conc.1.25 | conc.2 | conc.3 |
|-------|---------|-----------|----------|-----------|--------|-----------|--------|--------|
| 1     | 1       | 1.50      | 0.94     | 0.78      | 0.48   | 0.37      | 0.19   | 0.09   |
| 12    | 2       | 2.03      | 1.63     | 0.71      | 0.70   | 0.64      | 0.36   | 0.19   |
| 23    | 3       | 2.72      | 1.49     | 1.16      | 0.80   | 0.80      | 0.39   | 0.20   |
| 34    | 4       | 1.85      | 1.39     | 1.02      | 0.89   | 0.59      | 0.40   | 0.20   |
| 45    | 5       | 2.05      | 1.04     | 0.81      | 0.39   | 0.30      | 0.23   | 0.10   |
| 56    | 6       | 2.31      | 1.44     | 1.03      | 0.84   | 0.64      | 0.42   | 0.20   |
|       |         | conc.5    | conc.6   | conc.8    |        |           |        |        |
| ..... |         |           |          |           |        |           |        |        |

# Reshaping: Wide to long

```
# From wide to long
```

```
df.l <- reshape(df.w, varying = list(2:12), v.names = "conc",  
  idvar = "Subject", direction = "long", times = c(0.25, 0.5,  
    0.75, 1, 1.25, 2, 3, 4, 5, 6, 8))
```

|        | Subject | time | conc |
|--------|---------|------|------|
| 1.0.25 | 1       | 0.25 | 1.50 |
| 2.0.25 | 2       | 0.25 | 2.03 |
| 3.0.25 | 3       | 0.25 | 2.72 |
| ....   |         |      |      |

```
df.l[order(df.l$Subject), ] # reorder by subject
```

|        | Subject | time | conc |
|--------|---------|------|------|
| 1.0.25 | 1       | 0.25 | 1.50 |
| 1.0.5  | 1       | 0.50 | 0.94 |
| 1.0.75 | 1       | 0.75 | 0.78 |
| ....   |         |      |      |

# Your turn!



- Create a data frame with 10 observations and the following columns:
  - id: character, respondents IDs
  - ses: factor, socio-economic level with three levels, low, medium, high (3 low, 5 medium, 2 high)
  - income: numeric (e.g., `runif(10, min = 1110, max = 5430)`) possibly coherent with the variable ses
- Filter the data set
  - respondents with 'ses == high'
  - respondents with `income > 2000`



# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output**
- 9 Basics of Programming

## Reading tabular txt files:

ASCII text files in tabular or spread sheet form (one line per observation, one column per variable) are read using `read.table()`

```
data = read.table("C:/Rcourse/file.txt", header = TRUE)
```

data is a data frame where the original numerical variables are converted in numeric vectors and character variables are converted in factors (not always).

Arguments:

- `header`: variable names in the first line? TRUE/FALSE
- `sep`: which separator between the columns (e.g., comma, `\t`)
- `dec`: 1.2 or 1,2?

## Reading other files

---

```
data = read.csv("C:/RcouRse/file.csv",  
                header = TRUE, sep = ";",  
                dec = ",")
```

From SPSS (file .sav):

```
install.packages("foreign")  
library(foreign)  
data = read.spss("data.sav", to.data.frame = TRUE)
```

## Combine data frames

---

If they have the same number of columns/rows

```
all_data = rbind(data, data1, data2) # same columns  
all_data = cbind(data, data1, data2) # same rows
```

If they have different rows/columns but they share at least one characteristic (e.g., ID):

```
all_data = merge(data1, data2,  
                  by = "ID")
```

If there are different IDs in the two datasets → added in new rows

all\_data contains all columns in data1 and data2. The columns of the IDs in data1 but not in data2 (and vice versa) will be filled with NAs accordingly

```
write.table(data, # what you want to write
            file = "mydata.txt", # its name + extension
            header = TRUE, # first row with col names?
            sep = "\t", # column separator
            ....) # other arguments
```

```
save(dat, file = "exp1_data.rda") # save something specific
save(file = "the_earth.rda")      # save the environment
load("the_earth.rda")             # load it back
```

# Table of Contents

---

- 1 Who aRe you?
- 2 Working directories
- 3 Basics
- 4 Get help
- 5 Be tidy
- 6 Structures in R
- 7 Data frames
- 8 Data input and output
- 9 Basics of Programming**



- It works fine until you have simple tests



**if () {} else {}**

---

If you have only one condition:

```
if (test_1) {  
  command_1  
} else {  
  command_2  
}
```

```
if ( ) {} else {}
```

Multiple conditions:

```
if (test_1) {
    command_1
} else if (test_2) {
    command_2
} else {
    command_3
}
```

test\_1 (and test\_2, if you have it) **must** evaluate in either TRUE or FALSE

```
if(!is.na(x)) y <- x^2 else stop("x is missing")
```



## Avoiding loops

## Don't loop, apply()!

## apply()

```
X <- matrix(rnorm(20),
            nrow = 5, ncol = 4)
apply(X, 2, max) # maximum for each column
```

```
[1] 1.4203744 1.0716663 2.3329026 0.9084482
```

# Avoiding loops

---

Don't loop, `apply()`!

`apply()`

```
X <- matrix(rnorm(20),  
            nrow = 5, ncol = 4)  
apply(X, 2, max) # maximum for each column
```

```
[1] 1.4203744 1.0716663 2.3329026 0.9084482
```

`for()`

```
y = NULL  
for (i in 1:ncol(X)) {  
  y[i] = max(X[, i])  
}
```