Data Structures (part 1)

Vectors and Dataframes

Enrico Toffalini

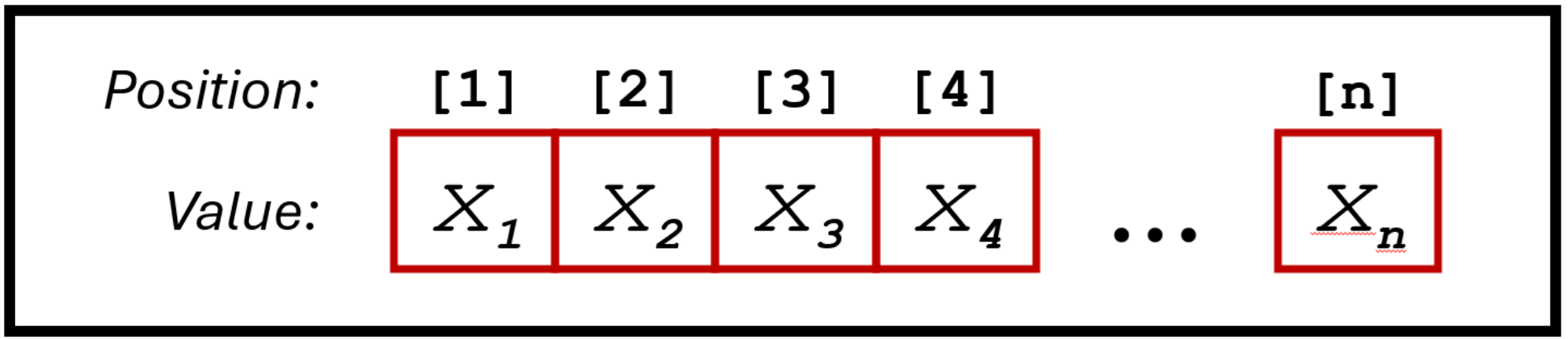
What are data structures

Data structures, like **vectors**, **matrices**, **dataframes**, **lists**, are fundamental tools that allow you to **organize and store complex information**, so that they can be easily **processes by functions** (e.g., `lm()` function to fit a linear model using variables stored in a dataframe)

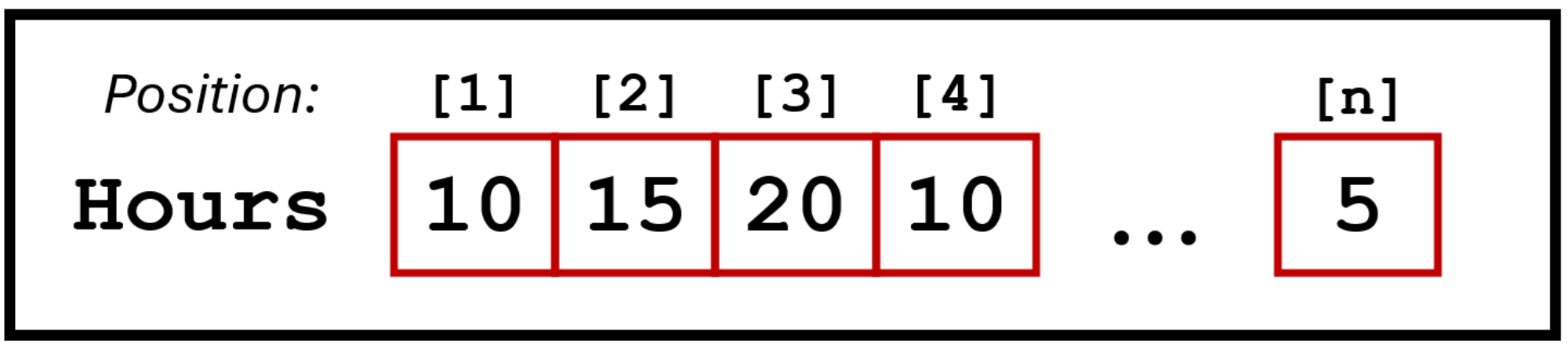
Most operations you will perform in R (e.g., *processing data*, *fitting models*, *plotting outputs*) are performed on these data structures

Vectors

Simple one-dimensional structures that store data of different types



Here is an actual **example** (of a *numerical* vector):



Vectors

Vectors can easily be **created using the `c()`** base function, with a sequence of elements separated by *commas* "`,`"

Vectors can be of different types. The following example shows a *character* vector (note the *quotes* `" "` around objects):

```
Teachers = c("Pastore", "Kiesner", "Granziol", "Toffalini",
             "Calignano", "Epifania", "Bastianelli")
```

or numeric:

```
Hours = c(10, 15, 20, 10, 15, 5, 15, 5)
```

Vectors

Vectors must contain elements of the **same type**. If you mix types, R will automatically **coerce** the elements to a single type, which may lead to undesired results.

Therefore, **avoid mixing data types!** Example:

```
        Hours = c(10, 15, 20, 10, 15, "tbd", 15, 5)
        Hours
```

```
[1] "10"  "15"  "20"  "10"  "15"  "tbd" "15"  "5"
```

everything was coerced to become a character!

If needed, use **NA** (Not Available):

```
        Hours = c(10, 15, 20, 10, 15, NA, 15, 5)
        Hours # remains a numerical vector, NA does not affect typ
```

```
[1] 10 15 20 10 15 NA 15  5
```

Vectors

Select/extract elements with **INDEXING** using square brackets `[]`:

```
Hours = c(10, 15, 20, 10, 15, 5, 15, 5)
Hours[4] # a single element
```

```
[1] 10
```

```
Hours[5:7] # a range of elements
```

```
[1] 15  5 15
```

```
Hours[c(1,3,6)] # specific elements
```

```
[1] 10 20  5
```

Know the **length** of a vector using the `length()` function, and use it:

```
length(Hours)
```

```
[1] 8
```

```
Hours[length(Hours)] # use it to extract the last element
```

```
[1] 5
```

Vectors

Negative indexing

You can use the *minus* sign **-** to select **all elements except some** from a vector. (This method is also applicable to dataframes)

```
        Hours = c(10, 15, 20, 10, 15, 5, 15, 5)
        Hours[-4] # ALL BUT a single element
```

```
[1] 10 15 20 15  5 15  5
```

```
        Hours[-c(5:7)] # ALL BUT a range of elements
```

```
[1] 10 15 20 10  5
```

```
        Hours[-c(1,3,6)] # ALL BUT specific elements
```

```
[1] 15 10 15 15  5
```

```
        Hours[-length(Hours)] # ALL BUT the last element
```

```
[1] 10 15 20 10 15  5 15
```

Vectors

Logical indexing

Often, you'll need to extract values from a vector based on specific ***logical*** conditions. Here's an example:

```
        Hours = c(10, 15, 20, 10, 15, 5, 15, 5)
        Hours[Hours >= 15] # extract only values greater than or equal to 15
```

```
[1] 15 20 15 15
```

This is called ***logical indexing*** because you are selecting elements based on a logical vector (i.e., a sequence of TRUE, FALSE):

```
        Hours >= 15 # the logical vector actually inside the square brackets
```

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

Also, you can use a vector to extract values **from another vector**:

```
        Teachers[Hours >= 15]
```

```
[1] "Kiesner"     "Granziol"    "Calignano"   "Bastianelli"
```

Vectors

Operations

you can simultaneously apply an operation to a whole vector, like

```
Hours = c(10, 15, 20, 10, 15, 5, 15, 5)
Hours / 5
```

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

Of course, this is useful when you want to save the result as a new vector:

```
ECTS = Hours / 5
```

Similarly, you can apply functions to all elements of a vector:

```
sqrt(Hours) # computes square root of each element
```

```
[1] 3.162278 3.872983 4.472136 3.162278 3.872983 2.236068 3.872983 2.236068
```

```
log(Hours) # computes the natural logarithm of each element
```

```
[1] 2.302585 2.708050 2.995732 2.302585 2.708050 1.609438 2.708050 1.609438
```

Vectors

Summary statistics

A whole vector may serve to compute summary statistics, for example using functions such as **`mean()`**, **`sd()`**, **`median()`**, **`quantile()`**, **`max()`**, **`min()`**:

```
        mean(Hours) # returns the average value (mean) of the vecto
```

```
[1] 11.875
```

```
        sd(Hours) # returns the Standard Deviation of the vector
```

```
[1] 5.303301
```

```
        median(Hours) # returns the median value of the vector
```

```
[1] 12.5
```

Vectors

Summary statistics

A whole vector may serve to compute summary statistics, for example using functions such as **`mean()`**, **`sd()`**, **`median()`**, **`quantile()`**, **`max()`**, **`min()`**:

```
        quantile(Hours, probs=c(.25, .50, .75)) # returns desired q
```

```
  25%   50%   75%
 8.75 12.50 15.00
```

```
        max(Hours) # returns largest value
```

```
[1] 20
```

```
        min(Hours) # returns smallest value
```

```
[1] 5
```

Vectors

Summary statistics - Managing missing (NA) values

All of the previous summary statistics will **fail** if there is even a single NA value:

```
        Hours = c(10, 15, 20, 10, 15, NA, 15, 5)

        mean(Hours) # a single NA value implies that the average is impossible
```

```
[1] NA
```

```
        quantile(Hours, probs=c(.25, .75)) # quantile() will even return an Er
```

```
Error in quantile.default(Hours, probs = c(0.25, 0.75)): missing values
and NaN's not allowed if 'na.rm' is FALSE
```

You can easily manage missing values by adding the **na.rm=TRUE** argument:

```
        mean(Hours, na.rm=TRUE) # NA values are ignored
```

```
[1] 12.85714
```

```
        quantile(Hours, probs=c(.25, .75), na.rm=TRUE) # NA values are ignored
```

```
25% 75%
 10  15
```

Vectors

Example: replacing NA with the average value

Replacing a missing value with the average across valid values is risky, as it may alter many other summary statistics, but it is a good example for understanding different concepts seen so far:

```
Hours = c(10, 15, 20, 10, 15, NA, 15, 5)

# compute the average value ignoring NAs, and put it wherever
# there is a NA value in the vector
Hours[is.na(Hours)] = mean(Hours, na.rm=TRUE)

# now let's inspect the updated content of the vector
Hours
```

```
[1] 10.00000 15.00000 20.00000 10.00000 15.00000 12.85714 15.00000  5.00000
```

```
# by the way... na.rm=TRUE is no longer needed now, as NA is no longer
mean(Hours)
```

```
[1] 12.85714
```

Vectors

Summary statistics

Another useful summary statistic is the **frequency count**, which shows how often each unique value appears in a vector. You can use the `table()` function to calculate frequencies easily:

```
        type = c("METHODOLOGY", "METHODOLOGY", "PROGRAMMING", "SOFT SKILLS", "S
                 "METHODOLOGY", "SOFT SKILLS", "METHODOLOGY", "PROGRAMMING")
        table(type)
```

```
type
METHODOLOGY PROGRAMMING SOFT SKILLS
          4           2           3
```

Be careful: R is case sensitive!

```
        type = c("METHODOLOGY", "methodology", "PROGRAMMING", "SOFT SKILLS", "S
                 "METHODOLOGY", "SOFT SKILLS", "METHODOLOGY", "Programming")
        table(type)
```

```
type
methodology METHODOLOGY Programming PROGRAMMING SOFT SKILLS
          1           3           1           1           3
```

A type of data structure you are already familiar with

	A	B	C	D	E	F	G	H	I
1	Year	Type of course	Title	Teacher	Hours	ECTS	Mandatory	Delivery	Language
2	1	METHODOLOGY	CURRENT ISSUES IN STATISTICAL INFERENCE FOR	MASSIMILIANO PASTORE	10	2	YES	IN PERSON	ENGLISH
3	1	METHODOLOGY	LINEAR AND MIXED EFFECT MODELS WITH SPSS	JEFF KIESNER	15	3	NO	IN PERSON	ENGLISH
4	1	METHODOLOGY	BASICS OF STATISTICAL INFERENCE WITH R	UMBERTO GRANZIOL	20	4	YES	IN PERSON	ENGLISH
5	1	PROGRAMMING	BASICS OF R FOR DATA SCIENCE	ENRICO TOFFALINI	10	2	YES	IN PERSON	ENGLISH
6	1	METHODOLOGY	PSYCHOLOGICAL MEASUREMENT	LUCA STEFANUTTI	15	3	YES	IN PERSON	ENGLISH
7	1	METHODOLOGY	POWER AND DESIGN ANALYSIS	GIANMARCO ALTOE	5	1	YES	IN PERSON	ENGLISH
8	1	METHODOLOGY	EVALUATION OF OUTLIERS AND INFLUENTIAL	GIANMARCO ALTOE	5	1	NO	IN PERSON	ENGLISH
9	1	METHODOLOGY	QUESTIONABLE MEASUREMENT PRACTICES AND	TATIANA MARCI	5	1	YES	IN PERSON	ENGLISH
10	1	METHODOLOGY	DATA VISUALISATION WITH GGPLOT2	MICHELE VICOVARO	5	1	NO	IN PERSON	ENGLISH
11	1	SOFT SKILLS	CRAFTING EFFECTIVE SCIENTIFIC	FILIPPO GAMBAROTA	5	1	NO	IN PERSON	ENGLISH
12	1	PROGRAMMING	BASICS OF MATLAB FOR DATA SCIENCE	LUCA STEFANUTTI	10	2	NO	IN PERSON	ENGLISH
13	1	PROGRAMMING	BASICS OF PYTHON FOR DATA SCIENCE	ENRICO TOFFALINI	10	2	NO	IN PERSON	ENGLISH
14	2-3	SOFT SKILLS	ADVANCING RESEARCH PARADIGMS: OPEN	GIULIA CALIGNANO	5	1	NO	IN PERSON	ENGLISH
15	2-3	THEMATIC COURSE	NEUROPSYCHOLOGY OF VISION	LUCA BATTAGLINI	5	1	NO	IN PERSON	ENGLISH
16	2-3	METHODOLOGY	METHODOLOGY IN NEUROSCIENCES	SIMONE CUTINI	10	2	NO	IN PERSON	ENGLISH
17	2-3	METHODOLOGY	BAYESIAN DATA ANALYSIS IN PSYCHOLOGICAL	MASSIMILIANO PASTORE	10	2	NO	IN PERSON	ENGLISH
18	2-3	METHODOLOGY	GENERALISED LINEAR MODELS	FILIPPO GAMBAROTA	15	3	NO	IN PERSON	ENGLISH
19	2-3	METHODOLOGY	STRUCTURAL EQUATION MODELING	TOMMASO FERACO	20	4	NO	IN PERSON	ENGLISH
20	2-3	METHODOLOGY	CONDUCTING SYSTEMATIC REVIEWS	ENRICO SELLA	5	1	NO	IN PERSON	ENGLISH
21	2-3	METHODOLOGY	INTRODUCTION TO ITEM RESPONSE THEORY	MARINA OTTAVIA EPIFANIA	15	3	NO	IN PERSON	ENGLISH
22	2-3	SOFT SKILLS	HOW TO WIN RESEARCH GRANTS	CHRISTIAN AGRILLO	5	1	NO	IN PERSON	ENGLISH
23	2-3	SOFT SKILLS	CAREER COUNSELING	NICOLA CELLINI	10	2	NO	IN PERSON	ENGLISH
24	2-3	SOFT SKILLS	OUTSIDE ACADEMIA	ALESSIA BASTIANELLI	5	1	NO	IN PERSON	ENGLISH
25	2-3	METHODOLOGY	INTRODUCTION TO META-ANALYSIS WITH	GIANMARCO ALTOE	5	1	NO	IN PERSON	ENGLISH
26	2-3	METHODOLOGY	DATA SIMULATION IN PSYCHOLOGICAL STUDIES	MASSIMILIANO PASTORE	10	2	NO	IN PERSON	ENGLISH
27	2-3	SOFT SKILLS	PUBLISHING IN HIGH-IMPACT JOURNALS	MARA CADINU	15	3	NO	IN PERSON	ENGLISH
28	2-3	THEMATIC COURSE	PSYCHONEUROENDOCRINOLOGY	JEFF KIESNER	5	1	NO	IN PERSON	ENGLISH
29	2-3	PROGRAMMING	BASICS OF LINUX FOR DATA SCIENCE	FRANCESCO VESPIGNANI	5	1	NO	IN PERSON	ENGLISH

Dataframes

here is how I would import it in R (download here), and display the first few rows:

```
library(readxl)
df = data.frame(read_excel("data/Courses40Cycle.xlsx"))
head(df)
```

```
  Year TypeOfCourse                                                    Title
1    1  METHODOLOGY CURRENT ISSUES IN STATISTICAL INFERENCE FOR PSYCHOLOGY
2    1  METHODOLOGY               LINEAR AND MIXED EFFECT MODELS WITH SPSS
3    1  METHODOLOGY                 BASICS OF STATISTICAL INFERENCE WITH R
4    1  PROGRAMMING                             BASICS OF R FOR DATA SCIENCE
5    1  METHODOLOGY                            PSYCHOLOGICAL MEASUREMENT
6    1  METHODOLOGY                            POWER AND DESIGN ANALYSIS
              Teacher Hours ECTS MandatoryAttendance DeliveryMethod Language
1 MASSIMILIANO PASTORE    10    2                YES      IN PERSON  ENGLISH
2         JEFF KIESNER    15    3                 NO      IN PERSON  ENGLISH
3     UMBERTO GRANZIOL    20    4                YES      IN PERSON  ENGLISH
4     ENRICO TOFFALINI    10    2                YES      IN PERSON  ENGLISH
5      LUCA STEFANUTTI    15    3                YES      IN PERSON  ENGLISH
6      GIANMARCO ALTOE     5    1                YES      IN PERSON  ENGLISH
```

Dataframes

In fact, **dataframes** are just collections (lists) of **vectors** of different types, all with the same length. Each column in a dataframe is a vector (a variable):

```
df$Teacher
```

```
 [1] "MASSIMILIANO PASTORE"    "JEFF KIESNER"
 [3] "UMBERTO GRANZIOL"        "ENRICO TOFFALINI"
 [5] "LUCA STEFANUTTI"         "GIANMARCO ALTOE"
 [7] "GIANMARCO ALTOE"         "TATIANA MARCI"
 [9] "MICHELE VICOVARO"        "FILIPPO GAMBAROTA"
[11] "LUCA STEFANUTTI"         "ENRICO TOFFALINI"
[13] "GIULIA CALIGNANO"        "LUCA BATTAGLINI"
[15] "SIMONE CUTINI"           "MASSIMILIANO PASTORE"
[17] "FILIPPO GAMBAROTA"       "TOMMASO FERACO"
[19] "ENRICO SELLA"            "MARINA OTTAVIA EPIFANIA"
[21] "CHRISTIAN AGRILLO"       "NICOLA CELLINI"
[23] "ALESSIA BASTIANELLI"     "GIANMARCO ALTOE"
[25] "MASSIMILIANO PASTORE"    "MARA CADINU"
[27] "JEFF KIESNER"            "FRANCESCO VESPIGNANI"
```

```
df$Hours
```

```
 [1] 10 15 20 10 15  5  5  5  5  5 10 10  5  5 10 10 15 20  5 15  5 10  5  5
10
[26] 15  5  5
```

Dataframes

To know the names of all variables in a dataframe, use the **`names()`** function:

```
names(df)
```

```
[1] "Year"                "TypeOfCourse"        "Title"
[4] "Teacher"             "Hours"               "ECTS"
[7] "MandatoryAttendance" "DeliveryMethod"      "Language"
```

Use the **`dim()`** function to view the dimensions of a dataframe:

```
# first value is number of rows, second is number of columns (variables
dim(df)
```

```
[1] 28  9
```

Alternatively, you can use **`nrow()`** and **`ncol()`**:

```
nrow(df) # number of rows
```

```
[1] 28
```

```
ncol(df) # number of columns
```

```
[1] 9
```

Dataframes

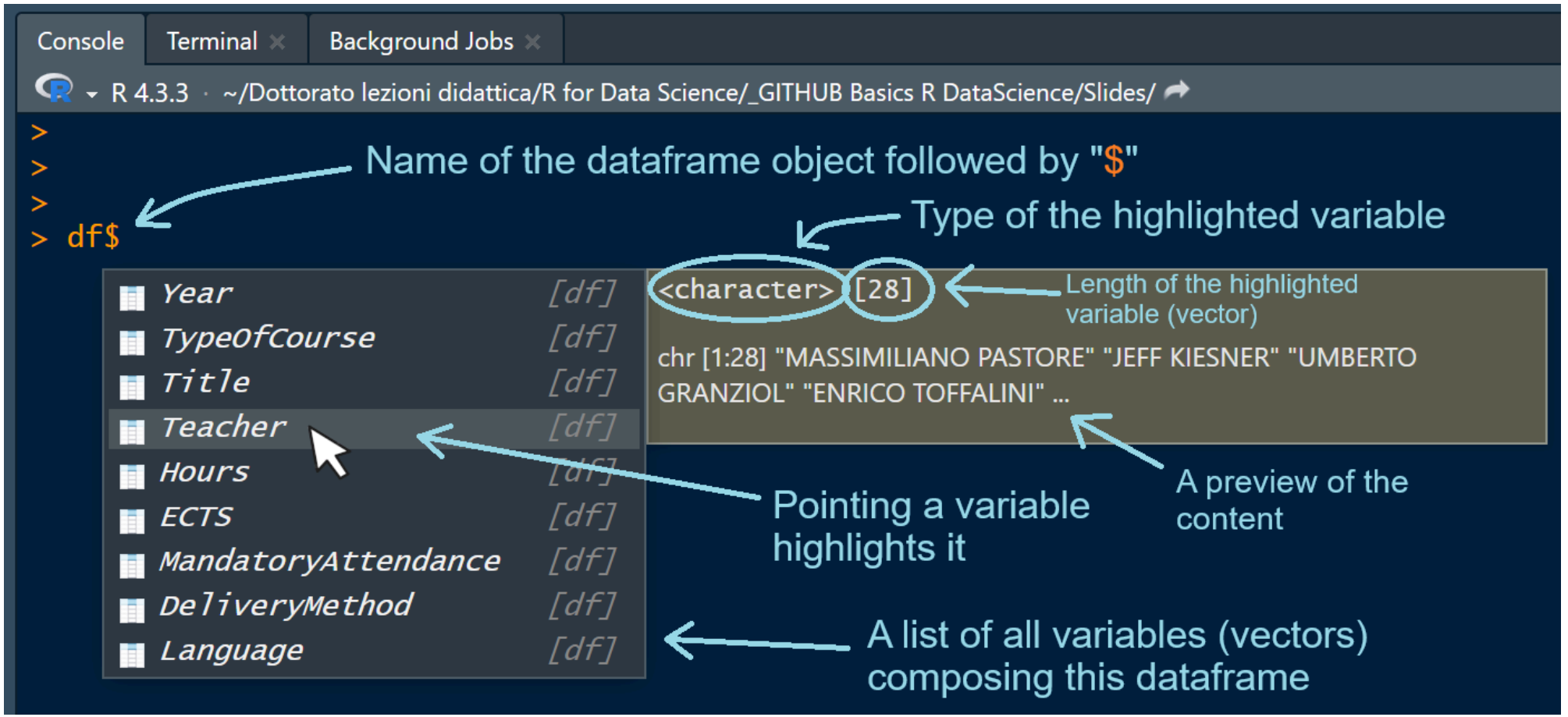
The **`str()`** function provides a quick overview of the structure of a dataframe, including its dimensions, variables, their data types, and first few observations:

```
str(df)
```

```
'data.frame':   28 obs. of  9 variables:
 $ Year               : chr  "1" "1" "1" "1" ...
 $ TypeOfCourse       : chr  "METHODOLOGY" "METHODOLOGY" "METHODOLOGY"
"PROGRAMMING" ...
 $ Title              : chr  "CURRENT ISSUES IN STATISTICAL INFERENCE FOR
PSYCHOLOGY" "LINEAR AND MIXED EFFECT MODELS WITH SPSS" "BASICS OF STATISTICAL
INFERENCE WITH R" "BASICS OF R FOR DATA SCIENCE" ...
 $ Teacher            : chr  "MASSIMILIANO PASTORE" "JEFF KIESNER" "UMBERTO
GRANZIOL" "ENRICO TOFFALINI" ...
 $ Hours              : num  10 15 20 10 15 5 5 5 5 5 ...
 $ ECTS               : num  2 3 4 2 3 1 1 1 1 1 ...
 $ MandatoryAttendance: chr  "YES" "NO" "YES" "YES" ...
 $ DeliveryMethod     : chr  "IN PERSON" "IN PERSON" "IN PERSON" "IN PERSON"
...
 $ Language           : chr  "ENGLISH" "ENGLISH" "ENGLISH" "ENGLISH" ...
```

Dataframes

The "$" (dollar) operator is essential to access variables in a dataframe:



Dataframes

As an exercise, let's check whether `ECTS` is actually always `Hours*5`

We may use many different, increasingly sophisticated, strategies:

```
        df$Hours / df$ECTS
```

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

```
        (df$Hours / df$ECTS) == 5
```

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

```
        sum((df$Hours / df$ECTS) == 5)
```

```
[1] 28
```

```
        sum((df$Hours / df$ECTS) == 5) == nrow(df)
```

```
[1] TRUE
```

```
        sum((df$Hours / df$ECTS) != 5)
```

```
[1] 0
```

Dataframes

Variables in a dataframe can be manipulated like any other vector:

```
log(df$Hours)
```

```
 [1] 2.302585 2.708050 2.995732 2.302585 2.708050 1.609438 1.609438 1.609438
 [9] 1.609438 1.609438 2.302585 2.302585 1.609438 1.609438 2.302585 2.302585
[17] 2.708050 2.995732 1.609438 2.708050 1.609438 2.302585 1.609438 1.609438
[25] 2.302585 2.708050 1.609438 1.609438
```

Also, new variables can easily be created and added at any time:

```
df$newVar = log(df$Hours)
names(df)
```

```
 [1] "Year"                "TypeOfCourse"        "Title"
 [4] "Teacher"             "Hours"               "ECTS"
 [7] "MandatoryAttendance" "DeliveryMethod"      "Language"
[10] "newVar"
```

```
df$newVar
```

```
 [1] 2.302585 2.708050 2.995732 2.302585 2.708050 1.609438 1.609438 1.609438
 [9] 1.609438 1.609438 2.302585 2.302585 1.609438 1.609438 2.302585 2.302585
[17] 2.708050 2.995732 1.609438 2.708050 1.609438 2.302585 1.609438 1.609438
[25] 2.302585 2.708050 1.609438 1.609438
```

Dataframes

Indexing elements in a dataframe

In addition to using the "$" (dollar) operator, you can directly access a variable of a dataframe using **indexing** with **square brackets** `[]`:

```
        df[ , "Hours"]
```

```
 [1] 10 15 20 10 15  5  5  5  5  5 10 10  5  5 10 10 15 20  5 15  5 10  5  5
10
[26] 15  5  5
```

Notice the comma "`,`" above.

Unlike vectors, dataframes must be indexed by **both row and column**. In the example above, we're specifying only the desired column (`"Hours"`), leaving the row index blank before the comma `,` This selects all rows for the column named `"Hours"`. Remember that blank index means "all". Importantly, the "`,`" must always be there when indexing dataframes!

Dataframes

Indexing elements in a dataframe - Examples

```
        df[ 1 , "Hours"]
```

```
[1] 10
```

```
        df[ 1:5 , "Hours"]
```

```
[1] 10 15 20 10 15
```

```
        df[ 1 , c("Teacher","Hours","TypeOfCourse")]
```

```
              Teacher Hours TypeOfCourse
1 MASSIMILIANO PASTORE    10  METHODOLOGY
```

```
        df[ 1:5 , c("Teacher","Hours","TypeOfCourse")]
```

```
              Teacher Hours TypeOfCourse
1 MASSIMILIANO PASTORE    10  METHODOLOGY
2         JEFF KIESNER    15  METHODOLOGY
3     UMBERTO GRANZIOL    20  METHODOLOGY
4     ENRICO TOFFALINI    10  PROGRAMMING
5      LUCA STEFANUTTI    15  METHODOLOGY
```

```
        df[ 1 , c(4, 5, 2)]
```

```
              Teacher Hours TypeOfCourse
1 MASSIMILIANO PASTORE    10  METHODOLOGY
```

Dataframes

Logical indexing

Just like for vectors, you can use a logical condition for indexing a dataframe.

Let's consider this logical condition:

```
df$Teacher == "MASSIMILIANO PASTORE"
```

```
 [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25]  TRUE FALSE FALSE FALSE
```

Let's use it to extract some dataframe rows:

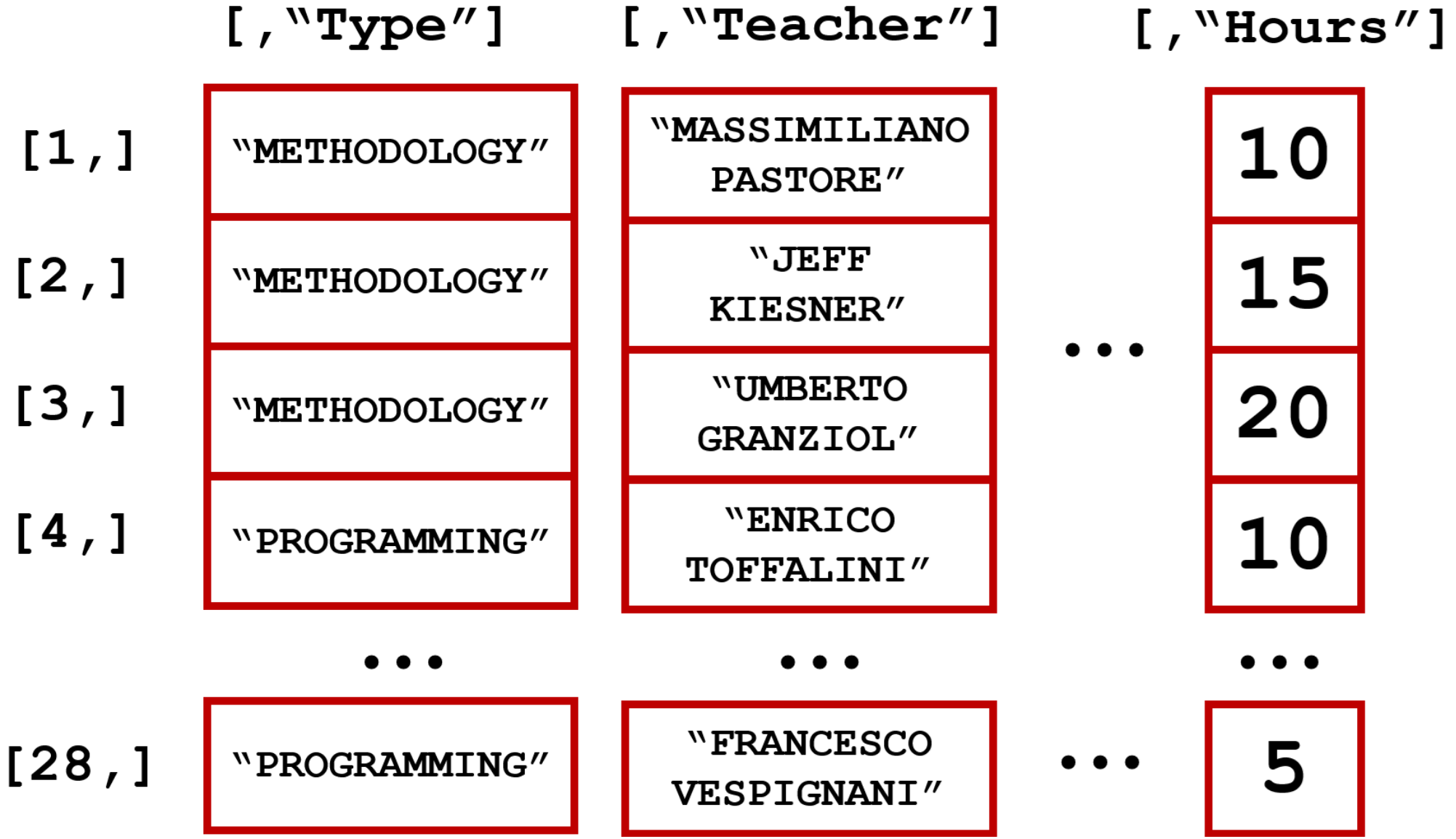
```
df[df$Teacher == "MASSIMILIANO PASTORE" , ] # some rows by condition ,
```

```
   Year TypeOfCourse                                                  Title
1     1  METHODOLOGY CURRENT ISSUES IN STATISTICAL INFERENCE FOR PSYCHOLOGY
16  2-3  METHODOLOGY       BAYESIAN DATA ANALYSIS IN PSYCHOLOGICAL RESEARCH
25  2-3  METHODOLOGY                 DATA SIMULATION IN PSYCHOLOGICAL STUDIES
                Teacher Hours ECTS MandatoryAttendance DeliveryMethod Language
1  MASSIMILIANO PASTORE    10    2                YES      IN PERSON  ENGLISH
16 MASSIMILIANO PASTORE    10    2                 NO      IN PERSON  ENGLISH
25 MASSIMILIANO PASTORE    10    2                 NO      IN PERSON  ENGLISH
     newVar
```

```
1  2.302585
16 2.302585
25 2.302585
```

Dataframes

Indexing elements in a dataframe - A summary


[,"Type"]
[,"Teacher"]
[,"Hours"]
[1,]
"METHODOLOGY"
"MASSIMILIANO PASTORE"
10
[2,]
"METHODOLOGY"
"JEFF KIESNER"
15
[3,]
"METHODOLOGY"
"UMBERTO GRANZIOL"
20
[4,]
"PROGRAMMING"
"ENRICO TOFFALINI"
10
[28,]
"PROGRAMMING"
"FRANCESCO VESPIGNANI"
5

Dataframes

Subset

Base function **`subset()`** can also be used as an alternative to indexing

```
subset(df, Teacher == "MASSIMILIANO PASTORE", select=c("Teacher","Hours
```

```
                Teacher Hours TypeOfCourse
1  MASSIMILIANO PASTORE    10  METHODOLOGY
16 MASSIMILIANO PASTORE    10  METHODOLOGY
25 MASSIMILIANO PASTORE    10  METHODOLOGY
```

```
df[df$Teacher == "MASSIMILIANO PASTORE" , c("Teacher","Hours","TypeOfC
```

```
                Teacher Hours TypeOfCourse
1  MASSIMILIANO PASTORE    10  METHODOLOGY
16 MASSIMILIANO PASTORE    10  METHODOLOGY
25 MASSIMILIANO PASTORE    10  METHODOLOGY
```

However, indexing with **`[ ]`** is more "computationally focused", computationally faster (especially if working with large datasets), and more similar to programming in other languages (e.g., **Python**), so should probably be favoured by data scientists!

Dataframes

Combine two dataframes using `rbind()`

Imagine you have two datasets collected by two students, each including different participants:

```
df1
```

```
  subjName age accuracy
1    Julie  12     0.92
2    Tommy  10     0.78
3     Phil  10     0.85
```

```
df2
```

```
  subjName age accuracy
1    Amber   9     0.87
2      Max  13     0.90
```

Our goal is to get one single dataset including all participant's data for the final analysis. Of course, you could combined these files manually outside R (e.g., in Excel). However, it's simpler and more efficient to do this directly in R using `rbind()`

Dataframes

Combine two dataframes using `rbind()`

```
dfTotal = rbind(df1, df2)

dfTotal
```

```
  subjName age accuracy
1    Julie  12     0.92
2    Tommy  10     0.78
3     Phil  10     0.85
4    Amber   9     0.87
5      Max  13     0.90
```

Important: for `rbind()` to work, the two to-be-combined dataframes must:

- have the **exact same number of columns**;
- the **column names must be identical** (remember that R is case-sensitive).

Dataframes

Merge two dataframes using `merge()`

Another frequent case is having data collected from the **same participants** across **different dataframes**, and having to analyze all information together:

```
df1
```

```
  subjName age
1    Julie  12
2    Amber   9
3    Tommy  10
4     Phil  10
```

```
df2
```

```
  subjName accuracy time
1    Julie     0.92 1203
2    Tommy     0.78 3302
3     Phil     0.85  994
4    Amber     0.87 1163
```

Dataframes

Merge two dataframes using `merge()`

You can merge the two dataframes into a single, comprehensive dataframe:

```
dfTotal = merge(df1, df2, by="subjName")

dfTotal
```

```
  subjName age accuracy time
1    Amber   9     0.87 1163
2    Julie  12     0.92 1203
3     Phil  10     0.85  994
4    Tommy  10     0.78 3302
```

Important: `merge()` will work even if some or even all values that should be used for merging do not match… but in that case part of or all data will be lost

Dataframes

Contingency tables

The **`table()`** function, which counts frequencies, can also be used on dataframes. Importantly, it can also create **contingency tables** when applied to multiple variables at once

```
table(df$Hours) # just counts frequencies
```

```
 5 10 15 20
13  8  5  2
```

```
table(df$TypeOfCourse, df$Hours) # creates contingency tab
```

```
                  5 10 15 20
  METHODOLOGY     6  4  4  2
  PROGRAMMING     1  3  0  0
  SOFT SKILLS     4  1  1  0
  THEMATIC COURSE 2  0  0  0
```