Using data in R

Alexandre Courtiol

Leibniz Institute of Zoo and Wildlife Research

June 2018

Table of contents

Table of contents

Vector

A vector is a sequence of data elements of the same basic type

- Vectors allow the organisation of entities (e.g. numbers, characters. . .) along one dimension which can be indexed

```r
height.girls <- c(178, 175, 159, 164, 183, 192)
height.boys <- c(181, 189, 174, 177)
```

```r
height.girls[2]

## [1] 175

height.boys[3]

## [1] 174
```

Vector

- They can be combined:

```
(height <- c(height.boys, height.girls))

##  [1] 181 189 174 177 178 175 159 164 183 192
```

Vector continued

- They can be indexed logically (i.e. indexed by anything leading to a vector of booleans):

```
(height > 168)

##  [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
## [10]  TRUE
```

```
height[height > 168]

## [1] 181 189 174 177 178 175 183 192

height[!(height == min(height))]

## [1] 181 189 174 177 178 175 164 183 192

height[height != min(height)]

## [1] 181 189 174 177 178 175 164 183 192
```

type of vectors

- character

```
x <- c("bla", "1", "sf", "xx3")
summary(x)

##    Length     Class      Mode
##         4 character character
```

- factor

```
x <- as.factor(c("bla", "1", "sf", "xx3"))
str(x)

##  Factor w/ 4 levels "1","bla","sf",..: 2 1 3 4
```

- logical

```
x <- c(TRUE, FALSE, T, F)
is.logical(x)

## [1] TRUE
```

- Dates

```
x <- c(as.Date("1999-01-12"), lubridate::ymd("1999-03-12"))
str(x)

##  Date[1:2], format: "1999-01-12" "1999-03-12"
```

cool stuff with dates

- Dates object

```
x

## [1] "1999-01-12" "1999-03-12"
```

- compute differences

```
x[1] - x[2]

## Time difference of -59 days

as.numeric(x[1] - x[2])

## [1] -59
```

- logical

```
x[1] > x[2]

## [1] FALSE

x[x > as.Date("1999-02-12")]

## [1] "1999-03-12"
```

Factors

- They work with other things than numbers:

```r
sex <- c("girl","girl","girl","girl","girl", "girl",
"boy","boy","boy","boy")
sex <- factor(sex)
sex

##  [1] girl girl girl girl girl girl boy  boy  boy  boy
## Levels: boy girl
```

```r
# Or
sex <- factor(c(rep("girl", times = 6),
              rep("boy", times = 4)))
# Or

sex <- factor(c(rep("girl", times = length(height.girls)),
              rep("boy", times = length(height.boys))))
```

Changing the order of levels of a factor

You have:

```
my_factor1

## [1] A A B B C
## Levels: A B C
```

You want:

```
my_factor2

## [1] A A B B C
## Levels: C B A
```

Changing the order of levels of a factor

You have:

```
my_factor1
## [1] A A B B C
## Levels: A B C
```

You want:

```
my_factor2
## [1] A A B B C
## Levels: C B A
```

You do:

```
## Using base:
my_factor2 <- factor(my_factor1, levels(my_factor1)[c(3, 2, 1)])
my_factor2
## [1] A A B B C
## Levels: C B A
```

Changing the order of levels of a factor

You have:

```
my_factor1

## [1] A A B B C
## Levels: A B C
```

You want:

```
my_factor2

## [1] A A B B C
## Levels: C B A
```

You do:

```
## Using base:
my_factor2 <- factor(my_factor1, levels(my_factor1)[c(3, 2, 1)])
my_factor2

## [1] A A B B C
## Levels: C B A
```

Note: the order of levels influences the output of linear models and plotting functions (e.g. order in the legend of a ggplot) . . .

Changing the levels of a factor

You have:

```
my_factor1

## [1] A A B B C
## Levels: A B C
```

You want:

```
my_factor2

## [1] A A A A D
## Levels: A D
```

You do:

```
## Using base:
levels(my_factor1)

## [1] "A" "B" "C"

my_factor2 <- my_factor1
levels(my_factor2) <- c("A", "A", "D") ## in same order!
my_factor2

## [1] A A A A D
## Levels: A D
```

```
## Using dplyr:
my_factor2 <- recode(my_factor1, A = "A", B = "A", C = "D")
my_factor2

## [1] A A A A D
## Levels: A D
```

Data frames

Data frames allow the organisation of entities as a matrix-like structure whose columns have the same length:

```
dataframe.ht <- data.frame(Height = height, Sex = sex)
dataframe.ht
```

```
##    Height  Sex
## 1     181 girl
## 2     189 girl
## 3     174 girl
## 4     177 girl
## 5     178 girl
## 6     175 girl
## 7     159  boy
## 8     164  boy
## 9     183  boy
## 10    192  boy
```

Data frames

It is good practice to always check their structure:

```
str(dataframe.ht)

## 'data.frame': 10 obs. of  2 variables:
##  $ Height: num  181 189 174 177 178 175 159 164 183 192
##  $ Sex   : Factor w/ 2 levels "boy","girl": 2 2 2 2 2 2 1 1 1 1
```

Data frames

You access the columns by means of the extractor $

```
height
##  [1] 181 189 174 177 178 175 159 164 183 192
rm(list = c("height", "sex")) # removing original vectors
height
## Error in eval(expr, envir, enclos):  object 'height' not found
dataframe.ht$Height #Or: with(data = dataframe.ht, Height)
##  [1] 181 189 174 177 178 175 159 164 183 192
```

⇒ What is the average height?

Data frames

Some functions can take a data frame as an input:

```
summary(dataframe.ht)
```

```
##      Height          Sex   
##  Min.   :159.0   boy :4  
##  1st Qu.:174.2   girl:6  
##  Median :177.5           
##  Mean   :177.2           
##  3rd Qu.:182.5           
##  Max.   :192.0           
```

Note: this will be the case of a lot of functions performing statistical tests!

Data frames

How to compute the average height per sex?

- simple

```
mean(dataframe.ht$Height[dataframe.ht$Sex == "boy"])

## [1] 174.5
```

- more elegant

```
tapply(X = dataframe.ht$Height, INDEX = dataframe.ht$Sex,
       FUN = mean)

##   boy  girl
## 174.5 179.0

# Or: with(data = dataframe.ht, tapply(X = Height,
#   INDEX = Sex, FUN = mean))
```

- even more elegant but dangerous

```
library(dplyr)
dataframe.ht %>% group_by(Sex) %>% summarize(mean = mean(Height))  ## be aware of the rounding

## # A tibble: 2 x 2
##   Sex     mean
##   <fct> <dbl>
## 1 boy    174.
## 2 girl   179
```

Data frames

They can also be indexed:

```
dataframe.ht[1, ]

##   Height  Sex
## 1    181 girl

dataframe.ht[, 1]  # Or: dataframe.ht[, "Sex"]

##  [1] 181 189 174 177 178 175 159 164 183 192
```

Data frames

They can be edited:

```
dataframe.ht[1, 1]

## [1] 181

dataframe.ht[1, 1] <- 171.3
dataframe.ht[1, 1]

## [1] 171.3

dataframe.ht$linenumber <- 1:nrow(dataframe.ht)  # add column
ncol(dataframe.ht) # try dim()

## [1] 3

dataframe.ht$linenumber <- NULL  # remove column
ncol(dataframe.ht)

## [1] 2
```

Data frames

They can be edited with dpylr way

add column with `mutate()`

```
dataframe.ht <- dataframe.ht %>% mutate(linenumber = 1:nrow(dataframe.ht))
head(dataframe.ht, n= 3)
```

```
##   Height  Sex linenumber
## 1  171.3 girl          1
## 2  189.0 girl          2
## 3  174.0 girl          3
```

Data frames

They can be edited with dpylr way

add column with `mutate()`

```
dataframe.ht <- dataframe.ht %>% mutate(linenumber = 1:nrow(dataframe.ht))
head(dataframe.ht, n= 3)
```

```
##   Height  Sex linenumber
## 1  171.3 girl          1
## 2  189.0 girl          2
## 3  174.0 girl          3
```

select columns with `select()`

```
dataframe.ht.sex <- dataframe.ht %>% select(Sex)
head(dataframe.ht.sex, n= 3)
```

```
##    Sex
## 1 girl
## 2 girl
## 3 girl
```

Data frames

They can be edited with dpylr way

add column with `mutate()`

```
dataframe.ht <- dataframe.ht %>% mutate(linenumber = 1:nrow(dataframe.ht))
head(dataframe.ht, n= 3)
```

```
##   Height  Sex linenumber
## 1  171.3 girl          1
## 2  189.0 girl          2
## 3  174.0 girl          3
```

select columns with `select()`

```
dataframe.ht.sex <- dataframe.ht %>% select(Sex)
head(dataframe.ht.sex, n= 3)
```

```
##    Sex
## 1 girl
## 2 girl
## 3 girl
```

select rows with `filter()`

```
dataframe.ht.female <- dataframe.ht %>% filter(Sex == "girl")
head(dataframe.ht.female, n= 3)
```

```
##   Height  Sex linenumber
## 1  171.3 girl          1
## 2  189.0 girl          2
## 3  174.0 girl          3
```

Data frames cool stuff with dpylr

They can be edited with dpylr way
combining `summarise()` and `mutate()`

```
dataframe.ht <- dataframe.ht %>% mutate(linenumber = 1:nrow(dataframe.ht))
```

reshaping data frame

you have wide data:

```
head(my_df1)
```

```
##    age4  Sex linenumber age1  age2  age3
## 1 171.3 girl          1 71.3 146.3 161.3
## 2 189.0 girl          2 89.0 164.0 179.0
## 3 174.0 girl          3 74.0 149.0 164.0
## 4 177.0 girl          4 77.0 152.0 167.0
## 5 178.0 girl          5 78.0 153.0 168.0
## 6 175.0 girl          6 75.0 150.0 165.0
```

```
dim(my_df1)
```

```
## [1] 10  6
```

you want long data:

```
head(my_df2)
```

```
##    Sex  Age Height
## 1 girl age1   71.3
## 2 girl age1   89.0
## 3 girl age1   74.0
## 4 girl age1   77.0
## 5 girl age1   78.0
## 6 girl age1   75.0
```

```
dim(my_df2)
```

```
## [1] 50  3
```

reshaping data frame

you have wide data:

```
head(my_df1)
```

```
##    age4  Sex linenumber age1  age2  age3
## 1 171.3 girl          1 71.3 146.3 161.3
## 2 189.0 girl          2 89.0 164.0 179.0
## 3 174.0 girl          3 74.0 149.0 164.0
## 4 177.0 girl          4 77.0 152.0 167.0
## 5 178.0 girl          5 78.0 153.0 168.0
## 6 175.0 girl          6 75.0 150.0 165.0
```

```
dim(my_df1)
```

```
## [1] 10  6
```

you want long data:

```
head(my_df2)
```

```
##    Sex  Age Height
## 1 girl age1   71.3
## 2 girl age1   89.0
## 3 girl age1   74.0
## 4 girl age1   77.0
## 5 girl age1   78.0
## 6 girl age1   75.0
```

```
dim(my_df2)
```

```
## [1] 50  3
```

you do:

```
my_df2 <- my_df1 %>% gather("Age", "Height", -Sex) %>% arrange(Age)
```

reshaping data frame

you have wide data:

```
head(my_df1)
```

```
##    age4  Sex linenumber age1  age2  age3
## 1 171.3 girl          1 71.3 146.3 161.3
## 2 189.0 girl          2 89.0 164.0 179.0
## 3 174.0 girl          3 74.0 149.0 164.0
## 4 177.0 girl          4 77.0 152.0 167.0
## 5 178.0 girl          5 78.0 153.0 168.0
## 6 175.0 girl          6 75.0 150.0 165.0
```

```
dim(my_df1)
```

```
## [1] 10  6
```

you want long data:

```
head(my_df2)
```

```
##    Sex  Age Height
## 1 girl age1   71.3
## 2 girl age1   89.0
## 3 girl age1   74.0
## 4 girl age1   77.0
## 5 girl age1   78.0
## 6 girl age1   75.0
```

```
dim(my_df2)
```

```
## [1] 50  3
```

you do:

```
my_df2 <- my_df1 %>% gather("Age", "Height", -Sex) %>% arrange(Age)
```

one row = one observation, one column = one variable

joining data frame

you have df1:

```
my_df1
```

```
##        ID      age
## 1    ID-1 12.49418
## 2    ID-2 15.73457
## 3    ID-3 11.65749
## 4    ID-4 21.38112
## 5    ID-5 16.31803
## 6    ID-6 11.71813
## 11  ID-11 21.04712
```

you have df2:

```
my_df2
```

```
##       ID school grade origin
## 1   ID-4 Youhou 76.42 French
## 2   ID-5 bababa 71.88  Swiss
## 3   ID-1 genius 78.38 French
## 4  ID-12 Youhou 75.64 German
## 5   ID-7 bababa 61.49 German
## 6   ID-3 genius 20.21 French
## 7   ID-8 Youhou 72.40 German
## 8   ID-6 bababa 58.88 German
## 9   ID-2 genius 56.88  Swiss
## 10 ID-10 Youhou 30.58 French
```

You want to merge the two data frames

joining data frame with merge

You can use `merge()`

```
my_df3 <- merge(my_df1, my_df2)
```

```
my_df3
```

```
##     ID      age school grade origin
## 1 ID-1 12.49418 genius 78.38 French
## 2 ID-2 15.73457 genius 56.88  Swiss
## 3 ID-3 11.65749 genius 20.21 French
## 4 ID-4 21.38112 Youhou 76.42 French
## 5 ID-5 16.31803 bababa 71.88  Swiss
## 6 ID-6 11.71813 bababa 58.88 German
```

joining data frame with dpylr join

or use `inner_join()`

```r
library(dplyr)
my_df3 <- inner_join(my_df1, my_df2)
        ## Joining, by = "ID"
```

```r
my_df3
```

```
##      ID      age school grade origin
## 1 ID-1 12.49418 genius 78.38 French
## 2 ID-2 15.73457 genius 56.88  Swiss
## 3 ID-3 11.65749 genius 20.21 French
## 4 ID-4 21.38112 Youhou 76.42 French
## 5 ID-5 16.31803 bababa 71.88  Swiss
## 6 ID-6 11.71813 bababa 58.88 German
```

joining data frame with dpylr left_join

`left_join()` or `right_join()` keep all the rows of the data frame on the left (or right) adds NA when no data are present

```
library(dplyr)
my_df3 <- left_join(my_df1, my_df2)
          ## Joining, by = "ID"
```

```
my_df3
##      ID      age school grade origin
## 1  ID-1 12.49418 genius 78.38 French
## 2  ID-2 15.73457 genius 56.88  Swiss
## 3  ID-3 11.65749 genius 20.21 French
## 4  ID-4 21.38112 Youhou 76.42 French
## 5  ID-5 16.31803 bababa 71.88  Swiss
## 6  ID-6 11.71813 bababa 58.88 German
## 7 ID-11 21.04712   <NA>    NA   <NA>
```

joining data frame with dpylr full_join

`full_join()` keep all the rows of the two data frame

adds NA when no data are present

```
library(dplyr)
my_df3 <- full_join(my_df1, my_df2)
## Joining, by = "ID"
```

```
my_df3
```

```
##        ID      age school grade origin
## 1    ID-1 12.49418 genius 78.38 French
## 2    ID-2 15.73457 genius 56.88  Swiss
## 3    ID-3 11.65749 genius 20.21 French
## 4    ID-4 21.38112 Youhou 76.42 French
## 5    ID-5 16.31803 bababa 71.88  Swiss
## 6    ID-6 11.71813 bababa 58.88 German
## 7   ID-11 21.04712   <NA>    NA   <NA>
## 8   ID-12       NA Youhou 75.64 German
## 9    ID-7       NA bababa 61.49 German
## 10   ID-8       NA Youhou 72.40 German
## 11  ID-10       NA Youhou 30.58 French
```

extanding data frame

cheating data frame

plenty of informative cheatsheets on: `https://www.rstudio.com/resources/cheatsheets/`

Lists

Lists allow the organisation of any set of entities into a single R object:

```
list.ht <- list(girls = height.girls, boys = height.boys)
list.ht
```

```
## $girls
## [1] 178 175 159 164 183 192
##
## $boys
## [1] 181 189 174 177
```

Lists

Lists can also be indexed and their elements extracted:

```
list.ht$girls
```

```
## [1] 178 175 159 164 183 192
```

```
list.ht["boys"]  # still a list
```

```
## $boys
## [1] 181 189 174 177
```

```
list.ht[["boys"]]  # vector
```

```
## [1] 181 189 174 177
```

```
list.ht[[2]][3]
```

```
## [1] 174
```

Lists

Some functions can take a list as an input:

```
lapply(list.ht, FUN = mean)
```

```
## $girls
## [1] 175.1667
##
## $boys
## [1] 180.25
```

Summary

```
dataframe.ht
```

```
##    Height  Sex linenumber
## 1   171.3 girl          1
## 2   189.0 girl          2
## 3   174.0 girl          3
## 4   177.0 girl          4
## 5   178.0 girl          5
## 6   175.0 girl          6
## 7   159.0  boy          7
## 8   164.0  boy          8
## 9   183.0  boy          9
## 10  192.0  boy         10
```

```
list.ht
```

```
## $girls
## [1] 178 175 159 164 183 192
##
## $boys
## [1] 181 189 174 177
```

Summary

- `data.frame`
  - All columns have same length
  - Each column can have its own class (e.g. `numeric`, `factor`, `character`)
- `list`
  - Each element can have its own length
  - Each element can have its own class (e.g. `numeric`, `factor`, `character`)