Getting to use data in **R**

Alexandre Courtiol & Colin Vullioud

Leibniz Institute of Zoo and Wildlife Research

June 2018

Getting started with **R**

Handling data in **R**

There are many types of objects designed to store data in **R**.

We will focus on:

- vectors
- matrices (and arrays)
- data frames (and tibbles)
- list

Note: if you master those, we are pretty much all set!

Handling data in **R**

- vector
  - A single row of data
  - All elements have the same type (e.g. `numeric`, `factor`, `character`...)
- `matrix` & `array`
  - All rows & columns have same length
  - All rows & columns have the same type (`numeric` or `character`)
- `data.frame` & `tibble`
  - All rows & columns have same length
  - Each column can have its own type (`numeric`, `factor`, `character`...)
- `list`
  - Each element can have its own length
  - Each element can have its own type (e.g. `numeric`, `factor`, `character`...)

Getting started with **R**

Vector

The vector is the simplest way to store data in **R**; it is a sequence of data elements of the same kind.

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

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

```
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
```

Types of vectors

- character

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

```
##  chr [1:4] "bla" "1" "sf" "xx3"
```

- 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)
str(x)
```

```
##  logi [1:4] TRUE FALSE TRUE FALSE
```

- 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"
```

Getting started with **R**

Factors

- They work with other things than numbers:

```
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
```

```
# 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
```

Getting started with **R**

Getting started with **R**

Getting started with **R**

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
```

Getting started with **R**

dplyr

- `dplyr` is a useful package for data manipulation

dplyr

- `dplyr` is a useful package for data manipulation

- `dplyr` is a grammar of data manipulation: one verb = one operation

dplyr

- `dplyr` is a useful package for data manipulation
- `dplyr` is a grammar of data manipulation: one verb = one operation
- operations can be chained with the pipe operator %>%

dplyr

- `dplyr` is a useful package for data manipulation
- `dplyr` is a grammar of data manipulation: one verb = one operation
- operations can be chained with the pipe operator %>%
- the pipe operator %>% takes the output from one function as input of another function.

Data frames

Useful `dplyr` verbs

add column with `mutate()`

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

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

Data frames

Useful `dplyr` verbs

add column with `mutate()`

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

```
##   Height  Sex ID
## 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

Useful dplyr verbs

add column with mutate()

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

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

mutate_if()

you want to change all numeric variables into character variables

you have:

```
## 'data.frame': 10 obs. of  6 variables:
##  $ Height  : num  171 189 174 177 178 ...
##  $ Sex     : chr  "girl" "girl" "girl" "girl" ...
##  $ ID      : int  1 2 3 4 5 6 7 8 9 10
##  $ mean_H  : num  177 177 177 177 177 ...
##  $ median_H: num  176 176 176 176 176 ...
##  $ n       : int  6 6 6 6 6 6 4 4 4 4
```

you want

```
## 'data.frame': 10 obs. of  6 variables:
##  $ Height  : chr  "171.3" "189" "174" "177" ...
##  $ Sex     : chr  "girl" "girl" "girl" "girl" ...
##  $ ID      : chr  "1" "2" "3" "4" ...
##  $ mean_H  : chr  "177.38" "177.38" "177.38" "177.38" ...
##  $ median_H: chr  "176" "176" "176" "176" ...
##  $ n       : chr  "6" "6" "6" "6" ...
```

you do:

```
x_numeric <- x %>% mutate_if(is.numeric, ~ as.character(.))
```

group_by()

- `group_by()` allow you to perform operation on gouped data.

group_by()

- `group_by()` allow you to perform operation on gouped data.

- it is mostly used with `summarize()` -> one value per group

group_by()

- `group_by()` allow you to perform operation on gouped data.
- it is mostly used with `summarize()` -> one value per group
- or with `mutate()` -> one value per observation

group_by() with summarize()

You want the mean height of males and females, the median height and the number in each group:

you do:

```
x <- dataframe.ht %>%
  group_by(Sex) %>%
  summarize(mean_H = mean(Height, na.rm = T),
            median_H = median(Height, na.rm = T),
            n = n())
```

you get:

```
as.data.frame(x)
```

```
##    Sex   mean_H median_H n
## 1  boy 174.5000    173.5 4
## 2 girl 177.3833    176.0 6
```

group_by() with mutate()

You want the mean height of males and females, the median height and the number in each group but get the value for each individual

you do:

```
x <- dataframe.ht %>%
  group_by(Sex) %>%
  mutate(mean_H = mean(Height, na.rm = T),
           median_H = median(Height, na.rm = T),
           n = n())
```

you get:

```
as.data.frame(x)
```

```
##    Height  Sex ID   mean_H median_H n
## 1   171.3 girl  1 177.3833    176.0 6
## 2   189.0 girl  2 177.3833    176.0 6
## 3   174.0 girl  3 177.3833    176.0 6
## 4   177.0 girl  4 177.3833    176.0 6
## 5   178.0 girl  5 177.3833    176.0 6
## 6   175.0 girl  6 177.3833    176.0 6
## 7   159.0  boy  7 174.5000    173.5 4
## 8   164.0  boy  8 174.5000    173.5 4
## 9   183.0  boy  9 174.5000    173.5 4
## 10  192.0  boy 10 174.5000    173.5 4
```

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 base R

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()`

```
library(dplyr)
my_df3 <- inner_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
```

joining data frame with 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 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
```

Getting started with **R**

reshaping data frame

- one row = one observation, one column = one variable

reshaping data frame

- one row = one observation, one column = one variable

- `gather()` turns wide data into long

reshaping data frame

- one row = one observation, one column = one variable
- `gather()` turns wide data into long
- `spread()` turns long data into wide

reshaping data frame

you have wide data:

```
head(my_df1)
```

```
##   ID  Sex age1  age2  age3  age4
## 1  1 girl 71.3 146.3 161.3 171.3
```

```
dim(my_df1)
```

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

you want long data:

```
head(my_df2)
```

```
##   ID  Sex  Age Height
## 1  1 girl age1   71.3
## 2  1 girl age2  146.3
## 3  1 girl age3  161.3
## 4  1 girl age4  171.3
```

```
dim(my_df2)
```

```
## [1] 4 4
```

reshaping data frame

you have wide data:

```
head(my_df1)
```

```
##   ID  Sex age1  age2  age3  age4
## 1  1 girl 71.3 146.3 161.3 171.3
```

```
dim(my_df1)
```

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

you want long data:

```
head(my_df2)
```

```
##   ID  Sex  Age Height
## 1  1 girl age1   71.3
## 2  1 girl age2  146.3
## 3  1 girl age3  161.3
## 4  1 girl age4  171.3
```

```
dim(my_df2)
```

```
## [1] 4 4
```

you do:

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

reshaping data frame

you have wide data:

```
head(my_df1)
```

```
##   ID  Sex age1  age2  age3  age4
## 1  1 girl 71.3 146.3 161.3 171.3
```

```
dim(my_df1)
```

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

you want long data:

```
head(my_df2)
```

```
##   ID  Sex  Age Height
## 1  1 girl age1   71.3
## 2  1 girl age2  146.3
## 3  1 girl age3  161.3
## 4  1 girl age4  171.3
```

```
dim(my_df2)
```

```
## [1] 4 4
```

you do:

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

or:

```
my_df2 <- my_df1 %>% gather("Age", "Height", 3:ncol(my_df1)) %>% arrange(ID, Age)
```

reshaping data frame

The reverse is done with `spread()`

you have wide data:

```
head(my_df2)
```

```
##   ID  Sex  Age Height
## 1  1 girl age1   71.3
## 2  1 girl age2  146.3
## 3  1 girl age3  161.3
## 4  1 girl age4  171.3
```

```
dim(my_df2)
```

```
## [1] 4 4
```

you want long data:

```
head(my_df1)
```

```
##   ID  Sex age1  age2  age3  age4
## 1  1 girl 71.3 146.3 161.3 171.3
```

```
dim(my_df1)
```

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

reshaping data frame

The reverse is done with `spread()`

you have wide data:

```
head(my_df2)
```

```
##   ID  Sex  Age Height
## 1  1 girl age1   71.3
## 2  1 girl age2  146.3
## 3  1 girl age3  161.3
## 4  1 girl age4  171.3
```

```
dim(my_df2)
```

```
## [1] 4 4
```

you want long data:

```
head(my_df1)
```

```
##   ID  Sex age1  age2  age3  age4
## 1  1 girl 71.3 146.3 161.3 171.3
```

```
dim(my_df1)
```

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

you do:

```
my_df2 %>% spread(-Sex, -ID)
```

```
##   ID  Sex age1  age2  age3  age4
## 1  1 girl 71.3 146.3 161.3 171.3
```

some other useful functions

`unite()` merges 2 columns of a data frame

```
my_df3 <- my_df2 %>% unite(New_col, ID, Sex)
head(my_df3)
```

```
##   New_col  Age Height
## 1  1_girl age1   71.3
## 2  1_girl age2  146.3
## 3  1_girl age3  161.3
## 4  1_girl age4  171.3
```

some other useful functions

`unite()` merges 2 columns of a data frame

```
my_df3 <- my_df2 %>% unite(New_col, ID, Sex)
head(my_df3)
```

```
##   New_col  Age Height
## 1  1_girl age1   71.3
## 2  1_girl age2  146.3
## 3  1_girl age3  161.3
## 4  1_girl age4  171.3
```

`separate()` separate 2 columns of a data frame

```
my_df3 %>% separate(New_col, c("ID", "Sex"))
```

```
##   ID  Sex  Age Height
## 1  1 girl age1   71.3
## 2  1 girl age2  146.3
## 3  1 girl age3  161.3
## 4  1 girl age4  171.3
```

cheating data frame

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

Getting started with **R**

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 ID
## 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`)

Getting started with **R**

Working directory

```
getwd()  # to change, use setwd()
```

```
## [1] "/Users/alex/Dropbox/Boulot/Mes_projets_de_recherche/R_packages/BeginR_project/BeginR/sources_vignettes/usingdata"
```

```
dir() # listing all files in the working directory
```

```
## [1] "usingdata.nav"      "usingdata.pdf"
## [3] "usingdata.pdf.asis" "usingdata.Rnw"
## [5] "usingdata.snm"      "usingdata.tex"
## [7] "usingdata.toc"      "usingdata.vrb"
```

```
dir(pattern = "*.csv")
```

```
## character(0)
```

Exporting and importing data in R

```r
write.csv(dataframe.ht,
    file = "my.first.R.dataframe.csv", row.names = FALSE)

rm(list = ls())  # deleting everything in R

dataframe.ht <- read.csv("my.first.R.dataframe.csv")
```

R cannot read/write .xls files out of the box
Packages can do that but it is safer to use .csv files
Excel can read and write .csv files!

Challenge #2

Create a dataframe using your favorite spreadsheet software
and import it in R!