Automating web scraping with R

Alex Sanchez and Francesc Carmona
Genetics Microbiology and Statistics Department
Universitat de Barcelona
October 2022

Outline

Introduction

Introduction

- We have introduced R as a *a language (a tool), to manage and analyze data*.
- It is also a *programming language*
  - It is simple and versatile
  - The user can create new functions that adapt to their needs
  - It is widely used (2nd most widely used in Data Science)
  - Users provide the community with a high variety of solutions ("packages")
  - As a programming language it is not, however, very efficient

Example 1: Why we need programming

- Assume we have the diabetes dataset and want to make a summary of every variable it contains.
- Some variables are "naturally" categorical but the system cannot recognnize them, so we need to transform them.

```
library(readr)
library(dplyr)
library(janitor)
MunicipisBARC ← read_csv("datasets/MunicipisBARC.csv")
MunicipisBreu ← MunicipisBARC %>% select (1,2,9,10,27,28) %>%
  janitor::clean_names()
summary(MunicipisBreu)
```

Example 1: Why we need programming

```
## Rows: 311 Columns: 30
## ── Column specification ────────────────────────────────────────────────────
## Delimiter: ","
## chr  (23): _id, Codi INE, Nom del municipi, Nom curt del municipi, Article d…
## dbl   (6): Codi de comarca, Codi de la província, Fax, Nombre d'habitants, E…
## dttm  (1): _lastChange
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(janitor)
```

```
##
## Attaching package: 'janitor'

## The following objects are masked from 'package:stats':
```

Example 1: Why we need programming

- A simple solution: Convert text variables into factors.

```
library(forcats)
MunicipisBreu$codi_de_comarca ← as_factor(MunicipisBreu$codi_de_comarca)
MunicipisBreu$nom_de_la_comarca ← as_factor(MunicipisBreu$nom_de_la_comarca)

summary(MunicipisBreu)
```

```
##       id              codi_ine         codi_de_comarca       nom_de_la_comarca
##  Length:311         Length:311         24     : 47     Osona           : 47
##  Class :character   Class :character   41     : 39     Vallès Oriental: 39
##  Mode  :character   Mode  :character   6      : 33     Anoia           : 33
##                                        7      : 30     Baix Llobregat : 30
##                                        11     : 30     Bages           : 30
##                                        14     : 30     Maresme         : 30
##                                        (Other):102     (Other)         :102
##  nombre_dhabitants      extensio
##  Min.   :     27.0   Min.   :  0.40
##  1st Qu.:    734.5   1st Qu.: 11.01
##  Median :   3219.0   Median : 21.17
##  Mean   :  18375.3   Mean   : 24.90
##  3rd Qu.:  11010.5   3rd Qu.: 34.12
##  Max.   :1636732.0   Max.   :102.90
##
```

- But how shoulde we proceed if there were dozens or hundreds of variables that need to be changed?
- What if, besides, we were required to do ths repeatedly, on distinct files every day?
- One solution may consists of:
  - providing some way to encapsulate all steps needed to do the transformation
  - in such a way that they can be easily applied to a file everytime they are required.

User defined functions

Functions are named expressions

- A function is a set of statements organized together to perform a specific task.
- R has a large number of in-built functions.
- Users can create their own functions, for those situations where they wish to apply the same set of instructions more than once.

```
function_name ← function(arg_1, arg_2, ... ) {
   sentence 1
    ...
   sentence n
   return(result)
}
```

Go here for more information on functions.

A preprocessing function

We can encapsulate preprocessing in a function

```
preprocessa ← function(unMunicipi){
  unMunicipiBreu ← unMunicipi %>% select (1,2,9,10,27,28) %>%
  janitor::clean_names()
  unMunicipiBreu$codi_de_comarca ← as_factor(unMunicipiBreu$codi_de_comarca)
  unMunicipiBreu$nom_de_la_comarca ← as_factor(unMunicipiBreu$nom_de_la_coma
  return(unMunicipiBreu)
}
```

And use it whenever is required

```
BCNBreu ← preprocessa(MunicipisBARC)
GiroBreu ← preprocessa(MunicipisGIRO)
LleidaBreu ← preprocessa(MunicipisLleida)
```

Scraping a recipes site

- Imagine we are scraping a recipes site
- The code below extracts (without cleaning it) a recipe for brownies.

```
library(rvest)
```

```
##
## Attaching package: 'rvest'

## The following object is masked from 'package:readr':
##
##     guess_encoding
```

```
brownies ← read_html("https://www.allrecipes.com/recipe/25080/mmmmm-brownies/")

ingredients ← brownies %>%
  html_elements( ".mntl-structured-ingredients__list") %>%
  html_text2() %>% stringr::str_split("\\n")

xpath4Directions ← '//*[(@id = "mntl-sc-block_2-0-1")] | //*[(@id = "mntl-sc-block_2-0-9")] | //*[(@id

directions ←  brownies %>%
  html_elements( xpath=xpath4Directions) %>%
  html_text2()
```

The scraped recipe

```
show(ingredients)
```

```
## [[1]]
##  [1] "½ cup white sugar"                    ""
##  [3] "2 tablespoons butter"                 ""
##  [5] "2 tablespoons water"                  ""
##  [7] "1 ½ cups semisweet chocolate chips" ""
##  [9] "2 large eggs, beaten"                 ""
## [11] "½ teaspoon vanilla extract"           ""
## [13] "⅔ cup all-purpose flour"              ""
## [15] "½ teaspoon salt"                      ""
## [17] "¼ teaspoon baking soda"
```

```
show(directions)
```

```
## [1] "½ cup white sugar\n\n2 tablespoons butter\n\n2 t
## [2] "Directions"
## [3] "Preheat the oven to 325 degrees F (165 degrees C
## [4] "Preheat the oven to 325 degrees F (165 degrees C
## [5] "Combine sugar, butter, and water in a medium sau
## [6] "Bake in the preheated oven until top is dry and
```

A function to scrape brownies

```
scrape_recipes ← function(URL) {
  aDessert ← read_html(URL)
  ingredients ← aDessert %>%
    html_elements( ".mntl-structured-ingredients__list") %>%
    html_text2() %>% stringr::str_split("\\n")

  xpath4Directions ← '//*[(@id = "mntl-sc-block_2-0-1")] | //*[(@id = "mntl-sc-block_2-0-9")] | //*[(@

  directions ←  aDessert %>%
    html_elements( xpath=xpath4Directions) %>%
    html_text2()
  return(list(Ingredientes=ingredients, Recepta=directions))
}
```

The scraped recipe (2)

```
library(rvest)
recipeURL ← "https://www.allrecipes.com/recipe/25080/mmmmm-brownies/"
brownies ← scrape_recipes (recipeURL)
```

```
show(brownies[["Ingredientes"]])
```

```
## [[1]]
##  [1] "½ cup white sugar"                    ""
##  [3] "2 tablespoons butter"                 ""
##  [5] "2 tablespoons water"                  ""
##  [7] "1 ½ cups semisweet chocolate chips" ""
##  [9] "2 large eggs, beaten"                 ""
## [11] "½ teaspoon vanilla extract"           ""
## [13] "⅔ cup all-purpose flour"              ""
## [15] "½ teaspoon salt"                      ""
## [17] "¼ teaspoon baking soda"
```

```
show(show(brownies[["Recepta"]]))
```

```
## [1] "½ cup white sugar\n\n2 tablespoons butter\n\n2 t
## [2] "Directions"
## [3] "Preheat the oven to 325 degrees F (165 degrees C
## [4] "Preheat the oven to 325 degrees F (165 degrees C
## [5] "Combine sugar, butter, and water in a medium sau
## [6] "Bake in the preheated oven until top is dry and
## NULL
```

Changing the flow

Changing the flow of execution

Scripts are executed *lineally*

- R, as most ordinary programming languages, is executed lineally, that is from the first to last line.
- Sometimes this needs to be changed.
  - Taking alternative flows according to certain conditions
  - Repeating some instructions while certain condition holds, or a fixed number of times,...
- This can be acomplished using *Flow Control Structures*

Loop controlled by a counter: `for`

- Loops are used in programming to repeat a specific block of code made by one or more instructions.
- Syntax of `for` loops:

```
for (val in sequence)
{
statement
}
```

- `sequence` is a vector and `val` takes on *each of its values* during the loop.
- In each iteration, `statement` is evaluated.

Example of `for` loop

- A `for` loop can be used to preprocess a list of selected files
- Assume we have the list of four files to be processed, and ***we know they have the same structure***.
- To process them all in one step do (not run):

```
llistaMunicipis ← c("MunicipisBAR.csv", "MunicipisGIR.csv",
                    "MunicipisLLE.csv", "MunicipisTAR.csv" )
for (nomFitxerMunicipis in llistaMunicipis) {
  municipisProvincia ← read_csv("nomFitxerMunicipis")
  preprocessa(municipisProvincia)
  summary(municipisProvincia)
}
```

Exercise

- Create a `for` loop that reads all .csv filenames in your datasets directory (or the directory you decide) and prints the name of the file and the column names in the screen.

Scraping multiple recipes

- Imagine we want to process not one but many desserts' recipes from the web "https://www.allrecipes.com/".
- THis can be done using a simple for loop:

```
recipe_urls ← c("https://www.allrecipes.com/recipe/25080/mmmmm-brownies/",
                "https://www.allrecipes.com/recipe/27188/crepes/",
                "https://www.allrecipes.com/recipe/22180/waffles-i/")
listOfRecipes ← list()
for (i in 1:length(recipe_urls)) {
  listOfRecipes[i] ← scrape_recipes(recipe_urls[i])
}
```

Exercise

- Write a simple function to print one recipes obtained using the function `scrape_recipes`
- Use this function to print all the recipes collected in the list "listOfRecipes"

Conditional statements: `if - else`.

- Conditional statements allow different blocks to be executed whether a certain condition is TRUE or FALSE.

```
if (test_expression) {
    statement
  }
```

- If the `test_expression` is TRUE, the `statement` gets executed. But if it's FALSE, nothing happens.
- Here, `test_expression` can be a logical or numeric vector, but only the first element is taken into consideration.
- In numeric vectors, zero is taken as FALSE, rest as TRUE.

Conditional statements: `if - else`.

```
if (test_expression) {
    statement_1
}else{
    statement_2
  }
```

- If the `test_expression` is TRUE, then `statement_1` gets executed.
- If it's FALSE then `statement_2` gets executed.