Exercise 2 - Data Curation

Christopher Diebel & Jan-Hendrik Schmidt - ISE Darmstadt

Note: Thanks are also due here to Gregor Albrecht and Jakob Lorz, whose exercise session from the summer semester of 2021 also forms the basis of this semester's exercise.

- :house_with_garden: Home
- :open_book: `dplyr` Documentation
- :open_book: `readr` Documentation
- :open_book: `lubridate` Documentation
- :open_book: `rmarkdown` Cheatsheet
- :open_book: Further Reading R for Data Science, R für Einsteiger or Einführung in R

Preparation

1. Install the `weights` package by running `install.packages('weights')`
2. Install the `tidyfst` package by running `install.packages('tidyfst')`
3. Install the `rmarkdown` package by running `install.packages('rmarkdown')`

Refresher: Star Wars Tasks

1. Find all characters with yellow eyes.
2. Remove all Gungans. How many characters are left?
3. What is the average mass of all droids?
4. Calculate the BMI for all humans (‘mass / ((height / 100) ^ 2)‘)
5. Which character has the longest name?
6. What is the earliest birth year for each species? (‘birth_year‘ is measured in BBY = Before Battle of Yavin, so high values = earlier)
7. What are the most populated planets?
8. What is the average height of all female humans?
9. Which planet has the most droids?
10. Who is the oldest character of each species?
11. Which is the most prevalent eye color on each planet?
12. How many unique eye colors are there?

How to Import Data in R

There are a lot of packages in the `tidyverse` for importing data, but you should mostly to only care about `readr`† and `readxl`‡.

- `read_csv()`† for comma-separated values (csv) files
- `read_csv2()`† for csv files that use semicolons as delimiters
- `read_tsv()`† for csv files that use tabs as delimiters
- `read_delim()`† for csv files that use anything else as delimiters
- `read_excel()`‡ for Excel files

As you can see, there are a lot of different functions with very specific use cases.

Important: There are also very similar functions from other (partly the basic) packages, which are called e.g. `read.csv()`. Pay attention to the correct notation/source package accordingly. That means: Remember to load (and install if necessary) the necessary package to be able to use the functions.

To read a file, it's easiest if the file is in your current working directory (CWD). To find out what your CWD is, use `getwd()`.

```
getwd()
```

Necessary CSV File

Download the CSV file friends.csv from GitHub to your private computer. To get the file, you can either download everything as `.zip` or create a new empty file and copy&paste the raw file content (for those of you who know what that is, you can also, of course, clone the repository).

- **Option 1**: Download as `.zip`: Go to the `wellbeing` repository, click on the green `Code` button and click on "Download ZIP". Then put it into your CWD. Also see: "How to download as ZIP?"
- **Option 2**: Create an empty file and copy&paste: Open RStudio, type `write.csv('hello there', 'friends.csv')` and press `Enter`. You should now see a new file in RStudio (Section "files") that you can open (Select the file and press "View File") and replace the content with the raw content of the file in the repository. Don't forget to save the changes.
- **Option 3**: Use `getwd()` to read your current working directory and copy it to your clipboard. Then open the raw content of the file in the repository, use the `right mouse button` and click "`Save as`". Save the csv file (pay attention to the file type) in your working directory.

After the file itself is created, we should also save it to a variable.

In the so created tibble, we can see that three observations in the "vaccine" column have no entry (`NA`). For our first evaluation, we are **not interested in which vaccine** a character received, but **only whether he received one or not**. For this we can use dummy variables.

Dummy Variables

To make our lives easier, it is possible in R to define our own functions and then use them. This is very simple:

```
ourFunction <- function() {

}
```

Let's go through these lines step by step. We see that we define an object name, `ourFunction`, and this object is now assigned something, `<-`. With the keyword `function` we say that `ourFunction` should be a function. In our example, the function has no parameters, so there is nothing in the braces, `()`. The curly braces form the `function body`, which now contains all the statements that belong to the function. Everything after the closing brace no longer belongs to the function.†

† Source

But now we want the function to do something specific, because it is still empty and nothing is executed. Let's build a function that calculates a simple mathematical function:

```
ourFunction <- function(x) {
  result <- 4*x + 5
  return(result)
}

ourFunction(9)
```

```
## [1] 41
```

For our use case, we first need our own defined function `has_been_vaccinated()` that returns `1` if the passed parameter is defined and `0` if not. We can declare our own function as follows:

```
has_been_vaccinated <- function(vaccine) {
  is_vaccinated = ifelse(is.na(vaccine), 0, 1)
  return(is_vaccinated)
}
```

Now, we want to use this function to dummify the column `vaccine` into a new column `is_vaccinated`. `1` means "vaccinated" and `0` means "not vaccinated". It **doesn't matter which** vaccine the character has, **only if** they have one.

```
friends %>%
  mutate(is_vaccinated = has_been_vaccinated(vaccine))
```

```
## # A tibble: 11 x 3
##    name      vaccine     is_vaccinated
##    <chr>     <chr>               <dbl>
##  1 Luke      AstraZeneca             1
##  2 Leia      Moderna                 1
##  3 Han       BioNTech                1
##  4 Rey       <NA>                    0
##  5 Chewbacca J&J                     1
##  6 C-3PO     Microsoft               1
##  7 R2-D2     Microsoft               1
##  8 Obi-Wan   BioNTech                1
##  9 JarJar    <NA>                    0
## 10 Boba      Sputnik V               1
## 11 Bossk     <NA>                    0
```

Now, we do want to know which vaccine each character received. In theory, we could do this by checking for each possibility separately:

But, as you can see, this is quite tedious. Luckily, there is the function `dummify()` from the package `weights`. So load the package with `library(weights)` (install if necessary with `install.packages('weights')`and then let's try to use `dummify()`:

```
library(weights)
dummify(friends$vaccine)
```

This will show you an error: `variable needs to be a factor`. But what are factors?

> R uses factors to represent categorical variables that have a known set of possible values.†

† Source

So how do we create factors? In this case, it's simple. First, we remove all `NA` values:

```
friends <- friends %>%
  mutate(vaccine = replace_na(vaccine, 'NONE'))
```

Then, we make the `vaccine` column a factor:

```
friends_factorized <- friends %>%
  mutate(vaccine = as.factor(vaccine))
```

Practice: Breakout Session

1. Install the `tidyfst` package: `install.packages("tidyfst")`. Afterwards, load it with `library(tidyfst)`.
2. Download the `starwars_dd.csv` file from GitHub. For ease of use: place it in the folder which is shown in the `Files` section of RStudio.
3. Import `starwars_dd.csv` into a variable named `starwars_dd`. The contents are different from the tidyverse-internal `starwars` dataset and simplify the next steps for you.
4. Import the `tidyfst` package. Then generate dummies for all films and store them in the variable `starwars_films_dd`. Inspect them in RStudio (using the `View` function) afterwards. You may use the :book: `dummy_dt` function from `tidyfst`. Make sure to exclude rows which hold no additional information, we are focusing on the different films here (Hint: Each character has multiple rows for each film, vehicle and starship)
5. How many characters starred in `A New Hope`?
6. Which of those characters are female?

Group Session: Working with Dates

Because working with dates can be cumbersome, the `tidyverse` contains a very helpful package for that: `lubridate`.

```
library(lubridate)
```

```
##
## Attache Paket: 'lubridate'
```

```
## Die folgenden Objekte sind maskiert von 'package:base':
##
##     date, intersect, setdiff, union
```

If the date you are trying to work with contains a date like `4/14/22`, you can tell `lubridate` that this is a date:

```
date <- parse_date_time('4/14/22', 'mdy')
```

The `'mdy'` represents `[m]onth [d]ay [year]`. The resulting `date` is a

"POSIXct date-time object" (you can verify that with `is.instant(date)` or `class(date)`) that makes it very easy to work with it. For instance, you can find out which week the date refers to:

```
week(date)
```

```
## [1] 15
```

We can also do some calculations with timestamps:

```
start <- now()

# wait...
Sys.sleep(3.14159265359)
end <- now()

elapsed <- start %--% end
# ... and find out how long you waited
as.duration(elapsed)
```

```
## [1] "3.23255109786987s"
```