Data Handling: Import, Cleaning and Visualisation

Exercise to lecture 4: csv and arrays

Dr. Aurélien Sallin

Working with a data frame

Set your script

Set your `R` Script.

```
#######################################################################
# Data Handling Course: Example Script for Data Gathering and Import
#
# Imports data from ...
# Input: import c to data sources (data comes in ... format)
# Output: cleaned data as CSV
#
# A. Sallin, St. Gallen, 2023
#######################################################################

# SET UP --------------
# load packages
library(readr)


# SET PATH ------------------
# If not in NUVOLOS, set correct path!
# financial_data <- read.csv("Path/to/my/file/financial_data.txt")
```

Import data

Have a look at the file `financial_data.txt` using your favorite text editor. What do you notice?

Import the table using the `read.csv()` function in your environment. Make sure you have the right path to access the .txt document. What does this parser do? Explore the data.frame. What is its structure? What are its dimensions?

```
# IMPORT RAW DATA FROM CSVs -------------

# From simply opening the .txt data from the "Files" Panel on the left side
# in RStudio, we can see that the file has a structure with ":" separated
# value, has 5 columns and should have 30 rows.

# However, when importing the file with "read_csv" or "read.csv", we notice
# that the output is not correct: R does not understand that the value separator
# is a ":" and not a comma. For this reason, the file is not correctly read.
financial_data <- read.csv("financial_data.txt")
# financial_data <- read_csv("financial_data.txt")

# To indicate that the value separator is actually a ":", we need to tell R
# to use ":" as a separator.  This can be done using the option "sep = ":"" with
# read.csv().
# With read_csv() and the readr package, we need to use the function read_delim()
# instead.
financial_data <- read.csv("financial_data.txt", sep = ":")
# financial_data <- read_delim("financial_data.txt", delim = ":")
# When using read.csv(), we notice the following warning:
# "Error in type.convert.default(data[[i]], as.is = as.is[i], dec = dec,  :
# invalid multibyte string at '<f6>'"
# This means there is an encoding problem in the file. A visual exploration shows
# us that the encoding problem is in row 10, col 3.

# We have the character \xF6. This could be any encoding. In this case, I am guessing
# the encoding by trying different things. The function "string::stri_enc_detect()", upon
# which "guess_encoding()" is built, does not help me much.
# I did a Google search and realized it is likely to be Latin-1, which is "ISO-8859-1" and
# ISO-8859- family
stringi::stri_enc_detect("\xF6")
```

```
## [[1]]
##   Encoding Language Confidence
## 1    UTF-8                0.15
```

```
iconv("\xF6", from = "ISO-8859-1", to = "UTF-8")
```

```
## [1] "ö"
```

```
iconv("\xF6", from = "ISO-8859-2", to = "UTF-8")
```

```
## [1] "ö"
```

```
iconv("\xF6", from = "ISO-8859-2", to = "UTF-8")
```

```
## [1] "ö"
```

```
# This seems to be OK. Therefore, I can specify Latin-1 in my encoding.
financial_data <- read.csv("financial_data.txt",
                           sep = ":",
                           fileEncoding = "ISO-8859-1")
```

```
# The data looks now like what I expect... except for the variable "Revenue", which
# is a character. I remove the special ö character from the encoding issue and
# coerce
head(financial_data, 10)
```

```
##     Firm Year Revenue    Profit Category
## 1  FirmA 2017    4355  897.4552  Finance
## 2  FirmB 2017    4919 1091.3730   Health
## 3  FirmC 2017    4065 1231.3810     Tech
## 4  FirmD 2017    4989  860.2956     Tech
## 5  FirmE 2017    4172 1684.9384     Tech
## 6  FirmA 2018    2003  361.5399     Tech
## 7  FirmB 2018    1622  330.1158   Health
## 8  FirmC 2018    3952 1963.5914     Tech
## 9  FirmD 2018    3692 1561.4979  Finance
## 10 FirmE 2018   1933ö  621.1375  Finance
```

```
str(financial_data)
```

```
## 'data.frame':    30 obs. of  5 variables:
##  $ Firm    : chr  "FirmA" "FirmB" "FirmC" "FirmD" ...
##  $ Year    : int  2017 2017 2017 2017 2017 2018 2018 2018 2018 2018 ...
##  $ Revenue : chr  "4355" "4919" "4065" "4989" ...
##  $ Profit  : num  897 1091 1231 860 1685 ...
##  $ Category: chr  "Finance" "Health" "Tech" "Tech" ...
```

```
financial_data[10, 3] <- 1933

# Coerce to numeric
financial_data$Revenue <- as.numeric(financial_data$Revenue)

# Another way of writing the column selection
financial_data[10, "Revenue"]
```

```
## [1] 1933
```

```
financial_data[10, "Revenue"] <- 1933

financial_data[, "Revenue"] <- as.numeric(financial_data[, "Revenue"])

# The data is now ready. You are ready to compute the rest of the exercise.
# END - for now
```

Summary statistics of your data

Compute the summary statistics for each variable using the `summary()` command. What does this command give you? What do you notice? Make the necessary changes.

```
# Check summary again
summary(financial_data)
```

```
##      Firm                Year         Revenue         Profit      
##  Length:30          Min.   :2017   Min.   :1269   Min.   : 164.4  
##  Class :character   1st Qu.:2018   1st Qu.:2332   1st Qu.: 665.5  
##  Mode  :character   Median :2020   Median :3610   Median : 990.4  
##                     Mean   :2020   Mean   :3317   Mean   :1001.0  
##                     3rd Qu.:2021   3rd Qu.:4048   3rd Qu.:1245.4  
##                     Max.   :2022   Max.   :4989   Max.   :1963.6  
##    Category        
##  Length:30         
##  Class :character  
##  Mode  :character  
##                    
##                    
##                    
```

Variable creation

Create a new variable “costs”, which is the revenue - profit. [There are many ways to create a variable in a data frame. Here, use the `$` index.]

```
financial_data$costs <- financial_data$Revenue - financial_data$Profit
```

Factor variable

Which variable is (should be) a factor? Recode this variable as a factor. What are the levels? Should we have the variable `Firm` as a factor?

```
financial_data$Category <- as.factor(financial_data$Category)

levels(financial_data$Category)
```

```
## [1] "Finance" "Health"  "Tech"
```

Nests - more difficult question… but still exam relevant 👻

Split your data using the factor variable into three data frames that are contained in a list. Compute the mean profit for each data frame.

- Hint: use the function `split` .
- Hint: use a `for-loop` over each list element to compute the mean

```
list_financial_data <- split(financial_data, financial_data$Category)

for (i in 1:length(list_financial_data)){
  print(mean(list_financial_data[[i]]$Profit))
}
```

```
## [1] 982.0813
## [1] 877.2573
## [1] 1143.627
```

```
# Or, using lapply (not exam relevant)
lapply(list_financial_data, function(x) mean(x$Profit))
```

```
## $Finance
## [1] 982.0813
##
## $Health
## [1] 877.2573
##
## $Tech
## [1] 1143.627
```

Advanced: map (not exam relevant)

Do the same as the exercise above using the `map` function. Install the packages `tidyr` , `dplyr` , and `purrr` .

```
# Or (advanced!) with a nested tibble and map
library(tidyr)
library(dplyr)
library(purrr)

tibble_financial_data <- financial_data |>
  group_by(Category) |>
  nest()

map(tibble_financial_data$data, ~mean(.$Profit))
```