Let’s see how you could use the {tidyxl} package to go from a human readable Excel Workbook to a tidy  
data set (or flat file, as they are also called). Some people then contributed their solutions,  
which is always something I really enjoy when it happens. This way, I also get to learn things! Let’s see below:

You can find the data I will use [here](http://statistiques.public.lu/stat/ReportFolders/ReportFolder.aspx?IF_Language=eng&MainTheme=3&FldrName=1&RFPath=14306). Click on the “Time use” folder and you can download the workbook.

The Excel workbook contains several sheets (in French and English) of the amount of time Luxembourguish citizens spend from Monday to Sunday. For example, on average, people that are in employment spend almost 8 hours sleeping during the week days, and 8:45 hours on Saturday.

As you can see from the screenshot, each sheet contains several tables that have lots of headers and these tables are next to one another. Trying to import these sheets with good ol’ readxl::read\_excel() produces a monster.

This is where {tidyxl} comes into play. Let’s import the workbook with {tidyxl}:

library(tidyverse)

library(tidyxl)

time\_use\_xl <- xlsx\_cells("time-use.xlsx")

Let’s see what happened:

head(time\_use\_xl)

## # A tibble: 6 x 21

## sheet address row col is\_blank data\_type error logical numeric

## <chr> <chr> <int> <int> <lgl> <chr> <chr> <lgl> <dbl>

## 1 Index A1 1 1 FALSE character <NA> NA NA

## 2 Index B1 1 2 TRUE blank <NA> NA NA

## 3 Index C1 1 3 TRUE blank <NA> NA NA

## 4 Index D1 1 4 TRUE blank <NA> NA NA

## 5 Index E1 1 5 TRUE blank <NA> NA NA

## 6 Index F1 1 6 TRUE blank <NA> NA NA

## # … with 12 more variables: date <dttm>, character <chr>,

## # character\_formatted <list>, formula <chr>, is\_array <lgl>,

## # formula\_ref <chr>, formula\_group <int>, comment <chr>, height <dbl>,

## # width <dbl>, style\_format <chr>, local\_format\_id <int>

As you can see, the sheet was imported, but the result might be unexpected. Actually, time\_use\_xl is a tibble object, where each row is one cell of the Excel sheet. This might seem very complicated to handle, but you will see that it actually makes things way easier.

I only want to work on the English sheets so I use the following code to ignore the French ones:

sheets <- xlsx\_sheet\_names("time-use.xlsx") %>%

keep(grepl(pattern = ".\*day$", .))

Also, there’s a sheet that aggregates the results for week days and weekends, which I also ignore.

Now, to extract the tables from each sheet I wrote the following function:

extract\_data <- function(sheet){

activities <- sheet %>%

filter(col == 2) %>%

select(row, character) %>%

filter(row %in% seq(6,58)) %>%

rename(activities = character) %>%

select(-row)

cols\_to\_extract <- sheet %>%

filter(grepl("Population who completed.\*", character)) %>%

pull(col)

headers\_pos <- cols\_to\_extract - 1

headers <- sheet %>%

filter(col %in% headers\_pos, row == 3) %>%

pull(character)

cols\_to\_extract %>%

map(~filter(sheet, col %in% .)) %>%

map(~select(., sheet, address, row, col, character)) %>%

map(~filter(., row %in% seq(6,58))) %>%

map(~select(., character)) %>%

map2(.x = ., .y = headers, ~mutate(.x, "population" = .y)) %>%

map(., ~bind\_cols(activities, .)) %>%

bind\_rows()

}

Let’s study it step by step and see how it works. First, there’s the argument, sheet. This function will be mapped to each sheet of the workbook. Then, the first block I wrote, extracts the activities:

activities <- sheet %>%

filter(col == 2) %>%

select(row, character) %>%

filter(row %in% seq(6,58)) %>%

rename(activities = character) %>%

select(-row)

I only keep the second column (filter(col == 2)); col is a column of the tibble and if you look inside the workbook, you will notice that the activities are on the second column, or the B column. Then, I select two columns, the row and the character column. row is self-explanatory and character actually contains whatever is written inside the cells. Then, I only keep rows 6 to 58, because that is what interests me; the rest is either empty cells, or unneeded. Finally, I rename the character column to activities and remove the row column.

The second block:

cols\_to\_extract <- sheet %>%

filter(grepl("Population who completed.\*", character)) %>%

pull(col)

returns the index of the columns I want to extract. I am only interested in the people that have completed the activities, so using grepl() inside filter(), I located these columns, and use pull()… to pull them out of the data frame! cols\_to\_extract is thus a nice atomic vector of columns that I want to keep.

In the third block, I extract the headers:

headers\_pos <- cols\_to\_extract - 1

Why - 1? This is because if you look in the Excel, you will see that the headers are one column before the column labeled “People who completed the activity”. For example on column G, I have “People who completed the activity” and on column F I have the header, in this case “Male”.

Now I actually extract the headers:

headers <- sheet %>%

filter(col %in% headers\_pos, row == 3) %>%

pull(character)

Headers are always on the third row, but on different columns, hence the col %in% headers\_pos. I then pull out the values inside the cells with pull(character). So my headers object will be an atomic vector with “All”, “Male”, “Female”, “10 - 19 years”, etc… everything on row 3.

Finally, the last block, actually extracts the data:

cols\_to\_extract %>%

map(~filter(sheet, col %in% .)) %>%

map(~select(., sheet, address, row, col, character)) %>%

map(~filter(., row %in% seq(6,58))) %>%

map(~select(., character)) %>%

map2(.x = ., .y = headers, ~mutate(.x, "population" = .y)) %>%

map(., ~bind\_cols(activities, .)) %>%

bind\_rows()

cols\_to\_extract is a vector with the positions of the columns that interest me. So for example “4”, “7”, “10” and so on. I map this vector to the sheet, which returns me a list of extracted data frames. I pass this down to a select() (which is inside map()… why? Because the input parameter is a list of data frames). So for each data frame inside the list, I select the columns sheet, address, row, col and character. Then, for each data frame inside the list, I use filter() to only keep the rows from position 6 to 58. Then, I only select the character column, which actually contains the text inside the cell. Then, using map2(), I add the values inside the headers object as a new column, called population. Then, I bind the activities column to the data frame and bind all the rows together.

Time to use this function! Let’s see:

clean\_data <- sheets %>%

map(~filter(time\_use\_xl, sheet %in% .)) %>%

set\_names(sheets) %>%

map(extract\_data) %>%

map2(.x = ., .y = sheets, ~mutate(.x, "day" = .y)) %>%

bind\_rows() %>%

select(day, population, activities, time = character)

glimpse(clean\_data)

## Observations: 2,968

## Variables: 4

## $ day <chr> "Year 2014\_Monday til Friday", "Year 2014\_Monday til …

## $ population <chr> "All", "All", "All", "All", "All", "All", "All", "All…

## $ activities <chr> "Personal care", "Sleep", "Eating", "Other personal c…

## $ time <chr> "11:07", "08:26", "01:47", "00:56", "07:37", "07:47",…

So I map my list of sheets to the tibble I imported with readxl, use set\_names to name the elements of my list (which is superfluous, but I wanted to show this; might interest you!) and then map this result to my little function. I could stop here, but I then add a new column to each data frame that contains the day on which the data was measured, bind the rows together and reorder the columns. Done!

Now, how did I come up with this function? I did not start with a function. I started by writing some code that did what I wanted for one table only, inside one sheet only. Only when I got something that worked, did I start to generalize to several tables and then to several sheets. Most of the time spent was actually in trying to find patterns in the Excel sheet that I could use to write my function (for example noticing that the headers I wanted where always one column before the column I was interested in). This is my advice when working with function programming; always solve the issue for one element, wrap this code inside a function, and then simply map this function to a list of elements!

Let’s see into deeper as per the below code

I’ll first show my code (with comments of edits I’ve made), then Bruno’s code, and then that our results are the same:

**Set Up**

library(tidyverse)

library(tidyxl)

library(magrittr)

time\_use\_xl <- xlsx\_cells("data/time\_use.xlsx")

**My Code**

sheets <- # use object we wrote it and i prefer str\_detect

unique(time\_use\_xl$sheet) %>%

str\_subset("day$")

# avoid hard coding 6 and 58

after\_na\_to\_before\_na <- function(df, col\_with\_na){

rows\_na <- is.na(df[[col\_with\_na]])

first\_row <- min(which(!rows\_na))

last\_row <- first\_row + min(which(rows\_na[first\_row:nrow(df)])) - 2

first\_row:last\_row

}

# this function has a variety of edits, the first is that day here lets us

# use imap later which is nice!

extract\_data <- function(sheet, day){

activities <-

sheet %>%

filter(col == 2) %>%

select(row, character)

rows\_of\_interest <- # again, this is done to avoid hard coding 6 and 58

activities$row[after\_na\_to\_before\_na(activities, "character")]

# no reason this can't be a vector here

activities %<>%

filter(row %in% rows\_of\_interest) %>%

rename(activities = character) %>%

pull(activities)

cols\_to\_extract <- sheet %>% # i personally prefer str\_detect

filter(str\_detect(character, "^Population who completed")) %>%

pull(col)

headers\_pos <- cols\_to\_extract - 1

headers <- sheet %>%

filter(col %in% headers\_pos, row == 3) %>%

pull(character)

# can avoid all of the maps!

sheet %>%

filter(col %in% cols\_to\_extract) %>%

arrange(col) %>%

select(sheet, address, row, col, character) %>%

filter(row %in% rows\_of\_interest) %>%

select(col, character) %>%

mutate(

population = rep(headers, times = table(col)),

col = NULL,

activities = rep(activities, times = n\_distinct(col)),

day = day

)

}

clean\_data\_ben <-

# i like split better than map filter, and it has bonus of keeping names

split(time\_use\_xl, time\_use\_xl$sheet)[sheets] %>%

# imap uses list name to populate day (see 2nd parameter of extract\_data

imap(extract\_data) %>%

bind\_rows() %>%

select(day, population, activities, time = character)

Now, it would be too bad not to further analyze this data. I’ve been wanting to play around with  
the {flexdashboard} package for some time now, but never really got the opportunity to do so.  
The opportunity has now arrived. Using the cleaned data from the last post, I will further tweak  
it a little bit, and then produce a very simple dashboard using {flexdashboard}.

To make the data useful, I need to convert the strings that represent the amount of time spent  
doing a task (for example “1:23”) to minutes. For this I use the {chron} package:

clean\_data <- clean\_data %>%

mutate(time\_in\_minutes = paste0(time, ":00")) %>% # I need to add ":00" for the seconds else it won't work

mutate(time\_in\_minutes =

chron::hours(chron::times(time\_in\_minutes)) \* 60 +

chron::minutes(chron::times(time\_in\_minutes)))

rio::export(clean\_data, "clean\_data.csv")

Now we’re ready to go! Below is the code to build the dashboard; if you want to try, you should  
copy and paste the code inside a Rmd document:

---

shiny

---

`` `{r setup, include=FALSE}

library(flexdashboard)

library(shiny)

library(tidyverse)

library(plotly)

library(ggthemes)

main\_categories <- c("Personal care",

"Employment",

"Study",

"Household and family care",

"Voluntary work and meetings",

"Social life and entertainment",

"Sports and outdoor activities",

"Hobbies and games",

"Media",

"Travel")

df <- read.csv("clean\_data.csv") %>%

rename(Population = population) %>%

rename(Activities = activities)

`` `

Inputs {.sidebar}

-----------------------------------------------------------------------

`` `{r}

selectInput(inputId = "activitiesName",

label = "Choose an activity",

choices = unique(df$Activities))

selectInput(inputId = "dayName",

label = "Choose a day",

choices = unique(df$day),

selected = "Year 2014\_Monday til Friday")

selectInput(inputId = "populationName",

label = "Choose a population",

choices = unique(df$Population),

multiple = TRUE, selected = c("Male", "Female"))

`` `

The Time Use Survey (TUS) aims to measure accurately how people allocate their time across different day-to-day activities. To this end, people are asked to keep records of all their activities in a time diary. For each activity, additional information is collected about whether or not the person was alone doing it or together with other persons, where did the activity take place, etc. The main studies on time use have been conducted to calculate indicators making possible comparative analysis of quality of life within the same population or between countries. International studies care more about specific activities such as work (unpaid or not), free time, leisure, personal care (including sleep), etc.

Source: http://statistiques.public.lu/en/surveys/espace-households/time-use/index.html

Layout based on https://jjallaire.shinyapps.io/shiny-biclust/

Row

-----------------------------------------------------------------------

### Minutes spent per day on certain activities

`` `{r}

dfInput <- reactive({

df %>% filter(Activities == input$activitiesName,

Population %in% input$populationName,

day %in% input$dayName)

})

dfInput2 <- reactive({

df %>% filter(Activities %in% main\_categories,

Population %in% input$populationName,

day %in% input$dayName)

})

renderPlotly({

df1 <- dfInput()

p1 <- ggplot(df1,

aes(x = Activities, y = time\_in\_minutes, fill = Population)) +

geom\_col(position = "dodge") +

theme\_minimal() +

xlab("Activities") +

ylab("Time in minutes") +

scale\_fill\_gdocs()

ggplotly(p1)})

`` `

Row

-----------------------------------------------------------------------

### Proportion of the day spent on main activities

`` `{r}

renderPlotly({

df2 <- dfInput2()

p2 <- ggplot(df2,

aes(x = Population, y = time\_in\_minutes, fill = Activities)) +

geom\_bar(stat="identity", position="fill") +

xlab("Proportion") +

ylab("Proportion") +

theme\_minimal() +

scale\_fill\_gdocs()

ggplotly(p2)

})

`` `

You will see that I have defined the following atomic vector:

main\_categories <- c("Personal care",

"Employment",

"Study",

"Household and family care",

"Voluntary work and meetings",

"Social life and entertainment",

"Sports and outdoor activities",

"Hobbies and games",

"Media",

"Travel")

If you go back to the raw Excel file, you will see that these main categories are then split into  
secondary activities. The first bar plot of the dashboard does not distinguish between the main and  
secondary activities, whereas the second barplot only considers the main activities. I could  
have added another column to the data that helped distinguish whether an activity was a main or secondary one,  
but I was lazy. The source code of the dashboard is very simple as it uses R Markdown. To have  
interactivity, I’ve used Shiny to dynamically filter the data, and built the plots with {ggplot2}.  
Finally, I’ve passed the plots to the ggplotly() function from the {plotly} package for some  
quick and easy javascript goodness!