Last week It was written a [blog post](https://www.brodrigues.co/blog/2019-01-04-newspapers/) where It was analyzed  
one year of newspapers ads from 19th century newspapers. The data is made available by the  
[national library of Luxembourg](https://data.bnl.lu/data/historical-newspapers/).  
In this blog post, which is part 1 of a 2 part series, I extract data from the 257gb archive, which  
contains 10 years of publications of the *L’Union*, another 19th century Luxembourguish newspaper  
written in French. As explained in the previous post, to make life easier to data scientists,  
the national library also included ALTO and METS files (which are a XML files used to  
describe the layout and contents of physical text sources, such as pages of a book or newspaper)  
which can be easily parsed by R.

This is how a ALTO file looks like:

Each page of the newspaper of a given day has one ALTO file.  
This is how a METS file looks like:

For each daily issue of the newspaper, there is a METS file. So 1 METS file for 4 ALTO files.

In my last blog post, I only extracted the words from the ALTO file (red rectangles of the first  
screenshot) and did not touch the METS file.  
The problem of doing this is that I get all the words for each page, without knowing which  
come from the same article. If want to know which words come from the same article, need to use  
the info from the METS file. From the METS file I have the ID of the article, and some other  
metadata, such as the title of the article and the type of the article (which can be *article*,  
*advertisement*, etc). The information highlighted with the green rectangles in the METS file  
can be linked to the green rectangles from the ALTO files. My goal is to get the following data  
frame from the METS file:

and this data frame from the ALTO files:

As you can see, by combining both data frames I can know which words come from the same article,  
which will be helpful for further analysis.  
[A lot of things happened in the 1860s.](https://en.wikipedia.org/wiki/1860s)  
I am really curious to see if and how these events where reported in a Luxembourguish newspaper.  
I am particularly curious about how long it took to report certain news from far away, such as the  
assassination of Abraham Lincoln. But before that I need to extract the data!

I will only focus on the METS file. The logic for the ALTO file is the same. All the source code  
will be in the appendix of this blog post.

First, let’s take a look at a METS file:

library(tidyverse)

mets <- read\_file("1533660\_newspaper\_lunion\_1860-11-14/1533660\_newspaper\_lunion\_1860-11-14-mets.xml")

This is how it looks like:

"\r\n\r\n \r\n \r\n CCS docWORKS/METAe Version 6.4-3\r\n docWORKS-ID: 101636\r\n \r\n \r\n \r\n \r\n \r\n \r\n lunion\r\n \r\n L'UNION.\r\n \r\n \r\n \r\n \r\n \r\n \r\n \r\n \r\n \r\n \r\n Chemins de fer. — Service d'hiver.\r\n \r\n \r\n fr\r\n ...."

As usual when you import text files like this, it’s always a good idea to split the file. I will  
split at the "DMDID" character. Take a look back at the second screenshot. The very first tag,  
first row, first word after div is "DMDID". By splitting at this level, I will get back a list,  
where each element is the content of this div DMDID block. This is exactly what I need, since  
this block contains the information from the green rectangles.  
So let’s split the mets variable at this level:

mets\_articles <- mets %>%

str\_split("DMDID") %>%

flatten\_chr()

Let’s take a look at mets\_articles:

str(mets\_articles)

chr [1:25] "\r\n

Doesn’t seem to be very helpful, but actually it is. We can see that mets\_articles is a now a list  
of 25 elements.

This means that for each element of mets\_articles, I need to get the identifier, the label, the type  
(the red rectangles from the screenshot), but also the information from the "BEGIN" element (the green  
rectangle).

To do this, I’ll be using regular expressions. In general, I start by experimenting in the console,  
and then when things start looking good, I write a function. Here is this function:

extractor <- function(string, regex, all = FALSE){

if(all) {

string %>%

str\_extract\_all(regex) %>%

flatten\_chr() %>%

str\_extract\_all("[:alnum:]+", simplify = FALSE) %>%

map(paste, collapse = "\_") %>%

flatten\_chr()

} else {

string %>%

str\_extract(regex) %>%

str\_extract\_all("[:alnum:]+", simplify = TRUE) %>%

paste(collapse = " ") %>%

tolower()

}

}

This function may seem complicated, but it simply encapsulates some pretty standard steps to get  
the data I need. I had to consider two cases. The first case is when I need to extract all the  
elements with str\_extract\_all(), or only the first occurrence, with str\_extract().  
Let’s test it on the first article of the mets\_articles list:

mets\_articles\_1 <- mets\_articles[1]

extractor(mets\_articles\_1, "ID", all = FALSE)

## [1] "id"

Let’s see what happens with all = TRUE:

extractor(mets\_articles\_1, "ID", all = TRUE)

## [1] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [15] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [29] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [43] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [57] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [71] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [85] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [99] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [113] "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID" "ID"

## [127] "ID" "ID" "ID" "ID" "ID"

This seems to work as intended. Since I need to call this function several times, I’ll be writing  
another function that extracts all I need:

extract\_mets <- function(article){

id <- article %>%

extractor("(?<=ID)(.\*?)(?=LABEL)")

label <- article %>%

extractor("(?<=LABEL)(.\*?)(?=TYPE)")

type <- article %>%

extractor("(?<=TYPE)(.\*?)(?=>)")

begins <- article %>%

extractor("(?<=BEGIN)(.\*?)(?=BETYPE)", all = TRUE)

tibble::tribble(~label, ~type, ~begins, ~id,

label, type, begins, id) %>%

unnest()

}

This function uses complex regular expressions to extract the strings I need, and then puts  
the result into a data frame, with the tibble() function. I then use unnest(), because label,  
type, begins and id are not the same length. label, type and id are of length 1, while  
begins is longer. This means that when I put them into a data frame it looks like this:

tribble(~a, ~b,

"a", rep("b", 4))

## # A tibble: 1 x 2

## a b

##

## 1 a

With unnest(), I get a nice data frame:

tribble(~a, ~b,

"a", rep("b", 4)) %>%

unnest()

## # A tibble: 4 x 2

## a b

##

## 1 a b

## 2 a b

## 3 a b

## 4 a b

Now, I simply need to map this function to all the files and that’s it! For this, I will write yet  
another helper function:

mets\_csv <- function(page\_path){

page <- read\_file(page\_path)

doc\_name <- str\_extract(page\_path, "(?<=/).\*")

mets\_articles <- page %>%

str\_split("DMDID") %>%

flatten\_chr()

mets\_df <- map\_df(mets\_articles, extract\_mets)

mets\_df <- mets\_df %>%

mutate(document = doc\_name)

write\_csv(mets\_df, paste0(page\_path, ".csv"))

}

This function takes the path to a METS file as input, and processes it using the steps I explained  
above. The only difference is that I add a column containing the name of the file that was processed,  
and write the resulting data frame directly to disk as a data frame. Finally, I can map this function to all the METS  
files:

# Extract content from METS files

pages\_mets <- str\_match(list.files(path = "./", all.files = TRUE, recursive = TRUE), ".\*mets.xml") %>%

discard(is.na)

library(furrr)

plan(multiprocess, workers = 8)

tic <- Sys.time()

future\_map(pages\_mets, mets\_csv)

toc <- Sys.time()

toc - tic

I use {furrr} to extract the data from all the files in parallel, by putting 8 cores of my CPU to  
work. This took around 3 minutes and 20 seconds to finish.

That’s it for now, stay tuned for part 2 where I will analyze this fresh data!

Hope you enjoyed! If you found this blog post useful, you might want to follow  
me on [twitter](https://www.twitter.com/brodriguesco) for blog post updates and  
[buy me an espresso](https://www.buymeacoffee.com/brodriguesco) or [paypal.me](https://www.paypal.me/brodriguesco).

[Buy me an Espresso](https://www.buymeacoffee.com/brodriguesco)

**Appendix**

extract\_alto <- function(article){

begins <- article[1] %>%

extractor("(?<=^ID)(.\*?)(?=HPOS)", all = TRUE)

content <- article %>%

extractor("(?<=CONTENT)(.\*?)(?=WC)", all = TRUE)

tibble::tribble(~begins, ~content,

begins, content) %>%

unnest()

}

alto\_csv <- function(page\_path){

page <- read\_file(page\_path)

doc\_name <- str\_extract(page\_path, "(?<=/text/).\*")

alto\_articles <- page %>%

str\_split("TextBlock ") %>%

flatten\_chr()

alto\_df <- map\_df(alto\_articles, extract\_alto)

alto\_df <- alto\_df %>%

mutate(document = doc\_name)

write\_csv(alto\_df, paste0(page\_path, ".csv"))

}

alto <- read\_file("1533660\_newspaper\_lunion\_1860-11-14/text/1860-11-14\_01-00001.xml")

# Extract content from alto files

pages\_alto <- str\_match(list.files(path = "./", all.files = TRUE, recursive = TRUE), ".\*/text/.\*.xml") %>%

discard(is.na)

library(furrr)

plan(multiprocess, workers = 8)

tic <- Sys.time()

future\_map(pages\_alto, alto\_csv)

toc <- Sys.time()

toc - tic

#Time difference of 18.64776 mins