The polarsteps company makes money by selling you a automagically created a photo album of your travels. I did not really like that photo album, so I want to do something with the texts themselves.  
There are several options: I could scrape the webpages that contain my travels. Or I could download my data and work with that. Polarsteps explains that the data you create remains yours and you can download a copy. Which is what I did.

Now my approach is a bit roundabout and probably not the most effective but I thought it would  
demonstrate how to work with lists.I first extract the ‘steps’ (individual posts) data and turn that all into a rectangular format finally I extract those elements again and turn that into a document again (other post). I could have gone a more direct route.

**Loading the data**

*First enable the tools:*

library(tidyverse)

## ── Attaching packages ─────────────────────────────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ ggplot2 3.3.0 ✓ purrr 0.3.3

## ✓ tibble 3.0.0 ✓ dplyr 0.8.5

## ✓ tidyr 1.0.2 ✓ stringr 1.4.0

## ✓ readr 1.3.1 ✓ forcats 0.5.0

## ── Conflicts ────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──

## x dplyr::filter() masks stats::filter()

## x dplyr::lag() masks stats::lag()

library(jsonlite)

##

## Attaching package: 'jsonlite'

## The following object is masked from 'package:purrr':

##

## flatten

The data comes in a zip which contains two folders trip and user data.  
I’m interested in the trip data.

user\_data

├── trip

│ └── New\ Zealand\_3129570

└── user

└── user.json

The trip data contains two json files and folders for photos and videos.

New\ Zealand\_3129570

├── locations.json

├── photos

├── trip.json

└── videos

The locations.json contains only gps coordinates and names linking back to the trip. but the  
trip also contains these coordinates so for me this locations.json file is less relevent.  
I extracted both of these json files but I will only work with the trip.json.

trip <- jsonlite::read\_json("trip.json")

When you receive your file it is in a json format. Which is a bunch of lists inside lists.  
We can work with lists in R, but usually we want to work with rectangular data, such as data.frames.  
because that is just so much more easy with the tidyverse tools.

names(trip)

[1] "featured" "feature\_date" "likes"

[4] "id" "fb\_publish\_status" "step\_count"

[7] "total\_km" "featured\_priority\_for\_new\_users" "feature\_text"

[10] "open\_graph\_id" "start\_date" "type"

[13] "uuid" "user\_id" "cover\_photo\_path"

[16] "slug" "all\_steps" "views"

[19] "end\_date" "cover\_photo" "visibility"

[22] "planned\_steps\_visible" "cover\_photo\_thumb\_path" "language"

[25] "name" "is\_deleted" "timezone\_id"

[28] "summary"

The top of the trip file contains an overview of the trip: how many posts are there, what is the name etc.  
However I’m more focused on the details in every ‘step’. If you explore the all\_steps, it contains all of the individual posts. Every post is another list. I’m turning these list into a data.frame.

I’m approaching this in the following way:

* extract one example,
* create helper functions that work on that one example,
* apply the helper functions with the purrr package on the entire list of all\_steps.

I think I got this approach from Jenny Bryan (see bottom for references).

**Extract one example**

all\_steps <- trip$all\_steps

# try one,

example <- all\_steps[[1]]

So what can we find in this one example list?

glimpse(example)

For all the steps we have the following information *(I have redacted this a bit, the google crawler is mighty and everything on the internet lives forever and I don’t want to share everything with my readers)*:

List of 23

$ weather\_temperature : num 11

$ likes : int 0

$ supertype : chr "normal"

$ id : int 24041483

$ fb\_publish\_status : NULL

$ creation\_time : num 1.58e+09

$ main\_media\_item\_path: NULL

$ location :List of 9

..$ detail : chr "Netherlands"

..$ name : chr "REDACTED"

..$ uuid : chr "REDACTED"

..$ venue : NULL

..$ lat : num 99999

..$ lon : num 99999

..$ full\_detail : chr "Netherlands"

..$ id : int 999999999

..$ country\_code: chr "NL"

$ open\_graph\_id : NULL

$ type : NULL

$ uuid : chr "REDACTED"

$ comment\_count : int 0

$ location\_id : int 99999999

$ slug : chr "REDACTED"

$ views : int 0

$ description : chr "Roel: We zijn er klaar voor hoor, alles ligt bij de koffers (hopen dat het past \U0001f605) onze ochtendkoffie "| \_\_truncated\_\_

$ start\_time : num 1.58e+09

$ trip\_id : int 3129570

$ end\_time : NULL

$ weather\_condition : chr "rain"

$ name : chr "Laatste voorbereidingen"

$ is\_deleted : logi FALSE

$ timezone\_id : chr "Europe/Amsterdam"

Of interest here:

* I wanted the texts and they live in ‘description’.
* The title of the post is in ‘name’
* The polarsteps application is deeply integrated with facebook (scary!)
* time is in unix timestamps
* Temperature is in degrees Celsius (the international norm)
* The description is in utf-8 but my printing here is not and does not show this emoji correctly.

**Create extractor functions**

Most things I care about in this file are one level deep. I can create a general function that extracts them, based on the name of the field: start\_time, weather\_temperature, description, etc.

But I quickly realised I wanted to do something special with the location and time so they get their own functions.

#' General extractor function

#'

#' Give it the name of a field and it extracts that.

#' Also deals with non existing or empty fields (can happen in lists)

#' by replacing that with empty character field.

#' Alternative is to use purrr::safely

extract\_field <- function(item, field){

result = item[[field]]

if(is.null(result)){result = ""}

result

}

#' Extractor for location

#'

#' Extracts location list and pastes together the name of the location, country code and

#' latitude and longitude.

extract\_location\_string <- function(item){

location = item[["location"]]

paste0(

"In ",location[["name"]], " ",location[["full\_detail"]], " (",location[["country\_code"]], ") ",

"[",location[["lat"]],",",location[["lon"]],"]"

)

}

#' Time extractor

#'

#' Turns unix timestamp into real time, and uses the correct timezone.

#' this might be a bit of an overkill because I'm immediately turning this

#' into text again.

extract\_time = function(item){

timezone = item[["timezone\_id"]]

start\_time = item[["start\_time"]] %>% anytime::anytime(asUTC = FALSE,tz = timezone)

paste(start\_time, collapse = ", ")

}

**Apply the extractors on the example**

extract\_field(example, "name")

extract\_location\_string(example)

extract\_time(example)

"Laatste voorbereidingen"

"In Leiden Netherlands (NL) [52.1720626,4.5076576]"

"2020-02-01 09:23:07"

**Apply all extractors on all steps in the trip**

First create an empty data.frame and add new columns for the fields i’m interested in.

base <- tibble(

stepnr = [seq.int](http://seq.int)(from = 1, to = length(all\_steps), by=1)

)

tripdetails <-

base %>%

mutate(

title = purrr::map\_chr(all\_steps, ~extract\_field(.x, "name")),

description = purrr::map\_chr(all\_steps, ~extract\_field(.x, "description")),

slug = purrr::map\_chr(all\_steps, ~extract\_field(.x, "slug")),

temperature = purrr::map\_dbl(all\_steps, ~extract\_field(.x, "weather\_temperature")),

temperature = round(temperature, 2),

weather\_condition = purrr::map\_chr(all\_steps, ~extract\_field(.x, "weather\_condition")),

location = purrr::map\_chr(all\_steps, extract\_location\_string),

time = purrr::map\_chr(all\_steps, extract\_time)

)

End result

**Conclusion**

I wanted to print the descriptions etc into a word file or something for printing but  
that can be found in the next post.

**State of the machine**

At the moment of creation (when I knitted this document ) this was the state of my machine:

sessioninfo::session\_info()

## ─ Session info ───────────────────────────────────────────────────────────────

## setting value

## version R version 3.6.3 (2020-02-29)

## os macOS Mojave 10.14.6

## system x86\_64, darwin15.6.0

## ui X11

## language (EN)

## collate en\_US.UTF-8

## ctype en\_US.UTF-8

## tz Europe/Amsterdam

## date 2020-04-22

##

## ─ Packages ───────────────────────────────────────────────────────────────────

## package \* version date lib source

## assertthat 0.2.1 2019-03-21 [1] CRAN (R 3.6.0)

## backports 1.1.5 2019-10-02 [1] CRAN (R 3.6.0)

## blogdown 0.18 2020-03-04 [1] CRAN (R 3.6.1)

## bookdown 0.18 2020-03-05 [1] CRAN (R 3.6.1)

## broom 0.5.5 2020-02-29 [1] CRAN (R 3.6.0)

## cellranger 1.1.0 2016-07-27 [1] CRAN (R 3.6.0)

## cli 2.0.2 2020-02-28 [1] CRAN (R 3.6.0)

## colorspace 1.4-1 2019-03-18 [1] CRAN (R 3.6.0)

## crayon 1.3.4 2017-09-16 [1] CRAN (R 3.6.0)

## DBI 1.1.0 2019-12-15 [1] CRAN (R 3.6.0)

## dbplyr 1.4.2 2019-06-17 [1] CRAN (R 3.6.0)

## digest 0.6.25 2020-02-23 [1] CRAN (R 3.6.0)

## dplyr \* 0.8.5 2020-03-07 [1] CRAN (R 3.6.0)

## ellipsis 0.3.0 2019-09-20 [1] CRAN (R 3.6.0)

## evaluate 0.14 2019-05-28 [1] CRAN (R 3.6.0)

## fansi 0.4.1 2020-01-08 [1] CRAN (R 3.6.0)

## forcats \* 0.5.0 2020-03-01 [1] CRAN (R 3.6.0)

## fs 1.3.2 2020-03-05 [1] CRAN (R 3.6.0)

## generics 0.0.2 2018-11-29 [1] CRAN (R 3.6.0)

## ggplot2 \* 3.3.0 2020-03-05 [1] CRAN (R 3.6.0)

## glue 1.3.2 2020-03-12 [1] CRAN (R 3.6.0)

## gtable 0.3.0 2019-03-25 [1] CRAN (R 3.6.0)

## haven 2.2.0 2019-11-08 [1] CRAN (R 3.6.0)

## hms 0.5.3 2020-01-08 [1] CRAN (R 3.6.0)

## htmltools 0.4.0 2019-10-04 [1] CRAN (R 3.6.0)

## httr 1.4.1 2019-08-05 [1] CRAN (R 3.6.0)

## jsonlite \* 1.6.1 2020-02-02 [1] CRAN (R 3.6.0)

## knitr 1.28 2020-02-06 [1] CRAN (R 3.6.0)

## lattice 0.20-38 2018-11-04 [1] CRAN (R 3.6.3)

## lifecycle 0.2.0 2020-03-06 [1] CRAN (R 3.6.0)

## lubridate 1.7.4 2018-04-11 [1] CRAN (R 3.6.0)

## magrittr 1.5 2014-11-22 [1] CRAN (R 3.6.0)

## modelr 0.1.6 2020-02-22 [1] CRAN (R 3.6.0)

## munsell 0.5.0 2018-06-12 [1] CRAN (R 3.6.0)

## nlme 3.1-144 2020-02-06 [1] CRAN (R 3.6.3)

## pillar 1.4.3 2019-12-20 [1] CRAN (R 3.6.0)

## pkgconfig 2.0.3 2019-09-22 [1] CRAN (R 3.6.0)

## purrr \* 0.3.3 2019-10-18 [1] CRAN (R 3.6.0)

## R6 2.4.1 2019-11-12 [1] CRAN (R 3.6.0)

## Rcpp 1.0.4 2020-03-17 [1] CRAN (R 3.6.1)

## readr \* 1.3.1 2018-12-21 [1] CRAN (R 3.6.0)

## readxl 1.3.1 2019-03-13 [1] CRAN (R 3.6.0)

## reprex 0.3.0 2019-05-16 [1] CRAN (R 3.6.0)

## rlang 0.4.5 2020-03-01 [1] CRAN (R 3.6.0)

## rmarkdown 2.1 2020-01-20 [1] CRAN (R 3.6.0)

## rstudioapi 0.11 2020-02-07 [1] CRAN (R 3.6.0)

## rvest 0.3.5 2019-11-08 [1] CRAN (R 3.6.0)

## scales 1.1.0 2019-11-18 [1] CRAN (R 3.6.0)

## sessioninfo 1.1.1 2018-11-05 [1] CRAN (R 3.6.0)

## stringi 1.4.6 2020-02-17 [1] CRAN (R 3.6.0)

## stringr \* 1.4.0 2019-02-10 [1] CRAN (R 3.6.0)

## tibble \* 3.0.0 2020-03-30 [1] CRAN (R 3.6.2)

## tidyr \* 1.0.2 2020-01-24 [1] CRAN (R 3.6.0)

## tidyselect 1.0.0 2020-01-27 [1] CRAN (R 3.6.0)

## tidyverse \* 1.3.0 2019-11-21 [1] CRAN (R 3.6.0)

## vctrs 0.2.4 2020-03-10 [1] CRAN (R 3.6.0)

## withr 2.1.2 2018-03-15 [1] CRAN (R 3.6.0)

## xfun 0.12 2020-01-13 [1] CRAN (R 3.6.0)

## xml2 1.2.2 2019-08-09 [1] CRAN (R 3.6.0)

## yaml 2.2.1 2020-02-01 [1] CRAN (R 3.6.0)

##

## [1] /Library/Frameworks/R.framework/Versions/3.6/Resources/library