Exercise 5: APIs

Karsten Donnay

18.06.2021

Contents

```
library(knitr)

## Global options
options(max.print="75")
opts_chunk$set(echo=FALSE,
                 cache=FALSE,
               prompt=FALSE,
               tidy=TRUE,
               comment=NA,
               message=FALSE,
               warning=FALSE)
opts_knit$set(width=75)
rm(list = ls())
```

Preparation

Install and loadrequired packages

```
# Uncomment to install packages toInstall <- c('httr','jsonlite',
# 'yaml','guardianapi','maps','dplyr','ggplot2','digest','ROAuth','rtweet')
# install.packages(pkgs = toInstall)

library(httr)
```

```r
library(jsonlite)
library(yaml)
library(ggplot2)
```

Useful ressources

- List of APIs: https://www.programmableweb.com
- R Packages: https://ropensci.org/packages/

Application: Guardian API

Registration

- Go to: https://open-platform.theguardian.com/access
- Register for a developer key
- Save key in yaml file in a secure location (e.g. **not** in a Dropbox folder)
- Load key into R and set global variable

```r
guardian_creds <- yaml::yaml.load_file("~/Documents/Keys/GuardianAPI_key.yaml")
Sys.setenv(GU_API_KEY = guardian_creds)
```

Build your own access to the API

Procedure

1. Read the documentation of the API: https://open-platform.theguardian.com/documentation/

- which endpoints are there?
- How do queries look like?
- Which rules do I need to follow?

2. Enpoints:

- */search*: returns all pieces of content in the API.
  - Example: 'https://content.guardianapis.com/search?q=debates'
- */tags*: returns all tags in the API
- */sections*: returns all sections in the API.
- */editions*: returns all editions in the API.

3. Build queries in R and test them. Start with simple queries and then add functionality.

Create a function that creates appropriate queries

- Needs to be in ASCII-Encoding / "Percent-Encoding" / URL-Encoding (can be done with utils::URLEncode). This is signalled by these '%20' in the url.

```r
print(utils::URLencode("This is a test."))
```

```
[1] "This%20is%20a%20test."
```

```r
print(utils::URLdecode("This%20is%20a%20test."))
```

```
[1] "This is a test."
```

To build an own query we need to:

- understand the syntax of the queries we will send.
  - Starts every time with: 'https://content.guardianapis.com'
  - Then specifies the endpoint (e.g. '/search')

    - Then adds a query introduced by '?q=' and keywords that can be concatenated by 'AND' or 'AND NOT'
    - Optional parameters that specify addiditonal fields that should be returned (e.g. the word count of an article)
    - Mandatory provision of your API key that is being used to identify yourself and to manage rate limits.
    - Finally: queries need to be URL encoded (see above)
- send queries with valid syntax to the API with httr::GET()

```r
build_guardian_query <- function(content = NULL, not_content = NULL, tag = NULL,
    from_date = NULL, to_date = NULL) {
    base_url <- "https://content.guardianapis.com"
    # Currently we only want to request the 'search' endpoint
    endpoint <- "/search"

    # Paste the basic query and the endpoint together
    query_url <- paste(base_url, endpoint, sep = "")

    # Append desired keywords with the 'AND' if existent
    ifelse(is.null(content), query_url <- query_url, query_url <- paste(query_url,
        "?q=", paste(content, collapse = " AND "), sep = ""))

    # Append the not desired keywords with the 'AND NOT' if existent
    ifelse(is.null(not_content), query_url <- query_url, query_url <- paste(query_url,
        " AND NOT ", paste(not_content, collapse = " AND NOT "), sep = ""))

    # We also want to get the word count of articles
    query_url <- paste(query_url, "&show-fields=wordcount", sep = "")

    # Append the API key to the query
    query_url <- paste(query_url, "&api-key=", guardian_creds, sep = "")
    # query_url <- paste(query_url, '&api-key=', guardian_creds$api_key,
    # sep='')

    # Finally: ASCII encode our query to make it valid
    query_url <- utils::URLencode(query_url)

    return(query_url)
}
```

Send a request to API with **httr::GET**

```r
query_url <- build_guardian_query(content = "UK", not_content = "brexit")
print(query_url)
```

```
[1] "https://content.guardianapis.com/search?q=UK%20AND%20NOT%20brexit&show-fields=wordcount&api-key=79
```

```r
articles_get <- httr::GET(query_url)
save(articles_get, file = paste("data/guardianapi-UKnews-notbrexit-", Sys.Date(),
    ".Rda", sep = ""))
```

What is returned has several properties:

- articles_get**$url**: the url used to query the API
- articles_get**$status_code**: the corresponding status code of the query. 200 means it was without any error.
- articles_get**$content**: contains the content of the query. This is what we are mainly interested in.

- articles_get**$date**: Date when the query was sent.

Next steps: transform it into a dataframe so we can work with it.

- Transform it to JSON
- Transform JSON to data frame

```r
load("data/guardianapi-UKnews-notbrexit-2021-06-18.Rda")

# Transforms the response to a JSON data format
articles_text <- httr::content(articles_get, "text")
substr(articles_text, start = 1, stop = 500)
```

```
[1] "{\"response\":{\"status\":\"ok\",\"userTier\":\"developer\",\"total\":414749,\"startIndex\":1,\"pag
```

This looks like JSON. Next, we need to use **jsonlite** to transform it into a data.frame.

```r
# Transforms the JSON Data to an R object
articles_json <- jsonlite::fromJSON(articles_text)

# Transforms every part of the response to a dataframe (e.g. also response
# status)
articles_df <- as.data.frame(articles_json)

# Extracts only the results (content)
articles_df <- articles_json$response$results
head(articles_df)
```

```
                                                                       id
1                       travel/2021/jun/08/warming-to-holidays-in-the-uk
2            travel/2021/may/28/uk-campsites-with-half-term-availability
3    politics/2021/jun/11/is-the-uk-sleepwalking-into-authoritarian-rule
4 business/2021/jun/16/uk-government-extends-moratorium-commercial-rents
5  uk-news/2021/feb/08/storm-darcy-snow-ice-further-disruption-across-uk
6              world/2021/jun/01/zero-daily-covid-deaths-announced-in-uk
     type sectionId sectionName   webPublicationDate
1 article    travel      Travel 2021-06-08T17:16:56Z
2 article    travel      Travel 2021-05-28T11:28:03Z
3 article  politics    Politics 2021-06-11T15:57:24Z
4 article  business    Business 2021-06-16T15:30:45Z
5 article   uk-news     UK news 2021-02-08T17:10:52Z
6 article     world  World news 2021-06-01T15:39:12Z
                                                        webTitle
1                       Warming to holidays in the UK | Letters
2                      UK campsites with half-term availability
3     Is the UK sleepwalking into authoritarian rule? | Letters
4          UK government extends moratorium on commercial rents
5 UK weather: Storm Darcy to cause further disruption across UK
6                       Zero daily Covid deaths announced in UK
                                                                                           webUrl
1                       https://www.theguardian.com/travel/2021/jun/08/warming-to-holidays-in-the-uk
2            https://www.theguardian.com/travel/2021/may/28/uk-campsites-with-half-term-availability
3    https://www.theguardian.com/politics/2021/jun/11/is-the-uk-sleepwalking-into-authoritarian-rule
4 https://www.theguardian.com/business/2021/jun/16/uk-government-extends-moratorium-commercial-rents
5  https://www.theguardian.com/uk-news/2021/feb/08/storm-darcy-snow-ice-further-disruption-across-uk
6              https://www.theguardian.com/world/2021/jun/01/zero-daily-covid-deaths-announced-in-uk
                                                                                                  apiU
```

```
1                       https://content.guardianapis.com/travel/2021/jun/08/warming-to-holidays-in-the-u
2            https://content.guardianapis.com/travel/2021/may/28/uk-campsites-with-half-term-availabili
3    https://content.guardianapis.com/politics/2021/jun/11/is-the-uk-sleepwalking-into-authoritarian-rul
4 https://content.guardianapis.com/business/2021/jun/16/uk-government-extends-moratorium-commercial-ren
5  https://content.guardianapis.com/uk-news/2021/feb/08/storm-darcy-snow-ice-further-disruption-across-u
6              https://content.guardianapis.com/world/2021/jun/01/zero-daily-covid-deaths-announced-in-u
  wordcount isHosted         pillarId pillarName
1       272    FALSE pillar/lifestyle  Lifestyle
2      1315    FALSE pillar/lifestyle  Lifestyle
3       595    FALSE      pillar/news       News
4       757    FALSE      pillar/news       News
5       507    FALSE      pillar/news       News
6       501    FALSE      pillar/news       News
```

Start using 'guardianapi'

There is already an existing package that enables retrieval of data from the Guardian API. Just to showcase its simplicity, we provide some basic code here.

```
gres <- guardianapi::gu_content(query = c("UK"), from_date = "2021-06-16", to_date = "2021-06-17")
save(gres, file = paste("data/guardianapi-GuardianR-UKnews-2021-06-16--2021-06-17--date-",
    Sys.Date(), ".Rda", sep = ""))
```

```
load("data/guardianapi-GuardianR-UKnews-2021-06-16--2021-06-17--date-2021-06-18.Rda")
head(gres)
```

```
# A tibble: 6 x 45
  id    type  section_id section_name web_publication_da~ web_title web_url
  <chr> <chr> <chr>      <chr>        <dttm>              <chr>     <chr>
1 busi~ arti~ business   Business     2021-06-16 15:30:45 UK gover~ https:~
2 worl~ arti~ world      World news   2021-06-16 13:48:17 UK criti~ https:~
3 medi~ arti~ media      Media        2021-06-16 16:07:59 Sony Mus~ https:~
4 worl~ live~ world      World news   2021-06-16 22:55:22 UK repor~ https:~
5 envi~ arti~ environme~ Environment  2021-06-16 05:01:20 UK faili~ https:~
6 soci~ arti~ society    Society      2021-06-16 12:37:16 UK chari~ https:~
# ... with 38 more variables: api_url <chr>, tags <list>, is_hosted <lgl>,
#   pillar_id <chr>, pillar_name <chr>, headline <chr>, standfirst <chr>,
#   trail_text <chr>, byline <chr>, main <chr>, body <chr>,
#   newspaper_page_number <chr>, wordcount <chr>,
#   first_publication_date <dttm>, is_inappropriate_for_sponsorship <chr>,
#   is_premoderated <chr>, last_modified <chr>, newspaper_edition_date <date>,
#   production_office <chr>, publication <chr>, short_url <chr>,
#   should_hide_adverts <chr>, show_in_related_content <chr>, thumbnail <chr>,
#   legally_sensitive <chr>, lang <chr>, is_live <chr>, body_text <chr>,
#   char_count <chr>, should_hide_reader_revenue <chr>,
#   show_affiliate_links <chr>, byline_html <chr>, live_blogging_now <chr>,
#   display_hint <chr>, sensitive <chr>, comment_close_date <dttm>,
#   commentable <chr>, star_rating <chr>
```

You see, via the API we get much more data, much more conveniently. We could get the same data by modifying the optional parameters in our functions but would advise using existing packages if they exist.

Application: APIs with OAuth (Twitter)

There are many established packages that connect with the Twitter API. Nevertheless, we use the Twitter API as an example to establish a different connection to an API: while the Guardian API used HTTR access, the Twitter API uses OAuth to handle requests to their APIs.

However, Twitter limited the access to their API severly after the Cambridge Analytica scandal. Subsequently, developers need to apply for a verified account that may then create apps that access the API. You will not be able to follow these steps immediately, but only after being accepted as a developer.

In case of acceptance and after creating an application of your own, you will receive 4 keys and secrets that you may use for authentication with the Twitter API. Keys and secrets are identifier and Twitter ratelimits based on a consumer basis: each consumer gets a ratelimit assigned per application he or she has authenticated. Practivally, this means that an application can request data for each consumer who authenticated the application. The results are the four identifier below:

- Consumer key: identifier for a Twitter account that authorized the specific application
- Consumer secret: second identifier for a Twitter account that authorized the application
- Application key: identifier for the application
- Application secret: second identifier for the application

Specify the authentication object (load data from a stored yaml file.)

```r
# Save the credentials of the application x auth to a yaml file in the form of:
# consumerkey : xxx consumersecret : xxx token : xxx-xxx secret : xxx
twitter_creds <- yaml::yaml.load_file("~/Documents/Keys/TwitterAPI_key.yaml")

# Feed it into a list
oauth <- list(consumer_key = twitter_creds$consumerkey, consumer_secret = twitter_creds$consumersecret,
    access_token = twitter_creds$token, access_token_secret = twitter_creds$secret)
my_oauth <- ROAuth::OAuthFactory$new(consumerKey = oauth$consumer_key, consumerSecret = oauth$consumer_
    oauthKey = oauth$access_token, oauthSecret = oauth$access_token_secret, needsVerifier = FALSE,
    handshakeComplete = TRUE, verifier = "1", requestURL = "https://api.twitter.com/oauth/request_token
    authURL = "https://api.twitter.com/oauth/authorize", accessURL = "https://api.twitter.com/oauth/acc
    signMethod = "HMAC")
```

Example: request the Twitter friends of a Twitter account (@guardian)

- Look up the URL in the documentation: https://developer.twitter.com/en
- https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-friends-ids

```r
url <- "https://api.twitter.com/1.1/friends/ids.json"
```

Request the first 5.000 friends of The Guardian

```r
friends_json <- my_oauth$OAuthRequest(URL = url, params = list(screen_name = "guardian",
    stringify_ids = "true"), method = "GET")
save(friends_json, file = paste("data/guardian-friends-json-", Sys.Date(), ".Rda",
    sep = ""))
```

Recognize it as JSON and parse it accordingly into R.

```r
load("data/guardian-friends-json-2021-06-18.Rda")

# Parse it into R
friends <- jsonlite::fromJSON(friends_json)

# Extract the IDs
```

```
friend_ids <- friends$ids
save(friend_ids, file = paste("data/guardian-friends-ids-", Sys.Date(), ".Rda", sep = ""))
length(friend_ids)
```

```
[1] 1072
```

Working with cursors

In the previous case, we did not reach the limit of 5.000 Twitter friends. If we exceed this threshold (i.e. if the Guardian had more than 5.000 friends), we would require additional queries. These are done via cursors: a query returns a cursor which can be used for subsequent queries. The idea is that we do not request the same data again, but starting from the results of the last query.

Let's test it with the followers of the Guardian which should be more than 5.000 and who share the same rate limits (5.000; https://developer.twitter.com/en/docs/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids)

```
url <- "https://api.twitter.com/1.1/followers/ids.json"
```

Request the first 5.000 followers of The Guardian

```
followers1_json <- my_oauth$OAuthRequest(URL = url, params = list(screen_name = "guardian",
    stringify_ids = "true"), method = "GET")
save(followers1_json, file = paste("data/guardian-followers-1-json-", Sys.Date(),
    ".Rda", sep = ""))
```

```
load("data/guardian-followers-1-json-2021-06-18.Rda")

# Parse it into R
followers <- jsonlite::fromJSON(followers1_json)
```

How many did we get?

```
length(followers$ids)
```

```
[1] 5000
```

How does the cursor look like?

```
followers$next_cursor_str
```

```
[1] "1702674173178714197"
```

Query another chunk of friends with this cursor as a parameter

```
followers2_json <- my_oauth$OAuthRequest(URL = url, params = list(screen_name = "guardian",
    stringify_ids = "true", cursor = followers$next_cursor_str), method = "GET")
save(followers2_json, file = paste("data/guardian-followers-2-json-", Sys.Date(),
    ".Rda", sep = ""))
```

Compare the two chunks:

- both have a length of 5.000
- None share the same IDs

```
load("data/guardian-followers-1-json-2021-06-18.Rda")
load("data/guardian-followers-2-json-2021-06-18.Rda")

# Parse it into R
followers1 <- jsonlite::fromJSON(followers1_json)
```

```r
followers2 <- jsonlite::fromJSON(followers2_json)

# Compare their numbers
length(followers1$ids)
```

```
[1] 5000
```

```r
length(followers2$ids)
```

```
[1] 5000
```

```r
# Do they overlap?
table(followers1$ids %in% followers2$ids)
```

```
FALSE
 5000
```

Application: Twitter API (rtweet)

Showcase

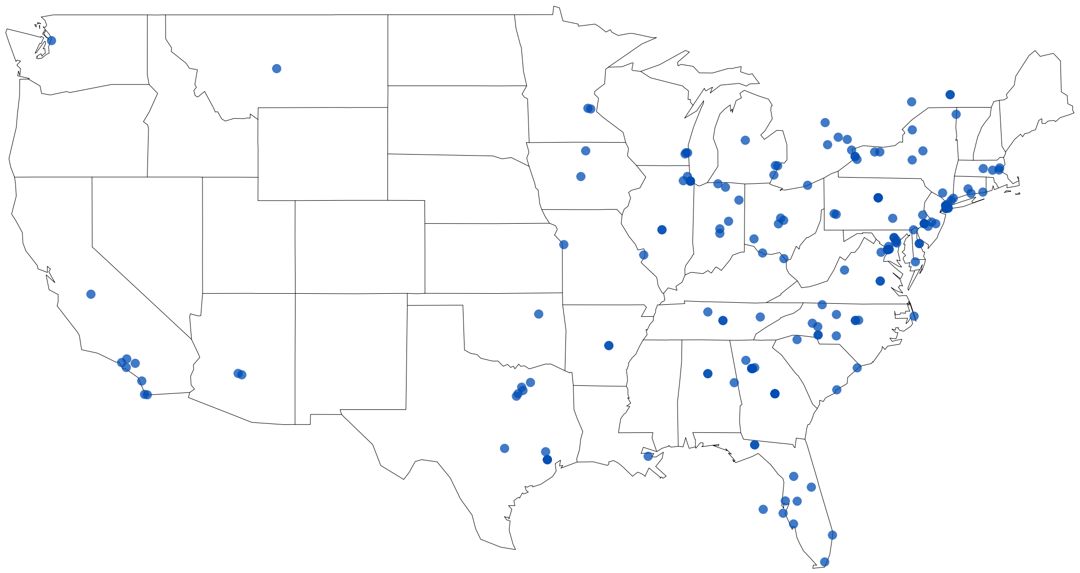
```r
library(rtweet)
## search for 10,000 tweets sent from the US
rt <- rtweet::search_tweets("lang:en", geocode = rtweet::lookup_coords("usa"), n = 10000)
save(rt, file = paste("data/tweets_us_", Sys.Date(), ".Rda", sep = ""))
```

```r
library(rtweet)
## Or: load pre-downloaded Tweets
load("data/tweets_us_2021-06-18.Rda")

## create lat/lng variables using all available tweet and profile geo-location
## data
rt <- lat_lng(rt)

## plot state boundaries
par(mar = c(0, 0, 0, 0))
maps::map("state", lwd = 0.25)

## plot lat and lng points onto state map
with(rt, points(lng, lat, pch = 20, cex = 0.75, col = rgb(0, 0.3, 0.7, 0.75)))
```



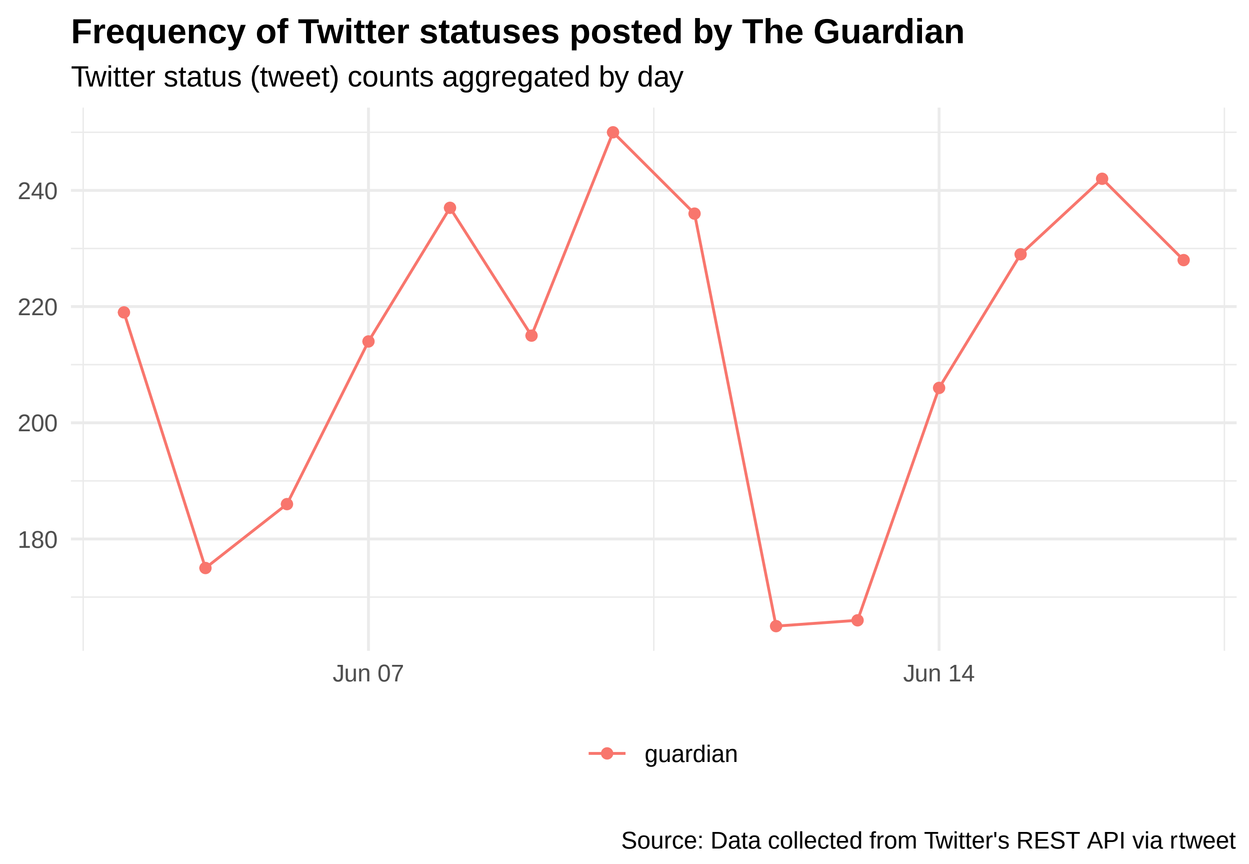
Guardian Timeline

```
tml <- rtweet::get_timelines(user = "guardian", n = 3200)
save(tml, file = paste("data/guardian-timeline-rtweet-", Sys.Date(), ".Rda", sep = ""))
```

```
load("data/guardian-timeline-rtweet-2021-06-18.Rda")
```

Plot

```
## plot the frequency of tweets for each user over time
tml %>%
    dplyr::filter(created_at > "2021-06-01") %>%
    dplyr::group_by(screen_name) %>%
    rtweet::ts_plot("days", trim = 1L) + ggplot2::geom_point() + ggplot2::theme_minimal() +
    ggplot2::theme(legend.title = ggplot2::element_blank(), legend.position = "bottom",
        plot.title = ggplot2::element_text(face = "bold")) + ggplot2::labs(x = NULL,
    y = NULL, title = "Frequency of Twitter statuses posted by The Guardian", subtitle = "Twitter statu
    caption = "\nSource: Data collected from Twitter's REST API via rtweet")
```



Extract urls

```
tml$expanded_url <- as.character(tml$urls_expanded_url)
```

Prepare for reproducibility

To allow other researchers to reproduce our results we need to provide a detailed documentation of our data set and some information that allows them to reproduce our data.

On Twitter, we have the problem that Tweets encompass personal information. People can:

- set their profile to 'protected' and, thus, dissallow further retrieval of their information.
- delete their already published Tweets and, thus, remove access to them.

We need to respect this privacy and adapt to this situation. Despite making it potentially impossible for other researchers to reproduce our results, we will provide the Tweet IDs (status_id) of our retrieved Tweets. This enables other researchers to retrieve the Tweets from the API, and Twitter itself will handle the provision of information: if e.g. a person set their profile to 'private' or deleted their Tweet, the Twitter API will not yield the information for a specific status ID.

```
tml_ids <- tml$status_id
save(tml_ids, file = paste("data/tml-publication-", Sys.Date(), ".Rda", sep = ""))
save(tml, file = paste("data/tml-private-", Sys.Date(), ".Rda", sep = ""))
```

Despite us being able to save the complete data set for our personal purpose, another researcher could use the status IDs in the following way to reproduce our results.

```
load(paste("data/tml-publication-", Sys.Date(), ".Rda", sep = ""))
statuses_repr <- rtweet::lookup_statuses(tml_ids)
save(statuses_repr, file = paste("data/tml-reproduced-", Sys.Date(), ".Rda", sep = ""))
```

```r
load("data/tml-reproduced-2021-06-18.Rda")
table(tml$status_id %in% statuses_repr$status_id)
```

```
TRUE
3200
```

In case you are evaluating Twitter accounts, this might get more tricky. One solution might be to introduce new identifiers for Twitter accounts that enables their distinction but not their identification. We may use the *digest*-package to create e.g. an SHA-2 Hash.

Hashes are a 'message digest' of text - the algorithmic product of an application of an algorithm to the text. They always have the same length and they only work one way: you can transform text into a hashed form, but you cannot recreate the text from a hashed version of itself. They are commonly used to verify e.g. downloads and to save passwords. As they are unique, we can use them to hash the user IDs. The result are distinct identifier which are able to distinguish between accounts but someone who obtained these can not make conclusions about an individual Twitter account.

```r
# Get e.g. the retweeters of one status of The Guardian.
accounts <- rtweet::get_retweeters(status_id = "1254605442124255234")
save(accounts, file = paste("data/retweeters-1254605442124255234-", Sys.Date(), ".Rda",
    sep = ""))
```

```r
# Load the previously downloaded Retweeters
load("data/retweeters-1254605442124255234-2021-06-18.Rda")

# Get additional information
accounts_info <- rtweet::lookup_users(accounts$user_id)

# Extract only information we are interested in (account information)
info <- accounts_info[, c("user_id", "protected", "followers_count", "friends_count",
    "statuses_count", "favourites_count", "account_created_at", "verified")]
head(info)
```

```
# A tibble: 6 x 8
  user_id protected followers_count friends_count statuses_count
  <chr>   <lgl>               <int>         <int>          <int>
1 471290~ FALSE                 524          1051          81529
2 254745~ FALSE                 338           505          25816
3 760776~ FALSE                 217           356          42524
4 104800~ FALSE                 152           352          54061
5 387165~ FALSE                  11            77            160
6 121093~ FALSE                 335          1209           1259
# ... with 3 more variables: favourites_count <int>, account_created_at <dttm>,
#   verified <lgl>
```

```r
# Hash the IDs of the Twitter users to anonymize them.
info$user_id <- sapply(info$user_id, digest::digest, algo = "sha2")
table(duplicated(info$user_id))
```

```
FALSE
   41
```

```r
head(info)
```

```
# A tibble: 6 x 8
  user_id protected followers_count friends_count statuses_count
  <chr>   <lgl>               <int>         <int>          <int>
1 ce4a53~ FALSE                 524          1051          81529
```

```
2 b5bfc8~ FALSE                  338          505          25816
3 0bec7e~ FALSE                  217          356          42524
4 2afab0~ FALSE                  152          352          54061
5 15d07c~ FALSE                   11           77            160
6 9a2f0b~ FALSE                  335         1209           1259
# ... with 3 more variables: favourites_count <int>, account_created_at <dttm>,
#   verified <lgl>
```

```r
save(info, file = paste("data/retweeters-1254605442124255234-", Sys.Date(), "-anonymized.Rda",
    sep = ""))
```