Exercise 5: APIs

Philipp Kling

19.06.2020

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,root.dir = "/home/philipp/Documents/fds-2020-exercise")
rm(list = ls())
```

Preparation

Installation of required packages

```
toInstall <- c("httr", "jsonlite", "yaml", "GuardianR", "maps", "dplyr", "ggplot2",
    "digest")
install.packages(pkgs = toInstall)
```

Load some

```
library(httr)
library(jsonlite)
```

```r
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

```r
guardian_creds <- yaml::yaml.load_file("/home/philipp/Documents/keys/guardian_api/api_key1.yaml")
```

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()

```
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=',
    # 'DEVKEYPLACEHOLDER', 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**

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

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

```
articles_get <- httr::GET(query_url)
save(articles_get, file = paste("./data/ex5/guardianapi-own-debate-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/ex5/guardianapi-own-debate-notbrexit-2020-04-30.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\":2148485,\"startIndex\":1,\"pa
```

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         politics/2016/mar/21/david-cameron-backs-chancellor-despite-budget-triggering-tory-turmoil
2                                         politics/2012/oct/15/alex-salmond-scotland-referendum-deal
3                               books/2017/jul/16/fall-down-7-times-get-up-8-naoki-higashida-review-autism
4                            football/2017/jul/07/gold-cup-2017-predictions-usa-mexico-costa-rica-football
5 world/2017/jul/15/stream-of-floating-bodies-near-mosul-raises-fears-of-reprisals-by-iraqi-militias
6                             football/2017/jul/06/romelu-lukaku-bargain-everton-manchester-united-chelsea
     type sectionId sectionName   webPublicationDate
1 article  politics    Politics 2016-03-21T12:56:43Z
2 article  politics    Politics 2012-10-15T19:03:59Z
3 article     books       Books 2017-07-16T06:00:13Z
4 article  football    Football 2017-07-07T09:00:08Z
5 article     world  World news 2017-07-15T08:00:01Z
6 article  football    Football 2017-07-06T18:00:04Z
                                                                    webTitle
1                 David Cameron backs chancellor despite Tory turmoil over budget
2              Alex Salmond hails historic day for Scotland after referendum deal
3             Fall Down 7 Times Get Up 8 review - a window on the world of autism
4         Gold Cup picks: USA to tip under-strength Mexico and in-form Costa Rica
5 Stream of floating bodies near Mosul raises fears of reprisals by Iraqi militias
6           Romelu Lukaku: £75m is never a bargain but Everton striker is worth it

1         https://www.theguardian.com/politics/2016/mar/21/david-cameron-backs-chancellor-despite-budge
2                                         https://www.theguardian.com/politics/2012/oct/15/alex-salmond-
3                               https://www.theguardian.com/books/2017/jul/16/fall-down-7-times-get-up-8-naok
4                            https://www.theguardian.com/football/2017/jul/07/gold-cup-2017-predictions-usa-m
5 https://www.theguardian.com/world/2017/jul/15/stream-of-floating-bodies-near-mosul-raises-fears-of-re
```

```
6                        https://www.theguardian.com/football/2017/jul/06/romelu-lukaku-bargain-everton-m

1         https://content.guardianapis.com/politics/2016/mar/21/david-cameron-backs-chancellor-despite-b
2                                 https://content.guardianapis.com/politics/2012/oct/15/alex-sal
3                        https://content.guardianapis.com/books/2017/jul/16/fall-down-7-times-get-up-8-
4                     https://content.guardianapis.com/football/2017/jul/07/gold-cup-2017-predictions-u
5 https://content.guardianapis.com/world/2017/jul/15/stream-of-floating-bodies-near-mosul-raises-fears-o
6                        https://content.guardianapis.com/football/2017/jul/06/romelu-lukaku-bargain-ever
  wordcount isHosted     pillarId pillarName
1       901    FALSE  pillar/news       News
2      1492    FALSE  pillar/news       News
3      1232    FALSE  pillar/arts       Arts
4      1633    FALSE pillar/sport      Sport
5      1037    FALSE  pillar/news       News
6      1013    FALSE pillar/sport      Sport
```

Start using 'GuardianR'

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.

```r
gres <- GuardianR::get_guardian(keywords = c("uk"), from.date = "2020-04-20",
    to.date = "2020-04-21", api.key = guardian_creds$api_key)
save(gres, file = paste("../data/ex5/guardianapi-GuardianR-uknews-2020-04-20--2020-04-21--date-",
    Sys.Date(), ".Rda", sep = ""))
```

```r
load("./data/ex5/guardianapi-GuardianR-uknews-2020-04-20--2020-04-21--date-2020-04-30.Rda")
head(gres)
```

```
                                                                              id
1          education/2020/apr/20/thousands-urge-uk-government-to-keep-schools-closed
2 business/2020/apr/20/uk-youth-employment-prospects-crumbling-in-coronavirus-crisis
  sectionId sectionName   webPublicationDate
1 education   Education 2020-04-20T15:54:59Z
2  business    Business 2020-04-20T09:05:38Z
                                                       webTitle
1             Thousands urge UK government to keep schools closed
2 'UK youth employment prospects crumbling' in coronavirus crisis

1          https://www.theguardian.com/education/2020/apr/20/thousands-urge-uk-government-to-keep-schoo
2 https://www.theguardian.com/business/2020/apr/20/uk-youth-employment-prospects-crumbling-in-coronavir

1          https://content.guardianapis.com/education/2020/apr/20/thousands-urge-uk-government-to-keep-s
2 https://content.guardianapis.com/business/2020/apr/20/uk-youth-employment-prospects-crumbling-in-coron
  newspaperPageNumber
1                <NA>
2                  29
                                                                                       trailText
1  Education workers say enforcing social distancing among primary pupils is â€˜almost impossibleâ€
2             Employers warn of grim summer for graduates as firms, big and small, trim recruitment
                                                       headline
1             Thousands urge UK government to keep schools closed
2 'UK youth employment prospects crumbling' in coronavirus crisis
  showInRelatedContent         lastModified hasStoryPackage score
```

```
1                  true 2020-04-20T17:43:54Z              NA    NA
2                  true 2020-04-20T19:55:02Z              NA    NA

1 <p>Education workers say enforcing social distancing among primary pupils is â€~almost impossibleâ€</p
2            <p>Employers warn of grim summer for graduates as firms, big and small, trim recruitment<
              shortUrl wordcount commentable allowUgc isPremoderated
1 https://gu.com/p/dy74y       681        <NA>       NA          false
2 https://gu.com/p/dkf69       696        <NA>       NA          false
                             byline     publication newspaperEditionDate
1 Sally Weale Education correspondent theguardian.com                 <NA>
2                       Jasper Jolly    The Guardian 2020-04-21T00:00:00Z
  shouldHideAdverts liveBloggingNow commentCloseDate
1             false            <NA>             <NA>
2             false            <NA>             <NA>

1 <p>A letter from an NHS nurse urging the government to keep schools closed because of the health risk
```

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

```r
install.packages("ROauth")
```

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("/home/philipp/Documents/keys/twitter_api/api_feedanalysis_159916

# 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.
    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

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

Request the first 5.000 friends of The Guardian

```
friends_json <- my_oauth$OAuthRequest(URL = url, params = list(screen_name = "guardian",
    stringify_ids = "true"), method = "GET")
save(friends_json, file = paste("./data/ex5/guardian-friends-json-", Sys.Date(),
    ".Rda", sep = ""))
```

Recognize it as JSON and parse it accordingly into R.

```
load("./data/ex5/guardian-friends-json-2020-04-30.Rda")

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

# Extract the IDs
friend_ids <- friends$ids
save(friend_ids, file = paste("./data/ex5/guardian-friends-ids-", Sys.Date(),
    ".Rda", sep = ""))
length(friend_ids)
```

```
[1] 1085
```

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/ex5/guardian-followers-1-json-",
    Sys.Date(), ".Rda", sep = ""))
```

```
load("./data/ex5/guardian-followers-1-json-2020-04-30.Rda")
```

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

How many did we get?

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

```
[1] 5000
```

How does the cursor look like?

```r
followers$next_cursor_str
```

```
[1] "1665334458923733656"
```

Query another chunk of friends with this cursor as a parameter

```r
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/ex5/guardian-followers-2-json-",
    Sys.Date(), ".Rda", sep = ""))
```

Compare the two chunks:

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

```r
load("./data/ex5/guardian-followers-1-json-2020-04-30.Rda")
load("./data/ex5/guardian-followers-2-json-2020-04-30.Rda")

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

Installation

```r
install.packages("rtweet")
```

Showcase

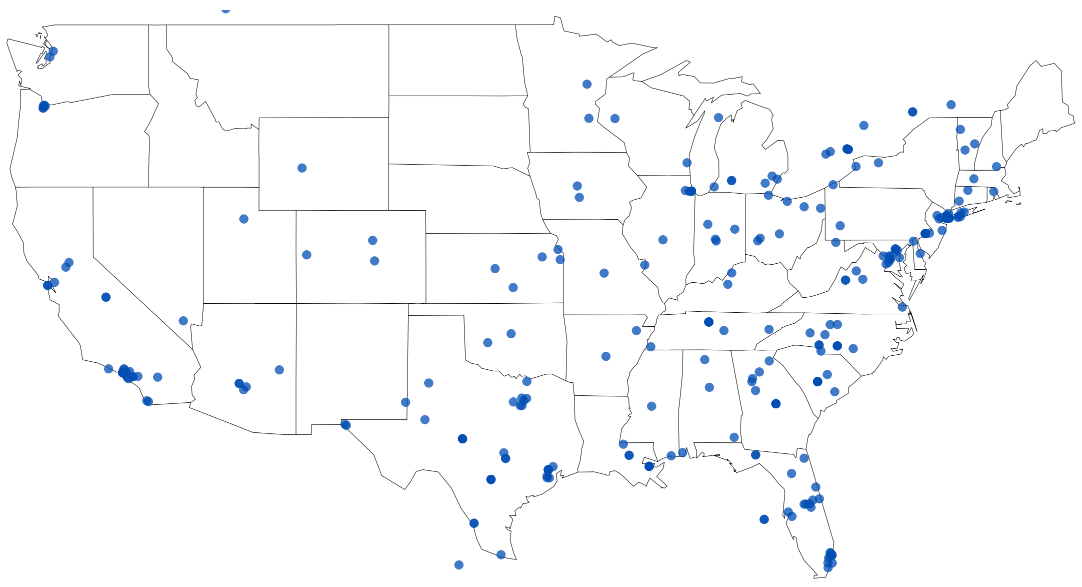
```r
library(rtweet)
## search for 10,000 tweets sent from the US
rt <- rtweet::search_tweets("lang:en", geocode = rtweet::lookup_coords("usa"),
    n = 10000)
save(rt, file = paste("./data/ex5/tweets_us_", , Sys.Date(), ".Rda", sep = ""))
```

```r
library(rtweet)
## Or: load pre-downloaded Tweets
load("./data/ex5/tweets_us_20200422.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)))
```



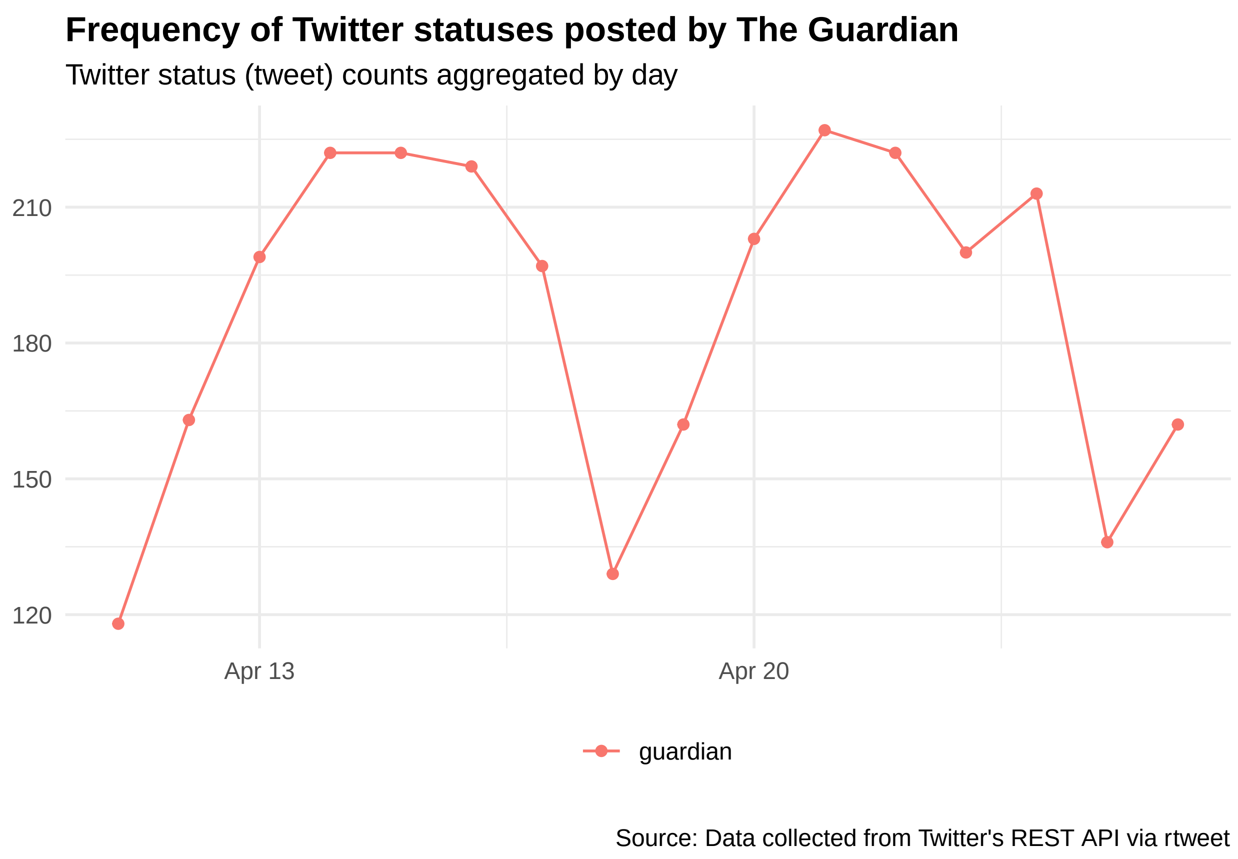
Guardian Timeline

```r
tml <- rtweet::get_timelines(user = "guardian", n = 3200)
save(tml, file = paste("./data/ex5/guardian-timeline-rtweet-", Sys.Date(), ".Rda",
    sep = ""))
```

```r
# setwd('./ex5')
load("./data/ex5/guardian-timeline-rtweet-2020-04-27.Rda")
```

Plot

```r
## plot the frequency of tweets for each user over time
tml %>% dplyr::filter(created_at > "2017-10-29") %>% 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 status (tweet) counts aggregated by day", caption = "\nSource: Data collected f
```



Extract urls

```r
tml$expanded_url <- unlist(tml$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/ex5/tml-publication-", Sys.Date(), ".Rda",
    sep = ""))
save(tml, file = paste("./data/ex5/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/ex5/tml-publication-", Sys.Date(), ".Rda", sep = ""))
statuses_repr <- rtweet::lookup_statuses(tml_ids)
save(statuses_repr, file = paste("../data/ex5/tml-reproduced-", Sys.Date(),
    ".Rda", sep = ""))
```

```
load("./data/ex5/tml-reproduced-2020-04-30.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.

```
# Get e.g. the retweeters of one status of The Guardian.
accounts <- rtweet::get_retweeters(status_id = "1254605442124255234")
save(accounts, file = paste("./data/ex5/retweeters-1254605442124255234-", Sys.Date(),
    ".Rda", sep = ""))
```

```
# Load the previously downloaded Retweeters
load("./data/ex5/retweeters-1254605442124255234-2020-04-30.Rda")

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

# Extract only information we are interested in (account information)
```

```r
info <- accounts_info[, c("user_id", "protected", "followers_count", "friends_count",
    "statuses_count", "favourites_count", "account_created_at", "verified")]
head(info)

# 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))
head(info)
save(info, file = paste("./data/ex5/retweeters-1254605442124255234-", Sys.Date(),
    "-anonymized.Rda", sep = ""))
```