Automatic Sampling and Analysis of YouTubeData

Collecting Data With the tuber Package for R

Julian Kohne
Johannes Breuer
M. Rohangis Mohseni

2021-02-24

The `tuber` Package

With the `tuber` package you can access the *YouTube* API via `R`. The author of the package has written a short vignette to introduce some of its main functions.

Note: You can also access/open this vignette in `R`/*RStudio* with the following command:

```
vignette("tuber-ex")
```

Essentially, there are three main units of analysis for which you can access data with the `tuber` package:

1. Videos
2. Channels
3. Playlists

In this workshop, we will focus on accessing and working with data for individual **videos**.

Setup

```
install.packages("tuber")
install.packages("tidyverse")
library(tuber)
library(tidyverse)
```

If your version of `R` is < 4.0.0, you should also run the following command before collecting data from the *YouTube* API:

```
options(stringsAsFactors = FALSE)
```

Excursus: Keep Your Secrets Secret

If you save your API credentials in an `R` script, anyone who has that script can potentially use your API key. This can affect your quota and might even cause costs for you (if you have billing enabled for your project).

Sidenote: If you use environment variables in `R` and store your app ID and secret there, your credentials won't be shared if you share your scripts. However, the information is still stored locally in a plain text file. You can run `usethis::edit_r_environ()` to see its contents.

Excursus: Keep Your Secrets Secret

Excursus: Keep Your Secrets Secret

One way of keeping your credentials safe(r) is using the `keyring` package.

```r
install.packages("keyring")
library(keyring)
```

Excursus: The `keyring` Package

To store your credentials in an encrypted password-protected keyring you first need to create a new keyring.

```
keyring_create("YouTube_API")
```

You will then be prompted to set a password for this keyring. As always: This password should be easy to remember (for you) but hard to guess (for anyone else). You might want to write it down or store it in your password manager. In the next step, you can store your *YouTube* app secret in the keyring (*Note*: You can also store your app ID the same way).

```
key_set("app_secret",
        keyring ="YouTube_API")
```

When you are prompted to enter the password this time you should enter your app secret. After that - as well as whenever you unlock a keyring to use the credentials stored inside - you should lock the keyring again.

```
keyring_lock("YouTube_API")
```

Authenticate Your App

```
yt_oauth(app_id = "my_app_id", # paste your app ID here
         app_secret = "my_app_secret", # paste your app secret here
         token="")
```

or, if you stored your app secret in a keyring

```
yt_oauth(app_id = "my_app_id",  # paste your app ID here
         app_secret = key_get("app_secret", keyring ="YouTube_API"),
         token="")
```

Note: If you use a keyring, you will be prompted to enter the password for that keyring in `R`/*RStudio*.

Authentication

When running the `yt_oauth()` function, there will be a prompt in the console asking you whether you want to save the access token in a file. If you select "Yes", `tuber` will create a local file (in your working directory) to cache OAuth access credentials between `R` sessions. The token stored in this file will be valid for one hour (and can be automatically refreshed). Doing this makes sense if you know you will be running multiple `R` sessions that need your credentials (example: knitting an `RMarkdown` document).

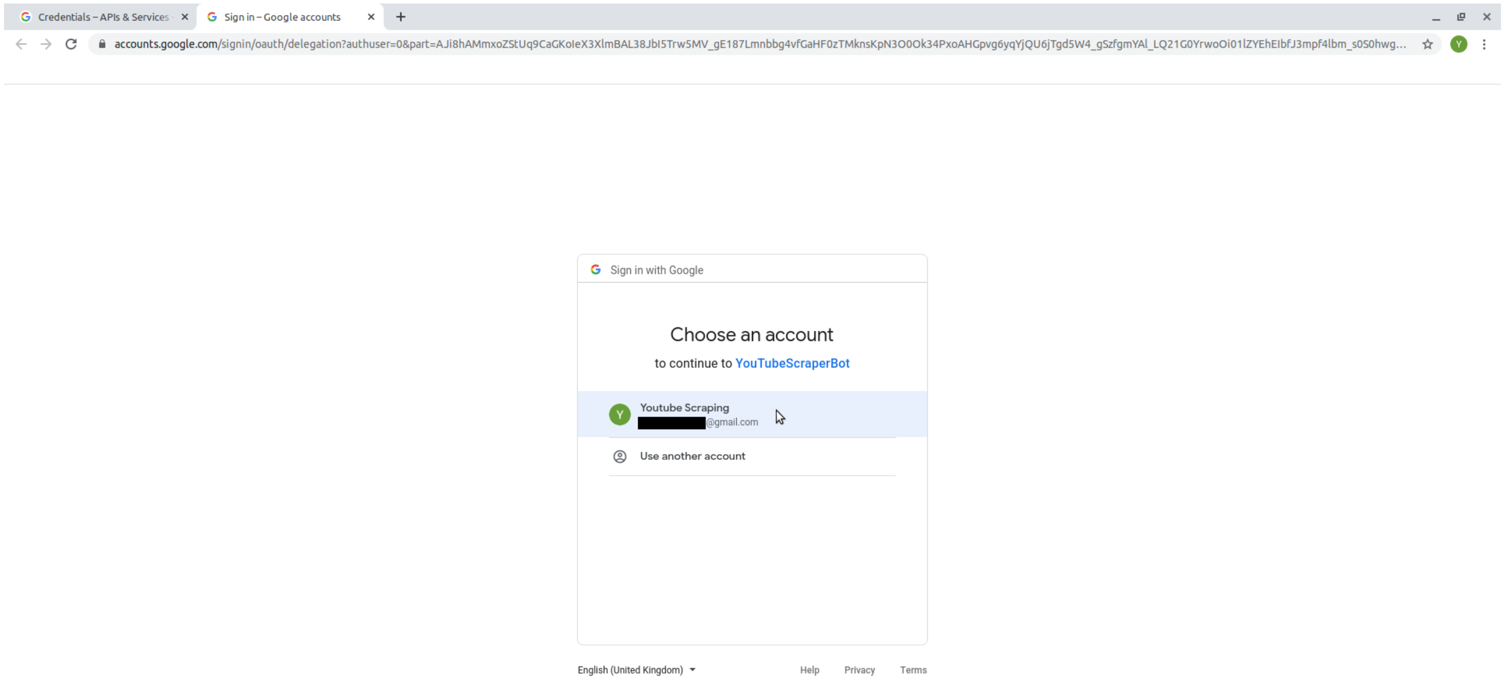
Authentication

Similar to the app key and secret, you should never share the `.httr-oauth` as the file contains a list with your app ID and the access token. Hence, it is usually ok (and safer) to select "No" when asked whether to store a local `.httr-oauth` file. If you have created a `.httr-oauth` file and want to check its contents, you can do so as follows in `R`:

```
# read the .httr-oauth file
token <- readRDS("~/.httr-oauth") # this assumes the file is stored
# app ID
token$`0123456789abcdefghij`$app$key # the token object is a list of
# app secret
token$`0123456789abcdefghij`$app$secret
# If you are unsure which token is the correct one, you can check ou
token$`0123456789abcdefghij`$app$appname
```

App Authorization (1)

After making your choice regarding the `.httr-oauth` file, a browser window should open, prompting you to log in with your *Google* account and to give permissions.



Credentials – APIs & Services

Sign in – Google accounts

accounts.google.com/signin/oauth/delegation?authuser=0&part=AJi8hAMmxoZStUq9CaGKoIeX3XlmBAL38JbI5Trw5MV_gE187Lmnbbg4vfGaHF0zTMknsKpN3O0Ok34PxoAHGpvg6yqYjQU6jTgd5W4_gSzfgmYAl_LQ21G0YrwoOi01lZYEhEIbfJ3mpf4lbm_s0S0hwg...

Sign in with Google

Choose an account

to continue to YouTubeScraperBot

Youtube Scraping
[redacted]@gmail.com

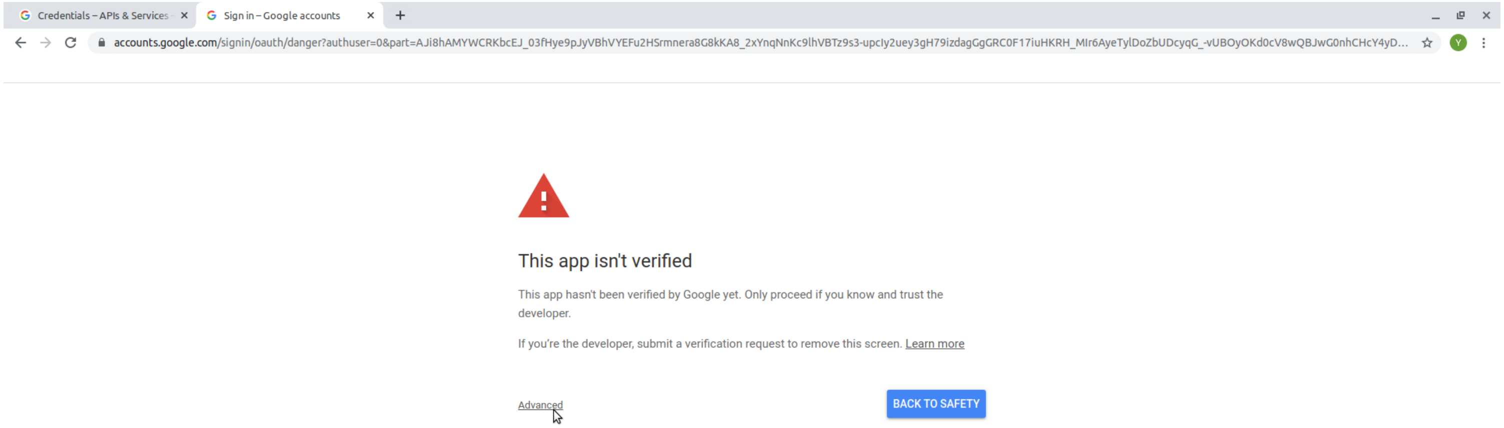
Use another account

English (United Kingdom)

Help Privacy Terms

App Authorization (2)

Google will mark the app as unsecure as it has not been verified. However, since you created the app and presumably are its sole user, you can go ahead and trust yourself: Click "Extended".



Credentials – APIs & Services
Sign in – Google accounts

accounts.google.com/signin/oauth/danger?authuser=0&part=AJi8hAMYWCRKbcEJ_03fHye9pJyVBhVYEFu2HSrmnera8G8kKA8_2xYnqNnKc9lhVBTz9s3-upcly2uey3gH79izdagGgGRC0F17iuHKRH_MIr6AyeTylDoZbUDcyqG_-vUBOyOKd0cV8wQBJwG0nhCHcY4yD...

This app isn't verified

This app hasn't been verified by Google yet. Only proceed if you know and trust the developer.

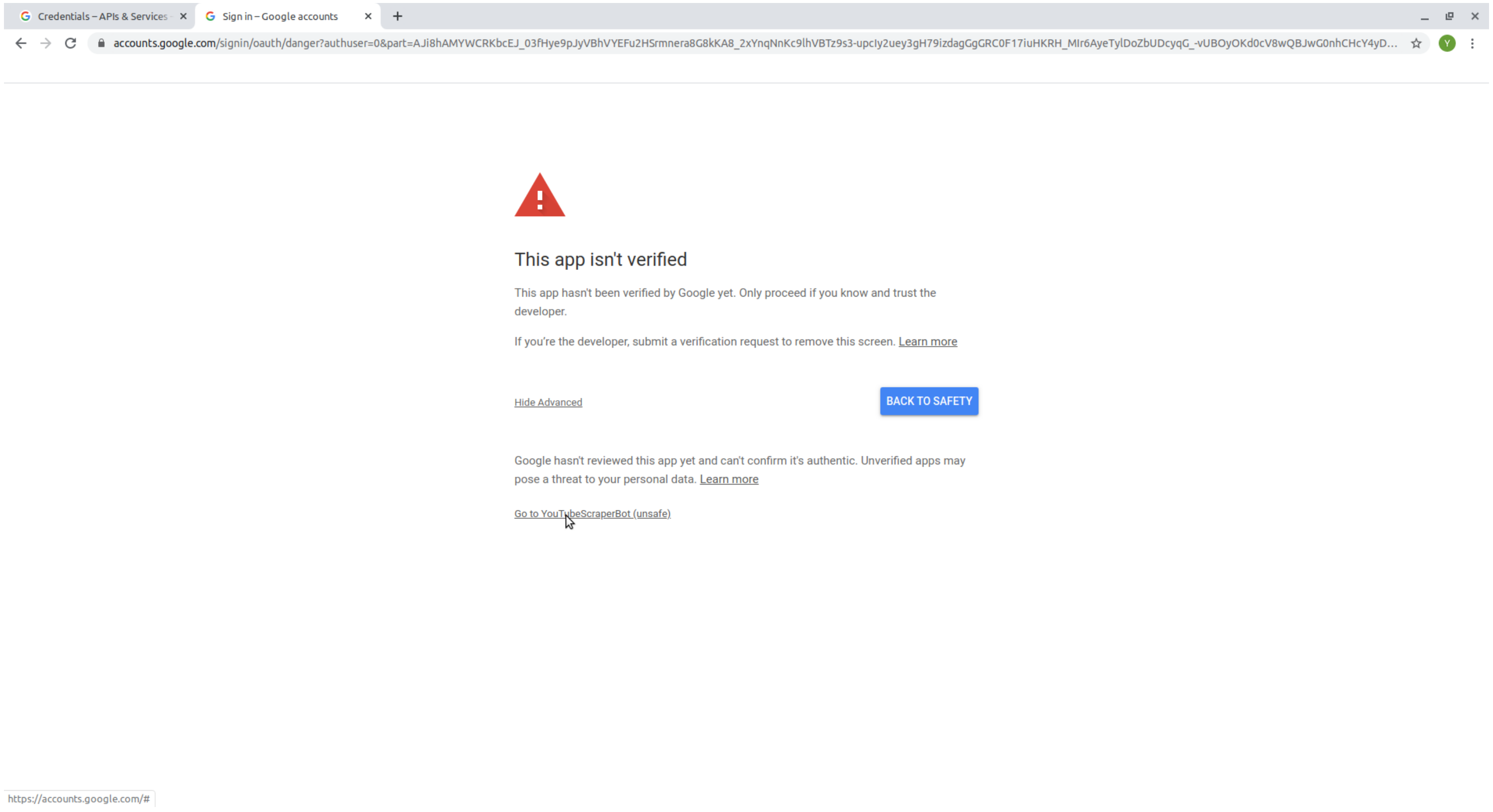
If you're the developer, submit a verification request to remove this screen. Learn more

Advanced

BACK TO SAFETY

App Authorization (3)

Click "Go to Name_of_your_app(unsafe)".


Credentials – APIs & Services
Sign in – Google accounts
accounts.google.com/signin/oauth/danger?authuser=0&part=AJi8hAMYWCRKbcEJ_03fHye9pJyVBhVYEFu2HSrmnera8G8kKA8_2xYnqNnKc9lhVBTz9s3-upcly2uey3gH79izdagGgGRC0F17iuHKRH_MIr6AyeTylDoZbUDcyqG_-vUBOyOKd0cV8wQBJwG0nhCHcY4yD...

This app isn't verified

This app hasn't been verified by Google yet. Only proceed if you know and trust the developer.

If you're the developer, submit a verification request to remove this screen. Learn more

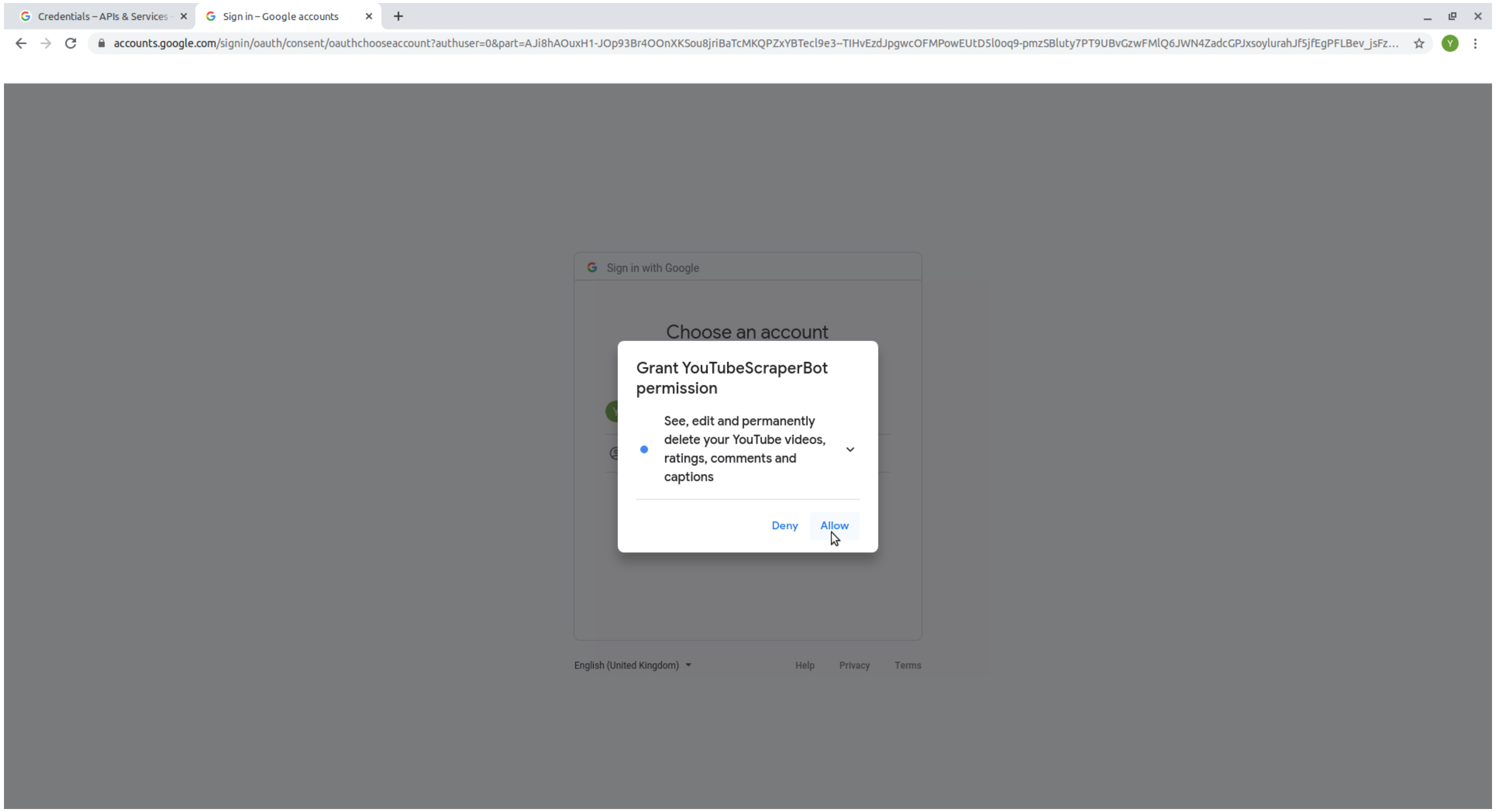
Hide Advanced
BACK TO SAFETY

Google hasn't reviewed this app yet and can't confirm it's authentic. Unverified apps may pose a threat to your personal data. Learn more

Go to YouTubeScraperBot (unsafe)
https://accounts.google.com/#

App Authorization (4)

Click "Allow" to grant the app the required permissions this your account.


Credentials – APIs & Services
Sign in – Google accounts
accounts.google.com/signin/oauth/consent/oauthchooseaccount?authuser=0&part=AJi8hAOuxH1-JOp93Br4OOnXKSou8jriBaTcMKQPZxYBTecl9e3--TIHvEzdJpgwcOFMPowEUtD5l0oq9-pmzSBluty7PT9UBvGzwFMlQ6JWN4ZadcGPJxsoylurahJf5jfEgPFLBev_jsFz...
Sign in with Google
Choose an account
English (United Kingdom)
Help
Privacy
Terms

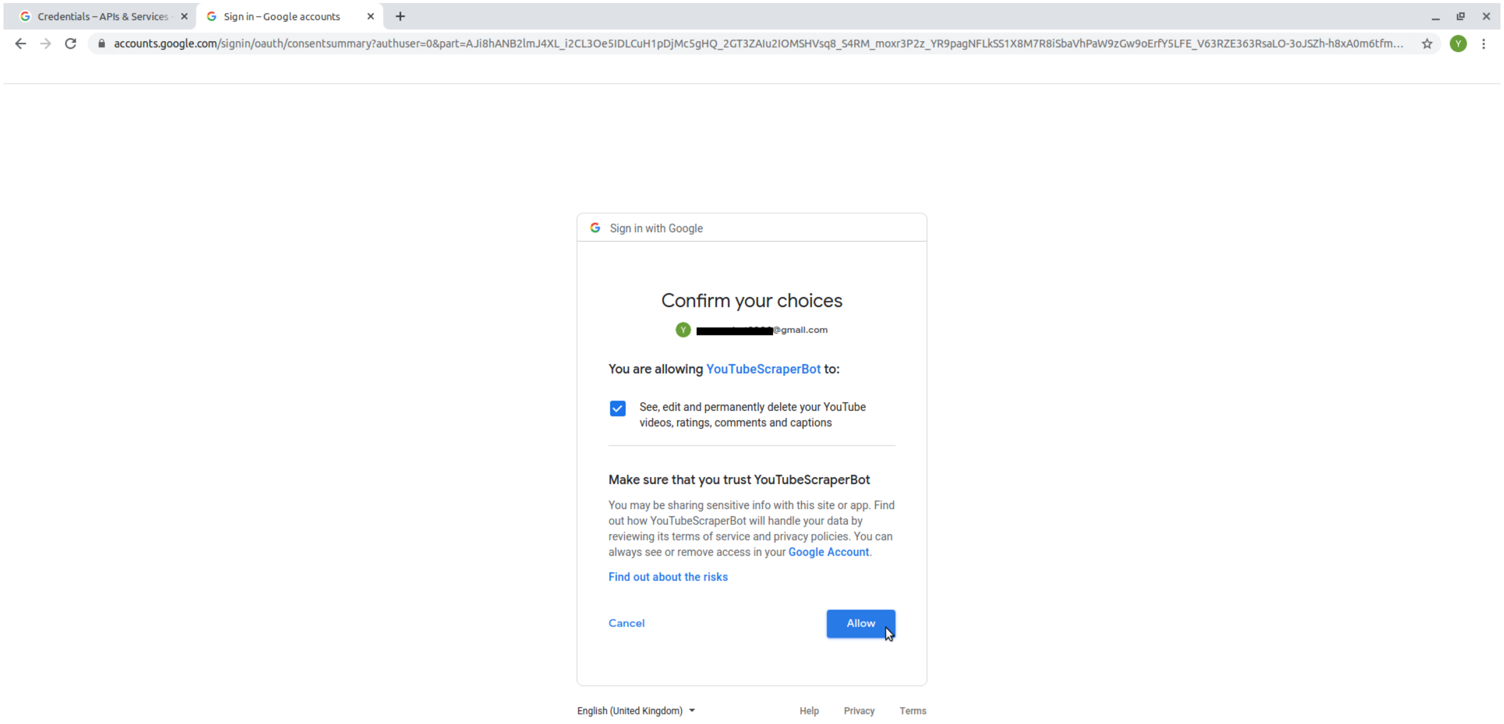
Grant YouTubeScraperBot permission

See, edit and permanently delete your YouTube videos, ratings, comments and captions

Deny Allow

App Authorization (5)

Click "Allow" again to confirm your choice


Credentials – APIs & Services
Sign in – Google accounts
accounts.google.com/signin/oauth/consentsummary?authuser=0&part=AJi8hANB2lmJ4XL_i2CL3Oe5IDLCuH1pDjMc5gHQ_2GT3ZAIu2IOMSHVsq8_S4RM_moxr3P2z_YR9pagNFLkSS1X8M7R8iSbaVhPaW9zGw9oErfY5LFE_V63RZE363RsaLO-3oJSZh-h8xA0m6tfm...
Sign in with Google

Confirm your choices

[redacted]@gmail.com

You are allowing YouTubeScraperBot to:

- [✓] See, edit and permanently delete your YouTube videos, ratings, comments and captions

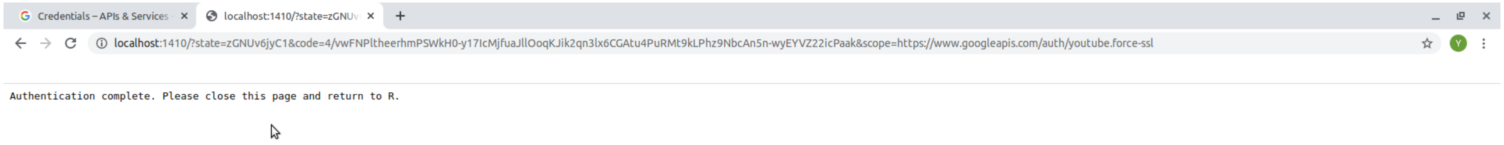
Make sure that you trust YouTubeScraperBot

You may be sharing sensitive info with this site or app. Find out how YouTubeScraperBot will handle your data by reviewing its terms of service and privacy policies. You can always see or remove access in your Google Account.

Find out about the risks

App Authorization (6)

After that, you can close the browser tab/window and return to R/*RStudio*.



localhost:1410/?state=zGNUv6jyC1&code=4/vwFNPltheerhmPSWkH0-y17IcMjfuaJllOoqKJik2qn3lx6CGAtu4PuRMt9kLPhz9NbcAn5n-wyEYVZ22icPaak&scope=https://www.googleapis.com/auth/youtube.force-ssl

```
Authentication complete. Please close this page and return to R.
```

Channel Statistics (1)

After authorizing your app, you can test if everything works by getting some statistics for a channel. You can find the channel ID by clicking on the user name for a video on the *YouTube* website (the channel ID is at the end of the URL, after "/channel/"), by using the *Commentpicker* tool or directly through the API (see this Stackoverflow discussion).

```
hadley_stats <- get_channel_stats("UCxOhDvtaoXDAB336AolWs3A")
```

```
## Channel Title: Hadley Wickham
## No. of Views: 67692
## No. of Subscribers: 4300
## No. of Videos: 6
```

Note: If you want additional data on channels/video creators and their performance on *YouTube*, you can also check out the statistics provided by *Socialblade*.

Channel Statistics (2)

`get_channel_stats()` returns a list which can then be wrangled into a dataframe (or a tibble as in the code below).

```
hadley_stats_df <- hadley_stats$statistics %>%
  as_tibble() %>%
  mutate(
    across(where(is.character), as.numeric),
    channel_id = hadley_stats$id,
    channel_name = hadley_stats$snippet$title,
    channel_published = hadley_stats$snippet$publishedAt)

hadley_stats_df
```

```
## # A tibble: 1 x 7
##   viewCount subscriberCount hiddenSubscribe~ videoCount channel_id channel_name
##       <dbl>           <dbl> <lgl>                 <dbl> <chr>      <chr>
## 1     67692            4300 FALSE                     6 UCxOhDvta~ Hadley Wick~
## # ... with 1 more variable: channel_published <chr>
```

Video Statistics

To get statistics for a video you need its ID. The video IDs are the characters after the "v=" URL parameter. For example, for the video https://www.youtube.com/watch?v=DcJFdCmN98s the ID is "DcJFdCmN98s".

```
dayum_stats <- get_stats("DcJFdCmN98s")

dayum_stats_df <- dayum_stats %>%
  as_tibble() %>%
  mutate(across(c(2:6), as.numeric))

dayum_stats_df
```

```
## # A tibble: 1 x 6
##   id          viewCount likeCount dislikeCount favoriteCount commentCount
##   <chr>           <dbl>     <dbl>        <dbl>         <dbl>        <dbl>
## 1 DcJFdCmN98s  41650484    594938         7279             0        51458
```

Viewer Comments

There are two `tuber` functions for collecting viewer comments for specific videos (both return a dataframe by default):

1. `get_comment_threads()` collects comments for a video. By default, it collects the 100 most recent comments. The number of comments can be changed with the argument `max_results` (needs to be >= 20). If the number is > 100, the function fetches all results (see `?get_comment_threads`). **NB:** This function does not collect replies to comments. Also, when we tested it, the resulting dataframe only contained repetitions of the 100 most recent comments when `max_results` > 100.

2. `get_all_comments()` collects all comments for a video, including replies to comments. **NB:** This function only collects up to 5 replies per comment (the related issue in the `tuber` GitHub repo is still open). Depending on the number of comments for a video, running this function can take some time and might also deplete your API quota limit (more on that later).

Our recommendation is that if you want more than the most recent 100 comments, you should use `get_all_comments()`, even if you do not want to analyze replies to comments (you can simply filter those out later on).

Exercise time 🏋️ 💪 🏃 🚴

Solutions

Collecting Comments for Multiple Videos

If you want to collect comments for multiple videos, you need a vector with all video IDs (in the code example below, this vector is named `video_ids`). If you have that, you can use `map_df()` from the `purrr` package to iterate over video IDs and create a combined dataframe (of course, you could also do this with a for-loop).

```
comments <- purrr::map_df(.x = video_ids,
                          .f = get_all_comments)
```

Important notice: Be aware that your daily API quota limit very likely won't allow you to collect all comments for videos with a high number of comments (more on the API quota limit in a bit).

Searching Video IDs with `tuber`

`tuber` provides the function `yt_search()` for using the API to search *YouTube*. However, using this function is extremely costly in terms of API queries/quotas.

A single search query with the `yt_search()` function can easily exceed your daily quota limit as, by default, it returns data for all search results (you can check the number of results for a search query via the *YouTube* Data API Explorer). Hence, if you want to use the `yt_search()` function, we would strongly recommend setting the `get_all` argument to `FALSE`. This returns up to 50 results.

```
yt_search(term = "tidyverse",
          get_all = F)
```

Multiple search result pages (1)

If you want to use `yt_search()` and want more than 50 search results, you can use the `page_token` argument to get more than one page of results.

```
search_p1 <- yt_search(term = "search term",
                       simplify = F,
                       get_all = F)

page_token <-search_p1$nextPageToken

search_p2 <- yt_search(term = "search term",
                       get_all = F,
                       page_token = page_token)
```

Multiple search result pages (2)

```r
# turn search_p1 from a list into a dataframe
search_p1_snippet <- lapply(search_p1$items, function(x) unlist(x$snippet))
search_p1_ids <- lapply(search_p1$items, function(x) unlist(x$id))
ids_p1_df<- purrr::map_dfr(search_p1_ids, bind_rows)
search_p1_df <- purrr::map_dfr(search_p1_snippet, bind_rows)

# combine search results from different pages into one dataframe
search_results <- bind_rows(search_p1_df, search_p2) %>%
  mutate(videoId = case_when(is.na(videoId) ~ as.character(video_id),
                             TRUE ~ as.character(videoId))) %>%
  select(-video_id) %>%
  relocate(videoId)
```

Example: Two pages of search results

```r
search_p1 <- yt_search(term = "weezer",
                       simplify = F,
                       get_all = F)

page_token <-search_p1$nextPageToken

search_p2 <- yt_search(term = "weezer",
                       get_all = F,
                       page_token = page_token)

search_p1_snippet <- lapply(search_p1$items, function(x) unlist(x$snippet))
search_p1_ids <- lapply(search_p1$items, function(x) unlist(x$id))
ids_p1_df<- purrr::map_dfr(search_p1_ids, bind_rows) %>%
  select(videoId)
search_p1_df <- purrr::map_dfr(search_p1_snippet, bind_rows) %>%
  bind_cols(ids_p1_df)
search_results <- bind_rows(search_p1_df, search_p2) %>%
  mutate(videoId = case_when(is.na(videoId) ~ as.character(video_id),
                             TRUE ~ as.character(videoId))) %>%
  select(-video_id) %>%
  relocate(videoId)

glimpse(search_results)
```

Example: Two pages of search results

```
## Rows: 100
## Columns: 17
## $ videoId                   <chr> "ENXvZ9YRjbo", "AGPdXYG1msg", "erG5rgNYSd...
## $ publishedAt               <chr> "2009-06-17T05:26:28Z", "2021-01-21T05:01...
## $ channelId                 <chr> "UCjXfz0RBwagVrDo7uL6liqg", "UC7JDBUzkcwR...
## $ title                     <chr> "Weezer - Say It Ain&#39;t So (Official M...
## $ description               <chr> "Music video by Weezer performing Say It ...
## $ thumbnails.default.url    <chr> "https://i.ytimg.com/vi/ENXvZ9YRjbo/defau...
## $ thumbnails.default.width  <chr> "120", "120", "120", "120", "120", "120",...
## $ thumbnails.default.height <chr> "90", "90", "90", "90", "90", "90", "90",...
## $ thumbnails.medium.url     <chr> "https://i.ytimg.com/vi/ENXvZ9YRjbo/mqdef...
## $ thumbnails.medium.width   <chr> "320", "320", "320", "320", "320", "320",...
## $ thumbnails.medium.height  <chr> "180", "180", "180", "180", "180", "180",...
## $ thumbnails.high.url       <chr> "https://i.ytimg.com/vi/ENXvZ9YRjbo/hqdef...
## $ thumbnails.high.width     <chr> "480", "480", "480", "480", "480", "480",...
## $ thumbnails.high.height    <chr> "360", "360", "360", "360", "360", "360",...
## $ channelTitle              <chr> "WeezerVEVO", "weezer", "WeezerVEVO", "We...
## $ liveBroadcastContent      <chr> "none", "none", "none", "none", "none", "...
## $ publishTime               <chr> "2009-06-17T05:26:28Z", "2021-01-21T05:01...
```

Searching Video IDs: Alternatives

There are two alternatives to using the `yt_search()` function from the `tuber` package to search for *YouTube* video IDs:

1. Manual search via the *YouTube* website: This works well for a small number of videos but is not really feasible if you want a large number of video IDs for your analyses.

2. Web scraping: Freelon (2018) argues that researchers interested in social media and other internet data should know/learn how to scrape the web in what he calls the "post-API age". However, you should use this method with caution. The robots.txt of *YouTube* disallows the scraping of results pages. In practical terms, the (over)use of web scraping might, e.g., get your IP address blocked (at least temporarily). A complete introduction to web scraping would be beyond the scope of this workshop, but we have created some slides with a short tutorial (in the folder with the slides on the *YouTube* API) and there are many online tutorials on this (e.g., this one).

Other Useful `tuber` Functions

Function	Output
get_video_details('video_id')	list with metadata for the video (e.g., title, description, tags)
get_all_channel_video_stats('channel_id')	list with statistics for all videos in a channel
get_playlists(filter=c(channel_id='channel_id'))	list with playlists for a channel
get_playlist_items(filter=c(playlist_id='playlist_id'))	dataframe with items from a playlist

For the full list of functions in the package, see the `tuber` reference manual.

As always when using `R` (packages), it helps to check the documentation for the functions that you (want to) use. For example:

```
?get_video_details
?get_all_comments
```

YouTube API Quota Limits

In order to be able to estimate the quota costs of a specific API call via the `tuber` package, you need to know the specific queries that the function you use produces. If you want to do this, you can print the function code (you can do this in `R` by executing the function name without `()` at the end: e.g., `get_stats`). You can then use the information from that code for the Quota Calculator or the API Explorer (by checking the respective method in the YouTube API documentation). The help file for the function of interest can also be useful here (`?get_stats`).

`tuber` Function Code

```
get_stats
```

```
## function (video_id = NULL, ...)
## {
##     if (!is.character(video_id))
##         stop("Must specify a video ID.")
##     querylist <- list(part = "statistics", id = video_id)
##     raw_res <- tuber_GET("videos", querylist, ...)
##     if (length(raw_res$items) == 0) {
##         warning("No statistics for this video are available.\n
##         return(list())
##     }
##     res <- raw_res$items[[1]]
##     stat_res <- res$statistics
##     c(id = res$id, stat_res)
## }
## <bytecode: 0x000000001641c1b8>
## <environment: namespace:tuber>
```

Exemplary Quota Costs for `tuber` Functions

Function	Costs	API resource	API method	API parts
get_channel_stats('channel_id')	1	Channels	list	statistics,snippet
get_stats('video_id')	1	Videos	list	statistics
get_comment_threads(filter = c(video_id = 'video_id'), max_results = 100)	1	CommentThreads	list	snippet

The costs increase with the number of returned items/results.

For reference: `yt_search()` has a quota cost of 100 per page of results (with a default number of max. 50 results per page).

How do I Know the Quota Costs When I use `tuber`?

a) Estimate the costs based on costs for the specific query and the number of returned results. For example, if a video has 4000 comments and you wish to collect all of them, you can estimate your cost as follows: 100 results per page = 40 pages & cost per page for `get_all_comments()` = 1 -> overall cost = 40. In practice, this is a bit more difficult to estimate as both the website of the video and `get_stats()` give you the comment count including all replies, but `get_all_comments()` only returns up to 5 replies per comment.

b) You can use the *Google* Cloud Platform for monitoring your API quota use: Choose *APIs & Services* from the Navigation Menu on the left side and look at the stats for the *YouTube Data API v3* at the bottom of the *Dashboard* view. If you want a more detailed visualization, you can use the *Google* Developer Console: Choose your project, then on the *Dashboard* click on the name of the API in the list on the bottom (*YouTube Data API v3*), select *Quotas* from the menu on the left, and choose *Queries per minute* from the dropdown-menu for *Queries* as well as an appropriate timeframe (e.g., 6 hrs). **NB**: It takes a bit for the numbers to update after sending your API calls.

Managing Your API Quota

The daily quota for your app resets at midnight Pacific Time. There are 3 ways of dealing with the quota limit:

1. Schedule your scripts. You can either do this manually (by splitting your data collection into batches and running the scripts on different days) or automatically (e.g., using the taskScheduleR package for Windows or the cronR package for Linux/Unix).

2. Request an increase through the Quotas menu in the developer console for your app. However, this will quite likely cost money or might not work at all.

3. Use multiple apps and/or accounts for your data collection. This requires some manual scheduling/planning. Keep in mind that there is a limited number of projects per account for the *Google* Cloud Platform. Note that - depending on your setup and use - this strategy might get your API quota, app(s), account(s) or IP address(es) suspended/blocked (at least temporarily) if employed excessively (and we can't put a number on what 'excessively' means here, so this is just a word of caution).