Automatic Sampling and Analysis of YouTube Data

The YouTube API

Julian Kohne
Johannes Breuer
M. Rohangis Mohseni

2022-02-21

The YouTube API

How do we get Data From Websites?

- Theoretically, we could gather all the information manually by clicking on the things that are interesting to us and copy/pasting them. However, this is tedious and time-consuming. **We want a way of automatizing this task**. The solution to our problem is...

- Webscraping

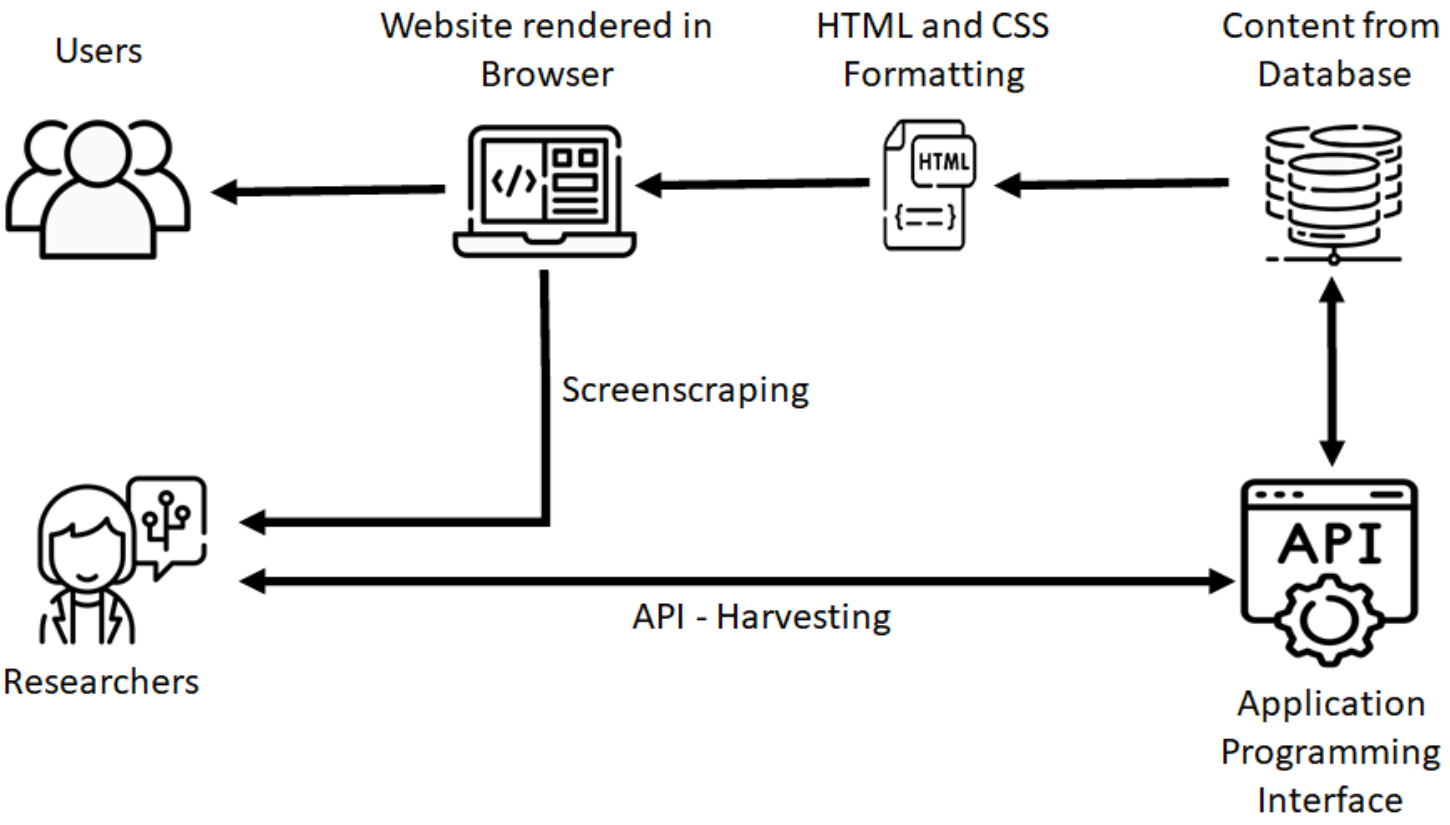
  1) **Screenscraping:** Getting the HTML-code out of your browser, parsing & formatting it, then analyzing the data

  2) **API harvesting:** Sending requests directly to the database and only getting back the information that you want and need

Overview

- All data on *YouTube* is stored in a MySQL database
- The website itself is an HTML page, which loads content from this database
- The HTML is rendered by a web browser so the user can interact with it
- Through interacting with the rendered website, we can either retrieve content from the database or send information to the database
- The YouTube website is
  - built in HTML
  - uses CSS for the "styling"
  - dynamically loads content using Ajax from the Database

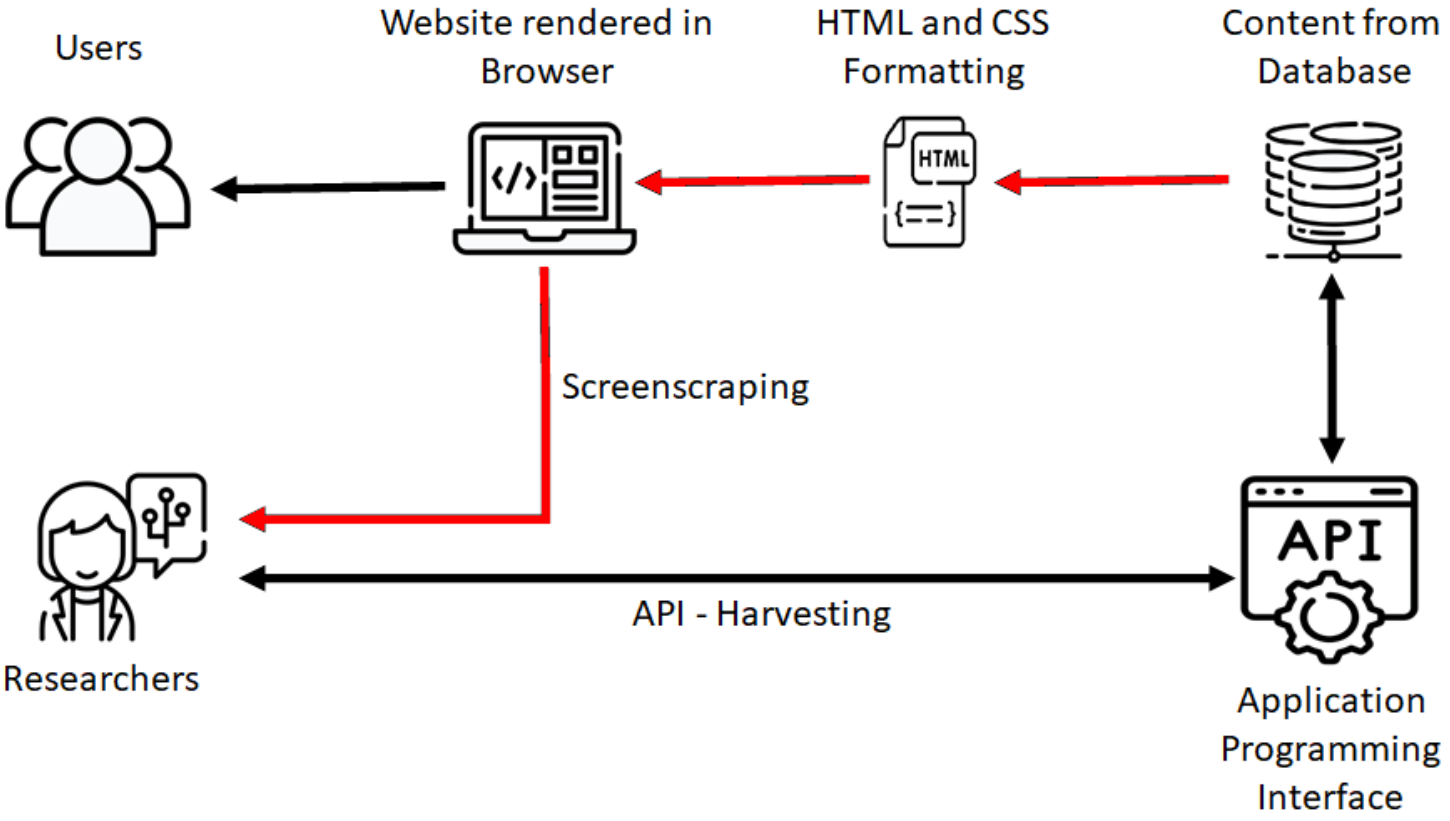
Overview


Users
Website rendered in Browser
HTML and CSS Formatting
Content from Database
HTML
Screenscraping
API
API - Harvesting
Researchers
Application Programming Interface

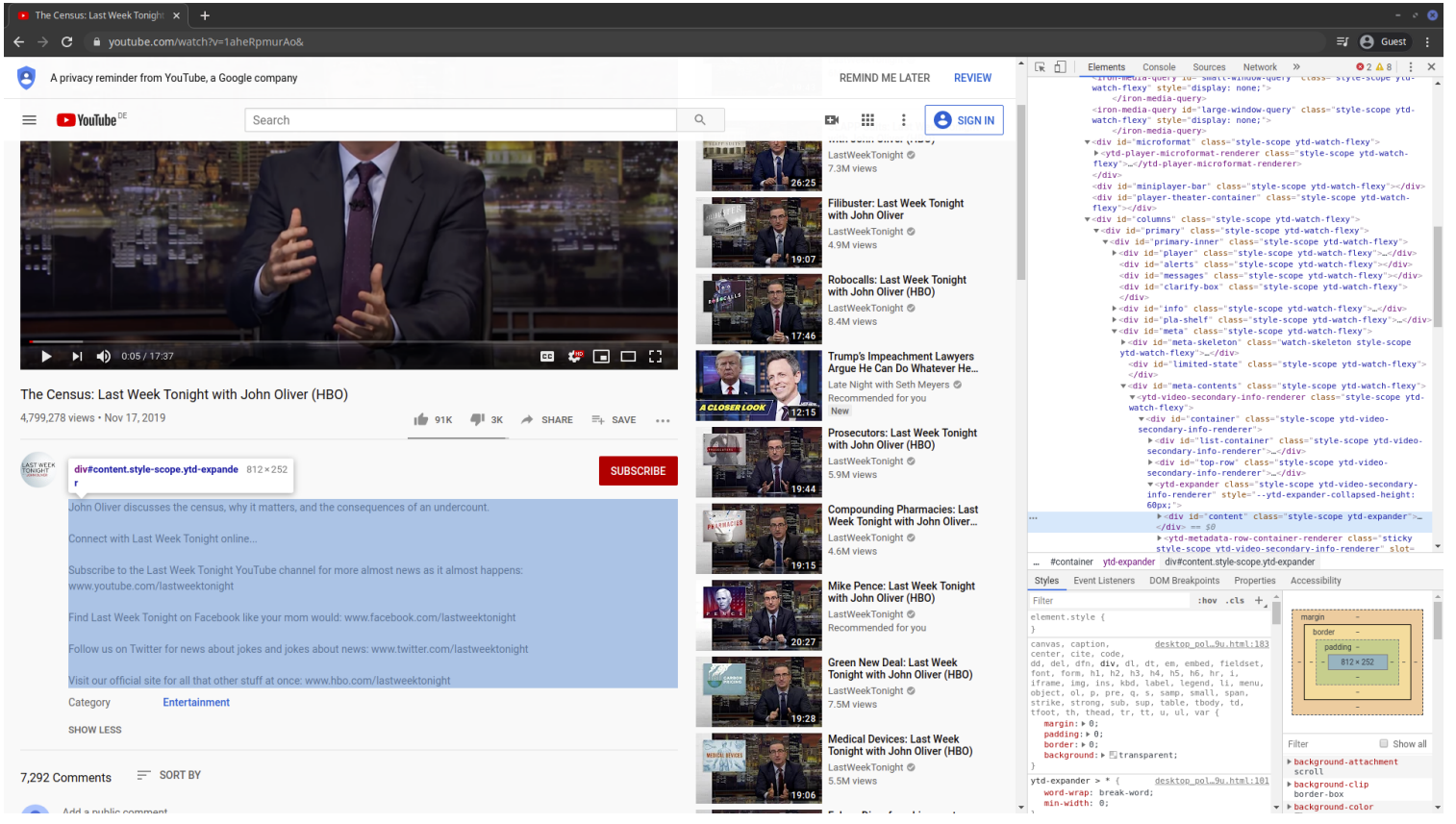
Screenscraping

Screenscraping

- Screenscraping means that we are downloading the HTML text file, which contains the content we are interested in but also a lot of unnecessary clutter that describes how the website should be rendered by the browser



Screenscraping



Screenscraping

- To automatically obtain data, we can use a so-called GET request
- A GET request is an HTTP method for asking a server to send a specific resource (usually an HTML page) back to your local machine. It is implemented in many different libraries such as curl
- This is the basic principle that all the scraping packages are built on
- We will not use this directly and will let the higher-level applications handle this under the hood

Screenscraping - Examples

- From the console in Linux or MacOS (saves html to a file)

```
curl "https://www.youtube.com/watch?v=1aheRpmurAo/" >
YT.html
```

- Online, using the code from above

```
curl "https://www.youtube.com/watch?v=1aheRpmurAo/"
```

- In R

```
# Warning about incomplete final line can (usually) be ignored
library(curl)
html_text <-
readLines(curl("https://www.youtube.com/watch?v=1aheRpmurAo/"))
```

Screenscraping

- Advantages of Screenscraping:
  - You can access everything that you are able to access from your browser
  - You are (theoretically) not restricted in how much data you can get
  - (Theoretically) Independent from API-restrictions
- Disadvantages of Screenscraping:
  - Extremely tedious to get information out of HTML-pages
  - You have to manually look up the Xpaths/CSS/HTML containers to get specific information
  - Reproducibility: The website might be tailored to stuff in your Cache, Cookies, Accounts etc.
  - There is no guarantee that even pages that look the same have the same underlying HTML structure
  - You have to manually check the website and your data to make sure that you get what you want
  - If the website changes anything in their styling, your scripts won't work anymore
  - Legality depends on country

API-Harvesting

API Harvesting

- An **A**pplication **P**rogramming **I**nterface...
  - is a system built for developers
  - directly communicates with the underlying database(s)
  - is a voluntary service of the website
  - controls what information is accessible, to whom, how, and in which quantities

API-Harvesting

- APIs can be used to:
  - embed content in other applications
  - create bots that do something automatically
  - scheduling/moderation for content creators
  - collect data for (market) research purposes
- Not every website has their own API. However, most large social media Websites do, e.g.:
  - Facebook
  - Twitter
  - Instagram
  - Wikipedia
  - Google Maps

API Harvesting - Examples

- From the console
  (API Key needs to be added before execution)

```
curl "https://www.googleapis.com
    /youtube/v3/search?
    part=snippet&q=Brexit&
    key=INSERT-API-KEY-HERE"
```

- Online, using code from above (API Key needs to be added before execution)

- In R (API Key needs to be added before execution, data needs to be converted to JSON format)

```
library(curl)
library(jsonlite)
api_response <- fromJSON(curl("https://www.googleapis.com/
                              youtube/v3/search?
                              part=snippet&q=Brexit&
                              key=INSERT-API-KEY-HERE"))
```

API Harvesting

- Advantages of API Harvesting:
  - No need to interact with HTML files, you only get the information you asked for
  - The data you get is already nicely formatted (usually JSON files)
  - You can be confident that what you do is legal (if you adhere to the Terms of Service and respect data privacy and copyright regulations)
- Disadvantages of API Harvesting:
  - Not every website has an API
  - You can only get what the API allows you to get
  - There are often restricting quotas (e.g., daily limits)
  - Terms of Service can restrict how you may use the data (e.g., with regard to sharing or publishing it)
  - There is no standard language to make queries, you have to check the documentation
  - Not every API has a (good) documentation

Screenscraping vs. API-Harvesting

If you can, use an API, if you must, use screenscraping instead

The YouTube API

Summary

- To find an API for a given website, Programmable Web is a good starting point
- Fortunately, *YouTube* has its own, well-documented API that developers can use to interact with their database (most *Google* services do)
- We will use the YouTube API in this workshop

Let's Check Out the API!

- Google provides a sandbox for their API that we can use to get a grasp of how it operates
- We can, for example, use our credentials to search for videos with the keyword "Brexit"
- Example
- Keep in mind: We have to log in with the *Google* account we used to create the app to use the API
- What we get back is a JSON-formatted response with the formats and information we requested in the API sandbox

API Key vs. OAuth2.0

- There are two different ways to authenticate with the YouTube API
    - API Key: Text string identifying the App and user, grants access to public data
    - OAuth2.0: Token created from Client secret and Client ID, grant access to everything the user can access
- For most API calls, the API key is enough
- Tuber is using OAuth2.0 authentification because it also let's you change your account information from R

Constructing API calls

We can construct all calls to the API according to the same logic

YouTube Data API v3 – Call Construction

https://youtube.googleapis.com/youtube/v3/search?maxResults=10&pageToken=2&q=Omicron&key=[YOUR_API_KEY]

https://youtube.googleapis.com/youtube/v3/	API Address, this is always constant
search	Type of resource to retrieve
?	Separator to distinguish resources from parameters
maxResults=10 pageToken=2 q=Omicron	Parameters for specifying format and content of resource
&	Separator to distinguish parameters from each other
key=[YOUR_API_KEY]	Your API key

What is JSON?

- Java Script Object Notation
- Language-independent data format (like .csv)
- Like a nested List of Key:Value pairs
- Standard data format for many APIs and web applications
- Better than tabular formats (.csv / .tsv) for storing large quantities of data by not declaring missing data
- Represented in `R` as a list of lists that typically needs to be transformed into a regular dataframe (this can be tedious)

What is JSON?

```
'{
  "first name": "John",
  "last name": "Smith",
  "age": 25,
  "address": {
    "street address": "21 2nd Street",
    "city": "New York",
    "postal code": "10021"
  },
  "phone numbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "mobile",
      "number": "646 555-4567"
    }
  ],
  "sex": "male"
}'
```

Most Important Parameters

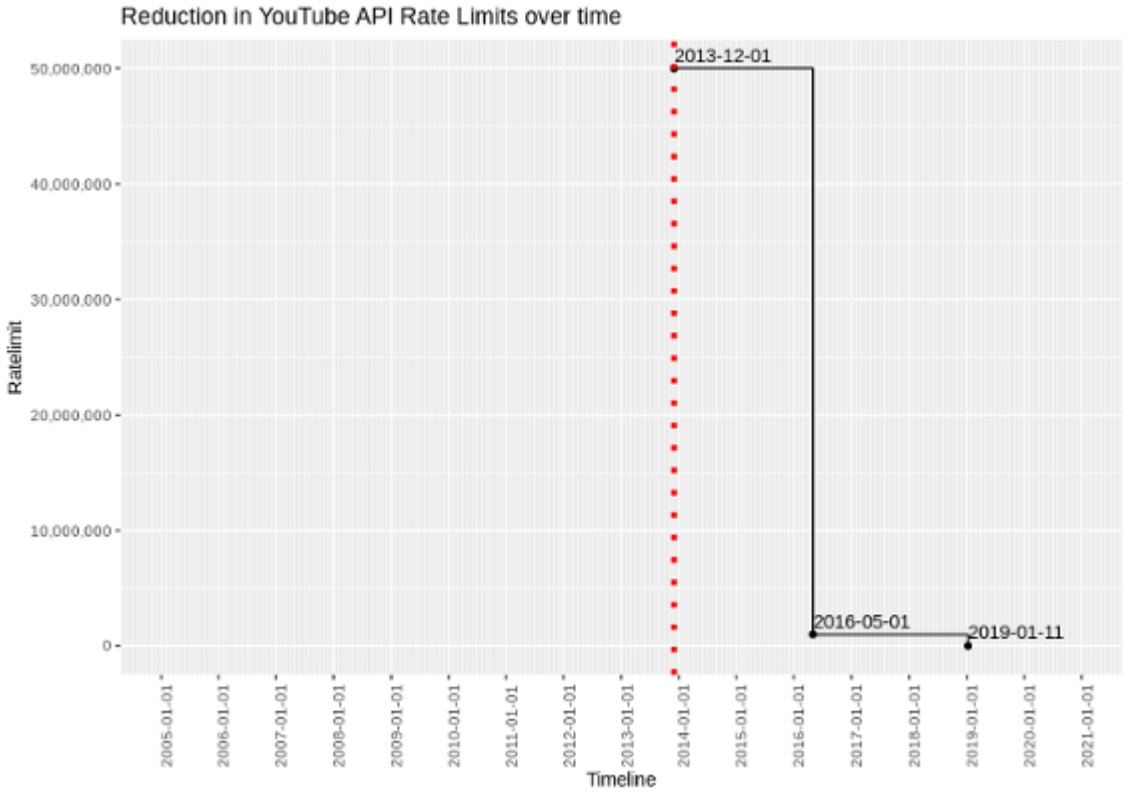
- All possible resources for the *YouTube* API are listed here
- For our workshop, the most important methods will be `search`, `Comments`, `CommentThreads`, and `videos`
- Some information is only visible to owners of a channel or author of a video
- Not all information is necessarily available for all videos (e.g., live videos)
- Public data requires an API key, getting user data requires OAuth2.0 authentication

Using it from R

- We can simplify the process of interacting with the YouTube API by using a dedicated `R` package
- The package handles the authentication with our credentials and translates `R` commands into API calls
- It also simplifies the JSON response to a standard dataframe automatically for many requests
- In essence, we can run `R` commands and get nicely formatted API results back
- For this workshop, we will mostly use the tubeR package, and briefly mention the vosonSML package

Rate Limits

- With the API, you have a limit of how much data you can get
- This limit has constantly decreased over the last decade



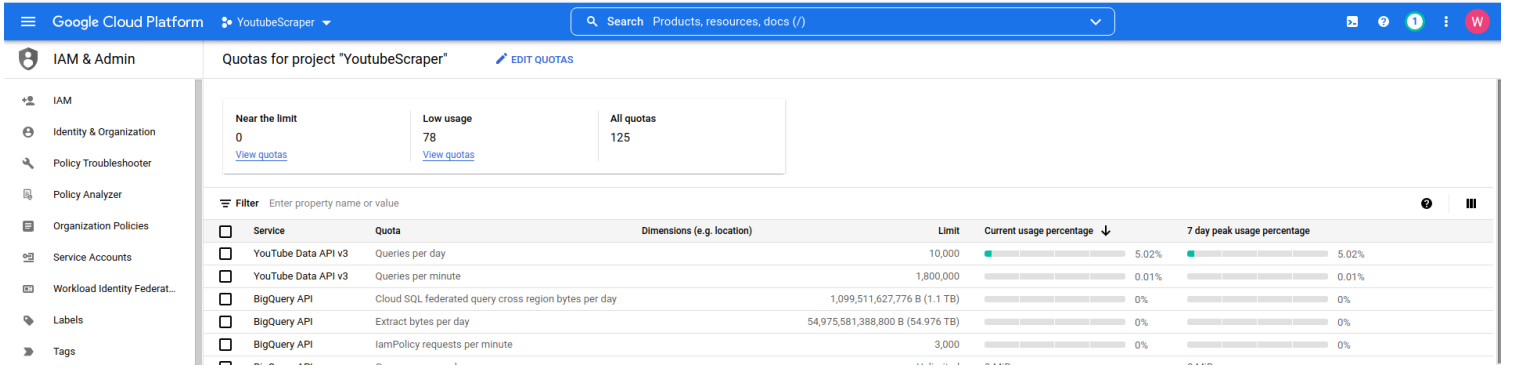
Rate Limits

- Currently (02.2022), you have a quota of **10.000** units per day
- Each request (even invalid ones) costs a certain amount of units
- There are two factors influencing the quota cost of each request:
  - different types of requests (e.g., write operation: 50 units; video upload: 1600 units)
  - how many parts the requested resource has (playlist:2 ; channel:6 ; video:10)
- **You should only request parts that you absolutely need to make the most of your units. More on that in the data collection session.**

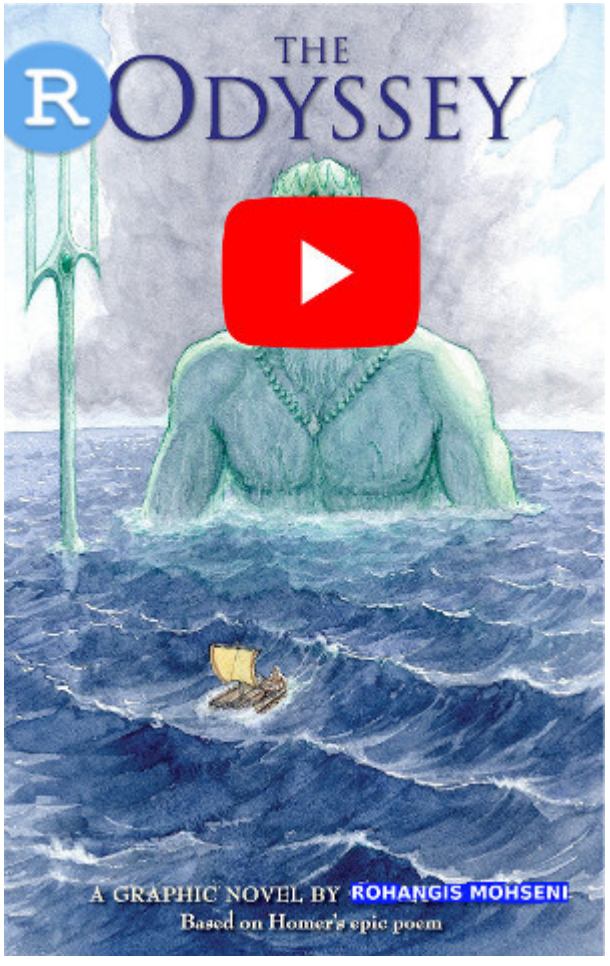
NB: Sending incorrect requests can fill up your daily quota

Rate Limits

- You can check the rate limits in the *YouTube* API Documentation
- You can see how much of your quota you have already used up in the *Google* Developer Console



Can I Increase my Rate Limit?



Trying to Raise the *YouTube* API Quota

- Study that needs large datasets in a short amount of time
- RQ: Is there a u-shaped relationship between success and number of uploads?
- Sample: 600 popular channels (identified via SocialBlade)
- Request for higher quota (October 11, 2019)
- Problem: Same application form for (web) apps and research
- Hard to figure what applies to research and what to write into the form
- Experience: Stuck in an infinite loop with e-mails from *Google* support on this issue

Any questions?

Exercise time 🏋️💪🏃🚴

Solutions