A test

Camille Landesvatter

2021-12-06

Contents

Preface

The present online book provide a review of APIs that may be useful for social scientists. Please start by reading the Introduction. The material was/is being developed by various contributors that you can find above and in the contributor section of the corresponding github repository. If you are interested in contributing please check out the Section How to contribute in the github README.

The material is licensed under a Apache License 2.0 license. Where we draw on other authors material other licenses may apply. We are extremely grateful for feedback and if you find errors please let us know.

This document was generated with R, RMarkdown and Bookdown.

Chapter 1

Introduction

The project *APIs for Social Scientists: A collaborative Review* is an outcome of the seminar *Computational Social Science* (CSS) taught at the University of Mannheim in 2021. While teaching the seminar we had trouble finding short reviews of APIs with quick R-code examples. Fortunately, almost everyone participating in the seminar was motivated enough to write a quick API review. Hopefully, our resource will be help future students to start diving into different APIs.

Below we review different data- and service-APIs that may be useful to social scientists. The chapters always include a simple R code example as well as references to social science research that has relied on them. The idea is to provide short reviews of max. 10 pages for the corresponding API with code to get you started. Each chapter follows a systematic set of questions:

- What data/service is provided by the API? (+ who provides it?)
- What are the prerequisites to access the API (e.g., authentication)?
- What does a simple API call look like?
- How can we access the API from R (httr + other packages)? * Are there social science research examples using the API?

1.1 Prerequesits: Authentication

A lot of the APIs require that you authenticate with the API provider. The underlying script of this review is written in such a way that it contains R chunks for authentication, however they will not be visible in the examples below (we only show placeholders for you to recognize at which step you will need to authenticate). These chunks in most cases make use of so-called keys in JSON format (e.g., service account key for Google APIs). However cloning the corresponding repository of this review will not result in giving you the keys, hence in order to replicate our API calls, you will have to generate and use your own individual keys.

1.2 Prerequesits: Software & packages

The code examples rely R and different packages thereof. It's probably easiest if you install all of them in one go. The `p_load()` function (`pacman` package) checks whether packages are installed. If not they are installed and loaded.

```
library(pacman)
pacman::p_load(
  dplyr, # needed for almost any chapter in this review
  ggplot2, # e.g. CH5
  tidytext, # comes in handy whenever text data is being pre-processed (CH2)
  devtools, # to download any package from github (e.g., RCrowdTangle in CH4)
  jsonlite, # import of JSON formats (e.g., CH4)
  httr, # Tools for Working with URLs and HTTP (various chapters if an API call can me
  googleLanguageR, # allows different API calls for languga processing (CH2, CH3)
  RCrowdTangle, #CH4
  googleway, #CH5
  mapsapi, #CH5
  stars, #CH5
  httr,
  WikipediR # CH12
)

# Move these installations before pacman?
devtools::install_github("quanteda/quanteda.corpora")
devtools::install_github("cbpuschmann/RCrowdTangle")
```

Chapter 2

Google Natural Language API

Paul C. Bauer, Camille Landesvatter, Malte Söhren

2.1 Provided services/data

- *What data/service is provided by the API?*

The API is provided by Google.

Google Cloud offers two Natural Language Products: AutoML Natural Language and Natural Language API. See here to read about which of the two products is the one more useful to you. In short, option 1, the Auto Machine Learning (ML) Natural Language allows you to train a new, custom model to either classify text, extract entities or detect sentiment. For instance, you could provide an already pre-labeled subset of your data which the API will then use to train a custom classifier. With this classifier at hand you could then classify and analyze further similar data of yours. This API review focuses on option 2, the Natural Language API. This API uses pre-trained models to analyze your data. Put differently, instead of providing only a pre-labeled subset of your data, here you normally provide the API with your complete (unlabeled) data which it will then analyze.

The following requests are available:

- Analyzing Sentiment (`analyzeSentiment`)
- Analyzing Entities (`analyzeEntities`)
- Analyzing Syntax (`analyzeSyntax`)

- Analyzing Entity Sentiment (`analyzeEntitySentiment`)
- Classifying Content (`classifyText`)

A demo of the API that allows you to input text and explore the different classification capabilities can be found here.

2.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

The prerequisite to access Google Natural Language API is a Google Cloud Project. To create this you will need a Google account to log into the Google Cloud Platform (GCP). Within your Google Cloud Platform, you must enable the Natural Language API for your respective Google Cloud Project here. Additionally, if you are planning to request the Natural Language API from outside a Google Cloud environment (e.g., R) you will be required to use a private (service account) key. This can be achieved by creating a service account which in turn will allow you to download your private key as a JSON file. To create your API key for authentication from within the GCP, go to APIs & Services > Credentials. Below we provide an example of how to authenticate from within the Google Cloud Platform (Cloud Shell + API key) and how to authenticate from within R (authentication via JSON key file).

2.3 Simple API call

- *What does a simple API call look like?*

Here we describe how a simple API call can be made from within the Google Cloud Platform environment via the Google Cloud Shell:

- To activate your Cloud Shell, inspect the upper right-hand corner of your Google Cloud Platform Console and click the icon called "Activate Shell". Google Cloud Shell is a command line environment running in the cloud.
- Via the Cloud Shell command line, add your individual API key to the environment variables, so it is not required to be called for each request.

```
export API_KEY=<YOUR_API_KEY>
```

- Via the built-in Editor in Cloud Shell create a JSON file (call it for instance 'request.json') with the text that you would like to perform analysis on. Consider that text can be uploaded in the request (shown below) or integrated with Cloud Storage. Supported types of your text are PLAIN_TEXT (shown below) or HTML.

```
{
  "document":{
    "type":"PLAIN_TEXT",
    "content":"Enjoy your vacation!"
  },
  "encodingType": "UTF8"
}
```

- For sending your data, pass a curl command to your Cloud Shell command line where you refer (via @) to your request.json file from the previous step.

```
curl "https://language.googleapis.com/v1/documents:analyzeEntities?key=${API_KEY}" -s -X POST -H
```

- Depending on to which endpoint you send the request (here: analyzeEntities) you will receive your response with many different insights into your text data.

2.4 API access

- *How can we access the API from R (httr + other packages)?*

The input (i.e., text data) one provides to the API most often will go beyond a single word or sentence. The most convenient way which also produces the most insightful and structured results (on which you can directly perform further analysis on) are achieved when using the 'googleLanguageR' R package - a package which among other options (there are other examples in this review) allows calling the Natural Language API:

In this small example we demonstrate how to..

- .. authenticate with your Google Cloud Account within R
- .. how to analyze the syntax of exemplary twitter data (we are using twitter data from two popular german politicians, which we (via the Google Translation API) beforehand also translated to english)
- .. how to extract terms that are nouns only
- .. plot your nouns in a word cloud

Step 1: Load package

```r
library(googleLanguageR)
library(tidyverse)
library(tm)#stopwords
```

Step 2: Authentication

```r
gl_auth("./your-key.json")
```

Step 3: Analysis

Start with loading your text data. For this example, we retrieve data inherit to the quanteda.corpora R package which in a broader sense is associated with the famous quanteda package.

The data we choose to download ('data_corpus_guardian') contains Guardian newspaper articles in politics, economy, society and international sections from 2012 to 2016. See here for a list of even more publicy available text corpora from the quanteda.corpora package.

```r
# Download and store corpus
guardian_corpus <- quanteda.corpora::download("data_corpus_guardian")

# Keep text only from the corpus
text <- guardian_corpus[["documents"]][["texts"]]

# For demonstration purposes, subset the text data to 20 observations only
text <- text[1:20]

# Turn text into a data frame and add an identifier
df <- as.data.frame(text)
df <- tibble::rowid_to_column(df, "ID")
```

Note: Whenever you choose to work with textual data, a very common procedure is to pre-process the data via a set of certain transformations. For instance, you will convert all letters to lower case, remove numbers and punctuation, trim words to their word stem and eventually remove so-called stopwords. There are many tutorials (for example here or here).

After having retrieved and prepared our to-be-analyzed (text) data, we can now call the API via the function `gl_nlp()`. Here you will have to specify the quantity of interest (here: `analyzeSyntax`). Depending on what specific argument you make use of (e.g., `analyzeSyntax`, `analyzeSentiment`, etc.) a list with information on different characteristics of your text is returned, e.g., sentences, tokens, tags of tokens.

```r
syntax_analysis <- gl_nlp(df$text, nlp_type = "analyzeSyntax")
```

Importantly, find the list `tokens` inherit to the large list `syntax_analysis`. This list stores two variables: `content` (contains the token) and `tag` (contains the tag, e.g., verb or noun). Let's have a look at the first document.

```r
head(syntax_analysis[["tokens"]][[1]][,1:3])
```

```
##        content beginOffset   tag
## 1       London           0  NOUN
## 2 masterclass            7  NOUN
## 3           on          19   ADP
## 4      climate          22  NOUN
## 5       change          30  NOUN
## 6            |          37 PUNCT
```

Now imagine you are interested in all the nouns that were used in the Guardian Articels while removing all other types of words (e.g., adjectives, verbs, etc.). We can simply filter for those using the "tag"-list.

```r
# Add tokens from syntax analysis to original dataframe
df$tokens <- syntax_analysis[["tokens"]]

# Keep nouns only
df <- df %>% dplyr::mutate(nouns = map(tokens,
                             ~ dplyr::filter(., tag == "NOUN")))
```

Step 4: Visualization

Finally, we can also plot our nouns in a wordcloud using the ggwordcloud package.

```r
# Load package
library(ggwordcloud)
```

```r
# Create the data for the plot
data_plot <- df %>%
  # only keep content variable
  mutate(nouns = map(nouns,
                             ~ select(., content))) %>%
  # Write tokens in all rows into a single string
  unnest(nouns) %>% # unnest tokens
  # Unnest tokens
  tidytext::unnest_tokens(output = word, input = content) %>% # generate a wordcloud
  anti_join(tidytext::stop_words) %>%
  dplyr::count(word) %>%
  filter(n > 10) #only plot words that appear more than 10 times

# Visualize in a word cloud
data_plot %>%
  ggplot(aes(label = word,
             size = n)) +
  geom_text_wordcloud() +
  scale_size_area(max_size = 10) +
  theme_minimal()
```

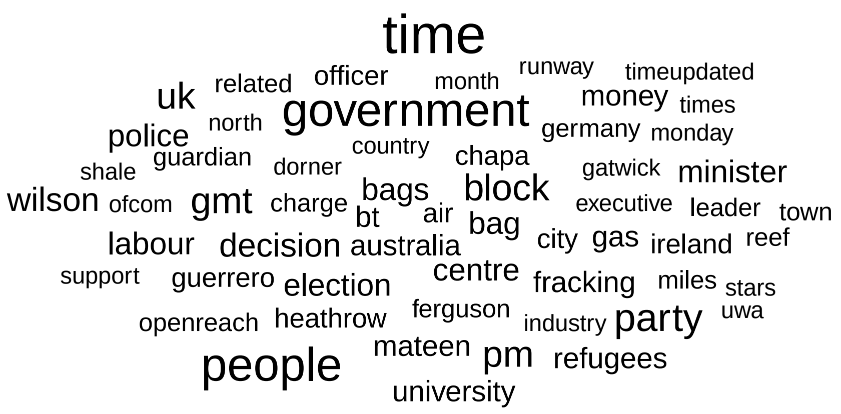


Figure 2.1: Wordcloud of nouns found within guardian articles

2.5 Social science examples

- *Are there social science research examples using the API?*

Text-as-data has become quite a common approach in the social sciences (see e.g., **?** for an overview). For the usage of Google's Natural Language API we however have the impression that it is relatively unknown in NLP and among social scientists. Hence, we want to emphasize the usefulness and importance the usage of Google's NLP API could have in many research projects.

However, if you are considering making use of it, keep two things in mind:

- some might interpret using the API as a "blackbox" approach (see **?** for very recent developments of "glass-box machine learning appraoches" for text analysis) potentially standing in way of transparency and replication (two important criteria of good research?). However it is always possible to perform robustness and sensitivity analysis and to add the version of the API one was using.
- depending on how large your corpus of text data, Google might charges you some money. However for up to 5,000 units (i.e., terms) the different variants of sentiment and syntax analysis are free. Check this overview by Google to learn about prices for more units here. Generally, also consider that if you are pursuing a CSS-related project in which the GCP Products would come in useful, there is the possibility to achieve Google Cloud Research credits (see here).

Chapter 3

Google Translation API

Paul C. Bauer, Camille Landesvatter

3.1 Provided services/data

- *What data/service is provided by the API?*

The API is provided by Google.

Google's Translation API translates texts into more than one hundred languages. Note that the approach via the API is a lot more refined than the free version on Googles' translation website and of course comes in very useful if text in large scale needs to be translated (possibly with longer and more complex content or syntax). For instance, you can choose to specify a certain model to further improve the translation (Neural Machine Translation vs. Phrase-Based Machine Translation).

The API limits in three ways: characters per day, characters per 100 seconds, and API requests per 100 seconds. All can be set in the API manager of your Google Cloud Project.

Consider that additionally to the Translation API which we demonstrate in this review, Google provides us with two further APIs for translation: AutoML Translation and the Advanced Translation API (see here for a short comparison).

3.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

To access and to use the API the following steps are necessary:

- Create a google account (if you do not already have one).
- Using this google account login to the google cloud platform and create a Google Cloud Project.
- Within this Google Cloud Project enable the Google Translation API.
- For authentication you will need to create an API key (which you additionally should restrict to the Translation API). If however, you are planning to request the Natural Language API from outside a Google Cloud environment (e.g., R) you will be required to use a private (service account) key. This can be achieved by creating a service account which in turn will allow you to download your private key as a JSON file (we show an example below).

3.3 Simple API call

- *What does a simple API call look like?*

Here we describe how a simple API call can be made from within the Google Cloud Platform environment via the Google Cloud Shell:

- To activate your Cloud Shell, inspect the upper right-hand corner of your Google Cloud Platform Console and click the icon called "Activate Shell". Google Cloud Shell is a command line environment running in the cloud.
- Via the built-in Editor in Cloud Shell create a JSON file (call it for instance 'request.json') with the text that you would like to perform analysis on. Consider that text can be uploaded in the request (shown below) or integrated with Cloud Storage. Supported types of your text are PLAIN_TEXT (shown below) or HTML.

```
{
 "q": ["To administer medicine to animals is frequently a very difficult matter, and y
"target": "de"
}
```

- For sending your data, pass a curl command to your Cloud Shell command line where you refer (via @) to your request.json file from the previous step.
- Don't forget to insert your individual API key in the curl command (alternatively, you could define it beforehand via adding a global variable to your environment → see example in the API call for Googles' NLP API earlier in this document).

```
curl "https://translation.googleapis.com/language/translate/v2?key=APIKEY" -s -X POST -H "Content
```

3.4 API access

- *How can we access the API from R (httr + other packages)?*

The following example makes use of the 'googleLanguageR' package from R which among other options (e.g., syntax analysis → see Chapter 2 of this review) allows calling the Cloud Translation API.

In this small example we demonstrate how to...

- ... authenticate with your Google Cloud Account within R
- ... translate an exemplary sentence

Step 1: Load package

```
library(googleLanguageR)
```

Step 2: Authentication

```
gl_auth("./your-key.json")
```

Step 3: Analysis - API call and Translation

First, we create exemplary data. For demonstration purposes, we here again make use of the sentence from above. Of course, for your own project use a complete vector containing text data from within your data.

```
df_original <- data.frame(text = "To administer medicine to animals is frequently a very difficul

df_original$text <- as.character(df_original$text)
```

Next, using this data, call the API via the function 'gl_translate()' and importantly specify the target language (german in this example) within the 'target' argument.

```
df_translate <- gl_translate(df_original$text, target = "de")
```

This API call eventually provides reuslts in a dataframe with three columns: translatedText (→ contains the translation), detectedSourceLangauge (→ contains a label for the original language being detected) and text (→ contains the original text).

Let's check the translation.

```
df_translate$translatedText
```

Tieren Medikamente zu verabreichen ist oft eine sehr schwierige Angelegenheit, und doch ist es manchmal notwendig

3.5 Social science examples

- *Are there social science research examples using the API?*

Searching on literature using Google Translation Services we found a study by **?** (Assessing gender bias in machine translation: a case study with Google Translate).

We are not aware of any further research / publications that made explicit usage of Google's Translation API. However, we assume that the API (or at least Google's Free Translation Service) is involved in many research projects. Generally, we are convinced that making use of automated translation (as well as converting speech to text (→ example also in this review) eventually in combination with translation) can be of great advantage for all kinds of qualitative or mixed-methods research projects. For instance, automated translation could be very useful for easily and very reliably translating data from qualitative interviews or other (field) experiments or observational data (where data might be collected in a foreign language). Also consider the other way around where data is available in the principal language of the research project but some of the textual data has to be communicated and presented in (for instance) english language.

Chapter 4

CrowdTangle API

Lion Behrens and Pirmin Stöckle

CrowdTangle is a public insights tool, whose main intent was to monitor what content overperformed in terms of interactions (likes, shares, etc.) on Facebook and other social media platforms. In 2016, CrowdTangle was acquired by Facebook that now provides the service.

4.1 Provided services/data

- *What data/service is provided by the API?*

CrowdTangle allows users to systematically follow and analyze what is happening with public content on the social media platforms of Facebook, Twitter, Instagram and Reddit. The data that can be assessed through the CrowdTangle API consists of any post that was made by a public page, group or verified public person who has ever acquired more than 110,000 likes since the year 2014 or has ever been added to the list of tracked public accounts by any active API user. If a new public page or group is added, data is pulled back from day one.

Data that is tracked:

- Content (the content of a post, including text, included links, links to included images or videos)
- Interactions (count of likes, shares, comments, emoji-reactions)
- Page Followers
- Facebook Video Views
- Benchmark scores of all metrics from the middle 50% of posts in the same category (text, video) from the respective account

Data that is not tracked:

- Comments (while the number of comments is included, the content of the comments is not)
- Demographical data
- Page reach, traffic and clicks
- Private posts and profiles
- Ads only appear in the ad library (which is public), boosted content cannot differentiated from organic content

CrowdTangle's database is updated once every fifteen minutes and comes as time-series data which merges the content of a post on one of the included platforms (a text post, video, or image) alongside aggregate information on the post's views, likes and interactions.

When connecting to the user interface via the CrowdTangle website, the user can either manually set up a list of pages of interest whose data should be acquired. Alternatively, one can choose from an extensive number of pre-prepared lists covering a variety of topics, regions, or socially and politically relevant events such as inaugurations and elections. Data can be downloaded from the user interface as csv files or as json files via the API.

4.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

Full access to the CrowdTangle API is only given to Facebook partners who are in the business of publishing original content or fact-checkers as part of Facebook's Third-Party Fact-Checking program. From 2019, the CrowdTangle API and user interface is also available for academics and researchers in specific fields. Currently, this prioritization includes research on one of the following fields: misinformation, elections, COVID-19 racial justice, well-being. To get access to CrowdTangle, a formal request has to be filed via an online form, asking for a short description of the research project and intended use of the data.

As a further restriction, CrowdTangle currently only allows academic staff, faculty and registered PhD students permission to obtain a CrowdTangle account. This does not include individuals enrolled as students at a university unless they are employed as research assistants. Also, certain access policies differ between academics and the private sector. Usage of CrowdTangle for research purposes does currently not provide access to any content posted on Reddit given that data is retrieved via the Application Programming Interface. Content from Reddit is open to every registered user only when navigating through the company's dynamic user interface that does not imply usage of any scripting language. Finally, the CrowdTangle API requires researchers to log in using an existing Facebook account. Overall, access to the API is quite restrictive, both because of the prioritization of certain research areas, and because the access request will be decided individually so that an immediate access is not possible. If access is granted, CrowdTangle provides quite extensive onboarding and training resources to use the API.

Replicability

Access to CrowdTangle is gated and Facebook does not allow data from CrowdTangle to be published. So researchers can publish aggregate results from analyses on the data, but not the original data, which might be problematic for the replicability of research conducted with the API. A possible workaround is that you can pull ID numbers of posts in your dataset, which can then be used by anyone with a CrowdTangle API access to recreate your dataset. CrowdTangle also provides some publicly available features such as a Link Checker Chrome Extension, allowing users to see how often a specific link has been shared on social media, and a curated public hub of Live displays, giving insight about specific topics on Facebook, Instagram and Reddit.

4.3 Simple API call

- *What does a simple API call look like?*

All requests to the CrowdTangle API are made via GET to https://api.crowdtangle.com/.

In order to access data, users log in on the website with their Facebook account and acquire a personalized token. The CrowdTangle API expects the API token to be included in each query. With one of these available endpoints, each of which comes with a set of specific parameters:

GET /posts	Retrieve a set of posts for the given parameters.
GET /post	Retrieves a specific post.
GET /posts/search	Retrieve a set of posts for the given parameters and search terms.
GET /leaderboard	Retrieves leaderboard data for a certain list or set of accounts.
GET /links	Retrieve a set of posts matching a certain link.

GET /lists	Retrieve the lists, saved searches and saved post lists of the dashboard associated with the token sent in.

A simple example: Which party or parties posted the 10 most successful Facebook posts this year?

On the user interface, I created a list of the pages of all parties currently in the German Bundestag. We want to find out which party or parties posted the 10 most successful posts (i.e. the posts with the most interactions) this year.

The respective API call looks like that: https://api.crowdtangle.com/posts?token=token&listIds=listIDs
01-01&count=10, where token is the personal API key, and listIDs is the ID of the list created with the user interface. Here, we sortBy total interactions with the startDate at the beginning of this year and the output restricted to count 10 posts.

4.4 API access

- *How can we access the API from R (httr + other packages)?*

Instead of typing the API request into our browser, we can use the httr package's GET function to access the API from R.

```r
# Option 1: Accessing the API with base "httr" commands
library(httr)

ct_posts_resp <- GET("https://api.crowdtangle.com/posts",
    query=list(token = token, # API key has to be included in every query
               listIds = listIds, # ID of the created list of pages or groups
               sortBy = "total_interactions",
               startDate = "2021-01-01",
               count = 10))

ct_posts_list <- content(ct_posts_resp)
class(ct_posts_list) # verify that the output is a list

# List content
str(ct_posts_list, max.level = 3) # show structure & limit levels

# with some list operations we can get a dataframe with the account name and post date
list_part <- rlist::list.select(ct_posts_list$result$posts, account$name, date)
rlist::list.stack(list_part)
```

Alternatively, we can use a wrapper function for R, which is provided by the RCrowdTangle package available on github. The package provides wrapper functions for the /posts, /posts/search, and /links endpoints. Conveniently, the wrapper function directly produces a dataframe as output, which is typically what we want to work with. As the example below shows, the wrapper function may not include the specific information we are looking for, however, as the example also shows, it is relatively straightforward to adapt the function on our own depending on the specific question at hand. To download the package from github, we need to load the devtools package, and to use the wrapper function, we need dplyr and jsonlite.

```
# Option 2: There is a wrapper function for R, which can be downloaded from github

library(devtools) # to download from github

install_github("cbpuschmann/RCrowdTangle")
library(RCrowdTangle)

# The R wrapper relies on jsonlite and dplyr
library(dplyr)
library(jsonlite)

ct_posts_df <- ct_get_posts(listIds, startDate = "2021-01-01", token = token)

#conveniently, the wrapper function directly produces a dataframe
class(ct_posts_df)

# to sort by total interactions we have to compute that figure because it is not part of the data
ct_posts_df %>%
  mutate(total_interactions = statistics.actual.likeCount+statistics.actual.shareCount+ statistic
           statistics.actual.angryCount+ statistics.actual.thankfulCount+ statistics.actual.careC
  arrange(desc(total_interactions)) %>%
  select(account.name, date) %>%
  head(n=10)

# alternatively, we can adapt the wrapper function by ourselves to include the option to sort by
ct_get_posts <- function(x = "", searchTerm = "", language = "", types= "", minInteractions = 0,
{
  endpoint.posts <- "https://api.crowdtangle.com/posts"
  query.string <- paste0(endpoint.posts, "?listIds=", x, "&searchTerm=", searchTerm, "&language="
  response.json <- try(fromJSON(query.string), silent = TRUE)
  status <- response.json$status
  nextpage <- response.json$result$pagination$nextPage
  posts <- response.json$result$posts %>% select(-expandedLinks, -media) %>% flatten()
  return(posts)
}
```

```
ct_posts_df <- ct_get_posts(listIds, sortBy = "total_interactions", startDate = "2021-0

ct_posts_df %>%
  select(account.name, date) %>%
  head(n=10)
```

4.5 Social science examples

- *Are there social science research examples using the API?*

A common use case is to track the spread of specific links containing misinformation, e.g. conspiracy around the connection of COVID-19 and 5G (**?**). **?** provide an in-depth analysis of a specific page involved in online health misinformation and investigate factors driving interactions with the respective posts. They find that users mainly interact to foster social relations, not to spread misinformation. CrowdTangle has also been used to study changes in the framing of vaccine refusal by analyzing content of posts by pages opposing vaccinations over time (**?**). Another approach is to monitor political communication of political actors, specifically in the run-up to elections. **?** investigates a one-month period before the 2018 Swedish election and finds that right-wing political actors are more successful than mainstream actors in engaging their Facebook followers, often using sensational rhetoric and hate-mongering.

Chapter 5

Google Places API

Lukas Isermann and Clara Husson

5.1 Provided services/data

- *What data/service is provided by the API?*

The following five requests are available: Place Search, Place Details, Place Photos, Place Autocomplete and Query Autocomplete. Place Search returns a list of places along with summary information about each place based on a user's location (by proximity) or search string. Once you find a place_id from a Place Search, you can request more details about a particular place by doing a Place Details request. A Place Details request returns more detailed information about the indicated place such as its complete address, phone number, user rating and reviews. Place Photos provides access to the millions of place-related photos stored in Google's Place database. When you get place information using a Place Details request, photo references will be returned for relevant photographic content. Find Place, Nearby Search, and Text Search requests also return a single photo reference per place, when relevant. Place Autocomplete automatically fills in the name and/or address of a place as users type. Query Autocomplete service provides a query prediction for text-based geographic searches, by returning suggested queries as you type.

Note: You can display Places API results on a Google Map, or without a map but it is prohibited to use Places API data on a map that is not a Google map.

5.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

The prerequisites to access Google Places API are a Google Cloud project (to create it you need a Google account to log into the Google Cloud Platform) and an API Key. Before creating your API Key, don't forget to enable Places API! To create your API key, go the APIs & Services > Credentials > API key page.

5.3 Simple API call

- *What does a simple API call look like?*

The API provides different searches and services that can be accessed via HTTP Urls. These Urls requests all take the same general form and pattern:

https://maps.googleapis.com/maps/api/place/service/output?parameters

Here, service can take the inputs findplacefromtext for find place requests, nearbysearch to look for nearby places, details for a request of place details and more. output may take the value json or xml, dependent on the requested output format.

Furthermore, certain parameters are required for all requests. Most importantly, every request must entail a key parameter, indicating the API key. Second, all search places requests take an input parameter that identifies the search target and an inputtype parameter that identifies the type of input given in the input parameter. For place requests, the inputtype parameter can take the values textquery and phonenumber.

Nearby requests take a location parameter setting longitude and latitude of the requested place as well as a radius parameter. Detail request, however, take a mandatory parameter place_id, which indicates the place for which the details are requested.

Additionally, different optional parameters can be used. These entail a language parameter, a fields parameter indicating the types of place data to return and more.

An examples for an API request for pizza places in Mannheim can look like this:

https://maps.googleapis.com/maps/api/place/textsearch/xml?query=
pizza&location=49.487459,8.466039&radius=5000&key=YOUR_API_KEY

5.4 API access

- *How can we access the API from R (httr + other packages)?*

Instead of typing the API request into our browser, we can use the httr package's GET function to access the API from R.

```
# Option 1: Accessing the API with base "httr" commands
library(httr)

key <- "YOURAPIKEY"

res<-GET("https://maps.googleapis.com/maps/api/place/textsearch/json?", query = list(
           query = "pizza",
           location = "49.487459,8.466039",
           radius = 5000,
           key = key
         ))
```

Alternatively, we can use a wrapper function for R provided by the R-Package googleway.

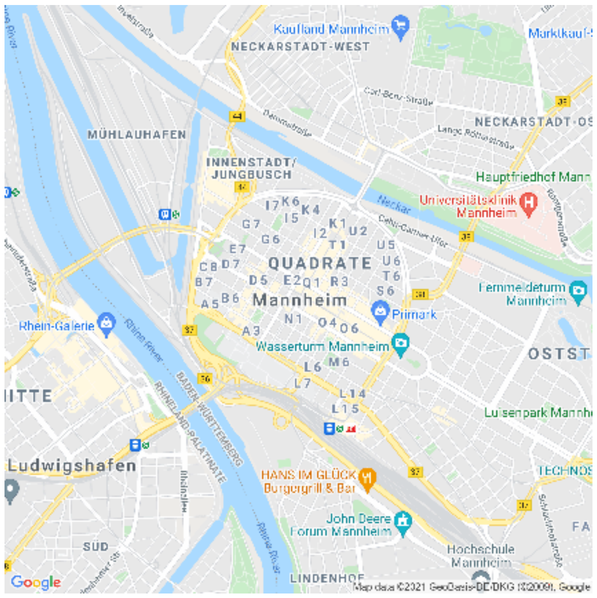
Authentication

```
key <- "YOURAPIKEY"
set_key(key)
```

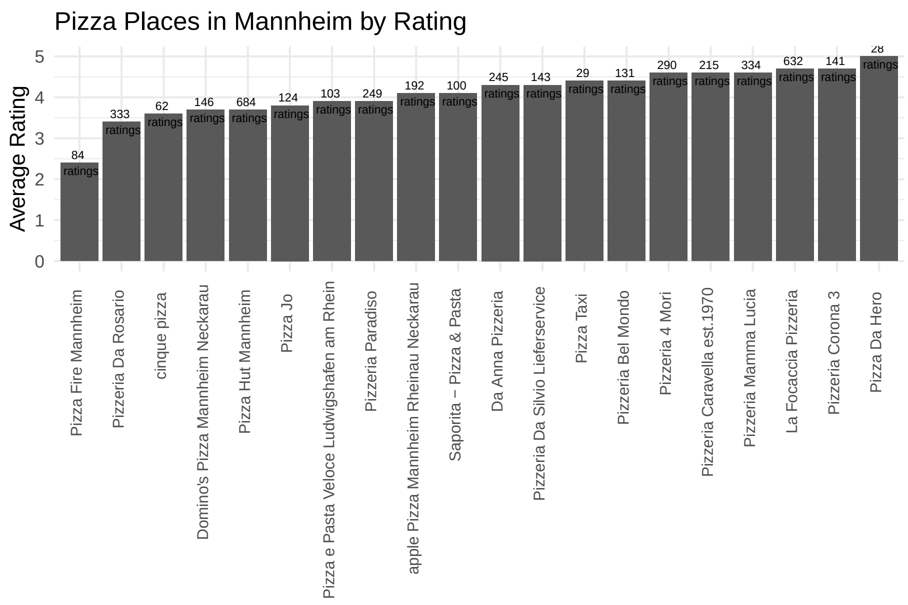
API call

```
# Option 2: Accessing the API with googleway

library(ggplot2)
library(tidyverse)
library(googleway)

# Request 'Mannheim' to get latitude and longitude information
location <- googleway::google_places("Mannheim")


# Save latitude and longitude information in vector
location <- location$results$geometry
location <- c(location$location$lat, location$location$lng)


# Plot places to google map
```

```
library(mapsapi)
# for this you will also need to activate the "maps static API"
r = mapsapi::mp_map(center = ("49.48746,8.466039"), zoom = 14, key = key, quiet = TRUE)
library(stars)
plot(r)
```



```
# Google places request with googleway
pizza <- google_places("Pizza", location = location, radius = 5000, place_type = "food"

# Plot rankings as barplot
pizza$results %>%
  ggplot() +
  geom_col(aes(x = reorder(name, rating), y = rating)) +
  geom_text(aes(x = reorder(name, rating), y = rating),
              label = paste0(pizza$results$user_ratings_total, " \n ratings"), size =
  ylab("Average Rating")+
  xlab("") +
  ggtitle("Pizza Places in Mannheim by Rating") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 90, size = 8, hjust=0.95,vjust=0.2))
```



```
# Plot pizza places to google map
#important: in order to display the map correctly, you will also have to enable the Maps JavaScr
# unfortunately we can not display an intercative card in this document, but check out the below

map<-googleway::google_map(location = location)
googleway::add_markers(map, data = pizza$results$geometry$location)
```

5.5 Social science examples

- *Are there social science research examples using the API?*

In his study "Using Google places data to analyze changes in mobility during the COVID-19 pandemic", **?** looked at the "popular times" data provided by Google Places to measure the effect of social distancing effort on mobility.

Chapter 6

Google Speech-to-Text API

Camille Landesvatter

6.1 Provided services/data

- *What data/service is provided by the API?*

Google's Speech-to-Text API allows you to convert audio files to text by applying powerful neural network models. Audio content can be transcribed in real time and of course (and possibly of higher relevance for social science research) from stored files.

The API currently recognizes more than 125 languages. It supports multiple audio formats, and audio files can either be transcribed directly (if the content does not exceed 60 seconds) or perform asynchronous requests for audio files longer than 60 seconds.

A demo of the API that allows you to record text via your microphone (or to upload an audio file) and explore the transcript can be found here.

Also consider that there is a Text-to-Speech API - simply performing operations the other way around - offered by Google.

6.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

To access and to use the API the following steps are necessary:

- Create a google account (if you do not already have one).
- With this google account login to the google cloud platform and create a Google Cloud Project.
- Within this Google Cloud Project enable the Google Speech-to-text API.
- For authentication you will need to create an API key (which you additionally should restrict to the Translation API). If however, you are planning to request the Natural Language API from outside a Google Cloud environment (e.g., R) you will be required to use a private (service account) key. This can be achieved by creating a service account which in turn will allow you to download your private key as a JSON file (we show an example below).

6.3 Simple API call

- *What does a simple API call look like?*

Note. For both Google's Translation API as well as Google's Natural-Language API, in this review we demonstrate an example for a simple API call via the Google Cloud Shell. In principle (and in a very similar procedure) this can be achieved for the Speech-to-Text API. However, your audio file will need some pre-processing. Audio data (such as our exemplary file in wav-format) is binary data. To make your REST request (via the Google Cloud Shell) however JSON is used. JSON eventually does not support binary data which is why you will have to transform your binary audio file into text using Base64 encoding (also refer to this documentation from the Google Website for more information). If you enter audio data which is not Base64 encoded, the Google Cloud Shell will give you an error 400 stating that Base64 decoding failed for your (wav-)file. Nevertheless, in the box below we will provide the basic structure of the request.

- To activate your Cloud Shell, inspect the upper right-hand corner of your Google Cloud Platform Console and click the icon called "Activate Shell". Google Cloud Shell is a command line environment running in the cloud.
- Via the built-in Editor in Cloud Shell create a JSON file (call it for instance 'request.json'). You can either upload your audio file directly via the Google Cloud Shell (search for the three-dotted "More" menu in the Shell and select "Upload file"), alternatively audio content can be integrated with Cloud Storage.
- The wav.file we uploaded for this example is an exemplary wav.file that comes along with the 'googleLanguageR' R package.

```
{
  "audio": {
    "content": "woman1_wb"
  },
  "config": {
    "enableAutomaticPunctuation": true,
    "encoding": "LINEAR16",
    "languageCode": "en-US",
    "model": "default"
  }
}
```

- For sending your data, pass a curl command to your Cloud Shell command line where you refer (via @) to your request.json file from the previous step.
- Don't forget to insert your individual API key (alternatively, you could define it beforehand via a variable in your environment -> see example in the API call for Google's NLP API later in this document).

```
curl "https://speech.googleapis.com/v1p1beta1/speech:recognize?key=APIKEY" -s -X POST -H "Content
```

6.4 API access

- *How can we access the API from R (httr + other packages)?*

Example using R-Package 'googleLanguageR'

In this small example we demonstrate how to..

*.. authenticate with your Google Cloud Account within R

*.. how to import an exemplary audiofile from the "GoogleLanguageR" package

*.. how to transcribe this audio file and calculate a confidence score

For the usage of further arguments, also read the `gl_speech()` documentation and this vignette.

1. Load packages

```
library(tidyverse)
library(googleLanguageR)
```

Step 2: Authentication

```
gl_auth("./your-key.json")
```

Step 3: Analysis

We will now get a sample source file which comes along with the `googleLanuageR` package. The transcript of this file is: "To administer medicine to animals is frequently a very difficult matter, and yet sometimes it's necessary to do so" - which according to **?** (one of the authors of the 'googleLanguageR' R package) is a fairly difficult sentence for computers to parse.

```
test_audio <- system.file("woman1_wb.wav", package = "googleLanguageR")
```

We can now call the API via the function `gl_speech()`. Here you will have to specify the quantity of interest, namely the `audio_source` (this can either be a local file or a Google Cloud Storage URI) as well as the `languageCode` (language spoken in your audio file).

```
audio_data <- gl_speech(audio_source=test_audio, languageCode = "en-GB")
```

The result is a list containing two dataframes: `transcript` and `timings`.

```
dimnames(audio_data$transcript)
```

```
## [[1]]
## [1] "1"
##
## [[2]]
## [1] "transcript"   "confidence"   "languageCode" "channelTag"
```

The `timings` dataframe stores timestamps telling us when each specific term was recognised. The `transcript` dataframe importantly provides the transcript as well as a confidence score. We can see that the transcript misses one term ("a") and indicates its confidence with a score close to 1.0.

```
audio_data$transcript$transcript
```

to administer medicine to animals is frequently very difficult matter and yet sometimes it's necessary to do so

```
audio_data$transcript$confidence #0.92
```

```
## [1] "0.9151855"
```

6.5 Social science examples

- *Are there social science research examples using the API?*

Similar to our note on social science research examples for Google's Translation API, we are not aware of research that made explicit usage of Google's Speech-to-text API. However, and especially in combination with the Translation API, we are convinced that speech-to-text conversion can be of great advantage for all kinds of qualitative or mixed-methods research projects.

Chapter 7

Instagram Graph API

Philipp Kadel

7.1 Provided services/data

- *What data/service is provided by the API?*

The Instagram Graph API is provided by Facebook. There are two main APIs for Instagram, the Instagram Basic Display API and the Instagram Graph API. The latter is described in the following.

The API can be used to get and manage published photos, videos, and stories as well as getting basic data about other Instagram Business users and Creators. It is also possible to moderate comments and their replies and to measure media and profile interaction. Photos and videos can be published directly from the API. It can also be used to discover hashtagged media and mentions.

For photos and videos different metrics can be obtained:

- engagement – Total number of likes and comments on the media object.
- Impressions – Total number of times the media object has been seen.
- Reach – Total number of unique accounts that have seen the media object.
- Saved – Total number of unique accounts that have saved the media object.
- Video_views – (Videos only) Total number of times the video has been seen. Returns 0 for videos in carousel albums.

Likewise, there are several metrics about stories that are provided by the API:

- Exits – Number of times someone exited the story.

- Impressions – Total number of times the story has been seen.
- Reach – Total number of unique accounts that have seen the story.
- Replies – Total number of replies to the story.
- Taps_forward – Total number of taps to see this story's next photo or video.

7.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

For most endpoints you need an Instagram Business Account, a Facebook Page that is connected to that account, a Facebook Developer Account and a Facebook App with Basic settings configured. Facebook provides a tutorial for setting this up here.

7.3 Simple API call

- *What does a simple API call look like?*

Below you can find expamples of simple API calls for the Instagram Graph API.

Get Fields and Edges on an IG Media. Fields can be e.g., "caption", "comments_count", "like_count", or "timestamp".

- GET ("https://graph.facebook.com/v10.0/%7Big-media-id%7D ?fields={fields} &access_token={access-token}")

Example:

```
GET ("https://graph.facebook.com/v10.0/17895695668004550
      ?fields=id,media_type,media_url,owner,
      timestamp&access_token=IGQVJ...")
```

Response:

```
{
  "id": "17895695668004550",
  "media_type": "IMAGE",
  "media_url": "https://fb-s-b-a.akamaihd.net/h-ak-fbx/t51.2885-9/21227247_16409624126
  "owner": {
    "id": "17841405822304914"
  },
  "timestamp": "2017-08-31T18:10:00+0000"
```

Return Fields and Edges on an IG Hashtag. Field can be the name of the hashtag without the “#” symbol or a hashtag ID.

- GET (“https://graph.instagram.com/%7Big-hashtag-id%7D ?fields={fields} &access_token={access-token}”)

Example:

```
GET ("https://graph.facebook.com/17841593698074073
?fields=id,name
&access_token=EAADd...")
```

Response:

```
{ "id": "17841593698074073",
  "name": "coke" }
```

Get fields and edges on an Instagram Business or Creator Account. Fields can be e.g., “biography”, “id”, “followers_count”, or “media_count”.

- GET (“https://graph.facebook.com/v10.0/%7Big-user-id%7D?fields= %7Bfields%7D &access_token={access-token}”)

Example:

```
GET ("https://graph.facebook.com/v3.2/17841405822304914
      ?fields=biography%2Cid%2Cusername%2Cwebsite&access_token=EAACwX...")
```

Response:

```
{  "biography": "Dino data crunching app",
  "id": "17841405822304914",
  "username": "metricsaurus",
  "website": "http://www.metricsaurus.com/" }
```

7.4 API access

- *How can we access the API from R (httr + other packages)?*

The `httr` package can be used to access the Instagram Graph API. There used to be a `instaR` package but it was made for the old Instagram API and can not be used anymore. The `FBinsightsR` package provides access to the Insights API. Its `fbins_insta` function can be used to collect Instagram insights. Detailed information on the packages’ functions can be found here and more information on the deprecated `instaR` package here.

7.5 Social science examples

- *Are there social science research examples using the API?*

In their study, **?** tried to infer personality traits from the way users take pictures and apply filters to them. The authors found distinct picture features (e.g., hue, brightness, saturation) that are related to personality traits. **?** investigated the link between acute suicidality and language use as well as activity on Instagram. Differences in activity and language use on Instagram were not associated with acute suicidality. The goal of a study by **?** was to automatically detect and predict incidents of cyberbullying on Instagram. Based on a sample data set consisting of Instagram images and their associated comments, media sessions were labeled for cyberbullying. Associations are investigated between cyberbullying and a host of features such as cyber aggression, profanity, social graph features, temporal commenting behavior, linguistic content, and image content.

Chapter 8

Instagram Basic Display API

Madleen Meier-Barthold

Initially released in 2010, Instagram currently counts 1+ billion monthly active users with 50+ billion photos stored (**?**). User engagement is high, with an average of 28 minutes per day spent on the platform in 2020 (**?**). Needless to say, the photo and video sharing social networking service holds invaluable data that could be leveraged by social researchers.

8.1 Provided services/data

- *What data/service is provided by the API?*

Instagram offers two types of APIs that allow an application to access data on the platform: the Instagram Graph API and the Instagram Basic Display API.

Previously, other APIs were available which allowed developers and researchers a less restricted access to data collection (**?**). These older APIs are now depreciated.

The Instagram Basic Display API gives read-access to basic profile information, photos and videos on authenticated users' accounts (**?**). Particularly, it is possible to fetch a user's profile, including fields like account type and account name, as well as a user's media (images, videos and albums), including fields like media type, caption, URL and timestamp. The API does not allow to modify data like publishing media or moderating comments (see Instagram Graph API).

It is a RESTful API, meaning that queries are made for static information at the current moment. Queries are subject to rate limits. Responses are in the form of JSON-formatted objects containing the default and requested fields and edges.

8.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

In order to access the Instagram Basic Display API, developers are required to first register as a Facebook developer on developers.facebook.com, to further create a Facebook App here and to submit the application for review.

Another prerequisite to access the API is to get authentication. API authentication is handled through Instagram User Access Tokens that conform to the OAuth 2.0 protocol. The process of getting an access token includes two parts. First, each application user must grant an application permission to read the user node and/or the media node. These permissions are controlled via an Authorization Window.

```
https://api.instagram.com/oauth/authorize
  ?client_id={appId}
  &redirect_uri={redirectURI}
  &scope=user_profile,user_media
  &response_type=code
```

Give this URL to the application users. The next steps have to be completed by each user.

1. Open a new browser window and load the Authorization Window URL.
2. Authenticate your Instagram test user by signing into the Authorization Window
3. Click Authorize to grant your app access to your profile data.
4. Upon success, the page will redirect you to the redirect URI you included in the previous step and append an Authorization Code. For example: https://mmeierba.github.io/?code=AQD...#_
5. Copy the code except for the #_ portion in the end. Send this Authorization Code to researcher.

When a user successfully grants the application permission to access their data, the user is redirected to a redirect URI which appended with an Authorization Code. Second, the Authorization Code can be exchanged for a short-lived access token (i.e., valid for 1 hour).

Then, the API can be queried.

```
library(httr)

appId = "126..." #use Instagram App ID and secret (not Facebook App)
appSecret = "b73..."
redirectUri = "https://mmeierba.github.io/" #example
code = "AQD..."

id<-POST("https://api.instagram.com/oauth/access_token",
     body=list(client_id=appId, client_secret=appSecret, grant_type="authorization_code", redirec
```

8.3 Simple API call

- *What does a simple API call look like?*

The Instagram Basic Display API is http-based. To query a node or edge, a GET call can be used. The base URLs are api.instagram.com and graph.instagram.com.

```
accessToken = "..."
userId = "..."

## Query the user node
GET("https://graph.instagram.com/userId?fields=id,username&access_token=accessToken")
GET("https://graph.instagram.com/me?fields=id,username&access_token=accessToken") #alternative


## Query the user media edge
GET("https://graph.instagram.com/me/media?fields=id,caption&access_token=accessToken")

## Query the user media node
mediaId = "..."

GET("https://graph.instagram.com/mediaId?fields=id,media_type,media_url,username,timestamp&access
```

8.4 API access

- *How can we access the API from R (httr + other packages)?*

The Instagram Graphic Display API can be accessed from R using the httr package.

Current R packages specific to APIs from Instagram (e.g., instaR) are related to deprecated versions of Instagram APIs and are therefore no longer useful in the current version (**?**). To the knowledge of the authors, there are no R packages specific to the Instagram Basic Display API.

- *Are there social science research examples using the API?*

There are a couple of examples of studies in the field of social science that have used an Instagram API. However, the presented examples use older, depreciated versions of the Instagram API.

? collected 50 user profiles and their most recent photos, as well as users' lists of friends and followers using the Instagram API. Analyzing the data, the authors identified a range of photo categories and types of Instagram users.

? used the Instagram API to extract Instagram pictures from survey participants, who had previously filled in a personality questionnaire. 113 survey participants granted the researchers access to their Instagram accounts through the API. The authors found that distinct features of the Instagram pictures (i.e., hue, brightness, saturation) associated with users' personality traits.

These examples show the potential that extracting data using the Instagram Basic Display API has for social science researchers. Yet, a lot of data are not yet being leveraged (e.g., captions).

Chapter 9

GoogleTrends API

Jan Behnert, Dean Lajic

9.1 Provided services/data

- *What data/service is provided by the API?*

The API is provided by Google.

With Google Trends, one gets access to a largely unfiltered sample of actual search topics (up to 36h before your search) and a filtered and representative sample for search topics older than 36 hours starting from the year 2004. The data is anonymized, can be obtained from different Google products like "Web search", "News", "Images", "Shopping" and "Youtube," can be filtered by different categories to get the data for the correct meaning of the word, and is aggregated, which means that the searches of all cities/regions are aggregated to the federal state level, country level or world level. The results you get are a standardized measure of search volume for single search terms, a combination of search terms using operators (see table below), or comparisons (one input in relation to the other inputs) over a selected time period. Google calculates how much search volume in each region a search term or query had, relative to all searches in that region. Using this information, Google assigns a measure of popularity to search terms (scale of 0 - 100), leaving out repeated searches from the same person over a short period of time and searches with apostrophes and other special characters.

No quotation marks (e.g. Corona symptoms)	You get results for each word in your query

Quotation marks (e.g. “Corona symptoms”)	You get results for the coherent search phrase
Plus sign (e.g. corona +covid)	Serves as function of an OR-operator
Minus sign (e.g. corona -symptoms)	Excludes word after the operator

9.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

It can be used without an API key by anyone for free directly in the internet browser (no sign up needed).

9.3 Simple API call

- *What does a simple API call look like?*

Just click here.

9.4 API access

- *How can we access the API from R (httr + other packages)?*

Example using “httr” package:

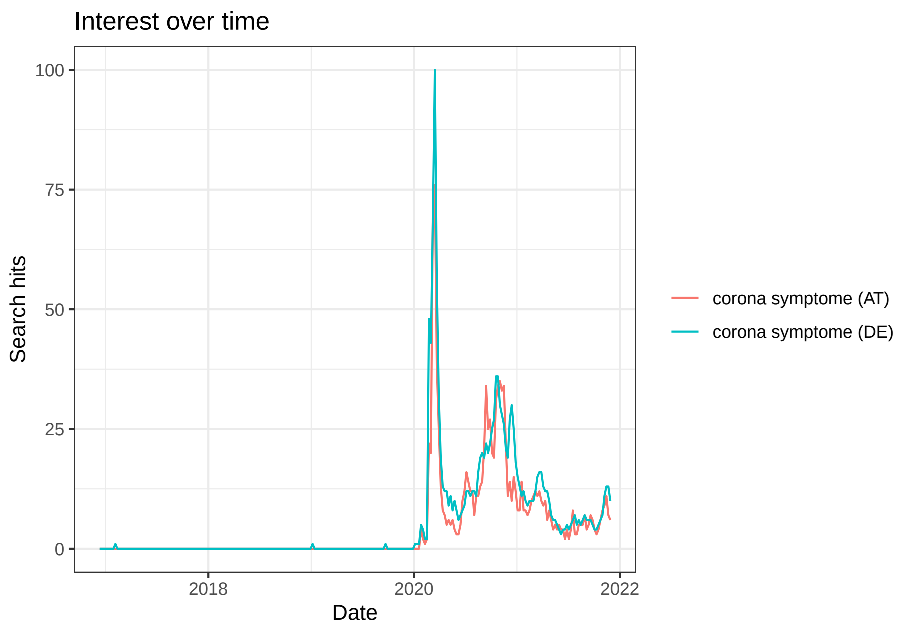
```r
library(httr)
GET("https://trends.google.com/trends/explore",
    query=list(q = "Covid",geo = "US"))
```

- but just html-output, we recommend to use the gtrendsR package

Example using “gtrendsR” package:

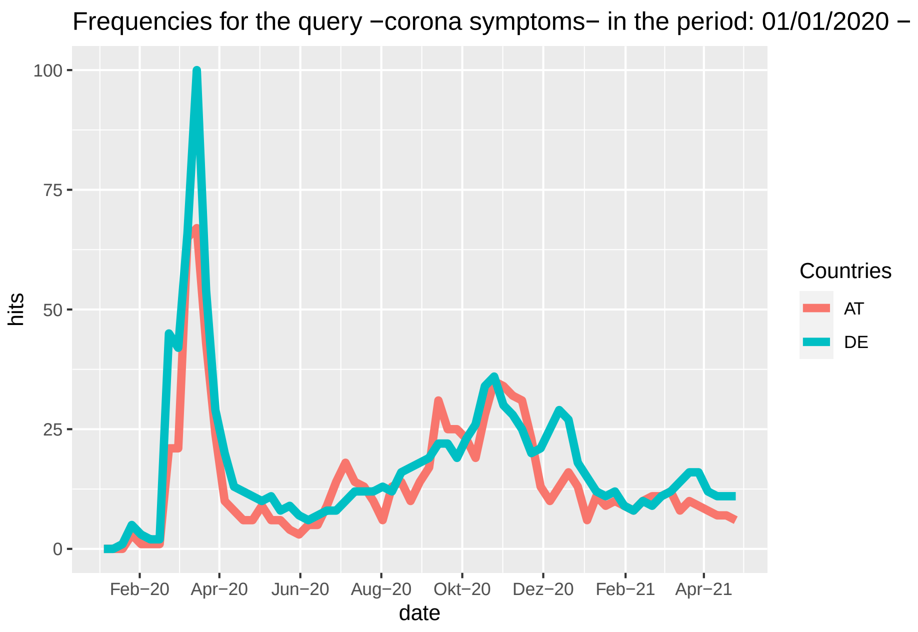
```r
# visualizing google searches for the word "corona symptoms" in
# Germany and Austria in the period 01/01/2020 - 27/04/2021
library(gtrendsR)
library(ggplot2)
library(dplyr)
```

```
data("countries") # get abbreviations of all countries to filter data
data("categories") # get numbers of all categories to filter data

# Simple call

res <- gtrends("corona symptome",geo=c("DE", "AT"))
plot(res)
```



- Note (1): the use of c() in the keyword argument of the gtrends function allows comparisons of up to 5 searches (separator = comma).
- Note (2): the use of the pattern '"xyz"' in the keyword argument of the gtrends function corresponds to the inverted commas in the table above, all other punctuation methods in the table above can be used as indicated in the table.

```
#Combination using dplyr and ggplot
trend = gtrends(keyword="corona symptome", geo=c("DE", "AT"), time = "2020-01-01 2021-04-27", gpr

trend_df <- trend$interest_over_time

trend_df <- trend_df %>%
    mutate(hits = as.numeric(hits), date = as.Date(date)) %>%
    replace(is.na(.), 0)

ggplot(trend_df, aes(x=date, y=hits, group=geo, col=geo)) + geom_line(size=2) +
scale_x_date(date_breaks = "2 months" , date_labels = "%b-%y") +
labs(color= "Countries") +
ggtitle("Frequencies for the query -corona symptoms- in the period: 01/01/2020 - 27/04/
```



9.5 Social science examples

- *Are there social science research examples using the API?*

Google Trends can be used to predict the outcomes of elections. For example a study by (**?**) uses Google Trends data to predict the past four elections in the United States and the past five in Canada, since Google first published its search statistics in 2004. They analysed which candidate had the most Google searches in the months leading up to election day and show, that with the help of this data, all actual winners in all the elections held since 2004 could be predicted.
Another example is a study by **?** which uses Google Trends data to predict the results of referendums (Scottish referendum 2014, Greek referendum 2015, British referendum 2016, Hungarian referendum 2016, Italian referendum 2016 and the Turkish referendum 2017). It can be shown that the results from Google

Trends data are quite similar to the actual referendum results and in some cases are even more accurate than official polls. It is argued that with the help of Google Trends data revealed preferences instead of users' stated preferences can be analyzed and this data source could be a helpful source to analyze and predict human behavior (given areas where the Internet is widely accessible and not restricted).
Furthermore, Google Trends data can also be utilized in other fields, for example to examine whether COVID-19 and the associated lockdowns initiated in Europe and America led to changes in well-being related topic search-terms. The study by **?** finds an increase in queries addressing boredom, loneliness, worry and sadness, and a decrease for search terms like stress, suicide and divorce. Indicating that the people's mental health could have been strongly affected by the pandemic and the lockdowns.

Chapter 10

Youtube API

Melike Kaplan, Jana Klein

10.1 Provided services/data

- *What data/service is provided by the API?*

The API is provided by Google, Youtube's parent company.

There are different types of Youtube APIs that serve different purposes:

1. **YouTube Analytics API:** retrieves your YouTube Analytics data.
2. **YouTube Data API v3:** provides access to YouTube data, such as videos, playlists, and channels.
3. **YouTube oEmbed API:** oEmbed is an elegant way to embed multimedia for a link.
4. **YouTube Reporting API:** Schedules reporting jobs containing your YouTube Analytics data and downloads the resulting bulk data reports in the form of CSV files.

The google developer site provides sample requests and a summary of the possible metrics that the API can give you data on. You can actually run your API requests there. All the possible calls you can make are provided on the page: Captions, ChannelBanners, Channels, ChannelSection, Comments, CommentThreads, i18nLanguages, i18nRegrions, Members, MembershipLevels, Playlistitems, Playlists, Search, Subscriptions, Thumbnails, VideoAbuseReportReasons, VideoCategories, and Videos.

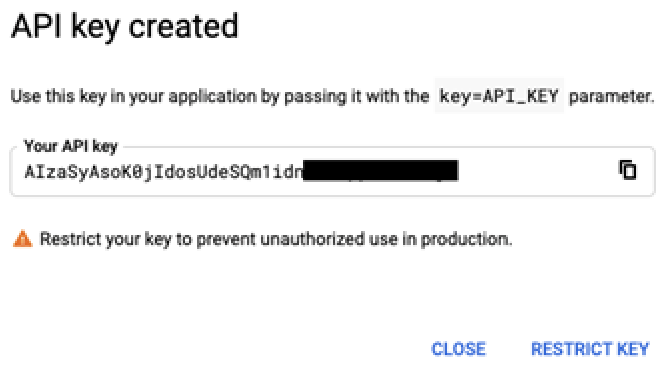
10.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

First, you will need a Google account which you will use to log into the Google Cloud Platform. You will need to create a new project unless you already have one (here you can find more information). Then, you can search for the four Youtube APIs (YouTube Analytics API, YouTube Data API v3, YouTube oEmbed API, YouTube Reporting API) mentioned above and enable them (here you can find more information).

Then, continue to the "APIs and Services" Page from the sidebar and click on "Credentials." Click on "+ Create Credentials" at the top of the page. You have three options here: API Key, OAuth client ID or Service account. An API Key will identify your project with a simple key to check quota and access but if you wish to use the YouTube API for your app, you should create an OAuth client ID which will request user consent so that your app can access the user's data. This is also necessary when you want to use the tuber package. A Service account enables server to server, app-level authentication using robot accounts. We will continue with the option of creating an API Key, and later we provide an example of using the OAuth Client ID with the tuber package.

When you click on "API Key" in the "+Create Credentials" list, a screen will appear like below:



API key created

Use this key in your application by passing it with the `key=API_KEY` parameter.

Your API key

AIzaSyAsoK0jIdosUdeSQm1idn[redacted]

Restrict your key to prevent unauthorized use in production.

Your key is created! It is important to restrict the key!

10.3 Simple API call

- *What does a simple API call look like?*

The base URL is https://www.googleapis.com/youtube/v3/. With the following API call we tried to get the channel statistics from the SWR youtube channel. The channel statistics include information on the viewer count, subscriber count, whether there are hidden subscribers and on the video count. https://youtube.googleapis.com/youtube/v3/channels?part=statistics&id=UCy4_zQ59zmS7zO4Dc6vbT_w&key=%5BYour_API_Key%5D However, this call did not work for us, we got an error code 400 that said that our API key is not valid.

10.4 API access

- *How can we access the API from R (httr + other packages)?*

Example to get channel statistics:

```
library(httr)
library(jsonlite)
library(here)
library(dplyr)
library(ggplot2)
```

```
#save your API key in the object key
key<-"Your_API_Key"

#YouTube channels either have a channel id or a user id
ZDF_Magazin_Royle<-"UCNNEMxGKV1LsKZRt4vaIbvw" #channel id
Boilerroom <- "brtvofficial" #user id

#save the base URL in the object base
base<- "https://www.googleapis.com/youtube/v3/"

#get channel info with channel id
api_params <-
  paste(paste0("key=", key),
        paste0("id=", ZDF_Magazin_Royle),
        "part=snippet,contentDetails,statistics",
        sep = "&")
api_call <- paste0(base, "channels", "?", api_params)
api_result <- GET(api_call)
json_result <- content(api_result, "text", encoding="UTF-8")

#format json into dataframe
channel.json <- fromJSON(json_result, flatten = T)
channel.df <- as.data.frame(channel.json)
```

```
#example with a username
api_params2 <-
  paste(paste0("key=", key),
        paste0("forUsername=", Boilerroom),
        "part=snippet,contentDetails,statistics",
        sep = "&")
api_call2 <- paste0(base, "channels", "?", api_params2)
api_result2 <- GET(api_call2)
json_result2 <- content(api_result2, "text", encoding="UTF-8")

#format json into dataframe
channel.json2 <- fromJSON(json_result2, flatten = T)
channel.df2 <- as.data.frame(channel.json2)
```

On CRAN we found the “tuber” package. The package enables you to get the comments posted on YouTube videos, number of likes of a video, search for videos with specific content and much more. You can also scrape captions from a few videos. To be able to use the tuber package, not an API key but the authentication with OAuth is necessary. OAuth (Open Authorization) uses authorization tokens to prove an identity between consumers and service providers. You can get your client ID and secret on the Google Cloud Platform under Credentials.

1) Setting up the “Consent Screen”
   First we had to configure the so-called OAuth consent screen, where we put “external” and then had to put an app name. For scopes we did not specify anything and just clicked save & continued. To be able to use the API you have to set your own google mail address that you use for the Google cloud.

2) Get OAuth credentials
   After setting up the consent screen you can go back and click “create credentials” and add a “OAuth client ID”. As a result you get an OAuth client id and secret. You can download this information stored in a JSON file. With the Yt_oauth function you can then authenticate yourself. This will forward us to logging into our google account. Allow the access to your google account. (like with google bigquery).

This page provides some example API calls you can make with the tuber package.

```
library(tuber) # youtube API
library(magrittr) # Pipes %>%, %T>% and equals(), extract().
library(tidyverse) # all tidyverse packages
library(purrr) # package for iterating/extracting data
```

```
#save client id and secret in an object
client_id<-"put client ID"
client_secret<-"put client secret"

# use the youtube oauth
yt_oauth(app_id = client_id,
         app_secret = client_secret,
         token = '')

#Downloading playlist data
#first get playlist ID
go_psych_playlist_id <- stringr::str_split(
  string = "https://www.youtube.com/playlist?list=PLD4cyJhQaFwWZ838zZhWVr3RG_nETlICM",
  pattern = "=",
  n = 2,
  simplify = TRUE)[ , 2]
go_psych_playlist_id

#use the tuber function get_playlist_items to collect the videos into a data frame

go_psych <- tuber::get_playlist_items(filter =
                                                c(playlist_id = "PLD4cyJhQaFwWZ838zZhWVr3RG_nETlI
                                              part = "contentDetails",
                                              # set this to the number of videos
                                              max_results = 200)

# check the data for go Psych
#now we have the video ids of all videos in that playlist
go_psych %>% dplyr::glimpse(78)
```

Package information:
* CRAN - Package tuber
* here you can find all the functions that the tuber package provides

10.5 Social science examples

- *Are there social science research examples using the API?*

In the study "Identifying Toxicity Within YouTube Video Comment" (**?**), the researchers utilized the YouTube Data API to collect the comments from eight YouTube channels that were either pro- or anti NATO. To the comments, five types of toxicity scores were assigned to analyze hateful comments. With word clouds the researchers were able to quantify the count of words from comments.

The final dataset contained 1,424 pro-NATO videos with 8,276 comments, and 3,461 anti-NATO videos with 46,464 comments.

The aim of the study “YouTube channels, uploads and views: A statistical analysis of the past 10 years” (?) was to give an overview on how YouTube developed over the past 10 years in terms of consumption and production of videos. The study utilizes a random sample of channel and video data to answer the question. The data is retrieved with the YouTube API (did not specify which one) combined with a tool that generated random string searches to find a near-random sample of channels created between 01.01.2016 and 31.12.2016. Results are that channels, views and video uploads differ according to video genre. Furthermore, the analysis revealed that the majority of views are obtained by only a few channels. On average, older channels have a larger amount of viewers.

In the study “From ranking algorithms to ‘ranking cultures’: Investigating the modulation of visibility in YouTube search results” (?), YouTube is conceptualized as an influential source of information that uses a socio-algorithmic process in order to place search recommendations in a hierarchy. This process of ranking is considered to be a construction of relevance and knowledge in a very large pool of information. Therefore, the search function serves as a curator of recommended content. The information that is being transmitted in this content can also impose certain perspectives on users which is why how the algorithm works is especially important when it comes to controversial issues. In order to better understand how the algorithms that determine search rankings on YouTube work, the authors use a scraping approach and the YouTube API v3 to study the ranking of certain sociocultural issues over time. Examples of the keywords that they use are ‘gamergate,’ ‘trump,’ ‘refugees’ and ‘syria.’ They find three general types of morphologies of rank change.

Chapter 11

CKAN API

Barbara K. Kreis

The CKAN API is an API offered by the open-source data management system (DMS) CKAN (Open Knowledge Foundation). Currently, CKAN is used as a DMS by many different users, governmental institutions and corporations alike. This API review will focus on the use of the CKAN API to access and work with open government data. As the CKAN DMS is used by various governments to offer open datasets, it is a helpful tool for researchers to access this treasure of publicly open information. CKAN hosts free datasets from various governments, such as from Germany, Canada, Australia, the Switzerland and many more.

11.1 Provided services/data

- *What data/service is provided by the API?*

All of CKAN's core features can be accessed via the CKAN API (Open Knowledge Foundation)
With the CKAN API, you can

- Get a JSON-formatted list of a site's objects, datasets or groups.
- Get a full JSON representation of an object, e.g. a dataset.
- Search for any packages (datasets) or resources that match a query.
- Get an activity stream of recently changed datasets on a site.

Please see the following link for more information on the services provided by the CKAN API and some specific examples.
When it comes to the specific datasets on the government sites, there are two types that can be accessed: specific datasets and meta data sets.

For example, the German and the US Government have a website each, where you can get access to metadata that include descriptions and URLs about the specific open datasets that can be accessed. These meta datasets can be a starting point for research on a specific topic.
The specific datasets include a variety of different contents from public administration, such as election results, data on schools, maps and many more. The German data portal govdata.de for example serves as a collection point for all those data from various institutions. Those specific administrative institutions are the ones that actually provide the data. Therefore, not every institution provides the same data on the same topic.

11.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

There are no prerequisites to access the CKAN API. Furthermore, there seem to be no prerequisites to access the open data from the various governmental institutions using CKAN.

11.3 Simple API call

- *What does a simple API call look like?*

When a user wants to make an API call, two use cases have to be distinguished: Calling meta-data and calling specific datasets.

Meta-datasets

When calling the meta data, the DCAT catalog has to be queried. DCAT-AP.de is a German metadata model to exchange open government data. For more information and information on the meta data structure, see this website.
The API call for the DCAT catalog can deliver three formats: RDF, Turtle and JSON-LD. The type of format can be specified at the end of the request (e.g. “format=jsonld”).
The following API call is an example for the search term “Kinder”. https://ckan.govdata.de/api/3/action/dcat_catalog_search?q=kinder&format=jsonld

Specific datasets from GovData

To look for specific datasets, not the meta data, only little has to be changed in the URL. In the case of querying specific datasets, the response format is JSON.
The following API call is an example when looking for the first 5 packages (datasets) that contain the search term “Kinder” (=children).
https://www.govdata.de/ckan/api/3/action/resource_show?q=kinder

11.4 API access

- *How can we access the API from R (httr + other packages)?*

The CKAN API can be accessed from R with the httr package or the ckanr package.
Please note that as a scientist you can only use GET requests. All kinds of POST requests are restricted to government employees that work at the institutions which provide the data sets.

```
# CKAN API #
# Option 1: Use the httr package to access the API

library(httr) # required to work with the API

# With the following query we get the same information as described in the paragraph above

base_url <- "https://www.govdata.de/ckan/api/3/action/resource_show"
berlin <- GET(base_url, query=list(q="kinder",rows=5))
```

```
# Option 2: Use the ckanr package to access the API

# load relevant packages
library(tidyverse)
library(ckanr)
library(jsonlite)
library(readxl)
library(curl)
library(readxl)

#connect to the website
url_site <- "https://www.govdata.de/ckan"
ckanr_setup(url = url_site)

# first, let's see which groups are on this site
group_list(as = "table")


#you can see there are different groups
#now we want to look at them in more detail
group_list(limit = 2)

# now you can look for the specific packages
package_list(as = "table")
```

```
# now, let's look at a specific package more closely, to get some more information
package_show("100-jahre-stadtgrun-stadtpark-und-volkspark")

# now, let's do a more specific search for specific resources (we look at Kinder = kid
x <- resource_search(q = "name:Kinder", limit = 3)
x$results

# here you get the name, the Description (not always filled out) and the data format

# now we want to have a closer look at the second resource (day care for children)
# we need to get the url, by using the resource number

url<-resource_show(id ="a8413550-bf4d-40f3-921a-941da3fce132")
url$url

# with the url, we can now import the data
url <- ("https://geo.sv.rostock.de/download/opendata/kindertagespflegeeinrichtungen/ki
destfile <- ("kindertagespflegeeinrichtungen.csv")
curl::curl_download(url, destfile)

kindertagespflegeeinrichtungen <- read_csv(destfile)
View(kindertagespflegeeinrichtungen)

# in this file, you can now for example look at the opening hours of the day cares in
```

11.5 Social science examples

- *Are there social science research examples using the API?*

When looking for social science research that used the CKAN API and Open Government data (OGD), it seems that there is more papers and research on the usage of those data, than on the data themselves (**?**, **?**). In a recent paper that examines the use of OGD (**?**), the authors come to the conclusion, that on the one hand many OGD portals lack information about data usage, and on the other hand, where those information can be found, it becomes obvious that the data are only rarely used.
For example, regarding the German OGD portal "GovData.de", I did not find any social science papers that specifically used data from GovData.de. However, there are a few papers available that describe the German open data initiative (**?**) and the metadata (**?**) that can be found on GovData.de.

Chapter 12

MediaWiki Action API

Noam Himmelrath, Jacopo Gambato

12.1 Provided services/data

- *What data/service is provided by the API?* To access *Wikipedia*, *MediaWiki* provides the MediaWiki Action API.

The API can be used for multiple things, such as accessing wiki features, interacting with a wiki and obtaining meta-information about wikis and public users. Additionally, the web service can provide access data and post changes of *Wikipedia*-webpages.

12.2 Prerequesites

- *What are the prerequisites to access the API (authentication)?*

No pre-registration is required to access the API. However, for certain actions, such as very large queries, a registration is required. Moreover, while there is no hard and fast limit on read requests, the system administrators heavily recommend limiting the request rate to secure the stability of the side. It is also best practice to set a descriptive User Agent header.

12.3 Simple API call

- *What does a simple API call look like?*

As mentioned, the API can be used to communicate with Wikipedia for a variety of actions. As it is most likely for social scientist to extract information rather than post changes to a Wikipedia page, we focus here on obtaining from Wikipedia the information we need.

We include a basic API call to obtain information about the Albert Einstein Wikipedia page

`https://en.wikipedia.org/w/api.php?action=query&format=json&prop=info&titles=Albert%20`

to be plugged into the search bar of a browser to obtain the basic information on the page.

Notice that the first line is common for all calls of the API, while the second line relates to the specific action you are trying to perform.

12.4 API access

- *How can we access the API from R (httr + other packages)?*

The most common tool is `WikipediR`, a wrapper around the Wikipedia API. It allows `R` to access information and "directions" for the relevant page or pages of Wikipedia and the content or metadata therein. Importantly, the wrapper only allows to gather information, which implies that the instrument needs to be accompanied by other packages such as `rvest` for scraping and `XML` or `jsonlite` for parsing.

We report here the code to obtain the same information as in the previous example through `R`:

```r
WikipediR::page_info(
    language = "en",
    project = "wikipedia",
    page = "Albert Einstein",
    properties = "url")
```

```
## $batchcomplete
## [1] ""
##
## $query
## $query$pages
## $query$pages$`736`
## $query$pages$`736`$pageid
## [1] 736
##
## $query$pages$`736`$ns
## [1] 0
##
## $query$pages$`736`$title
## [1] "Albert Einstein"
##
## $query$pages$`736`$contentmodel
## [1] "wikitext"
##
## $query$pages$`736`$pagelanguage
## [1] "en"
##
## $query$pages$`736`$pagelanguagehtmlcode
## [1] "en"
##
## $query$pages$`736`$pagelanguagedir
## [1] "ltr"
##
## $query$pages$`736`$touched
## [1] "2021-12-05T18:46:13Z"
##
## $query$pages$`736`$lastrevid
## [1] 1058773374
##
## $query$pages$`736`$length
## [1] 183211
##
## $query$pages$`736`$fullurl
## [1] "https://en.wikipedia.org/wiki/Albert_Einstein"
##
## $query$pages$`736`$editurl
## [1] "https://en.wikipedia.org/w/index.php?title=Albert_Einstein&action=edit"
##
## $query$pages$`736`$canonicalurl
## [1] "https://en.wikipedia.org/wiki/Albert_Einstein"
##
##
##
##
## attr(,"class")
## [1] "pageinfo"
```

12.5 Social science examples

- *Are there social science research examples using the API?*

Some papers using Wikipedia-information rely on the API to access the data. These papers cover a wide range of social and economical sciences. Political science papers are, for example, concerned with political elections, more specifically election prediction (**??**). Other papers use the data accessed through the API to analyze media coverage of the COVID-19 pandemic (**?**) or the interplay between online information and investment markets (**?**).

Chapter 13

References