Application Programming Interfaces

Bamberg Summer Institute in Computational Social Science

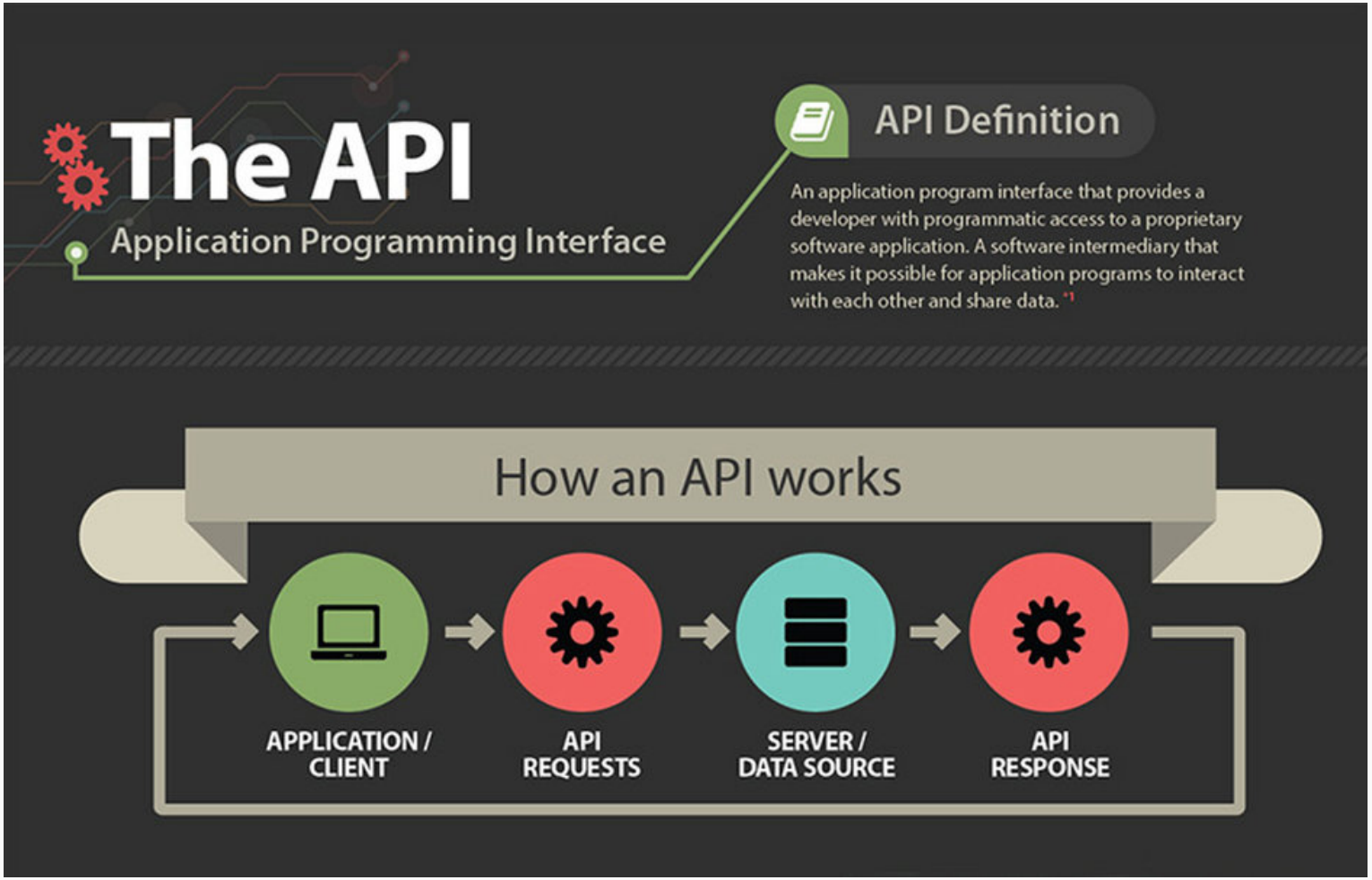
Carsten Schwemmer, University of Bamberg

2019-07-30

Many thanks to Chris Bail for providing material for this lecture

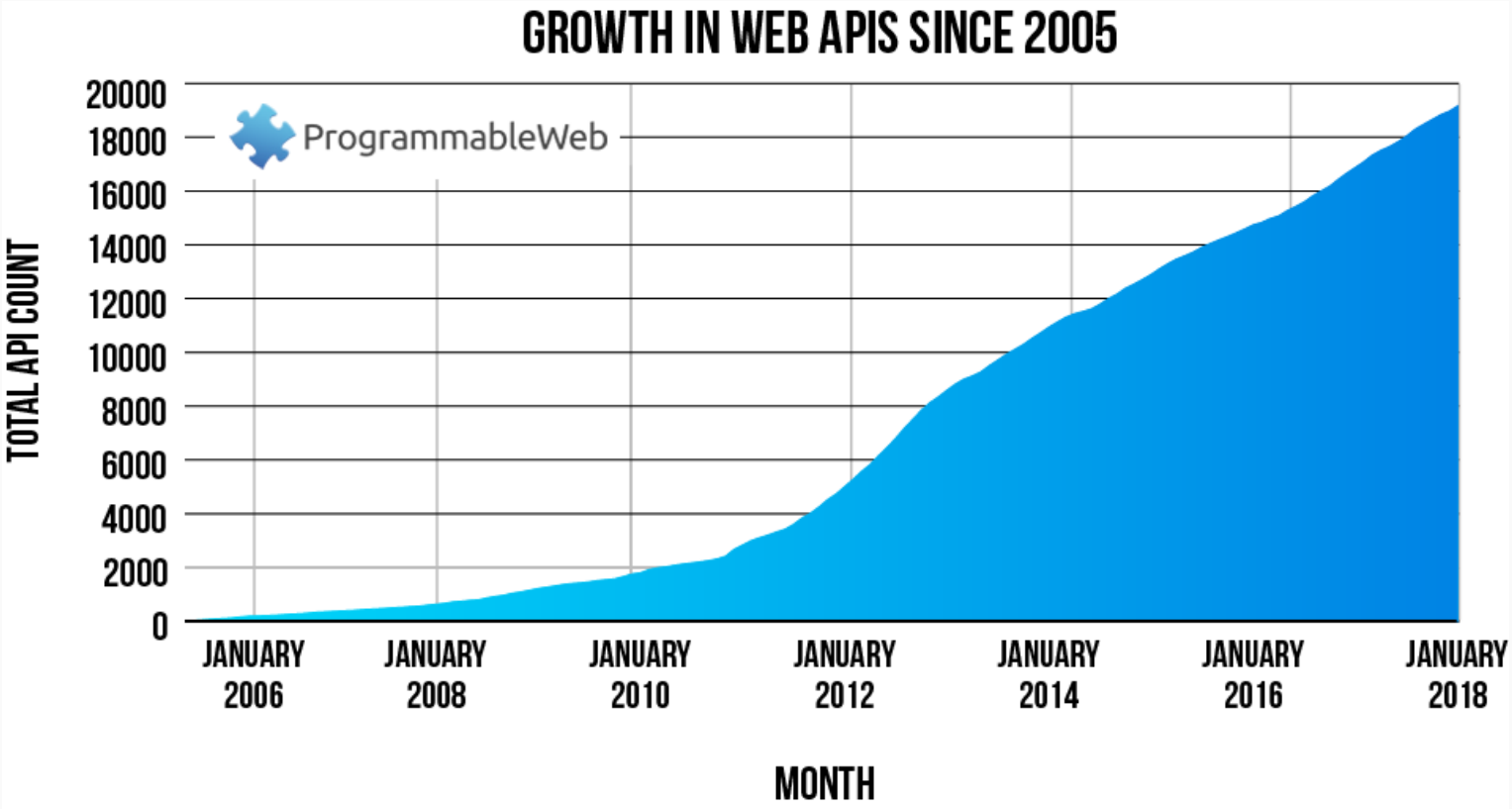
Application Programming Interfaces

What is an API?



{https://www.govtech.com/applications/Whats-an-API-and-Why-Do-You-Need-One.html}

Growth of APIs



{https://www.programmableweb.com/news/apis-show-faster-growth-rate-2019-previous-years/research/2019/07/17}

Strengths and weaknesses of APIs

Advantages:

- accessing APIs is legal and in most cases well defined(e.g. 10,000 requests per day)
- we do not have to apply scraping procedures for unstructured data (e.g. web pages)
- for some APIs R packages make it very easy to collect data (e.g. `rtweet`)

Disadvantages:

- we first have to learn how to use the API (some documentation are good, others are bad)
- Many APIs require authenification and / or are not available for free

Working with APIs in R

Required packages

```r
install.packages(c('usethis', 'httr', 'rtweet'))
```

Example - Guardian API

`http://open-platform.theguardian.com/explore`

- the Guardian API allows to retrieve meta data and full texts from articles of the british newspaper in structured `JSON`format.
- we can access data from different endpoints (see documentation), e.g. `/content`

Example - Guardian API explorer

vegan

Search content

Add filters...

Add API key...

Show All Filters Help Feedback

https://content.guardianapis.com/search?q=vegan&api-key=test


```
{
  - response: {
        status: "ok"
        userTier: "developer"
        total: 3570
        startIndex: 1
        pageSize: 10
        currentPage: 1
        pages: 357
        orderBy: "relevance"
      - results: [
          - {
              id: "fashion/2019/may/04/the-best-vegan-perfumes-sali-hughes"
              type: "article"
              sectionId: "fashion"
              sectionName: "Fashion"
              webPublicationDate: "2019-05-04T07:00:08Z"
              webTitle: "The best vegan perfumes | Sali Hughes"
              webUrl: https://www.theguardian.com/fashion/2019/may/04/the-best-vegan-perfumes-sali-hughes
              apiUrl: https://content.guardianapis.com/fashion/2019/may/04/the-best-vegan-perfumes-sali-hughes
```

Guardian API

For communication with the API we need:

- an API key
- the correct base URL
- a set of parameters accoring to the *rules* of the API. Parameters will influence what kind of data the API returns.

Guardian API - credentials

To retrieve a Guardian API key, fill out the developer register form:

`https://open-platform.theguardian.com/access/`

An easy way to include api keys in R scripts is to store them as strings:

```
guardian_key <- "your_api_key"
```

If you share your script, this has the advantage that others can see (and use) your key.

Guardian API - credentials

A better way is to store the key in your R environment with `usethis::edit_r_environ()` and to reload it afterwards:

```
guardian_key <- Sys.getenv('guardian_api_key')
substr(guardian_key, 1, 10) # first 10 characters of API key
```

```
## [1] "90f8720e-9"
```

Guardian API - parameters

The API document explains what parameters can be used to work with the API: `https://open-platform.theguardian.com/documentation/search`

```
base <- 'http://content.guardianapis.com/search'
params <- list('api-key' = guardian_key, # API key
               'page-size' = 50, # number of results
               'q' = 'vegan') # search query
```

Guardian API - sending a request

We will use the R package httr to retrieve Guardian articles with the `GET()` function.

```
library(httr)
guardian_request <- GET(base, query = params)
```

HTTP status code (200 means "OK"):

```
guardian_request$status_code
```

```
## [1] 200
```

Guardian API - parse JSON data

- JSON is one of the most common data formats returned by APIs
- the R package jsonlite is useful for converting `JSON` data to R data structures

```r
library(jsonlite)
json_data <- fromJSON(content(guardian_request, as = "text"))
names(json_data)
```

```
## [1] "response"
```

Guardian API - nested structures

JSON files can be nested several layers depp. For the guardian API, article data is returned under the key `results`, which in turn has to be accessed through `response` (see documentation).

```
df <- json_data$response$results
df$webTitle[1:3]
```

```
## [1] "The best vegan perfumes | Sali Hughes"
## [2] "Meera Sodha's vegan recipe for mango sticky rice"
## [3] "Greggs' vegan sausage rolls fuel profit boom"
```

Guardian API - pagination

- besides rate limits, the Guardian API (and many others) has a maximum number of items that can be retrieved with one single call
- to retrieve more data, pagination can be used inside a loop

```
json_data$response$currentPage # current page
```

```
## [1] 1
```

```
json_data$response$pageSize # items per request
```

```
## [1] 50
```

```
json_data$response$pages # total number of pages
```

```
## [1] 72
```

Guardian API - example pagination function

```r
library(tidyverse)
get_guardian_articles <- function(query, max_pages = 5) {
  # parameters
  base <- 'http://content.guardianapis.com/search'
  params <- list(
    'api-key' = guardian_key,
    'page-size' = 50,
    'page' = 1,
    'q' = query
  )
  # first request
  req <- GET(base, query = params)
  data <- fromJSON(content(req, 'text'))
  df <- data$response$results
  # remaining pages
  for (page in 2:data$response$pages) {
    if (page <= max_pages) {
      params$page <- page
      req <- GET(base, query = params)
      data <- fromJSON(content(req, 'text'))
      df <- bind_rows(df, data$response$results)
    }
  }
  return(df)
}
```

Using API-specific R packages

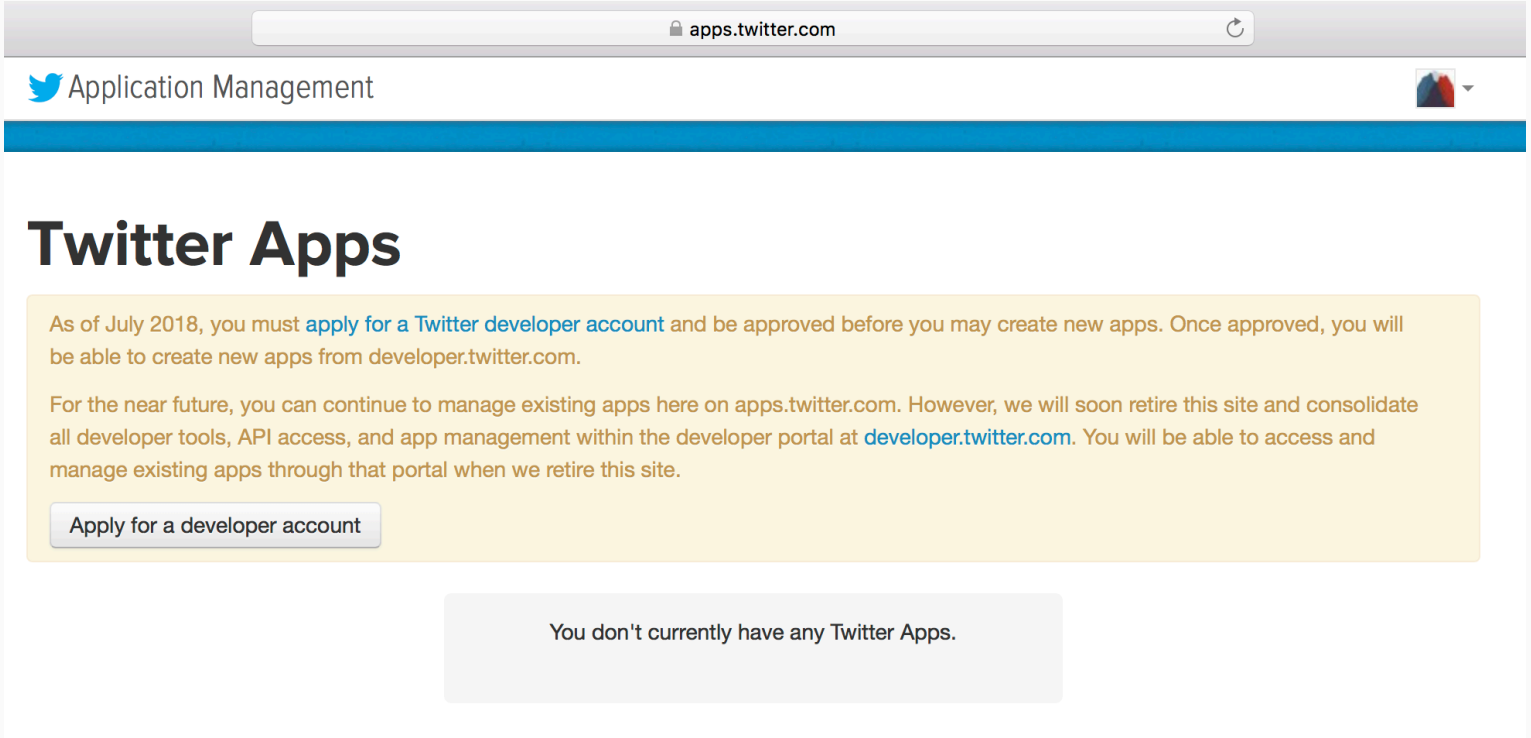
- data from most APi's can be retrieved with http requests and by applying loops (pagination)
- some packages like rtweet make it easier to work with APIs
- not all R packages (including those for APIs) are regularly maintained. Before using API-specific R packages, check whether developers are still active and the documentation is sufficient.
- `rtweet`is an example for a well-maintained package with active developers: `https://github.com/mkearney/rtweet`

Twitter's API

Using the Twitter API requires a developer account which can be used to create a Twitter application. This application in turn allows to create credentials used to acces the API.

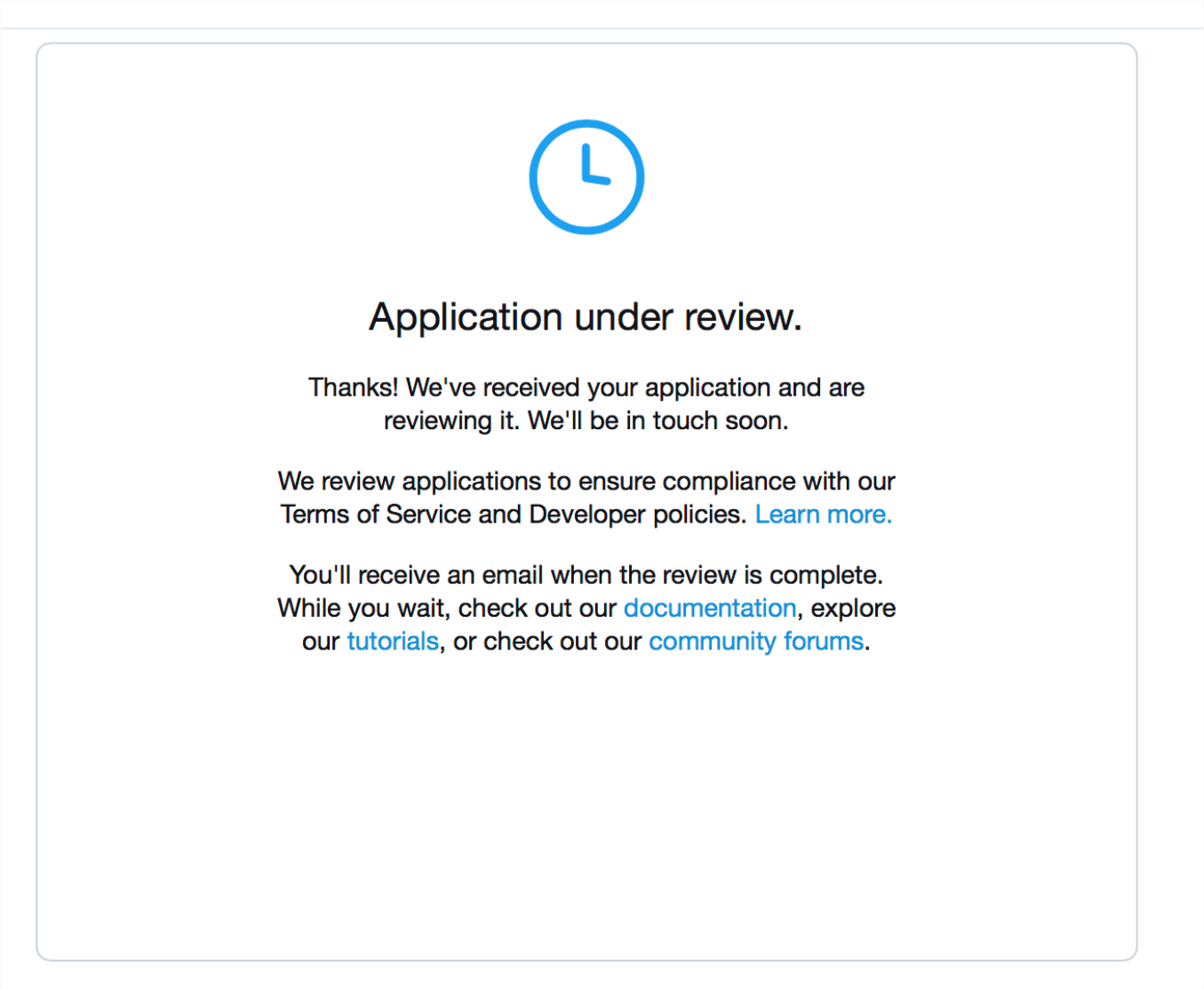
documentation: `https://developer.twitter.com/en/docs.html`

Twitter - apply for developer account



Guide: `https://www.carstenschwemmer.com/files/twitter_developer_application.pdf`

The waiting is the hardest part...



Your Twitter app

- go to apps.twitter.com to create app
- use `https://127.0.0.1:1410` as callback url for your Twitter app

Your Twitter app

Application Management

CelebrityFollowers

Test OAuth

Details | Settings | Keys and Access Tokens | Permissions

Test for class

https://docs.google.com/spreadsheets/d/1mEQVZr2re3FTzdpuFFTO8zU_xvT8QRo75tekY3pmzks/edit?usp=sharing

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.

Access level	Read-only (modify app permissions)
Consumer Key (API Key)	[redacted] (manage keys and access tokens)
Callback URL	http://127.0.0.1:1410
Sign in with Twitter	No
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

Application Actions

Delete Application

Twitter credentials

Details | Settings | Keys and Access Tokens | Permissions

Application Settings

Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.

Consumer Key (API Key)	YOUR CODE WILL APPEAR HERE
Consumer Secret (API Secret)	YOUR CODE WILL APPEAR HERE
Access Level	Read-only (modify app permissions)
Owner	chris_bail
Owner ID	

Application Actions

Regenerate Consumer Key and Secret | Change App Permissions

Your Access Token

This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.

Access Token	YOUR CODE WILL APPEAR HERE
Access Token Secret	YOUR CODE WILL APPEAR HERE
Access Level	Read-only
Owner	chris_bail
Owner ID	

Twitter credentials - simple

```
app_name <- "YOURAPPNAMEHERE"
consumer_key <- "YOURKEYHERE"
consumer_secret <- "YOURSECRETHERE"
access_token <- "YOURACCESSTOKENHERE"
access_token_secret <- "YOURACCESSTOKENSECRETHERE"
```

Twitter credentials - better

reminder: `usethis::edit_r_environ()` to set R environment variables

```
app_name <- Sys.getenv('twitter_app_name')
consumer_key <-Sys.getenv('twitter_consumer_key')
consumer_secret <- Sys.getenv('twitter_consumer_secret')
access_token <- Sys.getenv('twitter_access_token')
access_token_secret <- Sys.getenv('twitter_access_secret')
```

Authenticate yourself with Twitter API

```r
library(rtweet)
library(tidyverse) # for data analysis
create_token(app = app_name,
             consumer_key = consumer_key,
             consumer_secret = consumer_secret,
             access_token = access_token,
             access_secret = access_token_secret,
             set_renv = TRUE)
```

Your first Twitter API call

```r
hk_tweets <- search_tweets("#hongkong",
              n = 2000, include_rts = FALSE)
names(hk_tweets)[1:20]
```

```
##  [1] "user_id"              "status_id"            "created_at"
##  [4] "screen_name"          "text"                 "source"
##  [7] "display_text_width"   "reply_to_status_id"   "reply_to_user_id"
## [10] "reply_to_screen_name" "is_quote"             "is_retweet"
## [13] "favorite_count"       "retweet_count"        "quote_count"
## [16] "reply_count"          "hashtags"             "symbols"
## [19] "urls_url"             "urls_t.co"
```

Tweets are parsed into a DataFrame

```r
cat(hk_tweets$text[1])
```

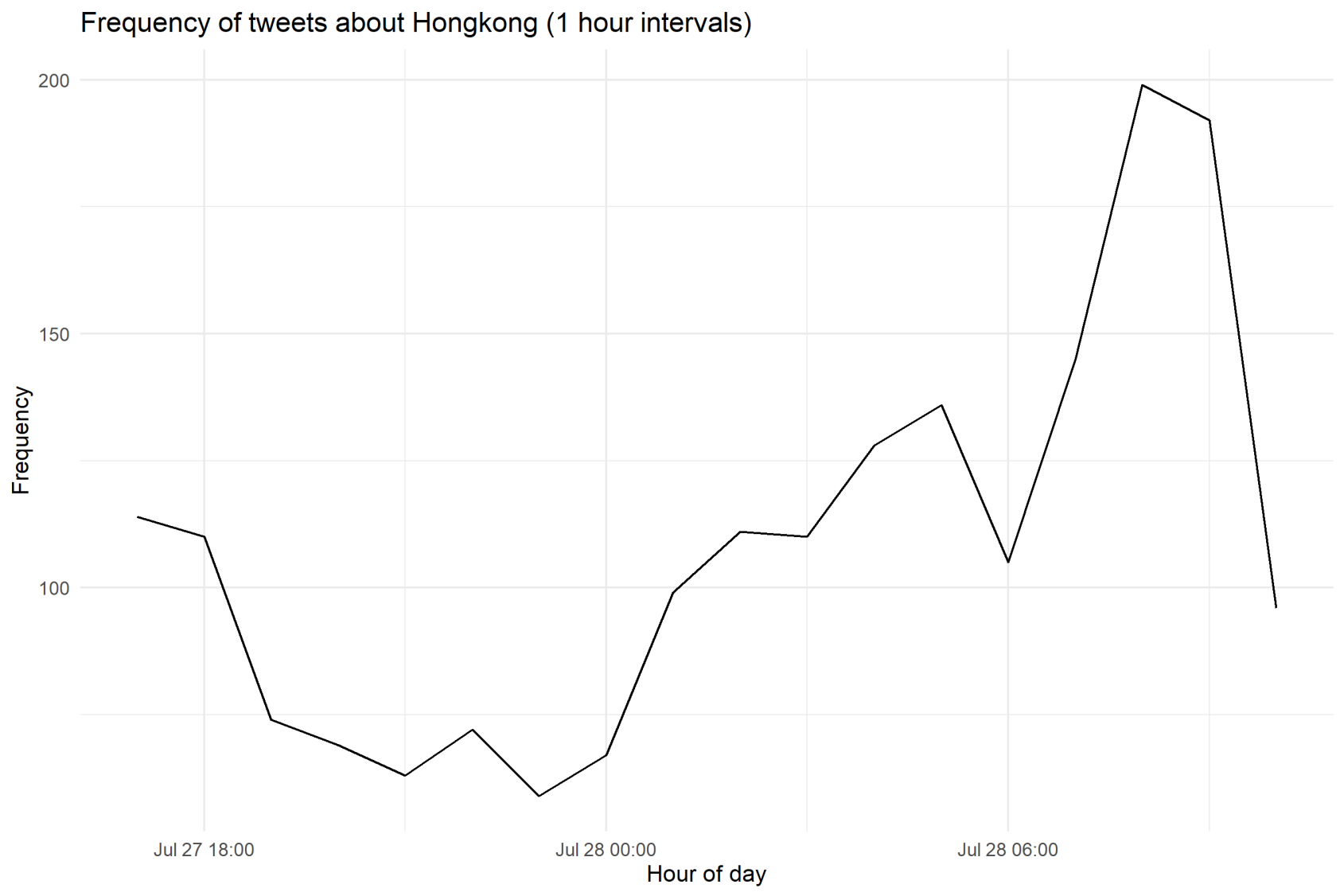
```
## #Internacional
##
## #HongKong , siete semanas de protestas
##
## https://t.co/zbWpAwpiuA https://t.co/vLshHnlLFz
```

Visualizing frequencies

`rtweet` includes a function for visualizing frequencies of tweets over a specified time:

```
ts_plot(hk_tweets, "1 hour") + theme_minimal() +
  labs(x = 'Hour of day', y = 'Frequency',
  title = "Frequency of tweets about Hongkong (1 hour intervals)")
```

Visualizing frequencies



Get Tweets from individual account

```r
sanders_tweets <- get_timeline("sensanders", n = 3)
cat(sanders_tweets$text[1])
```

```
## Public support for Medicare for All is rising. Why? The reason is pretty
obvious. It is getting harder and harder to defend a dysfunctional health care
system in which 30,000 people die every year because they don't get to a doctor
when they should. https://t.co/GTPVwfMsRW
```

Get general information about a user

```r
sanders_twitter_profile <- lookup_users("sensanders")

cat(sanders_twitter_profile$description)
```

```
## U.S. Senator Bernie Sanders of Vermont is the longest-serving independent in
congressional history.
```

```r
sanders_twitter_profile$followers_count
```

```
## [1] 8469144
```

Get users' favorites

```r
sanders_favorites <- get_favorites("sensanders", n = 3)
cat(sanders_favorites$text[1])
```

```
## "There is no reason for a drug as simple as insulin, which costs $21 in
Canada, to cost the equivalent of a mortgage payment." - @AOC #Insulin4all
#medicareforall https://t.co/o2gzATdekg
```

Get trending topics by location

```r
trends <- get_trends("United Kingdom")
trends$trend[1:3]
```

```
## [1] "#SundayMorning" "#Ridge"         "Harvey Elliott"
```

Check your Twitter rate limits

```r
rate_limits <- rate_limit()
head(rate_limits[,1:4])
```

```
## # A tibble: 6 x 4
##   query                    limit remaining reset
##   <chr>                    <int>     <int> <drtn>
## 1 lists/list                  15        15 14.99735 mins
## 2 lists/memberships           75        75 14.99735 mins
## 3 lists/subscribers/show      15        15 14.99735 mins
## 4 lists/members              900       900 14.99735 mins
## 5 lists/subscriptions         15        15 14.99735 mins
## 6 lists/show                  75        75 14.99735 mins
```

Post content

You can also posts tweets, like tweets, or follow users with `rtweet`:

```
post_tweet("I love APIs")
```

Note: this is very useful if you are building a bot

Wrapping API calls within a loop

```r
#load list of twitter handles for elected officials
elected_officials <- read_csv("https://cbail.github.io/Elected_Officials_Twitter_Handl
head(elected_officials)
```

```
## # A tibble: 6 x 2
##   name                screen_name
##   <chr>               <chr>
## 1 Sen Luther Strange  SenatorStrange
## 2 Rep. Mike Johnson   RepMikeJohnson
## 3 Ted Budd            RepTedBudd
## 4 Adriano Espaillat   RepEspaillat
## 5 Rep. Blunt Rochester RepBRochester
## 6 Nanette D. Barragán RepBarragan
```

Wrapping API calls within a loop

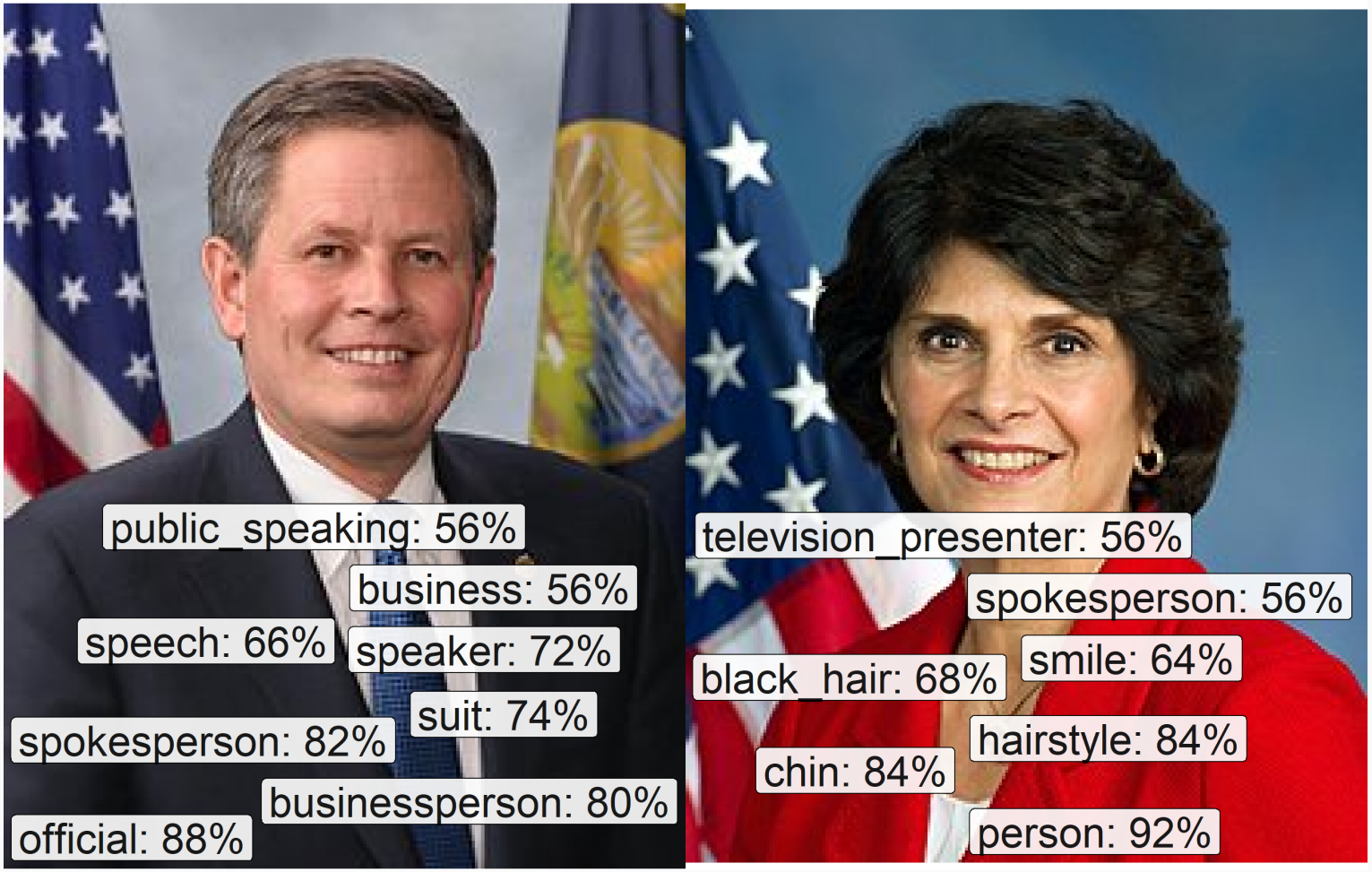
```
# create empty container to store tweets
elected_df <- data_frame()

for(i in 1:nrow(elected_officials)){
  # pull tweets
  tweets <- get_timeline(elected_officials$screen_name[i],
                         n = 100)
  # populate dataframe
  elected_df <- bind_rows(elected_df, tweets)
  # pause for one second to further prevent rate limiting
  Sys.sleep(1)
  # print number/iteration for monitoring progress
  print(i)
}
```

There are also APIs that do analysis for you!

For example, image recognition services such as Google Vision:

Google Vision labels for images of U.S. Members of Congress



R Packages for APIs

Here are a few:

- `RgoogleMaps`
- `rOpenSci`(combines many different APIs e.g. the Internet Archive)
- `WDI`
- `rOpenGov`
- `rtimes`
- `imgrec`
- `plumber` (build your own API)

Many more are available but not yet on CRAN (install from github or using devtools)

Lists of APIs of interest

https://github.com/public-apis/public-apis

https://www.programmableweb.com/

https://apilist.fun/

https://ropensci.org/packages/

Questions?