Querying APIs in R

Trafford Data Lab Follow
Jul 26, 2018 · 4 min read

Governments and other organisations often make open data available through Web service Application Programming Interfaces or APIs. The World Bank, UK Police, and Transport for London are just a few well-known examples. This article details the steps required to request data from these different Web service APIs using R.

Several R packages[1] have been developed as clients for Web service APIs. These don't assume any knowledge of API endpoints, HTTP requests, or data formats like XML and JSON. These are really convenient but sometimes you want to break into the 'black box' of APIs …

A quick introduction to APIs

APIs or Application Programming Interfaces are a set of rules that allow one software application to interact with another either in the same location or over a network. Inputs and outputs will vary between APIs but the process is the same: a 'request' that follows certain programmatic rules is submitted and a 'response' containing content in an expected format is returned.

There are many types of API including library-based (e.g. leafletJS) and class-based (e.g. Java) but one of the most common are Web service APIs. A client (browser) submits a Hypertext Transfer Protocol (HTTP) request to a server and the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

The parameters of an HTTP request are typically contained in the URL. For example, to return a map of Manchester using the Google Maps Static API we would submit the following request:

https://maps.googleapis.com/maps/api/staticmap?center=Manchester,England&zoom=13&size=600x300&maptype=roadmap

The request contains:

1. a URL to the API endpoint (https://maps.googleapis.com/maps/api/staticmap?) and;
2. a query containing the parameters of the request (center=Manchester,England&zoom=13&size=600x300&maptype=roadmap). In this case, we have specified the location, zoom level, size and type of map.

Web service APIs use two key HTTP verbs to enable data requests: GET and POST. A GET request is submitted within the URL with each parameter separated by an ampersand (&). A POST request is submitted in the message body which is separate from the URL. The advantage of using POST over GET requests is that there are no character limits and the request is more secure because it is not stored in the browser's cache.

There are several types of Web service APIs (e.g. XML-RPC, JSON-RPC and SOAP) but the most popular is Representational State Transfer or REST. RESTful APIs can return output as XML, JSON, CSV and several other data formats.

Each API has documentation and specifications which determine how data can be transferred. Unfortunately, the specifications tend to be different and the documentation can be hard to follow.

An example API request

Querying a Web service API typically involves the following steps:

1. submit the request
2. check for any server error
3. parse the response
4. convert to a data frame

In the following example we will submit a request for police reported crime data from the UK Police API. The API uses both HTTP GET and POST requests and provides content in JSON data format.

The two key R packages for submitting HTTP requests to Web service APIs and parsing the content of the response are httr and jsonlite. Let's load them into our R session. The tidyverse package is also loaded because it contains a suite of useful functions.

```
library(tidyverse) ; library(httr) ; library(jsonlite)
```

We would like to retrieve street level crimes within a mile radius of a specific location so we need to use https://data.police.uk/api/crimes-street/all-crime? as our API endpoint (see API documentation). Rather than retrieving 'all-crime' lets narrow our request to retrieve only reports of burglary. This will change our path to: https://data.police.uk/api/crimes-street/burglary?

```
path <- "https://data.police.uk/api/crimes-street/burglary?"
```

Next we need to build our API request and submit it. We will use the `GET` function from the httr package. First we supply the path to the API endpoint and provide search parameters in the form of a list to the `query` argument. There are three parameters available to us:

- `lat` = latitude
- `lng` = longitude
- `date` = and optional date in YYYY-MM format

```
request <- GET(url = path,
               query = list(
                 lat = 53.421813,
                 lng = -2.330251,
                 date = "2018-05")
               )
```

Let's check if the API returned an error. If the request fails the API will return a non-200 status code.

```
request$status_code
```

Next we parse the content returned from the server as text using the `content` function.

```
response <- content(request, as = "text", encoding = "UTF-8")
```

Then we'll parse the JSON content and and convert it to a data frame.

```
df <- fromJSON(response, flatten = TRUE) %>%
  data.frame()
```

Finally, we might strip out some of the variables and rename the remaining.

```
df <- select(df,
             month, category,
             location = location.street.name,
             long = location.longitude,
             lat = location.latitude)
```

Burglaries within 1m radius of specified location

month	category	location	long	lat
2018-05	burglary	On or near Priory Road	-2.308988	53.427603
2018-05	burglary	On or near Beaufort Avenue	-2.314616	53.415363
2018-05	burglary	On or near Trinity Avenue	-2.307520	53.420236

```
2018-05   burglary   On or near Hyde Grove        -2.323764   53.421289
```

That's it. We've submitted a request to the Police UK API and parsed the response into a data frame ready for use in R.

. . .

If you've found this useful you can find the R code needed to query other APIs like Nomis and the Food Standards Agency in our expanding Open Data Companion.

. . .

[1] Examples include eurostat, fingertipsR, and WHO.

API Rstats Open Data Crime