Retrieving data through APIs

Connor Gilroy

2017-10-24

Defining and demystifying APIs

APIs—application programming interfaces—are a structured way for pieces of software to communicate with each other.

Web APIs allow one computer (a client) to ask another computer (a server) for some resource over the internet.

APIs vs web scraping

You might be more familiar with the idea of **web scraping**, extracting information from the html of a web page.

Web scraping can be good for simple, static pages—if, for instance, you want a table from Wikipedia. It doesn't work at all for complicated, dynamic web sites like Facebook.

In addition, by following Terms of Service and going through an authentication process, you get a kind of *legitimacy* from API use that you don't necessarily get from web scraping.

Terminology and context

Modern web APIs use standard HTTP methods.

These are verbs: GET, POST, PUT, DELETE, and so on.

You make a **request** (to a url, or an *endpoint*) and get a **response**.

Responses have specific components: the status, the headers, and the body.

The body will contain the data you want in some format—most often, JSON.

Packages

Our main tool for interacting with APIs from R is the `httr` package:

```
library(httr)
```

We need a package to help us work with JSON-formatted files:

```
library(jsonlite)
```

And we'll also functions from the tidyverse:

```
library(tidyverse)
```

A simple illustration

```
r <- GET("https://http.cat/200")
r
```

```
## Response [https://http.cat/200]
##   Date: 2017-10-24 21:14
##   Status: 200
##   Content-Type: image/jpeg
##   Size: 27 kB
## <BINARY BODY>
```

Looking at the response

```
status_code(r)
```

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

```
r_body <- content(r, as = 'raw')
head(r_body)
```

```
## [1] ff d8 ff e0 00 10
```

```
write_file(r_body, "200.jpg")
```

Figure 1: source: http.cat

Example 1: The US Census API

Question: What are the largest Census-designated places in the US?

Example 1: The US Census API

First, read in your API key from a config file:

```
census_api_key <- read_lines("census_api_key.txt")
```

This is the base url for the 2015 5-year American Community Survey:

```
acs_url <- "https://api.census.gov/data/2015/acs/acs5"
```

Documentation

`https://www.census.gov/data/developers/data-sets/acs-5year.html`

examples:

`https://api.census.gov/data/2015/acs5/examples.html`

all variables:

`https://api.census.gov/data/2015/acs5/variables.html`

Using the documentation

What syntax do I use to make a request?

For the Census API, we request variables through the `get` field, separated by commas. (`"get=VAR1,VAR2"`)

We specify different geographies with the `for` or `in` field, using `*` to get all locations of a particular type. (`"for=region:01"`, `"for=tract:*&in=state:01"`)

What is the information I want called?

For total population, the estimate is a variable named `B01003_001E`, and the margin of error is `B01003_001M`. We also want the `NAME` of each place.

Constructing a query

We want to attach a query to the end of the url:

`https://url.com/data?q=query&key=YOURKEY`

`httr` will do this for us if we put the fields we want into a list:

```
acs_query <- list(
  get = "B01003_001E,B01003_001M,NAME",
  `for` = "place:*",
  # "for" is a reserved keyword in R!
  # You need backticks to use it as a name
  key = census_api_key
)
```

Request

```
acs_r <- GET(url = acs_url, query = acs_query)
```

Response

Look at the response. The body is a JSON array that looks like this:


```
[["B01003_001E","B01003_001M","NAME","state","place"],
["52","52","Abanda CDP, Alabama","01","00100"],
["2646","23","Abbeville city, Alabama","01","00124"],
["4454","21","Adamsville city, Alabama","01","00460"],
["682","154","Addison town, Alabama","01","00484"],
["293","84","Akron town, Alabama","01","00676"],
["31905","63","Alabaster city, Alabama","01","00820"],
...
```

Data processing

We want to turn this JSON object into a data frame, using `jsonlite::fromJSON`.

The first row contains the names of the columns; the remaining rows are values.

```
acs_m <- fromJSON(content(acs_r, as = "text"))
acs_df <- as_data_frame(acs_m[-1, ])
colnames(acs_df) <- acs_m[1, ]
```

```
head(acs_df)
```

B01003_001E	B01003_001M	NAME	state	place
52	52	Abanda CDP, Alabama	01	00100
2646	23	Abbeville city, Alabama	01	00124
4454	21	Adamsville city, Alabama	01	00460
682	154	Addison town, Alabama	01	00484
293	84	Akron town, Alabama	01	00676
31905	63	Alabaster city, Alabama	01	00820

What are the 10 largest cities in the US?

```
acs_df <-
  acs_df %>%
  mutate(B01003_001E = as.numeric(B01003_001E),
         B01003_001M = as.numeric(B01003_001M)) %>%
  arrange(desc(B01003_001E)) %>%
  select(-B01003_001M)
```

```
head(acs_df, 10)
```

B01003_001E	NAME	state	place
8426743	New York city, New York	36	51000
3900794	Los Angeles city, California	06	44000
2717534	Chicago city, Illinois	17	14000
2217706	Houston city, Texas	48	35000
1555072	Philadelphia city, Pennsylvania	42	60000
1514208	Phoenix city, Arizona	04	55000
1413881	San Antonio city, Texas	48	65000
1359791	San Diego city, California	06	66000
1260688	Dallas city, Texas	48	19000
1000860	San Jose city, California	06	68000

Example 2: The World Bank

The World Bank API offers access to more than 8000 indicators for the countries of the world, often over long periods of time.

`https://datahelpdesk.worldbank.org/knowledgebase/topics/125589`

We will use it to look at the total population over time for one country, South Africa.

Example 2: The World Bank

```
wb_url <- "http://api.worldbank.org"

wb_query <- list(
  format = "json"
)
```

In this case, we will build our queries using path rather than query syntax, as described here:

`https://datahelpdesk.worldbank.org/knowledgebase/articles/898581-api-basic-call-structure`

This query retrieves basic information about South Africa:

```
r_za <- GET(wb_url,
            path = "countries/za",
            query = wb_query)
# to view:
# prettify(content(r_za, as = "text"))
```

This query describes the SP.POP.TOTL indicator:

```
r_pop <- GET(wb_url,
             path = "indicators/SP.POP.TOTL",
             query = wb_query)
```

Total population for South Africa

To get the total population indicator for South Africa, we combine the two previous queries:

```
r_za_pop <-
  GET(wb_url,
      path = "countries/za/indicators/SP.POP.TOTL",
      query = wb_query)
```

Paginated responses

Often, APIs only return a fixed amount of information, e.g. 50 items. If you want more, you have to ask.

The most recent years are on “page” 2 of 2, which we must query specifically.

```
wb_query_pg2 <- list(
  page = 2,
  format = "json"
)

r_za_pop_pg2 <-
  GET(wb_url,
      path = "countries/za/indicators/SP.POP.TOTL",
      query = wb_query_pg2)
```

Data processing

R can parse this data into a list, but then we need to turn it into a data frame.

Here is a function to help us do that:

```
process_wb_data <- function(d) {
  as_data_frame(d) %>%
    unnest() %>%
    select(-decimal) %>%
    group_by(value, date) %>%
    summarise_all(first) %>%
    # try out ‘last‘ instead of ‘first‘ here!
    ungroup()
}
```

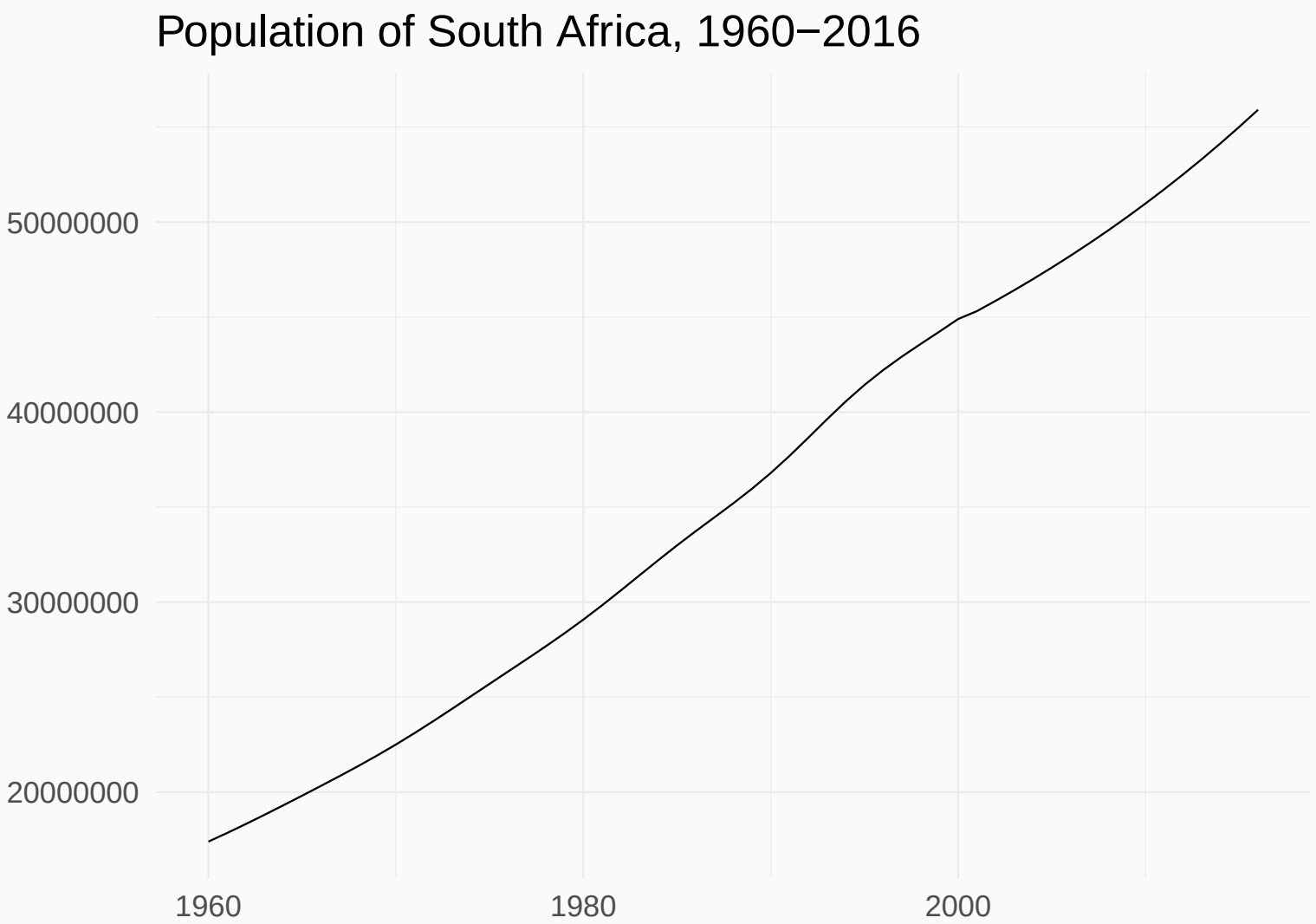
Data processing

```
df_pg1 <-
  map(content(r_za_pop, as = "parsed")[[2]],
      process_wb_data) %>%
  bind_rows()

df_pg2 <-
  map(content(r_za_pop_pg2, as = "parsed")[[2]],
      process_wb_data) %>%
  bind_rows()

r_za_pop_df <-
  bind_rows(df_pg2, df_pg1) %>%
  mutate(value = as.numeric(value),
         date = as.numeric(date))
```

Plot population over time

```
options(scipen = 99)
za_plot <-
  ggplot(r_za_pop_df, aes(x = date, y = value)) +
  geom_line() +
  theme_minimal(base_size = 20) +
  labs(title = "Population of South Africa, 1960-2016",
       x = NULL, y = NULL)
```

```
za_plot
```


Population of South Africa, 1960−2016
50000000
40000000
30000000
20000000
1960
1980
2000

Something old, something new

These examples have focused on accessing more traditional sources of data in a new way.

But the obvious promise of APIs is the ability to access *new* data sources—Google, Yelp, Twitter, Facebook, and so on. Sometimes the authentication might be more complicated, or the documentation more technical, but the principle is the same.

An alternative: specialized packages

Many of the most common APIs you might want to use have dedicated R packages that wrap them and make them easier to interact with.

For example, there's `tidycensus` for the Census, and `rtimes` for the New York Times.

Often, these packages are based on `httr`!

But...

- ...what if the API you want to use doesn't have a package?
- ...what if you want to access API functionality the package doesn't support?
- ...what if the API updates, and the package breaks?

Takeaways

General mental model for how APIs work, and a versatile tool for interacting with them, `httr`.

Sense of how to find the information you want from the original documentation.

Some comfort with manipulating data from raw JSON into a form you can use.

Exercises

Exercise 1: Use the 2016 1-year ACS to get the population of every state. What are the 5 smallest states?

url: `https://api.census.gov/data/2016/acs/acs1`

Be sure to look at the 1-year ACS documentation on `https://www.census.gov/developers/` rather than the 5-year documentation.

Exercise 2: Request population indicators for three other countries from the World Bank API and plot their population change over time.

Use two-digit country codes for countries: `https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2`

Exercises

Challenge exercise: Check out this list of APIs for social science maintained by Chris Bail:

`https://docs.google.com/spreadsheets/d/1ZEr3okdlb0zctmX0MZKo-gZKPsq5WGn1nJOxPV7al-Q/edit?usp=sharing`

Sign up for one that looks interesting and request a piece of data from it. Try to figure out how to get that data into an R-friendly format like a data frame, and display some of it. Why might the data be of interest to demographers?

Resources

The `httr` quickstart vignette: `https://cran.r-project.org/web/packages/httr/vignettes/quickstart.html`

Another tutorial for the New York Times API: `https://github.com/ccgilroy/nyt-api-httr-demo`