Querying APIs

Justin Post

Reading Data

Data comes in many formats such as

- 'Delimited' data: Character (such as ',' , '>', or [' ']) separated data
- Fixed field data
- Excel data
- From other statistical software, Ex: SPSS formatted data or SAS data sets
- From a database
- From an Application Programming Interface (API)

APIs

Application Programming Interfaces (APIs) - a defined method for asking for information from a computer

- Basically a protocol for computers to talk to one another
- Useful for getting data
- Useful for allowing others to access something you make (say a model)

APIs

- Most major sites with data now have an API. A key is usually required
  - Documentation can be spotty
  - Some have written functions for us :)
- Consider theCensus API
  - A `tidycensus` package exists!

```
library(tidycensus) #install first!
```

Census APIs

- Consider the American Community Survey
  - Accessed via `get_acs()` function
  - Variable list available

Census APIs

- Consider the American Community Survey
  - Accessed via `get_acs()` function
  - Variable list available

```
rent <- "DP04_0142PE" #PE means percentage
rent_data <- get_acs(variables = rent,
        geography = "county",
        geometry = TRUE,#returns the polygon data and allows for maps easily
        survey = "acs5",
        show_call = TRUE,
        key = "e267f117801b2ef741e54620602b0903c5f4d3c8"
        ) #can add state and other things
```

```
##
Downloading: 1 B
Downloading: 1 B
Downloading: 1.4 kB
Downloading: 1.4 kB
Downloading: 5.5 kB
Downloading: 5.5 kB
Downloading: 6.8 kB
```



```
Downloading: 40 kB
```

Plotting Census Data

- A great package can be combined for easy plots!

```
#install mapview
rent_data |>
  mapview::mapview(zcol = "estimate",
                   layer.name = "Median rent as a % of gross income")
```

Census API

- Ok, what is going on with the `get_acs()` function?
    - It calls `load_data_acs()` which builds the URL for us!

```
load_data_acs <- function(geography, formatted_variables, key, year, state = NULL,
                          county = NULL, zcta = NULL, survey, show_call = FALSE) {
  base <- paste("https://api.census.gov/data",
                as.character(year), "acs",
                survey, sep = "/")

  if (grepl("^DP", formatted_variables)) {
    message("Using the ACS Data Profile")
    base <- paste0(base, "/profile")
  }
...
```

API Access in R

- Awesome! When someone has done the work it is great :)
- Some resources on API packages:
  - Someone's Github List
  - Another one!
- List of APIs

API Example: Building it Ourselves

- Let's investigate the National Hockey League's (NHL) API
- Google shows a number of packages... but they get out of date or aren't maintained. Let's do it ourselves!
- Unfortunately, the NHL API is very poorly documented...
    - Thanks Zmalski, this helps!

API Example: Building it Ourselves

Process:

- Build the appropriate URL
- Use `httr:GET()` to contact the web site
- Data is usually JSON (or possibly XML). Parse it!
- Try to put into a data frame

Aside: JSON Data

- Most APIs return data in JSON format
    - **JSON** - JavaScript Object Notation
    - Can represent usual 2D data or heirarchical data

Aside: JSON Data

- Uses key-value pairs

```
{
  {
    "name": "Barry Sanders"
    "games" : 153
    "position": "RB"
  },
  {
    "name": "Joe Montana"
    "games": 192
    "position": "QB"
  }
}
```

JSON Packages in R

Four major R packages

1. `rjson`
2. `RJSONIO`

3. `jsonlite`

    - many nice features
    - a little slower implementation

4. `tidyjson`

`jsonlite` Package

`jsonlite` basic functions:

Function	Description
`fromJSON`	Reads JSON data from file path or character string. Converts and simplfies to R object
`toJSON`	Writes R object to JSON object
`stream_in`	Accepts a *file connection* - can read streaming JSON data

Build the URL

- First we want to build the URL to contact a particular end point of the API
- Suppose we first want team information. Documentation says

Teams

Get Team Information

- **Endpoint:** `/{lang}/team`
- **Method:** GET
- **Description:** Retrieve list of all teams.
- **Parameters:**
  - `lang` (string) - Language code
- **Response:** JSON format

Example using cURL:

```
curl -X GET "https://api.nhle.com/stats/rest/en/team"
```

Build the URL

We create a string for the URL:

```
URL_ids <- "https://api.nhle.com/stats/rest/en/team"
```

- Now use `GET` from `httr` package

```
id_info <- httr::GET(URL_ids)
str(id_info, max.level = 1)
```

```
## List of 10
##  $ url        : chr "https://api.nhle.com/stats/rest/en/team"
##  $ status_code: int 200
##  $ headers    :List of 15
##   ..- attr(*, "class")= chr [1:2] "insensitive" "list"
##  $ all_headers:List of 1
##  $ cookies    :'data.frame':	0 obs. of  7 variables:
##  $ content    : raw [1:6664] 7b 22 64 61 ...
##  $ date       : POSIXct[1:1], format: "2025-05-07 18:55:30"
##  $ times      : Named num [1:6] 0 0.161 0.177 0.456 0.515 ...
##   ..- attr(*, "names")= chr [1:6] "redirect" "namelookup" "connect" "pretransfer" ...
##  $ request    :List of 7
                                    equest"
                                    dle' <externalptr>
                                    onse"
```

NC STATE UNIVERSITY

Build the URL

- Must parse this a bit... Usually data is in `content` or `results` element
  - Often use `rawToChar()` with `jsonlite::fromJSON()`

```
library(jsonlite)
parsed <- fromJSON(rawToChar(id_info$content))
team_info <- as_tibble(parsed$data)
team_info
```

```
## # A tibble: 62 x 6
##      id franchiseId fullName             leagueId rawTricode triCode
##   <int>       <int> <chr>                   <int> <chr>      <chr>
## 1    32          27 Quebec Nordiques          133 QUE        QUE
## 2     8           1 MontrÃ©al Canadiens       133 MTL        MTL
## 3    58           5 Toronto St. Patricks      133 TSP        TSP
## 4     7          19 Buffalo Sabres            133 BUF        BUF
## 5    46          13 Oakland Seals             133 OAK        OAK
## # i 57 more rows
```

Build the URL

- Now we can get some team stats through the same process!

Get Team Stats

- **Endpoint**: `/{lang}/team/{report}`
- **Method**: GET
- **Description**: Retrieve team stats for a specific report.
- **Parameters**:
  - `report` (string) - Team report
  - `lang` (string) - Language code
- **Request Parameters**:
  - `isAggregate` (query, boolean) - Optional
  - `isGame` (query, boolean) - Optional
  - `factCayenneExp` (query, string) - Optional
  - `include` (query, string) - Optional
  - `exclude` (query, string) - Optional
  - `cayenneExp` (query, string) - Optional
  - `sort` (query, string) - Optional
  - `dir` (query, string) - Optional
  - `start` (query, int) - Optional
  - `limit` (query, int) - Optional (**Note:** a limit of -1 will return all results)
- **Response**: JSON format

Example using cURL:

```
curl -X GET "https://api.nhle.com/stats/rest/en/team/summary?sort=shotsForPerGame&cayenneExp=seasonId=2
```

Build the URL

- A few things can be modified but it isn't clear here what the values could be.

```
URL_team_stats <-
"https://api.nhle.com/stats/rest/en/team/summary?sort=wins&cayenneExp=seasonId=20232024%20and%20gameTypeId=2"
```

- `GET()` it and parse it with the same process

```
team_stats_return <- httr::GET(URL_team_stats)
parsed_team_stats <- fromJSON(rawToChar(team_stats_return$content))
team_stats <- as_tibble(parsed_team_stats$data)
```

Check it Out

```
 team_stats |>
   select(teamId, teamFullName, everything())
```

```
## # A tibble: 32 x 24
##    teamId teamFullName faceoffWinPct gamesPlayed goalsAgainst goalsAgainstPerGame
##     <int> <chr>                <dbl>       <int>        <int>                <dbl>
## 1      28 San Jose Sh~         0.490          82          326                 3.98
## 2      16 Chicago Bla~         0.463          82          289                 3.52
## 3      24 Anaheim Duc~         0.466          82          293                 3.57
## 4      29 Columbus Bl~         0.472          82          298                 3.63
## 5       8 MontrÃ©al C~         0.515          82          281                 3.43
## # i 27 more rows
## # i 18 more variables: goalsFor <int>, goalsForPerGame <dbl>, losses <int>,
## #   otLosses <int>, penaltyKillNetPct <dbl>, penaltyKillPct <dbl>,
## #   pointPct <dbl>, points <int>, powerPlayNetPct <dbl>, powerPlayPct <dbl>,
## #   regulationAndOtWins <int>, seasonId <int>, shotsAgainstPerGame <dbl>,
## #   shotsForPerGame <dbl>, ties <lgl>, wins <int>, winsInRegulation <int>,
## #   winsInShootout <int>
```

Implementing a Model in Production

Later: Need a way to make your model available to others

- Can write an API that accesses your model
- Hosted on a server or locally
- Not traditionally done in R but can be!

Recap

- APIs are a common tool used for communicating about data
    - Can be used for other things as well
- Accessing data through an API involves building appropriate communication message (URL usually)
- Some API packages already exist
- Others, we need to parse the data ourselves!