QSS20: Modern Statistical Computing

Unit 06: APIs

Outline

- **API: terminology and basics**
- Example 1: API with no credentials and no wrapper (NAEP data API)
- Example 2: API with credentials and no wrapper (Yelp API)
- Example 3: API with credentials and wrapper (wrapper for Twitter API)

Terminology

▶ **API:** application programming interface; way to ask an app or website for something and get something in return

▶ **Call the API**: sending a request for something to the API

▶ **Response**: can think of this as a message back telling us *whether* we got something back or whether the call returned an error

▶ **JSON:** if we get something back, oftentimes it'll be stored in json format, which is basically a text string with a particular structure that is similar to the *data structure* of a dictionary; can pretty easily convert to a pandas dataframe

▶ **Wrapper:** a language-specific module or package that helps simplify the process of calling an API with code written in a particular language (e.g., later we'll review a Python wrapper for the Twitter API; there are also R wrappers for the Twitter API)

Main use in our context: data acquisition

Three general routes to acquiring data:

Exists already:
csv or excel data we've been working with for problem sets

API:
Use to create data or flat files for use; most relevant for "high-velocity" data that changes frequently (e.g., tweets; job postings) and also for using code rather than point/click to get data

Scraping:
APIs are a sort of "front door" to a website, where the developers provide an easy way to get content but also set limits (e.g., what content you can get; how much content you can pull in a given period); scraping is a back door for when there's no API or when we need content beyond the structured fields the API returns

Why go through the effort to use data that doesn't exist already?

I got the data for my thesis from web scraping, which I mostly learned from Google. I highly recommend learning how to scrape website information. It is often a guarantee that you are working with original data, which means you are uncovering an original thesis topic (Source: `https://qss.dartmouth.edu/sites/program_quantitative_social_science/files/zach-schnell-thesis-advice.pdf`)

Outline

- ▶ API: terminology and basics
- ▶ **Example 1: API with no credentials and no wrapper (NAEP data API)**
- ▶ Example 2: API with credentials and no wrapper (Yelp API)
- ▶ Example 3: API with credentials and wrapper (wrapper for Twitter API)

High-level overview of steps: APIs that don't need credentials

1. Construct a query that tells the API what we want to pull
2. Use `requests.get(querystring)` to call the API
3. Examine the response: message from the API telling us whether it returned something
4. If the response returned something, extract the content of the response and make it usable

Step 1: construct a query

▶ Generic example:
  "https://baseurl.com/onething=something&anotherthing=somethingelse"

▶ Specific NAEP example (use the (syntax to split across lines)

```
1  example_naep_query = (
2  'https://www.nationsreportcard.gov/'
3  'Dataservice/GetAdhocData.aspx?'
4  'type=data&subject=writing&grade=8&'
5  'subscale=WRIRP&variable=GENDER&',
6  'jurisdiction=NP&stattype=MN:MN&',
7  'Year=2011')
```

▶ Breaking things down:
  ▶ `nationsreportcard`: this is the "base url" we're using for the API call and what we add parameters to
  ▶ `subject`: type of parameter
  ▶ `subject=writing`: specific value for that parameter (error if we feed it a subject that doesn't exist)
  ▶ And so on...

Steps 2-4: call the API, examine the response, and if response indicates something usable, extract content

```python
1  ## use requests.get to call the API
2  naep_resp = requests.get(example_naep_query)
3
4  ## we got usable response, so get json of status and
       result
5  naep_resp_j = naep_resp.json()
6
7  ## extract contents in `result' key
8  ## and convert to dataframe
9  naep_resp_d = pd.DataFrame(naep_resp_j['result'])
```

What do I mean by "no wrapper"?

- ▶ We write a query to request something from the API
- ▶ While the request syntax differs across languages, the query is the same– eg could use same query and run below in R to get content

```
1 ## packages
2 library(httr)
3 library(jsonlite)
4
5 ## ping API
6 return_q = GET(example_naep_query)
7
8 ## get data from that ping
9 data = fromJSON(rawToChar(return_q$content))$result
```

Break for code walk-through and group example

- ▶ Example of executing a query that doesn't have errors
- ▶ Example of executing a query that returns nothing
- ▶ Working together to write a function to do multiple calls to the API

Outline

- ▶ API: terminology and basics
- ▶ Example 1: API with no credentials and no wrapper (NAEP data API)
- ▶ **Example 2: API with credentials and no wrapper (Yelp API)**
- ▶ Example 3: API with credentials and wrapper (wrapper for Twitter API)

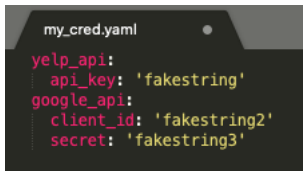## What changes about the general steps?

1. Acquire credentials for the API: these may be an API key (single string) or a client ID and secret (two strings; can store in a `yaml` creds file that I'll outline)
2. Construct a query that tells the API what we want to pull
3. Two paths:
   3.1 **Use credentials to authenticate and then call the API:** we'll later see example of this with Twitter API/wrapper
   3.2 **Feed API your credentials when you call the API:** we'll see example of this with Yelp
4. Examine the response: message from the API telling us whether it returned something
5. If the response returned something, extract the content of the response and make it usable

Step 1: acquire credentials

▶ Varies across APIs, but general involves going to the "developer's portal," creating an account, and obtaining credentials

▶ Examples:
  ▶ Google developer's console (things like google geocoding API; maps API): https://console.cloud.google.com
  ▶ Facebook: https://developers.facebook.com/docs/development
  ▶ Twitter (via Tweepy wrapper): https://docs.tweepy.org/en/latest/auth_tutorial.html
  ▶ Yelp: https://www.yelp.com/developers/documentation/v3/authentication

▶ Note weird-ish terminology for social science applications since you often set up "an application" in order to get credentials (but we're often doing a one-way pull of data and not developing an app. that repeatedly calls it)

Step 1: store those credentials somewhere

▶ Your key or client ID/secret are meant to be unique to you like a password, so you shouldn't generally print in code

▶ Can use any text editor to make a yaml file (structured similar to a dictionary); screenshot below from Sublime text with fake credentials

Step 1: load the file with credentials

```
1 ## imports
2 import yaml
3
4 ## load creds
5 with open("../private_data/my_cred.yaml", 'r') as stream:
6     try:
7         creds = yaml.safe_load(stream)
8     except yaml.YAMLError as exc:
9         print(exc)
10
11 ## can then get the relevant key
12 creds['yelp_api']['api_key']
```

Step 2: construct a query

Same exact process as before; here focusing on **Yelp Fusion API**; API has different endpoints shown in the screenshot; we'll initially focus on Business Search, since that returns a Yelp-specific ID (https://www.yelp.com/developers/documentation/v3/get_started)

Name	Path	Description
Business Search	/businesses/search	Search for businesses by keyword, category, location, price level, etc.
Phone Search	/businesses/search/phone	Search for businesses by phone number.
Transaction Search	/transactions/{transaction_type}/search	Search for businesses which support food delivery transactions.
Business Details	/businesses/{id}	Get rich business data, such as name, address, phone number, photos, Yelp rating, price levels and hours of operation.
Business Match	/businesses/matches	Find the Yelp business that matches an exact input location. Use this to match business data from other sources with Yelp businesses.
Reviews	/businesses/{id}/reviews	Get up to three review excerpts for a business.
Autocomplete	/autocomplete	Provide autocomplete suggestions for businesses, search keywords and categories.

Step 2: construct a query

```python
## defining inputs
base_url = "https://api.yelp.com/v3/businesses/search?"
my_name = "restaurants"
my_location = "Hanover,NH,03755"

## combining them into a query
yelp_genquery = (
    '{base_url}'
    'term={name}'
    '&location={loc}').format(
            base_url = base_url,
            name = my_name,
            loc = my_location)
```

Step 3: authenticate and call the API

For Yelp, we feed a dictionary with our key directly into the get call via the optional header parameter; for other APIs, we sometimes authenticate in a separate step

```
1 ## construct my http header dict
2 header = {'Authorization': f'Bearer {API_KEY}'}
3
4 ## call the API
5 yelp_genresp = requests.get(yelp_genquery, headers =
    header)
```

Step 3: output of successful and unsuccessful call

▶ Successful call:

```
<Response [200]>
```

▶ Unsuccessful call (put Hanover,WY,09999 as the location, which doesn't exist)

```
<Response [400]>

{'error': {'code': 'LOCATION_NOT_FOUND',
   'description': 'Could not execute search, try specifying a more exact location.'}}
```

Step 4: if output successful, make results usable

See that 'businesses' key of json file has a dictionary for each business, but some nesting to deal with variable lengths (e.g., within 'location', 'address1', 'address2', etc.) that might produce odd things when we concat. to a df:

```
{'id': '8ybF6YyRldtZmU9jil4xlg',
 'alias': 'mollys-restaurant-and-bar-hanover',
 'name': "Molly's Restaurant & Bar",
 'image_url': 'https://s3-media2.fl.yelpcdn.com/bphoto/1YkJFic4Czt9b2FsZyOrwQ/o.jpg',
 'is_closed': False,
 'url': 'https://www.yelp.com/biz/mollys-restaurant-and-bar-hanover?adjust_creative=A
gn=yelp_api_v3&utm_medium=api_v3_business_search&utm_source=ABQTB3e9fTiSiyqs0c-3Bg',
 'review_count': 403,
 'categories': [{'alias': 'tradamerican', 'title': 'American (Traditional)'},
  {'alias': 'burgers', 'title': 'Burgers'},
  {'alias': 'pizza', 'title': 'Pizza'}],
 'rating': 4.0,
 'coordinates': {'latitude': 43.701144, 'longitude': -72.2894249},
 'transactions': ['delivery'],
 'price': '$$',
 'location': {'address1': '43 South Main St',
  'address2': '',
  'address3': '',
  'city': 'Hanover',
  'zip_code': '03755',
  'country': 'US',
  'state': 'NH',
  'display_address': ['43 South Main St', 'Hanover, NH 03755']},
 'phone': '+16036432570',
 'display_phone': '(603) 643-2570',
 'distance': 250.8301601841674}
```

Approach 1 to step 4: more automatic pd.concat that leaves those as lists

```python
1 yelp_gendf = pd.DataFrame(yelp_genjson['businesses'])
```

Approach 2 to step 4: only retaining columns that are already strings

```python
def clean_yelp_json(one_biz):

    ## restrict to str cols
    d_str = {key:value for key, value in one_biz.items()
    if type(value) == str}

    df_str = pd.DataFrame(d_str, index = [d_str['id']])
    return(df_str)

yelp_stronly = [clean_yelp_json(one_b)
        for one_b in yelp_genjson['businesses']]
yelp_stronly_df = pd.concat(yelp_stronly)
```

Activity

▶ Try running a business search query for your hometown or another place by constructing a query similar to 'yelp_genquery' but changing the location parameter

▶ Other endpoints require feeding what's called the fusion id into the API. Take an id from 'yelp_stronly.id' and use the documentation here to pull the reviews for that business: https://www.yelp.com/developers/documentation/v3/business_reviews

▶ **Challenge**: generalize the previous step by writing a function that (1) takes a list of ids as an input, (2) calls the reviews API for each id, (3) returns the results, and (4) rowbinds all results