STAT679
Computing for Data Science and Statistics

Lecture 14: APIs
Previously: Scraping Data from the Web

We used BeautifulSoup to process HTML that we read directly
   We had to figure out where to find the data in the HTML
   This was okay for simple things like Wikipedia…
   …but what about large, complicated data sets?
       E.g., Climate data from NOAA; Twitter/reddit/etc.; Google maps

Many websites support APIs, which make these tasks simpler

Instead of scraping for what we want, just ask!

Example: ask Google Maps for a computer repair shop near a given address
Recall the implementation-interface distinction. APIs are an example of this!

The API provides a set of tools or functions for interacting with a web service.

**Example:** Google Maps supplies tools for asking about addresses and directions:
- Get information about a specific address
- Get directions from one address to another
- Get traffic information

These are supplied as interfaces that we can use...

...but their inner workings are hidden from us as end users.
Three common API approaches

Via a Python package
Service (e.g., Google maps, ESRI*) provides library for querying DB
**Example:** from arcgis.gis import GIS

Via a command-line tool
**Example:** twurl https://developer.twitter.com/

Via HTTP requests
We submit an HTTP request to a server
Supply additional parameters in URL to specify our query
**Example:** https://www.yelp.com/developers/documentation/v3/business_search

* ESRI is a GIS service, to which the university has a subscription: https://developers.arcgis.com/python/

Ultimately, all three of these approaches end up submitting an HTTP request to a server, which typically returns information in the form of a JSON or XML file.
Reminder: Client-server model

Request can be as simple as “give me website X”...
...but we can also make more complicated requests.
Web service APIs

Step 1: Create URL with query parameters

Example (non-working): www.example.com/search?key1=val1&key2=val2

Step 2: Make an HTTP request

Communicates to the server what kind of action we wish to perform


Step 3: Server returns a response to your request

May be as simple as a code (e.g., 404 error)...

...but typically a JSON or XML file (e.g., in response to a DB query)
HTTP Requests

Allows a client to ask a server to perform an action on a resource
  E.g., perform a search, modify a file, submit a form

Two main parts of an HTTP request:
  **URI**: specifies a resource on the server
  **Method**: specifies the action to be performed on the resource

HTTP request also includes (optional) additional information
  E.g., specifying message encoding, length and language

More information:
RFC specifying HTTP requests: https://tools.ietf.org/html/rfc7231#section-4
HTTP Request Methods

GET: retrieves information from the server
POST: sends information to the server (e.g., a file for upload)
PUT: replace the URI with a client-supplied file
DELETE: delete the file indicated by the URI
CONNECT: establishes a tunnel (i.e., connection) with the server


See also Representational State Transfer:
https://en.wikipedia.org/wiki/Representational_state_transfer
Submitting HTTP Requests: Parameters

We submit an HTTP request to a URL, e.g., api.example.com but often we want to further specify our request with parameters

Example: when we ask Google Maps for directions, we need to specify:
- Start location, destination
- Mode of transportation (e.g., walking, bike, bus, plane, train, automobile)

We do this with URL parameters, passed as key-value pairs

Example: api.example.com/server?course=STAT679&location=UWMadison

Passes two parameters: course, with value STAT679 and location, with value UWMadison.

Roughly comparable to Python keyword arguments.
Refresher: JSON

JavaScript Object Notation
https://en.wikipedia.org/wiki/JSON

Commonly used by website APIs

Basic building blocks:
attribute–value pairs
array data

Example (right) from wikipedia:
Possible JSON representation of a person
Refresher: Python `json` module

```python
import json

json_string = '{"first_name":"John", "last_name":"Bardeen", "alma_mater":"University of Wisconsin"}'

parsed_json = json.loads(json_string)

print(parsed_json)

json.dumps(parsed_json)

'{"first_name": "John", "last_name": "Bardeen", "alma_mater": "University of Wisconsin"}'
```

- JSON string encoding information about physicist John Bardeen
- `json.loads` parses a string and returns a JSON object.
- `json.dumps` turns a JSON object back into a string.
Refresher: Python `json` module

```python
parsed_json

{'alma_mater': 'University of Wisconsin',
 'first_name': 'John',
 'last_name': 'Bardeen'}

parsed_json['alma_mater']

'University of Wisconsin'

parsed_json['first_name']

'John'

parsed_json['middle_name']

Traceback (most recent call last)
<ipython-input-9-e0447f76c1d5> in <module>()
----> 1 parsed_json['middle_name']

KeyError: 'middle_name'
```

JSON object returned by `json.loads` acts just like a Python dictionary.
Example: Querying Yelp’s Business Search Service

I am sitting at my desk, woefully under-caffeinated

I could open a new browser tab and search for coffee nearby…
  ...but why leave the comfort of my Jupyter notebook?

Yelp provides several services under their “Fusion API”
  https://www.yelp.com/developers/documentation/v3/get_started

We’ll use the business search endpoint
  Supports queries that return businesses reviewed on Yelp
Example: Querying Yelp’s Business Search Service

```python
import requests
url = 'https://api.yelp.com/v3/businesses/search'
headers = {'Authorization': 'Bearer %s %s yelp_api_key'}
url_params = {'term': 'coffee',  # Search for coffee...
              'radius': 1000,  # ...within 1000 meters...
              'location': '1300 University Ave, Madison WI'}
r = requests.get(url, headers=headers, params=url_params)
r.json()
```

URL to which to direct our request, specified in Yelp’s documentation.

Example: Querying Yelp’s Business Search Service

```python
import requests
url = 'https://api.yelp.com/v3/businesses/search'
headers = {'Authorization': 'Bearer %s' % yelp_api_key}
url_params = {'term': 'coffee',  # Search for coffee...
              'radius': 1000,  # ...within 1000 meters...
              'location': '1300 University Ave, Madison WI'}
r = requests.get(url, headers=headers, params=url_params)
r.json()
```

Yelp requires that we obtain an API key to use for authentication. You must register with Yelp to obtain such a key.

Example: Querying Yelp’s Business Search Service

```python
import requests
url = 'https://api.yelp.com/v3/businesses/search'
headers = {'Authorization': 'Bearer %s' % yelp_api_key}
url_params = {'term': 'coffee',
              'radius': 1000,
              'location': '1300 University Ave, Madison WI'}
r = requests.get(url, headers=headers, params=url_params)
r.json()
```

The resulting URL looks like this (can be accessed with `r.url`):

Notice that if you try to follow that link, you’ll get an error asking for an authentication token.

Example: Querying Yelp’s Business Search Service

```python
import requests
url = 'https://api.yelp.com/v3/businesses/search'
headers = {'Authorization': 'Bearer %s % yelp_api_key'}
url_params = {'term': 'coffee', # Search for coffee...
              'radius': 1000, # ...within 1000 meters...
              'location': '1300 University Ave, Madison WI'}

r = requests.get(url, headers=headers, params=url_params)
r.json()
```

This line actually submits the GET request to the URL, and includes the authorization header and our search parameters. `requests` handles all the annoying formatting and construction of the HTTP request for us.

Example: Querying Yelp’s Business Search Service

```python
import requests
url = 'https://api.yelp.com/v3/businesses/search'
headers = {'Authorization': 'Bearer %s' % yelp_api_key}
url_params = {'term': 'coffee', # Search for coffee...
              'radius': 1000, # ...within 1000 meters...
              # ...near the statistics department
              'location': '1300 University Ave, Madison WI'}
r = requests.get(url, headers=headers, params=url_params)
r.json()
```

requests packages up the JSON object returned by Yelp, if we ask for it. Recall that JSON objects in Python are really just dictionaries, so it makes sense that `r.json()` is a dictionary.

The businesses attribute of the JSON object returned by Yelp is a list of dictionaries, one dictionary per result. The name of each business is stored in its alias key.

More interesting API services

National Oceanic and Atmospheric Administration (NOAA)
https://www.ncdc.noaa.gov/cdo-web/webservices/v2

ESRI ArcGIS
https://developers.arcgis.com/python/

MediaWiki (includes API for accessing Wikipedia pages)
https://www.mediawiki.org/wiki/API:Main_page

Open Movie Database (OMDb)
https://omdbapi.com/

Major League Baseball
http://statsapi.mlb.com/docs

Of course, these are just examples. Just about every large tech company provides an API, as do most groups/agencies that collect data.