Using Web APIs with Python

What is an API?

API: Application programming interface

An API is how we interact with an external program in order to exchange data.

This is how applications do things like

- Access your calendar
- Give you up-to-date weather information

Why use an API?

If we wanted to, we could scrape web pages for most of the data that we want.

- Load website
- Make a search
- Scrape the page for relevant information

This means loading lots of unnecessary data, as well as being restricted to (mostly) public data.

Why use an API?

With an API, we can make requests to the server for specific information.

- We can make authenticated requests for private information (MUCH harder when scraping)
- We don't have to load whole websites
- We don't have to hunt through HTML for the information we care about.

Useful APIs

- Quandl Financial Data - Information on many unique financial metrics
- Twitter Search - Track historic trends on a given topic (realtime)
- Sentiment Analysis - What is the sentiment in a given block of text?
- Sports Updates - API to gather data on US sports teams (Soccer)
- Mapping Data - Collect information on routes, distances, directions, etc.
- Spotify - Check out song, album, and playlist info

Cool! So how can I use one?

Let's walk through API use in Python with the Google Distance Matrix API.

First, let's get set up:

- Need a Google account
- Need to register on Google Developer services
- Need to set up an API Key (ties our requests to us for billing purposes, but Google offers everyone a $200 credit per month for maps API requests)

WARNING

Using APIs frequently requires setting up a billing account.

- When we scrape the web, we pretend to be customers, so websites allow us to load their content at no cost (which is why many sites prohibit scraping)
- When we use APIs, we pay for access, because we are purchasing access to the firm's data, rather than looking to purchase goods/services
- PAY ATTENTION TO THE REQUESTS YOU MAKE... THEY COST MONEY

Setting up a Google Developer API

First, head to console.cloud.google.com, so we can set up our accounts.

- You will get a $300 credit to play around with on the platform, but we won't need it, since the Maps API offers its own monthly credit
- We need to create a project (do this in the blue bar)
- Head to the API menu, and choose credentials
- Follow the create credentials instructions for an API key

Using our API

Now, let's go back to the Google Distance Matrix API

- We can use the API key that we just created to complete the example link provided in the documentation
- When we copy and paste the link, we will see a page with JSON in our browser

Making Requests via URLs

When we want to request data from the Google Maps API, we do so with a custom URL that specifies

- the information we would like to collect
- additional parameters to clarify our request

The response provided when accessing that URL gives us the information that we requested from the API.

Automating API Requests

```
import requests
import json
import time
import datetime
import pandas as pd
```

- `requests` enables Python to process url requests
- `json` provides native JSON handling
- `time` helps us to check the system clock
- `datetime` provides the ability to parse dates and times.

Automating API Requests

```
def fetch_data(url):
    success = False
    while success is False:
        try:
            response = requests.get(url)

            if response.status_code == 200:
                success = True
        except:
            time.sleep(5)

            print("Error... Retrying")
    return response.text
```

Automating API Requests

```
results = {"timestamp" : [],
  "travel_time" : [],
  "distance" : []
  }

data = json.loads(fetch_data(req_url))
results['timestamp'].append(
  datetime.datetime.now())
results['travel_time'].append(
  data['rows'][0]['elements'][0]['duration']['value'])
results['distance'].append(
  data['rows'][0]['elements'][0]['distance']['value'])

results = pd.DataFrame(results)
```

Automating API Requests

```
api_key = "AIzaSyAwVHvWNPNOV05zA-hXBHC7DxOBK8AT0qs" # example key (not valid)
origin = '6708+Pine+Street+Omaha+NE' # work
destination = '2010+184th+Ave+NE+Redmond+WA' #my childhood home

site = "https://maps.googleapis.com" +
  "/maps/api/distancematrix/json?"
origin = "origins={}&".format(origin)
destination = "destinations={}&".format(destination)
key = "key={}".format(api_key)

req_url = site + origin + destination + key
```

We can start to see how we can use parameters to create an automated URL builder. Let's formalize this as a function

Automating API Requests

```
def map_data(api_key, origin, destination,
        frequency, duration):
    site = "https://maps.googleapis.com" +
      "/maps/api/distancematrix/json?" # put above
    origin = "origins={}&".format(origin)
    destination = "destinations={}&".format(destination)
    key = "key={}".format(api_key)

    req_url = site + origin + destination + key

    results = {
      "timestamp" : [],
      "travel_time" : [],
      "distance" : []
      }
    step = 1
    ... # to be continued on the next slide
```

Automating API Requests

```
def map_data(api_key, origin, destination,
    frequency, duration):
  ... # continued from last slide
  while (step <= int(duration*60 / frequency)):
    data = json.loads(fetch_data(req_url))
    results['timestamp'].append(
      datetime.datetime.now())
    results['travel_time'].append(
      data['rows'][0]['elements'][0]['duration']['value'])
    results['distance'].append(
      data['rows'][0]['elements'][0]['distance']['value'])

    print("Query Completed at {}".format(
      datetime.datetime.now()))
    step+=1
    time.sleep(frequency*60)

  return pd.DataFrame(results)
```

Executing our Query

We now have the functions in place to make queries automatically:

```
from ipywidgets import widgets

api_key = widgets.Text(placeholder="API Key")
origin = widgets.Text(placeholder="Origin Address")
destination = widgets.Text(placeholder="Destination Address")
frequency = widgets.Text(placeholder="Frequency (in minutes)")
duration = widgets.Text(placeholder="Duration (in hours)")

elements = [api,origin, destination, frequency, duration]
widgets.Box(elements)
```

Executing our Query

We now have the functions in place to make queries automatically:

```
if __name__ == '__main__':
  data = map_data(api_key, origin, destination,
    frequency, duration)
```

At this point, we will (eventually) get a DataFrame of our results when the function terminates

Results

timestamp	travel_time	distance
2019-04-09 11:37:30.747668	1343	23942
2019-04-09 11:38:30.958629	1343	23942
2019-04-09 11:39:31.504978	1343	23942
2019-04-09 11:40:31.738479	1343	23942
2019-04-09 11:41:31.931956	1343	23942
...	...	...

Lab Time!