Lab Report

Title: GUI Goes the Way of the Dinosaur: Building ETL Pipelines to Three APIs
Notice: Dr. Bryan Runck
Author: Cecelia Isaac
Date: 2/11/2021

Project Repository: *https://github.com/CeceliaAi/GIS5572/tree/master/Lab1*

Abstract

In this lab, we will explore how to decompose APIs to make ETL pipelines. An ETL pipeline is a way to extract, transform, and load data, but the process is different for different APIs. Therefore, we must develop a unique method for each of the three APIs we want to access. The resulting code will have some similarities in the construction of URLs and the use of the Requests library. The models will show the differences in the structure of the data. Our results will show three pipelines that construct a URL, pull in data, and save that data to our local machine in some file format. We can verify our results by check if the file downloaded properly.

Problem Statement

Our challenge is to create, and then compare and contrast, three different ETL pipelines from three APIs. The pipelines must be different because the way the APIs are structured changes how we will extract the data, and the format it will be in once extracted. The APIs we will use are the Minnesota Geospatial Commons, Google Places, and NDAWN. The first step will be to compare and contrast the conceptual models of the APIs, since this will tell us how the data is structured and help inform the code. In practice though, the process is more iterative, so first the model will have to be formed based on initial thoughts, then the code can be written, and then the model can be edited based on this experience.

Table 1. Data Sources

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
1	Minnesota Geospatial Commons	CKANS API				Use ETL to specify format
2	Google Places	Google Places API				Use ETL to specify format
3	NDAWN	NDAWN API				Use ETL to specify format

Input Data

There is no input data, as our goal is to download the data we need. We use Python to access the APIs of three different websites. We will download data from each API. For NDAWN, we downloaded a csv of Station 78's weather data from February 7th, 2021. From Minnesota Geospatial Commons, we downloaded a zipfile of a shapefile. For Google Places, we obtained a kml.

Table 2. Data

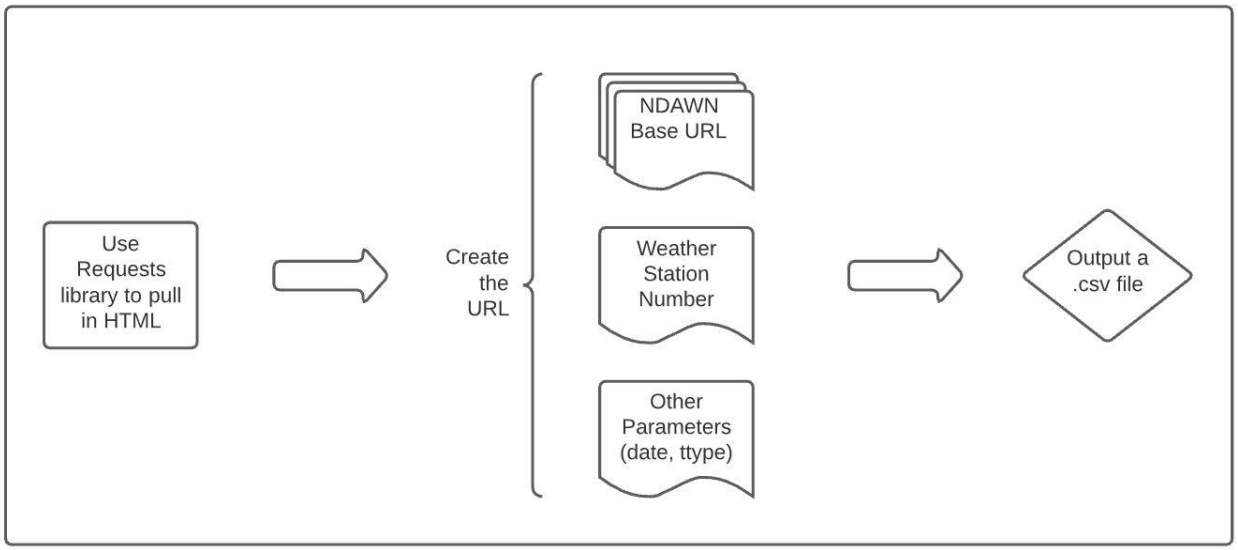
#	Title	Purpose in Analysis	Link to Source
1	Minnesota Geospatial Commons		https://gisdata.mn.gov/content/?q=help/api
2	Google Places		https://developers.google.com/places/web-service/overview
3	NDAWN		https://ndawn.ndsu.nodak.edu/

Methods

NDAWN

To create an ETL pipeline for NDAWN, we set the base URL. Then we created a for loop that would loop through the weather stations and return URLs for the day of February 7th. We called one of those URLs with requests.get and assigned that a variable. We then used that variable to write the result to a CSV file.

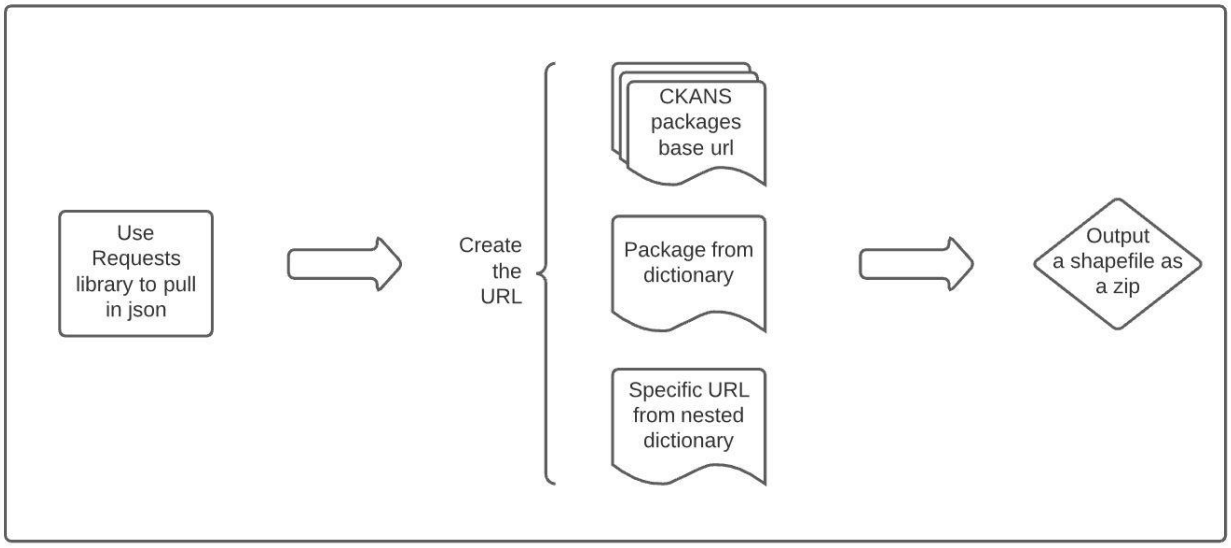
Figure 1. NDAWN API flow diagram.



Minnesota Geospatial Commons

To create an ETL pipeline for Minnesota Geospatial Commons, we used the packages URL and requests.get to pull in a JSON dictionary of the packages content. Then we chose a package to assign to the "package" variable. We built the URL as with NDAWN, but this time had to move through the nested dictionaries to find the 'url' key that we could use to build the URL we wanted to use. Once this was done, we used requests.get again to write the zip file to the chosen directory.

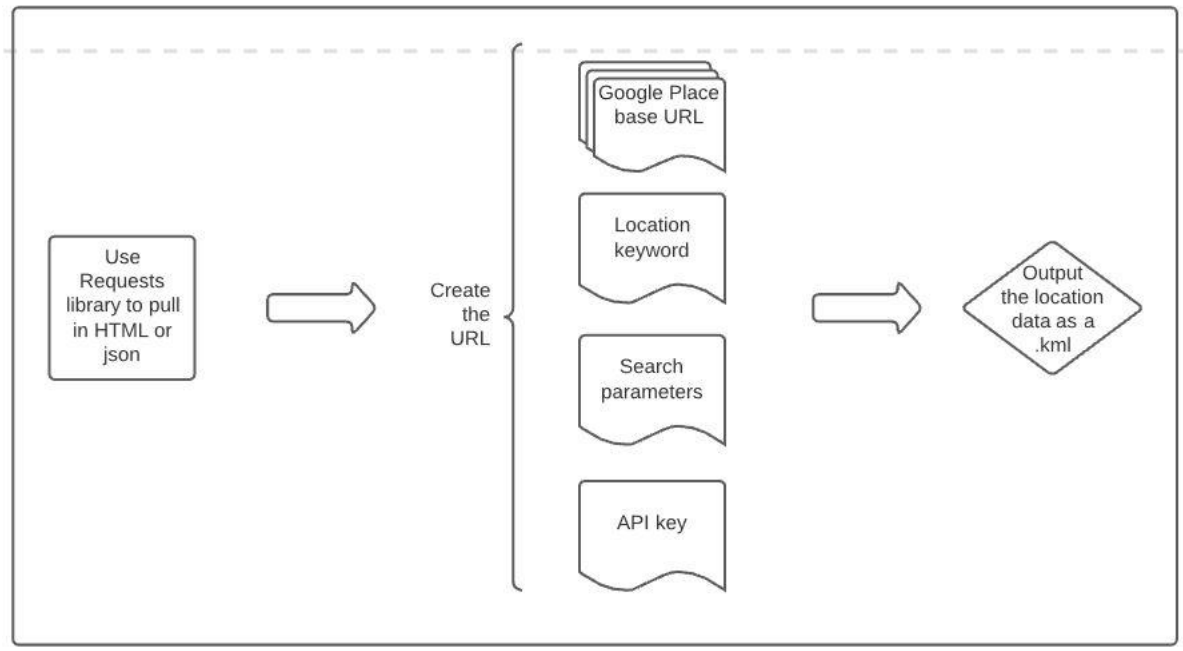
Figure 2. MN Geo API flow diagram.



Google Places

As with before, we assigned a variable to the base URL for Google Place. We then used requests.get to call the URL, which is composed of the base URL and the parameters, including the unique API key. Finally, we called the URL with JSON to see the contents.

Figure 3. Google Places API flow diagram.



Results

The results are three notebooks of Python script, one for each API. The scripts can be broken down into three parts: setting exploring the server setup through pulling in the HTML or JSON, parsing the data to create a URL to pull in the specific file, and saving that file to the local server. In our case, we saved a CSV file and a zip of a shapefile to the computer. As mentioned in the Methods section, the code differs for each API, but result is a usable file in the desired format.

Figure 4. NDAWN HTML parsing to CSV

```
#set the base URL as a variable
base = "https://ndawn.ndsu.nodak.edu"
```

```
#create a for loop that will create URLs for each station
for i in range (0,140):

    print("https://ndawn.ndsu.nodak.edu/get-table.html?station=" + str(i) + "&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_da
```

```
ps://ndawn.ndsu.nodak.edu/get-table.html?station=0&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=1&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=2&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=3&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=4&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=5&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=6&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=7&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=8&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
ps://ndawn.ndsu.nodak.edu/get-table.html?station=9&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=2021-02-06&end_date=202
2-06
```

```
#call for that URL and set it as a variable
downloadstation1 = r.get("https://ndawn.ndsu.nodak.edu/table.csv?station=78&variable=ddmxt&year=2021&ttype=daily&quick_pick=&begin_date=
```

```
#write the result to a csv file

open("stationdata78.csv","wb").write(downloadstation1.content)
```

Figure 5. MN Geo JSON dictionaries to zipped shapefile

```
#assign a variable to a package
#this package can later be swapped out for another
package = 'us-mn-state-metrogis-plan-regonal-parcels-2020'
```

```
#put together the base url and the selected package
base_url = 'https://gisdata.mn.gov/api/3/action/package_show?id='
package_information_url = base_url + package

#call the URL and load with json as above
package_information = r.get(package_information_url, verify = False)
package_dict = json.loads(package_information.content)


package_dict = package_dict['result']
```

```
C:\Program Files\ArcGIS\Pro\bin\Python\envs\arcgispro-py3\lib\site-packages\urllib3\connectionpool.py:988: InsecureRequestWarning: Unverifi
ed HTTPS request is being made to host 'gisdata.mn.gov'. Adding certificate verification is strongly advised. See: https://urllib3.readthed
ocs.io/en/latest/advanced-usage.html#ssl-warnings
  InsecureRequestWarning,
```

```
#the dictionaries are nested, so we have to go another level in to format the URL
data_url = package_dict['resources'][1]['url'] #number can be changed to get to the shapefile
print(data_url)
```

```
Data url:     https://resources.gisdata.mn.gov/pub/gdrs/data/pub/us_mn_state_metrogis/plan_regonal_parcels_2020/fgdb_plan_regonal_parcels_2
020.zip
```

```
#download the selected shapefile as a new zip file to the desktop
ParcelData = r.get(data_url)
with open("/Users/celia/Desktop/ParcelData.zip", "wb") as zip:
    zip.write(ParcelData.content)
```

Figure 6. Google Places JSON to kml file

```
import arcpy
import requests as r
#demo = r.get(https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=Museum%20of%20Contemporary%20Art%20Australia&inpu
```

```
goog_base = "https://maps.googleapis.com/maps/api/place/findplacefromtext/"

demo = r.get("https://maps.googleapis.com/maps/api/place/findplacefromtext/json?" +
             "input=Mongolia" + #this is keyword
             "&inputtype=textquery" +
             "&fields=photos,formatted_address,name,geometry"
             + "&key=AIzaSyDh4Bkc9OeXeMI4w9XOWjEJHCLeyeMEIEc")

demo.json()
```


```
{'candidates': [{'formatted_address': 'Mongolia', 'geometry': {'location': {'lat': 46.862496, 'lng': 103.846656}, 'viewport': {'northeast':
{'lat': 52.148355, 'lng': 119.9315098}, 'southwest': {'lat': 41.581833, 'lng': 87.7344789}}}, 'name': 'Mongolia', 'photos': [{'height': 40
6, 'html_attributions': ['<a href="https://maps.google.com/maps/contrib/103436265642766943887">A Google User</a>'], 'photo_reference': 'ATt
YBwKM6QjkJRQuZwchleIFrC3ctBFblDQJNaz4_J3IRkeA84i2dD0s0SAaV_y52_O8trx4rmXcTkWB3FSXednczcaipzqFHfRrjs3sXvpJpO7s5brJ2F19z_nbsA_Xspifu9vDPkm0Es
6eU3u-SZB4A3TJ8Dv62vjc733jovI1osKvmZLl', 'width': 720}]}], 'status': 'OK'}
```


Results Verification

The first way the results can be verified is by producing code without any error messages. The other way is by locating the file on the local computer and making sure it is not corrupted or empty. The status of the URL code also be checked before downloading anything, to make sure a 404 error did not come back.

Discussion and Conclusion

I learned how to construct URLs and uses them to call information in HTML or JSON format. I also learned how to download this information to my computer programmatically. This connects to one part of the main problem, building pipelines. The second part of the main problem was comparing and contrasting these pipelines. While I was able to work with different APIs and build pipelines to download data, comparing and contrasting them is still difficult for me. I do not fully understand what is going on behind the scenes in the API. I can see that, for example, MN Geo uses a nested dictionary, but I still feel that there are a lot of gaps in my knowledge, and therefore the code I produced. The models were difficult to write out because of this feeling that there are blank spots in my understanding. I did learn a lot more about APIs, so I will have more strategies to use in the future.

References

Saba, G. (n.d.). *Python Download File Tutorial – How To Download File From Internet Using Python*. Retrieved February 5, 2021, from https://www.simplifiedpython.net/python-download-file/

DataCareer. (n.d.). *Tutorial for retrieving data from the Swiss Open Data portal*. Retrieved February 5, 2021, from https://www.pythonsherpa.com/tutorials/2/

Garazov, P. (n.d.). *What Is An API*. Retrieved February 5, 2021, from https://www.freecodecamp.org/news/what-is-an-api-in-english-please-b880a3214a82/

Self-score

Category	Description	Points Possible	Score
Structural Elements	All elements of a lab report are included **(2 points each)**: Title, Notice: Dr. Bryan Runck, Author, Project Repository, Date, Abstract, Problem Statement, Input Data w/ tables, Methods w/ Data, Flow Diagrams, Results, Results Verification, Discussion and Conclusion, References in common format, Self-score	28	**28**
Clarity of Content	Each element above is executed at a professional level so that someone can understand the goal, data, methods, results, and their validity and implications in a 5 minute reading at a cursory-level, and in a 30 minute meeting at a deep level **(12 points)**. There is a clear connection from data to results to discussion and conclusion **(12 points)**.	24	**24**
Reproducibility	Results are completely reproducible by someone with basic GIS training. There is no ambiguity in data flow or rationale for data operations. Every step is documented and justified.	28	**28**
Verification	Results are correct in that they have been verified in comparison to some standard. The standard is clearly stated **(10 points)**, the method of comparison is clearly stated **(5 points)**, and the result of verification is clearly stated **(5 points)**.	20	**15**
		100	**95**