Lab Report

Title: *Lab1: Deconstructing the API's of NDAWN, Google Places, and MNGEO*
Notice: Dr. Bryan Runck
Author: Lucas Rosen
Date: 2/11/2021

Project Repository: ***https://github.com/LRosen656/GIS5572.git***

Abstract
The purpose of this assignment is to compare the APIs of NDAWN, Google Places, and MNGEO for viewing online. Each API was requested into Jupyter notebook. NDAWN data was parsed from the url. A search inquiry was used to narrow data in Google Places and MNGEO. MNGEO had to be further parsed to get to the metadata. Results created a csv from NDAWN, a json from Google Places, and a shapefile from MNGEO.

Problem Statement

The purpose of this lab is to compare the Application Programing Interfaces (APIs) of North Dakota Agricultural Weather Network (NDAWN), Google Places, and Minnesota Geospatial Commons (MNGEO). The results will show data requested from each API.

Table 1.The required data is the APIs from NDAWN, Google Places, and MNGEO

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
1	NDAWN	North Dakota Agricultural Weather Network Website	NA	NA	NA	NA
2	Google Places	Google Places API	NA	NA	NA	NA
3	MNGEO	Minnesota Geospatial Commons API	NA	NA	NA	NA
4						

Input Data

There was not any required data. Rather, the goal of the assignment was to view the API. I did download data from each site, but that was only to prove that the API functioned correctly. NDAWN links to tables of weather stations, Google Places finds locations in google, and MNGEO links to spatial data submitted by Minnesota regulatory agencies such as MPCA, DNR, MDA etc.

Table 2. Links to the sites used for the API

#	Title	Purpose in Analysis	Link to Source
1	NDAWN	View the API in Jupyter Notebook	https://ndawn.ndsu.nodak.edu/
2	Google Places	View the API in Jupyter Notebook	https://maps.googleapis.com/maps/
3	MNGEO	View the API in Jupyter Notebook	https://gisdata.mn.gov/

Methods

All methods were done in Arc GIS Pro Jupyter Notebook.

NDAWN

NDAWN has two main links: one that links to a table called "get-table", and one that downloads a table as a csv called "table.csv". To start, I created variables for both called "Weather_table" and "Weather_csv" respectively. After that, variables needed to be created for the following parameters: station number, variable (desired data), timing type, quick time, begin date, and end date. Station Number connects to the weather station, for example, station 118 connects to Becker, MN (not all stations are in North Dakota). Variable is what you want measured for example windspeed has the variable "ws". Timing type describes how you want it measured. I can be hourly, daily, weekly, monthly, or yearly (among others but that is outside of the main site). Finally, begin date and end date puts the date ranges you want in the data and quick time automatically fills the fields if used in the GUI (or can be typed but that would be redundant).

Once variables were created for each parameter, I connected them to the "Weather_table" and separated them a string of the desired variable and "&". I first made a for loop the station number and had it set to the range of stations (2-142). I then set the variable for hourly rain called "hdr" (variable is connected to timing type) and therefore, the timing type was "hourly". I left the quick pick blank, and finally set the begin and end date to 2/5/2021. The result showed a list of hourly rain by station number.

To download a csv, I imported requests and used "requests.get" to get the weather_csv of hourly rainfall in station 2 (Eldred, MN). The parameters are the same as the "Weather_table" except I did station[0] (that was station 2 in the list) so that I would just get one csv. Finally, I downloaded the csv using "open". The Notebook is in the directory and the flow diagram can be seen in **Figure 1**.

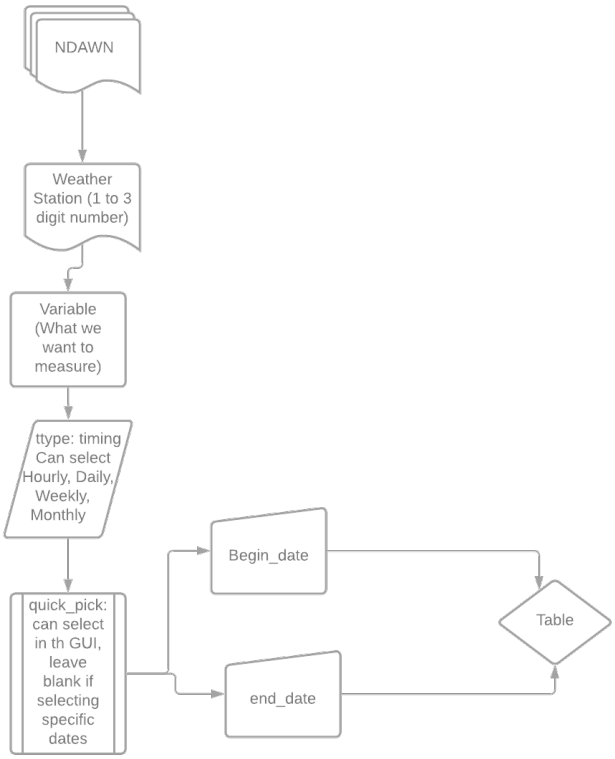

Figure 1: NDAWN Flow Diagram.

Google Places

Google Places has three main place searches: and find place (called “find place from text”), a query search (called “text search”), and a nearby search (called “nearby search). Because the near by search requires a location (latitude and longitude), I just did find place and query search in the notebook. However, before I could access the API, I first had to create an API key by creating an account.

Once I had a key, I started to do a “find place”. In this case, the place I was searching for was the O. Meredith Wislon Library. So, I create a list that separated each word with a comma to add to the url. Once the url was created (see Notebook), I added the list by joining it with “%20” (that is how this variable recognizes spaces) and did a request. The result showed a place id as a json.

To make sure it was correct, I also did a query search. The parameters were the same except I then joined the library list with a “+”. The result showed a bigger json that included a location, a photo, and other details. More importantly, it included a place id that was the same as the previous search. **Figure 2** shows the flow diagram.

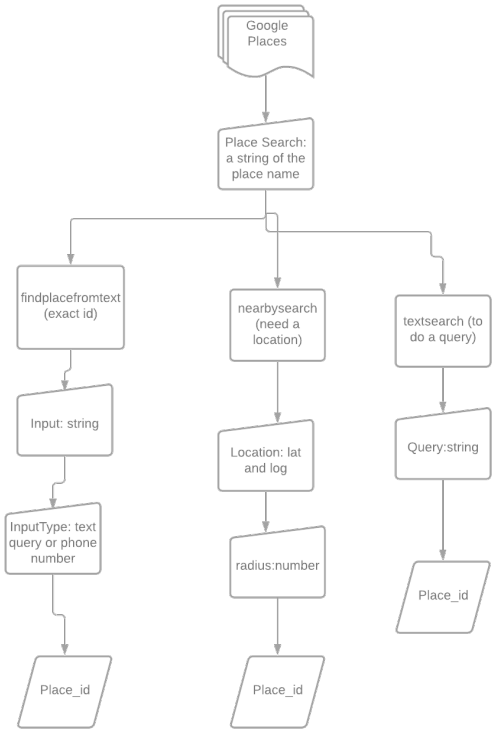

Figure 2: Google Places Flow Diagram

MNGEO

MNGEO uses a CKAN API. This is an open-source API that divides variables into packages, groups, and tags. Once I created a package variable for the MNGEO url (see Notebook) I could perform a search inquiry. One of the queries in the tags was "ecological" so that is the one I went with. I then created a base query search and added the tag. Once that was done, I requested it as a json a made it into a dictionary variable. From there, I iterated to the useful information ("result") and then to the metadata ("results" (yes, it is confusing)).

The metadata is where all the datatypes are including (in this case) the desired shapefile. To find the shapefile, I did some trial and error by printing the url (zip file to download) and the name (shows the datatype). Once I found a shapefile, I downloaded the zip file and then imported the zipfile package so I could convert it to a shapefile. **Figure 3** shows the flow diagram.

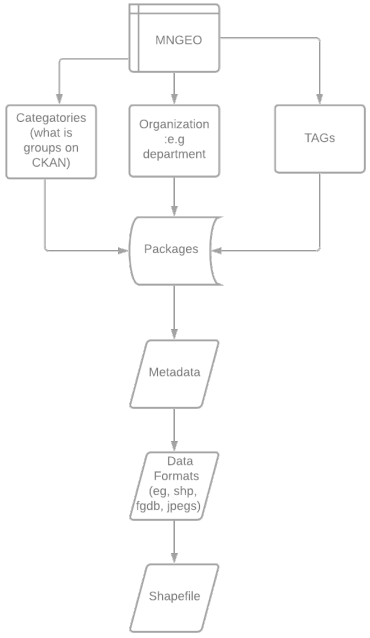

Figure 3: MNGEO Flow Diagram

Results

Each Notebook shows details of the APIs and downloaded data from them. The data downloaded from NDAWN was a csv, the data download from Google places was a json, and the data downloaded from MNGEO was a shapefile.

Results Verification

I was able to download data from each API so that shows that it worked at least once. However, making sure the API works in every possible condition is a bit tricky. When viewing the NDAWN tables, I noticed that not all the station numbers had results. To resolve this, I made sure that the csv download would be one with a valid station number. For Google Places, I compared the same search using two different methods. However, it would be unreasonable to exhaust every possible search. Finally, I was just seeking one possible query in MNGEO and did not check different results or ids.

Discussion and Conclusion

NDAWN

NDAWN was the only website that I did not need to see the json metadata. This is because all the variables were neatly displayed in the url. Even with such a simple API, NDAWN can create many combinations of weather data. One note is that the quick pick option is not useful in the notebook since I had to fill out the beginning and end date anyway.

Google Places

There are many more variables in the Places API than just the Place Search including Place Details and Autocomplete. But that is far outside of the scope of this assignment. Also, it took me a while to figure out how to get an API key. Because an authorization key with payment information is required, it was a major obstacle just to get started.

MNGEO

MNGEO is a convoluted json filled with dictionaries and a list that ultimately get the metadata. I had to get some help from other students and outside resources to get it to work. However, as soon as I got to the metadata, I was able to get to the data types and the shapefile.

Conclusion

While all the APIs have different methods and formats, the goal of each API is fundamentally the same: to provide the user with data in an understandable and comprehensive manner.

References

API guide¶. (n.d.). Retrieved February 8, 2021, from https://docs.ckan.org/en/2.9/api/

Overview | Places API | Google developers. (n.d.). Retrieved February 3, 2021, from https://developers.google.com/places/web-service/overview

Pythonsherpa. (n.d.). Retrieved February 9, 2021, from https://www.pythonsherpa.com/tutorials/2/

Self-score

Fill out this rubric for yourself and include it in your lab report. The same rubric will be used to generate a grade in proportion to the points assigned in the syllabus to the assignment.

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	21
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	26
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	20	19

	verification is clearly stated **(5 points)**.		
		100	94