Lab Report

Title: Lab 1
Notice: Dr. Bryan Runck
Author: Cole Anderson
Date: 2/10/21

Project Repository: https://github.com/and04671/GIS5572/tree/main/Lab1

Abstract

In this lab, three different APIs were decomposed, modeled, and utilized to download data over the web. CKAN, Google Places, and NDAWN were tested for the lab. The conceptual models all followed a call-download format, requiring search terms or HTML elements to acquire the desired data. Models varied on the type of search query and output data. All of the APIs successfully downloaded the desired data, which verified as the same as manual download. The user learned how to write fast API scripts for future laboratory.

Problem Statement

Decompose API interfaces to concept models. Compare and contrast models and ETL routines. Create python scripts to utilize CKAN, NDAWN, and Places APIS to download specified data.

Table 1. Lab Required Components

#	Requirement	Defined As	Spatial Data	Attribute Data	Dataset	Preparation
1	Decompose models/compare	Break apart APIs into conceptual models and illustrate differences	n/a	n/a	n/a	Inspect HTML and URL
2	Write API ETLs	Create Jupyter notebooks to retrieve data from APIs	n/a	n/a	n/a	Read API documentation
3	Use API to download data	Successfully use aforementioned notebooks to download data to computer	Output of ETL	Output of ETL	Output of ETL	n/a
4						

Input Data

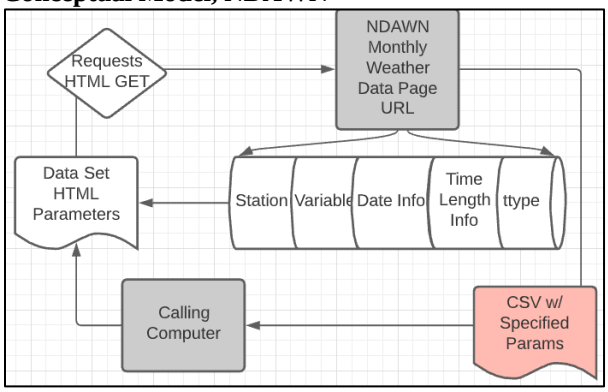
Describe the data in two paragraphs max. Fill out the table.

Table 2. APIs Tested/Utilized

#	Title	Purpose in Analysis	Link to Source
1	MN Geospatial Commons CKAN	Metadata based API, JSON dictionary and ZIP output	Minnesota Geospatial Commons
2	Google Places API	Search based API, dictionary output	Google Places
3	NDAWN	HTML based API, CSV output	NDAWN

Methods

Conceptual Model, NDAWN



Method Flow, NDAWN

The NDAWN model offered the most HTML based API. In order to call information from this API, we needed to inspect the HTML elements in the web page's URL. These elements include the station number, variable recorded, and temporal variables. In this URL, note that the required elements to locate that particular table are connected by ampersands: "*...edu/get-table.html?station=78&variable=mdmxt&year=2021&ttype=monthly&…".* The information for each element can be discovered via the inspected HTML code. Here, the names for each selection box on the webpage can be located:

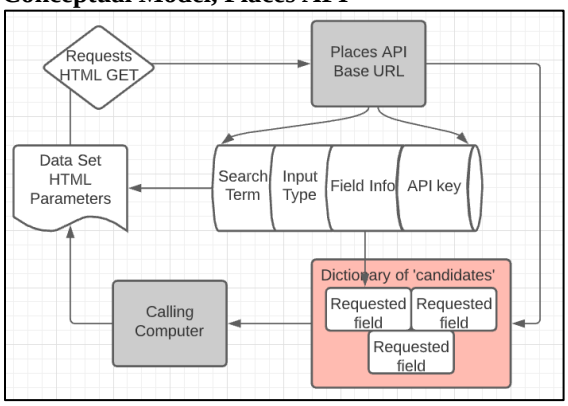
```
<div class="col-group date-num-months">
  <p class="instr">…</p>
  <label for="begin-date">Begin date:</label>
  <input id="begin-date" name="begin_date" maxl
  "2020-01">
  <br>
  <label for="num-months">Number of months:</la
  <input id="num-months" name="count" size="3"
  <button class="submit ui-button ui-corner-all
```

Each of these elements can then be listed as part of a data dictionary in the calling script, set as a 'payload' in our example.

```
payload = {'station': station,
      'variable':variable,
      'year': year,
      'ttype':'monthly',
      'quickpick':'',
       #begin date needs a DD if weekly is selected
      'begin_date': str(year)+'-01',
      'count':12}
```

The keys in the dictionary correspond to the items on the left of the equal sign in each element in the URL!
Now that the parameters to be passed to the NDAWN webpage are established, a GET requests is sent to the base URL with that information. While the actual NDAWN table view page lists the end of the URL as "get-table.html", a manual test download reveals it is instead "table.csv". The latter need be in the URL instead. The response to the GET request creates a data object to be saved to the calling computer. The content of the response is saved to a file with a representative name, and closed. A CSV is created with each loop iteration: each station requested, variable requested, and year.

Conceptual Model, Places API



Method Flow, Places API

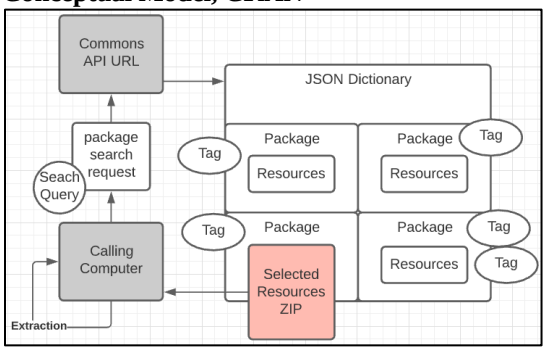
The Places API is similar to the NDAWN API with some notable exceptions. First, an API key is required. Second, the returned response is not a CSV but a dictionary of results. Third, the structure of 'search terms' in the GET request is notably different. The base (without search terms) URL depends on the desired type of search. For this example, it is 'find a place from the text provided'. The additional search add-ons are a set of search words separated by '%20', a type of input corresponding to that search query, and a set of fields to return. Note that this is fields to return corresponding to a place, NOT a set of field values used to narrow down to a place/result, making it different from the NDAWN version. The resulting URL to be sent via GET, with the information in the below image, looks like this:

Base URL + input=Potato%20Museum&inputtype=textquery&fields=place_id,formatted_address&key=AIz…

```
search = 'Potato%20Museum'
#the terms to search for. put in %20 for space
inp_type = 'textquery'
#input type
fields = 'place_id,formatted_address'
#desired fields
```

The API key required at the end must be acquired via Google Cloud Platform access. With this information, a GET request is sent from the calling computer to the API. The returned response is a JSON dictionary of possible places, with the fields requested.

Conceptual Model, CKAN



Method Flow, CKAN

The CKAN API works in a rather different way than the other two API interfaces, utilizing metadata instead of direct data links. Digging downward through very long JSON objects is central, instead of using URL elements like before. The URL seems actually rather short. The Minnesota Geospatial Commons uses the CKAN, and has the base URL 'https://gisdata.mn.gov/api/3/action/<xxxxxx>'. The last element represented by X's is the type of search plus the search query. For example, this might be a package (dataset) search, a search by tags, by recent changes. The search query is similar to the Places API. In this example, it is 'package_search?q=waterways'. This completed URL

is again sent to the data page via a GET request, except the verification certificate must be forcefully ignored. The response must be translated into a JSON, and then parsed through.

```
response = requests.get(big_url, verify = False)
#turn result into JSON
json_response = json.loads(response.content)
#dig down through dictionary layers
result_options = json_response['result']['results']
chosen_result = result_options[result_num]
resources_under_result= chosen_result['resources'][resource_num]
chosen_resource = resources_under_result['url']
print(chosen_resource)
```

As illustrated in the concept diagram, the dictionary contains multiple package (dataset pages) results. The user must select the desired package via slicing. The layer underneath the package contains multiple resources, usually documents like an SHP or XML. The user must once again choose a resource by slicing. The resource has an item in its dictionary that contains the URL to that resource. There are often other metadata items at each level of depth, creating the need to go two levels to go from the very top to the results. Note the set of two [][] brackets at those levels. This new whole URL is sent out as a GET request. Again, the response to the calling computer is saved. This time, the file must also be unzipped for some applications (not for ArcOnline, but for ArcPro). This is done via the zipfile module.

Results

Show the results in figures and maps. Describe how they address the problem statement.

The result of the NDAWN API script is a csv.
The GET request (one of several):
https://ndawn.ndsu.nodak.edu/table.csv/station=3&variable=mdmxt&year=2020&ttype=monthly&quickpick=&begin_date=2020-01&count=12
The corresponding result csv:

	A	B	C	D	E	F	G	H	I	J
1	Data from	North Dakota Agricultural Weather Network https://ndawn.ndsu.nodak.edu								
2	Monthly Observation Table for January 1 2020									
3	Flag Definition Line: M - Missing; E - Estimated; N/A - Not Available									
4	Station Na	Latitude	Longitude	Elevation	Year	Month	Max Temp	Number M	Number Estimated	
5		deg	deg	ft			Degrees F			
6	Perley	47.179	-96.68	895	2020	1	18.892	0	0	
7	Perley	47.179	-96.68	895	2020	2	21.24	0	0	
8	Perley	47.179	-96.68	895	2020	3	34.226	0	0	
9	Perley	47.179	-96.68	895	2020	4	48.161	0	0	
10	Perley	47.179	-96.68	895	2020	5	64.988	0	0	
11	Perley	47.179	-96.68	895	2020	6	81.899	0	0	
12	Perley	47.179	-96.68	895	2020	7	83.288	0	0	
13	Perley	47.179	-96.68	895	2020	8	79.931	0	0	
14	Perley	47.179	-96.68	895	2020	9	69.049	0	0	
15	Perley	47.179	-96.68	895	2020	10	48.212	0	0	
16	Perley	47.179	-96.68	895	2020	11	43.178	0	0	
17	Perley	47.179	-96.68	895	2020	12	30.282	0	0	

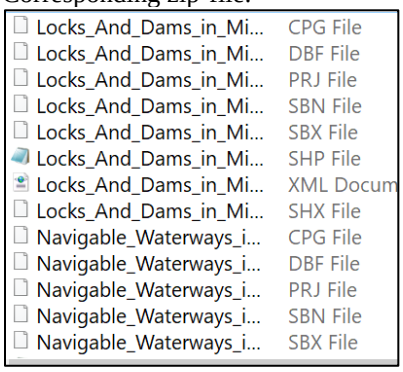
The result of the Places API is a JSON dictionary.
The GET request:
https://maps.googleapis.com/maps/api/place/findplacefromtext/json?input=Potato%20Museum&inputtype=textquery&fields=place_id,formatted_address&key=AIzaSyBjn4F98m1QtOZC0VzPfbQpOXicJPOzEN8
The corresponding result dictionary:

```
b'{
b'   "candidates" : ['
b'      {'
b'         "formatted_address" : "130 NW Main St, Blackfoot, ID 83221, United States",'
b'         "place_id" : "ChIJPTg-hZ0aVVMRiEnRXIW3Lt0"'
b'      }'
b'   ],'
b'   "status" : "OK"'
b'}'
```

The result of the CKAN API is a JSON dictionary and eventually a zip file.
The first GET request (0th package, 0th resource): https://gisdata.mn.gov/api/3/action/package_search?q=waterways
Corresponding JSON dictionary :
{'help': 'https://gisdata.mn.gov/api/3/action/help_show?name=package_search', 'success': True, **'result'**: {'count': 4, ' sort': 'score desc, metadata_modified desc', 'facets': {}, **'results':** [{'license_title': 'License not specified', 'maintainer'

: None, 'relationships_as_object': [], 'private': False, 'maintainer_email': None, 'num_tags': 6, 'id': 'f3773893-e1e4-49a8-86fa-622066b34764', 'metadata_created': '2018-07-19T10:02:16.070039', 'metadata_modified': '2021-02-04T09:09:54.727898', 'author': None, 'author_email': None, 'state': 'active', 'version': None, 'creator_user_id': '61044ca7-ee56-4019-aef7-f7c3b5e06e3a', 'type': 'dataset', **'resources'**: [{'mimetype': None, 'cache_url': None, 'hash': '', 'description': '', 'gdrsResGuid': '{ab6ef069-321b-434e-997b-421e53171ba0}', 'cache_last_updated': None, 'url': **'https://resources.gisdata.mn.gov/pub/gdrs/data/pub/us_mn_state_dot/trans_water_navigations/shp_trans_water_navigations.zip'**, 'name': 'Shapefile', 'format': 'SHP', 'package_id': 'f3773893-e1e4-49a8-86fa-622066b34764', 'created': '2021-02-04T09:09:55.242348', 'state': 'active', 'mimetype_inner': None, 'last_modified': None, 'position': 0, 'revision_id': 'e0d488f9-e423-45b2-89f7-f36fe00d30e3', 'url_type': None, 'id': 'a9a7f141-e930-42ec-9c4c-5cf121602f42', 'resource_type': 'shp', 'size': None….

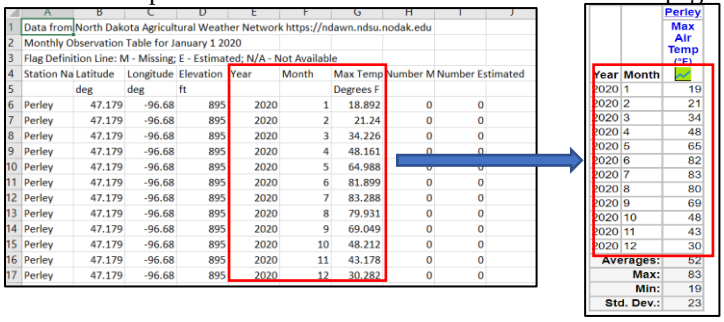
The second GET request: https://resources.gisdata.mn.gov/pub/gdrs/data/pub/us_mn_state_dot/trans_water_navigations/shp_trans_water_navigations.zip
Corresponding zip-file:



Results Verification

To verify that the calling script is working correctly and not just giving a lucky result, the script download and manual download of each data result was compared.

NDAWN script: NDAWN table page:

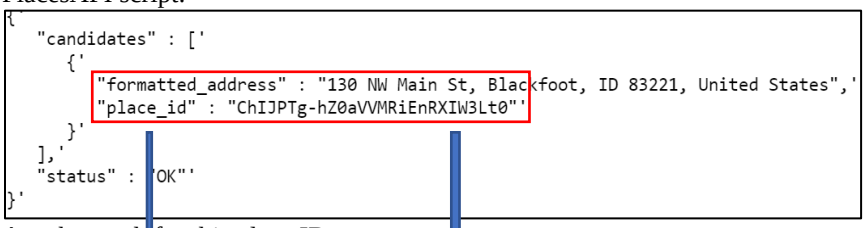


	A	B	C	D	E	F	G	H	I	J
1	Data from	North Dakota Agricultural Weather Network https://ndawn.ndsu.nodak.edu								
2	Monthly Observation Table for January 1 2020									
3	Flag Definition Line: M - Missing; E - Estimated; N/A - Not Available									
4	Station Na	Latitude	Longitude	Elevation	Year	Month	Max Temp	Number M	Number Estimated	
5		deg	deg	ft			Degrees F			
6	Perley	47.179	-96.68	895	2020	1	18.892	0	0	
7	Perley	47.179	-96.68	895	2020	2	21.24	0	0	
8	Perley	47.179	-96.68	895	2020	3	34.226	0	0	
9	Perley	47.179	-96.68	895	2020	4	48.161	0	0	
10	Perley	47.179	-96.68	895	2020	5	64.988	0	0	
11	Perley	47.179	-96.68	895	2020	6	81.899	0	0	
12	Perley	47.179	-96.68	895	2020	7	83.288	0	0	
13	Perley	47.179	-96.68	895	2020	8	79.931	0	0	
14	Perley	47.179	-96.68	895	2020	9	69.049	0	0	
15	Perley	47.179	-96.68	895	2020	10	48.212	0	0	
16	Perley	47.179	-96.68	895	2020	11	43.178	0	0	
17	Perley	47.179	-96.68	895	2020	12	30.282	0	0	

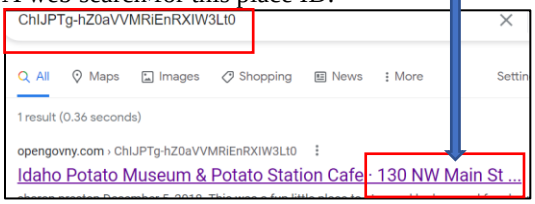
	Perley Max Air Temp (F)
Year Month	
2020 1	19
2020 2	21
2020 3	34
2020 4	48
2020 5	65
2020 6	82
2020 7	83
2020 8	80
2020 9	69
2020 10	48
2020 11	43
2020 12	30
Averages:	52
Max:	83
Min:	19
Std. Dev.:	23

The station, temperatures, dates, and month numbers all match.

PlacesAPI script:

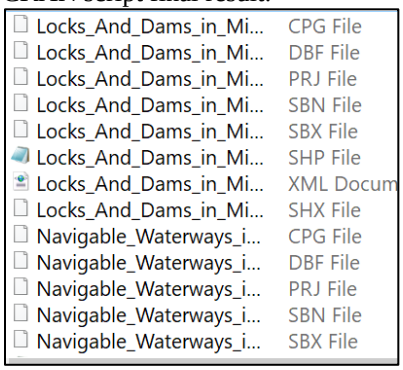


A web search for this place ID:

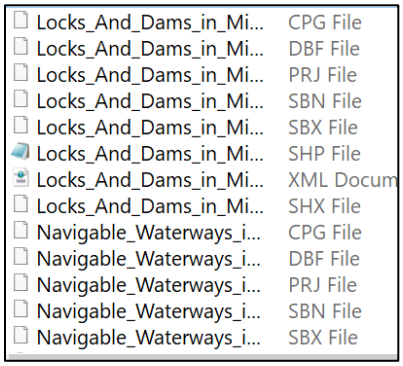


The web search for the same place ID turns up the same address.

CKAN script final result:



CKAN Commons Website download:



These two results are exactly the same files, additionally checked by adding both sets to ArcMap.

The results of each type of download for all three API scripts are identical, meaning the GET requests were able to successfully reach the desired resource in all cases! These are not lucky results but the intended ones.

Discussion and Conclusion

What did you learn? How does it relate to the main problem?

The first thing that varied between APIS was the structure of the URL in the GET request; what type of information was required. For NDAWN, the URL required the base URL of the webpage, plus a value for each type of narrowing parameter. Effectively, Base + narrowing params. The Places API still uses a base URL, but it designates a type of search and not a webpage. Meanwhile, the params added are a search term and the desired response fields. Effectively, Base + query + desired returned fields. The CKAN API still uses a base URL for each type of search, but only a search query. Most of the 'narrowing' happens after the first return. Only the CKAN API requires both a metadata GET request and a final file GET request. All of the API scripts made use of a requests.get call with a URL corresponding to the desired content.

The second major difference between the three APIs was the structure of the GET response. NDAWN returns a simple data table, which is simply downloaded. The Places API returns a dictionary object with the fields requested in the GET request. There is nothing to download here, but instead possible locations. The CKAN API also returns an enormous dictionary/list object. This is not a simple list, but levels of different datasets and results from the query. It is lengthy to sort through. The second GET request in CKAN returns a zipfile of the selected resource.

At a high level, the process of obtaining data from each API is similar. First, inspect the actual base data page, URL, and documentation for clues that determine what is downloaded and from where. Find what types of URL add ons

are needed. Second, send relevant parameters in a GET request to the URL containing those parameters for the data you need. Third, the API sends a data file back to the computer, where it can be downloaded. The types of parameters involved, and how they can be determined, are different for each model. The parameters are all effectively a way to 'search' the API for data and select the correct information.

This was a tricky lab at the beginning, but once it was realized that all APIs follow a similar type of flow, it became easier to write the scripts for the latter APIs. The documentation for the CKAN was terrible, resulting in a large amount of guess work and trial/error testing. The other two had solid reference material. Overall, this lab taught how APIs are similar, and how to write code to utilize them. It was successful on both accounts, and will be my dominant form of large data retrieval in the future.

References

Gette, C. (2018, May 18). Two (and a half) ways to get weather Data: Part 2- using an API and NOAA. Retrieved February 07, 2021, from https://gettecr.github.io/noaa-api.html

Google Maps Platform. (2020). Place search | places api | google developers. Retrieved February 07, 2021, from https://developers.google.com/places/web-service/search?hl=en_US

Morales, F. (2020, May 20). Google maps: Places api. Retrieved February 07, 2021, from https://medium.com/swlh/google-maps-places-api-28b8fdf28082

Open Knowledge Foundation. (2012). The ckan api¶. Retrieved February 07, 2021, from https://docs.ckan.org/en/ckan-2.1.5/api.html

Prewitt, N., & Larson, S. M. (2020). Requests: Http for humans™¶. Retrieved February 07, 2021, from https://requests.readthedocs.io/en/master/

Ronquillo, A. (2020, November 07). Python's requests Library (guide). Retrieved February 07, 2021, from https://realpython.com/python-requests/#the-get-request

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	**26**
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	**22**
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 verification is clearly stated **(5 points)**.	20	**19**
		100	**93**