Lab Report

Title: Lab 1
Notice: Dr. Bryan Runck
Author: Emily Cavazos
Date: Oct 6, 2021

Project Repository: https://github.com/cavaz020/GIS5571/
Google Drive Link: N/A
Time Spent: 20 hours

Abstract

The purpose of this report is to compare and contrast conceptual models for the Minnesota Geospatial Commons API, the Google Places API and the NDAWN API. This report also details the process of using arcpy to transform datasets using two datasets downloaded from the MNGEO API as an example. For the transformation of two datasets I used two shapefiles out of the three shapefiles of the different levels of the Ecological Classification System (ECS). Specifically, I used the provinces shapefile and the subsections shapefile. For methods I used various modules imported into jupyter notebooks in arcgis pro and wrote code to perform the requests from the APIs and to perform the transformations on the MNGEO data. The steps I took to do so are thoroughly detailed below with accompanying screenshots and data flow charts. For results, I compared and contrasted all three APIs based on various categories. Within the discussion, I detailed my experience working with the three APIs and why I used the modules that I did. My sources are cited and I reported a self score at the end of the report.

Problem Statement

Compare and contrast the conceptual models for the following API's - Minnesota Geospatial Commons, Google Places, and NDAWN. Create Jupyter notebooks that can programmatically get data from each of these APIs. Using Jupyter notebooks, build a pipeline that downloads two data sets, transform both datasets to the same coordinate reference system (geographic and projected), spatially joins them, prints to screen the head of the table showing the merged attributes, and saves the integrated dataset to a geodatabase.

Table 1. APIs to be compared in this lab

#	Requirement	Defined As	(Spatial) Data	Attribute Data	Dataset	Preparation
1	MN Geospatial Commons API	CKAN - has a built-in REST-ful API	Produces data in shapefile format	N/A	N/A	N/A
2	Google Places API	Places API is a service that returns information about places using HTTP requests	Produces data in either JSON or XML formats	N/A	N/A	Enable API within Google Cloud Account and generate unique API Key.
3	NDAWN API	No documentation found	Produces data in CSV format	N/A	N/A	N/A

Input Data

This data is actually an output after sending a request to the MNGEO API - CKAN. This data is used for the transformations (adding to a geodatabase, projecting to a new coordinate system, performing a spatial join and adding the resulting shapefile to a geodatabase).

Table 2. Data used in transformations using ArcPy

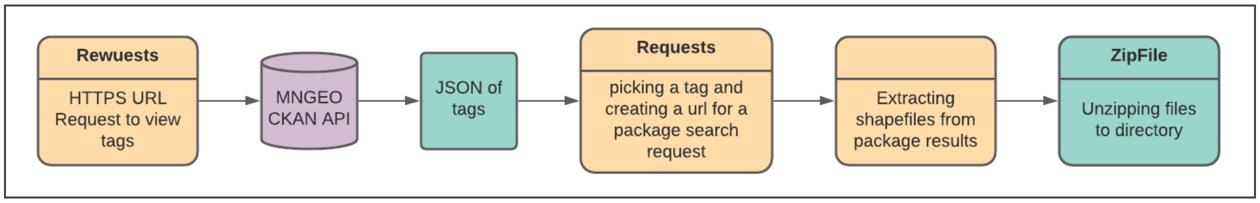
#	Title	Purpose in Analysis	Link to Source
1	MNGEO Provinces, Sections and Subsections (3 shapefiles)	Output dataset for transformation from MNGEO via requests to the CKAN API used for transformations using ArcPy (projection, spatial join, and adding to a geodatabase)	Shapefiles of the three levels of the Ecological Classification System (ECS): Provinces, Sections, and Subsections

Method:
I learned that APIs take requests in an HTTP URL format. This URL is specific to each API and requires you to build it using a base URL, a specific request, and various parameters or inputs. To use APIs through the python notebook, you must first import 'requests'. Most of the methods are documented within my notebooks within the comments as well. This section with its figures, images, and text should be used in conjunction with the three notebooks.

MN Geospatial Commons

Requesting Data

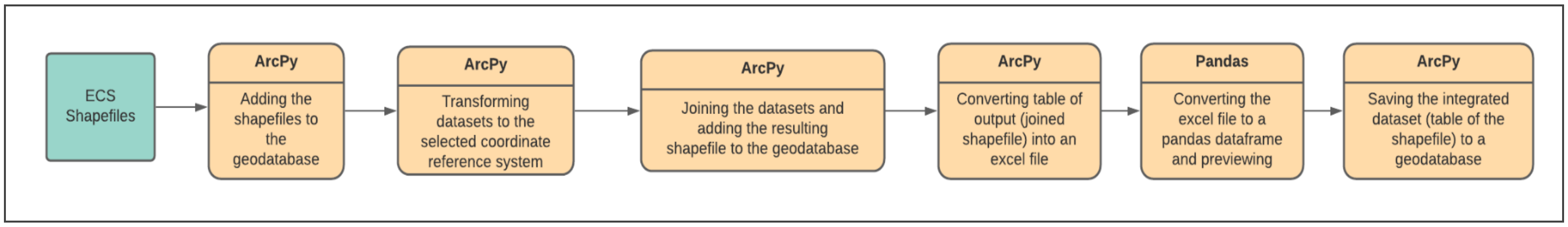
Figure 1. Data flow diagram of Requests to MNGEO API


Rewuests
HTTPS URL Request to view tags
MNGEO CKAN API
JSON of tags
Requests
picking a tag and creating a url for a package search request
Extracting shapefiles from package results
ZipFile
Unzipping files to directory

When using the MNGEO API, CKAN, you can send requests to get JSON-formatted lists of a site's datasets, groups or other CKAN objects. This information can be looked through to send another request to actually download more specified datasets. As you can see within my code in my notebook file, we first sent a request to get a JSON-formatted list of the tags within CKAN and then we chose the tag 'ecological'. This tag was then appended to the base URL to build out the request URL.

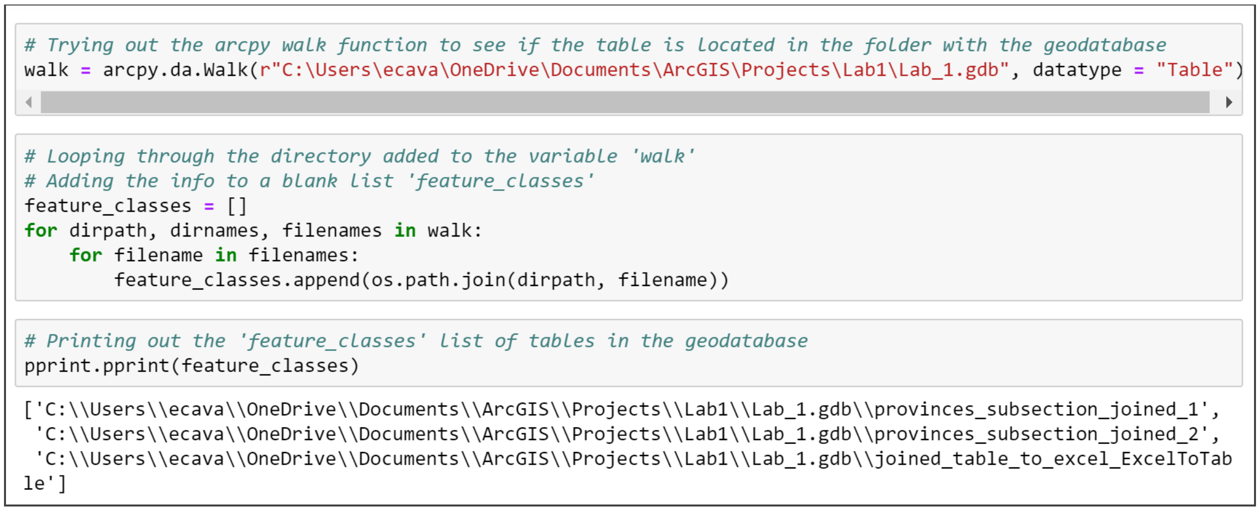
Transforming the two datasets

Figure 2. Data flow diagram of transformations to 2 datasets


ECS Shapefiles
ArcPy
Adding the shapefiles to the geodatabase
ArcPy
Transforming datasets to the selected coordinate reference system
ArcPy
Joining the datasets and adding the resulting shapefile to the geodatabase
ArcPy
Converting table of output (joined shapefile) into an excel file
Pandas
Converting the excel file to a pandas dataframe and previewing
ArcPy
Saving the integrated dataset (table of the shapefile) to a geodatabase

After we downloaded the datasets, we explored using ArcPy to perform various transformations. Specifically, we used the 'project' tool to project the shapefiles to the same coordinate reference system, performed a spatial join on two of the shapefiles, and then saved the dataset to the geodatabase. I used pandas to preview the output table. I did not look into how to preview the table using arcpy but I used pandas because I have previous experience with pandas and knew it is easy to convert shapefiles into dataframes. Also, at the end, I decided to try out the ArcPy.da.Walk function to catalog the data and ensure that the tables were added to the geodatabase. The results of this code are pictured below.

Image 1. Code using the 'Walk' function in Arcpy to check if the tables were added to the geodatabase



```
# Trying out the arcpy walk function to see if the table is located in the folder with the geodatabase
walk = arcpy.da.Walk(r"C:\Users\ecava\OneDrive\Documents\ArcGIS\Projects\Lab1\Lab_1.gdb", datatype = "Table")
```

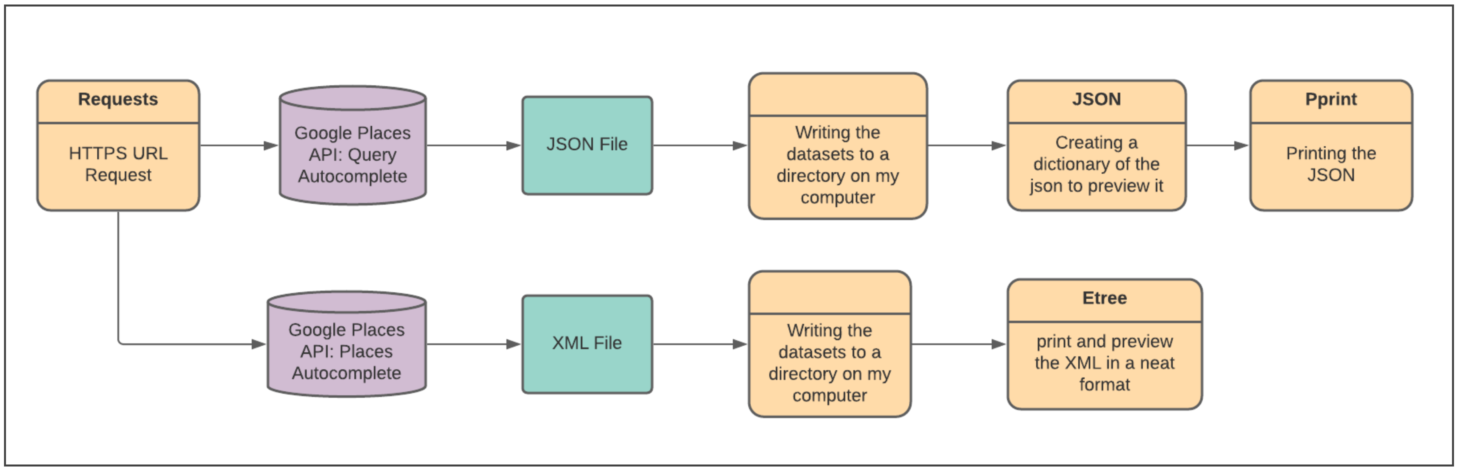
```
# Looping through the directory added to the variable 'walk'
# Adding the info to a blank list 'feature_classes'
feature_classes = []
for dirpath, dirnames, filenames in walk:
    for filename in filenames:
        feature_classes.append(os.path.join(dirpath, filename))
```

```
# Printing out the 'feature_classes' list of tables in the geodatabase
pprint.pprint(feature_classes)
```

```
['C:\\Users\\ecava\\OneDrive\\Documents\\ArcGIS\\Projects\\Lab1\\Lab_1.gdb\\provinces_subsection_joined_1',
 'C:\\Users\\ecava\\OneDrive\\Documents\\ArcGIS\\Projects\\Lab1\\Lab_1.gdb\\provinces_subsection_joined_2',
 'C:\\Users\\ecava\\OneDrive\\Documents\\ArcGIS\\Projects\\Lab1\\Lab_1.gdb\\joined_table_to_excel_ExcelToTab
le']
```

Google Places

Figure 3. Data flow diagram of Requests to Google Places API

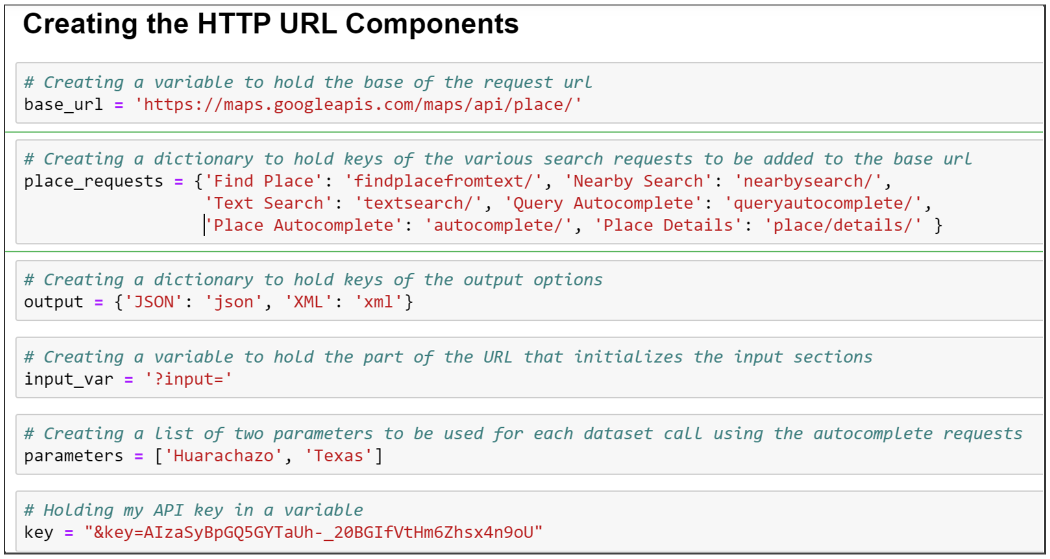

Requests
HTTPS URL Request
Google Places API: Query Autocomplete
JSON File
Writing the datasets to a directory on my computer
JSON
Creating a dictionary of the json to preview it
Pprint
Printing the JSON
Google Places API: Places Autocomplete
XML File
Writing the datasets to a directory on my computer
Etree
print and preview the XML in a neat format

Google Places API requires that you enable the API to your google account and generate a unique, personal API key that will be appended to the HTTP URL to run requests. There are various requests that may be used and each can be called using a unique HTTP URL:

- Place Search returns a list of places based on a user's location or search string.
- Place Details returns more detailed information about a specific place, including user reviews.
- Place Photos provides access to the millions of place-related photos stored in Google's Place database.
- Place Autocomplete automatically fills in the name and/or address of a place as users type.
- Query Autocomplete provides a query prediction service for text-based geographic searches, returning suggested queries as users type.

I downloaded two datasets - one using the query autocomplete request and one using the place autocomplete request. I downloaded one JSON and one XML. I preview the downloaded datasets using the modules json, pprint, and etree from lxml.

Image 2. Creating the URL by assigning all of the necessary components to variables



Creating the HTTP URL Components

```
# Creating a variable to hold the base of the request url
base_url = 'https://maps.googleapis.com/maps/api/place/'
```

```
# Creating a dictionary to hold keys of the various search requests to be added to the base url
place_requests = {'Find Place': 'findplacefromtext/', 'Nearby Search': 'nearbysearch/',
                  'Text Search': 'textsearch/', 'Query Autocomplete': 'queryautocomplete/',
                  'Place Autocomplete': 'autocomplete/', 'Place Details': 'place/details/' }
```

```
# Creating a dictionary to hold keys of the output options
output = {'JSON': 'json', 'XML': 'xml'}
```

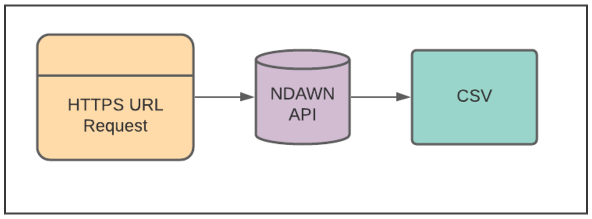
```
# Creating a variable to hold the part of the URL that initializes the input sections
input_var = '?input='
```

```
# Creating a list of two parameters to be used for each dataset call using the autocomplete requests
parameters = ['Huarachazo', 'Texas']
```

```
# Holding my API key in a variable
key = "&key=AIzaSyBpGQ5GYTaUh-_20BGIfVtHm6Zhsx4n9oU"
```

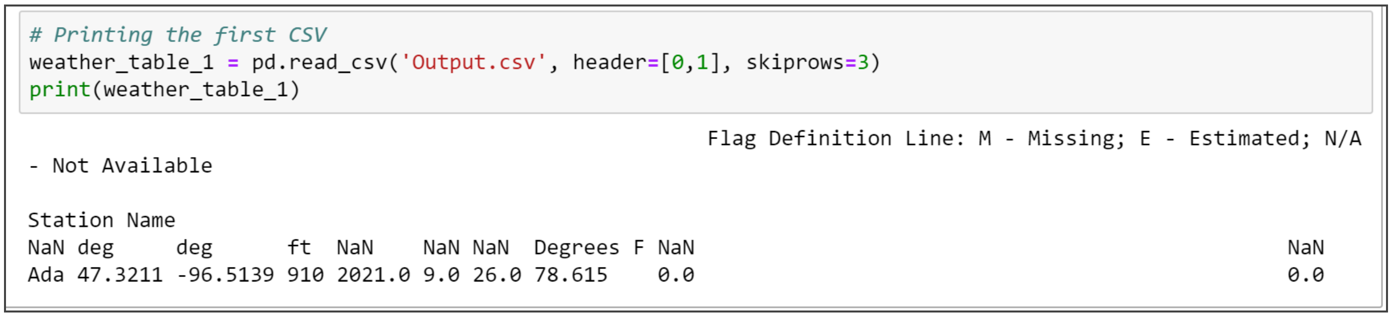
NDAWN

Figure 4. Data flow diagram of Requests to NDAWN API



Using the NDAWN API was confusing because there is no documentation available on the NDAWN website. I found that the requests take in parameters as well as variables to describe the data that you want to download. The parameters for a dataset can include the following: station (station number), begin date (yyyy-mm-dd), end date (yyyy-mm-dd), 'ttype' (meaning the type of data - daily, weekly, monthly, etc). There are also variables that can be used - 'wdmxt' = max temp; 'wdmnt'= min temp; 'wdavt' = average temp; 'wdbst' = bare soil temp;
##'wdtst' = turf soil temp; 'wdws' = average wind speed; 'wdmxws' = max wind speed; 'wdsr' = solar radiation;
##'wdr' = rain fall; 'wddp'= dew point; 'wdwc' = wind chill. The w in front of each variable should stand for 'weekly' and can be changed according to what amount of time your data measures such as d for daily. Once I wrote the CSVs to the directory on my computer, I used pandas to read the CSVs and preview them within my notebook. One example is pictured below.

Image 3. Using Pandas to view the CSV



```
# Printing the first CSV
weather_table_1 = pd.read_csv('Output.csv', header=[0,1], skiprows=3)
print(weather_table_1)
```

```
                                                   Flag Definition Line: M - Missing; E - Estimated; N/A
- Not Available

Station Name
NaN deg     deg      ft  NaN     NaN NaN  Degrees F NaN                                                NaN
Ada 47.3211 -96.5139 910 2021.0 9.0 26.0 78.615    0.0                                                0.0
```

Results
Show the results in figures and maps. Describe how they address the problem statement.
The table below shows my thoughts on the differences between the three tools in performing the same function.

Table 1. Comparison between the three APIs

Site using API	**Type of data on Site**	**Spatial Data Returned**	**Data Type returned**	**Specificity Required in HTTP URL?**	**API Key needed?**	**Documentation?**
MN Geospatial Commons	Geographic data of many types (ecology, weather, etc)	Yes	Shapefile	Yes, need to choose a tag and package from lists	Sometimes	Yes (CKAN Documentation)
Google Places	All data on Google	Yes	JSON or XML	Can use autocomplete and just provide a key word of interest	Yes	Yes (Places API Documentation)
NDAWN	Weather and agricultural data within North Dakota and some of Minnesota	Only City	CSV	Yes - need to provide site number, date, length of time for data collection etc. and hard because not much documentation of these parameters	No	Not that I could find

Results Verification
This assignment did not require much output but was more about the experience of working with three APIs so the results verification is more so just speaking on my experiences and findings of working with these APIs.

Discussion and Conclusion

Comparing 3 Systems

The three systems were all very new for me to use. I had never retrieved data from an API before this assignment so everything was new and nothing was too easy. I thought that the Google Places API was the easiest to use because you can just use one of the autocomplete requests to search for a dataset which only requires a few parameters including the API key. I thought the MNGEO API was difficult because there is so much data on that website and it is hard to sift through it all using API requests. However, I think that the output dataset format of a shapefile is the easiest and most convenient to use within ArcGIS Pro or for visualization purposes. I was not sure if there were other formats that the data could be exported as because I am new to using this API but I am interested to know if there are other formats. NDAWN was the worst API to use because there was no documentation that I could find on the internet. Also, the data format, a CSV, that it outputs is not as useful as a JSON or Shapefile which has plenty of spatial data which can be utilized by ArcGIS pro. Overall, I think all of these APIs are useful for different purposes. NDAWN for example is very specialized data so it might be useful if it matches the purpose of your research. I feel like using the API is almost not worth it because you can easily download the CSVs from their site in a similar fashion. Perhaps it would be useful to use the API if you're getting many datasets from pre-specified sites and times.

The MNGEO API and Google Places API are both well documented, and have a wealth of data but are much more complicated to use.

References

1. "Convert CSV to JSON with Python. I Got Help from a Youtube Tutorial… | by Hannah | Medium." Accessed October 6, 2021. https://medium.com/@hannah15198/convert-csv-to-json-with-python-b8899c722f6d.
2. "Get Started—ArcGIS REST APIs | ArcGIS Developers." Accessed October 6, 2021. https://developers.arcgis.com/rest/services-reference/enterprise/get-started-with-the-services-directory.htm.
3. Gette, Cody. Get Weather Data. Python, 2018. https://github.com/gettecr/get_us_weather/blob/ebf1aeed77a4530b932909c3e844ef34b302860e/get_weather_ndawn.py.
4. Earth Data Science - Earth Lab. "GIS in Python: Intro to Coordinate Reference Systems in Python," February 5, 2018. https://www.earthdatascience.org/courses/use-data-open-source-python/intro-vector-data-python/spatial-data-vector-shapefiles/intro-to-coordinate-reference-systems-python/.
5. "GIS in Python: Intro to Coordinate Reference Systems in Python | Earth Data Science - Earth Lab." Accessed October 6, 2021. https://www.earthdatascience.org/courses/use-data-open-source-python/intro-vector-data-python/spatial-data-vector-shapefiles/intro-to-coordinate-reference-systems-python/.
6. GeeksforGeeks. "Json.Loads() in Python," June 16, 2020. https://www.geeksforgeeks.org/json-loads-in-python/.
7. "Pretty Printing XML in Python - Stack Overflow." Accessed October 6, 2021. https://stackoverflow.com/questions/749796/pretty-printing-xml-in-python.
8. Rees, Eric van. "Tutorial: Creating a Pandas DataFrame from a Shapefile." Accessed October 6, 2021. https://geospatialtraining.com/tutorial-creating-a-pandas-dataframe-from-a-shapefile/.
9. "Table To DBASE (Conversion)—ArcGIS Pro | Documentation." Accessed October 6, 2021. https://pro.arcgis.com/en/pro-app/latest/tool-reference/conversion/table-to-dbase.htm.
10. "Table To Geodatabase (Conversion)—ArcGIS Pro | Documentation." Accessed October 6, 2021. https://pro.arcgis.com/en/pro-app/latest/tool-reference/conversion/table-to-geodatabase.htm.
11. "The CKAN API — CKAN Documentation 2.1.5 Documentation." Accessed October 6, 2021. https://docs.ckan.org/en/ckan-2.1.5/api.html.
12. "Walk—ArcGIS Pro | Documentation." Accessed October 6, 2021. https://pro.arcgis.com/en/pro-app/latest/arcpy/data-access/walk.htm.

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	**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	**20**
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	**20**
		100	**92**