Project Documentation Requirements -

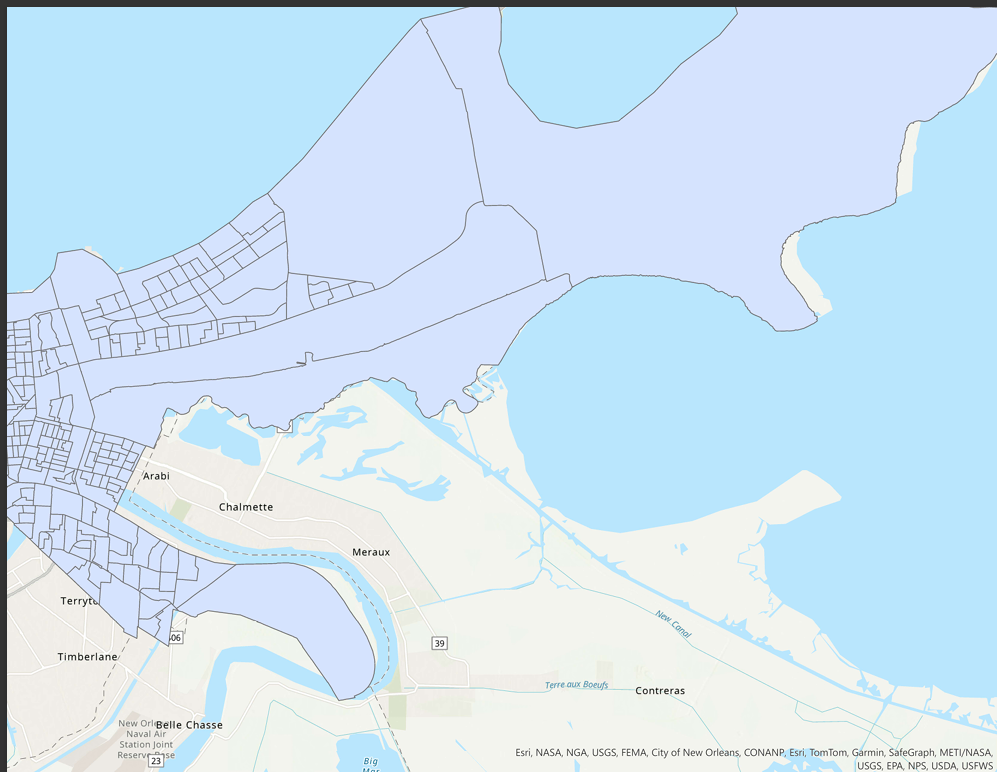
A comprehensive report

The primary objective of this project is to pull, convert, import, and turn into a map the data from the New Orleans JSON. The data is held with Well-Known Text format and has values hidden within it, like land values, thus in order to do what needs to be done that data must be turned into something that can be read by a map generating program. In this case I will be using Visual Studio Code in order to turn the JSON file into something that can be processed by ArcGIS Pro to turn the data into a map.

The selected JSON file was chosen from Data.gov and describes the land values from 2018 in New Orleans. It contains both metadata, names, spatial data, WKT and geometry. In order to turn the offending JSON file into a usable file and export the appropriate data there are a few processes that we must do. First, of course, we must import the JSON file into the Visual Studio Code then, using available information or packages, code Visual Studio Code to read the presented data. Secondly, we must extract the important and relevant data from the JSON file and turn it into handleable data such as geometric data. The final step is to use the geometric data as a pathway to create a shapefile that can then be used to generate the map. As previously described, Visual Studio Code and ArcGIS Pro were the two tools used in pulling, converting, importing, and turning the JSON file into a map. The Visual Studio Code was used to import, extract, and convert the JSON file data into a usable shapefile as well as create a code that allows the shapefile to create a map. The ArcGIS Pro program was used to read the shapefile and the resulting geometric data and turn it into a viewable, exportable map.

The first code created was the exploration of the JSON file, the creation of a feature class and write fields, and the process of adding data to the feature class. The code needed for these steps is as follows. First one must import the JSON file, make sure it properly closes when not in use, and create the ability for your chosen program to read the contents of the file. Then you must extract the meta column data and create a list of fields so that all of the data can be properly listed. Next you must import the arcpy module and convert the Well-Known Text format into geometric data. For my own code I repeated this twice while adding steps such as looping the conversion through the JSON file, converting specifically the 8th index of each row into geometry and looping it through the JSON file. Then I imported an os module, defined a feature class name, set up a workplace directory, and combined the feature class name with the workspace. Additionally, I made it so that it checks if the created feature class already exists and if it does to delete it. Following that I strung together another feature class creation, a save path, a new name for the feature class, the specific geometric data type I wanted (polygon), and set the spatial reference to 4236. Following that I once again described the feature class metadata, looped it through the JSON file, and printed said names. Then I retrieved the JSON column and defined a list of field types to add to the feature class, and returned the length of the list. I then basically repeated this code just to make sure it stuck. Next I filtered out any field named the\_geom, iterated through the list, printed the list soon repeating the basics of the code to look it over the fields and rename id to id\_{index}, reduced all names to a maximum of 10 characters, and replaced all spaces and periods with underscores. Following this I printed the field names then looped over the field names list in order to add them to each feature class. Then I appended SHAPE@ to the list of names as it holds geometric information as well as printing the field names, adding them to the feature class. Finally, I created an insert cursor that then adds data to the feature class, creates an inner loop to process the data of the JSON file and appends the value and geometry to the new row, and prints the length of the new row for each row. The second code I created extracts the JSON file data and converts it into a shapefile. The first step of this code is to import arcpy module, JSON module, and the os module. Following this I defined a function to take three parameters, a new workspace, JSON file name, and output name for the shapefile. Then I opened the Json file, newly named, and loaded it into the original JSON variable. Following this I created a code to process the data by iterating through each data entry and converting all value index 8s into ArcGIS geometric data. Then I created the shapefile by assigning out\_fc parameters, joining the workspace with the full path, and checking to see if the shapefile already exists then deleting it if it does. Following that I strung together another feature class creation, a save path, a new name for the feature class, the specific geometric data type I wanted (polygon), and set the spatial reference to 4236. Then I retrieved the shapefile descriptions, iterating over the fields, and printing the names of each field. Following this I added the fields from the previously mentioned JSON variable by retrieving the column definitions, iterating over each column, to then print it. Then I set up the field names and type processes. I did this by defining the list of field types within the shape file, initializing an empty list to store the field names, iterating through the fields list, retrieving the name of the field, skipping the process for field the\_geom, renaming id to avoid conflicts, limiting the characters, trimming the name length, and adding the field names to the list. Next, I made sure that spaces and periods are replaced with underscores. Following this I iterated through the list of field names, added each field to the shapefile, and appended SHAPE@ to the field names. Immediately after, I created an insert cursor that then added rows to the shapefile that iterates through each row, initialized an empty list for new rows and iterated through each value in the current row, skip value at index 8, replace none values with empty strings, add processed value and geometry to the rows, and inserted the new rows into the shapefile. Then I defined the main function, imported the sys module, retrieved the first command-line from the output shapefile, and called a function that then passed the output shapefile name. Finally, I executed the following code by checking that the script runs directly and calls upon the main function to execute the script. The final bit of coding I did was the creation of parameter information, describing the name, display name, direction, parameter type, and data type of workspace, JSON, and output. Workspace and JSON were inputs while output was the output. The final bit of coding being the end of this section creates a core function for the previously decided parameters that when executed creates a shapefile using the selected JSON within the specified workspace.

The results of the previously explained code are a map of the sections of the land values within New Orleans in 2018. It shows a clean and exact colored map highlighting the previously encoded metadata and geometric data from the JSON file downloaded from data.org.



ArcGIS Pro and Visual Studio Code and many other coding and gis programs have the ability to do many things though they are limited, to an extent, to the types of files they support. Some can access Well-Known Text while others cannot. Some can read JSON files while others cannot. Each different program has its strengths and weaknesses though not all programs are interchangeable or co-usage. This lack of usage across all programs creates a disturbing lack of multiple functions and usage. Though mainly ultimately fixable it is a rather lengthy and undeterminable process. Creating modules or programs that can automatically formulate, extract, read, ect. what is needed within the unmatching file would be a huge time saver. That or altering the programs to be more readable/extractable for different programs and file types would be, most likely complicated but ultimately be easier and faster.

# References

“Data.Gov - Data.Gov Dataset.” Catalog. Accessed December 9, 2024. https://catalog.data.gov/dataset/.

Wang, Lei. “Project1.Mkv: Powered by Box.” Box. Accessed December 9, 2024. https://lsu.app.box.com/s/6gpbv9juj9v69to9pqd5u38afzknoeam.

Wang, Lei. “GEOG4057/Project/Guided Project 1.MD at Main · LEIWANGLSU/GEOG4057.” GitHub. Accessed December 9, 2024. https://github.com/leiwanglsu/geog4057/blob/main/project/guided%20project%201.md.

Wang, Lei. “GEOG4057/Project/Guided Project 1.MD at Main · LEIWANGLSU/GEOG4057.” GitHub. Accessed December 9, 2024. https://github.com/leiwanglsu/geog4057/blob/main/project%20project%201.md.