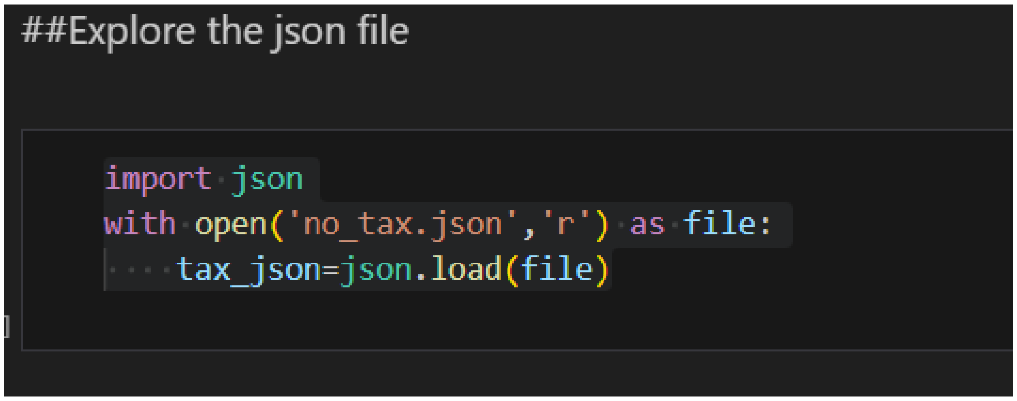
Name: Sudeep Kuikel

Guided Project1: GEOG 4057

In this project, we are using a JSON file developed from data.gov page of the 2018 Land Value for New Orleans. It contains WKT (well-known-text) format storing geometries and attribute informations which could be used in GIS. Since, ArcGIS cannot directly import it using the tools in Toolboxes (such as JSON to Feature).The objective is to provide a script tool to import the JSON file to a shapefile and visualize it in a project.

The snapshots and explanations of the code that we used for this task is explained in detail below:

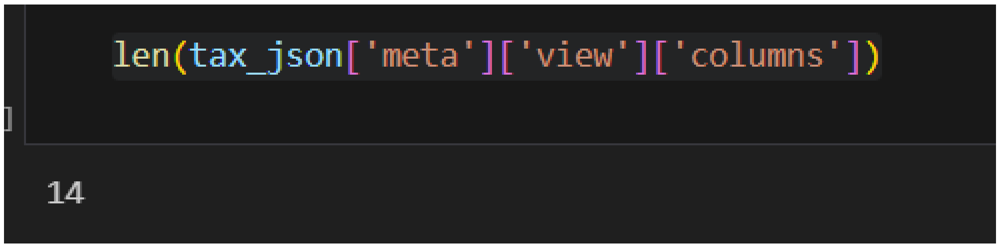
The very First step is to Explore the JSON file. So, below is the demo of how we explored the data.



##Explore the json file

```
import json
with open('no_tax.json','r') as file:
    tax_json=json.load(file)
```

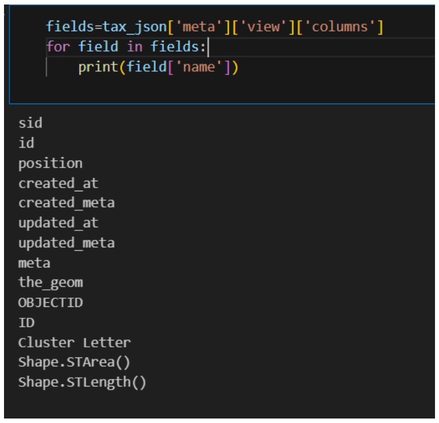
This code imports json and reads the content of no_tax.json file and stores it in a variable named tax_json for further use afterward.



```
len(tax_json['meta']['view']['columns'])
```

```
14
```

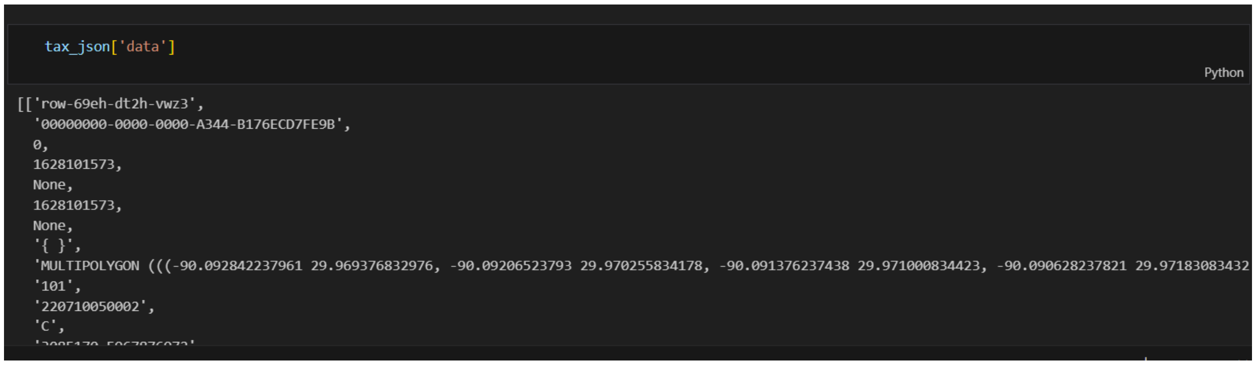
After reading the file and storing it in tax_json we wanted to count the elements within the columns field in the JSON data. If the columns include metadata or descriptions of columns, it said the total column as 14.



```
fields=tax_json['meta']['view']['columns']
for field in fields:
    print(field['name'])
```

```
sid
id
position
created_at
created_meta
updated_at
updated_meta
meta
the_geom
OBJECTID
ID
Cluster Letter
Shape.STArea()
Shape.STLength()
```

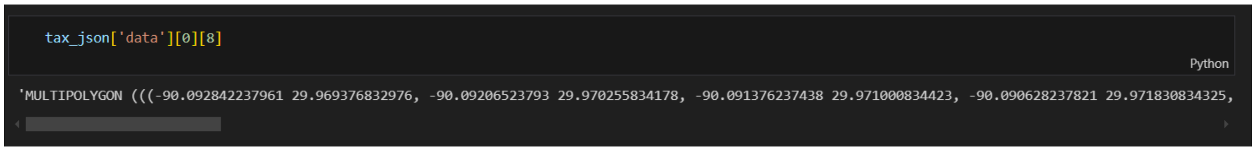
This code assigns the list of columns to the variable fields and then iterates through each element in the fields list. Lastlu, it prints the value of name key from each field dictionary. It means, for every column described in the columns list of the JSON, the code prints its name.



```
tax_json['data']
```

```
[['row-69eh-dt2h-vwz3',
  '00000000-0000-0000-A344-B176ECD7FE9B',
  0,
  1628101573,
  None,
  1628101573,
  None,
  '{ }',
  'MULTIPOLYGON (((-90.092842237961 29.969376832976, -90.09206523793 29.970255834178, -90.091376237438 29.971000834423, -90.090628237821 29.97183083432
  '101',
  '220710050002',
  'C',
```

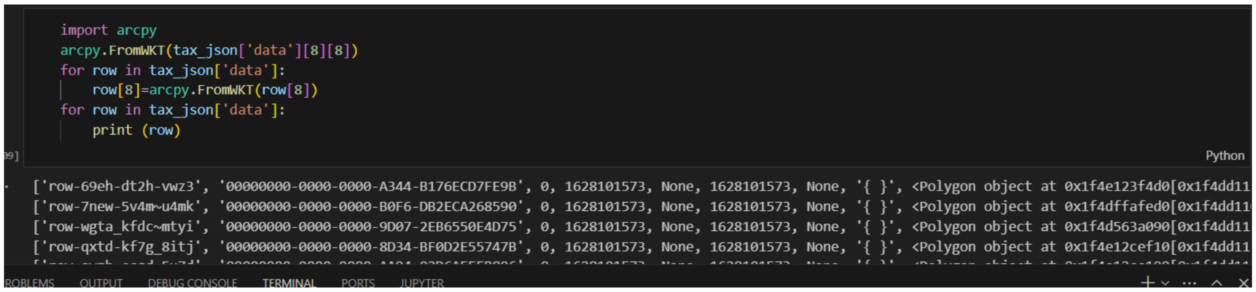
It helped us know the data (rows of information) that corresponds to the column definition.



```
tax_json['data'][0][8]
```

```
'MULTIPOLYGON (((-90.092842237961 29.969376832976, -90.09206523793 29.970255834178, -90.091376237438 29.971000834423, -90.090628237821 29.971830834325,
```

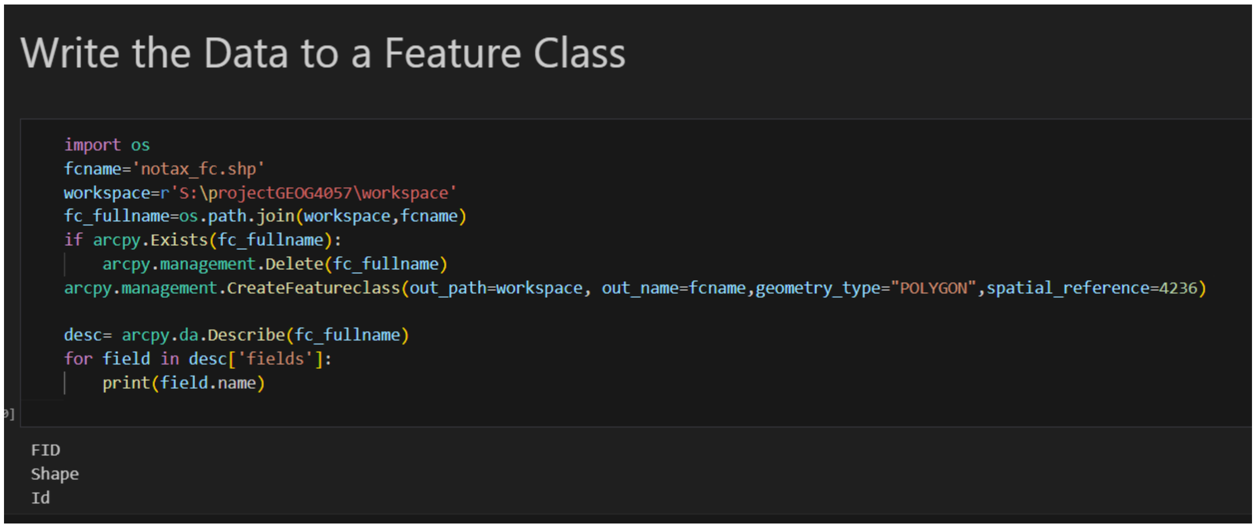
This code retrieved geospatial information from the first row of the dataset and the 9th element from the first row. We know from the output that the specific column corresponding to the multipolygon.



```
import arcpy
arcpy.FromWKT(tax_json['data'][8][8])
for row in tax_json['data']:
    row[8]=arcpy.FromWKT(row[8])
for row in tax_json['data']:
    print (row)
```

```
['row-69eh-dt2h-vwz3', '00000000-0000-0000-A344-B176ECD7FE9B', 0, 1628101573, None, 1628101573, None, '{ }', <Polygon object at 0x1f4e123f4d0[0x1f4dd11
['row-7new-5v4m~u4mk', '00000000-0000-0000-B0F6-DB2ECA268590', 0, 1628101573, None, 1628101573, None, '{ }', <Polygon object at 0x1f4dffafed0[0x1f4dd11
['row-wgta_kfdc~mtyi', '00000000-0000-0000-9D07-2EB6550E4D75', 0, 1628101573, None, 1628101573, None, '{ }', <Polygon object at 0x1f4d563a090[0x1f4dd11
['row-qxtd-kf7g_8itj', '00000000-0000-0000-8D34-BF0D2E55747B', 0, 1628101573, None, 1628101573, None, '{ }', <Polygon object at 0x1f4e12cef10[0x1f4dd11
```

Here we imported arcpy library and created geometry from Well Known Text (WKT) using arcpy.FromWKT. The code iterates through each row in the data list and converted WKT string in the 9th column of each row into an ArcPy geometry object. The code replaces the original WKT string in row[8] with the corresponding geometry object. Finally, the code prints each row of tax_json['data'].

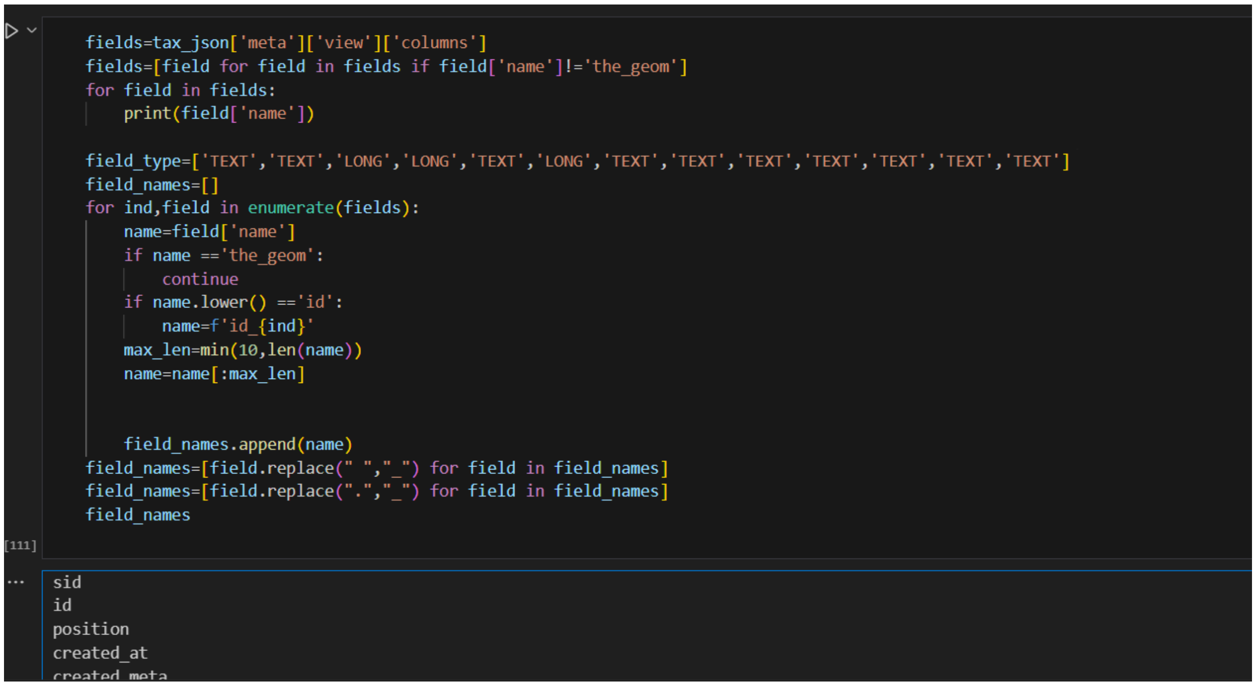


Write the Data to a Feature Class

```
import os
fcname='notax_fc.shp'
workspace=r'S:\projectGEOG4057\workspace'
fc_fullname=os.path.join(workspace,fcname)
if arcpy.Exists(fc_fullname):
    arcpy.management.Delete(fc_fullname)
arcpy.management.CreateFeatureclass(out_path=workspace, out_name=fcname,geometry_type="POLYGON",spatial_reference=4236)

desc= arcpy.da.Describe(fc_fullname)
for field in desc['fields']:
    print(field.name)
```

```
FID
Shape
Id
```

The main task of the project is now to create a shapefile to store geospatial data and describe its field. We imported os in this section to help us manipulate file paths and check the file existence. We assigned the name of the shapefile to be created as notax_fc.shp in our defined workspace, the fc_fullname helps to combine the workspace path and the shapefile name into the full file path. We wanted to try if the file already exists and want to delete if it does. We used arcpy.Exists(fc_fullname) to check if the shapefile already exists at the specified path and arcpy.management.Delete(fc_fullname) to delete the existing shapefile to ensure a clean slate. Then we created a shapefile using arcpy.management.CreateFeatureclass, specified the directory where the file will be created by setting the out_path=workspace. We defined the geometry type of the shapefile as polygons and set the spatial reference as EPSG 4236 which refers to WGS 84. In the last part we tried to describe the shapefile, we used arcpy.da.Describe which returns a dictionary containing metadata about th nwly created shapefile including its fields, geoetry type and spatial reference and then print the field names.

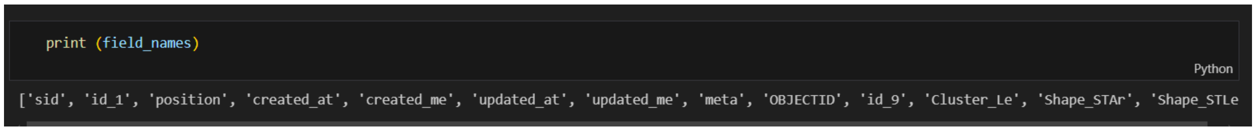


```
fields=tax_json['meta']['view']['columns']
fields=[field for field in fields if field['name']!='the_geom']
for field in fields:
    print(field['name'])

field_type=['TEXT','TEXT','LONG','LONG','TEXT','LONG','TEXT','TEXT','TEXT','TEXT','TEXT','TEXT','TEXT']
field_names=[]
for ind,field in enumerate(fields):
    name=field['name']
    if name =='the_geom':
        continue
    if name.lower() =='id':
        name=f'id_{ind}'
    max_len=min(10,len(name))
    name=name[:max_len]


    field_names.append(name)
field_names=[field.replace(" ","_") for field in field_names]
field_names=[field.replace(".","_") for field in field_names]
field_names
```

[111]

```
sid
id
position
created_at
created_meta
```

The main aim of this part of the code is to process fields from a JSON strcrure, filter them and prepare them for using in a shapefile. The first two lines filters the fields. The field named ‘the_geom’. The for loop loops through the fields which are filtered and prints the name of each field. The next work we did is defined field properties by assigning field_type as TEXT and LONG. An empty list named field_names is created to store processed field names. The next step iterates over the fields and we decided to skip the field named ‘the_geom’, renamed the field ‘id’ with ‘id_1’ to avoid the duplication and limited the field name length at 10 characters then added the perocessed name to the field_name list. Finally, we replaced spaces with underscores to make it GIS compatible and also replaced periods with underscore to avoid invalid character. The final output was a list of processed and saitized field names.

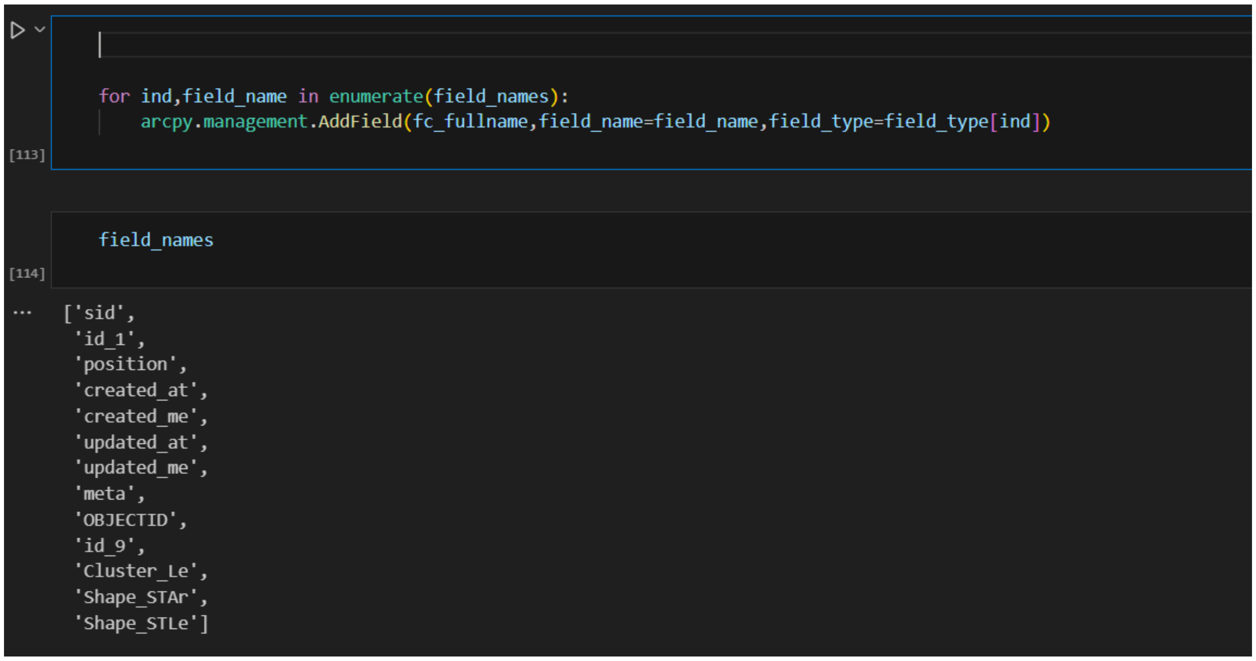


```
print (field_names)
```

Python

```
['sid', 'id_1', 'position', 'created_at', 'created_me', 'updated_at', 'updated_me', 'meta', 'OBJECTID', 'id_9', 'Cluster_Le', 'Shape_STAr', 'Shape_STLe
```

This line was written to print the field_names and be sure with the names of the fields. It shows the name with space and period are replaced as we wanted.

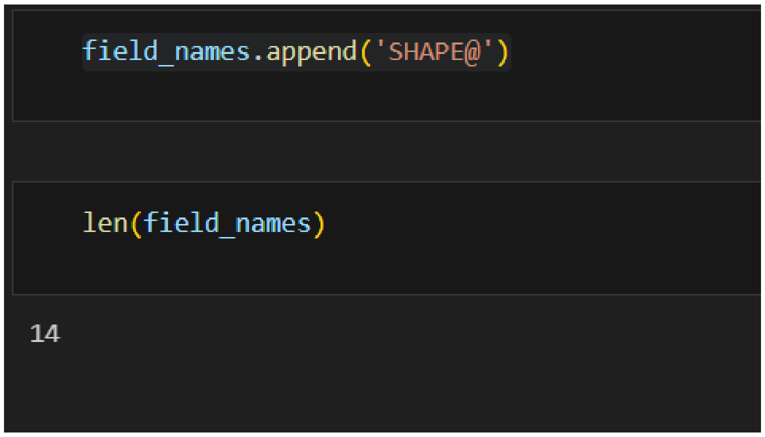


```
[113]
for ind,field_name in enumerate(field_names):
    arcpy.management.AddField(fc_fullname,field_name=field_name,field_type=field_type[ind])
```

```
[114]
field_names
```

```
... ['sid',
 'id_1',
 'position',
 'created_at',
 'created_me',
 'updated_at',
 'updated_me',
 'meta',
 'OBJECTID',
 'id_9',
 'Cluster_Le',
 'Shape_STAr',
 'Shape_STLe']
```

In these codes, the loop added the fields defined in field_names to the shapefile fc_fullname using specified data types in field_type. The objective of these lines is to prepare the shapefile for sturing structured data.

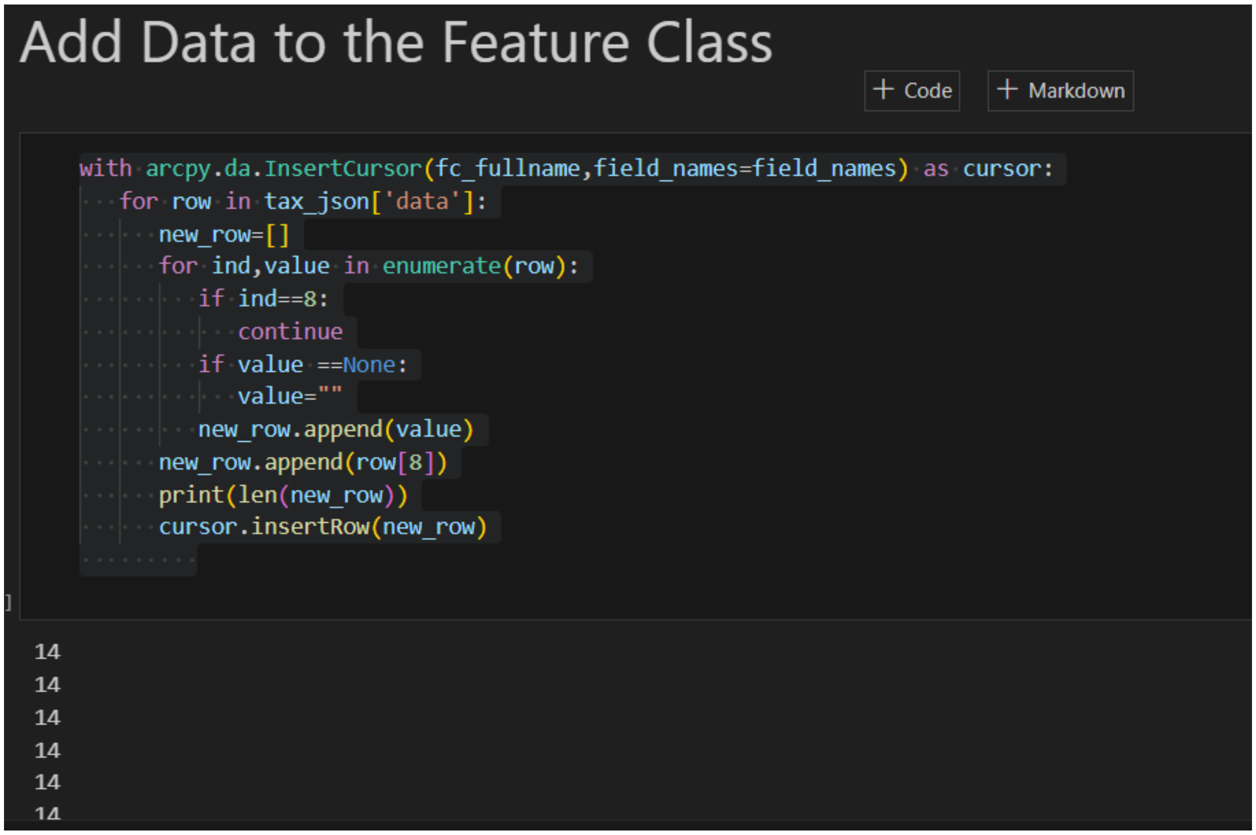


```
field_names.append('SHAPE@')
```

```
len(field_names)
```

```
14
```

We added string ‘SHAPE@’ to the end of the field_names list. Here, SHAPE@ is a special token which is used in ArcPy modules to work with geometry fields. It represents the geometry object (e.g., polygon, point) for each row. In the next part we checked the length of field_names and the output was 14.



Add Data to the Feature Class

```
with arcpy.da.InsertCursor(fc_fullname,field_names=field_names) as cursor:
    for row in tax_json['data']:
        new_row=[]
        for ind,value in enumerate(row):
            if ind==8:
                continue
            if value ==None:
                value=""
            new_row.append(value)
        new_row.append(row[8])
        print(len(new_row))
        cursor.insertRow(new_row)
```

```
14
14
14
14
14
14
```

This is the final part of our programming in notebook where we insert rows into the feature class (fc_fullname) using ArcPy InsertCursor. We used arcpy.da.InsertCursor to insert rows into the feature class. Then we used loop through each row in the data key of the JSON file and initialized an empty list to build a new row which matches the field in the shapefile. Then we populated the new row and added the geometry data to new_row from index 8 of the original row. This part ensured that the geometry is included as the last item corresponding to the SHAPE@ field. Then we checked the length of new row and printed it and got the length as 14 which varified that it matched with the number of fields in field_names. In the last, line we inserted new_row into the feature class using InsertCursor.

Layout of Map prepared as part of Guided Project1 is shown below:

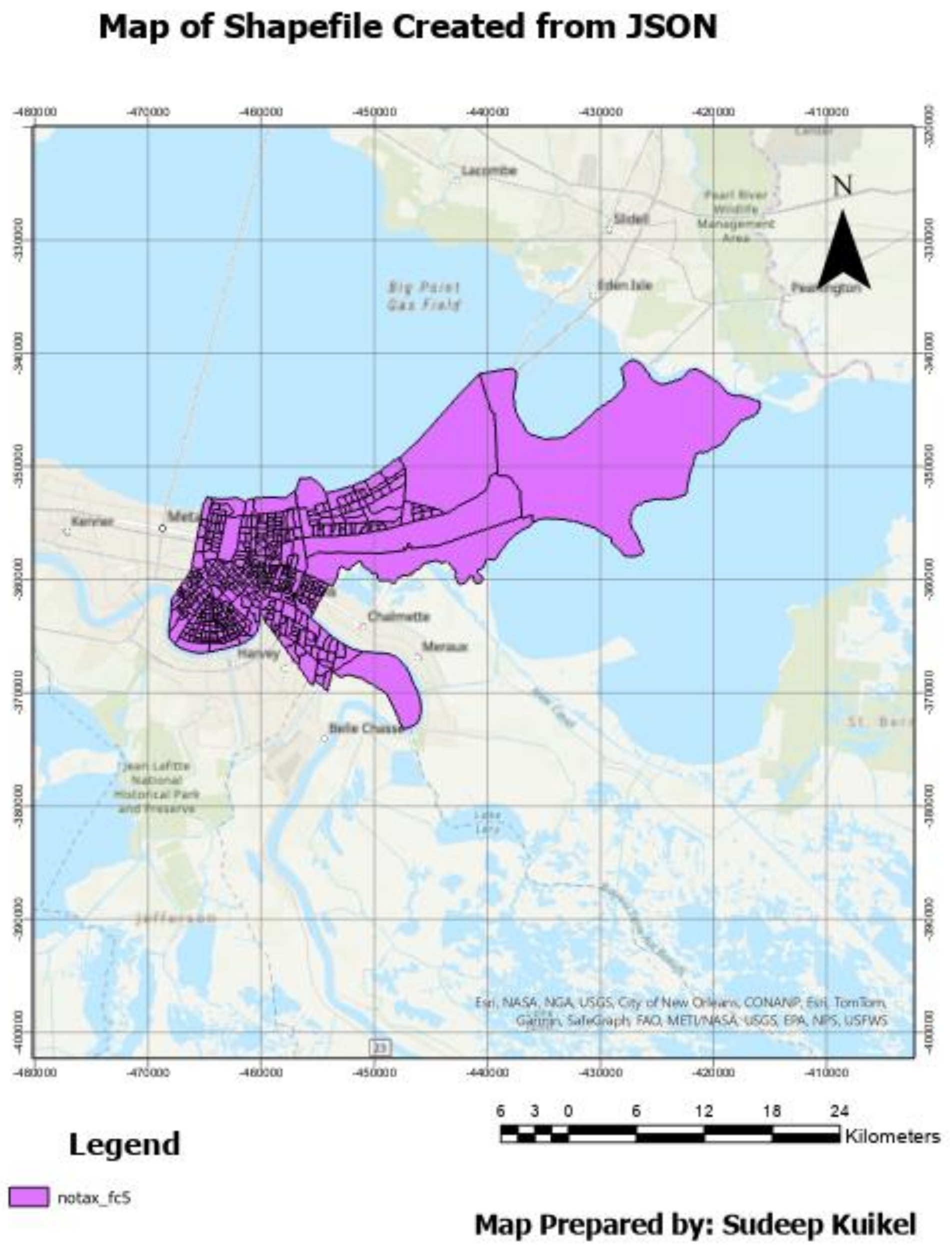

Map of Shapefile Created from JSON
N
Lacombe
Slidell
Pearl River Wildlife Management Area
Big Point Gas Field
Eden Isle
Kenner
Chalmette
Meraux
Harvey
Belle Chasse
Jean Lafitte National Historical Park and Preserve
Jefferson
Esri, NASA, NGA, USGS, City of New Orleans, CONANP, Esri, TomTom, Garmin, SafeGraph, FAO, METI/NASA, USGS, EPA, NPS, USFWS
-480000
-470000
-460000
-450000
-440000
-430000
-420000
-410000
-320000
-330000
-340000
-350000
-360000
-370000
-380000
-390000
-400000
6 3 0 6 12 18 24
Kilometers
Legend
notax_fc5
Map Prepared by: Sudeep Kuikel

Figure 1: Map of shapefile created from JSON file

Some more snapshots of our work can be found in the pictures below:

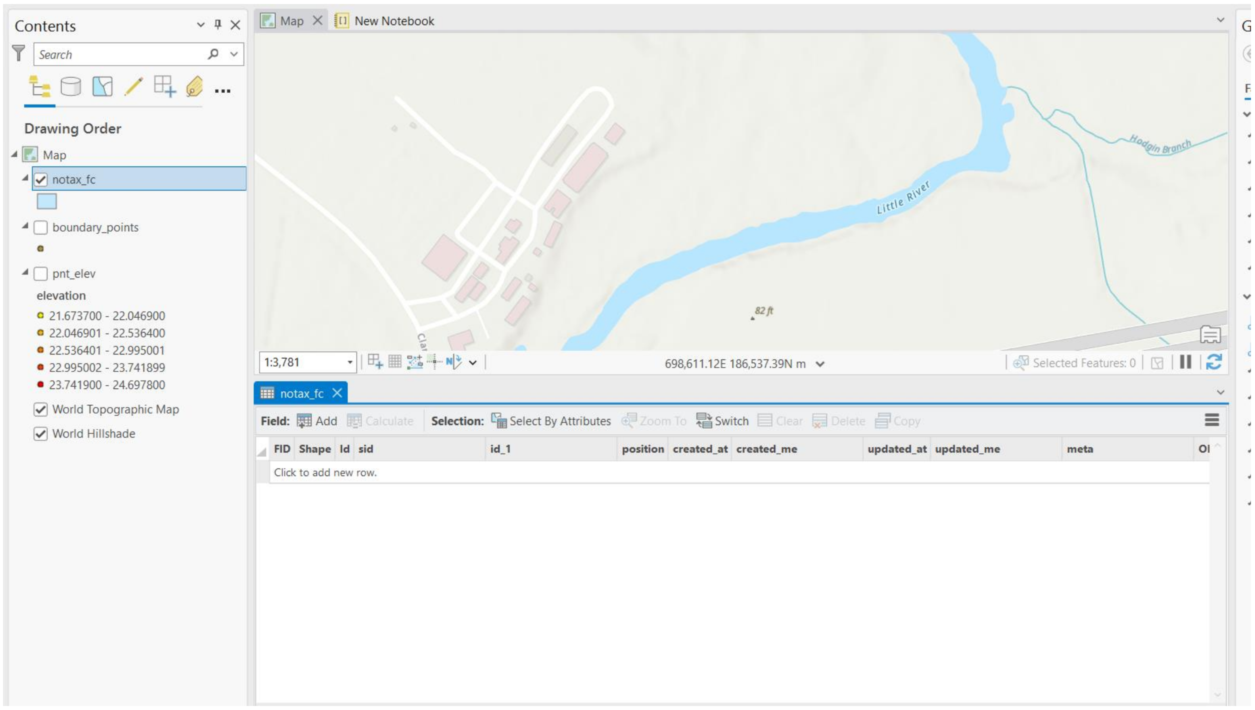


Figure 2:Added Fields in Feature Class

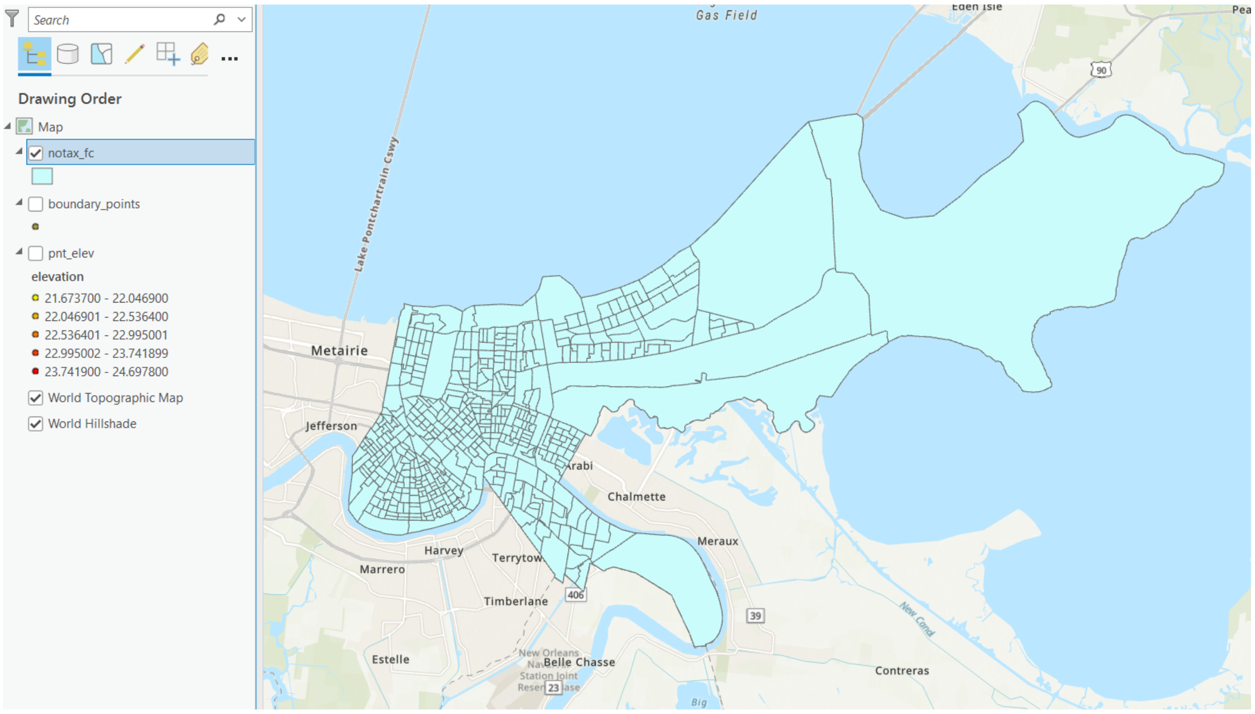
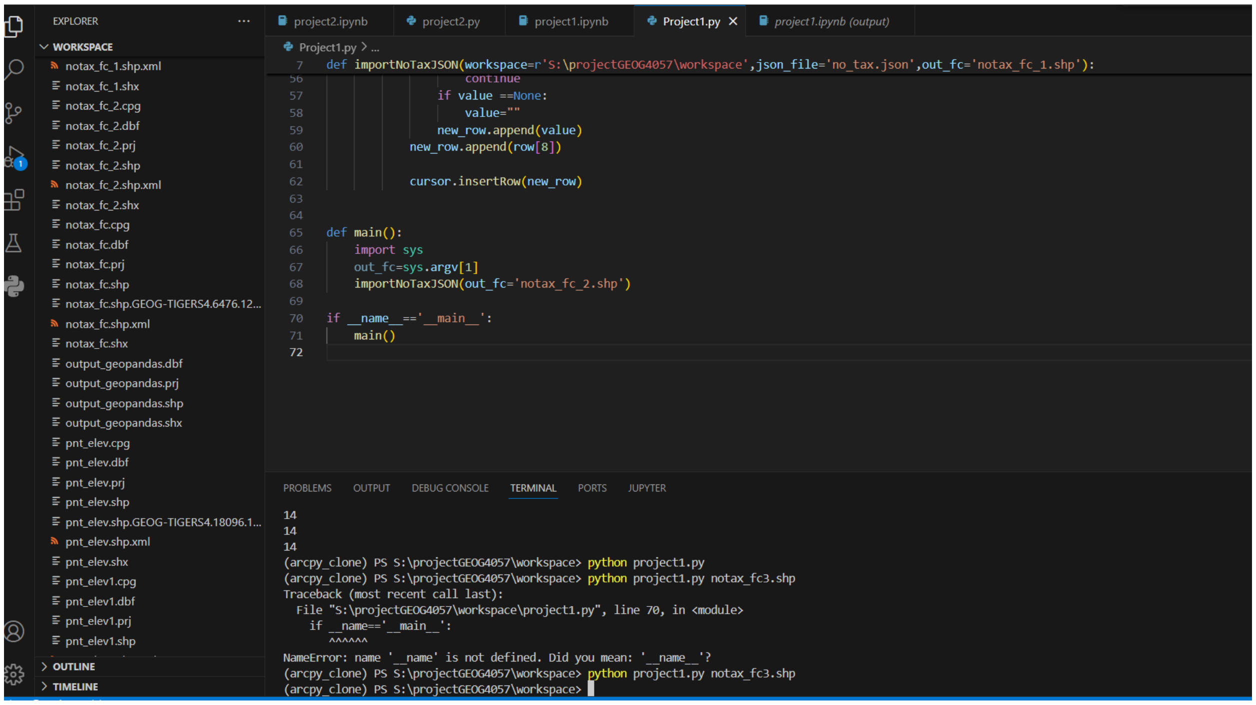


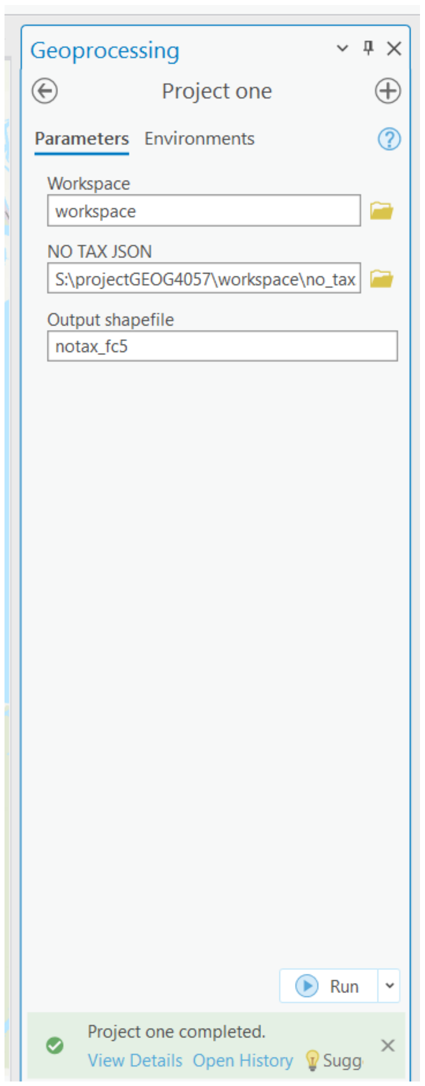
Figure 3: Created Polygons from JSon file



```python
def importNoTaxJSON(workspace=r'S:\projectGEOG4057\workspace',json_file='no_tax.json',out_fc='notax_fc_1.shp'):
                    continue
                if value ==None:
                    value=""
                new_row.append(value)
            new_row.append(row[8])

            cursor.insertRow(new_row)


def main():
    import sys
    out_fc=sys.argv[1]
    importNoTaxJSON(out_fc='notax_fc_2.shp')

if __name__=='__main__':
    main()
```

```
14
14
14
(arcpy_clone) PS S:\projectGEOG4057\workspace> python project1.py
(arcpy_clone) PS S:\projectGEOG4057\workspace> python project1.py notax_fc3.shp
Traceback (most recent call last):
  File "S:\projectGEOG4057\workspace\project1.py", line 70, in <module>
    if __name=='__main__':
       ^^^^^^
NameError: name '__name' is not defined. Did you mean: '__name__'?
(arcpy_clone) PS S:\projectGEOG4057\workspace> python project1.py notax_fc3.shp
(arcpy_clone) PS S:\projectGEOG4057\workspace>
```

Figure 4: Reference for successful execution of code as python file



Workspace

workspace

NO TAX JSON

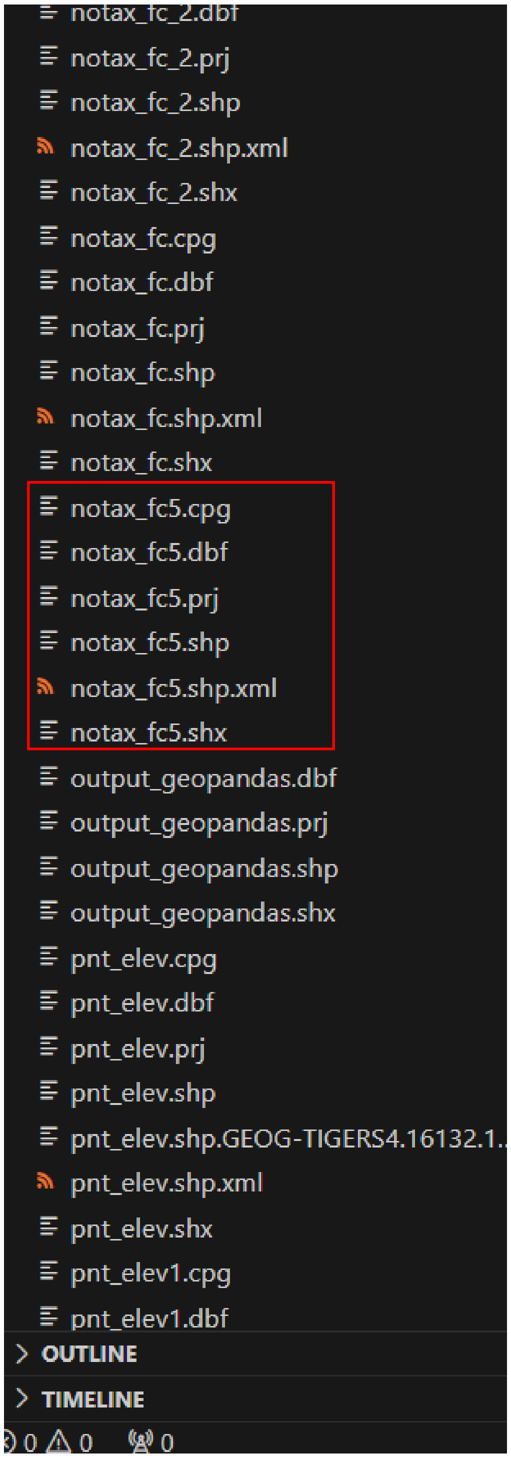
S:\projectGEOG4057\workspace\no_tax

Output shapefile

notax_fc5

Project one completed.

Figure 5: Reference of successfully running the ArcGIS tool



notax_fc_2.dbf
notax_fc_2.prj
notax_fc_2.shp
notax_fc_2.shp.xml
notax_fc_2.shx
notax_fc.cpg
notax_fc.dbf
notax_fc.prj
notax_fc.shp
notax_fc.shp.xml
notax_fc.shx
notax_fc5.cpg
notax_fc5.dbf
notax_fc5.prj
notax_fc5.shp
notax_fc5.shp.xml
notax_fc5.shx
output_geopandas.dbf
output_geopandas.prj
output_geopandas.shp
output_geopandas.shx
pnt_elev.cpg
pnt_elev.dbf
pnt_elev.prj
pnt_elev.shp
pnt_elev.shp.GEOG-TIGERS4.16132.1..
pnt_elev.shp.xml
pnt_elev.shx
pnt_elev1.cpg
pnt_elev1.dbf

Figure 6: The shapefile named ntax_fc5.shp created in the workspace from our tool