Land use Landcover Area Calculation Tool Preparation Using Python Script

Link to Git Hub: https://github.com/MichaelAgyiri/Final_Project_GEOG4057.git

Code Development

Mizbah Ahmed Sresto: Toolbox structure (Toolbox and Tool classes), parameter setup (getParameterInfo), raster projection/clipping, clipping workflow using arcpy.ProjectRaster_management and arcpy.Clip_management, Map Integgration.
Agyiri, Michael Nii Attoh: Area calculations (NumPy/pandas), chart generation using matplotlib and automated its saving., raster-to-polygon/dissolve workflows using (arcpy.RasterToPolygon and arcpy.Dissolve).

Report

Mizbah Ahmed Sresto: Methodology documentation, tool validation, spatial analysis, , technical workflow documentation.
Agyiri, Michael Nii Attoh: Visual outputs (charts/tables), statistical summaries.

Collaborative Contribution: Equal testing, debugging, and finalization.

1. Introduction

1.1 Problem Statement

Manual LULC analysis requires multiple disjointed tools (e.g., clipping rasters, calculating areas in Excel, generating charts), leading to inefficiencies and potential errors. This toolbox automates the entire process, ensuring accuracy and saving time. The custom ArcGIS Python Toolbox (pyt) is designed to simplify the analysis of land use/land cover (LULC) data within a defined geographic area. It processes LULC raster data by clipping it to a user-defined boundary shapefile, calculates the area of each class, and outputs a detailed statistics table alongside a bar chart for visual interpretation. Additionally, it generates a dissolved vector shapefile, merging polygons by class and annotating them with area attributes. The output is automatically added to the current ArcGIS Pro map, ensuring an efficient and seamless workflow. The script utilizes arcpy for GIS operations, numpy and pandas for data processing, and matplotlib for visualizations, offering a comprehensive approach to spatial analysis and effective presentation.

1.2. Toolbox Importance

The toolbox is important for several reasons. It enhances automation by reducing repetitive tasks such as clipping, area calculation, and charting, thereby saving time and effort. It promotes standardization by ensuring a consistent methodology across different users and projects. The toolbox also enables seamless integration by combining raster and vector processing, data analysis, and visualization within a single platform. Its accessibility simplifies complex workflows, making it easier for non-experts such as planners and students to use.

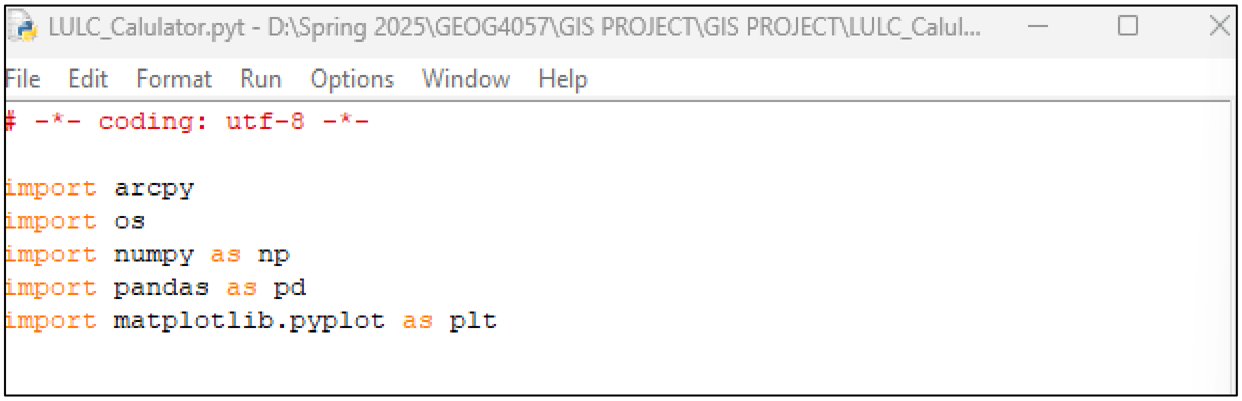
1.3 Goal of the project

To automate LULC analysis by streamlining raster-to-vector conversion, area calculation, and visualization within ArcGIS Pro, enhancing efficiency and accuracy for data-driven decision-making.

2. Methodology

2.1. Import Libraries

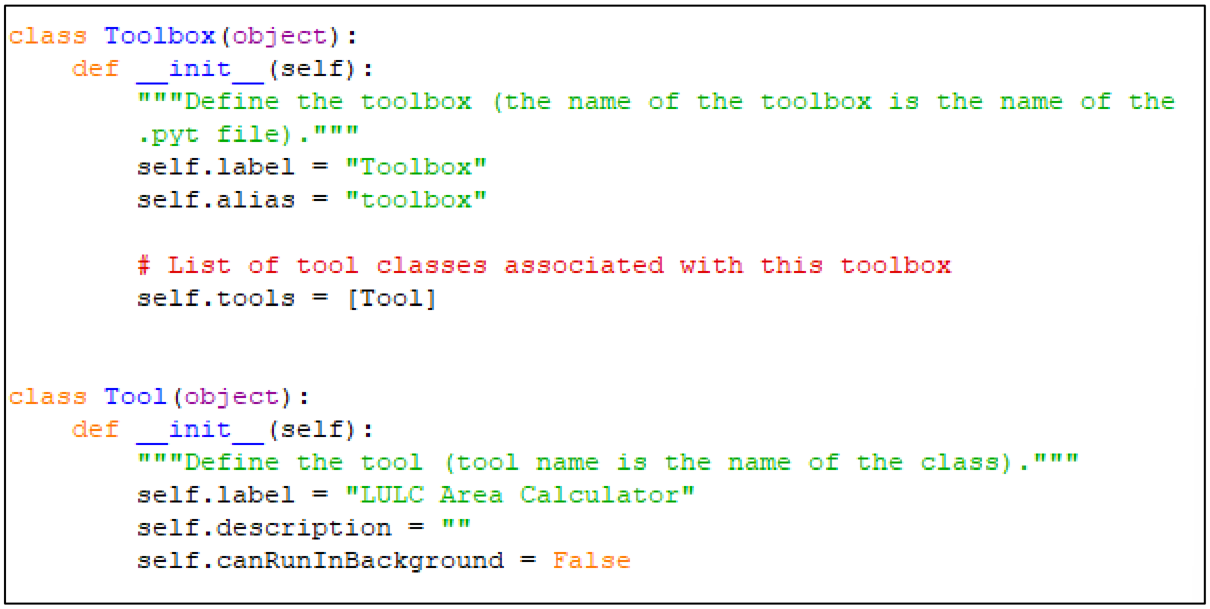
arcpy: For GIS operations (projecting/clipping rasters, converting to polygons).
os: To manage file paths and folders.
numpy and pandas: To analyze raster data and export results to CSV.
matplotlib: To visualize area statistics as a bar chart.


LULC_Calulator.pyt - D:\Spring 2025\GEOG4057\GIS PROJECT\GIS PROJECT\LULC_Calul...
File Edit Format Run Options Window Help

```python
# -*- coding: utf-8 -*-

import arcpy
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

2.2. Toolbox & Tool Setup

The Toolbox and Tool classes define the ArcGIS Pro toolbox structure. The Tool class sets metadata (name, description) and links to the execute method where the workflow runs.

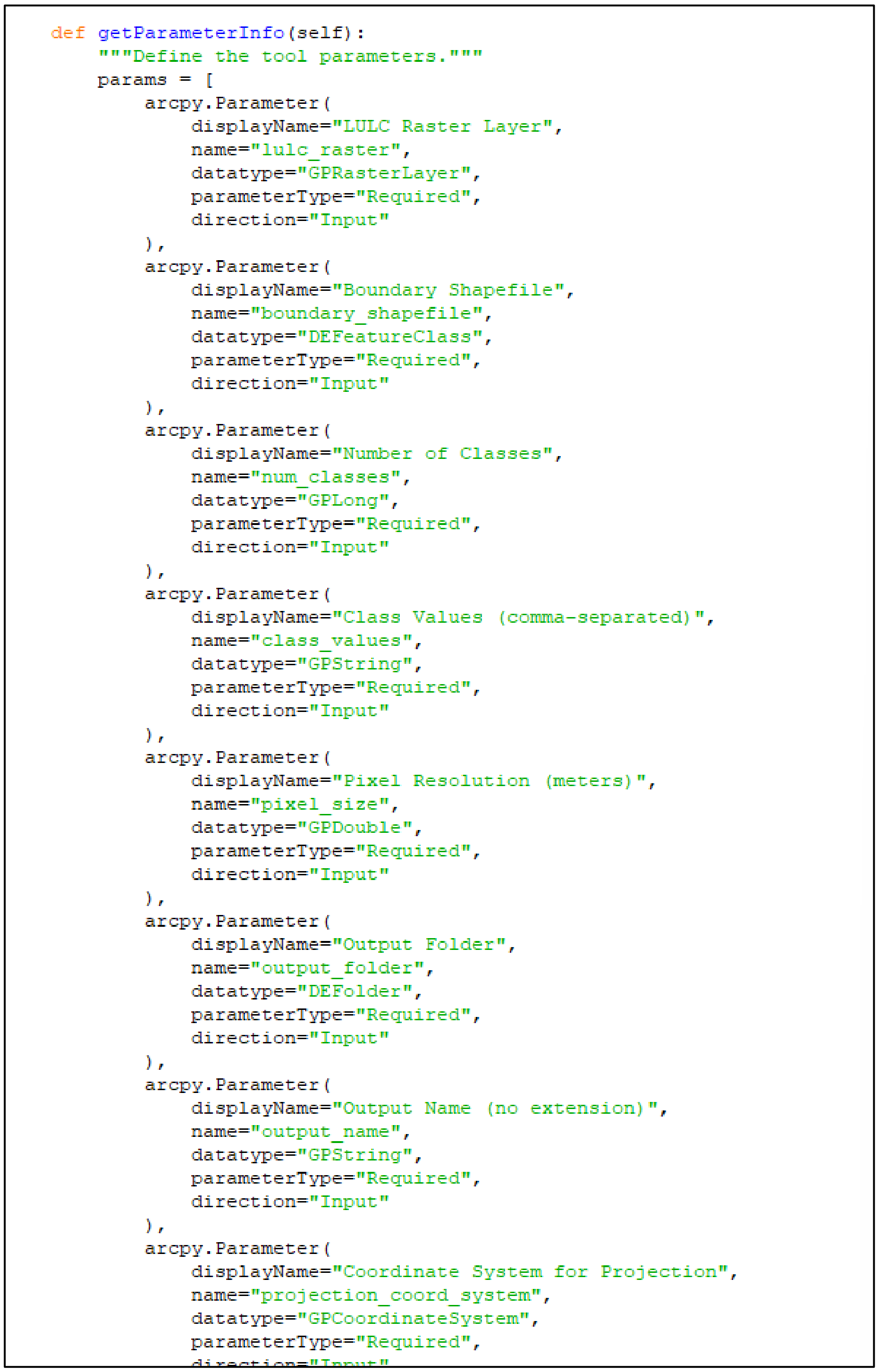


```python
class Toolbox(object):
    def __init__(self):
        """Define the toolbox (the name of the toolbox is the name of the
        .pyt file)."""
        self.label = "Toolbox"
        self.alias = "toolbox"

        # List of tool classes associated with this toolbox
        self.tools = [Tool]


class Tool(object):
    def __init__(self):
        """Define the tool (tool name is the name of the class)."""
        self.label = "LULC Area Calculator"
        self.description = ""
        self.canRunInBackground = False
```

2.3. Define Input Parameters

The getParameterInfo method configures user inputs:

Raster layer, boundary shapefile, class values, pixel size, and output paths. Parameters are validated (e.g., ensuring the number of classes matches input values).



```
def getParameterInfo(self):
    """Define the tool parameters."""
    params = [
        arcpy.Parameter(
            displayName="LULC Raster Layer",
            name="lulc_raster",
            datatype="GPRasterLayer",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Boundary Shapefile",
            name="boundary_shapefile",
            datatype="DEFeatureClass",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Number of Classes",
            name="num_classes",
            datatype="GPLong",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Class Values (comma-separated)",
            name="class_values",
            datatype="GPString",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Pixel Resolution (meters)",
            name="pixel_size",
            datatype="GPDouble",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Output Folder",
            name="output_folder",
            datatype="DEFolder",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Output Name (no extension)",
            name="output_name",
            datatype="GPString",
            parameterType="Required",
            direction="Input"
        ),
        arcpy.Parameter(
            displayName="Coordinate System for Projection",
            name="projection_coord_system",
            datatype="GPCoordinateSystem",
            parameterType="Required",
            direction="Input"
```

2.4. Execute the Workflow

In execution, user inputs (e.g., raster path, class values) are parsed. Class values are split into a list of integers (e.g., 1,2,3 → [1,2,3]).

```
def execute(self, parameters, messages):
    """The source code of the tool."""
    arcpy.env.overwriteOutput = True

    # Get user inputs
    raster_layer = parameters[0].valueAsText
    boundary_shapefile = parameters[1].valueAsText
    num_classes = int(parameters[2].valueAsText)
    class_values_str = parameters[3].valueAsText
    pixel_size = float(parameters[4].valueAsText)
    output_folder = parameters[5].valueAsText
    output_name = parameters[6].valueAsText
    projection_coord_system = parameters[7].valueAsText

    class_values = [int(x.strip()) for x in class_values_str.split(',') if x.strip()]
    if len(class_values) != num_classes:
        raise ValueError("Number of class values must match the specified number of classes.")

    if not os.path.exists(output_folder):
        os.makedirs(output_folder)

    raster = arcpy.Raster(raster_layer)
    projected_raster_path = os.path.join(output_folder, f"{output_name}_projected.tif")
    clipped_raster_path = os.path.join(output_folder, f"{output_name}_clipped.tif")
    vector_output = os.path.join(output_folder, f"{output_name}.shp")
    dissolved_output = os.path.join(output_folder, f"{output_name}_dissolved.shp")
    csv_output = os.path.join(output_folder, f"{output_name}.csv")
    chart_output = os.path.join(output_folder, f"{output_name}_chart.png")

    # Project raster
    messages.addMessage("Projecting raster to selected coordinate system...")
    arcpy.ProjectRaster_management(raster, projected_raster_path, projection_coord_system)
```

2.5. Project the Raster

The input raster is projected to a user-specified coordinate system (e.g., UTM) using arcpy. ProjectRaster_management. This ensures area calculations use linear units (meters/km²) instead of degrees.

```
# Project raster
messages.addMessage("Projecting raster to selected coordinate system...")
arcpy.ProjectRaster_management(raster, projected_raster_path, projection_coord_system)
```

2.6. Clipping the Raster and Calculate Class Areas

The projected raster is clipped to the boundary shapefile using arcpy. Clip management. This restricts analysis to the study area, ignoring pixels outside the boundary.

The clipped raster is converted to a NumPy array. For each class:

- Pixel count: np.sum(raster_array == class_val) counts pixels matching the class.
- Area: Calculated as (pixel count × pixel size²), converted to km².
- Results are stored in a panda DataFrame and saved to a CSV file.

```
# Clip raster to boundary
messages.addMessage("Clipping raster to specified boundary shapefile...")
arcpy.Clip_management(
    in_raster=projected_raster_path,
    out_raster=clipped_raster_path,
    in_template_dataset=boundary_shapefile,
    nodata_value="0",
    clipping_geometry="ClippingGeometry"
)

clipped_raster = arcpy.Raster(clipped_raster_path)
raster_array = arcpy.RasterToNumPyArray(clipped_raster)

messages.addMessage("Calculating area statistics by class...")
result = []
for class_val in class_values:
    pixel_count = np.sum(raster_array == class_val)
    area_m2 = pixel_count * (pixel_size ** 2)
    area_km2 = area_m2 / 1e6
    result.append({
        "Class": class_val,
        "Pixel_Count": int(pixel_count),
        "Area_km2": round(area_km2, 4)
    })

df = pd.DataFrame(result)
df.to_csv(csv_output, index=False)
messages.addMessage(f"✅ Area stats saved to: {csv_output}")
```

2.7. Generate a Bar Chart

A bar chart is plotted with matplotlib, showing area per class. The chart is saved as a PNG file for quick visual analysis.

```
# Plotting bar chart
plt.figure(figsize=(8, 6))
plt.bar(df['Class'].astype(str), df['Area_km2'], color='skyblue')
plt.xlabel("LULC Class")
plt.ylabel("Area (km²)")
plt.title("LULC Class Area Distribution")
plt.tight_layout()
plt.savefig(chart_output)
plt.close()
messages.addMessage(f"📊 Area chart saved to: {chart_output}")
```

2.8. Convert Raster to Vector & Dissolve

The clipped raster is converted to polygons (arcpy.RasterToPolygon). Polygons of the same class are merged using arcpy.Dissolve. A new field Area_km2 is added to store polygon areas calculated with SHAPE@.getArea.

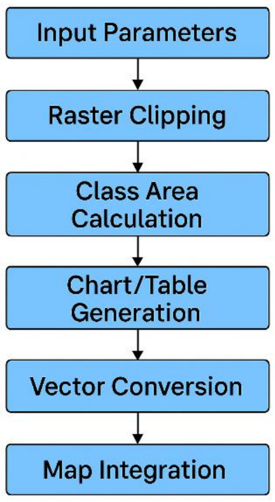
```
    # Convert raster to polygon and dissolve
    messages.addMessage("Converting clipped raster to polygon...")
    arcpy.RasterToPolygon_conversion(clipped_raster, vector_output, "NO_SIMPLIFY", "Value")

    messages.addMessage("Dissolving polygons by class value...")
    arcpy.Dissolve_management(vector_output, dissolved_output, "gridcode")

    messages.addMessage("Calculating area per class polygon...")
    arcpy.AddField_management(dissolved_output, "Area_km2", "DOUBLE")
    with arcpy.da.UpdateCursor(dissolved_output, ["SHAPE@", "Area_km2"]) as cursor:
        for row in cursor:
            row[1] = round(row[0].getArea("PLANAR", "SQUAREKILOMETERS"), 4)
            cursor.updateRow(row)
```

2.9. Add Results to ArcGIS Pro Map

The dissolved polygon layer is added to the active ArcGIS Pro map using arcpy.mp.ArcGISProject allowing users to visualize class boundaries directly in GIS.

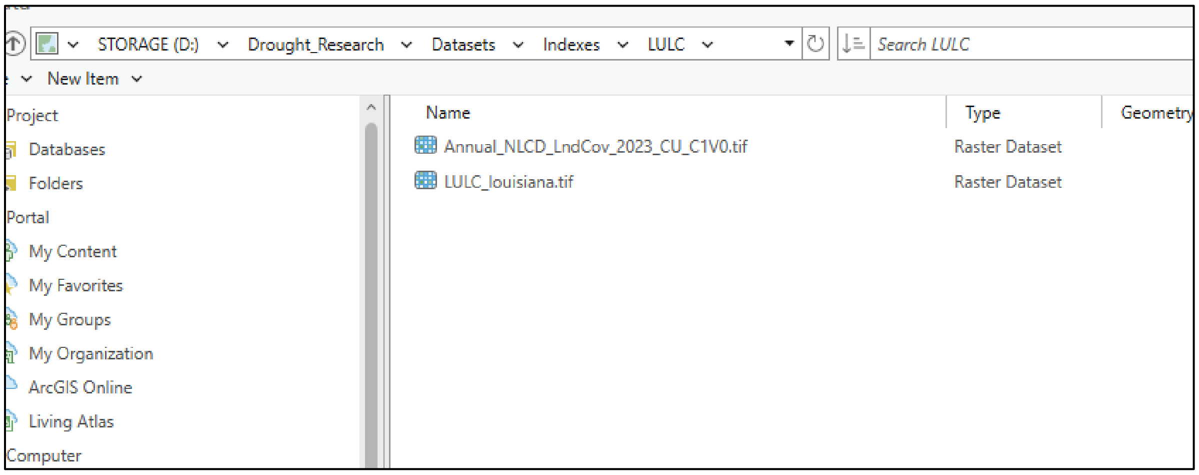
```
    # Add to map
    aprx = arcpy.mp.ArcGISProject("CURRENT")
    map_view = aprx.activeMap
    map_view.addDataFromPath(dissolved_output)

    messages.addMessage("🎉 Done! Vector layer and class area chart have been generated and addec

    return

def postExecute(self, parameters):
    """This method takes place after outputs are processed and
    added to the display."""
    return
```

2.10 Methodological Framework of the Toolbox



3. Running the toolbox in Arc GIS Pro

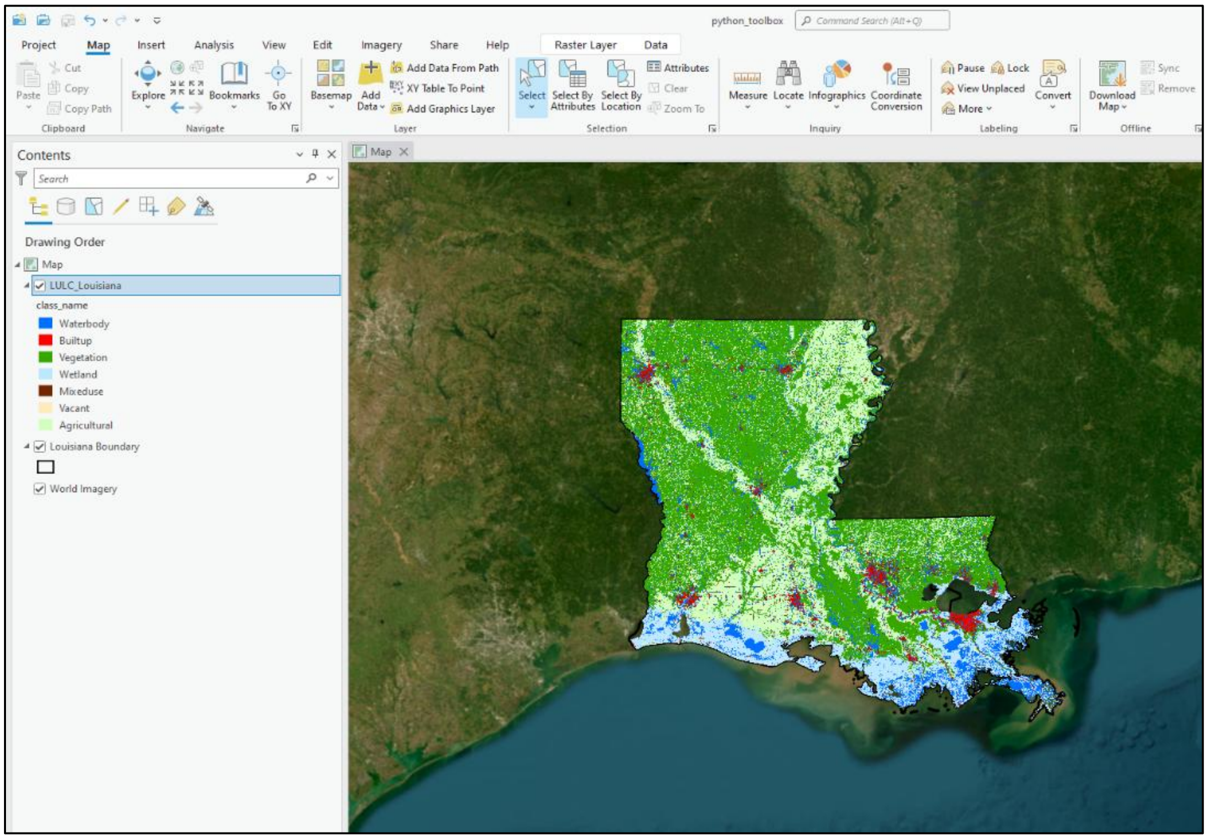
3.1 Adding the Datasets



Name	Type	Geometry
Annual_NLCD_LndCov_2023_CU_C1V0.tif	Raster Dataset	
LULC_louisiana.tif	Raster Dataset	

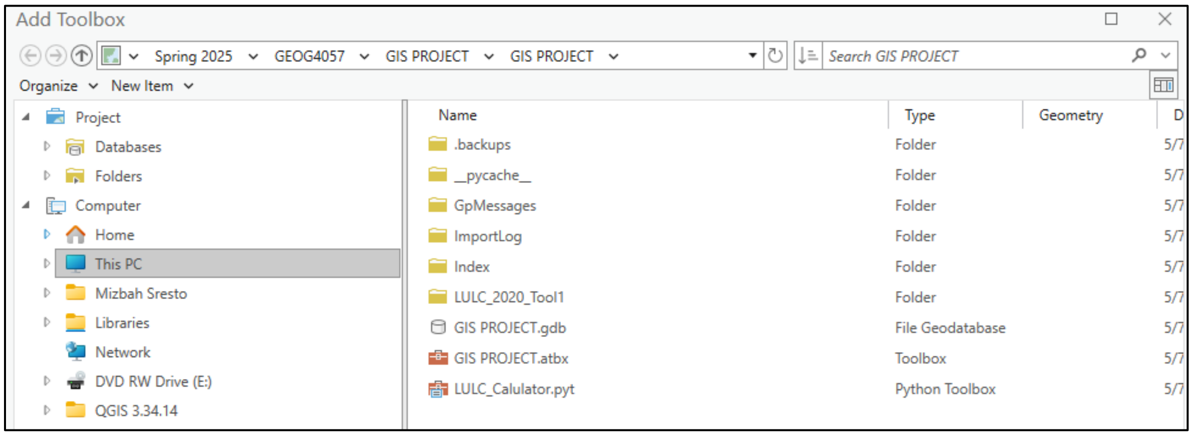
3.2 Map Layer in Arc GIS Pro

From the add data section, LULC raster data has been added.

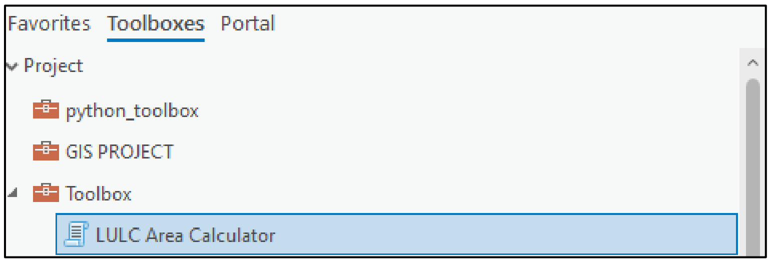


3.3 Adding the Toolbox

In this step the toolbox was added from add toolbox option.



Name	Type	Geometry
.backups	Folder	
__pycache__	Folder	
GpMessages	Folder	
ImportLog	Folder	
Index	Folder	
LULC_2020_Tool1	Folder	
GIS PROJECT.gdb	File Geodatabase	
GIS PROJECT.atbx	Toolbox	
LULC_Calulator.pyt	Python Toolbox	



3.4 The Toolbox

The script starts by extracting and validating the user-defined parameters to ensure the correct format and completeness. It converts the Class Values input into a list of integers, checks the consistency of the Number of Classes with the provided class values, and verifies the output directory's existence, creating it if needed.

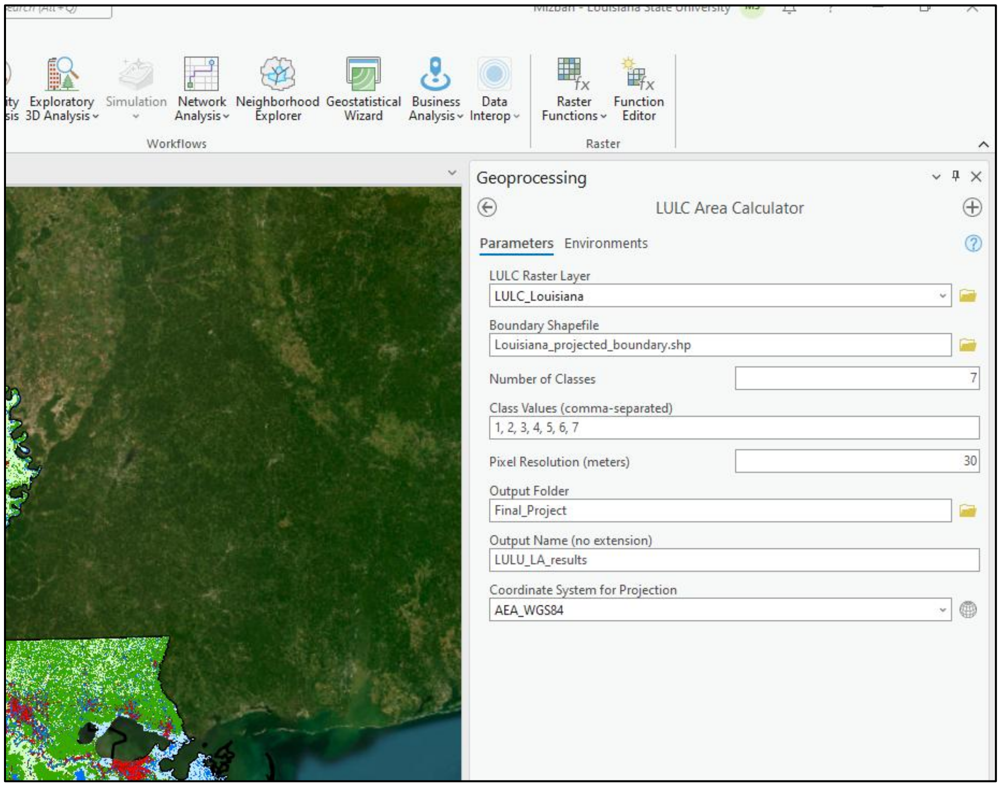
The LULC raster is reprojected into the specified coordinate system for spatial consistency, a critical step when working with datasets from multiple sources. The raster is then clipped using the boundary shapefile to focus on the area of interest, minimizing unnecessary processing.

The clipped raster is converted to a NumPy array to facilitate efficient pixel-wise analysis. For each LULC class, the script counts the pixels and calculates the corresponding area in square kilometers, with the results exported to a .csv file for easy access and further analysis.

A bar chart is generated to visually represent the area of each LULC class in square kilometers. This provides a clear, intuitive comparison of the land cover distribution. The chart is saved as a .png image, suitable for inclusion in reports or presentations.

The script converts the raster into vector format, turning raster cells into polygons, each associated with a class value (gridcode). It then dissolves the polygons by class, merging adjacent polygons with the same value, and calculating the area for each class. The dissolved polygons are embedded with area attributes, and the resulting shapefile is ready for further analysis.

To complete the workflow, the script automatically adds the dissolved shapefile to the current ArcGIS Pro map. This integration allows for immediate visualization and spatial analysis.



Geoprocessing

LULC Area Calculator

Parameters Environments

LULC Raster Layer: LULC_Louisiana

Boundary Shapefile: Louisiana_projected_boundary.shp

Number of Classes: 7

Class Values (comma-separated): 1, 2, 3, 4, 5, 6, 7

Pixel Resolution (meters): 30

Output Folder: Final_Project

Output Name (no extension): LULU_LA_results

Coordinate System for Projection: AEA_WGS84

3.5 Output

Area Calculation (CSV Output):

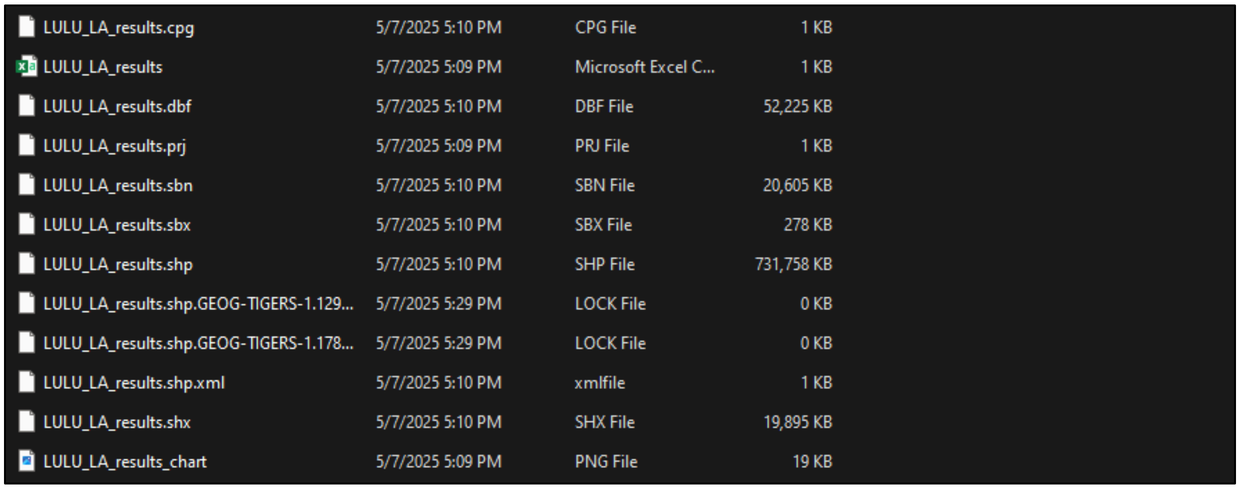
It calculates the area of each land use class by converting the clipped raster into a NumPy array, counting pixels per class, and computing the corresponding area in square kilometers.

Bar Chart Generation:

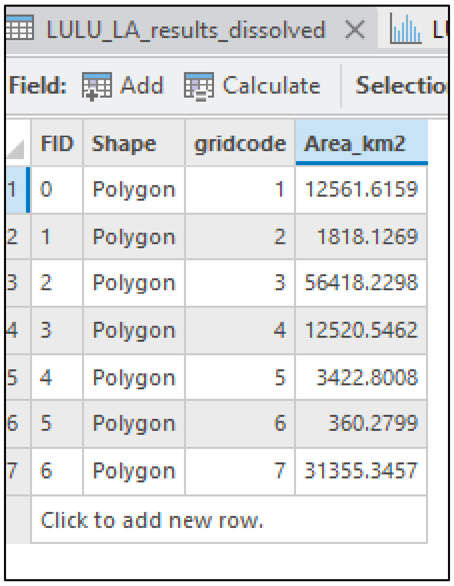
For visual interpretation, the toolbox creates a bar chart showing the area distribution of each LULC class. This graphical output helps users quickly compare land cover types and is saved as a PNG image for reporting or presentation purposes.

Polygon Output (Raster-to-Vector Conversion):

The toolbox converts the classified raster into vector format, transforming raster cells into polygons. It dissolves adjacent polygons with the same class value, calculates area attributes, and generates a final shapefile. This output is ready for advanced spatial analysis and integration into GIS projects.



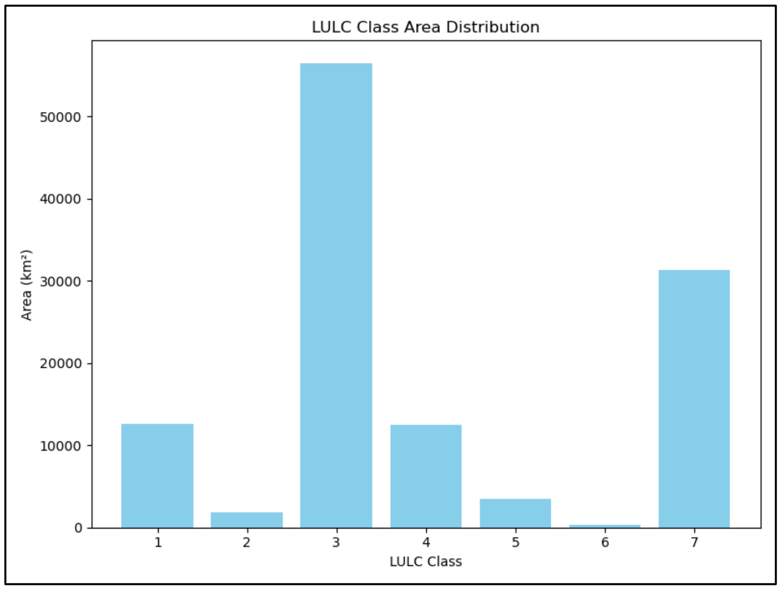
```
LULU_LA_results.cpg                          5/7/2025 5:10 PM    CPG File                1 KB
LULU_LA_results                              5/7/2025 5:09 PM    Microsoft Excel C...    1 KB
LULU_LA_results.dbf                          5/7/2025 5:10 PM    DBF File           52,225 KB
LULU_LA_results.prj                          5/7/2025 5:09 PM    PRJ File                1 KB
LULU_LA_results.sbn                          5/7/2025 5:10 PM    SBN File           20,605 KB
LULU_LA_results.sbx                          5/7/2025 5:10 PM    SBX File              278 KB
LULU_LA_results.shp                          5/7/2025 5:10 PM    SHP File          731,758 KB
LULU_LA_results.shp.GEOG-TIGERS-1.129...     5/7/2025 5:29 PM    LOCK File               0 KB
LULU_LA_results.shp.GEOG-TIGERS-1.178...     5/7/2025 5:29 PM    LOCK File               0 KB
LULU_LA_results.shp.xml                      5/7/2025 5:10 PM    xmlfile                 1 KB
LULU_LA_results.shx                          5/7/2025 5:10 PM    SHX File           19,895 KB
LULU_LA_results_chart                        5/7/2025 5:09 PM    PNG File               19 KB
```

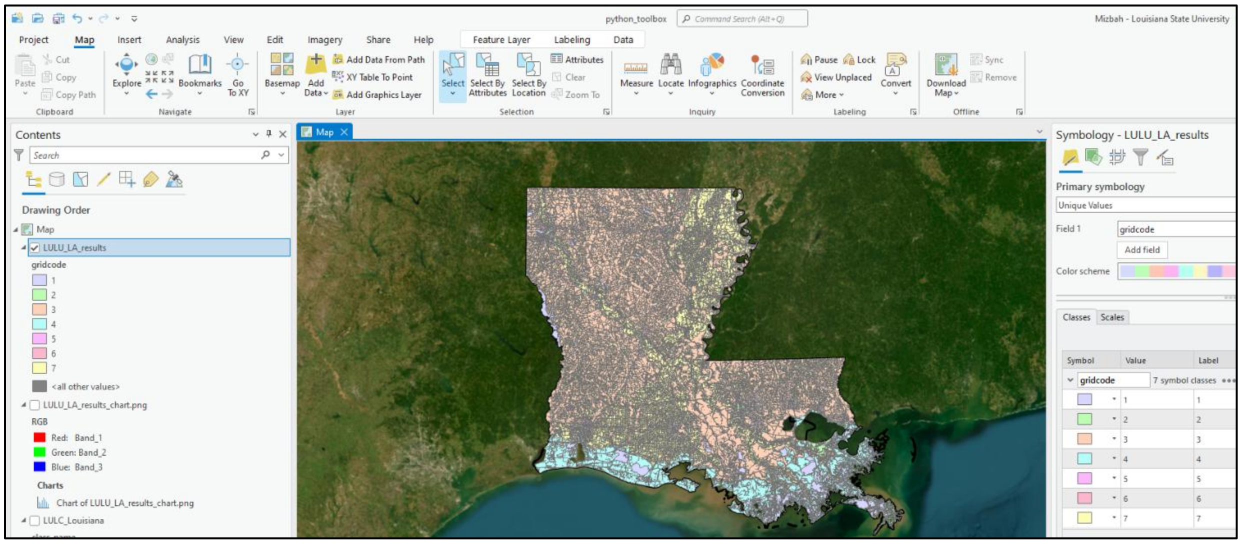


LULU_LA_results_dissolved

Field: Add Calculate Selectio

	FID	Shape	gridcode	Area_km2
1	0	Polygon	1	12561.6159
2	1	Polygon	2	1818.1269
3	2	Polygon	3	56418.2298
4	3	Polygon	4	12520.5462
5	4	Polygon	5	3422.8008
6	5	Polygon	6	360.2799
7	6	Polygon	7	31355.3457
	Click to add new row.			





Drawing Order

- Map
  - ☑ LULU_LA_results
    - gridcode
    - 🟪 1
    - 🟩 2
    - 🟧 3
    - 🟦 4
    - 🟪 5
    - 🟥 6
    - 🟨 7
    - ⬛ <all other values>
  - ☐ LULU_LA_results_chart.png
    - RGB
    - 🟥 Red: Band_1
    - 🟩 Green: Band_2
    - 🟦 Blue: Band_3
    - Charts
    - Chart of LULU_LA_results_chart.png
  - ☐ LULC_Louisiana

Symbology - LULU_LA_results

Primary symbology: Unique Values

Field 1: gridcode

Add field

Color scheme

Classes Scales

Symbol	Value	Label
gridcode	7 symbol classes	
🟪	1	1
🟩	2	2
🟧	3	3
🟦	4	4
🟪	5	5
🟥	6	6
🟨	7	7

4. Discussion and Conclusion

The developed toolbox offers a streamlined and efficient solution for land use/land cover (LULC) analysis, with notable strengths in speed, accuracy, and seamless integration with ArcGIS Pro. It automates essential tasks such as area calculation, chart generation, and raster-to-vector conversion, producing outputs that are easily interpretable and ready for further spatial analysis. These features make it highly valuable for GIS professionals, researchers, and educators.

Despite its strengths, the toolbox has some limitations. It relies on the assumption that input rasters are correctly classified and currently supports only Euclidean area calculations, which may not account for projection-related distortions. Addressing these limitations in future versions such as adding support for other area units (acres, hectares) and allowing for custom classification schemes could enhance its versatility and accuracy.