



CoRE Stack

Winter Internship

IIT Delhi

Abhinav Kumar Pandey

Documentation



Acknowledgments

I am immensely grateful to Prof. Aaditeshwar Seth of the Department of Computer Science and Engineering, IIT Delhi, under whom I have done this Internship and learned so much. I would also like to express my heartfelt thanks to Shivani Ma'am and the entire ICTD lab who have kept my spirits up and have always been ready to help me. Last but not in anyway the least, I am eternally grateful to my parents who have supported me throughout thick and thin in this project and beyond.

Contents

1	Introduction	1
2	Analysis of Tabular Data by LLMs	2
1	Description of the Excel file	2
2	Defining the functions	2
3	Different Excel functionalities through LLMs	3
3.A	Displaying columns and rows from the Excel	3
3.B	Complex functions	3
3.C	Plotting a pie chart	3
3	Analysis of data from HTML files	4
1	Defining the functions	4
2	Function executions	4
3	Text based queries	5
4	Image based queries	5
4.A	Query for analysis based on image in HTML file.	5
4.B	Query to display image present in the HTML file	5
4	Geo-spatial analysis by LLMs	6
1	Plotting and analyzing the intersection of shapefiles	6
1.A	Defining the functions	6
1.B	Plotting the intersection with QGIS	6
1.C	Plotting the intersection in Python	7
1.D	Plotting the intersection with the LLM	7
2	Plotting and analyzing the intersection of GeoJSON files	7
2.A	Defining the functions	8
2.B	Plotting the intersection	8
5	Bypassing the errors	10
1	Pie chart from Excel file	10
2	Plotting the intersection of shapefiles	12
3	Plotting the intersection of GeoJSON files	12
6	Conclusions	14

Abstract

The Commoning for Resilience and Equality (CoRE) stack is a collaborative effort with several academic research institutions, civil society organizations working on ecological sustainability and rural livelihood, innovation catalysts, and technology-led social enterprises. Being designed as a Digital Public Good, the CoRE stack enables an open-access co-creation network to innovate and scale digital technology solutions for ecosystem sustainability. Within the overall realm of CoRE Stack, my work in the ICTD lab focuses mainly on leveraging LLMs to analyze data in different forms, be it HTML, Excel, GeoJSON, Vector and Raster images. Eventually based on the abilities, we hope to one day be able to develop a chatbot which people in rural areas can access to get information about their area. Be it deforestation amounts, Land Use Land Cover(LULC) data, or soil profile for the area, the user should be able to access them in a matter of minutes.

Chapter 1

Introduction

A watershed, also known as a drainage basin or catchment area, is an area of land where all the water that falls or drains into it ultimately drains into a common outlet, such as a river, lake, or ocean. This includes surface water runoff, precipitation, and even underground water flow. Watersheds play a crucial role in the hydrological cycle, influencing the distribution and movement of water across the landscape.

Based on the size, the hydrological units are termed as basin, sub-basin, catchment, sub-catchment, watershed, sub-watershed and micro-watershed. India's Water Resources Information System (WRIS) divides the Indian subcontinent into 25 major river basins with major river basin of Ganga-Brahmaputra-Meghna followed by Indus, Mahanadi, Godavari and Krishna.

These sub divisions are represented in two forms - Vector and Raster Images. Raster and vector files are the two most popular formats used for visual content. They represent images in very different ways, so there's a lot to consider when deciding which one to use.

Raster files are images built from pixels — tiny colour squares that, in great quantity, can form highly detailed images such as photographs. The more pixels an image has, the higher quality it will be, and vice versa. The number of pixels in an image depends on the file type (for example, JPEG, GIF or PNG).

Vector files use mathematical equations, lines and curves with fixed points on a grid to produce an image. There are no pixels in a vector file. A vector file's mathematical formulae capture shape, border and fill colour to build an image. Because the mathematical formula recalibrates to any size, you can scale a vector image up or down without affecting its quality.

All the files analyzed in this document can be found at the GramVaani [Geoserver](#).

The colab files and images used in this project are located in a public [repository](#) hosted by Github. For the analysis present in this document, I have used GPT-4o-mini through API key.

The use of Large Language Models (LLMs) in geospatial analysis is a recent development, spurred by the advancements in LLMs' ability to reason and understand semantic information. The development of leveraging LLMs for geospatial analysis has been rapid, with increasing sophistication in approaches. The field is still in its nascent stage, with ongoing research addressing challenges like the uncertainty of LLM outputs, the need for specialized training data in GIS, and the development of more robust and adaptable frameworks. Nonetheless, the progress made so far points towards a promising future for LLMs as powerful tools in geospatial research and applications.

Chapter 2

Analysis of Tabular Data by LLMs

1 Description of the Excel file

For my first analysis, I provided GPT-4o-mini with an Excel file labeled as "Masalia_data". This file had several sheets with different type of data on each one. The way I proceeded was to first define four functions, one which loads the Excel file, one which uses the Excel file to provide context to the LLM, the query to the LLM, and finally the main function which executes the LLM call. A representation of the first sheet is shown in Fig.2.1 and the function pipeline is shown as a flow chart in Fig.2.2.

UID	area_in_ha	terrainCluster_ID	Terrain_Description	% of area_hill_slope	% of area_plain_slope	% of area_ridge_area	% of area_slopy_area	% of area_water_area
12_305795	771.284439	3	Broad Plains and Slopes	0.579553325	60.34057151	7.586122448	12.345150518	3.17504761
12_305796	113.000000	3	Broad Plains and Slopes	0.453755325	60.34057151	7.586122448	12.345150518	3.17504761
12_308402	1410.86473	3	Broad Plains and Slopes	0.435262316	60.29266506	5.380878356	26.65604589	3.517504782
12_308404	1618.388478	3	Misty Plains	0.435262316	81.30200000	2.889925452	16.38045891	4.435807787
12_308405	1717.000000	3	Misty Plains	0.325937154	78.21269394	5.380878356	16.853386189	2.358874889
12_312558	1850.157169	3	Misty Plains	0.139574374	78.55453111	2.510694645	16.21628395	1.579117937
12_312559	1950.000000	3	Misty Plains	0.323056816	78.47455972	18.20254793	16.21628395	2.220967259
12_313377	760.205669	3	Misty Plains	0.323056816	78.47455972	1.140672985	18.20254793	0.584107051
12_314841	1620.684416	3	Broad Plains and Slopes	0.411006438	66.93146704	5.762059814	15.50627148	4.489104823
12_314842	1620.684416	3	Broad Plains and Slopes	0.411006438	58.13200000	7.055902005	15.50627148	3.770504789
12_317061	1782.290065	3	Misty Plains	2.040402931	75.37395831	2.262093402	18.52487842	1.820967259
12_318531	2342.956564	3	Misty Plains	0.80574826	76.15909218	8.009838118	12.747954527	2.747093204
12_318532	2342.956564	3	Broad Plains and Slopes	0.325937154	65.31269394	7.151322597	12.747954527	4.020967259
12_319548	2389.498236	3	Misty Plains	0.186432809	78.569664	6.377846714	11.4909067	3.77514481
12_319781	632.060033	3	Broad Plains and Slopes	0.593165773	45.41874327	11.60679321	15.49051609	8.30382841
12_320080	728.883198	3	Misty Plains	0.015339808	84.36234144	3.00000000	15.49051609	0.00000000
12_320081	728.883198	3	Misty Plains	0.015339808	77.00000000	2.063920217	13.51011203	0.24986095
12_321768	788.1564991	3	Misty Plains	0	76.00000000	4.02534616	18.585084	1.48057956
12_322101	1635.946028	3	Misty Plains	0.061137631	75.74970587	4.190259891	16.00456143	3.99113518
12_322311	1073.795148	2	Misty Plains	0.26685431	79.78004300	2.3744059725	15.41209444	1.06618308

Figure 2.1: Structure of the Excel file

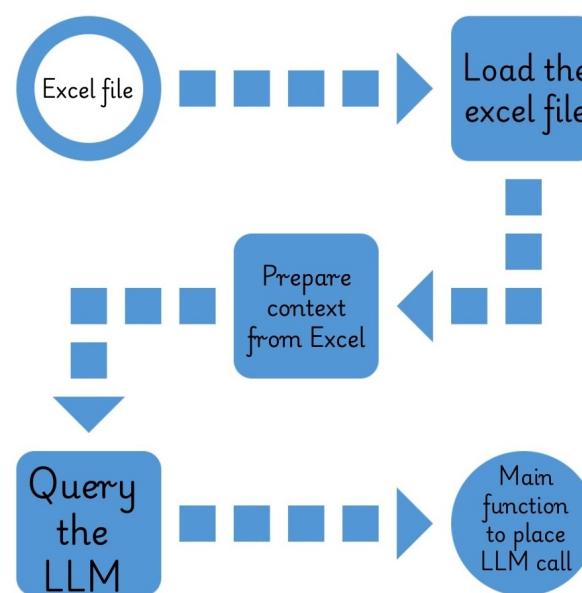


Figure 2.2: function pipeline for Excel analysis

2 Defining the functions

The Code snippets displayed in Fig.2.3, 2.4, and 2.5 show how I have defined the functions.

```

def load_excel_file(file_path):
    """
    Loads an Excel file and returns all sheet names and data from each sheet.
    """
    xls = pd.ExcelFile(file_path)
    sheet_names = xls.sheet_names

    sheets_data = {}
    for sheet in sheet_names:
        df = pd.read_excel(file_path, sheet_name=sheet)
        sheets_data[sheet] = df

    return sheet_names, sheets_data
  
```

Figure 2.3: function to load the Excel file

```

def prepare_context_from_excel(sheet_names, sheets_data):
    """
    Prepares a text-based context from the Excel file's data.
    """
    context = "Excel File Content:\n\n"
    for sheet in sheet_names:
        context += f"Sheet Name: {sheet}\n"
        context += f"Columns: {', '.join(sheets_data[sheet].columns)}\n"
        context += f"First 5 Rows:{'\n'.join(sheets_data[sheet].head().to_string(index=False))}\n\n"
    return context

def ask_question_to_llm(context, question):
    """
    Sends the context and user question to the LLM and retrieves the answer.
    """
    messages = [
        {"role": "system", "content": "You are an expert at analyzing and summarizing data from Excel files."},
        {"role": "user", "content": f"Context:{context}\nQuestion: {question}"}
    ]
    response = openai.ChatCompletion.create(
        model="gpt-4o-instruct",
        messages=messages,
        max_tokens=1000,
        temperature=0.7
    )
    return response.choices[0].message["content"]
  
```

Figure 2.4: functions to generate context and ask question to the LLM

```

# Main Function
def main(file_path, question):
    sheet_names, sheets_data = load_excel_file(file_path)
    context = prepare_context_from_excel(sheet_names, sheets_data)

    if len(context) > 100000:
        context = context[:100000]

    answer = ask_question_to_llm(context, question)
    return answer
  
```

Figure 2.5: Main function

3 Different Excel functionalities through LLMs

3.A Displaying columns and rows from the Excel

Now what we want to explore is whether the LLM can display the Excel columns which it has analyzed.

```
file_path = '/content/Masalia_data.xlsx'
question = "Can you display the columns and
↪ their first few rows from the Excel file?"
answer = main(file_path, question)
print(answer)
```

Here are the columns and their first few rows. From each sheet in the provided Excel file:							
## Sheet: terrain							
UID	area_in_hac	terrainCluster_ID	Terrain_Description	% of area hill_slope	% of area plain_area	% of area ridge_area	% of area slope_area
12_385795	771.728346	1	Broad Plains and Slopes	11.131331	51.816508	7.685622	17.245198
12_387989	2158.443393	1	Broad Plains and Slopes	14.544277	68.346056	7.144277	8.019658
12_388492	1419.864739	1	Broad Plains and Slopes	3.435826	68.829685	5.588878	26.636843
12_388493	1419.864739	1	Mostly Plains	1.183213	81.822311	4.886213	13.174451
12_389095	1554.517784	1	Mostly Plains	8.182937	91.212693	5.638018	8.815288

## Sheet: terrain_wlc_plain						
UID	area_in_hac	terrainCluster_ID	Terrain_Description	Cluster_Type	% of area barren	% of area for
12_385795	771.728346	1	Broad Plains and Slopes	Mix of Cropped(<5%) and Trees/Forests(>20%)	8.815846	35.297367
12_387989	2158.443393	1	Broad Plains and Slopes	Mix of Cropped(<5%) and Trees/Forests(>20%)	8.819982	25.561745
12_388492	1419.864739	1	Broad Plains and Slopes	Mix of Cropped(<5%) and Trees/Forests(>20%)	8.805569	20.000000
12_388493	1419.864739	1	Mostly Plains	Mix of Cropped(<5%) and Trees/Forests(>20%)	8.846484	22.699529
12_389095	1554.517784	1	Mostly Plains	Mostly Trees/Forests(>30%)	8.859473	2.355644

Figure 2.6: Output which displays the Excel file

As we can see in Fig.2.6, the LLM is able to display the columns of Excel files with excellent accuracy.

3.B Complex functions

In this subsection, we explore if the LLM can perform more complex functions like addition and multiplication albeit simultaneously. What I am going to ask the LLM to compute is the total area which comes under hill slope and ridge area for a particular UID from the Excel file. For this, it would have to access the percentages of hill slope and ridge area, multiply them independently to the total area and then add them.

```
file_path = '/content/Masalia_data.xlsx'
question = "Can you tell me the total amount
↪ of area hill slope and ridge area for UID
↪ 12_305795?"
answer = main(file_path, question)
print(answer)
```

As we can see in Fig.2.7, the LLM gives us the whole calculation for the process and is able to process the total area.

```
To find the total area for the hill slope and ridge area for UID 12_305795, we will refer to the data provided in the "terrain" sheet. In the "terrain" sheet, for UID 12_305795, we have the following data:
- % of area hill_slope:: 11.131331%
- % of area ridge_area:: 7.685622%
- %area_in_hac:: 771.728344 hac
Now, we can calculate the areas as follows:
1. **Area of hill slopes:
\text{Area of hill slope} = \left(\frac{\% \text{ of area hill_slope}}{100}\right) \times \text{area\_in\_hac}
\text{Area of hill slope} = \left(\frac{11.131331}{100}\right) \times 771.728344 \approx 85.8474 \text{ (hac)}
2. **Area of ridge areas:
\text{Area of ridge area} = \left(\frac{\% \text{ of area ridge_area}}{100}\right) \times \text{area\_in\_hac}
\text{Area of ridge area} = \left(\frac{7.685622}{100}\right) \times 771.728344 \approx 59.3975 \text{ (hac)}
Now, add the two areas together:
\text{Total area} = \text{Area of hill slope} + \text{Area of ridge area}
\text{Total area} \approx 85.8474 + 59.3975 \approx 145.2449 \text{ (hac)}
Thus, the total amount of area for hill slope and ridge area for UID 12_305795 is approximately ==145.24 hac==.
```

Figure 2.7: LLM output for total area

3.C Plotting a pie chart

Now we are going to explore whether the LLM can produce visual outputs like a Pie chart based on the Excel data that it has.

```
file_path = '/content/Masalia_data.xlsx'
question = "Can you plot a pie chart of hill
↪ slope, plane area, ridge area, slopy area
↪ and valley area for UID 12_307609"
answer = main(file_path, question)
print(answer)
```

```
To plot a pie chart of the area distribution for UID 12_307609, we can extract the relevant percentage values from the "terrain" sheet. Based on the provided data for UID 12_307609 in the "terrain" sheet, we have the following percentage areas:
- % of area hill_slope: 4.597656
- % of area plain_area: 68.346056
- % of area ridge_area: 7.144277
- % of area slope_area: 24.733317
- % of area valley_area: 3.175952
To create a pie chart, we can use a data visualization tool or software like Excel, Python, or any other tool that supports chart creation.
## Python Code Example
```python
import matplotlib.pyplot as plt

Data for UID 12_307609
labels = ['Hill Slope', 'Plain Area', 'Ridge Area', 'Slopy Area', 'Valley Area']
sizes = [4.597656, 68.346056, 7.144277, 24.733317, 3.175952]
colors = ['red', 'blue', 'green', 'orange', 'purple']
explode = (0.1, 0, 0, 0, 0) # explode the 1st slice

Create pie chart
plt.figure(figsize=(6, 6))
plt.pie(sizes, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=140)
plt.title("Area Distribution for UID 12_307609")
plt.axis('equal') # Equal aspect ratio ensures that pie chart is circular.
plt.show()

Steps to Create the Pie Chart in Excel:
1. Open Excel and input the data:
 - In column A, enter the labels:
 - Hill Slope
 - Plain Area
 - Ridge Area
 - Slopy Area
 - Valley Area
 - In the adjacent column, enter the corresponding percentage values.
2. Select the data range (both columns).
3. Go to the "Insert" tab in the Excel ribbon.
4. Click on "Pie Chart" and select the type of pie chart you prefer (e.g., 2-D Pie).
5. Format the chart as needed, including adding data labels and a chart title.
By following either of these methods, you can successfully visualize the distribution of area types for UID 12_307609 in a pie chart.
```

Figure 2.8: output for pie chart creation

Contrary to our expectations, the LLM is not able to plot the pie chart. Instead it gives us the complete Python code which we can execute to plot the pie chart and the instructions on how to execute it as well. This is the limitation that we will be exploring in this document: whether LLMs are able to generate and analyze images for answers.

# Chapter 3

## Analysis of data from HTML files

In this chapter, we explore the analysis of data from a .html file. The key challenge in applying LLMs to HTML data analysis lies in effectively bridging the gap between the unstructured nature of HTML and the structured data formats that LLMs are typically trained on.

### 1 Defining the functions

The HTML file I use over here for my analysis is a Micro-watershed report of Pindwara, Rajasthan. What we want to explore is whether the LLM can analyze and retrieve information from inside the HTML file. We use libraries like Pytesseract, BeautifulSoup, BytesIO and Image to analyze the data from the HTML file.



Figure 3.1: functions flow

Here as well, we use five functions. One for extracting content from the HTML, one for processing images, one for preparing context to the LLM, the question pipeline and lastly the main function where we place the LLM call.

### 2 Function executions

The code snippets provided in Figs. 3.2, 3.3, 3.4, and 3.5 display the code I used to write the functions for the HTML analysis.

```
def extract_html_content(file_path):
 """
 Extracts text, images, and other data from an HTML file.
 """
 with open(file_path, "r", encoding="utf-8") as file:
 soup = BeautifulSoup(file, "html.parser")

 text_content = soup.get_text(separator="\n", strip=True)

 images = []
 for img_tag in soup.find_all("img"):
 img_src = img_tag.get("src")
 if img_src.startswith("data:image"):
 img_data = base64.b64decode(img_src.split(",")[1])
 images.append(Image.open(BytesIO(img_data)))
 elif img_src.startswith("http"):
 response = requests.get(img_src)
 images.append(Image.open(BytesIO(response.content)))
 else:
 img_path = os.path.join(os.path.dirname(file_path), img_src)
 if os.path.exists(img_path):
 images.append(Image.open(img_path))

 graphs = [img for img in images if "graph" in img.format.lower()]

 return text_content, images, graphs
```

Figure 3.2: function to extract content

```
def process_images(images):
 """
 Extracts text from images using OCR.
 """
 image_texts = []
 for img in images:
 try:
 text = pytesseract.image_to_string(img)
 image_texts.append(text.strip())
 except Exception as e:
 image_texts.append(f"Error processing image: {str(e)}")
 return image_texts

def prepare_context(text, image_texts):
 """
 Combines textual data and extracted image text into a single context for the LLM.
 """
 context = f"Text Content:\n{text}\n\n"
 for i, img_text in enumerate(image_texts):
 context += f"Image {i+1} Text:\n{img_text}\n\n"
 return context
```

Figure 3.3: functions combining image and text for common context

```

def ask_question_to_llm(context, question):
 """
 Sends the context and user question to the LLM and retrieves the answer.
 """
 messages = [
 {"role": "system", "content": "You are an expert at analyzing and summarizing complex data from HTML files."},
 {"role": "user", "content": f"Context:\n{context}\n\nQuestion: {question}"}
]
 response = openai.ChatCompletion.create(
 model="gpt-4-0613",
 messages=messages,
 max_tokens=1000,
 temperature=0.7
)
 return response.choices[0].message["content"]

```

Figure 3.4: Question pipeline

```

Main function
def main(html_file_path, question):
 text, images = extract_html_content(html_file_path)
 image_texts = process_images(images)
 context = prepare_context(text, images, image_texts)

 if len(context) > 100000:
 context = context[:100000]

 answer = ask_question_to_llm(context, question)
 return answer

```

Figure 3.5: Main function

### 3 Text based queries

First We will explore whether the LLM is able to answer questions from the text present in the HTML file.

```

html_file_path =
"/content/micro_watershed_report_pindwara.html"
question = "Can you display the distribution
↪ of land use types in microwatersheds?"
answer = main(html_file_path, question)
print(answer)

```

Fig 3.6 showcases that the LLM is able to answer queries based on text present in the LLM.

```

Based on the provided micro-watershed report, the distribution of land use types in the micro-watershed area from 2017 to 2022 is as follows:
- **Farmlands**: 59.2%
- **Barren Areas**: 2.3%
- **Shrubs**: 5.4%
This indicates that the predominant land use type in the micro-watershed is farmlands, comprising a significant majority of the land use distribution.

```

Figure 3.6: Output

### 4 Image based queries

Now what we want to explore is whether the LLM can answer queries base on images in the LLM or whether it can display images from the HTML file.

#### 4.A Query for analysis based on image in HTML file.

First, we are placing a query to the LLM based on a figure of the graph of rainfall in mm vs time in

years.

```

html_file_path =
"/content/micro_watershed_report_pindwara.html"
question = "What is the Rainfall trends for
↪ for UID 14_57832"
answer = main(html_file_path, question)
print(answer)

```

As displayed in Fig.3.7 the LLM gives us a hallucinated answer comprising of the text analysed from above the graph but is not able to retrieve any information from the actual graph itself.

```

The rainfall trends for UID 14_57832 indicate the following key points:
1. **Good Rainfall Years**: Between 2017 and 2018, the area experienced good rainfall years with an average annual rainfall of approximately 754 mm. The scenario
2. **Groundwater Depletions**: Despite the favorable rainfall, these years resulted in negative groundwater recharge, with an average groundwater change of -38.6
3. **Surface Runoff Potential**: During the good rainfall years, around 21.0% of the rainfall contributed to surface runoff, which presents potential for water
4. **Bad Rainfall Years**: In contrast, 2021 was marked as a bad rainfall year, resulting in significantly reduced annual rainfall averaging 446.1 mm. If
5. **Reduced Runoff**: During the bad rainfall years, runoff decreased to 14% of the total rainfall, which diminished the volume of harvestable water.

Overall, the trends highlight the need for improved water management practices, especially during drought years, to stabilize water availability and support agri

```

Figure 3.7: Failure to do image analysis

#### 4.B Query to display image present in the HTML file

Since we have seen that the LLM is not able to analyse images present in the HTML file, we will now explore whether the LLM is able to display the same graphical image present in the HTML file.

```

html_file_path =
"/content/micro_watershed_report_pindwara.html"
question = "Can you display the rainfall
↪ trends for UID 14_57832?"
answer = main(html_file_path, question)
print(answer)

```

Unfortunately, the LLM is not able to this either which leads us to conclude that the LLM is not able to perform image analysis or process images from an HTML file.

```

The provided context does not include specific rainfall data for UID 14_57832. However, it does mention general rainfall patterns in the micro-watershed area, indicating:
1. **Good Rainfall Years**:
 - In 2017 to 2018, the area experienced good rainfall years with an average annual rainfall of approximately 754 mm.
 - Rainfall during these years led to negative groundwater recharge, with an average groundwater change of -38.6 m, indicating depletion of groundwater resources.
 - Surface runoff potential became a critical concern, supporting potential for water harvesting.

2. **Bad Rainfall Years**:
 - In 2021, the area faced bad rainfall conditions, averaging around 446.1 mm of rainfall.
 - Groundwater recharge was significantly lower, averaging 77.2 m depletion.
 - Runoff reduced to 14% of the total rainfall, diminishing the harvestable water.

For a more detailed analysis specific to UID 14_57832, additional data specific to that identifier would be required. If you have specific rainfall data or trends for UID 14_57832,

```

Figure 3.8: Failure to display image from HTML file.

# Chapter 4

## Geo-spatial analysis by LLMs

In this final chapter, we will explore whether the LLM can analyze and display geo-spatial data.

### 1 Plotting and analyzing the intersection of shapefiles

I will be using the angul\_angul and the angul\_drought shapefiles for the first part of my analysis. I will first plot their intersection in QGIS, then display it by Python, then query the LLM on whether it is able to plot the intersection. I have downloaded these files in .zip format from the geoserver which I had mentioned in the Introduction chapter.

#### 1.A Defining the functions



Figure 4.1: Function pathway

As we can see in Fig4.1 we first unzip the files, store the shapefiles in a separate folder and access them, read them, prepare context based on these files, and lastly, place the LLM call for analysis of these shapefiles.

```
Function to unzip files
def unzip_files(zip_path, extract_to):
 with zipfile.ZipFile(zip_path, 'r') as zip_ref:
 zip_ref.extractall(extract_to)

Paths to ZIP files (replace with your file paths)
zip_file_1 = "/content/angul_drought.zip"
zip_file_2 = "/content/angul_angul.zip"

Directories to extract the shapefiles
extract_dir_1 = "extracted_shapefiles_1"
extract_dir_2 = "extracted_shapefiles_2"

Unzip the files
unzip_files(zip_file_1, extract_dir_1)
unzip_files(zip_file_2, extract_dir_2)

Locate shapefiles in the directories
shapefile_1 = [f for f in os.listdir(extract_dir_1) if f.endswith('.shp')][0]
shapefile_2 = [f for f in os.listdir(extract_dir_2) if f.endswith('.shp')][0]

Read the shapefiles
gdf1 = gpd.read_file(os.path.join(extract_dir_1, shapefile_1))
gdf2 = gpd.read_file(os.path.join(extract_dir_2, shapefile_2))
```

Figure 4.2: Unzip files and read shapefiles

```
Prepare data for LLM
def prepare_geodata_context(gdf1, gdf2):
 ...
 # Prepare a text-based context from geospatial data for the LLM.
 context = [
 f"Dataset 1: {len(gdf1)} geometries\n",
 f"Dataset 2: {len(gdf2)} geometries\n"
]
 return context

LLM analysis and plotting request
context = prepare_geodata_context(gdf1, gdf2)
prompt = (
 f"The context describes two geographical datasets. Use the following data to compute and plot their intersection:\n"
 f"{context}\n"
 "Please compute the intersection of these two datasets and plot the results, ensuring proper visualization of all geometries."
)

response = openai.ChatCompletion.create(
 model="gpt-4-0613",
 messages=[
 {"role": "system", "content": "You are a geospatial analyst capable of computing and plotting shapefile intersections."},
 {"role": "user", "content": prompt}
]
)
```

Figure 4.3: Context preparation and LLM call

Fig.4.2 and 4.3 contain the code snippets which I used to access the shapefiles and process the intersection.

#### 1.B Plotting the intersection with QGIS

Firstly I will plot the intersection in QGIS. Plotting the intersection of two shapefiles involves a simple library call. Fig.4.4 displays the angul\_angul file in orange, the angul\_drought in gray and the intersection in purple.

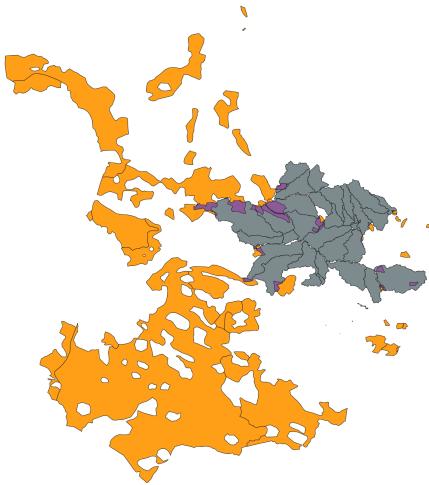


Figure 4.4: Intersection in QGIS

### 1.C Plotting the intersection in Python

To verify our diagram constructed in Python, I will plot the intersection in Python as well.

```
Compute the intersection
intersection = gpd.overlay(gdf1, gdf2,
 how='intersection')

Plot the result
plt.figure(figsize=(10, 8))
ax = plt.gca()
gdf1.plot(ax=ax, color='blue', alpha=0.5,
 label='Shapefile 1')
gdf2.plot(ax=ax, color='green', alpha=0.5,
 label='Shapefile 2')
if not intersection.empty:
 intersection.plot(ax=ax, color='red',
 alpha=0.7, label='Intersection')
else:
 print("No intersection found.")
ax.set_aspect('auto')
plt.legend()
plt.title('Intersection of Two Shapefiles')
plt.show()
```

As we can observe in Fig.4.5, the intersection has been plotted successfully with angul\_angul in green, angul\_drought in purple and the intersection in red.

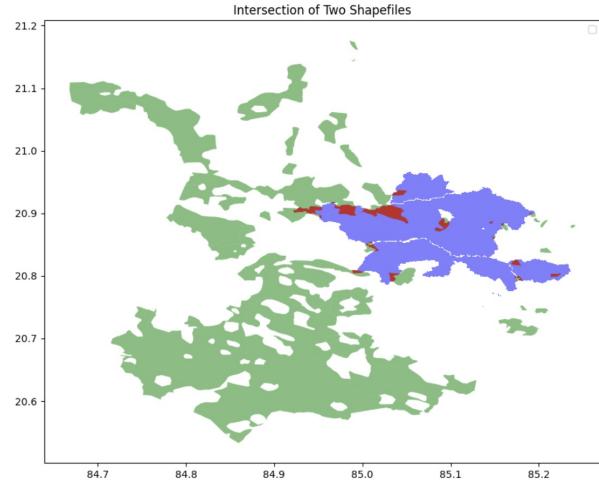


Figure 4.5: Intersection in Python

### 1.D Plotting the intersection with the LLM

Now that we know how the final plot should look like, we will query the LLM to plot the intersection

```
Output the LLM's response
print("Response from LLM:")
print(response['choices'][0]['message']['content'])
```

Unlike what we expect, the LLM is not able to plot the intersection. Instead it gives us the Python code to plot the intersection.

```
Analysis from LLM:
To compute and plot the intersection of two geographical datasets with geometries, here's a general process you can follow, assuming you are using a programming language such as Python:
1. *Load the datasets: Read the shapefiles or any geographical data formats into Geodataframes.
2. *Compute the intersection: Use the 'intersection' method to find the overlapping areas between the two datasets.
3. *Plot the results: Create a visualization to show the original geometries and the intersections.

Here is an example of how you might implement this in Python:
python
import geopandas as gpd
import matplotlib.pyplot as plt

Load shapefiles assuming they are in GeoPandas format
Replace path_to_dataset1.shp and path_to_dataset2.shp with actual file paths
dataset1 = gpd.read_file('path_to_dataset1.shp')
dataset2 = gpd.read_file('path_to_dataset2.shp')

Ensure both datasets are in the same Coordinate Reference System (CRS)
if dataset1.crs != dataset2.crs:
 dataset1 = dataset1.to_crs(dataset2.crs)

Compute the intersection
intersection = gpd.overlay(dataset1, dataset2, how='intersection')

Plot the datasets
fig, ax = plt.subplots(figsize=10, 8)
Plot Dataset 1
dataset1.plot(ax=ax, color='blue', alpha=0.5, label='Dataset 1')
Plot Dataset 2
dataset2.plot(ax=ax, color='green', alpha=0.5, label='Dataset 2')
Plot Intersections
intersection.plot(ax=ax, color='red', alpha=0.7, label='Intersection')

Plot styling
plt.title('Intersection of Two Geospatial Datasets')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
plt.show()

Key Steps:
- Make sure to replace the paths with the actual paths to your datasets.
- Make sure to correct the CRS if the two datasets are not using the same system.
- Adjust the colors and labels as necessary for clarity.

By running the above code, you will compute the intersection of the two datasets and create a plot that visually represents both the original geometries and their intersection.
```

Figure 4.6: Failure of the LLM to plot the intersection.

## 2 Plotting and analyzing the intersection of GeoJSON files

For our last analysis, we will try to plot the intersection of GEOJSON files of angul boundaries

and micro-watersheds(mws) of angul. These files can be found in the Github repository I mentioned in the Introduction chapter.

## 2.A Defining the functions

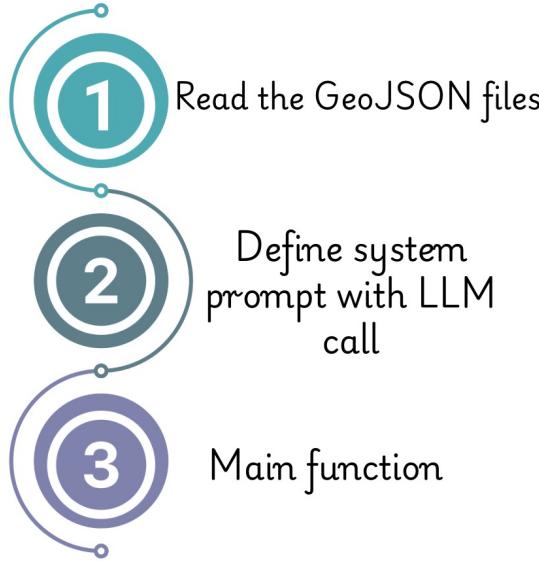


Figure 4.7: Function pathway

As we can see in Fig.4.7 we will define functions in such a manner that they read the GeoJSON files, defined the system propmts with the LLM call and then execute the code in the main function. Figs.4.8 and 4.9 contain the code snippets for the same.

```

Function to read GeoJSON from file paths
def read_geojson(file_path):
 with open(file_path, "r") as f:
 return json.load(f)

Function to call the LLM with GeoJSON files
def call_llm_with_geojson_files(file_path1, file_path2):
 # Read the GeoJSON files
 geojson1 = read_geojson(file_path1)
 geojson2 = read_geojson(file_path2)

 # Prepare the GeoJSON data as strings
 geojson1_str = json.dumps(geojson1)
 geojson2_str = json.dumps(geojson2)

```

Figure 4.8: function to read the GeoJSON files

```

Define the system prompt
prompt = """
You are an assistant that can execute Python code. The user has provided two GeoJSON files.
Your task is:
1. Parse the GeoJSON files.
2. Compute their intersection using GeoPandas.
3. Plot GeoJSON 1 in blue, GeoJSON 2 in green, and their intersection in red using Matplotlib.
4. Return the plot as output.

GeoJSON 1:
{geojson1_str}
GeoJSON 2:
{geojson2_str}

Perform the task and show the resulting plot.
"""

Call the LLM with the prompt
response = openai.ChatCompletion.create(
 model="gpt-4o-mini",
 messages=[
 {"role": "system", "content": "You can execute Python code to answer questions."},
 {"role": "user", "content": prompt},
]
)

Return the LLM response
return response

```

Figure 4.9: System prompt with LLM call

## 2.B Plotting the intersection

Now we will try to plot the intersection by leveraging the LLM.

```

if __name__ == "__main__":
 # Paths to the GeoJSON files
 file_path1 =
 "/content/anugul_purunakot.geojson"
 file_path2 =
 "/content/mws_anugul_purunakot.geojson"

 # Call the function
 response =
 call_llm_with_geojson_files(file_path1,
 file_path2)

 print(response["choices"][0]["message"]
 ["content"])

```

As we can see in Fig.4.10 the LLM throws an error that our request exceeds the token limits for GPT-4o-mini.

```

 File "/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py", line 37
 response = openai.ChatCompletion.create(
 ^~~~~
 File "/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py", line 763
 stream_error = stream and "error" in resp.data
 ^~~~~
 File "/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py", line 764
 if stream_error or not 200 <= rcode < 300:
 ^~~~~
 File "/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py", line 765
 raise self.handle_error_response(
 ^~~~~
 File "/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py", line 766
 rbody, rcode, resp.data, rheaders, stream_error=stream_error
 ^~~~~
 File "/usr/local/lib/python3.10/dist-packages/openai/api_requestor.py", line 767
)

RateLimitError: Request too large for gpt-4o-mini in organization org-wfisfhq0CvudGZcVrC9 on tokens per min (TPM): Limit 200000, Requested 429854. The input or output tokens must be reduced in order to run successfully. Visit https://platform.openai.com/account/rate-limits to learn more.

```

Figure 4.10: Failure to plot intersection

---

```
Limit the size of the data sent to the LLM
summary_gdf1 =
 ↪ gdf1[['geometry']].head(5).to_json()
summary_gdf2 =
 ↪ gdf2[['geometry']].head(5).to_json()
summary_intersection =
 ↪ intersection[['geometry']].head(5).to_json()
 ↪ if not intersection.empty else "[]"
```

---

Instead, let us try to summarize the GeoJSON file using the LLM and then plot the intersection using that data. This way, we will be reducing the tokens and may fall into the safe limit.

```
Response from LLM:
To visualize the provided GeoJSON datasets, we can create a plot using appropriate libraries.

```python
import geopandas as gpd
import matplotlib.pyplot as plt

# Load the GeoJSON datasets
dataset_1 = {
    "type": "FeatureCollection",
    "features": [
        # Add Dataset 1 features here as per the provided summary
    ]
}

dataset_2 = {
    "type": "FeatureCollection",
    "features": [
        # Add Dataset 2 features here as per the provided summary
    ]
}

intersection_data = {
    "type": "FeatureCollection",
    "features": [
        # Add Intersection features here as per the provided summary
    ]
}

# Create GeoDataFrames
gdf1 = gpd.GeoDataFrame.from_features(dataset_1['features'])
gdf2 = gpd.GeoDataFrame.from_features(dataset_2['features'])
gdf_intersection = gpd.GeoDataFrame.from_features(intersection_data['features'])

# Plotting
fig, ax = plt.subplots(figsize=(15, 10))

# Plot Dataset 1 with blue color
gdf1.plot(ax=ax, color='blue', edgecolor='k', alpha=0.5, label='Dataset 1')

# Plot Dataset 2 with green color
gdf2.plot(ax=ax, color='green', edgecolor='k', alpha=0.5, label='Dataset 2')

# Plot Intersection with red color
gdf_intersection.plot(ax=ax, color='red', edgecolor='k', alpha=0.8, label='Intersection')

# Customize the plot
plt.title('GeoJSON Data Visualization')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend()
plt.grid()

# Show the plot
plt.show()
```

```

Figure 4.11: Failure to plot intersection

As in the case of the shapefiles, the LLM again returns the Python code to plot the intersection between these two files. This is visible in Fig.4.11.

# Chapter 5

## Bypassing the errors

Now that we have identified the specific errors encountered with each data type, we can systematically address these issues with a straightforward remedy. The root problem, as observed across the various file formats—be it Excel files, shapefiles, or GeoJSON files—is consistent: the model fails to directly display the desired images or graphs. Instead, it generates and provides the complete code required for us to create these visualizations independently.

This limitation suggests that while the model is able to generate correct and actionable code, it cannot execute the code to produce and display the resulting visualizations directly. This realization simplifies our task considerably. We can focus on a simpler solution: extracting the code provided in the output, executing it, and ensuring that the resulting graphs or images are correctly saved and displayed as required.

### 1 Pie chart from Excel file

---

```
Path to the Excel file

file_path = '/content/Masalia_data.xlsx'
question = "Can you plot a pie chart of hill slope, plane area, ridge area, slopy area and
→ valley area for UID 12_307609"
last_output = main(file_path, question)

print(last_output)
```

---

This code, as we know from Fig.2.8, produces the Python code to plot the pie chart. But, in the above code, we have modified the original code in such a way that we save the output in a function named last\_output. Now we will write the code for a second function to extract python code from the output and execute it.

```
import re

def execute_code_from_previous_output():
 global last_output

 if not last_output:
 print("No output available to process.")
 return

 # Defines a pattern to match Python code blocks within triple backticks
 code_blocks = re.findall(r"```python(.*)?```", last_output, re.DOTALL)

 if not code_blocks:
```

```

print("No Python code blocks found in the output.")
return

for i, code in enumerate(code_blocks, start=1):
 print(f"Executing Code Block {i}:\n{'='*40}\n{code.strip()}\n{'='*40}")
 try:
 # Execute the code block
 exec(code.strip(), globals())
 except Exception as e:
 print(f"Error executing Code Block {i}: {e}")

Executes the Python code from the last_output

execute_code_from_previous_output()

```

---

This code will be the Master code for each of the data types. Now all we have to do is define last output in each data type and then this code will execute the relevant python code from it.

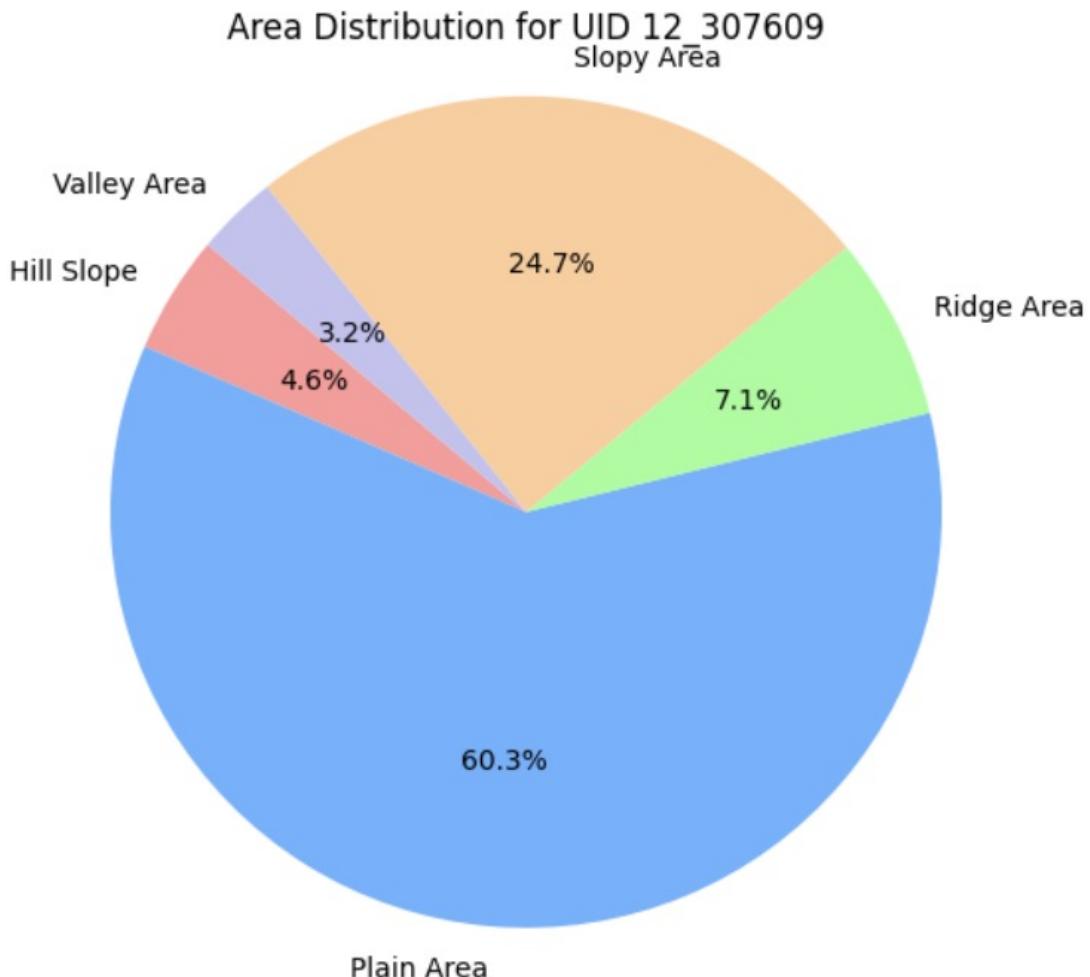


Figure 5.1: Successful creation of Pie Chart

As we can see in Fig.5.1 this code successfully plots the pie chart as we requested.

## 2 Plotting the intersection of shapefiles

Again, we will modify the original code to store the output.

```
last_output = response['choices'][0]['message']
['content']
print("Analysis from LLM:")
print(last_output)
```

We will then execute the Master code we defined in the previous section.

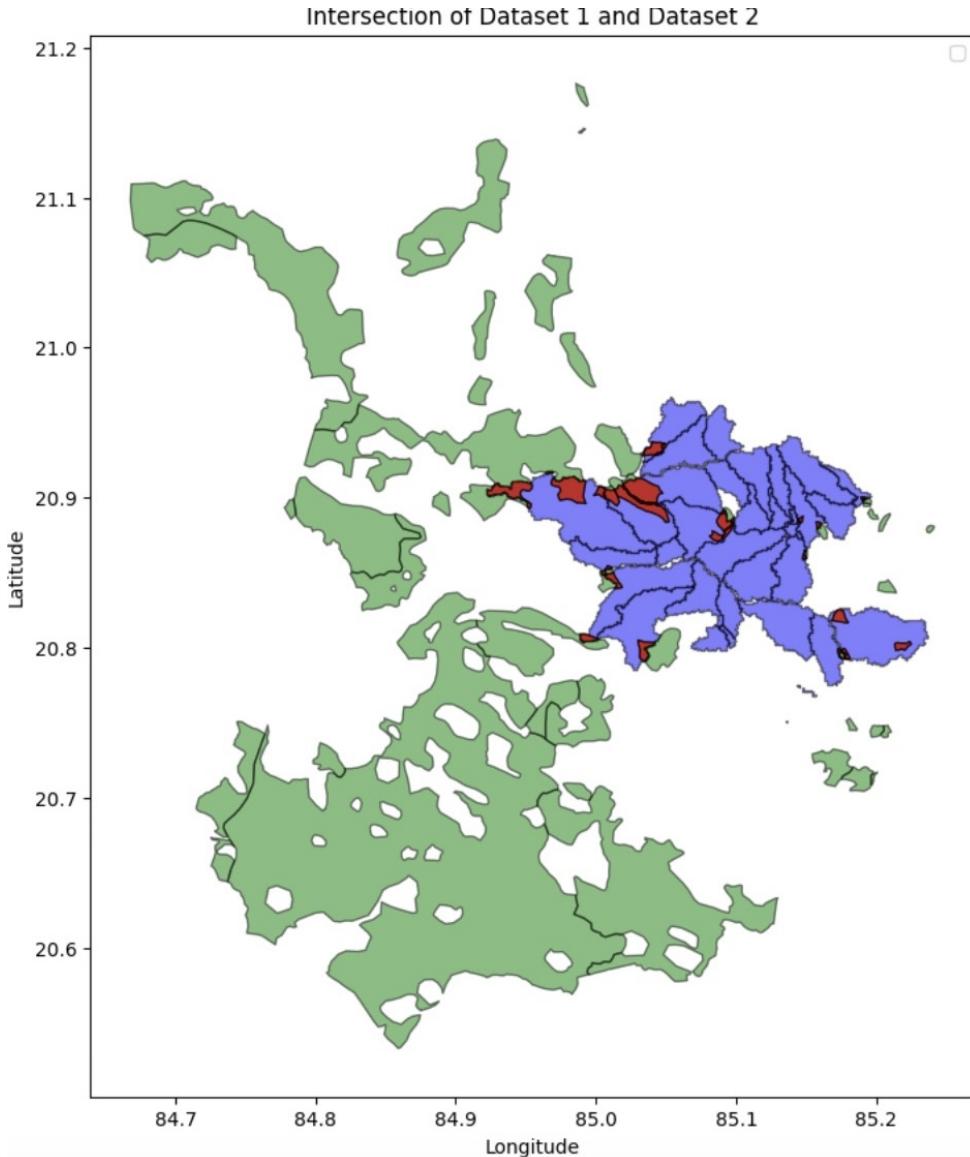


Figure 5.2: Successful Intersection of Shapefiles

As we can see Fig.5.2 matches Figs.4.4 and 4.5 perfectly with angul\_angul in green, angul\_drought in purple and the intersection in red.

## 3 Plotting the intersection of GeoJSON files

Once again, we simply have to modify the original code.

---

```
Save the output from the LLM
last_output = response['choices'][0]['message']
['content']
print("Response from LLM:")
print(last_output)
```

---

Once again, we will execute the Master code after this code cell.

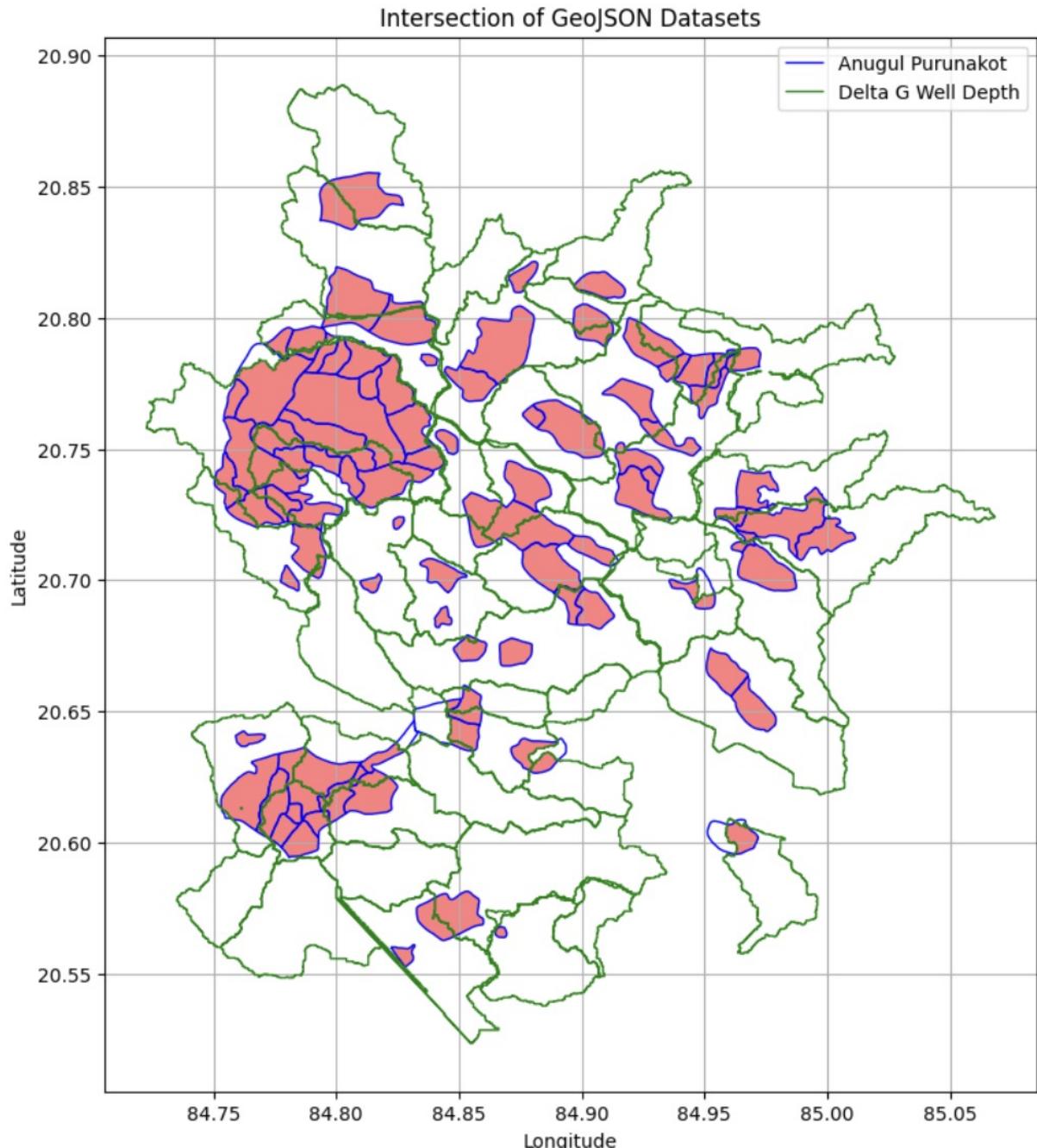


Figure 5.3: Successful Intersection of GeoJSON files

As we can see in Fig.5.3 the intersection has successfully been plotted.

# Chapter 6

## Conclusions

This research focused on overcoming the challenges associated with integrating and processing diverse datasets—such as Excel files, shapefiles, and GeoJSON files—while ensuring the seamless visualization of outputs. A major insight gained during this work was that while the LLM consistently generated accurate and efficient code for data manipulation and visualization, it lacked the capability to directly execute the code and display the desired outputs, such as graphs and maps.

To address this limitation, a systematic approach was developed to extract the generated code, execute it using Python, and store the resulting visualizations in an organized manner. This streamlined the process, bridging the gap between code generation and result utilization, thereby significantly enhancing the workflow.

The relevance of these findings for the backend of a chatbot is paramount. By automating the execution and visualization processes, the backend becomes capable of directly delivering actionable insights and visual outputs to end users. This eliminates the need for users to engage in additional steps, thereby improving accessibility and usability. Such advancements make the chatbot more versatile and effective in handling real-world tasks, ranging from geospatial analysis to dynamic data visualization. The work conducted here lays the foundation for robust, user-friendly systems capable of simplifying complex operations.

In conclusion, this internship has not only enhanced proficiency in pioneering AI methodologies but also cultivated a deeper understanding of their societal impact, setting the stage for future innovations in geo-spatial analysis and technology-enabled community empowerment.