



explain the code line by line and if there is error in the execution of the code then remove it by updating the code snippet and state the reason for the update



To assist you effectively, please provide the specific code snippet you'd like me to explain, or let me know if you need help with a certain type of code. Once I have that, I can break it down line by line, and if there are errors, I will address them, update the code accordingly, and explain why the update is necessary.



Generate Beautiful map using GeoPandas
Imports and dataset

```
!pip install geopandas
```

```
gpd.__version__
```

```
'0.13.0'
```

```
import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd
```

```
df = gpd.read_file('/content/district.shp')
df.head()
```

```
OBJECTID  DISTRICT  ProvinceSCode  DCode  SDD  SUM_length  SUM_area
Shape_Leng Shape_Area  DistLabel  Name_Nep  geometry
0  1  ACHHAM  Sudur Pashchim  7  69  769  1702.678726  1702.678726  2.669525
0.157592  Achham अछाम  POLYGON ((81.46795 29.27188, 81.46795 29.27188...
1  2  ARGHAKHANCHI  5  5  51  551  1239.074609  1239.074609  2.379111  0.113578
Arghakhanchi  अर्घाखाँची  POLYGON ((83.05092 28.11160, 83.05104 28.11159...
2  3  BAGLUNG  Gandaki  4  45  445  1835.741748  1835.741748  2.885574  0.168958
Baglung बागलुङ  POLYGON ((83.10729 28.63221, 83.10708 28.63187...
3  4  BAITADI  Sudur Pashchim  7  74  774  1496.314027  1496.314027  2.499107  0.138875
Baitadi  बैतडी  POLYGON ((80.79590 29.68829, 80.79590 29.68829...
4  5  BAJHANG  Sudur Pashchim  7  68  768  3464.299347  3464.299347  3.156896
0.322441  Bajhang बझाङ  POLYGON ((81.09080 30.05419, 81.09085 30.05420...
Basic map with df.plot()
```

```
df.plot()
```

<Axes: >

```
df.plot(column="SUM_area")
```

<Axes: >

```
df.plot(column="SUM_area", cmap='RdYlBu', legend=True, vmin=1000, vmax=8000,
        legend_kwds={"label": "Shape Area"})
```

<Axes: >

Interacting with axis and label

```
fig, ax1 = plt.subplots(1,1,figsize=(10,7))
ax1.set_title("Nepal Shape Area")
plot = df.plot(column="SUM_area", ax=ax1, cmap='hsv', legend=True, vmin=1000, vmax=8000,
               legend_kwds={"label": "Shape Area"})
ax1.set_xlabel("Longitude")
ax1.set_ylabel("Latitude")
```

```
# ax1.set_xticks([])
# ax1.set_yticks([])
```

```
Text(92.09722222222221, 0.5, 'Latitude')
```

Automation on maps

```
provinces = df['Province'].unique()
provinces
```

```
array(['Sudur Pashchim', '5', 'Gandaki', '2', 'Bagmati', '1', 'Karnali'],
      dtype=object)
```

```
fig, ax = plt.subplots(2,4,figsize=(15,10))
axis = ax.flatten()
```

```
axis[7].remove()
print(axis)
for i, (ax, province) in enumerate(zip(axis,provinces)):
```

```
    df_province = df[df['Province']==province]
```

```
    ax.set_title(f'{province} Province (Shape Area)')
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', edgecolor='#000000')
    ax.set_xticks([])
    ax.set_yticks([])
```

[<Axes: > <Axes: > <Axes: > <Axes: > <Axes: > <Axes: > <Axes: > <Axes: >]

```
import numpy as np

# getting start with matplotlib suplots
fig, ax = plt.subplots(2,4,figsize=(20,12))
axis = ax.flatten()

# remove the last unnecessary axis
axis[7].remove()

# for loop based on axis and province
for i, (ax, province) in enumerate(zip(axis,provinces)):

    # select the province
    df_province = df[df['Province']==province]

    # calculate geometry bounds
    x1, y1, x2, y2 = df_province.geometry.total_bounds

    # zoom to the selected feature
    max_range = np.array([x2-x1, y2-y1]).max()/2.0
    ax.set_xlim((x1+x2)*0.5-max_range-0.05, (x1+x2)*0.5+max_range+0.05)
    ax.set_ylim((y1+y2)*0.5-max_range-0.05, (y1+y2)*0.5+max_range+0.05)

    # label feature on map
    df_province.apply(lambda x: ax.annotate(text=x.DISTRICT, xy=x.geometry.centroid.coords[0],
ha='center', size=7),axis=1)

    # plot province shape area
    ax.set_title(f"{province} Province (Shape Area)")
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn')
    ax.set_xticks([])
    ax.set_yticks([])

# Remove unnecessary white spaces
fig.tight_layout()
fig.savefig('/content/final_map.jpg', dpi=500, bbox_inches='tight')
```

```
import numpy as np

# getting start with matplotlib suplots
fig, ax = plt.subplots(2,4,figsize=(20,12))
axis = ax.flatten()

# remove the last unnecessary axis
axis[7].remove()

# for loop based on axis and province
```

```
for i, (ax, province) in enumerate(zip(axis, provinces)):

    # select the province
    df_province = df[df['Province']==province]

    # calculate geometry bounds
    x1, y1, x2, y2 = df_province.geometry.total_bounds

    # zoom to the selected feature
    max_range = np.array([x2-x1, y2-y1]).max()/2.0
    ax.set_xlim((x1+x2)*0.5-max_range-0.05, (x1+x2)*0.5+max_range+0.05)
    ax.set_ylim((y1+y2)*0.5-max_range-0.05, (y1+y2)*0.5+max_range+0.05)

    # label feature on map
    df_province.apply(lambda x: ax.annotate(text=x.DISTRICT, xy=x.geometry.centroid.coords[0],
ha='center', size=7), axis=1)

    # plot province shape area
    ax.set_title(f"{province} Province (Shape Area)")
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', vmin=500, vmax=8000)
    ax.set_xticks([])
    ax.set_yticks([])

# Remove unnecessary white spaces

plot.legend()
fig.tight_layout()
fig.savefig('/content/final_map.jpg', dpi=500, bbox_inches='tight')

WARNING:matplotlib.legend:No artists with labels found to put in legend. Note that artists whose
label start with an underscore are ignored when legend() is called with no argument.

Thank you
Please make sure to suscribe to my YouTube Channel: GeoDev
```



Let's break down the code you provided, explain it line by line, and also fix any issues that may arise during its execution.

1. Install GeoPandas:

```
python

!pip install geopandas
```

This command installs the `geopandas` package, which is necessary for working with geospatial data.

2. Check GeoPandas version:

```
python

gpd.__version__
```

This checks the installed version of GeoPandas, which is `'0.13.0'` in your case.

3. Import necessary libraries:

```
python

import geopandas as gpd
import matplotlib.pyplot as plt
import pandas as pd
```

- `'geopandas'`: For handling geospatial data.
- `'matplotlib.pyplot'`: For plotting data.
- `'pandas'`: Used for handling regular tabular data.

4. Load the shapefile:

```
python

df = gpd.read_file('/content/district.shp')
df.head()
```

- `'gpd.read_file()'` reads the shapefile (`'shp'`) containing the geographic data.
- `'df.head()'` shows the first few rows of the data.

If there is any issue with the file path or format, ensure the shapefile is in the correct directory and properly formatted.

5. Basic map plotting:

```
python

df.plot()
```

This plots the basic map without any specific styling.

6. Map based on the "SUM_area" column:

python

```
df.plot(column="SUM_area")
```

This plots the map, coloring the regions according to the values in the `"SUM_area"` column.

7. Customized map with color map (`cmap``):

python

```
df.plot(column="SUM_area", cmap='RdYlBu', legend=True, vmin=1000, vmax=8000, legend_kwds=
{"label": "Shape Area"})
```

- This uses the `'RdYlBu'` color map.
- Adds a legend with limits set by `vmin`` and `vmax`` between 1000 and 8000.
- `legend_kwds`` specifies the label of the legend.

8. Customizing axis and labels:

python

```
fig, ax1 = plt.subplots(1, 1, figsize=(10, 7))
ax1.set_title("Nepal Shape Area")
plot = df.plot(column="SUM_area", ax=ax1, cmap='hsv', legend=True, vmin=1000, vmax=8000, l
egend_kwds={"label": "Shape Area"})
ax1.set_xlabel("Longitude")
ax1.set_ylabel("Latitude")
```

- Creates a customized figure and sets the title, x-label, and y-label.
- `figsize=(10, 7)`` defines the size of the plot.

9. Automating map creation for each province:

python

```
provinces = df['Province'].unique()
fig, ax = plt.subplots(2, 4, figsize=(15, 10))
axis = ax.flatten()
axis[7].remove() # Remove the last empty axis
```

- `df['Province'].unique()`` retrieves the unique provinces.
- `plt.subplots(2, 4)`` creates a grid of subplots (2 rows and 4 columns).
- `axis[7].remove()`` removes an extra axis in the plot.

10. Plotting each province's map:

python

```
for i, (ax, province) in enumerate(zip(axis, provinces)):
    df_province = df[df['Province'] == province] # Filter the province
    ax.set_title(f"{province} Province (Shape Area)")
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', edgecolor='#000000')
    ax.set_xticks([]) # Remove x-ticks
    ax.set_yticks([]) # Remove y-ticks
```

- This loop iterates over each province and plots the map for that province, with custom titles.
- `edgecolor='#000000'` adds black borders around the polygons.

11. Zooming to the selected province:

python

```
for i, (ax, province) in enumerate(zip(axis, provinces)):
    df_province = df[df['Province'] == province]
    x1, y1, x2, y2 = df_province.geometry.total_bounds
    max_range = np.array([x2 - x1, y2 - y1]).max() / 2.0
    ax.set_xlim((x1 + x2) * 0.5 - max_range - 0.05, (x1 + x2) * 0.5 + max_range + 0.05)
    ax.set_ylim((y1 + y2) * 0.5 - max_range - 0.05, (y1 + y2) * 0.5 + max_range + 0.05)
    df_province.apply(lambda x: ax.annotate(text=x.DISTRICT, xy=x.geometry.centroid.coords
[0], ha='center', size=7), axis=1)
    ax.set_title(f"{province} Province (Shape Area)")
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', vmin=500, vmax=8000)
    ax.set_xticks([]) # Remove x-ticks
    ax.set_yticks([]) # Remove y-ticks
```

- This zooms into each province using the bounds of its geometry.
- Labels each district with its name at the centroid of its shape.

12. Remove unnecessary white spaces and save the figure:

python

```
fig.tight_layout()
fig.savefig('/content/final_map.jpg', dpi=500, bbox_inches='tight')
```

- `tight_layout()` removes unnecessary white spaces between subplots.
- `savefig()` saves the final map as an image.

13. Add a legend:

python

```
plot.legend()
```

You received the following warning:

text

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.
```

This warning occurs because the `plot.legend()` was called without specifying which elements to include in the legend. To fix this, update the code to correctly add labels.

Updated Code for Legend:

python

```
df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', vmin=500, vmax=8000, legend=True, legend_kwds={"label": "Shape Area"})
```

This update adds the legend directly in the plot by setting `legend=True` and specifying `legend_kwds`.

Key Takeaways:

- If the legend does not appear or is missing labels, ensure that the `legend` and `legend_kwds` are properly defined within the plotting function.
- Always verify that file paths are correct when reading shapefiles.

By running the updated code, you should get a fully functional and beautifully formatted map!



i think you missed some codes to explain



You're right! Let me go over the missing parts step by step and ensure that all code is fully explained.

1. Load the dataset:

python

```
df = gpd.read_file('/content/district.shp')
df.head()
```

- `gpd.read_file()` reads the shapefile that contains geographic data for different districts.
- `df.head()` shows the first five rows of the loaded geospatial data, which includes columns such as `OBJECTID`, `DISTRICT`, `Province`, `geometry`, etc.

2. Basic map plotting:


```
python
```

```
df.plot()
```

- This plots a basic map of all the regions in the shapefile. The default behavior is to draw polygons based on the `geometry` column, without applying any styling.

3. Plotting by "SUM_area":

```
python
```

```
df.plot(column="SUM_area")
```

- This generates a map where the regions are colored according to the values in the `SUM_area` column (which might represent the area of each district). No customization is applied yet beyond this.

4. Applying color map, legend, and value limits:

```
python
```

```
df.plot(column="SUM_area", cmap='RdYlBu', legend=True, vmin=1000, vmax=8000, legend_kwds={"label": "Shape Area"})
```

- `cmap='RdYlBu'`: This applies the Red-Yellow-Blue colormap to visualize differences in the `SUM_area` column.
- `legend=True`: Displays a legend indicating the meaning of the colors.
- `vmin=1000, vmax=8000`: These define the minimum and maximum values for the color scale.
- `legend_kwds={"label": "Shape Area"}`: Specifies the label for the legend.

5. Customizing the map with axis labels and title:

```
python
```

```
fig, ax1 = plt.subplots(1, 1, figsize=(10, 7))
ax1.set_title("Nepal Shape Area")
plot = df.plot(column="SUM_area", ax=ax1, cmap='hsv', legend=True, vmin=1000, vmax=8000, legend_kwds={"label": "Shape Area"})
ax1.set_xlabel("Longitude")
ax1.set_ylabel("Latitude")
```

- `plt.subplots(1, 1, figsize=(10, 7))`: Creates a figure with one subplot, with a size of 10x7 inches.
- `ax1.set_title("Nepal Shape Area")`: Sets the title of the map.
- The map is plotted on `ax1` using the 'hsv' colormap (Hue-Saturation-Value).
- `ax1.set_xlabel()` and `ax1.set_ylabel()` label the axes as "Longitude" and "Latitude", respectively.

The commented-out lines:

python

```
# ax1.set_xticks([])
# ax1.set_yticks([])
```

These would remove the tick marks on the x and y axes if uncommented.

6. Automating map creation for each province:

python

```
provinces = df['Province'].unique()
provinces
```

- `df['Province'].unique()` extracts all unique values from the 'Province' column.
- `provinces` now holds the array of province names, e.g., `['Sudur Pashchim', '5', 'Gandaki', '2', 'Bagmati', '1', 'Karnali']`.

7. Creating subplots for each province:

python

```
fig, ax = plt.subplots(2, 4, figsize=(15, 10))
axis = ax.flatten()
axis[7].remove() # Remove the last empty axis
```

- `plt.subplots(2, 4, figsize=(15, 10))`: Creates a 2x4 grid of subplots, where each subplot will hold a map for one province.
- `axis = ax.flatten()`: Flattens the 2D array of subplots into a 1D array so that we can loop over them.
- `axis[7].remove()`: Since there are only 7 provinces and 8 subplots, this removes the extra axis (the 8th subplot).

8. Looping over provinces to create maps:

python

```
for i, (ax, province) in enumerate(zip(axis, provinces)):
    df_province = df[df['Province'] == province] # Filter the data for the current province
    ax.set_title(f"{province} Province (Shape Area)") # Set title
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', edgecolor='#000000')
# Plot each province
ax.set_xticks([]) # Remove x-ticks
ax.set_yticks([]) # Remove y-ticks
```

- `zip(axis, provinces)` pairs each subplot with a unique province.

- `df[df['Province'] == province]` filters the data to include only rows where the 'Province' column matches the current province.
- `plot = df_province.plot(...)` creates a map for the current province and displays it on the respective axis (`ax`).
- `ax.set_xticks([])` and `ax.set_yticks([])` remove tick marks to make the plot cleaner.

9. Adjusting axis limits and annotating districts:

python

```
for i, (ax, province) in enumerate(zip(axes, provinces)):
    df_province = df[df['Province'] == province]
    x1, y1, x2, y2 = df_province.geometry.total_bounds # Get the bounding box of the province
    max_range = np.array([x2 - x1, y2 - y1]).max() / 2.0
    ax.set_xlim((x1 + x2) * 0.5 - max_range - 0.05, (x1 + x2) * 0.5 + max_range + 0.05) # Adjust x limits
    ax.set_ylim((y1 + y2) * 0.5 - max_range - 0.05, (y1 + y2) * 0.5 + max_range + 0.05) # Adjust y limits

    # Label each district in the map by placing the text at the centroid of the district
    df_province.apply(lambda x: ax.annotate(text=x.DISTRICT, xy=x.geometry.centroid.coords[0], ha='center', size=7), axis=1)

    ax.set_title(f"{province} Province (Shape Area)")
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', vmin=500, vmax=8000)
# Create the map
ax.set_xticks([]) # Remove x-ticks
ax.set_yticks([]) # Remove y-ticks
```

- `geometry.total_bounds` returns the bounds (x1, y1, x2, y2) of the geometry of the filtered province.
- The limits of the plot are adjusted using `ax.set_xlim()` and `ax.set_ylim()` so the map zooms in to fit the province's bounds.
- `df_province.apply(lambda x: ax.annotate(...))`: Annotates each district by placing its name at the centroid of its shape (`geometry.centroid`).

10. Saving the figure:

python

```
fig.tight_layout() # Remove unnecessary white spaces between subplots
fig.savefig('/content/final_map.jpg', dpi=500, bbox_inches='tight') # Save the final figure as an image
```

- `tight_layout()` ensures there is no overlap between subplots.
- `savefig()` saves the image with high resolution (`dpi=500`).

11. Fixing the legend issue:

You encountered the following warning:

text

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.
```

The issue arises because `plot.legend()` was called without specifying what elements to include in the legend. Here's the update to fix it:

Updated Code for Adding Legend:

python

```
plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', vmin=500, vmax=8000, legend=True, legend_kwds={"label": "Shape Area"})
```

- This ensures that the legend appears correctly in the plot by setting `legend=True` and using `legend_kwds` to specify the label for the legend.

Now all parts of the code are explained in detail! If you run the updated code, you should get beautiful province-level maps with correct legends, titles, and district labels.



explain the concept



The concept behind the code revolves around **visualizing geospatial data** using Python libraries like **GeoPandas** and **Matplotlib**. The overall goal is to load geographical information, create maps with it, and customize those maps by coloring them based on some numerical data (like the area of districts), labeling the regions, and adjusting the visual aspects for better clarity.

Here's a breakdown of the key concepts involved:

1. Geospatial Data and Shapefiles

- **Shapefile (.shp):** It is a widely used geospatial data format for representing geometries like points, lines, and polygons. Each shape can have attributes, such as a district name, population, or area, and this additional data is stored alongside the geometric information.
- **Geometry Column:** In the shapefile, each row corresponds to a geographic region (like a district), and the `geometry` column contains the polygon (set of coordinates) defining the region's boundary on the map.

2. GeoPandas

GeoPandas extends Pandas, making it possible to work with geospatial data, like reading shapefiles and manipulating geographic information.

- **GeoDataFrame:** This is a core object in GeoPandas, which functions similarly to a Pandas DataFrame but includes a `geometry` column that stores spatial data.
 - **Geometry:** The `geometry` column defines the shapes of regions on a map, such as a district's boundaries represented as polygons.
 - **Attributes:** Columns like `DISTRICT`, `Province`, and `SUM_area` are the attributes of these geometries.

For example, this part of the code:

```
python

df = gpd.read_file('/content/district.shp')
```

Reads a shapefile and stores the data (geometries and attributes) in a GeoDataFrame `df`.

3. Mapping and Plotting

Plotting Maps: The core idea is to visualize geographic regions (like districts or provinces) on a map and apply customizations, such as coloring the regions based on the `SUM_area` (shape area) or adding labels (district names).

- **Basic Plotting:** GeoPandas makes it easy to plot geospatial data:

```
python

df.plot()
```

This command generates a simple map of the districts using their geometries.

- **Customized Maps:** You can use a column like `"SUM_area"` to apply color gradients based on the area of each district:

```
python

df.plot(column="SUM_area", cmap='RdYlBu', legend=True)
```

Here, different regions are colored using the Red-Yellow-Blue (`RdYlBu`) colormap, where each color corresponds to a range of area values.

4. Matplotlib Integration

Matplotlib is a widely used library for generating plots and is integrated with GeoPandas for further customizations, like adding titles, axes labels, or even creating subplots.

- **Subplots:** The code creates multiple maps in a grid layout, with each map representing a different province. This allows for side-by-side comparison of provinces:

```
python

fig, ax = plt.subplots(2, 4, figsize=(15, 10))
```

This line creates a 2x4 grid of subplots, where each subplot will display a province's map.

5. Filtering Data for Specific Regions

GeoPandas allows you to filter data based on specific conditions. For example:

```
python

df_province = df[df['Province'] == province]
```

This filters the GeoDataFrame `df` to include only rows where the 'Province' column matches a given province. It allows the code to plot individual maps for each province.

6. Color Maps and Legends

Color maps (`cmap`) are used to visually distinguish between regions based on numeric values. In this code, regions are colored based on their area:

```
python

df.plot(column="SUM_area", cmap='RdYlBu', legend=True, vmin=1000, vmax=8000)
```

- `cmap='RdYlBu'`: Uses the Red-Yellow-Blue colormap to color regions.
- `vmin` and `vmax`: These set the range for the color scale (from 1000 to 8000).
- `legend=True`: Displays a legend indicating the meaning of the colors.

7. Adding Labels to the Map

You can annotate maps by placing text at specific coordinates. In this case, the code annotates the map with district names:

```
python

df_province.apply(lambda x: ax.annotate(text=x.DISTRICT, xy=x.geometry.centroid.coords[0],
ha='center', size=7), axis=1)
```

- `x.geometry.centroid.coords[0]`: This gets the central point (centroid) of each district's polygon to place the label in the middle of the district.

8. Zooming and Adjusting Axes

The code also zooms into each province's map by adjusting the axis limits (`xlim` and `ylim`) to focus on a specific region:

python

```
ax.set_xlim((x1 + x2) * 0.5 - max_range - 0.05, (x1 + x2) * 0.5 + max_range + 0.05)
```

This calculates the bounding box of the region and adjusts the view to fit the specific province.

9. Saving the Final Map

After creating the map, it can be saved as an image:

python

```
fig.savefig('/content/final_map.jpg', dpi=500, bbox_inches='tight')
```

This saves the figure with a resolution of 500 dots per inch (DPI) and removes extra white space around the image (`bbox_inches='tight'`).

Key Concepts Summary:

- **GeoPandas:** Handles geospatial data, allows reading from shapefiles, and plotting geographic shapes.
- **Matplotlib:** Provides additional customization (axes labels, titles, subplots, etc.) to enhance the map's appearance.
- **Color Maps:** Visualize the numerical data (like district area) by coloring regions differently.
- **Annotations:** Labels the districts directly on the map by finding their centroid (center).
- **Subplots:** Allows for creating multiple maps in a grid layout to show different regions (provinces) side-by-side.

In essence, this code demonstrates how to load geospatial data from a shapefile, customize it by coloring based on numeric attributes (such as area), zoom into specific regions, and annotate the map with labels using Python's GeoPandas and Matplotlib libraries.



write a term paper for this notebook



Term Paper: Visualizing Geospatial Data Using GeoPandas and Matplotlib

1. Introduction

Geospatial data, which represents the geographic locations and attributes of objects, has become crucial for decision-making in various domains such as urban planning, environmental monitoring, and policy-making. Effective visualization of this data is key to uncovering patterns and relationships that may not be evident from raw numerical data alone. The Python libraries GeoPandas and Matplotlib provide powerful tools for visualizing and analyzing geospatial data in an accessible and customizable way.

This term paper discusses the practical use of these libraries to create detailed maps of geographic regions, focusing on province-level divisions, district-level visualization, and automated map generation based on spatial attributes such as area. We explore the concepts, techniques, and steps involved in loading geospatial data from shapefiles, plotting basic and customized maps, and adding useful visual enhancements like color scales and labels.

2. Background and Literature Review

Geospatial data visualization has a long history, dating back to the earliest forms of map-making. With the advent of Geographic Information Systems (GIS) and digital cartography, geospatial visualization has expanded its reach, allowing for intricate analysis of spatial patterns and relationships. Traditionally, GIS software like ArcGIS or QGIS has been used to handle such tasks. However, in recent years, open-source Python libraries such as GeoPandas and Matplotlib have democratized access to geospatial visualization, enabling more customized and programmatic approaches.

GeoPandas, an extension of the popular data manipulation library Pandas, allows Python users to easily read, write, and process geospatial data in formats such as shapefiles, GeoJSON, and more. It integrates seamlessly with Matplotlib, a Python library for 2D plotting, to create static, interactive, and publication-quality maps. Together, these tools provide a robust environment for researchers and data scientists to visualize spatial data without the need for expensive proprietary software.

3. Methodology

3.1 Dataset and Tools

The dataset used in this term paper is a shapefile containing the geographic boundaries of districts within a country. Shapefiles are a common geospatial data format that store both the shape (geometry) of spatial features and their attributes, such as district names or areas. The tools employed for analysis and visualization include:

- **GeoPandas:** For reading the shapefile and performing geographic operations.
- **Matplotlib:** For visualizing the geographic data and customizing the appearance of maps.

3.2 Loading the Data

The shapefile is loaded using the `read_file` function from GeoPandas:

```
python

df = gpd.read_file('/content/district.shp')
```


This command reads the shapefile and stores the geographic data in a `GeoDataFrame`, a specialized data structure that combines spatial data with tabular data, allowing for easy manipulation and visualization.

The dataset contains columns such as ``DISTRICT``, ``Province``, and ``SUM_area``, which represent each district's name, the province it belongs to, and the district's area, respectively.

3.3 Basic Map Visualization

After loading the data, a basic map can be created using the ``plot()`` method:

```
python

df.plot()
```

This command generates a simple plot of the district boundaries based on their geometries. Although informative, this basic map lacks customization, such as color-coding based on attributes or labels, which will be added in subsequent steps.

3.4 Color-Coding Based on Attributes

One of the key functionalities of GeoPandas is the ability to color regions based on their attribute values. In this case, the districts are color-coded by their area (``SUM_area``), using a gradient color map (``RdYlBu``):

```
python

df.plot(column="SUM_area", cmap='RdYlBu', legend=True, vmin=1000, vmax=8000, legend_kwds=
{"label": "Shape Area"})
```

Here, the ``cmap`` parameter specifies the color map to use, while ``vmin`` and ``vmax`` define the value range for the color gradient. The result is a map where districts with larger areas are displayed in one color (e.g., red), and smaller districts in another (e.g., blue). The ``legend`` option is enabled to provide a reference for interpreting the colors.

3.5 Adding Titles and Labels

In most maps, providing context through titles and labels is essential for clarity. Using Matplotlib's ``subplots()`` function, we can add custom titles and axis labels to the map:

```
python

fig, ax1 = plt.subplots(1, 1, figsize=(10, 7))
ax1.set_title("Nepal Shape Area")
plot = df.plot(column="SUM_area", ax=ax1, cmap='hsv', legend=True, vmin=1000, vmax=8000, l
egend_kwds={"label": "Shape Area"})
ax1.set_xlabel("Longitude")
ax1.set_ylabel("Latitude")
```

This block of code generates a map with a title ("Nepal Shape Area") and labeled axes ("Longitude" and "Latitude"). Customizing maps in this way makes them more informative and easier to interpret.

3.6 Automating Map Creation for Each Province

To provide a more granular view of the data, we automate the process of creating maps for each province in the dataset. First, we extract the unique province names:

```
python

provinces = df['Province'].unique()
```

Next, we use Matplotlib's `subplots()` to create multiple maps (one for each province) arranged in a grid:

```
python

fig, ax = plt.subplots(2, 4, figsize=(15, 10))
axis = ax.flatten()
axis[7].remove() # Remove the last empty axis

for i, (ax, province) in enumerate(zip(axis, provinces)):
    df_province = df[df['Province'] == province] # Filter data by province
    ax.set_title(f"{province} Province (Shape Area)") # Set title
    plot = df_province.plot(column="SUM_area", ax=ax, cmap='RdYlGn', edgecolor='#000000')
    ax.set_xticks([]) # Remove x-ticks
    ax.set_yticks([]) # Remove y-ticks
```

This loop iterates over the unique provinces, creating individual maps for each one, while adjusting visual properties such as tick marks, titles, and color schemes.

3.7 Customizing Maps with Annotations and Zoom

For enhanced readability, the map includes district names, placed at the centroid of each district. The following code zooms in on each province and annotates the map with district names:

```
python

df_province.apply(lambda x: ax.annotate(text=x.DISTRICT, xy=x.geometry.centroid.coords[0],
ha='center', size=7), axis=1)
```

This command uses the `annotate` method to place the district name at the centroid of its geometry, ensuring that the labels are clearly positioned within each district's boundaries.

Additionally, the code calculates the bounding box for each province and adjusts the zoom level to focus on that region:

python

```
x1, y1, x2, y2 = df_province.geometry.total_bounds
ax.set_xlim((x1 + x2) * 0.5 - max_range - 0.05, (x1 + x2) * 0.5 + max_range + 0.05)
ax.set_ylim((y1 + y2) * 0.5 - max_range - 0.05, (y1 + y2) * 0.5 + max_range + 0.05)
```

3.8 Saving the Final Map

The final composite map, consisting of multiple subplots for each province, is saved as a high-resolution image:

python

```
fig.savefig('/content/final_map.jpg', dpi=500, bbox_inches='tight')
```

This command saves the map with a resolution of 500 dots per inch (DPI), making it suitable for printing or sharing in reports.

4. Results and Discussion

The resulting maps provide an informative visualization of district boundaries, their relative areas, and their positions within each province. The color-coding based on the `SUM_area` attribute effectively highlights differences in district sizes. Moreover, the automated map creation for each province allows for easy comparison between regions. By annotating the districts with their names and zooming in on each province, the map becomes more user-friendly and readable.

The use of subplots helps in creating a comparative visualization of multiple geographic regions in a single figure, a feature particularly useful for policymakers, urban planners, or researchers conducting regional analysis.

However, there are challenges to consider. Large maps with many subplots can become cluttered, and the automatic zooming and label placement might occasionally overlap or be poorly positioned in smaller or irregularly shaped districts. Fine-tuning the layout and placement of labels and adjusting zoom levels can improve the clarity of the final map.

5. Conclusion

This paper demonstrates the power and flexibility of Python's GeoPandas and Matplotlib libraries in visualizing geospatial data. The tools allow for easy manipulation and analysis of geographic datasets, as well as customizable visual outputs. By integrating these libraries, we can generate detailed, informative maps that are both aesthetically pleasing and highly functional.

GeoPandas provides a powerful way to handle geospatial data programmatically, making it an essential tool for researchers and analysts working in fields where geographic data is central. Matplotlib complements GeoPandas by enabling high-quality, customizable visualizations. Together,

these libraries are excellent for creating detailed maps with custom features such as automated province-level visualization, color gradients, and annotations.

Future work may involve incorporating interactive elements using libraries like Folium or Plotly, enabling users to zoom and pan on maps or click on regions for additional information.

6. References

- GeoPandas Documentation. (n.d.). Retrieved from <https://geopandas.org/>
- Matplotlib Documentation. (n.d.). Retrieved from <https://matplotlib.org/>
- QGIS Project. (n.d.). Shapefiles. Retrieved from <https://qgis.org/en/site/>

This term paper outlines the concepts and steps involved in using GeoPandas and Matplotlib to visualize geospatial data. It provides a comprehensive overview of the methodology, results, and implications of these visualizations for researchers and practitioners.

ChatGPT can make mistakes. Check important info.