Script for Exploratory Data Analysis (EDA) and Reporting of Geospatial Datasets

Overview

This script automates the processing and reporting of geospatial datasets stored in a directory. The script:

1. **Finds geospatial files** (`.shp` , `.gpkg` , `.csv`) in the dataset folder.
2. **Analyses each file** to extract metadata (geometry type, record count, CRS, etc.).
3. **Generates an interactive map** for each geospatial file.
4. **Captures a screenshot** of the map using a headless browser.
5. **Creates individual reports** (`.docx`) for each file, which include:
   - Metadata about the dataset.
   - Folder contents (only in the first report).
   - A dataset analysis section.
   - A map screenshot.
   - A sample of the dataset in GeoJSON format.
6. **Merges multiple reports** into a single document while keeping:
   - The **main report title** (`{dataset} Analysis Report`) only in the first document.
   - The **folder contents** section only once.
   - A **level-1 heading** for each shapefile analysis.

Step-by-Step Breakdown

1. Import Required Libraries

The script imports several Python libraries:

- **os**: Manages file paths and sizes.
- **geopandas & pandas**: Reads geospatial data from `.shp` , `.gpkg` , and `.csv` files.
- **folium**: Generates interactive maps.
- **asyncio & pyppeteer**: Captures map screenshots using a headless browser.
- **docx**: Creates `.docx` reports.
- **docx.shared**: Formats document elements such as images.

---

2. Define the `process_dataset` Function

This function:

- Takes a dataset name as input.
- Finds geospatial files in the dataset's folder.

- Iterates through each file, processing and analysing it.
- Saves results in individual reports.
- Merges reports if multiple files exist.

It first defines the folder structure for the dataset.

---

3. Identify Geospatial Files

The script scans the dataset directory and identifies files with the extensions `.shp`, `.gpkg`, or `.csv`. If no files are found, the script terminates.

---

4. Analyse Folder Contents

Before analysing the geospatial files, the script:

- Extracts the **size of each file**.
- Calculates the **total size of the folder**.
- Stores this information for inclusion in the final report.

This information is **only added to the first report**.

---

5. Process Each Geospatial File

For each geospatial file found, the script:

1. **Loads the file into memory**.
2. **Extracts metadata** such as:
   - Geometry type.
   - Record count.
   - Feature columns.
   - Coordinate reference system (CRS).
   - File size.
3. **Saves a sample of the dataset** in GeoJSON format (first 50 records).
4. **Generates an interactive map** for the dataset.
5. **Captures a screenshot** of the map using a headless Chrome browser.
6. **Creates a `.docx` report** summarising the dataset.

---

6. Loading the Geospatial File

- **Shapefiles (`.shp`)** and **GeoPackages (`.gpkg`)** are read using `geopandas`.
- **CSV files (`.csv`)** are read using `pandas`.
- If an error occurs while loading a file, the script logs the error and moves to the next file.

---

7. Extracting Metadata

The script retrieves key details about the dataset:

- **Geometry Type**: The type of spatial objects (e.g., `Point`, `Polygon`).
- **Record Count**: The number of features in the dataset.
- **Feature Columns**: A list of attribute fields.
- **CRS (Coordinate Reference System)**: The spatial reference used.
- **File Size**: The size of the dataset in bytes.

This metadata is included in the report.

---

8. Saving a Sample as GeoJSON

The script saves the first 50 records of each dataset in GeoJSON format. This allows a preview of the dataset structure.

---

9. Generating an Interactive Map

The script creates an interactive map using `folium` and saves it as an HTML file.

---

10. Capturing a Screenshot of the Map

Since `folium` maps are interactive, they must be converted into images. The script:

1. **Opens the saved HTML map** in a headless Chrome browser using `pyppeteer`.
2. **Waits for the map to fully load**.
3. **Takes a screenshot** and saves it as a PNG file.

If an error occurs (e.g., if Chrome is not installed or accessible), the script logs the error and continues.

---

11. Creating an Individual Report

A `.docx` report is generated for each geospatial file. Each report contains:

- **Main Title** (`{dataset} Analysis Report`, only in the first report).
- **Folder Contents** (only in the first report).
- **A Section for the Current File** (`{geospatial_file} Analysis`).
- **Metadata** (geometry type, record count, CRS, etc.).
- **Dataset Plot** (map screenshot).
- **GeoJSON Sample Data** (preview of the dataset in GeoJSON format).

If a dataset does not contain an image or GeoJSON sample, placeholders are added.

---

12. Merging Reports

If multiple geospatial files exist, the script:

1. **Merges all individual reports into a single document**.

2. **Keeps the main report title and folder contents only in the first report**.
3. **Adds a level-1 heading before each dataset's analysis** to structure the document properly.

The final document is saved in the dataset folder.

Final Output

After execution, the script generates:

- **Individual Reports** (`.docx` files) for each geospatial dataset.
- **A Merged Report** (if multiple files exist), structured as follows:
    1. **Title (** `{dataset} Analysis Report` **)**.
    2. **Folder Contents** (only once).
    3. **Shapefile Analyses**, each separated by a heading.
    4. **Dataset Plots** (screenshots).
    5. **GeoJSON Samples**.

Each dataset analysis section is properly formatted for clarity.

Key Features

✅ **Handles multiple geospatial files** and generates structured reports.
✅ **Automatically extracts and saves key dataset information**.
✅ **Creates an interactive map for each dataset**.
✅ **Captures high-quality map screenshots**.
✅ **Formats reports professionally** in `.docx` format.
✅ **Merges reports into a single document with proper headings**.

How to Run the Script

1. **Ensure all dependencies are installed**:
    - `pip install geopandas pandas folium pyppeteer python-docx`
2. **Set the dataset name**:
    - `dataset = "your_dataset_name"`
3. **Run the function**:
    - `await process_dataset(dataset)`

The script will automatically process all geospatial files and generate reports.

In [5]:
```
dataset = ""
```

In [ ]:
```
import os
import geopandas as gpd
import pandas as pd
import folium
```

```python
import asyncio
from pyppeteer import launch
from docx import Document
from docx.shared import Inches, Pt

async def process_dataset(dataset):
    """
    Processes all shapefiles in a dataset folder by analysing each, generating indi
    and then merging them into a single consolidated DOCX report.

    Args:
        dataset (str): Name of the dataset folder.
    """

    dataset_folder = f"datasets/{dataset}"
    folder_path = f"{dataset_folder}/files"

    def find_geospatial_files(folder_path):
        """Finds geospatial files (.shp, .gpkg, .csv) in the folder."""
        return [file for file in os.listdir(folder_path) if file.endswith(('.shp',

    print(f"Looking for geospatial files in: {folder_path}")
    geospatial_files = find_geospatial_files(folder_path)

    if not geospatial_files:
        print("No geospatial files found.")
        return

    print("Geospatial files found:", geospatial_files)

    report_files = []

    def analyse_folder(folder_path):
        """Analyses the folder and returns file sizes."""
        details = {file: f"Size: {os.path.getsize(os.path.join(folder_path, file))}
                   for file in os.listdir(folder_path)}
        details["Total Folder Size"] = f"{sum(os.path.getsize(os.path.join(folder_p
        return details

    folder_description = analyse_folder(folder_path)

    for index, geospatial_file in enumerate(geospatial_files):
        print(f"\nProcessing: {geospatial_file} ({index + 1}/{len(geospatial_files)

        geojson_output_file = f"{dataset_folder}/{dataset}_{geospatial_file}_sample
        data_plot_file = f"{dataset_folder}/{dataset}_{geospatial_file}_plot.png"
        report_output_file = f"{dataset_folder}/{dataset}_{geospatial_file}_analysi

        def load_geospatial_file(folder_path, file_name):
            """Loads a geospatial file (.shp, .gpkg, .csv) into a GeoDataFrame."""
            file_path = os.path.join(folder_path, file_name)
            try:
                if file_name.endswith(('.shp', '.gpkg')):
                    return gpd.read_file(file_path, engine="pyogrio", on_invalid="i
                elif file_name.endswith('.csv'):
                    return pd.read_csv(file_path)
                else:
                    raise ValueError("Unsupported file format.")
            except Exception as e:
                print(f"Error loading file: {e}")
                return None

        gdf = load_geospatial_file(folder_path, geospatial_file)
        if gdf is None:
            continue

        def analyse_geospatial_file(gdf, geojson_output_file):
            """Analyses geospatial data and saves a sample as GeoJSON."""
            try:
                gdf.head(50).to_file(geojson_output_file, driver="GeoJSON")
                print(f"Sample GeoJSON saved: {geojson_output_file}")
            except Exception as e:
                print(f"Error saving GeoJSON: {e}")
                return None

            return {
                "geometry_type": ', '.join(gdf.geom_type.unique()),
                "record_count": len(gdf),
                "features": ', '.join(gdf.columns),
                "crs": str(gdf.crs) if gdf.crs else "Unknown",
                "file_size": os.path.getsize(os.path.join(folder_path, geospatial_f
                "geojson_file": geojson_output_file
            }

        geospatial_data = analyse_geospatial_file(gdf, geojson_output_file)
        if geospatial_data is None:
            continue

        CHROME_PATH = "C:/Program Files/Google/Chrome/Application/chrome.exe"

        m = gdf.iloc[:1].explore()
        map_html_path = f"{dataset_folder}/map_{geospatial_file}.html"
        m.save(map_html_path)

        async def capture_map():
            """Captures a screenshot of an interactive map using a headless browser
            try:
                browser = await launch(headless=True, executablePath=CHROME_PATH, a
                page = await browser.newPage()
                await page.goto(f"file://{os.path.abspath(map_html_path)}")
                await asyncio.sleep(3)
                await page.screenshot({"path": data_plot_file, "fullPage": True})
                print(f"Map saved: {data_plot_file}")
            except Exception as e:
                print(f"Error capturing map: {e}")
            finally:
                await browser.close()

        await capture_map()

        def generate_report(folder_data, geospatial_data, output_file, image_file,
            """
            Generates a report summarising the dataset, including metadata, folder
            a map plot, and a GeoJSON sample.

            Args:
                folder_data (dict): Folder contents.
                geospatial_data (dict): Geospatial metadata.
                output_file (str): Report file path.
                image_file (str): Dataset plot path.
                first_report (bool): Whether this is the first report (keeps the ma
            """
            doc = Document()

            if first_report:
                doc.add_heading(f"{dataset} Analysis Report", 0)

                # Folder contents only in the first report
                doc.add_heading('Folder Contents:', level=1)
                for file, details in folder_data.items():
                    doc.add_paragraph(f"{file}: {details}", style='List Bullet')

            doc.add_heading(f"{geospatial_file} Analysis:", level=1)
            doc.add_paragraph(f"Geometry Type: {geospatial_data['geometry_type']}",
            doc.add_paragraph(f"Number of Records: {geospatial_data['record_count']
            doc.add_paragraph(f"Features: {geospatial_data['features']}", style='Li
            doc.add_paragraph(f"CRS: {geospatial_data['crs']}", style='List Bullet'
            doc.add_paragraph(f"File Size: {geospatial_data['file_size']} bytes", s

            doc.add_heading("Dataset Plot", level=1)
            if os.path.exists(image_file):
                doc.add_picture(image_file, width=Inches(6))
                doc.add_paragraph(f"Figure: First-row dataset visualisation ({image
            else:
                doc.add_paragraph("No image available for visualisation.")

            doc.add_heading("GeoJSON Sample Data", level=1)
            if os.path.exists(geospatial_data["geojson_file"]):
                with open(geospatial_data["geojson_file"], 'r') as geojson_file:
                    doc.add_paragraph("Sample GeoJSON content:")
                    doc.add_paragraph(geojson_file.read(2000))
            else:
                doc.add_paragraph("No GeoJSON file available.")

            doc.save(output_file)
            print(f"Report saved: {output_file}")
            return doc

        first_report = (index == 0)
        generate_report(folder_description, geospatial_data, report_output_file, da
        report_files.append(report_output_file)

    if len(report_files) > 1:
        final_report_path = f"{dataset_folder}/{dataset}_final_analysis_report.docx
        merged_doc = Document(report_files[0])

        for report in report_files[1:]:
            doc_to_merge = Document(report)
            merged_doc.add_page_break()
            doc_heading = f"{geospatial_files[report_files.index(report)]} Analysis
            merged_doc.add_heading(doc_heading, level=1)
            for element in doc_to_merge.paragraphs:
                merged_doc.add_paragraph(element.text)

        merged_doc.save(final_report_path)
        print(f"\nFinal merged report saved: {final_report_path}")

# Example usage:
await process_dataset(dataset)
```

In [ ]: