# Interactive Mapping Application for Nepal

Guragain Prabesh, Bhattrai Ankit

Abstract
This project provides a powerful, interactive geospatial visualization tool specifically for Nepal's administrative boundaries. Built with Streamlit for an intuitive interface, Plotly for dynamic mapping, and GeoPandas for robust spatial data handling, it allows users to effortlessly explore national, provincial, and district borders. Users gain full control over boundary visibility and styling, enabling a tailored viewing experience. A key innovation is the seamless integration of custom data: users can upload their own CSV files, and the map intelligently overlays this information. Numerical data is displayed as choropleth maps (heatmaps), while categorical information is represented by customizable icons placed precisely on the relevant districts. The tool offers extensive personalization, allowing users to modify layer names, tooltip labels, colors, and boundary details for both the base map and custom layers. Features like fuzzy string matching ensure accurate data-to-district correlation, even with minor spelling variations. With multi-layer plotting, users can compare different datasets, enhancing visualization and analytical capabilities. The system supports persistent sessions, remembering map views and settings. This versatile tool provides a scalable foundation for critical applications in public policy, disaster management, and demographic research, facilitating more informed decision-making across Nepal. This simpler way of plotting geodata of Nepal will enable more people to create geospatial visualizations of Nepali data.

Keyword: Map Visualizer, Nepal, Streamlit, Plotly, GeoPandas, web application.

## I. Introduction

This project delivers an intuitive, interactive map of Nepal. It allows users to easily view and explore the country's national, provincial, and district boundaries. Beyond just showing borders, its core purpose is to help people visualize their own information directly on the map. Users can upload simple data files, and the map will automatically display this information, whether it's numbers shown with colors or specific points marked with icons, on the relevant districts.

This kind of tool is essential in Nepal because understanding geographical data is critical for so many activities, from planning development projects and responding to disasters to managing local governance and distributing resources effectively. By making it easy for anyone to overlay their specific data onto an accurate map, this project helps transform raw information into clear visual insights, supporting better decision-making and a deeper understanding of trends across the country.

Geospatial visualization is critical for understanding regional data patterns in Nepal's complex topography. This application addresses limitations in existing tools by providing:

- Boundary toggling (districts/provinces)
- Custom data integration without GIS expertise
- Responsive design for field researchers
- Compare and visualize numerical data against categorical data to identify correlations related to geographical regions.

## II. System Architecture

### *A. Component Diagram*

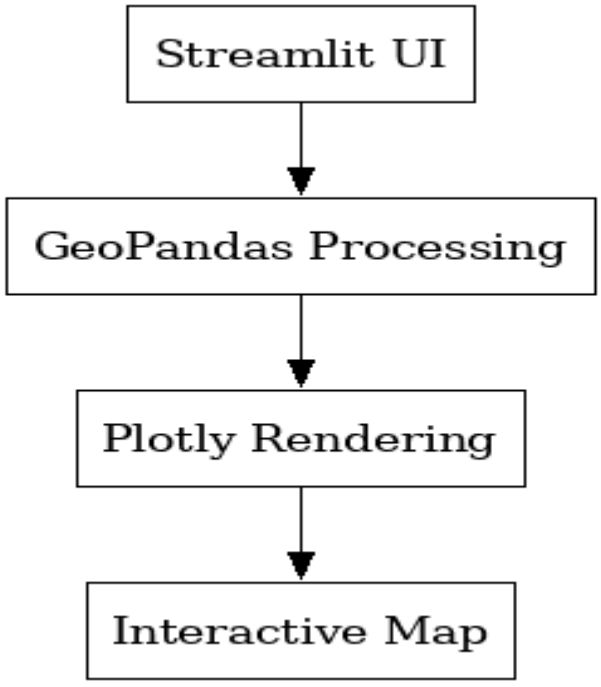


*Figure 1 Core workflow diagram.*

This above diagram illustrates the core components of the Nepal interactive map project. In the diagram, the Streamlit UI serves as the primary user interface, through which all user interactions and controls are managed. This interface sends commands and data to the GeoPandas Processing module,

which is responsible for handling and manipulating the geographical data, such as the district and province boundaries of Nepal. Following processing, the data is passed to Plotly Rendering, where it is transformed into interactive graphical elements. Finally, Plotly Rendering outputs the complete Interactive Map that users see and engage with.

The next diagram below shows a separate flow for custom user data. A User CSV file, containing location-based information, is uploaded by the user. This data then undergoes Fuzzy Matching, a process that intelligently attempts to align the locations specified in the user's CSV with the official district names within the map's geospatial data. Once matched, the data proceeds to Data Integration, where it is merged with the geographical data being processed by GeoPandas. This integrated data then feeds into the Plotly Rendering stage, allowing the custom information to be displayed as an overlay on the Interactive Map, alongside the base geographical features.

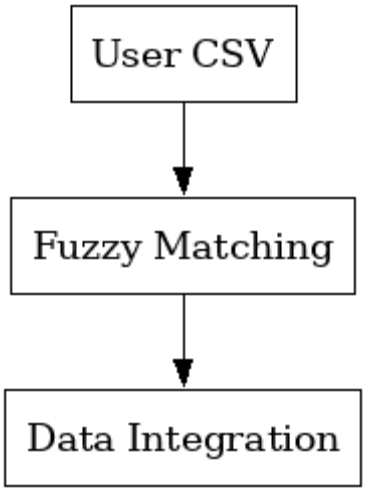


*Figure 2 User data loading flow.*

### B. *Technical Stack*

- UI: Streamlit, CSS (Markdown)
- Geospatial: GeoPandas, Shapely
- Visualization: Plotly, ColorScales
- Data Processing: Pandas, FuzzyWuzzy

## III. Implementation Details

### A. *Boundary Rendering*

#### 1) *District/Province Visualization:*

```
def add_border_trace(geom, fig, line_color,
line_width):
    if isinstance(geom, MultiPolygon):
        for poly in geom.geoms:
            lons, lats =
poly.exterior.coords.xy
            fig.add_trace(go.Scatter(x=lons,
y=lats, mode='lines',

line=dict(color=line_color,
width=line_width)))
```

- Precision: 0.5px district borders (`dimgray`)
- Hierarchy: 1.5px province borders (`black`)

### B. *Data Integration*

#### 1) *CSV Processing:*

1. Validate column structure (`Location` + data column)
2. Apply fuzzy matching (≥80% similarity threshold)
3. Merge with official district geometries
4. Classify data type (numeric/categorical)

#### 2) *Matching Algorithm:*

The administrative names of some districts are different from the common name so to make it user-freidly we implemented word matching algorithm.

```
from fuzzywuzzy import process
def get_match(loc):
    match, score = process.extractOne(loc,
official_districts)
    return match if score >= 80 else None
```

### C. *Visualization Modes*

#### 1) *Numeric Data*

- Gradient scaling:

  `interp(value, [min_val, max_val], [^2])`

- Dynamic colormap (Heatmap):

  `sample_colorscale([[0,'white'],[1,selecte d_color]], normalized_val)`

*2) Categorical Data*

- Icon placement at district centroids:

```
fig.add_trace(go.Scatter(x=[centroid.x],
y=[centroid.y],
                         mode='text',
text=selected_icon))
```

- Anti-overlap jitter:

```
±0.01° coordinate randomization
```

*D. Session Management*

*1) Persistent States:*
*Preserves: Zoom level, pan position, uploaded files, UI settings*

```
if 'map_view' not in st.session_state:
    st.session_state.map_view = {
        'zoom': 8.5,
        'center': {'lat': 28.3949, 'lon':
84.1240}
    }
```

## IV. Feature Analysis

*A. Core Functionality*

*1) Interactive Nepal Map: Displays an interactive map of Nepal showing national, provincial, and district boundaries using Plotly and GeoPandas.*

*2) Customizable Base Layer:*
- Toggles for showing/hiding district and province borders.
- Option to color provinces individually with a color picker and control their visibility.

*3) CSV Data Upload:*
*Allows users to upload a CSV file to add custom data layers to the map.*
- CSV Formatting Rules: Provides clear guidelines on the required CSV format (two columns: Location for district names and a data column).
- Sample CSV Links: Includes links to sample CSV files for user reference.

*4) Dynamic Data Layer Controls:*
*For each uploaded CSV, users can:*
- Edit the layer's display name and tooltip label.
- Toggle layer visibility.
- Numeric Data Visualization: If the data column is numeric, it colors districts based on the values using a gradient from white to a user-selected color.
- Categorical/String Data Visualization: If the data column is categorical or string, it displays user-selected icons at the centroid of the relevant districts.

*5) Smart String Matching (FuzzyWuzzy):*
Automatically matches user-provided location names in the CSV to official district names, with a toast notification for successful matches (if FuzzyWuzzy is available).

*6) Customizable Tooltips:*
- Option to show/hide the district name in tooltips.
- Option to show/hide values from custom data layers in tooltips.

*7) Persistent Map View: Remembers the last zoomed/panned map view across interactions.*

*8) Error Handling: Includes basic error handling for file not found and other unexpected errors.*

*B. Performance Measures*
- Rendering Speed: <3s for 77 districts
- Data Capacity: Can load 10+ CSV layers
- Layer: 1 numeric data /district + more than 5 categorical data
- Accuracy: district match rate with fuzzy logic

## V. Usage Scenarios

*A. 5.1. Public Health Deployment (Numeric Data)*

```
Location,Cholera_Cases
Kathmandu,142
Rupandehi,89
Kailali,67
```

*Output*: Heatmap highlighting outbreak clusters

*B. Infrastructure Planning (Categorical Data)*

```
Location,Project_Status
Mustang,Construction
Sindhupalchok,Proposed
```

*Output*: Icon-based status markers (🏠/ ⛲ )

### C. Tourism Hotspot analysis (Numeric + Categorical Data)

```
Csv1: Toutist spot
Location,Spot
Mustang,Muktinath Temple
Bhaktaput,Bhaktaput Durbar Square
Csv2: Torist arrival
Location,Spot
Mustang,123456
Bhaktaput,76542
```

*Output*: Icons over heatmap

## VI. Error Handling System

### A. GeoJSON File Not Found:

If the `districts.geojson` or `provinces.geojson` files are not found in the `geo_data` folder, a `FileNotFoundError` is caught, and an informative error message is displayed to the user, suggesting they check the file paths.

### B. CSV File Reading Errors:

When a user uploads a CSV file, the code attempts to read it using `pd.read_csv`. If any `Exception` occurs during this process (e.g., malformed CSV, incorrect encoding), an error message is shown to the user, indicating the file that failed and the specific error encountered.

### C. Missing Value Column in CSV:

After a CSV is uploaded, the code checks if a column other than 'Location' (which is expected to be the data column) can be identified. If no such column is found, an error message is displayed, prompting the user to ensure one column is named 'Location'.

### D. No Matching Locations in Custom Data:

If, after attempting to merge user-uploaded data with the district GeoDataFrame, no matching locations are found (especially after fuzzy matching), a warning message is displayed to the user for that specific data layer.

### E. General Unexpected Errors:

A broad `except Exception as e` block is used to catch any other unexpected errors that might occur during the map generation process. This block displays a general error message and then provides the full exception traceback for debugging purposes.

## VII. Future Enhancements and Limitations

### A. Future Enhancements

- **Advanced Data Layer Styling for Numeric Data:** Implement options for users to select from a wider range of pre-defined Plotly color scales (e.g., 'Viridis', 'Plasma') and define custom min/max values for the color scale, rather than solely relying on data's min/max.
- **Manual Location Mapping:** If the fuzzy string matching fails to provide an accurate match or returns no match for a user's Location in the CSV, allow users to manually map their provided location names to the official district names.
- **Data Aggregation Options:** For custom data layers, provide users with choices on how to aggregate data if multiple entries in their CSV correspond to the same district. This could include options like sum, average, or count for numeric data, or listing all values for categorical data.
- **Search and Zoom Functionality:** Integrate a search bar that allows users to find specific districts by name and automatically zoom the map to the selected district's boundaries.

### B. Limitations

- **Performance for Large Datasets and API Integration:** The application's reliance on loading and processing GeoJSON files and potentially large user-uploaded CSVs can lead to slower performance and responsiveness. This inherent latency could make real-time updates or seamless integration with external APIs (which often require quick data retrieval and rendering) impractical due to high loading times.
- **No Offline Functionality:** The application is built using Streamlit, which requires a persistent connection to a server to run. There is no provision for offline use or client-side caching of map data, limiting its utility in environments with unreliable internet connectivity.
- **Geographic Scope Limited to Nepal:** The base model of the application is specifically for visualizing Nepal's administrative boundaries (provinces and districts). It cannot, as of now, visualize other countries or regions without significant modifications to include their respective GeoJSON data and adjust the map's initial view.

## VIII. Conslucsion

This application establishes a simpler solution for geospatial analysis in Nepal, combining cartographic precision with accessible data integration. It dynamic and easy to use system provides:

- **Policy Support**:
  Visual evidence for resource allocation
- **Research Flexibility**:
  Custom data layering without GIS dependencies
- **Technical Extensibility**:
  Foundation for advanced geospatial modules

## IX. Appendices

### A. Data Dictionary

| COLUMN | DESCRIPTION | CONSTRAINTS |
|---|---|---|
| `LOCATION` | Official district names | Matching algorithm |
| `[VALUE]` | User-defined metric | Numeric/string |

### B. Sample CSV Structure

```
Location,Population_Density
Jhapa,501.2
Morang,398.7
Sunsari,312.4
```

### C. Requirements

```
streamlit==1.28.0
plotly==5.18.0
geopandas==0.14.0
fuzzywuzzy==0.18.0
python-Levenshtein==0.23.0
```

### D. Start Application

```
cd Nepal_visualizer
pip install requirements.txt
streamlit run main.py
```

A browser will appear in screen running in localhost.

### E. Output files

1. Final Base Visualization


*Figure 3 Final projejct base layer*

2. Map with Province color on


*Figure 4 Project screenshot with province border ON*

3. Custom Data Visualizer

Map with census.csv


*Figure 5 Screenshot of applicatoin with custom numeric data*

4. Plotting multiple values

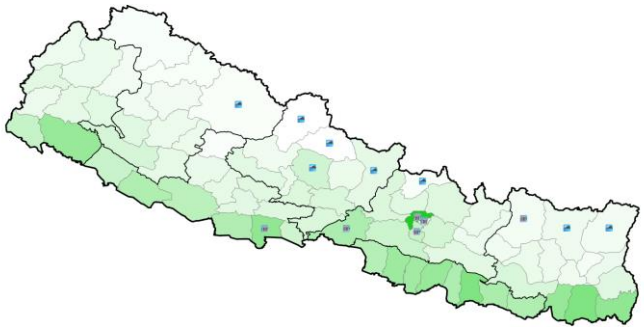

*Figure 6 Polot of tourist vs toursim spot vs UNESCO site*