CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE · INDIA

LAB 6

By
Nishant Rodrigues
2348045
5 MSc Data Science
(Geospatial Analysis)

Introduction

Spatial data analysis is a critical approach for uncovering patterns and relationships in geospatial datasets, aiding in decision-making and policy development. This study focuses on understanding spatial dependencies and clustering patterns by analyzing soil composition data across Indian districts. By employing techniques like Moran's I autocorrelation for spatial relationships and clustering algorithms such as DBSCAN and K-means, this study aims to identify spatial clusters, detect outliers, and uncover insights into the spatial distribution of soil properties.

Objectives

1. **Moran's I Autocorrelation**:
   - Compute a spatial weights matrix and create a Moran scatterplot to visualize spatial relationships.
   - Calculate global spatial autocorrelation using Moran's I to measure spatial dependency.
   - Detect spatial clusters and outliers using local spatial autocorrelation methods like LISA.
2. **Spatial Clustering**:
   - Develop a workflow for spatial clustering using algorithms such as DBSCAN and K-means.
   - Perform spatial clustering on point data to identify and analyze geospatial patterns.
   - Visualize and interpret spatial clusters using tools like QGIS to derive actionable insights.

Dataset Description

The study utilizes two datasets for spatial analysis and clustering:

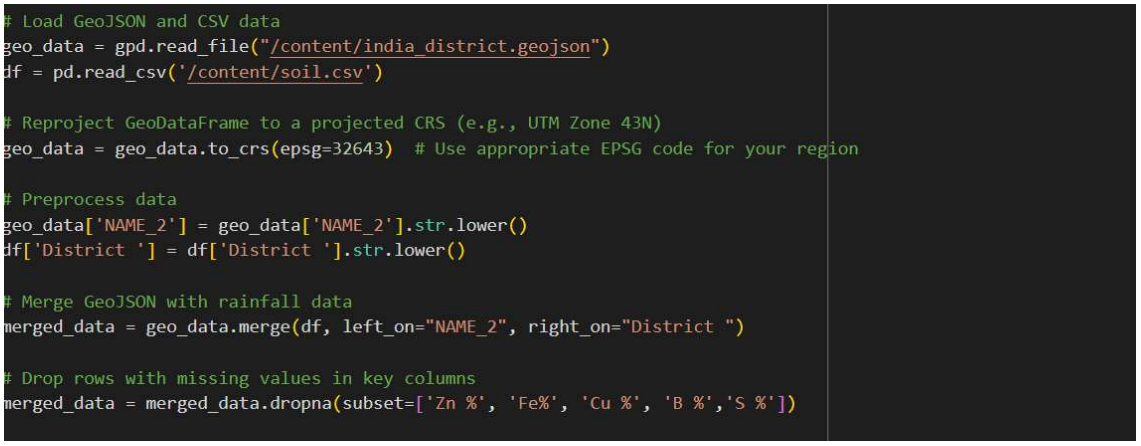
1. GeoJSON Dataset (india_district.geojson):
   - Type: Geospatial data file containing district boundaries in India.
   - Key Features:
     - NAME_2: District names, which serve as the primary key for merging datasets.
     - Geometries: Polygon data defining district boundaries, used for spatial analysis and visualization.
2. Soil Composition Dataset (soil.csv):
   - Type: Tabular dataset containing soil composition data for Indian districts.

- Key Features:
  - District: Name of the district, used to merge with GeoJSON data.
  - Zn %, Fe %, Cu %, B %, S %: Percentage values representing the concentration of essential soil nutrients.
- Purpose: Provides soil-related attributes for spatial autocorrelation and clustering analysis.

Both datasets are essential for combining spatial and attribute data to perform meaningful geospatial analyses and detect patterns in soil composition across Indian districts.

Data Cleaning and Preprocessing

The data cleaning and preprocessing involved reprojecting the GeoJSON dataset to a suitable CRS for accurate spatial analysis. District names in both datasets were standardized to lowercase to ensure a consistent key for merging. The GeoJSON and soil composition datasets were then merged using district names. Rows with missing values in critical soil nutrient columns (Zn %, Fe %, Cu %, B %, S %) were removed to maintain data integrity. The resulting dataset was cleaned, aligned, and ready for spatial analysis and clustering.



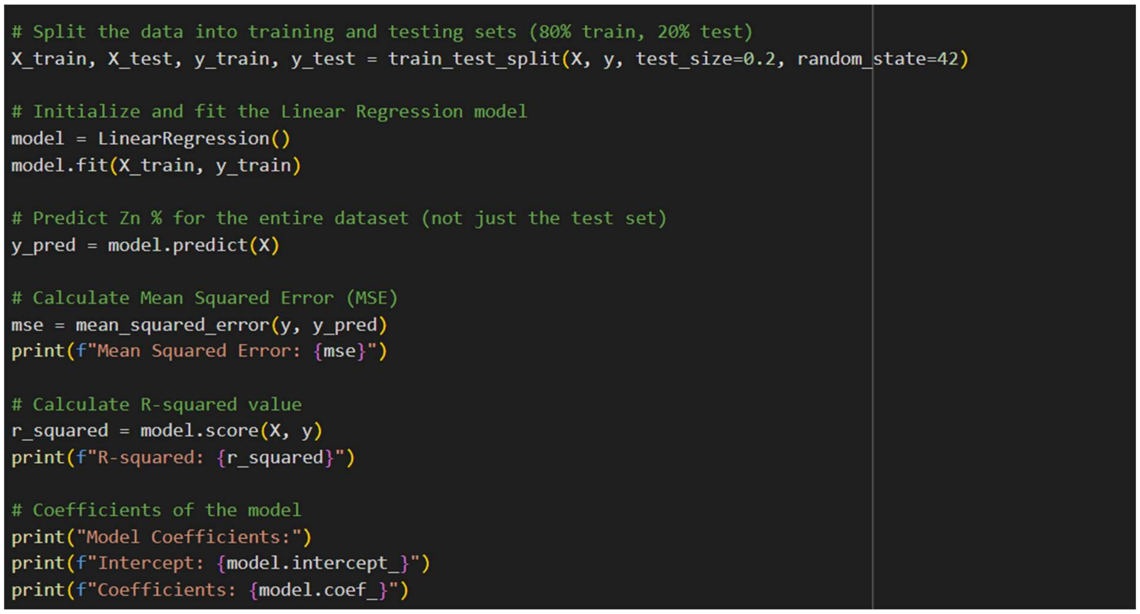
```
# Load GeoJSON and CSV data
geo_data = gpd.read_file("/content/india_district.geojson")
df = pd.read_csv('/content/soil.csv')

# Reproject GeoDataFrame to a projected CRS (e.g., UTM Zone 43N)
geo_data = geo_data.to_crs(epsg=32643)  # Use appropriate EPSG code for your region

# Preprocess data
geo_data['NAME_2'] = geo_data['NAME_2'].str.lower()
df['District '] = df['District '].str.lower()

# Merge GeoJSON with rainfall data
merged_data = geo_data.merge(df, left_on="NAME_2", right_on="District ")

# Drop rows with missing values in key columns
merged_data = merged_data.dropna(subset=['Zn %', 'Fe%', 'Cu %', 'B %','S %'])
```

Q1. Moran's I Autocorrelation

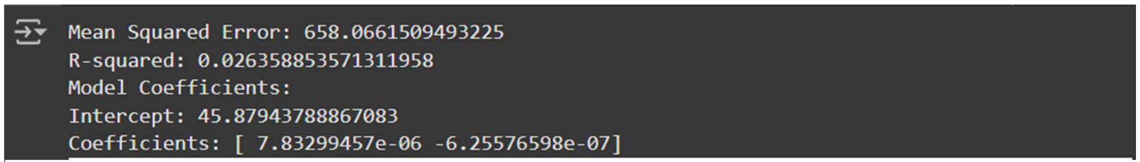
a. create a spatial weights matrix and Moran scatterplot

b. Calculate global spatial autocorrelation

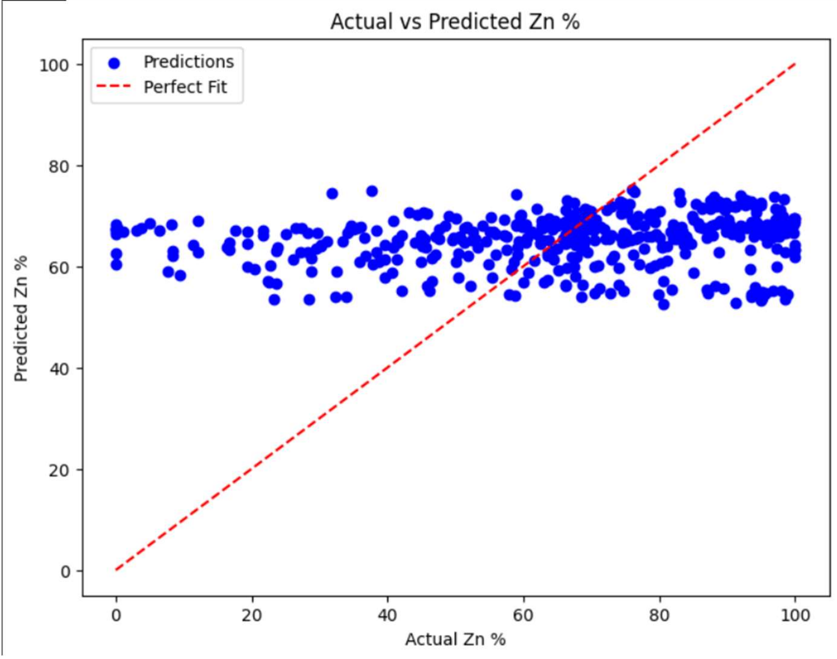
c. Detect clusters using the local spatial autocorrelation



```python
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict Zn % for the entire dataset (not just the test set)
y_pred = model.predict(X)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error: {mse}")

# Calculate R-squared value
r_squared = model.score(X, y)
print(f"R-squared: {r_squared}")

# Coefficients of the model
print("Model Coefficients:")
print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
```



```
Mean Squared Error: 658.0661509493225
R-squared: 0.026358853571311958
Model Coefficients:
Intercept: 45.87943788867083
Coefficients: [ 7.83299457e-06 -6.25576598e-07]
```



1. Mean Squared Error (MSE) = 658.07:

- The **MSE** indicates that the model's predictions deviate considerably from the actual values. A high MSE suggests poor model performance in predicting Zn %.

2. R-squared (R^2) = 0.026:

- **R^2** measures how much variance in the target variable (Zn %) is explained by the model. A value of **0.026** indicates that the model explains only **2.6%** of the variance in Zn %, meaning the model is not capturing the underlying pattern in the data well. The predictor variables (latitude and longitude) do not have a strong linear relationship with Zn %.

3. Model Coefficients:

- **Intercept = 45.88**: The predicted Zn % when both latitude and longitude are zero, though this might not be meaningful geographically.
- **Latitude coefficient = 7.83e-06**: A very small positive relationship between latitude and Zn %. This means as latitude increases, Zn % slightly increases, but the effect is minimal.
- **Longitude coefficient = -6.26e-07**: A very small negative relationship between longitude and Zn %. As longitude increases, Zn % decreases slightly, though the effect is negligible.

4. Global Moran's I = 0.2479:

- **Moran's I** measures spatial autocorrelation. A value of **0.2479** indicates **positive spatial autocorrelation**, meaning there is a tendency for districts with similar Zn % values to be clustered together. A higher value would indicate stronger clustering of similar values.

5. P-value for Moran's I = 0.001:

- The **p-value** of **0.001** indicates that the positive spatial autocorrelation detected is statistically significant. This suggests that the clustering of similar Zn % values is unlikely to be due to random chance.

6. Disconnected Components and Islands:

- There are **6 disconnected components** in the spatial weights matrix, meaning there are groups of districts that are not directly connected to the rest.
- **4 islands** (districts with no neighbors): The districts with IDs **25, 26, 305, 309** are isolated and do not have neighboring districts to form a meaningful spatial relationship.

Q2. Spatial Clustering

a. Develop an workflow for spatial clustering

b. Perform spatial clustering on point data using QGIS.

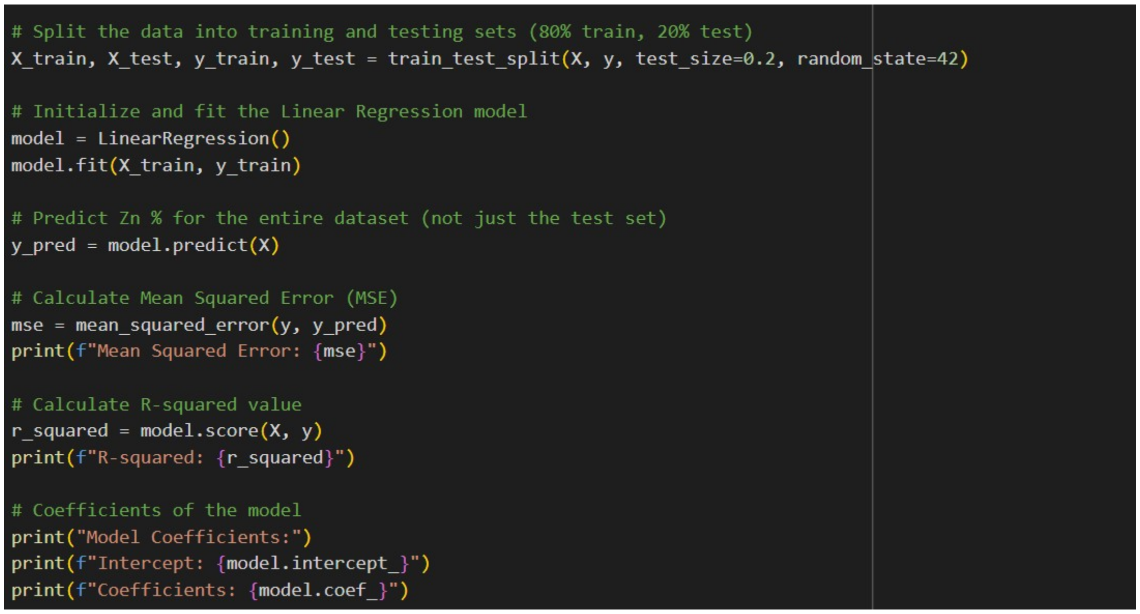
c. Employ algorithms such as DBSCAN or K-means to identify and analyze clusters.

Spatial clustering allows us to group points in a geographical area based on their proximity or other features. The two algorithms you can use for clustering in a spatial context are **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)** and **K-Means**.

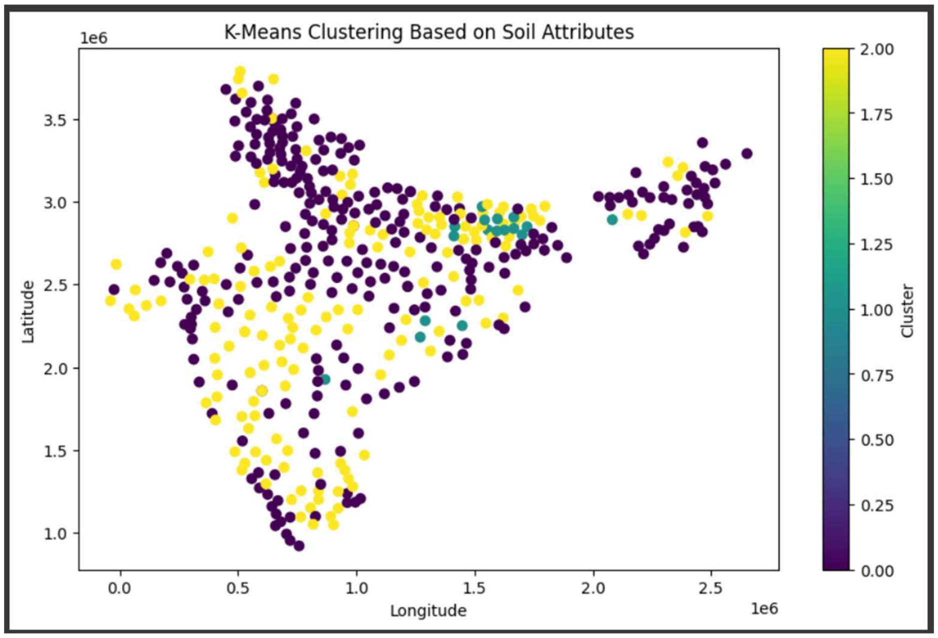
Clustering Algorithm:

> **K-Means**: Divides the data into a predefined number of clusters. This method requires the number of clusters (k) to be specified beforehand.
>
> **DBSCAN**: A density-based clustering algorithm that groups together points that are closely packed, with a specified radius. It can also identify outliers as noise.

Clustering with K-Means



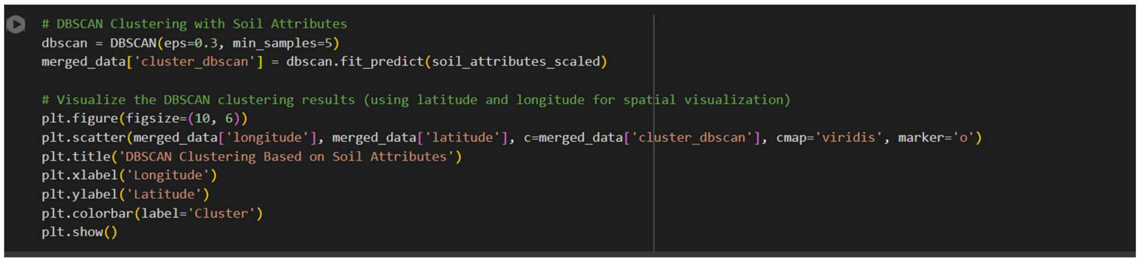
```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize and fit the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predict Zn % for the entire dataset (not just the test set)
y_pred = model.predict(X)

# Calculate Mean Squared Error (MSE)
mse = mean_squared_error(y, y_pred)
print(f"Mean Squared Error: {mse}")

# Calculate R-squared value
r_squared = model.score(X, y)
print(f"R-squared: {r_squared}")

# Coefficients of the model
print("Model Coefficients:")
print(f"Intercept: {model.intercept_}")
print(f"Coefficients: {model.coef_}")
```



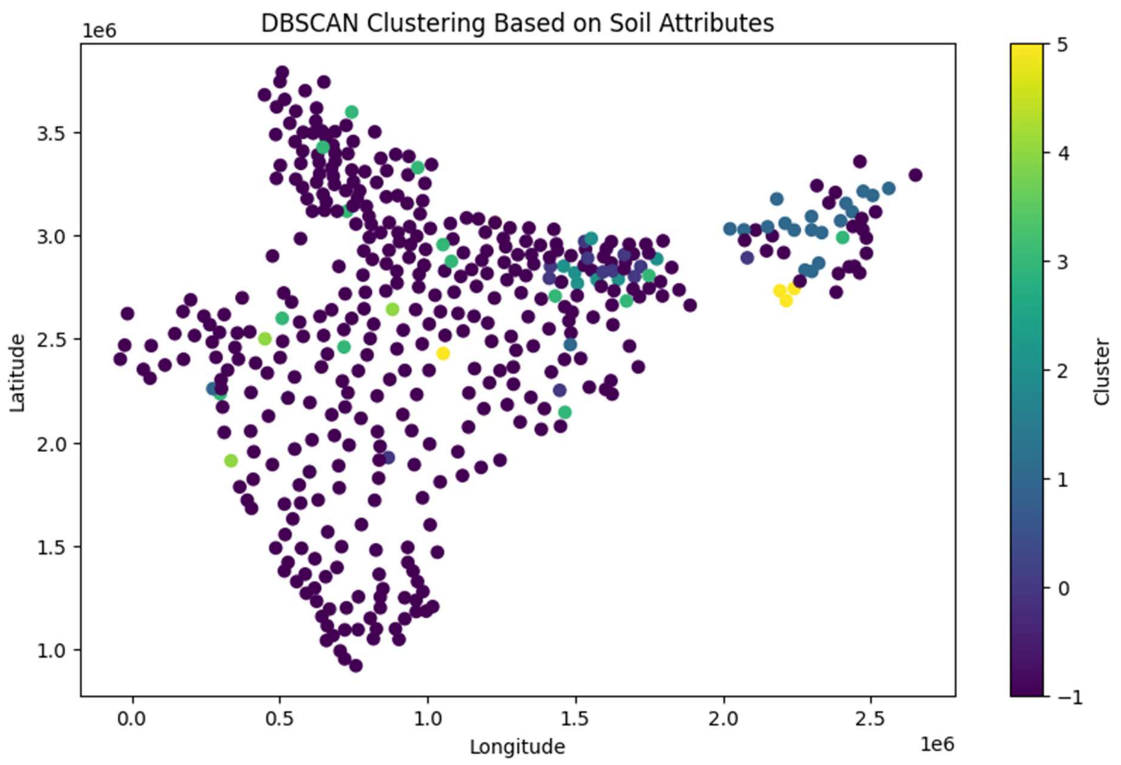
Interpretation of the Plot:

1. **Clustering based on soil attributes**:
   - The districts are divided into different clusters based on their **soil characteristics**, not just their geographical location.
   - The clusters are represented by different colors, with the **color scale** showing the cluster number.
     - **Yellow** represents one cluster.
     - **Purple** and **greenish-blue** represent other clusters.
2. **Geographical Distribution**:
   - The points are plotted based on their **latitude** and **longitude**, showing the geographical distribution of the districts.
   - The clusters are not purely geographically dependent, meaning districts with similar soil properties are grouped together regardless of their location.
3. **Insights from the Clusters**:
   - The plot reveals that districts with **similar soil characteristics** (e.g., Zn %, Fe %, etc.) are located in geographically close or distant regions, indicating that soil properties can show spatial patterns.
   - The **cluster of districts** (yellow) may represent regions with high concentrations of specific soil nutrients, while the **purple and greenish-blue clusters** may represent other soil properties with different nutrient profiles.
   - Clusters may overlap across geographical regions, showing that similar soil conditions can exist in different parts of India, possibly influenced by environmental factors like climate, topography, or human activity.

DBSCAN Clustering



```
# DBSCAN Clustering with Soil Attributes
dbscan = DBSCAN(eps=0.3, min_samples=5)
merged_data['cluster_dbscan'] = dbscan.fit_predict(soil_attributes_scaled)

# Visualize the DBSCAN clustering results (using latitude and longitude for spatial visualization)
plt.figure(figsize=(10, 6))
plt.scatter(merged_data['longitude'], merged_data['latitude'], c=merged_data['cluster_dbscan'], cmap='viridis', marker='o')
plt.title('DBSCAN Clustering Based on Soil Attributes')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.colorbar(label='Cluster')
plt.show()
```



The DBSCAN clustering plot shows districts grouped based on soil attributes, with each cluster represented by a different colour. The **purple points** are **outliers** (noise), indicating districts that don't fit into any main clusters. The clustering reflects regions with similar soil characteristics, and the **spread of clusters** suggests spatial patterns in soil composition. DBSCAN does not require the number of clusters to be predefined and can identify **dense areas** with similar soil properties while marking isolated regions as noise.