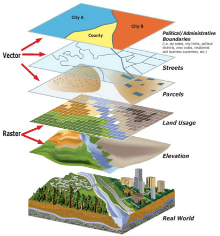
Geographic Information Systems

Dana Golden



Environmental and Natural Resource Economics - **December 9, 2024**

Presentation Outline

Introduction to Geographic Information Systems (GIS)

- **Definition of GIS:** Geographic Information Systems (GIS) are tools used to capture, store, analyze, and visualize spatial or geographic data.
- **Applications of GIS:** - Urban planning (e.g., zoning and land use) - Environmental management (e.g., conservation, climate change analysis) - Transportation planning - Public health (e.g., mapping disease outbreaks)
- **Core Components of GIS:**
    - **Data** (spatial and attribute)
    - **Hardware and Software** for data processing and visualization
    - **Methods and Algorithms** for spatial analysis
    - **People** who interpret and apply the insights gained

Types of GIS Data — Vector and Raster

- **Vector Data:** Represents spatial data using points, lines, and polygons.
  - **Points**: Single coordinate pairs, e.g., locations of wells, cities.
  - **Lines**: Series of connected points, e.g., roads, rivers.
  - **Polygons**: Closed shapes formed by lines, e.g., lakes, property boundaries.
- **Raster Data:** Represents spatial data using a grid of cells or pixels, each with a value.
  - Commonly used for continuous data like elevation, temperature, and satellite imagery.
  - Each cell in a raster has a resolution, e.g., 10m x 10m, which determines detail.

Vector vs. Raster

- **Key Differences:** - **Vector Data**: High precision, best for discrete features, smaller file sizes for certain data types. - **Raster Data**: Suitable for continuous data, allows analysis based on cell values, larger file sizes.
- Choosing Between Vector and Raster: The choice depends on the data type and the analysis goals; for example, vector is often used for infrastructure, while raster is used for environmental variables.

Vector Data Structure and Applications

- **Structure of Vector Data:** - Consists of coordinates defining points, lines, or polygons. - Associated attribute data provides information about each feature (e.g., a city's population or a road's name).
- **Advantages of Vector Data:** - High precision in representing boundaries and discrete features. - Compact file size for well-defined features. - Ideal for network analyses, like finding shortest paths on road networks.
- **Applications of Vector Data:**
  - **Urban Planning:** Mapping infrastructure, zoning areas.
  - **Transportation:** Analyzing road networks, traffic flows.
  - **Public Health:** Mapping disease cases by locations, health facility access.

Raster Data Structure and Applications

- **Structure of Raster Data:** - Consists of a grid of cells (pixels), each containing a value. - Cell size (resolution) determines the detail of the data; smaller cells provide more detail.
- **Advantages of Raster Data:** - Suitable for continuous data like elevation, land cover, temperature. - Facilitates mathematical operations on cell values, such as overlay analysis.
- **Applications of Raster Data:**
  - **Environmental Science:** Analyzing temperature, vegetation, soil moisture.
  - **Remote Sensing:** Interpreting satellite and aerial imagery for land use or disaster assessment.
  - **Agriculture:** Crop monitoring, soil condition mapping.

Vector and Raster Data

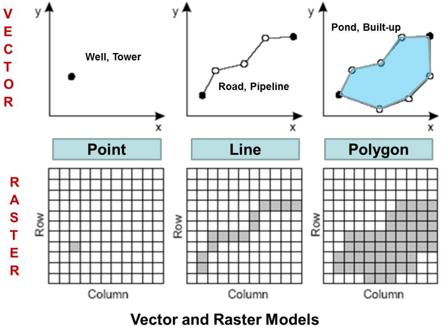


Figure 1: Vector and Raster visual

Tips for Cleaning Geospatial Data

- **Understand the Data:**
  - Review metadata for CRS (Coordinate Reference System), units, and attributes.
  - Identify the purpose of the dataset.
- **Check for Missing or Invalid Values:**
  - Inspect for missing geometries or attribute values.
  - Handle invalid geometries (e.g., self-intersecting polygons).
- **Reproject to a Consistent CRS:**
  - Ensure all layers share a common CRS for analysis.
- **Simplify Geometry:**
  - Reduce complexity for faster processing while retaining essential details.
- **Remove Duplicates or Unnecessary Data:**
  - Filter irrelevant features or rows to improve clarity and performance.

Cleaning Geospatial Data in R - CRS and Validity

Set Up Libraries and Read Data:

```
library(sf)
library(dplyr)

# Load geospatial data
data <- st_read("path/to/your/data.shp")

# Check the Coordinate Reference System (CRS)
st_crs(data)

# Reproject to a consistent CRS if necessary
data <- st_transform(data, crs = 4326)
```

Cleaning Geospatial Data in R - CRS and Validity

```
# Check for invalid geometries
invalid <- st_is_valid(data)
if (any(!invalid)) {
  data <- st_make_valid(data)
}
```

Cleaning Geospatial Data in R - Simplification and Missing Values

Simplify Geometry and Handle Missing Data:

```r
# Simplify geometries for faster processing
data <- st_simplify(data, dTolerance = 0.001)

# Inspect for missing values in attributes
missing_summary <- data %>%
  summarise(across(everything(), ~ sum(is.na(.))))

# Handle missing values (e.g., fill with default or
   remove)
data <- data %>%
  mutate(attribute = ifelse(is.na(attribute), "default
     _value", attribute)) %>%
  filter(!is.na(geometry))
```

Where to Find Geospatial Data

Sources of Geospatial Data:

- **Government Portals:**
  - data.gov (United States)
  - EEA Data and Maps (Europe)
- **Open Data Platforms:**
  - Open Data Commons
  - Open Data Initiative
- **Specialized Datasets:**
  - Natural Earth Data (Global data)
  - GADM (Administrative boundaries)
  - HIFLD (Infrastructure data)

Where to Find Geospatial Data

- **Satellite Data:**
  - USGS Earth Explorer
  - Landsat Missions
  - Copernicus Sentinel Data
- **Academic and Research Platforms:**
  - Harvard Dataverse
  - Global Biodiversity Information Facility (GBIF)
- **Crowdsourced Data:**
  - OpenStreetMap (OSM)
  - GeoJSON.io

Basics of Visualizing GIS Data

- **Purpose of GIS Visualization:** - Communicate spatial patterns and insights effectively. - Make data accessible and interpretable for decision-making.
- **Map Elements:** Essential components of a GIS map include:
  - **Title:** Brief description of the map's purpose.
  - **Legend:** Key to understanding symbols and colors used.
  - **Scale:** Shows the relationship between map distance and real-world distance.
  - **North Arrow:** Indicates the map's orientation.
  - **Coordinate System:** Specifies the projection, such as latitude/longitude.
- **Data Layers:** GIS maps often contain multiple layers (e.g., roads, rivers, population data) that are visualized together to provide context.

Types of GIS Maps

- **Choropleth Maps:** Display data values through color shading, commonly used for visualizing density, population, or income across areas. - **Example:** Population density by county.
- **Heat Maps:** Show intensity or concentration of a variable over space, often used for displaying crime rates, pollution levels, or disease spread.
- **Topographic Maps:** Represent elevation and terrain using contour lines or color gradients. - **Example:** Elevation maps for hiking or environmental studies.
- **Dot Density Maps:** Use dots to represent the occurrence of a phenomenon, with each dot representing a specific quantity. - **Example:** Mapping population or wildlife distributions.
- **Proportional Symbol Maps:** Use symbols of varying sizes to represent the magnitude of a variable. - **Example:** Circle sizes to show city populations.

Chloropleth

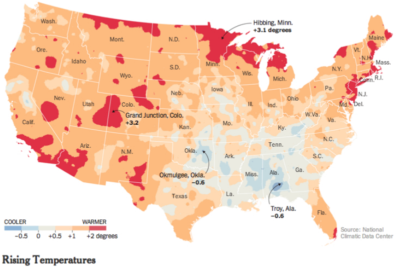


Figure 2: Someone better raise interests because the US is running hot!

Geospatial Heat Map

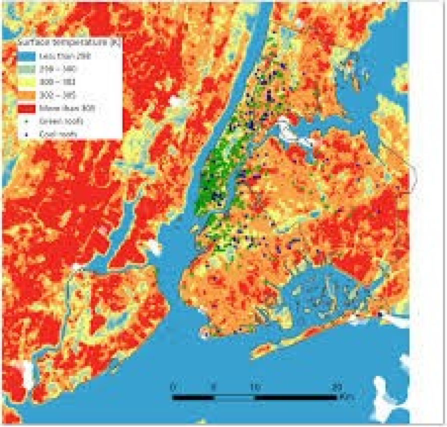


Figure 3: Hot in NYC

Dot Density Map

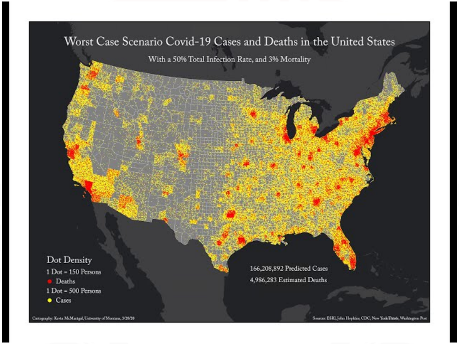


Figure 4: It's been a rough couple years gang...

Effective GIS Data Visualization Techniques

- **Color Choice:** - Use color schemes that are easy to interpret and colorblind-friendly. - **Sequential colors** for ordered data (e.g., income levels). - **Diverging colors** for data with a meaningful midpoint (e.g., temperature changes).
- **Symbology:** Choose symbols and sizes that accurately represent data magnitudes, avoiding misleading scales. - For proportional symbol maps, ensure symbols don't overlap excessively, which can obscure data.
- **Layer Transparency:** Adjust transparency for overlapping layers to allow multiple datasets to be visualized together without obscuring details.
- **Data Classification:** - Choose the right classification method (e.g., equal intervals, natural breaks) based on the data distribution. - Avoid too many classes; 4–7 classes are generally easier for viewers to interpret.

Example GIS Visualization Workflow

- **Step 1: Define the Purpose** — Determine what spatial patterns or insights the map should communicate.
- **Step 2: Data Preparation** — Clean and preprocess spatial data, including projecting all data layers to the same coordinate system.
- **Step 3: Choose Map Type and Symbology** — Select the most suitable map type (e.g., choropleth, heat map) and appropriate symbols to represent data.
- **Step 4: Add Map Elements** — Include title, legend, scale, and north arrow to ensure context and readability.
- **Step 5: Refine and Test for Clarity** — Adjust colors, transparency, and data classes; test with different audiences for interpretability.
- **Step 6: Finalize for Presentation** — Export in suitable formats (e.g., PDF for print, interactive maps for web use).

Tools for Working with Geospatial Data

Software Tools for Geospatial Data:

- **GIS Software:**
  - **QGIS:** Free and open-source desktop GIS for spatial analysis and visualization.
  - **ArcGIS:** Comprehensive GIS software with robust analysis and mapping capabilities.
- **Programming Languages:**
  - **R:** Powerful for statistical analysis and visualization of geospatial data.
  - **Python:** Popular for geospatial processing with libraries like GeoPandas and PyProj.

Tools for Working with Geospatial Data

Software Tools for Geospatial Data:

- **Web-based Tools:**
    - **Google Earth Engine:** Cloud-based platform for planetary-scale analysis.
    - **Mapbox/Leaflet:** JavaScript libraries for interactive web maps.
- **Databases:**
    - **PostGIS:** Extension of PostgreSQL for spatial queries.
    - **SpatiaLite:** Lightweight geospatial database based on SQLite.

R Packages for Geospatial Data

Core R Packages for Geospatial Data:

- **sf:** Simple Features for geospatial data handling.
- **sp:** Legacy package for spatial data classes and methods.
- **raster:** Analysis and manipulation of raster data.
- **terra:** Modern replacement for raster with enhanced functionality.

R Packages for Geospatial Data

Visualization:

- **ggplot2:** Integration with geospatial data using `geom_sf`.
- **tmap:** Thematic mapping for both static and interactive visualizations.
- **leaflet:** Interactive web-based maps.
- **plotly:** Interactive maps with dynamic tooltips and zooming.

R Packages for Geospatial Data

Specialized Tools:

- **rgdal:** Interface to GDAL for data reading/writing.
- **rgeos:** Geometric operations for spatial objects.
- **lwgeom:** Advanced geometry operations.
- **sfnetworks:** For spatial network analysis.
- **geosphere:** Distance and area calculations on a spherical Earth.

Loading and Inspecting Geospatial Data

Load and Inspect Geospatial Data:

```
library(sf)

# Load shapefile
data <- st_read("path/to/shapefile.shp")

# Inspect the first few rows
print(head(data))

# Plot basic geometry
plot(st_geometry(data))
```

Visualizing Geospatial Data with ggplot2

```r
library(ggplot2)

# Basic map
ggplot(data) +
  geom_sf() +
  ggtitle("Basic Map of Geospatial Data") +
  theme_minimal()
```

Adding Attributes to the Map

Coloring by Attribute:

```
# Map with color based on an attribute
ggplot(data) +
  geom_sf(aes(fill = ATTRIBUTE_COLUMN)) +
  scale_fill_viridis_c() +
  ggtitle("Map Colored by Attribute") +
  theme_minimal()
```

Overlaying Layers

Combining Multiple Layers:

```r
# Load additional geospatial data
overlay <- st_read("path/to/overlay.shp")

# Combine layers
ggplot() +
  geom_sf(data = data, fill = "lightblue", color = "black") +
  geom_sf(data = overlay, fill = NA, color = "red", linetype = "dashed") +
  ggtitle("Overlaying Multiple Geospatial Layers") +
  theme_minimal()
```

Using leaflet for Interactive Visuals

```r
library(leaflet)

# Create interactive map
leaflet(data) %>%
  addTiles() %>%
  addPolygons(fillColor = ~pal(ATTRIBUTE_COLUMN),
              color = "black", weight = 1) %>%
  addLegend("bottomright", pal = pal, values = ~
      ATTRIBUTE_COLUMN,
             title = "Legend")
```

Using Plotly for Interactive Visuals

```r
library(sf)
library(plotly)

# Load geospatial data
data <- st_read("path/to/shapefile.shp")

# Convert to a format compatible with Plotly
data_geojson <- sf::st_as_sf(data)
```

Using Plotly for Interactive Visuals

```r
# Create a Plotly map
plot_ly() %>%
  add_trace(
    type = "scattergeo",
    geojson = data_geojson,
    locations = ~id, # Replace 'id' with the unique
        identifier for each geometry
    z = ~ATTRIBUTE_COLUMN, # Replace with the
        attribute to visualize
    colorscale = "Viridis",
    marker = list(line = list(width = 0.5, color = "
        black"))
  ) %>%
  layout(
    title = "Interactive Map with Plotly",
    geo = list(fitbounds = "locations")
  )
```

Visualization with Tableau

Why Use Tableau for Visualization?

- **User-Friendly:** Intuitive drag-and-drop interface for creating complex visualizations.
- **Dynamic Visualizations:** Create interactive dashboards and charts.
- **Integration:** Seamlessly connects to various data sources (CSV, SQL, cloud platforms, etc.).
- **Collaboration:** Share visualizations with others through Tableau Public or Tableau Server.

Visualization with Tableau

Popular Visualization Features:

- **Bar Charts, Line Charts, and Scatter Plots:** Standard visualizations with advanced customization.
- **Dashboards:** Combine multiple visualizations for holistic analysis.
- **Filters and Parameters:** Add interactivity and user control.
- **Storytelling:** Narrate insights through data stories.

Best Practices:

- Clean and preprocess your data before importing.
- Use color and labels judiciously to avoid clutter.
- Design for your audience and focus on key insights.

Connecting to a SQL server in Tableau

Connecting to a SQL Database:

- **Supported Databases:**
  - Tableau supports popular databases like MySQL, PostgreSQL, SQL Server, and Oracle.
- **Steps to Connect:**
  1. Go to `Data` > `Connect to Data`.
  2. Select your database type (e.g., `PostgreSQL`, `MySQL`).
  3. Provide connection details:
     - `Server:` Hostname or IP address.
     - `Database:` Name of the database.
     - `Username` and `Password`.
  4. Click `Sign In` and select tables or write custom SQL queries.

Connecting to an API Tableau

Connecting to an API:

- **Tableau Native API Connectors:**
  - Use pre-built connectors for platforms like Salesforce, Google Analytics, and others.
- **Using Web Data Connectors:**
  1. Create or use an existing **Web Data Connector** (WDC) script.
  2. In Tableau, go to `Data ¿ Connect to Data`.
  3. Select `Web Data Connector` and provide the WDC URL.
  4. Authenticate and configure API settings in the WDC interface.

Best Practices:

- Optimize SQL queries for performance before importing into Tableau.
- For APIs, ensure the WDC script handles pagination and authentication securely.
- Test connections for latency and refresh frequency.

Spatial Data in Tableau

Working with Spatial Data in Tableau:

- **Importing Spatial Data:**
  - Tableau supports spatial files like `Shapefiles (.shp)`, `KML`, and `GeoJSON`.
  - Connect directly to spatial databases like PostGIS.
- **Built-in Geographic Roles:**
  - Automatically recognizes geographic data (e.g., countries, states, ZIP codes).
  - Assign geographic roles to custom fields.
- **Custom Maps:**
  - Use Tableau's built-in maps or connect to Mapbox for customized backgrounds.
  - Overlay spatial data for detailed geographic insights.

Spatial Data in Tableau

Visualization Features:

- **Choropleth Maps:** Visualize data with colors based on geographic areas.
- **Point Maps:** Use latitude and longitude data for precise locations.
- **Heat Maps:** Highlight density and patterns in spatial data.
- **Flow Maps:** Visualize movements or connections between locations.

Advanced Features:

- Combine spatial data with non-spatial datasets for comprehensive analysis.
- Use filters, tooltips, and layers for interactive spatial dashboards.

Visualization, Dashboard, and Story in Tableau

1. **Visualization: Analyze Air Quality**
   - Create a **Choropleth Map** to display average air pollution (e.g., PM2.5 levels) by region.
   - Example:
     - Data: Air quality monitoring stations.
     - Insight: Highlight regions with high pollution levels for targeted policy interventions.
   - Features:
     - Interactive tooltips showing detailed station-level data.
     - Filters to explore data by time period or pollutant type.

Visualization, Dashboard, and Story in Tableau

2. Dashboard: Renewable Energy Overview

- Combine multiple visualizations:
  - Bar chart: Energy production by source (solar, wind, hydro, etc.).
  - Line chart: Monthly trends in renewable energy adoption.
  - Map: Geographical distribution of renewable energy facilities.
- Example:
  - Data: Energy production data by state.
  - Insight: Monitor renewable energy growth and identify underperforming regions.
- Features:
  - Add filters for year, energy type, and region.
  - Use KPIs (e.g., total renewable energy generated).

Visualization, Dashboard, and Story in Tableau

3. Story: Climate Change Impacts

- Sequence of visualizations and dashboards to narrate insights.
- Example:
  - Story points:
    1. Historical temperature changes (line chart).
    2. Impact of rising sea levels (geospatial map).
    3. Correlation between emissions and temperature anomalies (scatter plot).
- Insight: Illustrate the progression of climate change and the urgency for action.
- Features:
  - Add annotations to highlight key findings.
  - Include captions summarizing each point.

Dashboard in Tableau

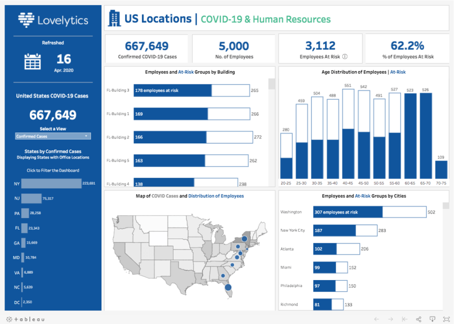


Figure 5: Employers like Dashboards!

Regression with Spatial Data

What is Spatial Regression?

- A regression model that accounts for spatial dependencies in the data.
- Useful when the values of the dependent variable are influenced by neighboring locations.

Common Spatial Regression Models:

- **Spatial Lag Model (SLM):**
  - Includes a lagged dependent variable to capture spatial autocorrelation.

$$y = \rho W y + X\beta + \epsilon$$

  - Wy: Spatially lagged dependent variable.
- **Spatial Error Model (SEM):**
  - Accounts for spatial autocorrelation in the error term.

$$y = X\beta + \epsilon, \quad \epsilon = \lambda W \epsilon + u$$

Estimation of Spatial Regression in R

R Code for Spatial Regression:

```
# Load required libraries
library(spdep)
library(spatialreg)

# Example dataset: Simulated spatial data
set.seed(123)
coords <- cbind(runif(100, 0, 10), runif(100, 0, 10))
    # Coordinates
neighbors <- knearneigh(coords, k = 4)
   # Nearest neighbors
weights <- nb2listw(knn2nb(neighbors))
   # Spatial weights
```

Estimation of Spatial Regression in R

```
# Simulated data
data <- data.frame(
    y = rnorm(100),          # Dependent variable
    x1 = rnorm(100),         # Independent variable
    x2 = rnorm(100)          # Another independent
       variable
)

# Fit a Spatial Lag Model
slm <- lagsarlm(y ~ x1 + x2, data = data, listw =
   weights)
summary(slm)

# Fit a Spatial Error Model
sem <- errorsarlm(y ~ x1 + x2, data = data, listw =
   weights)
summary(sem)
```

Estimation of Spatial Regression in R
Explanation

- **Weights Matrix:** Defines spatial relationships between locations.
- **lagsarlm():** Fits a spatial lag model.
- **errorsarlm():** Fits a spatial error model.

Spatial Autocorrelation

What is Spatial Autocorrelation?

- Measures the degree to which a spatial variable is correlated with itself across a geographic area.
- Indicates whether similar values occur near each other (spatial clustering) or are randomly distributed.

Key Concepts:

- **Positive Spatial Autocorrelation:**
  - Nearby locations have similar values (e.g., high-income neighborhoods clustering together).
- **Negative Spatial Autocorrelation:**
  - Nearby locations have dissimilar values (e.g., alternating land use patterns).
- **No Spatial Autocorrelation:**
  - Values are randomly distributed.

Spatial Autocorrelation

Measuring Spatial Autocorrelation:

- **Moran's I:** Global measure of spatial autocorrelation.
- **Geary's C:** Compares squared differences in neighboring values.
- **Local Indicators of Spatial Association (LISA):** Measures local clusters and outliers.

Applications:

- Urban planning (e.g., detecting neighborhood clustering).
- Environmental monitoring (e.g., mapping pollutant concentrations).
- Public health (e.g., identifying disease hotspots).

Spatial Autocorrolation Visualized

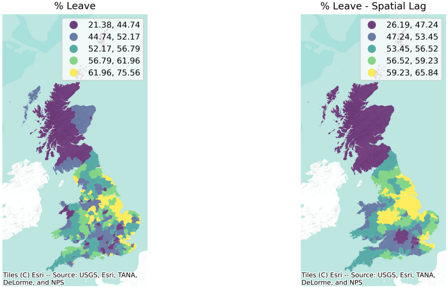


Figure 6: Brexit means Brexit!

Detecting and Dealing with Spatial Autocorrelation

Detecting Spatial Autocorrelation:

- **Global Moran's I:**
  - Measures overall spatial autocorrelation.
- **Local Moran's I (LISA):**
  - Identifies local clusters and outliers.

R Code for Moran's I:

```
moran <- moran.test(data\$y, weights)
```

Dealing with Spatial Autocorrelation:

- Use spatial regression models (SLM or SEM).
- Improve model specification by including relevant covariates.
- Reassess spatial weights matrix (W) for accuracy.

K-means Clustering for Geospatial Data

What is K-means Clustering?

- An unsupervised learning algorithm that partitions data into k clusters.
- Minimizes the within-cluster sum of squared distances.

Steps in K-means:

1. Choose the number of clusters k.
2. Randomly initialize k centroids.
3. Assign each data point to the nearest centroid.
4. Update centroids as the mean of the assigned points.
5. Repeat until convergence.

K-means Applications in Geospatial Data

- Identifying regional hotspots (e.g., crime, disease).
- Grouping locations based on proximity and characteristics.
- Resource allocation (e.g., emergency response, delivery zones).

Geospatial Adaptation:

- Use geographic coordinates (latitude, longitude) for clustering.
- Optionally incorporate other features (e.g., population, income).

Example Code: K-means for Geospatial Data in R

R Code for Geospatial K-means Clustering:

```
# Load required libraries
library(tidyverse)
library(sf)           # For geospatial data handling
library(ggplot2)      # For visualization

# Example dataset: Simulated geographic points
set.seed(123)
data <- tibble(
    id = 1:100,
    lat = runif(100, 35, 45),        # Latitude
    lon = runif(100, -120, -110)     # Longitude
)
```

Example Code: K-means for Geospatial Data in R

R Code for Geospatial K-means Clustering:

```

# Prepare data for clustering

coords <- data %>% select(lon, lat)

# Add cluster assignments to the data

data <- data % > % mutate(cluster = kmeans_result)
```

Example Code: K-means for Geospatial Data in R

```
# Apply K-means clustering (choose k = 3)

kmeans_ result <- kmeans(coords, centers = 3,
    nstart = 25)
```

Example Code: K-means for Geospatial Data in R

```
# Visualize the clusters

ggplot(data, aes(x = lon, y = lat, color = as.factor(cluster))) +

geom_point(size = 3) +

labs(title = "K-means␣Clustering␣of␣Geospatial␣Data",
     x = "Longitude", y = "Latitude", color = "Cluster") +

  theme_minimal()
```

Explanation of R Code for Spatial K-means

- **Input Data:** Geographic coordinates (latitude, longitude).
- **K-means Clustering:** Applied to geographic data with $k = 3$.
- **Visualization:** Plots clusters on a scatterplot of coordinates.

K-means General

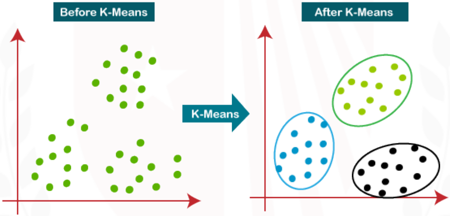


Figure 7: K-means Clustering

K-means Spatial

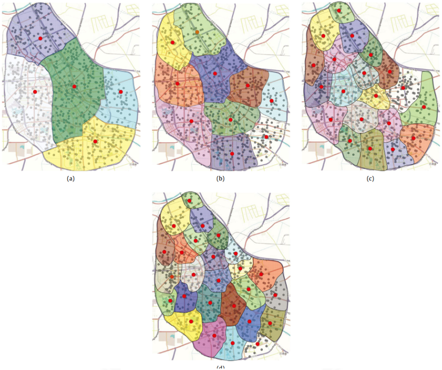


Figure 8: Spatial K-means

The Elbow Curve

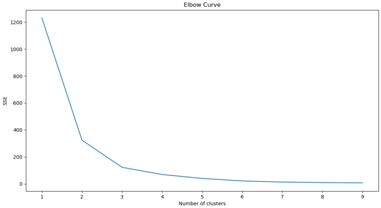


Figure 9: Choosing Number of Clusters

Thank You So Much!

List of References