Chapter 2: Pre-processing Spatial Data for Machine Learning

Topic 1: Data Cleaning and Transformation

Topic Overview: Learning Goals and Conceptual Background

Conceptual Background:
Raw spatial data, whether from satellite sensors, GPS trackers, or administrative records, is rarely perfect. It often contains imperfections such as missing values, inconsistent scales, and misaligned spatial resolutions. The principle of "garbage in, garbage out" is especially critical in spatial machine learning. The spatial nature of the data means that errors do not exist in isolation; a single missing data point can disrupt neighborhood calculations, bias spatial statistics, and ultimately cause a model to fail or produce unreliable results. This initial phase of preprocessing is arguably the most critical step in the entire spatial machine learning workflow. It involves a systematic process of cleaning, correcting, and transforming the data to ensure its integrity, comparability, and suitability for sophisticated algorithms. This topic lays the foundation by addressing how to handle imperfections, make disparate datasets comparable through scaling, and align spatial grids for integrated analysis.

Learning Goals:

Upon completing this topic, students will be able to:

1. Identify and diagnose different types of missing data (MCAR, MAR, MNAR) in both vector and raster datasets.
2. Select and apply appropriate deletion or imputation methods, with a focus on spatially-aware techniques.
3. Understand the unique challenges of missing data in remote sensing imagery (e.g., cloud cover) and the advanced methods used to address them.
4. Differentiate between data normalization and standardization and understand their impact on machine learning algorithms.
5. Implement feature scaling techniques using Python's scikit-learn library.
6. Recognize the importance of domain-specific calibration (e.g., radiometric correction) for remote sensing data before applying statistical scaling.
7. Distinguish between spatial interpolation and raster resampling.
8. Choose and apply the correct interpolation or resampling method based on the data type (continuous vs. discrete) and analytical objective.
9. Utilize Python libraries such as GeoPandas, Rasterio, and Scikit-learn to perform data cleaning and transformation tasks.

Key Concepts Table: Data Cleaning and Transformation Methods

Category	Method/Technique	Description	Primary Use Case / Data Type
Missing Data Handling	Listwise/Pairwise Deletion	Removal of entire records or use of records only when all variables for a specific analysis are present.	Simple, but risky if data is not Missing Completely at Random (MCAR) or if the dataset is small.
	Simple Imputation	Replacing missing values with a statistic like the mean, median, or mode of the column.[2]	Quick fix for small amounts of missing data; can distort variance.
	Spatial Imputation	Using values from geographic neighbors to estimate a missing value, based on Tobler's First Law.[1]	Vector and raster data where spatial autocorrelation is present.
	Areal Interpolation	A geostatistical method to predict values for polygon gaps using data from surrounding polygons.[5]	Filling missing data in administrative units (e.g., census tracts).
	DINEOF / HANTS	Advanced spatiotemporal methods using Empirical Orthogonal Functions or Harmonic Analysis to reconstruct missing data in time-series imagery.[6]	Large, persistent gaps in satellite imagery (e.g., seasonal cloud cover).
Feature Scaling	Standardization (Z-Score)	Transforms data to have a mean of 0 and a standard deviation of 1.[8]	Algorithms assuming a Gaussian distribution (e.g., SVMs, Logistic Regression) and for PCA.
	Normalization (Min-Max)	Rescales data to a fixed range, typically .[8]	Distance-based algorithms (e.g., k-NN) and neural networks. Image processing.
	Radiometric Calibration	Converts raw Digital Numbers (DNs) from a sensor to physically meaningful values like radiance or backscatter (σ^o).[9]	Pre-step for all scientific analysis of raw SAR, hyperspectral, and optical satellite imagery.

Grid Alignment	Spatial Interpolation (IDW, Kriging)	Estimates values at unsampled locations based on known point values to create a continuous surface.[11]	Vector point data (e.g., weather stations, soil samples) to be converted to a raster.
	Resampling (Nearest Neighbor)	Assigns the output cell the value of the nearest input cell. Does not create new values.[13]	Discrete/categorical raster data (e.g., land cover maps).
	Resampling (Bilinear, Cubic)	Calculates the output cell value as a weighted average of the 4 (Bilinear) or 16 (Cubic) nearest input cells, creating a smoother output.[13]	Continuous raster data (e.g., elevation, temperature, reflectance).

Detailed Explanations and Slide Content

Module 1.1: Foundations of Data Integrity - Handling Missing Spatial Data

Slides 1-6: Introduction to Missing Spatial Data

- **Slide 1: Title Slide: Handling Missing Spatial Data**
  - Chapter 2: Pre-processing Spatial Data for Machine Learning
  - Topic 1.1: Foundations of Data Integrity

    - Your Name, Course Title, University
- **Slide 2: The "Why": The Impact of Missingness**
    - **Concept:** Missing data is not just a nuisance; it's a critical flaw that can invalidate your entire analysis. In spatial ML, this is amplified.
    - **Why we are learning this:**
        - Machine learning models cannot process null or NaN values. The first step to any modeling is ensuring data completeness.
        - Spatial algorithms rely on neighborhood relationships. A missing polygon or pixel creates a "hole," breaking spatial contiguity and corrupting calculations for spatial features like the spatial lag.
        - Ignoring missing data can lead to severely biased model parameters, reduced statistical power, and inaccurate conclusions about spatial patterns.[15]
    - **Visualization:** A diagram showing a grid of polygons. One polygon in the center is missing. Arrows from its neighbors point to the empty space, labeled "Neighborhood calculations fail."
- **Slide 3: Real-World Scenarios of Missing Spatial Data**
    - **Urban Planning:** A census tract is missing demographic data due to a reporting error.
    - **Environmental Science:** A network of air quality sensors has gaps in its time-series data due to sensor malfunction or power loss.
    - **Transport Networks:** Traffic volume data is missing for a specific highway segment during a major event because the sensor was overwhelmed.
    - **Remote Sensing:** A crucial part of a satellite image for crop health monitoring is obscured by clouds.[6]
    - **Diagram:** Four quadrants, each with an icon (city skyline, factory, road, satellite) and the corresponding example text.
- **Slide 4: Mechanisms of Missingness - Part 1**
    - **Concept:** To handle missing data correctly, we must first understand *why* it is missing. The underlying cause, or "mechanism," determines which methods are safe to use.
    - **1. Missing Completely at Random (MCAR):**
        - The probability of a value being missing is completely random. It does not depend on any other variable, observed or unobserved.
        - **Example:** A lab technician accidentally skips a soil sample in a batch for chemical analysis.
        - **Implication:** Deleting data under the MCAR assumption is generally safe and will not introduce bias, though it reduces the dataset size.[2]
    - **Visualization:** A grid of data points. A few points are randomly removed (highlighted in red). An arrow points to them labeled "No underlying pattern."
- **Slide 5: Mechanisms of Missingness - Part 2**
    - **2. Missing at Random (MAR):**
        - The probability of a value being missing is related to another *observed* variable in the dataset, but not the missing value itself.
        - **Example:** In a survey about income, individuals with higher education levels are less likely to disclose their income. The missingness in 'income' can be predicted by the 'education' variable.
        - **Implication:** Simple deletion is risky as it can introduce bias. More advanced imputation methods that use the related variables (like regression imputation) are preferred.[3]
    - **Visualization:** A table with 'Education' and 'Income' columns. Several 'Income' cells are missing, but primarily for rows where 'Education' is 'High'.
- **Slide 6: Mechanisms of Missingness - Part 3**
    - **3. Missing Not at Random (MNAR):**
        - The probability of a value being missing is related to the unobserved value itself. This is the most

problematic case.
    - **Example:** A pollution sensor stops working when pollution levels exceed a certain threshold, causing it to overheat. The highest pollution values are systematically missing.[3]
    - **Implication:** This introduces significant bias. Simple deletion or imputation will lead to models that underestimate the true range of values. This requires advanced modeling or domain-specific knowledge to handle.
  - **Visualization:** A time-series graph of pollution levels. The line is cut off whenever it reaches a high peak, with the missing segments highlighted. An annotation reads: "Data is missing *because* it is high."

Slides 7-15: Detection and Diagnosis

- **Slide 7: Step 1: Detecting Missing Data in Vector Datasets**
  - **Concept:** Before we can fix the problem, we must find it. In vector data (points, lines, polygons), missingness can occur in attribute columns or in the geometry itself.
  - **Tool:** geopandas library in Python.
  - **Detecting Attribute Nulls:** Use the .isnull().sum() methods to get a count of missing values per column.
  - **Python Code Example:**

```
Python
import geopandas as gpd
# Load a shapefile of world countries
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
# Check for missing values in attribute columns
print(world.isnull().sum())
```

  - **Output Snippet:**

```
pop_est       0
continent     0
name          0
iso_a3        0
gdp_md_est    0
geometry      0
dtype: int64
# (Example with no missing attributes)
```

  -
- **Slide 8: Missing vs. Empty Geometries: A Crucial Distinction**
  - **Why we are learning this:** geopandas makes a critical distinction that affects spatial operations. Treating them the same can lead to errors or incorrect results.[16]
  - **Missing Geometry (None):** The value is truly unknown. It's a placeholder. In spatial operations, it typically propagates (e.g., the area of None is NaN).[16]
  - **Empty Geometry (POLYGON EMPTY):** It is a valid geometry object, but it contains no coordinates. It can result from a spatial operation like the intersection of two non-overlapping polygons. Its area is 0.[16]
  - **Visualization:** A table with three rows.
    - Row 1: A polygon shape, isna()=False, is_empty=False, area=10.5.
    - Row 2: A None placeholder, isna()=True, is_empty=False, area=NaN.
    - Row 3: An "EMPTY" placeholder, isna()=False, is_empty=True, area=0.0.
- **Slide 9: Detecting Missing and Empty Geometries in Python**
  - **Concept:** Use specific geopandas methods to identify each case.

- **Code Example:**

```
Python
import geopandas as gpd
from shapely.geometry import Polygon

# Create a GeoSeries with valid, missing, and empty geometries
p1 = Polygon([(0, 0), (1, 0), (1, 1)])
p_empty = Polygon()
s = gpd.GeoSeries([p1, None, p_empty])

# Detect missing geometries
print("Missing geometries (isna):\n", s.isna())

# Detect empty geometries
print("\nEmpty geometries (is_empty):\n", s.is_empty)
```

- **Output Snippet:**

```
Missing geometries (isna):
 0    False
 1     True
 2    False
dtype: bool

Empty geometries (is_empty):
 0    False
 1    False
 2     True
dtype: bool
```

- **Summary:** s.isna() for missing, s.is_empty for empty.

- **Slide 10: Filtering for Valid Geometries**
  - **Concept:** Often, the goal is to work only with features that have valid, non-empty geometries. This can be done by combining the boolean masks from the previous slide.
  - **Code Example:**

```
Python
# s is the GeoSeries from the previous slide

# Filter out both missing AND empty geometries
valid_geometries = s[~(s.isna() | s.is_empty)]
print(valid_geometries)
```

  - **Output Snippet:**

```
0    POLYGON ((0 0, 1 0, 1 1, 0 0))
dtype: geometry
```

  - **Explanation:** The | operator is a logical OR. The ~ operator inverts the boolean mask, selecting only the False

values (i.e., the valid geometries).

- **Slide 11: Detecting Missing Data in Raster Datasets**
  - **Concept:** In raster data, missing values are not None but are represented by a specific numeric placeholder called a nodata value.
  - **Common nodata values:** <Null>, 0, -9999, or a very large positive/negative number.[4] The correct value is specified in the raster's metadata.
  - **Tool:** rasterio library in Python.
  - **Code Example:**

```python
import rasterio
import numpy as np

# Assume 'dem.tif' is a digital elevation model
with rasterio.open('dem.tif') as src:
    nodata_value = src.nodata
    print(f"The nodata value is: {nodata_value}")

    # Read the first band into a numpy array
    dem_array = src.read(1)

    # Count the number of nodata pixels
    missing_count = np.sum(dem_array == nodata_value)
    total_pixels = dem_array.size
    print(f"{missing_count} missing pixels out of {total_pixels}")
```

  - **Challenge:** Sometimes nodata is not properly defined in the metadata. In such cases, one might need to inspect the data visually or by sorting pixel values to identify an anomalous value used as a placeholder.[4]

- **Slide 12: Visual Diagnosis: Mapping the Missingness**
  - **Concept:** A critical best practice is to visualize the spatial distribution of missing data. This provides clues about the missingness mechanism.[1]
  - **Why we are learning this:** A map of missing values can reveal if the problem is systematic. Are missing values clustered in mountainous regions? Along the coast? In low-income neighborhoods? This spatial pattern is a strong hint that the data is not MCAR.
  - **Visualization:** Two maps side-by-side.
    - Map 1: "MCAR Pattern" - shows missing data points scattered randomly across the study area.
    - Map 2: "Clustered (Potential MNAR/MAR)" - shows missing data points concentrated in one specific region (e.g., a mountain range).
  - **Quote:** "Determine if missing data values are clustered or located on the periphery or in the core of your study area... Patterns can indicate that data is not missing at random."
- **Slide 13: Application: Diagnosing Cloud Cover in Satellite Imagery**
  - **Geo-Informatics Application:** Cloud cover in optical imagery is a classic example of spatially clustered, MNAR data. Clouds are more likely to form over certain topographies or during specific seasons.
  - **Process:**
    1. Use a cloud detection algorithm (or a pre-existing quality band) to create a binary cloud mask.
    2. Visualize this mask as a map.

3. Analyze the spatial pattern. Does it correlate with elevation? With coastlines?
    - **Real-world Connection:** A land cover classification model trained only on cloud-free data from valleys will likely fail when asked to classify land cover on frequently cloudy mountaintops. Diagnosing the spatial pattern of missingness is the first step to mitigating this critical model bias.
    - **Visualization:** A Sentinel-2 image on the left. A binary cloud mask of the same area on the right, showing large, contiguous white patches (clouds) that are clearly not random.
- **Slide 14: Best Practices for Initial Diagnosis**
    - **Know your nodata value:** Always check the raster metadata for the nodata placeholder.[4]
    - **Quantify the problem:** Determine the percentage of missing values. A common guideline is to be very cautious if more than 5% of the data is missing.[4]
    - **Map the missingness:** Always visualize the spatial pattern of missing data to get clues about the underlying mechanism.
    - **Check distributions:** Compare the statistical distribution (mean, std. dev., histogram) of variables for observations with and without missing data. Significant differences suggest the data is not MCAR.[4]
- **Slide 15: Summary of Detection & Diagnosis**
    - Missing data can be in attributes or geometries (vector) or represented by nodata values (raster).
    - geopandas distinguishes between missing (.isna()) and empty (.is_empty) geometries.
    - rasterio allows you to read the nodata value from metadata.
    - **Crucially, always visualize the spatial pattern of missingness.** This is the first and most important diagnostic step in any spatial analysis.

Slides 16-20: Deletion Methods

- **Slide 16: Handling Missing Data: Deletion Methods**
    - **Concept:** The simplest strategy for dealing with missing data is to remove it.
    - **Methods:**
        1. **Listwise Deletion (or Case Deletion):** Remove the entire row (observation) if it contains even a single missing value.
        2. **Pairwise Deletion:** For a specific analysis (e.g., a correlation between two variables), use only the rows where both variables are present.
        3. **Dropping Variables:** Remove an entire column (feature) if it has too many missing values.
    - **When to use:** Deletion is only truly safe if the data is MCAR and the number of missing values is small.[2]
- **Slide 17: Listwise Deletion in geopandas**
    - **Concept:** Listwise deletion is the default behavior in many statistical packages and is easy to implement.
    - **Code Example:**

```python
import pandas as pd
import numpy as np

# Create a sample DataFrame with a missing value
df = pd.DataFrame({'temp': [25, 28, np.nan, 30],
                   'humidity': })
```

```
print("Original DataFrame:\n", df)

# Apply listwise deletion
df_dropped = df.dropna()
print("\nDataFrame after dropna():\n", df_dropped)
```

- **Output Snippet:**

```
Original DataFrame:
   temp  humidity
0  25.0        60
1  28.0        65
2   NaN        68
3  30.0        70

DataFrame after dropna():
   temp  humidity
0  25.0        60
1  28.0        65
3  30.0        70
```

- **Explanation:** The entire row with the NaN temperature value was removed.

- **Slide 18: Pros and Cons of Listwise Deletion**
  - **Pros:**
    - **Simplicity:** Very easy to implement (.dropna()).
    - **Unbiased (under MCAR):** If the data is truly MCAR, the resulting analysis on the complete cases will be unbiased.
  - **Cons:**
    - **Loss of Power:** Reduces the sample size, which can decrease the statistical power of your models. This is a major issue with small datasets.
    - **Risk of Bias:** If the data is MAR or MNAR, listwise deletion will introduce systematic bias. You are removing a non-random subset of your data, skewing the results.
  - **Visualization:** A diagram showing a large dataset with a few missing values. An arrow points to a much smaller dataset after listwise deletion, with a caption: "Significant loss of data and statistical power."
- **Slide 19: Dropping Variables (Columns)**
  - **Concept:** If a particular feature (column) is missing a very large proportion of its values, it may contain little useful information and could be more harmful than helpful to a model.
  - **Guideline:** There's no hard rule, but if a column is >50-60% missing, consider dropping it unless it is critically important for the analysis.[3]
  - **Code Example:**

```
Python
# df is the DataFrame from the previous slide
# Let's add a new column with many missing values
df['wind_speed'] = [10, np.nan, np.nan, np.nan]
print("DataFrame with mostly-missing column:\n", df)
```

```
# Drop columns with more than 50% missing values
df_dropped_cols = df.dropna(thresh=len(df)*0.5, axis='columns')
print("\nDataFrame after dropping columns:\n", df_dropped_cols)
```

    - **Explanation:** thresh specifies the minimum number of non-NaN values required for a column to be kept. axis='columns' tells pandas to drop columns instead of rows.
- **Slide 20: Summary of Deletion Methods**
    - **Summary:** Deletion is a quick but blunt instrument.
    - **Best Practice:**
        - Only use listwise deletion if you have a large dataset, the percentage of missing values is very small (<5%), and you have strong reason to believe the data is MCAR.
        - Always report the number of observations you deleted from your analysis.
        - Consider imputation as a more robust alternative in most cases.
    - **Next Up:** Imputation - The art of filling in the gaps intelligently.

Slides 21-40: Imputation Methods

- **Slide 21: Imputation: Filling in the Gaps**
    - **Concept:** Imputation is the process of replacing missing data with substituted values. The goal is to create a complete dataset that can be used for modeling while preserving the statistical properties of the data as much as possible.
    - **Why we are learning this:** Imputation is generally preferred over deletion because it preserves all cases, retaining statistical power and potentially reducing bias if done correctly.
    - **Spectrum of Methods:**
        - **Simple:** Mean, Median, Mode
        - **Intermediate:** Regression, K-Nearest Neighbors
        - **Advanced/Spatial:** Spatial Neighbor Imputation, Areal Interpolation, Spatiotemporal Methods
    - **Flowchart:** A flowchart showing a "Missing Data" box leading to two paths: "Deletion" (with a caution sign) and "Imputation" (leading to the different methods).
- **Slide 22: Simple Imputation: Mean, Median, Mode**
    - **Concept:** The most basic imputation technique. Replace missing values in a column with the mean, median, or mode of the non-missing values in that same column.[3]
    - **Mean:** Use for normally distributed, continuous data without significant outliers.
    - **Median:** Use for skewed continuous data or data with outliers, as it is more robust.
    - **Mode:** Use for categorical data.
    - **Tool:** sklearn.impute.SimpleImputer.
    - **Challenge:** This method is easy but has significant drawbacks. It adds no new information, artificially reduces the variance of the data, and weakens the correlation between variables. It should be used with caution.
- **Slide 23: Code Example: SimpleImputer**
    - **Concept:** scikit-learn provides a convenient transformer for simple imputation that fits seamlessly into ML

pipelines.

- **Code Example:**

```
Python
from sklearn.impute import SimpleImputer
import numpy as np

# Sample data with a missing value
X = [, [np.nan, 3], ]

# Impute using the mean
imputer = SimpleImputer(strategy='mean')
imputer.fit(X)
X_imputed = imputer.transform(X)

print("Imputed Data:\n", X_imputed)
```

- **Output Snippet:**

```
Imputed Data:
 [[1. 4.5]
 [4. 3. ]
 [7. 6. ]]
```

- **Explanation:** The fit step calculates the mean of the first column as (1+7)/2 = 4. The transform step then fills the NaN with this value.

- **Slide 24: K-Nearest Neighbors (k-NN) Imputation**
  - **Concept:** A more sophisticated approach. A missing value for an observation is imputed using the average value from the *k-nearest neighbors* for that observation. "Closeness" is determined in the multi-dimensional feature space, not necessarily geographic space.[3]
  - **How it works:**
    1. For a row with a missing value, find the k other rows (that are complete) that are most similar to it based on the other available features.
    2. The missing value is then filled with the average (or mode) of that feature from these k neighbors.
  - **Tool:** sklearn.impute.KNNImputer.[17]
  - **Advantage:** Can be more accurate than simple imputation as it uses information from related features.
- **Slide 25: Code Example: KNNImputer**
  - **Concept:** Demonstrating how KNNImputer uses other features to make a more intelligent guess.
  - **Code Example:**

```
Python
from sklearn.impute import KNNImputer
import numpy as np

X = [[1, 2, np.nan], , [np.nan, 6, 5], ]

# Impute using 2 nearest neighbors
imputer = KNNImputer(n_neighbors=2)
X_imputed = imputer.fit_transform(X)
```

```
print(X_imputed)
```

- **Output Snippet:**
```
[[1. 2. 4.]
 [3. 4. 3.]
 [5.5 6. 5.]
 [8. 8. 7.]]
```

- **Explanation:** The first NaN was filled with 4.0 because its two nearest neighbors (rows 2 and 4) have values of 3 and 5 in that column. The second NaN was filled with 5.5 because its neighbors (rows 2 and 4) have values of 3 and 8 in the first column.

- **Slide 26: Spatial Imputation (Geoimputation)**
  - **Concept:** This is where we explicitly use geography. Based on Tobler's First Law ("everything is related to everything else, but near things are more related than distant things"), we can use the values of *geographic neighbors* to estimate a missing value.
  - **Why we are learning this:** For spatial data, this is almost always a better approach than non-spatial methods like mean or even k-NN imputation, because it respects the underlying spatial structure of the data.
  - **Methods:**
    - Average of N nearest neighbors.
    - Average of neighbors within a certain distance.
    - Median, Minimum, or Maximum of neighbors.[1]
  - **Visualization:** A map with a missing polygon. Arrows from adjacent polygons point to it, with their values (e.g., 10, 12, 14). The imputed value in the center is calculated as the average (12).
- **Slide 27: Choosing a Spatial Imputation Method**
  - **Concept:** The choice of statistic (mean, median, min, max) is a modeling decision that depends on the context and desired outcome.
  - **Use Average/Median:** For a typical, representative estimate.
  - **Use Minimum:** When you want to be conservative and underestimate (e.g., estimating social benefits, where overestimation could deplete budgets).[4]
  - **Use Maximum:** When you want to err on the side of caution and overestimate (e.g., estimating childhood lead poisoning risk, where underestimation could lead to public health failures).
  - **Application Context:** A public health analyst mapping disease risk should use the max of neighboring areas to fill a gap, ensuring potential hotspots are not missed. This improves the real-world utility of the model by prioritizing safety.
- **Slide 28: Areal Interpolation for Polygon Data**
  - **Concept:** A geostatistical method specifically for filling gaps in polygon data. It predicts values for polygons with missing data by using the spatial variation of data across the entire study area.[5]
  - **How it works:** It treats the polygon data as a continuous surface, interpolates this surface, and then extracts the predicted value for the missing polygon from that surface.
  - **Tool:** Available in GIS software like ArcGIS Pro (Geostatistical Wizard) and can be implemented in R.
  - **Application:** Ideal for demographic or socioeconomic data at the census tract or county level, where data for a few administrative units might be missing.

    - **Visualization:** A map of Polish *powiaty* (counties) showing population data. Several counties are blank (missing data). An arrow points to a second map where the gaps are filled in, creating a complete surface.[5]
- **Slide 29: Best Practices for Imputation**
    - **Create a Missingness Indicator:** Before imputing, create a new binary column that flags which rows were originally missing. This can be used as a feature in the model itself, which can sometimes capture important patterns.
    - **Don't Estimate from Estimates:** Be sure not to fill in missing values with values you have already filled in. This compounds error.[4]
    - **Check Distributions:** Compare the histogram and descriptive statistics (mean, std. dev.) of the variable before and after imputation. The best imputation method will result in a distribution that is most similar in shape to the original.[4]
    - **Map Before and After:** Just as you map the missingness, map the final imputed data to ensure the filled values are spatially plausible and don't create artificial "bulls-eyes" or other artifacts.
- **Slide 30: Geo-Informatics Deep Dive: Cloud Removal in Hyperspectral Imagery**
    - **The Challenge:** Clouds and their shadows are a major source of missing data in optical and hyperspectral remote sensing. Simple imputation is not sufficient because large, contiguous areas are missing, and the underlying surface can be highly heterogeneous.[18]
    - **Visualization:** A hyperspectral image with a large white cloud and a dark shadow next to it, completely obscuring the land cover underneath.
- **Slide 31: Traditional Methods for Cloud Removal**
    - **1. Temporal Compositing:**
        - **Concept:** Create a "best pixel" composite image from a time-series of images. For each pixel location, find all observations over a period (e.g., one month) and select the one that is most cloud-free.
        - **Pros:** Simple and effective for phenomena that do not change quickly (e.g., land cover).
        - **Cons:** Not suitable for dynamic phenomena (e.g., crop growth stages, flood monitoring). Can create mosaic-like artifacts.[6]
    - **2. Spatial Interpolation:**
        - **Concept:** Use spatial filters or interpolation methods (like IDW or Kriging) to fill in small gaps from surrounding valid pixels.[6]
        - **Pros:** Works for small, isolated cloudy pixels.
        - **Cons:** Fails for large cloud patches; produces blurry results.
- **Slide 32: Advanced Spatiotemporal Imputation: HANTS & DINEOF**
    - **Why we are learning this:** When gaps are large and persistent, we need methods that leverage both spatial and temporal patterns.
    - **HANTS (Harmonic ANalysis of Time Series):**
        - **Concept:** Models the time series of each pixel's reflectance as a sum of sine and cosine waves (harmonics). It fits this curve to the valid data points and uses the fitted curve to predict values for the cloudy (outlier) pixels.[6]
        - **Best for:** Data with clear seasonality, like vegetation indices (NDVI).
    - **DINEOF (Data INterpolating Empirical Orthogonal Functions):**
        - **Concept:** A powerful method from oceanography. It uses Empirical Orthogonal Functions (EOFs), which are analogous to Principal Components, to identify the dominant modes of spatiotemporal variability in an image time series. It reconstructs the data using a limited number of EOFs, which effectively fills the gaps based on the learned dominant patterns.[7]
        - **Best for:** Large, coherent gaps in spatiotemporal datasets. Computationally intensive.[7]
- **Slide 33: Application of HANTS and DINEOF**

- **Geo-Informatics Application:** Monitoring ocean color (chlorophyll concentration) is often hampered by clouds. A researcher used DINEOF on a time-series of MODIS imagery over the Argentinean sea, where huge regions were missing data for 3 months every year. DINEOF successfully reconstructed the chlorophyll data, enabling a continuous analysis of ocean productivity.[6]
- **Real-world Connection:** By filling these gaps, scientists can build more accurate models of the marine carbon cycle and fisheries health, which would be impossible with incomplete data.

- **Slide 34: State-of-the-Art: Deep Learning for Cloud Removal**
  - **Concept:** Modern approaches use deep learning, particularly Generative Adversarial Networks (GANs) and Transformers, to "inpaint" or reconstruct cloud-covered regions.
  - **How it works:** These models are trained on pairs of cloudy and corresponding clear images. They learn the complex statistical relationship between the cloud-free image content and the surrounding context, allowing them to generate realistic, physically plausible pixels to fill the cloud gaps.[18]
  - **Visualization:** A diagram showing a cloudy image being fed into a "black box" deep learning model, which then outputs a clear, reconstructed image.
- **Slide 35: Leveraging Auxiliary Data: SAR-Optical Fusion**
  - **Concept:** The most powerful reconstruction methods often use auxiliary data from sensors that are not affected by clouds. Synthetic Aperture Radar (SAR) is a prime example.[18]
  - **How it works:**
    1. Acquire a cloudy optical/hyperspectral image and a contemporaneous SAR image of the same area.
    2. Train a deep learning model (e.g., a GAN or Transformer) to learn the mapping: f(Cloudy_Optical, SAR) -> Clear_Optical.
    3. The model learns to use the texture and structural information from the SAR data (which penetrates clouds) to guide the reconstruction of the spectral information in the optical image.[18]
  - **Real-world Connection:** This technique is transformative for monitoring in persistently cloudy regions like the tropics. It allows for continuous land cover monitoring for applications like deforestation tracking, which would otherwise have large temporal gaps.
- **Slide 36: Handling Missing Data in SAR Imagery**
  - **The Challenge:** While SAR sees through clouds, its data can still be missing or corrupted. This can happen due to sensor interruptions (creating gaps in the raw data stream) or processing errors.[20] In InSAR time-series analysis for deformation monitoring (e.g., landslides), gaps can occur if satellite acquisitions are missed or if surface changes cause decorrelation.[21]
  - **Methods:**
    - For missing raw data, specialized signal processing techniques are used to reconstruct the image.[20]
    - For time-series InSAR, deep learning models like sequence-to-sequence LSTMs are being developed. These models learn the non-linear temporal deformation trend from the available data points and can predict the deformation during the missing intervals, accounting for external factors.[21]
  - **Application:** This is critical for long-term infrastructure and landslide monitoring, ensuring that a continuous deformation history is available for risk assessment.
- **Slide 37: Case Study: Landslide Monitoring with InSAR**
  - **Scenario:** A landslide is being monitored using Sentinel-1 SAR data. A satellite is decommissioned, creating a long data gap.
  - **Problem:** Conventional interpolation methods fail because they don't capture the complex, non-linear movement of the landslide and can't account for external triggers like rainfall.
  - **Solution:** A Seq2Seq LSTM model is trained on the historical InSAR time-series data. It learns the temporal dynamics of the landslide's movement.
  - **Result:** The model successfully imputes the missing deformation values during the data gap, providing a

continuous and reliable record for hazard analysis. This outperforms traditional methods significantly.[21]
    - **Visualization:** A graph showing InSAR deformation time-series data points. There is a large gap in the middle. A dotted line shows a simple linear interpolation, while a solid curved line shows the more realistic imputation from the LSTM model.
- **Slide 38: Deeper Understanding: Imputation as Feature Engineering**
    - The choice of an imputation method is a form of implicit feature engineering.
    - **Mean Imputation:** Assumes no spatial structure and actively reduces data variance.
    - **Spatial Imputation:** Explicitly encodes Tobler's First Law into the dataset before modeling. A model trained on this data is already primed to recognize spatial dependency.
    - **Advanced Methods (DINEOF, Deep Learning):** These are forms of representation learning. They learn complex spatiotemporal patterns and embed that learned structure back into the data.
    - **Takeaway:** Imputation is not just "fixing" data; it's the first and one of the most fundamental modeling decisions you make.
- **Slide 39: Deeper Understanding: The Importance of the Missingness Mechanism**
    - The concepts of MCAR, MAR, and MNAR are not just academic. They have profound real-world consequences.
    - If data is MNAR (e.g., clouds over mountains), and you simply delete the cloudy data, you are training a model that has never seen a mountain. Its predictions for mountainous terrain will be unreliable.
    - This connects the abstract statistical concept of MNAR to the operational reliability of a geo-engineering model. An agricultural model that has never seen data from a drought (because sensors failed) cannot predict crop yields during a drought.
    - **Takeaway:** Always question *why* data is missing. Mapping the missingness is your first tool for this diagnosis.
- **Slide 40: Summary of Imputation**
    - Imputation is generally preferred over deletion.
    - The method should match the data type and underlying process.
    - **For spatial data, use spatial methods.** Geoimputation is a robust starting point.
    - For time-series imagery with large gaps, advanced methods like HANTS, DINEOF, or deep learning are necessary.
    - Leveraging auxiliary data (like SAR for optical) is a powerful strategy.
    - Always document and evaluate the impact of your imputation strategy.

Slides 41-60: Case Study - Land Cover Data Cleaning for ML Models

(This section will integrate the concepts from Modules 1.1, 1.2, and 1.3 into a cohesive workflow)

- **Slide 41: Case Study: Preparing Data for Land Cover Classification**
    - **Objective:** To prepare a set of diverse spatial datasets for training a machine learning model to classify land cover.
    - **Input Datasets:**
        1. **Sentinel-2 Image (10m):** Optical data, but with 15% cloud cover.
        2. **SRTM DEM (30m):** Elevation data.
        3. **Census Tracts (Vector):** Polygons with population density data, but some tracts are missing values.
    - **Goal:** Create a complete, aligned raster stack where every pixel has a value for all features (reflectance bands, elevation, population density).
- **Slide 42: Workflow Overview**
    - **Flowchart:**
        1. **Handle Missing Vector Data:** Impute missing population density in census tracts.
        2. **Rasterize Vector Data:** Convert the complete population density polygons to a 10m raster.
        3. **Align Raster Grids:** Resample the 30m DEM to match the 10m grid of the Sentinel-2 image.

4. **Handle Missing Raster Data:** Reconstruct the cloud-covered pixels in the Sentinel-2 image.
5. **Normalize & Standardize:** Scale all feature layers to be comparable.
6. **Stack Layers:** Combine all processed rasters into a final multi-band data stack ready for ML.

- **Slide 43: Step 1: Handling Missing Population Density (Vector)**
  - **Problem:** 5 out of 100 census tracts are missing POP_DENSITY.
  - **Diagnosis:** Map the missing tracts. They are scattered, suggesting an MAR or MCAR pattern.
  - **Method:** Spatial Imputation. We will use the average population density of neighboring (Queen contiguity) tracts.
  - **Python Pseudo-code:**

```
Python
# 1. Load census tracts with geopandas
tracts = gpd.read_file("census_tracts.shp")
# 2. Identify missing tracts
missing_tracts = tracts.isnull()]
# 3. For each missing tract:
#    a. Find its neighbors using pysal
#    b. Calculate the mean density of the neighbors
#    c. Fill the missing value with the mean
```

  - **Visualization:** Map showing the census tracts, with the 5 missing ones highlighted. Arrows show the neighbors used for imputation for one of them.
- **Slide 44: Step 2: Rasterizing the Vector Data**
  - **Problem:** The ML model will operate on a pixel-by-pixel basis. We need to convert the polygon-based population density into a raster grid that aligns with our imagery.
  - **Tool:** rasterio.features.rasterize.
  - **Process:** Create a new 10m resolution raster grid that matches the extent and CRS of the Sentinel-2 image. "Burn" the population density values from the polygons onto this grid.
  - **Visualization:** A map of the census polygons on the left. An arrow points to a rasterized grid of the same data on the right.
- **Slide 45: Step 3: Aligning Raster Grids (Resampling)**
  - **Problem:** The DEM is at 30m resolution, but our target grid is 10m. We need to upsample the DEM.
  - **Data Type:** Elevation is a continuous variable.
  - **Method Choice:** Nearest neighbor would create blocky, unrealistic terrain. We should use Bilinear or Cubic Convolution. Bilinear is faster and safer (won't create values outside the original range).[13] We choose Bilinear.
  - **Tool:** rasterio.
  - **Visualization:** A zoomed-in view of the 30m DEM (pixelated) next to the 10m resampled DEM (smoother).
- **Slide 46: Code Example: Resampling with rasterio**
  - **Concept:** rasterio's .read() method can resample on the fly by specifying a different out_shape.
  - **Code:**

```
Python
import rasterio
from rasterio.enums import Resampling

with rasterio.open('sentinel2_10m.tif') as s2_src:
    target_profile = s2_src.profile

with rasterio.open('srtm_30m.tif') as dem_src:
```

```
    # Read and resample to match the shape of the 10m image
    dem_resampled = dem_src.read(
        out_shape=(target_profile['height'], target_profile['width']),
        resampling=Resampling.bilinear
    )
```

- **Explanation:** We read the 30m DEM but tell rasterio to output it into an array with the dimensions of our 10m target grid, using bilinear interpolation to calculate the new pixel values.

- **Slides 47-50: Step 4: Reconstructing Cloudy Pixels (Raster Imputation)**
  - **Slide 47: Problem - Cloud Cover**
    - Our Sentinel-2 image has significant cloud cover, which is MNAR. Simple interpolation will fail.
    - We need a more advanced method. For this case study, we'll assume we have a contemporaneous SAR image available.
  - **Slide 48: Method - SAR-Optical Fusion**
    - We will use a (pre-trained) deep learning model that takes the cloudy Sentinel-2 bands and the corresponding SAR image as input, and outputs a reconstructed, cloud-free Sentinel-2 image.
    - The model leverages the SAR texture and structure to fill in the optical data.
  - **Slide 49: Conceptual Workflow**
    - **Diagram:**
      1. Input: Cloudy S2 image + SAR image.
      2. Box labeled "Trained Fusion Model (e.g., Pix2Pix GAN)".
      3. Output: Reconstructed Cloud-Free S2 Image.
  - **Slide 50: Result Visualization**
    - Side-by-side comparison of the original cloudy Sentinel-2 image and the reconstructed cloud-free version. The land cover under the former cloud is now visible.
- **Slides 51-55: Step 5: Normalization and Standardization**
  - **Slide 51: The Need for Scaling**
    - Our features now have vastly different ranges:
      - Sentinel-2 Reflectance: 0 - ~10000 (unitless integer)
      - DEM: meters (e.g., 50 - 1500)
      - Population Density: persons/km² (e.g., 10 - 5000)
    - For many classifiers (SVM, Neural Networks), these different scales will cause problems.
  - **Slide 52: Pre-Scaling - Calibration**
    - **Concept:** First, convert Sentinel-2 reflectance to a physical scale. We will normalize the integer values to floating-point reflectance by dividing by a scaling factor (e.g., 10000). This is a form of domain-specific normalization.
    - **Code:** s2_reflectance = s2_array / 10000.0
  - **Slide 53: Statistical Scaling**
    - **Method Choice:** We will use Standardization (StandardScaler) because it centers the data and is a robust choice for many algorithms. It is less sensitive to the exact min/max than Min-Max scaling.
    - **Process:** We will fit a StandardScaler on the training pixels for *all* features (all S2 bands, DEM, population) and then apply it to the entire dataset.
  - **Slide 54: Code Example: Scaling a Raster Stack**

```
Python
from sklearn.preprocessing import StandardScaler

# Assume `data_stack` is a (bands, height, width) numpy array
```

```
# Reshape for scikit-learn: (n_samples, n_features)
n_pixels = data_stack.shape * data_stack.shape
data_reshaped = data_stack.reshape((data_stack.shape, -1)).T

scaler = StandardScaler()
# Fit and transform the data
data_scaled_reshaped = scaler.fit_transform(data_reshaped)

# Reshape back to image format
data_scaled = data_scaled_reshaped.T.reshape(data_stack.shape)
```

    - **Slide 55: Visualization of Scaled Data**
        - Show histograms of the DEM and a single Sentinel-2 band before and after standardization. Before: different ranges and means. After: both centered at 0 with a standard deviation of 1.
- **Slides 56-58: Step 6: Stacking and Final Output**
    - **Slide 56: Creating the Final Data Stack**
        - **Concept:** Combine all the processed, aligned, and scaled layers into a single multi-band raster file.
        - **Process:** Use numpy.vstack to stack the 2D arrays of each feature into a 3D array (bands, height, width).
        - **Output:** A single GeoTIFF file that is ready to be used for training a machine learning model.
    - **Slide 57: Visualizing the Data Stack**
        - Show the final data stack as a "cube" of data. The x and y dimensions are spatial, and the z dimension represents the different features (S2 Band 1, S2 Band 2,..., DEM, Pop Density).
    - **Slide 58: Case Study Summary**
        - We started with disparate, imperfect datasets.
        - We imputed missing vector data using spatial neighbors.
        - We resampled and rasterized data to create a common 10m grid.
        - We used an advanced technique to reconstruct data missing due to clouds.
        - We scaled all features to make them comparable.
        - The result is a clean, complete, and analysis-ready data stack.
- **Slides 59-60: Summary and Key Takeaways**
    - **Slide 59: Summary of Workflow**
        - Reiterate the flowchart from Slide 42. Emphasize that preprocessing is a sequential, logical process.
    - **Slide 60: Key Takeaways**
        - Data cleaning is not optional; it's the foundation of reliable spatial ML.
        - Always choose your method based on the data type (vector/raster, continuous/discrete) and the underlying spatial/temporal process.
        - For remote sensing data, domain-specific calibration (e.g., radiometric) must precede statistical scaling.
        - Visualization is a key tool at every step: to diagnose problems and to verify solutions.

Slides 61-80: Summary, Challenges, and Self-Assessment

- **Slide 61: Topic Summary: Data Cleaning & Transformation**
    - We covered three core areas:
        1. **Handling Missing Data:** From simple deletion to advanced spatiotemporal imputation for satellite imagery.
        2. **Normalization & Standardization:** Making features comparable, including the critical step of radiometric calibration for remote sensing data.
        3. **Interpolation & Resampling:** Aligning datasets to a common spatial grid.

- The goal is to produce an **Analysis-Ready Data (ARD)** stack.

- **Slide 62: Challenge: The Curse of Dimensionality in Hyperspectral Data**
  - **Concept:** Hyperspectral sensors collect data in hundreds of contiguous bands. This high dimensionality presents several challenges [23]:
    - **Computational Cost:** Processing hundreds of bands is slow and memory-intensive.
    - **Hughes Phenomenon:** With a fixed number of training samples, classifier performance can actually decrease as the number of features (bands) increases beyond a certain point.
    - **Multicollinearity:** Adjacent bands are highly correlated, providing redundant information.
  - **Solution Preview:** This is a major motivation for feature extraction and dimensionality reduction techniques (like PCA), which will be covered later in the course.
- **Slide 63: Challenge: Speckle Noise in SAR Data**
  - **Concept:** SAR images are inherently affected by "speckle," a granular noise that results from the coherent imaging process. It is multiplicative, meaning its intensity is proportional to the signal strength.[25]
  - **Impact:** Speckle degrades image quality, makes interpretation difficult, and can confuse classification algorithms.
  - **Solution Preview:** Speckle filtering (e.g., Lee filter, Frost filter) is a crucial preprocessing step for most SAR applications. Averaging methods like multilooking can also reduce speckle at the cost of spatial resolution.[25]
- **Slide 64: Challenge: The Modifiable Areal Unit Problem (MAUP)**
  - **Concept:** The results of a spatial analysis can change depending on the scale (e.g., census tracts vs. counties) or zoning (how the boundaries are drawn) of the geographic units used.
  - **Impact:** A pattern of clustering that is significant at the tract level might disappear or change when aggregated to the county level.
  - **Relevance to Preprocessing:** The choice of spatial resolution for resampling or the units for rasterization is a form of MAUP. There is no single "correct" scale; the scale must be chosen based on the phenomenon being studied.
- **Slide 65: Challenge: Data Leakage in Preprocessing**
  - **Concept:** Data leakage occurs when information from the test set "leaks" into the training process, leading to overly optimistic performance estimates.
  - **How it happens in preprocessing:** If you calculate the mean for imputation or the mean/std. dev. for standardization using the *entire dataset* before splitting into training and test sets, the training process has already "seen" the test data.
  - **Best Practice:** Always split your data into training and testing sets *first*. Then, fit your preprocessors (imputers, scalers) *only* on the training data, and use .transform() to apply the learned parameters to both the training and test data.[28] scikit-learn Pipelines make this easy to manage correctly.
- **Slides 66-75: Self-Assessment Quiz**
  - **Slide 66: Question 1:** You have a land cover raster (categorical data) at 10m resolution that you need to align with a 30m raster. Which resampling method should you use and why? (Answer: Nearest Neighbor or Majority, because they do not create new, invalid category values).
  - **Slide 67: Question 2:** A feature in your dataset has a skewed distribution with several extreme outliers. Would StandardScaler or MinMaxScaler be more affected? What is a more robust alternative? (Answer: MinMaxScaler would be more affected. RobustScaler is a better alternative).
  - **Slide 68: Question 3:** What is the fundamental difference between k-NN Imputation and Spatial Imputation? (Answer: k-NN finds neighbors in feature space; Spatial Imputation finds neighbors in geographic space).
  - **Slide 69: Question 4:** Why is it incorrect to apply StandardScaler directly to the raw Digital Number (DN) values of a SAR image for a scientific application? (Answer: DNs are not physical values. Radiometric calibration to convert DNs to backscatter (σ^o) must be performed first).

- **Slide 70: Question 5:** You observe that missing data for air quality sensors only occurs on days with very high humidity. What type of missingness mechanism is this likely to be? (Answer: MAR, as the missingness is predictable from another observed variable, humidity).
- **Slide 71: Question 6:** What is the primary advantage of Kriging over IDW for interpolation? (Answer: Kriging provides an estimate of the prediction error/uncertainty).
- **Slide 72: Question 7:** Explain in one sentence the purpose of row-standardizing a spatial weights matrix. (Answer: It transforms the weights so that the spatial lag becomes a weighted average of the neighboring values).
- **Slide 73: Question 8:** What is the main drawback of using mean imputation? (Answer: It artificially reduces the variance of the dataset).
- **Slide 74: Question 9:** You are trying to fill a large gap in a time-series of ocean temperature satellite images caused by seasonal clouds. Would simple spatial interpolation be effective? Name a more suitable advanced method. (Answer: No. DINEOF or HANTS would be more suitable).
- **Slide 75: Question 10:** What is data leakage and how can you avoid it when scaling your data? (Answer: It's when test set information influences the training process. Avoid it by splitting data first, then fitting the scaler only on the training set).

- **Slides 76-80: Further Reading and Next Steps**
  - **Slide 76:** Recommended Python Libraries Review (Logos and short descriptions for GeoPandas, Rasterio, Scikit-learn, PySAL).
  - **Slide 77:** Advanced Topics (Brief mention of more complex topics like non-linear dimensionality reduction and advanced spatiotemporal modeling).
  - **Slide 78:** Further Reading (List of key papers or book chapters on spatial data preprocessing, e.g., from sources like [1]).
  - **Slide 79:** Practical Assignment Overview (Brief description of the upcoming lab/assignment where students will apply these techniques to a real-world dataset).
  - **Slide 80:** Next Chapter Preview: Feature Engineering for Spatial Data (A teaser for the next topic, explaining that now that the data is clean, we can start creating powerful predictive features from it).

Topic 2: Feature Engineering for Spatial Data

Topic Overview: Learning Goals and Conceptual Background

Conceptual Background:
Once data is cleaned and transformed, the next critical phase is feature engineering. This is often described as more of an art than a science and is frequently the key differentiator between a good machine learning model and a great one.29 Feature engineering is the process of using domain knowledge and creativity to construct new input variables (features) from raw data that help machine learning algorithms perform better.30 In the context of spatial data, this means translating geography—the complex web of locations, distances, densities, and neighborhood contexts—into a numerical format that standard, spatially-unaware algorithms like Random Forests or Gradient Boosting can understand and leverage. This topic delves into the methods for explicitly encoding spatial relationships and context into the feature

set, thereby enabling powerful, general-purpose ML models to "see" the geography of the problem.
Learning Goals:
Upon completing this topic, students will be able to:

1. Derive new features based on vector geometry (area, shape), proximity (distance to nearest feature), and density (counts within a buffer).
2. Understand the concept of texture in raster data and extract texture features using the Gray-Level Co-occurrence Matrix (GLCM).
3. Apply domain-specific feature engineering techniques for advanced Geo-Informatics data types like SAR and hyperspectral imagery.
4. Define spatial autocorrelation and understand its importance as summarized by Tobler's First Law of Geography.
5. Construct and interpret a spatial weights matrix (W) using different criteria (contiguity, distance) with the PySAL library.
6. Calculate and interpret global spatial autocorrelation statistics like Moran's I and Geary's C.
7. Create a spatial lag feature and understand its role as a powerful summary of local context.
8. Incorporate more advanced forms of spatial context, such as neighborhood summary statistics and graph-based features, into a machine learning workflow.

Key Concepts Table: Feature Engineering for Spatial Data

Category	Method/Technique	Description	Application Example
Vector Features	Geometric Features	Deriving features from the shape and size of geometries, such as area, perimeter, or compactness ratio.[32]	Calculating the area of a parcel to include in a property value model.
	Proximity Features	Measuring the distance from each feature to the nearest feature of another type (e.g., nearest park, nearest hospital).[33]	Creating a "distance_to_transit" feature for a model predicting house prices.
	Density/Count Features	Counting the number of points or features within a defined neighborhood or buffer around a	Counting the number of crime incidents within a 500m radius of each residential

		target feature.[33]	property.
Raster Features	Texture (GLCM) Features	Quantifying the spatial arrangement of pixel values using statistics derived from a Gray-Level Co-occurrence Matrix (e.g., Contrast, Homogeneity).[36]	Using texture from a SAR image to differentiate between smooth water and rough forest canopy.
Spatial Context	Spatial Weights Matrix (W)	A formal N x N matrix that defines the "neighborhood" for each of the N features in a dataset, based on contiguity or distance.[37]	A prerequisite for calculating spatial autocorrelation and spatial lag.
	Spatial Autocorrelation	A statistical measure (e.g., Moran's I, Geary's C) of the degree to which nearby features have similar values.[39]	Testing whether high-crime areas are significantly clustered together.
	Spatial Lag	The weighted average of a variable's values in the neighborhood of a feature, as defined by the spatial weights matrix W.[41]	Creating a feature for "average neighborhood income" to help predict individual household income.
	Spectral-Spatial Features	For hyperspectral data, combining the spectral information of a pixel with spatial information from its surrounding pixels (e.g., using a 3D CNN).[42]	Improving land cover classification accuracy by reducing "salt-and-pepper" noise.

Detailed Explanations and Slide Content

Module 2.1: Extracting Spatial Features

Slides 81-85: Introduction to Spatial Feature Engineering

- **Slide 81: Title Slide: Feature Engineering for Spatial Data**
  - Chapter 2: Pre-processing Spatial Data for Machine Learning
  - Topic 2.1: Extracting Spatial Features
  - Your Name, Course Title, University
- **Slide 82: The "Why": Making Space Explicit for Non-Spatial Models**
  - **Concept:** Standard machine learning algorithms like Random Forest, XGBoost, and Support Vector Machines are incredibly powerful but are inherently "spatially blind." They operate on tabular data and have no native understanding of location, distance, or neighborhood.[30]
  - **Why we are learning this:** Spatial feature engineering is the bridge that allows us to use these state-of-the-art algorithms on spatial problems. We translate geographical concepts into numerical columns in a feature table. The algorithm doesn't know it's "space," but it can learn the patterns from the numbers we provide.
  - **Visualization:** A diagram showing a map with points and polygons on the left. An arrow labeled "Spatial Feature Engineering" points to a standard data table (matrix) on the right. The table has columns like feature_1, feature_2, distance_to_river, neighbor_count. An arrow from the table points to a "black box" ML model.
- **Slide 83: A Hypothesis-Driven Process**
  - Feature engineering is not a blind, mechanical task. Every feature you create should represent a hypothesis about the phenomenon you are modeling.
  - **Hypothesis:** "Proximity to public transit increases property value."
    - **Feature:** distance_to_nearest_metro_station.
  - **Hypothesis:** "Areas with a high density of restaurants are more likely to be commercial zones."
    - **Feature:** restaurant_count_in_500m_buffer.
  - **Hypothesis:** "Rougher land surfaces (like forests) have a different SAR backscatter signature than smoother surfaces (like fields)."
    - **Feature:** GLCM_contrast.
  - The ML model's job is then to test these hypotheses by determining the predictive power of each feature.
- **Slide 84: Categories of Spatial Features**
  - We will explore several categories of engineered features:
    1. **Vector-Based Features:** Derived from point, line, and polygon geometries.
       - **Geometric Features:** Based on the shape and size of individual features.
       - **Proximity Features:** Based on distances between features.
       - **Density/Count Features:** Based on the characteristics of a feature's neighborhood.
    2. **Raster-Based Features:** Derived from grid-based data.
       - **Texture Features:** Based on the spatial arrangement of pixel values.
    3. **Domain-Specific Features:** Specialized techniques for SAR, Hyperspectral, and other advanced data types.

- **Slide 85: The Importance of Scale**
  - The spatial scale at which you engineer a feature is as important as the feature itself.
  - **Example:** A feature for cafe_density might be most predictive within a 500m "walkable" radius, while a feature for school_district_quality is relevant over a much larger area.
  - **Best Practice: Multi-Scale Feature Engineering.** Instead of choosing one scale, create features at multiple scales (e.g., crime_count_100m, crime_count_500m, crime_count_1km). This allows the model to learn which spatial scale is most influential for the problem at hand.
  - **Visualization:** A central point with three concentric buffer rings around it (100m, 500m, 1km), each labeled with the count of other points inside it.

Slides 86-95: Vector-Based Features with GeoPandas

- **Slide 86: Geometric Features**
  - **Concept:** These are the simplest features to create, derived directly from the geometry column of a GeoDataFrame using built-in attributes.[32]
  - **Key Attributes:**
    - .area: The area of a polygon.
    - .length: The length of a line.
    - .centroid: The geometric center of a feature.
    - .boundary: The line that represents the boundary of a polygon.
  - **Application:** In a model predicting agricultural yield per parcel, the .area of the parcel is a fundamental feature.
- **Slide 87: Code Example: Calculating Geometric Features**
  - **Tool:** geopandas.
  - **Code:**

```Python
import geopandas as gpd

# Load NYC boroughs data
gdf = gpd.read_file(gpd.datasets.get_path('nybb'))
# NOTE: Data is in a projected CRS (units are feet). Area will be in sq. feet.
gdf = gdf.set_index('BoroName')

# Calculate area in square kilometers
gdf['area_sqkm'] = gdf.area / 1e6 * (0.3048**2)

# Calculate the centroid of each borough
gdf['centroid'] = gdf.centroid

print(gdf[['area_sqkm', 'centroid']].head())
```

  - **Output Snippet:** A pandas DataFrame showing the new area_sqkm and centroid columns for each borough.
- **Slide 88: Advanced Geometric Feature: Compactness**
  - **Concept:** We can combine basic geometric features to create more descriptive ones. The "isoperimetric quotient" or "polygons-circularity index" measures how compact or circular a polygon is, compared to a

perfect circle.
    - **Formula:** $Compactness = \frac{4 \pi \times Area}{Perimeter^2}$
    - **Interpretation:** A value of 1.0 means a perfect circle. A value close to 0 means a very irregular, non-compact shape.
    - **Application:** In urban morphology, this could help distinguish between planned, grid-like neighborhoods and organically grown, irregular ones. In ecology, it can describe the shape of habitat patches (less compact shapes have more "edge effect").
- **Slide 89: Proximity Features: Measuring Distance**
    - **Concept:** Proximity features measure the distance from a target observation to another location or feature. This is fundamental for modeling concepts of accessibility and influence.[34]
    - **Methods:**
        - **Distance to a single point:** Use the .distance() method to calculate the distance from every feature in a GeoDataFrame to a single shapely Point.[32]
        - **Distance to the nearest of many features:** Use a spatial join. geopandas.sjoin_nearest is a powerful function for this.
    - **Important Precaution:** Distance calculations must be performed in a **projected Coordinate Reference System (CRS)** where units are meters or feet. Calculating distance on geographic coordinates (latitude/longitude) will give incorrect results.
- **Slide 90: Code Example: Distance to Nearest Feature**
    - **Scenario:** For each Airbnb location in San Diego, find the distance to the nearest Point of Interest (POI, e.g., a restaurant or bar).
    - **Tools:** geopandas.
    - **Code:**

```python
import geopandas as gpd

# Assume 'airbnbs' and 'pois' are GeoDataFrames in the same projected CRS
# airbnbs: Point geometry for each rental
# pois: Point geometry for each restaurant/bar

# Perform the nearest neighbor spatial join
# This will find the single nearest POI for each Airbnb
airbnbs_with_nearest_poi = gpd.sjoin_nearest(airbnbs, pois, distance_col="dist_to_poi")

print(airbnbs_with_nearest_poi[['geometry', 'dist_to_poi']].head())
```

    - **Explanation:** sjoin_nearest merges the two GeoDataFrames, adding attributes of the nearest pois feature to each airbnbs feature. The distance_col argument automatically calculates the distance and adds it as a new column.
- **Slide 91: Density and Count Features: The Buffer-and-Count Method**
    - **Concept:** A common and intuitive way to quantify the "neighborhood" of a feature.
    - **Process:**
        1. **Buffer:** Create a polygon (a buffer) of a specified radius (e.g., 500 meters) around each feature of interest.
        2. **Spatial Join:** Perform a spatial join between the buffers and another set of features (e.g., crime incidents).
        3. **Count:** Group the results by the original feature and count how many incidents fall within each buffer.[33]
    - **Visualization:** A diagram showing a point, a 500m buffer circle around it, and several other points inside the circle being counted.

- **Slide 92: Code Example: Buffer-and-Count**
  - **Scenario:** For each Airbnb, count the number of POIs within a 500-meter radius.
  - **Tools:** geopandas.
  - **Code:**

```python
# 1. Create 500m buffers around each Airbnb
airbnb_buffers = airbnbs.copy()
airbnb_buffers['geometry'] = airbnbs.geometry.buffer(500)

# 2. Perform a spatial join to find which POIs are within which buffer
# The 'predicate="within"' ensures we only join POIs inside the buffers
joined = gpd.sjoin(pois, airbnb_buffers, how="inner", predicate="within")

# 3. Group by the Airbnb's unique ID and count the POIs
poi_counts = joined.groupby('airbnb_id').size()

# Merge the counts back to the original Airbnb data
airbnbs['poi_count_500m'] = airbnbs.index.map(poi_counts).fillna(0)
```

  - **Explanation:** This workflow creates a new feature, poi_count_500m, that every standard ML model can use.
- **Slide 93: Geo-Informatics Application: Feature Engineering for Ship Detection in SAR**
  - **The Challenge:** Detecting ships in SAR imagery. Ships typically appear as bright points due to strong "double-bounce" scattering between the ship's vertical side and the flat water surface.[45] However, other objects (e.g., buoys, small islands) can also appear as bright spots.[46]
  - **Feature Engineering Approach:**
    - **Initial Detection:** Identify all bright pixel clusters (potential targets).
    - **Geometric Features:** For each cluster, calculate geometric features like area, aspect ratio (length/width), and compactness. Ships are typically elongated.
    - **Proximity/Contextual Features:** Calculate the distance of each target to the coastline (ships are in the water), and distance to known shipping lanes or ports.
    - **Texture Features:** Analyze the texture of the area surrounding the target. The sea surface has a characteristic texture that differs from land.
  - **Real-world Connection:** By combining these engineered features (brightness, shape, location context, background texture), a classifier can more accurately distinguish true ship targets from false alarms, improving the reliability of maritime surveillance systems.[47]
- **Slide 94: Deeper Understanding: Feature Engineering as Hypothesis Testing**
  - The act of creating a feature is a way to embed domain knowledge into a model.
  - When an urban planner creates a distance_to_park feature, they are hypothesizing that green space access is a relevant factor.
  - When a remote sensing scientist creates a GLCM_homogeneity feature from a SAR image, they are hypothesizing that different land cover types have different levels of surface roughness that the radar can detect.
  - Feature engineering allows us to move beyond a "black box" approach and build models that are more interpretable and grounded in theory. The model's feature importance scores then tell us which of our hypotheses were most predictive.
- **Slide 95: Summary of Vector-Based Features**
  - geopandas provides powerful tools for creating features from vector data.

- **Geometric features** describe the shape and size of objects.
- **Proximity features** measure accessibility and influence.
- **Density/Count features** quantify the neighborhood context.
- **Key Pre-requisite:** Always work in a projected CRS for accurate distance and area calculations.

Slides 96-105: Raster-Based Features (Texture)

- **Slide 96: Introduction to Texture**
  - **Concept:** Texture refers to the spatial arrangement and variation of pixel intensity values in an image. It describes qualities like smoothness, coarseness, regularity, and directionality.[49]
  - **Why we are learning this:** A single pixel's value (e.g., its brightness) tells only part of the story. Its relationship to its neighbors—its texture—provides crucial contextual information that can dramatically improve classification.
  - **Visualization:** Side-by-side images of different textures with the same average brightness: a smooth surface (e.g., sand) and a rough surface (e.g., gravel). A simple classifier based on average brightness would fail, but one that uses texture can easily distinguish them.
- **Slide 97: Gray-Level Co-occurrence Matrix (GLCM)**
  - **Concept:** The GLCM is a statistical method for examining texture. It is a matrix that tabulates how often different combinations of pixel brightness values (gray levels) occur in an image at a specific distance and orientation.[36]
  - **How it works:**
    1. Define an **offset** (a distance d and an angle θ). For example, "1 pixel to the right" (d=1, θ=0°).
    2. Scan the image pixel by pixel.
    3. For each pixel, look at the pixel at the specified offset.
    4. The pair of values (value at pixel i, value at pixel j) corresponds to a cell in the GLCM. Increment the count in that cell.
  - **Visualization:** A small 4x4 pixel grid with integer values. A step-by-step calculation of the GLCM for a `` offset (1 pixel to the right).[36]
- **Slide 98: Interpreting the GLCM**
  - The GLCM itself reveals properties of the texture.
  - **Smooth/Homogeneous Texture:** Values will be concentrated along the diagonal of the GLCM, because adjacent pixels have similar values.
  - **Rough/Coarse Texture:** Values will be more spread out across the GLCM, because adjacent pixels are more likely to have different values.
  - **Visualization:** Two GLCMs displayed as heatmaps. One for a smooth texture, with a bright diagonal line. One for a rough texture, with values more dispersed.
- **Slide 99: Deriving Texture Features from the GLCM**
  - **Concept:** The GLCM is too large to use as a feature directly. Instead, we compute statistical measures from it. These become our texture features.[36]
  - **Common GLCM Features:**
    - **Contrast:** Measures local intensity variation. High for coarse textures.
    - **Dissimilarity:** Similar to contrast.
    - **Homogeneity (Inverse Difference Moment):** Measures closeness of GLCM elements to the diagonal. High for smooth textures.

    - **Energy (Angular Second Moment - ASM):** Measures uniformity. High when pixel values are constant.
    - **Correlation:** Measures the linear dependency of gray levels of neighboring pixels.
  - [36]
- **Slide 100: Implementing GLCM in Python**
  - **Tool:** scikit-image library, specifically the skimage.feature module.
  - **Functions:**
    - graycomatrix(): Computes the GLCM. Takes the image, a list of distances, and a list of angles as input.
    - graycoprops(): Calculates the texture properties from the computed GLCM.
  - [52]
- **Slide 101: Code Example: Calculating GLCM Features**
  - **Scenario:** Calculate the 'contrast' and 'homogeneity' for a single-band raster image.
  - **Code:**

```
Python
from skimage.feature import graycomatrix, graycoprops
import numpy as np

# Assume 'image' is a 2D numpy array (e.g., one band of a SAR image)
# Image must be integer type (e.g., uint8)
image = (image / image.max() * 255).astype(np.uint8)

# Calculate GLCM with a 1-pixel offset in 4 directions (0, 45, 90, 135 deg)
glcm = graycomatrix(image, distances=,
                    angles=[0, np.pi/4, np.pi/2, 3*np.pi/4],
                    levels=256, symmetric=True, normed=True)

# Calculate texture features
contrast = graycoprops(glcm, 'contrast')
homogeneity = graycoprops(glcm, 'homogeneity')

# To get a single rotation-invariant feature, average across all angles
avg_contrast = np.mean(contrast)
print(f"Average Contrast: {avg_contrast}")
```

  - **Explanation:** We calculate the GLCM for multiple directions and then average the resulting feature values to get a more robust, rotation-invariant measure of texture.
- **Slide 102: Geo-Informatics Application: Texture in SAR Imagery**
  - **Geo-Informatics Application:** SAR backscatter is highly sensitive to surface roughness and geometry, making texture a primary feature for land cover classification.[53]
  - **Examples:**
    - **Smooth Water:** Low backscatter, very smooth texture (low contrast, high homogeneity).
    - **Urban Areas:** High backscatter due to double-bounce from buildings, very rough and heterogeneous texture (high contrast).
    - **Forest Canopy:** Medium-high backscatter from volume scattering, moderately rough texture.
    - **Agricultural Fields:** Can be smooth or rough depending on crop type and tillage.
  - **Real-world Connection:** By calculating GLCM features on a SAR image, we can create new data layers that explicitly represent "roughness." A random forest classifier can then use these texture features, along with the original backscatter intensity, to produce a much more accurate land cover map than using intensity alone.[54]

- **Slide 103: Geo-Informatics Application: Spectral-Spatial Features in Hyperspectral Imagery**
  - **The Challenge:** Classifying hyperspectral data pixel-by-pixel (using only the spectral vector) is prone to "salt-and-pepper" noise because it ignores spatial context. The goal is to combine the rich spectral information with spatial features.[42]
  - **Feature Engineering Methods:**
    - **Simple Approach:** Calculate GLCM texture on one or more principal components of the hyperspectral cube and stack these texture bands with the original spectral bands.
    - **Advanced Approach: 3D Descriptors:** Treat the hyperspectral cube (x, y, λ) as a 3D volume. Methods like 3D Local Binary Patterns or 3D Gabor filters can extract features that capture patterns across both space and spectrum simultaneously.[42]
    - **Deep Learning:** 3D Convolutional Neural Networks (CNNs) automatically learn the optimal spectral-spatial features from the data cube, representing the state-of-the-art.
  - **Visualization:** A diagram showing a 3x3xN patch being extracted from a hyperspectral cube and fed into a 3D CNN.
- **Slide 104: Deeper Understanding: The Scale of Texture**
  - The choice of parameters for GLCM calculation—the **window size** over which it's computed and the **offset distance**—is critical.
  - A small window and small offset will capture micro-textures.
  - A large window and large offset will capture macro-textures.
  - For example, in a forest, a small window might capture the texture of individual tree crowns, while a large window might capture the texture of the entire forest stand.
  - This is another example of multi-scale analysis. Generating texture features at multiple scales and feeding them all to a model allows it to discover the most discriminative texture scale for each class.
- **Slide 105: Summary of Raster-Based Features**
  - Texture provides vital spatial context that complements single-pixel spectral information.
  - The Gray-Level Co-occurrence Matrix (GLCM) is a standard method for quantifying texture.
  - Features like Contrast, Homogeneity, and Energy are derived from the GLCM and used in ML models.
  - scikit-image provides the tools to calculate GLCMs and their properties.
  - Texture is a particularly powerful feature for analyzing SAR and hyperspectral imagery.

Module 2.2: Spatial Autocorrelation and Spatial Lag

Slides 106-110: Spatial Autocorrelation

- **Slide 106: Tobler's First Law of Geography**
  - **Quote:** "Everything is related to everything else, but near things are more related than distant things." - Waldo Tobler
  - **Concept:** This is the fundamental principle of spatial analysis. It suggests that values of properties at nearby locations are not independent.
  - **Examples:**
    - House prices in a neighborhood tend to be similar.

        - Temperatures at nearby weather stations are highly correlated.
        - Land cover is generally continuous; a forest pixel is likely to be surrounded by other forest pixels.
    - **Spatial Autocorrelation:** The statistical measurement of this principle. It quantifies the degree of dependency among observations in a geographic space.
- **Slide 107: Types of Spatial Autocorrelation**
    - **Positive Spatial Autocorrelation:** Similar values are clustered together. High values are near other high values; low values are near other low values. This is the most common pattern in geographic data. (e.g., clusters of high-income neighborhoods).
    - **Negative Spatial Autocorrelation:** Dissimilar values are clustered together. High values are near low values. This creates a checkerboard-like pattern and often indicates competition. (e.g., competing store locations).
    - **Zero Spatial Autocorrelation:** No spatial pattern; values are randomly distributed across the landscape. This is the null hypothesis for statistical tests.
    - **Visualization:** Three small maps illustrating each pattern: a clustered map (hotspots/coldspots), a checkerboard map, and a random "salt-and-pepper" map.
- **Slide 108: Measuring Global Spatial Autocorrelation: Moran's I**
    - **Concept:** Moran's I is the most common statistic for measuring global spatial autocorrelation. It evaluates whether the overall pattern is clustered, dispersed, or random.[39]
    - **Interpretation:**
        - $I > 0$: Positive spatial autocorrelation (clustering).
        - $I < 0$: Negative spatial autocorrelation (dispersion).
        - $I \approx 0$: No spatial autocorrelation (randomness).
    - **Statistical Significance:** The Moran's I value is evaluated against a null hypothesis of complete spatial randomness. A p-value and z-score are calculated to determine if the observed pattern is statistically significant.[39]
    - **Formula:** $I = \frac{N}{\sum_i \sum_j w_{ij}} \frac{\sum_i \sum_j w_{ij} (x_i - \bar{x})(x_j - \bar{x})}{\sum_i (x_i - \bar{x})^2}$
        - N is the number of features.
        - x_i is the value for feature i.
        - $\bar{x}$ is the mean of x.
        - w_{ij} is the spatial weight between feature i and j.
- **Slide 109: Moran Scatterplot**
    - **Concept:** A powerful visualization that plots a variable (z) against its spatial lag (Wz). Each point on the plot represents one feature.
    - **Quadrants:**
        - **Upper-Right (High-High):** High value surrounded by high values (hotspot).
        - **Lower-Left (Low-Low):** Low value surrounded by low values (coldspot).
        - **Upper-Left (Low-High):** Low value surrounded by high values (spatial outlier).
        - **Lower-Right (High-Low):** High value surrounded by low values (spatial outlier).
    - **Interpretation:** The slope of the regression line through the scatterplot is equivalent to the global Moran's I statistic. A strong positive slope indicates strong positive spatial autocorrelation.
    - **Visualization:** A classic Moran scatterplot with the four quadrants labeled.
- **Slide 110: Other Statistics: Geary's C**
    - **Concept:** Another measure of global spatial autocorrelation. It uses the sum of squared differences between neighboring values rather than the covariance-like cross-product of Moran's I.[40]
    - **Interpretation (inversely related to Moran's I):**
        - $C < 1$: Positive spatial autocorrelation (clustering).

    - $C > 1$: Negative spatial autocorrelation (dispersion).
    - $C \approx 1$: No spatial autocorrelation (randomness).
  - **Difference from Moran's I:** Geary's C is more sensitive to local differences, whereas Moran's I is more of a global measure of covariance.[40]

Slides 111-120: The Spatial Weights Matrix (W)

- **Slide 111: Defining "Neighborhood": The Spatial Weights Matrix (W)**
  - **Concept:** Before we can measure spatial autocorrelation or create a spatial lag, we must formally define what "neighbor" means. The spatial weights matrix, W, is the structure that does this.[37]
  - **Structure:** For a dataset with N features, W is an N x N matrix where the element w_{ij} represents the spatial connection between feature i and feature j.
  - **Binary vs. Weighted:**
    - **Binary:** $w_{ij} = 1$ if i and j are neighbors, 0 otherwise.
    - **Weighted:** w_{ij} can be a continuous value, for example, the inverse of the distance between i and j.
  - **Tool:** The libpysal.weights module from the PySAL ecosystem is the standard tool for creating spatial weights in Python.[56]
- **Slide 112: Contiguity-Based Weights (for Polygons)**
  - **Concept:** Neighbors are defined as features that touch each other.
  - **Types:**
    - **Rook Contiguity:** Polygons are neighbors if they share a common edge.
    - **Queen Contiguity:** Polygons are neighbors if they share a common edge or a common vertex. (Queen is more common as it is more inclusive).
  - **Visualization:** A diagram of a 3x3 grid of squares. The central square's Rook neighbors (4 squares) are highlighted. A second diagram shows the central square's Queen neighbors (8 squares) highlighted.
- **Slide 113: Code Example: Creating Queen Weights**
  - **Tool:** pysal.
  - **Code:**
    Python
    ```
    import geopandas as gpd
    import libpysal

    # Load a GeoDataFrame of US states
    states = gpd.read_file(...)

    # Create Queen contiguity weights
    # The 'idVariable' ensures the weights are indexed by state name
    w_queen = libpysal.weights.Queen.from_dataframe(states, idVariable='STATE_NAME')

    # Inspect the neighbors of 'Texas'
    print("Neighbors of Texas:", w_queen.neighbors)
    ```
  - **Output Snippet:** Neighbors of Texas: ['Oklahoma', 'Arkansas', 'Louisiana', 'New Mexico']
- **Slide 114: Distance-Based Weights**

- **Concept:** Neighbors are defined based on the distance between their centroids or points.
- **Types:**
  - **K-Nearest Neighbors (KNN):** An object's neighbors are its k closest objects. This results in an adaptive neighborhood size and ensures every object has exactly k neighbors.
  - **Distance Band (or Radius):** Objects are neighbors if the distance between them is less than a specified threshold d. This uses a fixed neighborhood size, which can result in some objects having many neighbors (in dense areas) and others having none (in sparse areas).
- **Visualization:** Two diagrams. One shows KNN with k=3, where each point is connected to its 3 closest neighbors. The other shows a distance band, where points are connected to all other points within a fixed radius circle.

- **Slide 115: Code Example: Creating KNN Weights**
  - **Tool:** pysal.
  - **Code:**

```
Python
import geopandas as gpd
import libpysal

# Load a GeoDataFrame of city points
cities = gpd.read_file(...)

# Create K-Nearest Neighbor weights with k=5
w_knn = libpysal.weights.KNN.from_dataframe(cities, k=5)

# Inspect the neighbors of the first city
print("Neighbors of city 0:", w_knn.neighbors)
```

  - **Explanation:** w_knn now stores, for each city, the indices of its 5 nearest neighbors.
- **Slide 116: Row Standardization**
  - **Concept:** A common and important transformation applied to a weights matrix. Each weight w_{ij} is divided by the sum of all weights for that row i.
  - **Formula:** $w_{ij}^{std} = \frac{w_{ij}}{\sum_j w_{ij}}$
  - **Result:** Every row in the standardized matrix sums to 1.
  - **Why we do this:** When we later calculate the spatial lag, using a row-standardized matrix means the lag will be the **weighted average** of the neighboring values, which is an intuitive and interpretable measure.[37]
  - **Implementation:** In pysal, this is done by setting the .transform attribute: w.transform = 'r'.
- **Slide 117: Deeper Understanding: W as a Hyperparameter**
  - The choice of W (Queen vs. KNN, k=5 vs. k=10) is one of the most critical modeling decisions in spatial analysis. It is a "hyperparameter" that is not learned from the data but is set by the analyst based on theory.
  - This choice defines the **scale of analysis**. A KNN model with k=3 looks at very local patterns. A distance band of 50km looks at regional patterns.
  - The choice should be justified based on the underlying process being studied. For modeling disease spread by contact, a contiguity matrix makes sense. For modeling a city's economic influence, a distance-based matrix might be better.
- **Slide 118: Geo-Informatics Application: Autocorrelation in SAR**
  - **Application:** As discussed before, land cover changes from events like landslides can be detected by measuring the change in spatial autocorrelation in multi-temporal SAR images.[57]
  - **How it works with W:** To calculate Moran's I on the raster, a weights matrix is constructed for the pixels. A

Queen contiguity W is typically used, where each pixel's neighbors are the 8 pixels surrounding it.

- **Process:**
  1. Create a Queen W for the raster grid.
  2. Calculate the Log-Ratio change image.
  3. Compute Moran's I on the Log-Ratio image using the W.
  4. A significant increase in Moran's I indicates a new cluster has formed, signaling an event like a landslide.

- **Slide 119: Code Example: Moran's I with esda**
  - **Tool:** The esda (Exploratory Spatial Data Analysis) package from the pysal family.
  - **Code:**

```
Python
import geopandas as gpd
import libpysal
import esda
import matplotlib.pyplot as plt
from splot.esda import moran_scatterplot

# Assume 'gdf' is a geodataframe with a column 'crime_rate'
# Assume 'w' is a pre-calculated spatial weights object (e.g., Queen)

# Calculate Moran's I
moran = esda.Moran(gdf['crime_rate'], w)

print(f"Moran's I: {moran.I:.4f}")
print(f"P-value: {moran.p_sim:.4f}")

# Create a Moran Scatterplot
fig, ax = moran_scatterplot(moran, aspect_equal=True)
plt.show()
```

  - [58]

- **Slide 120: Summary of Spatial Autocorrelation & Weights**
  - Spatial autocorrelation measures the degree of clustering or dispersion in spatial data.
  - Moran's I is the most common statistic for this.
  - A Spatial Weights Matrix (W) is required to define "neighborhoods" and is a critical modeling choice.
  - pysal is the primary Python library for creating W and calculating spatial autocorrelation statistics.

Slides 121-130: The Spatial Lag Feature

- **Slide 121: The Spatial Lag: Your Neighborhood's Average**
  - **Concept:** The spatial lag is one of the most powerful and widely used features in spatial machine learning. For a given variable, it calculates the weighted average of that variable in the immediate neighborhood of each location.[41]
  - **Formula:** $Lag(y)_i = \sum_j w_{ij} y_j$ (where W is row-standardized).
  - **Interpretation:** It captures the local context. If y is house price, the spatial lag is the average price of neighboring houses. If y is vegetation index, the spatial lag is the average "greenness" of the surrounding

pixels.

- **Slide 122: Why the Spatial Lag is so Powerful**
  - **Why we are learning this:** The spatial lag is the bridge between the world of spatial statistics and mainstream machine learning.[61]
  - It distills the complex concept of "neighborhood influence" into a single, powerful number.
  - By adding the spatial lag as a feature, you are explicitly providing a non-spatial ML model with information about each observation's spatial context.
  - In many cases, the value of a variable at a location is best predicted by the values of that same variable at nearby locations (e.g., today's temperature is best predicted by today's temperature at a nearby station).
- **Slide 123: Code Example: Calculating the Spatial Lag**
  - **Tool:** pysal.
  - **Code:**

```
Python
import geopandas as gpd
import libpysal

# Assume 'gdf' is a geodataframe with a column 'house_price'
# Assume 'w' is a row-standardized spatial weights object

# Calculate the spatial lag of house prices
gdf['lag_price'] = libpysal.weights.lag_spatial(w, gdf['house_price'])

# Display the original price and the calculated neighborhood average
print(gdf[['house_price', 'lag_price']].head())
```

  - **Explanation:** The lag_spatial function takes the weights object and the data series and efficiently computes the spatial lag for every feature. The new lag_price column can now be used as an independent variable in a regression model.
- **Slide 124: Application: Predicting House Prices**
  - **Scenario:** We want to predict the price of a house.
  - **Traditional Features:** num_bedrooms, square_footage, age_of_house.
  - **Spatial Feature:** lag_price (average price of neighboring houses).
  - **Model:** A Random Forest Regressor is trained with all four features.
  - **Real-world Connection:** The lag_price feature will almost certainly have very high feature importance. This tells the model that "a house is worth what its neighbors are worth," a fundamental concept in real estate appraisal that we have now encoded mathematically. This makes the model both more accurate and more interpretable.
- **Slide 125: Application: RADAR Precipitation Estimation**
  - **Geo-Informatics Application:** Quantitative Precipitation Estimation (QPE) from radar data aims to estimate rainfall intensity at the ground.
  - **Concept:** Rainfall is a highly spatially continuous phenomenon. The rainfall rate at one pixel is strongly correlated with the rates in surrounding pixels.
  - **Feature Engineering:** To predict rainfall at pixel (i, j), we can use the radar reflectivity at that pixel as a primary feature. We can then create a spatial lag feature: the average reflectivity of the 8 surrounding pixels.
  - **Real-world Connection:** This lag feature helps the model understand the spatial context of the storm. A high reflectivity value might be noise if its neighbors are all zero, but it's likely a real storm cell if its neighbors also have high reflectivity. This improves the robustness of the precipitation estimate.

- **Slide 126: Deeper Understanding: Spatial Lag vs. Spatial Autoregressive Models**
  - **Spatial Lag as a Feature:** This is what we have been discussing. We pre-calculate the spatial lag of the dependent variable (or an independent variable) and add it to our feature matrix. We then use a standard OLS or Random Forest model. This is a simple and powerful approach.
  - **Spatial Autoregressive (SAR) Model:** A specialized regression model where the spatial lag of the *dependent variable* is included as a predictor on the right-hand side of the equation: $y = \rho Wy + X\beta + \epsilon$. This model explicitly accounts for spatial dependence but requires specialized estimators (not OLS) to avoid bias.[61]
  - **Takeaway:** Using the spatial lag as a feature is a feature engineering approach to approximate the effect of a full spatial regression model. It's often easier to implement and integrate into standard ML pipelines.
- **Slides 127-128: Local Indicators of Spatial Association (LISA)**
  - **Concept:** While Global Moran's I gives one value for the entire dataset, Local Moran's I (also known as LISA) calculates a statistic for each individual feature. This allows us to identify the location of statistically significant hotspots, coldspots, and spatial outliers.[39]
  - **Output:** For each feature, LISA provides:
    - A local I-statistic.
    - A p-value for significance.
    - A quadrant classification (HH, LL, LH, HL).
  - **Application:** LISA is used to create "cluster maps" that show the locations of significant spatial patterns, moving beyond a single global statistic.
  - **Visualization:** A map of a city, with certain neighborhoods colored red (significant hotspots of crime) and others colored blue (significant coldspots).
- **Slide 129: Code Example: LISA with esda**
  - **Tool:** esda.
  - **Code:**

```
Python
import esda
from splot.esda import lisa_cluster

# Assume 'gdf' and 'w' are defined
# Calculate Local Moran's I
lisa = esda.Moran_Local(gdf['crime_rate'], w)

# Create a cluster map, showing significant features
# p=0.05 sets the significance level
lisa_cluster(lisa, gdf, p=0.05, figsize=(9,9))
plt.show()
```

  - **Explanation:** The lisa_cluster function from splot automatically generates a map that colors features based on their LISA quadrant, but only if their p-value is below the specified threshold. Non-significant features are colored gray.
- **Slide 130: Summary of Spatial Lag**
  - The spatial lag is a powerful feature that summarizes the neighborhood context of a variable.
  - It is the bridge that allows standard ML models to understand spatial dependency.
  - It is calculated using a spatial weights matrix (W) and the pysal library.
  - LISA statistics extend this concept to identify the locations of significant local clusters.

Module 2.3: Incorporating Spatial Context into Features

Slides 131-140: Advanced Spatial Context

- **Slide 131: Beyond the Spatial Lag**
  - **Concept:** The spatial lag (neighborhood average) is a powerful first-order summary of context. But the real world is more complex. Advanced methods aim to capture richer details about the neighborhood and spatial relationships.[63]
  - **Why we are learning this:** For complex problems, a single average value may not be enough. Capturing the heterogeneity, structure, and non-local connections within the data can provide a significant performance boost.
  - **Methods we will explore:**
    1. Neighborhood Summary Statistics
    2. Graph-Based Features
    3. Learned Contextual Features (Deep Learning)
- **Slide 132: Neighborhood Summary Statistics**
  - **Concept:** Instead of just calculating the mean of the neighbors (spatial lag), we can calculate a whole suite of descriptive statistics for the neighborhood defined by W.
  - **Example Features:**
    - neighborhood_std_dev: Standard deviation of the variable among neighbors. (Measures local heterogeneity. Is it a uniform or diverse neighborhood?)
    - neighborhood_min: Minimum value among neighbors.
    - neighborhood_max: Maximum value among neighbors.
    - neighborhood_median: Robust alternative to the mean.
  - **Application:** In house price prediction, the neighborhood_std_dev of prices could be a feature for "market volatility" or "neighborhood diversity," which might be a useful predictor in itself.
- **Slide 133: Graph-Based Features**
  - **Concept:** Think of your spatial data as a network or graph. The features are the nodes, and the spatial weights matrix W defines the edges between them. We can then use tools from graph theory to derive features.
  - **Example Features:**
    - **Node Degree:** The number of neighbors a feature has. In a contiguity graph, this measures how connected a polygon is.
    - **Centrality Measures (e.g., Betweenness Centrality):** Identifies features that are "bridges" or critical links in the spatial network.
  - **Real-world Connection:** This approach moves beyond simple proximity. Two locations might be far apart in Euclidean distance but highly connected through a transport network (e.g., two subway stops on the same line). A graph representation can capture this functional relationship, which Euclidean distance cannot.
- **Slide 134: Learned Context: The Deep Learning Approach**
  - **Concept:** The methods discussed so far involve "hand-crafting" features based on our hypotheses. The deep learning paradigm shifts from *feature engineering* to *representation learning*. We design a network architecture that can *learn* the optimal spatial context features automatically from the data.

- **Example:** A Convolutional Neural Network (CNN) applied to an image. The convolutional filters are not pre-defined; they are learned during training. Early layers might learn to detect simple spatial features like edges and corners. Deeper layers combine these to learn more complex features like textures, shapes, and eventually, objects. The network learns the most useful spatial representations for the given task.[43]

- **Slide 135: Geo-Informatics Deep Dive: Spatial Context in Hyperspectral Classification**
  - **The Challenge:** The "salt-and-pepper" effect in pixel-wise classification arises from ignoring spatial context. The label of a pixel is strongly dependent on its neighbors.[43]
  - **Solution: Combining Spectral and Spatial Information.**
    - **Method 1: Patch-Based CNNs:** Instead of feeding a single pixel's 200-band spectrum into a model, we extract a small spatial patch (e.g., 5x5 pixels) around the target pixel. This 5x5x200 "minicube" is the input to a 3D-CNN. The model learns both the spectral patterns and the local spatial patterns simultaneously.[55]
    - **Method 2: Graph Convolutional Networks (GCNs):** The image can be modeled as a graph where each pixel (or superpixel) is a node. GCNs then pass information between neighboring nodes, allowing the model to learn contextual relationships over longer ranges than a small CNN patch would allow.[43]
  - **Visualization:** A diagram comparing three approaches: (1) A single pixel vector going into a model. (2) A 5x5 pixel patch going into a model. (3) The whole image represented as a graph of connected nodes.
- **Slide 136: Geo-Informatics Deep Dive: Spatial Context in RADAR Precipitation Estimation**
  - **The Challenge:** Estimating rainfall at a single point from a single radar reflectivity value at that point is unreliable. The context of the surrounding weather system is crucial.[66]
  - **Solution: Using CNNs to Learn Spatiotemporal Patterns.**
    - **Input:** Instead of a single pixel value, the input to the model is a time-series of radar image "stamps" (e.g., a 64x64 pixel area) centered on the target location over the last hour.[67]
    - **Process:** A CNN learns the spatial patterns (shape of the storm, intensity gradients) within each stamp. A Recurrent Neural Network (RNN) or LSTM layer then processes the sequence of feature maps from the CNN to learn the temporal patterns (movement, intensification/decay of the storm).[19]
    - **Output:** A prediction of rainfall intensity at the central pixel for the next time step.
  - **Real-world Connection:** This is how modern "nowcasting" systems work. They learn the physics of storm motion and evolution directly from vast amounts of historical radar data, using deep learning to capture the complex spatiotemporal context.[67]
- **Slide 137: Deeper Understanding: Non-Euclidean and Non-Local Context**
  - Simple neighborhood definitions (distance, contiguity) assume space is flat (Euclidean) and that only immediate neighbors matter (local).
  - Real-world relationships are often more complex. Two neighborhoods might be geographically distant but functionally close because they are connected by a highway or subway line. Two research labs in different countries might be "close" in a collaboration network.
  - Graph-based representations (W matrices, GCNs) are powerful because they can model these arbitrary, non-Euclidean, and non-local relationships. This allows us to move beyond a simple grid-based view of space to a more flexible and powerful network-based view.
- **Slide 138: Case Study: A Scikit-learn Pipeline with a Custom Spatial Transformer**
  - **Concept:** We can integrate our custom spatial feature engineering steps into a standard scikit-learn pipeline for a clean, reproducible workflow. This involves creating a custom transformer class.[68]
  - **Process:**
    1. Define a custom transformer class that inherits from BaseEstimator and TransformerMixin.
    2. In its transform method, it will take a GeoDataFrame, calculate a spatial feature (e.g., distance to nearest POI), and return the modified DataFrame.
    3. Combine this custom transformer with standard scikit-learn steps (like StandardScaler and a final model)

in a Pipeline object.

- 69

- **Slide 139: Code Example: Custom Transformer for Proximity Feature**

```python
from sklearn.base import BaseEstimator, TransformerMixin
import geopandas as gpd

class ProximityTransformer(BaseEstimator, TransformerMixin):
    def __init__(self, poi_gdf, col_name):
        self.poi_gdf = poi_gdf
        self.col_name = col_name

    def fit(self, X, y=None):
        return self # No fitting needed

    def transform(self, X, y=None):
        # Assumes X is a GeoDataFrame
        joined = gpd.sjoin_nearest(X, self.poi_gdf, distance_col=self.col_name)
        # Handle cases where multiple points have the same nearest POI
        joined = joined[~joined.index.duplicated(keep='first')]
        return joined[[self.col_name]] # Return just the new feature column
```

  - **Explanation:** This class encapsulates the logic for calculating the distance to the nearest POI. It can now be used as a step in a pipeline.
- **Slide 140: Summary of Incorporating Spatial Context**
  - Spatial context provides information beyond what a single observation contains.
  - The spatial lag is a foundational context feature.
  - Richer context can be captured with neighborhood statistics or graph-based features.
  - Deep learning methods, especially CNNs and GCNs, excel at learning complex spectral-spatial and spatiotemporal context automatically from the data.
  - Custom transformers allow for the integration of spatial feature engineering into scikit-learn pipelines.

Slides 141-160: Summary, Challenges, and Self-Assessment

- **Slide 141: Topic Summary: Feature Engineering**
  - We covered three core areas:
    1. **Extracting Spatial Features:** Deriving features from vector (geometry, proximity, density) and raster (texture) data.
    2. **Spatial Autocorrelation & Lag:** Quantifying spatial dependency and creating a powerful neighborhood-average feature.
    3. **Advanced Spatial Context:** Moving beyond simple averages to capture heterogeneity and learned representations.
  - The goal is to translate geography into a numerical format that powerful, general-purpose ML models can understand.
- **Slide 142: Challenge: Choosing the Right Features**

- **Problem:** It's possible to generate hundreds or even thousands of potential features (multi-scale buffers, different GLCM parameters, etc.). How do you choose the best ones?
- **Solutions:**
  - **Domain Knowledge:** Start with features that are theoretically relevant.
  - **Feature Importance:** Use model-based methods (e.g., Random Forest feature importance) to rank features by their predictive power.
  - **Dimensionality Reduction:** Use techniques like Principal Component Analysis (PCA) to combine correlated features into a smaller set of components.
  - **Regularization:** Use models (like Lasso regression) that automatically perform feature selection by shrinking the coefficients of unimportant features to zero.

- **Slide 143: Challenge: Computational Scale**
  - **Problem:** Many spatial feature engineering tasks are computationally expensive. Calculating a distance matrix for 1 million points is a trillion calculations. Building a dense weights matrix for 100,000 polygons is infeasible.
  - **Solutions:**
    - **Use Spatial Indices:** Ensure your data has a spatial index, which dramatically speeds up spatial queries like joins and buffer operations. geopandas does this automatically.
    - **Use Sparse Matrices:** For weights matrices, use a sparse representation that only stores the non-zero neighbor relationships. pysal does this by default.
    - **Approximate Nearest Neighbors:** For very large datasets, use algorithms that find approximate nearest neighbors instead of exact ones.
    - **Parallel Processing:** Use libraries like Dask to distribute computations across multiple CPU cores.
- **Slide 144: Challenge: Transferability of Features**
  - **Problem:** Features engineered for one geographic area may not work well in another. The relationship between "distance to city center" and house price is very different in a monocentric city (like Paris) versus a polycentric city (like Los Angeles).
  - **Best Practice:** Be cautious when applying a model trained in one region to another. The spatial context is different. It's often necessary to retrain the model with locally-derived features. This is a key aspect of **spatial heterogeneity**, which we will discuss in later chapters.
- **Slides 145-155: Self-Assessment Quiz**
  - **Slide 145: Question 1:** You want to add a feature to your house price model representing "walkability." Describe a feature you could engineer using the buffer-and-count method. (Answer: Create a 500m buffer around each house and count the number of cafes, restaurants, and shops inside).
  - **Slide 146: Question 2:** A SAR image of an agricultural area shows two fields with the same average backscatter intensity. One is a smooth, bare soil field, and the other is a mature cornfield. Which GLCM feature would be most useful to distinguish them? (Answer: Contrast or Dissimilarity. The cornfield would have a much higher contrast due to its rougher texture).
  - **Slide 147: Question 3:** What does a Global Moran's I value of -0.8 with a p-value of 0.001 indicate? (Answer: A statistically significant pattern of strong negative spatial autocorrelation, or dispersion).
  - **Slide 148: Question 4:** You have created a Queen contiguity weights matrix w and a GeoDataFrame gdf with a column 'population'. You run w.transform = 'r'. What will the code lag_spatial(w, gdf['population']) calculate? (Answer: For each polygon, it will calculate the average population of its touching neighbors).
  - **Slide 149: Question 5:** What is the primary advantage of using a patch-based 3D-CNN for hyperspectral classification over a pixel-wise classifier? (Answer: It incorporates local spatial context, which helps reduce "salt-and-pepper" noise and improves accuracy).
  - **Slide 150: Question 6:** Why must you use a projected CRS when calculating a distance-based feature? (Answer: Because distances calculated using degrees from a geographic CRS are not uniform and are incorrect).

    - **Slide 151: Question 7:** What is the difference between the spatial lag of a variable and its neighborhood standard deviation? What do they each measure? (Answer: The lag measures the central tendency (average) of the neighborhood, while the standard deviation measures the dispersion or heterogeneity of the neighborhood).
    - **Slide 152: Question 8:** What is the key difference between feature engineering and representation learning? (Answer: In feature engineering, the user manually designs features based on hypotheses. In representation learning (e.g., deep learning), the model learns the optimal features automatically).
    - **Slide 153: Question 9:** Name two types of distance-based spatial weights matrices. (Answer: K-Nearest Neighbors (KNN) and Distance Band/Radius).
    - **Slide 154: Question 10:** How does the Moran Scatterplot visually represent Global Moran's I? (Answer: The slope of the regression line fitted to the scatterplot points is equal to the Global Moran's I statistic).
    - **Slide 155: Quiz Answers**
- **Slides 156-160: Further Reading and Next Steps**
    - **Slide 156:** Recommended Python Libraries Review (Logos and short descriptions for PySAL, esda, splot, scikit-image).
    - **Slide 157:** Advanced Topics (Brief mention of spatial fixed effects, geographically weighted regression (GWR), and graph neural networks).
    - **Slide 158:** Further Reading (List of key resources, e.g., the Geographic Data Science with Python book).
    - **Slide 159:** Practical Assignment Overview (Brief description of the upcoming lab where students will engineer features for a predictive modeling task).
    - **Slide 160:** Next Chapter Preview: Geospatial Data Visualization (A teaser for the next topic, explaining that now that we have cleaned data and engineered features, we need to learn how to effectively visualize our data and results).

Topic 3: Geospatial Data Visualization

Topic Overview: Learning Goals and Conceptual Background

Conceptual Background:
Geospatial data visualization is far more than just "making pretty maps." It is a critical component of the entire analytical workflow, serving as the primary tool for Exploratory Spatial Data Analysis (ESDA), hypothesis generation, model diagnostics, and the effective communication of final results.74 A well-crafted map can reveal spatial patterns, clusters, and outliers that are completely invisible in a table of numbers. However, a poorly designed map can just as easily mislead or obscure the truth. This topic explores both the science and art of cartography in the context of data science. It covers the fundamental principles of effective visualization—clarity, accuracy, and purpose—and the practical tools needed to implement them, from creating static, publication-quality maps with matplotlib to building interactive web maps for data exploration with folium.
Learning Goals:
Upon completing this topic, students will be able to:

1. Understand and apply the core principles of effective cartographic design (e.g., simplicity, visual hierarchy,

purpose).

2. Choose the appropriate map type (e.g., choropleth, proportional symbol, heat map) for different types of spatial data and analytical questions.
3. Understand the importance of map projections and select an appropriate projection for a given task to avoid spatial distortion.
4. Apply different data classification schemes (e.g., quantiles, natural breaks) and perceptually uniform color palettes to create clear and accurate choropleth maps.
5. Create static, multi-layered maps in Python using geopandas and matplotlib.
6. Add essential cartographic elements to a map, including basemaps, scale bars, and north arrows.
7. Create interactive web maps using folium, complete with markers, custom popups, and layer controls.
8. Apply specialized visualization techniques for remote sensing data, such as true-color and false-color composites for hyperspectral and optical imagery.

Key Concepts Table: Principles and Techniques in Geovisualization

Category	Method/Technique	Description	Key Consideration
Core Principles	Purpose and Audience	Defining the goal of the map (exploration, explanation) and for whom it is intended (experts, public).[76]	The goal dictates all other design choices, from complexity to symbology.
	Simplicity and Clarity	Removing any visual element that does not contribute to understanding the data (e.g., excessive gridlines, clutter).[74]	A clean design with ample white space enhances comprehension.
	Visual Hierarchy	Using visual cues like color saturation, line thickness, and font size to guide the viewer's eye to the most important information.[77]	Darker, thicker, larger elements appear more important.
Cartographic Choices	Map Projection	The mathematical transformation used to represent the 3D	Use equal-area projections for thematic maps to

		Earth on a 2D surface. Every projection distorts shape, area, distance, or direction.[76]	avoid misrepresenting size.
	Data Classification	The method used to group continuous data into a finite number of classes for symbology (e.g., for a choropleth map).[76]	The choice of method (Quantiles, Natural Breaks, etc.) can drastically change the visual pattern.
	Color Schemes	The selection of colors to represent data. Can be sequential, diverging, or qualitative.[76]	Use perceptually uniform, colorblind-friendly palettes (e.g., 'viridis').
Map Types	Choropleth Map	A map where areas (polygons) are shaded or patterned in proportion to a statistical variable.[76]	Best for normalized data (rates, densities), not raw counts.
	Proportional Symbol Map	A map that uses symbols of varying sizes (e.g., circles) to represent a numeric variable at specific locations.[76]	Good for showing raw counts at point locations.
	Heat Map	A map that uses color intensity to represent the density of point events.[76]	Visualizes "hotspots" of activity.
Python Libraries	matplotlib / geopandas	The standard combination for creating high-quality, static, multi-layered maps for publication.[79]	Highly customizable but not interactive.
	folium	A library for creating interactive, web-	Excellent for exploratory data

		based Leaflet.js maps with markers, popups, and layers.[80]	analysis and web dashboards.

Detailed Explanations and Slide Content

Module 3.1: Principles of Geospatial Data Visualization

Slides 161-175: The "Art and Science" of Cartography

- **Slide 161: Title Slide: Geospatial Data Visualization**
  - Chapter 2: Pre-processing Spatial Data for Machine Learning
  - Topic 3.1: Principles of Effective Geovisualization
  - Your Name, Course Title, University
- **Slide 162: The "Why": From Data to Insight**
  - **Concept:** Visualization is not just an endpoint for presenting results; it is a powerful analytical tool in its own right. A map is often the fastest and most intuitive way to understand spatial data.[75]
  - **Why we are learning this:**
    - **Exploratory Spatial Data Analysis (ESDA):** Maps help us identify patterns, spot outliers, and formulate hypotheses before formal modeling.[82]
    - **Model Diagnostics:** Mapping the residuals (errors) of a spatial model is a critical step to check for remaining spatial autocorrelation.
    - **Communication:** A compelling map is one of the most effective ways to communicate complex spatial findings to both technical and non-technical audiences.[76]
  - **Visualization:** A cycle diagram: Data -> Visualize -> Identify Pattern -> Form Hypothesis -> Model -> Visualize Results -> Refine Hypothesis.
- **Slide 163: A Map is a Model of Reality**
  - It is crucial to understand that a map is not a perfect representation of the world; it is a simplified model.
  - Every cartographic choice you make—projection, classification, color, scale—is an act of abstraction and generalization.
  - A choropleth map, for example, assumes a variable is uniform within a polygon, which is rarely true in reality.
  - **Takeaway:** A critical map-reader (and map-maker) understands how these modeling choices influence the story the map tells.
- **Slide 164: Core Principle 1: Define Purpose and Audience**
  - **Concept:** The first and most important question is: **Why** are you making this map, and **for whom**? [77]
  - **Exploratory Map (for yourself/experts):**
    - Goal: Uncover patterns, check data quality, find outliers.

        - Characteristics: Can be complex, may not need extensive labeling or a perfect layout. Interactivity is highly valuable.
    - **Explanatory Map (for a report/public):**
        - Goal: Communicate a specific finding or story clearly and concisely.
        - Characteristics: Must be simple, clear, and focused. Requires a clear title, legend, and source information.
    - **Real-world Connection:** A map for an internal meeting of city planners can be dense with technical data. A map for a public town hall meeting about the same project must be simplified to convey a single, clear message.
- **Slide 165: Core Principle 2: Simplicity and Visual Hierarchy**
    - **Concept:** Good design is about removing the unnecessary. Every element on the map should serve a purpose in communicating the data's story.[74]
    - **Simplicity:**
        - Limit the number of thematic layers.
        - Avoid visual clutter like excessive gridlines, borders, or decorative elements.
        - Use clear, intuitive symbols.
    - **Visual Hierarchy:**
        - Guide the viewer's eye to the most important information first.
        - Use darker/more saturated colors, thicker lines, and larger fonts for the most important elements (the "figure").
        - Use lighter/more transparent colors and thinner lines for contextual elements (the "ground," e.g., basemaps, boundaries).[77]
    - **Visualization:** Two maps side-by-side. The first is cluttered with labels, a heavy grid, and multiple distracting layers. The second is clean, showing only the key data layer on a simple basemap with a clear title.
- **Slide 166: Key Choice: Map Projections**
    - **Concept:** A map projection is the mathematical method for transforming the Earth's spherical surface onto a flat map. Every projection introduces distortion in at least one of four properties: Shape, Area, Distance, or Direction.[76]
    - **Why it matters:** The choice of projection can dramatically alter the visual impression of size and spatial relationships. Using the wrong projection can lead to serious misinterpretations.[74]
    - **Visualization:** A world map in Web Mercator projection next to a world map in a Mollweide (equal-area) projection. Annotations highlight the massive distortion of Greenland in Mercator compared to its more accurate relative size in Mollweide.
- **Slide 167: Common Map Projections and When to Use Them**
    - **Conformal Projections (Preserve Shape):**
        - Examples: Mercator, Lambert Conformal Conic.
        - Use for: Navigation, weather maps where angles are important.
        - Warning: Severely distorts area, especially near the poles. **Avoid for global thematic maps.**
    - **Equal-Area Projections (Preserve Area):**
        - Examples: Albers Equal Area, Mollweide, Cylindrical Equal-Area.
        - Use for: **Thematic maps** like choropleths, where it's important to compare the sizes of different regions accurately.
    - **Equidistant Projections (Preserve Distance):**
        - Example: Azimuthal Equidistant.
        - Use for: Radio and seismic work, where distance from a central point is key.
    - [76]
- **Slide 168: Key Choice: Data Classification for Choropleth Maps**

    - **Concept:** When visualizing a continuous variable on a choropleth map, you must divide its range into a discrete number of classes (bins). The method you use to define the breaks between these classes drastically affects the map's appearance.[78]
    - **Common Methods:**
        - **Equal Interval:** Divides the data range into equal-sized intervals. Easy to understand but can be skewed by outliers.
        - **Quantiles:** Divides the data so there is an equal number of features in each class. Can be misleading if data is clustered around a few values.
        - **Natural Breaks (Jenks):** An optimization method that identifies natural groupings in the data by minimizing variance within classes and maximizing variance between classes. Often a good default choice.
        - **User-Defined:** Breaks are set manually based on meaningful thresholds (e.g., poverty line, freezing point).
    - [76]
- **Slide 169: Visualization: The Impact of Classification**
    - **Visualization:** Show the same dataset (e.g., US state population density) mapped three times using three different classification schemes: Equal Interval, Quantiles, and Natural Breaks. The visual patterns will look strikingly different, demonstrating how classification is a powerful (and potentially manipulative) cartographic tool.
- **Slide 170: Key Choice: Color Schemes**
    - **Concept:** Color is the most powerful tool for encoding information on a map. The choice of color palette should match the nature of the data.[74]
    - **Types of Palettes:**
        - **Sequential:** A gradient from light to dark in a single hue. Used for ordered data that progresses from low to high (e.g., population density).
        - **Diverging:** Two sequential palettes joined by a neutral midpoint. Used for data with a meaningful central value (e.g., values above/below an average, positive/negative change).
        - **Qualitative (Categorical):** Distinct, unrelated colors. Used for nominal data where each color represents a different category (e.g., land cover types).[76]
    - **Visualization:** Show examples of each palette type from Matplotlib (e.g., 'viridis' for sequential, 'coolwarm' for diverging, 'tab10' for qualitative).
- **Slide 171: Best Practices for Color**
    - **Use Perceptually Uniform Palettes:** Use palettes like 'viridis', 'plasma', 'cividis' where a change in data value corresponds to an equal perceived change in color. This prevents visual distortion.[74]
    - **Be Mindful of Colorblindness:** About 8% of men have some form of color vision deficiency. Avoid red-green palettes

Works cited

1. Dealing with Missing Data | Summer 2017 | ArcUser - Esri, accessed on October 21, 2025, https://www.esri.com/about/newsroom/arcuser/dealing-with-missing-data
2. The prevention and handling of the missing data - PMC, accessed on October 21, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC3668100/
3. How to Deal with Missing Data | Master's in Data Science, accessed on October 21, 2025, https://www.mastersindatascience.org/learning/how-to-deal-with-missing-data/
4. How Fill Missing Values works—ArcGIS Pro | Documentation, accessed on October 21, 2025, https://pro.arcgis.com/en/pro-app/latest/tool-reference/space-time-pattern-mining/learnmorefillmissingvalues.htm
5. Fill gaps in your data with areal interpolation | Documentation - Learn ArcGIS, accessed on October 21,

2025, https://learn.arcgis.com/en/projects/fill-gaps-in-your-data-with-areal-interpolation/
6. How do I handle the problem of missing data due to cloud cover in ocean colour images?, accessed on October 21, 2025, https://www.researchgate.net/post/How_do_I_handle_the_problem_of_missing_data_due_to_cloud_cover_in_ocean_colour_images
7. An approach to fill in missing data from satellite imagery using data ..., accessed on October 21, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC9138164/
8. Normalization vs. Standardization: Key Differences Explained - DataCamp, accessed on October 21, 2025, https://www.datacamp.com/tutorial/normalization-vs-standardization
9. Radiometric Correction and Calibration of SAR Images - ASPRS, accessed on October 21, 2025, https://www.asprs.org/wp-content/uploads/pers/1989journal/sep/1989_sep_1295-1301.pdf
10. RADIOMETRIC CALIBRATION OF SAR IMAGE DATA ABSTRACT: KEYWORDS: - GeoSpatial WareHouse, accessed on October 21, 2025, https://geospatialwarehouse.wordpress.com/wp-content/uploads/2017/04/212_xxix-part1.pdf
11. 11. Spatial Analysis (Interpolation) - QGIS resources, accessed on October 21, 2025, https://docs.qgis.org/latest/en/docs/gentle_gis_introduction/spatial_analysis_interpolation.html
12. Spatial Interpolation Methods, accessed on October 21, 2025, https://iri.columbia.edu/~rijaf/CDTUserGuide/html/interpolation_methods.html
13. Resample (Data Management)—ArcGIS Pro | Documentation, accessed on October 21, 2025, https://pro.arcgis.com/en/pro-app/latest/tool-reference/data-management/resample.htm
14. Raster Resampling for Discrete and Continuous Data - GIS Geography, accessed on October 21, 2025, https://gisgeography.com/raster-resampling/
15. Missing Data and Multiple Imputation | Columbia University Mailman School of Public Health, accessed on October 21, 2025, https://www.publichealth.columbia.edu/research/population-health-methods/missing-data-and-multiple-imputation
16. Missing and empty geometries — GeoPandas 1.1.1+0.ge9b58ce ..., accessed on October 21, 2025, https://geopandas.org/en/stable/docs/user_guide/missing_empty.html
17. sklearn.preprocessing — scikit-learn 1.7.2 documentation, accessed on October 21, 2025, https://scikit-learn.org/stable/api/sklearn.preprocessing.html
18. Cloud removal for hyperspectral remotely sensed images based on ..., accessed on October 21, 2025, https://www.researchgate.net/publication/324799757_Cloud_removal_for_hyperspectral_remotely_sensed_images_based_on_hyperspectral_information_fusion
19. Gap-Filling and Missing Information Recovery for Time Series of ..., accessed on October 21, 2025, https://www.mdpi.com/2072-4292/14/19/4692
20. Diagram of SAR azimuth missing data - ResearchGate, accessed on October 21, 2025, https://www.researchgate.net/figure/Diagram-of-SAR-azimuth-missing-data_fig1_335341777
21. A Missing Data Imputation Method for Waste Dump Landslide ..., accessed on October 21, 2025, https://www.mdpi.com/2072-4292/17/17/2962
22. Cell size and resampling in analysis - ArcMap Resources for ArcGIS Desktop, accessed on October 21, 2025, https://desktop.arcgis.com/en/arcmap/latest/extensions/spatial-analyst/performing-analysis/cell-size-and-resampling-in-analysis.htm
23. (PDF) Hyperspectral Remote Sensing Data Analysis and Future Challenges - ResearchGate, accessed on October 21, 2025, https://www.researchgate.net/publication/260622818_Hyperspectral_Remote_Sensing_Data_Analysis_and_Future_Challenges
24. Five preprocessing strategies for enhancing Hyperspectral Data ..., accessed on October 21, 2025, https://www.redpoint-ai.com/five-preprocessing-strategies-for-enhancing-hyperspectral-data-analysis
25. Handling speckle noise on SAR images - KappaZeta, accessed on October 21, 2025, https://kappazeta.ee/blog/handling-speckle-noise-on-sar-images/

26. Speckle Noise Reduction Technique for SAR Images Using Statistical Characteristics of Speckle Noise and Discrete Wavelet Transform - MDPI, accessed on October 21, 2025, https://www.mdpi.com/2072-4292/11/10/1184
27. SAR Image Despeckling Using a Convolutional Neural Network, accessed on October 21, 2025, https://engineering.jhu.edu/vpatel36/wp-content/uploads/2018/08/despeckle_SPL_v5.pdf
28. 7.3. Preprocessing data — scikit-learn 1.7.2 documentation, accessed on October 21, 2025, https://scikit-learn.org/stable/modules/preprocessing.html
29. Chapter 3 Intro to geospatial machine learning, Part 1 | Public Policy Analytics, accessed on October 21, 2025, https://urbanspatial.github.io/PublicPolicyAnalytics/intro-to-geospatial-machine-learning-part-1.html
30. Spatial Feature Engineering - Oracle Help Center, accessed on October 21, 2025, https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/cspai/spatial-feature-engineering.html
31. Feature Engineering | Python Data Science Handbook, accessed on October 21, 2025, https://jakevdp.github.io/PythonDataScienceHandbook/05.04-feature-engineering.html
32. GeoPandas Tutorial: An Introduction to Geospatial Analysis - DataCamp, accessed on October 21, 2025, https://www.datacamp.com/tutorial/geopandas-tutorial-geospatial-analysis
33. Spatial Feature Engineering - Geographic Data Science with Python, accessed on October 21, 2025, https://geographicdata.science/book/notebooks/12_feature_engineering.html
34. Proximity analysis – Knowledge and References - Taylor & Francis, accessed on October 21, 2025, https://taylorandfrancis.com/knowledge/Engineering_and_technology/Engineering_support_and_special_topics/Proximity_analysis/
35. Proximity Analysis - Kaggle, accessed on October 21, 2025, https://www.kaggle.com/code/alexisbcook/proximity-analysis
36. Texture Analysis Using Gray-Level Co-Occurrence Matrix - MATLAB & Simulink - MathWorks, accessed on October 21, 2025, https://www.mathworks.com/help/images/texture-analysis-using-the-gray-level-co-occurrence-matrix-glcm.html
37. Introduction · Geographic Data Science with PySAL and the pydata ..., accessed on October 21, 2025, http://darribas.org/gds_scipy16/
38. Feature engineering with geography - PySAL, accessed on October 21, 2025, https://pysal.org/scipy2019-intermediate-gds/stochastic/gds3-geography-as-feature.html
39. How Spatial Autocorrelation (Global Moran's I) works—ArcGIS Pro ..., accessed on October 21, 2025, https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/h-how-spatial-autocorrelation-moran-s-i-spatial-st.htm
40. Geary's C - Wikipedia, accessed on October 21, 2025, https://en.wikipedia.org/wiki/Geary%27s_C
41. Chapter 4 Intro to geospatial machine learning, Part 2 | Public Policy ..., accessed on October 21, 2025, https://urbanspatial.github.io/PublicPolicyAnalytics/intro-to-geospatial-machine-learning-part-2.html
42. Spectral-spatial Feature Extraction for Hyperspectral Image ..., accessed on October 21, 2025, https://openresearch-repository.anu.edu.au/items/1d1cac44-24ac-4483-8bf3-e59c04ceb599
43. Hyperspectral Image Classification With Context-Aware Dynamic Graph Convolutional Network - Shirui Pan, accessed on October 21, 2025, https://shiruipan.github.io/publication/tgrs-2020-wan/tgrs-2020-wan.pdf
44. Proximity Analysis in Python - MichaelMinn.net, accessed on October 21, 2025, https://michaelminn.net/tutorials/python-proximity
45. Detect ships with SAR imagery | Documentation - Learn ArcGIS, accessed on October 21, 2025, https://learn.arcgis.com/en/projects/detect-ships-with-sar-imagery/
46. SSE-Ship: A SAR Image Ship Detection Model with Expanded Detection Field of View and Enhanced Effective Feature Information - Scirp.org., accessed on October 21, 2025, https://www.scirp.org/journal/paperinformation?paperid=124602

47. Context-aware SAR image ship detection and recognition network - Frontiers, accessed on October 21, 2025, https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2024.1293992/full
48. Ship detection in SAR imagery: A comparison study - ResearchGate, accessed on October 21, 2025, https://www.researchgate.net/publication/319141509_Ship_detection_in_SAR_imagery_A_comparison_study
49. Texture analysis | Images as Data Class Notes - Fiveable, accessed on October 21, 2025, https://fiveable.me/images-as-data/unit-4/texture-analysis/study-guide/6rcT9p8U0KbNBwq9
50. Looking at the Statistical Texture Approach Applied to Weather Radar Rainfall Fields, accessed on October 21, 2025, https://www.scirp.org/journal/paperinformation?paperid=115239
51. Modeling of texture quantification and image classification for change prediction due to COVID lockdown using Skysat and Planetscope imagery - PubMed Central, accessed on October 21, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC8384559/
52. Scikit Image - GLCM Texture Features - Tutorials Point, accessed on October 21, 2025, https://www.tutorialspoint.com/scikit-image/scikit-image-glcm-texture-features.htm
53. Exploiting spatial correlation features for SAR image analysis - ResearchGate, accessed on October 21, 2025, https://www.researchgate.net/publication/3202367_Exploiting_spatial_correlation_features_for_SAR_image_analysis
54. Segmentation of polarimetric radar imagery using statistical texture - EGUsphere, accessed on October 21, 2025, https://egusphere.copernicus.org/preprints/2023/egusphere-2023-181/
55. Full article: Spatial-spectral feature classification of hyperspectral image using a pretrained deep convolutional neural network - Taylor & Francis Online, accessed on October 21, 2025, https://www.tandfonline.com/doi/full/10.1080/22797254.2021.1942225
56. Spatial Weights - Geographic Data Science with Python, accessed on October 21, 2025, https://geographicdata.science/book/notebooks/04_spatial_weights.html
57. Measures of Spatial Autocorrelation Changes in Multitemporal SAR ..., accessed on October 21, 2025, https://www.mdpi.com/2072-4292/9/6/554
58. esda.Moran — esda v2.8.1.dev2+gcb1f0d777 Manual - PySAL, accessed on October 21, 2025, https://pysal.org/esda/generated/esda.Moran.html
59. splot.esda.moran_scatterplot — splot v1.1.5.post1+13.geffb183 Manual, accessed on October 21, 2025, https://splot.readthedocs.io/en/latest/generated/splot.esda.moran_scatterplot.html
60. esda_morans_viz - PySAL, accessed on October 21, 2025, https://pysal.org/notebooks/viz/splot/esda_morans_viz.html
61. Spatial Lag Model - Oracle Help Center, accessed on October 21, 2025, https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/cspai/spatial-lag-model.html
62. Spatial Autoregression (Spatial Statistics)—ArcGIS Pro | Documentation, accessed on October 21, 2025, https://pro.arcgis.com/en/pro-app/latest/tool-reference/spatial-statistics/spatial-autoregression.htm
63. Incorporating simulated spatial context information improves the effectiveness of contrastive learning models - ResearchGate, accessed on October 21, 2025, https://www.researchgate.net/publication/379313550_Incorporating_simulated_spatial_context_information_improves_the_effectiveness_of_contrastive_learning_models
64. Integration of spatial data context into machine learning models, accessed on October 21, 2025, https://repositories.lib.utexas.edu/items/5fde455e-ad18-4498-a4fa-cc0b714e775e
65. Hyperspectral Image Classification Using Spectral–Spatial Double-Branch Attention Mechanism - MDPI, accessed on October 21, 2025, https://www.mdpi.com/2072-4292/16/1/193
66. Comparing and Optimizing Four Machine Learning Approaches to Radar-Based Quantitative Precipitation Estimation - MDPI, accessed on October 21, 2025, https://www.mdpi.com/2072-4292/16/24/4713
67. Improving Doppler Radar Precipitation Prediction with Citizen ..., accessed on October 21, 2025,

https://pmc.ncbi.nlm.nih.gov/articles/PMC12197204/
68. Custom Transformers in scikit-learn Pipelines - GeeksforGeeks, accessed on October 21, 2025, https://www.geeksforgeeks.org/data-science/custom-transformers-in-scikit-learn-pipelines/
69. Principles and Techniques of Data Science - 14 Sklearn and Feature Engineering, accessed on October 21, 2025, https://ds100.org/fa23-course-notes/feature_engineering/feature_engineering.html
70. Scikit-learn Pipeline with Feature Engineering | ANotes - Adam Novotny notes, accessed on October 21, 2025, https://adamnovotny.com/blog/custom-scikit-learn-pipeline.html
71. Using Pipelines and FeatureUnions in scikit-learn - Michelle Fullwood, accessed on October 21, 2025, https://michelleful.github.io/code-blog/2015/06/20/pipelines/
72. Pipeline — scikit-learn 1.7.2 documentation, accessed on October 21, 2025, https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html
73. 7.1. Pipelines and composite estimators — scikit-learn 1.7.2 documentation, accessed on October 21, 2025, https://scikit-learn.org/stable/modules/compose.html
74. 7 Geospatial Data Visualization – Geospatial Data Science with R, accessed on October 21, 2025, https://warin.ca/geospatial/07-geospatial_data_visualization.html
75. Geospatial Data Visualization - NamasteDev Blogs, accessed on October 21, 2025, https://namastedev.com/blog/geospatial-data-visualization/
76. Geospatial Data Visualization | Big Data Analytics and Visualization ..., accessed on October 21, 2025, https://fiveable.me/big-data-analytics-and-visualization/unit-11/geospatial-data-visualization/study-guide/RVk5AgX4nmgeuSGW
77. Geospatial Visualization of Environmental Data - EDM - ITRC, accessed on October 21, 2025, https://edm-1.itrcweb.org/geospatial-visualization-of-environmental-data/
78. Creating a choropleth map using GeoPandas and financial data | by ..., accessed on October 21, 2025, https://mleblog.medium.com/creating-a-choropleth-map-using-geopandas-and-financial-data-c9839a51c187
79. Mapping and plotting tools — GeoPandas 1.1.1+0.ge9b58ce.dirty documentation, accessed on October 21, 2025, https://geopandas.org/en/stable/docs/user_guide/mapping.html
80. Web Maps with Folium - Python-Fiddle, accessed on October 21, 2025, https://python-fiddle.com/tutorials/folium
81. Interactive leaflet maps in Python with folium, accessed on October 21, 2025, https://python-charts.com/spatial/interactive-maps-folium/
82. Chapter 62 Introduction to exploratory spatial data analysis and visualization using QGIS | EDAV Fall 2021 Tues/Thurs Community Contributions, accessed on October 21, 2025, https://jtr13.github.io/cc21fall2/introduction-to-exploratory-spatial-data-analysis-and-visualization-using-qgis.html