Spatial Data, GIS, and Remote Sensing

Note on data. This problem set uses (i) a free, georeferenced remote-sensing raster downloaded in the tutorial and (ii) **synthetic** vector labels (toy buildings and roads) created for teaching. The goal is to practice spatial data workflows (CRS, vector/raster integration) and a lightweight segmentation pipeline—not to make substantive claims about real-world land use.

Conceptual Questions

Please write three to ten sentence explanations for each of the following questions. **You are only required to answer ONE of the two questions below.**

1. Explain why coordinate reference systems (CRS) matter in social science GIS workflows. In your answer, describe one concrete error that can occur if layers are in mismatched CRSs (e.g., wrong distances/areas; failed spatial joins; misaligned rasterization), and explain how you would detect and fix the problem in practice.
2. Remote-sensing and GIS data often involve multiple file types and processing steps (raster windows, vector overlays, rasterization, model training). Describe a reproducible pipeline architecture for such a project. In your answer, mention (i) how you would cache intermediate artifacts, (ii) how you would document transforms/CRS choices, and (iii) one strategy to reduce measurement error when building labels from imperfect spatial data.

Applied Exercises

Use the code in the week's code tutorial and the lecture slides to answer the following questions.

3. **Raster workflow: download, metadata, and windowing.** Using the provided Python tutorial script:
   - Download and open the georeferenced raster. Report key metadata: CRS, bounds, width/height, band count, dtype, and affine transform.
   - Extract a 256×256 window (patch) using `rasterio.windows.Window`. Plot the full raster and the patch.
   - Report the patch shape and compute its approximate bounding box in the raster CRS.
   - In 3–6 sentences, explain why windowing is useful for both spatial analysis and machine learning workflows.
4. **Vector workflow: GeoPandas, CRS transforms, and spatial join.** Using the synthetic vector geometries in the script:
   - Create the buildings GeoDataFrame (polygons) and roads GeoDataFrame (buffered line polygons), both in the raster CRS.
   - Transform both layers to WGS84 (`EPSG:4326`) and print the head of each transformed GeoDataFrame.

- Generate random points in the patch bounds, then run a spatial join to determine which points fall within building polygons. Report the number (and share) of points that fall inside buildings.
- Create one plot showing the patch extent with buildings, roads, and points overlaid (a simple matplotlib plot is fine).

5. **Rasterization + segmentation + panoptic-style output.** Using the tutorial workflow:
   - Rasterize the roads and buildings to create:
     (a) a semantic mask with classes {0 background, 1 road, 2 building}, and
     (b) an instance mask for buildings (IDs 1..K; 0 for not-building).
   - Train the small encoder–decoder CNN on the patch to predict the semantic mask. Report the final training loss and pixel accuracy.
   - Post-process predicted building pixels using connected components to produce a "panoptic-style" output (stuff class + building instances). Report the number of predicted building instances.
   - Save (or show) a figure with three panels: raster patch, predicted semantic classes, and panoptic-style IDs.
   - In 6–10 sentences, discuss at least two limitations of this toy setup (e.g., synthetic labels, single-patch overfitting, no train/test split, simplistic instance extraction) and one improvement you would make to move toward a realistic remote-sensing pipeline.
6. **Challenge Question (Optional — if you finish early):** Add one evaluation or robustness check beyond pixel accuracy. Choose **ONE** of the following options:
   (a) Compute per-class Intersection-over-Union (IoU) for the semantic mask (background/road/building) and briefly interpret which class is hardest and why.
   (b) Run a small sensitivity analysis that varies one modelling knob (e.g., number of epochs, learning rate, window location) across **three** settings. Report how pixel accuracy and the number of predicted building instances change, and interpret the pattern (5–8 sentences).