Write up of Project 1:

Modular Coding:

My algorithm is modular, breaking down the task of finding the smallest border into clear, purpose-specific functions. This structure makes the code self-documenting, easy to understand and maintain thanks to docstrings and comments. This enables data to be validated before being returned to another function, enhancing error handling and robustness.

The Main Algorithm:
The find_shortest_border() function compares each country in the world as country A, with a country B to locate the shortest border. This is achieved with a nested loop. Finding the length of a border for each pairwise comparison requires GeoPandas' computationally intensive geometry intersection tests. Thus, reducing the number of comparisons and intersection tests is essential.

R-Trees:

Only geographically close countries can share borders, so the algorithm uses an R-tree – a balanced tree, spatial index data structure - to create a list of neighbouring countries. This sub-list replaces the need to check against the entire world database, allowing the algorithm to avoid, in most cases hundreds, of unnecessary comparisons with distant countries.

The R-tree is populated by iterating over the world GeoSeries, adding only bounding box data. Later, when full geometry data needs to be retrieved, the algorithm makes use of iloc[] to pull it from 'world'. This is faster than using a Generator function, potentially because these functions are better optimised for GeoPandas, or handle bounding data directly rather than yielding an entire record from world at a time.

The root node of the R-tree contains a large bounding box that encompasses the bounding boxes of all countries, while each level of child nodes below it contains progressively smaller bounding boxes that group spatially close countries. This structure allows highly efficient queries for intersections with country A's bounding box. If a bounding box at a child node does not contain country A, all its child nodes can be ignored, so only relevant branches of the tree are traversed. The time complexity of this traversal is O(log n), which is significantly faster than iterating through a list of all countries at O(n) complexity. Additionally, once created and populated, the R-tree can be reused for multiple queries without the memory overhead of generating a new tree each time. By storing only bounding boxes and index pointers to the 'world' dataset, the R-tree efficiently handles small amounts of data, avoiding the need to process full geometry information in contrast to geometry intersection checks. This approach reduces both computational load and memory usage.

However, bounding box intersections do not guarantee the geometry of these countries share a border, so intersection tests are still required to confirm and produce border geom's for measurement. Nonetheless, this proximity-based R-tree search substantially reduces the number of function calls later in the algorithm, runs on a more scalable time complexity, and offsets any memory trade-off introduced by initialising and storing the data structure and bounding box data.

Redundant comparisons:

Within the inner loop, further redundant comparisons are prevented by avoiding self-comparisons and reverse comparisons. Self-comparisons occur when a country's geometry is compared with itself, which is unnecessary. To avoid this, each loop iteration stops if idx_a (country A) matches idx_b (country B), thus preventing unnecessary calls and saving memory and time. Additionally, redundant reverse comparisons are avoided. For example, a comparison between Spain (idx_a) and Germany (idx_b) should not be repeated as Germany (idx_a) and Spain (idx_b). By ensuring idx_a is always less than idx_b, we eliminate reverse comparisons, skipping cases where idx_a is greater, as they have already been handled in the reverse order. This effectively halves the number of function calls. This can be better explained visually below.

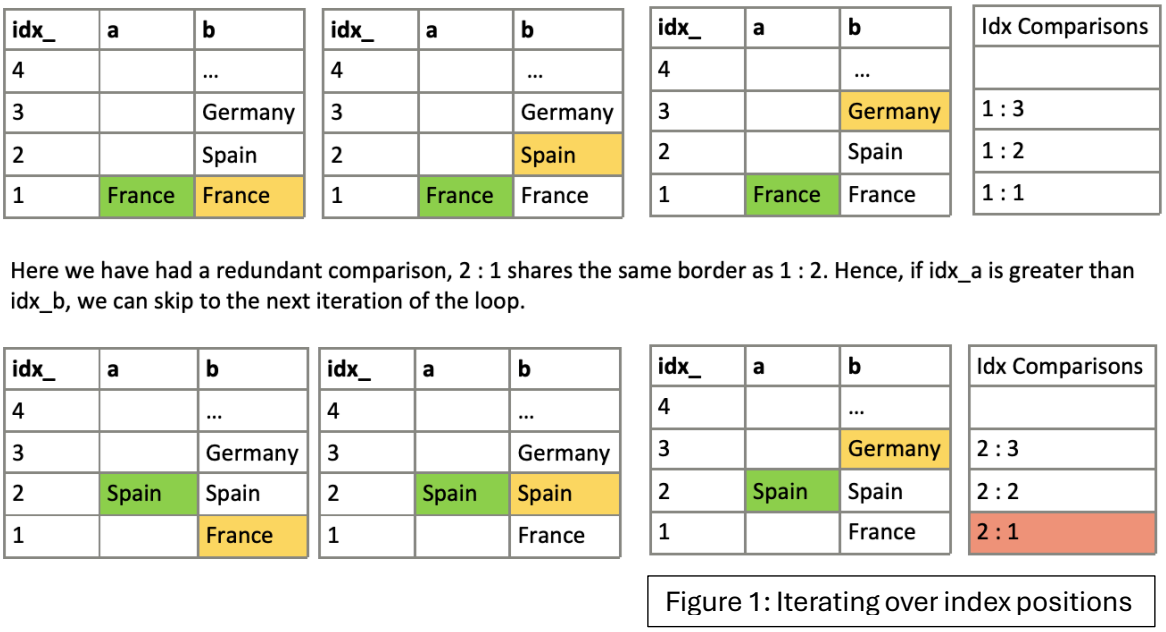

Figure 1: Iterating over index positions

Despite these optimizations, the algorithm's nested loop has O(n^2) complexity, meaning performance will degrade significantly as dataset size increases. While this quadratic complexity is okay for the small dataset in this project, it would become inefficient with larger datasets.

Calculating Border Length:

The calculate_length function uses geodesic distance calculations with the WGS84 global datum (EPSG:4326) to optimise between computational efficiency and spatial accuracy. Geodesic calculations compute the shortest path between two points on the Earth's ellipsoidal surface, accounting for curvature. Implemented with Pyproj's Geod class, this method achieves high global accuracy, typically within a few metres, without needing planar projections. Keeping the GeoDataFrame in WGS84 reduces computational overhead by avoiding reprojections for each border calculation, enhancing efficiency.

Geoidal calculations using a local Geod datums would be the most accurate way of measuring these distances; however, implimentation of using local Geoids when running calculations over the entire globe encounters similar issues with locality. A few datums could have been chosen for key areas, and hardcoded in, but this is not a particularly elegant solution. Employing the WGS84 global datum is still an accurate choice, pragmatic, and is efficient by avoiding re-projections or retrieving local datums for each area dynamically.

Local UTM projections, dynamically identified with GeoPandas, could have been used alongside planar distance measurements like Pythagoras theorum. However, this would require reprojecting the dataset for each border calculation, which is computationally inefficient. Furthermore, UTM zones are less effective for borders spanning multiple zones, potentially introducing distortions and reducing accuracy. Finally, given the difference between the calculated size of the border between Italy and the Vatican, and it's actual size referenced online, it seems that the dataset Is likely both relatively coarse and imprecise, making the accuracy gains of just a few meters achieved by local projections, seem trivial.

Drawing the Map:

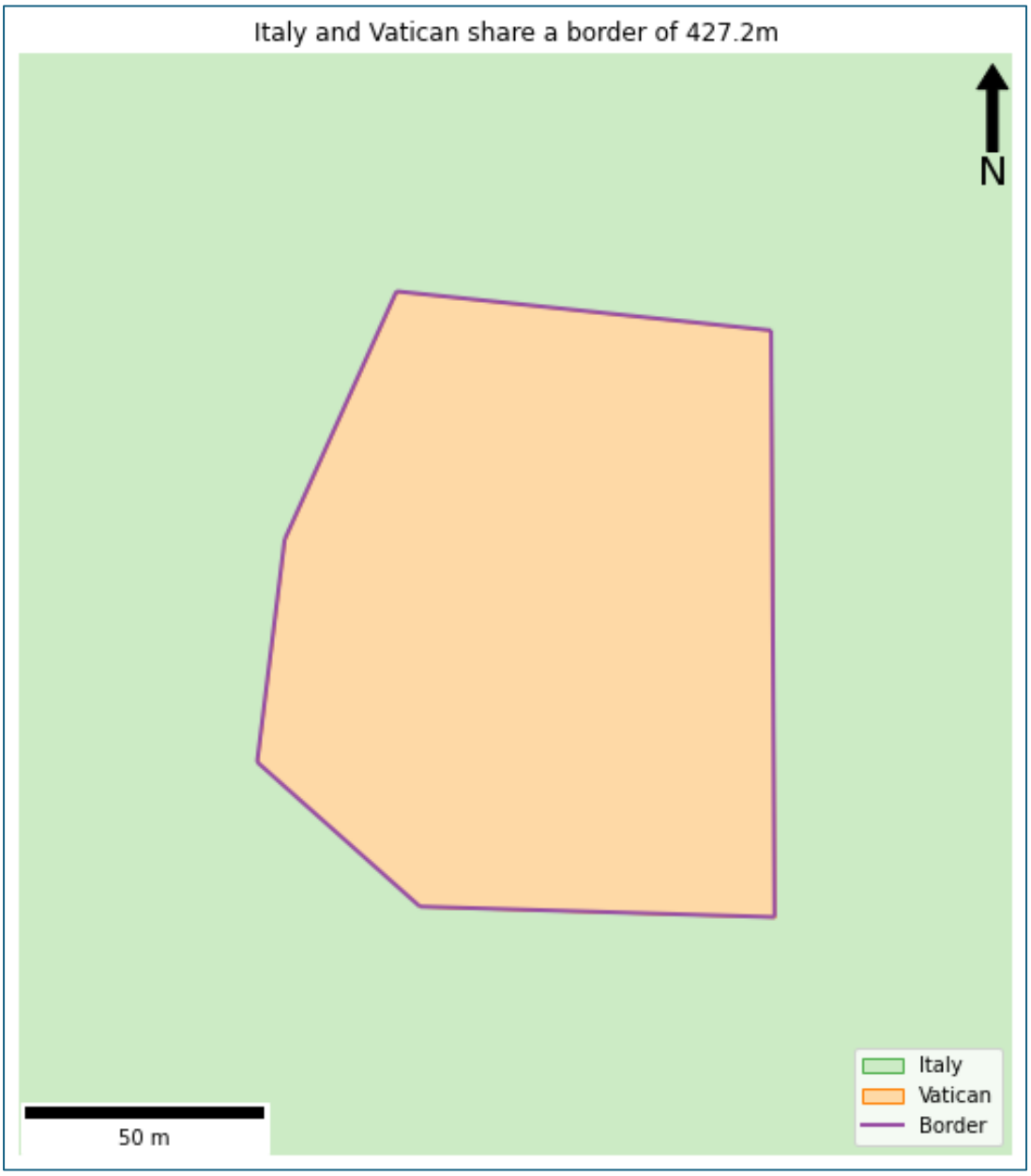

Figure 2: Map of the shortest border

The draw_map function visualizes the shortest bord[illegible]GS84 (geographic CRS), the data is reprojected to the Universal Transverse Mercator (UTM) projection, which is dynamically estimated as the best fit to the border region. This UTM projection is then

applied consistently to all elements plotted, including the countries surrounding the border, ensuring spatial accuracy across the map. UTM reduces scale and area distortions by dividing the Earth into 60 zones that preserve accurate distances and areas within each zone. Since this task focuses on the shortest border, UTM is suitable, as the border itself is unlikely to span multiple zones. While neighbouring regions, such as Italy, may extend across zones and experience minor distortions, UTM ensures accuracy for the immediate area of interest. Additionally, a buffer is applied, limiting the visual extent so that Italy's full shape isn’t visible, further minimizing distortion concerns.