The Loneliest Place on Earth

Sunny Simon David

A DISSERTATION

Submitted to

The University of Liverpool

in partial fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

Student Declaration

SIGNATURE	Sunny Simon David
DATE	October 4, 2024

Acknowledgments

I would like to express my sincere gratitude to all those who have supported me throughout the journey of completing this dissertation.
First and foremost, I extend my deepest appreciation to my supervisor, Prof. Sebastian Wild, whose guidance, expertise, and unwavering support have been invaluable throughout this research process. Their insightful feedback and encouragement have significantly shaped both this project and my academic growth.
To my family and friends, thank you for your patience, understanding, and encouragement throughout this process. Your unwavering belief in me has been a constant source of motivation.
I am also grateful to the Department of Computer Science at the University of Liverpool for providing the resources and academic environment conducive to research.

THE LONELIEST PLACE ON EARTH

Abstract

Finding the point on Earth furthest from human population or in a more amusing tone "The Lonliest Place on Earth" is a fascinating computational problem with applications in a number of fields such ecology, conservation, and urban planning. Given a chosen global population density dataset in GeoTIFF format (with a resolution of 30 arc-seconds or approximately 1km at the equator), I explored multiple approaches to identify the most isolated location with varying degree of results.

The initial solution employed an antipodal approach, calculating the weighted centroid of the global population using K-means and identifying its antipodal point as the most remote location. While computationally efficient, this method oversimplified population distribution. I then developed a Voronoi diagram-based solution, which divides the Earth's surface into regions based on proximity to populated areas. The vertices of these Voronoi polygons represent equidistant points from multiple population centers, making them candidates for remote locations. To handle the computational challenges of global-scale data, methods to down-sample the data were implemented. However, this approach still proved to be a challenge to get computationally efficient.

Additionally, a Binary Masking approach to calculate the furthest distance from any human population using a distance transform was used that provided the most reasonably accurate result giving us a location approximately in the Antarctica.

Each method presented trade-offs between accuracy, computational efficiency, and memory usage. The Binary Masking approach offered a good balance, providing reasonably accurate results while remaining computationally feasible. Key challenges included handling the large dataset, ensuring accurate land/water discrimination, and managing computational resources effectively.

Statement of Ethical Compliance

The guide to the Ethical Compliance was followed on this project.
Data Category: B
Participant Category: 0

Contents

List of Figures

Chapter 1

Introduction

There are some days when one just wants to go away from everyone and anyone. This predicament leaves us with an interesting question. What place is furthest away from any human population? Disregarding galaxies at the edge of the observable universe, we can narrow down our search to find the point furthest away from any human population on planet Earth itself. Or, “The Loneliest Place on Planet Earth.”

. My approach here involves using population density data of the human population to calculate a point that is furthest away from any human population.

1.1 Scope

There are many remote places on Earth to look at in search for an answer to the aforementioned question. There are numerous poles of inaccessibility[1] established on all continents. These are points on planet earth that are furthest away from any sort of boundary including political boundaries, land, oceans, etc. Included in these is the point Nemo[2]. Point Nemo is a fascinating spot in the Pacific Ocean, known for being the Oceanic pole of Inaccessibility. This means it is the farthest point from any land on Earth. Located at coordinates 48°52.6’S 123°23.6’W, Point Nemo is about 1,450 nautical miles (2,688 kilometres) from the nearest landmasses, which are Ducie Island, Motu Nui (part of the Easter Island chain), and Maher Island in Antarctica. The name ”Point Nemo” comes from the Latin word ”nemo,” meaning ”no one,” and was popularised by Jules Verne’s character Captain Nemo in the novel ”Twenty Thousand Leagues Under the Sea.” This name reflects the isolation of the location. Interestingly, Point Nemo is not only remote from human habitation but also lies beneath a part of the Pacific Ocean known as the South Pacific Gyre. This area is devoid of nutrient-rich waters, making it one of the least biologically active regions of the world’s oceans. It is often referred to as an ”oceanic desert.”

However, can this be verified to be furthest away from any human population? Or is there some other place on planet Earth that is the loneliest? This is the question that has been tackled in this project. In this study, regions both on land and water are being considered to find a truly remote place.

1.2 Problem Statement

This study aims to find a point on planet earth such that, it is the furthest away from any human population using population density data that is publicly available.

1.3 Approach

This study includes three main menthodologies that use different techniques in order to compute the answer. See section 2.1, 2.2, and 2.3 for more detail.

1.4 Outcome

The expected outcome of this study is to find a specefiv co-ordinate or a region on the planet Earth furthest away from any human population. The output of this would be in an interactive map format with the remote location visualised. Additionally, this has to be done using reasonable computationalal resources and efficiently.

Chapter 2

Background

The first most obvious way to start was Lukatela's Nemo Library API [3]. The Lukatela Nemo Library is a comprehensive collection of software tools designed to handle complex geodetic computations and spatial data transformations. It includes various functions to perform operations such as projecting points from spherical coordinates to planar coordinates and vice versa, generating random points on a sphere, and performing stack operations. However, for this particular task, the resolution of the data proved to be too high to use the functionalities provided by this API efficiently. However, there were a few other approaches that could be used. For this project a spatial clustering paradigm was picked to begin with.

Spatial clustering is a crucial technique applicable in numerous areas like urban planning, ecology, and data analysis. It allows for the detection of patterns, clustering of similar data points, and understanding spatial distributions more effectively using geospatial data indexing. The H3 library, created by Uber Technologies, is a robust open-source tool for geospatial indexing and analysis that was the initial preference for this project. Global spatial indexing systems employ a worldwide grid where each cell is uniquely indexed and can be zoomed in for different levels of precision. For example, the Open Location Code (OLC) [4] uses a grid based on square cells. In contrast, the Hexagonal Hierarchical Spatial Index (H3) utilises a combination of hexagons and pentagons to tile the Earth's ellipsoidal surface, providing efficient and flexible geospatial data indexing.

The paper titled "An Enhanced Earthquake Risk Analysis using H3 Spatial Indexing"[5] by A.N. Aini et al., presented at the Second International Seminar on Earth Sciences and Technology (ISEST-2023), introduces the application of the H3 spatial indexing system[17] to improve earthquake risk assessment, with Sorong City as a case study. The study compares H3-based risk assessments with traditional methods used in Indonesia, highlighting H3's advantages in providing detailed, non-boundary-dependent, and efficient risk evaluations. The performance evaluation indicates that H3 offers faster and more accurate spatial analysis compared to conventional methods, making it suitable for large-scale and high-resolution data assessments.

However, given the nature of the dataset [6] chosen, H3 proved to be very computationally expensive. Hence, some other approaches needed to be investigated. Based off the limited literature available on this problem, the following approaches seemed to be the most promising.

2.1 Voronoi Diagrams

[7] Given a set of n points randomly placed on a flat surface, a Voronoi diagram creates n distinct regions - one around each point. Each region

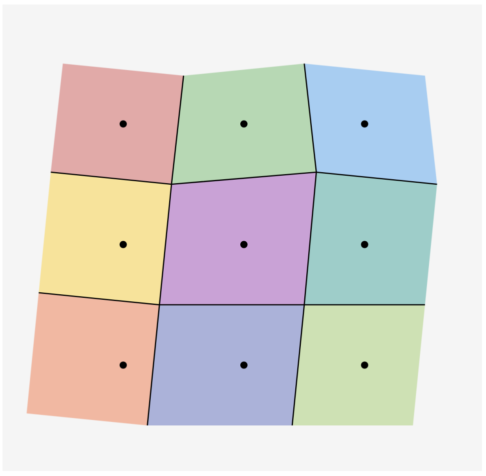


Figure 2.1: Example of a Voronoi Tessellation

contains all the space that's closer to its central point than to any other point in the set. These regions fit together perfectly like puzzle pieces to cover the entire surface, with no gaps or overlaps.

See figure 2.1 for an example of how this would look. In this figure, each point is surrounded by its own cell. The boundaries between cells are always equidistant from the two closest points - so if you were to pick any spot on a boundary line and measure the distance to the points on either side, those distances would be equal. This property of Voronoi cells ensures that all locations in a cell are closer to the central point of the cell than any other point on the diagram.
Mathematically this can be defined in the following way:

Given that *P=p1,p2,. . . ,pm* is a set of m points in a *n*-dimensional space, a set of partitions Vi can be created containing all points in Rn that are closer to pi than to any other point such that:

$$V_i = \{x : \forall j \neq i, d\left(x, p_i\right) \leq d\left(x, p_j\right)\}, with\ i, j \epsilon \{1, 2, ..., m\}$$

where, *d(x,y)* typically gives the Euclidean distance between the arguments.

Voronoi diagrams have a close relationship with the k-nearest neighbors (k-NN) algorithm, which is widely used in machine learning for tasks like classification, regression, and clustering. The k-NN algorithm works by looking at the k closest data points in your training set to make predictions about new data points. When k=1 (meaning you only look at the single nearest neighbor), the boundaries of Voronoi cells perfectly match the decision boundaries in the 1-NN algorithm.

2.2 K-Means

K-Means is a very popular clustering algorithm that groups similar data points (population densities in this case) together. Depending on the number of clusters created, there can be a number of remote locations that can be identified as the lonliest place on earth. However, in this project K-means was used to find a weighted average of the entire human population. This would be the centroid of the cluster that defined the human population the best.

K-means works by minimising the objective function which is the squared error function[8]. It can be mathematically defined as follows:

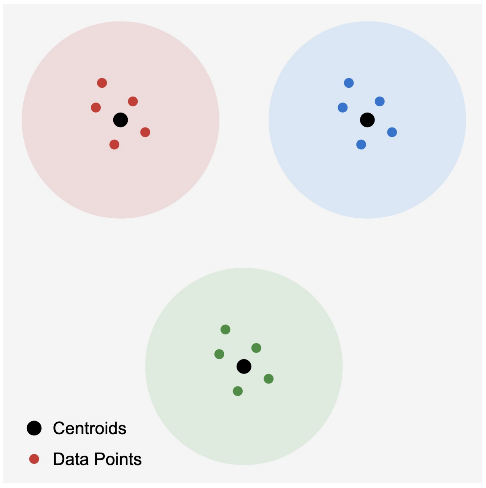


Figure 2.2: K-Means Illustration

$$J = \sum_{i=1}^{k} \sum_{j=1}^{n} (\|x_i - v_j\|)^2 = 1$$

where,

$$(\|x_i - v_j\|)$$

is the euclidean distance.

The figure 2.2 illustrates how a K-means algorithm clusters:
In the visualisation, the black dots represent the centroids and the coloured dots represent the data points. The data points are grouped based on the proximity to the nearest centroid.

Applied to the population density dataset we can find a centroid on planet earth such that it is the weighted average of the human population distribution. Logically, the furthest place away from such a point would be the "anti-podal" point or simply put, the point on the exact opposite side of the earth. This point would lie on the surface of the earth and on the line that would connect the centroid found by the k-means algorithm and the center of the earth.

2.3 Binary Masking

Another approach that proved to be much more efficient and accurate was using a binary mask [9] and then using the Haversine distance (see section 2.4) transform to calculate a point on planet earth furthest away from any human population. This works more efficiently due to the basic paradigm of how the binary mask works on the Region of Interest(ROI) in our population dataset. Since the furthest distance from any human population is being computed, any pixel in the data set with any population will be covered in the binary mask. Mask pixel values of 1 indicate image pixels that belong to the ROI. Alternatively, mask pixel values of 0 indicate image pixels that are part of the background or in the context of this project, regions that have no human population. This gives a clear mask that covers all of the areas that have a human population.

Computing a point furthest away from the mask becomes much more easier with this approach combined with the Haversine distance formula and seemed to provide the most accurate result as well.

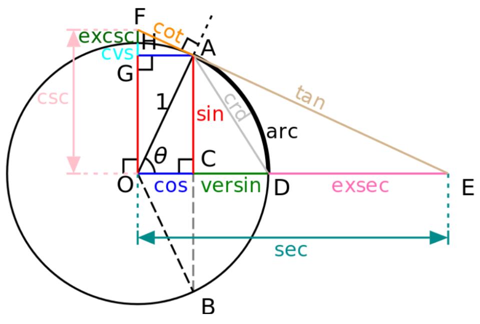


Figure 2.3: Circular Geometry

2.4 Haversine Distance

The Haversine formula[10] is a mathematical equation used to determine the great-circle distance between two points on a sphere given their latitudes and longitudes. This formula is particularly important in navigation, geospatial analysis, and geographic information systems (GIS).

the "Ha" in "Haversine" stands for "half versed sine" where $haversin\,(\theta) = versin\,(\theta/2)$. (See figure 2.3)

The formula for the Haversine Distance can be written as follows:

$$d = 2R \times \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right)$$

where,

$\phi\, denotes\, the\, latitudes, \lambda\, denotes\, the\, longitudes,\; and\, R\, is\, the\, radius\, of\, the\, Earth$

Chapter 3

Implementation and Results

3.1 Data Source

For this particular implementation, the Spatial Distribution of population in 2020 [6] dataset is used. This dataset is in a Geotiff format and follows the WGS84 format with a resolution of 30 arc-seconds. This would be approximately 1 Km at the equator. This dataset had the most reasonable resolution to make the analyses more feasible without blowing up the computational cost. The dataset was aggregated by School of Geography and Environmental Science, University of Southampton; Department of Geography and Geosciences, University of Louisville; Departement de Geographie, Universite de Namur) and Center for International Earth Science Information Network (CIESIN), Columbia University and was funded by the Bill and Melinda Gates Foundation.

3.2 Libraries Used

While the aforementioned approaches were implemented separately, a lot of libraries were shared between the different approaches. All the implementation was done in Python 3.9 and the following libraries were used in the realisation of these ideas:

3.2.1 Rasterio

Rasterio[11] reads and writes GeoTIFF and other formats to organize and store gridded raster datasets which include satellite images and models for terrains, etc. It provides a Python API based on Numpy N-dimensional arrays and GeoJSON.

3.2.2 sklearn

sklearn or scikit-learn[12] is an open source tool that provides a lot of simple and efficient tools for predictive data analysis.

3.2.3 numpy

numpy[13] is a very popular and a fundamental package for scientific computing with numerous mathematical functionalities.

3.2.4 Geopandas

Geopandas[14] is an open source library that extends the functionalities of the Pandas library to allow spatial data analysis and operations on geometric types.

3.2.5 Shapely

Shapely[15] is used to manipulation of planar geometric objects. This library was helpful in converting projected distances to actual distances on the surface of the planet.

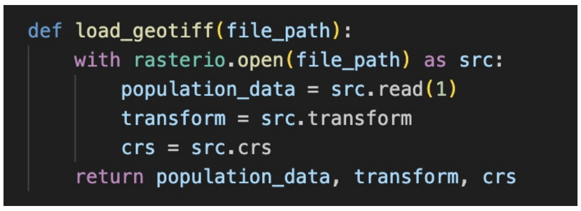
3.2.6 Folium

Folium[16] is a visualisation tool that outputs manipulated python data on a leaflet map.

3.3 Methods Used

While each of the approaches is a separate file, there are methods that have been reused between the different approaches. There are namely the loadpopulationdata and createmap methods.

3.3.1 load_population_data

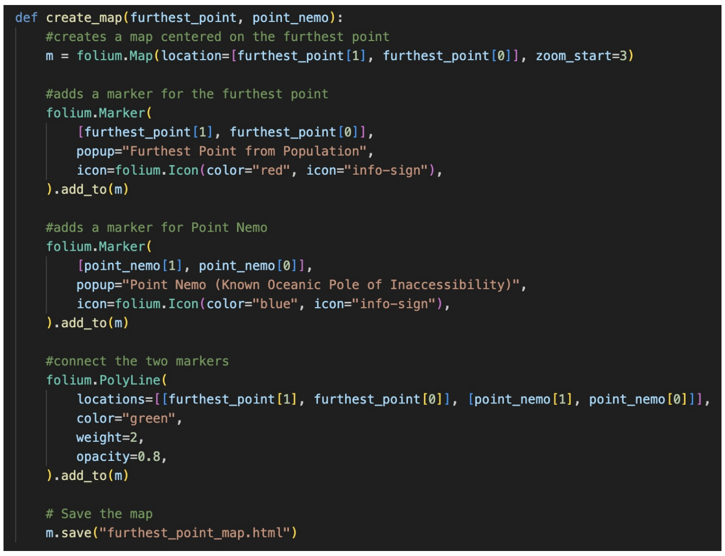


```python
def load_geotiff(file_path):
    with rasterio.open(file_path) as src:
        population_data = src.read(1)
        transform = src.transform
        crs = src.crs
    return population_data, transform, crs
```

Figure 3.1: Load population data method

This method reads the first band of the Geotiff file and performs and affine transformation on it converting the pixel values to geometric coordinates. This makes the dataset ready to be analysed.

3.3.2 create_map

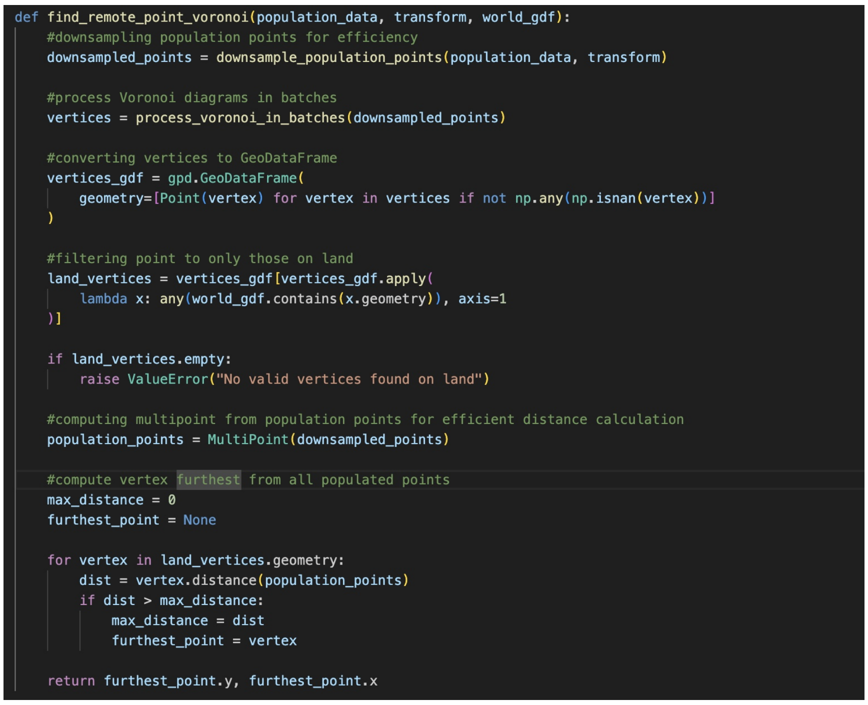


```python
def create_map(furthest_point, point_nemo):
    #creates a map centered on the furthest point
    m = folium.Map(location=[furthest_point[1], furthest_point[0]], zoom_start=3)

    #adds a marker for the furthest point
    folium.Marker(
        [furthest_point[1], furthest_point[0]],
        popup="Furthest Point from Population",
        icon=folium.Icon(color="red", icon="info-sign"),
    ).add_to(m)

    #adds a marker for Point Nemo
    folium.Marker(
        [point_nemo[1], point_nemo[0]],
        popup="Point Nemo (Known Oceanic Pole of Inaccessibility)",
        icon=folium.Icon(color="blue", icon="info-sign"),
    ).add_to(m)

    #connect the two markers
    folium.PolyLine(
        locations=[[furthest_point[1], furthest_point[0]], [point_nemo[1], point_nemo[0]]],
        color="green",
        weight=2,
        opacity=0.8,
    ).add_to(m)

    # Save the map
    m.save("furthest_point_map.html")
```

Figure 3.2: Create Map Method

As the name suggests, this method takes the manipulated data and creates a map visualising the output on a map. It uses the Folium library to do this.

3.4 The Voronoi Tessellation Approach and Results



```
def find_remote_point_voronoi(population_data, transform, world_gdf):
    #downsampling population points for efficiency
    downsampled_points = downsample_population_points(population_data, transform)

    #process Voronoi diagrams in batches
    vertices = process_voronoi_in_batches(downsampled_points)

    #converting vertices to GeoDataFrame
    vertices_gdf = gpd.GeoDataFrame(
        geometry=[Point(vertex) for vertex in vertices if not np.any(np.isnan(vertex))]
    )

    #filtering point to only those on land
    land_vertices = vertices_gdf[vertices_gdf.apply(
        lambda x: any(world_gdf.contains(x.geometry)), axis=1
    )]

    if land_vertices.empty:
        raise ValueError("No valid vertices found on land")

    #computing multipoint from population points for efficient distance calculation
    population_points = MultiPoint(downsampled_points)

    #compute vertex furthest from all populated points
    max_distance = 0
    furthest_point = None

    for vertex in land_vertices.geometry:
        dist = vertex.distance(population_points)
        if dist > max_distance:
            max_distance = dist
            furthest_point = vertex

    return furthest_point.y, furthest_point.x
```

Figure 3.3: Voronoi Tessellation

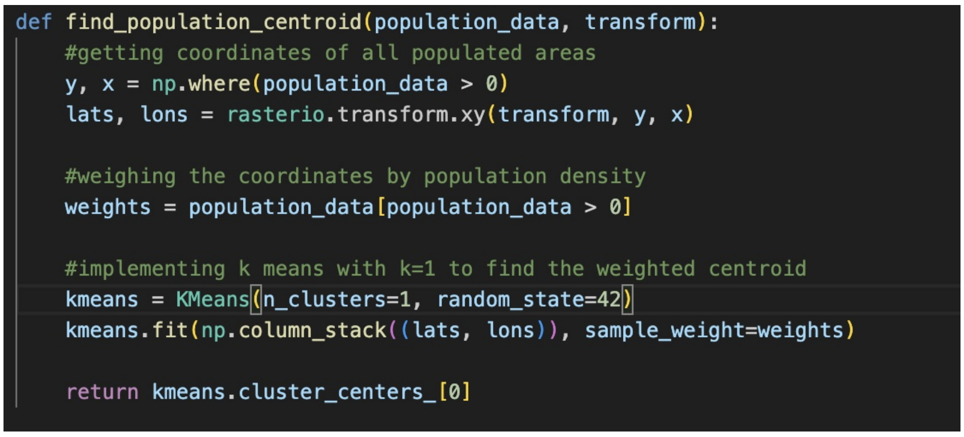
This method works by dividing the space into various regions where each region consists of all points closer to a specific populated area than to any other. The edges of these regions form lines that are equidistant from two populated points. The vertices (intersections) of these lines are points that are equidistant from three or more populated points. These Voronoi regions are being computed in batches to save up on memory cost. Finally each of these vertices are checked as a potential candidate for the furthest away point.

3.4.1 Output

While this methodology is mathematically elegant, it is computationally still very expensive and resulted in memory overflows even after trying downsampling and batch processing.

3.5 The K-Means Approach and Results

Owing to the global nature of the dataset, a K-means clustering algorithm with k=1 is implemented to find the weighted centroid of the global population. The weights are the population densities, ensuring that areas with higher population have more influence on the centroid location. Next, the antipodal point (point on the opposite side of the Earth) is calculated using the population centroid. This point represents the location furthest from the global population center.



```python
def find_population_centroid(population_data, transform):
    #getting coordinates of all populated areas
    y, x = np.where(population_data > 0)
    lats, lons = rasterio.transform.xy(transform, y, x)

    #weighing the coordinates by population density
    weights = population_data[population_data > 0]

    #implementing k means with k=1 to find the weighted centroid
    kmeans = KMeans(n_clusters=1, random_state=42)
    kmeans.fit(np.column_stack((lats, lons)), sample_weight=weights)

    return kmeans.cluster_centers_[0]
```

Figure 3.4: K-Means Method

3.5.1 Output

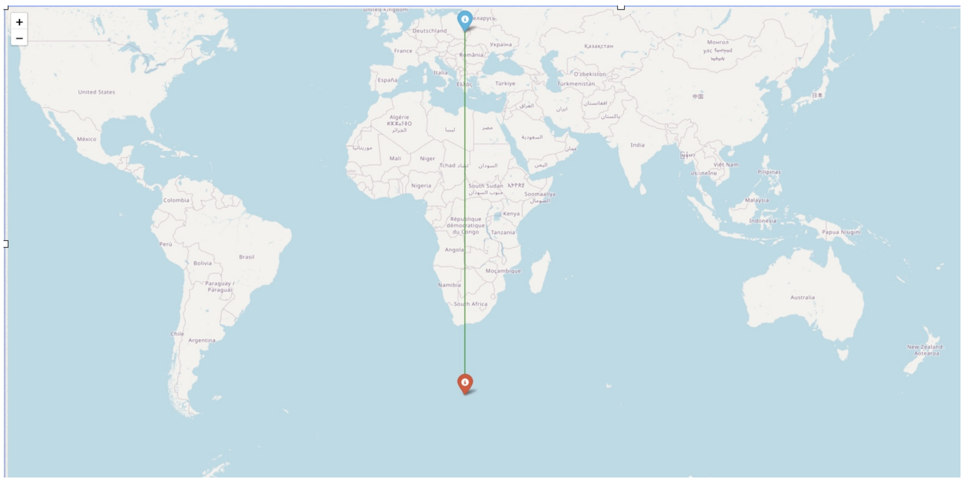
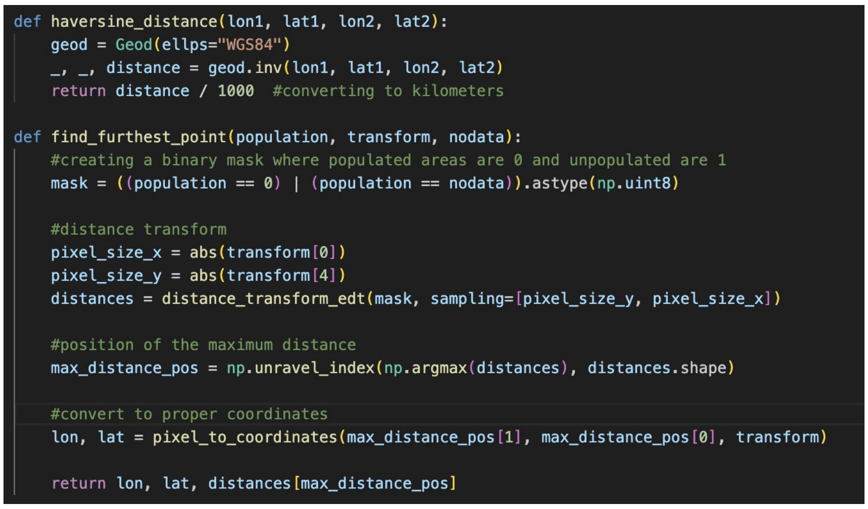


Figure 3.5: Output for K-means

As seen in figure 3.5, the algorithm gives us an output where it computes the furthest away point south of the continent of Africa (marked by the red pin) with the centroid of the human population being in mainland Europe (marked by the blue pin). However this does not seem to be accurate based of intuition or logical reasoning. This may be partly because the algorithm is oversimplifying how population spread can be interpreted. With a larger value of K, it is possible to get more accurate results, however making it computationally feasible has proven to be a challenge.

3.6 Binary Mask Methodology

In this method (see figure 3.6) , the continuous population density data has been transformed into a simple binary (0 or 1) representation. This simplifies the dataset hugely since the precise population density of various geographical locations on planet Earth is not needed to compute a point furthest away from any human population. Any populated pixel regardless of its population density will be a part of the mask. Once the mask has been identified, the haversine distance to a point furthest away from the mask is calclated.



```python
def haversine_distance(lon1, lat1, lon2, lat2):
    geod = Geod(ellps="WGS84")
    _, _, distance = geod.inv(lon1, lat1, lon2, lat2)
    return distance / 1000  #converting to kilometers

def find_furthest_point(population, transform, nodata):
    #creating a binary mask where populated areas are 0 and unpopulated are 1
    mask = ((population == 0) | (population == nodata)).astype(np.uint8)

    #distance transform
    pixel_size_x = abs(transform[0])
    pixel_size_y = abs(transform[4])
    distances = distance_transform_edt(mask, sampling=[pixel_size_y, pixel_size_x])

    #position of the maximum distance
    max_distance_pos = np.unravel_index(np.argmax(distances), distances.shape)

    #convert to proper coordinates
    lon, lat = pixel_to_coordinates(max_distance_pos[1], max_distance_pos[0], transform)

    return lon, lat, distances[max_distance_pos]
```

Figure 3.6: Binary Mask Method

3.6.1 Output

See figure 3.7. The Binary Mask Methodology gives us a point at the coordinates longitude 76.5821, latitude -71.9921. This point lies close to the South Pole and is 4705.12 km away from the nearest populated area. This point is also 6505.25 km away from Point Nemo. This result seems to be reasonably accurate and is the furthest a point has been computed by any of the aforementioned methodologies. However, one key point must be noted here. None of the currently available datasets include the population of researchers and other individuals that live at various Antarctica Research Stations. If this data were available, the point would have likely shift towards the Pacific closer to Point Nemo.

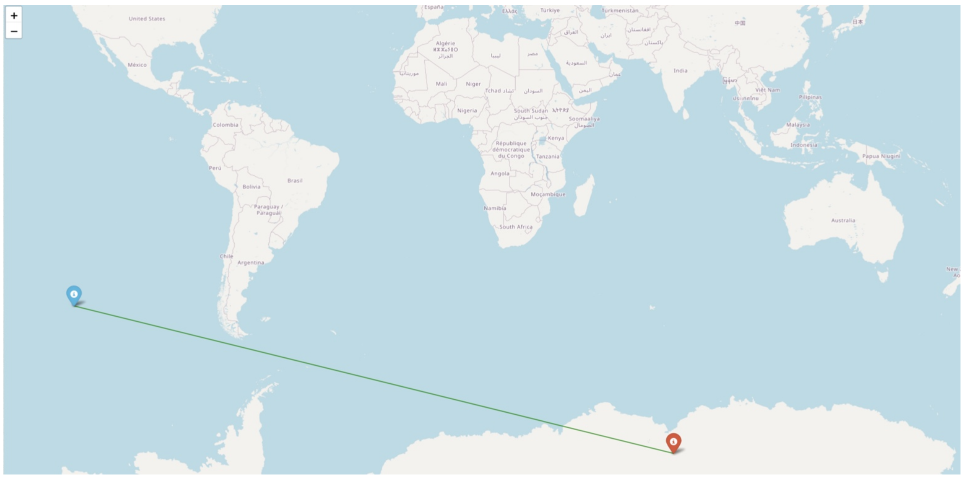


Figure 3.7: Binary Mask Output

3.6.2 Conclusion and Future Work

This project successfully developed a methodology to identify the point furthest from human population using geospatial analysis techniques. Or, in the spirit of the title, the Loneliest Place on Earth is in Antarctica close to the South Pole. By employing techniques like K-means, clustering, Voronoi Tessellation, and binary masking and distance transform algorithms on population density data, a robust approach to finding remote locations has been created. This project has laid a foundation for identifying and analyzing remote locations using geospatial data processing. While the current implementation has its limitations such as rudimentary distance calculations and simplification of population densities, the potential for improvement and expansion is significant. The potential applications of this study involve identifying locations for astronomical observatories or sensitive scientific equipment that requires minimal human interference, conservation, emergency planning, etc. As global population distributions continue to change and data quality improves, the ability to accurately identify and analyze remote locations will become increasingly valuable for a wide range of applications.

Chapter 4

BCS Project Criteria and Self Reflection

4.1 An ability to apply practical and analytical skills gained during the degree programme.

I applied techniques and methodologies learnt over the past year of my degree programme in modules such as last data visualisation, optimisation, data science, artificial intelligence, and algorithms.These include applying algorithms like K-means, binary masking, etc. See chapter 2.

4.2 Innovation and/or creativity.

I have used techniques that I have never learned before in an attempt to solve this problem. These include working with geospatial data and using techniques such a Voronoi Diagrams to help me achieve the desired outcome. See chapter 2.

4.3 Synthesis of information, ideas and practices to provide a quality solution together with an evaluation of that solution.

I have used multiple techniques to achive the desired solution and analysed them accordingly. I have analysed the pros and cons of each methodology and justified why one may be better than the other.

4.4 That your project meets a real need in a wider context.

The potential applications of this study involve identifying locations for astronomical observatories or sensitive scientific equipment that requires minimal human interference, conservation, emergency planning, etc. This study can be used as a foundation to build upon and come up with more robust solutions.

4.5 An ability to self-manage a significant piece of work.

While, I faced a lot of challenges over the 3 months of developing a solution to this project, I can safely say that I have managed this project to the best of my abilities. However, I could work more on managing my time and other resources more efficiently.

4.6 Critical self-evaluation of the process.

While I have successfully seen this project to completion, it was not without its hurdles. I have realised that I need to work on my time management skills harder. I also need to work on asking for help and communicating with my advisor about the project when needed. Additionally, I need to learn how to not let my mental health affect my work and vice versa.

Bibliography

[1] https://inaccessibility.net/

[2] https://oceanservice.noaa.gov/facts/nemo.html

[3] http://www.lukatela.com/nemoLibrary/index.html

[4] https://github.com/google/open-location-code

[5] A N Aini et al 2023 IOP Conf. Ser.: Earth Environ. Sci. 1245 012014

[6] WorldPop (www.worldpop.org - School of Geography and Environmental Science, University of Southampton; Department of Geography and Geosciences, University of Louisville; Departement de Geographie, Universite de Namur) and Center for International Earth Science Information Network (CIESIN), Columbia University (2018). Global High Resolution Population Denominators Project - Funded by The Bill and Melinda Gates Foundation (OPP1134076). https://dx.doi.org/10.5258/SOTON/WP00647

[7] https://zbmath.org/0946.68144

[8] https://uc-r.github.io/kmeans_clustering

[9] https://link.springer.com/chapter/10.1007/978-3-031-26588-4_5

[10] https://ui.adsabs.harvard.edu/abs/2020JPhCS1450a2080M/abstract

[11] https://rasterio.readthedocs.io/en/stable/

[12] https://scikit-learn.org/stable/

[13] https://numpy.org/

[14] https://geopandas.org/en/stable/

[15] https://pypi.org/project/shapely/

[16] https://python-visualization.github.io/folium/latest/

[17] https://www.uber.com/en-GB/blog/h3/

[18] Keogh E, Mueen A. Encyclopedia of Machine Learning and Data Mining. Curse of Dimensionality. 2nd ed. Springer, Boston, MA, 2017, 314–315