Member-only story

An Open Data-Driven Approach to Optimising Healthcare Facility Locations Using Python

A tutorial in Python with an open data stack

Parvathy Krishnan · Following

14 min read · Draft

Listen Share More

Image generated by Authors in Midjourney

This work was co-authored with Prof Joaquim Gromicho and Kai Kaiser. The author(s) are responsible for all errors and omissions.

According to research published in 2020 on global maps of travel time to healthcare facilities, 43.3%

> (3.16 billion people) cannot reach a healthcare facility on foot within one hour.

Accurately calculating travel time and distance to healthcare facilities is fundamental in assessing healthcare accessibility, particularly in regions where barriers to access can significantly impact public health outcomes. These calculations are vital for resource allocation, emergency services, healthcare utilization, equitable healthcare access, and strategic planning for future facilities. However, to calculate this, a lot of data crunching is needed, including the location of hospitals, population distribution, and travel time calculations based on road network data such as OpenStreetMaps or APIs such as Google or Mapbox. Geographic variability, such as differing terrains, road conditions, and weather, can drastically affect travel times, especially in mountainous or poorly connected regions. The availability and type of transportation also play crucial roles, with many rural areas lacking reliable public or personal transport options, complicating access to healthcare. Furthermore, the accuracy and availability of geographic and infrastructural data are often limited, particularly in developing countries, leading to less precise travel time estimations. In addition, travel times are not static; they fluctuate with daily traffic patterns and seasonal weather changes, requiring dynamic assessment methods.

This blog leverages the power of open-source data and tools to address the challenge of calculating physical access in countries and administrative regions, especially when population censuses are infrequent, and road network and health facility data are not regularly updated. This is exemplified in Timor-Leste, where healthcare access is hindered by systemic inefficiencies and financial burdens, making the need for up-to-date information even more critical.

> This blog summarises how to download data on healthcare facilities, and population to calculate the percenatge of people who have

We start by downloading Timor-Leste's administrative boundary data from GADM and augment this with high-resolution population data from Meta and healthcare facility locations from OpenStreetMap. We integrate and visualize these datasets

using Python libraries like geopandas and Folium, creating an informative geographic overview. Next, we employ the Openroute service API to calculate travel

This leads to a detailed visualization of healthcare coverage on an interactive map, highlighting areas with and without access.

Finally, we run an optimization model to identify potential locations for new healthcare facilities, further enhancing our understanding of healthcare distribution. This approach offers insights into Timor-Leste's healthcare landscape. It serves as a model that can be replicated in other regions, showcasing the power of open data and geospatial analysis in public health planning.

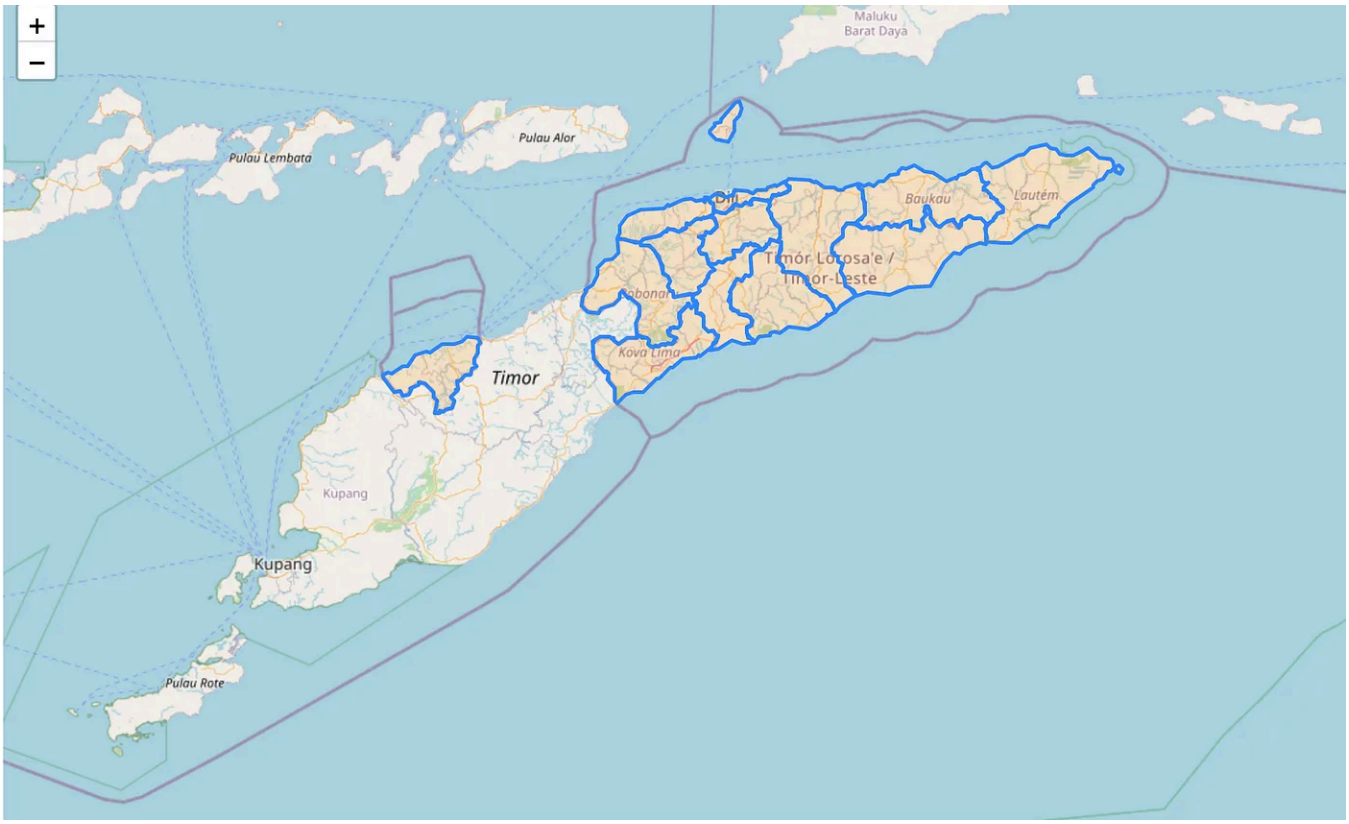
Extracting Timor-Leste's Administrative Boundaries Using GADM Data and visualizing using Folium in Python

The following code snippet initializes a downloader for GADM (Global Administrative Areas) data, specifying version 4.0. It then retrieves the administrative boundary data for Timor-Leste, focusing on the first administrative level, such as districts or provinces. After obtaining the geospatial data, the script visualizes these boundaries on a Folium map, using OpenStreetMap as the base map. The map includes simplified geometries for each administrative region, styled with an orange fill colour, and popups displaying each region's name. This visualization helps in understanding the geographical layout of Timor-Leste's administrative divisions.

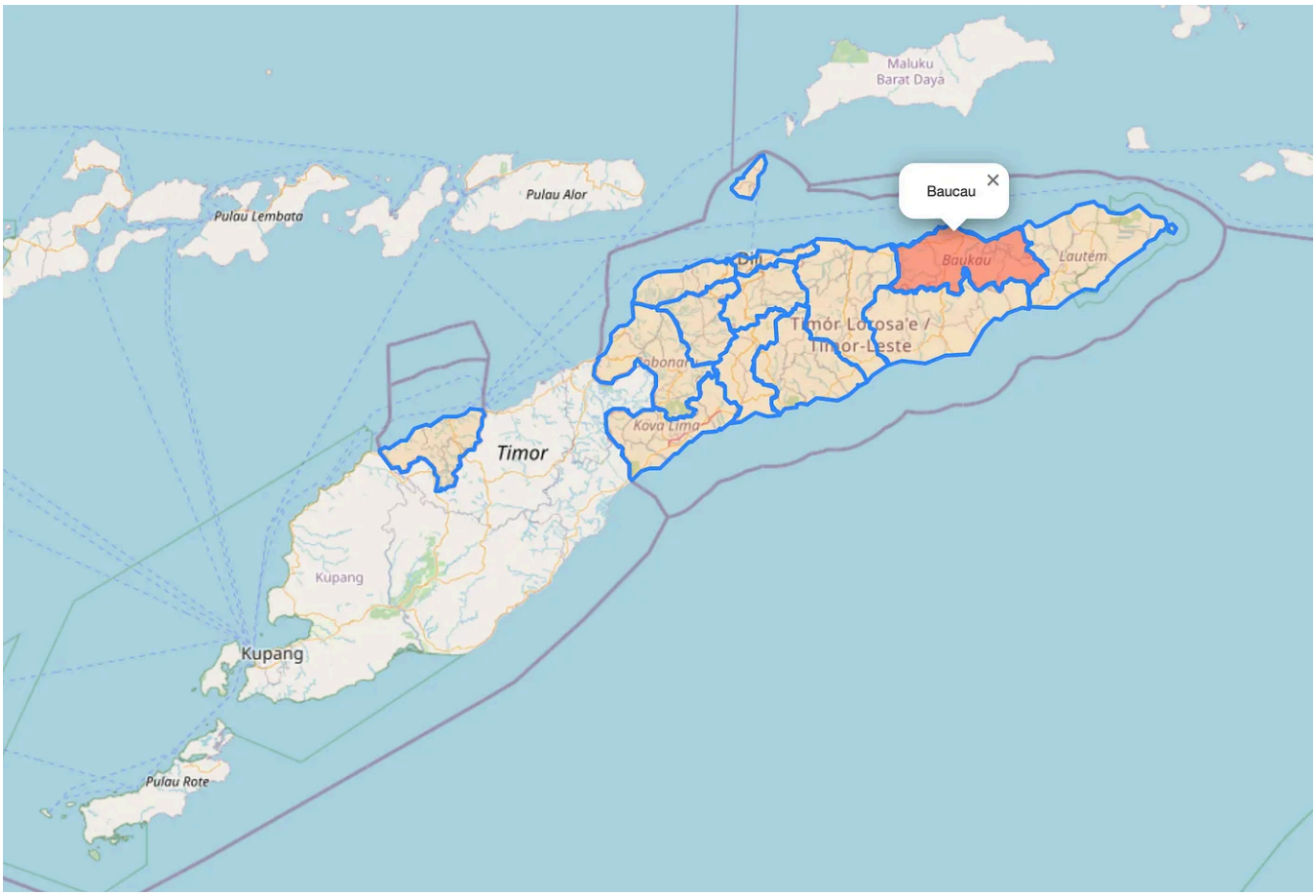
```python
# Initialize the GADMDownloader with the specified version (in this case, version 4.0)
downloader = GADMDownloader(version="4.0")

# Define the country name for which you want to retrieve administrative boundary data
country_name = "Timor-Leste"

# Specify the administrative level you are interested in (e.g., 1 for districts or provinces)
ad_level = 1

# Retrieve the geospatial data for the selected country and administrative level
gdf = downloader.get_shape_data_by_country_name(country_name=country_name, ad_level=ad_level)

# Display the first 5 rows of the obtained geospatial data for a quick preview
gdf.head(5)
```

pisa_gadm.py hosted with ❤ by GitHub view raw

ID_0	COUNTRY	ID_1	NAME_1	VARNAME_1	NL_NAME_1	TYPE_1	ENGTYPE_1	CC_1	HASC_1	ISO_1	geometry
TLS	Timor-Leste	TLS.1_1	Aileu			Distrito	District		TP.AL		MULTIPOLYGON (((125.58068 -8.82766, 125.57981 ...
TLS	Timor-Leste	TLS.2_1	Ainaro			Distrito	District		TP.AN		MULTIPOLYGON (((125.61008 -8.93925, 125.60912 ...
TLS	Timor-Leste	TLS.3_1	Ambeno	Oecussi		Distrito	District		TP.AM		MULTIPOLYGON (((124.35570 -9.48334, 124.35561 ...
TLS	Timor-Leste	TLS.4_1	Baucau			Distrito	District		TP.BC		MULTIPOLYGON (((126.45296 -8.67774, 126.45297 ...
TLS	Timor-Leste	TLS.5_1	Bobonaro			Distrito	District		TP.BB		MULTIPOLYGON (((125.42563 -9.00174, 125.42568 ...



Timor Leste's Administrative Boundaries (Level 1) extracted through GADMDownloader (Image by Authors)



Bacau — Administrative Boundary selected for analysis (Image by Authors)

Downloading and visualizing High-Resolution Population Density Maps from Meta

This section discusses downloading and visualizing high-resolution population density maps from Meta, developed in collaboration with CIESIN. These maps provide detailed population estimates at a 30-meter resolution, with demographic breakdowns covering various groups, and are publicly available for over 160 countries. The maps are created by modelling population growth using census data, analyzing satellite imagery to detect buildings, calculating building density, and distributing population data across tiles based on this density. These maps, downloadable from HDX or AWS, are crucial for urban planning, public health, disaster response, and environmental studies.

In the following Python code snippet, a function `fb_pop_data` is defined to retrieve 2020 Facebook population data for Timor-Leste (identified by the ISO code 'TLS'). The data is downloaded, processed into a GeoDataFrame, and aligned with the selected administrative boundary's coordinate reference system. The population within the area of interest (the selected administrative region) is then calculated and displayed.

```
print('Total Population:',round(population_meta['population'].sum()/1000000,2),'million')
```

```
Data downloaded
Total Population: 2.67 million
```

```
population_meta.head(2)
```

	ID	xcoord	ycoord	population	geometry
0	0	125.761250	-8.004306	10.489956	POINT (125.76125 -8.00431)
1	1	125.761528	-8.004306	10.489956	POINT (125.76153 -8.00431)

```
# Perform a spatial join to find population within the selected administrative boundary
population_aoi = gpd.sjoin(population_meta, selected_gadm)
print('Total Population (Area of Interest -', selected_adm1,'):',round(population_aoi['population'].sum()))
```

```
Total Population (Area of Interest - Baucau ): 123895
```

> *This returns a population of 123,895 in Baucau.*

Mapping Healthcare Facilities in Timor-Leste Using OpenStreetMap Data

This code segment is dedicated to retrieving and analyzing data on healthcare facilities in Timor-Leste, specifically hospitals and clinics. It performs the following functions:

- Queries OpenStreetMap via the Overpass API for hospital and clinic locations across Timor-Leste.
- Extracts essential data such as names, coordinates, and amenities and stores it in separate DataFrames for hospitals (`df_hospitals`) and clinics (`df_clinics`).

Merges these datasets into a single GeoDataFrame (`df_health_osm`) for spatial analysis.

- Executes a spatial join to determine the count of healthcare facilities within a specified area of interest (AOI), providing valuable insights for assessing healthcare accessibility in Timor-Leste.

```
Number of hospitals and clinics extracted: 130
Number of hospitals and clinics in AOI ( Baucau ): 14
CPU times: user 55.2 ms, sys: 879 µs, total: 56.1 ms
Wall time: 1.11 s
```

> *This returns a total of **14 hospitals** and clinics in Baucau.*

Assessing Healthcare Accessibility with Isochrone Analysis

This section of the blog introduces a function, `get_isochrone_osm`, designed to calculate isochrones for healthcare facilities in Timor-Leste. An isochrone is a polygon representing areas reachable within a specified time from a location, typically a hospital. The function uses the OpenRouteService API to generate these

polygons based on a **60-minute travel time** with the mode as **walking**. The resultant geometric shapes are then used to determine the population with access to each facility. This analysis, visualized on a Folium map, is crucial for understanding healthcare accessibility and planning for improved regional healthcare coverage.

Population with Access = 47.11%

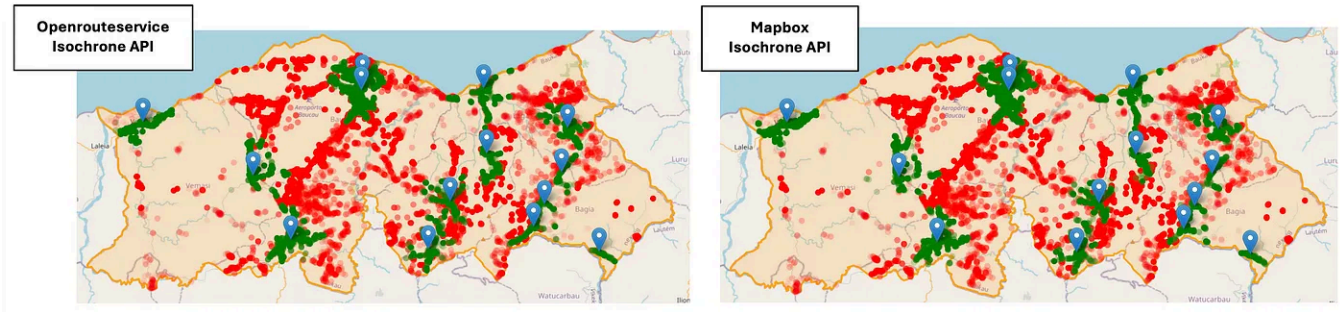
A crucial aspect of assessing healthcare accessibility is understanding the reliability and sensitivity of our analysis based on the underlying data sources. The road network data from OpenStreetMap (OSM) is used in calculating travel times to healthcare facilities in the previous step. However, this data can vary significantly in

quality and completeness, which may affect the accuracy of our isochrone maps and, subsequently, our conclusions about accessibility. So, it is advisable to conduct sensitivity analysis using alternative data sources to address these uncertainties. For regions where OSM data might be less reliable or outdated, utilizing APIs like Mapbox or Google can provide a different perspective on accessibility based on possibly more current or complete road network information. If resources allow, these APIs can complement or validate our findings from OSM data.

To illustrate, let's integrate Mapbox to perform an isochrone analysis. This example shows how to calculate the area accessible within a 60-minute walk from existing hospitals using the **Isochrone API** by Mapbox.

Population with Access = 48.75%

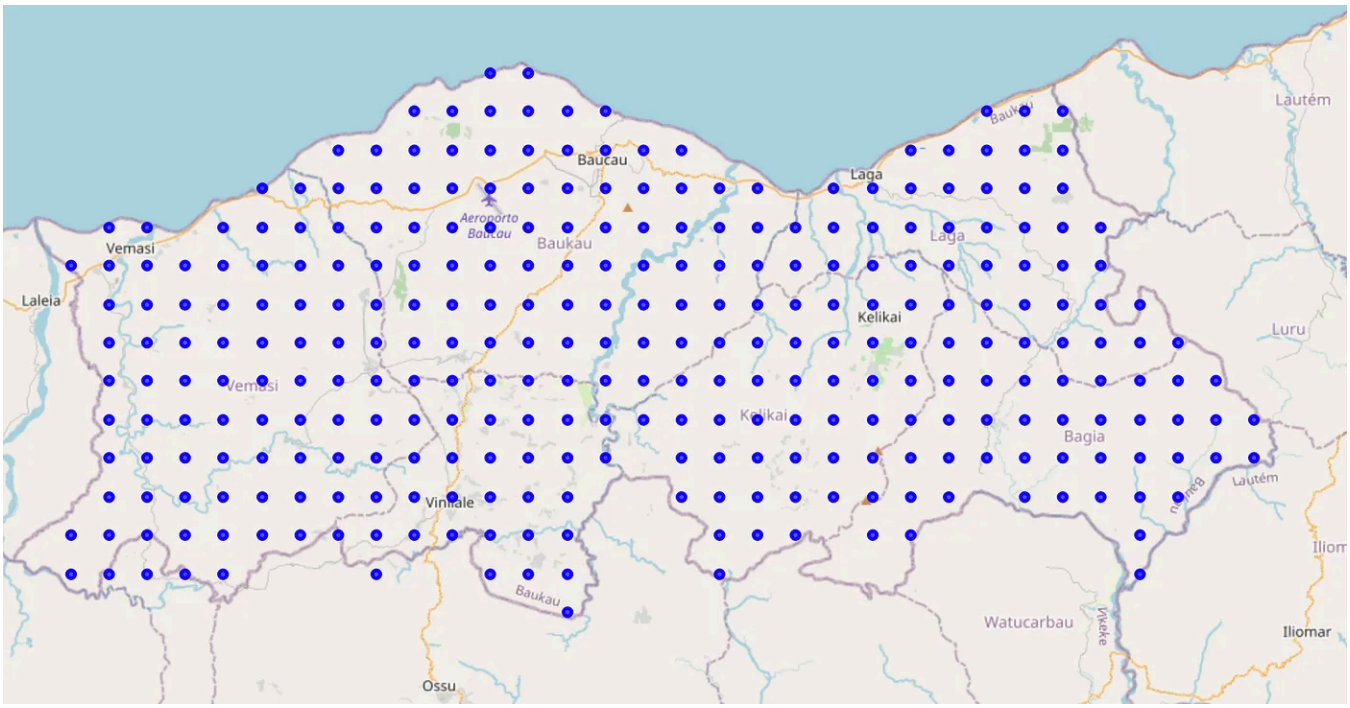
The code below allows us to create a multi-layered display on the map. It outlines administrative boundaries with orange GeoJson objects and marks hospital locations with blue pins, each featuring a popup with the hospital's name. The map also differentiates populations based on their access to healthcare, using red circle markers for those without access and green for those with access. The opacity of these markers is determined by population counts, enabling a clear visual distinction between densely populated and well-served areas versus those that are sparsely populated and underserved, thus illustrating disparities in healthcare accessibility.



Left: Openrouteservice Analysis (47.11% Coverage) and Right: Mapbox Analysis (48.75% Coverage) — This map compares healthcare facility catchment areas in Baucau, Timor-Leste, using two different routing services — Openrouteservice and Mapbox. (Image by Authors)

Representational Potential Locations Grid of the Administrative Boundary

Optimising the placement of new hospitals and clinics is essential to ensure comprehensive healthcare coverage. This often involves analyzing potential locations which, in the absence of specific site recommendations from official sources, can be approximated through a representational grid within the target area. Such a grid provides a starting framework for considering where to establish new healthcare facilities to maximize accessibility for the underserved population. The code snippet outlines a Python function `generate_grid_in_polygon` designed to create this representational grid within a given geographic area, represented as an `MultiPolygon` object. The function generates a series of evenly spaced points—determined by the `spacing` parameter—across the extent of the provided geometry, resulting in *307 potential locations in Baucau.*



Potential locations as a grid of 0.02 degrees (Image by Authors)

To optimize the placement of potential healthcare facilities, our analysis involves calculating isochrones for each proposed location, utilizing the same 60-minute walking parameter previously applied to existing facilities. This method identifies the geographic areas that would be accessible within one hour by foot from any given potential site. These isochrone calculations are critical for the mathematical model that determines optimal facility locations, as they allow us to assess the potential impact and accessibility of new healthcare centres. Utilizing this data, we decide which population segments lack access and can be served by these potential facilities. We then aggregate access data from existing and potential locations to calculate the ***maximum possible access*** if all the potential locations are opened along with existing facilities.

> Maximum access attainable with this potential location list = 86.17 %

Mathematical Optimization

We will now employ *Mathematical Optimization* to determine the optimal subset of hospitals to open. For those unfamiliar with _Mathematical Optimization_, we recommend starting with a hands-on introduction in this Jupyter book.

At its core, *Mathematical Optimization* involves creating mathematical models that act as digital twins of the real-world scenarios we aim to optimize. After developing those models, we input relevant data to create specific instances of an optimization

problem. These instances are then solved using an appropriate solver to discover the best feasible solutions, which we call the optimal solution.

Modelling is a conceptual process while coding the models is a technical craft. We use the package pyomo for the latter as the Jupyter book does. Please note that the model discussed subsequently is featured as Exercise 3.1 in a forthcoming textbook from Cambridge University Press, the aforementioned Jupyter book being its online companion.

> *A mathematical optimization model can be seen as a blueprint for the intended optimal solution. Once instantiated, the model is processed by a solver that seeks the optimal solution if one exists.*

We can utilize powerful solvers to solve instances. For problems like the one we will describe, Gurobi is an outstanding commercial solver, while HiGHS, an excellent open-source alternative, is available under the MIT license.

The modelling process begins by identifying the decisions to be made, leading to defining decision variables. After naming these decisions and variables, we formalize the objective and constraints using the functions of those variables.

- The **objective function** measures the quality of a solution.
- **Constraints** ensure that a solution adheres to all necessary rules to be considered feasible.

In our case, and many others, the functions will be linear, and our variables will have a *binary* nature to represent *yes/no* decisions.

For example, we need one variable per household to determine if an accessible open hospital serves it and another variable per hospital to indicate whether it is open. Typical mathematical notation for expressing models starts by naming the sets supporting the variables' indices and the model parameters derived from the data.

For our optimization challenge, these include:

Sets

- I — the set of households

- J — the set of potential hospital locations
- Ji — the set of potential hospital locations within reach of household $i \in I$. Note: $Ji \subseteq J$.

Parameters

- vi — the headcount of household $i \in I$.

Model `maximal covering` as in the article by Church and ReVelle

This model defines variables zi for each household $i \in I$ to indicate if that household can be served by a hospital that is opened at $j \in J$, leading to the complete model as follows:

$$
\begin{aligned}
\max \quad & \sum_{i \in I} v_i z_i & \\
\text{subject to:} \quad & z_i \leq \sum_{j \in J_i} x_j & \forall i \in I \\
& \sum_{j \in J} x_j \leq p & \\
& x_j \in \{0, 1\} & \forall j \in J \\
& z_i \in \{0, 1\} & \forall i \in I
\end{aligned}
$$

The first line states the objective of maximizing the total headcount of the households served, while the second line (after _subject to:_) lists the first constraint: each household is only served if at least one hospital within reach is open. Then, the number of hospitals to open constraints the selection, and finally, the binary nature of the variables used is specified.

The model above selects up to p hospitals. In the original paper, Church and ReVelle selected exactly p hospitals, but our model has advantages to be discussed later.

Implementing the mathematical model

After the model is designed, implementation translates the concepts from the above mathematical expressions into code. The translation is more or less one-on-one, the main difference being that the variables are declared before using them, as always in programming. In contrast, the mathematical formulation traditionally declares the variables at the end. We use the package `Pyomo` to code our model.

1. **Defining the Optimization Model**

The `model_max_covering` function defines the mathematical optimization model using Pyomo. It takes several parameters:

- `w` : A dictionary containing population counts for each household ID
- `I` : A set of household IDs
- `J` : A set of potential hospital location IDs
- `JI` : A dictionary mapping household IDs to sets of potential hospital IDs that can serve them
- `p` : The maximum number of hospitals to open

The model sets up the decision variables `x` (binary, whether to open a hospital at location j) and `z` (binary, whether household i is served). The objective maximizes the total population covered, subject to constraints that ensure households are only marked as served if a reachable hospital is opened. The total number of opened hospitals does not exceed the budget - `p` .

2. Preparing Input Data

The code then extracts relevant data from the DataFrames:

- `J_existing` : Set of existing hospital IDs
- `J_potential` : Set of potential new hospital location IDs
- `I1` : Set of all household IDs
- `IJ_existing` : Dictionary mapping existing hospital IDs to sets of households they can serve

- `IJ_potential` : Dictionary mapping potential hospital IDs to sets of households they can serve
- `JI1` : A reverse mapping from the previous two, giving sets of potential hospitals for each household

3. Calling the Optimization Model

The `get_ids` function is defined to instantiate and solve the optimization model for a given budget (number of hospitals to open). It creates the model instance `m1` , fixes the existing hospitals to be open using a constraint, and adjusts the budget `p` by adding the number of existing hospitals.

The model is then solved using the HiGHS solver, and the IDs of the selected (opened) hospitals are returned.

4. Finding the Optimal Solution

In the following code block, a list `result_list` is created to store the results for different budgets (*number of hospitals to open*).

The loop iterates over different budget values, from 1 to the total number of potential locations. For each budget:

- The `get_ids` function is called to get the optimal set of hospital IDs to open
- The populations with and without access are calculated based on the selected hospitals
- The percentage of the population with access is computed
- This percentage and the corresponding budget are appended to the `result_list`

This process allows you to explore the trade-off between the number of hospitals opened and the percentage of the population covered, ultimately leading to a Pareto Curve visualization.

The ***Pareto curve*** visualization refers to plotting the results from the optimization model runs with different budgets (number of hospitals to open) on a 2D graph. This allows for visualising the trade-off between the number of facilities opened and the resulting population coverage. Specifically, the x-axis represents the number of hospitals opened (budget), while the y-axis shows the percentage of the population with access to healthcare facilities. As the budget (and consequently, the number of open hospitals) increases, the population coverage tends to improve. However, the rate of improvement usually diminishes, leading to a curve that flattens out as more hospitals are added beyond a certain point. This curve is known as the Pareto curve or Pareto frontier, as it represents the set of non-dominated solutions — solutions where an increase in one objective (e.g., population coverage) can only be achieved by sacrificing the other objective (e.g., increasing the number of hospitals/budget).

The Pareto curve visualization is valuable in this context because it helps decision-makers understand the trade-offs involved and make informed choices. For example:

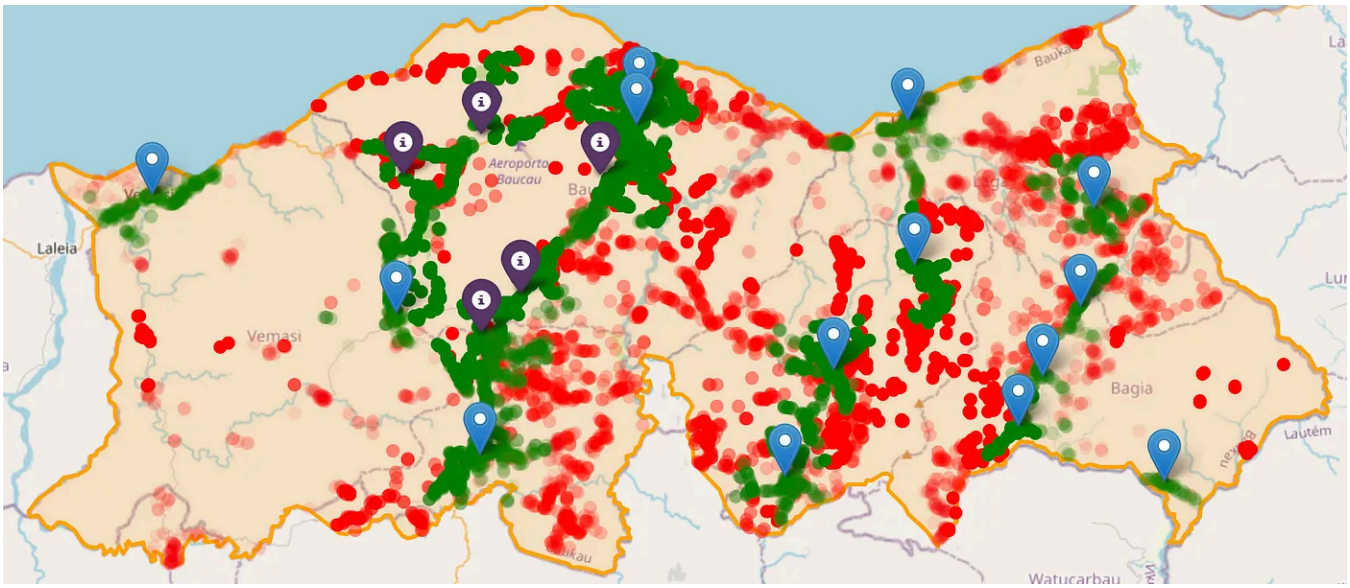
1. If the curve has a steep initial slope, adding a few hospitals can significantly improve population coverage, making such investments highly effective.
2. If the curve flattens quickly, opening additional hospitals beyond a certain point may not substantially increase coverage, suggesting diminishing returns.
3. The curve can reveal “knees” or points where the trade-off becomes less favourable, helping identify potential sweet spots balancing the two objectives.

The code below uses the Plotly.express module to visualise the Pareto frontier for the optimization results.

Pareto curve depicting the trade-off between the number of healthcare facilities (existing and new) and the percentage of the population with access in Baucau, Timor Leste (Image by Authors)

Now, to the most critical question: Where are these optimized locations?

Our optimization model has identified several strategic locations for new healthcare facilities, enhancing overall access. With a budget constraint of opening 5 additional facilities, the analysis suggests these locations as the most effective choices for enhancing coverage, increasing the percentage of the population with access from 48.7% (with only existing facilities) to 63.6%.



Strategically Optimized: This map showcases the **five new healthcare facilities** selected by the optimization model. Marked in violet, these locations are positioned to expand healthcare access to 63.6% of the population, up from the initial 48.7%. (Image by Authors)

In conclusion, this blog post demonstrates the power of leveraging open data, geospatial analysis, and mathematical optimization techniques to tackle the critical challenge of equitable healthcare access planning. By harnessing Python's capabilities and integrating diverse data sources, we have developed a robust methodology for evaluating healthcare facility locations and identifying optimal sites for new facilities. The comprehensive approach, starting with data acquisition and processing, moving on to accessibility analysis using isochrone calculations, and finally employing optimization models, has proven to be an effective decision-support framework. The Pareto curve visualization enables stakeholders to understand the trade-offs between the number of facilities and population coverage, facilitating informed decision-making based on resource constraints and desired accessibility levels. The interactive map showcasing the optimized facility locations is a valuable communication tool, allowing stakeholders to visualize the proposed solutions and assess their potential impact on various regions. By highlighting densely populated underserved areas, this analysis pinpoints the regions that would benefit most from the strategically placed new facilities, ensuring that healthcare resources are allocated efficiently and equitably.

> While the case study focused on Timor-Leste, the methodology presented in this blog is universally applicable and can be adapted to other regions or countries facing similar healthcare accessibility challenges. By leveraging the power of open data and Python's ecosystem, this approach empowers decision-makers with data-driven insights, enabling them to prioritize investments and enhance healthcare accessibility for needy populations.

The complete code for this tutorial can be found in the Colab Notebook. Even if you are not a Python programmer, we hope this contribution gives you an intuitive sense of the possibilities and processes for leveraging openly available geospatial data for a new generation of decision support using optimization models.

Geospatial | Optimization | Data Science | Healthcare | Python

Following

Written by Parvathy Krishnan

981 Followers

Lead Data Scientist | CTO at Analytics for a Better World | Public Sector Consultant

Recommended from Medium

mo husseini

50 Completely True Things

This is a repost of a list of posts I made to Threads last fall.

5 min read · May 3, 2024

9.4K 191

Hazel Paradise

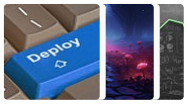
How I Create Passive Income With No Money

many ways to start a passive income today

5 min read · Mar 27, 2024

11.6K 280

Lists



Predictive Modeling w/ Python

20 stories · 1182 saves

Coding & Development

11 stories · 607 saves

Practical Guides to Machine Learning

10 stories · 1426 saves

ChatGPT prompts

47 stories · 1544 saves

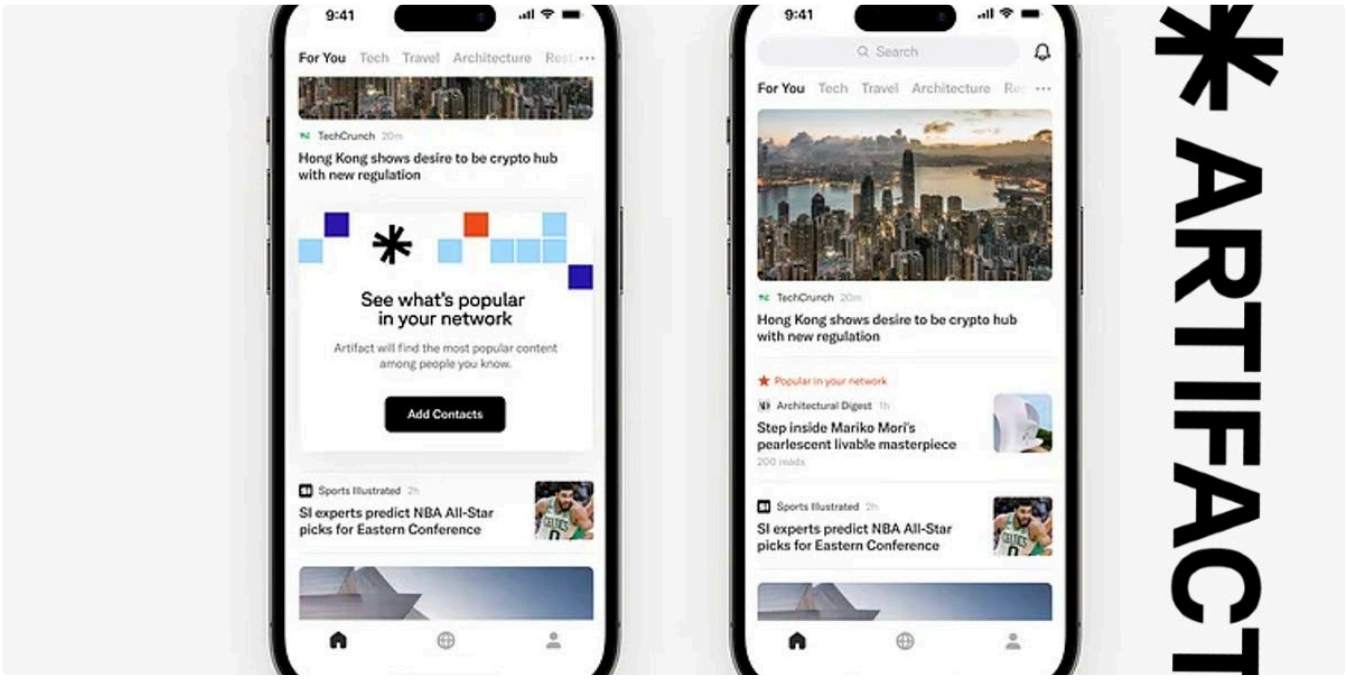
Kallol Mazumdar in ILLUMINATION

I Went on the Dark Web and Instantly Regretted It

Accessing the forbidden parts of the World Wide Web, only to realize the depravity of humanity

8 min read · Mar 13, 2024

18.9K 351



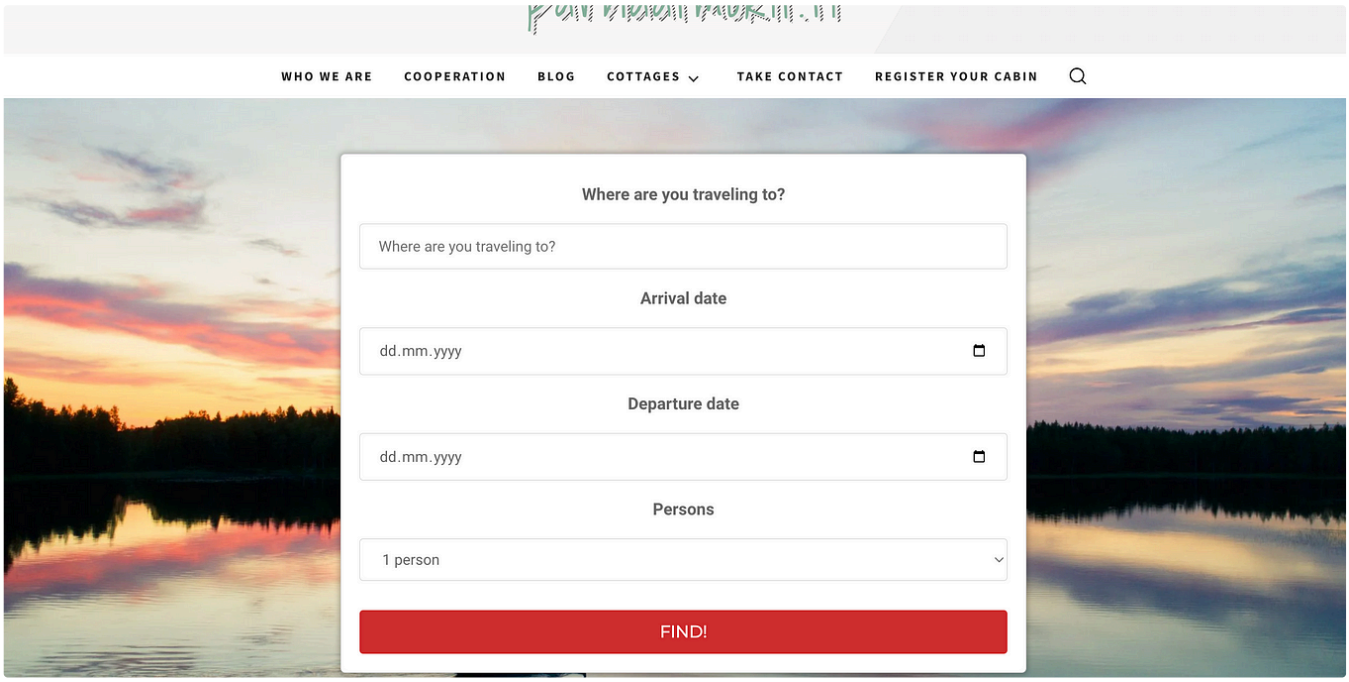
Gowtham Oleti

Apps I Use And Why You Should Too.

Let's skip past the usual suspects like YouTube, WhatsApp and Instagram. I want to share with you some less familiar apps that have become...

11 min read · Nov 14, 2023

22K 424



Artturi Jalli

I Built an App in 6 Hours that Makes $1,500/Mo

Copy my strategy!

· 3 min read · Jan 23, 2024

18.1K 194

Erin Anne Lynch

What I wish I'd known about unemployment

Have you ever watched a friend spend months unemployed?

8 min read · Apr 22, 2024

7.3K 209

See more recommendations