City of Melbourne Open Data | DATA SCIENCE Team

Distance Matrix Implementation for Route Optimization

Introduction

The distance matrix is a crucial component in the optimization of waste collection routes. It represents the shortest and most efficient paths between multiple points, which in this project are the locations of waste bins. This document outlines the methodology used to generate and utilize the distance matrix, its role in route optimization, and its impact on the operational efficiency of waste collection services.

Distance Matrix Definition

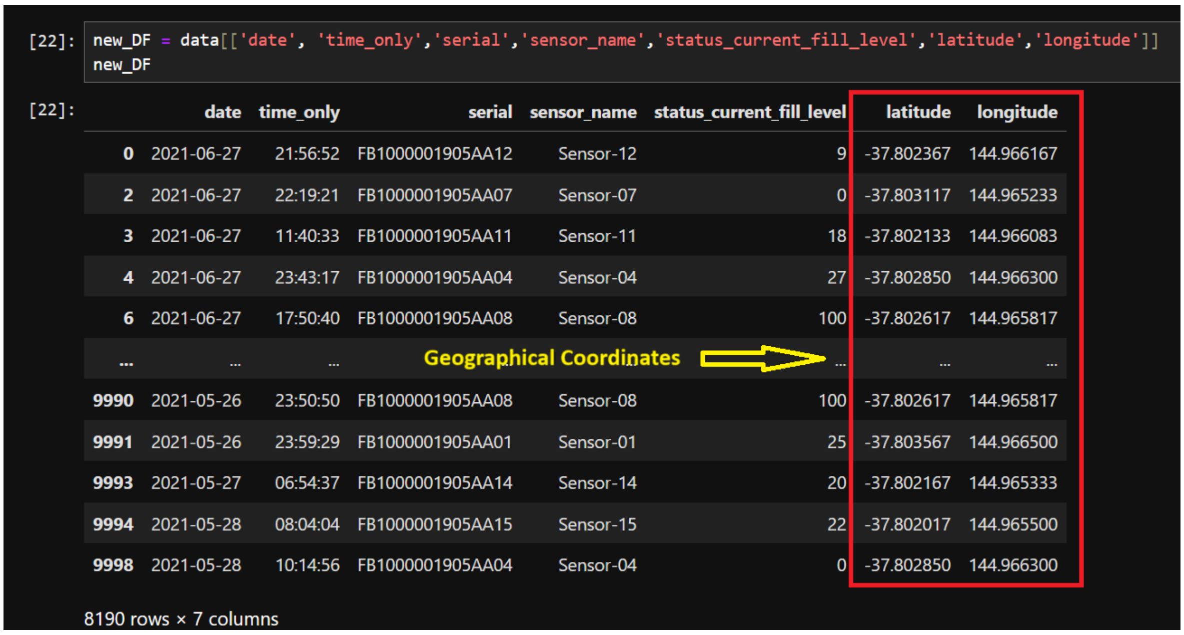
A distance matrix is a two-dimensional array where each element $(\boldsymbol{i}, \boldsymbol{j})$ represents the distance from point $(\boldsymbol{i})$ to point $(\boldsymbol{j})$. When considering this project, each point corresponds to a bin location, and the distance is typically measured in terms of travel distance between these points.

Methodology

- **Data Collection**

  The geographical coordinates (latitude and longitude) of each bin are collected from the data set provided by **City of Melbourne Open Data** and the data set was cleaned in the python environment using Jupiter notebook.

  The below image chows the cleaned data set.



```
[22]: new_DF = data[['date', 'time_only','serial','sensor_name','status_current_fill_level','latitude','longitude']]
      new_DF
```

```
[22]:
            date  time_only             serial  sensor_name  status_current_fill_level   latitude   longitude
   0  2021-06-27   21:56:52  FB1000001905AA12    Sensor-12                          9  -37.802367  144.966167
   2  2021-06-27   22:19:21  FB1000001905AA07    Sensor-07                          0  -37.803117  144.965233
   3  2021-06-27   11:40:33  FB1000001905AA11    Sensor-11                         18  -37.802133  144.966083
   4  2021-06-27   23:43:17  FB1000001905AA04    Sensor-04                         27  -37.802850  144.966300
   6  2021-06-27   17:50:40  FB1000001905AA08    Sensor-08                        100  -37.802617  144.965817
 ...         ...        ...                ...          ...                        ...         ...         ...
9990  2021-05-26   23:50:50  FB1000001905AA08    Sensor-08                        100  -37.802617  144.965817
9991  2021-05-26   23:59:29  FB1000001905AA01    Sensor-01                         25  -37.803567  144.966500
9993  2021-05-27   06:54:37  FB1000001905AA14    Sensor-14                         20  -37.802167  144.965333
9994  2021-05-28   08:04:04  FB1000001905AA15    Sensor-15                         22  -37.802017  144.965500
9998  2021-05-28   10:14:56  FB1000001905AA04    Sensor-04                          0  -37.802850  144.966300

8190 rows × 7 columns
```

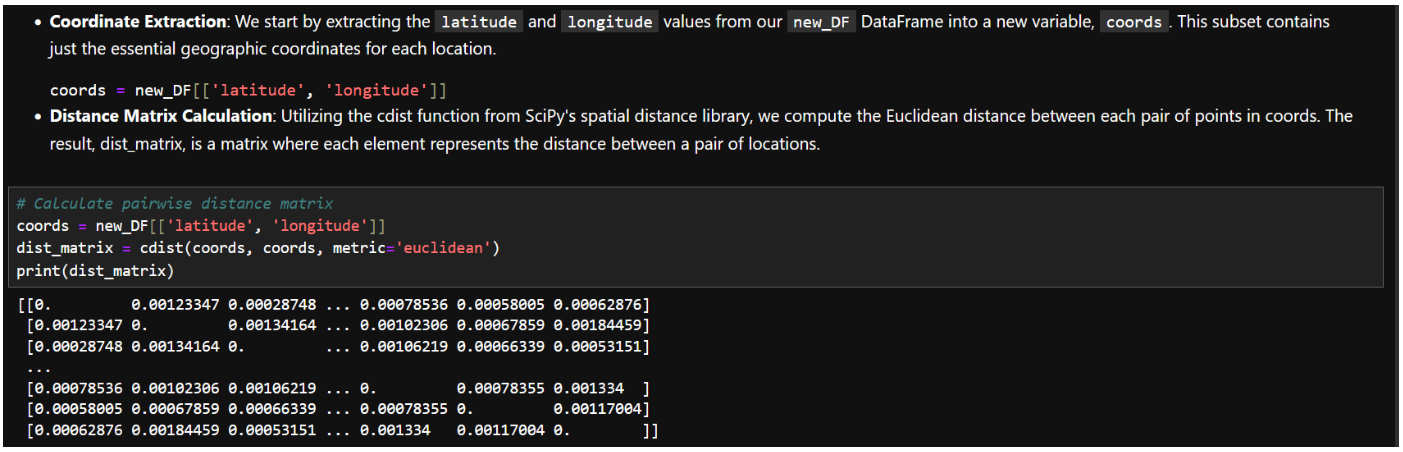
- **Calculation**

The distance matrix is computed using the **cdist** function from the **SciPy** library, which calculates the distance between each pair of points in the dataset. The metric used is **Euclidean**, which computes the straight-line distance between points in a Euclidean plane. The formula used is:

$$d(i,j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

$\boldsymbol{i}$ Were $(x_i - y_i)$ and $(x_j - y_j)$ are the geographical coordinates of bins $\boldsymbol{i}$ and $\boldsymbol{j}$ respectively.

The below image shows the code used.



- **Coordinate Extraction**: We start by extracting the `latitude` and `longitude` values from our `new_DF` DataFrame into a new variable, `coords`. This subset contains just the essential geographic coordinates for each location.

```
coords = new_DF[['latitude', 'longitude']]
```

- **Distance Matrix Calculation**: Utilizing the cdist function from SciPy's spatial distance library, we compute the Euclidean distance between each pair of points in coords. The result, dist_matrix, is a matrix where each element represents the distance between a pair of locations.

```
# Calculate pairwise distance matrix
coords = new_DF[['latitude', 'longitude']]
dist_matrix = cdist(coords, coords, metric='euclidean')
print(dist_matrix)
```

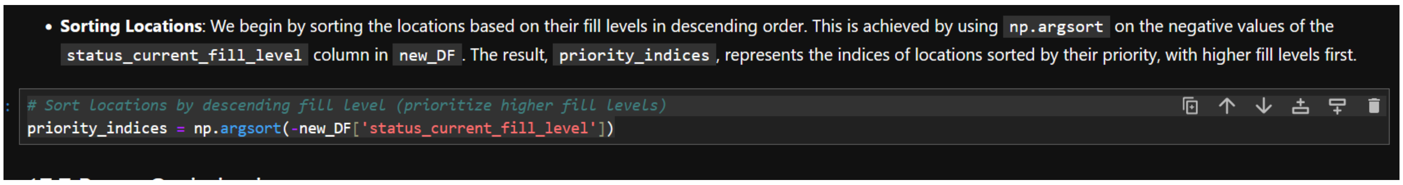
```
[[0.         0.00123347 0.00028748 ... 0.00078536 0.00058005 0.00062876]
 [0.00123347 0.         0.00134164 ... 0.00102306 0.00067859 0.00184459]
 [0.00028748 0.00134164 0.         ... 0.00106219 0.00066339 0.00053151]
 ...
 [0.00078536 0.00102306 0.00106219 ... 0.         0.00078355 0.001334  ]
 [0.00058005 0.00067859 0.00066339 ... 0.00078355 0.         0.00117004]
 [0.00062876 0.00184459 0.00053151 ... 0.001334   0.00117004 0.        ]]
```

Integration in Route Optimization

The distance matrix serves as the input for the route optimization algorithm. The algorithm iteratively selects the shortest distances from the matrix to determine the optimal route that minimizes travel time and maximizes the efficiency of the collection process.

Usage in the Project

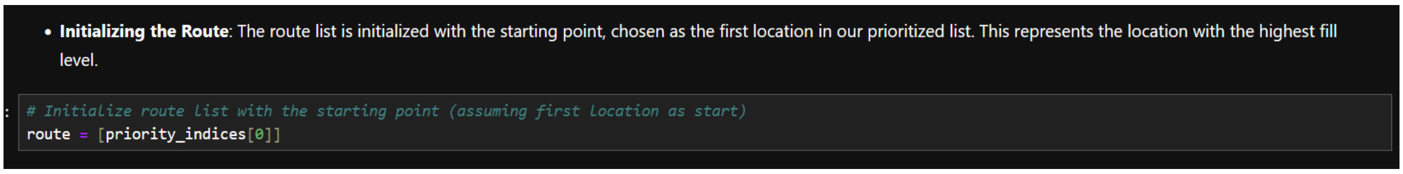
- **Route Calculation:** Once the distance matrix is calculated, it is used to identify the nearest next bin to visit, starting from the bin with the highest fill level. This approach ensures that bins most in need of service are prioritized, while also minimizing the travel distance between stops.



- **Sorting Locations**: We begin by sorting the locations based on their fill levels in descending order. This is achieved by using `np.argsort` on the negative values of the `status_current_fill_level` column in `new_DF`. The result, `priority_indices`, represents the indices of locations sorted by their priority, with higher fill levels first.

```
# Sort locations by descending fill level (prioritize higher fill levels)
priority_indices = np.argsort(-new_DF['status_current_fill_level'])
```

- **Initializing the Route**: The route list is initialized with the starting point, chosen as the first location in our prioritized list. This represents the location with the highest fill level.



- **Initializing the Route**: The route list is initialized with the starting point, chosen as the first location in our prioritized list. This represents the location with the highest fill level.

```
# Initialize route list with the starting point (assuming first location as start)
route = [priority_indices[0]]
```

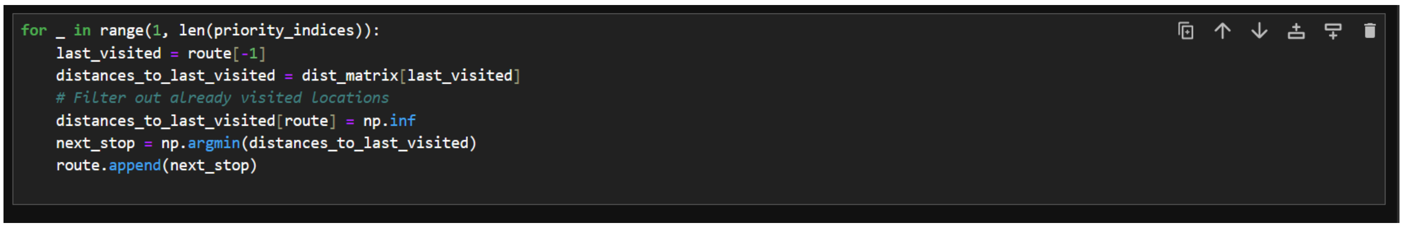
- **Determine Next Stop:**To determine the next stop, we follow a simple heuristic: from the current location, move to the nearest unvisited location. For each subsequent location:

  1. Calculate the distances from the last visited location to all others using the precomputed distance matrix, **dist_matrix.**

  2. Temporarily set the distance to already visited locations to infinity to exclude them from consideration.

  3. Identify the next stop as the location with the minimum distance to the last visited location. This location is then added to the route.

- This methodical approach to route optimization, prioritizing bins by their necessity for service (based on fill levels) and sequentially choosing the next nearest location, significantly enhances efficiency. It not only ensures the most critical bins are attended to promptly but also minimizes the operational costs associated with travel distances.



```python
for _ in range(1, len(priority_indices)):
    last_visited = route[-1]
    distances_to_last_visited = dist_matrix[last_visited]
    # Filter out already visited locations
    distances_to_last_visited[route] = np.inf
    next_stop = np.argmin(distances_to_last_visited)
    route.append(next_stop)
```

Benefits

- **Efficiency:** Significantly reduces the computational complexity compared to calculating distances on-the-fly during optimization.
- **Scalability:** Easily scales with increasing numbers of bins as only the matrix dimensions increase.
- **Accuracy:** Provides a reliable and consistent basis for route planning decisions.

Conclusion

The distance matrix is a powerful tool in the arsenal of route optimization for waste management. By pre-calculating distances between all relevant points, it allows for rapid, dynamic adjustments to routes based on real-time data on bin fill levels. This methodology not only improves the efficiency of waste collection but also contributes to environmental sustainability by reducing unnecessary fuel consumption.

Author

Sachitha 2024.v1