

# 

# **SC4020 Project 2**

# **Technical Review Report**

| **Name** | **Matriculation No.** | **Contribution** |
| --- | --- | --- |
| Lee Ying Ying, Pamela | U2121263A | 25% |
| Pearlina Tan Qinlin | U2221690F | 25% |
| Wong Yi Xian | U2121401C | 25% |
| Gwee Jia Xiang | U2122287H | 25% |

# **Analysis of Co-occurrence Patterns of Points of Interest (POI)**

The goal of this project is to analyse co-occurrence patterns of different Points of Interest (POI) within each city. Specifically, the task aims to identify types of POIs that frequently appear together within the same grid cell using the Apriori algorithm.

The **Apriori algorithm** implemented in mlxtend is specifically used for extracting frequent itemsets. Some key modules used were as follows:

* TransactionEncoder: Converts categorical data into a binary matrix.
* Apriori(): Extracts frequent itemsets based on a minimum support threshold.
* Association\_rules(): Derives association rules from frequent itemsets.

## 1.1 Data Preparation (we use city\_A as example)

The dataset is grouped by spatial coordinates (x,y) to aggregate all POI categories within the same grid cell (Figure 1). Each grid cell is treated as a "basket," and the POI categories within the same grid form a transaction.

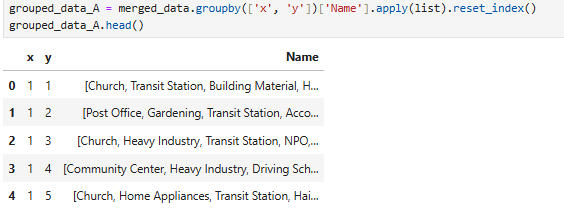


Figure 1. Code for data preparation

## 1.2 One-Hot Encoding

Convert the list of POI categories in each grid into a binary matrix. This is required because the Apriori Algorithm operates on binary data.

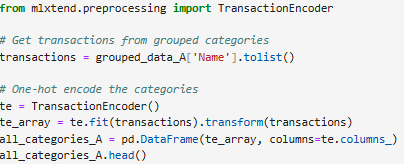


Figure 2. Code for one-hot encoding

## 1.3 Apriori Algorithm

The Apriori algorithm, implemented using the mlxtend package, is used to extract frequent itemsets. The algorithm iteratively generates candidate itemsets and prunes those that do not meet the minimum support threshold.

**Frequent Itemset Mining**

The Apriori algorithm (Figure 3) was applied to the one-hot encoded dataset to identify frequent itemsets with a minimum support threshold of 0.25. Frequent itemsets represent combinations of POIs that co-occur within a significant proportion of grid cells.

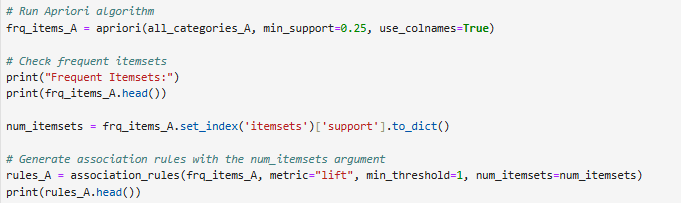


Figure 3. Code for Apriori Algorithm

**Association Rule Mining**

Using the frequent itemsets, association rules were generated with a minimum lift threshold of 1. Rules identify relationships between POI categories, highlighting which categories are likely to appear together.



Figure 4. Code for Association Rule Mining

## 1.4 Results

The results are sorted as seen in Figure 5.



Figure 5. Code for Top Lift Rules

Table 1 displays the top lift rules for each city, with the following columns:

* Antecedents: set of categories that frequently appear together.
* Consequents: POI category that is likely to occur given the antecedent
* Support: frequency of the entire rule appearing in the dataset
* Confidence: probability of consequent given antecedent
* Lift : measure of the strength of the association between antecedents and consequents in a rule.

| City | Antecedents | Consequents | support | confidence | lift |
| --- | --- | --- | --- | --- | --- |
| A | (Real Estate) | (Hair Salon) | 0.266951 | 0.716303 | 1.911089 |
| B | (Real Estate, Bank, Hair Salon) | (Building Material, Laundry, Hospital) | 0.050088 | 0.688253 | 10.047393 |
| C | (Japanese restaurant, Bank, Elderly Care Home) | (Convenience Store, Hair Salon, Hospital, Real Estate) | 0.100584 | 0.753456 | 4.850468 |
| D | (Hospital) | (Hair Salon) | 0.105934 | 0.735777 | 3.195543 |

Table 1. Top lift rules for each city

For city A, there seem to be strong associations involving categories like {Real Estate} and {Hair Salon} which suggests that residential or commercial areas often include personal care services, indicating mixed-use urban planning.

City B shows a very strong relationship between business-oriented areas and complementary services like hospitals and laundries.

For City C, we can observe high confidence and lift which indicates that such clusters are designed to meet the needs of both residential and commercial zones.

City D highlights the clustering of healthcare facilities with personal care services, suggesting a pattern of complementary develop

# **Mining Sequential Patterns**

## 2.1 Data Preparation

For all cities, we filtered and dropped rows with invalid coordinates that exceed the 500m by 500m grid. We also dropped rows after 30 days as seen in Figure 6.

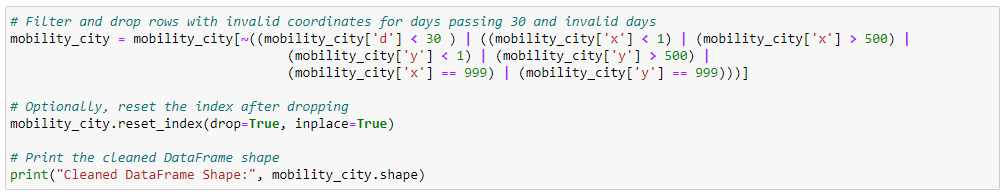


Figure 7. Code for Data Preparation

Next, we converted the mobility dataframe into a GeoDataFrame in order to match trackintel requirements. This included renaming columns such as date, time, as well as converting the x, y coordinates into Points. This allows us to create positionfixes pfs\_city as seen in Figure 7.

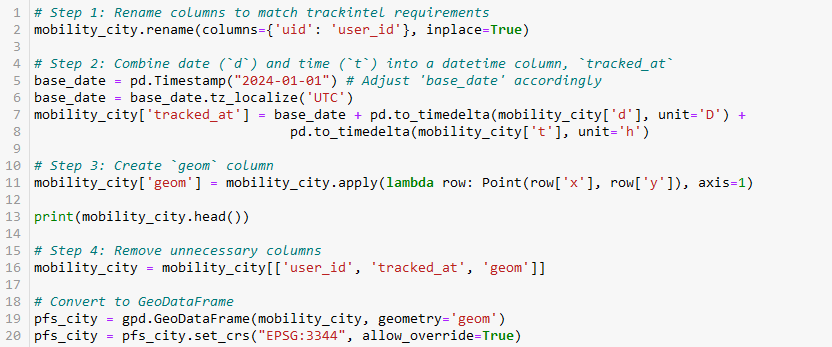


Figure 7. Code for Positionfixes Generation

## 2.2 Generating Staypoints

Using the code below (Figure 8) , staypoints sp\_city are generated using the positionfixes that were previously generated. The spread of staypoints generated across cities A, B, C and D can be seen in Table 2.

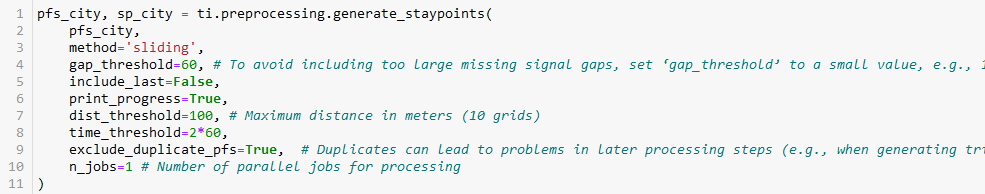


Figure 8. Code for Staypoints Generation

| CityA | CityB | CityC | CityD |
| --- | --- | --- | --- |
|  |  |  |  |

Table 2. Staypoints across the different cities

## 2.3 Generating Triplegs

Triplegs are then generated from the staypoints in Figure 9.

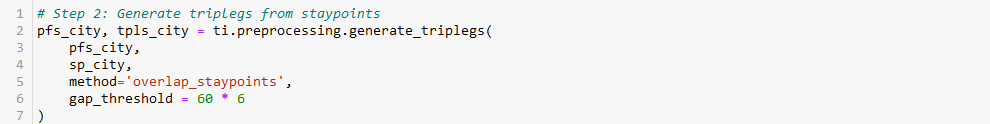


Figure 9. Code for Triplegs Generation

Table 3 shows the final data that we worked with for each city. We reduced the number of days for City A due to memory constraints.

| City | Num\_of\_Days | Num\_of\_Staypoints | Num\_of\_Triplegs | MinSup Value |
| --- | --- | --- | --- | --- |
| CityA | 3 | 9973 | 274,832 | 0.01 |
| CityB | 31 | 31408 | 90650 | 0.01 |
| CityC | 30 | 25003 | 72035 | 0.01 |
| CityD | 32 | 9344 | 21778 | 0.01 |

Table 3. Triplegs and Staypoints for GSP

## 2.4 Generalised Sequential Patterns (GSP)

The triplegs generated are then converted into a list of sequences in Figure 10. We used the gsppy python package to implement the GSP algorithm to mine sequential patterns from the list of sequences (Figure 11). The results are then saved into a csv file (Appendix A).

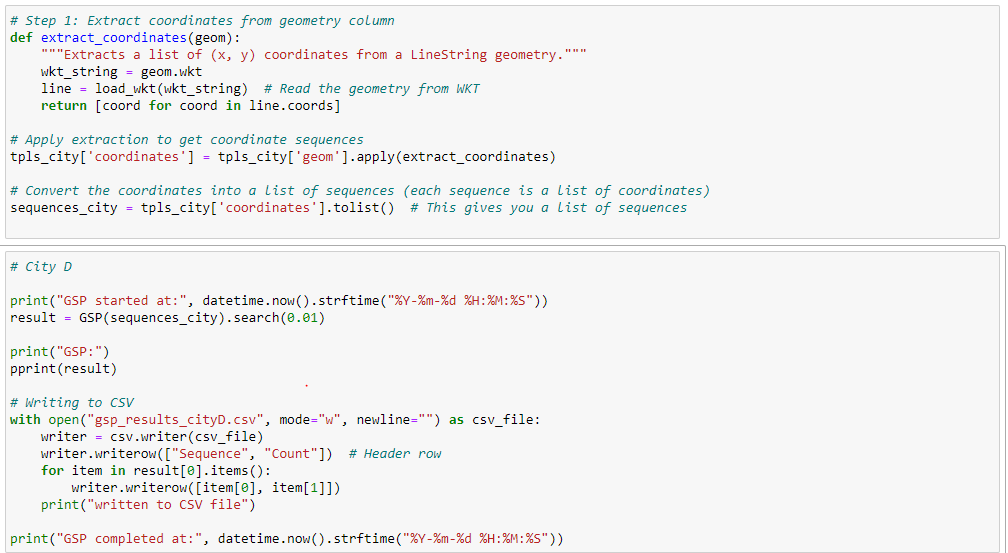


Figure 10. Code for Converting Triplegs into Sequences

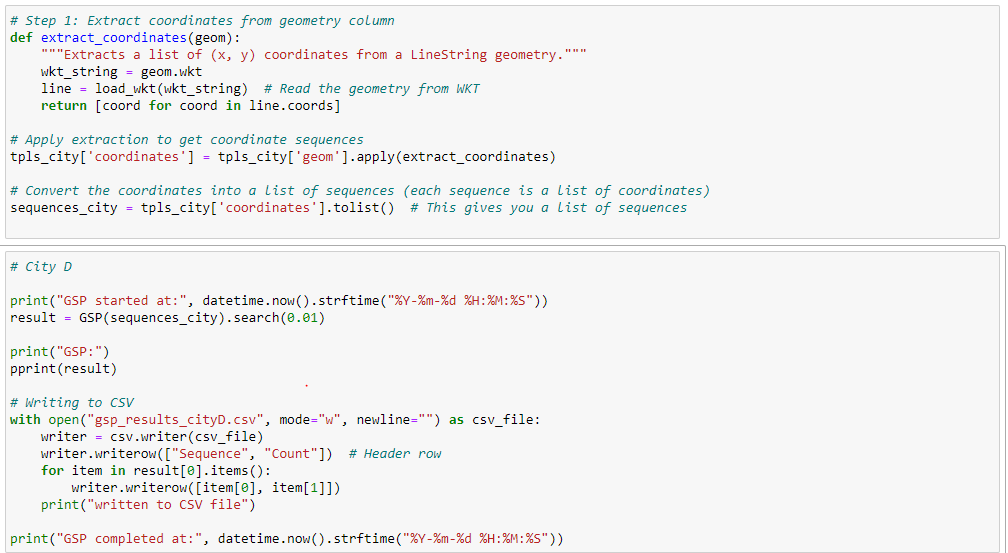


Figure 11. Code for Generalised Sequential Pattern Generation

**gsppy Package Implementation**

The GSP algorithm operates as seen below.

| GSP(frequent\_patterns, k):  candidates ← []  For each pattern1 in frequent\_patterns:  For each pattern2 in frequent\_patterns:  If the first k-1 items of pattern1 == last k-1 items of pattern2:  candidate ← Concatenate(pattern1, pattern2)  candidates ← candidates ∪ {candidate}  Return candidates |
| --- |

In each iteration, the algorithm takes the frequent sequences discovered in the previous step and attempts to combine them to form new candidate sequences. For example, given two frequent sequences of length 𝑘 − 1, such as ['Bread', 'Milk'] and ['Milk', 'Diaper'], the algorithm will combine them to form a new candidate sequence ['Bread', 'Milk', 'Diaper'].

The candidates are generated by combining subsequences that share common elements.

After generating the candidate sequences, the algorithm filters out those that are unlikely to be frequent. If a candidate sequence does not meet the minimum support threshold, it is discarded early to reduce unnecessary computation in later iterations.

# Open Advanced Task

**Problem Statement**: Predicting the user’s next location

**Dataset**: Trip Legs generated in task 2

**Solution**: Use the Triplegs generated in task 2 as input into an LSTM model to predict a user's next location.

## 3.1 Data Preparation

Step 1: Convert WKT String into *linestring* objects to extract sequences of coordinate tuples.

Step 2: Pad Sequences into the same length to give us a fixed length of 1471 sequences long.

Step 3: Split the dataset into train (70%), validation (15%) and test (15%) datasets.

## 3.2 Defining LSTM Model Implementation

| Model Parameters | Value |
| --- | --- |
| Input Size | 2 |
| Hidden Size | 64 |
| Output Size | 2 |
| Epochs | 50 |
| Optimizer | Adam |
| Loss Function | MSELoss |

Table 4: Parameters used for model training

**Input Layer**:

* the model receives sequences of coordinates tuples in the shape of (batch\_size, sequence\_length, input\_size)
* Batch Size is the number of sequences in each batch
* Sequence Length is the length of each sequence including padding
* Input size is the (x,y) coordinate

**LSTM Layer**

* Input Layer: Receives the (batch\_size, sequence\_length, input\_size) coordinates
* Hidden Size: Hidden size of 64 neurons
* Output Layer: Final hidden state sequences for all time steps with a dimensionality of 64

**Fully-Connected Layer**

* Input Layer: Hidden State representation of batch sequences
* Linear Layer: Linear Layers with 128 neurons
* Activation Function: ReLU Activation Function
* Output Layer: Predicted coordinates for the next point.

## 3.3 Training the Model

The Adam Optimiser and MSE loss function are used to train the model. After each epoch, the validation loss is calculated to check how well the model has generalised towards unseen data. After calculating the validation loss for each epoch, the model weights that produce the lowest validation loss score are saved and loaded again for inference on the test dataset. The

## 3.4 Inference Results on Test Dataset

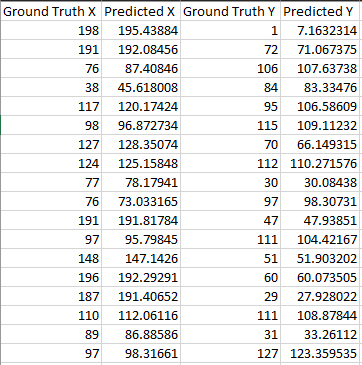


Figure 12: Model Prediction Results

Figure 12 shows a portion of the prediction (x,y) coordinates with the resulting MSE Loss on the test dataset at 83.6137. In general, the lower the MSE score, the better the model is at prediction.