NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Group Members	Matriculation Number	Contribution
Lim Kuan Hwee Sean	U2121484D	Task 3
WONG ZI JUN	U2221712A	Task 2
Bhupathiraju Mihir Varma	U2223870K	Task 1
Batra Parth	U2222378E	Task 3

Task 1 - Analysis of co-occurrence of POI

Overview

The primary objective of this task is to conduct a thorough analysis of the co-occurrence patterns of Points of Interest (POI) that exist within the individual grid cells of a city. To achieve this, we utilise the Apriori algorithm, which is a well-established data mining technique that helps in identifying frequent itemsets. This method specifically focuses on uncovering the various combinations of POI categories that tend to appear together within the same grid cell. Understanding these patterns is essential for effective urban planning and strategic resource allocation, as it provides valuable insights into how different types of POIs are spatially related and interact with one another within the urban environment.

Analysis

- **Grid Cell Formation:** The code generates a unique “grid_id” by merging the x and y coordinates of every POI. This categorises POIs into separate spatial units, making sure that all POIs located within the same physical grid are handled collectively. This phase is essential for structuring the data into "baskets" for further analysis.

- **Basket formation:** In the Basket Formation process, Points of Interest (POIs) within each grid cell are organised by grid_id. The categories of these POIs are compiled into lists, forming baskets of POIs. To eliminate redundancy, duplicate categories within a basket are removed by using set().

- **One-Hot Encoding:** In order to use the Apriori algorithm, categorical data is transformed into a binary matrix through one-hot encoding. Each column reflects a category of Point of Interest (POI), while the rows denote the grid cells. A value of 1 in a cell signifies the existence of a particular POI category within that grid cell. Next we created a dataframe for grid baskets and merged it with one-hot encoding.

- **Applying the Apriori algorithm:** This allows the Apriori algorithm to identify frequent combinations of POI categories that appear together within the same grid cell. The apriori function is applied to identify frequent itemsets based on a minimum support threshold (0.25 for city A and C & 1 for city B and D), which is in line with the analysis of frequent itemsets. Next we displayed the top association rules sorted by confidence and lift in descending order with a minimum lift of 1.

- **Verification of support counts:** We calculated the support counts by checking if the items are present in each grid cell from the itemset in order to check the support values calculated by the Apriori algorithm and displayed the updated frequent itemsets with the support.

- **Sorting with Lift:** Lift is an essential metric in association rules. It evaluates the strength of the link between the antecedents and consequents: A lift value greater than 1 signifies a strong positive association. Organising by lift ensures that the most influential rules are shown at the forefront. The leading association rules (those with the highest lift values) are presented for evaluation. These rules are the most critical in detecting patterns of POI co-occurrence.

Refer to the appendix for the observations from task 1 like “The top association rules”, “Frequent itemsets with manually verified support” and “Top rules with lift values” for all cities A,B,C&D.

Task 2 - Mining Sequential Patterns

Overview

The objective of task 2 is to analyse the movement sequences of residents in each city to uncover common patterns of mobility through sequential pattern mining. We will be using 4 data sets, possibly related to the cities Kotae, Sapporo, Hiroshima and Kumamoto. The TrackIntel library is used to generate staypoints and triplegs, and Generalised Sequential Pattern(GSP) algorithm is used to mine sequential patterns from the triplegs.

Data Preprocessing

Data Transformation

We will be using GeoDataFrame as input data type for the TrackIntel library.

‘tracked_at’ column in the format “YYYY-MM-DD HH:MM:SS”. From the dataset, per unit of d is a day, per unit of t is 30 minutes. We set 2024-01-01 as a reference start date, as the dataset does not possess exact dates.

“geometry” column in the format POINT(x,y), where x and y represent the horizontal and vertical axes of the grid, respectively. Each grid unit corresponds to a distance of 500 metres. It is converted into latitude and longitude **as only haversine calculation is possible in TrackIntel** generate staypoints. To convert the grid coordinates into geographic latitude and longitude, the following assumptions and steps are applied:

1. **Linear Conversion:**
    - **One** degree of latitude and longitude is approximated as equivalent to **111,320 metres**.
    - The latitude and longitude values are calculated based on this linear approximation.
2. **Reference Point(Additional Feature - Not used, set to 0,0):**
    - The conversion is performed relative to a reference latitude and longitude obtained from the **country** dataset, and added to grid values to generate the final latitude and longitude values.
3. **GeoDataFrame Creation:**
    - The geometry column is updated with POINT objects constructed using the computed longitude and latitude.

Limit to 30 Days

There is a total of up to 75 days of data tracked per individual. As this is too computationally intensive, we only mine up to day 30 to reduce data points while still having sufficient data to observe meaningful patterns.

Get Valid Sequences

We remove short sequences with insufficient data points, minimum 3, for pattern analysis in a rolling window of 7 days. This ensures that the dataset is cleaner and better suited to identify meaningful patterns .

TrackIntel

Generate PositionFixies(pfs)

We pass the processed GeoDataFrame into ti.io.read_positionfixes_gpd to generate pfs.

Generate Staypoints

We pass the generated pfs GeoDataFrame into the generate_staypoints method. Notable parameters:

1. **Dist_threshold:**
    - 1000m, max distance between pfs to be considered a staypoint
2. **Time_threshold:**
    - 60 min within dist_threshold to be considered staypoint
3. **Gap_threshold:**
    - 240 min. Assume user left and re-entered area if no updates for 240 min but still within dist_threshold

Generate Triplegs

We pass the pfs and staypoints into preprocessing.generate_triplegs to generate triplegs.

Data Analysis

Processing Triplegs

We floor triplegs coordinates to the nearest even number- e.g. (133,135) to (132,134) due to lack of computational power to mine meaningful patterns at lower support levels. This increases the frequency of events, allowing for more subsequences to meet min_sup threshold, thus, enabling us to visualise patterns. However, the patterns observed are more generalised and less precise as a result.

GSP Algorithm

Reference: https://github.com/jacksonpradolima/gsp-py/blob/master/gsppy/gsp.py

The **CUSTOMGSP** class implements the Generalized Sequential Pattern (GSP) mining algorithm to identify frequent subsequences in transactional datasets. The code can be found in customgsp.py. We use a minsup value of 0.0001. The algorithm is broken down to as follows:

1. **Preprocessing: Converts raw transactions into standardized format and retrieves unique candidates(len 1)**
    - Optimisations
        i. Represented transactions as tuples to enable quick lookups
2. **Subsequence Matching: Check if a candidate sequence is a subsequence within a transaction**

- Optimisations
    i. Used an iterator, allowing for sequential access to the elements in transaction, thus only scanning through the transaction once

3. **Frequency Calculation: Calculate support of elements**
    - Optimisations
        i. Pruning. Filtered those below minsup early to avoid unnecessary candidate generation

4. **Candidate Generation: Generate potential frequent candidate sequences(similar to apriori candidate generation)**
    - Optimisations
        i. Slicing to check overlap
        ii. Combining only overlapping patterns, reducing possible combinations generated

5. **Search(GSP): Implements GSP algorithm**
    - Explanation
        i. Begin with length 1 candidates
        ii. Evaluate sequences(subsequence matching, frequency calculation)
        iii. Iteratively generates longer sequences if meets minsup, append to list
        iv. Repeat (ii) and (iii)
        v. Stop when no more candidate in list OR when subsequence length greater than maximum transaction size
    - Optimisations
        i. Sequential filtering to ensure only frequent patterns are passed to next iteration

Sequential Mining Results: The results are stored in csv files. Only subsequences greater than length 2 are saved.

Kotae_freq_subseq.csv - City A
Hiroshima_freq_subseq.csv - City B
Sapporo_freq_subseq.csv - City C
Kumamoto_freq_subseq.csv - City D

Task 3 - Predicting User Movements with LSTM and POI Data

Overview

We predicted a user's next location within a city by integrating mobility data, Points of Interest (POIs), and sequential movement patterns to generate meaningful predictions. This solution combines data preprocessing, feature engineering, and LSTM models to capture temporal and spatial dependencies, enriched with real-world POI mapping for contextual relevance. The workflow enables actionable insights for applications such as navigation, urban planning, and personalised recommendations, bridging technical rigour with practical utility.

Data Pre-processing

1. **Loading Data**
   We work with three critical datasets:
- The **mobility data**, from kumamoto_challengedata.csv, contains user locations in grid coordinates (x, y) along with timestamps. This data captures user movements over time.
- The **POI distribution data**, from POIdata_cityD.csv, details Points of Interest within City D. It includes fields like x, y, category_id, and the count of POIs within a grid.
- The **POI category mappings**, from POI_datacategories.csv, link POI categories—such as parks or cafés—with numerical IDs for easy integration.

Together, these datasets provide a comprehensive view of user movements and the city's structure.

2. **Loading Task Outputs**
   To enhance predictions, we incorporate results from previous analyses:
- **Task 1 Output**, from frequent_itemsets_cityD.csv, identifies co-occurring POIs within the same grid. This data helps analyse user behaviour in specific areas.
- **Task 2 Output**, from frequent_sequences_cityD.csv, captures sequential movement patterns as tuples of coordinates. These patterns are converted into Python objects like lists or tuples for seamless integration.

These outputs help us improve mobility data by linking movement sequences with meaningful patterns and nearby POIs.

Augmentation

The main goal here is to augment mobility data by combining it with POI features and movement patterns. Specifically:

- We link each mobility record with nearby POIs based on grid coordinates.
- We enrich user movement records with frequent itemsets from Task 1 and sequential patterns from Task 2.
  This leads to feature engineering, where we generate features like:
  1. Nearby POI categories.
  2. Presence of frequent POI itemsets (from task 1 output).
  3. Matches frequent movement patterns (from task 2 output).

Model Architecture

The LSTM model is designed to process sequential data. Its structure includes:

- An **input layer** for sequential features.
- An **LSTM layer** that outputs a 128-dimensional representation, capturing temporal dependencies.
- Two dense layers:
  1. A 64-neuron layer.
  2. A 32-neuron layer with ReLU activations for non-linearity.
- An **output layer** that predicts two values, the x and y coordinates, using a linear activation function.

```python
# Define the model
model = Sequential([
    LSTM(128, input_shape=(X_train.shape[1], X_train.shape[2]), return_sequences=False),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(2, activation='linear')  # Predict (x, y) coordinates
])
```

Figure.

The loss function is Mean Squared Error, ideal for regression tasks, while the Adam optimizer ensures efficient learning.

Model: "sequential_4"

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 128)	68,608
dense_12 (Dense)	(None, 64)	8,256
dense_13 (Dense)	(None, 32)	2,080
dense_14 (Dense)	(None, 2)	66

Total params: 79,010 (308.63 KB)

Trainable params: 79,010 (308.63 KB)

Non-trainable params: 0 (0.00 B)

Figure.

The model summary reveals a lightweight architecture with **79,010** trainable parameters, making it efficient for both training and inference.

Model Training

The LSTM model was trained to predict the next (x, y) location based on sequential mobility data. The dataset was split into training (80%), validation (10%), and test (10%) sets.

Training Procedure

- Loss Function: Mean Squared Error (MSE) to minimise prediction error.
- Optimizer: Adam, for adaptive learning rates.
- Metrics: Mean Absolute Error (MAE) to track accuracy.
- Batch Size: 3000 sequences per batch.
- Early Stopping: Stopped training if validation loss did not improve for 10 consecutive epochs, restoring the best model weights.

Result

The models achieved consistent performance across all four cities:

- **Training Loss:** ~43.7 (MSE).
- **Training MAE**: ~3.4.
- **Validation Los**s: ~43.8 (MSE).
- **Validation MAE**: ~3.5.

We trained the model to obtain the weights, one for each city. All four models yielded the similar MAE scores of 3.4 -3.5 which means that the predicted location is, on average, 3 to 4 grid units away from the true location. Early stopping is also implemented to reduce overfitting and efficient training.

Application

The application focuses on next location prediction. Based on the user's input, the code loads the pretrained LSTM model and predicts based on frequent POI movements the next location for the user to go to.

```
# Example input
city = 'D'
current_location = [10, 15]

# Run prediction
result = predict_next_location(city, current_location)

# Display the result
print("Predicted Results:")
print(f"Predicted Next Location (x, y): {result['predicted_location']}")
print(f"Nearest POI Location (x, y): {result['nearest_poi_location']}")
print(f"POI: {result['poi']}")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 112ms/step
Predicted Results:
Predicted Next Location (x, y): [53, 46]
Nearest POI Location (x, y): [54, 46]
POI: Hair Salon
```

Appendix

Task 1

Figures for City A:

```
Top Association Rules:
             antecedents          consequents  antecedent support  \
19            (Laundry )     (Heavy Industry)            0.318624
10         (Real Estate)     (Heavy Industry)            0.372679
27  (Accountant Office)     (Heavy Industry)            0.332324
23          (Hair Salon)     (Heavy Industry)            0.374814
8          (Real Estate)  (Building Material)            0.372679

    consequent support   support  confidence      lift  leverage  conviction  \
19            0.559069  0.251266    0.788596  1.410553  0.073133    2.085731
10            0.559069  0.292316    0.784363  1.402982  0.083963    2.044788
27            0.559069  0.258414    0.777595  1.390876  0.072622    1.982562
23            0.559069  0.286062    0.763210  1.365145  0.076515    1.862121
8             0.453291  0.277921    0.745738  1.645164  0.108989    2.150179

    zhangs_metric
19       0.427163
10       0.457872
27       0.420906
23       0.427836
8        0.625131
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/r
  warnings.warn(
```

```
Frequent Itemsets with Manually Verified Support:
    support             itemsets  check_support
0  0.559069     (Heavy Industry)       0.559069
1  0.458255    (Home Appliances)       0.458255
2  0.453291  (Building Material)       0.453291
3  0.428522             (Church)       0.428522
4  0.406979    (Transit Station)       0.406979
```

```
Top Rules with Lift Values:
                                   rule      lift
0             Real Estate -> Hair Salon  1.911089
1             Hair Salon -> Real Estate  1.911089
2      Real Estate -> Building Material  1.645164
3      Building Material -> Real Estate  1.645164
4       Building Material -> Hair Salon  1.601028
```

Figures for City B:

```
Top Association Rules:
                           antecedents          consequents  \
100    (Building Material, Hair Salon)    (Transit Station)
17                              (Bank)    (Transit Station)
12                          (Laundry )    (Transit Station)
36                       (Real Estate)  (Building Material)
112  (Home Appliances, Heavy Industry)  (Building Material)

     antecedent support  consequent support   support  confidence      lift  \
100            0.123520            0.359711  0.100395    0.812777  2.259531
17             0.133056            0.359711  0.105984    0.796540  2.214392
12             0.162210            0.359711  0.121219    0.747297  2.077496
36             0.163744            0.270934  0.121328    0.740964  2.734852
112            0.145989            0.270934  0.107628    0.737237  2.721097

     leverage  conviction  zhangs_metric
100  0.055963    3.419934       0.635988
17   0.058123    3.147009       0.632577
12   0.062870    2.533765       0.619070
36   0.076965    2.814535       0.758559
112  0.068075    2.774618       0.740624
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages
  warnings.warn(
```

```
Frequent Itemsets with Manually Verified Support:
    support             itemsets  check_support
0  0.375055             (Church)       0.375055
1  0.359711    (Transit Station)       0.359711
2  0.303924     (Heavy Industry)       0.303924
3  0.270934  (Building Material)       0.270934
4  0.269838    (Home Appliances)       0.269838
```

```
Top Rules with Lift Values:
                                         rule      lift
0                     Hair Salon -> Laundry    3.490690
1                     Laundry  -> Hair Salon   3.490690
2                  Hair Salon -> Real Estate   3.380272
3                  Real Estate -> Hair Salon   3.380272
4  Hair Salon -> Transit Station, Building Material   3.261651
```

Figures for City C:

```
Top Association Rules:
                          antecedents        consequents  antecedent support  \
182   (Building Material, Hair Salon)  (Transit Station)            0.271301
158         (Real Estate, Hair Salon)  (Transit Station)            0.285451
165  (Real Estate, Accountant Office)  (Transit Station)            0.278376
146  (Real Estate, Elderly Care Home)  (Transit Station)            0.272224
20                         (Hospital)  (Transit Station)            0.301446

     consequent support   support  confidence      lift  leverage  conviction  \
182            0.571516  0.255306    0.941043  1.646572  0.100253    7.267740
158            0.571516  0.267302    0.936422  1.638487  0.104163    6.739538
165            0.571516  0.258382    0.928177  1.624060  0.099286    5.965809
146            0.571516  0.252230    0.926554  1.621220  0.096650    5.833968
20             0.571516  0.278683    0.924490  1.617608  0.106402    5.674512

     zhangs_metric
182       0.538875
158       0.545352
165       0.532492
146       0.526509
20        0.546562
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/m
  warnings.warn(
```

```
Frequent Itemsets with Manually Verified Support:
    support            itemsets  check_support
0  0.571516   (Transit Station)       0.571516
1  0.482621              (Park)       0.482621
2  0.472778 (Building Material)       0.472778
3  0.434943   (Home Appliances)       0.434943
4  0.426946    (Heavy Industry)       0.426946
```

```
Top Rules with Lift Values:
                                       rule      lift
0  Hair Salon -> Transit Station, Real Estate  2.242471
1  Transit Station, Real Estate -> Hair Salon  2.242471
2              Hair Salon -> Park, Real Estate  2.215365
3              Park, Real Estate -> Hair Salon  2.215365
4              Real Estate -> Park, Hair Salon  2.187702
```

Figures for City D:

```
Top Association Rules:
                        antecedents          consequents  antecedent support  \
34                       (Hospital)         (Hair Salon)            0.143975
10                       (Hospital)    (Transit Station)            0.143975
138   (Heavy Industry, Hair Salon)  (Building Material)            0.139880
120  (Hair Salon, Transit Station)  (Building Material)            0.142064
52                    (Real Estate)  (Building Material)            0.174827

     consequent support   support  confidence      lift  leverage  conviction  \
34             0.230251  0.105934    0.735777  3.195543  0.072783    2.913260
10             0.308063  0.105752    0.734513  2.384293  0.061398    2.606295
138            0.322625  0.102111    0.729993  2.262671  0.056983    2.508737
120            0.322625  0.101656    0.715567  2.217955  0.055823    2.381493
52             0.322625  0.125046    0.715252  2.216980  0.068642    2.378863

     zhangs_metric
34        0.802622
10        0.678238
138       0.648798
120       0.640064
52        0.665237
/Library/Frameworks/Python.framework/Versions/3.10/lib/python3.10/site-packages/m
  warnings.warn(
```

```
Frequent Itemsets with Manually Verified Support:
    support            itemsets  check_support
0  0.360939            (Church)       0.360939
1  0.334274    (Heavy Industry)       0.334274
2  0.331361   (Home Appliances)       0.331361
3  0.322625 (Building Material)       0.322625
4  0.308063   (Transit Station)       0.308063
```

```
Top Rules with Lift Values:
                                               rule      lift
0                          Hospital -> Hair Salon  3.195543
1                          Hair Salon -> Hospital  3.195543
2                         Laundry  -> Real Estate  2.986421
3                         Real Estate -> Laundry   2.986421
4  Transit Station, Building Material -> Hair Salon  2.860391
```