National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Valentina Bernal Gomez
Student ID: 23395745

School of Computing
National College of Ireland

Supervisor: Furqan Rustam

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Valentina Bernal Gomez
Student ID:	23395745
Programme:	Data Analytics **Year:** 2025
Module:	MSc Research Project
Lecturer:	Furqan Rustam
Submission Due Date:	11 August 2025
Project Title:	Enhancing Dublin's Bike-Sharing Network: Leveraging Deep Learning for Growth and Efficiency
Word Count:	735 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Valentina Bernal Gomez

Date: 9 August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Valentina Bernal
Student ID: 23395745

1 Introduction

This configuration manual provides the necessary information to replicate the experimental setup and results of the research project "Enhancing Dublin's bike-sharing network: Leveraging deep learning for growth and efficiency". The research includes clustering techniques with Graph Neural Networks (GNN) models to identify new potential bike-sharing locations in Dublin. The models are trained using a enriched dataset which was combine with historical station usage, demographic data and commuting modes.

2 System specification

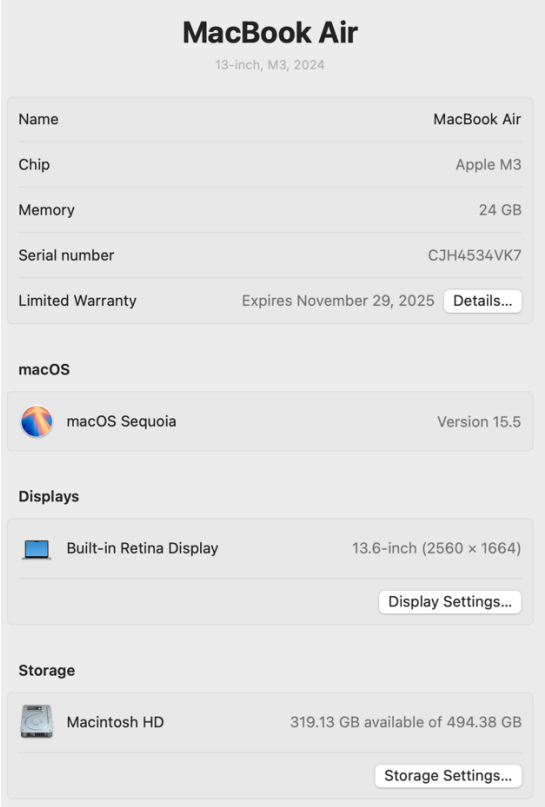

Figure 1: System Specification

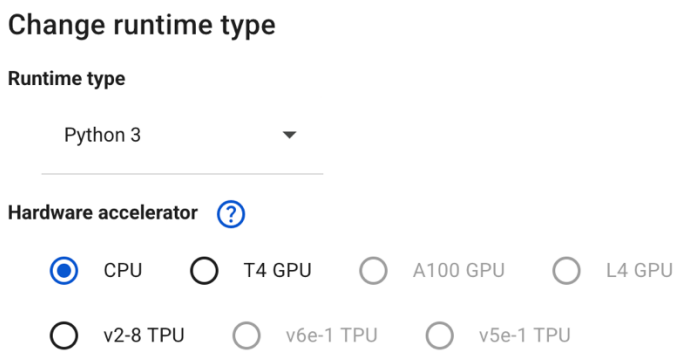

Figure 2. Google Collab configuration

The system is outfitted with the Apple M3 processor, with 24 GB of RAM, ensuring a efficiently processing operations. The device's unique identifier is CJH4532VK7 which serves as a specific marker for system identification. The system runs on macOS Sequoia, version 15.5 which is benefit from the latest feature.
While the MacBook Air was used for development and preliminary data analysis and preprocessing, the implementation of different variations of the model ad intensive tasks were executed in Google Collab, leveraging its cloud-based GPU environment.

3 Software Requirements

Main development of this research took place in Google Collab, which provided a efficient Python environment with GPU acceleration, while the local machine was used for dataset preparation, merge of datasets and notebook management.

Programming Language:

- Python 3.127

Development Environments:

- Jupyter Notebook and Google Collab

Package installation in Google Collab:

The following commands were executed to install required packages:

```
!pip install -q torch-scatter torch-sparse torch-cluster torch-spline-conv torch-geometric -f https://data.pyg.org/whl/torch-2.0.0+cpu.html
```

Figure 3. First package installed

```
!pip install torch torchvision torchaudio
```

Figure 4. Second package installed

```
pip install contextily
```

Figure 5. Third package installed

Libraries used:

```
# General-purpose libraries
import pandas as pd
import geopandas as gpd
import numpy as np
import random
import warnings
warnings.filterwarnings("ignore")

# Data visualization
import matplotlib.pyplot as plt
import folium
from folium.plugins import MarkerCluster
import contextily as ctx

# Geospatial processing
from shapely.geometry import Point
from geopy.distance import geodesic

# Machine Learning and preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import KMeans
from sklearn.metrics import (silhouette_score, accuracy_score, precision_score, recall_score, f1_score)
from sklearn.utils.class_weight import compute_class_weight
from sklearn.model_selection import train_test_split, KFold
from imblearn.under_sampling import RandomUnderSampler

# Sparse and graph utilities
from scipy.sparse import coo_matrix

# PyTorch & PyTorch Geometric
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import LeakyReLU
from torch_geometric.nn import GCNConv, SAGEConv, GATConv
from torch_geometric.utils import (from_scipy_sparse_matrix, dense_to_sparse)
from torch_geometric.data import Data
```

Figure 6. Libraries

4 Implementation and setup

4.1 Local data preparation using Jupyter Notebook

The process included the merge of 2024 monthly bike usage dataset and combined them into a dataset “bike_usage_summary_2024.csv”. Then a spatial join using GeoJSON file, which was merged the “bike_usage_summary_2024.csv” and the GeoJSON file, the result was “bike_usage_summary_with_zones.csv”, finally, the merge with age, commuting and employment datasets at the zone level, the final output was “df_enriched.csv” used for clustering and GNN training.

January and February

```
process_month("January.csv", "bike_summary_january.csv", 1)
process_month("February.csv", "bike_summary_february.csv", 2)
```

```
Processed and saved: bike_summary_january.csv
Processed and saved: bike_summary_february.csv
```

Figure 7. Bike summary January and February

May to December

```
process_month_may_to_dec("May.csv", "bike_summary_may.csv", 5)
process_month_may_to_dec("June.csv", "bike_summary_june.csv", 6)
process_month_may_to_dec("July.csv", "bike_summary_july.csv", 7)
process_month_may_to_dec("August.csv", "bike_summary_august.csv", 8)
process_month_may_to_dec("September.csv", "bike_summary_september.csv", 9)
process_month_may_to_dec("October.csv", "bike_summary_october.csv", 10)
process_month_may_to_dec("November.csv", "bike_summary_november.csv", 11)
process_month_may_to_dec("December.csv", "bike_summary_december.csv", 12)
```

```
Processed and saved: bike_summary_may.csv
Processed and saved: bike_summary_june.csv
Processed and saved: bike_summary_july.csv
Processed and saved: bike_summary_august.csv
Processed and saved: bike_summary_september.csv
Processed and saved: bike_summary_october.csv
Processed and saved: bike_summary_november.csv
Processed and saved: bike_summary_december.csv
```

Figure 8. Bike summary May to December

```
# Fix all random seeds
torch.manual_seed(42)
np.random.seed(42)
random.seed(42)
torch.cuda.manual_seed(42)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

# Load combined bike usage dataset
df = pd.read_csv("bike_usage_summary_2024 (1).csv")

# Load the GeoJSON file with CSO small areas
urban_areas = gpd.read_file("SMALL_AREA_2022_Genralised_20m_view_-6111806341570938324.geojson")

# Create a GeoDataFrame with station points
stations_gdf = gpd.GeoDataFrame(df,geometry=gpd.points_from_xy(df.longitude, df.latitude),crs="EPSG:4326")

# Ensure both use the same CRS
urban_areas = urban_areas.to_crs(stations_gdf.crs)

# Spatial join to assign urban area to each station point
stations_with_zone = gpd.sjoin(stations_gdf, urban_areas, how="left", predicate="within")

# Check columns and save
print(stations_with_zone.columns)
stations_with_zone.to_csv("bike_usage_summary_with_zones.csv", index=False)
```

```
Index(['station_id', 'station_name', 'capacity', 'latitude', 'longitude',
       'month', 'hour', 'avg_available_bikes', 'avg_occupancy_rate',
       'geometry', 'index_right', 'OBJECTID', 'SA_GUID_2016', 'SA_GUID_2022',
       'SA_PUB2011', 'SA_PUB2016', 'SA_PUB2022', 'SA_GEOGID_2022',
       'SA_CHANGE_CODE', 'SA_URBAN_AREA_FLAG', 'SA_URBAN_AREA_NAME',
       'SA_NUTS1', 'SA_NUTS1_NAME', 'SA_NUTS2', 'SA_NUTS2_NAME', 'SA_NUTS3',
       'SA_NUTS3_NAME', 'ED_GUID', 'ED_OFFICIAL', 'ED_ENGLISH', 'ED_GAEILGE',
       'ED_ID_STR', 'ED_PART_COUNT', 'COUNTY_CODE', 'COUNTY_ENGLISH',
       'COUNTY_GAEILGE', 'CSO_LEA'],
      dtype='object')
```

Figure 9. Load bike summary and GeoJSON file

Data cleaning and preprocessing:

Upload dataset with sociodemographic data

```
df_age = pd.read_csv("Small_areas_age.csv")
```

```
df_commuting = pd.read_csv("Small_areas_commuting.csv")
```

```
df_employment = pd.read_csv("Small_areas_occupation.csv")
```

Figure 10. Upload sociodemographic dataset

Clean data

Select key columns

```
key_columns = ['Statistic Label','CSO Small Areas 2022', 'Age', 'UNIT', 'VALUE']

df_age = df_age[key_columns]
```

```
key_columns = ['Statistic Label','CSO Small Areas 2022', 'Means of Travel', 'UNIT', 'VALUE']

df_commuting = df_commuting[key_columns]
```

```
key_columns = ['Statistic Label','CSO Small Areas 2022', 'Sex','Principle Economic Status','UNIT', 'VALUE']

df_employment = df_employment[key_columns]
```

Compute the proportion of each category for normalization

Age dataset

```
# Total population by area
df_age_total = df_age.groupby('CSO Small Areas 2022')['VALUE'].sum().reset_index()
df_age_total.rename(columns={'VALUE': 'Total_Population'}, inplace=True)

# Proportions by age
df_age_prop = df_age.merge(df_age_total, on='CSO Small Areas 2022')
df_age_prop['Proportion'] = df_age_prop['VALUE'] / df_age_prop['Total_Population']
```

Means of transport dataset

```
# Total population who travel by area
df_commuting_total = df_commuting.groupby('CSO Small Areas 2022')['VALUE'].sum().reset_index()
df_commuting_total.rename(columns={'VALUE': 'Total_Travellers'}, inplace=True)

# Proportions by means of transport
df_commuting_prop = df_commuting.merge(df_commuting_total, on='CSO Small Areas 2022')
df_commuting_prop['Proportion'] = df_commuting_prop['VALUE'] / df_commuting_prop['Total_Travellers']
```

Occupation dataset

```
# Total employmnet by area (all employment values by sex and status)
df_employment_total = df_employment.groupby('CSO Small Areas 2022')['VALUE'].sum().reset_index()
df_employment_total.rename(columns={'VALUE': 'Total_Employment'}, inplace=True)

# Proportions by category
df_employment_prop = df_employment.merge(df_employment_total, on='CSO Small Areas 2022')
df_employment_prop['Proportion'] = df_employment_prop['VALUE'] / df_employment_prop['Total_Employment']
```

Figure 11. Data cleaning

Merge four datasets

```
df_enriched = gdf_clusters.merge(df_age_pivot, on='SA_PUB2022', how='left')
df_enriched = df_enriched.merge(df_commuting_pivot, on='SA_PUB2022', how='left')
df_enriched = df_enriched.merge(df_employment_pivot, on='SA_PUB2022', how='left')
```

```
df_enriched.head()
```

Clustering with enriched data

```
# Select features
age_cols = [col for col in df.columns if col.startswith('Age') and col not in ['Age 0-4', 'Age 5-9', 'Age 10-14', 'Age 15-19']]

features_kmeans = [
    'avg_available_bikes', 'avg_occupancy_rate',
    'Bicycle','Bus, minibus or coach', 'Car Driver', 'Motorcycle or scooter',
    'Train, DART or LUAS', 'On Foot',
    'At work', 'Student', 'Work mainly at or from home'
] + age_cols

X = df[features_kmeans].fillna(0)

# Scale
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow & Silhouette to find optimal k
inertia = []
silhouette = []

for k in range(2, 10):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(X_scaled)
    inertia.append(km.inertia_)
    silhouette.append(silhouette_score(X_scaled, km.labels_))

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(range(2, 10), inertia, marker='o')
plt.title("Elbow Method")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
```

Figure 12. Clustering with enriched data

Migrate the clustering to df_enriched

```
df_enriched = df_enriched.merge(df[['station_id', 'cluster']], on='station_id', how='left')
```

```
print(df_enriched.columns)
```

```
Index(['station_id', 'station_name', 'latitude', 'longitude', 'capacity',
       'avg_available_bikes', 'avg_occupancy_rate', 'SA_PUB2022', 'geometry',
       'cluster_x', 'Age 0-4', 'Age 10-14', 'Age 15-19', 'Age 20-24',
       'Age 25-29', 'Age 30-34', 'Age 35-39', 'Age 40-44', 'Age 45-49',
       'Age 5-9', 'Age 50-54', 'Age 55-59', 'Age 60-64', 'Age 65-69',
       'Age 70-74', 'Age 75-79', 'Age 80-84', 'Age 85 and over', 'Total_x',
       'Bicycle', 'Bus, minibus or coach', 'Car Driver', 'Car passenger',
       'Motorcycle or scooter', 'Not stated', 'On Foot', 'Other (incl. lorry)',
       'Total_y', 'Train, DART or LUAS', 'Van', 'Work mainly at or from home',
       'At work', 'Long term unemployed', 'Looking after home/family',
       'Looking for first regular job', 'Other', 'Retired',
       'Short term unemployed', 'Student', 'Total',
       'Unable to work due to permanent sickness or disability', 'cluster_y'],
      dtype='object')
```

Figure 13. Migrate to final dataset

4.2 Model training using Google Collab

The training and evaluation of GNN models were implemented in Google Collab.

Original GCN model

Add label to real stations and generate negative samples using Jittering

```
df_enriched['label'] = 1
```

```
jitter_amount = 0.0005
np.random.seed(42)

jittered_geometries = []
for point in df_enriched.geometry:
    jittered_x = point.x + np.random.uniform(-jitter_amount, jitter_amount)
    jittered_y = point.y + np.random.uniform(-jitter_amount, jitter_amount)
    jittered_geometries.append(Point(jittered_x, jittered_y))

negative_points = gpd.GeoDataFrame(geometry=jittered_geometries, crs="EPSG:4326")
```

Select relevant feature columns for training, excluding identifiers, geometry and labels.

```
feature_cols = [col for col in df_enriched.columns if col not in ['station_id', 'station_name', 'geometry', 'label', 'SA_PUB2022']]
```

Figure 14. Add label to real and synthetic stations

Create DataFrame for negative nodes: We copied the feature values from the nearest real station to each negative sample to preserve contextual consistency.

```
negative_df = pd.DataFrame(0, index=np.arange(len(negative_points)), columns=feature_cols)
negative_df['geometry'] = negative_points.geometry
negative_df['label'] = 0
```

```
# Coordinates from real stations and negatives
stations_coords = df_enriched[['latitude', 'longitude']].values
neg_coords = negative_df['geometry'].apply(lambda point: (point.y, point.x)).to_list()

# Find nearest station
knn = NearestNeighbors(n_neighbors=1)
knn.fit(stations_coords)
distances, indices = knn.kneighbors(neg_coords)

# Copy features from nearest real station
for i, idx in enumerate(indices.flatten()):
    negative_df.iloc[i, :-2] = df_enriched.iloc[idx][feature_cols].values
```

Figure 15. Create dataframe for negative samples

Combine both into a single dataset named df_full

```
df_full = pd.concat([
    df_enriched[feature_cols + ['geometry', 'label']],
    negative_df[feature_cols + ['geometry', 'label']]
], ignore_index=True)

# Check class balance and missing values
print(df_full['label'].value_counts())

print(df_full.isnull().sum().sort_values(ascending=False).head())

df_full.fillna(0, inplace=True)
```

```
label
1    115
0    115
Name: count, dtype: int64
latitude               0
longitude              0
capacity               0
avg_available_bikes    0
avg_occupancy_rate     0
dtype: int64
```

Figure 16. Combine both samples to a final dataset

Create edge_index (graph structure-core for my GCN model)

```
# Extract coordinates for distance calculation
coords = df_full[['latitude', 'longitude']].values

# Fit k-NN
k = 5
knn = NearestNeighbors(n_neighbors=k)
knn.fit(coords)
distances, indices = knn.kneighbors(coords)

# Build edge index (sparse matrix)
rows = []
cols = []

for i in range(len(coords)):
    for j in indices[i]:
        rows.append(i)
        cols.append(j)

# Convert to torch_geometric edge_index
adj_matrix = coo_matrix((np.ones(len(rows)), (rows, cols)), shape=(len(coords), len(coords)))
edge_index, _ = from_scipy_sparse_matrix(adj_matrix)

print(edge_index.shape)
print(edge_index[:, :10])
```

```
torch.Size([2, 1150])
tensor([[  0,   0,   0,   0,   0,   1,   1,   1,   1,   1],
        [  0, 115, 165,  50,   8,   1, 116, 214,  99,  77]])
```

Create the graph data object

```
data = Data(x=x, edge_index=edge_index, y=y)
```

Define the train/test split

```
num_nodes = data.num_nodes
num_train = int(num_nodes * 0.8)

perm = torch.randperm(num_nodes)
train_mask = torch.zeros(num_nodes, dtype=torch.bool)
train_mask[perm[:num_train]] = True
test_mask = ~train_mask

data.train_mask = train_mask
data.test_mask = test_mask
```

Figure 17. Built graph structure

Define, initializes and train the GCN: We defined and trained a 2-layer GCN model with dropout and class weights to address class imbalance.

```python
class GCN(torch.nn.Module):
    def __init__(self, in_channels, hidden_channels, out_channels):
        super().__init__()
        self.conv1 = GCNConv(in_channels, hidden_channels)
        self.dropout = nn.Dropout(p=0.3)
        self.conv2 = GCNConv(hidden_channels, out_channels)

    def forward(self, x, edge_index):
        x = self.conv1(x, edge_index)
        x = F.relu(x)
        x = self.dropout(x)
        x = self.conv2(x, edge_index)
        return x

gcn_model = GCN(data.num_node_features, 16, 2)
optimizer = torch.optim.Adam(gcn_model.parameters(), lr=0.01, weight_decay=5e-4)

# Compute class weights
weights = torch.tensor([0.4,0.6], dtype=torch.float)
```

```python
# Training
gcn_model.train()
for epoch in range(201):
    optimizer.zero_grad()
    out = gcn_model(data.x, data.edge_index)
    loss = F.cross_entropy(out[data.train_mask], data.y[data.train_mask], weight=weights)
    loss.backward()
    optimizer.step()

    if epoch % 20 == 0:
        pred = out[data.train_mask].argmax(dim=1)
        acc = (pred == data.y[data.train_mask]).sum().item() / data.train_mask.sum().item()
        print(f"Epoch {epoch:03d}, Loss: {loss:.4f}, Train Accuracy: {acc:.4f}")
```

```
Epoch 000, Loss: 5.3452, Train Accuracy: 0.5109
Epoch 020, Loss: 1.2107, Train Accuracy: 0.5380
Epoch 040, Loss: 0.8489, Train Accuracy: 0.4891
Epoch 060, Loss: 0.7213, Train Accuracy: 0.5000
Epoch 080, Loss: 0.7058, Train Accuracy: 0.4891
Epoch 100, Loss: 0.7001, Train Accuracy: 0.5163
Epoch 120, Loss: 0.6893, Train Accuracy: 0.5000
Epoch 140, Loss: 0.6827, Train Accuracy: 0.4837
Epoch 160, Loss: 0.6862, Train Accuracy: 0.5000
Epoch 180, Loss: 0.6800, Train Accuracy: 0.4728
Epoch 200, Loss: 0.6794, Train Accuracy: 0.4783
```

Figure 18. Define and train model

Evaluate model on test set: using accuracy, precision, recall, and F1-score to assess generalization performance.

```python
gcn_model.eval()
with torch.no_grad():
    out = gcn_model(data.x, data.edge_index)
    pred_test = out[data.test_mask].argmax(dim=1)

# True and predicted labels
true = data.y[data.test_mask].cpu()
pred = pred_test.cpu()

# Metrics
accuracy = accuracy_score(true, pred)
precision = precision_score(true, pred)
recall = recall_score(true, pred)
f1 = f1_score(true, pred)

print(f"Accuracy:  {accuracy:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall:    {recall:.3f}")
print(f"F1-score:  {f1:.3f}")
```

```
Accuracy:  0.587
Precision: 0.587
Recall:    1.000
F1-score:  0.740
```

Figure 19. Evaluate model

Analyze the model's confidence scores

```
probs = F.softmax(out, dim=1)
probs_test = probs[data.test_mask]

# Top 10 highest probabilities for class 1
top_10_probs = probs_test[:, 1].topk(10)
print("Top 10 probs for class 1 in test set:", top_10_probs.values.detach().numpy())
```

```
Top 10 probs for class 1 in test set: [0.6968188  0.670793   0.6597198  0.6476739  0.62366    0.6219809
 0.6204664  0.6204664  0.6196621  0.61730087]
```

Test different thresholds to balance precision and recall, and identify the optimal decision boundary.

```
probs = F.softmax(out, dim=1)
probs_test = probs[data.test_mask]
y_true = data.y[data.test_mask].cpu()

thresholds = [0.5, 0.52, 0.55, 0.57, 0.6, 0.62, 0.65]

print("Threshold | Accuracy | Precision | Recall | F1-score")
print("-----------------------------------------------------")

for thresh in thresholds:
    y_pred = (probs_test[:, 1] > thresh).long().cpu()

    acc  = accuracy_score(y_true, y_pred)
    prec = precision_score(y_true, y_pred, zero_division=0)
    rec  = recall_score(y_true, y_pred)
    f1   = f1_score(y_true, y_pred)

    print(f"{thresh:9.2f} | {acc:8.3f} | {prec:9.3f} | {rec:6.3f} | {f1:8.3f}")
```

```
Threshold | Accuracy | Precision | Recall | F1-score
-----------------------------------------------------
     0.50 |    0.587 |     0.587 |  1.000 |    0.740
     0.52 |    0.587 |     0.587 |  1.000 |    0.740
     0.55 |    0.565 |     0.600 |  0.778 |    0.677
     0.57 |    0.478 |     0.565 |  0.481 |    0.520
     0.60 |    0.435 |     0.538 |  0.259 |    0.350
     0.62 |    0.413 |     0.500 |  0.148 |    0.229
     0.65 |    0.435 |     0.667 |  0.074 |    0.133
```

Figure 20. Identify optimal decision boundary

Generate candidate points

```
# Define bounding box for Dublin city
min_lat, max_lat = 53.30, 53.38
min_lon, max_lon = -6.35, -6.20

# Generate random candidate points
num_candidates = 50
np.random.seed(42)

candidate_points = []
while len(candidate_points) < num_candidates:
    rand_lat = np.random.uniform(min_lat, max_lat)
    rand_lon = np.random.uniform(min_lon, max_lon)
    candidate = Point(rand_lon, rand_lat)
    candidate_points.append(candidate)

candidates = gpd.GeoDataFrame(geometry=candidate_points, crs="EPSG:4326")
```

Figure 21. Generate candidate points for prediction

Build new graph with candidates

```
# Use coordinates of all nodes
coords = df_predict[['latitude', 'longitude']].values

# Build k-NN graph
k = 5
knn = NearestNeighbors(n_neighbors=k)
knn.fit(coords)
distances, indices = knn.kneighbors(coords)

# Build edge list
rows, cols = [], []
for i in range(len(coords)):
    for j in indices[i]:
        rows.append(i)
        cols.append(j)

# Convert to sparse adjacency matrix and then to edge_index
adj_matrix = coo_matrix((np.ones(len(rows)), (rows, cols)), shape=(len(coords), len(coords)))
edge_index, _ = from_scipy_sparse_matrix(adj_matrix)

# Build the graph
data_all = Data(x=x, edge_index=edge_index, y=y)
```

Figure 22. Build graph structure for candidate points

Predict station probabilities for candidates: Predicted the probability of each candidate being a station and selected those exceeding a 0.50 threshold.

```
# Run the trained model in evaluation mode
gcn_model.eval()
out = gcn_model(data_all.x, data_all.edge_index)

# Get softmax probabilities
probs_all = F.softmax(out, dim=1)

# Get probabilities for class 1 (station) in candidates
start_idx = len(df_full)
end_idx = len(df_predict)
probs_candidates = probs_all[start_idx:end_idx, 1]

# Apply custom threshold
threshold = 0.50
candidate_preds = (probs_candidates > threshold).long()

# Add predictions to the candidates GeoDataFrame
candidates['predicted_label'] = candidate_preds.cpu().numpy()

# Filter only those predicted as station (1)
suggested_locations = candidates[candidates['predicted_label'] == 1].copy()

print(f"Suggested new stations: {len(suggested_locations)}")
```

```
Suggested new stations: 27
```

```
# Center of Dublin
dublin_center = [53.35, -6.26]

# Create base map
m = folium.Map(location=dublin_center, zoom_start=13, tiles="CartoDB positron")

# Cluster for existing stations
existing_cluster = MarkerCluster(name="Existing Stations").add_to(m)
for _, row in df_enriched.iterrows():
    folium.CircleMarker(
        location=[row['latitude'], row['longitude']],
        radius=5,
        color='blue',
        fill=True,
        fill_color='blue',
        fill_opacity=0.7,
        popup=row.get("station_name", "Existing Station")
    ).add_to(existing_cluster)

# Cluster for suggested stations
suggested_cluster = MarkerCluster(name="Suggested Stations").add_to(m)
for _, row in suggested_locations.iterrows():
    folium.Marker(
        location=[row.geometry.y, row.geometry.x],
        icon=folium.Icon(color='green', icon='star', prefix='fa'),
        popup="Suggested Station"
    ).add_to(suggested_cluster)

# Add layer control
folium.LayerControl().add_to(m)

# Plot
m
```

Figure 23. Predict and visualize