006fin

April 21, 2025

1 Predicting High-risk Areas for Theft in London using Machine Learning

[1]:
```
import time
start_time = time.time()
```

Preparation

- See project in Github: Github link
- Number of words: 1497
- Runtime: 3.57 minutes (*Memory 32 GB, CPU Intel(R) Core(TM) Ultra 5 125H @3.60GHz*)
- Coding environment: VS Code + Python 3.12 (Windows 11)
- License: this notebook is made available under the Creative Commons Attribution license.
- Used packages:

[2]:
```
# for data cleaning and processing
import pandas as pd
import osmnx as ox # OSMnx is a Python package to get access to geospatial␣
↪features from OpenStreetMap. (Boeing, G. 2024)
import geopandas as gpd
import numpy as np
# for traditional modeling
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error, root_mean_squared_error,␣
↪r2_score
# for machine leaning method
from xgboost import XGBRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
import shap
# for visualization
```

```
from tabulate import tabulate # for table visualization
import seaborn as sns
import matplotlib.pyplot as plt
import geopandas as gpd
from mpl_toolkits.axes_grid1 import make_axes_locatable
from mapclassify import NaturalBreaks
```

1.1 Table of contents

1.2 Introduction

According to Office for Natinal Statistics (ONS) in 2024, London has one of the lowest rates of violent crime but also the highest overall rate of crime per 1,000 people. A main contributor for this situation is the high volume of various theft crime.(Hill, 2025)

With open data provided by Metropolitan Police Service (MPS) and ONS, this research seeks to predict high-risk areas and uncover key elements that influence their distributions. To enhance spatial interpretability, this study further visualized SHAP values across LSOAs, which has rarely been explored in previous works.

[go back to the top]

1.3 Literature review

Crime has been proved to be shaped by underlying socio-economic conditions and spatial processes. Classical theories like Routine Activity Theory (Cohen and Felson, 1979) have guided statistical modelling of crime, often through regression models (Glasson and Cozens, 2011).

Machine learning (ML) can capture non-linear and high-dimensional relationships, usually outperforming traditional models in terms of prediction (Yin, 2022; Yunus and Loo, 2024). Despite their accuracy, these "black-box" models have been criticized for lacking interpretability—making them difficult to apply in policy-making and urban governance (Mandalapu et al., 2023).

To address this, Zhang et al. (2022) applied XGBoost combined with SHAP (Shapley Additive exPlanations) to predict crime rate, showing that the proportion of non-local residents and age group contribute the most to crime prediction.

[go back to the top]

1.4 Research questions

According to previous studies, this study focusing on theft in London, aims to investigate whether interpretable machine learning methods can help to identify and predict crime hotspots, and explain the key drivers of crime rate.

To achieve this goal, the study is divided into three research questions:

- Does machine learning model outperform traditional statistical regression in predicting theft risk across different areas of London?
- What are the most influential factors contributing to theft risk?
- How can contributions of the factors be interpreted in different areas of London using machine learning model?

[go back to the top]

1.5 Methodology

1.5.1 Crime Rate Prediction

In recent years, machine learning models have been widely used in crime analysis. Among various machine learning algorithms, XGBoost (eXtreme Gradient Boosting) was selected due to its superior performance on tabular data and its capacity for handling nonlinear relationships. Compared with Random Forest, XGboost offers more efficient training through gradient boosting. Other ML methods like neural networks and K-nearest neighbors, had relatively high demands of data size and low interpretability. This research compare ML model with traditional statistical method to test their accuracy.

1.5.2 Feature Interpretation

In order to interpret feature importance of XGBoost, this study employs SHAP (SHapley Additive Explanations). Its core idea is to calculate the marginal contribution of features to the model output, and then interpret the "black-box" model from both the global and local levels.(Retzlaff et al., 2024)

Figure 1. Methodology [go back to the top] ***

1.6 Data

1.6.1 Data Source

This research combines multiple datasets related to crime, demographics, housing, and deprivation, aggregated at the LSOA (Lower Super Output Area) level in London. The research based on data from 2015-2019 to avoid the influce of Covid-19 after 2020, ensuring a more stable socio-economic environment for modeling, reducing noise from pandemic-related anomalies.

All data were spatially joined to LSOA boundaries 2011. Although the crime dataset was provided under the LSOA 2021 structure, only records that matched with the 2011 boundary were retained. A total of 4653 lsoas were used, resulting in a sample size of the data set of 23265 (4653*5=23265) observations.

The main target, theft rate per 1000 people, was calculated wiht the data from the Metropolitan Police's Crime records.

The predicting variables (Table 1) were selected based on prior studies (e.g., Zhang et al., 2022; Hojman, 2004) suggesting socio-economic and urban factors significantly influence crime risk. This study choose the supporting dataset like population density and POI counts to capture activity levels, while housing price and Indices of Deprivation (ID) to reflect socio-economic vulnerability.

Table 1. Used Variables

Variable	**Type**	**Description**	**Notes**
Theft_Rate_per1k	Numeric	Theft rate per 1,000 people. Used as the dependent variable.	Theft counts / Population
Population_Density	Numeric	Number of people per square kilometer.	Given by ONS
Vulnerable_Ratio	Numeric	Proportion of elderly and young people.	Age 0–15 & 65+ / Population
House_Price	Numeric	Median house price.	Given by ONS
Income_Score_rate	Numeric	Proportion of the population experiencing deprivation relating to low income.	Given by Gov.uk
Employment_Score_rate	Numeric	Proportion of working-age population involuntarily excluded from the labor market.	Given by Gov.uk
Education_Skills_and_Training_Score	Numeric	Lack of attainment and skills in the local population.	Given by Gov.uk
Health_Deprivation_and_Disability_Score	Numeric	Risk of premature death or reduced quality of life due to poor health.	Given by Gov.uk
Barriers_to_Housing_and_Services_Score	Numeric	Physical and financial inaccessibility of housing and local services.	Given by Gov.uk
Living_Environment_Score	Numeric	Quality of the local living environment.	Given by Gov.uk
tran_poi_count	Numeric	Number of transportation POIs.	Count from OSM
shop_poi_count	Numeric	Number of shop POIs.	Count from OSM

Variable	Type	Description	Notes
LSOA_Code	Categorical	Lower Super Output Area code, used for spatial join.	From London Datastore
Year	Numeric	Year of observation.	2015–2019

1.6.2 Data Cleaning and Preprocessing

The raw dataset contains many years of data and multiple metrics, and the following code and analysis helped with the collection and cleaning of the data.

[3]:
```
## 1. Crime data:
# noted that data is provided monthly.

# read raw crime data
df = pd.read_csv("https://raw.githubusercontent.com/meimao76/006assessment/refs/
↪heads/master/data/MPS_LSOA_Level_Crime_(Historical).csv")

# Filter major_category as Theft
df_theft = df[df["Major Category"].str.upper().str.contains("THEFT")]

# Filter data between 201501-201512
date_cols = [col for col in df_theft.columns if col.isdigit()]
date_cols = [col for col in date_cols if "201501" <= col <= "201912"]

# only keep lsoa column and time
df_filtered = df_theft[["LSOA Code"] + date_cols]

# turn the dataset into long format
df_long = pd.melt(
    df_filtered,
    id_vars=["LSOA Code"],
    value_vars=[col for col in df_filtered.columns if col.isdigit() and␣
↪"201501" <= col <= "201912"],
    var_name="Month",
    value_name="Count"
)

# add a column named year
df_long["Year"] = df_long["Month"].str[:4].astype(int)

# calculate the total count of a year
df_yearly = (
    df_long
    .groupby(["LSOA Code", "Year"])["Count"]
    .sum()
    .reset_index()
    .rename(columns={"Count": "Theft Count"})
```

```
)

print(df_yearly.shape)
print(df_yearly.head(2))

# save the lsoa in crime data, as a reference to select lsoas in London area
london_lsoa = df_yearly["LSOA Code"].unique()
```

```
(24940, 3)
   LSOA Code  Year  Theft Count
0  E01000006  2015            3
1  E01000006  2016            8
```

[4]:
```
## 2. Population Density:
# To align with the 2011 LSOA structure used in other datasets,
# population density values were restructured and filtered accordingly.
# Noted population density is provided in wide format.

# read density raw data
den = pd.ExcelFile("https://raw.githubusercontent.com/meimao76/006assessment/
↪refs/heads/master/data/sapelsoapopulationdensity20112022.xlsx")
sheet_names_den = den.sheet_names
# print(sheet_names_den)
# choose data sheet and filtered out the year
df_den = den.parse(sheet_name="Mid-2011 to mid-2022 LSOA 2021", skiprows=3,␣
↪header=0)
df_den.columns = df_den.columns.str.strip()
df_den = df_den.rename(columns={"LSOA 2021 Code": "LSOA Code",
                                "Mid-2015: People per Sq Km": "2015",
                                "Mid-2016: People per Sq Km": "2016",
                                "Mid-2017: People per Sq Km": "2017",
                                "Mid-2018: People per Sq Km": "2018",
                                "Mid-2019: People per Sq Km": "2019"})
df_den = df_den[["LSOA Code", "2015", "2016", "2017", "2018", "2019"]]

# turn into long format
den_long = pd.melt(df_den,
                   id_vars=["LSOA Code"],
                   var_name="Year",
                   value_name="Population Density")

den_long["Year"] = den_long["Year"].astype(int)
den_long["LSOA Code"] = den_long["LSOA Code"].astype(str)

# select London areas
london_den = den_long[den_long["LSOA Code"].isin(london_lsoa)]
```

```
print(london_den.shape)
print(london_den.head(2))
```

```
(24940, 3)
   LSOA Code  Year  Population Density
4  E01000006  2015        13158.253752
5  E01000007  2015        10790.000000
```

[5]:
```
## 3. Population and Vulnerable Group:
# Vulnerable group is defined as the proportion of people younger than 15 and␣
↪elder than 65,
# which are considered as the potential vistims.

# read population raw data
pop = pd.ExcelFile("https://raw.githubusercontent.com/meimao76/006assessment/
↪refs/heads/master/data/sapelsoabroadage20112022.xlsx")
sheet_names = pop.sheet_names
# print(sheet_names)

# defined the needed sheets and column
target_sheets =['Mid-2015 LSOA 2021', 'Mid-2016 LSOA 2021', 'Mid-2017 LSOA␣
↪2021', 'Mid-2018 LSOA 2021', 'Mid-2019 LSOA 2021']
target_year = {'Mid-2015 LSOA 2021': 2015,
               'Mid-2016 LSOA 2021': 2016,
               'Mid-2017 LSOA 2021': 2017,
               'Mid-2018 LSOA 2021': 2018,
               'Mid-2019 LSOA 2021': 2019}

# combine all the data sheets
pop_list=[]
for sheet in target_sheets:
    df_pop = pop.parse(sheet, skiprows=3, header=0)
    df_pop.columns = df_pop.columns.str.strip()
    df_pop = df_pop.rename(columns={
        "LSOA 2021 Code": "LSOA Code",
        "Total": "Population"
    })
    df_pop["Year"] = target_year[sheet]
    # calculating proportion of vulnerable group
    df_pop["Vulnerable_Group"] = df_pop["M65 and over"] + df_pop["F65 and␣
↪over"] + df_pop["F0 to 15"] + df_pop["M0 to 15"]
    df_pop["Vulnerable_Ratio"] = df_pop["Vulnerable_Group"] /␣
↪df_pop["Population"]

    pop_list.append(df_pop[["LSOA Code", "Year", "Population",␣
↪"Vulnerable_Ratio"]])
```

```
pop_fin = pd.concat(pop_list, ignore_index=True)

# select data in London areas
london_pop = pop_fin[pop_fin["LSOA Code"].isin(london_lsoa)]

print(london_pop.shape)
print(london_pop.head(2))
```

```
(24940, 4)
       LSOA Code  Year  Population  Vulnerable_Ratio
28767  E01000006  2015        1929          0.325557
28768  E01000007  2015        2158          0.297498
```

[6]:
```
## 4. House Price:
# Median house prices serve as an indicator of affluence and built environment␣
 ↪quality,
# potentially affecting the occurrence and attractiveness of theft-related␣
 ↪crime.
# Noted that this dataset in listed in 2011LSOA.

# read raw house price data
housing = pd.ExcelFile("https://raw.githubusercontent.com/meimao76/
 ↪006assessment/refs/heads/master/data/
 ↪hpssadataset46medianpricepaidforresidentialpropertiesbylsoa/median_price.
 ↪xlsx")
sheet_names_housing = housing.sheet_names
# print(sheet_names_housing)
# select used data sheet
df_housing = housing.parse(sheet_name="1a", skiprows=5, header=0)
df_housing.columns = df_housing.columns.str.strip()
df_housing = df_housing.rename(columns={"LSOA code": "LSOA Code"})

# turn into long format
housing_long = pd.melt(df_housing,
                  id_vars=["LSOA Code"],
                  var_name="Date",
                  value_name="House Price")

# extract the year out as a new column
housing_long["House Price"] = pd.to_numeric(housing_long["House Price"],␣
 ↪errors="coerce")
housing_long["Year"] = housing_long["Date"].str.extract(r"(\d{4})")
housing_long = housing_long.dropna(subset=["Year"])
housing_long["Year"] = housing_long["Year"].astype(int)
housing_yearly = housing_long.groupby(["LSOA Code", "Year"])["House Price"].
 ↪mean().reset_index()
```

```
# select data in London and between 2015-2019
london_housing = housing_yearly[housing_yearly["LSOA Code"].isin(london_lsoa) &
                               (housing_yearly["Year"].between(2015, 2019))].
 ↪copy()

# fill in the missing value in the dataset
london_housing["House Price"] = london_housing.groupby("LSOA Code")["House␣
 ↪Price"].transform(lambda x: x.fillna(x.mean()))
london_housing["House Price"] = london_housing["House Price"].
 ↪fillna(london_housing["House Price"].median())

print(london_housing.shape)
print(london_housing.head(2))
```

```
(23265, 3)
      LSOA Code  Year  House Price
136  E01000006  2015     196125.0
137  E01000006  2016     349062.5
```

[7]:
```
## 5. Scores for the Indices of Deprivation:
# In addition to the crime domain, other six scores served to capture␣
 ↪structural vulnerabilities of each area.
# Noted that this dataset in listed in 2011LSOA.

# read raw id score data
IMD = pd.ExcelFile("https://raw.githubusercontent.com/meimao76/006assessment/
 ↪refs/heads/master/data/File_5_-_IoD2019_Scores.xlsx")
sheet_names_IMD = IMD.sheet_names
# print(sheet_names_IMD)
# select data sheet
df_IMD = IMD.parse(sheet_name="IoD2019 Scores", header=0)
df_IMD.columns = df_IMD.columns.str.strip()
df_IMD = df_IMD.rename(columns={"LSOA code (2011)": "LSOA Code"})
# select data in London areas
df_IMD = df_IMD[df_IMD["LSOA Code"].isin(london_lsoa)]
# keep usefull columns
keep_cols3 = ["LSOA Code", "Income Score (rate)",
              "Employment Score (rate)",
              "Education, Skills and Training Score",
              "Health Deprivation and Disability Score",
              "Barriers to Housing and Services Score",
              "Living Environment Score"]
IMD_fin = df_IMD[keep_cols3]

print(IMD_fin.shape)
print(IMD_fin.head(2))
```

```
(4653, 7)
```

```
   LSOA Code  Income Score (rate)  Employment Score (rate)  \
4  E01000006                0.117                    0.059
5  E01000007                0.207                    0.107

   Education, Skills and Training Score  \
4                                14.798
5                                11.385

   Health Deprivation and Disability Score  \
4                                   -0.359
5                                   -0.027

   Barriers to Housing and Services Score  Living Environment Score
4                                  45.171                    26.888
5                                  50.420                    25.995
```

[8]:

```
# save the lsoa list in cleaned ID dataset as the final analysed lsoas
lsoa11 = IMD_fin["LSOA Code"].unique()

# filtered all the dataset again
df_yearly = df_yearly[df_yearly["LSOA Code"].isin(lsoa11)]
london_pop = london_pop[london_pop["LSOA Code"].isin(lsoa11)]
london_den = london_den[london_den["LSOA Code"].isin(lsoa11)]
london_housing = london_housing[london_housing["LSOA Code"].isin(lsoa11)]
```

[9]:

```
## 6. POI Counts:
# Select the number of transportation and shops in the area,
# as they might act as crime attractors and influence crime opportunities.

# read lsoa 2011 boundaries
lsoa_gdf = gpd.read_file("https://raw.githubusercontent.com/meimao76/
 ↪006assessment/refs/heads/master/data/statistical-gis-boundaries-london/
 ↪statistical-gis-boundaries-london/ESRI/LSOA_2011_London_gen_MHW.shp")

# get poi data from open street map
# code from osmnx website
place = "London, UK"
tags1 = {"railway": "station", "highway": "bus_stop"}  # subway stations and␣
 ↪bus stations
tags2 = {"shop": True } # shops
gdf1 = ox.features_from_place(place, tags1)
gdf2 = ox.features_from_place(place, tags2)

# set the same crs
tran_poi_gdf = gdf1.to_crs(lsoa_gdf.crs)
shop_poi_gdf = gdf2.to_crs(lsoa_gdf.crs)
```

```python
# change all types of geospatial features into points
shop_poi_gdf["geometry"] = shop_poi_gdf.geometry.centroid

# join the transportation points with the lsoas
joined_tran = gpd.sjoin(tran_poi_gdf, lsoa_gdf, how='inner', predicate='within')
# counts the points within each lsoa
tran_poi_count = joined_tran.groupby('LSOA11CD').size().
 ↪reset_index(name='tran_poi_count')
tran_poi_count = tran_poi_count.rename(columns={"LSOA11CD": "LSOA Code"})

# same with the shop points
joined_shop = gpd.sjoin(shop_poi_gdf, lsoa_gdf, how='inner', predicate='within')
shop_poi_count = joined_shop.groupby('LSOA11CD').size().
 ↪reset_index(name='shop_poi_count')
shop_poi_count = shop_poi_count.rename(columns={"LSOA11CD": "LSOA Code"})

print(tran_poi_count.head(2))
print(shop_poi_count.head(2))
```

```
   LSOA Code  tran_poi_count
0  E01000007               5
1  E01000008               1
   LSOA Code  shop_poi_count
0  E01000005               1
1  E01000007              13
```

[10]:
```python
# merge data
df_model = df_yearly.copy()
df_model = df_model.merge(london_pop, on=["LSOA Code", "Year"], how="left")
df_model = df_model.merge(london_den, on=["LSOA Code", "Year"], how="left")
df_model = df_model.merge(london_housing, on=["LSOA Code", "Year"], how="left")
df_model = df_model.merge(IMD_fin, on="LSOA Code", how="left")
df_model = df_model.merge(tran_poi_count, on="LSOA Code", how="left")
df_model["tran_poi_count"] = df_model["tran_poi_count"].fillna(0).astype(int)
df_model = df_model.merge(shop_poi_count, on="LSOA Code", how="left")
df_model["shop_poi_count"] = df_model["shop_poi_count"].fillna(0).astype(int)

# calculate crime rate
df_model["Theft_Rate_per1k"] = df_model["Theft Count"] / df_model["Population"]␣
 ↪* 1000

# delete the column that no longer used
df_model = df_model.drop(columns=["Theft Count", "Population"])

# amend column names
df_model.columns = df_model.columns.str.replace(" ", "_")
df_model.columns = df_model.columns.str.replace(r"[()/]", "", regex=True)
```

```
print(df_model.head(2))
print(df_model.shape)
```

```
   LSOA_Code  Year  Vulnerable_Ratio  Population_Density  House_Price  \
0  E01000006  2015          0.325557        13158.253752     196125.0
1  E01000006  2016          0.319872        12837.653479     349062.5

   Income_Score_rate  Employment_Score_rate  \
0              0.117                  0.059
1              0.117                  0.059

   Education,_Skills_and_Training_Score  \
0                                14.798
1                                14.798

   Health_Deprivation_and_Disability_Score  \
0                                   -0.359
1                                   -0.359

   Barriers_to_Housing_and_Services_Score  Living_Environment_Score  \
0                                  45.171                    26.888
1                                  45.171                    26.888

   tran_poi_count  shop_poi_count  Theft_Rate_per1k
0               0               0          1.555210
1               0               0          4.250797
(23265, 14)
```

Figure 2 shows that the distribution of variables used for theft prediction. Several variables, such as house prices, POI counts, and theft rate, are right-skewed, indicating the presence of extreme values or heavy tails.

[11]:
```
# select variables to show distribution
variables = df_model.drop(columns=["Year", "LSOA_Code"]).columns.tolist()
# creat a 3*4 figure
fig, axes = plt.subplots(3, 4, figsize=(18, 12))
axes = axes.flatten()
# for loop
for i, var in enumerate(variables):
    sns.histplot(df_model[var], kde=True, bins=30, ax=axes[i])
    axes[i].set_title(f"{var}")
    axes[i].set_xlabel("")
    axes[i].set_ylabel("")
# show picture
plt.tight_layout()
plt.suptitle("Distributions of Model Variables", fontsize=16, y=1.02)
plt.show()
```

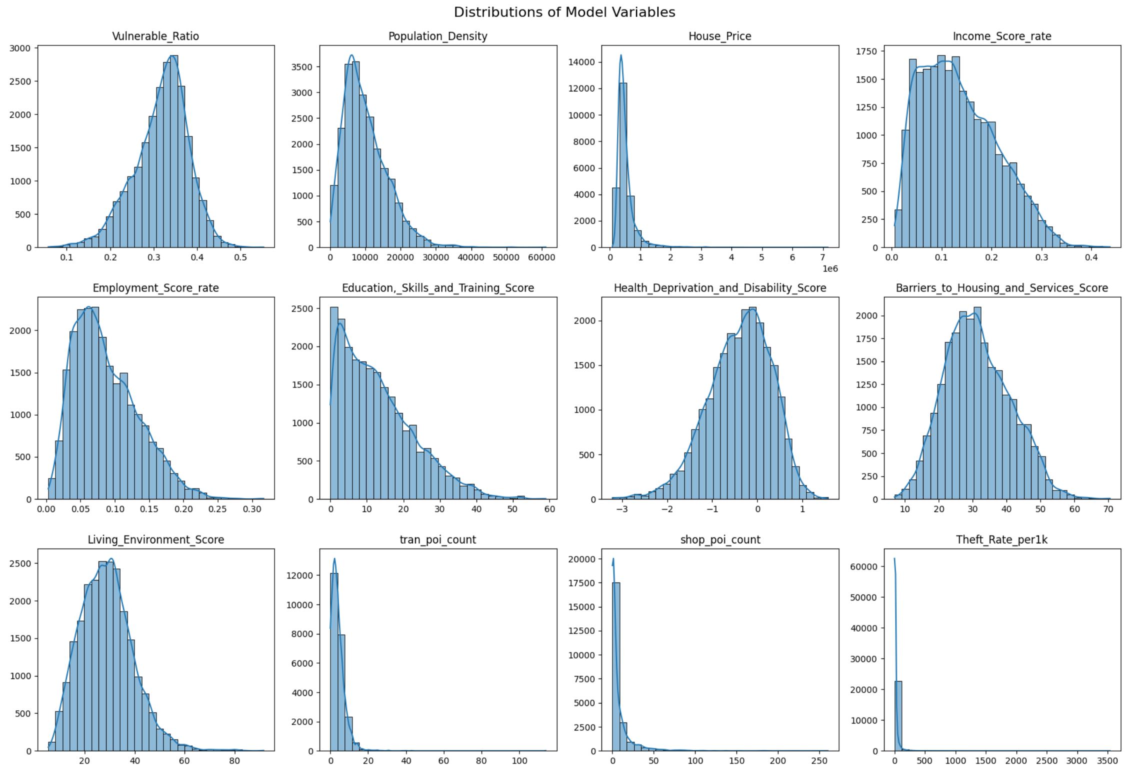

Figure 2. Variables Distribution

Figure 3 shows that correlation matrix of all variables used in the model. While most variables exhibit low moderate correlations, a strong positive correlation between Income_Score_rate and Employment_Score_rate suggests that may require further consideration.

[12]:
```python
df_model_var = df_model.drop(columns=["Theft_Rate_per1k", "LSOA_Code"])

plt.figure(figsize=(10, 10))
corr = df_model_var.corr()
sns.heatmap(corr, annot=False, cmap="viridis", square=True)
plt.title("Correlation Matrix of Model Features")
plt.tight_layout()
plt.show()
```

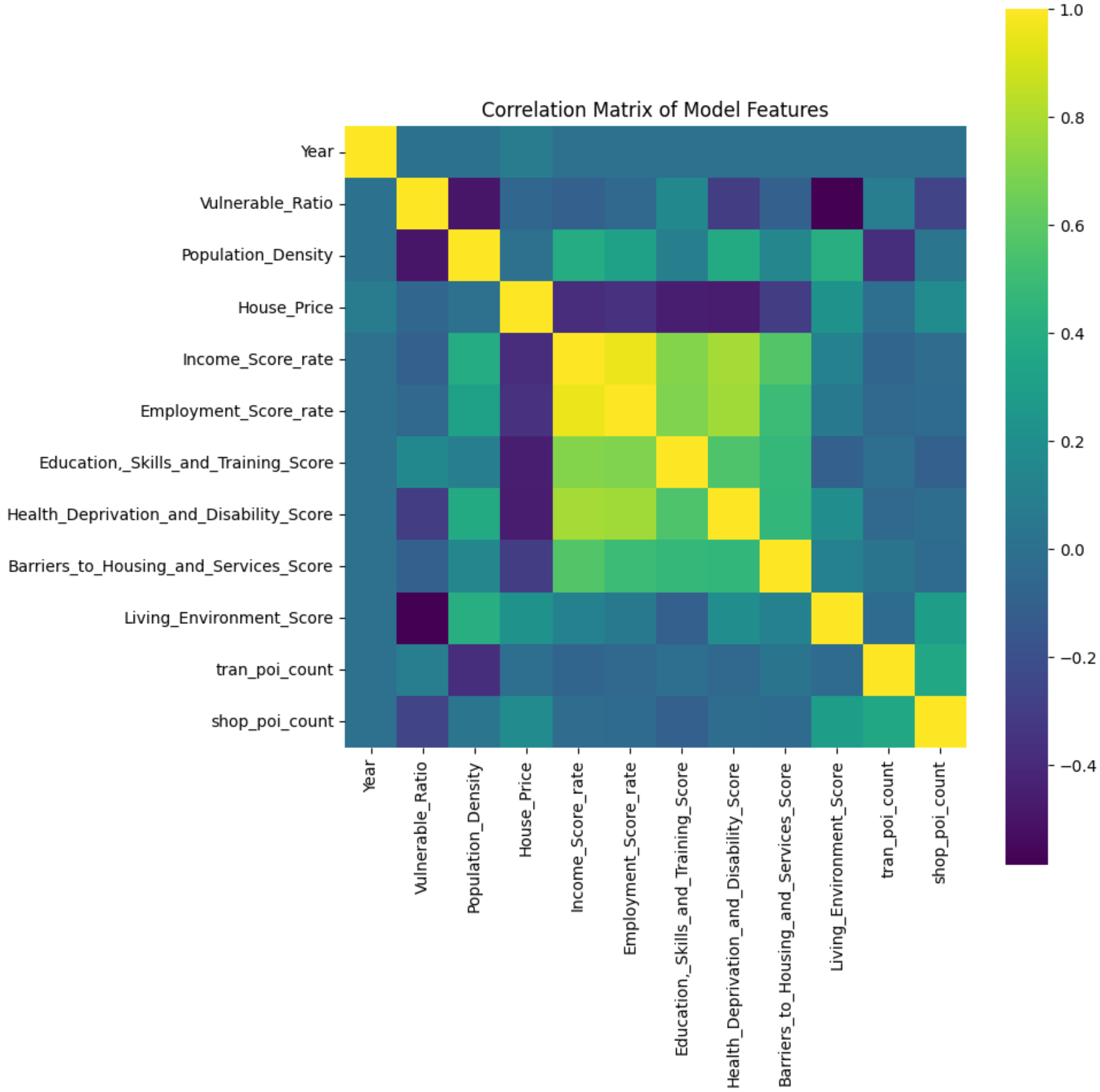

Figure 3. Correlation Matrix

[go back to the top]

1.7 Results

1.7.1 Prediction Modeling

OLS model To improve the performance and validity of the OLS regression model, standardization of the variables are necessary preprocessing steps. Additionally, Variance Inflation Factor (VIF) analysis is needed before the regression.

[13]:
```python
# Log transformation
df_model_ols = df_model.copy() # set a copy of data frame for OLS regression

# Standardization
scaler = StandardScaler()
features = df_model_ols.drop(columns=["LSOA_Code", "Employment_Score_rate",␣
↪"Theft_Rate_per1k"])
X_scaled = scaler.fit_transform(features)
```

[14]:
```python
X_ols = pd.DataFrame(X_scaled, columns=features.columns)
vif = pd.DataFrame()
vif["Variable"] = X_ols.columns
vif["VIF"] = [variance_inflation_factor(X_ols.values, i) for i in range(X_ols.
↪shape[1])]
print(vif)
```

```
                                   Variable       VIF
0                                      Year  1.009638
1                          Vulnerable_Ratio  2.128764
2                        Population_Density  1.980572
3                               House_Price  1.583846
4                         Income_Score_rate  4.506444
5      Education,_Skills_and_Training_Score  2.470923
6   Health_Deprivation_and_Disability_Score  3.461406
7     Barriers_to_Housing_and_Services_Score  1.568236
8                  Living_Environment_Score  1.738073
9                            tran_poi_count  1.416755
10                           shop_poi_count  1.368538
```

Based on the VIF results, the selected variables can be safely used in the OLS regression model.

[15]:
```python
# OLS
X_ols = sm.add_constant(X_ols)  #
y_ols = df_model_ols["Theft_Rate_per1k"]
ols_model = sm.OLS(y_ols, X_ols).fit()
y_pred_ols = ols_model.predict(X_ols)

# Model evaluation

r2_ols = r2_score(y_ols, y_pred_ols)
rmse_ols = root_mean_squared_error(y_ols, y_pred_ols)
mae_ols = mean_absolute_error(y_ols, y_pred_ols)
```

XGBoost model

[16]:
```python
df_model_xgb = df_model.copy() # set a copy of data frame for XGBoost

# XGBoost
```

```python
X_xgb = df_model_xgb.drop(columns=["Theft_Rate_per1k", "LSOA_Code",␣
↪"Employment_Score_rate"])
y_xgb = df_model_xgb["Theft_Rate_per1k"]
lsoa_code = df_model_xgb["LSOA_Code"].astype(str)

# set train and test group
X_train_xgb, X_test_xgb, y_train_xgb, y_test_xgb, lsoa_train, lsoa_test =␣
↪train_test_split(
    X_xgb, y_xgb, lsoa_code, test_size=0.2, random_state=42
)

# construct the GridSearch + XGBoost model
xgb = XGBRegressor(objective='reg:squarederror',
                   early_stopping_rounds = 10,
                   eval_metric= 'rmse',
                   random_state=42)

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.05, 0.1],
    'subsample': [0.7, 0.8],
    'colsample_bytree': [0.7, 0.8]
}

grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    scoring='neg_root_mean_squared_error',
    cv=3,
    verbose=1,
    n_jobs=-1
)

grid_search.fit(
    X_train_xgb, y_train_xgb,
    **{
        "eval_set": [(X_train_xgb, y_train_xgb),(X_test_xgb, y_test_xgb)],
        "verbose": False
    }
)

# get the best model and retrain
best_model = grid_search.best_estimator_
# get a record of the training process
eval_result = best_model.evals_result()
```

```python
# model evaluation
y_pred_xgb = best_model.predict(X_test_xgb)
rmse_xgb = root_mean_squared_error(y_test_xgb, y_pred_xgb)
mae_xgb = mean_absolute_error(y_test_xgb, y_pred_xgb)
r2_xgb = r2_score(y_test_xgb, y_pred_xgb)
```

```
Fitting 3 folds for each of 48 candidates, totalling 144 fits
```

With XGBoost learning curve showed in Figure 4, test RMSE sharply declines in the early stage and gradually stabilizes, suggesting that the model generalizes well without severe overfitting.

[17]:
```python
# 6. Learning Curve
train_errors = eval_result['validation_0']['rmse']
test_errors = eval_result['validation_1']['rmse']

plt.figure(figsize=(10, 5))
plt.plot(train_errors, label="Train RMSE")
plt.plot(test_errors, label="Test RMSE")
plt.xlabel("Boosting Round")
plt.ylabel("RMSE")
plt.title("XGBoost Learning Curve")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

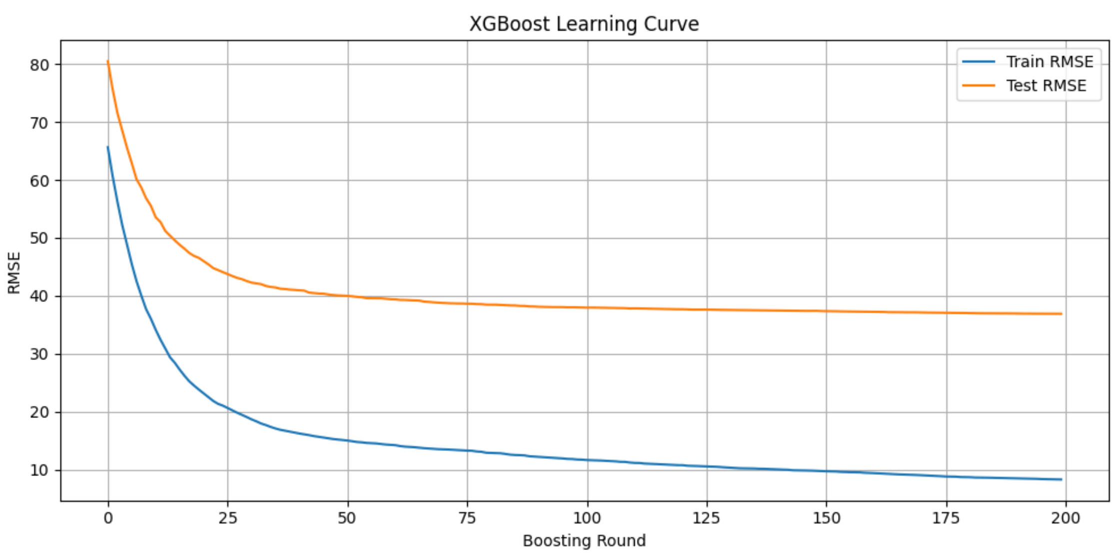

Figure 4. Learning Curve of XGBoost

Comparison As shown in Table 2, the XGBoost model achieves an R^2 of 0.799 while OLS model only reaches 0.474. Additionally, the RMSE of the XGBoost model (36.88) is markedly lower than that of the OLS model (51.11), and the MAE drops from 19.47 to just 8.27, demonstrating its effectiveness in minimizing prediction errors.

Table 2. Evaluation of OLS and XGboost

[18]:
```
# make the evaluation of two models into a table
comparison = pd.DataFrame({
    "Model": ["OLS", "XGBoost"],
    "R²": [r2_ols, r2_xgb],
    "RMSE": [rmse_ols, rmse_xgb],
    "MAE": [mae_ols, mae_xgb]
})

print(tabulate(comparison, headers='keys', tablefmt='github', showindex=False))
```

```
| Model   |       R² |    RMSE |      MAE |
|---------|----------|---------|----------|
| OLS     | 0.473621 | 51.1116 | 19.4711  |
| XGBoost | 0.79854  | 36.8811 |  8.27737 |
```

Figure 5 shows the actual and predicted theft rate. Points closer to the diagonal line represent better predictions. XGBoost predictions show a tighter alignment with actual values, indicating higher accuracy.

[19]:
```
# creat figure
plt.figure(figsize=(10, 5))

# OLS actual vs predict
sns.scatterplot(x=y_ols, y=y_pred_ols, label="OLS Prediction", alpha=0.5,␣
↪color="skyblue")
# XGBoost actual vs predict
sns.scatterplot(x=y_test_xgb, y=y_pred_xgb, label="XGBoost Prediction", alpha=0.
↪5, color="tomato")

# ideal line y = x
min_val = min(y_ols.min(), y_test_xgb.min())
max_val = max(y_ols.max(), y_test_xgb.max())
plt.plot([min_val, max_val], [min_val, max_val], color='gray', linestyle='--',␣
↪label="Ideal Fit")

plt.title("Actual vs Predicted: OLS vs XGBoost")
plt.xlabel("Actual Theft Rate per 1k")
plt.ylabel("Predicted Theft Rate per 1k")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

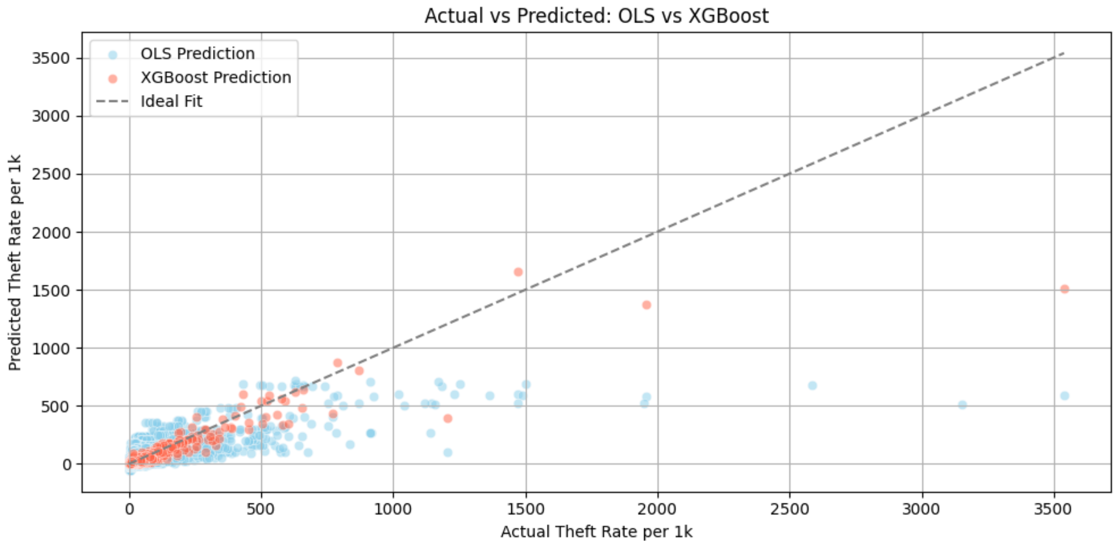

Figure 5. Actual vs Predicted theft rates for OLS and XGBoost models

Figure 6 presents the residuals vs fitted values plot for both models. The residuals of the OLS model show a wider and more structured spread compared to those from XGBoost, indicating that the ML model captures the variance in data more effectively and with less bias.

[20]:
```
# residual value
residuals_ols = y_ols - y_pred_ols
residuals_xgb = y_test_xgb - y_pred_xgb

# fitted value
fitted_ols = y_pred_ols
fitted_xgb = y_pred_xgb

# creat figure
plt.figure(figsize=(10, 5))

plt.scatter(fitted_ols, residuals_ols, alpha=0.5, label="OLS", color="skyblue")
plt.scatter(fitted_xgb, residuals_xgb, alpha=0.5, label="XGBoost",␣
  ↪color="tomato")

# ideal line
plt.axhline(y=0, color="gray", linestyle="--")

plt.xlabel("Fitted Values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted Values (OLS vs XGBoost)")
plt.legend()
plt.grid(True)
```

```
plt.tight_layout()
plt.show()
```

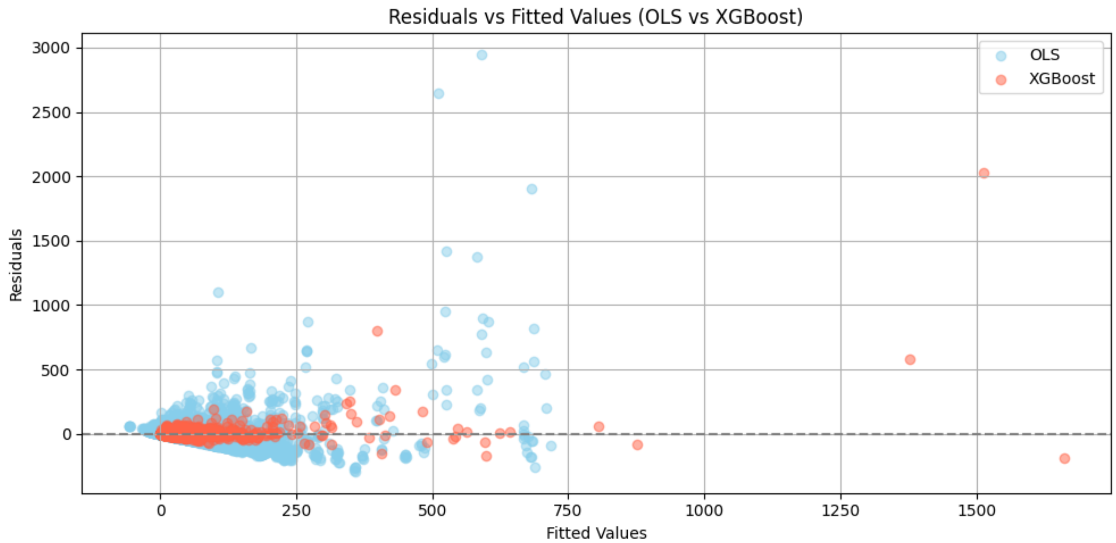

Figure 6. Residuals vs Fitted value for OLS and XGBoost models

Figure 7 compares the distribution of residuals for both OLS and XGBoost. The XGBoost model exhibits a sharper and more centralized residual distribution while OLS shows a wider spread and heavier tails, suggesting XGBoost has a better overall fit and fewer large prediction errors compared to OLS.

[21]:
```
# creat figure
plt.figure(figsize=(10, 5))

sns.kdeplot(residuals_ols, label="OLS", fill=True, color="skyblue", alpha=0.5,␣
↪linewidth=2)
sns.kdeplot(residuals_xgb, label="XGBoost", fill=True, color="salmon", alpha=0.
↪5, linewidth=2)

plt.axvline(0, color='gray', linestyle='--', linewidth=1)
plt.xlim(-200, 200)  # focusing on tha main part
plt.xlabel("Residual", fontsize=12)
plt.ylabel("Density", fontsize=12)
plt.title("Residual Distribution: OLS vs XGBoost", fontsize=14)
plt.legend(title="Model")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()

plt.show()
```

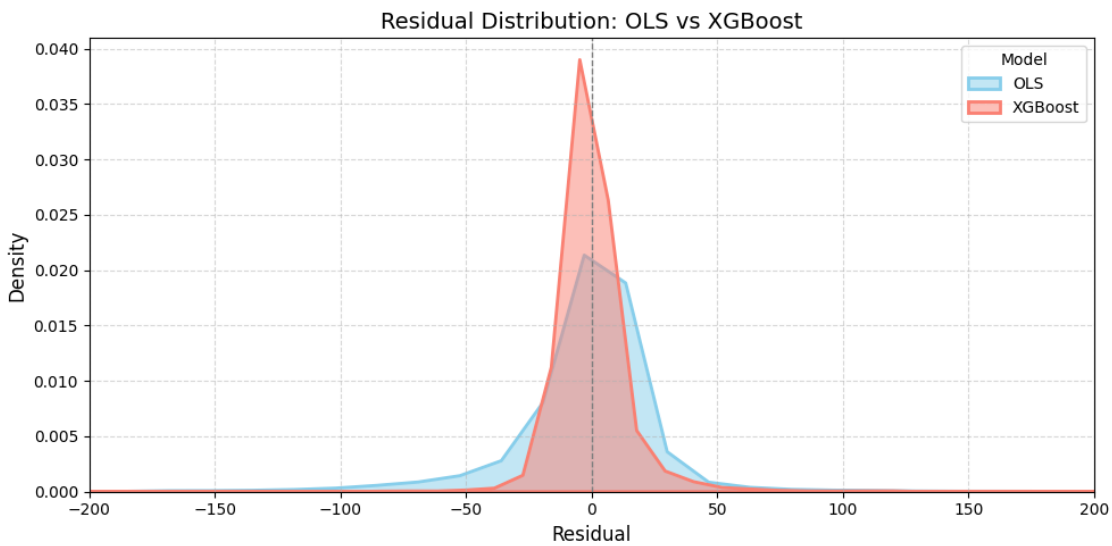

Figure 7. Distribution of Residuals

Figure 8 illustrates the spatial distribution of theft risk across London as predicted by the XGBoost model. The spatial pattern highlights central London and several surrounding districts as potential theft hotspots.

[22]:
```
lsoa_code = df_model.loc[X_xgb.index, "LSOA_Code"].astype(str)

lsoa_used = lsoa_code.unique()
lsoa_gdf = lsoa_gdf.rename(columns={"LSOA11CD": "LSOA_Code"})
lsoa_gdf = lsoa_gdf[lsoa_gdf["LSOA_Code"].isin(lsoa_used)].copy()

df_pred = pd.DataFrame({
    "LSOA_Code": lsoa_test,
    "predicted_theft": y_pred_xgb
})

threshold = df_pred["predicted_theft"].quantile(0.90)
df_pred["hotspot"] = df_pred["predicted_theft"] >= threshold

pred_gdf = lsoa_gdf.merge(df_pred, on="LSOA_Code")

classifier = NaturalBreaks(pred_gdf["predicted_theft"], k=5)  # 5 level natural␣
 ↪breaks

# add the classification back to gdf
pred_gdf["theft_class"] = classifier.yb  # yb = Classification

pred_gdf.plot(column="theft_class", cmap="OrRd", legend=True, figsize=(10, 10))
```

```
plt.title("Predicted Theft Rate (Natural Breaks)")
plt.axis("off")
plt.tight_layout()
plt.show()
```

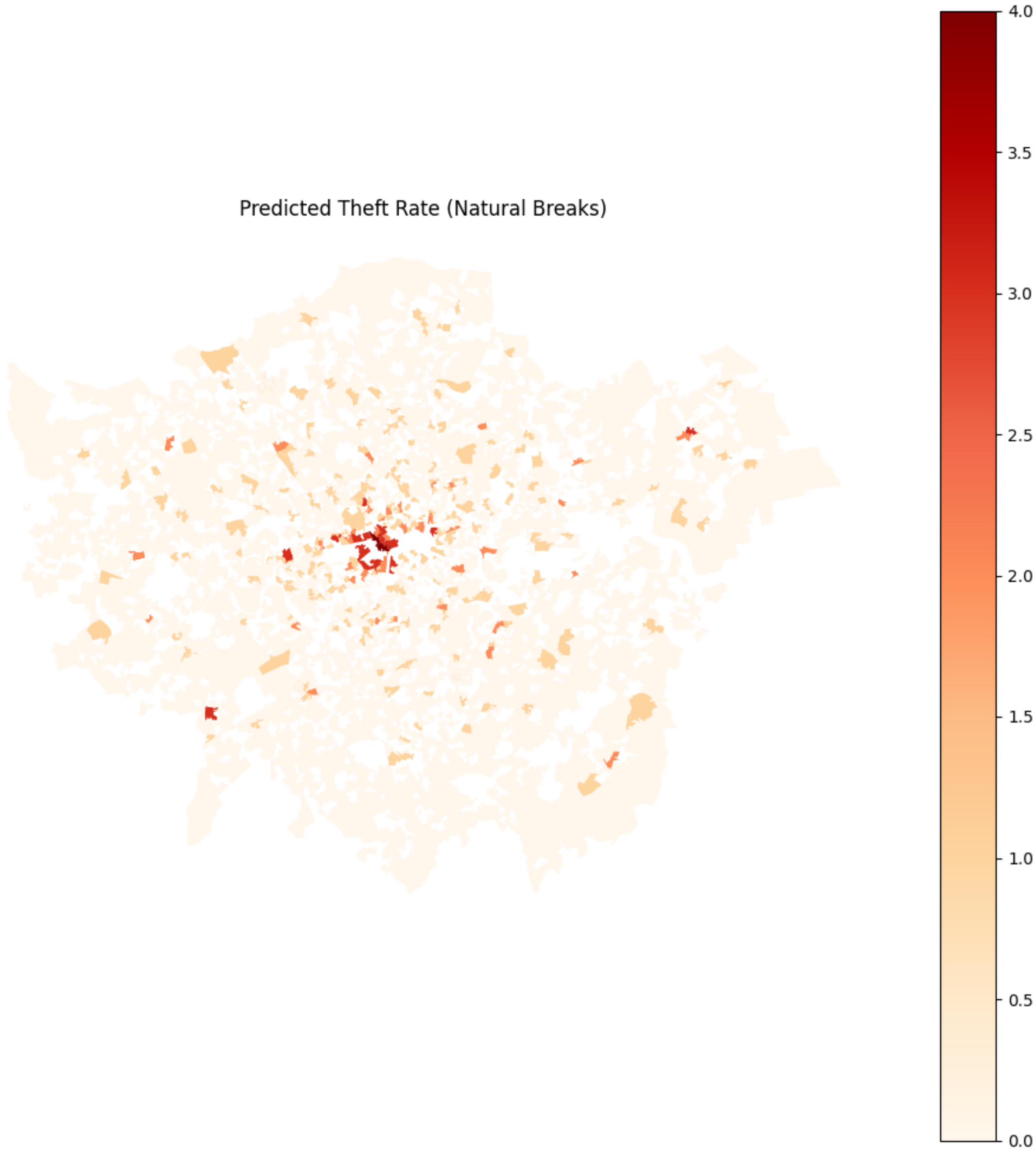

Figure 8. Predicted Theft Rate Map

1.7.2 Influencing Features

OLS coefficients The OLS coefficients provide a straightforward interpretation of feature effects as shown in Figure 9.

```
[23]: # extract column
coef_df = pd.DataFrame({
    "Variable": X_ols.columns,
    "Coefficient": ols_model.params
}).sort_values(by="Coefficient", ascending=False)
coef_df = coef_df[coef_df["Variable"] != "const"] # exclude intercept

plt.figure(figsize=(10, 5))
sns.barplot(x="Coefficient", y="Variable", data=coef_df, palette="coolwarm")
plt.title("OLS Coefficients for Theft Rate Model")
plt.xlabel("Coefficient Value")
plt.ylabel("Variable")
plt.grid(True, axis='x', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_37816\820080869.py:9: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x="Coefficient", y="Variable", data=coef_df, palette="coolwarm")
```

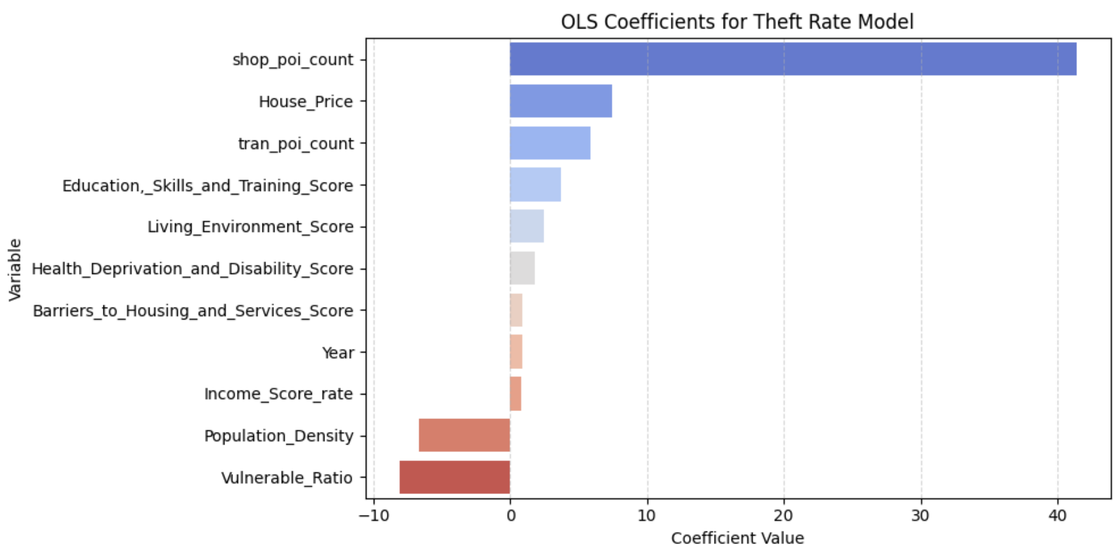

Figure 9. OLS Coefficients

SHAP SHAP allows XGBoost to explain the factors on both local and global level, as showed in Figure 10.

[24]:
```
# calculate SHAP value
explainer = shap.Explainer(best_model, X_train_xgb)
shap_values = explainer(X_train_xgb)
shap.plots.bar(shap_values)
```

```
99%|===================| 18516/18612 [02:01<00:00]
```

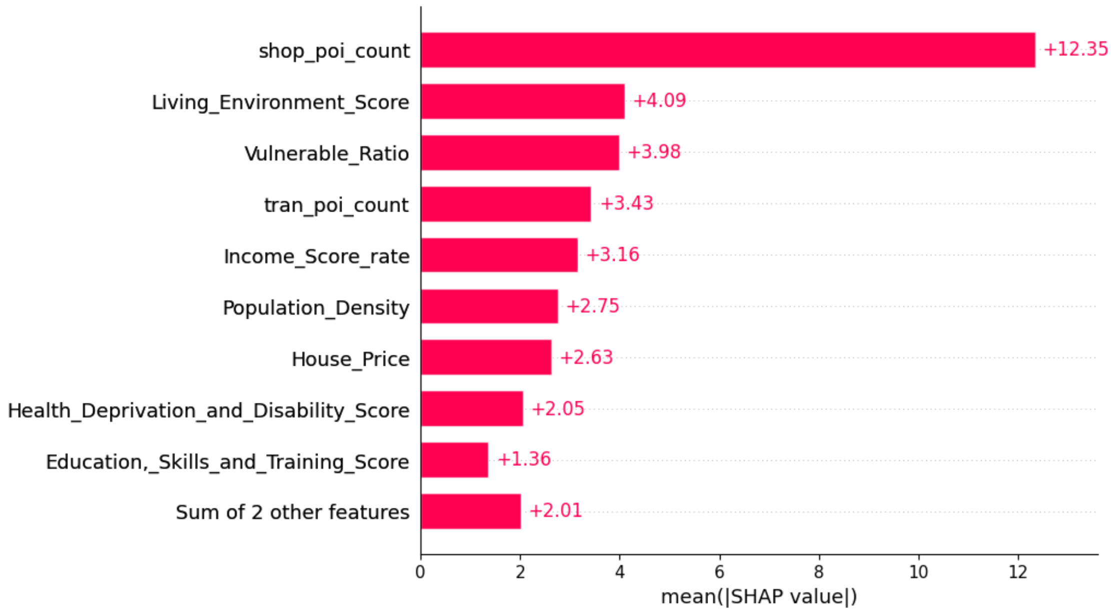

Figure 10. Ranking of the Absolute Value of SHAP Value of All Features

Both OLS and XGBoost models identified similar top two predictors. However, the rankings diverge for subsequent features. This may be attributed to the non-linear interactions captured by XGBoost but overlooked in the linear OLS model.

The actual SHAP value (Figure 11) of all grids allowes for a better understand on the impact of each variable. It indicates that higher shop density (represented by red dots) consistently contributes to higher predicted theft rates, suggesting that commercial activity may serve as crime attractors.

Similarly, the proportion of vulnerable population is also strongly associated with theft risk. High values of this feature are linked to increased model outputs, indicating that areas with more vulnerable individuals may experience higher theft rates—potentially.

[25]:
```
shap.plots.beeswarm(shap_values)
```

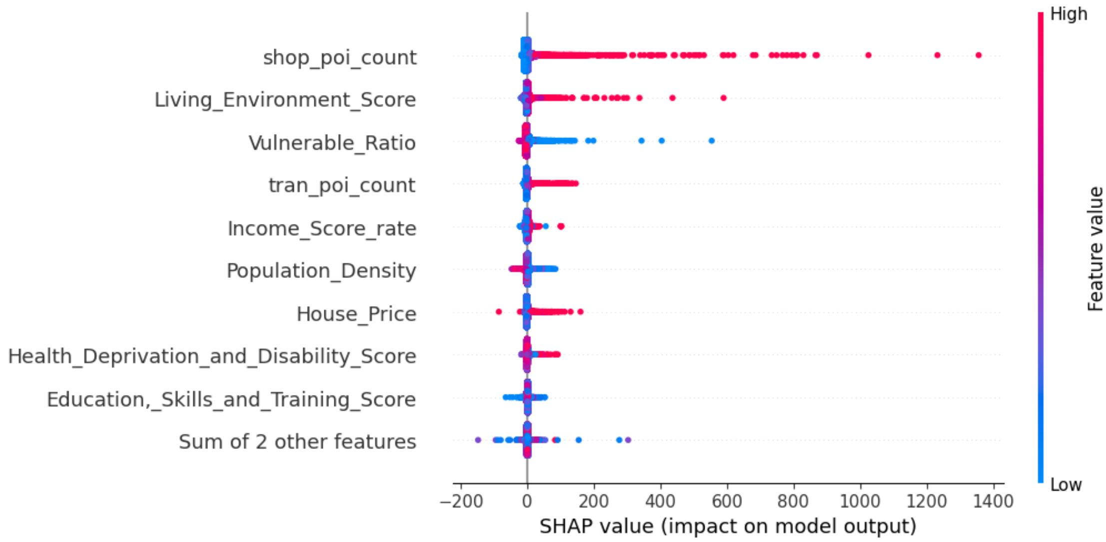

Figure 11. Distribution of SHAP Values of All Samples

Local Various By analysing the local variation in SHAP values across different grids, local interpretability allows for a detailed examination of how each feature contributes to individual predictions. Figure 12 provides a spatial perspective by showing its localization contribution using the number of shops as an example. This implies that the same factor can have varying influences across different LSOAs.

[26]:
```
# extract the shap value
shap_df = pd.DataFrame(shap_values.values, columns=X_train_xgb.columns)
# merge shap value into lsoa_gdf
shap_df["LSOA_Code"] = lsoa_train.reset_index(drop=True)
lsoa_gdf["LSOA_Code"] = lsoa_gdf["LSOA_Code"].astype(str)
gdf_with_shap = lsoa_gdf.merge(shap_df, on="LSOA_Code", how="left")

# extract the shap value of shop_poi_count
shop_shap = "shop_poi_count"

# creat figure
fig, ax = plt.subplots(1, 1, figsize=(10, 10))

gdf_with_shap["centroid"] = gdf_with_shap.geometry.centroid
gdf_points = gdf_with_shap.set_geometry("centroid")

# background layer
gdf_with_shap.plot(ax=ax, color="lightgrey", edgecolor="white")

# scatter layer
bubble = gdf_points.plot(
```

```
    ax=ax,
    column=shop_shap,
    cmap="Reds",
    markersize=gdf_points[shop_shap].clip(lower=0) * 0.5,
    alpha=0.7,
    legend=False,
    vmax=gdf_with_shap[shop_shap].quantile(0.95)
)

# colorbar legend
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.1)

sm = plt.cm.ScalarMappable(
    cmap="Reds",
    norm=plt.Normalize(
        vmin=gdf_points[shop_shap].min(),
        vmax=gdf_points[shop_shap].quantile(0.95)
    )
)
sm._A = []
cbar = plt.colorbar(sm, cax=cax)
cbar.set_label("SHAP Value of Shop Count", fontsize=12)

ax.set_title("SHAP Bubble Map: Shop Counts", fontsize=14)
ax.axis("off")
plt.tight_layout()
plt.show()
```

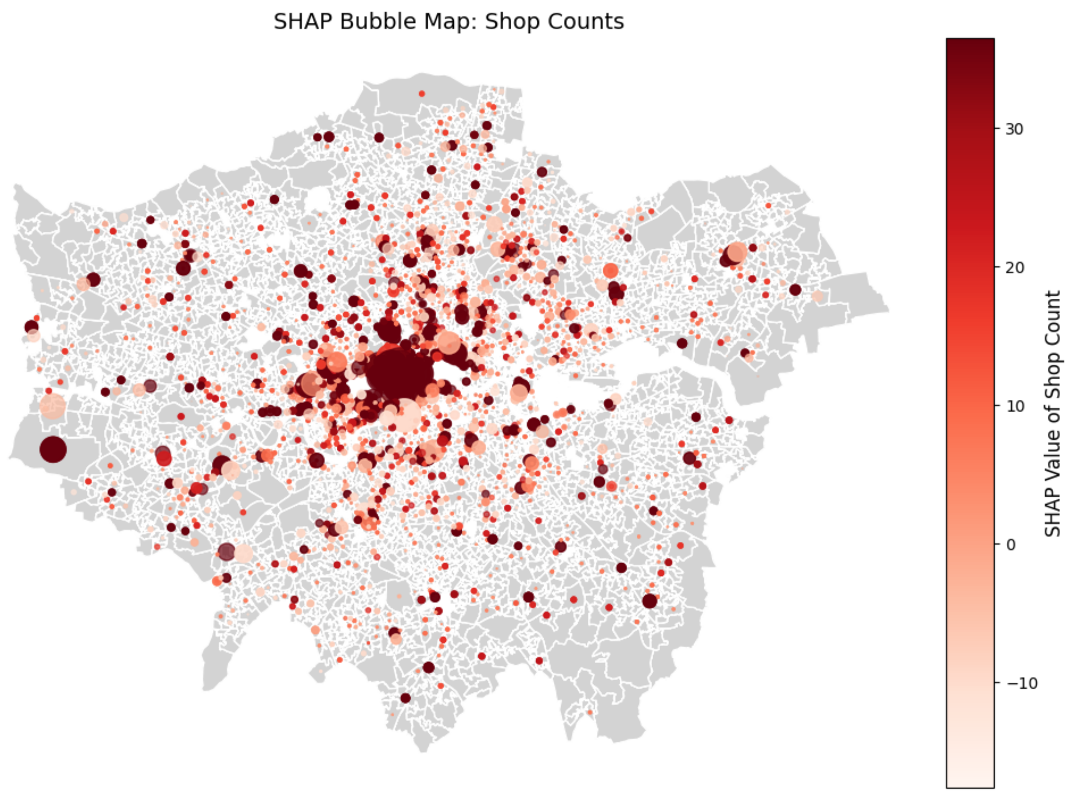

Figure 12. Distribution of SHAP Values of Shop Counts

Analyzing the SHAP values of individual features within a single LSOA can provide deeper insights into their local influence. As shown in Figure 13 and 14, LSOA with the highest prediction is mainly pushed up by low income and the lack of accessibility of housing and local services. On the contrast, lowest prediction area is mainly driven down by shop counts.

[27]:
```
# get the id of the highest prediction area
max_pred_idx = y_pred_xgb.argmax()
# get the id of the lowest prediction area
min_pred_idx = y_pred_xgb.argmin()
```

[28]:
```
shap.plots.waterfall(shap_values[max_pred_idx])
```

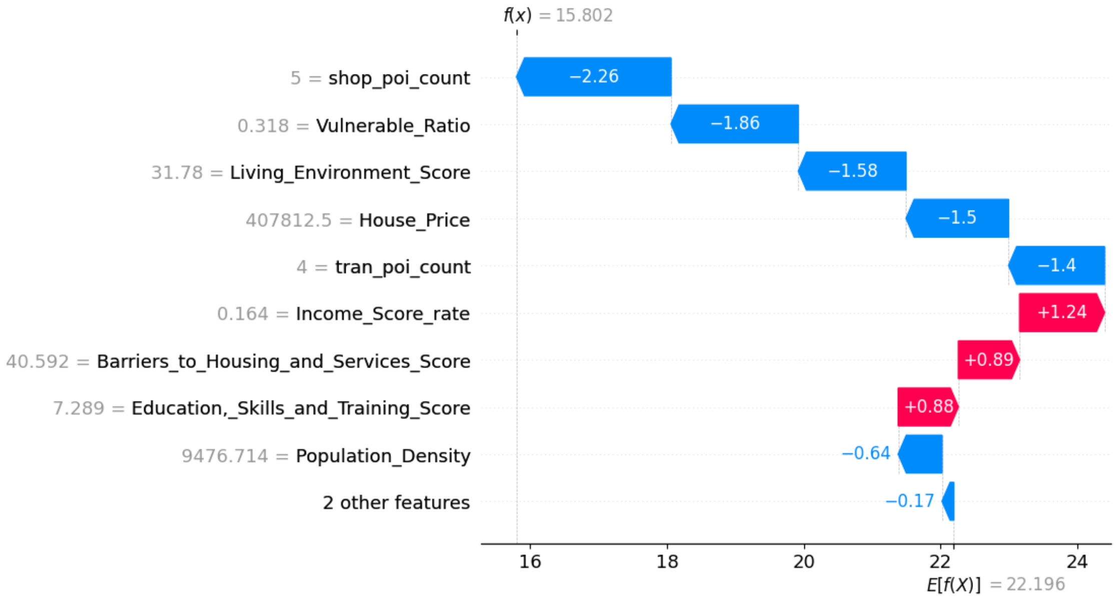

Figure 13. SHAP Force Plot for Highest Prediction

[29]:
```
shap.plots.waterfall(shap_values[min_pred_idx])
```

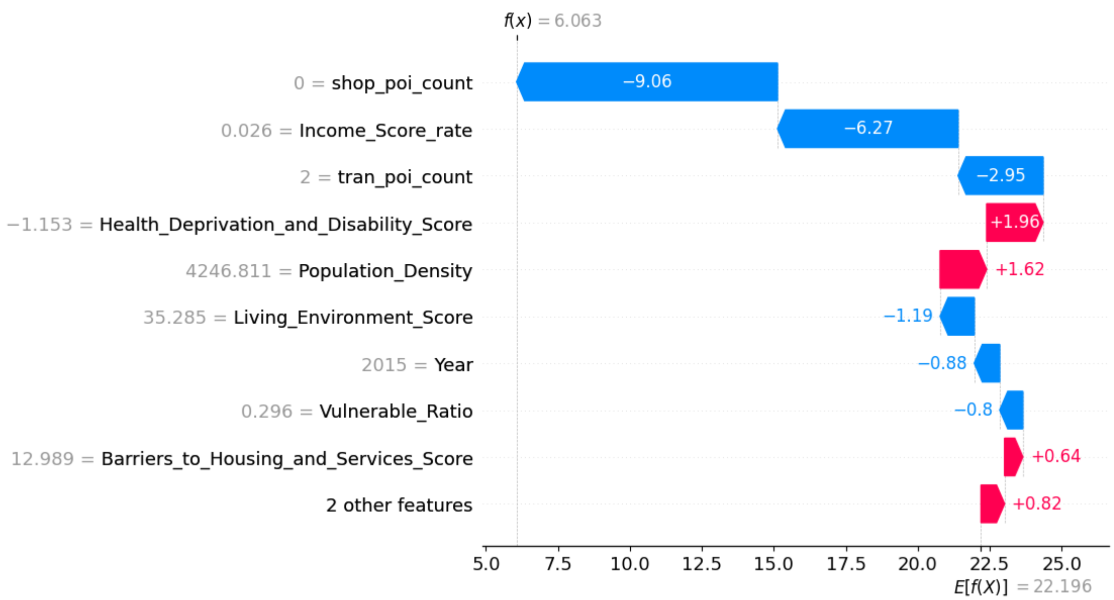

Figure 14. SHAP Force Plot for Lowest Prediction

[go back to the top]

1.8 Discussion

Several limitations remain in the study: - The absence of log transformation in the OLS model may have lead to reducing model robustness and underestimating the effects of skewed variables. - Despite the interpretability of SHAP values, XGBoost model remains limited in its capacity to quantify the causal impact of individual features on theft rates. - Models do not explicitly incorporate temporal dynamics, which may limit their ability to capture changes in feature importance or theft patterns over time.

[go back to the top] ***

1.9 Conclusion

This research has carried two different models to predict the theft rate across London areas, and tried to explain the driving factors.

In conclusion, ML model especially tree-based model, are more suitable for crime prediction tasks involving non-linear and complex interactions among variables. It not only provides a more accurate prediction, but also gives an in-depth explanation of the impact factors.

Generally, the number of shops and the porportion of both young and elderly in an area affect the theft rate more than other elements. However, different areas have different causes. It's important to have targeted policies and management measures.

[go back to the top]

---

[30]:
```
end_time = time.time()
print(f"Total runtime: {(end_time - start_time)/60:.2f} minutes")
```

```
Total runtime: 4.59 minutes
```

1.10 References

Boeing, G. 2024. "Modeling and Analyzing Urban Networks and Amenities with OSMnx." Working paper. URL: https://geoffboeing.com/publications/osmnx-paper/

Cohen, L. E. and Felson, M. (1979) 'Social change and crime rate trends: A routine activity approach', American Sociological Review, 44(4), pp. 588–608. doi: 10.2307/2094589.

Glasson, J. and Cozens, P. (2011) 'Making communities safer from crime: An undervalued element in impact assessment', Environmental Impact Assessment Review, 31(1), pp. 25–35. doi: 10.1016/j.eiar.2010.03.007.

Hill, D. (2025) 'London's recent rises in crimes of theft - some three-year patterns and trends', OnLondon, 8 February. Available at: https://www.onlondon.co.uk/londons-recent-rises-in-crimes-of-theft-some-three-year-patterns-and-trends/ (Accessed: 14 April 2025).

Junxiang Yin (2022) 'Crime prediction methods based on machine learning: A survey', Computers, Materials and Continua, 74(2), pp. 4601–4629. doi: 10.32604/cmc.2023.034190.

Mandalapu, V. et al. (2023) 'Crime prediction using machine learning and deep learning: A systematic review and future directions', IEEE Access, 11, pp. 60153–60170. doi: 10.1109/ACCESS.2023.3286344.

Retzlaff, C. O. et al. (2024) 'Post-hoc vs ante-hoc explanations: XAI design guidelines for data scientists', Cognitive Systems Research, 86, p. 101243. doi: 10.1016/j.cogsys.2024.101243.

Yunus, A. and Loo, J. (2024) 'London street crime analysis and prediction using crowd-sourced dataset', Journal of Computational Mathematics and Data Science, 10, p. 100089. doi: 10.1016/j.jcmds.2023.100089.

Zhang, X. et al. (2022) 'Interpretable machine learning models for crime prediction', Computers, Environment and Urban Systems, 94, p. 101789. doi: 10.1016/j.compenvurbsys.2022.101789.

[go back to the top]