The pattern of the spatial distribution of road safety inequalities in the London Boroughs for a vulnerable group comprising older people (60 + years) and children (0 - 15 years) and the extent to which indices of deprivation influence the road safety inequalities suffered by this vulnerable group

Student Number: 23191686

Word Count: 2056

1. Introduction

On the Earth, 1.3 million people die each year caused by traffic accidents, which account for a significant proportion of total deaths (Leijdesdorff et al., 2020; Sun et al., 2021). According to Kluczyńska et al. (2024), vulnerable groups typically include the elderly and children by definition. Leijdesdorff et al. (2020) find in their study that the average age of those killed by cyclists was 71.1 years. So the elderly as a group need to be given special attention in traffic safety protection. O'Toole and Christie (2018) identify that children as a group have a higher proportion of injuries in pedestrian traffic accidents than any other group. The children's group was three times more likely to be involved in serious road accidents in more deprived areas than in less deprived areas (O'Toole and Christie, 2018). Therefore, it is meaningful to research the inequalities in traffic accident casualty rates in different deprived severity areas for a vulnerable group comprising elderly people and children.

The study area for this essay is London, UK, and the study subject is a vulnerable group comprising children aged 0-15 years and older people aged 65 years and over. This essay will first investigate the pattern of spatial distribution of inequality in the traffic accident casualty rates at the London Boroughs study scale. It will then study to what extent Indices of Deprivation influence road safety inequalities for this vulnerable group at the London Boroughs and London LSOAs research scales. Finally, based on the above investigation and research, possible recommendations for mitigating traffic safety inequalities for vulnerable groups in London are proposed.

2. Literature review

The K-Mean clustering algorithm and Global Moran's I spatial autocorrelation algorithm could be used to research the spatial patterns of traffic accident inequalities. According to Sitohang et al. (2020), the K-Mean clustering algorithm is used to identify different severity of traffic accidents, which could contribute to the police's goal of reducing the number of traffic accidents. Hazaymeh et al. (2022) investigate the temporal and spatial distribution patterns of traffic accidents for the Irbid Governorate, Jordan using Global Moran's I spatial autocorrelation algorithm, which concludes that traffic accidents are spatially clustered in their study area.

Based on the research of O'Toole and Christie (2018), the spatial inequalities in traffic accident rates for groups of children are related to the environmental deprivation Index, the economic income deprivation Index and other Indices of Deprivation, and their study is scaled to individuals' postcodes, which makes it very micro-level. However, policy development and implementation may need to be based on more macro-level studies such as London Boroughs or London LSOAs. Therefore, micro-level research constitutes a possible research limitation.

Multiple linear regression models and random forest classification prediction models allow to study to what extent which factors are correlated with the spatial distribution of inequalities in road traffic accident casualty rates. Kourouma et al. (2019) use Ordinary Least Squares (OLS) regression to analyse trends in road traffic accidents in Guinea between 2015 and 2017, focusing on the factors that influence the occurrence of these accidents. Khanum et al. (2023) employ the Random Forest Classification algorithm to predict different levels of severity of road traffic accidents and also to assess the importance of explanatory variables.

3. Research question

Research questions: What are the spatial patterns of road accidents for children (0 - 15 years) and old people (more than or equal to 65 years) living in London at the Boroughs level? To what extent do these deprivation indices in IMD influence road safety inequalities of children (0 - 15 years) and the elderly (60 +)?

As we discussed in 1. Introduction, the vulnerable group consisting of older people and children is high risk of being injured in traffic accidents. Therefore it will be meaningful to study the pattern of spatial distribution of road safety inequalities experienced by this vulnerable group in the London Boroughs and the extent to which indices of deprivation influence the road safety inequalities.

The specific research objectives are:

(1) Analysing the characteristics of spatial distribution inequalities in traffic accident casualty rates suffered by the vulnerable group, which consists of children aged 0-15 years old and elderly people aged 65 years old and above, under the London Boroughs research scale.

(2) Studying the extent to which 'Income Deprivation index', 'Employment Deprivation index', 'Education, Skills and Training Deprivation index', 'Health Deprivation index', 'Crime Deprivation index', 'Barriers to Housing and Services Deprivation index' and 'Living Environment Deprivation index' in the Indices of Deprivation in London Boroughs and London LSOAs research scales influence inequalities in the traffic accidents calsulty rates.

(3) Proposing possible recommendations that might mitigate traffic safety inequalities for this vulnerable group.

4. Presentation of data

This section will be divided into three parts: data introduction, data pre-processing and data presentation

4.1. Data introduction

'Road traffic safety data', 'Population data', 'Indices of deprivation' data, 'Base map of London Boroughs and London Living Standards Survey Areas' data are described in Table 4.1. below:

In [1]:
```
# for printing tables in more academic styles
from tabulate import tabulate
```

In [2]:
```
def wrap_text(text, width):
    """
    Wraps text for each cell in the table to ensure it fits within a specified width.
    """
    import textwrap
    return '\n'.join(textwrap.wrap(text, width))
```

In [3]:
```
# Table 4.1. Data introduction
# set the terminal width
terminal_width = 91
title_table_41 = "Table 4.1. Data introduction"
# print the table title and the table
print(title_table_41.center(terminal_width))

# Table 4.1. (1) Road traffic safety data
# table title of table 4.1. Data introduction
# table data where long text has been handled with line breaks by the wrap_text function
data_411 = [
```

```
    ["Name:", "Road traffic safety data"],
    ["Date:", "2019"],
    ["Source:",
     wrap_text("Transport for London (https://tfl.gov.uk/cdn/static/cms/documents/2019-gla-data
    ["Useful fields included:", wrap_text("'Easting', 'Northing', 'Casualty Age (Banded)', 'No.
]
# table title
title_table_411 = "Table 4.1. (1) Road traffic safety data"
# generate tables
table_411 = tabulate(data_411, headers='', tablefmt='simple_grid')
table_411_lines = table_411.split('\n')
# centre the table
centered_table_411 = "\n".join(
    line.center(terminal_width) for line in table_411_lines)
# print the table title and the table
print(title_table_411.center(terminal_width))
print(centered_table_411)

# Table 4.1. (2) Population data
# table data where long text has been handled with line breaks by the wrap_text function
data_412 = [
    ["Name:", "Population data"],
    ["Date:", "2019"],
    ["Source:",
     wrap_text("Office for National Statistics(https://www.ons.gov.uk/file?uri=/peoplepopulatio
    ["Useful fields included:", wrap_text("'LSOA Code', 'LSOA NAME', 'Mid-2019 population'", 40
]
# table title
title_table_412 = "Table 4.1. (2) Population data"
# generate tables
table_412 = tabulate(data_412, headers='', tablefmt='simple_grid')
table_412_lines = table_412.split('\n')
# centre the table
centered_table_412 = "\n".join(
    line.center(terminal_width) for line in table_412_lines)
# print the table title and the table
print(title_table_412.center(terminal_width))
print(centered_table_412)

# Table 4.1. (3) Indices of Deprivation
# set the terminal width
# table data where long text has been handled with line breaks by the wrap_text function
data_413 = [
    ["Name:", "London Datastore"],
    ["Date:", "2019"],
    ["Source:",
     wrap_text(
         "London Datastore(https://data.london.gov.uk/download/indices-of-deprivation/9ee0cf66-
    ["Useful fields included:",
     "Useful sheet 1 - 'Borough domain summaries'\n"+"(Useful fields included):"+
               "\n'Local Authority District code (2019)',\n"+
               "'Local Authority District name (2019)',\n"+
               "'Income - Average score',\n"+
               "'Employment - Average score',\n"+
               "'Education, Skills and Training - Average score',\n"+
               "'Health Deprivation and Disability - Average score',\n"+
               "'Crime - Average score',\n"+
               "'Barriers to Housing and Services - Average score',\n"+
               "'Living Environment - Average score';\n"+
               "\nUseful sheet 2 - 'IMD 2019'\n"+"(Useful fields included):\n"+
               "'LSOA code (2011)',\n"+
               "'Income Score (rate)',\n"+
               "'Employment Score (rate)',\n"+
               "'Education, Skills and Training Score',\n"+
               "'Health Deprivation and Disability Score',\n"+
               "'Crime Score',\n"+
               "'Barriers to Housing and Services Score',\n"+
```

```
                "'Living Environment Score'"]
]
# table title
title_table_413 = "Table 4.1. (3) Indices of Deprivation"
# generate tables
table_413 = tabulate(data_413, headers='', tablefmt='simple_grid')
table_413_lines = table_413.split('\n')
# centre the table
centered_table_413 = "\n".join(
    line.center(terminal_width) for line in table_413_lines)
# print the table title and the table
print(title_table_413.center(terminal_width))
print(centered_table_413)

# Table 4.1. (4) Basemaps of London Boroughs and London LSOAs
# set the terminal width
# table data where long text has been handled with line breaks by the wrap_text function
data_414 = [
    ["Name:", "Basemaps of London Boroughs and London LSOAs"],
    ["Date:", "2014"],
    ["Source:",
     wrap_text(
         "London Datastore (https://data.london.gov.uk/download/statistical-gis-boundary-files-
    ["London Boroughs basemap\n"+"(Useful fields included):",
     wrap_text("'Easting', 'Northing', 'Casualty Age (Banded)', 'No. of Casualties'", 41)],
    ["London LSOAs basemap\n"+"(Useful fields included)",
     wrap_text("'LSOA11CD', 'LSOA11NM', 'LAD11CD', 'LAD11NM', 'geometry'", 41)]
]
# table title
title_table_414 = "Table 4.1. (4) Basemaps of London Boroughs and London LSOAs"
# generate tables
table_414 = tabulate(data_414, headers='', tablefmt='simple_grid')
table_414_lines = table_414.split('\n')
# centre the table
centered_table_414 = "\n".join(
    line.center(terminal_width) for line in table_414_lines)
# print the table title and the table
print(title_table_414.center(terminal_width))
print(centered_table_414)
```

Table 4.1. Data introduction

Table 4.1. (1) Road traffic safety data

Name:	Road traffic safety data
Date:	2019
Source:	Transport for London (https://tfl.gov.uk/cdn/static/cms/documents/2019-gla-data-extract-casualty.csv)
Useful fields included:	‘Easting’, ‘Northing’, ‘Casualty Age (Banded)’, ‘No. of Casualties’

Table 4.1. (2) Population data

Name:	Population data
Date:	2019
Source:	Office for National Statistics(https://www.ons.gov.uk/file?uri=/peoplepopulationandcommunity/populationandmigration/populationestimates/datasets/lowersuperoutputareapopulationdensity/mid2019sape22dt11/sape22dt11mid2019lsoapopulationdensity.zip)
Useful fields included:	‘LSOA Code’, ‘LSOA NAME’, ‘Mid-2019 population’

Table 4.1. (3) Indices of Deprivation

Name:	London Datastore
Date:	2019
Source:	London Datastore(https://data.london.gov.uk/download/indices-of-deprivation/9ee0cf66-e6f9-4e38-8eec-79c1d897e248/ID%202019%20for%20London.xlsx)
Useful fields included:	Useful sheet 1 - ‘Borough domain summaries’ (Useful fields included): ‘Local Authority District code (2019)’, ‘Local Authority District name (2019)’, ‘Income - Average score’, ‘Employment - Average score’, ‘Education, Skills and Training - Average score’, ‘Health Deprivation and Disability - Average score’, ‘Crime - Average score’, ‘Barriers to Housing and Services - Average score’, ‘Living Environment - Average score’; Useful sheet 2 - ‘IMD 2019’ (Useful fields included): ‘LSOA code (2011)’, ‘Income Score (rate)’, ‘Employment Score (rate)’, ‘Education, Skills and Training Score’, ‘Health Deprivation and Disability Score’, ‘Crime Score’, ‘Barriers to Housing and Services Score’, ‘Living Environment Score’

Table 4.1. (4) Basemaps of London Boroughs and London LSOAs

Name:	Basemaps of London Boroughs and London LSOAs
Date:	2014
Source:	London Datastore (https://data.london.gov.uk/download/statistical-gis-boundary-files-london/9ba8c833-6370-4b11-abdc-314aa020d5e0/statistical-gis-boundaries-london.zip)
London Boroughs basemap (Useful fields included):	'Easting', 'Northing', 'Casualty Age (Banded)', 'No. of Casualties'
London LSOAs basemap (Useful fields included)	''LSOA11CD', 'LSOA11NM', 'LAD11CD', 'LAD11NM', 'geometry'

4.2. Data pre-processing

Please see the code comments.

4.2.1. Reading data online

In [4]:
```
import geopandas as gpd
import pandas as pd
# for converting the DataFrame to GeoDataFrame
from shapely.geometry import Point
# For plotting
import matplotlib.pyplot as plt
```

4.2.1.1. Road safety data:

In [5]:
```
# read the road safety data online
df_road_safety_raw = pd.read_csv(
    "https://github.com/Jinting914113/CASA0006_Data_Science_for_Spatial_Systems_Assessment/raw/
    na_values=[" "],
    low_memory=False,
)
```

In [6]:
```
# converting the DataFrame 'df_road_safety_raw'to GeoDataFrame 'gdf_road_safety_raw'
# convert Easting and Northing into points
df_road_safety_raw['geometry'] = df_road_safety_raw.apply(
    lambda row: Point(row['Easting'], row['Northing']), axis=1
)

# creating a GeoDataFrame with Points
gdf_road_safety_raw = gpd.GeoDataFrame(df_road_safety_raw, geometry='geometry')

# set the coordinate reference system of the GeoDataFrame to BNG (EPSG:27700)
gdf_road_safety_raw = gdf_road_safety_raw.set_crs("EPSG:27700", inplace=True)
```

In [7]:
```
# gdf_road_safety_raw.head(3)
```

4.2.1.2. Population data:

In [8]:
```
# for reading file
import zipfile
import requests
from io import BytesIO
```

In [9]:
```
# download zip file
url = 'https://www.ons.gov.uk/file?uri=/peoplepopulationandcommunity/populationandmigration/pop
response = requests.get(url)

# unzip the zip file and read the Excel file
with zipfile.ZipFile(BytesIO(response.content), 'r') as z:
    file_list = z.namelist()
    excel_file = [f for f in file_list if f.endswith('.xlsx')][0]
    with z.open(excel_file) as f:
        # read the specified worksheet in the Excel file and skip the first four rows, labellin
        df_pop_raw = pd.read_excel(f, sheet_name='Mid-2019 Population Density', header=4)
```

In [10]:
```
# display data
# df_pop_raw.head()
```

4.2.1.3. Indices of Deprivation data:

In [11]:
```
# read excel file of Indices of Deprivation
excel_file = "https://data.london.gov.uk/download/indices-of-deprivation/9ee0cf66-e6f9-4e38-8ee
df_id_bor_raw = pd.read_excel(excel_file, sheet_name='Borough domain summaries')
df_id_lsoa_raw = pd.read_excel(excel_file, sheet_name='IMD 2019')
```

In [12]:
```
# df_id_bor_raw.info()
# df_id_lsoa_raw.info()
```

4.2.1.4. London Boroughs and London LSOAs:

In [13]:
```
%%capture captured_output
# the url of London Boroughs basemap
url = "https://data.london.gov.uk/download/statistical-gis-boundary-files-london/9ba8c833-6370-

! wget $url

# hide the output or use the next line to show it
# captured_output.show()
```

In [14]:
```
# read the basemap of London Boroughs
gdf_bor_raw = gpd.read_file(
    f"zip://statistical-gis-boundaries-london.zip!statistical-gis-boundaries-london/ESRI/London
# read the basemap of London LSOAs
gdf_lsoa_raw = gpd.read_file(
    f"zip://statistical-gis-boundaries-london.zip!statistical-gis-boundaries-london/ESRI/LSOA_2
```

In [15]:
```
# gdf_bor_raw.info()
# gdf_lsoa_raw.info()
```

In [16]:
```
# gdf_bor_raw.info()
```

4.2.2. Pre-processing of raw data into processed data

4.2.1.1. Population:

In [17]:
```
# add the new 'LSOA Code' and 'Mid-2019 population' column to gdf_lsoa_raw GeoDataFrame
gdf_pop_lsoa = gdf_lsoa_raw.merge(df_pop_raw[['LSOA Code', 'Mid-2019 population']],
                                  left_on='LSOA11CD', right_on='LSOA Code', how='left')
```

In [18]:
```
# gdf_pop_lsoa.info()
```

In [19]:
```
# filter 'LSOA11CD' and 'Mid-2019 population' columns from gdf_pop_lsoa
df_pop_lsoa = gdf_pop_lsoa[['LSOA11CD','Mid-2019 population']].copy()
```

```
# rename the 'Mid-2019 population' column to 'pop'.
df_pop_lsoa = df_pop_lsoa.rename(columns={'Mid-2019 population': 'pop'})
```

In [20]:
```
# df_pop_lsoa.info()
```

In [21]:
```
# filter 'LAD11CD','LAD11NM', 'geometry' and 'Mid-2019 population' columns from gdf_pop_lsoa
df_pop_bor_raw = gdf_pop_lsoa[['LAD11CD','LAD11NM', 'geometry', 'Mid-2019 population']].copy()
```

In [22]:
```
# df_pop_bor_raw.info()
```

In [23]:
```
# sum the 'Mid-2019 population' columns according to 'LAD11CD'
df_pop_bor = df_pop_bor_raw.groupby(['LAD11CD', 'LAD11NM'])['Mid-2019 population'].sum().reset_
```

In [24]:
```
# df_pop_bor.info()
```

4.2.1.2. Road safety data:

In [25]:
```
# define filter conditions
# age_band_condition = gdf_road_safety['Casualty Age (Banded)'].isin(['0-15', '16-24', '60+'])
age_band_condition = gdf_road_safety_raw['Casualty Age (Banded)'].isin(['0-15', '60+'])
# apply filter conditions
gdf_road_safety_filtered = gdf_road_safety_raw[(age_band_condition)].reset_index().copy()
```

In [26]:
```
# filter 'Borough','No. of Casualties' and 'geometry' columns from gdf_road_safety_filtered
gdf_road_safety_filtered = gdf_road_safety_filtered[['Borough','No. of Casualties', 'geometry']
```

In [27]:
```
# gdf_road_safety_filtered.info()
```

In [28]:
```
# convert the coordinate system of gdf_bor_raw to EPSG:27700
gdf_bor_raw = gdf_bor_raw.to_crs('EPSG:27700')
```

In [29]:
```
# gdf_bor_raw.crs
```

In [30]:
```
# Use the sjoin() function to add point data to polygons data
gdf_road_safety_bor = gpd.sjoin(gdf_bor_raw, gdf_road_safety_filtered, how='left', predicate='i
```

In [31]:
```
# gdf_road_safety_bor.info()
```

In [32]:
```
# sum the 'No. of Casualties' column from the 'GSS_CODE' column
casualties_sum = gdf_road_safety_bor.groupby('GSS_CODE')['No. of Casualties'].sum().reset_index

# rename the new column name to 'amount_of_casualties'
casualties_sum = casualties_sum.rename(columns={'No. of Casualties': 'amount_of_casualties'})

# remove duplicate 'GSS_CODE' column rows, keep the first one
gdf_road_safety_bor_unique = gdf_road_safety_bor.drop_duplicates('GSS_CODE')

# merge the summed results into the original data
gdf_road_safety_bor_unique = gdf_road_safety_bor_unique.merge(casualties_sum, on='GSS_CODE', ho

# select the columns to be retained
columns_to_keep = ['GSS_CODE', 'NAME', 'geometry', 'amount_of_casualties']
gdf_road_safety_bor = gdf_road_safety_bor_unique[columns_to_keep]
```

In [33]:
```
# gdf_road_safety_bor.info()
```

In [34]:
```
# convert the coordinate system of gdf_bor_raw to EPSG:27700
gdf_lsoa_raw = gdf_lsoa_raw.to_crs('EPSG:27700')
```

In [35]:
```
# Use the sjoin() function to add point data to polygons data
gdf_road_safety_lsoa = gpd.sjoin(gdf_lsoa_raw, gdf_road_safety_filtered, how='left', predicate=
```

In [36]:
```
# gdf_road_safety_lsoa.info()
```

In [37]:
```
# sum the 'No. of Casualties' column from the 'GSS_CODE' column
casualties_sum_lsoa = gdf_road_safety_lsoa.groupby('LSOA11CD')['No. of Casualties'].sum().reset
```

In [38]:
```
# rename the new column name to 'amount_of_casualties'
casualties_sum_lsoa = casualties_sum_lsoa.rename(columns={'No. of Casualties': 'amount_of_casua

# remove duplicate 'GSS_CODE' column rows, keep the first one
gdf_road_safety_lsoa_unique = gdf_road_safety_lsoa.drop_duplicates('LSOA11CD')

# merge the summed results into the original data
gdf_road_safety_lsoa_unique = gdf_road_safety_lsoa_unique.merge(casualties_sum_lsoa, on='LSOA11

# select the columns to be retained
columns_to_keep_lsoa = ['LSOA11CD', 'LSOA11NM', 'geometry', 'amount_of_casualties']
gdf_road_safety_lsoa = gdf_road_safety_lsoa_unique[columns_to_keep_lsoa]
```

In [39]:
```
# gdf_road_safety_lsoa.to_csv('gdf_road_safety_lsoa.csv', index=False)
```

In [40]:
```
# gdf_road_safety_lsoa.info()
```

4.2.1.3. Indices of Deprivation data:

In [41]:
```
#df_id_bor_raw
#df_id_lsoa_raw
```

In [42]:
```
# select the columns to be retained
columns_to_keep_id_bor = [
    'Local Authority District code (2019)',
    'Local Authority District name (2019)',
    'Income - Average score',
    'Employment - Average score',
    'Education, Skills and Training - Average score',
    'Health Deprivation and Disability - Average score',
    'Crime - Average score','Barriers to Housing and Services - Average score',
    'Living Environment - Average score']
columns_to_keep_id_lsoa = [
    'LSOA code (2011)',
    'LSOA name (2011)',
    'Income Score (rate)',
    'Employment Score (rate)',
    'Education, Skills and Training Score',
    'Health Deprivation and Disability Score',
    'Crime Score',
    'Barriers to Housing and Services Score',
    'Living Environment Score']
df_id_bor_filtered = df_id_bor_raw[columns_to_keep_id_bor].copy()
df_id_lsoa_filtered = df_id_lsoa_raw[columns_to_keep_id_lsoa].copy()
```

4.2.3. Pre-processing of data for research qustion 1 and research question 2

4.2.3.1. Pre-processing of data for research qustion 1

In [43]:
```
# add the 'LAD11CD' and 'Mid-2019 population' columns to gdf_road_safety_bor GeoDataFrame
gdf_road_safety_bor = gdf_road_safety_bor.merge(df_pop_bor[['LAD11CD', 'Mid-2019 population']],
                                  left_on='GSS_CODE', right_on='LAD11CD', how='left')
```

In [44]:
```
# gdf_road_safety_bor.info()
```

In [45]:
```
gdf_road_safety_bor['casualty_rates'] = (gdf_road_safety_bor['amount_of_casualties'] /
                                         gdf_road_safety_bor['Mid-2019 population']) * 1000
```

4.2.3.2. Pre-processing of data for research qustion 2

In [46]:
```
# add the 'pop' column to gdf_road_safety_bor GeoDataFrame
gdf_road_safety_lsoa = gdf_road_safety_lsoa.merge(df_pop_lsoa[['LSOA11CD', 'pop']],
                                    left_on='LSOA11CD', right_on='LSOA11CD', how='left')
```

In [47]:
```
# gdf_road_safety_lsoa.info()
```

In [48]:
```
# add the 'GSS_CODE' and 'casualty_rates' columns to df_id_bor_filtered DataFrame
df_bor = df_id_bor_filtered.merge(gdf_road_safety_bor[['GSS_CODE', 'casualty_rates']],
                                    left_on='Local Authority District code (2019)',
                                    right_on='GSS_CODE', how='left')
```

In [49]:
```
# df_bor.info()
```

In [50]:
```
gdf_road_safety_lsoa['casualty_rates'] = (gdf_road_safety_lsoa['amount_of_casualties'] /
                                         gdf_road_safety_lsoa['pop']) * 1000
```

In [51]:
```
# add the 'LSOA11CD' and 'casualty_rates' columns to df_id_lsoa_filtered DataFrame
df_lsoa = df_id_lsoa_filtered.merge(gdf_road_safety_lsoa[['LSOA11CD', 'casualty_rates']],
                                    left_on='LSOA code (2011)',
                                    right_on='LSOA11CD', how='left')
```

In [52]:
```
# df_lsoa.info()
```

4.3. Data presentation

Data presentation at the scale of London Boroughs and London LSOAs separately:

In [53]:
```
# Through the previous inspection, it was found that the 'Casualty Rates' field in df_bor has a
# which may cause an error for clustering, so it is deleted.
max_value_bor = gdf_road_safety_bor['casualty_rates'].max()
gdf_road_safety_bor_no_max = gdf_road_safety_bor[gdf_road_safety_bor['casualty_rates'] != max_v
```

In [54]:
```
# Histograms of description of data (plotting)

# create subplot
fig, ax = plt.subplots(3, 3, figsize=(11, 5.9))

# Plotting the histogram
ax[0,0].hist(df_bor.casualty_rates, bins=9, color='skyblue', edgecolor='black', alpha=0.7)
ax[0,1].hist(df_bor['Income - Average score'], bins=9, color='skyblue', edgecolor='black', alph
ax[0,2].hist(df_bor['Employment - Average score'], bins=7, color='skyblue', edgecolor='black',
ax[1,0].hist(df_bor['Education, Skills and Training - Average score'], bins=8, color='skyblue',
ax[1,1].hist(df_bor['Health Deprivation and Disability - Average score'], bins=11, color='skybl
ax[1,2].hist(df_bor['Crime - Average score'], bins=7, color='skyblue', edgecolor='black', alpha
ax[2,0].hist(df_bor['Barriers to Housing and Services - Average score'], bins=8, color='skyblue
ax[2,1].hist(df_bor['Living Environment - Average score'], bins=11, color='skyblue', edgecolor=
ax[2,2].hist(gdf_road_safety_bor_no_max.casualty_rates, bins=7, color='skyblue', edgecolor='bla

# Adding labels
ax[0,0].set_xlabel('Casualty rates', fontsize=10, labelpad=1)
ax[0,0].set_ylabel('Frequency', fontsize=11, labelpad=0.5)
ax[0,1].set_xlabel('Income', fontsize=10, labelpad=1)
ax[0,1].set_ylabel('Frequency', fontsize=11, labelpad=0.5)
ax[0,2].set_xlabel('Employment', fontsize=10, labelpad=1)
```

```
ax[0,2].set_ylabel('Frequency', fontsize=11, labelpad=0.5)
ax[1,0].set_xlabel('Education, Skills and Training', fontsize=10, labelpad=1)
ax[1,0].set_ylabel('Frequency', fontsize=11, labelpad=5)
ax[1,1].set_xlabel('Health)', fontsize=10, labelpad=1)
ax[1,1].set_ylabel('Frequency', fontsize=11, labelpad=0.5)
ax[1,2].set_xlabel('Crime)', fontsize=10, labelpad=1)
ax[1,2].set_ylabel('Frequency', fontsize=11, labelpad=0.5)
ax[2,0].set_xlabel('Barriers to Housing and Services', fontsize=10, labelpad=1)
ax[2,0].set_ylabel('Frequency', fontsize=11, labelpad=5)
ax[2,1].set_xlabel('Living Environment', fontsize=10, labelpad=1)
ax[2,1].set_ylabel('Frequency', fontsize=11, labelpad=0.5)
ax[2,2].set_xlabel('Casualty rates (no max)', fontsize=10, labelpad=1)
ax[2,2].set_ylabel('Frequency', fontsize=11, labelpad=0.5)

fig.text(0.46, -0.0,
         'Figure 4.3.1. Presentation of data at the scale of London Boroughs',
         ha='center', va='bottom', fontsize=13)

# Adding grid lines (optional)
ax[0,0].grid(True)
ax[0,1].grid(True)
ax[0,2].grid(True)
ax[1,0].grid(True)
ax[1,1].grid(True)
ax[1,2].grid(True)
ax[2,0].grid(True)
ax[2,1].grid(True)
ax[2,2].grid(True)

# plt.savefig('description.png', dpi=311, bbox_inches='tight')
plt.subplots_adjust(wspace=0.19, hspace=0.3)


# Display the plot
plt.show()
```

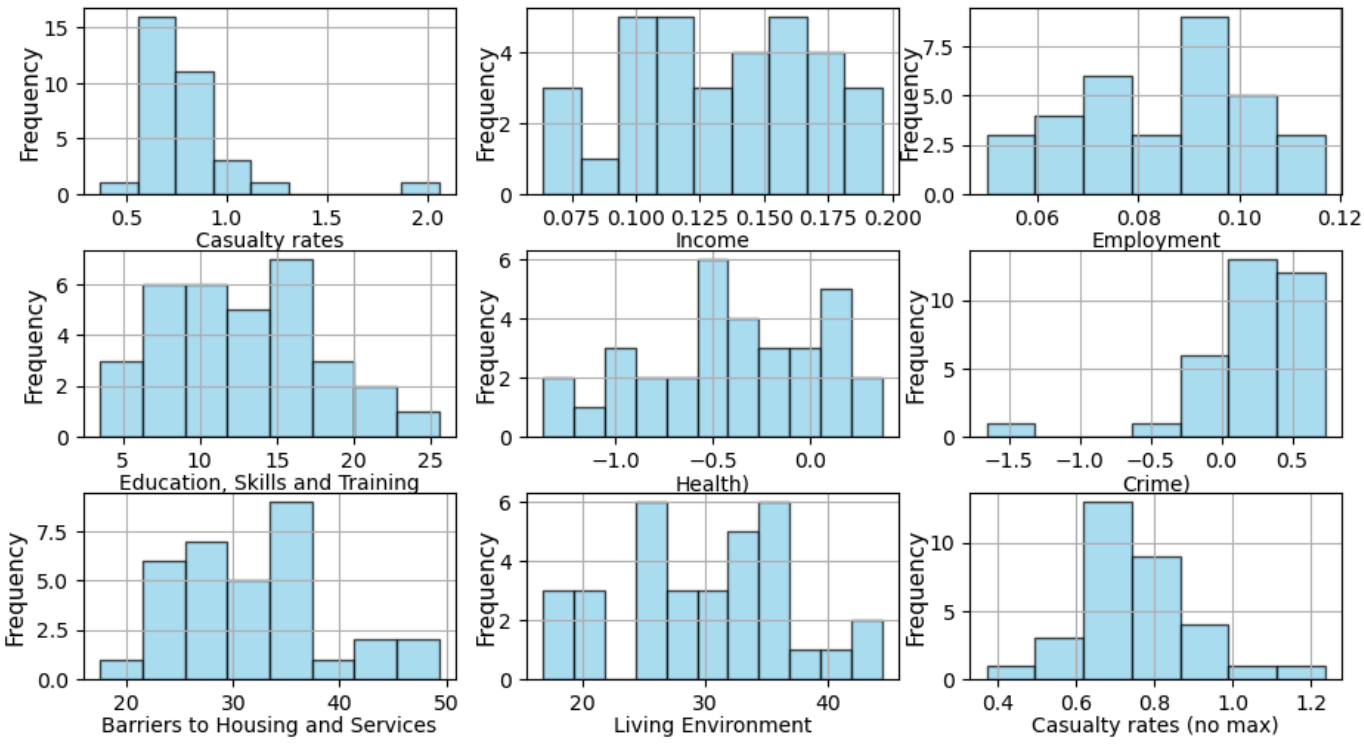

Figure 4.3.1. Presentation of data at the scale of London Boroughs

In [55]:
```
# Histograms of description of data (plotting)

# create subplot
fig, ax = plt.subplots(3, 3, figsize=(11, 5.9))
```

```
# Plotting the histogram
ax[0,0].hist(df_lsoa.casualty_rates, bins=91, color='skyblue', edgecolor='black', alpha=0.7)
ax[0,1].hist(df_lsoa['Income Score (rate)'], bins=91, color='skyblue', edgecolor='black', alpha
ax[0,2].hist(df_lsoa['Employment Score (rate)'], bins=91, color='skyblue', edgecolor='black', a
ax[1,0].hist(df_lsoa['Education, Skills and Training Score'], bins=91, color='skyblue', edgecol
ax[1,1].hist(df_lsoa['Health Deprivation and Disability Score'], bins=91, color='skyblue', edge
ax[1,2].hist(df_lsoa['Crime Score'], bins=91, color='skyblue', edgecolor='black', alpha=0.7)
ax[2,0].hist(df_lsoa['Barriers to Housing and Services Score'], bins=91, color='skyblue', edgec
ax[2,1].hist(df_lsoa['Living Environment Score'], bins=91, color='skyblue', edgecolor='black',

# hide unused subgraphs ax[2,2].
ax[2,2].set_visible(False) # make ax[2,2] invisible

# Adding labels
ax[0,0].set_xlabel('Casualty rates', fontsize=10, labelpad=1)
ax[0,0].set_ylabel('Frequency', fontsize=9, labelpad=0.5)
ax[0,1].set_xlabel('Income', fontsize=10, labelpad=1)
ax[0,1].set_ylabel('Frequency', fontsize=9, labelpad=0.5)
ax[0,2].set_xlabel('Employment', fontsize=10, labelpad=1)
ax[0,2].set_ylabel('Frequency', fontsize=9, labelpad=0.5)
ax[1,0].set_xlabel('Education, Skills and Training', fontsize=10, labelpad=1)
ax[1,0].set_ylabel('Frequency', fontsize=9, labelpad=5)
ax[1,1].set_xlabel('Health)', fontsize=10, labelpad=1)
ax[1,1].set_ylabel('Frequency', fontsize=9, labelpad=0.5)
ax[1,2].set_xlabel('Crime)', fontsize=10, labelpad=1)
ax[1,2].set_ylabel('Frequency', fontsize=9, labelpad=0.5)
ax[2,0].set_xlabel('Barriers to Housing and Services', fontsize=10, labelpad=1)
ax[2,0].set_ylabel('Frequency', fontsize=9, labelpad=5)
ax[2,1].set_xlabel('Living Environment', fontsize=10, labelpad=1)
ax[2,1].set_ylabel('Frequency', fontsize=9, labelpad=0.5)

# plt.title('Histogram of Residuals of the Linear Model')

fig.text(0.46, -0.0,
         'Figure 4.3.2. Presentation of data at the scale of London LSOAs',
         ha='center', va='bottom', fontsize=13)

# Adding grid lines (optional)
ax[0,0].grid(True)
ax[0,1].grid(True)
ax[0,2].grid(True)
ax[1,0].grid(True)
ax[1,1].grid(True)
ax[1,2].grid(True)
ax[2,0].grid(True)
ax[2,1].grid(True)

# plt.savefig('descriptio', dpi=311, bbox_inches='tight')
plt.subplots_adjust(wspace=0.19, hspace=0.3)


# Display the plot
plt.show()
```

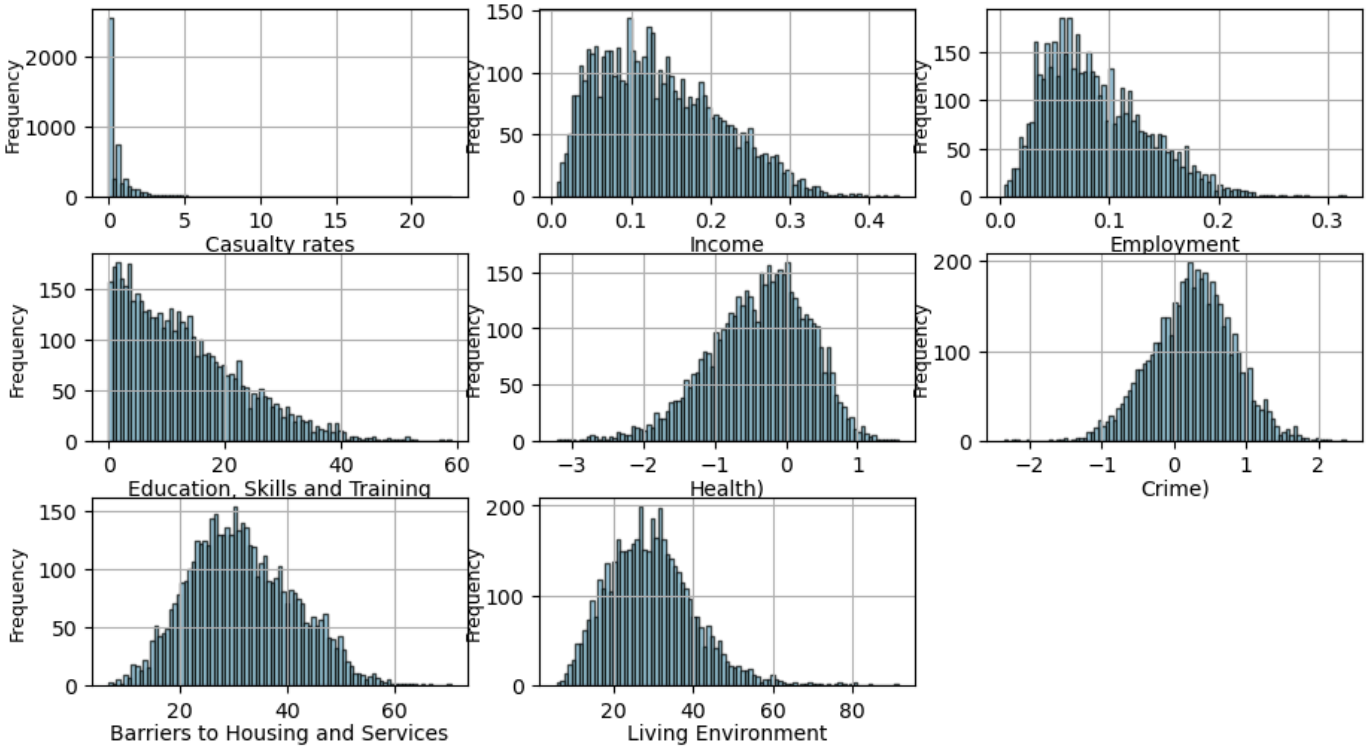

Figure 4.3.2. Presentation of data at the scale of London LSOAs

5. Methodology

In this section, the methodology used in the research of the research question 1 and research question 2 posed earlier in this essay would be discussed respectively.

5.1. Research question 1

The continuous variable of traffic accident casualty rates does not easily enable direct analysis of their spatial distribution patterns in a visual way. Therefore, we categorise the severity of traffic accident casualty rates into 2-4 clusters based on the data of traffic accident casualty rates in London Boroughs to be easier to compare. From 'Figure 4.3.1. Presentation of data at the scale of London Boroughs ', we could find that the raw data of traffic accident casualty rates has a maximum value which is much larger than the other values. As a result it could be labelled as 'Casualty rates are relatively high'. After removing this maximum value the K-Means clustering algorithm is used and the Silouette score is calculated for different numbers of clusters. This score reflects the quality of the clusters from -1 (very poor) to 1 (very strong) (Chen, 2024a). The number of clusters with the best clustering results is obtained by choosing the one with the highest score.

After visualising the results of the completed clustering, the global Moran I could be calculated.The purpose of this is to further identify whether inequality casualty rates are clustered, discrete or randomly distributed at the Borough level (Hazaymeh et al., 2022).

5.2. Research question 2

5.2.1. At the London Boroughs level

Before building the multiple regression model, the correlation matrix needs to be built and the VIF values of each variable need to be calculated to prevent multicollinearity from affecting the model fitting results. The independent variables with the VIF greater than 5 should be removed before proceeding with the model fitting.

After fitting the OLS model, the residuals of this model need to be analysed to verify the reliability of the regression model. A scatter plot of the residuals versus the fitted traffic accident casualty rate is produced to analyse whether there is heteroskedasticity in the model, a histogram of the model residuals is produced to see whether the residuals show a normal distribution, and a QQ plot of the residuals is produced to compare the theoretical distribution of the residuals with the residuals in the samples (Chen, 2024b). Finally, all the results obtained above are analysed.

5.2.2. At the London LSOAs level

Because it is too much time spent on predicting the model and getting the importance parameters by directly using the random forest regression algorithm. Using Random Forest classification algorithm could effectively avoid the problem of excessive computational time. Since the research question was to study the extent to which Indices of Deprivation correlates with inequality in casualty rates, classifying the casualty rates data into different levels of severity would also work out the final feature importance result. However, this result is certainly not as accurate as without clustering, and thus this is part of the limitations of this study.

The predictive dependent variable for the Random Forest classification algorithm needs to be categorical data, so firstly the traffic accident casualty rates data is categorised in terms of severity. With 'Figure 4.3.2. Presentation of data at the scale of London LSOAs' we could find that there are particularly high values of 0 for casualty rates, so 0 was categorised as the lowest severity category. Afterwards, the remaining data is calculated using K-Means clustering algorithm. The Silouette scores for the different clusters are calculated and the group with the highest score is selected. The processed data are predicted using Random Forest Regression classification algorithm and the parameters are continuously adjusted to get the optimal result. Finally, different feature Importances are calculated and analysed to get the answer to research question 2.

6. Results

6.1. Research question 1

6.1.1. K-means clustering and visualisation

In [56]:
```
# import packages
import sklearn
import sklearn.cluster as sklc  # For clustering
import sklearn.metrics as sklm  # For the silhouette score
# for printing tables in more academic styles
from tabulate import tabulate
```

In [57]:
```
# data pre-processing and k-means clustering
## data pre-processing
### there is noly one variable, so Data standardisation is not necessary
## choose the best n clusters by caculating silhouette score
X = gdf_road_safety_bor_no_max[["casualty_rates"]]
random_state_seed =19
df_silhouette_score = pd.DataFrame(
    {"n_cluster": [2, 3, 4, 5, 6, 7], "silhouette_score": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
)
for index, row in df_silhouette_score.iterrows():
    n_clusters = int(row["n_cluster"])
    clusterer = sklc.KMeans(n_clusters=n_clusters, random_state=random_state_seed, n_init=10).f
    cluster_labels = clusterer.labels_
    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
```

```
    silhouette_avg = sklm.silhouette_score(X, cluster_labels)
#    print(
#        "For n_clusters =",
#        n_clusters,
#        "The average silhouette_score is :",
#        silhouette_avg,
#    )
    df_silhouette_score.at[index, "silhouette_score"] = silhouette_avg

#print(df_silhouette_score)
```

In [58]:
```
# show the result of silhouette score for different numbers of clusters
# settings
terminal_width = 91
# table (a)
# print the centered title for the table
title_table_611 = "Table 6.1.1. Results of silhouette score"
# print the names of the columns that were deleted
table_611 = tabulate(df_silhouette_score,
                     headers='keys',
                     tablefmt='mixed_outline',
                     showindex=False)
table_611_lines = table_611.split('\n')
centered_table_611 = "\n".join(
    line.center(terminal_width) for line in table_611_lines)

print(title_table_611.center(terminal_width))
print(centered_table_611)
```

Table 6.1.1. Results of silhouette score

n_cluster	silhouette_score
2	0.540914
3	0.494179
4	0.521945
5	0.500433
6	0.503266
7	0.472542

In [59]:
```
## do k-means cluster and join the result to df_try
# we fix the random_state so that the kmeans result is reproducible
random_state_seed = 19
num_clusters = 2
kmeans_output = sklc.KMeans(
    n_clusters=num_clusters, random_state=random_state_seed, n_init=10
).fit(gdf_road_safety_bor_no_max[["casualty_rates"]])

# Sanity check
# print(kmeans_output)

# This line creates a list giving the final cluster number of each point:
clustering_ids_kmeans = kmeans_output.labels_

# You can print the clustering IDs to get an ordered list of labels
# print(clustering_ids_kmeans)

# we will combine the clustering IDs to the dataframe
gdf_bor_no_max = gdf_road_safety_bor_no_max.assign(cluster_id=clustering_ids_kmeans)

# Have a look at the result:
# print(gdf_bor_no_max)
```

In [60]:
```
# gdf_bor_no_max.info()
```

In [61]:
```
# gdf_road_safety_bor.info()
```

In [62]:
```
# add the 'LSOA11CD' and 'casualty_rates' columns to df_id_lsoa_filtered DataFrame
gdf_bor = gdf_road_safety_bor.merge(gdf_bor_no_max[['GSS_CODE', 'cluster_id']],on='GSS_CODE',ho
```

In [63]:
```
# gdf_bor.info()
```

In [64]:
```
# for using np.nan
import numpy as np
## convert the resluts of clustering to text
# replace the value of the cluster_id column in gdf_bor
gdf_bor["cluster_id"] = gdf_bor["cluster_id"].replace(
    {
        0: "Casualty rates are relatively low",
        1: "Casualty rates are moderate",
        np.nan: "Casualty rates are relatively high"
    }
)
```

In [65]:
```
# gdf_bor.info()
```

In [66]:
```
### do visualization
import matplotlib.pyplot as plt
# 'ListedColormap' is used to create a custom colour map that directly corresponds to the diffe
from matplotlib.colors import ListedColormap
# the 'Patch' object is used to create colour blocks in the legend that correspond to the accid
from matplotlib.patches import Patch
import geopandas as gpd

# set colours and labels
colors = ['green', 'yellow', 'darkred']
labels = [
    "Casualty rates are relatively low",
    "Casualty rates are moderate",
    "Casualty rates are relatively high"
]

cmap = ListedColormap(colors)
label_to_int = {label: i for i, label in enumerate(labels)}
gdf_bor['cluster_id_int'] = gdf_bor['cluster_id'].map(label_to_int)

#  Build a graph with one row two columns
fig, axes = plt.subplots(1, 2, figsize=(11, 4), edgecolor="black", linewidth=1)
gdf_bor.plot(column="casualty_rates", ax=axes[0], legend=True, cmap="Reds")
gdf_bor.plot(column="cluster_id_int", ax=axes[1], legend=False, cmap=cmap)

# create the legend entry
legend_elements = [Patch(facecolor=color, label=label) for color, label in zip(colors, labels)]

# add legends to appropriate axes
axes[1].legend(handles=legend_elements, loc='lower left', bbox_to_anchor=(1, 0))

axes[0].text(-0.1, 1.19, "(a)", transform=axes[0].transAxes)
axes[1].text(-0.1, 1.04, "(b)", transform=axes[1].transAxes)
axes[0].tick_params(axis="both", labelsize=8)
axes[1].tick_params(axis="both", labelsize=8)

for ax in axes:
    for spine in ax.spines.values():
        spine.set_edgecolor("black")
        spine.set_linewidth(1)

# remove all ticks
axes[0].set_xticks([])
axes[0].set_yticks([])
```

```
axes[1].set_xticks([])
axes[1].set_yticks([])

plt.subplots_adjust(wspace=0.3)

# add the map title
fig.text(0.6, -0.0,
         'Figure 6.1.1. (a) Spatial distribution map of the traffic accident casualty Rate in L
         ha='center', va='bottom', fontsize=13)

# save the image - before plt.show()
# plt.savefig('The traffic accident casualty Rate spatial distribution map.png', dpi=311, bbox_

# display the map
plt.show()
```

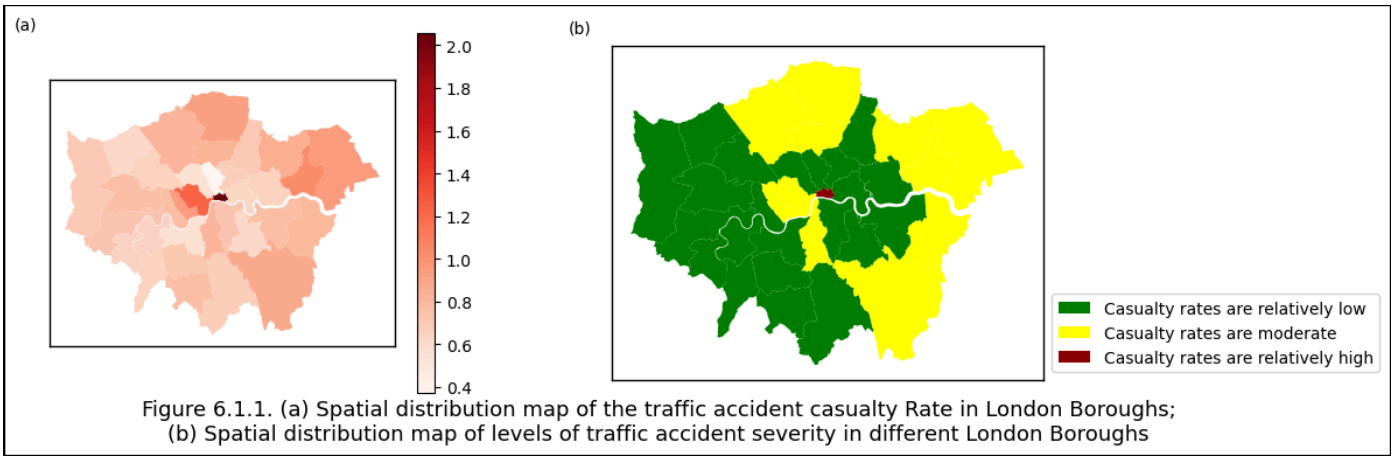

Figure 6.1.1. (a) Spatial distribution map of the traffic accident casualty Rate in London Boroughs;
(b) Spatial distribution map of levels of traffic accident severity in different London Boroughs

6.1.2. Global Moran's I

In [67]:
```
# pip install libpysal
```

In [68]:
```
# @jit(nopython=False) # for avoiding warnings in the next code block
```

In [69]:
```
import libpysal as lps
from esda.moran import Moran
from splot.esda import moran_scatterplot
```

```
/opt/conda/lib/python3.11/site-packages/libpysal/cg/alpha_shapes.py:38: NumbaDeprecationWarning:
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit defa
ult value for this argument is currently False, but it will be changed to True in Numba 0.59.0.
See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-mode
-fall-back-behaviour-when-using-jit for details.
  @jit
/opt/conda/lib/python3.11/site-packages/libpysal/cg/alpha_shapes.py:164: NumbaDeprecationWarnin
g: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit d
efault value for this argument is currently False, but it will be changed to True in Numba 0.59.
0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-m
ode-fall-back-behaviour-when-using-jit for details.
  @jit
/opt/conda/lib/python3.11/site-packages/libpysal/cg/alpha_shapes.py:198: NumbaDeprecationWarnin
g: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit d
efault value for this argument is currently False, but it will be changed to True in Numba 0.59.
0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-m
ode-fall-back-behaviour-when-using-jit for details.
  @jit
/opt/conda/lib/python3.11/site-packages/libpysal/cg/alpha_shapes.py:260: NumbaDeprecationWarnin
g: The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit d
efault value for this argument is currently False, but it will be changed to True in Numba 0.59.
0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-of-object-m
ode-fall-back-behaviour-when-using-jit for details.
  @jit
```

In [70]:
```
# create a Queen contiguity spatial weights matrix from the DataFrame 'gdf_bor'
w = lps.weights.Queen.from_dataframe(gdf_bor)
# Transform the spatial weights matrix to row-standardized form
w.transform = 'r'
```

In [71]:
```
# selection of columns to be analysed
y = gdf_bor['casualty_rates']

# set the random seed for reproducibility
np.random.seed(914113)

# calculate Moran's I
mi = Moran(y, w)

# create a DataFrame to hold results
df_results_moran = pd.DataFrame({
    "Moran's I": [mi.I],
    "P-value": [mi.p_sim]
})

# determine the appropriate width for centering based on a typical terminal size
terminal_width = 91

# generate the table with 'mixed_outline' format and ensure the content is centered
table_str = tabulate(df_results_moran, headers='keys',
                     tablefmt='mixed_outline',
                     numalign="center",
                     stralign="center",
                     showindex=False)

# center the table by padding
table_lines = table_str.split('\n')
centered_table = "\n".join(line.center(terminal_width) for line in table_lines)

# print the centered title for the table
title = "Table 6.1.2. The result of Global Moran's I"
print(title.center(terminal_width))
print(centered_table)
```

Table 6.1.2. The result of Global Moran's I

Moran's I	P-value
-0.0354757	0.48

6.2. Research question 2

6.2.1. At the London Boroughs level

6.2.1.1. Before building regression model

In [72]:
```
# import the basic libraries
import pandas as pd
from sklearn.linear_model import LinearRegression
import statsmodels
import statsmodels.api as sm

import numpy as np

import sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

```
import matplotlib.pyplot as plt
import seaborn as sn

pd.set_option('display.max_rows', 300) # specifies number of rows to show
pd.options.display.float_format = '{:40,.4f}'.format # specifies default number format to 4 dec
plt.style.use('ggplot') # specifies that graphs should use ggplot styling
%matplotlib inline
```

In [73]:
```
import statsmodels
print("statsmodels version: ", statsmodels.__version__)
print("pandas version: ", pd.__version__)
```

```
statsmodels version:  0.14.0
pandas version:  2.1.0
```

In [74]:
```
# df_bor.info()
# df_lsoa.info()
```

(1) Checking the correlation between variables

Before modelling, we always want to know the correlation between variables before building models. We will use the dataframe.corr() function to generate the correlation matrix and plot it.

We can check the collinearity between attributes using a correlation matrix as below.

In [75]:
```
# do a correlation matrix and plot it
df_factors_bor = df_bor.select_dtypes(include=[np.number])
correlation_matrix_bor = df_factors_bor.corr()
#print(correlation_matrix_bor)
plt.rcParams["axes.grid"] = False
f = plt.figure(figsize=(5, 5))
plt.matshow(df_factors_bor.corr(), fignum=f.number)
plt.xticks(range(df_factors_bor.shape[1]), df_factors_bor.columns, fontsize=6, rotation=85)
plt.yticks(range(df_factors_bor.shape[1]), df_factors_bor.columns, fontsize=6)
cb = plt.colorbar()
cb.ax.tick_params(labelsize=9)
# add the figure title
plt.title('Figure 6.2.1. Correlation Matrix', fontsize=13, y=-0.19)
# save image before plt.show()
#plt.savefig('lrgm1Correlation Matrix.png', dpi=311, bbox_inches='tight')
plt.show()
```

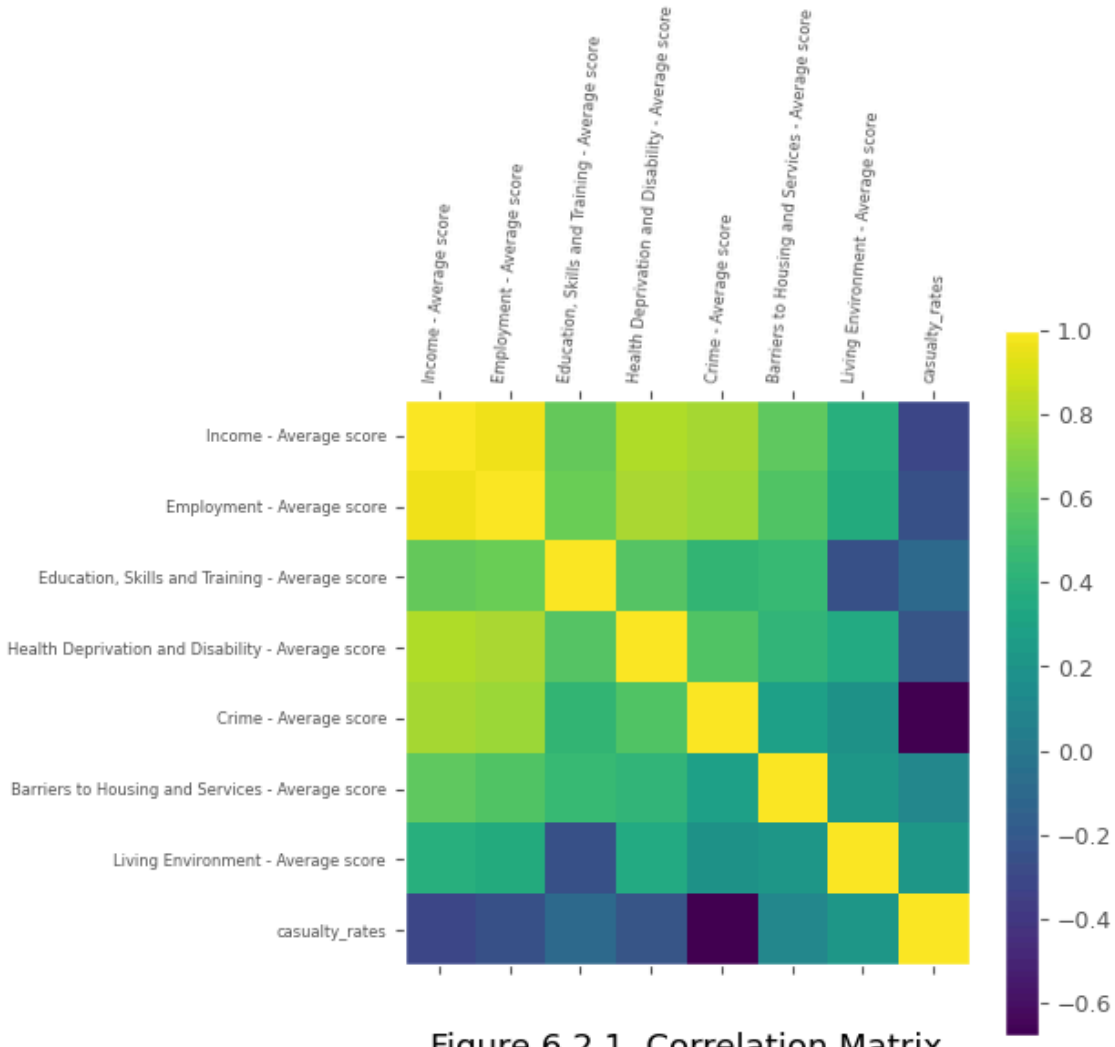

Figure 6.2.1. Correlation Matrix

(2) Using VIF to deal with multicollinearity

Here we introduce VIF to automatically deal with multicollinearity.

The Variance Inflation Factor (VIF) is a measure of multicollinearity among predictors within a multiple regression task.

In [76]:
```
# calculating VIF
# This function is adjusted from: https://stackoverflow.com/a/51329496/4667568
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

def drop_column_using_vif_(df, thresh=5):
    '''
    Calculates VIF each feature in a pandas dataframe, and repeatedly drop the columns with the
    A constant must be added to variance_inflation_factor or the results will be incorrect

    :param df: the pandas dataframe containing only the predictor features, not the response va
    :param thresh: (default 5) the threshould VIF value. If the VIF of a variable is greater th
    :return: dataframe with multicollinear features removed
    '''
    dropped_cols = []  # this step is used to record the names of the columns that were deleted
    while True:
        df_with_const = add_constant(df)
        vif_df = pd.Series([variance_inflation_factor(df_with_const.values, i)
                            for i in range(df_with_const.shape[1])],
                           name="VIF", index=df_with_const.columns).to_frame()

        # drop the constant term
```

```
        vif_df = vif_df.drop('const')

        # if the largest VIF is above the threshold, remove a variable with the largest VIF
        if vif_df.VIF.max() > thresh:
            index_to_drop = vif_df.index[vif_df.VIF == vif_df.VIF.max()].tolist()[0]
            dropped_cols.append(index_to_drop)  # record deleted columns
            df = df.drop(columns=index_to_drop)
        else:
            break

    return df, dropped_cols
```

In [77]:
```
# remove the dependent variable 'casualty_rates'
df_bor_f = df_factors_bor.drop('casualty_rates', axis=1)
df_bor_f, dropped_columns = drop_column_using_vif_(df_bor_f)
terminal_width = 91
# print the centered title for the table
title = "Table 6.2.1.1. (2)-(a) The result of Dropped Columns"
print(title.center(terminal_width))
# print the names of the columns that were deleted
table_dropped_cols = tabulate([[col] for col in dropped_columns], headers=['Dropped Columns:'],
table_dropped_cols_lines = table_dropped_cols.split('\n')
centered_table_dropped_cols = "\n".join(line.center(terminal_width) for line in table_dropped_c
print(centered_table_dropped_cols)
```

Table 6.2.1.1. (2)-(a) The result of Dropped Columns

Dropped Columns:
Income - Average score
Employment - Average score

In [78]:
```
# df_bor_f.info()
```

In [79]:
```
# adding a constatnt item to the data. add_constant is a function from statsmodels (see the imp
df_with_const_bor = add_constant(df_bor_f)

vif_df_bor = pd.Series([variance_inflation_factor(df_with_const_bor.values, i)
        for i in range(df_with_const_bor.shape[1])], name= "VIF",
        index=df_with_const_bor.columns).to_frame()

# drop the const
vif_df_bor = vif_df_bor.drop('const')
```

In [80]:
```
terminal_width = 91
# print the centered title for the table
title = "Table 6.2.1.1. (2)-(b) The result of VIF"
print(title.center(terminal_width))
# print the names of the columns that were deleted
table_VIF = tabulate(vif_df_bor, headers='keys', tablefmt='mixed_outline')
table_VIF_lines = table_VIF.split('\n')
centered_table_VIF = "\n".join(line.center(terminal_width) for line in table_VIF_lines)
print(centered_table_VIF)
```

Table 6.2.1.1. (2)-(b) The result of VIF

	VIF
Education, Skills and Training - Average score	2.88944
Health Deprivation and Disability - Average score	2.55302
Crime - Average score	1.53355
Barriers to Housing and Services - Average score	1.5223
Living Environment - Average score	1.99689

In [81]:
```
df_factors_bor = df_factors_bor.drop(['Income - Average score',
                                      'Employment - Average score'], axis=1)
```

In [82]:
```
# df_factors_bor.info()
```

6.2.1.2. Fitting the OLS model

In [83]:
```
model_casualty_rates_bor = sm.OLS(df_factors_bor[['casualty_rates']], exog=sm.add_constant(df_b
```

In [84]:
```
# model_casualty_rates_bor.summary()
```

In [85]:
```
data_a = [
    ["Dep. Variable:", "casualty_rates", "R-squared:", 0.759],
    ["Model:", "OLS", "Adj. R-squared:", 0.715],
    ["Method:", "Least Squares", "F-statistic:", 17.04],
    ["No. Observations:", 33, "Prob (F-statistic):", 1.30e-07]
]
data_b = [
    ["const", -0.3933, 0.123],
    ["Education, Skills and Training - Average score", 0.0328, 0.001],
    ["Health Deprivation and Disability - Average score", -0.1995, 0.036],
    ["Crime - Average score", -0.5896, 0.000],
    ["Barriers to Housing and Services - Average score", 0.0027, 0.545],
    ["Living Environment - Average score", 0.0241, 0.000]
]
headers_b = ["Variable", "P>|t|"]
data_c = [
    ["Durbin-Watson:", "Jarque-Bera (JB):", "Cond. No."],
    [2.176, 2.147, 460.]
]

# settings
terminal_width = 91
# table (a)
# print the centered title for the table
title_table_a = "Table 6.2.1.2. (a) Results of OLS Regression"
# print the names of the columns that were deleted
table_a = tabulate(data_a,
                   headers="firstrow",
                   tablefmt='simple_grid')
table_a_lines = table_a.split('\n')
centered_table_a = "\n".join(
    line.center(terminal_width) for line in table_a_lines)

# table (b)
# print the centered title for the table
title_table_b = "Table 6.2.1.2. (b) Results of OLS Regression"
# print the names of the columns that were deleted
table_b = tabulate(
    data_b, headers=headers_b, tablefmt='simple_grid')
table_b_lines = table_b.split('\n')
centered_table_b = "\n".join(
    line.center(terminal_width) for line in table_b_lines)

# table (c)
# print the centered title for the table
title_table_c = "Table 6.2.1.2. (c) Results of OLS Regression"
# print the names of the columns that were deleted
table_c = tabulate(data_c,
                   headers="firstrow",
                   tablefmt='simple_grid')
table_c_lines = table_c.split('\n')
centered_table_c = "\n".join(
    line.center(terminal_width) for line in table_c_lines)
```

```
 # print all tables
print(title_table_a.center(terminal_width))
print(centered_table_a)
print(title_table_b.center(terminal_width))
print(centered_table_b)
print(title_table_c.center(terminal_width))
print(centered_table_c)
```

Table 6.2.1.2. (a) Results of OLS Regression

Dep. Variable:	casualty_rates	R-squared:	0.759
Model:	OLS	Adj. R-squared:	0.715
Method:	Least Squares	F-statistic:	17.04
No. Observations:	33	Prob (F-statistic):	1.3e-07

Table 6.2.1.2. (b) Results of OLS Regression

	Variable	P>|t|
const	-0.3933	0.123
Education, Skills and Training - Average score	0.0328	0.001
Health Deprivation and Disability - Average score	-0.1995	0.036
Crime - Average score	-0.5896	0
Barriers to Housing and Services - Average score	0.0027	0.545
Living Environment - Average score	0.0241	0

Table 6.2.1.2. (c) Results of OLS Regression

Durbin-Watson:	Jarque-Bera (JB):	Cond. No.
2.176	2.147	460

6.2.1.3. Residuals analysis

In [86]:
```
# Create a new figure with subplots
fig, axs = plt.subplots(1, 3, figsize=(10, 3))

# (a) Residuals vs. Fitted plot
axs[0].scatter(
    model_casualty_rates_bor.fittedvalues,
    model_casualty_rates_bor.resid,
    alpha=0.5,
    s=13)
axs[0].set_xlabel('Fitted casulties rates', fontsize=13, labelpad=1)
axs[0].set_ylabel('Residual', fontsize=13, labelpad=1)
axs[0].grid(True)

# (b) Histogram of residuals
axs[1].hist(model_casualty_rates_bor.resid,
            bins=9, color='skyblue', edgecolor='black', alpha=0.7)
axs[1].set_xlabel('Residuals', fontsize=13, labelpad=1)
axs[1].set_ylabel('Frequency', fontsize=13, labelpad=1)
axs[1].grid(True)

# (c) QQ plot of residuals with grid
sm.qqplot(model_casualty_rates_bor.resid,
          fit=True, line="45", ax=axs[2])
axs[2].grid(True)
```

```
# Add (a), (b), (c) labels in the top left corner of each subplot
fig.text(0.006, 0.91, '(a)', fontsize=19, fontweight='bold')
fig.text(0.338, 0.91, '(b)', fontsize=19, fontweight='bold')
fig.text(0.67, 0.91, '(c)', fontsize=19, fontweight='bold')

# Adjust the layout to prevent overlapping labels
plt.tight_layout()
fig.text(0.50, -0.091,
         'Figure 6.2.1.3. (a) Residual vs. Fitted Plot of poverty rate; '+
         '(b) Histogram of Residuals of the Linear Model; '+
         '(c) QQ Plot of Residuals. ',
         ha='center', va='bottom', fontsize=13)
# plt.savefig('residuals analysis.png', dpi=311, bbox_inches='tight')
# Display the combined plot
plt.show()
```

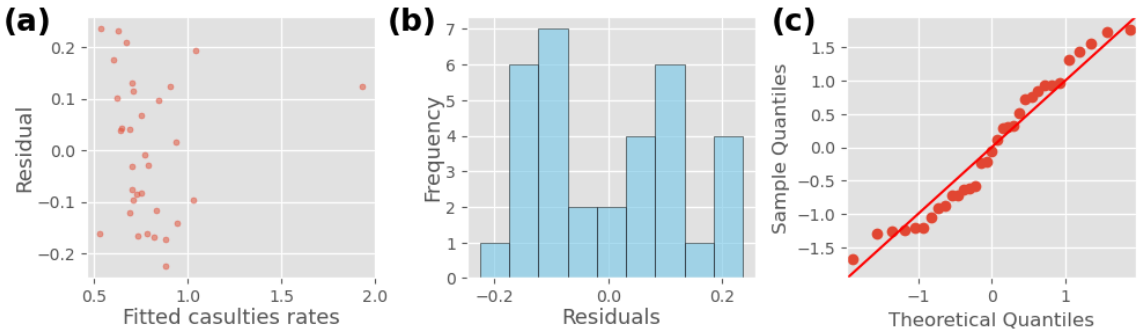

Figure 6.2.1.3. (a) Residual vs. Fitted Plot of poverty rate; (b) Histogram of Residuals of the Linear Model; (c) QQ Plot of Residuals.

6.2.2. At the London LSOAs level

In [87]:
```
# import the Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
# import train_test_split function
from sklearn.model_selection import train_test_split
# import evaluation metrics
from sklearn.metrics import accuracy_score, classification_report
# pip install imbalanced-learn # if the next line went wrong, please running this line
# import SMOTE for dealing with category imbalances
from imblearn.over_sampling import SMOTE
# import tools for cross-validation
from sklearn.model_selection import cross_val_score, KFold
# imported imbalance-learn pipeline for combined SMOTE and model training
# from imblearn.pipeline import Pipeline as ImbPipeline
# import functions from the rfpimp package to calculate and visualize feature importances
from rfpimp import importances, plot_importances
# import matplotlib for plotting
import matplotlib.pyplot as plt
```

In [88]:
```
# df_lsoa.info()
```

In [89]:
```
df_cls_lsoa = df_lsoa.copy()
```

In [90]:
```
df_cls_lsoa_no_zero = df_cls_lsoa[df_cls_lsoa['casualty_rates'] != 0]
```

In [91]:
```
# data pre-processing and k-means clustering
## data pre-processing
### there is noly one variable, so Data standardisation is not necessary
## choose the best n clusters by caculating silhouette score
X = df_cls_lsoa_no_zero[["casualty_rates"]]
random_state_seed =92
df_silhouette_score = pd.DataFrame(
    {"n_cluster": [2, 3, 4, 5, 6, 7], "silhouette_score": [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]}
```

```
)
for index, row in df_silhouette_score.iterrows():
    n_clusters = int(row["n_cluster"])
    clusterer = sklc.KMeans(n_clusters=n_clusters, random_state=random_state_seed, n_init=10).f
    cluster_labels = clusterer.labels_
    # The silhouette_score gives the average value for all the samples.
    # This gives a perspective into the density and separation of the formed
    # clusters
    silhouette_avg = sklm.silhouette_score(X, cluster_labels)
#    print(
#        "For n_clusters =",
#        n_clusters,
#        "The average silhouette_score is :",
#        silhouette_avg,
#    )
    df_silhouette_score.at[index, "silhouette_score"] = silhouette_avg

#print(df_silhouette_score)
```

In [92]:
```
# show the result of silhouette score for different numbers of clusters
# settings
terminal_width = 91
# table (a)
# print the centered title for the table
title_table_622 = "Table 6.2.2. (1) Results of silhouette score"
# print the names of the columns that were deleted
table_622 = tabulate(df_silhouette_score,
                     headers='keys',
                     tablefmt='mixed_outline',
                     showindex=False)
table_622_lines = table_622.split('\n')
centered_table_622 = "\n".join(
    line.center(terminal_width) for line in table_622_lines)

print(title_table_622.center(terminal_width))
print(centered_table_622)
```

Table 6.2.2. (1) Results of silhouette score

n_cluster	silhouette_score
2	0.818577
3	0.72798
4	0.68374
5	0.668011
6	0.632471
7	0.645241

In [93]:
```
## do k-means cluster and join the result to df_try
# fix the random_state so that the kmeans result is reproducible
random_state_seed = 91
num_clusters = 2
kmeans_output = sklc.KMeans(
    n_clusters=num_clusters, random_state=random_state_seed, n_init=10
).fit(df_cls_lsoa_no_zero[["casualty_rates"]])

# sanity check
# print(kmeans_output)

# this line creates a list giving the final cluster number of each point:
clustering_ids_kmeans = kmeans_output.labels_

# print the clustering IDs to get an ordered list of labels
# print(clustering_ids_kmeans)
```

```
# combine the clustering IDs to the dataframe
df_cls_lsoa_no_zero = df_cls_lsoa_no_zero.assign(cluster_id=clustering_ids_kmeans)
```

In [94]:
```
# have a look at the result:
# df_cls_lsoa_no_zero.head()
```

In [95]:
```
# add the 'cluster_id' column to df_cls_lsoa_filtered DataFrame
df_cls_lsoa = df_cls_lsoa.merge(
    df_cls_lsoa_no_zero[['LSOA code (2011)', 'cluster_id']],
    on='LSOA code (2011)',
    how='left')
```

In [96]:
```
## convert the resluts of clustering to text
# replace the value of the 'cluster_id' column in df_cls_lsoa
df_cls_lsoa["cluster_id"] = df_cls_lsoa["cluster_id"].replace(
    {
        0: 2,
        1: 3,
        np.nan: 1
    }
)
```

In [97]:
```
df_cls_num_lsoa = df_cls_lsoa.select_dtypes(
    include=[np.number]) # select columns
df_cls_num_lsoa = df_cls_num_lsoa.drop(
    'casualty_rates', axis=1) # drop 'casualty_rates'
df_cls_num_lsoa = df_cls_num_lsoa.rename(
    columns={'cluster_id': 'severity_levels'}) # rename
```

In [98]:
```
# df_cls_num_lsoa.info()
```

In [99]:
```
random_state_split = 100
train_x, test_x, train_y, test_y = train_test_split(
    df_cls_num_lsoa.drop(['severity_levels'], axis = 1),
    df_cls_num_lsoa.severity_levels,
    random_state=random_state_split)
```

In [100…
```
# double check the rows and columns of the outputs
print(train_x.shape)
print(train_y.shape)
print(test_x.shape)
print(test_y.shape)

# check the index of train_x and train_y - they should be identical. The index indicates which
print(train_x.index.identical(train_y.index))
print(test_x.index.identical(test_y.index))
```

```
(3626, 7)
(3626,)
(1209, 7)
(1209,)
True
True
```

In [101…
```
# Handle class imbalance
smote = SMOTE(random_state=191)
train_x_resampled, train_y_resampled = smote.fit_resample(train_x, train_y)

# Initialize the Random Forest classifier with parameters to reduce overfitting
clf = RandomForestClassifier(
    # Number of trees
    n_estimators=200,
    # Seed for random number generator
    random_state=91,
    # Minimum number of samples required to split an internal node
```

```
    min_samples_split=8,
    # Minimum number of samples required to be at a leaf node
    min_samples_leaf=2,
    # The number of features to consider when looking for the best split (sqrt)
    max_features='sqrt',
    # Adjust class weights to balance the class imbalance
    class_weight='balanced'
)

# Conduct cross-validation and set up cross-validation parameters
cv = KFold(n_splits=5, shuffle=True, random_state=42)

# Calculate cross-validation scores
cv_scores = cross_val_score(clf, train_x_resampled, train_y_resampled, cv=cv, scoring='accuracy

# Train the model
clf.fit(train_x_resampled, train_y_resampled)

# Make predictions on the test set
pred_test = clf.predict(test_x)

# Evaluate the model
test_accuracy = accuracy_score(test_y, pred_test)  # Calculate test accuracy

# Print cross-validation scores and test performance
# print(f"Cross-Validated Scores: {cv_scores}")
# print(f"Mean CV Score: {cv_scores.mean()}")
# print(f"Testing Accuracy: {test_accuracy}")
# print("\nClassification Report on Test Data:")
# print(classification_report(test_y, pred_test))
```

In [102…

```
# settings for printing Table 6.2.2. (2)
terminal_width = 91
# table 6.2.2. (2)
title_table_6222 = "Table 6.2.2. (2) Results of cross-validation scores and test performance"
# table 6.2.2. (2)-(a)
title_table_6222a = "Table 6.2.2. (2)-(a) Results of cross-validation scores and test performan
data_6222a = [
    ["Cross-Validated Scores:", "[0.67722603 0.67123288 0.66695205 0.66409597 0.67694944]"],
    ["Mean CV Score:", 0.6712912748999307],
    ["Testing Accuracy:", 0.48883374689826303]
]
headers_6222a = ["Different scores", "Results"]
table_6222a = tabulate(data_6222a, headers=headers_6222a, tablefmt='mixed_outline')
table_6222a_lines = table_6222a.split('\n')
centered_table_6222a = "\n".join(line.center(terminal_width) for line in table_6222a_lines)

# table 6.2.2. (2)-(b)
title_table_6222b = "Table 6.2.2. (2)-(b) Results of cross-validation scores and test performan
report = classification_report(test_y, pred_test, output_dict=True)
data_6222b = []
for label, metrics in report.items():
    if isinstance(metrics, dict):  # 确保是每个类别的报告，排除总结性统计如 accuracy
        row = [label] + [metrics['precision'], metrics['recall'], metrics['f1-score'], metrics[
        data_6222b.append(row)
headers_6222b = ["Classifications", "Precision", "Recall", "F1-Score", "Support"]
table_6222b = tabulate(data_6222b, headers=headers_6222b, tablefmt='mixed_outline')
table_6222b_lines = table_6222b.split('\n')
centered_table_6222b = "\n".join(line.center(terminal_width) for line in table_6222b_lines)

# print all tables
print(title_table_6222.center(terminal_width))
print(title_table_6222a.center(terminal_width))
print(centered_table_6222a)
print(title_table_6222b.center(terminal_width))
print(centered_table_6222b)
```

Table 6.2.2. (2) Results of cross-validation scores and test performance

Table 6.2.2. (2)-(a) Results of cross-validation scores and test performance

Different scores	Results
Cross-Validated Scores:	[0.67722603 0.67123288 0.66695205 0.66409597 0.67694944]
Mean CV Score:	0.6712912748999307
Testing Accuracy:	0.48883374689826303

Table 6.2.2. (2)-(b) Results of cross-validation scores and test performance

Classifications	Precision	Recall	F1-Score	Support
1.0	0.559585	0.528548	0.543624	613
2.0	0.485075	0.47532	0.480148	547
3.0	0.0744681	0.142857	0.0979021	49
macro avg	0.373043	0.382242	0.373891	1209
weighted avg	0.506212	0.488834	0.49684	1209

In [103…
```
# Calculate feature importances
imp = importances(clf, train_x_resampled, train_y_resampled)  # Use the training set data
# print(imp)
```

In [104…
```
# settings for printing Table 6.2.2. (3)
terminal_width = 91
# table 6.2.2. (3)
title_table_6223 = "Table 6.2.2. (3) Results of importances for different features"

headers_6223 = ["Features", "Importances"]
table_6223 = tabulate(imp, headers=headers_6223, tablefmt='mixed_outline')
table_6223_lines = table_6223.split('\n')
centered_table_6223 = "\n".join(line.center(terminal_width) for line in table_6223_lines)

# print all tables
print(title_table_6223.center(terminal_width))
print(centered_table_6223)
```

Table 6.2.2. (3) Results of importances for different features

Features	Importances
Living Environment Score	0.2992
Health Deprivation and Disability Score	0.2928
Crime Score	0.2878
Barriers to Housing and Services Score	0.2018
Education, Skills and Training Score	0.1778
Income Score (rate)	0.1752
Employment Score (rate)	0.129

In [105…
```
# Plot the feature importances
viz = plot_importances(imp)
# add the figure title
plt.title('Figure 6.2.2. The feature importances', fontsize=13, x=-1.4, y=-0.519)
plt.show()  # Show the plot
```

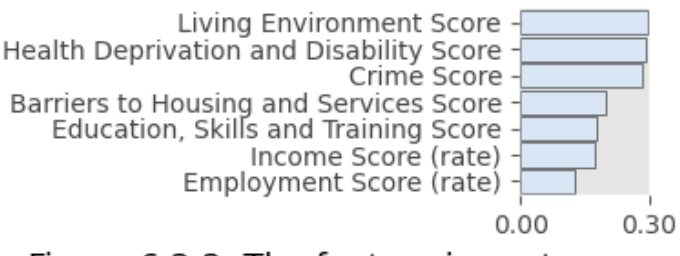

Figure 6.2.2. The feature importances

7. Discussion

This section discusses separately two research questions of this paper based on the previous investigation results.

7.1. Research question 1

The result 'Figure 6.1.1.' identifies Bromley, Havering, Barnet, Lambeth, Bexley, Enfield, Redbridge, Kensington and Chelsea, Westminster, Haringey, Barking and Dagenham, City of London. Westminster, Haringey, Barking and Dagenham, City of London are identified as being in the 'Casualty rates are moderate' and 'Casualty rates are relatively high' categories. It is suggested that the London government could focus on combating the high casualty rates in these areas.

With the result of global Moran's I in Table 6.1.2., it can be analysed that there is a slightly dispersed distribution of traffic accident casualty rates for vulnerable groups at Boroughs level in London. The p-value is too large, so the spatial pattern of casualty rates is randomly distributed at Boroughs level in London.

7.2. Research question 2

7.2.1. At the London Boroughs level

According to 'Table 6.2.1. Correlation Matrix', the correlation between income and employment is relatively high, which suggests that there may be multicollinearity. The results in 'Table 6.2.1.1. (2)-(a) The result in Dropped Columns' and 'Table 6.2.1.1. (2)-(b) The result of VIF ' demonstrate that the multicollinearity does exist. Therefore, the multiple linear regression model needs to be fitted after removing the two potential factors of income and employment.

According to the OLS regression results in Table 6.2.1.2., the model explained 75.9 percent of the variation in accident rates, with an adjusted R-squared of 71.5 percent, indicating a good explanatory power of the model. However, when considering degrees of freedom the explanatory power of the model decreases. The F-statistic of 17.04 with a very low p-value (1.30e-07) shows that the predictors in the model significantly influence the accident rates. Specifically, the effects on casualty rates of equal variations in factors are crime, health, education, and living environment, in descending order, and all of their regression coefficients are significant enough to reject the null hypothesis.

In the Residuals analysis of Figure 6.2.1.3., it could be seen that (a) the residuals overall show a random distribution, (b) the residuals in the graph are roughly normally distributed; and (c) most points in the graph show a normal distribution close to the reference line. However, the sample size of the dataset is only 32, which may introduce some randomness into the model and affect the reliability of the results.

7.2.2. At the London LSOAs level

Based on the results of the importance in Table 6.2.2. and Figure 6.2.2., it could be seen that at the London LSOAs level, the factors in terms of their impact on the inequitable distribution of traffic accident rates are living environment, health, crime, housing, education, income, and employment, in descending order. Therefore, it is recommended that the government should firstly reduce the deprivation of these three factors, which are living environment, health and crime, to reduce the inequitable distribution of traffic accidents.

However, according to the results of Random Forest classification model in Table 6.2.2., the test score in the cross-validated model is about 0.182 lower than the training score, which concludes that the Random Forest model is a bit overfitted. In addition, for the category '3.0', which is the category with high traffic accident casualty rate, the model has a relatively small number of test samples and the performance of the prediction is poorer than the other two categories. The above two points are also the limitations of this study.

8. Conclusion

Research question 1:

The spatial pattern is that the road traffic accidents of the vulnerable group indcluding children (0 - 15 years) and old people (more than or equal to 65 years) are randomly distributed at the London Borough leveivation.

Research question 2:

At the London Borough level, the impacts on spatial distribution inequalities in road safety for the vulnerable group are, in descending order, crime, health, education, and living environment in indices of deprivation.

At the London LSOAs level, the effects on the spatial distribution inequalities of road safety for the vulnerable group are, from largest to smallest, living environment, health, crime, housing, education, income, and employment in indices of deprivation.

References

Leijdesdorff, H. A., Gillissen, S., Schipper, I. B.,and Krijnen, P. (2020). 'Injury Pattern and Injury Severity of In-Hospital Deceased Road Traffic Accident Victims in The Netherlands: Dutch Road Traffic Accidents Fatalities', *World journal of surgery*, 44 (5), pp.1470–1477. doi: 10.1007/s00268-019-05348-6.

Sun, L., Zhang, Z., Chen, M., and Dong, S. (2021). 'Research on Characteristics of Perpetrators and casualties in Road Traffic Accidents in China - - Based on the panel data analysis of national traffic accidents from 2006 to 2019', E3S Web of Conferences, 275, pp. 2024-. doi: 10.1051/e3sconf/202127502024.

O'Toole, S. E. and Christie, N. (2018). 'Deprivation and road traffic injury comparisons for 4–10 and 11–15 year-olds', *Journal of transport & health*, 11, pp.221–229. doi: 10.1016/j.jth.2018.08.003.

Kluczyńska, U., Kłonkowska, A. M. and Bieńkowska, M. (2024). 'Researching Vulnerable Groups: Definitions, Controversies, Dilemmas, and the Researcher's Personal Entanglement', *Qualitative sociology review : QSR*, 20 (1), pp.10–28. doi: 10.18778/1733-8077.20.1.02.

Sitohang, H., Rosmiati, and Merni, S. (2020). 'K-Means method for analysis of accident-prone areas in Palangka Raya', *Journal of Physics: Conference Series*, 1566 (1), pp.12008-. doi: 10.1088/1742-6596/1566/1/012008.

Hazaymeh, K., Almagbile, A., and Alomari, A. H. (2022). 'Spatiotemporal Analysis of Traffic Accidents Hotspots Based on Geospatial Techniques', *ISPRS international journal of geo-information*, 11 (4), pp.260-. doi: 10.3390/ijgi11040260.

Kourouma, K., Delamou, A., Lamah, L., Camara, B. S., Kolie, D., Sidibé, S., Béavogui, A. H., Owiti, P., Manzi, M., Ade, S., and Harries, A. D. (2019). 'Frequency, characteristics and hospital outcomes of road traffic accidents and their victims in Guinea: A three-year retrospective study from 2015 to 2017', *BMC public health*, 19 (1), 1022–1022. doi: 10.1186/s12889-019-7341-9.

Khanum, H., Garg, A., and Faheem, M. I. (2023). 'Accident severity prediction modeling for road safety using random forest algorithm: an analysis of Indian highways', *F1000 research*, 12, pp.494-. doi:

10.12688/f1000research.133594.1.

Chen, H. (2024a). 'CASA0006 Workshop 6: Spatial Clustering', *CASA0006: Data Science for Spatial Systems*. University College London. Unpublished.

Chen, H. (2024b). 'CASA0006 Workshop 7: Regression', *CASA0006: Data Science for Spatial Systems*. University College London. Unpublished.

Chen, H. (2024c). 'CASA0006 Workshop 3: Tree-based methods', *CASA0006: Data Science for Spatial Systems*. University College London. Unpublished.