Coursework Coversheet

School of Geography. FACULTY OF ENVIRONMENT

GEOG5403 Creative Coding for Urban Problems

UNIVERSITY OF LEEDS

Student ID	201788996、201413609、201769905 201753800、201741443、201787698	Mark	
Group Number	Group 14	Less deduction (state reason)	
Assignment title	Building Connected Communities: 20-Minute Neighbourhood Model		
Marker		Final Mark	

Justification of mark (using specific text from criteria):

To improve your work for next time:
1.
2.
3.

Additional comments:

The 20-Minute neighbbourishoood model

Building connnected communities

Group 14

2024.05.07

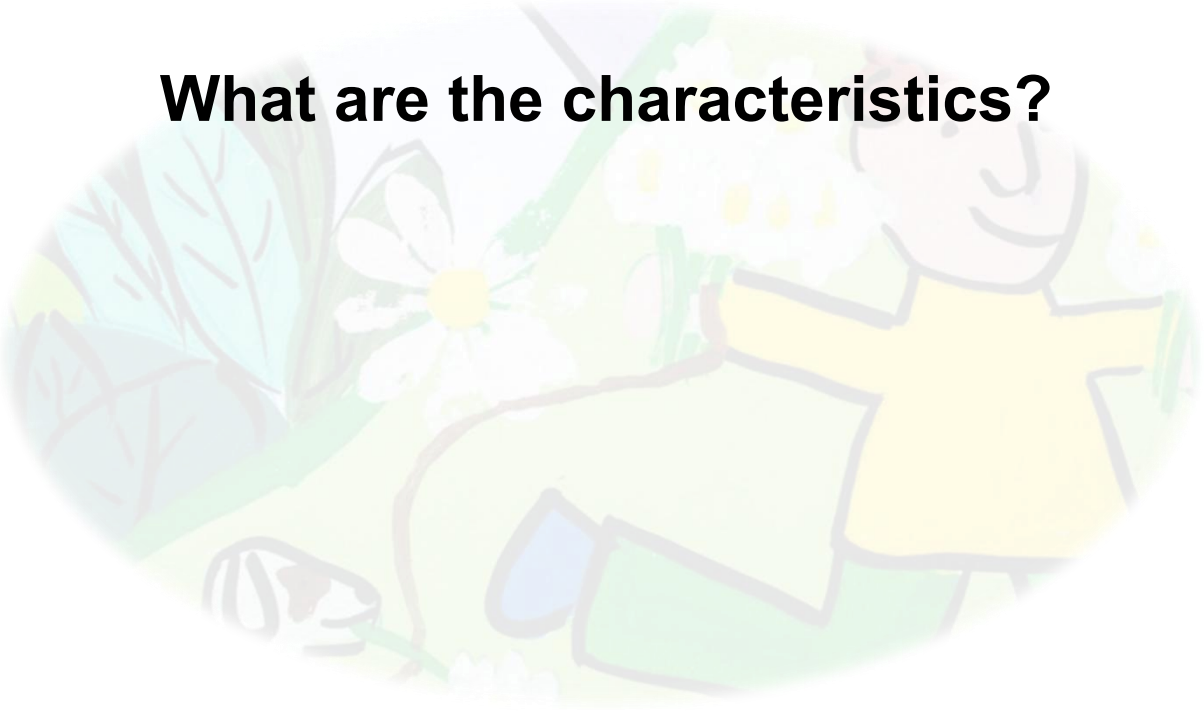
1 Background

What is a 20-minute neighbourhood?

A 20-minute neighbourhood is an urban design strategy aimed at ensuring that residents have access to most essential services within a 20-minute walk from their homes. This concept promotes sustainable urban development by enhancing community connectivity and accessibility.

1 Background

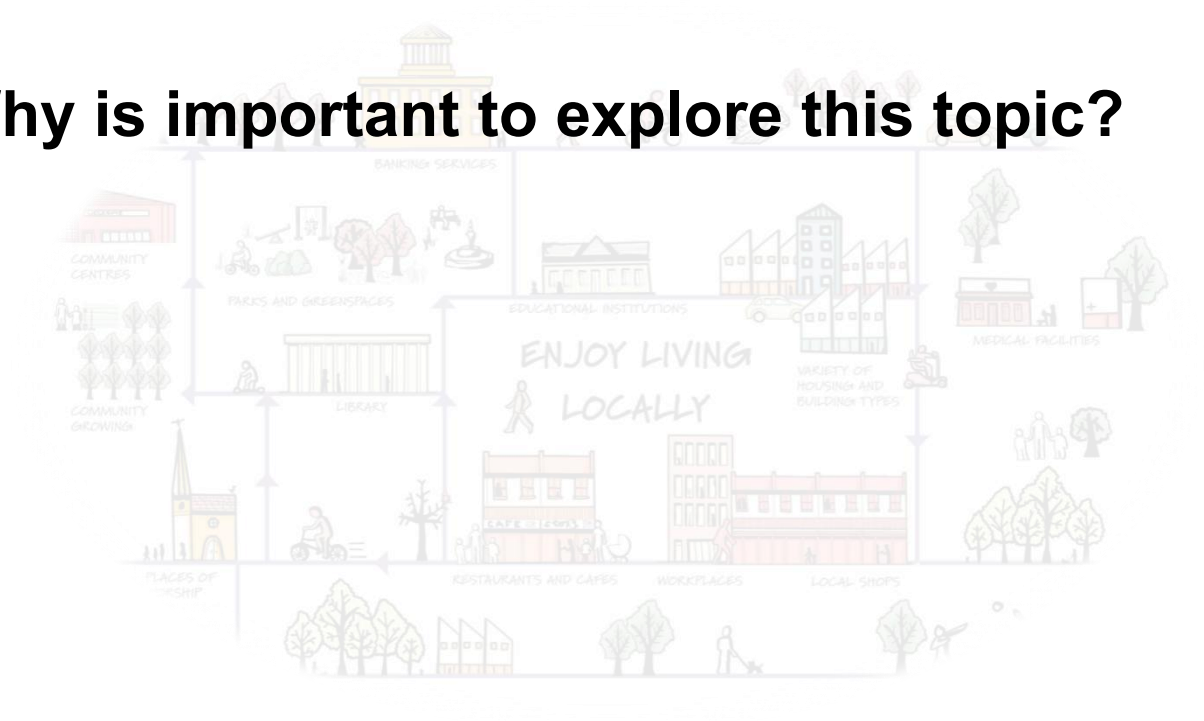
What are the characteristics?



- Accessible Public Services
- Multifunctional Communities
- Sustainable Transportation System
- Pedestrian-friendly Infrastructure
- Housing Diversity
- Active Community Engagement
- …

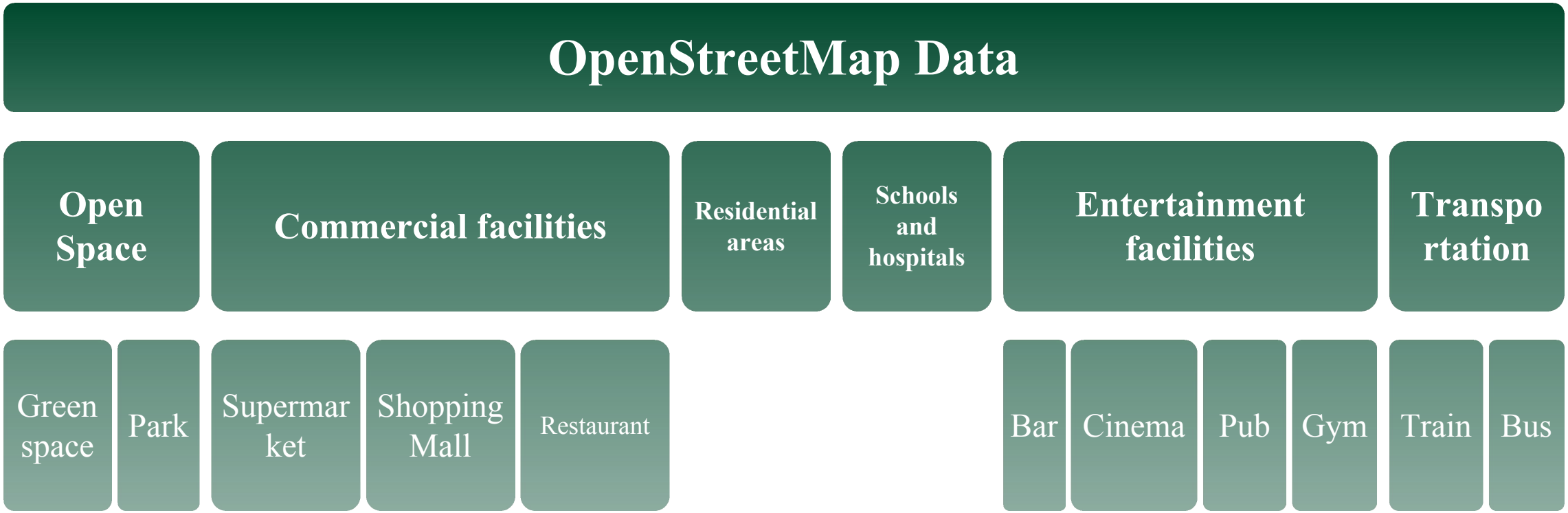
1 Background

Why is important to explore this topic?



- Residents can **quickly access to daily needs**, enhancing overall life quality in 20-min neighbourhood.
- Residents are more likely to **socialize and participate in community activities** within multifunctional communities.
- The 20-minute neighbourhood offers a more compact and efficient model for **urban sustainable development**

2 Data Description



2 Data Description

Datasets

LSOA	Includes geographic boundaries, LSOA name, LSOA code information for identifying and analyzing communities.
House Price	Contains asking prices and locations of properties to calculate average house prices and analyze housing market trends per LSOA area.
Crime Rate	Provides records of crime events along with geographic area codes and names to compute crime rates and assess safety across LSOA areas.
IMD	IMD assesses socioeconomic deprivation across income, employment, health, education, crime, and living environment.

3 Approach

What is the spatial distribution of 20-minute neighbourhoods across Bradford?
Are there areas better or worse suited to 20-minute neighbourhoods?

Data Standardization

- Use **StandardScaler** to standardize the data, ensuring each variable has equal influence in the clustering analysis.

Buffer

- Extract and integrate information from multiple data sources, and calculate the proximity and accessibility of residential areas and public service facilities within 800 meters by creating an 800-meter buffer zone.

Principal Component Analysis (PCA)

- PCA was performed to calculate different weights for different domains to create a comprehensive index and identify 20-minute neighbor gods.

Data Clustering

- Clustering model classifies residential areas into clusters based on similar characteristics to simplify the decision-making process Visualized interactive map based on LSOA that highlights hotspots for various amenities..

Performance Metrics Calculation

- Calculate the within-cluster sum of squares (WCSS) to determine the optimal number of clusters.

Data Visualization

- Use **matplotlib** and **plotly** libraries to draw various distribution maps, including distributions of education, parks, crime, etc.
- Create interactive maps with **Folium**, highlighting hotspots of various amenities.

3 Approach

Raw data

	LSOA	education	parks	hospitals	services	transport	residential	crime	house	IMD
0	E01010573	0.0	0.0	0.0	0.0	0.0	0.0	0.322581	173796.297013	8.0
1	E01010576	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	173796.297013	9.0
2	E01010568	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	160000.000000	8.0
3	E01010569	13.0	4.0	1.0	4.0	1.0	1.0	0.645161	173796.297013	2.0
4	E01010570	0.0	0.0	0.0	0.0	0.0	0.0	0.258065	173796.297013	9.0

Weight settings:

$$
\begin{aligned}
Weight = {} & education \times 0.33869342 + parks \times 0.19263579 \\
& + hospitals \times 0.11895071 + services \times 0.09969657 \\
& + hospitals \times 0.11895071 + transport \times 0.0888844 \\
& + residential \times 0.06553174 + crime \times 0.04485399 \\
& + house \times 0.02864435 + IMD \times 0.02210903
\end{aligned}
$$

Processed data

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house	IMD	index
0	E01010573	Bradford 013A	53.85777	-1.79493	1	1	1	1	1	1	3	4	5	2
1	E01010576	Bradford 016E	53.85259	-1.75631	1	1	1	1	1	1	1	4	6	2
2	E01010568	Bradford 016A	53.84526	-1.75196	1	1	1	1	1	1	1	3	5	2
3	E01010569	Bradford 016B	53.84155	-1.78365	8	6	1	6	1	1	6	4	1	9
4	E01010570	Bradford 018A	53.84833	-1.79227	1	1	1	1	1	1	2	4	6	2

Standardization processing

Assigning weights

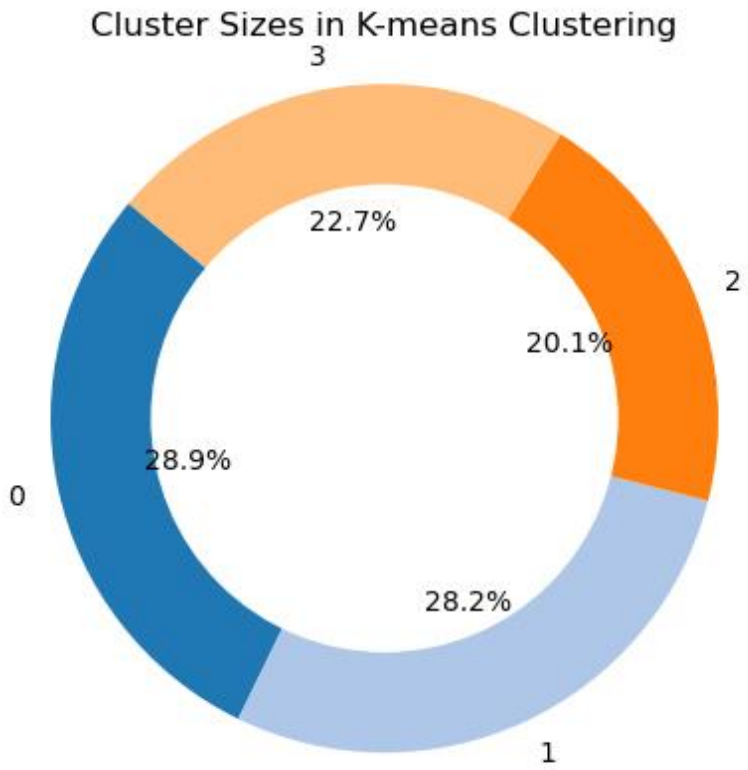
Rank sorting

4 Findings

4.1 Clustering Results

Cluster Distribution

Clusters are evenly distributed across Bradford, indicating no single type of neighborhood prevails, suggesting diverse neighborhood characteristics.



Cluster Characteristics

Clusters 0 and 1, with high index, are near the 20-minute neighbourhood ideal, needing minimal changes, **such as E01010573 and E01010570**. Clusters 2 and 3, with lower index or higher deprivation, require significant improvements, **such as E01010866 and E01033690**.

4 Findings

4.2 Heatmaps

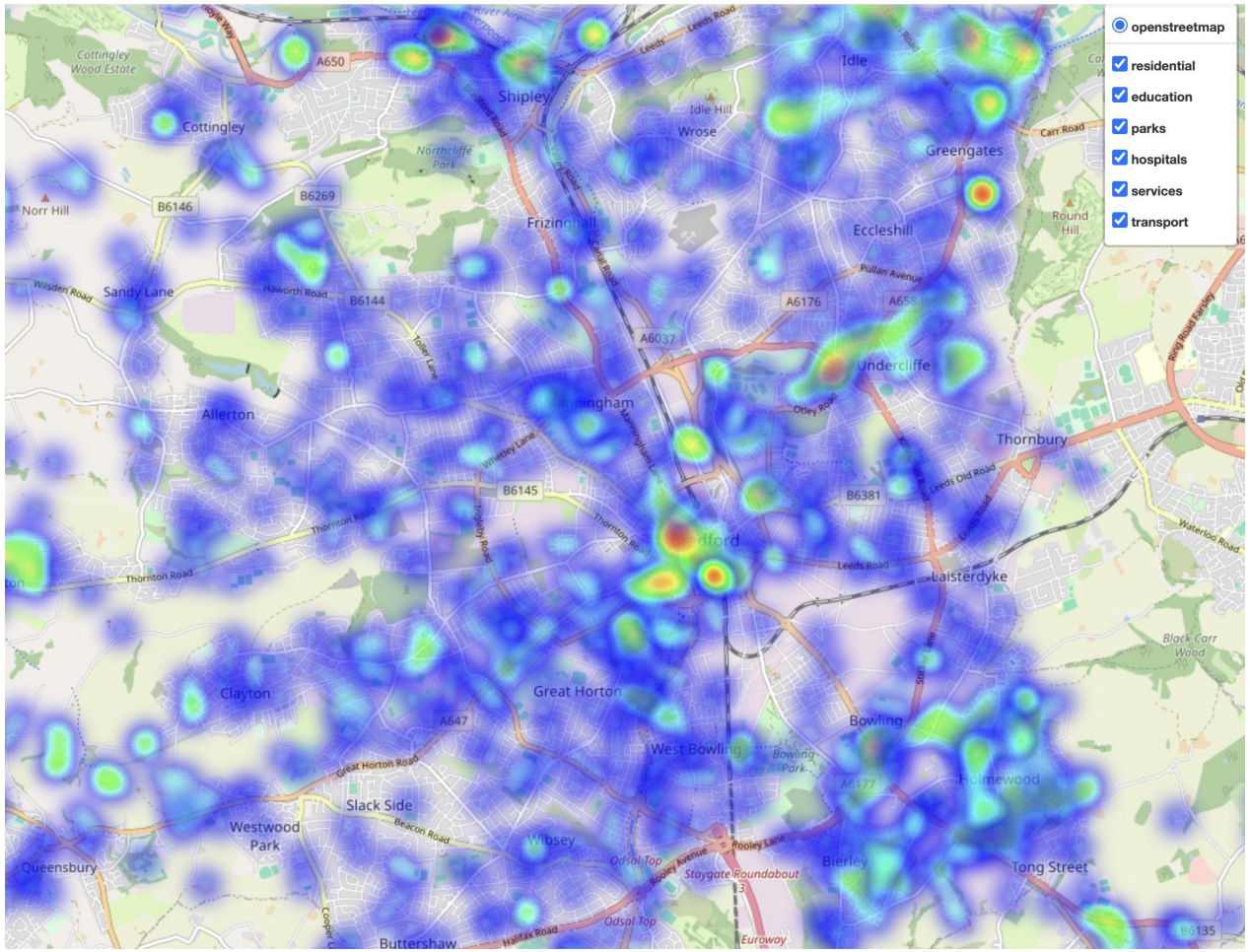
High Concentration Areas

Central Bradford: Good service and transport accessibility.

Shipley and Manningham: Ideal for families, with good parks and schools.

Moderate to Low Concentration Areas

Keighley and Denholme: Limited amenities and transport, impacting neighborhood viability.



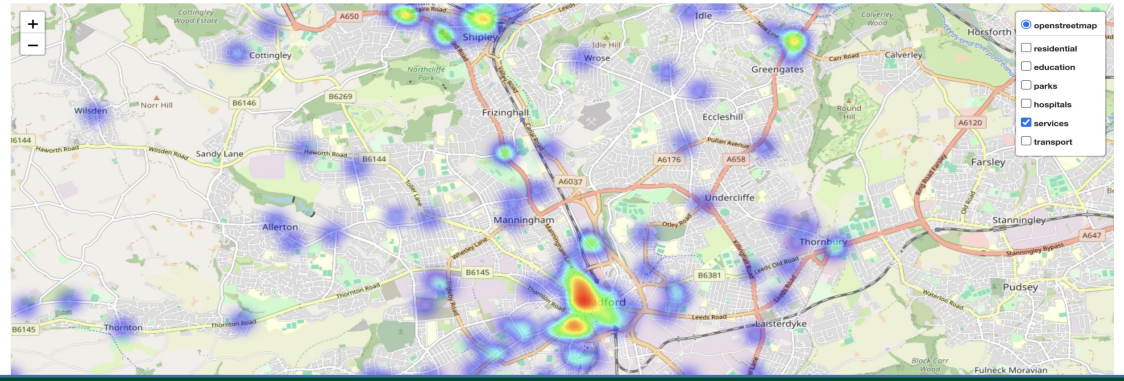
4 Findings

4.2 Heatmaps

Service Distribution

Services and education are well-distributed, reducing travel times.

Healthcare is centralized, disadvantaging outer areas.

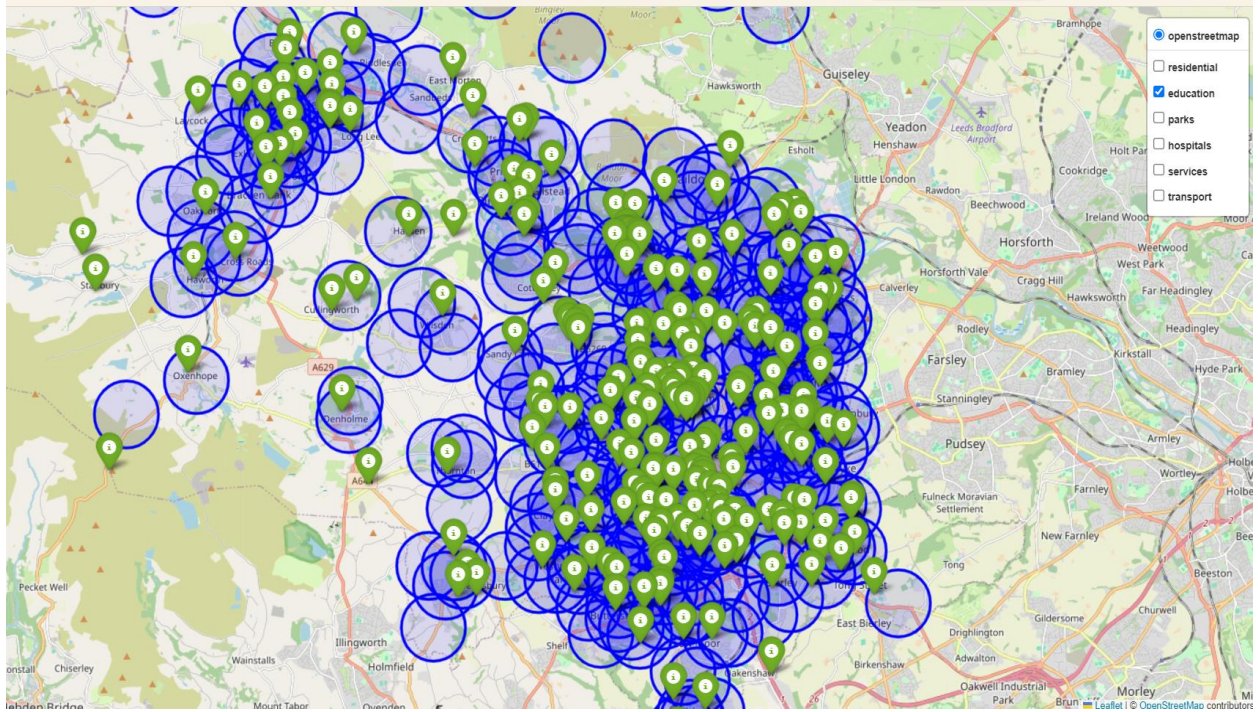


Strategic Implications

- Enhance accessibility in outskirts for equitable service distribution.
- Improve transport in underserved areas to extend 20-minute neighborhood reach.
- Develop green spaces in low-amenity areas to boost quality of life.

4 Findings

4.3 Interactive Map

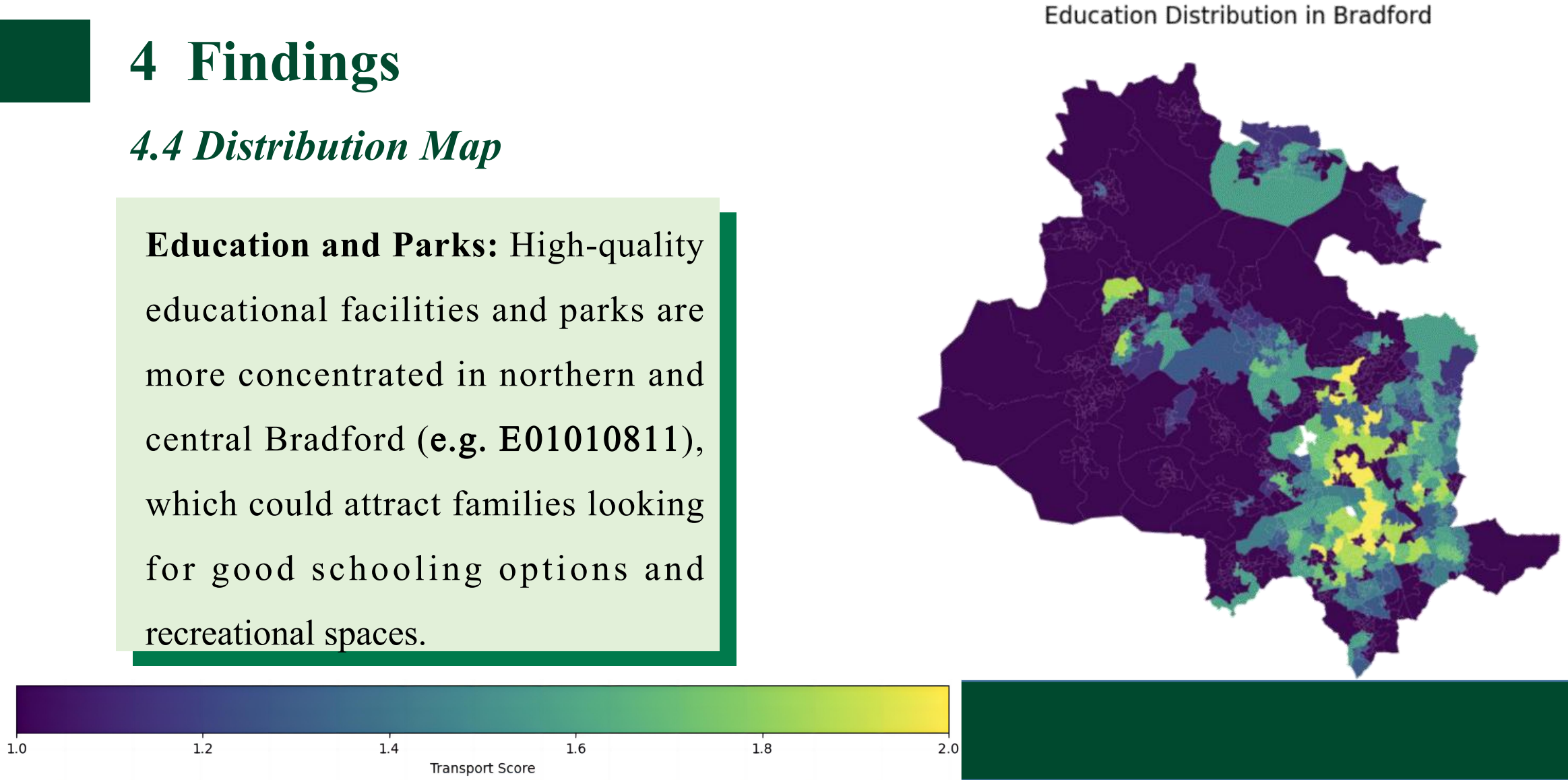


- **Central Bradford:** Meets amenity standards.
- **Amenity Distribution:** Central education and healthcare; scattered green spaces.
- **Transport & Services:** Adequate in center, insufficient in periphery.
- **Development Potential:** Targeted enhancements needed in outer areas.

4 Findings

4.4 Distribution Map

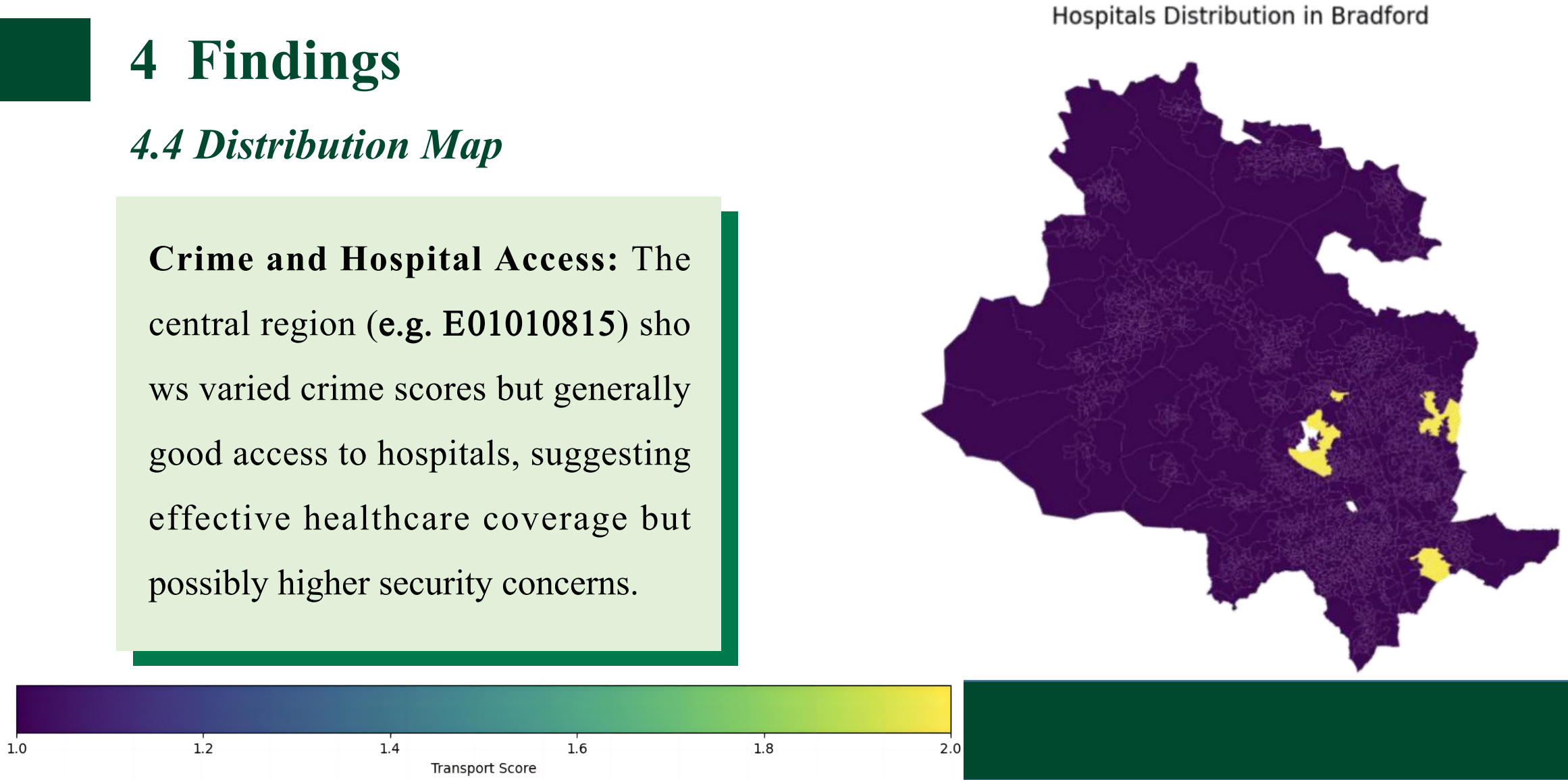
Education and Parks: High-quality educational facilities and parks are more concentrated in northern and central Bradford (**e.g. E01010811**), which could attract families looking for good schooling options and recreational spaces.



4 Findings

4.4 Distribution Map

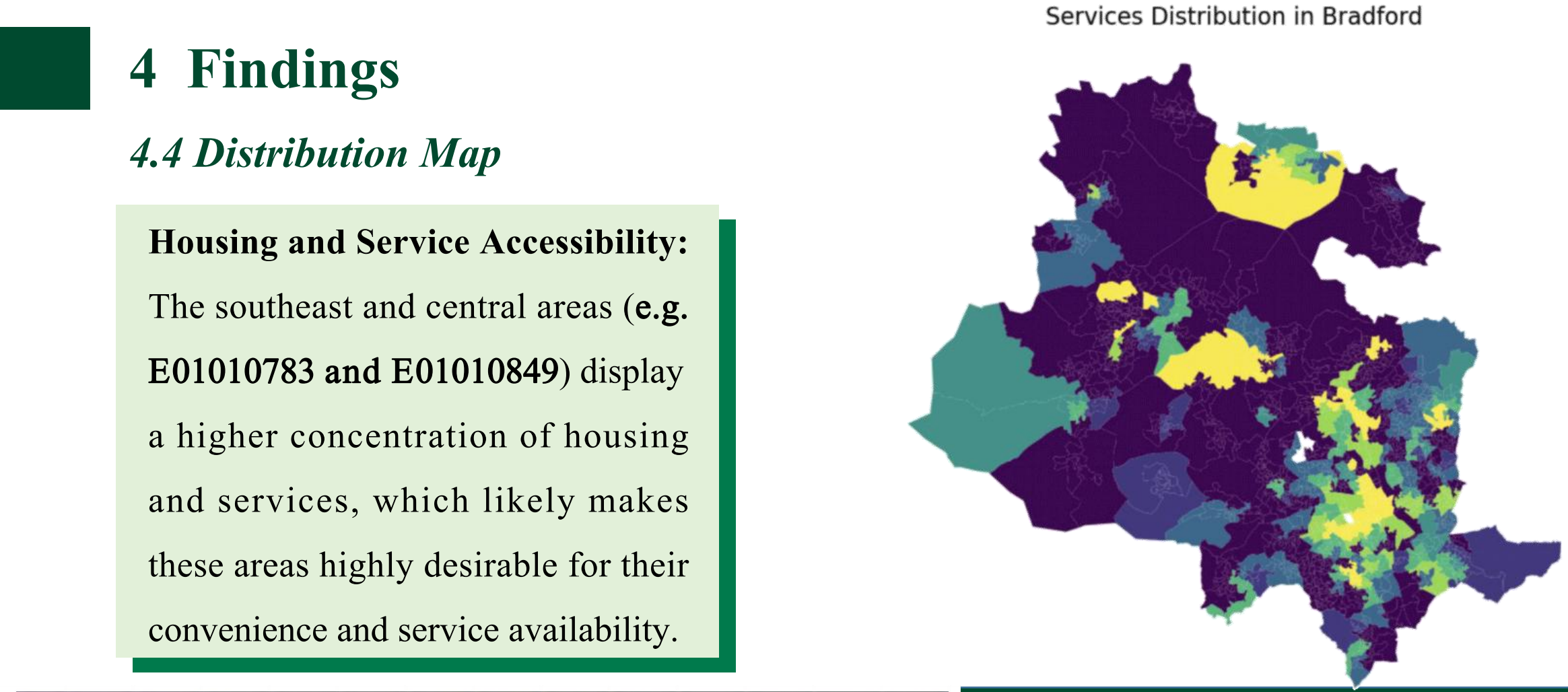
Crime and Hospital Access: The central region (**e.g. E01010815**) sho ws varied crime scores but generally good access to hospitals, suggesting effective healthcare coverage but possibly higher security concerns.



4 Findings

4.4 Distribution Map

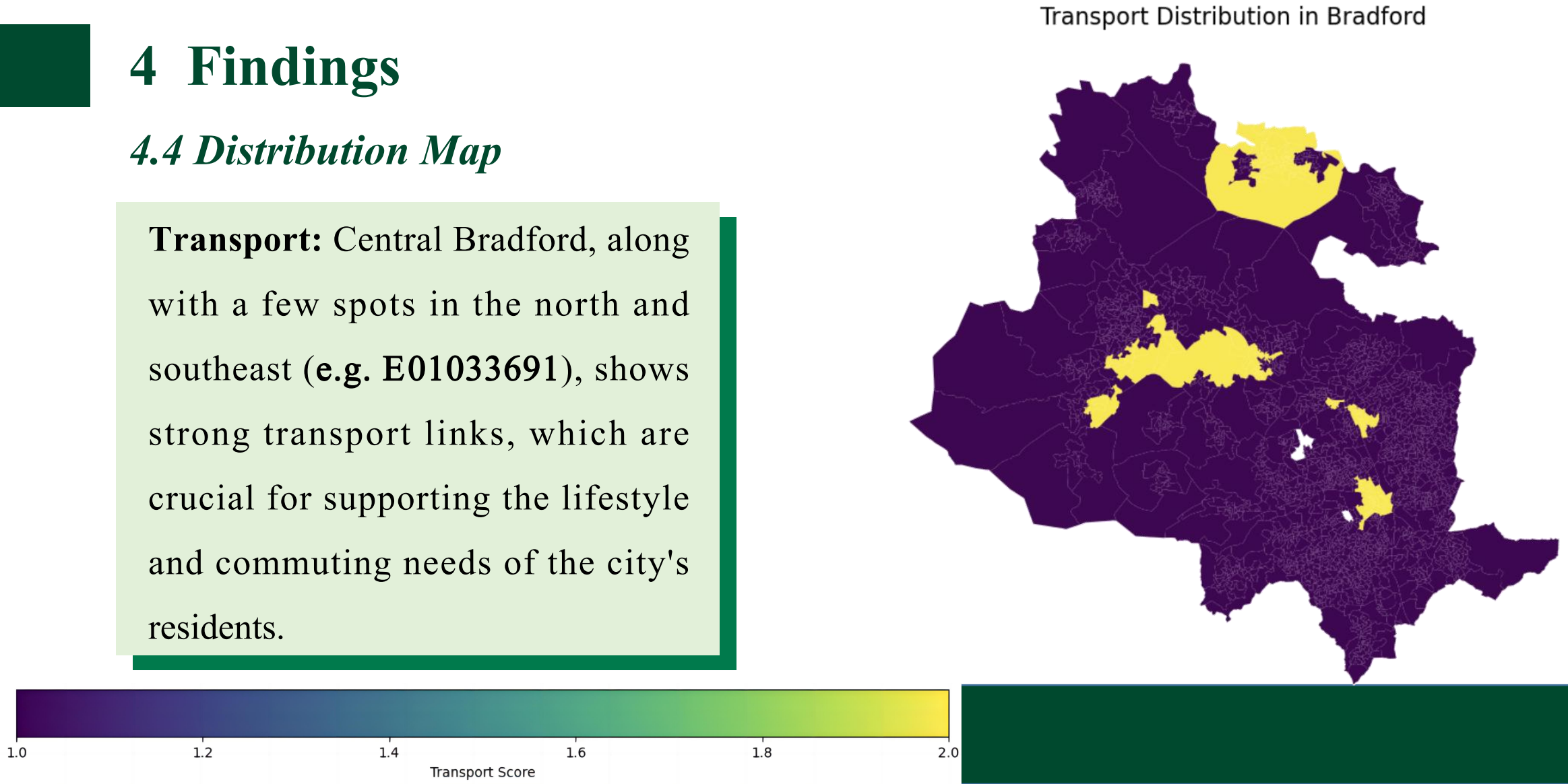
Housing and Service Accessibility: The southeast and central areas (**e.g. E01010783 and E01010849**) display a higher concentration of housing and services, which likely makes these areas highly desirable for their convenience and service availability.



4 Findings

4.4 Distribution Map

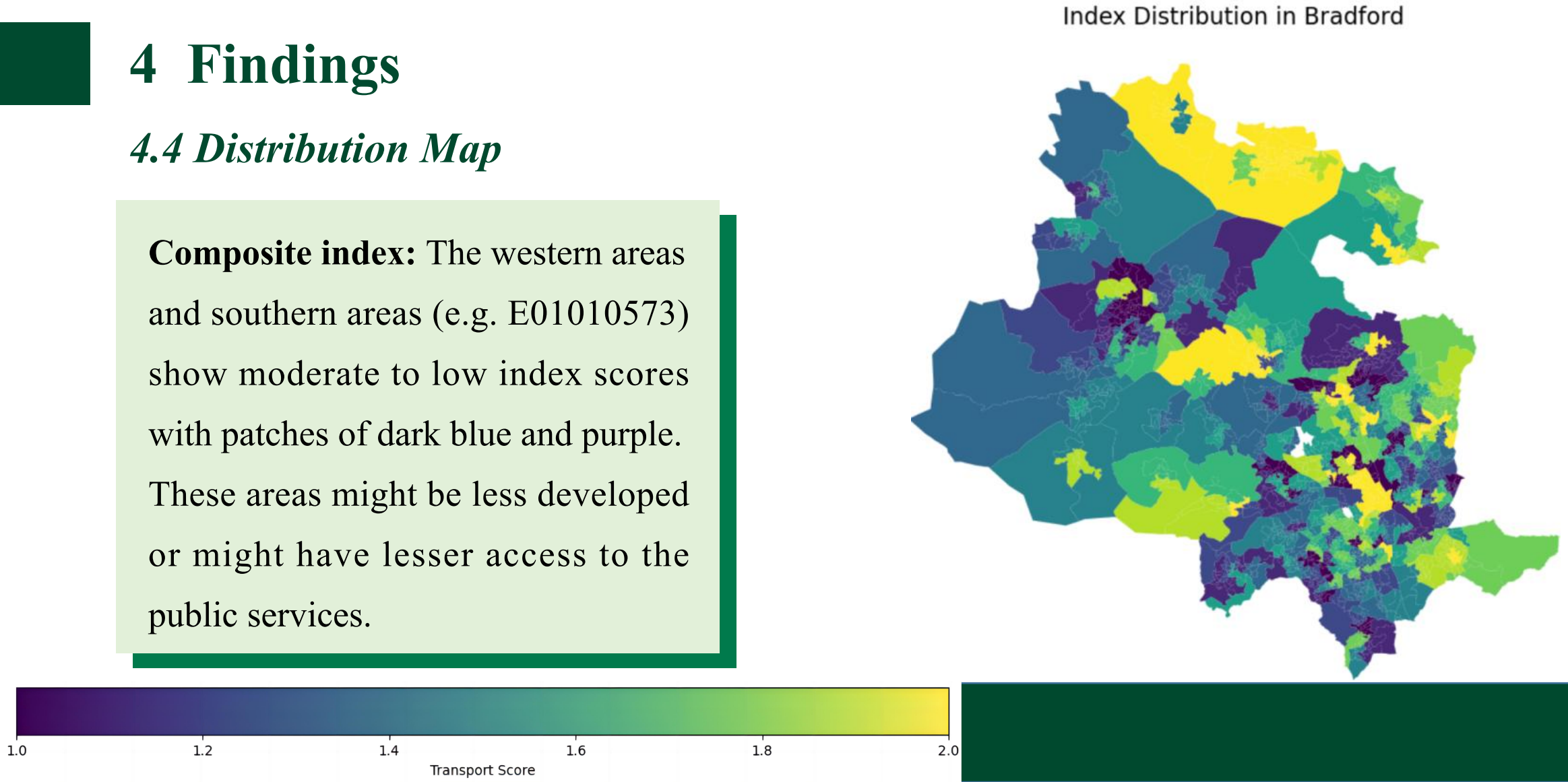
Transport: Central Bradford, along with a few spots in the north and southeast (**e.g. E01033691**), shows strong transport links, which are crucial for supporting the lifestyle and commuting needs of the city's residents.



4 Findings

4.4 Distribution Map

Composite index: The western areas and southern areas (e.g. E01010573) show moderate to low index scores with patches of dark blue and purple. These areas might be less developed or might have lesser access to the public services.



5 Solutions and Limitations

01 Solutions to build 20-minute neighbourhood

- **For some areas like E01010854 and E01010863,** governments can create more necessary services and facilities within convenient walking distance, in order to encourage residents to adopt healthier lifestyles and increase opportunities for physical exercise.
- **For some areas like E01010684 and E01010711,** governments can promote local economic development, and attract more business investment by conducting some policies

02 Limitations

Cluster analysis helps in the decision-making process by grouping areas based on amenities.
Visualizations such as heat maps and interactive maps enhance the presentation of results and make data easier to interpret.

Cluster analysis using K-means may oversimplify complex relationships between different amenities and attributes.
The solution does not address potential biases or limitations in the data collection process that could affect the accuracy of the results.

6 Future work

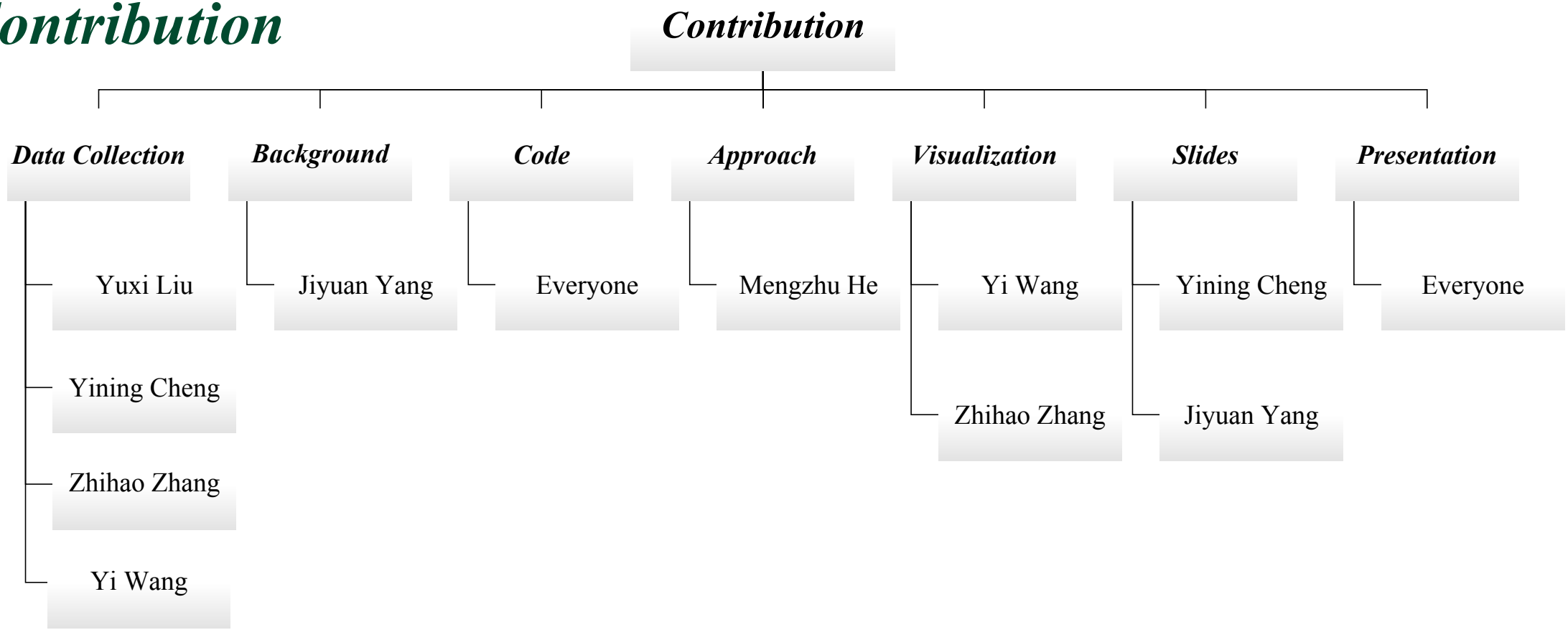
Extensions to Solution

- Incorporate social data: Incorporating social data such as demographics, income levels, and lifestyle preferences can provide a more comprehensive understanding of residents with different backgrounds.
- Machine learning algorithms: Use machine learning algorithms for predictive modeling and anomaly detection.

Approaches

- Community Engagement: Working with local communities to gather feedback on the availability and accessibility of amenities can provide valuable insights for future analyses.
- Predictive Modeling: Developing predictive models to predict changes in facility distribution and demand based on various factors can enhance the decision-making process.

Contribution



References

Bowie, P. "The English Indices of Deprivation 2019 (IoD2019)." GOV. UK (2019).

Gower, Alexa, and Carl Grodach. "Planning innovation or city branding? Exploring how cities operationalise the 20-minute neighbourhood concept." Urban policy and research 40.1 (2022): 36-52.

Helmreich, B., Hilliges, R., Schriewer, A. and Horn, H. 2010. Runoff pollutants of a highly trafficked urban road – Correlation analysis and seasonal influences. Chemosphere (Oxford). 80(9), pp.991–997.

Lovelace, R., Tennekes, M. and Carlino, D. 2022. ClockBoard: A zoning system for urban analysis.

Matsuoka, R.H. and Kaplan, R. 2008. People needs in the urban landscape: Analysis of Landscape And Urban Planning contributions. Landscape and urban planning. 84(1), pp.7–19.

Meng, Y., Li, Y., Liu, H. and Huang, Y. 2021. Study on Evaluation of Urban and Rural High-quality Living Space Based on Accessibility Analysis——Taking Shahe City, Hebei Province as an example. IOP conference series. Earth and environmental science. 769(2).

References

Oostenbach, Laura Helena, et al. "Do 20-minute neighbourhoods moderate associations between work and commute hours with food consumption?." Public Health Nutrition 26.10 (2023): 2026-2035.

Şanta, A.-M.I. 2022. Prosumers—A New Mindset for Citizens in Smart Cities. Smart cities (Basel). 5(4), pp.1409–1420.

Witten, Karen, et al. "Neighbourhood access to open spaces and the physical activity of residents: a national study." Preventive medicine 47.3 (2008): 299-303.

Wu, W. and Divigalpitiya, P. 2023. Availability and Adequacy of Facilities in 15 Minute Community Life Circle Located in Old and New Communities. Smart cities (Basel). 6(5), pp.2176–2195.

Yang, Guannan, et al. "Comparison of spatial approaches to assess the effect of residing in a 20-minute neighbourhood on body mass index." Spatial and spatio-temporal epidemiology 43 (2022): 100546.

UNIVERSITY OF LEEDS

Group14

Thank you

May. 7, 2024

In [ ]:
```
import osmnx as ox
import pandas as pd

# Define the place and tag specification for OpenStreetMap data retrieval
place = "Bradford, United Kingdom"

# Specifying the tags to retrieve different types of amenities and areas
tags = {
    'residential': {'landuse': 'residential'},  # Residential areas
    'education': {'amenity': ['school', 'college', 'university']},  # Educational institutions
    'hospitals': {'amenity': 'hospital'},  # Hospitals
    'parks': {'leisure': 'park'},  # Green spaces and parks
    'markets': {'shop': 'supermarket', 'landuse': 'commercial'},  # Markets and supermarkets
    'restaurants': {'amenity': 'restaurant'},  # Restaurants
    'leisure': {'amenity': ['bar', 'cinema', 'nightclub', 'casino', 'theatre']},  # Bars, cinemas, etc.
    'transport': {'amenity': ['bus_station', 'taxi'], 'railway': 'station'}  # Public transport stations
}

# Retrieve and organize data into separate DataFrames
df = {}

for key, tag in tags.items():
    # Retrieve the geometries based on the specified tags
    geometries = ox.geometries_from_place(place, tag)

    # If data is found, calculate the centroids and lat/lng
    if not geometries.empty:
        geometries['centroid'] = geometries.centroid
        geometries['longitude'] = geometries['centroid'].x
        geometries['latitude'] = geometries['centroid'].y
        df[key] = geometries
    else:
        df[key] = pd.DataFrame()  # Create an empty DataFrame if no data is found

residential=df['residential']
education=df['education']
parks=df['parks']
hospitals=df['hospitals']
```

```
services = pd.concat([df['markets'], df['leisure'], df['restaurants']], ignore_index=True)
transport=df['transport']
```

```
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
```

```
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:24: FutureWarning: The `geometries` module and `geometr
ies_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these instead. Th
e `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\111\AppData\Local\Temp\ipykernel_7792\2521463026.py:28: UserWarning: Geometry is in a geographic CRS. Resul
ts from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS before
this operation.

  geometries['centroid'] = geometries.centroid
```

In [ ]:

```
residential=residential[['name','geometry','longitude','latitude']]
education=education[['name','geometry','longitude','latitude']]
parks=parks[['name','geometry','longitude','latitude']]
hospitals=hospitals[['name','geometry','longitude','latitude']]
services = services[['name','geometry','longitude','latitude']]
transport=transport[['name','geometry','longitude','latitude']]
```

In [ ]:
```
import geopandas as gpd

# Assuming your GeoDataFrames are already converted to a CRS that uses meters (if not, convert them)
residential = residential.to_crs(epsg=3857)
education = education.to_crs(epsg=3857)
parks = parks.to_crs(epsg=3857)
hospitals = hospitals.to_crs(epsg=3857)
services = services.to_crs(epsg=3857)
transport = transport.to_crs(epsg=3857)

# Create a buffer of 800 meters around each residential area
residential['geometry_buffered'] = residential.geometry.buffer(800)

# Use this buffer as a geometry for spatial joins
buffered_residential = gpd.GeoDataFrame(residential, geometry='geometry_buffered')

# Function to count facilities within the residential buffer
def count_facilities(residential_df, facility_df):
    # Perform spatial join
    joined = gpd.sjoin(residential_df[['geometry_buffered']], facility_df, how='left', op='intersects', rsuffix='ri
    # Count facilities for each residential entry
    counts = joined.groupby(joined.index).size()
    return counts.reindex(residential_df.index, fill_value=0)

# Calculate counts for each facility type and add as new columns
residential['education'] = count_facilities(buffered_residential, education)
residential['parks'] = count_facilities(buffered_residential, parks)
residential['hospitals'] = count_facilities(buffered_residential, hospitals)
residential['services'] = count_facilities(buffered_residential, services)
residential['transport'] = count_facilities(buffered_residential, transport)

# Remove the buffered geometry if it's no longer needed
residential.drop(columns=['geometry_buffered'], inplace=True)

# Convert the CRS back to geographic coordinates if necessary
residential = residential.to_crs(epsg=4326)
```

```
D:\Leeds\software\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter is depr
ecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
D:\Leeds\software\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter is depr
ecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
D:\Leeds\software\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter is depr
ecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
D:\Leeds\software\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter is depr
ecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
D:\Leeds\software\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter is depr
ecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [ ]:

```python
lsoa= pd.read_csv('Output_Areas_2021_EW_BFC_V8_-1522627075007521410.csv')
lsoa = lsoa[lsoa['LSOA21NM'].str.contains('Bradford', case=False, na=False)]
lsoa = lsoa.drop_duplicates(subset='LSOA21CD', keep='first')
lsoa=lsoa[['LSOA21CD','LSOA21NM','LAT','LONG']]
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG
50292	E01010573	Bradford 013A	53.85777	-1.79493
50294	E01010576	Bradford 016E	53.85259	-1.75631
50297	E01010568	Bradford 016A	53.84526	-1.75196
50299	E01010569	Bradford 016B	53.84155	-1.78365
50301	E01010570	Bradford 018A	53.84833	-1.79227

In [ ]:

```python
import pandas as pd
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    # Convert to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
```

```
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371  # Radius of Earth in kilometers
    return c * r

# Calculate distance from each residential point to each LSOA center
for index, point in residential.iterrows():
    min_dist = float('inf')
    closest_lsoa = None
    for index2, lsoa_point in lsoa.iterrows():
        dist = haversine(point['latitude'], point['longitude'], lsoa_point['LAT'], lsoa_point['LONG'])
        if dist < min_dist:
            min_dist = dist
            closest_lsoa = lsoa_point['LSOA21NM']
    residential.at[index, 'LSOA21NM'] = closest_lsoa
```

In [ ]:

```
residential_counts = residential.groupby('LSOA21NM').size().reset_index(name='residential')
residential_average = residential.groupby('LSOA21NM')[['education', 'parks', 'hospitals', 'services', 'transport']]
residential_combined = residential_average.merge(residential_counts, on='LSOA21NM')
lsoa = lsoa.merge(residential_combined, on='LSOA21NM', how='left')
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential
0	E01010573	Bradford 013A	53.85777	-1.79493	NaN	NaN	NaN	NaN	NaN	NaN
1	E01010576	Bradford 016E	53.85259	-1.75631	NaN	NaN	NaN	NaN	NaN	NaN
2	E01010568	Bradford 016A	53.84526	-1.75196	NaN	NaN	NaN	NaN	NaN	NaN
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0
4	E01010570	Bradford 018A	53.84833	-1.79227	NaN	NaN	NaN	NaN	NaN	NaN

In [ ]:

```
lsoa = lsoa.dropna()
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0
5	E01010574	Bradford 013B	53.85102	-1.75998	4.0	4.0	1.0	14.0	1.0	1.0
9	E01010575	Bradford 013C	53.85597	-1.77122	2.0	2.5	1.0	1.0	1.0	2.0
10	E01010583	Bradford 018B	53.84318	-1.80077	3.0	6.0	1.0	1.0	1.0	1.0
11	E01010586	Bradford 022A	53.84141	-1.87549	1.0	1.0	1.0	1.0	1.0	5.0

In [ ]:

```
crime=pd.read_csv('2023-01-west-yorkshire-street.csv')
crime=crime[['Crime ID','LSOA code','LSOA name']]
crime = crime.dropna()
crime = crime[crime['LSOA name'].str.contains('Bradford', case=False, na=False)]
crime.head()
```

Out[ ]:

	Crime ID	LSOA code	LSOA name
4	04c76965a81915c19060c589c799b82ffe073c723b7ee9...	E01010647	Bradford 001B
5	1ca06834467dca272744b7784b4ba746cef214520b12a1...	E01010647	Bradford 001B
6	01c9ad6ef83394ac7ab34f29c68720193c724ccb60115e...	E01010647	Bradford 001B
9	fe48de8df5a77a1111b097f37e55478daa4a5108d307a2...	E01010692	Bradford 001D
10	ced28ddcd79ba4563b5551bb14bea93a6b124d276ebfc6...	E01010692	Bradford 001D

In [ ]:

```
crime_counts = crime.groupby('LSOA code').size()/31
crime_counts_df = crime_counts.reset_index(name='Crime')

lsoa = lsoa.merge(crime_counts_df, left_on='LSOA21CD', right_on='LSOA code', how='left')
lsoa = lsoa.drop(columns=['LSOA code'])
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	Crime
0	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.580645
1	E01010574	Bradford 013B	53.85102	-1.75998	4.0	4.0	1.0	14.0	1.0	1.0	0.032258
2	E01010575	Bradford 013C	53.85597	-1.77122	2.0	2.5	1.0	1.0	1.0	2.0	0.225806
3	E01010583	Bradford 018B	53.84318	-1.80077	3.0	6.0	1.0	1.0	1.0	1.0	0.451613
4	E01010586	Bradford 022A	53.84141	-1.87549	1.0	1.0	1.0	1.0	1.0	5.0	0.096774

In [ ]:

```
house=pd.read_csv('PropertyData_prices_asking_prices_2024-04-30.csv')

house=house[['Postcode','Asking price','Lat','Lng']]
house = house[house['Postcode'].str.contains('BD', case=False, na=False)]
house.head()
```

Out[ ]:

	Postcode	Asking price	Lat	Lng
0	BD18 3DL	325000	53.830395	-1.783322
1	BD4 6QJ	364000	53.766492	-1.718616
2	BD4 6QJ	364000	53.766490	-1.718620
3	BD13 2RZ	485000	53.772585	-1.814986
4	BD13 2RZ	485000	53.772540	-1.815370

In [ ]:

```
# Calculate distance from each house point to each LSOA center
for index, point in house.iterrows():
    min_dist = float('inf')
    closest_lsoa = None
    for index2, lsoa_point in lsoa.iterrows():
        dist = haversine(point['Lat'], point['Lng'], lsoa_point['LAT'], lsoa_point['LONG'])
        if dist < min_dist:
            min_dist = dist
            closest_lsoa = lsoa_point['LSOA21NM']
    house.at[index, 'LSOA21NM'] = closest_lsoa
```

```
house.head()
```

Out[ ]:

	Postcode	Asking price	Lat	Lng	LSOA21NM
0	BD18 3DL	325000	53.830395	-1.783322	Bradford 020C
1	BD4 6QJ	364000	53.766492	-1.718616	Bradford 057A
2	BD4 6QJ	364000	53.766490	-1.718620	Bradford 057A
3	BD13 2RZ	485000	53.772585	-1.814986	Bradford 049H
4	BD13 2RZ	485000	53.772540	-1.815370	Bradford 049H

In [ ]:

```
# Calculating the average asking price per LSOA21NM
average_prices = house.groupby('LSOA21NM')['Asking price'].mean()
average_prices_df = average_prices.reset_index()

# Merging the average prices with the lsoa DataFrame
lsoa = lsoa.merge(average_prices_df, on='LSOA21NM', how='left')

# Displaying the first few rows of the lsoa DataFrame to verify changes
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	Crime	Asking price
0	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.580645	NaN
1	E01010574	Bradford 013B	53.85102	-1.75998	4.0	4.0	1.0	14.0	1.0	1.0	0.032258	NaN
2	E01010575	Bradford 013C	53.85597	-1.77122	2.0	2.5	1.0	1.0	1.0	2.0	0.225806	NaN
3	E01010583	Bradford 018B	53.84318	-1.80077	3.0	6.0	1.0	1.0	1.0	1.0	0.451613	NaN
4	E01010586	Bradford 022A	53.84141	-1.87549	1.0	1.0	1.0	1.0	1.0	5.0	0.096774	NaN

In [ ]:

```
lsoa = lsoa.rename(columns={'Asking price': 'house'})
mean_house = lsoa['house'].mean()
# Replace NaNs in the 'Asking price' column with the calculated mean
lsoa['house'].fillna(mean_house, inplace=True)
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	Crime	hou
0	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.580645	172112.2828
1	E01010574	Bradford 013B	53.85102	-1.75998	4.0	4.0	1.0	14.0	1.0	1.0	0.032258	172112.2828
2	E01010575	Bradford 013C	53.85597	-1.77122	2.0	2.5	1.0	1.0	1.0	2.0	0.225806	172112.2828
3	E01010583	Bradford 018B	53.84318	-1.80077	3.0	6.0	1.0	1.0	1.0	1.0	0.451613	172112.2828
4	E01010586	Bradford 022A	53.84141	-1.87549	1.0	1.0	1.0	1.0	1.0	5.0	0.096774	172112.2828

In [ ]:
```
lsoa = lsoa.rename(columns={'Crime': 'crime'})
```

In [ ]:
```
imd = pd.read_csv('IMD.csv')
imd = imd[imd['LSOA name (2011)'].str.contains('Bradford', case=False, na=False)]
imd.rename(columns={'LSOA code (2011)': 'LSOA21CD'}, inplace=True)
imd.head()
```

Out[ ]:

	LSOA21CD	LSOA name (2011)	Local Authority District code (2019)	Local Authority District name (2019)	Index of Multiple Deprivation (IMD) Rank	Index of Multiple Deprivation (IMD) Decile
10261	E01010568	Bradford 016A	E08000032	Bradford	23,447	8
10262	E01010569	Bradford 016B	E08000032	Bradford	5,775	2
10263	E01010570	Bradford 018A	E08000032	Bradford	29,423	9
10264	E01010571	Bradford 016C	E08000032	Bradford	6,829	3
10265	E01010572	Bradford 016D	E08000032	Bradford	22,844	7

In [ ]:
```
lsoa = lsoa.merge(imd[['LSOA21CD', 'Index of Multiple Deprivation (IMD) Decile']], on='LSOA21CD', how='left')
lsoa.rename(columns={'Index of Multiple Deprivation (IMD) Decile': 'IMD'}, inplace=True)
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	hou
0	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.580645	172112.2828
1	E01010574	Bradford 013B	53.85102	-1.75998	4.0	4.0	1.0	14.0	1.0	1.0	0.032258	172112.2828
2	E01010575	Bradford 013C	53.85597	-1.77122	2.0	2.5	1.0	1.0	1.0	2.0	0.225806	172112.2828
3	E01010583	Bradford 018B	53.84318	-1.80077	3.0	6.0	1.0	1.0	1.0	1.0	0.451613	172112.2828
4	E01010586	Bradford 022A	53.84141	-1.87549	1.0	1.0	1.0	1.0	1.0	5.0	0.096774	172112.2828

In [ ]:

```
lsoa=lsoa.dropna()
lsoa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 254 entries, 0 to 259
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   LSOA21CD     254 non-null    object
 1   LSOA21NM     254 non-null    object
 2   LAT          254 non-null    float64
 3   LONG         254 non-null    float64
 4   education    254 non-null    float64
 5   parks        254 non-null    float64
 6   hospitals    254 non-null    float64
 7   services     254 non-null    float64
 8   transport    254 non-null    float64
 9   residential  254 non-null    float64
 10  crime        254 non-null    float64
 11  house        254 non-null    float64
 12  IMD          254 non-null    float64
dtypes: float64(11), object(2)
memory usage: 27.8+ KB
```

In [ ]:
```
cols = ['education', 'parks', 'hospitals', 'services', 'transport', 'residential', 'crime', 'house']

lsoa_pca=lsoa.copy()

# Convert columns to deciles
for col in cols:
    lsoa_pca[col] = pd.qcut(lsoa_pca[col], 10, labels=False, duplicates='drop') + 1

lsoa_pca.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house	IMD
0	E01010569	Bradford 016B	53.84155	-1.78365	8	5	1	5	1	1	6	4	2.0
1	E01010574	Bradford 013B	53.85102	-1.75998	5	5	1	7	1	1	1	4	8.0
2	E01010575	Bradford 013C	53.85597	-1.77122	2	4	1	1	1	1	2	4	8.0
3	E01010583	Bradford 018B	53.84318	-1.80077	4	7	1	1	1	1	4	4	1.0
4	E01010586	Bradford 022A	53.84141	-1.87549	1	1	1	1	1	4	1	4	10.0

In [ ]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

features = cols + ['IMD']
pca = PCA(n_components=len(features))
principal_components = pca.fit_transform(lsoa_pca[features])
```

In [ ]:
```
# Look at the variance explained by each component
explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[ ]:
```
array([4.36051980e-01, 1.83879534e-01, 1.12541040e-01, 9.99684636e-02,
       6.49057541e-02, 5.35159098e-02, 4.91373195e-02, 3.92770484e-34,
       6.88303425e-35])
```

In [ ]:
```
loadings = pd.DataFrame(pca.components_.T, columns=[f'PC{i+1}' for i in range(pca.n_components)], index=features)

print(loadings)
```

```
                     PC1           PC2           PC3           PC4  \
education   -4.137686e-01  4.896605e-01  3.167637e-01 -1.600037e-01
parks       -1.496858e-01  5.094907e-01 -2.234981e-01  4.942164e-01
hospitals    5.551115e-17 -1.110223e-16 -5.551115e-17 -5.551115e-17
services    -2.653149e-01  4.417571e-01 -3.391631e-01  4.389506e-02
transport   -1.084202e-19 -6.938894e-18  1.110223e-16  1.110223e-16
residential  1.648921e-01 -1.469211e-01 -3.849342e-01  5.104846e-01
crime       -5.776050e-01 -4.764818e-01 -1.788366e-01  3.019296e-01
house        1.471547e-01  2.982014e-02  7.084207e-01  6.126931e-01
IMD          5.945808e-01  2.366274e-01 -2.294800e-01  3.276117e-02

                     PC5           PC6           PC7           PC8  \
education    8.624060e-02  6.641424e-01 -1.206865e-01  2.851109e-17
parks        4.893123e-01 -3.515710e-01 -2.465429e-01 -6.036349e-18
hospitals   -2.081668e-16  5.551115e-17 -9.714451e-17 -9.809451e-01
services    -6.841598e-01 -1.664131e-01  3.489016e-01 -2.303830e-18
transport   -1.110223e-16  0.000000e+00 -5.551115e-17 -1.942850e-01
residential  1.453049e-01  5.775163e-01  4.333789e-01 -1.822674e-17
crime       -2.504517e-01  9.206094e-02 -4.949670e-01  3.115148e-17
house       -3.031213e-01 -7.075754e-02  5.757113e-02  2.100601e-17
IMD         -3.306650e-01  2.461960e-01 -6.056360e-01  7.515333e-17

                     PC9
education   -4.643247e-18
parks        1.406497e-19
hospitals    1.942850e-01
services     7.822659e-18
transport   -9.809451e-01
residential -2.998035e-17
crime        4.358081e-17
house        4.798886e-18
IMD          4.136243e-17
```

In [ ]:
```
weights = pca.components_[0]
print("Weights assigned to each feature:", weights)
```

```
Weights assigned to each feature: [-4.13768558e-01 -1.49685841e-01  5.55111512e-17 -2.65314913e-01
 -1.08420217e-19  1.64892101e-01 -5.77604974e-01  1.47154718e-01
  5.94580781e-01]
```

In [ ]:
```
weighted_scores = lsoa_pca[features].dot(weights)
lsoa_pca['weighted_scores'] = -weighted_scores
lsoa_pca.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house	IMD	w
0	E01010569	Bradford 016B	53.84155	-1.78365	8	5	1	5	1	1	6	4	2.0	
1	E01010574	Bradford 013B	53.85102	-1.75998	5	5	1	7	1	1	1	4	8.0	
2	E01010575	Bradford 013C	53.85597	-1.77122	2	4	1	1	1	1	2	4	8.0	
3	E01010583	Bradford 018B	53.84318	-1.80077	4	7	1	1	1	1	4	4	1.0	
4	E01010586	Bradford 022A	53.84141	-1.87549	1	1	1	1	1	4	1	4	10.0	

In [ ]:
```
lsoa_pca['weighted_scores'] = pd.qcut(lsoa_pca['weighted_scores'], 10, labels=False) + 1
lsoa_pca.head()
```

Out[ ]:

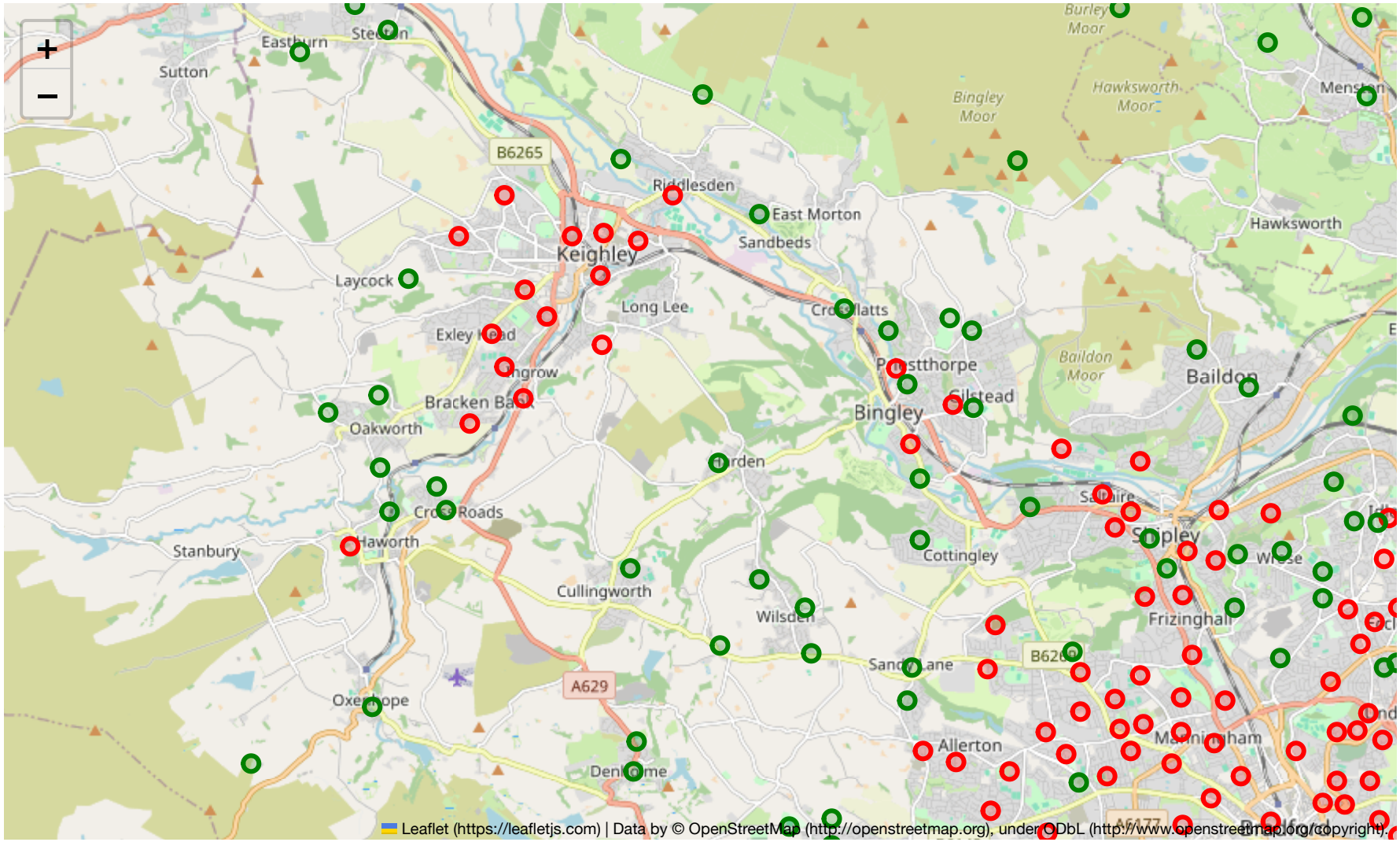
	LSOA21CD	LSOA21NM	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house	IMD	w
0	E01010569	Bradford 016B	53.84155	-1.78365	8	5	1	5	1	1	6	4	2.0	
1	E01010574	Bradford 013B	53.85102	-1.75998	5	5	1	7	1	1	1	4	8.0	
2	E01010575	Bradford 013C	53.85597	-1.77122	2	4	1	1	1	1	2	4	8.0	
3	E01010583	Bradford 018B	53.84318	-1.80077	4	7	1	1	1	1	4	4	1.0	
4	E01010586	Bradford 022A	53.84141	-1.87549	1	1	1	1	1	4	1	4	10.0	

In [ ]:

```
import folium

# Initialize the map centered around the average latitude and longitude of your data
m = folium.Map(location=[lsoa['LAT'].mean(), lsoa['LONG'].mean()], zoom_start=12, tiles='OpenStreetMap')

# Function to determine color based on weighted_average
def get_color(weighted_scores):
    if weighted_scores < 5:
        return 'green'
    else:
        return 'red'

# Add points to the map
for index, row in lsoa_pca.iterrows():
    folium.CircleMarker(
        location=(row['LAT'], row['LONG']),
        radius=5,
        color=get_color(row['weighted_scores']),
        fill=True,
        fill_color=get_color(row['weighted_scores']),
        popup=f"Weighted Average: {row['weighted_scores']:.2f}"
    ).add_to(m)

# Display the map
```

```
m  # Save the map to an HTML file
# Display in Jupyter Notebook: display(m) if you are using Jupyter
```

Out[ ]:



In [ ]:

In [ ]:
```
import osmnx as ox
import pandas as pd

# Define the place and tag specification for OpenStreetMap data retrieval
place = "Bradford, United Kingdom"

# Specifying the tags to retrieve different types of amenities and areas
tags = {
    'residential': {'landuse': 'residential'},  # Residential areas
    'education': {'amenity': ['school', 'college', 'university']},  # Educational institutions
    'hospitals': {'amenity': 'hospital'},  # Hospitals
    'parks': {'leisure': 'park'},  # Green spaces and parks
    'markets': {'shop': 'supermarket', 'landuse': 'commercial'},  # Markets and supermarkets
    'restaurants': {'amenity': 'restaurant'},  # Restaurants
    'leisure': {'amenity': ['bar', 'cinema', 'nightclub', 'casino', 'theatre']},  # Bars, cinemas, etc.
    'transport': {'amenity': ['bus_station', 'taxi'], 'railway': 'station'}  # Public transport stations
}

# Retrieve and organize data into separate DataFrames
df = {}

for key, tag in tags.items():
    # Retrieve the geometries based on the specified tags
    geometries = ox.geometries_from_place(place, tag)

    # If data is found, calculate the centroids and lat/lng
    if not geometries.empty:
        geometries['centroid'] = geometries.centroid
        geometries['longitude'] = geometries['centroid'].x
        geometries['latitude'] = geometries['centroid'].y
        df[key] = geometries
    else:
        df[key] = pd.DataFrame()  # Create an empty DataFrame if no data is found

residential=df['residential']
education=df['education']
parks=df['parks']
hospitals=df['hospitals']
```

```
services = pd.concat([df['markets'], df['leisure'], df['restaurants']], ignore_index=True)
transport=df['transport']
```

```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:28: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:24: FutureWarning: The `geometries` module a
nd `geometries_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these
instead. The `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:28: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:24: FutureWarning: The `geometries` module a
nd `geometries_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these
instead. The `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:28: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  geometries['centroid'] = geometries.centroid
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:24: FutureWarning: The `geometries` module a
nd `geometries_from_X` functions have been renamed the `features` module and `features_from_X` functions. Use these
instead. The `geometries` module and function names are deprecated and will be removed in the v2.0.0 release.
  geometries = ox.geometries_from_place(place, tag)
C:\Users\Administrator\AppData\Local\Temp\ipykernel_11820\2521463026.py:28: UserWarning: Geometry is in a geographic
CRS. Results from 'centroid' are likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected
CRS before this operation.

  geometries['centroid'] = geometries.centroid
```

In [ ]:

```
residential=residential[['name','geometry','longitude','latitude']]
education=education[['name','geometry','longitude','latitude']]
parks=parks[['name','geometry','longitude','latitude']]
hospitals=hospitals[['name','geometry','longitude','latitude']]
services = services[['name','geometry','longitude','latitude']]
transport=transport[['name','geometry','longitude','latitude']]
```

In [ ]:
```
import geopandas as gpd

# Assuming your GeoDataFrames are already converted to a CRS that uses meters (if not, convert them)
residential = residential.to_crs(epsg=3857)
education = education.to_crs(epsg=3857)
parks = parks.to_crs(epsg=3857)
hospitals = hospitals.to_crs(epsg=3857)
services = services.to_crs(epsg=3857)
transport = transport.to_crs(epsg=3857)

# Create a buffer of 800 meters around each residential area
residential['geometry_buffered'] = residential.geometry.buffer(800)

# Use this buffer as a geometry for spatial joins
buffered_residential = gpd.GeoDataFrame(residential, geometry='geometry_buffered')

# Function to count facilities within the residential buffer
def count_facilities(residential_df, facility_df):
    # Perform spatial join
    joined = gpd.sjoin(residential_df[['geometry_buffered']], facility_df, how='left', op='intersects', rsuffix='ri
    # Count facilities for each residential entry
    counts = joined.groupby(joined.index).size()
    return counts.reindex(residential_df.index, fill_value=0)

# Calculate counts for each facility type and add as new columns
residential['education'] = count_facilities(buffered_residential, education)
residential['parks'] = count_facilities(buffered_residential, parks)
residential['hospitals'] = count_facilities(buffered_residential, hospitals)
residential['services'] = count_facilities(buffered_residential, services)
residential['transport'] = count_facilities(buffered_residential, transport)

# Remove the buffered geometry if it's no longer needed
residential.drop(columns=['geometry_buffered'], inplace=True)

# Convert the CRS back to geographic coordinates if necessary
residential = residential.to_crs(epsg=4326)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter
is deprecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter
is deprecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter
is deprecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter
is deprecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
C:\ProgramData\anaconda3\Lib\site-packages\IPython\core\interactiveshell.py:3526: FutureWarning: The `op` parameter
is deprecated and will be removed in a future release. Please use the `predicate` parameter instead.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [ ]:

```
lsoa= pd.read_csv('Output_Areas_2021_EW_BFC_V8_-1522627075007521410.csv')
lsoa = lsoa[lsoa['LSOA21NM'].str.contains('Bradford', case=False, na=False)]
lsoa = lsoa.drop_duplicates(subset='LSOA21CD', keep='first')
lsoa=lsoa[['LSOA21CD','LSOA21NM','LAT','LONG']]
lsoa.head()
```

Out[ ]:

	LSOA21CD	LSOA21NM	LAT	LONG
50292	E01010573	Bradford 013A	53.85777	-1.79493
50294	E01010576	Bradford 016E	53.85259	-1.75631
50297	E01010568	Bradford 016A	53.84526	-1.75196
50299	E01010569	Bradford 016B	53.84155	-1.78365
50301	E01010570	Bradford 018A	53.84833	-1.79227

In [ ]:

```
import pandas as pd
import numpy as np

def haversine(lat1, lon1, lat2, lon2):
    # Convert to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])
    dlat = lat2 - lat1
```

```
    dlon = lon2 - lon1
    a = np.sin(dlat/2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon/2)**2
    c = 2 * np.arcsin(np.sqrt(a))
    r = 6371  # Radius of Earth in kilometers
    return c * r

# Calculate distance from each residential point to each LSOA center
for index, point in residential.iterrows():
    min_dist = float('inf')
    closest_lsoa = None
    for index2, lsoa_point in lsoa.iterrows():
        dist = haversine(point['latitude'], point['longitude'], lsoa_point['LAT'], lsoa_point['LONG'])
        if dist < min_dist:
            min_dist = dist
            closest_lsoa = lsoa_point['LSOA21CD']
    residential.at[index, 'LSOA'] = closest_lsoa
```

In [ ]:

```
residential_counts = residential.groupby('LSOA').size().reset_index(name='residential')
residential_average = residential.groupby('LSOA')[['education', 'parks', 'hospitals', 'services', 'transport']].mea
residential_combined = residential_average.merge(residential_counts, on='LSOA')
residential_combined.head()
```

Out[ ]:

	LSOA	education	parks	hospitals	services	transport	residential
0	E01010569	13.0	4.00	1.0	4.0	1.0	1
1	E01010574	4.0	4.00	1.0	14.0	1.0	1
2	E01010575	2.0	2.50	1.0	1.0	1.0	2
3	E01010578	2.0	6.50	1.0	14.0	2.5	2
4	E01010579	1.5	4.25	1.0	1.5	1.0	4

In [ ]:

```
lsoa = lsoa.rename(columns={'LSOA21CD': 'LSOA'})
lsoa = lsoa.rename(columns={'LSOA21NM': 'name'})
lsoa = lsoa.merge(residential_combined, on='LSOA', how='left')
lsoa.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential
0	E01010573	Bradford 013A	53.85777	-1.79493	NaN	NaN	NaN	NaN	NaN	NaN
1	E01010576	Bradford 016E	53.85259	-1.75631	NaN	NaN	NaN	NaN	NaN	NaN
2	E01010568	Bradford 016A	53.84526	-1.75196	NaN	NaN	NaN	NaN	NaN	NaN
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0
4	E01010570	Bradford 018A	53.84833	-1.79227	NaN	NaN	NaN	NaN	NaN	NaN

In [ ]:

```
lsoa.fillna(0, inplace=True)
lsoa.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential
0	E01010573	Bradford 013A	53.85777	-1.79493	0.0	0.0	0.0	0.0	0.0	0.0
1	E01010576	Bradford 016E	53.85259	-1.75631	0.0	0.0	0.0	0.0	0.0	0.0
2	E01010568	Bradford 016A	53.84526	-1.75196	0.0	0.0	0.0	0.0	0.0	0.0
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0
4	E01010570	Bradford 018A	53.84833	-1.79227	0.0	0.0	0.0	0.0	0.0	0.0

In [ ]:

```
crime=pd.read_csv('2023-01-west-yorkshire-street.csv')
crime=crime[['LSOA code','LSOA name']]
crime = crime[crime['LSOA name'].str.contains('Bradford', case=False, na=False)]
crime.head()
```

Out[ ]:

	LSOA code	LSOA name
2	E01010646	Bradford 001A
3	E01010646	Bradford 001A
4	E01010647	Bradford 001B
5	E01010647	Bradford 001B
6	E01010647	Bradford 001B

In [ ]:

```
crime_counts = crime.groupby('LSOA code').size()/31
crime_counts_df = crime_counts.reset_index(name='crime')

lsoa = lsoa.merge(crime_counts_df, left_on='LSOA', right_on='LSOA code', how='left')
lsoa = lsoa.drop(columns=['LSOA code'])
lsoa.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime
0	E01010573	Bradford 013A	53.85777	-1.79493	0.0	0.0	0.0	0.0	0.0	0.0	0.322581
1	E01010576	Bradford 016E	53.85259	-1.75631	0.0	0.0	0.0	0.0	0.0	0.0	0.129032
2	E01010568	Bradford 016A	53.84526	-1.75196	0.0	0.0	0.0	0.0	0.0	0.0	0.129032
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.645161
4	E01010570	Bradford 018A	53.84833	-1.79227	0.0	0.0	0.0	0.0	0.0	0.0	0.258065

In [ ]:

```
house=pd.read_csv('PropertyData_prices_asking_prices_2024-04-30.csv')

house=house[['Postcode','Asking price','Lat','Lng']]
house = house[house['Postcode'].str.contains('BD', case=False, na=False)]
house.head()
```

Out[ ]:

	Postcode	Asking price	Lat	Lng
0	BD18 3DL	325000	53.830395	-1.783322
1	BD4 6QJ	364000	53.766492	-1.718616
2	BD4 6QJ	364000	53.766490	-1.718620
3	BD13 2RZ	485000	53.772585	-1.814986
4	BD13 2RZ	485000	53.772540	-1.815370

In [ ]:

```
# Calculate distance from each house point to each LSOA center
for index, point in house.iterrows():
    min_dist = float('inf')
    closest_lsoa = None
    for index2, lsoa_point in lsoa.iterrows():
        dist = haversine(point['Lat'], point['Lng'], lsoa_point['LAT'], lsoa_point['LONG'])
        if dist < min_dist:
            min_dist = dist
            closest_lsoa = lsoa_point['LSOA']
    house.at[index, 'LSOA'] = closest_lsoa

house.head()
```

Out[ ]:

	Postcode	Asking price	Lat	Lng	LSOA
0	BD18 3DL	325000	53.830395	-1.783322	E01010787
1	BD4 6QJ	364000	53.766492	-1.718616	E01010814
2	BD4 6QJ	364000	53.766490	-1.718620	E01010814
3	BD13 2RZ	485000	53.772585	-1.814986	E01010763
4	BD13 2RZ	485000	53.772540	-1.815370	E01033695

In [ ]:

```
# Calculating the average asking price per LSOA21NM
average_prices = house.groupby('LSOA')['Asking price'].mean()
average_prices_df = average_prices.reset_index()
```

```
# Merging the average prices with the lsoa DataFrame
lsoa = lsoa.merge(average_prices_df, on='LSOA', how='left')
lsoa.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	Asking price
0	E01010573	Bradford 013A	53.85777	-1.79493	0.0	0.0	0.0	0.0	0.0	0.0	0.322581	NaN
1	E01010576	Bradford 016E	53.85259	-1.75631	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	NaN
2	E01010568	Bradford 016A	53.84526	-1.75196	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	160000.0
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.645161	NaN
4	E01010570	Bradford 018A	53.84833	-1.79227	0.0	0.0	0.0	0.0	0.0	0.0	0.258065	NaN

In [ ]:

```
lsoa = lsoa.rename(columns={'Asking price': 'house'})
mean_house = lsoa['house'].mean()
# Replace NaNs in the 'Asking price' column with the calculated mean
lsoa['house'].fillna(mean_house, inplace=True)
lsoa.fillna(0, inplace=True)
lsoa.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house
0	E01010573	Bradford 013A	53.85777	-1.79493	0.0	0.0	0.0	0.0	0.0	0.0	0.322581	173796.297013
1	E01010576	Bradford 016E	53.85259	-1.75631	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	173796.297013
2	E01010568	Bradford 016A	53.84526	-1.75196	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	160000.000000
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.645161	173796.297013
4	E01010570	Bradford 018A	53.84833	-1.79227	0.0	0.0	0.0	0.0	0.0	0.0	0.258065	173796.297013

In [ ]:

```
imd = pd.read_csv('IMD.csv')
imd = imd[imd['LSOA name (2011)'].str.contains('Bradford', case=False, na=False)]
imd.rename(columns={'LSOA code (2011)': 'LSOA'}, inplace=True)
imd.head()
```

Out[ ]:

	LSOA	LSOA name (2011)	Local Authority District code (2019)	Local Authority District name (2019)	Index of Multiple Deprivation (IMD) Rank	Index of Multiple Deprivation (IMD) Decile
10261	E01010568	Bradford 016A	E08000032	Bradford	23,447	8
10262	E01010569	Bradford 016B	E08000032	Bradford	5,775	2
10263	E01010570	Bradford 018A	E08000032	Bradford	29,423	9
10264	E01010571	Bradford 016C	E08000032	Bradford	6,829	3
10265	E01010572	Bradford 016D	E08000032	Bradford	22,844	7

In [ ]:

```
lsoa = lsoa.merge(imd[['LSOA', 'Index of Multiple Deprivation (IMD) Decile']], on='LSOA', how='left')
lsoa.rename(columns={'Index of Multiple Deprivation (IMD) Decile': 'IMD'}, inplace=True)
```

```
lsoa.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house
0	E01010573	Bradford 013A	53.85777	-1.79493	0.0	0.0	0.0	0.0	0.0	0.0	0.322581	173796.297013
1	E01010576	Bradford 016E	53.85259	-1.75631	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	173796.297013
2	E01010568	Bradford 016A	53.84526	-1.75196	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	160000.000000
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.645161	173796.297013
4	E01010570	Bradford 018A	53.84833	-1.79227	0.0	0.0	0.0	0.0	0.0	0.0	0.258065	173796.297013

In [ ]:

```
lsoa=lsoa.dropna()
lsoa.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 308 entries, 0 to 310
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   LSOA         308 non-null    object
 1   name         308 non-null    object
 2   LAT          308 non-null    float64
 3   LONG         308 non-null    float64
 4   education    308 non-null    float64
 5   parks        308 non-null    float64
 6   hospitals    308 non-null    float64
 7   services     308 non-null    float64
 8   transport    308 non-null    float64
 9   residential  308 non-null    float64
 10  crime        308 non-null    float64
 11  house        308 non-null    float64
 12  IMD          308 non-null    float64
dtypes: float64(11), object(2)
memory usage: 33.7+ KB
```

In [ ]:

```
cols = ['education', 'parks', 'hospitals', 'services', 'transport', 'residential', 'crime', 'house','IMD']

data_ranked=lsoa.copy()
data_ranked.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house
0	E01010573	Bradford 013A	53.85777	-1.79493	0.0	0.0	0.0	0.0	0.0	0.0	0.322581	173796.297013
1	E01010576	Bradford 016E	53.85259	-1.75631	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	173796.297013
2	E01010568	Bradford 016A	53.84526	-1.75196	0.0	0.0	0.0	0.0	0.0	0.0	0.129032	160000.000000
3	E01010569	Bradford 016B	53.84155	-1.78365	13.0	4.0	1.0	4.0	1.0	1.0	0.645161	173796.297013
4	E01010570	Bradford 018A	53.84833	-1.79227	0.0	0.0	0.0	0.0	0.0	0.0	0.258065	173796.297013

In [ ]:

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
data_ranked[cols] = scaler.fit_transform(data_ranked[cols])

pca = PCA()
principalComponents = pca.fit_transform(data_ranked[cols])

explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[ ]:

```
array([0.33869836, 0.19252156, 0.11892655, 0.09971548, 0.08888504,
       0.06564703, 0.0448438 , 0.02864703, 0.02211516])
```

In [ ]:

```
weights = pca.components_[:3]
print("Weights assigned to each feature:", weights)
```

```
Weights assigned to each feature: [[ 0.41952656  0.43185275  0.35193039  0.45333044  0.49288567  0.03693262
   0.1899267  -0.10388451 -0.13027948]
 [ 0.11921161 -0.17154523 -0.20830961  0.05033594 -0.20621922 -0.48364269
   0.45839335 -0.31661057 -0.56888566]
 [-0.35163625 -0.06196047 -0.13907993  0.17427796  0.03288792  0.50677798
   0.40678841 -0.59056886  0.22503974]]
```

In [ ]:
```
if weights.shape[0] != 3:
    weights = pca.components_[:3].T  # Ensure weights are correctly taking the top 3 components and transposing

weighted_scores = np.dot(principalComponents[:, :3], weights)
data_ranked['index'] = np.sum(weighted_scores, axis=1)
data_ranked.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	hou
0	E01010573	Bradford 013A	53.85777	-1.79493	-0.916831	-0.922714	-1.659871	-0.636525	-1.858096	-0.659830	-0.479898	-0.0036
1	E01010576	Bradford 016E	53.85259	-1.75631	-0.916831	-0.922714	-1.659871	-0.636525	-1.858096	-0.659830	-0.753872	-0.0036
2	E01010568	Bradford 016A	53.84526	-1.75196	-0.916831	-0.922714	-1.659871	-0.636525	-1.858096	-0.659830	-0.753872	-0.2702
3	E01010569	Bradford 016B	53.84155	-1.78365	2.952931	0.731154	0.188277	0.298116	0.232809	-0.462521	-0.023276	-0.0036
4	E01010570	Bradford 018A	53.84833	-1.79227	-0.916831	-0.922714	-1.659871	-0.636525	-1.858096	-0.659830	-0.571223	-0.0036

In [ ]:
```
final_cols=['education', 'parks', 'hospitals', 'services', 'transport', 'residential', 'crime', 'house','IMD','inde

for col in data_ranked[final_cols].columns:
    data_ranked[col] = pd.qcut(data_ranked[col], 10, labels=False, duplicates='drop') + 1

data_ranked.head()
```

Out[ ]:

	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house	IMD	inde
0	E01010573	Bradford 013A	53.85777	-1.79493	1	1	1	1	1	1	3	4	5	
1	E01010576	Bradford 016E	53.85259	-1.75631	1	1	1	1	1	1	1	4	6	
2	E01010568	Bradford 016A	53.84526	-1.75196	1	1	1	1	1	1	1	3	5	
3	E01010569	Bradford 016B	53.84155	-1.78365	8	6	1	6	1	1	6	4	1	
4	E01010570	Bradford 018A	53.84833	-1.79227	1	1	1	1	1	1	2	4	6	

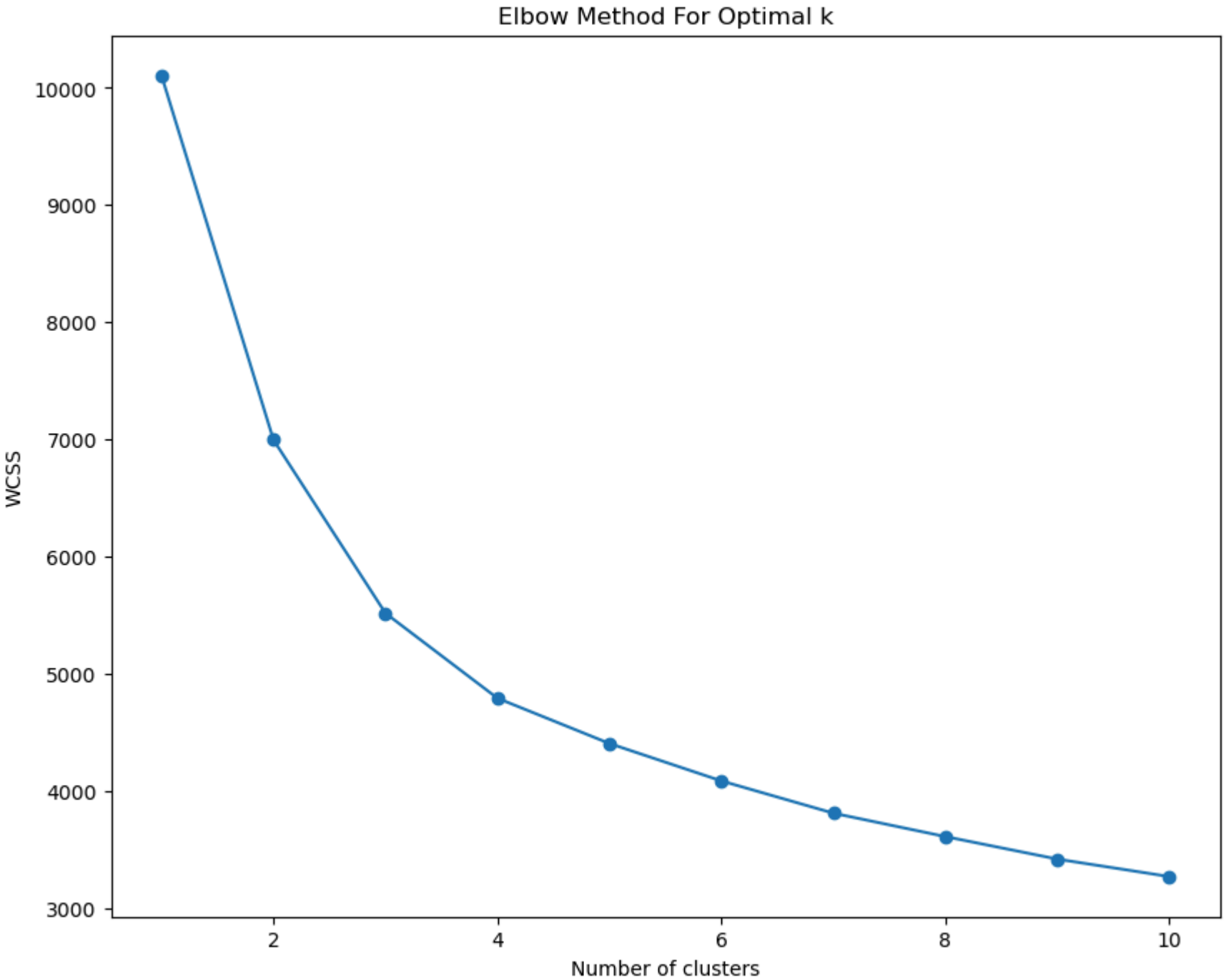
Clustering

In [ ]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler


features = data_ranked[cols]

wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state=42)
    kmeans.fit(features)
    wcss.append(kmeans.inertia_)

plt.figure(figsize=(10, 8))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('WCSS')
plt.show()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```


Elbow Method For Optimal k
10000
9000
8000
7000
6000
5000
4000
3000
WCSS
2
4
6
8
10
Number of clusters

In [ ]:
```
k_optimal = 4
kmeans = KMeans(n_clusters=k_optimal, init='k-means++', random_state=42)
clusters = kmeans.fit_predict(features)

data_ranked['cluster'] = clusters
data_ranked.head()
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\cluster\_kmeans.py:1412: FutureWarning: The default value of `n_i
nit` will change from 10 to 'auto' in 1.4. Set the value of `n_init` explicitly to suppress the warning
  super()._check_params_vs_input(X, default_n_init=10)
```

Out[ ]:

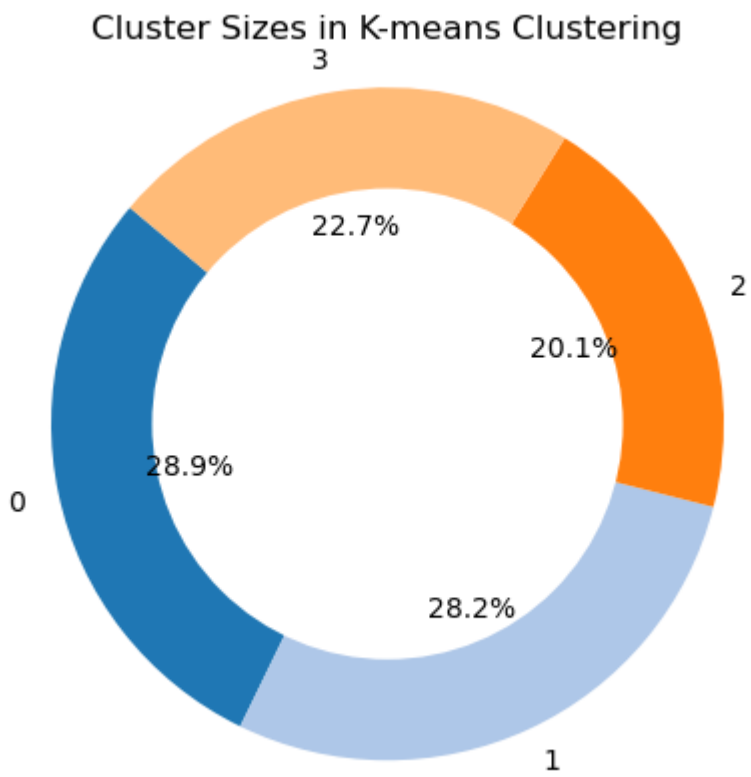
	LSOA	name	LAT	LONG	education	parks	hospitals	services	transport	residential	crime	house	IMD	inde
0	E01010573	Bradford 013A	53.85777	-1.79493	1	1	1	1	1	1	3	4	5	
1	E01010576	Bradford 016E	53.85259	-1.75631	1	1	1	1	1	1	1	4	6	
2	E01010568	Bradford 016A	53.84526	-1.75196	1	1	1	1	1	1	1	3	5	
3	E01010569	Bradford 016B	53.84155	-1.78365	8	6	1	6	1	1	6	4	1	
4	E01010570	Bradford 018A	53.84833	-1.79227	1	1	1	1	1	1	2	4	6	

In [ ]:
```
data_ranked.groupby('cluster').size()
```

Out[ ]:
```
cluster
0    89
1    87
2    62
3    70
dtype: int64
```

Visualisation

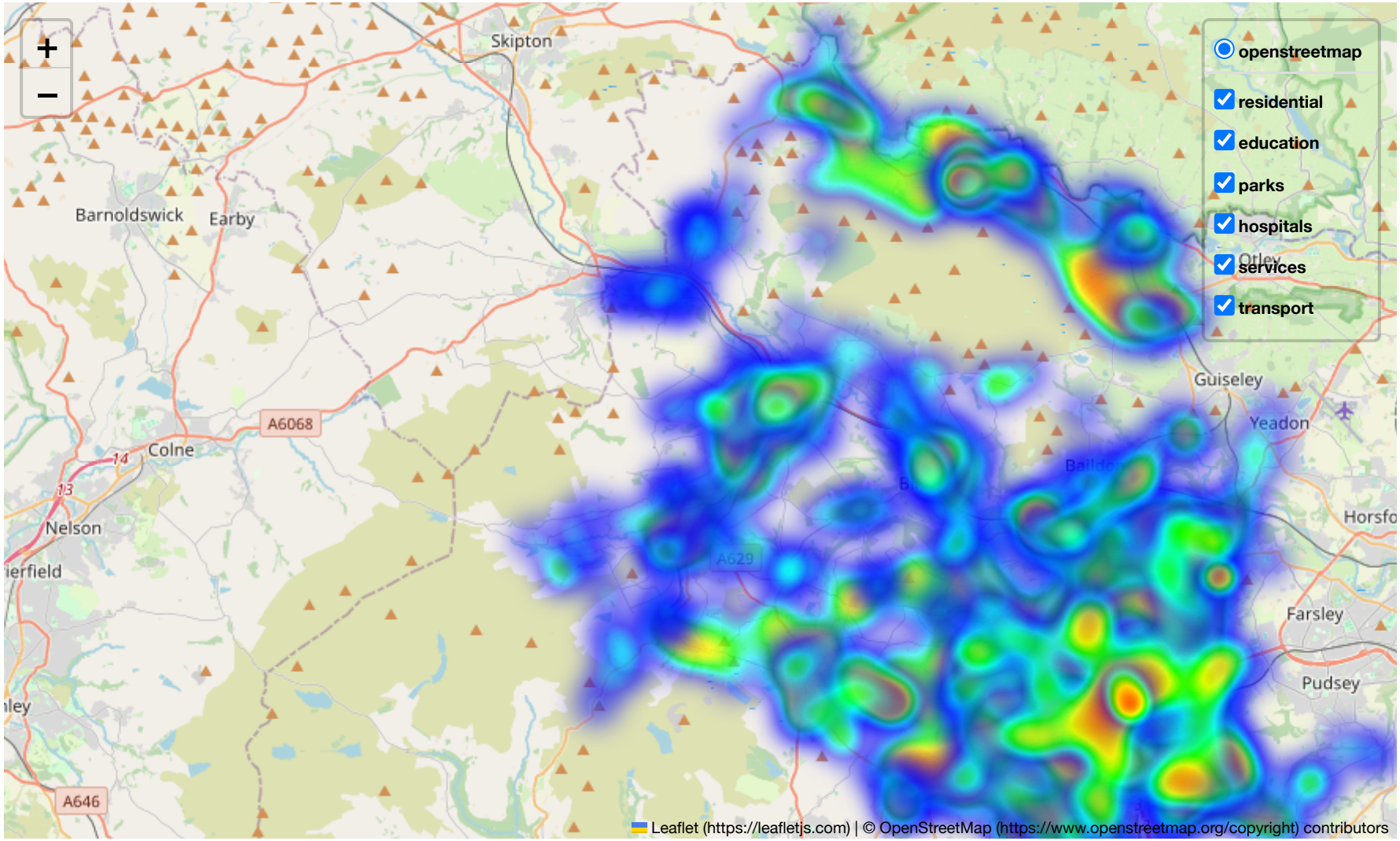
In [ ]:
```
import matplotlib.pyplot as plt

# Assuming 'cluster_sizes' is already calculated from data_ranked.groupby('cluster').size()
cluster_sizes = data_ranked.groupby('cluster').size()

# Create a pie chart
fig, ax = plt.subplots()
wedges, texts, autotexts = ax.pie(cluster_sizes, labels=cluster_sizes.index, autopct='%1.1f%%', startangle=140, col

# Draw a circle at the center of pie to make it a donut
centre_circle = plt.Circle((0,0),0.70,fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

# Equal aspect ratio ensures that pie is drawn as a circle
ax.axis('equal')

plt.title('Cluster Sizes in K-means Clustering')
plt.show()
```



In [ ]:
```
# Generate heat maps about different types of amenities:

import folium
from folium.plugins import HeatMap
from folium.map import LayerControl

# Create a map centered around the average latitude and longitude
map_center = [lsoa['LAT'].mean(), lsoa['LONG'].mean()]
main_map = folium.Map(location=map_center, zoom_start=12, tiles='OpenStreetMap')

# Dictionary to hold data for each type of amenity
facilities = {
    'residential': residential,
    'education': education,
    'parks': parks,
```

```
    'hospitals': hospitals,
    'services': services,
    'transport': transport
}

# Iterate over each amenity type to create separate heat maps
for facility_name, data in facilities.items():
    feature_group = folium.FeatureGroup(name=facility_name, show=True)
    HeatMap(data=data[['latitude', 'longitude']].values.tolist(), radius=15).add_to(feature_group)
    feature_group.add_to(main_map)

# Add layer control to toggle visibility of each amenity heat map
LayerControl(collapsed=False).add_to(main_map)

# Save the map as an HTML file
# main_map.save('amenities_heatmap.html')
# Display the map
main_map
```

Out[ ]:

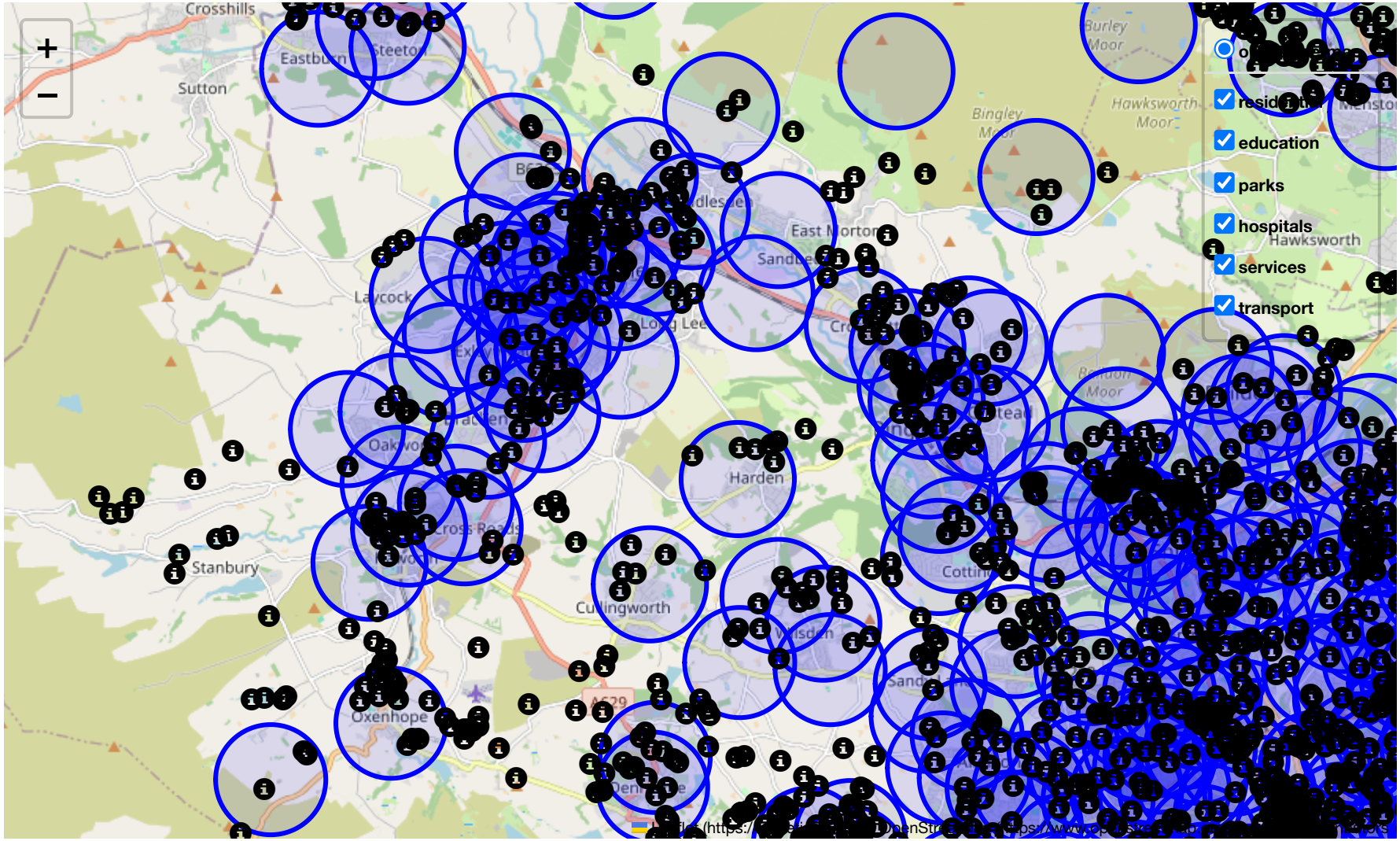


In [ ]:

```
# Show 20-minute walking distance(around 800 meters)

import folium

# Initialize the map at the average coordinates of the LSOA
map = folium.Map(location=[lsoa['LAT'].mean(), lsoa['LONG'].mean()], zoom_start=12, tiles='OpenStreetMap')

# Draw circles with a radius of 800 meters around each point
```

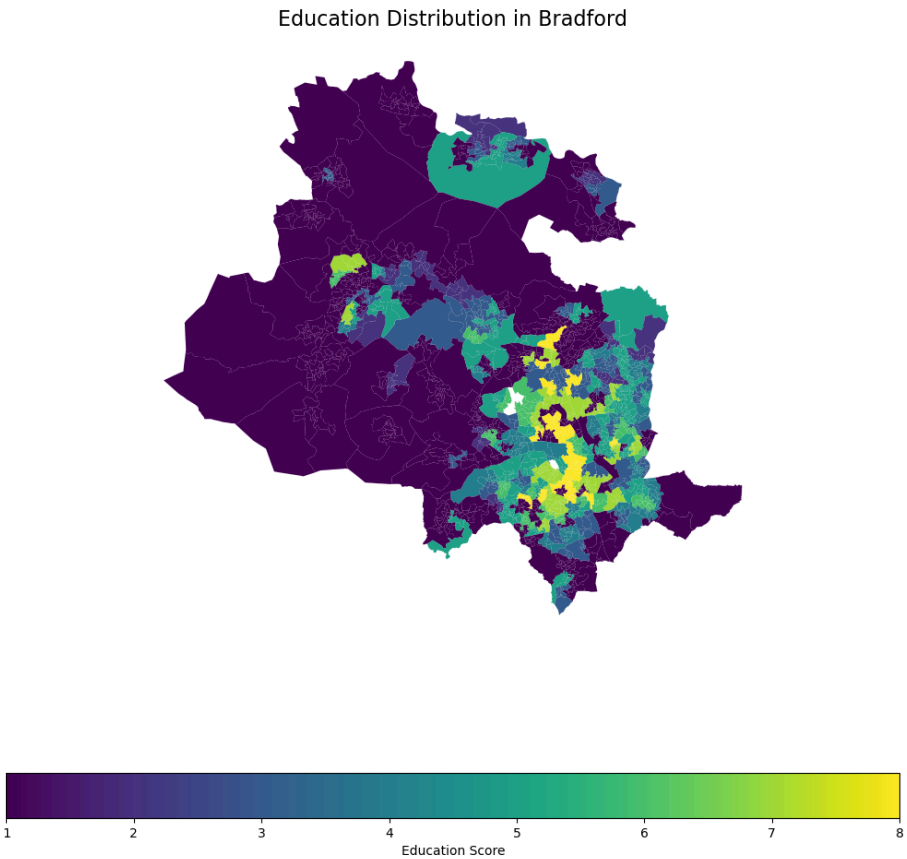
```python
for index, row in lsoa.iterrows():
    folium.Circle(
        location=[row['LAT'], row['LONG']],
        radius=800,  # Radius of 800 meters
        color='blue',
        fill=True,
        fill_opacity=0.1
    ).add_to(map)

# Define color for each type of facility
colors = {
    'residential': 'blue',
    'education': 'green',
    'parks': 'darkgreen',
    'hospitals': 'red',
    'services': 'purple',
    'transport': 'orange'
}

# Add markers for each type of facility
for facility, color in colors.items():
    feature_group = folium.FeatureGroup(name=facility, show=True)  # Create a feature group for each facility
    df = globals()[facility]  # Assuming each DataFrame is named and contains 'longitude' and 'latitude'
    for index, row in df.iterrows():
        folium.Marker(
            location=[row['latitude'], row['longitude']],
            popup=f"{facility.title()}: {row['name']}",  # Popup text showing the name of the facility
            icon=folium.Icon(color=color)
        ).add_to(feature_group)
    feature_group.add_to(map)

# Add Layer control to toggle facilities
folium.LayerControl(collapsed=False).add_to(map)

# Save the map as an HTML file
# map.save('accessibility_map.html')
map
```

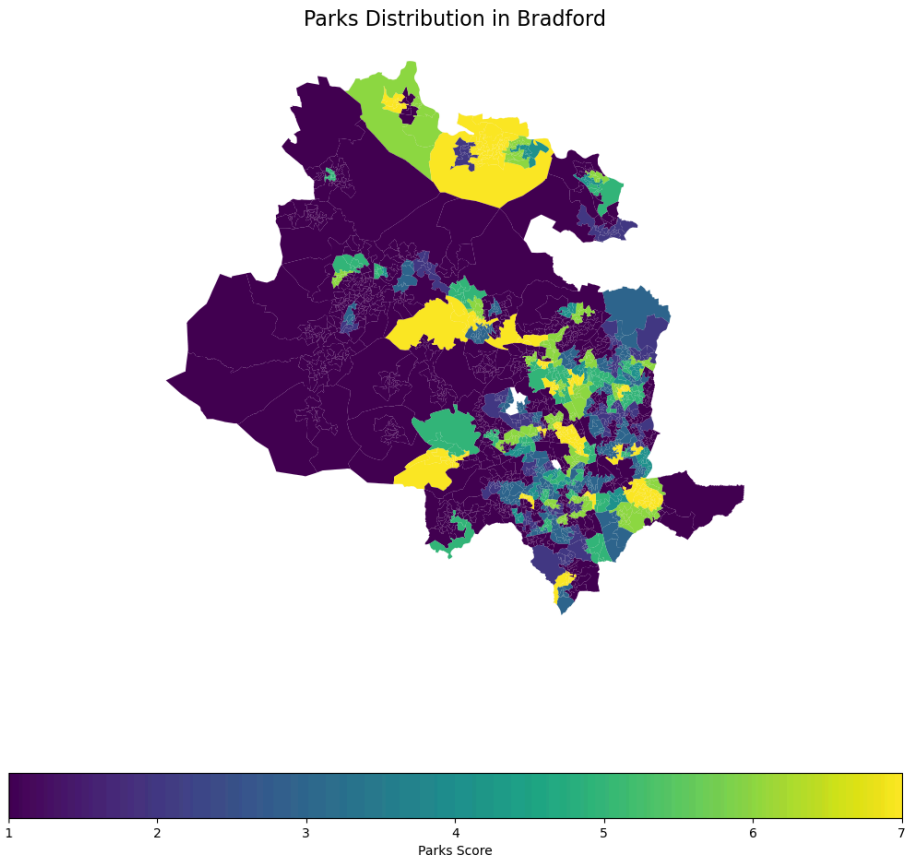
Out[ ]:

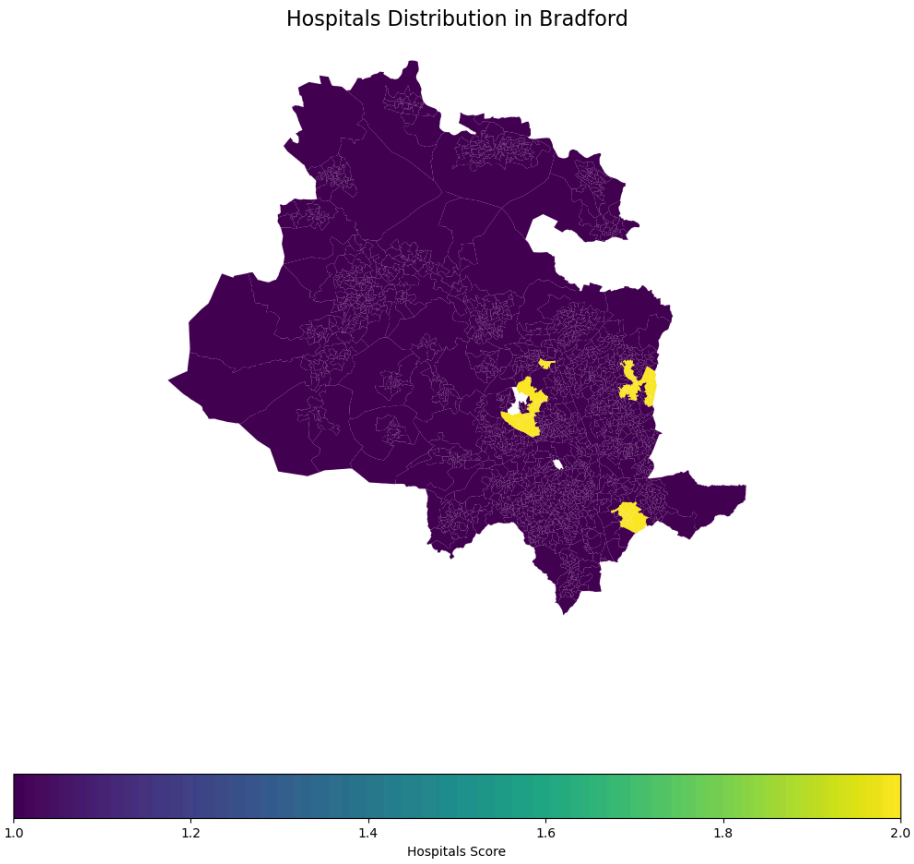

Crosshills
Eastburn
Steeton
Sutton
Burley Moor
Bingley Moor
Hawksworth Moor
Hawksworth
East Morton
Laycock
Harden
Stanbury
Cullingworth
Oxenhope
education
parks
hospitals
services
transport

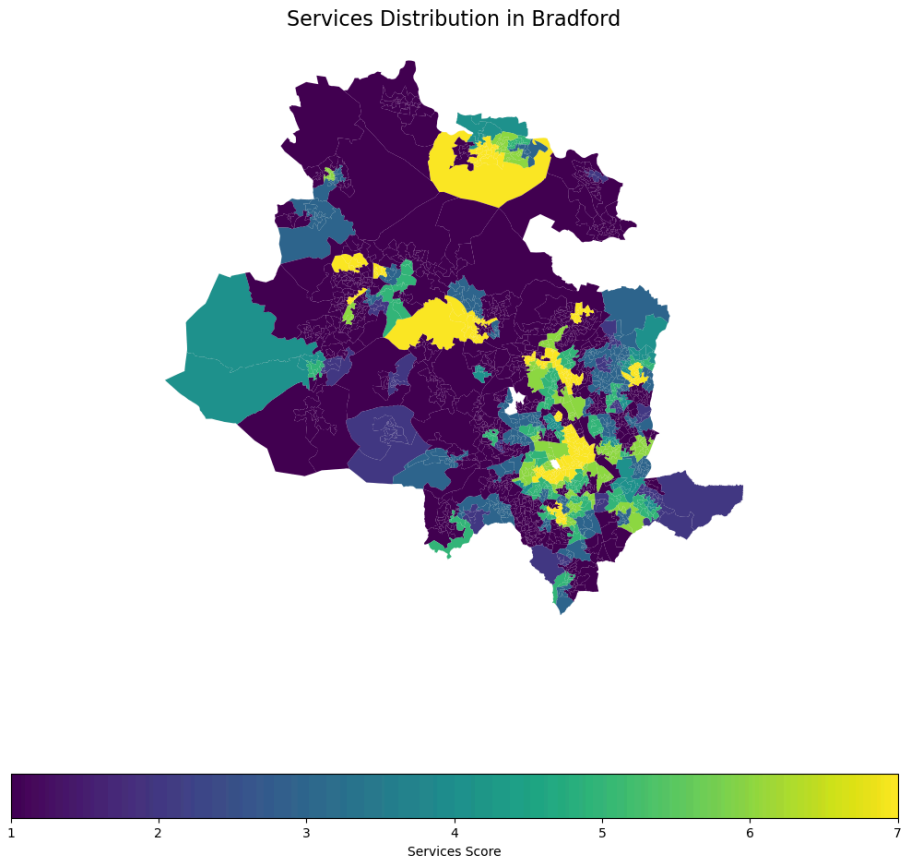
In [ ]:
```
import geopandas as gpd
import matplotlib.pyplot as plt

# Load the shapefile
gdf = gpd.read_file(r"Output_Areas_2021_Polygon/OA_2021_EW_BFC_V8.shp")

# Merge GeoDataFrame with your DataFrame on the LSOA code
gdf = gdf.merge(data_ranked, left_on='LSOA21CD', right_on='LSOA')
```

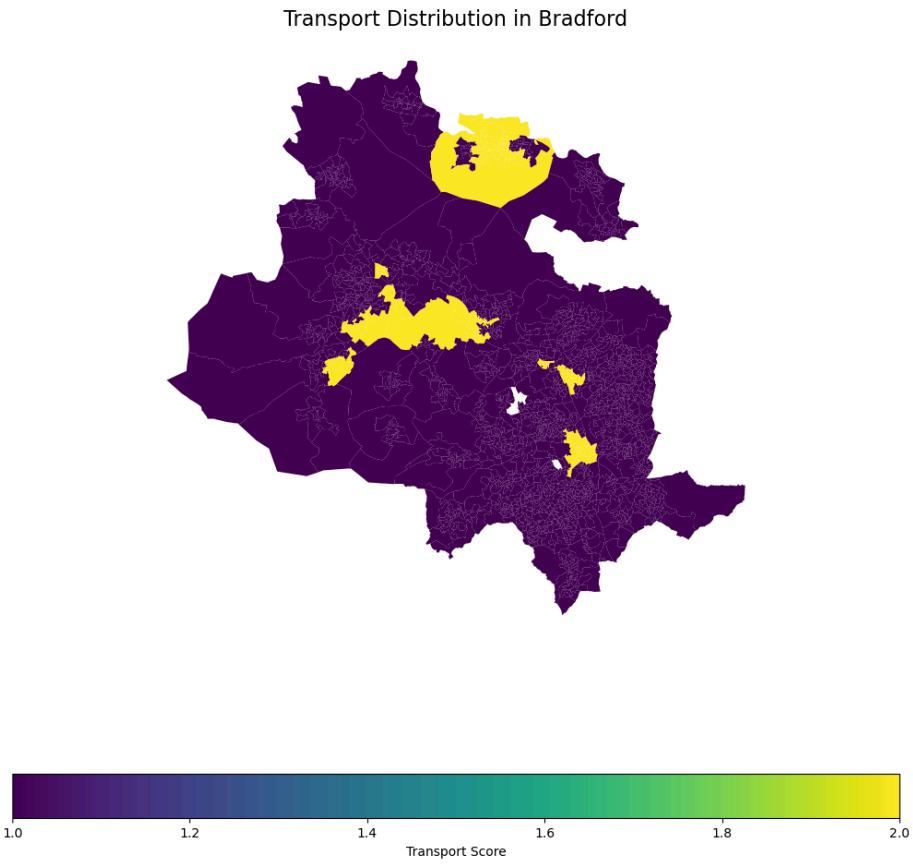
In [ ]:
```
# Columns to plot
columns = ['education', 'parks', 'hospitals', 'services', 'transport', 'residential', 'crime', 'house', 'IMD', 'ind

# Split columns into chunks for multiple figures
chunks = [columns[x:x+2] for x in range(0, len(columns), 2)]  # 2 maps per figure

for i, chunk in enumerate(chunks):
    fig, axs = plt.subplots(nrows=1, ncols=2, figsize=(20, 10))
    axs = axs.flatten()
    for ax, column in zip(axs, chunk):
        gdf.plot(column=column, ax=ax, legend=True, legend_kwds={'label': f"{column.capitalize()} Score", 'orientat
        ax.set_title(f"{column.capitalize()} Distribution in Bradford", fontsize=16)
        ax.axis('off')
    plt.tight_layout()
#   plt.savefig(f'bradford_{i+1}.png')  # Save each figure to a file
    plt.show()
```

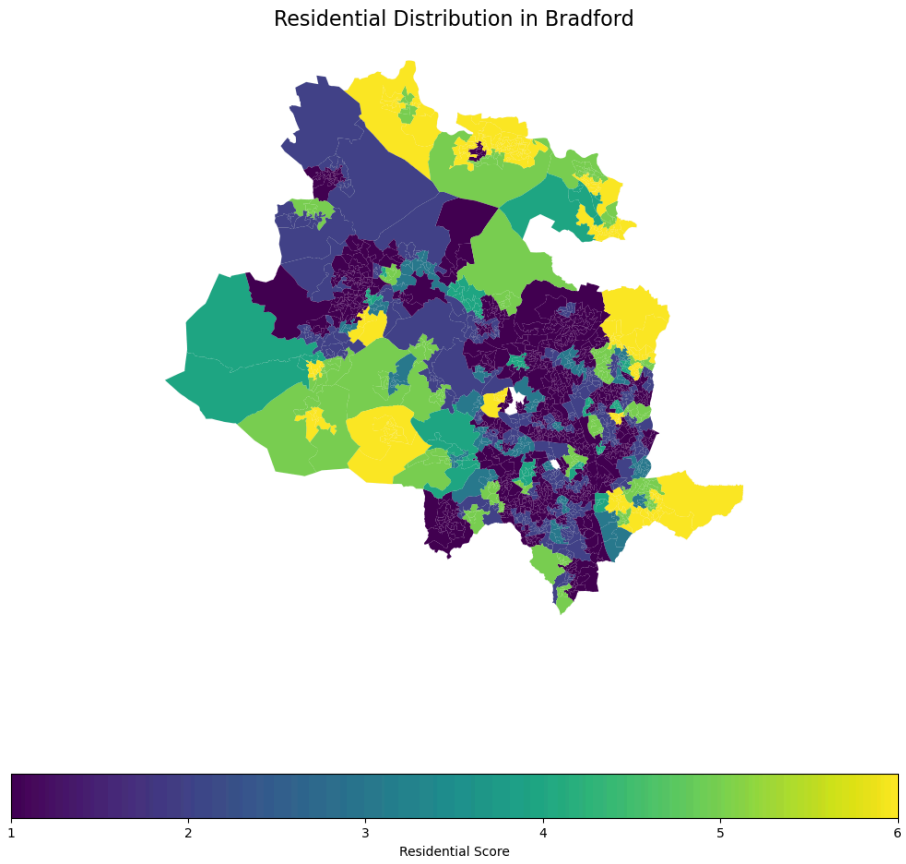

Education Distribution in Bradford
1
2
3
4
5
6
7
8
Education Score

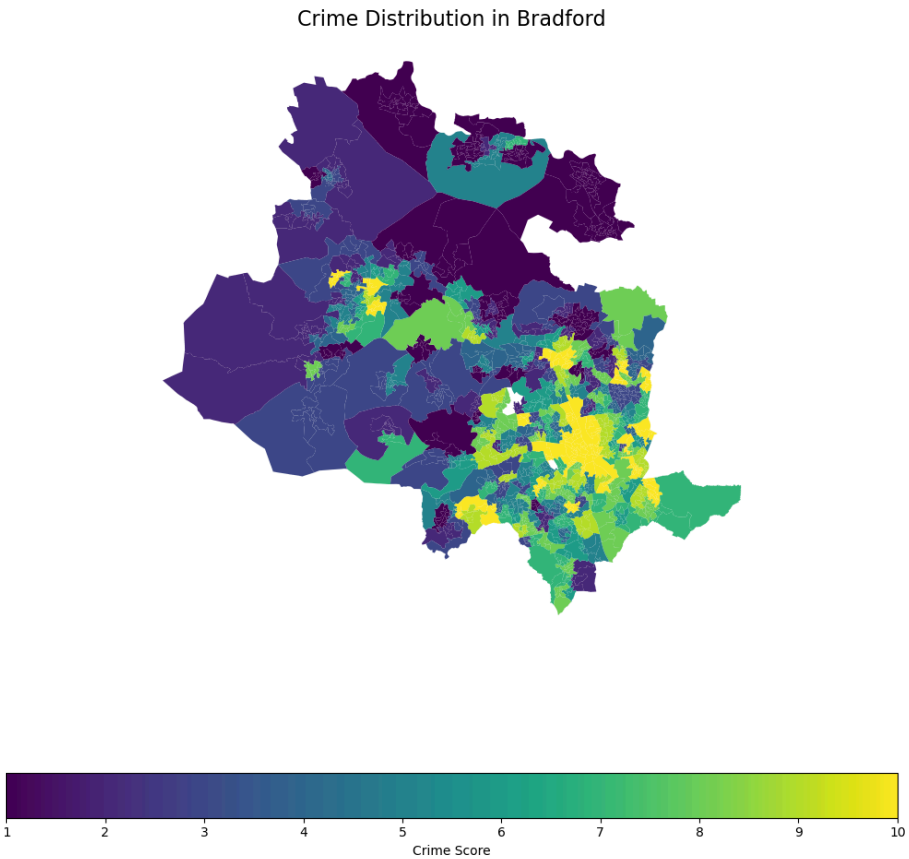

Parks Distribution in Bradford
1
2
3
4
5
6
7
Parks Score

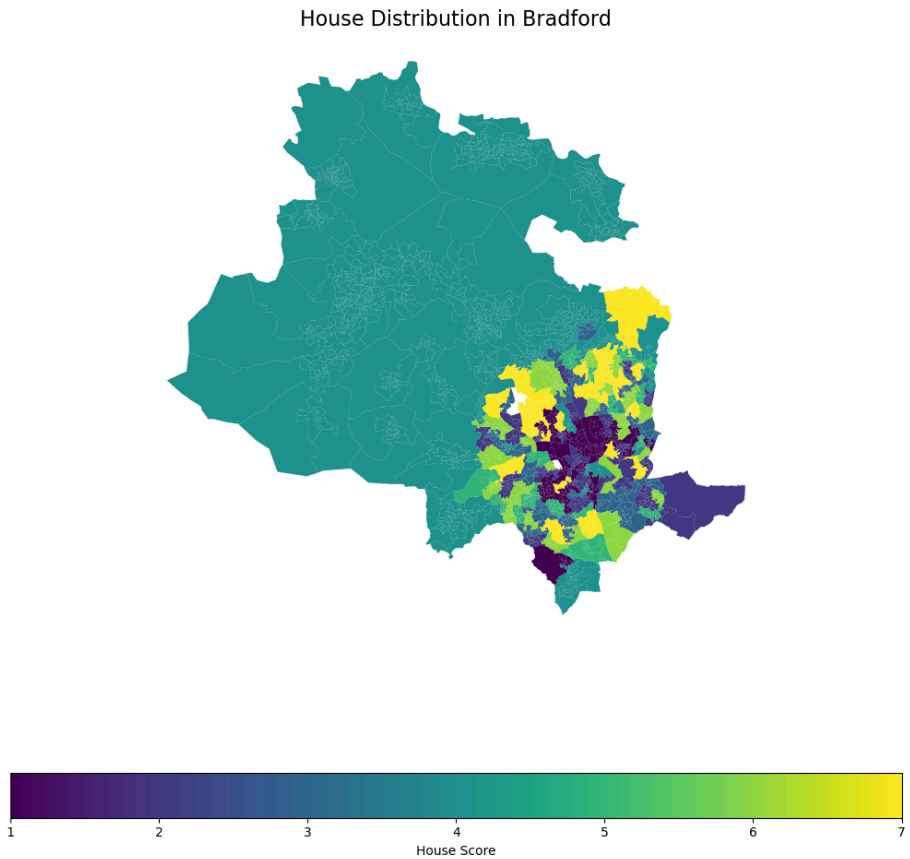

Hospitals Distribution in Bradford
1.0
1.2
1.4
1.6
1.8
2.0
Hospitals Score

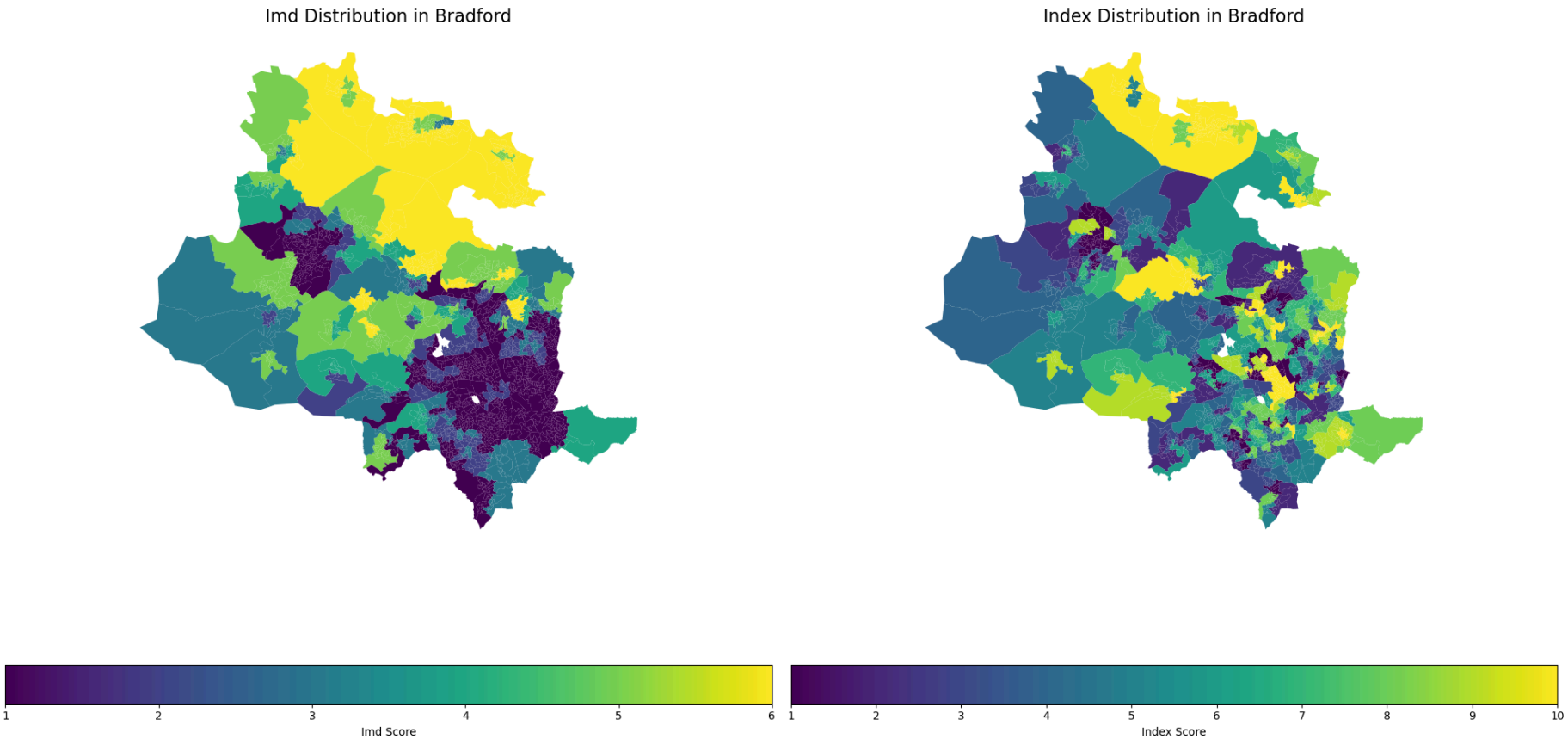

Services Distribution in Bradford
1
2
3
4
5
6
7
Services Score


Transport Distribution in Bradford
1.0
1.2
1.4
1.6
1.8
2.0
Transport Score

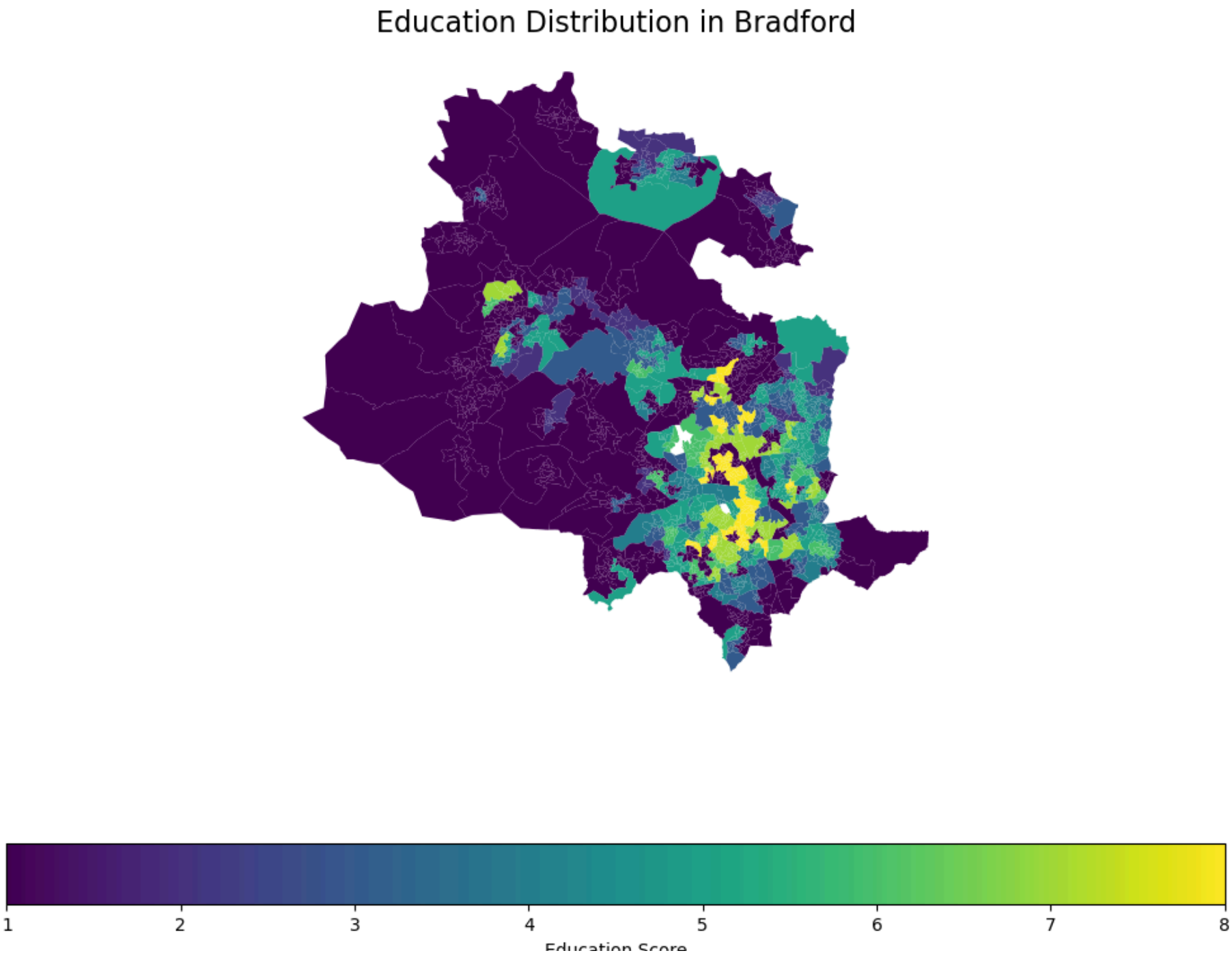

Residential Distribution in Bradford
1
2
3
4
5
6
Residential Score

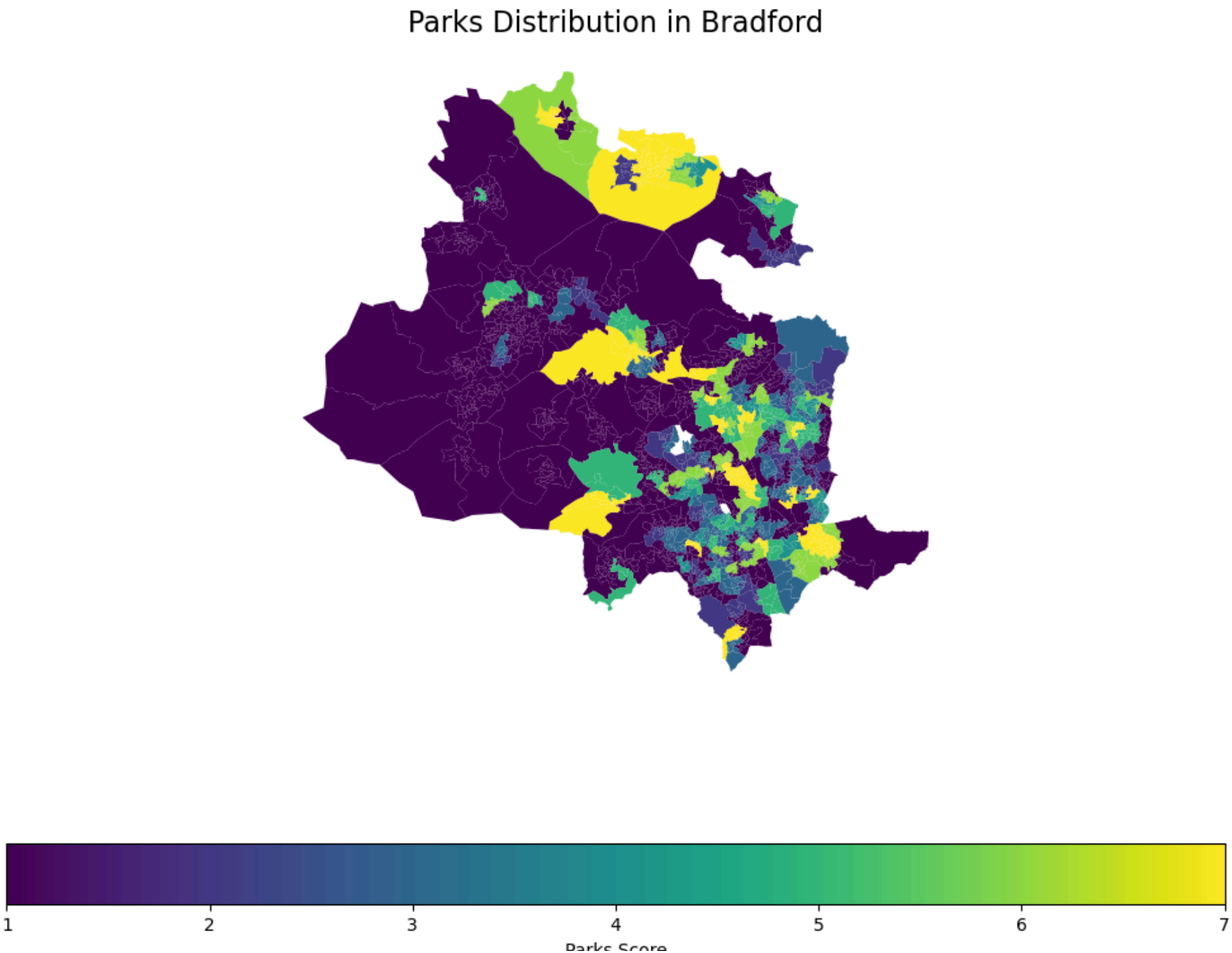

Crime Distribution in Bradford
1
2
3
4
5
6
7
8
9
10
Crime Score

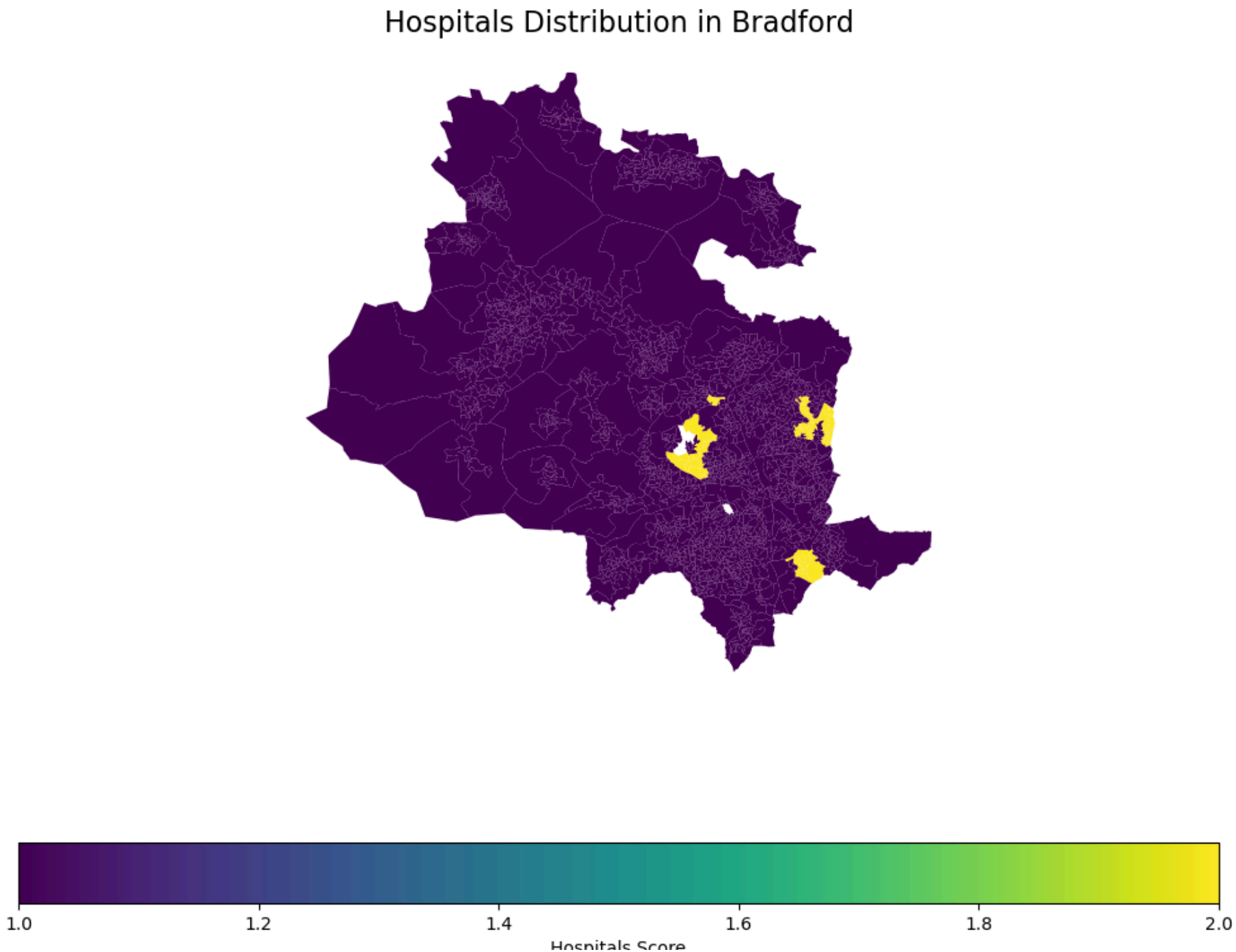

House Distribution in Bradford
1
2
3
4
5
6
7
House Score

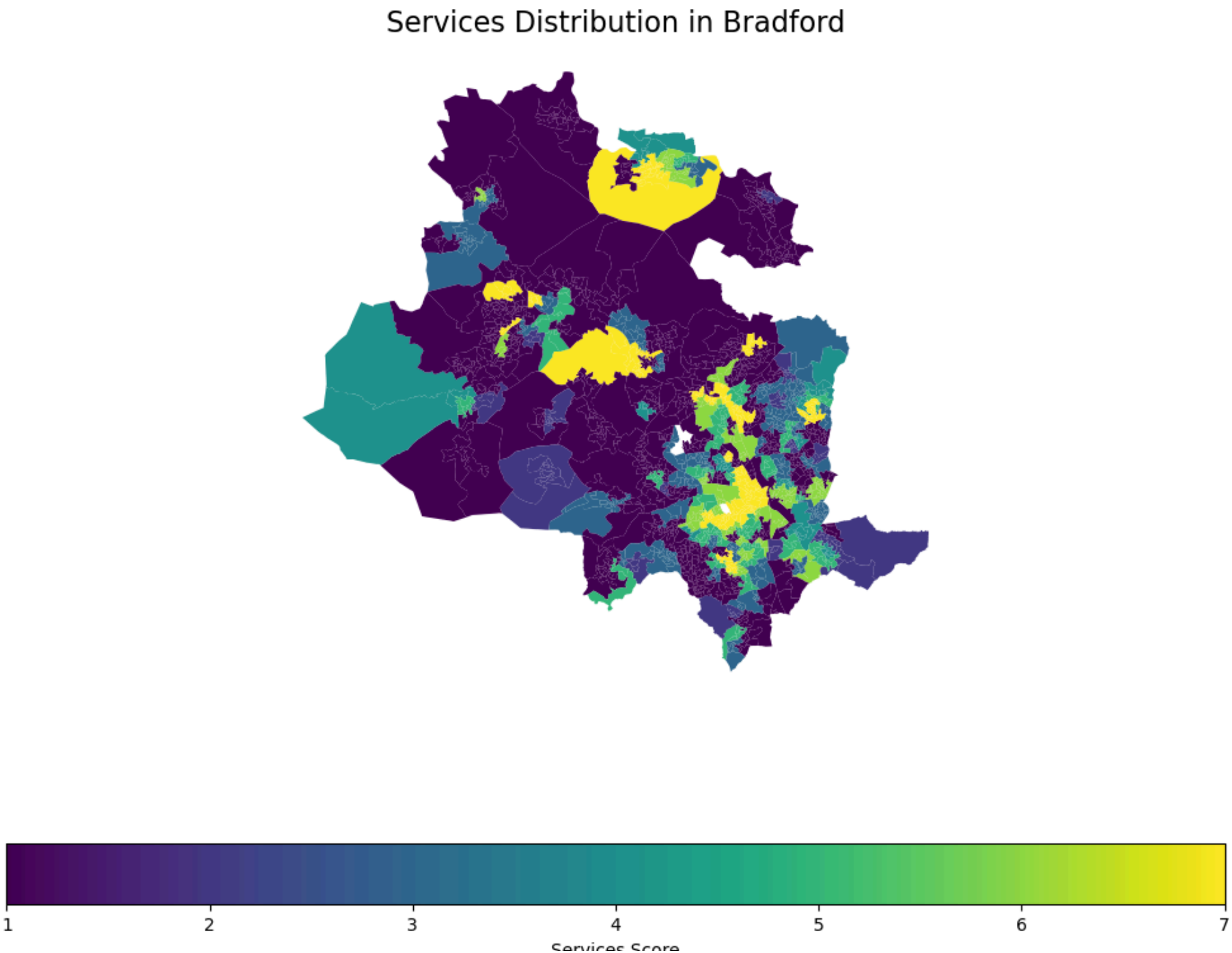


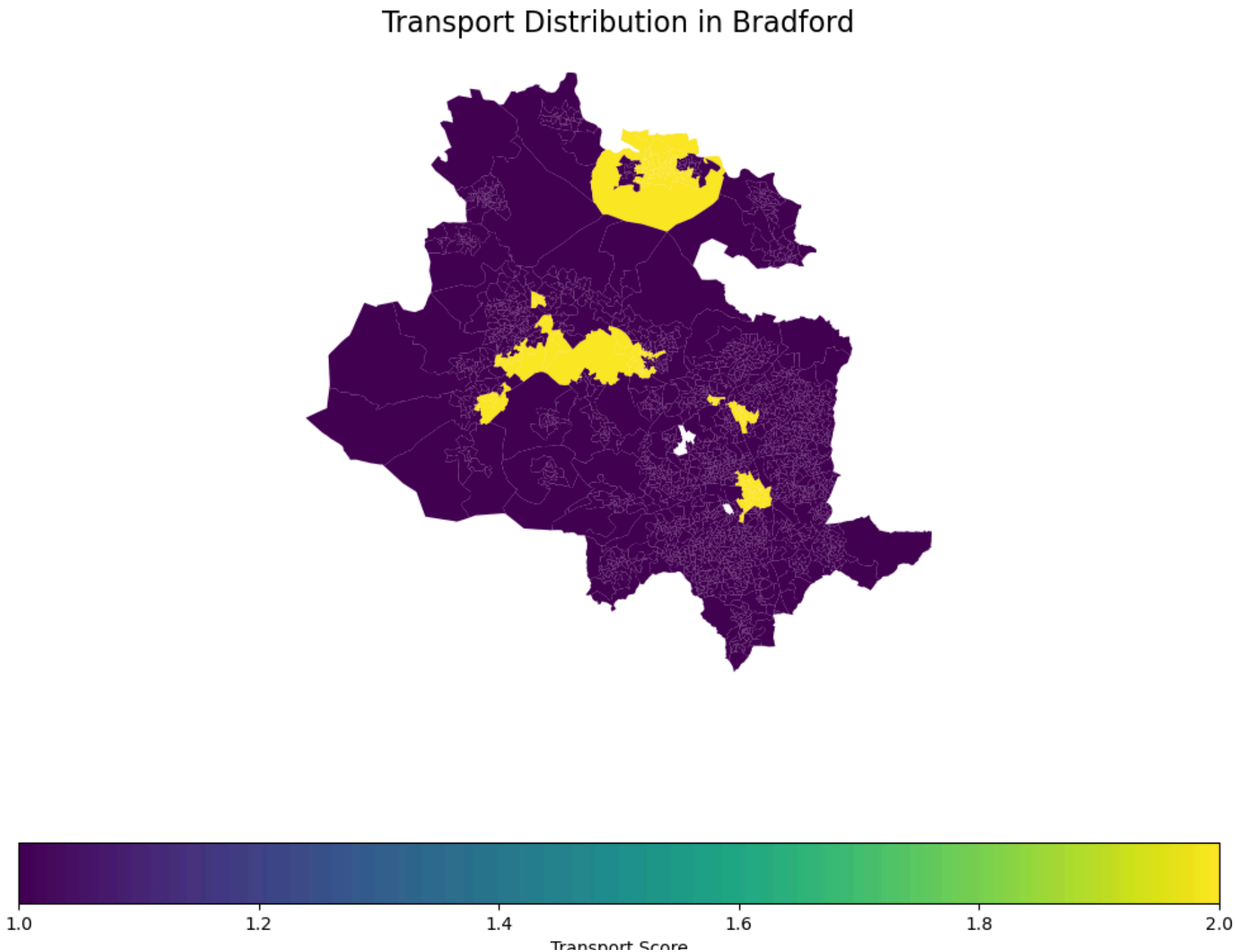
In [ ]:
```
for column in columns:
    fig, ax = plt.subplots(figsize=(10, 8))  # Large size for a single map
    gdf.plot(column=column, ax=ax, legend=True, legend_kwds={'label': f"{column.capitalize()} Score", 'orientation'
    ax.set_title(f"{column.capitalize()} Distribution in Bradford", fontsize=16)
    ax.axis('off')
    plt.tight_layout()
#   plt.savefig(f'bradford_{column}.png')  # Save to file
    plt.show()
```

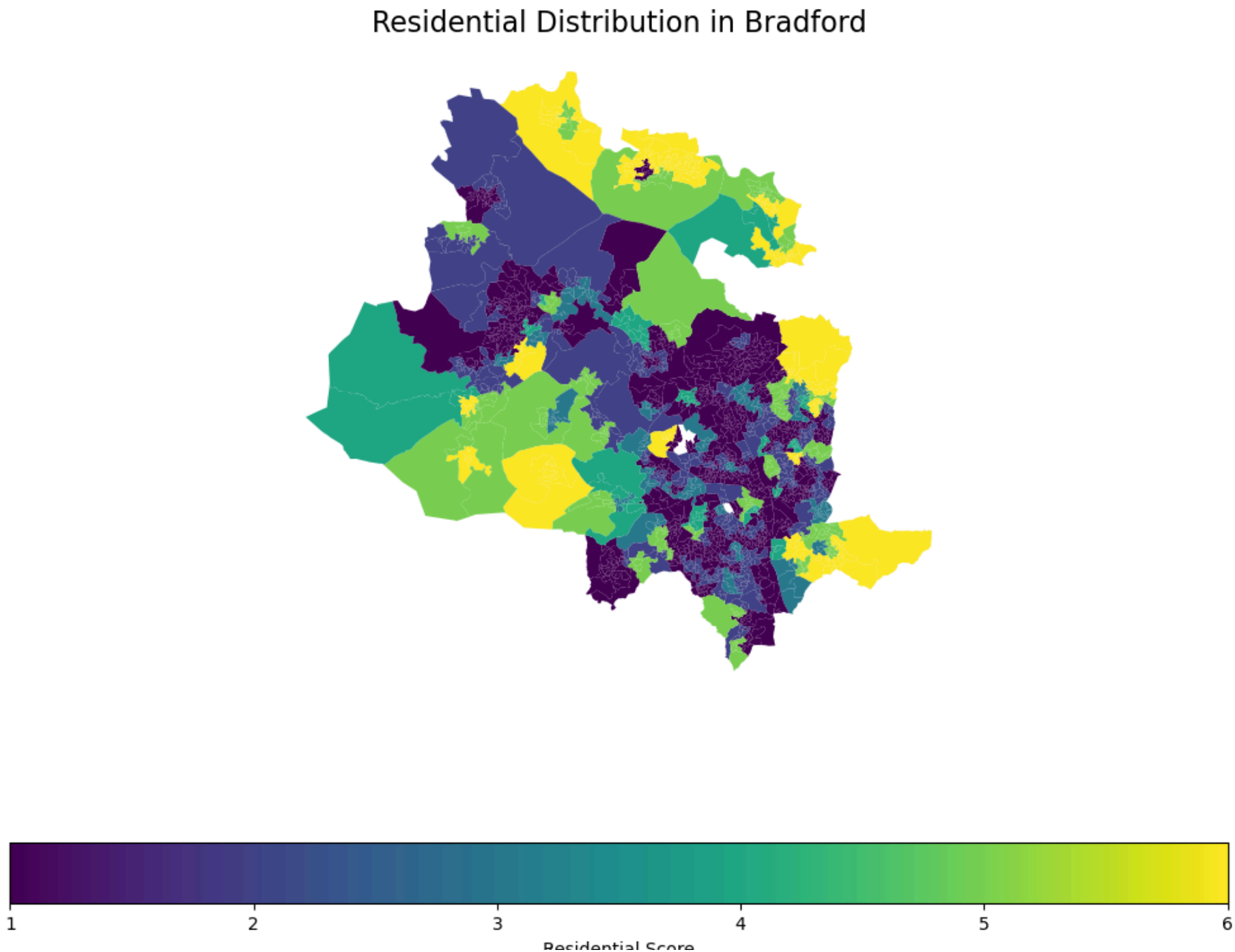

Education Distribution in Bradford
1
2
3
4
5
6
7
8
Education Score

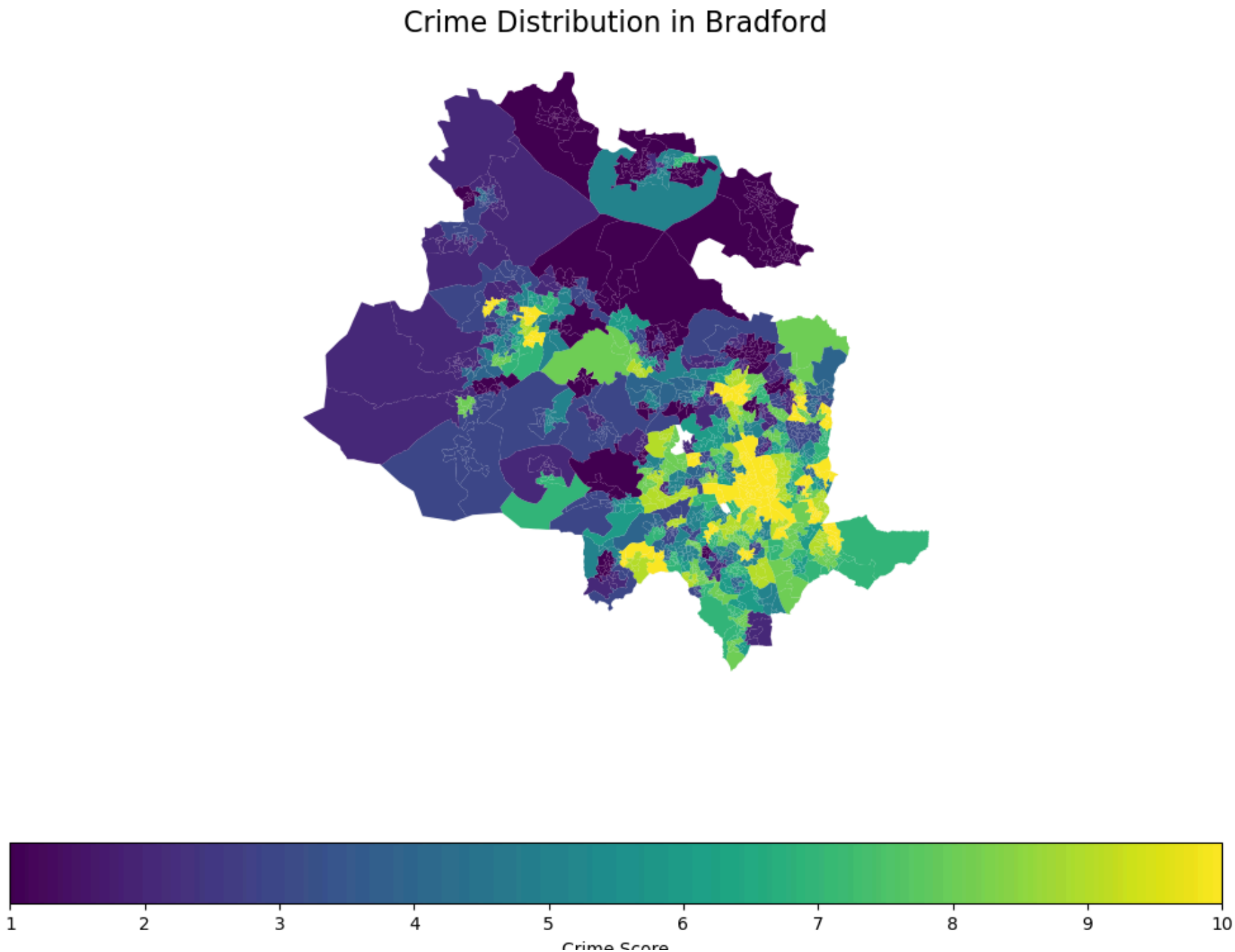

Parks Distribution in Bradford
1
2
3
4
5
6
7
Parks Score

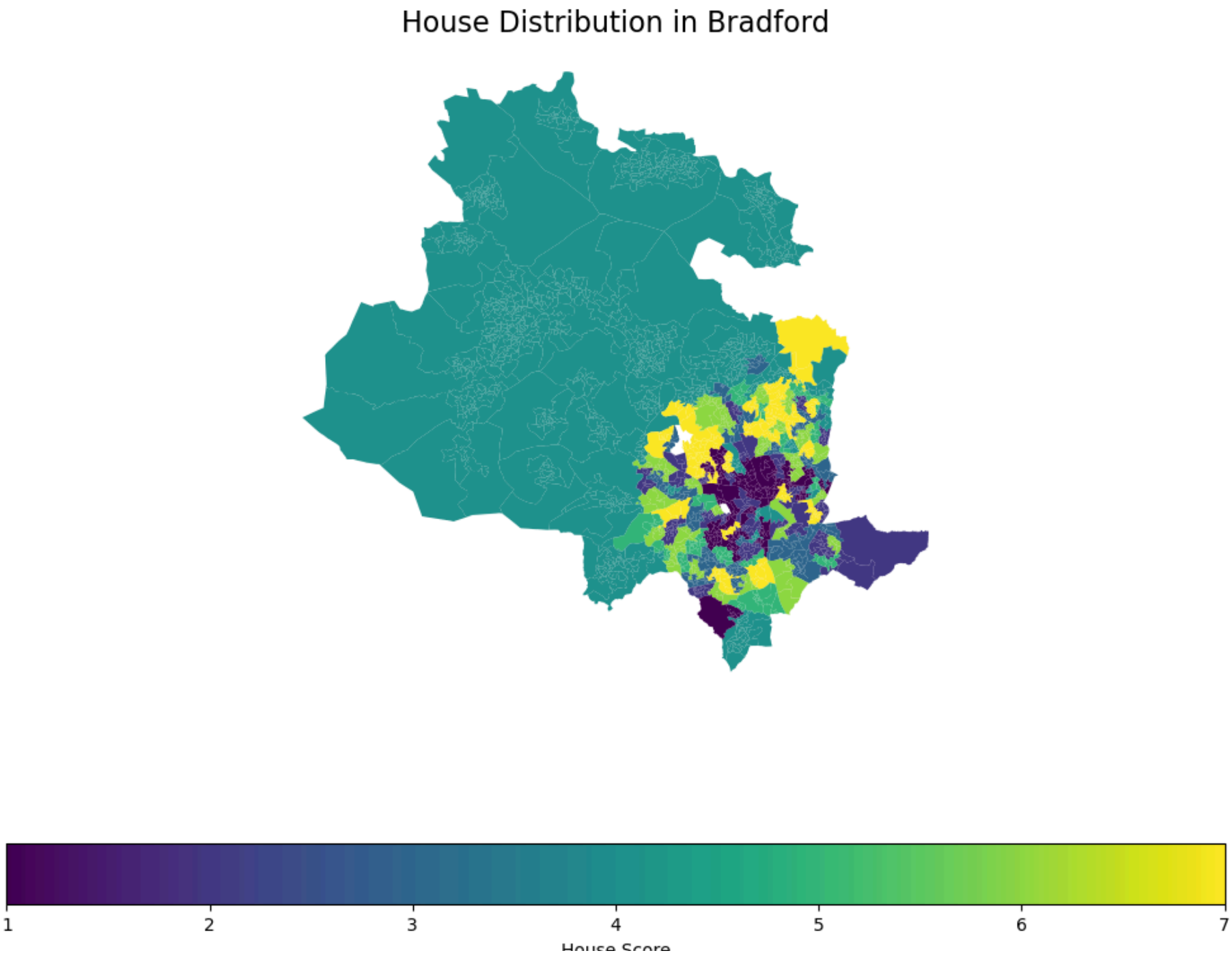

Hospitals Distribution in Bradford
1.0
1.2
1.4
1.6
1.8
2.0
Hospitals Score

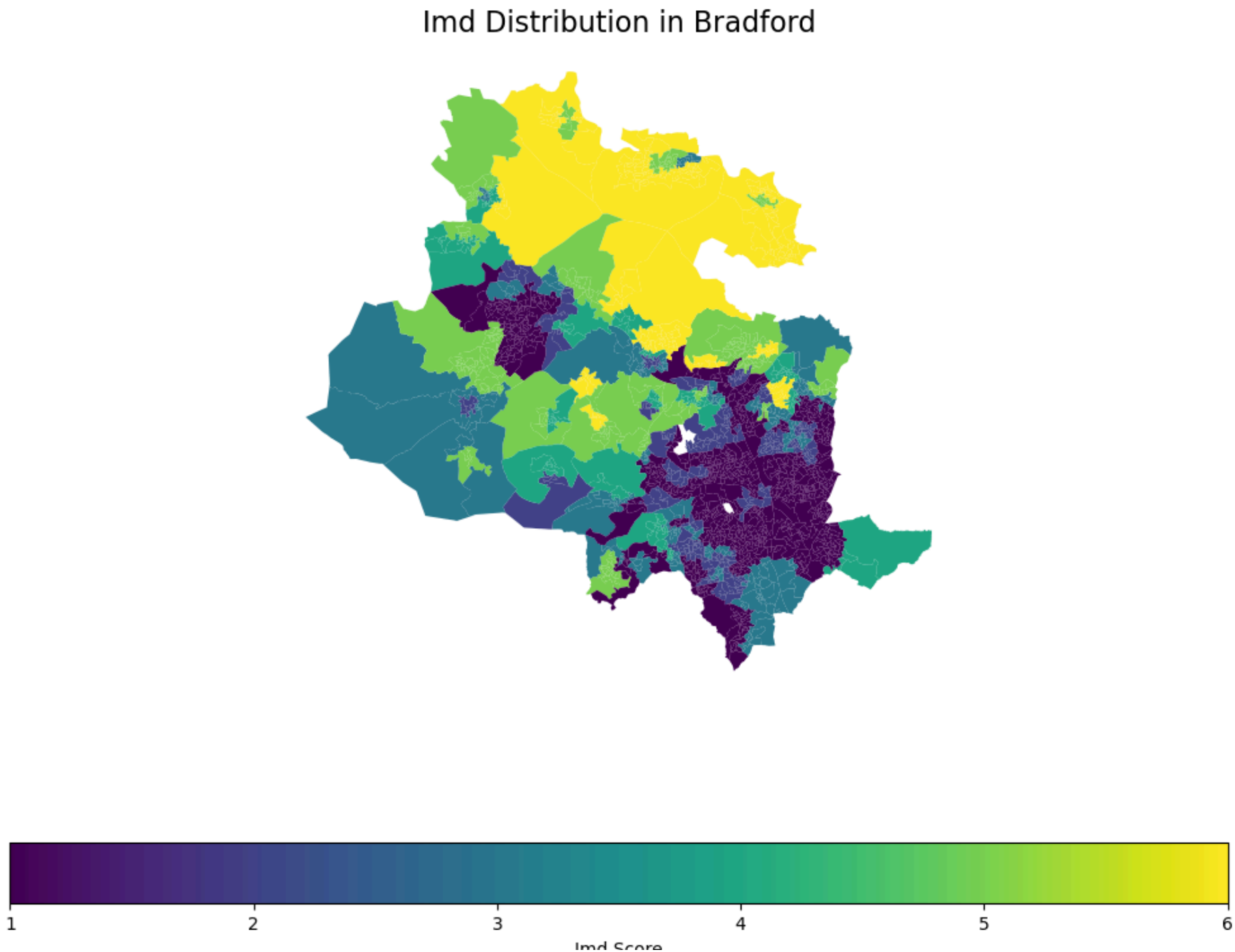

Services Distribution in Bradford
1
2
3
4
5
6
7
Services Score

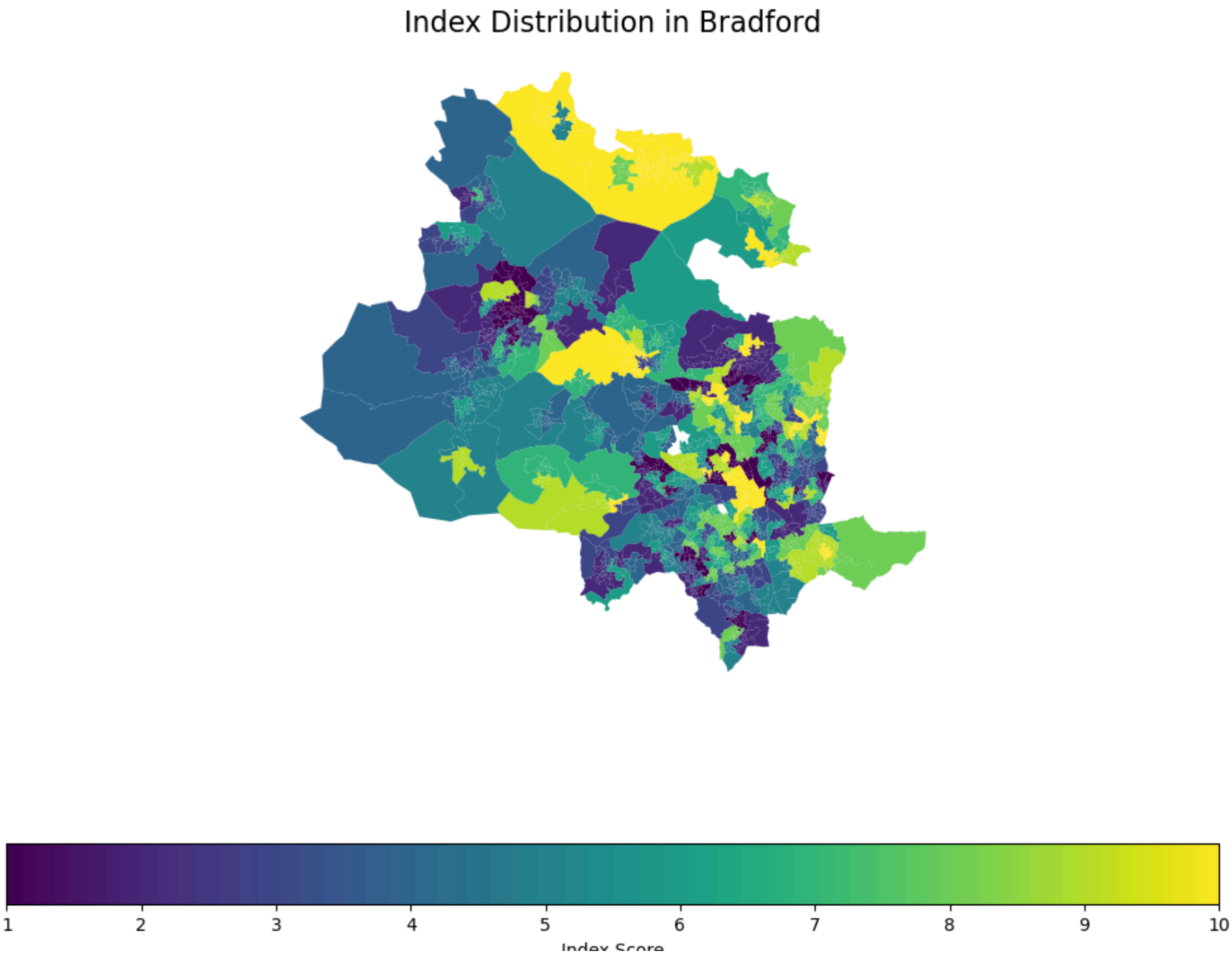

Transport Distribution in Bradford
1.0
1.2
1.4
1.6
1.8
2.0
Transport Score


Residential Distribution in Bradford
1
2
3
4
5
6
Residential Score


Crime Distribution in Bradford
1
2
3
4
5
6
7
8
9
10
Crime Score


House Distribution in Bradford
1
2
3
4
5
6
7
House Score


Imd Distribution in Bradford
1
2
3
4
5
6
Imd Score


Index Distribution in Bradford
1
2
3
4
5
6
7
8
9
10
Index Score