Investigate the dataset on the coverage of mobile networks

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import glob
import re
import chardet
import numpy as np
import pymongo
import folium
```

In [2]:
```
# Find the file names
!ls 2023J_TMA02_data/Ofcom_mobile
```

```
201909_mobile_laua_r01.csv
202009_mobile_laua_r01.csv
202109_mobile_laua_r01.csv
202209-about-mobile-coverage-local-and-unitary-authority.pdf
202209_mobile_laua_r01.csv
202304_mobile_laua_r01.csv
cn-2020-about-mobile-coverage-local-and-unitary-authority.pdf
cn-2021-about-mobile-laua.pdf
connected-nations-2019-about-mobile-local-unitary-authority-area.pdf
mobile-coverage-local-unitary-authority-202304.pdf
```

2019

In [3]:
```
df2019 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/201909_mobile_laua_r0
```

In [4]:
```
df2019.head()
```

Out[4]:

	laua	laua_name	2G_prem_out_0	2G_prem_out_1	2G_prem_out_2	2
0	E06000001	Hartlepool	0.08	0.14	3.49	
1	E06000002	Middlesbrough	NaN	NaN	2.18	
2	E06000003	Redcar and Cleveland	0.06	0.26	2.90	
3	E06000004	Stockton-on-Tees	0.04	0.17	3.17	
4	E06000005	Darlington	0.01	0.01	1.18	

5 rows × 147 columns

In [5]:
```
df2019.tail()
```

Out[5]:

	laua	laua_name	2G_prem_out_0	2G_prem_out_1	2G_prem_out_2
377	W06000020	Torfaen	0.03	0.10	1.96
378	W06000021	Monmouthshire	1.95	2.95	6.25
379	W06000022	Newport	NaN	0.20	1.4
380	W06000023	Powys	5.65	4.57	11.42
381	W06000024	Merthyr Tydfil	0.18	0.17	0.48

5 rows × 147 columns

In [6]:

```
# Search data types
df2019.dtypes
```

Out[6]:

```
laua                              object
laua_name                         object
2G_prem_out_0                    float64
2G_prem_out_1                    float64
2G_prem_out_2                    float64
                                   ...
Data_mway_ard_in_1               float64
Data_mway_ard_in_2               float64
Data_mway_ard_in_3               float64
Data_mway_ard_in_4               float64
per_sites_with_fibre_backhaul      int64
Length: 147, dtype: object
```

In [7]:

```
# Investigate columns of 4G
# Select only numerical columns
numerical_columns = df2019.select_dtypes(include=['number'])

# Use describe on numerical columns
numerical_description = numerical_columns.describe()

# Display the numerical description
display(numerical_description)
```

	2G_prem_out_0	2G_prem_out_1	2G_prem_out_2	2G_prem_out_3	2G_prem_
count	240.000000	288.000000	376.000000	382.000000	334.00
mean	0.664875	1.182500	6.107580	92.679162	3.92
std	1.032041	1.672305	6.481532	8.308294	4.56
min	0.000000	0.000000	0.000000	46.640000	0.00
25%	0.050000	0.090000	1.187500	88.802500	0.42
50%	0.250000	0.535000	4.200000	95.560000	2.28
75%	0.785000	1.650000	9.630000	98.882500	5.69
max	6.180000	11.330000	40.040000	100.000000	22.79

8 rows × 145 columns

In [8]:
```python
# Find null values
df2019.isnull().sum()
```

Out[8]:
```
laua                              0
laua_name                         0
2G_prem_out_0                   142
2G_prem_out_1                    94
2G_prem_out_2                     6
                               ...
Data_mway_ard_in_1              274
Data_mway_ard_in_2              198
Data_mway_ard_in_3               74
Data_mway_ard_in_4                0
per_sites_with_fibre_backhaul     0
Length: 147, dtype: int64
```

In [9]:
```python
# Investigate null values of 4G columns
import re

# Find columns containing '4g' (case insensitive)
columns_with_4g = [col for col in df2019.columns if re.search(r'4G', col,

# Check for missing values and calculate the sum for columns containing '
missing_values_4g = df2019[columns_with_4g].isnull().sum()

display(missing_values_4g)
```

```
4G_prem_out_0       208
4G_prem_out_1       156
4G_prem_out_2       107
4G_prem_out_3        42
4G_prem_out_4         1
4G_prem_in_0         87
4G_prem_in_1         59
4G_prem_in_2         13
4G_prem_in_3          4
4G_prem_in_4          1
4G_geo_out_0        155
4G_geo_out_1        110
4G_geo_out_2         72
4G_geo_out_3         25
4G_geo_out_4          1
4G_abrd_in_0        115
4G_abrd_in_1         75
4G_abrd_in_2         24
4G_abrd_in_3          8
4G_abrd_in_4          1
4G_mway_in_0         65
4G_mway_in_1         38
4G_mway_in_2          9
4G_mway_in_3          3
4G_mway_in_4          1
4G_mway_ard_in_0    151
4G_mway_ard_in_1     92
4G_mway_ard_in_2     28
4G_mway_ard_in_3      9
4G_mway_ard_in_4      1
dtype: int64
```

In [10]:
```
# Dataset
df2019.shape
```

Out[10]: (382, 147)

2020

In [11]:
```
df2020 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202009_mobile_laua_r0
df2020.head()
```

Out[11]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_count
0	E06000001	Hartlepool	9345	47125	665	0
1	E06000002	Middlesbrough	5387	67601	760	0
2	E06000003	Redcar and Cleveland	24471	68041	1469	0
3	E06000004	Stockton-on-Tees	20517	91340	1426	0
4	E06000005	Darlington	19746	54545	1114	127

5 rows × 151 columns

In [12]:
```
df2020.tail()
```

Out[12]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_co
374	W06000020	Torfaen	12612	45731	674	
375	W06000021	Monmouthshire	85053	45149	3680	
376	W06000022	Newport	19050	71924	1272	
377	W06000023	Powys	519528	69281	15700	
378	W06000024	Merthyr Tydfil	11178	28406	874	

5 rows × 151 columns

In [13]:
```
# data types
df2020.dtypes
```

Out[13]:
```
laua                   object
laua_name              object
pixel_count             int64
prem_count              int64
ab_rd_count             int64
                        ...
Data_mway_ard_in_0    float64
Data_mway_ard_in_1    float64
Data_mway_ard_in_2    float64
Data_mway_ard_in_3    float64
Data_mway_ard_in_4    float64
Length: 151, dtype: object
```

In [14]:
```python
# Find null values
df2019.isnull().sum()
```

Out[14]:
```
laua                              0
laua_name                         0
2G_prem_out_0                   142
2G_prem_out_1                    94
2G_prem_out_2                     6
                               ...
Data_mway_ard_in_1              274
Data_mway_ard_in_2              198
Data_mway_ard_in_3               74
Data_mway_ard_in_4                0
per_sites_with_fibre_backhaul     0
Length: 147, dtype: int64
```

In [15]:
```python
# Investigate columns of 4G
# Select only numerical columns
numerical_columns2 = df2020.select_dtypes(include=['number'])

# Use describe on numerical columns
numerical_description2 = numerical_columns2.describe()

# Display the numerical description
display(numerical_description2)
```

	pixel_count	prem_count	ab_rd_count	mway_count	mway_ard_count
count	3.790000e+02	379.000000	379.000000	379.000000	379.000000
mean	6.420102e+04	81992.424802	2652.023747	134.274406	1723.134565
std	1.606712e+05	54610.433868	3479.549582	191.255912	2130.623198
min	2.900000e+02	1623.000000	88.000000	0.000000	88.000000
25%	9.111500e+03	48622.000000	908.000000	0.000000	703.000000
50%	2.718800e+04	65530.000000	1626.000000	32.000000	1239.000000
75%	6.512100e+04	99397.500000	3047.000000	234.500000	1977.500000
max	2.610989e+06	474236.000000	40780.000000	1053.000000	28997.000000

8 rows × 149 columns

In [16]:
```python
# Investigate null values of 4G columns
import re

# Find columns containing '4g' (case insensitive)
columns_with_4g2 = [col for col in df2020.columns if re.search(r'4G', col

# Check for missing values and calculate the sum for columns containing '
missing_values_4g2 = df2020[columns_with_4g2].isnull().sum()

print(missing_values_4g2)
```

```
4G_prem_out_0       214
4G_prem_out_1       157
4G_prem_out_2       110
4G_prem_out_3        33
4G_prem_out_4         1
4G_prem_in_0         91
4G_prem_in_1         42
4G_prem_in_2         10
4G_prem_in_3          1
4G_prem_in_4          1
4G_geo_out_0        152
4G_geo_out_1        105
4G_geo_out_2         63
4G_geo_out_3         18
4G_geo_out_4          1
4G_abrd_in_0        129
4G_abrd_in_1         76
4G_abrd_in_2         22
4G_abrd_in_3          1
4G_abrd_in_4          1
4G_mway_in_0        375
4G_mway_in_1        349
4G_mway_in_2        239
4G_mway_in_3        182
4G_mway_in_4        171
4G_mway_ard_in_0    163
4G_mway_ard_in_1     98
4G_mway_ard_in_2     25
4G_mway_ard_in_3      1
4G_mway_ard_in_4      1
dtype: int64
```

In [17]:
```
# Investigate columns of 4G
# Use describe on numerical columns
numerical_description2 = numerical_columns2[columns_with_4g2].describe()

# Display the numerical description
display(numerical_description2)
```

	4G_prem_out_0	4G_prem_out_1	4G_prem_out_2	4G_prem_out_3	4G_prem
count	165.000000	222.000000	269.000000	346.000000	378.(
mean	0.347636	0.512297	0.801747	2.891474	96.5
std	0.622406	1.650327	1.434282	6.248743	5.1
min	0.000000	0.000000	0.000000	0.000000	49.1
25%	0.020000	0.020000	0.050000	0.270000	95.8
50%	0.070000	0.100000	0.280000	1.160000	98.9
75%	0.430000	0.427500	0.860000	3.735000	99.8
max	3.640000	22.140000	12.280000	95.630000	100.(

8 rows × 30 columns

In [18]:
```
# data volume
df2020.shape
```

Out[18]: (379, 151)

2021

In [19]:
```
df2021 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202109_mobile_laua_r0
df2021.head()
```

Out[19]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_count
0	E06000001	Hartlepool	9345	47585	665	0
1	E06000002	Middlesbrough	5387	68329	760	0
2	E06000003	Redcar and Cleveland	24471	68559	1469	0
3	E06000004	Stockton-on-Tees	20517	92889	1426	0
4	E06000005	Darlington	19746	54760	1114	127

5 rows × 151 columns

In [20]:
```
df2021.columns
```

Out[20]:
```
Index(['laua', 'laua_name', 'pixel_count', 'prem_count', 'ab_rd_count',
       'mway_count', 'mway_ard_count', '2G_prem_out_0', '2G_prem_out_1',
       '2G_prem_out_2',
       ...
       'Data_mway_in_0', 'Data_mway_in_1', 'Data_mway_in_2', 'Data_mway_
in_3',
       'Data_mway_in_4', 'Data_mway_ard_in_0', 'Data_mway_ard_in_1',
       'Data_mway_ard_in_2', 'Data_mway_ard_in_3', 'Data_mway_ard_in_
4'],
      dtype='object', length=151)
```

In [21]:
```
df2021.tail()
```

Out[21]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_co
369	W06000020	Torfaen	12612	46076	674	
370	W06000021	Monmouthshire	85053	45585	3680	
371	W06000022	Newport	19050	72445	1272	
372	W06000023	Powys	519528	69669	15700	
373	W06000024	Merthyr Tydfil	11178	28534	874	

5 rows × 151 columns

In [22]:
```
# data types
df2021.dtypes
```

Out[22]:
```
laua                      object
laua_name                 object
pixel_count                int64
prem_count                 int64
ab_rd_count                int64
                           ...
Data_mway_ard_in_0       float64
Data_mway_ard_in_1       float64
Data_mway_ard_in_2       float64
Data_mway_ard_in_3       float64
Data_mway_ard_in_4       float64
Length: 151, dtype: object
```

In [23]:
```
# Find null values
df2021.isnull().sum()
```

Out[23]:
```
laua                    0
laua_name               0
pixel_count             0
prem_count              0
ab_rd_count             0
                      ...
Data_mway_ard_in_0    320
Data_mway_ard_in_1    292
Data_mway_ard_in_2    214
Data_mway_ard_in_3     82
Data_mway_ard_in_4      0
Length: 151, dtype: int64
```

In [24]:
```
# Investigate null values of 4G columns

import re

# Find columns containing '4g' (case insensitive)
columns_with_4g3 = [col for col in df2021.columns if re.search(r'4G', col

# Check for missing values and calculate the sum for columns containing '
missing_values_4g3 = df2021[columns_with_4g3].isnull().sum()

print(missing_values_4g3)
```

```
4G_prem_out_0       190
4G_prem_out_1       131
4G_prem_out_2        88
4G_prem_out_3        26
4G_prem_out_4         1
4G_prem_in_0         80
4G_prem_in_1         43
4G_prem_in_2          9
4G_prem_in_3          1
4G_prem_in_4          1
4G_geo_out_0        154
4G_geo_out_1        106
4G_geo_out_2         60
4G_geo_out_3         17
4G_geo_out_4          1
4G_abrd_in_0        133
4G_abrd_in_1         80
4G_abrd_in_2         25
4G_abrd_in_3          3
4G_abrd_in_4          1
4G_mway_in_0        369
4G_mway_in_1        346
4G_mway_in_2        249
4G_mway_in_3        176
4G_mway_in_4        168
4G_mway_ard_in_0    166
4G_mway_ard_in_1    103
4G_mway_ard_in_2     26
4G_mway_ard_in_3      3
4G_mway_ard_in_4      1
dtype: int64
```

In [25]:
```
# Investigate columns of 4G
# Use describe on numerical columns
numerical_description3 = numerical_columns2[columns_with_4g3].describe()

# Display the numerical description
display(numerical_description3)
```

	4G_prem_out_0	4G_prem_out_1	4G_prem_out_2	4G_prem_out_3	4G_prem
count	165.000000	222.000000	269.000000	346.000000	378.(
mean	0.347636	0.512297	0.801747	2.891474	96.5
std	0.622406	1.650327	1.434282	6.248743	5.7
min	0.000000	0.000000	0.000000	0.000000	49.1
25%	0.020000	0.020000	0.050000	0.270000	95.8
50%	0.070000	0.100000	0.280000	1.160000	98.9
75%	0.430000	0.427500	0.860000	3.735000	99.8
max	3.640000	22.140000	12.280000	95.630000	100.(

8 rows × 30 columns

In [26]:
```
# Data volume
df2021.shape
```

Out[26]: (374, 151)

2022

In [27]:
```
df2022 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202209_mobile_laua_r0
df2022.head()
```

Out[27]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_count
0	E06000001	Hartlepool	9345	47806	665	0
1	E06000002	Middlesbrough	5387	69048	760	0
2	E06000003	Redcar and Cleveland	24471	69021	1469	0
3	E06000004	Stockton-on-Tees	20517	93406	1426	0
4	E06000005	Darlington	19746	55743	1114	127

5 rows × 151 columns

In [28]:
```
df2022.tail()
```

Out[28]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_co
369	W06000020	Torfaen	12612	46389	674	
370	W06000021	Monmouthshire	85053	46422	3680	
371	W06000022	Newport	19050	73210	1272	
372	W06000023	Powys	519528	70474	15700	
373	W06000024	Merthyr Tydfil	11178	28617	874	

5 rows × 151 columns

In [29]:
```
# Data types
df2022.dtypes
```

Out[29]:
```
laua                    object
laua_name               object
pixel_count              int64
prem_count               int64
ab_rd_count              int64
                         ...
Data_mway_ard_in_0     float64
Data_mway_ard_in_1     float64
Data_mway_ard_in_2     float64
Data_mway_ard_in_3     float64
Data_mway_ard_in_4     float64
Length: 151, dtype: object
```

In [30]:
```
# Select only numerical columns
numerical_columns4 = df2022.select_dtypes(include=['number'])

# Use describe on numerical columns
```

```
numerical_description4 = numerical_columns4.describe()

# Display the numerical description
display(numerical_description4)
```

	pixel_count	prem_count	ab_rd_count	mway_count	mway_ard_count
count	3.740000e+02	374.000000	374.000000	374.000000	374.000000
mean	6.515470e+04	84822.807487	2691.328877	136.516043	1749.267380
std	1.617038e+05	55941.528964	3502.205011	193.997066	2153.236784
min	2.900000e+02	1633.000000	88.000000	0.000000	88.000000
25%	9.346000e+03	50152.250000	906.500000	0.000000	700.000000
50%	2.766000e+04	68018.000000	1645.500000	32.500000	1243.500000
75%	6.713725e+04	102822.750000	3101.000000	235.000000	2024.000000
max	2.610989e+06	477349.000000	40780.000000	1053.000000	28997.000000

8 rows × 149 columns

In [31]:

```
# Find null values
df2022.isnull().sum()
```

Out[31]:

```
laua                  0
laua_name             0
pixel_count           0
prem_count            0
ab_rd_count           0
                   ...
Data_mway_ard_in_0    333
Data_mway_ard_in_1    304
Data_mway_ard_in_2    229
Data_mway_ard_in_3     91
Data_mway_ard_in_4      0
Length: 151, dtype: int64
```

In [32]:

```
# Investigate null values of 4G columns
import re

# Find columns containing '4g' (case insensitive)
columns_with_4g_4 = [col for col in df2022.columns if re.search(r'4G', co

# Check for missing values and calculate the sum for columns containing '
missing_values_4g_4 = df2022[columns_with_4g_4].isnull().sum()

print(missing_values_4g_4)
```

```
4G_prem_out_0       192
4G_prem_out_1       138
4G_prem_out_2        94
4G_prem_out_3        33
4G_prem_out_4         1
4G_prem_in_0         84
4G_prem_in_1         44
4G_prem_in_2         12
4G_prem_in_3          3
4G_prem_in_4          1
4G_geo_out_0        157
4G_geo_out_1        113
4G_geo_out_2         68
4G_geo_out_3         25
4G_geo_out_4          1
4G_abrd_in_0        134
4G_abrd_in_1         85
4G_abrd_in_2         33
4G_abrd_in_3         10
4G_abrd_in_4          1
4G_mway_in_0        369
4G_mway_in_1        343
4G_mway_in_2        250
4G_mway_in_3        181
4G_mway_in_4        168
4G_mway_ard_in_0    169
4G_mway_ard_in_1    101
4G_mway_ard_in_2     38
4G_mway_ard_in_3     10
4G_mway_ard_in_4      1
dtype: int64
```

In [33]:
```
# Investigate columns of 4G
df2022[columns_with_4g_4].describe()
```

Out[33]:

	4G_prem_out_0	4G_prem_out_1	4G_prem_out_2	4G_prem_out_3	4G_pre
count	182.000000	236.000000	280.000000	341.000000	37
mean	0.247582	0.429873	0.692393	2.539120	9
std	0.493976	1.104297	1.413166	6.417593	
min	0.000000	0.000000	0.000000	0.000000	4
25%	0.002500	0.007500	0.020000	0.130000	9
50%	0.050000	0.065000	0.170000	0.800000	9
75%	0.265000	0.397500	0.690000	3.210000	9
max	3.150000	12.930000	12.870000	96.140000	10

8 rows × 30 columns

In [34]:
```
# Data Volume
df2022.shape
```

Out[34]: (374, 151)

2023

In [35]:
```
df2023 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202304_mobile_laua_r0
df2023.head()
```

Out[35]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_count
0	S12000033	Aberdeen City	18582	129314	1624	NaN
1	S12000034	Aberdeenshire	631850	128401	20866	NaN
2	E07000223	Adur	4220	30000	349	NaN
3	E07000026	Allerdale	125779	52457	4713	NaN
4	E07000032	Amber Valley	26544	62594	1927	NaN

5 rows × 151 columns

In [36]:
```
df2023.tail()
```

Out[36]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_count
369	W06000006	Wrexham	50373	66935	3360	NaN
370	E07000238	Wychavon	66351	64446	3274	330.0
371	E07000128	Wyre	28170	58312	1240	151.0
372	E07000239	Wyre Forest	19548	49042	1434	NaN
373	E06000014	York	27192	96675	1570	NaN

5 rows × 151 columns

In [37]:
```
# Data types
df2023.dtypes
```

Out[37]:
```
laua                     object
laua_name                object
pixel_count               int64
prem_count                int64
ab_rd_count               int64
                          ...
Data_mway_ard_in_0      float64
Data_mway_ard_in_1      float64
Data_mway_ard_in_2      float64
Data_mway_ard_in_3      float64
Data_mway_ard_in_4      float64
Length: 151, dtype: object
```

In [38]:
```
# Investigate the columns of 4G
# Select only numerical columns
numerical_columns5 = df2023.select_dtypes(include=['number'])

# Use describe on numerical columns
numerical_description5 = numerical_columns5.describe()
```

```
# Display the numerical description
display(numerical_description5)
```

	pixel_count	prem_count	ab_rd_count	mway_count	mway_ard_count
count	3.740000e+02	374.000000	374.000000	207.000000	374.000000
mean	6.506730e+04	85090.874332	2689.344920	245.845411	1747.446524
std	1.616895e+05	56189.759145	3507.636312	202.054869	2156.522050
min	2.960000e+02	1690.000000	90.000000	1.000000	90.000000
25%	9.351750e+03	50517.250000	907.250000	99.000000	701.750000
50%	2.766500e+04	68153.000000	1639.500000	207.000000	1240.500000
75%	6.629100e+04	102400.500000	3102.750000	331.000000	2001.750000
max	2.610287e+06	478203.000000	40853.000000	1051.000000	29047.000000

8 rows × 149 columns

In [39]:
```
df2023.isnull().sum()
```

Out[39]:
```
laua                   0
laua_name              0
pixel_count            0
prem_count             0
ab_rd_count            0
                    ... 
Data_mway_ard_in_0    324
Data_mway_ard_in_1    296
Data_mway_ard_in_2    214
Data_mway_ard_in_3     93
Data_mway_ard_in_4      0
Length: 151, dtype: int64
```

In [40]:
```
# Investifate null values of 4G
# Find columns containing '4g' (case insensitive)
columns_with_4g_5 = [col for col in df2023.columns if re.search(r'4G', co

# Check for missing values and calculate the sum for columns containing '
missing_values_4g_5 = df2023[columns_with_4g_5].isnull().sum()

print(missing_values_4g_5)
```

```
4G_prem_out_0       210
4G_prem_out_1       174
4G_prem_out_2       116
4G_prem_out_3        48
4G_prem_out_4         1
4G_prem_in_0         97
4G_prem_in_1         61
4G_prem_in_2         20
4G_prem_in_3          5
4G_prem_in_4          1
4G_geo_out_0        161
4G_geo_out_1        117
4G_geo_out_2         75
4G_geo_out_3         25
4G_geo_out_4          1
4G_abrd_in_0        137
4G_abrd_in_1         91
4G_abrd_in_2         38
4G_abrd_in_3         12
4G_abrd_in_4          1
4G_mway_in_0        370
4G_mway_in_1        347
4G_mway_in_2        259
4G_mway_in_3        185
4G_mway_in_4        167
4G_mway_ard_in_0    176
4G_mway_ard_in_1    111
4G_mway_ard_in_2     44
4G_mway_ard_in_3     15
4G_mway_ard_in_4      1
dtype: int64
```

In [41]:
```
# Data volume
df2023.shape
```

Out[41]:
```
(374, 151)
```

For the first question:

4G_prem_out_4, 4G_prem_in_4, 4G_geo_out_4, 4G_abrd_in_4, and 4G_mway_in_4, among the 4G data columns, have only one null value and enough data to apply the analysis from 2019 to 2023. Therefore, these data will be used."

Explore to visualise maps further for the first question

In [42]:
```
# Use geojson data to visualise maps
import geopandas as gpd

geojson_path = '2023J_TMA02_data/Boundaries/Local_Authority_Districts_Dec
geojson_data = gpd.read_file(geojson_path)
```

In [43]:
```
print(geojson_data.columns)
```

```
Index(['FID', 'LAD22CD', 'LAD22NM', 'BNG_E', 'BNG_N', 'LONG', 'LAT',
       'GlobalID', 'geometry'],
      dtype='object')
```

In [44]:
```python
# Mapping dictionary for LAD22CD to nation
lad_to_nation = {
    'E': 'England',
    'N': 'Northern Ireland',
    'S': 'Scotland',
    'W': 'Wales'
}

# Create a new 'nation' column based on 'LAD22CD'
geojson_data['nation'] = geojson_data['LAD22CD'].str[0].map(lad_to_nation

# Print the updated DataFrame
print(geojson_data[['LAD22CD', 'nation']])
```

```
       LAD22CD   nation
0    E06000001  England
1    E06000002  England
2    E06000003  England
3    E06000004  England
4    E06000005  England
..         ...      ...
369  W06000020    Wales
370  W06000021    Wales
371  W06000022    Wales
372  W06000023    Wales
373  W06000024    Wales

[374 rows x 2 columns]
```

In [45]:
```python
# counts areas by nation
geojson_data['nation'].value_counts()
```

Out[45]:
```
nation
England             309
Scotland             32
Wales                22
Northern Ireland     11
Name: count, dtype: int64
```

In [46]:
```python
import folium
import json

# Convert 'nation' column to string to handle potential data type issues
geojson_data['nation'] = geojson_data['nation'].astype(str)

# Drop rows with NaN values in the 'nation' column
geojson_data = geojson_data.dropna(subset=['nation'])

# Create a Folium map centered at the mean latitude and longitude
map_center = [geojson_data['LAT'].mean(), geojson_data['LONG'].mean()]
mymap = folium.Map(location=map_center, zoom_start=6)

# Define a color mapping for each nation
color_mapping = {
    'England': 'blue',
    'Northern Ireland': 'green',
```

```
    'Scotland': 'red',
    'Wales': 'purple'
}

# Create a GeoJson object and add it to the map
folium.GeoJson(
    geojson_data,
    name='geojson',
    style_function=lambda feature: {
        'fillColor': color_mapping.get(feature['properties']['nation'], '
        'color': 'black',
        'weight': 2,
        'fillOpacity': 0.7,
    },
    tooltip=folium.features.GeoJsonTooltip(fields=['nation'], aliases=['N
).add_to(mymap)

# Add LayerControl to the map
folium.LayerControl().add_to(mymap)

# Display the map
mymap
```

Out[46]: Make this Notebook Trusted to load map: File -> Trust Notebook

The map above: No description has been provided for this image

This shows that it is possible to visualize nations using a choropleth map. Therefore, a comparison of 4G will be conducted.

Explore the segmentation to combine mobile data with the fixed broadband coverage dataset

In [47]:

```
#Try to segment the data using df2023
df2023
```

Out[47]:

	laua	laua_name	pixel_count	prem_count	ab_rd_count	mway_cou
0	S12000033	Aberdeen City	18582	129314	1624	N
1	S12000034	Aberdeenshire	631850	128401	20866	N
2	E07000223	Adur	4220	30000	349	N
3	E07000026	Allerdale	125779	52457	4713	N
4	E07000032	Amber Valley	26544	62594	1927	N
...	...	...	...	...	...	
369	W06000006	Wrexham	50373	66935	3360	N
370	E07000238	Wychavon	66351	64446	3274	33
371	E07000128	Wyre	28170	58312	1240	15
372	E07000239	Wyre Forest	19548	49042	1434	N
373	E06000014	York	27192	96675	1570	N

374 rows × 151 columns

In [48]:

```
# Hace a look at '4G_prem_in_4' as an example
df2023['4G_prem_in_4']
```

Out[48]:

```
0      96.14
1      67.92
2      90.81
3      65.14
4      86.85
       ...
369    72.08
370    73.02
371    74.79
372    79.92
373    89.69
Name: 4G_prem_in_4, Length: 374, dtype: float64
```

In [49]:

```
# Calculate the percentile

data_column = df2023['4G_prem_in_4']

# Calculate percentiles excluding NaN values
percentiles = np.nanpercentile(data_column, [25, 50, 75])

# Display the results
print("25th Percentile:", percentiles[0])
print("50th Percentile (Median):", percentiles[1])
print("75th Percentile:", percentiles[2])
```

```
25th Percentile: 72.72
50th Percentile (Median): 85.3
75th Percentile: 93.66
```

In [50]:

```
# Based on the percentile, segment the group

# Assuming df2023 is your DataFrame and '4G_prem_in_4' is the column you'
data_column = df2023['4G_prem_in_4']
```

```python
# Calculate percentiles excluding NaN values
percentiles = np.nanpercentile(data_column, [25, 50, 75])

# Segment data into four groups
bins = [-np.inf, percentiles[0], percentiles[1], percentiles[2], np.inf]
group_labels = ['Group 1', 'Group 2', 'Group 3', 'Group 4']

# Create a copy of the DataFrame
df2023_new = df2023.copy()

# Add a new column to the copied DataFrame indicating the group each valu
df2023_new['Group'] = pd.cut(data_column, bins=bins, labels=group_labels,

# Display the results
display(df2023_new[['4G_prem_in_4', 'Group']])
```

	4G_prem_in_4	Group
0	96.14	Group 4
1	67.92	Group 1
2	90.81	Group 3
3	65.14	Group 1
4	86.85	Group 3
...	...	...
369	72.08	Group 1
370	73.02	Group 2
371	74.79	Group 2
372	79.92	Group 2
373	89.69	Group 3

374 rows × 2 columns

In [51]:
```python
dfNew2023 = df2023_new[['laua','laua_name','4G_prem_in_4', 'Group']]
dfNew2023
```

Out[51]:

	laua	laua_name	4G_prem_in_4	Group
0	S12000033	Aberdeen City	96.14	Group 4
1	S12000034	Aberdeenshire	67.92	Group 1
2	E07000223	Adur	90.81	Group 3
3	E07000026	Allerdale	65.14	Group 1
4	E07000032	Amber Valley	86.85	Group 3
...	...	...	...	...
369	W06000006	Wrexham	72.08	Group 1
370	E07000238	Wychavon	73.02	Group 2
371	E07000128	Wyre	74.79	Group 2
372	E07000239	Wyre Forest	79.92	Group 2
373	E06000014	York	89.69	Group 3

374 rows × 4 columns

In [52]:

```
# Make sure the values of 'Group'
dfNew2023['Group'].value_counts()
```

Out[52]:

```
Group
Group 1    94
Group 3    94
Group 2    93
Group 4    92
Name: count, dtype: int64
```

In [53]:

```
# Investigate null values
dfNew2023['Group'].isnull().sum()
```

Out[53]:

```
1
```

In [54]:

```
# data type
dfNew2023.dtypes
```

Out[54]:

```
laua              object
laua_name         object
4G_prem_in_4     float64
Group           category
dtype: object
```

In [55]:

```
# Combine dfNew2023 with geojson_data

# Convert both columns to lowercase before merging
geojson_data['LAD22NM'] = geojson_data['LAD22NM'].str.lower()
dfNew2023['laua_name'] = dfNew2023['laua_name'].str.lower()

# Use .loc to avoid SettingWithCopyWarning
dfNew2023.loc[:, 'laua_name'] = dfNew2023['laua_name'].str.lower()

# Merge the DataFrames on the lowercase columns
df_combined_geo = pd.merge(geojson_data, dfNew2023, left_on='LAD22NM', ri
```

```python

# Drop the temporary lowercase columns
df_combined_geo = df_combined_geo.drop(columns=['LAD22NM'])

# Display the resulting DataFrame
display(df_combined_geo)
```

```
/tmp/ipykernel_171/370982054.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfNew2023['laua_name'] = dfNew2023['laua_name'].str.lower()
```

	FID	LAD22CD	BNG_E	BNG_N	LONG	LAT	GlobalID	
0	339	S12000033	387763	808479	-2.20398	57.16697	bc868205-08c7-47a9-88af-6cf72d57fbc8	MULTI ((-2.0
1	340	S12000034	352284	816277	-2.79208	57.23469	e9d43203-2a53-46ca-a3a6-762a33027034	MULTI ((-1.8
2	221	E07000223	518076	106472	-0.32417	50.84572	d56eedd2-29d2-40f8-a6b8-5f0b952ba9a4	(5
3	65	E07000026	317520	532997	-3.28090	54.68524	e6e9b854-1a61-4f46-928c-8ca3a83edbfc	5
4	71	E07000032	436166	348084	-1.46219	53.02884	2ff039cf-0a31-4ac2-a02a-beca575e827c	(5
...	...	...	...	...	...	...	...	
369	358	W06000006	333523	345387	-2.99203	53.00167	25f2369e-e2ec-44f1-86d0-45834489f0c9	(5
370	232	E07000238	398991	247839	-2.01614	52.12886	50dcf59b-5b4a-4ce9-8100-92e58cc1535a	5
371	150	E07000128	347295	445159	-2.80359	53.89991	480d5620-7908-4bd2-9ff4-656e7fd5a8be	MULTI ((-2.8
372	233	E07000239	384106	276388	-2.23494	52.38530	190f331e-8cdc-4a62-b8fd-f221930c5739	5
373	14	E06000014	460864	452589	-1.07375	53.96582	9b08b382-c47d-4230-8908-99ac65b82925	(5

374 rows × 13 columns

```
In [56]: # Check unique values and data types in the 'Group' column
print(df_combined_geo['Group'].unique())
print(df_combined_geo['Group'].dtype)

# Convert 'Group' column to string type
df_combined_geo['Group'] = df_combined_geo['Group'].astype(str)

# Print again to confirm changes
print(df_combined_geo['Group'].unique())
print(df_combined_geo['Group'].dtype)
```

```
['Group 4', 'Group 1', 'Group 3', 'Group 2', NaN]
Categories (4, object): ['Group 1' < 'Group 2' < 'Group 3' < 'Group 4']
category
['Group 4' 'Group 1' 'Group 3' 'Group 2' 'nan']
object
```

```
In [57]: import folium

# Map 'Group' values to integers
group_mapping = {'nan': 0, 'Group 1': 1, 'Group 2': 2, 'Group 3': 3, 'Gro


df_combined_geo['Group_Int'] = df_combined_geo['Group'].map(group_mapping

# Convert 'Group_Int' column to a list of integers
group_int_list = df_combined_geo['Group_Int'].astype(int).tolist()

# Create the GeoJSON file
df_combined_geo.to_file("geo_data.json", driver="GeoJSON")

# Create the map
map_combined = folium.Map(location=[df_combined_geo['LAT'].mean(), df_com

# Add choropleth layer for the change in the number of premises
folium.Choropleth(
    geo_data="geo_data.json",
    data=df_combined_geo,
    columns=['FID', 'Group_Int'],
    key_on='feature.properties.FID',
    fill_color='YlGnBu',  # You can choose a different color scale
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name='Group'
).add_to(map_combined)

# Display the map
map_combined
```

Out [57]: Make this Notebook Trusted to load map: File -> Trust Notebook

The map above: No description has been provided for this image

The analysis was conducted by segmenting the values of '4G_prem_in_4'. It is clear that 4G usage in Greater London is advanced. Similarly, concerning the combination with the fixed broadband coverage dataset, data from both mobile coverage and fixed broadband will be segmented together and applied to a map. For instance, high usage of 4G and Full Fibre is categorized as Group1, while moderate usage of 4G and Full Fibre belongs to Group4 or Group5. Since each dataset can be divided into four based on percentile rankings, the combination of both datasets results in eight distinct groups

Scatter Plot with mobile data and the fixed broadband coverage dataset¶

In [59]:
```
# Extract the fixed broadband coverage dataset in 2023
# Read the CSV file into a DataFrame
df2023_fixed = pd.read_csv('2023J_TMA02_data/Ofcom_fixed/202305_fixed_lau
df2023_fixed.head()
```

Out[59]:

	laua	laua_name	All Premises	All Matched Premises	SFBB availability (% premises)	UFBB (100Mbit/s) availability (% premises)	avail pre
0	S12000033	ABERDEEN CITY	129315	129197	97.2	84.8	
1	S12000034	ABERDEENSHIRE	128408	128070	85.9	25.5	
2	E07000223	ADUR	29985	29953	99.1	92.8	
3	E07000026	ALLERDALE	52482	52364	93.1	6.0	
4	E07000032	AMBER VALLEY	62512	62430	97.2	62.4	

5 rows × 40 columns

In [63]:

```
# Extract 4G_prem_in_4 from the mobile coverage and
#Full Fibre availability (% premises) from the fixed broadband coverage d

df_New2023_mobile = df2023_new[['laua','laua_name','4G_prem_in_4']]

df2023_fixed = df2023_fixed[['laua','laua_name', 'Full Fibre availability

display(df_New2023_mobile.head())
display(df2023_fixed.head())
```

	laua	laua_name	4G_prem_in_4
0	S12000033	Aberdeen City	96.14
1	S12000034	Aberdeenshire	67.92
2	E07000223	Adur	90.81
3	E07000026	Allerdale	65.14
4	E07000032	Amber Valley	86.85

	laua	laua_name	Full Fibre availability (% premises)
0	S12000033	ABERDEEN CITY	83.0
1	S12000034	ABERDEENSHIRE	25.4
2	E07000223	ADUR	65.4
3	E07000026	ALLERDALE	6.0
4	E07000032	AMBER VALLEY	59.0

In [65]:

```
# Convert 'laua_name' column in df_New2023_mobile to lowercase
df_New2023_mobile['laua_name'] = df_New2023_mobile['laua_name'].str.lower

# Convert 'laua_name' column in df2023_fixed to lowercase
df2023_fixed['laua_name'] = df2023_fixed['laua_name'].str.lower()

# Merge dataframes based on 'laua' and 'laua_name'
```

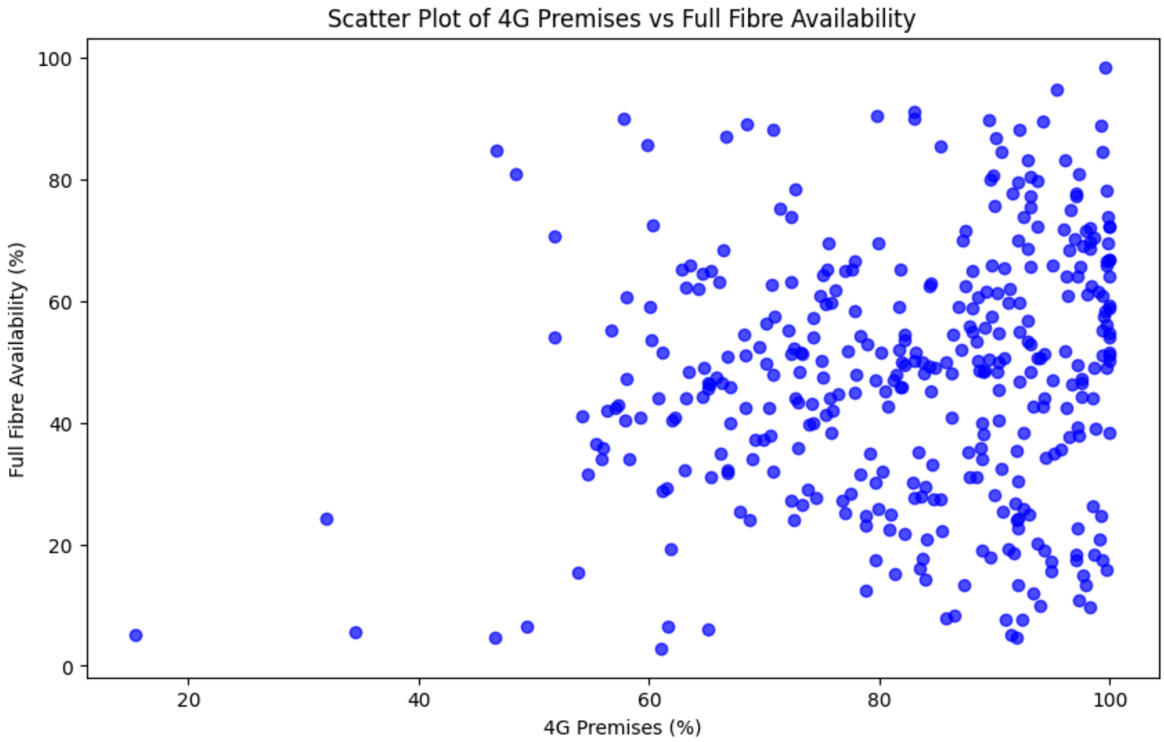
```
combined_df = pd.merge(df_New2023_mobile, df2023_fixed, on=['laua', 'laua

# Display the resulting dataframe
display(combined_df.head())
```

```
/tmp/ipykernel_171/3971252743.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_New2023_mobile['laua_name'] = df_New2023_mobile['laua_name'].str.lowe
r()
```

	laua	laua_name	4G_prem_in_4	Full Fibre availability (% premises)
0	S12000033	aberdeen city	96.14	83.0
1	S12000034	aberdeenshire	67.92	25.4
2	E07000223	adur	90.81	65.4
3	E07000026	allerdale	65.14	6.0
4	E07000032	amber valley	86.85	59.0

In [67]:

```
# Create the scatter plot

import matplotlib.pyplot as plt

# Data
laua_names = combined_df['laua_name']
prem_4G = combined_df['4G_prem_in_4']
full_fibre = combined_df['Full Fibre availability (% premises)']

# Scatter plot
plt.figure(figsize=(10, 6))
plt.scatter(prem_4G, full_fibre, color='blue', alpha=0.7)
plt.title('Scatter Plot of 4G Premises vs Full Fibre Availability')
plt.xlabel('4G Premises (%)')
plt.ylabel('Full Fibre Availability (%)')



plt.show()
```



Looking at the scatter plot, there is no correlation observed so far. It shows that, while there are high percentages of 4G usage, Full Fiber values have a wide range of percentages. Similarly, in the EMA, further investigation of 4G usage and Full Fiber using scatter plots will be conducted to find any interesting insights such as the correlation between 4G availability and Full Fibre availability.

```
In [ ]:
```