Import the dataset on the coverage of mobile networks

The data of each year were investigated in advance by Project diary.

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]:
```
# 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,

columns_with_4g
```

Out[6]:
```
['4G_prem_out_0',
 '4G_prem_out_1',
 '4G_prem_out_2',
 '4G_prem_out_3',
 '4G_prem_out_4',
 '4G_prem_in_0',
 '4G_prem_in_1',
 '4G_prem_in_2',
 '4G_prem_in_3',
 '4G_prem_in_4',
 '4G_geo_out_0',
 '4G_geo_out_1',
 '4G_geo_out_2',
 '4G_geo_out_3',
 '4G_geo_out_4',
 '4G_abrd_in_0',
 '4G_abrd_in_1',
 '4G_abrd_in_2',
 '4G_abrd_in_3',
 '4G_abrd_in_4',
 '4G_mway_in_0',
 '4G_mway_in_1',
 '4G_mway_in_2',
 '4G_mway_in_3',
 '4G_mway_in_4',
 '4G_mway_ard_in_0',
 '4G_mway_ard_in_1',
 '4G_mway_ard_in_2',
 '4G_mway_ard_in_3',
 '4G_mway_ard_in_4']
```

In [7]:
```
# choose relevant columns: laua and 4G

df2019 = df2019[['laua', 'laua_name', '4G_prem_out_4', '4G_prem_in_4', '4
df2019
```

Out[7]:

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4
0	E06000001	Hartlepool	98.56	58.84	95.21
1	E06000002	Middlesbrough	100.00	82.00	99.96
2	E06000003	Redcar and Cleveland	99.60	88.69	93.10
3	E06000004	Stockton-on-Tees	99.63	80.52	93.38
4	E06000005	Darlington	99.74	86.91	97.41
...	...	...	...	...	...
377	W06000020	Torfaen	99.46	85.10	88.04
378	W06000021	Monmouthshire	89.20	70.38	53.50
379	W06000022	Newport	99.20	85.56	94.82
380	W06000023	Powys	82.90	56.96	48.10
381	W06000024	Merthyr Tydfil	94.24	81.04	77.40

382 rows × 8 columns

There are many columns about 4G. To facilitate comparison across areas, it is necessary for each local authority to have a single value. Therefore, the values for each location are averaged and grouped by area.

In [8]:
```
# Create the column: average 4G percentage

# Selecting columns with 4G data
df_4g = df2019[['4G_prem_out_4', '4G_prem_in_4', '4G_geo_out_4', '4G_abrd

# Calculating the mean along the rows axis and assigning it to the new co
df2019.loc[:, 'AVG_4G'] = df_4g.mean(axis=1).round(2)

# Displaying the DataFrame with the new column


display(df2019)
```

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4	4
0	E06000001	Hartlepool	98.56	58.84	95.21	
1	E06000002	Middlesbrough	100.00	82.00	99.96	
2	E06000003	Redcar and Cleveland	99.60	88.69	93.10	
3	E06000004	Stockton-on-Tees	99.63	80.52	93.38	
4	E06000005	Darlington	99.74	86.91	97.41	
...	...	...	...	...	...	
377	W06000020	Torfaen	99.46	85.10	88.04	
378	W06000021	Monmouthshire	89.20	70.38	53.50	
379	W06000022	Newport	99.20	85.56	94.82	
380	W06000023	Powys	82.90	56.96	48.10	
381	W06000024	Merthyr Tydfil	94.24	81.04	77.40	

382 rows × 9 columns

2020

In [9]:
```
df2020 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202009_mobile_laua_r0
df2020.head()
```

Out[9]:

	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 [10]:
```
df2020.tail()
```

Out[10]:

	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 [11]:
```
# choose relevant columns: laua and 4G

df2020 = df2020[['laua', 'laua_name', '4G_prem_out_4', '4G_prem_in_4', '4
df2020
```

Out[11]:

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4
0	E06000001	Hartlepool	98.60	55.17	96.09
1	E06000002	Middlesbrough	99.97	77.83	99.89
2	E06000003	Redcar and Cleveland	99.40	88.74	90.98
3	E06000004	Stockton-on-Tees	99.70	81.44	93.84
4	E06000005	Darlington	99.63	86.81	97.42
...	...	...	...	...	...
374	W06000020	Torfaen	99.52	84.44	86.34
375	W06000021	Monmouthshire	90.15	72.49	57.06
376	W06000022	Newport	99.34	89.72	95.87
377	W06000023	Powys	82.15	60.76	49.43
378	W06000024	Merthyr Tydfil	94.36	81.53	76.87

379 rows × 8 columns

In [12]:
```
# Create the column: average 4G percentage

# Selecting columns with 4G data
df_4g_1 = df2020[['4G_prem_out_4', '4G_prem_in_4', '4G_geo_out_4', '4G_ab

# Calculating the mean along the rows axis and assigning it to the new co
df2020.loc[:, 'AVG_4G'] = df_4g_1.mean(axis=1).round(2)

# Displaying the DataFrame with the new column


display(df2020)
```

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4	4
0	E06000001	Hartlepool	98.60	55.17	96.09	
1	E06000002	Middlesbrough	99.97	77.83	99.89	
2	E06000003	Redcar and Cleveland	99.40	88.74	90.98	
3	E06000004	Stockton-on-Tees	99.70	81.44	93.84	
4	E06000005	Darlington	99.63	86.81	97.42	
...	...	...	...	...	...	
374	W06000020	Torfaen	99.52	84.44	86.34	
375	W06000021	Monmouthshire	90.15	72.49	57.06	
376	W06000022	Newport	99.34	89.72	95.87	
377	W06000023	Powys	82.15	60.76	49.43	
378	W06000024	Merthyr Tydfil	94.36	81.53	76.87	

379 rows × 9 columns

2021

In [13]:
```
df2021 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202109_mobile_laua_r0
df2021.head()
```

Out[13]:

	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 [14]:
```
df2021.columns
```

Out[14]:
```
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 [15]:
```
df2021.tail()
```

Out[15]:

	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 [16]:
```
# choose relevant columns: laua and 4G

df2021 = df2021[['laua', 'laua_name', '4G_prem_out_4', '4G_prem_in_4', '4

# Selecting columns with 4G data
df_4g_2 = df2021[['4G_prem_out_4', '4G_prem_in_4', '4G_geo_out_4', '4G_ab

# Calculating the mean along the rows axis and assigning it to the new co
df2021.loc[:, 'AVG_4G'] = df_4g_2.mean(axis=1).round(2)

# Displaying the DataFrame with the new column


display(df2021)
```

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4	4
0	E06000001	Hartlepool	98.75	62.46	96.25	
1	E06000002	Middlesbrough	99.82	75.20	99.61	
2	E06000003	Redcar and Cleveland	99.43	88.32	89.88	
3	E06000004	Stockton-on-Tees	99.61	81.86	93.54	
4	E06000005	Darlington	99.73	88.35	97.57	
...	...	...	...	...	...	
369	W06000020	Torfaen	99.24	81.58	86.58	
370	W06000021	Monmouthshire	89.96	71.21	58.41	
371	W06000022	Newport	99.41	89.57	96.31	
372	W06000023	Powys	82.31	61.69	49.60	
373	W06000024	Merthyr Tydfil	94.51	81.99	77.65	

374 rows × 9 columns

2022

In [17]:
```
df2022 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202209_mobile_laua_r0
df2022.head()
```

Out[17]:

	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 [18]:
```
df2022.tail()
```

Out[18]:

	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 [19]:
```
# choose relevant columns: laua and 4G

df2022 = df2022[['laua', 'laua_name', '4G_prem_out_4', '4G_prem_in_4', '4

# Selecting columns with 4G data
df_4g_3 = df2022[['4G_prem_out_4', '4G_prem_in_4', '4G_geo_out_4', '4G_ab

# Calculating the mean along the rows axis and assigning it to the new co
df2022.loc[:, 'AVG_4G'] = df_4g_3.mean(axis=1).round(2)

# Displaying the DataFrame with the new column


display(df2022)
```

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4	4
0	E06000001	Hartlepool	99.64	60.69	96.46	
1	E06000002	Middlesbrough	100.00	87.08	99.87	
2	E06000003	Redcar and Cleveland	99.49	84.34	93.44	
3	E06000004	Stockton-on-Tees	99.66	79.52	96.48	
4	E06000005	Darlington	99.77	88.22	98.19	
...	...	...	...	...	...	
369	W06000020	Torfaen	99.67	86.24	87.84	
370	W06000021	Monmouthshire	91.36	75.09	60.40	
371	W06000022	Newport	99.52	91.54	97.10	
372	W06000023	Powys	82.79	61.90	51.06	
373	W06000024	Merthyr Tydfil	94.30	81.28	79.17	

374 rows × 9 columns

2023

In [20]:
```
df2023 = pd.read_csv('2023J_TMA02_data/Ofcom_mobile/202304_mobile_laua_r0
df2023.head()
```

Out[20]:

	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 [21]:
```
df2023.tail()
```

Out[21]:

	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 [22]:

```
# choose relevant columns: laua and 4G

df2023 = df2023[['laua', 'laua_name', '4G_prem_out_4', '4G_prem_in_4', '4

# Selecting columns with 4G data
df_4g_4 = df2023[['4G_prem_out_4', '4G_prem_in_4', '4G_geo_out_4', '4G_ab

# Calculating the mean along the rows axis and assigning it to the new co
df2023.loc[:, 'AVG_4G'] = df_4g_4.mean(axis=1).round(2)

# Displaying the DataFrame with the new column


display(df2023)
```

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4	4G
0	S12000033	Aberdeen City	99.57	96.14	95.85	
1	S12000034	Aberdeenshire	89.95	67.92	62.99	
2	E07000223	Adur	99.92	90.81	87.61	
3	E07000026	Allerdale	95.27	65.14	73.10	
4	E07000032	Amber Valley	99.36	86.85	96.21	
...	...	...	...	...	...	
369	W06000006	Wrexham	96.09	72.08	76.72	
370	E07000238	Wychavon	98.13	73.02	95.92	
371	E07000128	Wyre	99.06	74.79	92.65	
372	E07000239	Wyre Forest	97.82	79.92	79.36	
373	E06000014	York	99.59	89.69	99.26	

374 rows × 9 columns

Merge 5 data

In [23]:

```
df2019_1 = df2019[['laua', 'laua_name', 'AVG_4G']]
df2019_1 = df2019_1.rename(columns={'AVG_4G': '2019'})
```

```
df2019_1.head(2)
```

Out[23]:

	laua	laua_name	2019
0	E06000001	Hartlepool	73.05
1	E06000002	Middlesbrough	86.68

In [24]:
```
df2020_1 = df2020[['laua', 'laua_name', 'AVG_4G']]
df2020_1 = df2020_1.rename(columns={'AVG_4G': '2020'})
df2020_1.head(2)
```

Out[24]:

	laua	laua_name	2020
0	E06000001	Hartlepool	72.89
1	E06000002	Middlesbrough	86.17

In [25]:
```
df2021_1 = df2021[['laua', 'laua_name', 'AVG_4G']]
df2021_1 = df2021_1.rename(columns={'AVG_4G': '2021'})
df2021_1.head(2)
```

Out[25]:

	laua	laua_name	2021
0	E06000001	Hartlepool	77.29
1	E06000002	Middlesbrough	83.39

In [26]:
```
df2022_1 = df2022[['laua', 'laua_name', 'AVG_4G']]
df2022_1 = df2022_1.rename(columns={'AVG_4G': '2022'})
df2022_1.head(2)
```

Out[26]:

	laua	laua_name	2022
0	E06000001	Hartlepool	78.15
1	E06000002	Middlesbrough	90.76

In [27]:
```
df2023_1 = df2023[['laua', 'laua_name', 'AVG_4G']]
df2023_1 = df2023_1.rename(columns={'AVG_4G': '2023'})
df2023_1.head(2)
```

Out[27]:

	laua	laua_name	2023
0	S12000033	Aberdeen City	93.60
1	S12000034	Aberdeenshire	65.64

In [28]:
```
# Merge 5 data

# Merging the DataFrames
merged_df = df2019_1.merge(df2020_1, on=['laua', 'laua_name']) \
                    .merge(df2021_1, on=['laua', 'laua_name']) \
                    .merge(df2022_1, on=['laua', 'laua_name']) \
                    .merge(df2023_1, on=['laua', 'laua_name'])

display(merged_df)
```

	laua	laua_name	2019	2020	2021	2022	2023
0	E06000001	Hartlepool	73.05	72.89	77.29	78.15	79.62
1	E06000002	Middlesbrough	86.68	86.17	83.39	90.76	92.62
2	E06000005	Darlington	82.35	87.16	88.45	89.07	89.98
3	E06000006	Halton	83.05	87.06	91.18	93.98	93.64
4	E06000007	Warrington	78.32	82.50	83.20	87.36	88.45
...	...	...	...	...	...	...	...
310	W06000020	Torfaen	77.01	81.74	79.63	81.19	84.67
311	W06000021	Monmouthshire	56.98	69.44	69.86	71.05	73.84
312	W06000022	Newport	81.62	90.38	90.88	91.33	93.72
313	W06000023	Powys	49.84	57.60	58.16	58.62	59.70
314	W06000024	Merthyr Tydfil	70.07	74.79	75.30	75.00	75.37

315 rows × 7 columns

In [29]:
```
merged_df.isnull().sum()
```

Out[29]:
```
laua         0
laua_name    0
2019         0
2020         0
2021         0
2022         0
2023         0
dtype: int64
```

The number of rows reduced from 345 to 315, indicating that some areas were not listed in certain years. This time, only the dataframes with non-null values will be considered.

Add the information about nation

To conduct a nation-by-nation analysis, information on each nation is added.

In [30]:
```
# 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)
print(geojson_data.columns)
```

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

In [31]:
```
# 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 [32]:
```
# counts areas by nation
geojson_data['nation'].value_counts()
```

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

In [33]:
```
geojson_data_1 = geojson_data[['LAD22CD', 'nation']]
display(geojson_data_1)
```

	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 × 2 columns

In [34]:
```
# Merging merged_df and geojson_data_1 on 'laua' and 'LAD22CD'
merged_nation = pd.merge(merged_df, geojson_data_1, left_on='laua', right

display(merged_nation)
```

	laua	laua_name	2019	2020	2021	2022	2023	LAD22CD	n
0	E06000001	Hartlepool	73.05	72.89	77.29	78.15	79.62	E06000001	En
1	E06000002	Middlesbrough	86.68	86.17	83.39	90.76	92.62	E06000002	En
2	E06000005	Darlington	82.35	87.16	88.45	89.07	89.98	E06000005	En
3	E06000006	Halton	83.05	87.06	91.18	93.98	93.64	E06000006	En
4	E06000007	Warrington	78.32	82.50	83.20	87.36	88.45	E06000007	En
...	...	...	...	...	...	...	...	...	
310	W06000020	Torfaen	77.01	81.74	79.63	81.19	84.67	W06000020	
311	W06000021	Monmouthshire	56.98	69.44	69.86	71.05	73.84	W06000021	
312	W06000022	Newport	81.62	90.38	90.88	91.33	93.72	W06000022	
313	W06000023	Powys	49.84	57.60	58.16	58.62	59.70	W06000023	
314	W06000024	Merthyr Tydfil	70.07	74.79	75.30	75.00	75.37	W06000024	

315 rows × 9 columns

Create a plot by nation

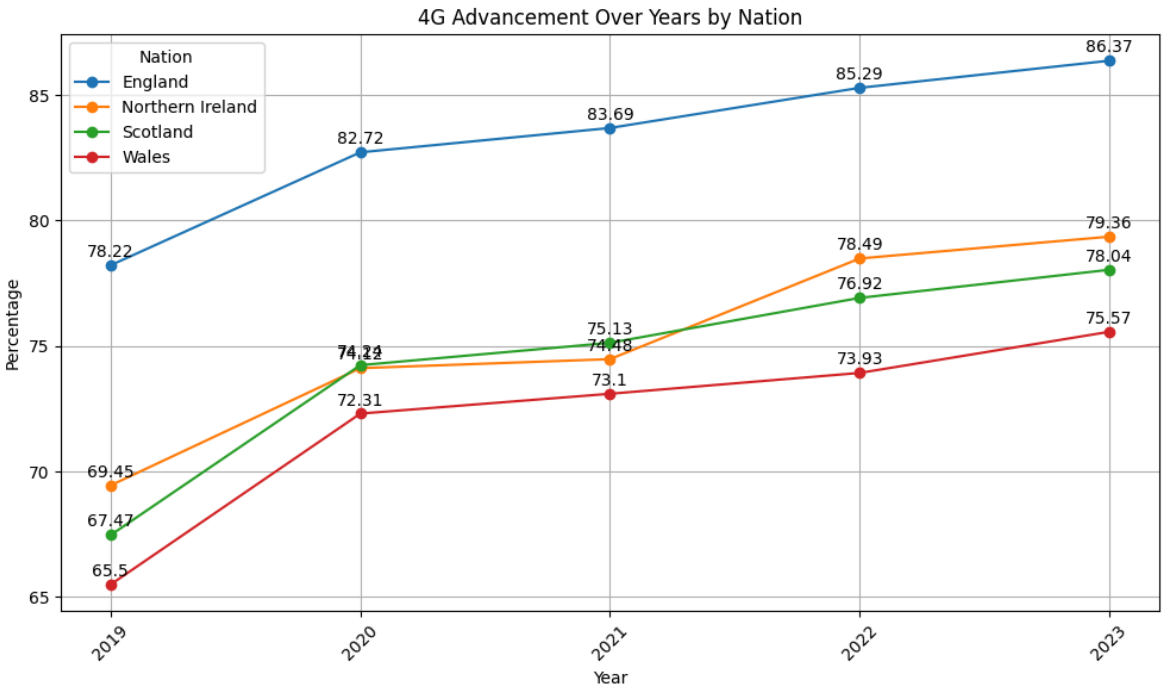
The advancement of 4G over a five-year period is depicted for each nation using a line graph to capture the trend.

In [35]:
```
# Group by 'nation' and calculate the mean for each year
grouped_nation = merged_nation.groupby('nation')[['2019', '2020', '2021',

display(grouped_nation)
```

	nation	2019	2020	2021	2022	2023
0	England	78.22	82.72	83.69	85.29	86.37
1	Northern Ireland	69.45	74.12	74.48	78.49	79.36
2	Scotland	67.47	74.24	75.13	76.92	78.04
3	Wales	65.50	72.31	73.10	73.93	75.57

In [ ]:
```
```

In [36]:
```
plt.figure(figsize=(10, 6))

for i in range(len(grouped_nation)):
    nation = grouped_nation['nation'].iloc[i]
    values = grouped_nation.iloc[i, 1:]
    plt.plot(grouped_nation.columns[1:], values, marker='o', label=nation

    # Annotate each point with the value
    for j, value in enumerate(values):
        plt.annotate(f'{value}', (grouped_nation.columns[1:][j], values.i

plt.title('4G Advancement Over Years by Nation')
plt.xlabel('Year')
plt.ylabel('Percentage')
plt.legend(title='Nation')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()

plt.show()
```



Among the nations, 4G coverage spread the most in England, averaging over 85% in 2023. Following England, Northern Ireland showed significant improvement, starting at less than 70% and reaching nearly 80% in 2023. Scotland experienced a similar trend to Northern Ireland, even surpassing it between 2020 and 2021. Wales saw a significant increase in coverage from 2019 to 2020 (65.50% to 72.31%). However, after that, Wales's coverage gradually increased, reaching 75.57% in 2023.

Increase for 5 years

Each nation experienced a similar increase from 2019 to 2023. Therefore, a bar graph showing the percentage increase from 2019 to 2023 is created.

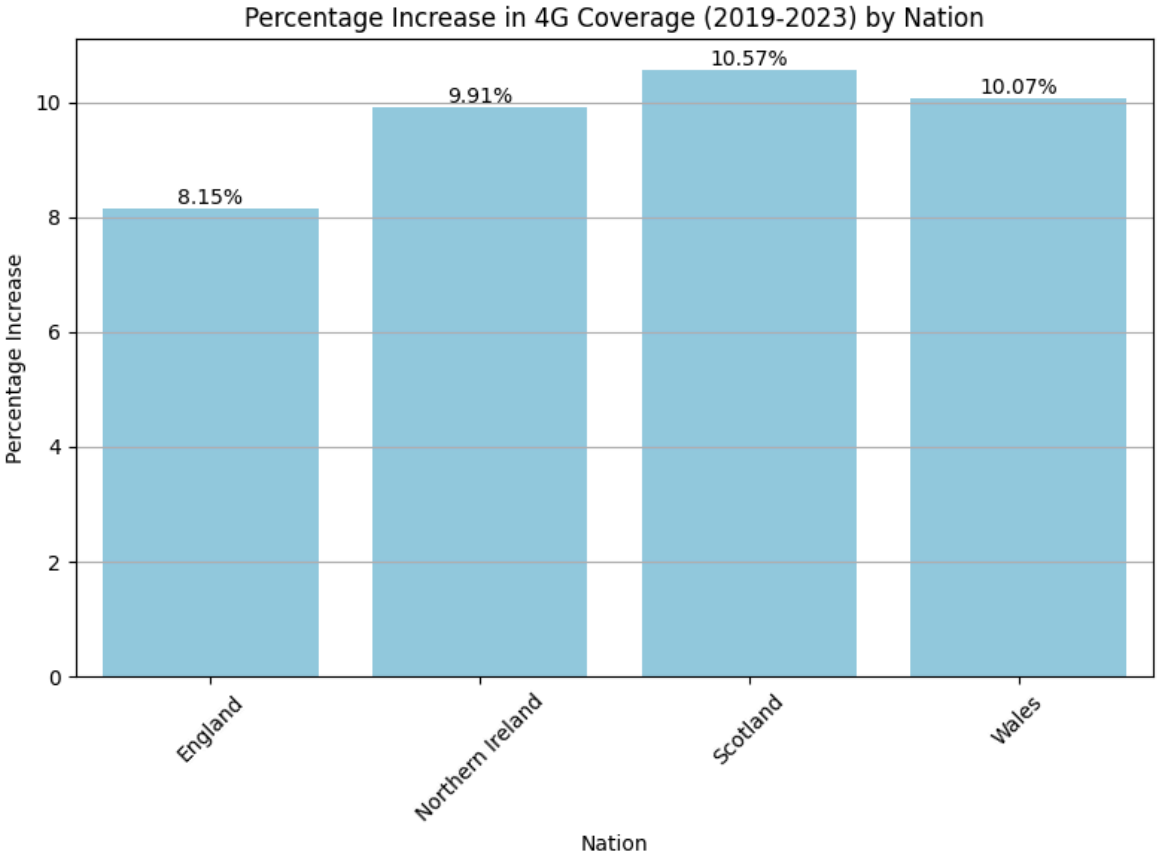
In [37]:
```
grouped_nation_1 = grouped_nation.copy()
```

In [38]:
```
grouped_nation_1['increase'] = grouped_nation_1['2023'] - grouped_nation_
```

In [39]:
```
display(grouped_nation_1)
```

	nation	2019	2020	2021	2022	2023	increase
0	England	78.22	82.72	83.69	85.29	86.37	8.15
1	Northern Ireland	69.45	74.12	74.48	78.49	79.36	9.91
2	Scotland	67.47	74.24	75.13	76.92	78.04	10.57
3	Wales	65.50	72.31	73.10	73.93	75.57	10.07

In [40]:
```
import seaborn as sns


# Create bar plot using Seaborn
plt.figure(figsize=(8, 6))
sns.barplot(x='nation', y='increase', data=grouped_nation_1, color='skybl
plt.title('Percentage Increase in 4G Coverage (2019-2023) by Nation')
plt.xlabel('Nation')
plt.ylabel('Percentage Increase')
plt.xticks(rotation=45)
plt.grid(axis='y')

# Annotate bars with values
for index, row in grouped_nation_1.iterrows():
    plt.text(index, row['increase'], f'{row["increase"]:.2f}%', ha='cente

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



Looking at the bar graph, Scotland saw the most increase in 4G coverage between 2019 and 2023.

2023 Local authorities

The current situation of 4G advancement is investigated at a granular level based on the recent data from 2023.

In [41]:
```
df2023.head()
```

Out[41]:

	laua	laua_name	4G_prem_out_4	4G_prem_in_4	4G_geo_out_4	4G_
0	S12000033	Aberdeen City	99.57	96.14	95.85	
1	S12000034	Aberdeenshire	89.95	67.92	62.99	
2	E07000223	Adur	99.92	90.81	87.61	
3	E07000026	Allerdale	95.27	65.14	73.10	
4	E07000032	Amber Valley	99.36	86.85	96.21	

In [42]:
```
# retrieve inforamtion of 2023


df_2023_info = df2023[['laua', 'laua_name', 'AVG_4G']]

# Print or use the resulting DataFrame
display(df_2023_info)
```

	laua	laua_name	AVG_4G
0	S12000033	Aberdeen City	93.60
1	S12000034	Aberdeenshire	65.64
2	E07000223	Adur	87.41
3	E07000026	Allerdale	68.39
4	E07000032	Amber Valley	87.89
...	...	...	...
369	W06000006	Wrexham	71.34
370	E07000238	Wychavon	83.28
371	E07000128	Wyre	83.07
372	E07000239	Wyre Forest	78.90
373	E06000014	York	92.40

374 rows × 3 columns

To analyse the current situation in the UK and explore variations between areas at a granular level, the data from 2023 is segmented into four groups (very high, high, medium, low). This segmentation is based on percentile statistics (25th, 50th, and 75th), and the results are then visualised on a map.

In [43]:
```
# Calculate the percentile

data_column = df_2023_info['AVG_4G']

# 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: 77.71
50th Percentile (Median): 86.58
75th Percentile: 93.75
```

In [44]:
```
# Based on the percentile, segment the group

# Assuming df_2023_info is your DataFrame and 'AVG_4G' is the column you'
data_column = df_2023_info['AVG_4G']

# 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
df_2023_info_new = df_2023_info.copy()
```

```
# Add a new column to the copied DataFrame indicating the group each valu
df_2023_info_new['Group'] = pd.cut(data_column, bins=bins, labels=group_l

# Display the results
display(df_2023_info_new[['AVG_4G', 'Group']])
```

	AVG_4G	Group
0	93.60	Group 3
1	65.64	Group 1
2	87.41	Group 3
3	68.39	Group 1
4	87.89	Group 3
...	...	...
369	71.34	Group 1
370	83.28	Group 2
371	83.07	Group 2
372	78.90	Group 2
373	92.40	Group 3

374 rows × 2 columns

In [45]:
```
dfNew2023 = df_2023_info_new[['laua','laua_name','AVG_4G', 'Group']]
dfNew2023
```

Out[45]:

	laua	laua_name	AVG_4G	Group
0	S12000033	Aberdeen City	93.60	Group 3
1	S12000034	Aberdeenshire	65.64	Group 1
2	E07000223	Adur	87.41	Group 3
3	E07000026	Allerdale	68.39	Group 1
4	E07000032	Amber Valley	87.89	Group 3
...	...	...	...	...
369	W06000006	Wrexham	71.34	Group 1
370	E07000238	Wychavon	83.28	Group 2
371	E07000128	Wyre	83.07	Group 2
372	E07000239	Wyre Forest	78.90	Group 2
373	E06000014	York	92.40	Group 3

374 rows × 4 columns

In [46]:
```
dfNew2023['Group'].value_counts()
```

Out[46]:
```
Group
Group 1    94
Group 2    93
Group 3    93
Group 4    93
Name: count, dtype: int64
```

Groups were almost equally segmented.

In [47]:
```
# Investigate null values
dfNew2023['Group'].isnull().sum()
```

Out[47]:
```
1
```

In [48]:
```
# Combine dfNew2023 with geojson_data
# Convert both columns to lowercase before merging
geojson_data['LAD22NM'] = geojson_data['LAD22NM'].str.lower()
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

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

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

	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

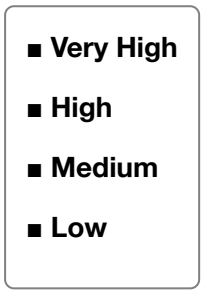
In [49]:
```
# Check unique values and data types in the 'Group' column
print(df_combined_geo['Group'].unique())
print(df_combined_geo['Group'].dtype)

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

In [50]:
```
import folium
import numpy as np

# Define four distinct colors
colors = ['blue', 'green', 'orange', 'red']

# Create a colormap function
def colormap(value):
    if value == 1:
        return 'blue'
    elif value == 2:
        return 'green'
    elif value == 3:
        return 'orange'
    elif value == 4:
        return 'red'
    else:
        return 'gray'  # Default color for NaN or undefined values

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

# Add GeoJson layer with custom colors
folium.GeoJson(
    data="geo_data.json",
    style_function=lambda feature: {
        'fillColor': colormap(feature['properties']['Group_Int']),
        'color': 'black',
        'weight': 1,
        'fillOpacity': 0.7,
    }
).add_to(map_combined)

# Add custom legend
legend_html = '''
<div style="position: fixed; bottom: 50px; left: 50px; z-index: 1000; fon
    <p><span style="color: red; font-weight: bold;">&#9632; Very High</sp
    <p><span style="color: orange; font-weight: bold;">&#9632; High</span
    <p><span style="color: green; font-weight: bold;">&#9632; Medium</spa
    <p><span style="color: blue; font-weight: bold;">&#9632; Low</span></

</div>
'''
```

```
# Add legend to the map
map_combined.get_root().html.add_child(folium.Element(legend_html))

# Display the map
map_combined
```

Out[50]: Make this Notebook Trusted to load map: File -> Trust Notebook



This map shows the 4G performance in the UK divided into four groups: Very High, High, Medium, and Low. It was clear that the capitals of nations and large cities such as London, Edinburgh, Manchester, and Birmingham had very high 4G performance. In contrast, rural areas had very low performance. However, 4G coverage was comparatively well spread (Medium) in southern England and some parts of northern England. High-performance groups are seen surrounding big cities (Very High) as well as in some parts of northeastern England.

Increase /decrease for 5 years

Increases and decreases are investigated at each local authority level to gain insights.

In [51]:
```
merged_nation.head()
```

Out[51]:

	laua	laua_name	2019	2020	2021	2022	2023	LAD22CD	nat
0	E06000001	Hartlepool	73.05	72.89	77.29	78.15	79.62	E06000001	Engla
1	E06000002	Middlesbrough	86.68	86.17	83.39	90.76	92.62	E06000002	Engla
2	E06000005	Darlington	82.35	87.16	88.45	89.07	89.98	E06000005	Engla
3	E06000006	Halton	83.05	87.06	91.18	93.98	93.64	E06000006	Engla
4	E06000007	Warrington	78.32	82.50	83.20	87.36	88.45	E06000007	Engla

In [52]:
```
merged_nation_1 = merged_nation[['2019', '2020', '2021', '2022', '2023']]
merged_nation_1
```

Out[52]:

	2019	2020	2021	2022	2023
0	73.05	72.89	77.29	78.15	79.62
1	86.68	86.17	83.39	90.76	92.62
2	82.35	87.16	88.45	89.07	89.98
3	83.05	87.06	91.18	93.98	93.64
4	78.32	82.50	83.20	87.36	88.45
...	...	...	...	...	...
310	77.01	81.74	79.63	81.19	84.67
311	56.98	69.44	69.86	71.05	73.84
312	81.62	90.38	90.88	91.33	93.72
313	49.84	57.60	58.16	58.62	59.70
314	70.07	74.79	75.30	75.00	75.37

315 rows × 5 columns

In [53]:
```
# Find the column positions of the max and min values
max_col_positions = merged_nation_1.idxmax(axis=1).str.extract(r'(\d+)',
min_col_positions = merged_nation_1.idxmin(axis=1).str.extract(r'(\d+)',

# Calculate the increase based on the condition
merged_nation_1.loc[:, 'increase/decrease'] = np.where(
    max_col_positions < min_col_positions,
    merged_nation_1.min(axis=1) - merged_nation_1.max(axis=1),
    merged_nation_1.max(axis=1) - merged_nation_1.min(axis=1)
)

# Display the DataFrame
merged_nation_1
```

```
/tmp/ipykernel_211/2913136132.py:6: 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
  merged_nation_1.loc[:, 'increase/decrease'] = np.where(
```

Out[53]:

	2019	2020	2021	2022	2023	increase/decrease
0	73.05	72.89	77.29	78.15	79.62	6.73
1	86.68	86.17	83.39	90.76	92.62	9.23
2	82.35	87.16	88.45	89.07	89.98	7.63
3	83.05	87.06	91.18	93.98	93.64	10.93
4	78.32	82.50	83.20	87.36	88.45	10.13
...	...	...	...	...	...	...
310	77.01	81.74	79.63	81.19	84.67	7.66
311	56.98	69.44	69.86	71.05	73.84	16.86
312	81.62	90.38	90.88	91.33	93.72	12.10
313	49.84	57.60	58.16	58.62	59.70	9.86
314	70.07	74.79	75.30	75.00	75.37	5.30

315 rows × 6 columns

In [54]:
```
# Create a boolean Series where 'increase' is less than 0
merged_nation_2 = merged_nation_1['increase/decrease'] < 0

# Count the number of True values in the boolean Series
negative_increase_count = merged_nation_2.value_counts()

# Display the count of negative increases
negative_increase_count
```

Out[54]:
```
increase/decrease
False    303
True      12
Name: count, dtype: int64
```

Each local authority experienced an increase or decrease from 2019 to 2023. Therefore, a bar graph showing the highest percentage increase /decrease by 2023 has been created.

In [55]:
```
# Add the 'increase' column to the original merged_nation DataFrame
merged_nation['increase/decrease'] = merged_nation_1['increase/decrease']

# Display the updated DataFrame
merged_nation
```

Out[55]:

	laua	laua_name	2019	2020	2021	2022	2023	LAD22CD	
0	E06000001	Hartlepool	73.05	72.89	77.29	78.15	79.62	E06000001	
1	E06000002	Middlesbrough	86.68	86.17	83.39	90.76	92.62	E06000002	
2	E06000005	Darlington	82.35	87.16	88.45	89.07	89.98	E06000005	
3	E06000006	Halton	83.05	87.06	91.18	93.98	93.64	E06000006	
4	E06000007	Warrington	78.32	82.50	83.20	87.36	88.45	E06000007	
...	...	...	...	...	...	...	...	...	
310	W06000020	Torfaen	77.01	81.74	79.63	81.19	84.67	W06000020	
311	W06000021	Monmouthshire	56.98	69.44	69.86	71.05	73.84	W06000021	
312	W06000022	Newport	81.62	90.38	90.88	91.33	93.72	W06000022	
313	W06000023	Powys	49.84	57.60	58.16	58.62	59.70	W06000023	
314	W06000024	Merthyr Tydfil	70.07	74.79	75.30	75.00	75.37	W06000024	

315 rows × 10 columns

In [56]:
```
# Sort the DataFrame by the 'increase/decrease' column in ascending order
merged_nation_sorted = merged_nation.sort_values(by='increase/decrease')

# Display the sorted DataFrame
merged_nation_sorted
```

Out[56]:

	laua	laua_name	2019	2020	2021	2022	2023	LAD22CD	nat
278	S12000023	Orkney Islands	30.68	36.46	36.52	33.20	26.35	S12000023	Scotla
250	E09000015	Harrow	97.57	87.43	88.26	88.95	89.68	E09000015	Engla
294	S12000050	North Lanarkshire	92.25	82.14	84.48	86.61	88.26	S12000050	Scotla
149	E07000169	Selby	78.90	84.56	80.51	78.18	81.84	E07000169	Engla
41	E07000010	Fenland	71.96	75.82	76.24	71.30	72.21	E07000010	Engla
...	...	...	...	...	...	...	...	...	
162	E07000189	South Somerset	61.49	73.50	71.14	77.91	80.35	E07000189	Engla
96	E07000094	Winchester	66.80	77.98	82.00	83.50	85.82	E07000094	Engla
274	S12000018	Inverclyde	68.48	77.42	79.06	86.75	88.36	S12000018	Scotla
293	S12000049	Glasgow City	77.65	93.21	93.64	96.97	97.67	S12000049	Scotla
45	E07000028	Carlisle	53.66	72.65	75.23	76.83	77.71	E07000028	Engla

315 rows × 10 columns

In [57]:
```
merged_nation_sorted.head()
```

Out[57]:

	laua	laua_name	2019	2020	2021	2022	2023	LAD22CD	nati
278	S12000023	Orkney Islands	30.68	36.46	36.52	33.20	26.35	S12000023	Scotla
250	E09000015	Harrow	97.57	87.43	88.26	88.95	89.68	E09000015	Engla
294	S12000050	North Lanarkshire	92.25	82.14	84.48	86.61	88.26	S12000050	Scotla
149	E07000169	Selby	78.90	84.56	80.51	78.18	81.84	E07000169	Engla
41	E07000010	Fenland	71.96	75.82	76.24	71.30	72.21	E07000010	Engla

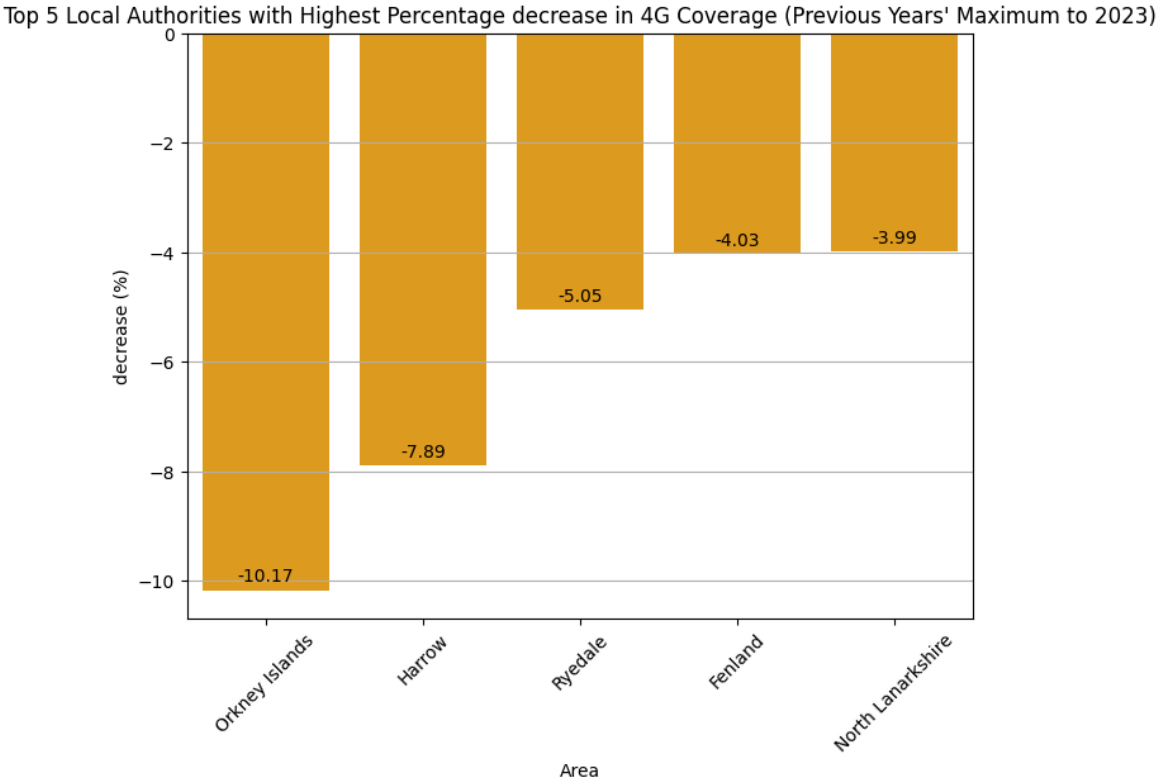
It is ensured that some areas did not see an increase but a decrease over the past five years. To compare the decrease in 4G coverage from previous years to the most recent year (2023), a DataFrame is created showing the top 5 local authorities with the highest percentage decrease in 4G coverage from their maximum value in the previous years to the value in 2023.

In [58]:
```
# Extract relevant columns
years = ['2019', '2020', '2021', '2022']

# Find max values and their corresponding years
merged_nation_sorted['max_value'] = merged_nation_sorted[years].max(axis=
max_year_dict = merged_nation_sorted[years].idxmax(axis=1).to_dict()
merged_nation_sorted['max_year'] = merged_nation_sorted.index.map(max_yea
merged_nation_sorted['2023_value'] = merged_nation_sorted['2023']

# Calculate percentage decrease from max to 2023
merged_nation_sorted['percentage_decrease'] = merged_nation_sorted['2023_

# Select top 5 decreases
top5_decreases = merged_nation_sorted.nsmallest(5, 'percentage_decrease')

# Select and reorder columns for the final DataFrame
result_merged_nation_sorted = top5_decreases[['laua_name','nation', 'max_

display(result_merged_nation_sorted)
```

	laua_name	nation	max_value	max_year	2023_value	percentage_decrease
278	Orkney Islands	Scotland	36.52	2021	26.35	-10.17
250	Harrow	England	97.57	2019	89.68	-7.89
147	Ryedale	England	71.62	2021	66.57	-5.05
41	Fenland	England	76.24	2021	72.21	-4.03
294	North Lanarkshire	Scotland	92.25	2019	88.26	-3.99

In [59]:
```
# Defining the plot size
plt.figure(figsize=(8, 6))


plots = sns.barplot(x="laua_name", y="percentage_decrease",  data=result_
```

```
# Iterating over the bars one-by-one
for bar in plots.patches:

  plots.annotate(format(bar.get_height(), '.2f'),
                 (bar.get_x() + bar.get_width() / 2,
                  bar.get_height()), ha='center', va='center',
                 size=10, xytext=(0, 8),
                 textcoords='offset points')



plt.title("Top 5 Local Authorities with Highest Percentage decrease in 4G
plt.xlabel('Area')
plt.ylabel('decrease (%)')
plt.xticks(rotation=45)
plt.grid(axis='y')



# Finally showing the plot
plt.show()
```



Comment:

Orkney Islands saw the most significant decrease, with a maximum 4G coverage of 36.52% in 2021, dropping to 26.35% in 2023, resulting in a percentage decrease of -10.17%. This indicates a substantial reduction in 4G coverage over the past two years. Three areas (Hoarrow, Ryedale, and Fenland) from England were ranked in the Top 5.

Next, to compare the increase in 4G coverage from previous years to the most recent year (2023) again, a DataFrame is created showing the top 5 local authorities with the highest percentage increase in 4G coverage from their minimum value in the previous years to the value in 2023.

In [60]:
```
# Find min values and their corresponding years
merged_nation_sorted['min_value'] = merged_nation_sorted[years].min(axis=
min_year_dict = merged_nation_sorted[years].idxmin(axis=1).to_dict()
merged_nation_sorted['min_year'] = merged_nation_sorted.index.map(min_yea

# Calculate percentage decrease from max to 2023
merged_nation_sorted['percentage_increase'] = merged_nation_sorted['2023_

# Select top 5 decreases
top5_increases = merged_nation_sorted.nlargest(5, 'percentage_increase')

# Select and reorder columns for the final DataFrame
result_merged_nation_sorted_2 = top5_increases[['laua_name','nation', 'mi

display(result_merged_nation_sorted_2)
```

	laua_name	nation	min_value	min_year	2023_value	percentage_increase
45	Carlisle	England	53.66	2019	77.71	24.05
293	Glasgow City	Scotland	77.65	2019	97.67	20.02
274	Inverclyde	Scotland	68.48	2019	88.36	19.88
96	Winchester	England	66.80	2019	85.82	19.02
162	South Somerset	England	61.49	2019	80.35	18.86

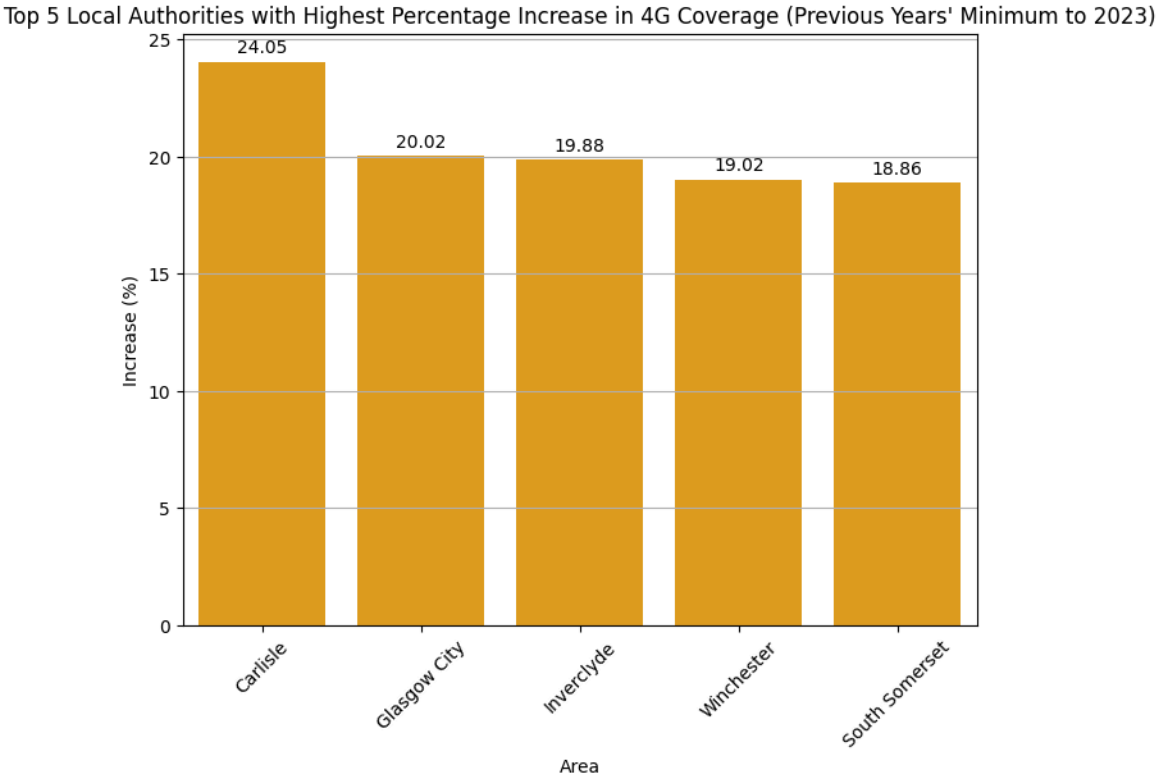
In [61]:
```
# create bar plot of the top 5 increase

top5 = result_merged_nation_sorted_2
top5
```

Out[61]:

	laua_name	nation	min_value	min_year	2023_value	percentage_increase
45	Carlisle	England	53.66	2019	77.71	24.05
293	Glasgow City	Scotland	77.65	2019	97.67	20.02
274	Inverclyde	Scotland	68.48	2019	88.36	19.88
96	Winchester	England	66.80	2019	85.82	19.02
162	South Somerset	England	61.49	2019	80.35	18.86

In [62]:
```
# Defining the plot size
plt.figure(figsize=(8, 6))


plots = sns.barplot(x="laua_name", y="percentage_increase",  data=top5, c
```

```
# Iterating over the bars one-by-one
for bar in plots.patches:

  plots.annotate(format(bar.get_height(), '.2f'),
                 (bar.get_x() + bar.get_width() / 2,
                  bar.get_height()), ha='center', va='center',
                 size=10, xytext=(0, 8),
                 textcoords='offset points')



plt.title("Top 5 Local Authorities with Highest Percentage Increase in 4G
plt.xlabel('Area')
plt.ylabel('Increase (%)')
plt.xticks(rotation=45)
plt.grid(axis='y')



# Finally showing the plot
plt.show()
```



Carlisle experienced a significant increase in 4G coverage, with the coverage percentage rising from 53.66% in 2022 to 77.71% in 2023. This represents a percentage increase of 24.05%, the highest among the listed local authorities. Notably, the relatively large city of Glasgow also had a high increase with coverage rising from 77.65% in 2022 to 97.67% in 2023, resulting in a percentage increase of 20.02%.

In [64]:
```
### Save the data
```

```
df_2023_info.to_csv('data/4G_2023.csv', index=False)
grouped_nation.to_csv('data/4G_nation.csv', index=False)
```

In [ ]:

In [ ]: