```
import os
import zipfile
import fiona
import geopandas as gpd
import pandas as pd
from shapely.ops import nearest_points
```

```
# -------------------------------------
# 1. Load & Clean Tabular CSV Features
# -------------------------------------

# Adjust these paths if your filenames differ - using raw GitHub URLs
hv_path     = 'https://raw.githubusercontent.com/IflyNY2PR/CASA0004/071702afad8b880e82c9ed335
ptal_path   = 'https://raw.githubusercontent.com/IflyNY2PR/CASA0004/071702afad8b880e82c9ed335
ls_path     = 'https://raw.githubusercontent.com/IflyNY2PR/CASA0004/071702afad8b880e82c9ed335
sent_path   = 'https://raw.githubusercontent.com/IflyNY2PR/CASA0004/071702afad8b880e82c9ed335

# Read CSVs (use latin-1 if you hit encoding errors)
hv   = pd.read_csv(hv_path, encoding='latin-1', low_memory=False)
ptal = pd.read_csv(ptal_path, encoding='latin-1', low_memory=False)
ls   = pd.read_csv(ls_path, encoding='latin-1', low_memory=False)
sent = pd.read_csv(sent_path, encoding='latin-1', low_memory=False)

# --- Housing Value ---
# Use LSOA code column (not the first column which is Local authority code)
hv_code   = 'LSOA code'  # Use the correct LSOA code column
# Find the column with 'Mar' and '2023' - use a safer approach
mar_2023_cols = [c for c in hv.columns if 'Mar' in c and '2023' in c]
if mar_2023_cols:
    hv_price = mar_2023_cols[0]
else:
    # Fallback to the last column if no Mar 2023 found
    hv_price = hv.columns[-1]
hv_sales  = next((c for c in hv.columns if 'sale' in c.lower()), None)
hv_cols   = [hv_code, hv_price] + ([hv_sales] if hv_sales else [])
hv_tab    = hv[hv_cols].copy()
hv_tab.columns = ['LSOA_CODE', 'AvgPrice'] + (['NumSales'] if hv_sales else [])

# Convert price from string to numeric (remove commas and convert to float)
hv_tab['AvgPrice'] = hv_tab['AvgPrice'].astype(str).str.replace(',', '').replace('', None)
hv_tab['AvgPrice'] = pd.to_numeric(hv_tab['AvgPrice'], errors='coerce')
```

```
# --- PTAL Stats ---
# Handle BOM character in column name
ptal_code_col = ptal.columns[0]  # This will be 'ï»¿LSOA21CD'
ptal_tab = ptal[[ptal_code_col,'MEAN_PTAL_2023','MAX_AI','MIN_AI']].rename(
    columns={ptal_code_col:'LSOA_CODE'}
)

# --- Demographics & Area ---
ls_code  = ls.columns[0]
# Find population column - try 2023 first, then any population column
ls_pop_candidates = [c for c in ls.columns if 'Population' in c and '2023' in c]
if not ls_pop_candidates:
    ls_pop_candidates = [c for c in ls.columns if 'Population' in c]
ls_pop = ls_pop_candidates[0] if ls_pop_candidates else None

# Find IMD column
ls_imd_candidates = [c for c in ls.columns if 'IMD' in c.upper()]
ls_imd = ls_imd_candidates[0] if ls_imd_candidates else None

# Find area column
ls_area_candidates = [c for c in ls.columns if 'AREA' in c.upper()]
ls_area = ls_area_candidates[0] if ls_area_candidates else None

# Build the columns list, only including found columns
ls_cols = [ls_code]
ls_col_names = ['LSOA_CODE']
if ls_pop:
    ls_cols.append(ls_pop)
    ls_col_names.append('Population')
if ls_imd:
    ls_cols.append(ls_imd)
    ls_col_names.append('IMD_Decile')
if ls_area:
    ls_cols.append(ls_area)
    ls_col_names.append('Area_km2')

ls_tab = ls[ls_cols].copy()
ls_tab.columns = ls_col_names

# --- Sentiment Stats ---
sent_tab = sent[['LSOA','Avg_Sentiment_Score','Sentiment_Std','Total_Reviews']].rename(
    columns={
```

```
        'LSOA':'LSOA_CODE',
        'Avg_Sentiment_Score':'MeanSentiment',
        'Sentiment_Std':'SentimentSD',
        'Total_Reviews':'ReviewCount'
    }
)

# Merge all tabular data
df_tab = (
    hv_tab
    .merge(ptal_tab, on='LSOA_CODE', how='outer')
    .merge(ls_tab,   on='LSOA_CODE', how='outer')
    .merge(sent_tab, on='LSOA_CODE', how='outer')
)
```

```
# ---------------------------------
# 2. Download & Unzip Shapefiles Layers
# ---------------------------------
import requests

zip_paths = {
    'lsoa':       'https://github.com/IflyNY2PR/CASA0004/raw/071702afad8b880e82c9ed33500e52ba
    'street':     'https://github.com/IflyNY2PR/CASA0004/raw/071702afad8b880e82c9ed33500e52ba
    'station':    'https://github.com/IflyNY2PR/CASA0004/raw/071702afad8b880e82c9ed33500e52ba
    'landuse':    'https://github.com/IflyNY2PR/CASA0004/raw/071702afad8b880e82c9ed33500e52ba
    'rail':       'https://github.com/IflyNY2PR/CASA0004/raw/071702afad8b880e82c9ed33500e52ba
}

# Download and extract each zip file
for name, zip_url in zip_paths.items():
    outdir = f'./{name}'
    zip_path = f'./{name}.zip'

    if not os.path.isdir(outdir):
        os.makedirs(outdir, exist_ok=True)

        # Download the zip file
        print(f"Downloading {name}...")
        response = requests.get(zip_url)
        response.raise_for_status()

        # Save the zip file
```

```
        with open(zip_path, 'wb') as f:
            f.write(response.content)

        # Extract the zip file
        print(f"Extracting {name}...")
        with zipfile.ZipFile(zip_path, 'r') as z:
            z.extractall(outdir)

        # Clean up the zip file
        os.remove(zip_path)

def find_shp(dirpath):
    for root, _, files in os.walk(dirpath):
        for f in files:
            if f.lower().endswith('.shp') and not f.startswith('._'):
                return os.path.join(root, f)
    raise FileNotFoundError(f"No .shp in {dirpath}")

def load_gdf(shp_path):
    with fiona.open(shp_path) as src:
        feats = list(src)
        return gpd.GeoDataFrame.from_features(feats, crs=src.crs)

# Load & reproject layers
lsoa_gdf    = load_gdf(find_shp('./lsoa')).to_crs('EPSG:27700')
street_gdf  = load_gdf(find_shp('./street')).to_crs('EPSG:27700')
station_gdf = load_gdf(find_shp('./station')).to_crs('EPSG:27700')
landuse_gdf = load_gdf(find_shp('./landuse')).to_crs('EPSG:27700')
rail_gdf    = load_gdf(find_shp('./rail')).to_crs('EPSG:27700')

# Determine the code column in LSOA shapefile
# Look for columns that might contain LSOA codes
possible_code_cols = [c for c in lsoa_gdf.columns if 'LSOA' in c.upper() and lsoa_gdf[c].dtyp
if possible_code_cols:
    shp_code = possible_code_cols[0]
else:
    # Fallback to 'code' column if no LSOA-named column found
    shp_code = 'code'

print(f"Using column '{shp_code}' as LSOA code column")
print(f"Sample values: {lsoa_gdf[shp_code].head().tolist()}")
```

```
Using column 'code' as LSOA code column
Sample values: ['E01000037', 'E01033729', 'E01000038', 'E01033730', 'E01000039']
```

```python
# --------------------------------
# 3. Comprehensive Input Data Check
# --------------------------------

# Dictionary of all loaded datasets for easy iteration
datasets = {
    "Merged Tabular Data (df_tab)": df_tab,
    "LSOA Polygons (lsoa_gdf)": lsoa_gdf,
    "Street Network (street_gdf)": street_gdf,
    "Stations (station_gdf)": station_gdf,
    "Land Use (landuse_gdf)": landuse_gdf,
    "Rail Network (rail_gdf)": rail_gdf
}

# Loop through and print a summary of each dataset
for name, data in datasets.items():
    print(f"--- Checking: {name} ---\n")

    if isinstance(data, pd.DataFrame):
        print(f"Shape: {data.shape}")
        print(f"Columns: {data.columns.tolist()}")

        # Check for missing values
        missing_values = data.isnull().sum().sum()
        print(f"Total Missing Values: {missing_values}")

        # Display info for GeoDataFrames
        if isinstance(data, gpd.GeoDataFrame):
            print(f"CRS: {data.crs}")
            print(f"Geometry Type: {data.geom_type.unique()}")

        print("\nHead:")
        print(data.head())

    else:
        print(f"'{name}' is not a DataFrame or GeoDataFrame.")

    print("\n" + "="*50 + "\n")
```

```
--- Checking: Merged Tabular Data (df_tab) ---

Shape: (35090, 10)
Columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Population', 'Area
Total Missing Values: 250189

Head:
   LSOA_CODE  AvgPrice MEAN_PTAL_2023      MAX_AI     MIN_AI  Population  \
0  E01000001  837500.0             6b   97.664050  43.344686      1615.0
1  E01000002  850000.0             6b  102.127724  44.802674      1493.0
2  E01000003  540000.0             6b   66.989479  36.995870      1573.0
3  E01000005       NaN             6b  104.154096  45.504887      1090.0
4  E01000006  241000.0              5   33.400903   2.836977      1612.0

    Area_km2  MeanSentiment  SentimentSD  ReviewCount
0  E01000001          0.427        0.312       1505.0
1  E01000002          0.524        0.188       5819.0
2  E01000003          0.317          NaN          3.0
3  E01000005          0.675        0.100       1067.0
4  E01000006            NaN          NaN          NaN

================================================

--- Checking: LSOA Polygons (lsoa_gdf) ---

Shape: (4719, 7)
Columns: ['geometry', 'code', 'name', 'label', 'Area', 'Half densi', 'Area2']
Total Missing Values: 0
CRS: EPSG:27700
Geometry Type: ['Polygon' 'MultiPolygon']

Head:
                                              geometry       code  \
0  POLYGON ((551549.998 187364.637, 551528.633 18...  E01000037
1  POLYGON ((544812 174524, 544819.775 174523.378...  E01033729
2  POLYGON ((550920.362 187341.138, 550921.876 18...  E01000038
3  POLYGON ((537938 177696, 537941.714 177678.043...  E01033730
4  POLYGON ((551431.061 186927.155, 551444.481 18...  E01000039

                        name                          label     Area  \
0  Barking and Dagenham 003B  E09000002E02000004E01000037   233488
1              Greenwich 030E  E09000011E02000342E01033729   691074
2  Barking and Dagenham 003C  E09000002E02000004E01000038   214094
```

```
3             Greenwich 035D  E09000011E02006928E01033730   187153
4  Barking and Dagenham 003D  E09000002E02000004E01000039  1532166

   Half densi    Area2
0        3764   233488
1        2034   691074
2        3519   214094
3        4950   187153
4         530  1532166

================================================

--- Checking: Street Network (street_gdf) ---

Shape: (115305, 2)
Columns: ['geometry', 'FID']
Total Missing Values: 0
CRS: EPSG:27700
Geometry Type: ['LineString']

Head:
                                      geometry  FID
0  LINESTRING (533496 194642, 533521 194711)    0
1  LINESTRING (532979 195192, 532937 194981)    1
2  LINESTRING (532451 194748, 532358 194770)    2
3  LINESTRING (532358 194770, 532225 194797)    3
4  LINESTRING (532019 194636, 532123 194669)    4

================================================

--- Checking: Stations (station_gdf) ---

Shape: (21002, 5)
Columns: ['geometry', 'osm_id', 'code', 'fclass', 'name']
Total Missing Values: 218
CRS: EPSG:27700
Geometry Type: ['Point']

Head:
                        geometry  osm_id  code           fclass  \
0  POINT (526506.781 196024.294)  659860  5601  railway_station
1  POINT (523318.835 178683.784)  702503  5601  railway_station
2   POINT (523190.202 180031.21)  780856  5601  railway_station
```

```
3  POINT (523344.763 180490.543)  780911  5601  railway_station
4   POINT (537270.73 188970.803)  818704  5601  railway_station

                    name
0             New Barnet
1            Hammersmith
2  Shepherd's Bush Market
3              Wood Lane
4     Walthamstow Central

================================================

--- Checking: Land Use (landuse_gdf) ---

Shape: (30775, 5)
Columns: ['geometry', 'osm_id', 'code', 'fclass', 'name']
Total Missing Values: 26239
CRS: EPSG:27700
Geometry Type: ['Polygon' 'MultiPolygon']

Head:
                                            geometry   osm_id  code fclass  \
0  POLYGON ((532065.726 197873.867, 532093.379 19...  2838058  7202   park
1  POLYGON ((532863.867 198097.732, 532874.832 19...  2903046  7202   park
2  POLYGON ((532433.383 197662.081, 532438.091 19...  2903368  7202   park
3  POLYGON ((532711.413 197548.732, 532736.071 19...  2903369  7202   park
4  POLYGON ((533583.521 196021.414, 533586.278 19...  2903398  7202   park

            name
0           None
1           None
2           None
3           None
4  Bush Hill Park

================================================

--- Checking: Rail Network (rail_gdf) ---

Shape: (11777, 5)
Columns: ['geometry', 'osm_id', 'code', 'fclass', 'name']
Total Missing Values: 5421
CRS: EPSG:27700
```

```
Geometry Type: ['LineString']

Head:
                                              geometry  osm_id  code  fclass  \
0  LINESTRING (538865.67 194026.578, 538953.872 1...   30804  6101    rail
1  LINESTRING (523318.835 178683.784, 523314.286 ...  101298  6103  subway
2  LINESTRING (524598.481 181595.074, 524626.148 ...  101486  6103  subway
3  LINESTRING (524694.206 181886.633, 524655.584 ...  101511  6101    rail
4  LINESTRING (525670.998 192344.253, 525634.231 ...  282898  6103  subway

                              name
0                  Lea Valley Lines
1           Hammersmith and City Line
2           Hammersmith and City Line
3             Great Western Main Line
4  Northern Line (High Barnet Branch)

================================================
```

```
print(hv)
```

```
      ï»¿Local authority code Local authority name  LSOA code  \
0                   E06000001           Hartlepool  E01011949
1                   E06000001           Hartlepool  E01011950
2                   E06000001           Hartlepool  E01011951
3                   E06000001           Hartlepool  E01011952
4                   E06000001           Hartlepool  E01011953
...                       ...                  ...        ...
34748               W06000024       Merthyr Tydfil  W01001320
34749               W06000024       Merthyr Tydfil  W01001321
34750               W06000024       Merthyr Tydfil  W01001322
34751               W06000024       Merthyr Tydfil  W01001324
34752               W06000024       Merthyr Tydfil  W01001898

                LSOA name Year ending Dec 1995 Year ending Mar 1996  \
0          Hartlepool 009A               34,750               34,500
1          Hartlepool 008A               25,000               25,000
2          Hartlepool 007A               27,000               27,000
3          Hartlepool 002A               44,500               44,500
4          Hartlepool 002B               22,000               27,000
...                    ...                  ...                  ...
34748  Merthyr Tydfil 007C               29,000               35,000
```

```
34749  Merthyr Tydfil 007D              23,500              24,000
34750  Merthyr Tydfil 007E              20,000              27,248
34751  Merthyr Tydfil 003E              34,748              34,998
34752  Merthyr Tydfil 008F              43,750              54,375

      Year ending Jun 1996 Year ending Sep 1996 Year ending Dec 1996  \
0                   30,500              30,000              29,950
1                   25,300              25,625              25,000
2                   27,250              28,950              28,500
3                   30,000              26,675              26,000
4                   27,000              20,600              20,000
...                    ...                 ...                 ...
34748               35,000              44,250              44,250
34749               26,000              30,000              29,475
34750               28,498              25,500              29,725
34751               32,500              33,000              31,000
34752               65,000              68,725              62,500

      Year ending Mar 1997  ... Year ending Dec 2020 Year ending Mar 2021  \
0                   29,000  ...               88,000               81,500
1                   24,800  ...               29,750               33,000
2                   28,950  ...               50,000               51,500
3                   25,500  ...               85,000                    :
4                   19,500  ...                    :                    :
...                    ...  ...                  ...                  ...
34748               43,500  ...              139,500              144,000
34749               22,725  ...               95,000              100,750
34750               28,750  ...               91,000              100,000
34751               29,000  ...              142,000              150,000
34752               59,294  ...                    :                    :

      Year ending Jun 2021 Year ending Sep 2021 Year ending Dec 2021  \
0                   80,500              89,000             101,500
1                   47,000              49,999              50,159
2                   53,000              58,574              60,000
3                   83,500              83,000              80,000
4                        :              95,000              92,500
...                    ...                 ...                 ...
34748              141,500             140,000             140,000
34749              107,000             110,000             109,500
34750              105,000             105,000             107,500
34751              130,500             146,000             137,500
34752                    :             142,500             165,000
```

```
      Year ending Mar 2022 Year ending Jun 2022 Year ending Sep 2022  \
0                  94,500              113,000               97,500
1                  50,159               46,000               43,500
2                  62,999               61,500               60,000
3                  76,000               75,000               75,000
4                  95,000               95,000               92,500
...                   ...                  ...                  ...
34748             142,500              141,975              149,500
34749             108,250              110,000              109,500
34750             108,500              113,500              119,000
34751             135,000              137,500              140,000
34752             180,000              182,500              182,500

      Year ending Dec 2022 Year ending Mar 2023
0                  102,500              106,500
1                   42,000               43,500
2                   65,500               66,000
3                   70,000               60,000
4                   93,750               92,500
...                    ...                  ...
34748              155,000              149,500
34749              113,500              120,000
34750              135,000              136,000
34751              140,000              145,000
34752              185,000              211,000

[34753 rows x 114 columns]
```

```python
# Let's examine the housing value dataset more carefully
print("=== HOUSING VALUE DATASET DIAGNOSIS ===")
print(f"Shape: {hv.shape}")
print(f"Columns: {list(hv.columns)}")
print("\nFirst few rows:")
print(hv.head())
print("\nData types:")
print(hv.dtypes)
print("\nColumn names with potential encoding issues:")
for i, col in enumerate(hv.columns):
    print(f"Column {i}: '{col}' (repr: {repr(col)})")
```

```
=== HOUSING VALUE DATASET DIAGNOSIS ===
```

```
Shape: (34753, 114)
Columns: ['ï»¿Local authority code', 'Local authority name', 'LSOA code', 'LSOA name', 'Year

First few rows:
  ï»¿Local authority code Local authority name  LSOA code        LSOA name  \
0               E06000001           Hartlepool  E01011949  Hartlepool 009A
1               E06000001           Hartlepool  E01011950  Hartlepool 008A
2               E06000001           Hartlepool  E01011951  Hartlepool 007A
3               E06000001           Hartlepool  E01011952  Hartlepool 002A
4               E06000001           Hartlepool  E01011953  Hartlepool 002B

  Year ending Dec 1995 Year ending Mar 1996 Year ending Jun 1996  \
0              34,750               34,500               30,500
1              25,000               25,000               25,300
2              27,000               27,000               27,250
3              44,500               44,500               30,000
4              22,000               27,000               27,000

  Year ending Sep 1996 Year ending Dec 1996 Year ending Mar 1997  ...  \
0              30,000               29,950               29,000  ...
1              25,625               25,000               24,800  ...
2              28,950               28,500               28,950  ...
3              26,675               26,000               25,500  ...
4              20,600               20,000               19,500  ...

  Year ending Dec 2020 Year ending Mar 2021 Year ending Jun 2021  \
0              88,000               81,500               80,500
1              29,750               33,000               47,000
2              50,000               51,500               53,000
3              85,000                    :               83,500
4                   :                    :                    :

  Year ending Sep 2021 Year ending Dec 2021 Year ending Mar 2022  \
0              89,000              101,500               94,500
1              49,999               50,159               50,159
2              58,574               60,000               62,999
3              83,000               80,000               76,000
4              95,000               92,500               95,000

  Year ending Jun 2022 Year ending Sep 2022 Year ending Dec 2022  \
0             113,000               97,500              102,500
1              46,000               43,500               42,000
2              61,500               60,000               65,500
```

```
3               75,000               75,000               70,000
4               95,000               92,500               93,750

  Year ending Mar 2023
0              106,500
1               43,500
2               66,000
3               60,000
4               92,500

[5 rows x 114 columns]

Data types:
ï»¿Local authority code    object
Local authority name       object
LSOA code                  object
LSOA name                  object
Year ending Dec 1995       object
                            ...
Year ending Mar 2022       object
Year ending Jun 2022       object
Year ending Sep 2022       object
Year ending Dec 2022       object
Year ending Mar 2023       object
Length: 114, dtype: object

Column names with potential encoding issues:
Column 0: 'ï»¿Local authority code' (repr: 'ï»¿Local authority code')
Column 1: 'Local authority name' (repr: 'Local authority name')
Column 2: 'LSOA code' (repr: 'LSOA code')
Column 3: 'LSOA name' (repr: 'LSOA name')
Column 4: 'Year ending Dec 1995' (repr: 'Year ending Dec 1995')
Column 5: 'Year ending Mar 1996' (repr: 'Year ending Mar 1996')
Column 6: 'Year ending Jun 1996' (repr: 'Year ending Jun 1996')
Column 7: 'Year ending Sep 1996' (repr: 'Year ending Sep 1996')
Column 8: 'Year ending Dec 1996' (repr: 'Year ending Dec 1996')
Column 9: 'Year ending Mar 1997' (repr: 'Year ending Mar 1997')
Column 10: 'Year ending Jun 1997' (repr: 'Year ending Jun 1997')
Column 11: 'Year ending Sep 1997' (repr: 'Year ending Sep 1997')
Column 12: 'Year ending Dec 1997' (repr: 'Year ending Dec 1997')
Column 13: 'Year ending Mar 1998' (repr: 'Year ending Mar 1998')
Column 14: 'Year ending Jun 1998' (repr: 'Year ending Jun 1998')
Column 15: 'Year ending Sep 1998' (repr: 'Year ending Sep 1998')
```

```
Column 16: 'Year ending Dec 1998' (repr: 'Year ending Dec 1998')
Column 17: 'Year ending Mar 1999' (repr: 'Year ending Mar 1999')
Column 18: 'Year ending Jun 1999' (repr: 'Year ending Jun 1999')
Column 19: 'Year ending Sep 1999' (repr: 'Year ending Sep 1999')
Column 20: 'Year ending Dec 1999' (repr: 'Year ending Dec 1999')
Column 21: 'Year ending Mar 2000' (repr: 'Year ending Mar 2000')
Column 22: 'Year ending Jun 2000' (repr: 'Year ending Jun 2000')
Column 23: 'Year ending Sep 2000' (repr: 'Year ending Sep 2000')
Column 24: 'Year ending Dec 2000' (repr: 'Year ending Dec 2000')
Column 25: 'Year ending Mar 2001' (repr: 'Year ending Mar 2001')
Column 26: 'Year ending Jun 2001' (repr: 'Year ending Jun 2001')
Column 27: 'Year ending Sep 2001' (repr: 'Year ending Sep 2001')
Column 28: 'Year ending Dec 2001' (repr: 'Year ending Dec 2001')
Column 29: 'Year ending Mar 2002' (repr: 'Year ending Mar 2002')
Column 30: 'Year ending Jun 2002' (repr: 'Year ending Jun 2002')
Column 31: 'Year ending Sep 2002' (repr: 'Year ending Sep 2002')
Column 32: 'Year ending Dec 2002' (repr: 'Year ending Dec 2002')
Column 33: 'Year ending Mar 2003' (repr: 'Year ending Mar 2003')
Column 34: 'Year ending Jun 2003' (repr: 'Year ending Jun 2003')
Column 35: 'Year ending Sep 2003' (repr: 'Year ending Sep 2003')
Column 36: 'Year ending Dec 2003' (repr: 'Year ending Dec 2003')
Column 37: 'Year ending Mar 2004' (repr: 'Year ending Mar 2004')
Column 38: 'Year ending Jun 2004' (repr: 'Year ending Jun 2004')
Column 39: 'Year ending Sep 2004' (repr: 'Year ending Sep 2004')
Column 40: 'Year ending Dec 2004' (repr: 'Year ending Dec 2004')
Column 41: 'Year ending Mar 2005' (repr: 'Year ending Mar 2005')
Column 42: 'Year ending Jun 2005' (repr: 'Year ending Jun 2005')
Column 43: 'Year ending Sep 2005' (repr: 'Year ending Sep 2005')
Column 44: 'Year ending Dec 2005' (repr: 'Year ending Dec 2005')
Column 45: 'Year ending Mar 2006' (repr: 'Year ending Mar 2006')
Column 46: 'Year ending Jun 2006' (repr: 'Year ending Jun 2006')
Column 47: 'Year ending Sep 2006' (repr: 'Year ending Sep 2006')
Column 48: 'Year ending Dec 2006' (repr: 'Year ending Dec 2006')
Column 49: 'Year ending Mar 2007' (repr: 'Year ending Mar 2007')
Column 50: 'Year ending Jun 2007' (repr: 'Year ending Jun 2007')
Column 51: 'Year ending Sep 2007' (repr: 'Year ending Sep 2007')
Column 52: 'Year ending Dec 2007' (repr: 'Year ending Dec 2007')
Column 53: 'Year ending Mar 2008' (repr: 'Year ending Mar 2008')
Column 54: 'Year ending Jun 2008' (repr: 'Year ending Jun 2008')
Column 55: 'Year ending Sep 2008' (repr: 'Year ending Sep 2008')
Column 56: 'Year ending Dec 2008' (repr: 'Year ending Dec 2008')
Column 57: 'Year ending Mar 2009' (repr: 'Year ending Mar 2009')
Column 58: 'Year ending Jun 2009' (repr: 'Year ending Jun 2009')
```

```
Column 59: 'Year ending Sep 2009' (repr: 'Year ending Sep 2009')
Column 60: 'Year ending Dec 2009' (repr: 'Year ending Dec 2009')
Column 61: 'Year ending Mar 2010' (repr: 'Year ending Mar 2010')
Column 62: 'Year ending Jun 2010' (repr: 'Year ending Jun 2010')
Column 63: 'Year ending Sep 2010' (repr: 'Year ending Sep 2010')
Column 64: 'Year ending Dec 2010' (repr: 'Year ending Dec 2010')
Column 65: 'Year ending Mar 2011' (repr: 'Year ending Mar 2011')
Column 66: 'Year ending Jun 2011' (repr: 'Year ending Jun 2011')
Column 67: 'Year ending Sep 2011' (repr: 'Year ending Sep 2011')
Column 68: 'Year ending Dec 2011' (repr: 'Year ending Dec 2011')
Column 69: 'Year ending Mar 2012' (repr: 'Year ending Mar 2012')
Column 70: 'Year ending Jun 2012' (repr: 'Year ending Jun 2012')
Column 71: 'Year ending Sep 2012' (repr: 'Year ending Sep 2012')
Column 72: 'Year ending Dec 2012' (repr: 'Year ending Dec 2012')
Column 73: 'Year ending Mar 2013' (repr: 'Year ending Mar 2013')
Column 74: 'Year ending Jun 2013' (repr: 'Year ending Jun 2013')
Column 75: 'Year ending Sep 2013' (repr: 'Year ending Sep 2013')
Column 76: 'Year ending Dec 2013' (repr: 'Year ending Dec 2013')
Column 77: 'Year ending Mar 2014' (repr: 'Year ending Mar 2014')
Column 78: 'Year ending Jun 2014' (repr: 'Year ending Jun 2014')
Column 79: 'Year ending Sep 2014' (repr: 'Year ending Sep 2014')
Column 80: 'Year ending Dec 2014' (repr: 'Year ending Dec 2014')
Column 81: 'Year ending Mar 2015' (repr: 'Year ending Mar 2015')
Column 82: 'Year ending Jun 2015' (repr: 'Year ending Jun 2015')
Column 83: 'Year ending Sep 2015' (repr: 'Year ending Sep 2015')
Column 84: 'Year ending Dec 2015' (repr: 'Year ending Dec 2015')
Column 85: 'Year ending Mar 2016' (repr: 'Year ending Mar 2016')
Column 86: 'Year ending Jun 2016' (repr: 'Year ending Jun 2016')
Column 87: 'Year ending Sep 2016' (repr: 'Year ending Sep 2016')
Column 88: 'Year ending Dec 2016' (repr: 'Year ending Dec 2016')
Column 89: 'Year ending Mar 2017' (repr: 'Year ending Mar 2017')
Column 90: 'Year ending Jun 2017' (repr: 'Year ending Jun 2017')
Column 91: 'Year ending Sep 2017' (repr: 'Year ending Sep 2017')
Column 92: 'Year ending Dec 2017' (repr: 'Year ending Dec 2017')
Column 93: 'Year ending Mar 2018' (repr: 'Year ending Mar 2018')
Column 94: 'Year ending Jun 2018' (repr: 'Year ending Jun 2018')
Column 95: 'Year ending Sep 2018' (repr: 'Year ending Sep 2018')
Column 96: 'Year ending Dec 2018' (repr: 'Year ending Dec 2018')
Column 97: 'Year ending Mar 2019' (repr: 'Year ending Mar 2019')
Column 98: 'Year ending Jun 2019' (repr: 'Year ending Jun 2019')
Column 99: 'Year ending Sep 2019' (repr: 'Year ending Sep 2019')
Column 100: 'Year ending Dec 2019' (repr: 'Year ending Dec 2019')
Column 101: 'Year ending Mar 2020' (repr: 'Year ending Mar 2020')
```

```
Column 102: 'Year ending Jun 2020' (repr: 'Year ending Jun 2020')
Column 103: 'Year ending Sep 2020' (repr: 'Year ending Sep 2020')
Column 104: 'Year ending Dec 2020' (repr: 'Year ending Dec 2020')
Column 105: 'Year ending Mar 2021' (repr: 'Year ending Mar 2021')
Column 106: 'Year ending Jun 2021' (repr: 'Year ending Jun 2021')
Column 107: 'Year ending Sep 2021' (repr: 'Year ending Sep 2021')
Column 108: 'Year ending Dec 2021' (repr: 'Year ending Dec 2021')
Column 109: 'Year ending Mar 2022' (repr: 'Year ending Mar 2022')
Column 110: 'Year ending Jun 2022' (repr: 'Year ending Jun 2022')
Column 111: 'Year ending Sep 2022' (repr: 'Year ending Sep 2022')
Column 112: 'Year ending Dec 2022' (repr: 'Year ending Dec 2022')
Column 113: 'Year ending Mar 2023' (repr: 'Year ending Mar 2023')
```

```python
# More focused housing value analysis
print("=== HOUSING VALUE FOCUSED ANALYSIS ===")
print(f"Dataset shape: {hv.shape}")
print(f"Number of columns: {len(hv.columns)}")

# Show first 10 column names to check for issues
print("\nFirst 10 columns:")
for i, col in enumerate(hv.columns[:10]):
    print(f"  {i}: {repr(col)}")

# Check if there are issues with column detection
print(f"\nSelected columns for processing:")
print(f"  Code column: {repr(hv_code)}")
print(f"  Price column: {repr(hv_price)}")
print(f"  Sales column: {hv_sales}")

# Check the processed housing value table
print(f"\nProcessed hv_tab shape: {hv_tab.shape}")
print(f"hv_tab columns: {list(hv_tab.columns)}")
print("\nhv_tab sample:")
print(hv_tab.head())

# Check for missing values and data types
print(f"\nhv_tab data types:")
print(hv_tab.dtypes)
print(f"\nMissing values in hv_tab:")
print(hv_tab.isnull().sum())
```

```
=== HOUSING VALUE FOCUSED ANALYSIS ===
```

```
Dataset shape: (34753, 114)
Number of columns: 114

First 10 columns:
  0: 'ï»¿Local authority code'
  1: 'Local authority name'
  2: 'LSOA code'
  3: 'LSOA name'
  4: 'Year ending Dec 1995'
  5: 'Year ending Mar 1996'
  6: 'Year ending Jun 1996'
  7: 'Year ending Sep 1996'
  8: 'Year ending Dec 1996'
  9: 'Year ending Mar 1997'

Selected columns for processing:
  Code column: 'LSOA code'
  Price column: 'Year ending Mar 2023'
  Sales column: None

Processed hv_tab shape: (34753, 2)
hv_tab columns: ['LSOA_CODE', 'AvgPrice']

hv_tab sample:
   LSOA_CODE  AvgPrice
0  E01011949  106500.0
1  E01011950   43500.0
2  E01011951   66000.0
3  E01011952   60000.0
4  E01011953   92500.0

hv_tab data types:
LSOA_CODE     object
AvgPrice     float64
dtype: object

Missing values in hv_tab:
LSOA_CODE      0
AvgPrice     886
dtype: int64
```

```
# Fix the housing value dataset processing
print("=== FIXING HOUSING VALUE DATASET ===")

# Re-process housing value data with corrections
# Use 'LSOA code' column instead of first column
hv_code_correct = 'LSOA code'  # Use the actual LSOA code column

# Find the March 2023 column (already found correctly)
print(f"Using price column: {hv_price}")

# Create corrected housing value table
hv_tab_corrected = hv[[hv_code_correct, hv_price]].copy()
hv_tab_corrected.columns = ['LSOA_CODE', 'AvgPrice']

# Convert price from string to numeric (remove commas and convert to float)
hv_tab_corrected['AvgPrice'] = hv_tab_corrected['AvgPrice'].astype(str).str.replace(',', '')

# Convert to numeric, handling any non-numeric values
hv_tab_corrected['AvgPrice'] = pd.to_numeric(hv_tab_corrected['AvgPrice'], errors='coerce')

print(f"Corrected hv_tab shape: {hv_tab_corrected.shape}")
print(f"Data types after correction:")
print(hv_tab_corrected.dtypes)
print(f"\nSample of corrected data:")
print(hv_tab_corrected.head(10))
print(f"\nMissing values after correction:")
print(hv_tab_corrected.isnull().sum())
print(f"\nPrice statistics:")
print(hv_tab_corrected['AvgPrice'].describe())

# Update the main hv_tab variable
hv_tab = hv_tab_corrected
```

```
=== FIXING HOUSING VALUE DATASET ===
Using price column: Year ending Mar 2023
Corrected hv_tab shape: (34753, 2)
Data types after correction:
LSOA_CODE     object
AvgPrice     float64
dtype: object

Sample of corrected data:
```

```
   LSOA_CODE  AvgPrice
0  E01011949  106500.0
1  E01011950   43500.0
2  E01011951   66000.0
3  E01011952   60000.0
4  E01011953   92500.0
5  E01011954  104000.0
6  E01011955  104750.0
7  E01011957   78000.0
8  E01011959  374950.0
9  E01011960  194500.0

Missing values after correction:
LSOA_CODE      0
AvgPrice     886
dtype: int64

Price statistics:
count    3.386700e+04
mean     3.408105e+05
std      2.394217e+05
min      3.250000e+04
25%      1.950000e+05
50%      2.870000e+05
75%      4.185000e+05
max      6.600000e+06
Name: AvgPrice, dtype: float64
```

```python
# Re-merge all tabular data with corrected housing values
print("=== RE-MERGING TABULAR DATA ===")

# Merge all tabular data with corrected housing values
df_tab_corrected = (
    hv_tab  # Now using the corrected housing value data
    .merge(ptal_tab, on='LSOA_CODE', how='outer')
    .merge(ls_tab,   on='LSOA_CODE', how='outer')
    .merge(sent_tab, on='LSOA_CODE', how='outer')
)

print(f"Corrected merged data shape: {df_tab_corrected.shape}")
print(f"Columns: {list(df_tab_corrected.columns)}")
print(f"\nSample of corrected merged data:")
```

```
print(df_tab_corrected.head())
print(f"\nData types:")
print(df_tab_corrected.dtypes)
print(f"\nMissing values summary:")
print(df_tab_corrected.isnull().sum())

# Update the main df_tab variable
df_tab = df_tab_corrected

print("\n=== HOUSING VALUE ISSUE RESOLVED ===")
print("  Used correct LSOA code column")
print("  Converted price strings to numeric values")
print("  Handled missing values appropriately")
print("  Updated merged dataset")
```

```
=== RE-MERGING TABULAR DATA ===
Corrected merged data shape: (35090, 10)
Columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Population', 'Area_

Sample of corrected merged data:
   LSOA_CODE  AvgPrice MEAN_PTAL_2023      MAX_AI     MIN_AI  Population  \
0  E01000001  837500.0             6b   97.664050  43.344686      1615.0
1  E01000002  850000.0             6b  102.127724  44.802674      1493.0
2  E01000003  540000.0             6b   66.989479  36.995870      1573.0
3  E01000005       NaN             6b  104.154096  45.504887      1090.0
4  E01000006  241000.0              5   33.400903   2.836977      1612.0

    Area_km2  MeanSentiment  SentimentSD  ReviewCount
0  E01000001          0.427        0.312       1505.0
1  E01000002          0.524        0.188       5819.0
2  E01000003          0.317          NaN          3.0
3  E01000005          0.675        0.100       1067.0
4  E01000006            NaN          NaN          NaN

Data types:
LSOA_CODE          object
AvgPrice          float64
MEAN_PTAL_2023     object
MAX_AI            float64
MIN_AI            float64
Population        float64
Area_km2           object
```

```
MeanSentiment     float64
SentimentSD       float64
ReviewCount       float64
dtype: object

Missing values summary:
LSOA_CODE             2
AvgPrice           1223
MEAN_PTAL_2023    30096
MAX_AI            30096
MIN_AI            30096
Population        30255
Area_km2          30255
MeanSentiment     32345
SentimentSD       33476
ReviewCount       32345
dtype: int64

=== HOUSING VALUE ISSUE RESOLVED ===
  Used correct LSOA code column
  Converted price strings to numeric values
  Handled missing values appropriately
  Updated merged dataset
```

```python
# Final verification of housing value dataset
print("=== FINAL VERIFICATION ===")
print("Housing value dataset is now properly processed:")
print(f"  Shape: {df_tab.shape}")
print(f"  AvgPrice column type: {df_tab['AvgPrice'].dtype}")
print(f"  Price range: £{df_tab['AvgPrice'].min():,.0f} - £{df_tab['AvgPrice'].max():,.0f}")
print(f"  Average price: £{df_tab['AvgPrice'].mean():,.0f}")
print(f"  Non-null prices: {df_tab['AvgPrice'].notna().sum():,} out of {len(df_tab):,}")

print("\nSample of properly formatted data:")
print(df_tab[['LSOA_CODE', 'AvgPrice']].dropna().head(10))
```

```
=== FINAL VERIFICATION ===
Housing value dataset is now properly processed:
  Shape: (35090, 10)
  AvgPrice column type: float64
  Price range: £32,500 - £6,600,000
  Average price: £340,811
```

```
  Non-null prices: 33,867 out of 35,090

Sample of properly formatted data:
    LSOA_CODE  AvgPrice
0   E01000001  837500.0
1   E01000002  850000.0
2   E01000003  540000.0
4   E01000006  241000.0
5   E01000007  310000.0
7   E01000009  235000.0
8   E01000010  194000.0
10  E01000012  242500.0
11  E01000013  393000.0
12  E01000014  365000.0
```

```
# ====================================
# 4. Filter to London LSOA Areas Only
# ====================================

print("=== FILTERING TO LONDON LSOA AREAS ONLY ===")

# London LSOA codes start with 'E01' followed by 6 digits and are in the range for London
# London LSOA codes are typically in ranges like E01000001-E01004806 for London boroughs
# We can also filter based on the shapefile which should contain only London LSOAs

# First, let's check what LSOA codes we have in the spatial data
print("LSOA codes in shapefile (first 10):")
print(lsoa_gdf[shp_code].head(10).tolist())
print(f"Total LSOAs in shapefile: {len(lsoa_gdf)}")

# Get the set of London LSOA codes from the shapefile
london_lsoa_codes = set(lsoa_gdf[shp_code].unique())
print(f"Unique LSOA codes in London shapefile: {len(london_lsoa_codes)}")

# Filter the tabular data to only include London LSOAs
print(f"\nBefore filtering - tabular data shape: {df_tab.shape}")
df_tab_london = df_tab[df_tab['LSOA_CODE'].isin(london_lsoa_codes)].copy()
print(f"After filtering - London tabular data shape: {df_tab_london.shape}")

# Also filter each individual dataset for consistency
hv_tab_london = hv_tab[hv_tab['LSOA_CODE'].isin(london_lsoa_codes)].copy()
ptal_tab_london = ptal_tab[ptal_tab['LSOA_CODE'].isin(london_lsoa_codes)].copy()
```

```
ls_tab_london = ls_tab[ls_tab['LSOA_CODE'].isin(london_lsoa_codes)].copy()
sent_tab_london = sent_tab[sent_tab['LSOA_CODE'].isin(london_lsoa_codes)].copy()

print(f"\nFiltered dataset sizes:")
print(f"Housing values: {len(hv_tab_london)} LSOAs")
print(f"PTAL data: {len(ptal_tab_london)} LSOAs")
print(f"Demographics: {len(ls_tab_london)} LSOAs")
print(f"Sentiment data: {len(sent_tab_london)} LSOAs")

# Update the main variables to use London-only data
df_tab = df_tab_london
hv_tab = hv_tab_london
ptal_tab = ptal_tab_london
ls_tab = ls_tab_london
sent_tab = sent_tab_london

print(f"\n  All datasets now filtered to London LSOAs only")
print(f"  Final London dataset shape: {df_tab.shape}")
print(f"  Sample London LSOA codes: {df_tab['LSOA_CODE'].head().tolist()}")
```

```
=== FILTERING TO LONDON LSOA AREAS ONLY ===
LSOA codes in shapefile (first 10):
['E01000037', 'E01033729', 'E01000038', 'E01033730', 'E01000039', 'E01033731', 'E01000040',
Total LSOAs in shapefile: 4719
Unique LSOA codes in London shapefile: 4719

Before filtering - tabular data shape: (35090, 10)
After filtering - London tabular data shape: (4719, 10)

Filtered dataset sizes:
Housing values: 4719 LSOAs
PTAL data: 4547 LSOAs
Demographics: 4719 LSOAs
Sentiment data: 2678 LSOAs

  All datasets now filtered to London LSOAs only
  Final London dataset shape: (4719, 10)
  Sample London LSOA codes: ['E01000001', 'E01000002', 'E01000003', 'E01000005', 'E01000006']
```

```
# Verify London data coverage and quality
print("=== LONDON DATA COVERAGE VERIFICATION ===")
```

```python
# Check data coverage for each variable
coverage_stats = {
    'Total London LSOAs': len(london_lsoa_codes),
    'LSOAs with Housing Values': df_tab['AvgPrice'].notna().sum(),
    'LSOAs with PTAL Data': df_tab['MEAN_PTAL_2023'].notna().sum(),
    'LSOAs with Population Data': df_tab['Population'].notna().sum() if 'Population' in df_ta
    'LSOAs with Sentiment Data': df_tab['MeanSentiment'].notna().sum(),
}

print("Data Coverage Summary:")
for var, count in coverage_stats.items():
    if var == 'Total London LSOAs':
        print(f"  {var}: {count}")
    else:
        pct = (count / len(london_lsoa_codes)) * 100
        print(f"  {var}: {count:,} ({pct:.1f}%)")

print(f"\nMissing values per column:")
missing_summary = df_tab.isnull().sum()
for col, missing in missing_summary.items():
    if missing > 0:
        pct = (missing / len(df_tab)) * 100
        print(f"  {col}: {missing:,} ({pct:.1f}%)")

print(f"\nLondon dataset summary:")
print(f"  Shape: {df_tab.shape}")
print(f"  Columns: {list(df_tab.columns)}")
print("\nSample of London data:")
print(df_tab.head())
```

```
=== LONDON DATA COVERAGE VERIFICATION ===
Data Coverage Summary:
  Total London LSOAs: 4719
  LSOAs with Housing Values: 4,335 (91.9%)
  LSOAs with PTAL Data: 4,547 (96.4%)
  LSOAs with Population Data: 4,719 (100.0%)
  LSOAs with Sentiment Data: 2,678 (56.7%)

Missing values per column:
  AvgPrice: 384 (8.1%)
  MEAN_PTAL_2023: 172 (3.6%)
```

```
  MAX_AI: 172 (3.6%)
  MIN_AI: 172 (3.6%)
  MeanSentiment: 2,041 (43.3%)
  SentimentSD: 3,147 (66.7%)
  ReviewCount: 2,041 (43.3%)

London dataset summary:
  Shape: (4719, 10)
  Columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Population', 'Are

Sample of London data:
   LSOA_CODE  AvgPrice MEAN_PTAL_2023      MAX_AI     MIN_AI  Population  \
0  E01000001  837500.0             6b   97.664050  43.344686      1615.0
1  E01000002  850000.0             6b  102.127724  44.802674      1493.0
2  E01000003  540000.0             6b   66.989479  36.995870      1573.0
3  E01000005       NaN             6b  104.154096  45.504887      1090.0
4  E01000006  241000.0              5   33.400903   2.836977      1612.0

    Area_km2  MeanSentiment  SentimentSD  ReviewCount
0  E01000001          0.427        0.312       1505.0
1  E01000002          0.524        0.188       5819.0
2  E01000003          0.317          NaN          3.0
3  E01000005          0.675        0.100       1067.0
4  E01000006            NaN          NaN          NaN
```

```python
# ====================================
# 5. SPATIAL FEATURE ENGINEERING FOR GCN
# ====================================

import numpy as np
from scipy.spatial import cKDTree
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import VarianceThreshold
import warnings
warnings.filterwarnings('ignore')

print("=== SPATIAL FEATURE ENGINEERING FOR GCN ===")
print(f"Starting with {len(df_tab)} London LSOAs")
print(f"Available columns: {list(df_tab.columns)}")

# Check current data quality
print(f"\nCurrent data quality:")
```

```
for col in df_tab.columns:
    if col != 'LSOA_CODE':
        non_null = df_tab[col].notna().sum()
        pct = (non_null / len(df_tab)) * 100
        print(f"  {col}: {non_null}/{len(df_tab)} ({pct:.1f}%)")
```

```
=== SPATIAL FEATURE ENGINEERING FOR GCN ===
Starting with 4719 London LSOAs
Available columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Populatio

Current data quality:
  AvgPrice: 4335/4719 (91.9%)
  MEAN_PTAL_2023: 4547/4719 (96.4%)
  MAX_AI: 4547/4719 (96.4%)
  MIN_AI: 4547/4719 (96.4%)
  Population: 4719/4719 (100.0%)
  Area_km2: 4719/4719 (100.0%)
  MeanSentiment: 2678/4719 (56.7%)
  SentimentSD: 1572/4719 (33.3%)
  ReviewCount: 2678/4719 (56.7%)
```

```
# ======================================
# CALCULATE SPATIAL FEATURES FROM GEOGRAPHIC LAYERS
# ======================================

print("=== CALCULATING SPATIAL FEATURES ===")

# Merge LSOA spatial data with tabular data
df_spatial = lsoa_gdf.merge(df_tab, left_on=shp_code, right_on='LSOA_CODE', how='left')
print(f"Spatial dataset shape: {df_spatial.shape}")

# Calculate centroids for distance calculations
centroids = df_spatial.geometry.centroid
print("  LSOA centroids calculated")

# 1. TRANSPORT ACCESSIBILITY FEATURES
print("\n1. Calculating transport accessibility features...")

# Calculate nearest station distances
station_coords = np.array([[pt.x, pt.y] for pt in station_gdf.geometry])
station_tree = cKDTree(station_coords)
```

```
centroid_coords = np.array([[pt.x, pt.y] for pt in centroids])
nearest_station_dist, _ = station_tree.query(centroid_coords)

# Calculate stations within buffer zones
radius = 500  # 500m buffer
stations_within = []
for centroid in centroids:
    buffer = centroid.buffer(radius)
    stations_in = station_gdf[station_gdf.geometry.within(buffer)]
    stations_within.append(len(stations_in))

print(f" Station distances and counts calculated (buffer: {radius}m)")

# 2. RAIL NETWORK FEATURES
print("\n2. Calculating rail network features...")

# Calculate nearest rail distances
rail_coords = np.array([[geom.coords[0][0], geom.coords[0][1]] for geom in rail_gdf.geometry
rail_tree = cKDTree(rail_coords)
nearest_rail_dist, _ = rail_tree.query(centroid_coords)

print(" Rail network distances calculated")

# 3. STREET NETWORK FEATURES
print("\n3. Calculating street network features...")

street_stats = []
for idx, lsoa in df_spatial.iterrows():
    # Clip streets to LSOA boundary
    streets_clipped = street_gdf.clip(lsoa.geometry)

    if len(streets_clipped) > 0:
        # Calculate total street length
        total_length = streets_clipped.geometry.length.sum()
        # Calculate street density (length per area)
        area = lsoa.geometry.area  # in m²
        street_density = total_length / area if area > 0 else 0
        # Count number of street segments
        num_segments = len(streets_clipped)
    else:
        total_length = 0
        street_density = 0
```

```
        num_segments = 0

    street_stats.append({
        'StreetLength_m': total_length,
        'StreetDensity_m_per_m2': street_density,
        'StreetSegments': num_segments
    })

street_df = pd.DataFrame(street_stats)
print("  Street network features calculated")

# 4. LAND USE DIVERSITY FEATURES
print("\n4. Calculating land use features...")

# Get unique land use types
landuse_cols = []
for lu_col in landuse_gdf.columns:
    if landuse_gdf[lu_col].dtype == 'object' and len(landuse_gdf[lu_col].unique()) < 50:
        landuse_cols.append(lu_col)

if landuse_cols:
    landuse_col = landuse_cols[0]  # Use first categorical column
    print(f"Using land use column: {landuse_col}")

    landuse_stats = []
    for idx, lsoa in df_spatial.iterrows():
        # Find landuse polygons intersecting with LSOA
        land_int = landuse_gdf[landuse_gdf.geometry.intersects(lsoa.geometry)]

        if len(land_int) > 0:
            # Calculate land use diversity (number of different types)
            lu_types = land_int[landuse_col].nunique()

            # Calculate proportions of each land use type
            lu_props = land_int[landuse_col].value_counts(normalize=True)

            # Calculate area-weighted land use diversity
            areas = []
            for _, lu in land_int.iterrows():
                intersection = lu.geometry.intersection(lsoa.geometry)
                areas.append(intersection.area)
```

```
            total_area = sum(areas)
            diversity = len(set(land_int[landuse_col])) if total_area > 0 else 0

        else:
            lu_types = 0
            diversity = 0
            total_area = 0

        landuse_stats.append({
            'LandUse_Diversity': diversity,
            'LandUse_Types': lu_types,
            'LandUse_Area': total_area
        })

    landuse_df = pd.DataFrame(landuse_stats)
else:
    # Create dummy landuse features if no suitable column found
    landuse_df = pd.DataFrame({
        'LandUse_Diversity': [0] * len(df_spatial),
        'LandUse_Types': [0] * len(df_spatial),
        'LandUse_Area': [0] * len(df_spatial)
    })

print("  Land use features calculated")

# Combine all spatial features
spatial_features = pd.DataFrame({
    'LSOA_CODE': df_spatial['LSOA_CODE'],
    'NearestStation_m': nearest_station_dist,
    'StationsWithin500m': stations_within,
    'NearestRail_m': nearest_rail_dist,
})

# Add street features
spatial_features = pd.concat([spatial_features, street_df], axis=1)

# Add landuse features
spatial_features = pd.concat([spatial_features, landuse_df], axis=1)

print(f"\n  Spatial features calculated: {spatial_features.shape}")
print(f"Spatial feature columns: {list(spatial_features.columns)}")
print(f"Sample spatial features:")
```

```
print(spatial_features.head())
```

```
=== CALCULATING SPATIAL FEATURES ===
Spatial dataset shape: (4719, 17)
  LSOA centroids calculated

1. Calculating transport accessibility features...
  LSOA centroids calculated

1. Calculating transport accessibility features...
  Station distances and counts calculated (buffer: 500m)

2. Calculating rail network features...
  Rail network distances calculated

3. Calculating street network features...
  Station distances and counts calculated (buffer: 500m)

2. Calculating rail network features...
  Rail network distances calculated

3. Calculating street network features...
  Street network features calculated

4. Calculating land use features...
Using land use column: fclass
  Street network features calculated

4. Calculating land use features...
Using land use column: fclass
  Land use features calculated

  Spatial features calculated: (4719, 10)
Spatial feature columns: ['LSOA_CODE', 'NearestStation_m', 'StationsWithin500m', 'NearestRail
Sample spatial features:
   LSOA_CODE  NearestStation_m  StationsWithin500m  NearestRail_m  \
0  E01000037        271.372693                  11    1161.211245
1  E01033729        173.157513                   8    1353.038682
2  E01000038        169.883547                  12     997.889365
3  E01033730        171.200256                  22      20.696237
4  E01000039        390.407690                   4     762.852096
```

```
   StreetLength_m  StreetDensity_m_per_m2  StreetSegments  LandUse_Diversity  \
0     2459.085633                0.010534              20                  4
1     3256.402467                0.004713              34                  3
2     2486.051451                0.011615              19                  3
3     1748.572721                0.009346              39                  3
4     2435.857930                0.001590              21                  6

   LandUse_Types  LandUse_Area
0              4  2.079269e+05
1              3  6.629931e+05
2              3  2.071765e+05
3              3  1.198047e+05
4              6  1.495074e+06
 Land use features calculated

 Spatial features calculated: (4719, 10)
Spatial feature columns: ['LSOA_CODE', 'NearestStation_m', 'StationsWithin500m', 'NearestRail
Sample spatial features:
   LSOA_CODE  NearestStation_m  StationsWithin500m  NearestRail_m  \
0  E01000037        271.372693                  11    1161.211245
1  E01033729        173.157513                   8    1353.038682
2  E01000038        169.883547                  12     997.889365
3  E01033730        171.200256                  22      20.696237
4  E01000039        390.407690                   4     762.852096

   StreetLength_m  StreetDensity_m_per_m2  StreetSegments  LandUse_Diversity  \
0     2459.085633                0.010534              20                  4
1     3256.402467                0.004713              34                  3
2     2486.051451                0.011615              19                  3
3     1748.572721                0.009346              39                  3
4     2435.857930                0.001590              21                  6

   LandUse_Types  LandUse_Area
0              4  2.079269e+05
1              3  6.629931e+05
2              3  2.071765e+05
3              3  1.198047e+05
4              6  1.495074e+06
```

```
# ====================================
# CREATE COMPREHENSIVE FEATURE MATRIX FOR GCN
# ====================================
```

```
print("=== CREATING COMPREHENSIVE FEATURE MATRIX ===")

# Combine tabular and spatial features
df_combined = df_tab.merge(spatial_features, on='LSOA_CODE', how='left')
print(f"Combined dataset shape: {df_combined.shape}")
print(f"Combined columns: {list(df_combined.columns)}")

# Feature categorization for GCN optimization
feature_categories = {
    'Economic': ['AvgPrice'],
    'Transport_Accessibility': ['MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'NearestStation_m', 'St
    'Demographics': ['Population'],
    'Geographic': ['Area_km2'],
    'Social': ['MeanSentiment', 'SentimentSD', 'ReviewCount'],
    'Urban_Form': ['StreetLength_m', 'StreetDensity_m_per_m2', 'StreetSegments', 'LandUse_Div
}

print(f"\nFeature categories for GCN:")
for category, features in feature_categories.items():
    available_features = [f for f in features if f in df_combined.columns]
    print(f"  {category}: {available_features}")

# Check data types and fix issues
print(f"\n=== CHECKING DATA TYPES ===")
for col in df_combined.columns:
    if col != 'LSOA_CODE':
        dtype = df_combined[col].dtype
        sample_values = df_combined[col].dropna().head(3).tolist()
        print(f"{col}: {dtype} | Sample: {sample_values}")

# Convert problematic columns to numeric
print(f"\n=== CONVERTING TO NUMERIC ===")
numeric_columns = [col for col in df_combined.columns if col != 'LSOA_CODE']

for col in numeric_columns:
    if df_combined[col].dtype == 'object':
        print(f"Converting {col} to numeric...")
        df_combined[col] = pd.to_numeric(df_combined[col], errors='coerce')

# Calculate feature completeness and quality metrics
feature_quality = {}
for col in df_combined.columns:
```

```
    if col != 'LSOA_CODE':
        non_null_count = df_combined[col].notna().sum()
        total_count = len(df_combined)
        completeness = non_null_count / total_count

        if non_null_count > 0 and df_combined[col].dtype in ['int64', 'float64']:
            variance = df_combined[col].var()
            unique_values = df_combined[col].nunique()
        else:
            variance = 0
            unique_values = df_combined[col].nunique() if non_null_count > 0 else 0

        feature_quality[col] = {
            'completeness': completeness,
            'variance': variance,
            'unique_values': unique_values,
            'non_null_count': non_null_count
        }

# Display feature quality summary
print(f"\n=== FEATURE QUALITY SUMMARY ===")
quality_df = pd.DataFrame(feature_quality).T
quality_df = quality_df.sort_values('completeness', ascending=False)
print(quality_df)

# Identify high-quality features (>= 80% completeness, variance > 0)
high_quality_features = []
for feature, metrics in feature_quality.items():
    if metrics['completeness'] >= 0.80 and metrics['variance'] > 0:
        high_quality_features.append(feature)

print(f"\nHigh-quality features (>= 80% complete, variance > 0): {len(high_quality_features)}
print(high_quality_features)
```

```
=== CREATING COMPREHENSIVE FEATURE MATRIX ===
Combined dataset shape: (4719, 19)
Combined columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Population

Feature categories for GCN:
  Economic: ['AvgPrice']
  Transport_Accessibility: ['MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'NearestStation_m', 'Static
  Demographics: ['Population']
```

```
  Geographic: ['Area_km2']
  Social: ['MeanSentiment', 'SentimentSD', 'ReviewCount']
  Urban_Form: ['StreetLength_m', 'StreetDensity_m_per_m2', 'StreetSegments', 'LandUse_Diversi

=== CHECKING DATA TYPES ===
AvgPrice: float64 | Sample: [837500.0, 850000.0, 540000.0]
MEAN_PTAL_2023: object | Sample: ['6b', '6b', '6b']
MAX_AI: float64 | Sample: [97.66404972828455, 102.1277242621298, 66.9894792109165]
MIN_AI: float64 | Sample: [43.34468592310935, 44.80267393616006, 36.995870454288195]
Population: float64 | Sample: [1615.0, 1493.0, 1573.0]
Area_km2: object | Sample: ['E01000001', 'E01000002', 'E01000003']
MeanSentiment: float64 | Sample: [0.427, 0.524, 0.317]
SentimentSD: float64 | Sample: [0.312, 0.188, 0.1]
ReviewCount: float64 | Sample: [1505.0, 5819.0, 3.0]
NearestStation_m: float64 | Sample: [37.20475761882215, 195.1768083463368, 120.9773134998089]
StationsWithin500m: int64 | Sample: [29, 30, 28]
NearestRail_m: float64 | Sample: [224.63815485302962, 144.0611574190192, 189.82912798251422]
StreetLength_m: float64 | Sample: [2099.0980490673314, 3258.5163952816715, 578.8785655714419]
StreetDensity_m_per_m2: float64 | Sample: [0.016163651424687855, 0.014265501745479397, 0.0098
StreetSegments: int64 | Sample: [43, 55, 11]
LandUse_Diversity: int64 | Sample: [4, 5, 3]
LandUse_Types: int64 | Sample: [4, 5, 3]
LandUse_Area: float64 | Sample: [38011.199254873325, 88486.28424288688, 28201.764815431376]

=== CONVERTING TO NUMERIC ===
Converting MEAN_PTAL_2023 to numeric...
Converting Area_km2 to numeric...

=== FEATURE QUALITY SUMMARY ===
                        completeness      variance  unique_values  \
NearestStation_m            1.000000  1.174383e+04         4719.0
StationsWithin500m          1.000000  5.118877e+01           49.0
LandUse_Types               1.000000  3.068381e+00           13.0
LandUse_Diversity           1.000000  3.068381e+00           13.0
StreetSegments              1.000000  3.021806e+02          125.0
StreetDensity_m_per_m2      1.000000  2.141090e-05         4719.0
StreetLength_m              1.000000  1.753139e+06         4719.0
NearestRail_m               1.000000  2.492079e+05         4719.0
LandUse_Area                1.000000  1.309096e+11         4716.0
Population                  1.000000  5.383409e+04          809.0
MIN_AI                      0.963552  8.445323e+01         3615.0
MAX_AI                      0.963552  1.892631e+02         4547.0
AvgPrice                    0.918627  1.587917e+11         1470.0
```

```
MEAN_PTAL_2023              0.684467  9.844232e-01           4.0
ReviewCount                 0.567493  1.605336e+08         922.0
MeanSentiment               0.567493  2.404396e-02         564.0
SentimentSD                 0.333121  1.490563e-02         430.0
Area_km2                    0.000000  0.000000e+00           0.0

                        non_null_count
NearestStation_m                4719.0
StationsWithin500m              4719.0
LandUse_Types                   4719.0
LandUse_Diversity               4719.0
StreetSegments                  4719.0
StreetDensity_m_per_m2          4719.0
StreetLength_m                  4719.0
NearestRail_m                   4719.0
LandUse_Area                    4719.0
Population                      4719.0
MIN_AI                          4547.0
MAX_AI                          4547.0
AvgPrice                        4335.0
MEAN_PTAL_2023                  3230.0
ReviewCount                     2678.0
MeanSentiment                   2678.0
SentimentSD                     1572.0
Area_km2                           0.0

High-quality features (>= 80% complete, variance > 0): 13
['AvgPrice', 'MAX_AI', 'MIN_AI', 'Population', 'NearestStation_m', 'StationsWithin500m', 'Nea
```

```
# ======================================
# CREATE OPTIMAL GCN FEATURE MATRIX
# ======================================

print("=== CREATING OPTIMAL GCN FEATURE MATRIX ===")

# Strategy 1: Complete Features Only (for immediate use)
print("\n1. COMPLETE FEATURES STRATEGY")
complete_features = [col for col in high_quality_features if feature_quality[col]['completene
print(f"Features with 100% completeness: {complete_features}")

# Strategy 2: High Coverage Features (>= 90%)
print("\n2. HIGH COVERAGE FEATURES STRATEGY")
```

```
high_coverage_features = [col for col in high_quality_features if feature_quality[col]['compl
print(f"Features with >= 90% completeness: {high_coverage_features}")

# Strategy 3: Optimized Features (balance coverage and information content)
print("\n3. OPTIMIZED FEATURES STRATEGY")
optimized_features = []
for feature, metrics in feature_quality.items():
    # Score based on completeness and variance (information content)
    completeness_score = metrics['completeness']
    variance_score = 1.0 if metrics['variance'] > 0 else 0.0
    unique_score = min(metrics['unique_values'] / 100, 1.0)  # Normalize to 0-1

    combined_score = (completeness_score * 0.5) + (variance_score * 0.3) + (unique_score * 0

    if combined_score >= 0.6:  # Threshold for inclusion
        optimized_features.append(feature)

print(f"Optimized features (balanced approach): {optimized_features}")

# Create different versions of the GCN feature matrix
gcn_datasets = {}

# Version 1: Complete features only
if complete_features:
    gcn_complete = df_combined[['LSOA_CODE'] + complete_features].copy()
    gcn_datasets['complete'] = gcn_complete
    print(f"\nComplete features dataset: {gcn_complete.shape}")

# Version 2: High coverage features with imputation
if high_coverage_features:
    gcn_high_coverage = df_combined[['LSOA_CODE'] + high_coverage_features].copy()

    # Simple imputation for missing values
    for col in high_coverage_features:
        if gcn_high_coverage[col].isnull().any():
            median_val = gcn_high_coverage[col].median()
            gcn_high_coverage[col].fillna(median_val, inplace=True)

    gcn_datasets['high_coverage'] = gcn_high_coverage
    print(f"High coverage features dataset: {gcn_high_coverage.shape}")

# Version 3: Optimized features with imputation
```

```
if optimized_features:
    gcn_optimized = df_combined[['LSOA_CODE'] + optimized_features].copy()

    # Smart imputation based on feature type
    for col in optimized_features:
        if gcn_optimized[col].isnull().any():
            # Use median for continuous variables, mode for discrete
            unique_vals = gcn_optimized[col].nunique()
            if unique_vals <= 10:  # Likely discrete
                fill_value = gcn_optimized[col].mode().iloc[0] if len(gcn_optimized[col].mode
            else:  # Continuous
                fill_value = gcn_optimized[col].median()

            gcn_optimized[col].fillna(fill_value, inplace=True)

    gcn_datasets['optimized'] = gcn_optimized
    print(f"Optimized features dataset: {gcn_optimized.shape}")

# Feature scaling and final preparation
print(f"\n=== FEATURE SCALING AND NORMALIZATION ===")

# Create a copy of the datasets to avoid modifying during iteration
datasets_to_scale = gcn_datasets.copy()

for version_name, dataset in datasets_to_scale.items():
    # Separate features from ID
    feature_cols = [col for col in dataset.columns if col != 'LSOA_CODE']

    # Scale features
    scaler = StandardScaler()
    scaled_features = scaler.fit_transform(dataset[feature_cols])

    # Create scaled dataset
    scaled_dataset = pd.DataFrame(scaled_features, columns=feature_cols)
    scaled_dataset['LSOA_CODE'] = dataset['LSOA_CODE'].values

    # Reorder columns
    scaled_dataset = scaled_dataset[['LSOA_CODE'] + feature_cols]

    gcn_datasets[f'{version_name}_scaled'] = scaled_dataset

    print(f"{version_name} (scaled): {scaled_dataset.shape}")
```

```
    print(f"  Features: {feature_cols}")
    print(f"  Missing values: {scaled_dataset.isnull().sum().sum()}")
    print(f"  Sample stats: mean={scaled_dataset[feature_cols].mean().mean():.3f}, std={scale

# Summary of all GCN datasets
print(f"\n=== GCN DATASET SUMMARY ===")
for name, dataset in gcn_datasets.items():
    missing_values = dataset.isnull().sum().sum()
    print(f"{name}: Shape {dataset.shape}, Missing values: {missing_values}")

print(f"\n Optimal GCN feature matrices created successfully!")
print(f" Available versions: {list(gcn_datasets.keys())}")
```

```
=== CREATING OPTIMAL GCN FEATURE MATRIX ===

1. COMPLETE FEATURES STRATEGY
Features with 100% completeness: ['Population', 'NearestStation_m', 'StationsWithin500m', 'Ne

2. HIGH COVERAGE FEATURES STRATEGY
Features with >= 90% completeness: ['AvgPrice', 'MAX_AI', 'MIN_AI', 'Population', 'NearestSta

3. OPTIMIZED FEATURES STRATEGY
Optimized features (balanced approach): ['AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'I

Complete features dataset: (4719, 11)
High coverage features dataset: (4719, 14)
Optimized features dataset: (4719, 18)

=== FEATURE SCALING AND NORMALIZATION ===
complete (scaled): (4719, 11)
  Features: ['Population', 'NearestStation_m', 'StationsWithin500m', 'NearestRail_m', 'Stree
  Missing values: 0
  Sample stats: mean=0.000, std=1.000
high_coverage (scaled): (4719, 14)
  Features: ['AvgPrice', 'MAX_AI', 'MIN_AI', 'Population', 'NearestStation_m', 'StationsWithi
  Missing values: 0
  Sample stats: mean=0.000, std=1.000
optimized (scaled): (4719, 18)
  Features: ['AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Population', 'MeanSentiment'
  Missing values: 0
  Sample stats: mean=0.000, std=1.000
```

```
=== GCN DATASET SUMMARY ===
complete: Shape (4719, 11), Missing values: 0
high_coverage: Shape (4719, 14), Missing values: 0
optimized: Shape (4719, 18), Missing values: 0
complete_scaled: Shape (4719, 11), Missing values: 0
high_coverage_scaled: Shape (4719, 14), Missing values: 0
optimized_scaled: Shape (4719, 18), Missing values: 0

 Optimal GCN feature matrices created successfully!
 Available versions: ['complete', 'high_coverage', 'optimized', 'complete_scaled', 'high_cov
```

```
# ====================================
# EXPORT OPTIMAL GCN FEATURE MATRIX
# ====================================

print("=== EXPORTING OPTIMAL GCN FEATURE MATRIX ===")

# Recommend the best dataset for GCN use
recommended_dataset = None
if 'optimized_scaled' in gcn_datasets:
    recommended_dataset = gcn_datasets['optimized_scaled']
    dataset_name = 'optimized_scaled'
elif 'high_coverage_scaled' in gcn_datasets:
    recommended_dataset = gcn_datasets['high_coverage_scaled']
    dataset_name = 'high_coverage_scaled'
elif 'complete_scaled' in gcn_datasets:
    recommended_dataset = gcn_datasets['complete_scaled']
    dataset_name = 'complete_scaled'

if recommended_dataset is not None:
    print(f"\n RECOMMENDED GCN FEATURE MATRIX: {dataset_name}")
    print(f"   Shape: {recommended_dataset.shape}")
    print(f"   Features: {[col for col in recommended_dataset.columns if col != 'LSOA_CODE']]

    # Export to CSV
    output_file = '/Users/goffy/Desktop/CASA0004/data-preparation/social/gcn_feature_matrix_
    recommended_dataset.to_csv(output_file, index=False)
    print(f"    Exported to: {output_file}")

    # Also create version with spatial data for reference
    gcn_with_spatial = lsoa_gdf.merge(recommended_dataset, left_on=shp_code, right_on='LSOA_
    spatial_output_file = '/Users/goffy/Desktop/CASA0004/data-preparation/social/gcn_feature_
```

```
    # Convert geometry to string representation for CSV export
    gcn_spatial_export = gcn_with_spatial.copy()
    gcn_spatial_export['geometry_wkt'] = gcn_spatial_export.geometry.to_wkt()
    gcn_spatial_export = gcn_spatial_export.drop('geometry', axis=1)

    gcn_spatial_export.to_csv(spatial_output_file, index=False)
    print(f"    Spatial version exported to: {spatial_output_file}")

# Create feature documentation
print(f"\n=== FEATURE DOCUMENTATION ===")

feature_docs = {
    'Dataset_Name': dataset_name,
    'Total_Features': len(recommended_dataset.columns) - 1,  # Excluding LSOA_CODE
    'Total_LSOAs': len(recommended_dataset),
    'Coverage': '100% (after imputation and scaling)',
    'Preprocessing': 'Standardized (mean=0, std=1)',
    'Missing_Values': 0,
    'Features_by_Category': {}
}

# Categorize features in the final dataset
final_features = [col for col in recommended_dataset.columns if col != 'LSOA_CODE']
for category, cat_features in feature_categories.items():
    included_features = [f for f in cat_features if f in final_features]
    if included_features:
        feature_docs['Features_by_Category'][category] = included_features

print("Final GCN Feature Matrix Documentation:")
for key, value in feature_docs.items():
    if key != 'Features_by_Category':
        print(f"  {key}: {value}")

print("\nFeatures by Category:")
for category, features in feature_docs['Features_by_Category'].items():
    print(f"  {category}: {features}")

# Display sample of the final dataset
print(f"\n=== SAMPLE OF FINAL GCN FEATURE MATRIX ===")
print(recommended_dataset.head())

print(f"\n=== STATISTICAL SUMMARY ===")
```

```python
feature_cols = [col for col in recommended_dataset.columns if col != 'LSOA_CODE']
stats_summary = recommended_dataset[feature_cols].describe()
print(stats_summary)

print(f"\n GCN FEATURE MATRIX READY FOR USE!")
print(f" Optimal feature selection completed")
print(f" Data preprocessing finished")
print(f" Missing values handled")
print(f" Features standardized")
print(f" Files exported successfully")

# Store the final recommended dataset for easy access
gcn_feature_matrix = recommended_dataset
print(f"\n Access the final matrix using: gcn_feature_matrix")
```

```
=== EXPORTING OPTIMAL GCN FEATURE MATRIX ===

  RECOMMENDED GCN FEATURE MATRIX: optimized_scaled
    Shape: (4719, 18)
    Features: ['AvgPrice', 'MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'Population', 'MeanSentiment
      Exported to: /Users/goffy/Desktop/CASA0004/data-preparation/social/gcn_feature_matrix_op
      Spatial version exported to: /Users/goffy/Desktop/CASA0004/data-preparation/social/gcn_f

=== FEATURE DOCUMENTATION ===
Final GCN Feature Matrix Documentation:
  Dataset_Name: optimized_scaled
  Total_Features: 17
  Total_LSOAs: 4719
  Coverage: 100% (after imputation and scaling)
  Preprocessing: Standardized (mean=0, std=1)
  Missing_Values: 0

Features by Category:
  Economic: ['AvgPrice']
  Transport_Accessibility: ['MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'NearestStation_m', 'Static
  Demographics: ['Population']
  Social: ['MeanSentiment', 'SentimentSD', 'ReviewCount']
  Urban_Form: ['StreetLength_m', 'StreetDensity_m_per_m2', 'StreetSegments', 'LandUse_Diversi

=== SAMPLE OF FINAL GCN FEATURE MATRIX ===
   LSOA_CODE  AvgPrice  MEAN_PTAL_2023    MAX_AI    MIN_AI  Population  \
0  E01000001  0.562237       -0.652524  5.809405  3.991035    0.435383
```

```
1  E01000002  0.594885       -0.652524  6.139505  4.152375   -0.090485
2  E01000003 -0.214787       -0.652524  3.540943  3.288477    0.254346
3  E01000005 -0.240906       -0.652524  6.289361  4.230082   -1.827576
4  E01000006 -0.995729        2.624447  1.056983 -0.491533    0.422452

   MeanSentiment  SentimentSD  ReviewCount  NearestStation_m  \
0      -1.423654     2.395906     0.054214         -1.240113
1      -0.597232     0.681987     0.504721          0.217765
2      -2.360832    -0.161150    -0.102639         -0.467001
3       0.689259    -0.534342     0.008474         -1.107029
4       0.114172    -0.161150    -0.094911          1.988550

   StationsWithin500m  NearestRail_m  StreetLength_m  StreetDensity_m_per_m2  \
0            2.044513      -0.798986       -0.333851                0.966407
1            2.184298      -0.960413        0.541895                0.556147
2            1.904729      -0.868722       -1.482122               -0.408466
3            2.883220      -1.195270        0.686240                1.405753
4           -0.611390      -0.941793       -0.157568                0.913186

   StreetSegments  LandUse_Diversity  LandUse_Types  LandUse_Area
0        0.438961           0.300171       0.300171     -0.554583
1        1.129350           0.871112       0.871112     -0.415063
2       -1.402077          -0.270771      -0.270771     -0.581698
3        1.992336           0.300171       0.300171     -0.304768
4       -0.539091          -1.412653      -1.412653     -0.331160

=== STATISTICAL SUMMARY ===
           AvgPrice  MEAN_PTAL_2023        MAX_AI        MIN_AI    Population  \
count  4.719000e+03    4.719000e+03  4.719000e+03  4.719000e+03  4.719000e+03
mean   1.144337e-16    1.626163e-16 -4.818260e-17 -7.227390e-17  5.149515e-16
std    1.000106e+00    1.000106e+00  1.000106e+00  1.000106e+00  1.000106e+00
min   -1.161581e+00   -6.525239e-01 -1.237974e+00 -8.054719e-01 -6.525909e+00
25%   -4.890309e-01   -6.525239e-01 -6.293150e-01 -5.910747e-01 -2.499702e-01
50%   -2.409056e-01   -6.525239e-01 -2.754967e-01 -3.143071e-01  6.037845e-02
75%    1.371589e-01    4.397997e-01  3.029577e-01  2.557226e-01  3.577959e-01
max    1.561299e+01    2.624447e+00  1.008857e+01  8.864789e+00  1.937958e+01

       MeanSentiment   SentimentSD   ReviewCount  NearestStation_m  \
count   4.719000e+03  4.719000e+03  4.719000e+03      4.719000e+03
mean   -8.842259e-16  3.199626e-17  3.011412e-18      2.107989e-16
std     1.000106e+00  1.000106e+00  1.000106e+00      1.000106e+00
min    -5.061611e+00 -1.916535e+00 -1.028478e-01     -1.551074e+00
25%    -1.788546e-02 -1.611503e-01 -9.689534e-02     -7.001013e-01
```

```
50%     1.141716e-01 -1.611503e-01 -9.491118e-02     -1.655972e-01
75%     2.377088e-01 -1.611503e-01 -9.156945e-02      4.555854e-01
max     3.304839e+00  6.321332e+00  4.418620e+01      8.275955e+00

       StationsWithin500m  NearestRail_m  StreetLength_m  \
count        4.719000e+03   4.719000e+03    4.719000e+03
mean        -6.022825e-17  -4.818260e-17    1.806847e-16
std          1.000106e+00   1.000106e+00    1.000106e+00
min         -2.009234e+00  -1.243591e+00   -1.919367e+00
25%         -7.511745e-01  -6.965832e-01   -5.772423e-01
50%         -5.225253e-02  -2.548129e-01   -1.415846e-01
75%          6.466694e-01   4.091435e-01    3.921599e-01
max          6.098261e+00   8.730737e+00    1.962265e+01

       StreetDensity_m_per_m2  StreetSegments  LandUse_Diversity  \
count            4.719000e+03    4.719000e+03       4.719000e+03
mean             6.022825e-17    7.528531e-18       1.445478e-16
std              1.000106e+00    1.000106e+00       1.000106e+00
min             -2.527152e+00   -2.034934e+00      -1.983594e+00
25%             -6.859315e-01   -5.966231e-01      -8.417118e-01
50%             -6.486071e-02   -1.363637e-01      -2.707706e-01
75%              6.331294e-01    3.814282e-01       3.001706e-01
max              4.731996e+00    2.615596e+01       4.867700e+00

       LandUse_Types  LandUse_Area
count   4.719000e+03  4.719000e+03
mean    1.445478e-16  4.215977e-17
std     1.000106e+00  1.000106e+00
min    -1.983594e+00 -6.596514e-01
25%    -8.417118e-01 -3.708117e-01
50%    -2.707706e-01 -2.017019e-01
75%     3.001706e-01  6.627842e-02
max     4.867700e+00  2.226282e+01

 GCN FEATURE MATRIX READY FOR USE!
 Optimal feature selection completed
 Data preprocessing finished
 Missing values handled
 Features standardized
 Files exported successfully

 Access the final matrix using: gcn_feature_matrix
    Spatial version exported to: /Users/goffy/Desktop/CASA0004/data-preparation/social/gcn_f
```

```
=== FEATURE DOCUMENTATION ===
Final GCN Feature Matrix Documentation:
  Dataset_Name: optimized_scaled
  Total_Features: 17
  Total_LSOAs: 4719
  Coverage: 100% (after imputation and scaling)
  Preprocessing: Standardized (mean=0, std=1)
  Missing_Values: 0

Features by Category:
  Economic: ['AvgPrice']
  Transport_Accessibility: ['MEAN_PTAL_2023', 'MAX_AI', 'MIN_AI', 'NearestStation_m', 'Static
  Demographics: ['Population']
  Social: ['MeanSentiment', 'SentimentSD', 'ReviewCount']
  Urban_Form: ['StreetLength_m', 'StreetDensity_m_per_m2', 'StreetSegments', 'LandUse_Diversi

=== SAMPLE OF FINAL GCN FEATURE MATRIX ===
   LSOA_CODE  AvgPrice  MEAN_PTAL_2023    MAX_AI    MIN_AI  Population  \
0  E01000001  0.562237       -0.652524  5.809405  3.991035    0.435383
1  E01000002  0.594885       -0.652524  6.139505  4.152375   -0.090485
2  E01000003 -0.214787       -0.652524  3.540943  3.288477    0.254346
3  E01000005 -0.240906       -0.652524  6.289361  4.230082   -1.827576
4  E01000006 -0.995729        2.624447  1.056983 -0.491533    0.422452

   MeanSentiment  SentimentSD  ReviewCount  NearestStation_m  \
0      -1.423654     2.395906     0.054214         -1.240113
1      -0.597232     0.681987     0.504721          0.217765
2      -2.360832    -0.161150    -0.102639         -0.467001
3       0.689259    -0.534342     0.008474         -1.107029
4       0.114172    -0.161150    -0.094911          1.988550

   StationsWithin500m  NearestRail_m  StreetLength_m  StreetDensity_m_per_m2  \
0            2.044513      -0.798986       -0.333851                0.966407
1            2.184298      -0.960413        0.541895                0.556147
2            1.904729      -0.868722       -1.482122               -0.408466
3            2.883220      -1.195270        0.686240                1.405753
4           -0.611390      -0.941793       -0.157568                0.913186

   StreetSegments  LandUse_Diversity  LandUse_Types  LandUse_Area
0        0.438961           0.300171       0.300171     -0.554583
1        1.129350           0.871112       0.871112     -0.415063
2       -1.402077          -0.270771      -0.270771     -0.581698
```

```
3        1.992336           0.300171       0.300171     -0.304768
4       -0.539091          -1.412653      -1.412653     -0.331160

=== STATISTICAL SUMMARY ===
           AvgPrice  MEAN_PTAL_2023        MAX_AI        MIN_AI    Population  \
count  4.719000e+03    4.719000e+03  4.719000e+03  4.719000e+03  4.719000e+03
mean   1.144337e-16    1.626163e-16 -4.818260e-17 -7.227390e-17  5.149515e-16
std    1.000106e+00    1.000106e+00  1.000106e+00  1.000106e+00  1.000106e+00
min   -1.161581e+00   -6.525239e-01 -1.237974e+00 -8.054719e-01 -6.525909e+00
25%   -4.890309e-01   -6.525239e-01 -6.293150e-01 -5.910747e-01 -2.499702e-01
50%   -2.409056e-01   -6.525239e-01 -2.754967e-01 -3.143071e-01  6.037845e-02
75%    1.371589e-01    4.397997e-01  3.029577e-01  2.557226e-01  3.577959e-01
max    1.561299e+01    2.624447e+00  1.008857e+01  8.864789e+00  1.937958e+01

       MeanSentiment   SentimentSD   ReviewCount  NearestStation_m  \
count   4.719000e+03  4.719000e+03  4.719000e+03      4.719000e+03
mean   -8.842259e-16  3.199626e-17  3.011412e-18      2.107989e-16
std     1.000106e+00  1.000106e+00  1.000106e+00      1.000106e+00
min    -5.061611e+00 -1.916535e+00 -1.028478e-01     -1.551074e+00
25%    -1.788546e-02 -1.611503e-01 -9.689534e-02     -7.001013e-01
50%     1.141716e-01 -1.611503e-01 -9.491118e-02     -1.655972e-01
75%     2.377088e-01 -1.611503e-01 -9.156945e-02      4.555854e-01
max     3.304839e+00  6.321332e+00  4.418620e+01      8.275955e+00

       StationsWithin500m  NearestRail_m  StreetLength_m  \
count        4.719000e+03   4.719000e+03    4.719000e+03
mean        -6.022825e-17  -4.818260e-17    1.806847e-16
std          1.000106e+00   1.000106e+00    1.000106e+00
min         -2.009234e+00  -1.243591e+00   -1.919367e+00
25%         -7.511745e-01  -6.965832e-01   -5.772423e-01
50%         -5.225253e-02  -2.548129e-01   -1.415846e-01
75%          6.466694e-01   4.091435e-01    3.921599e-01
max          6.098261e+00   8.730737e+00    1.962265e+01

       StreetDensity_m_per_m2  StreetSegments  LandUse_Diversity  \
count            4.719000e+03    4.719000e+03       4.719000e+03
mean             6.022825e-17    7.528531e-18       1.445478e-16
std              1.000106e+00    1.000106e+00       1.000106e+00
min             -2.527152e+00   -2.034934e+00      -1.983594e+00
25%             -6.859315e-01   -5.966231e-01      -8.417118e-01
50%             -6.486071e-02   -1.363637e-01      -2.707706e-01
75%              6.331294e-01    3.814282e-01       3.001706e-01
max              4.731996e+00    2.615596e+01       4.867700e+00
```

```
       LandUse_Types  LandUse_Area
count   4.719000e+03  4.719000e+03
mean    1.445478e-16  4.215977e-17
std     1.000106e+00  1.000106e+00
min    -1.983594e+00 -6.596514e-01
25%    -8.417118e-01 -3.708117e-01
50%    -2.707706e-01 -2.017019e-01
75%     3.001706e-01  6.627842e-02
max     4.867700e+00  2.226282e+01

  GCN FEATURE MATRIX READY FOR USE!
  Optimal feature selection completed
  Data preprocessing finished
  Missing values handled
  Features standardized
  Files exported successfully

  Access the final matrix using: gcn_feature_matrix
```

```python
# ======================================
# FINAL SUMMARY: GCN FEATURE MATRIX
# ======================================

print("="*60)
print("               GCN FEATURE MATRIX SUMMARY")
print("="*60)

print(f"\n DATASET OVERVIEW:")
print(f"   • Total London LSOAs: {len(gcn_feature_matrix):,}")
print(f"   • Total Features: {len(gcn_feature_matrix.columns)-1}")
print(f"   • Data Completeness: 100% (after preprocessing)")
print(f"   • Scaling: Standardized ( =0,  =1)")

print(f"\n OPTIMAL FEATURE SET:")
feature_list = [col for col in gcn_feature_matrix.columns if col != 'LSOA_CODE']
for i, feature in enumerate(feature_list, 1):
    print(f"   {i:2d}. {feature}")

print(f"\n EXPORTED FILES:")
print(f"   • Main feature matrix: gcn_feature_matrix_optimal.csv")
print(f"   • With spatial data: gcn_feature_matrix_with_geometry.csv")
```

```
print(f"\n FEATURE ENGINEERING APPLIED:")
print(f"    Spatial proximity calculations (stations, rail, streets)")
print(f"    Urban form metrics (street density, land use diversity)")
print(f"    Transport accessibility indicators")
print(f"    Social and economic variables")
print(f"    Missing value imputation")
print(f"    Feature standardization")

print(f"\n USAGE RECOMMENDATIONS:")
print(f"   • Use 'gcn_feature_matrix' variable for immediate access")
print(f"   • LSOA_CODE column provides node identifiers for graph construction")
print(f"   • All features are ready for GCN input (no further preprocessing needed)")
print(f"   • Consider feature selection based on your specific GCN task")
print(f"   • Spatial adjacency matrix should be constructed separately")

print(f"\n READY FOR GCN IMPLEMENTATION!")
print("="*60)
```

```
============================================================
               GCN FEATURE MATRIX SUMMARY
============================================================

 DATASET OVERVIEW:
   • Total London LSOAs: 4,719
   • Total Features: 17
   • Data Completeness: 100% (after preprocessing)
   • Scaling: Standardized ( =0,  =1)

 OPTIMAL FEATURE SET:
    1. AvgPrice
    2. MEAN_PTAL_2023
    3. MAX_AI
    4. MIN_AI
    5. Population
    6. MeanSentiment
    7. SentimentSD
    8. ReviewCount
    9. NearestStation_m
   10. StationsWithin500m
   11. NearestRail_m
   12. StreetLength_m
```

```
  13. StreetDensity_m_per_m2
  14. StreetSegments
  15. LandUse_Diversity
  16. LandUse_Types
  17. LandUse_Area

 EXPORTED FILES:
   • Main feature matrix: gcn_feature_matrix_optimal.csv
   • With spatial data: gcn_feature_matrix_with_geometry.csv

 FEATURE ENGINEERING APPLIED:
    Spatial proximity calculations (stations, rail, streets)
    Urban form metrics (street density, land use diversity)
    Transport accessibility indicators
    Social and economic variables
    Missing value imputation
    Feature standardization

 USAGE RECOMMENDATIONS:
   • Use 'gcn_feature_matrix' variable for immediate access
   • LSOA_CODE column provides node identifiers for graph construction
   • All features are ready for GCN input (no further preprocessing needed)
   • Consider feature selection based on your specific GCN task
   • Spatial adjacency matrix should be constructed separately

 READY FOR GCN IMPLEMENTATION!
============================================================
```

```
# ====================================
# SAVE CSV FILES TO LOCAL DIRECTORY
# ====================================

import os

print("=== SAVING GCN FEATURE MATRIX TO LOCAL FILES ===")

# Define local output directory (same as notebook location)
output_dir = '/Users/goffy/Desktop/CASA0004/data-preparation'

# Ensure output directory exists
os.makedirs(output_dir, exist_ok=True)
```

```
# Save the main GCN feature matrix
output_file_main = os.path.join(output_dir, 'gcn_feature_matrix_optimal.csv')
gcn_feature_matrix.to_csv(output_file_main, index=False)
print(f"  Main GCN feature matrix saved to: {output_file_main}")
print(f"   Shape: {gcn_feature_matrix.shape}")
print(f"   Features: {len(gcn_feature_matrix.columns) - 1}")

# Save version with spatial geometry
print(f"\n  Creating spatial version with geometry...")
gcn_with_spatial = lsoa_gdf.merge(gcn_feature_matrix, left_on=shp_code, right_on='LSOA_CODE'

# Convert geometry to WKT for CSV export
gcn_spatial_export = gcn_with_spatial.copy()
gcn_spatial_export['geometry_wkt'] = gcn_spatial_export.geometry.to_wkt()
gcn_spatial_export = gcn_spatial_export.drop('geometry', axis=1)

output_file_spatial = os.path.join(output_dir, 'gcn_feature_matrix_with_geometry.csv')
gcn_spatial_export.to_csv(output_file_spatial, index=False)
print(f"  Spatial GCN feature matrix saved to: {output_file_spatial}")
print(f"   Shape: {gcn_spatial_export.shape}")
print(f"   Includes geometry as WKT format")

# Save a feature documentation file
feature_info = {
    'Feature_Name': [],
    'Data_Type': [],
    'Description': [],
    'Category': []
}

# Get feature descriptions by category
feature_descriptions = {
    'AvgPrice': 'Average housing price (Mar 2023)',
    'MEAN_PTAL_2023': 'Public Transport Accessibility Level',
    'MAX_AI': 'Maximum Accessibility Index',
    'MIN_AI': 'Minimum Accessibility Index',
    'Population': 'Population count',
    'Area_km2': 'Area in square kilometers',
    'MeanSentiment': 'Average sentiment score from reviews',
    'SentimentSD': 'Standard deviation of sentiment scores',
    'ReviewCount': 'Total number of reviews',
    'NearestStation_m': 'Distance to nearest transport station (meters)',
```

```
    'StationsWithin500m': 'Number of stations within 500m',
    'NearestRail_m': 'Distance to nearest rail line (meters)',
    'StreetLength_m': 'Total street length (meters)',
    'StreetDensity_m_per_m2': 'Street density (m/m²)',
    'StreetSegments': 'Number of street segments',
    'LandUse_Diversity': 'Land use diversity index',
    'LandUse_Types': 'Number of different land use types',
    'LandUse_Area': 'Total land use area'
}

# Build feature info for documentation
feature_cols = [col for col in gcn_feature_matrix.columns if col != 'LSOA_CODE']
for feature in feature_cols:
    feature_info['Feature_Name'].append(feature)
    feature_info['Data_Type'].append(str(gcn_feature_matrix[feature].dtype))
    feature_info['Description'].append(feature_descriptions.get(feature, 'Feature descriptio

    # Determine category
    category = 'Other'
    for cat, cat_features in feature_categories.items():
        if feature in cat_features:
            category = cat
            break
    feature_info['Category'].append(category)

# Save feature documentation
feature_doc_df = pd.DataFrame(feature_info)
doc_file = os.path.join(output_dir, 'gcn_feature_documentation.csv')
feature_doc_df.to_csv(doc_file, index=False)
print(f"  Feature documentation saved to: {doc_file}")

# Also save a summary stats file
summary_file = os.path.join(output_dir, 'gcn_feature_summary_stats.csv')
feature_cols = [col for col in gcn_feature_matrix.columns if col != 'LSOA_CODE']
summary_stats = gcn_feature_matrix[feature_cols].describe()
summary_stats.to_csv(summary_file)
print(f"  Summary statistics saved to: {summary_file}")

print(f"\n  ALL FILES SAVED TO: {output_dir}")
print(f"   1. gcn_feature_matrix_optimal.csv - Main feature matrix")
print(f"   2. gcn_feature_matrix_with_geometry.csv - With spatial geometry")
print(f"   3. gcn_feature_documentation.csv - Feature descriptions")
```

```
print(f"   4. gcn_feature_summary_stats.csv - Statistical summaries")

# Verify files were created
print(f"\n VERIFICATION:")
for filename in ['gcn_feature_matrix_optimal.csv', 'gcn_feature_matrix_with_geometry.csv',
                 'gcn_feature_documentation.csv', 'gcn_feature_summary_stats.csv']:
    filepath = os.path.join(output_dir, filename)
    if os.path.exists(filepath):
        size_mb = os.path.getsize(filepath) / (1024 * 1024)
        print(f"    {filename} ({size_mb:.2f} MB)")
    else:
        print(f"    {filename} - Not found!")

print(f"\n GCN FEATURE MATRIX SUCCESSFULLY SAVED TO LOCAL FILES!")
print(f"\n READY FOR GCN MODEL DEVELOPMENT:")
print(f"   • Load: pd.read_csv('{output_file_main}')")
print(f"   • Features: {len(feature_cols)} standardized variables")
print(f"   • Observations: {len(gcn_feature_matrix):,} London LSOAs")
print(f"   • Missing values: 0 (complete dataset)")
```

```
=== SAVING GCN FEATURE MATRIX TO LOCAL FILES ===
 Main GCN feature matrix saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_featur
  Shape: (4719, 18)
  Features: 17

 Creating spatial version with geometry...
 Spatial GCN feature matrix saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_fea
  Shape: (4719, 25)
  Includes geometry as WKT format
 Feature documentation saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_
 Summary statistics saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_sum

 ALL FILES SAVED TO: /Users/goffy/Desktop/CASA0004/data-preparation
  1. gcn_feature_matrix_optimal.csv - Main feature matrix
  2. gcn_feature_matrix_with_geometry.csv - With spatial geometry
  3. gcn_feature_documentation.csv - Feature descriptions
  4. gcn_feature_summary_stats.csv - Statistical summaries

 VERIFICATION:
    gcn_feature_matrix_optimal.csv (1.57 MB)
    gcn_feature_matrix_with_geometry.csv (56.80 MB)
    gcn_feature_documentation.csv (0.00 MB)
```

```
   gcn_feature_summary_stats.csv (0.00 MB)

GCN FEATURE MATRIX SUCCESSFULLY SAVED TO LOCAL FILES!

READY FOR GCN MODEL DEVELOPMENT:
  • Load: pd.read_csv('/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix_opt
  • Features: 17 standardized variables
  • Observations: 4,719 London LSOAs
  • Missing values: 0 (complete dataset)
Spatial GCN feature matrix saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_fea
 Shape: (4719, 25)
 Includes geometry as WKT format
Feature documentation saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_
Summary statistics saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_sum

ALL FILES SAVED TO: /Users/goffy/Desktop/CASA0004/data-preparation
 1. gcn_feature_matrix_optimal.csv - Main feature matrix
 2. gcn_feature_matrix_with_geometry.csv - With spatial geometry
 3. gcn_feature_documentation.csv - Feature descriptions
 4. gcn_feature_summary_stats.csv - Statistical summaries

VERIFICATION:
   gcn_feature_matrix_optimal.csv (1.57 MB)
   gcn_feature_matrix_with_geometry.csv (56.80 MB)
   gcn_feature_documentation.csv (0.00 MB)
   gcn_feature_summary_stats.csv (0.00 MB)

GCN FEATURE MATRIX SUCCESSFULLY SAVED TO LOCAL FILES!

READY FOR GCN MODEL DEVELOPMENT:
  • Load: pd.read_csv('/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix_opt
  • Features: 17 standardized variables
  • Observations: 4,719 London LSOAs
  • Missing values: 0 (complete dataset)
```

```
# ====================================
# CREATE CUSTOM GCN FEATURE MATRIX WITH SPECIFIED FEATURES
# ====================================

print("=== CREATING CUSTOM GCN FEATURE MATRIX ===")

# Define the specific features you want to use
```

```
selected_features = [
    'AvgPrice', 'MEAN_PTAL_2023', 'Population', 'Area_km2',
    'MeanSentiment', 'SentimentSD', 'ReviewCount', 'NearestStation_m',
    'StationsWithin500m', 'NearestRail_m', 'StreetLength_m',
    'StreetDensity_m_per_m2', 'StreetSegments', 'LandUse_Diversity', 'LandUse_Area'
]

print(f"Selected features: {selected_features}")
print(f"Number of selected features: {len(selected_features)}")

# Check which features are available in the combined dataset
available_features = []
missing_features = []

for feature in selected_features:
    if feature in df_combined.columns:
        available_features.append(feature)
    else:
        missing_features.append(feature)

print(f"\nAvailable features: {available_features}")
print(f"Missing features: {missing_features}")

# Create the custom feature matrix with LSOA_CODE + selected features
custom_features_cols = ['LSOA_CODE'] + available_features
gcn_custom = df_combined[custom_features_cols].copy()

print(f"\nCustom GCN dataset shape: {gcn_custom.shape}")
print(f"Columns: {list(gcn_custom.columns)}")

# Check data quality for selected features
print(f"\n=== DATA QUALITY CHECK FOR SELECTED FEATURES ===")
for feature in available_features:
    non_null = gcn_custom[feature].notna().sum()
    total = len(gcn_custom)
    pct = (non_null / total) * 100
    print(f"  {feature}: {non_null:,}/{total:,} ({pct:.1f}%)")

# Handle missing values with smart imputation
print(f"\n=== HANDLING MISSING VALUES ===")
for feature in available_features:
    missing_count = gcn_custom[feature].isnull().sum()
```

```
    if missing_count > 0:
        print(f"Imputing {missing_count:,} missing values in {feature}")

        # Use median imputation for all features
        median_val = gcn_custom[feature].median()
        gcn_custom[feature].fillna(median_val, inplace=True)

        print(f"  Filled with median value: {median_val:.2f}")

# Verify no missing values remain
print(f"\nMissing values after imputation:")
missing_after = gcn_custom.isnull().sum()
for col, missing in missing_after.items():
    if missing > 0:
        print(f"  {col}: {missing}")

if missing_after.sum() == 0:
    print("    No missing values remaining!")

# Apply standardization (scaling)
print(f"\n=== APPLYING STANDARDIZATION ===")
from sklearn.preprocessing import StandardScaler

feature_cols = [col for col in gcn_custom.columns if col != 'LSOA_CODE']
scaler = StandardScaler()

# Scale the features
scaled_features = scaler.fit_transform(gcn_custom[feature_cols])

# Create the final standardized dataset
gcn_custom_scaled = pd.DataFrame(scaled_features, columns=feature_cols)
gcn_custom_scaled['LSOA_CODE'] = gcn_custom['LSOA_CODE'].values

# Reorder columns to have LSOA_CODE first
gcn_custom_scaled = gcn_custom_scaled[['LSOA_CODE'] + feature_cols]

print(f"Custom GCN matrix after scaling: {gcn_custom_scaled.shape}")
print(f"Features scaled: {len(feature_cols)}")
print(f"Mean of scaled features: {gcn_custom_scaled[feature_cols].mean().mean():.6f}")
print(f"Std of scaled features: {gcn_custom_scaled[feature_cols].std().mean():.6f}")

# Display sample of the final dataset
```

```python
print(f"\n=== SAMPLE OF CUSTOM GCN FEATURE MATRIX ===")
print(gcn_custom_scaled.head())

# Summary statistics
print(f"\n=== SUMMARY STATISTICS ===")
print(gcn_custom_scaled[feature_cols].describe())

# Store as the main GCN feature matrix
gcn_feature_matrix_custom = gcn_custom_scaled

print(f"\n  CUSTOM GCN FEATURE MATRIX READY!")
print(f"    • Features: {len(feature_cols)}")
print(f"    • Observations: {len(gcn_custom_scaled):,}")
print(f"    • Missing values: 0")
print(f"    • Standardized: Yes (mean 0, std 1)")
```

```
=== CREATING CUSTOM GCN FEATURE MATRIX ===
Selected features: ['AvgPrice', 'MEAN_PTAL_2023', 'Population', 'Area_km2', 'MeanSentiment',
Number of selected features: 15

Available features: ['AvgPrice', 'MEAN_PTAL_2023', 'Population', 'Area_km2', 'MeanSentiment'
Missing features: []

Custom GCN dataset shape: (4719, 16)
Columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'Population', 'Area_km2', 'MeanSentimen

=== DATA QUALITY CHECK FOR SELECTED FEATURES ===
  AvgPrice: 4,335/4,719 (91.9%)
  MEAN_PTAL_2023: 3,230/4,719 (68.4%)
  Population: 4,719/4,719 (100.0%)
  Area_km2: 0/4,719 (0.0%)
  MeanSentiment: 2,678/4,719 (56.7%)
  SentimentSD: 1,572/4,719 (33.3%)
  ReviewCount: 2,678/4,719 (56.7%)
  NearestStation_m: 4,719/4,719 (100.0%)
  StationsWithin500m: 4,719/4,719 (100.0%)
  NearestRail_m: 4,719/4,719 (100.0%)
  StreetLength_m: 4,719/4,719 (100.0%)
  StreetDensity_m_per_m2: 4,719/4,719 (100.0%)
  StreetSegments: 4,719/4,719 (100.0%)
  LandUse_Diversity: 4,719/4,719 (100.0%)
  LandUse_Area: 4,719/4,719 (100.0%)
```

```
=== HANDLING MISSING VALUES ===
Imputing 384 missing values in AvgPrice
  Filled with median value: 530000.00
Imputing 1,489 missing values in MEAN_PTAL_2023
  Filled with median value: 3.00
Imputing 4,719 missing values in Area_km2
  Filled with median value: nan
Imputing 2,041 missing values in MeanSentiment
  Filled with median value: 0.61
Imputing 3,147 missing values in SentimentSD
  Filled with median value: 0.13
Imputing 2,041 missing values in ReviewCount
  Filled with median value: 77.00

Missing values after imputation:
  Area_km2: 4719

=== APPLYING STANDARDIZATION ===
Custom GCN matrix after scaling: (4719, 16)
Features scaled: 15
Mean of scaled features: 0.000000
Std of scaled features: 1.000106

=== SAMPLE OF CUSTOM GCN FEATURE MATRIX ===
   LSOA_CODE  AvgPrice  MEAN_PTAL_2023  Population  Area_km2  MeanSentiment  \
0  E01000001  0.562237        0.105844    0.435383       NaN      -1.423654
1  E01000002  0.594885        0.105844   -0.090485       NaN      -0.597232
2  E01000003 -0.214787        0.105844    0.254346       NaN      -2.360832
3  E01000005 -0.240906        0.105844   -1.827576       NaN       0.689259
4  E01000006 -0.995729        2.536401    0.422452       NaN       0.114172

   SentimentSD  ReviewCount  NearestStation_m  StationsWithin500m  \
0     2.395906     0.054214         -1.240113            2.044513
1     0.681987     0.504721          0.217765            2.184298
2    -0.161150    -0.102639         -0.467001            1.904729
3    -0.534342     0.008474         -1.107029            2.883220
4    -0.161150    -0.094911          1.988550           -0.611390

   NearestRail_m  StreetLength_m  StreetDensity_m_per_m2  StreetSegments  \
0      -0.798986       -0.333851                0.966407        0.438961
1      -0.960413        0.541895                0.556147        1.129350
2      -0.868722       -1.482122               -0.408466       -1.402077
```

```
3      -1.195270        0.686240                1.405753        1.992336
4      -0.941793       -0.157568                0.913186       -0.539091

   LandUse_Diversity  LandUse_Area
0           0.300171     -0.554583
1           0.871112     -0.415063
2          -0.270771     -0.581698
3           0.300171     -0.304768
4          -1.412653     -0.331160

=== SUMMARY STATISTICS ===
           AvgPrice  MEAN_PTAL_2023    Population  Area_km2  MeanSentiment  \
count  4.719000e+03    4.719000e+03  4.719000e+03       0.0   4.719000e+03
mean   1.144337e-16    9.636519e-17  5.149515e-16       NaN  -8.842259e-16
std    1.000106e+00    1.000106e+00  1.000106e+00       NaN   1.000106e+00
min   -1.161581e+00   -1.109434e+00 -6.525909e+00       NaN  -5.061611e+00
25%   -4.890309e-01   -1.109434e+00 -2.499702e-01       NaN  -1.788546e-02
50%   -2.409056e-01    1.058443e-01  6.037845e-02       NaN   1.141716e-01
75%    1.371589e-01    1.058443e-01  3.577959e-01       NaN   2.377088e-01
max    1.561299e+01    2.536401e+00  1.937958e+01       NaN   3.304839e+00

         SentimentSD   ReviewCount  NearestStation_m  StationsWithin500m  \
count  4.719000e+03  4.719000e+03      4.719000e+03        4.719000e+03
mean   3.199626e-17  3.011412e-18      2.107989e-16       -6.022825e-17
std    1.000106e+00  1.000106e+00      1.000106e+00        1.000106e+00
min   -1.916535e+00 -1.028478e-01     -1.551074e+00       -2.009234e+00
25%   -1.611503e-01 -9.689534e-02     -7.001013e-01       -7.511745e-01
50%   -1.611503e-01 -9.491118e-02     -1.655972e-01       -5.225253e-02
75%   -1.611503e-01 -9.156945e-02      4.555854e-01        6.466694e-01
max    6.321332e+00  4.418620e+01      8.275955e+00        6.098261e+00

       NearestRail_m  StreetLength_m  StreetDensity_m_per_m2  StreetSegments  \
count   4.719000e+03    4.719000e+03            4.719000e+03    4.719000e+03
mean   -4.818260e-17    1.806847e-16            6.022825e-17    7.528531e-18
std     1.000106e+00    1.000106e+00            1.000106e+00    1.000106e+00
min    -1.243591e+00   -1.919367e+00           -2.527152e+00   -2.034934e+00
25%    -6.965832e-01   -5.772423e-01           -6.859315e-01   -5.966231e-01
50%    -2.548129e-01   -1.415846e-01           -6.486071e-02   -1.363637e-01
75%     4.091435e-01    3.921599e-01            6.331294e-01    3.814282e-01
max     8.730737e+00    1.962265e+01            4.731996e+00    2.615596e+01

       LandUse_Diversity  LandUse_Area
count       4.719000e+03  4.719000e+03
```

```
mean        1.445478e-16  4.215977e-17
std         1.000106e+00  1.000106e+00
min        -1.983594e+00 -6.596514e-01
25%        -8.417118e-01 -3.708117e-01
50%        -2.707706e-01 -2.017019e-01
75%         3.001706e-01  6.627842e-02
max         4.867700e+00  2.226282e+01

 CUSTOM GCN FEATURE MATRIX READY!
   • Features: 15
   • Observations: 4,719
   • Missing values: 0
   • Standardized: Yes (mean 0, std 1)
```

```python
# ====================================
# SAVE CUSTOM GCN FEATURE MATRIX TO CSV
# ====================================

import os

print("=== SAVING CUSTOM GCN FEATURE MATRIX ===")

# Define output directory
output_dir = '/Users/goffy/Desktop/CASA0004/data-preparation'

# Ensure output directory exists
os.makedirs(output_dir, exist_ok=True)

# Save the custom GCN feature matrix
output_file_custom = os.path.join(output_dir, 'gcn_feature_matrix_custom.csv')
gcn_feature_matrix_custom.to_csv(output_file_custom, index=False)
print(f"  Custom GCN feature matrix saved to: {output_file_custom}")
print(f"   Shape: {gcn_feature_matrix_custom.shape}")
print(f"   Features: {len(gcn_feature_matrix_custom.columns) - 1}")

# Create a version with spatial geometry
print(f"\n  Creating spatial version...")
gcn_custom_with_spatial = lsoa_gdf.merge(gcn_feature_matrix_custom, left_on=shp_code, right_

# Convert geometry to WKT and save
gcn_custom_spatial_export = gcn_custom_with_spatial.copy()
gcn_custom_spatial_export['geometry_wkt'] = gcn_custom_spatial_export.geometry.to_wkt()
```

```
gcn_custom_spatial_export = gcn_custom_spatial_export.drop('geometry', axis=1)

output_file_custom_spatial = os.path.join(output_dir, 'gcn_feature_matrix_custom_with_geometi
gcn_custom_spatial_export.to_csv(output_file_custom_spatial, index=False)
print(f"  Custom spatial GCN matrix saved to: {output_file_custom_spatial}")
print(f"   Shape: {gcn_custom_spatial_export.shape}")

# Create feature documentation for custom matrix
feature_info_custom = {
    'Feature_Name': [],
    'Data_Type': [],
    'Description': [],
    'Category': [],
    'Missing_Before_Imputation': [],
    'Imputation_Method': []
}

# Enhanced feature descriptions
feature_descriptions_detailed = {
    'AvgPrice': 'Average housing price in March 2023 (£)',
    'MEAN_PTAL_2023': 'Public Transport Accessibility Level (0-6 scale)',
    'Population': 'Total population count in LSOA',
    'Area_km2': 'Geographic area in square kilometers',
    'MeanSentiment': 'Average sentiment score from online reviews (-1 to 1)',
    'SentimentSD': 'Standard deviation of sentiment scores',
    'ReviewCount': 'Total number of reviews analyzed',
    'NearestStation_m': 'Distance to nearest transport station (meters)',
    'StationsWithin500m': 'Number of transport stations within 500m radius',
    'NearestRail_m': 'Distance to nearest rail line (meters)',
    'StreetLength_m': 'Total length of street network (meters)',
    'StreetDensity_m_per_m2': 'Street network density (meters per square meter)',
    'StreetSegments': 'Number of individual street segments',
    'LandUse_Diversity': 'Number of different land use types',
    'LandUse_Area': 'Total area covered by land use polygons (square meters)'
}

# Categorize features
feature_categories_custom = {
    'Economic': ['AvgPrice'],
    'Transport_Accessibility': ['MEAN_PTAL_2023', 'NearestStation_m', 'StationsWithin500m',
    'Demographics': ['Population'],
    'Geographic': ['Area_km2'],
```

```
    'Social': ['MeanSentiment', 'SentimentSD', 'ReviewCount'],
    'Urban_Form': ['StreetLength_m', 'StreetDensity_m_per_m2', 'StreetSegments', 'LandUse_Di
}

# Build documentation
feature_cols_custom = [col for col in gcn_feature_matrix_custom.columns if col != 'LSOA_CODE
for feature in feature_cols_custom:
    feature_info_custom['Feature_Name'].append(feature)
    feature_info_custom['Data_Type'].append('float64 (standardized)')
    feature_info_custom['Description'].append(feature_descriptions_detailed.get(feature, 'Fea

    # Determine category
    category = 'Other'
    for cat, cat_features in feature_categories_custom.items():
        if feature in cat_features:
            category = cat
            break
    feature_info_custom['Category'].append(category)

    # Check original missing values
    original_missing = df_combined[feature].isnull().sum() if feature in df_combined.columns
    feature_info_custom['Missing_Before_Imputation'].append(original_missing)
    feature_info_custom['Imputation_Method'].append('Median' if original_missing > 0 else 'No

# Save custom feature documentation
feature_doc_custom_df = pd.DataFrame(feature_info_custom)
doc_file_custom = os.path.join(output_dir, 'gcn_feature_matrix_custom_documentation.csv')
feature_doc_custom_df.to_csv(doc_file_custom, index=False)
print(f"  Custom feature documentation saved to: {doc_file_custom}")

# Save summary statistics
summary_file_custom = os.path.join(output_dir, 'gcn_feature_matrix_custom_summary_stats.csv')
summary_stats_custom = gcn_feature_matrix_custom[feature_cols_custom].describe()
summary_stats_custom.to_csv(summary_file_custom)
print(f"  Custom summary statistics saved to: {summary_file_custom}")

print(f"\n  CUSTOM GCN FILES SAVED TO: {output_dir}")
print(f"   1. gcn_feature_matrix_custom.csv - Main custom feature matrix")
print(f"   2. gcn_feature_matrix_custom_with_geometry.csv - With spatial geometry")
print(f"   3. gcn_feature_matrix_custom_documentation.csv - Feature descriptions")
print(f"   4. gcn_feature_matrix_custom_summary_stats.csv - Statistical summaries")
```

```
# Verification
print(f"\n VERIFICATION:")
custom_files = [
    'gcn_feature_matrix_custom.csv',
    'gcn_feature_matrix_custom_with_geometry.csv',
    'gcn_feature_matrix_custom_documentation.csv',
    'gcn_feature_matrix_custom_summary_stats.csv'
]

for filename in custom_files:
    filepath = os.path.join(output_dir, filename)
    if os.path.exists(filepath):
        size_mb = os.path.getsize(filepath) / (1024 * 1024)
        print(f"    {filename} ({size_mb:.2f} MB)")
    else:
        print(f"    {filename} - Not found!")

print(f"\n CUSTOM GCN FEATURE MATRIX SUCCESSFULLY CREATED AND SAVED!")
print(f"\n FINAL SUMMARY:")
print(f"   • Selected Features: {len(feature_cols_custom)}")
print(f"   • Total LSOAs: {len(gcn_feature_matrix_custom):,}")
print(f"   • Data Completeness: 100% (after imputation)")
print(f"   • Standardization: Applied (mean 0, std 1)")
print(f"   • Ready for GCN: Yes")

print(f"\n FEATURE LIST:")
for i, feature in enumerate(feature_cols_custom, 1):
    category = 'Other'
    for cat, cat_features in feature_categories_custom.items():
        if feature in cat_features:
            category = cat
            break
    print(f"   {i:2d}. {feature} ({category})")

print(f"\n USAGE:")
print(f"   import pandas as pd")
print(f"   gcn_data = pd.read_csv('{output_file_custom}')")
print(f"   # Features ready for GCN model input")
```

```
=== SAVING CUSTOM GCN FEATURE MATRIX ===
  Custom GCN feature matrix saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feat
    Shape: (4719, 16)
```

```
  Features: 15

Creating spatial version...
Custom spatial GCN matrix saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feat
  Shape: (4719, 23)
Custom feature documentation saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_f
Custom summary statistics saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feat

CUSTOM GCN FILES SAVED TO: /Users/goffy/Desktop/CASA0004/data-preparation
  1. gcn_feature_matrix_custom.csv - Main custom feature matrix
  2. gcn_feature_matrix_custom_with_geometry.csv - With spatial geometry
  3. gcn_feature_matrix_custom_documentation.csv - Feature descriptions
  4. gcn_feature_matrix_custom_summary_stats.csv - Statistical summaries

VERIFICATION:
   gcn_feature_matrix_custom.csv (1.31 MB)
   gcn_feature_matrix_custom_with_geometry.csv (56.54 MB)
   gcn_feature_matrix_custom_documentation.csv (0.00 MB)
   gcn_feature_matrix_custom_summary_stats.csv (0.00 MB)

CUSTOM GCN FEATURE MATRIX SUCCESSFULLY CREATED AND SAVED!

FINAL SUMMARY:
  • Selected Features: 15
  • Total LSOAs: 4,719
  • Data Completeness: 100% (after imputation)
  • Standardization: Applied (mean 0, std 1)
  • Ready for GCN: Yes

FEATURE LIST:
   1. AvgPrice (Economic)
   2. MEAN_PTAL_2023 (Transport_Accessibility)
   3. Population (Demographics)
   4. Area_km2 (Geographic)
   5. MeanSentiment (Social)
   6. SentimentSD (Social)
   7. ReviewCount (Social)
   8. NearestStation_m (Transport_Accessibility)
   9. StationsWithin500m (Transport_Accessibility)
  10. NearestRail_m (Transport_Accessibility)
  11. StreetLength_m (Urban_Form)
  12. StreetDensity_m_per_m2 (Urban_Form)
  13. StreetSegments (Urban_Form)
```

```
  14. LandUse_Diversity (Urban_Form)
  15. LandUse_Area (Urban_Form)

USAGE:
  import pandas as pd
  gcn_data = pd.read_csv('/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix_
  # Features ready for GCN model input
Custom spatial GCN matrix saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feat
  Shape: (4719, 23)
Custom feature documentation saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_f
Custom summary statistics saved to: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feat

CUSTOM GCN FILES SAVED TO: /Users/goffy/Desktop/CASA0004/data-preparation
  1. gcn_feature_matrix_custom.csv - Main custom feature matrix
  2. gcn_feature_matrix_custom_with_geometry.csv - With spatial geometry
  3. gcn_feature_matrix_custom_documentation.csv - Feature descriptions
  4. gcn_feature_matrix_custom_summary_stats.csv - Statistical summaries

VERIFICATION:
   gcn_feature_matrix_custom.csv (1.31 MB)
   gcn_feature_matrix_custom_with_geometry.csv (56.54 MB)
   gcn_feature_matrix_custom_documentation.csv (0.00 MB)
   gcn_feature_matrix_custom_summary_stats.csv (0.00 MB)

CUSTOM GCN FEATURE MATRIX SUCCESSFULLY CREATED AND SAVED!

FINAL SUMMARY:
  • Selected Features: 15
  • Total LSOAs: 4,719
  • Data Completeness: 100% (after imputation)
  • Standardization: Applied (mean 0, std 1)
  • Ready for GCN: Yes

FEATURE LIST:
   1. AvgPrice (Economic)
   2. MEAN_PTAL_2023 (Transport_Accessibility)
   3. Population (Demographics)
   4. Area_km2 (Geographic)
   5. MeanSentiment (Social)
   6. SentimentSD (Social)
   7. ReviewCount (Social)
   8. NearestStation_m (Transport_Accessibility)
   9. StationsWithin500m (Transport_Accessibility)
```

```
  10. NearestRail_m (Transport_Accessibility)
  11. StreetLength_m (Urban_Form)
  12. StreetDensity_m_per_m2 (Urban_Form)
  13. StreetSegments (Urban_Form)
  14. LandUse_Diversity (Urban_Form)
  15. LandUse_Area (Urban_Form)

 USAGE:
  import pandas as pd
  gcn_data = pd.read_csv('/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix_
  # Features ready for GCN model input
```

```
# ---------------------------------
# 3. Comprehensive Input Data Check
# ---------------------------------

# Dictionary of all loaded datasets for easy iteration
datasets = {
    "Merged Tabular Data (df_tab)": df_tab,
    "LSOA Polygons (lsoa_gdf)": lsoa_gdf,
    "Street Network (street_gdf)": street_gdf,
    "Stations (station_gdf)": station_gdf,
    "Land Use (landuse_gdf)": landuse_gdf,
    "Rail Network (rail_gdf)": rail_gdf
}

# Loop through and print a summary of each dataset
for name, data in datasets.items():
    print(f"--- Checking: {name} ---\n")

    if isinstance(data, pd.DataFrame):
        print(f"Shape: {data.shape}")
        print(f"Columns: {data.columns.tolist()}")

        # Check for missing values
        missing_values = data.isnull().sum().sum()
        print(f"Total Missing Values: {missing_values}")

        # Display info for GeoDataFrames
        if isinstance(data, gpd.GeoDataFrame):
            print(f"CRS: {data.crs}")
            print(f"Geometry Type: {data.geom_type.unique()}")
```

```
        print("\nHead:")
        print(data.head())

    else:
        print(f"'{name}' is not a DataFrame or GeoDataFrame.")

    print("\n" + "="*50 + "\n")
```

```
# Update datasets dictionary for London-only data
print("=== UPDATED LONDON DATASETS SUMMARY ===")

# Updated dictionary of all London-only datasets
london_datasets = {
    "London Tabular Data (df_tab)": df_tab,
    "London LSOA Polygons (lsoa_gdf)": lsoa_gdf,
    "London Street Network (street_gdf)": street_gdf,
    "London Stations (station_gdf)": station_gdf,
    "London Land Use (landuse_gdf)": landuse_gdf,
    "London Rail Network (rail_gdf)": rail_gdf
}

# Quick summary of each London dataset
for name, data in london_datasets.items():
    if isinstance(data, pd.DataFrame):
        print(f"{name}: {data.shape}")
        if isinstance(data, gpd.GeoDataFrame):
            print(f"  - CRS: {data.crs}")
            print(f"  - Geometry Type: {data.geom_type.unique()}")
    else:
        print(f"{name}: Not a DataFrame")

print(f"\n  All datasets now contain only London LSOA areas")
print(f"  Ready for spatial analysis and feature engineering")
```

```
=== UPDATED LONDON DATASETS SUMMARY ===
London Tabular Data (df_tab): (4719, 10)
London LSOA Polygons (lsoa_gdf): (4719, 7)
  - CRS: EPSG:27700
  - Geometry Type: ['Polygon' 'MultiPolygon']
London Street Network (street_gdf): (115305, 2)
  - CRS: EPSG:27700
```

```
  - Geometry Type: ['LineString']
London Stations (station_gdf): (21002, 5)
  - CRS: EPSG:27700
  - Geometry Type: ['Point']
London Land Use (landuse_gdf): (30775, 5)
  - CRS: EPSG:27700
  - Geometry Type: ['Polygon' 'MultiPolygon']
London Rail Network (rail_gdf): (11777, 5)
  - CRS: EPSG:27700
  - Geometry Type: ['LineString']

 All datasets now contain only London LSOA areas
 Ready for spatial analysis and feature engineering
```

```
# ============================================================================
# SPATIAL CORRELATION-BASED IMPUTATION FOR CUSTOM GCN FEATURE MATRIX
# ============================================================================

print(" Starting spatial correlation-based imputation for custom GCN feature matrix...")

# First, let's check which features have missing values in our custom matrix
missing_in_custom = gcn_custom.isnull().sum()
features_with_missing = missing_in_custom[missing_in_custom > 0]

print(f"\n Features with missing values in custom GCN matrix:")
for feature, count in features_with_missing.items():
    pct = (count / len(gcn_custom)) * 100
    print(f"  - {feature}: {count} missing ({pct:.1f}%)")

if len(features_with_missing) == 0:
    print(" No missing values found in custom GCN matrix!")
    gcn_custom_spatial_imputed = gcn_custom.copy()
else:
    print(f"\n Applying spatial correlation-based imputation to {len(features_with_missing)}

    # Create a copy of the custom matrix for spatial imputation
    gcn_custom_spatial_imputed = gcn_custom.copy()

    # Get the spatial data (LSOA polygons) and merge with our data
    spatial_data = lsoa_gdf[['code', 'geometry']].copy()

    # Merge spatial data with our feature matrix
```

```
gcn_with_geometry = pd.merge(gcn_custom_spatial_imputed,
                             spatial_data,
                             left_on='LSOA_CODE',
                             right_on='code',
                             how='left')
gcn_with_geometry = gpd.GeoDataFrame(gcn_with_geometry, geometry='geometry')

# For each feature with missing values, perform spatial imputation
imputation_results = {}

for feature in features_with_missing.index:
    print(f"\n    Processing feature: {feature}")

    # Get indices of missing values
    missing_mask = gcn_with_geometry[feature].isnull()
    missing_indices = gcn_with_geometry[missing_mask].index

    print(f"    - Missing values: {missing_mask.sum()}")

    if missing_mask.sum() > 0:
        # For each missing value, find spatial neighbors and impute
        imputed_values = []
        methods_used = []

        for idx in missing_indices:
            missing_geometry = gcn_with_geometry.loc[idx, 'geometry']

            if missing_geometry is None or pd.isna(missing_geometry):
                # If no geometry, use median
                fill_value = gcn_with_geometry[feature].median()
                method_used = 'median_fallback'
            else:
                # Find spatial neighbors using different distance thresholds
                distances = [500, 1000, 2000, 5000]  # meters
                fill_value = None
                method_used = 'median_fallback'

                for distance in distances:
                    # Create buffer around missing point
                    buffer = missing_geometry.buffer(distance)

                    # Find LSOAs that intersect with the buffer
```

```
                    intersecting = gcn_with_geometry[gcn_with_geometry.geometry.intersect

                    # Get non-missing values for this feature in neighboring areas
                    neighbor_values = intersecting[feature].dropna()

                    # Remove the current point itself
                    neighbor_values = neighbor_values[neighbor_values.index != idx]

                    if len(neighbor_values) >= 3:  # Need at least 3 neighbors
                        # Use distance-weighted average
                        weights = []
                        values = []

                        for neighbor_idx in neighbor_values.index:
                            neighbor_geom = gcn_with_geometry.loc[neighbor_idx, 'geometry
                            if neighbor_geom is not None:
                                dist = missing_geometry.distance(neighbor_geom)
                                if dist > 0:  # Avoid division by zero
                                    weight = 1 / (dist + 1)  # Add 1 to avoid infinite we
                                    weights.append(weight)
                                    values.append(neighbor_values.loc[neighbor_idx])

                        if len(values) > 0:
                            weights = np.array(weights)
                            values = np.array(values)
                            fill_value = np.average(values, weights=weights)
                            method_used = f'spatial_weighted_{distance}m'
                            break
                    elif len(neighbor_values) >= 1:
                        # Use simple average if we have some neighbors
                        fill_value = neighbor_values.mean()
                        method_used = f'spatial_mean_{distance}m'
                        break

                # If no spatial neighbors found, use median
                if fill_value is None:
                    fill_value = gcn_with_geometry[feature].median()
                    method_used = 'median_fallback'

            imputed_values.append(fill_value)
            methods_used.append(method_used)
```

```
            # Apply the imputed values
            gcn_custom_spatial_imputed.loc[gcn_custom_spatial_imputed[feature].isnull(), feat

            # Store results
            method_counts = pd.Series(methods_used).value_counts()
            imputation_results[feature] = {
                'total_imputed': len(imputed_values),
                'methods': method_counts.to_dict(),
                'mean_imputed_value': np.mean(imputed_values),
                'original_missing_pct': (missing_mask.sum() / len(gcn_with_geometry)) * 100
            }

            print(f"    - Imputation methods used: {dict(method_counts)}")
            print(f"    - Mean imputed value: {np.mean(imputed_values):.4f}")

print(f"\n Spatial imputation completed!")

# Verify that all missing values have been filled
remaining_missing = gcn_custom_spatial_imputed.isnull().sum().sum()
print(f" Remaining missing values after spatial imputation: {remaining_missing}")

if remaining_missing > 0:
    print("  Some missing values remain. Details:")
    remaining_missing_by_feature = gcn_custom_spatial_imputed.isnull().sum()
    for feature, count in remaining_missing_by_feature[remaining_missing_by_feature > 0].iter
        print(f"  - {feature}: {count} missing")

print(f"\n Imputation summary:")
for feature, results in imputation_results.items():
    print(f"  {feature}:")
    print(f"    - Imputed: {results['total_imputed']} values ({results['original_missing_pct
    print(f"    - Methods: {results['methods']}")
    print(f"    - Mean imputed: {results['mean_imputed_value']:.4f}")
```

```
 Starting spatial correlation-based imputation for custom GCN feature matrix...

 Features with missing values in custom GCN matrix:
  - Area_km2: 4719 missing (100.0%)

 Applying spatial correlation-based imputation to 1 features...

   Processing feature: Area_km2
```

```
   - Missing values: 4719
   - Imputation methods used: {'median_fallback': np.int64(4719)}
   - Mean imputed value: nan

 Spatial imputation completed!
 Remaining missing values after spatial imputation: 4719
  Some missing values remain. Details:
 - Area_km2: 4719 missing

 Imputation summary:
 Area_km2:
   - Imputed: 4719 values (100.0% of data)
   - Methods: {'median_fallback': 4719}
   - Mean imputed: nan
```

```python
# Check the structure of lsoa_gdf to find correct column names
print("  Checking lsoa_gdf structure...")
print(f"Columns in lsoa_gdf: {list(lsoa_gdf.columns)}")
print(f"Shape: {lsoa_gdf.shape}")
print(f"Sample of first few rows:")
print(lsoa_gdf.head())
```

```
  Checking lsoa_gdf structure...
Columns in lsoa_gdf: ['geometry', 'code', 'name', 'label', 'Area', 'Half densi', 'Area2']
Shape: (4719, 7)
Sample of first few rows:
                                            geometry       code  \
0  POLYGON ((551549.998 187364.637, 551528.633 18...  E01000037
1  POLYGON ((544812 174524, 544819.775 174523.378...  E01033729
2  POLYGON ((550920.362 187341.138, 550921.876 18...  E01000038
3  POLYGON ((537938 177696, 537941.714 177678.043...  E01033730
4  POLYGON ((551431.061 186927.155, 551444.481 18...  E01000039

                        name                          label     Area  \
0  Barking and Dagenham 003B  E09000002E02000004E01000037   233488
1             Greenwich 030E  E09000011E02000342E01033729   691074
2  Barking and Dagenham 003C  E09000002E02000004E01000038   214094
3             Greenwich 035D  E09000011E02006928E01033730   187153
4  Barking and Dagenham 003D  E09000002E02000004E01000039  1532166

   Half densi    Area2
0        3764   233488
```

```
1        2034   691074
2        3519   214094
3        4950   187153
4         530  1532166
```

```
# Check the structure of gcn_custom
print("\n  Checking gcn_custom structure...")
print(f"Index name: {gcn_custom.index.name}")
print(f"Columns: {list(gcn_custom.columns)}")
print(f"Shape: {gcn_custom.shape}")
print(f"Index sample: {gcn_custom.index[:5].tolist()}")
print(f"Sample of first few rows:")
print(gcn_custom.head())
```

```
  Checking gcn_custom structure...
Index name: None
Columns: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'Population', 'Area_km2', 'MeanSentiment
Shape: (4719, 16)
Index sample: [0, 1, 2, 3, 4]
Sample of first few rows:
   LSOA_CODE  AvgPrice  MEAN_PTAL_2023  Population  Area_km2  MeanSentiment  \
0  E01000001  837500.0             3.0      1615.0       NaN         0.4270
1  E01000002  850000.0             3.0      1493.0       NaN         0.5240
2  E01000003  540000.0             3.0      1573.0       NaN         0.3170
3  E01000005  530000.0             3.0      1090.0       NaN         0.6750
4  E01000006  241000.0             5.0      1612.0       NaN         0.6075

   SentimentSD  ReviewCount  NearestStation_m  StationsWithin500m  \
0        0.312       1505.0         37.204758                  29
1        0.188       5819.0        195.176808                  30
2        0.127          3.0        120.977313                  28
3        0.100       1067.0         51.625449                  35
4        0.127         77.0        387.054639                  10

   NearestRail_m  StreetLength_m  StreetDensity_m_per_m2  StreetSegments  \
0     224.638155     2099.098049                0.016164              43
1     144.061157     3258.516395                0.014266              55
2     189.829128      578.878566                0.009803              11
3      26.831724     3449.617678                0.018196              70
4     153.355797     2332.482543                0.015917              26
```

```
   LandUse_Diversity   LandUse_Area
0                  4   38011.199255
1                  5   88486.284243
2                  3   28201.764815
3                  4  128388.326345
4                  1  118840.250570
```

```
# ============================================================================
# FIX AREA_KM2 FEATURE USING SPATIAL DATA
# ============================================================================

print("  Fixing Area_km2 feature using spatial geometry...")

# Check the area columns in lsoa_gdf
print("Available area-related columns in lsoa_gdf:")
for col in lsoa_gdf.columns:
    if 'area' in col.lower() or 'Area' in col:
        print(f"  - {col}: {lsoa_gdf[col].describe()}")

# Let's use the existing 'Area' column and convert to km2
# Assuming the 'Area' column is in square meters
print(f"\nUsing 'Area' column from spatial data...")

# Create a mapping from LSOA code to area in km2
area_mapping = lsoa_gdf.set_index('code')['Area'] / 1000000  # Convert m2 to km2

print(f"Area statistics (km2):")
print(area_mapping.describe())

# Update the Area_km2 column in gcn_custom_spatial_imputed
print(f"\nUpdating Area_km2 in gcn_custom_spatial_imputed...")
gcn_custom_spatial_imputed['Area_km2'] = gcn_custom_spatial_imputed['LSOA_CODE'].map(area_ma

# Check if this fixed the missing values
area_missing_after = gcn_custom_spatial_imputed['Area_km2'].isnull().sum()
print(f"Missing values in Area_km2 after fix: {area_missing_after}")

if area_missing_after > 0:
    print("Some areas are still missing. Using geometry to calculate area...")

    # For remaining missing values, calculate from geometry
    missing_mask = gcn_custom_spatial_imputed['Area_km2'].isnull()
```

```
    if missing_mask.sum() > 0:
        # Merge with geometry for missing areas
        missing_codes = gcn_custom_spatial_imputed.loc[missing_mask, 'LSOA_CODE']

        for lsoa_code in missing_codes:
            geom = lsoa_gdf[lsoa_gdf['code'] == lsoa_code]['geometry']
            if not geom.empty:
                area_m2 = geom.iloc[0].area
                area_km2 = area_m2 / 1000000
                gcn_custom_spatial_imputed.loc[gcn_custom_spatial_imputed['LSOA_CODE'] == lsc

# Final check
final_missing = gcn_custom_spatial_imputed['Area_km2'].isnull().sum()
print(f"\nFinal missing values in Area_km2: {final_missing}")
print(f"Area_km2 statistics after fix:")
print(gcn_custom_spatial_imputed['Area_km2'].describe())
```

```
 Fixing Area_km2 feature using spatial geometry...
Available area-related columns in lsoa_gdf:
  - Area: count    4.719000e+03
mean     2.883627e+05
std      3.892889e+05
min      1.836900e+04
25%      1.342505e+05
50%      2.021800e+05
75%      3.110385e+05
max      9.936120e+06
Name: Area, dtype: float64
  - Area2: count       4719
unique      4686
top       237761
freq           2
Name: Area2, dtype: object

Using 'Area' column from spatial data...
Area statistics (km2):
count    4719.000000
mean        0.288363
std         0.389289
min         0.018369
25%         0.134250
```

```
50%         0.202180
75%         0.311038
max         9.936120
Name: Area, dtype: float64

Updating Area_km2 in gcn_custom_spatial_imputed...
Missing values in Area_km2 after fix: 0

Final missing values in Area_km2: 0
Area_km2 statistics after fix:
count    4719.000000
mean        0.288363
std         0.389289
min         0.018369
25%         0.134250
50%         0.202180
75%         0.311038
max         9.936120
Name: Area_km2, dtype: float64
```

```python
# ============================================================================
# FINAL CHECK AND SPATIAL IMPUTATION FOR ANY REMAINING MISSING VALUES
# ============================================================================

print("  Final check for missing values in custom GCN matrix...")

# Check for missing values in all features
missing_summary = gcn_custom_spatial_imputed.isnull().sum()
features_with_missing = missing_summary[missing_summary > 0]

if len(features_with_missing) == 0:
    print("  No missing values found! All features are complete.")
else:
    print(f"  Features still with missing values:")
    for feature, count in features_with_missing.items():
        pct = (count / len(gcn_custom_spatial_imputed)) * 100
        print(f"  - {feature}: {count} missing ({pct:.1f}%)")

    print(f"\n  Applying spatial imputation to remaining {len(features_with_missing)} feature

    # Apply spatial imputation for any remaining missing values
    # Merge with spatial data for spatial operations
```

```
gcn_with_geometry = pd.merge(gcn_custom_spatial_imputed,
                             lsoa_gdf[['code', 'geometry']],
                             left_on='LSOA_CODE',
                             right_on='code',
                             how='left')
gcn_with_geometry = gpd.GeoDataFrame(gcn_with_geometry, geometry='geometry')

spatial_imputation_results = {}

for feature in features_with_missing.index:
    if feature == 'LSOA_CODE':  # Skip the identifier column
        continue

    print(f"\n    Processing feature: {feature}")

    # Get indices of missing values
    missing_mask = gcn_with_geometry[feature].isnull()
    missing_indices = gcn_with_geometry[missing_mask].index

    print(f"    - Missing values: {missing_mask.sum()}")

    if missing_mask.sum() > 0:
        imputed_values = []
        methods_used = []

        for idx in missing_indices:
            missing_geometry = gcn_with_geometry.loc[idx, 'geometry']

            if missing_geometry is None or pd.isna(missing_geometry):
                # If no geometry, use median of non-missing values
                fill_value = gcn_with_geometry[feature].median()
                method_used = 'median_fallback'
            else:
                # Find spatial neighbors using different distance thresholds
                distances = [500, 1000, 2000, 5000, 10000]  # meters
                fill_value = None
                method_used = 'median_fallback'

                for distance in distances:
                    # Create buffer around missing point
                    buffer = missing_geometry.buffer(distance)
```

```
                    # Find LSOAs that intersect with the buffer
                    intersecting = gcn_with_geometry[gcn_with_geometry.geometry.intersect

                    # Get non-missing values for this feature in neighboring areas
                    neighbor_values = intersecting[feature].dropna()

                    # Remove the current point itself
                    neighbor_values = neighbor_values[neighbor_values.index != idx]

                    if len(neighbor_values) >= 3:  # Need at least 3 neighbors
                        # Use distance-weighted average
                        weights = []
                        values = []

                        for neighbor_idx in neighbor_values.index:
                            neighbor_geom = gcn_with_geometry.loc[neighbor_idx, 'geometry
                            if neighbor_geom is not None:
                                dist = missing_geometry.distance(neighbor_geom)
                                if dist > 0:  # Avoid division by zero
                                    weight = 1 / (dist + 100)  # Add 100 to reduce weight
                                    weights.append(weight)
                                    values.append(neighbor_values.loc[neighbor_idx])

                        if len(values) > 0:
                            weights = np.array(weights)
                            values = np.array(values)
                            fill_value = np.average(values, weights=weights)
                            method_used = f'spatial_weighted_{distance}m'
                            break
                    elif len(neighbor_values) >= 1:
                        # Use simple average if we have some neighbors
                        fill_value = neighbor_values.mean()
                        method_used = f'spatial_mean_{distance}m'
                        break

                # If no spatial neighbors found, use median
                if fill_value is None or np.isnan(fill_value):
                    fill_value = gcn_with_geometry[feature].median()
                    method_used = 'median_fallback'

            imputed_values.append(fill_value)
            methods_used.append(method_used)
```

```
            # Apply the imputed values
            gcn_custom_spatial_imputed.loc[gcn_custom_spatial_imputed[feature].isnull(), feat

            # Store results
            method_counts = pd.Series(methods_used).value_counts()
            spatial_imputation_results[feature] = {
                'total_imputed': len(imputed_values),
                'methods': method_counts.to_dict(),
                'mean_imputed_value': np.mean(imputed_values),
                'original_missing_pct': (missing_mask.sum() / len(gcn_with_geometry)) * 100
            }

            print(f"    - Imputation methods used: {dict(method_counts)}")
            print(f"    - Mean imputed value: {np.mean(imputed_values):.4f}")

# Final verification
final_missing = gcn_custom_spatial_imputed.isnull().sum().sum()
print(f"\n Spatial imputation completed!")
print(f" Total remaining missing values: {final_missing}")

if final_missing > 0:
    print("  Some missing values remain:")
    remaining = gcn_custom_spatial_imputed.isnull().sum()
    for feature, count in remaining[remaining > 0].items():
        print(f"  - {feature}: {count} missing")
else:
    print(" All missing values have been successfully imputed!")

print(f"\n Final dataset shape: {gcn_custom_spatial_imputed.shape}")
print(f" Final dataset info:")
print(gcn_custom_spatial_imputed.dtypes)
```

```
 Final check for missing values in custom GCN matrix...
 No missing values found! All features are complete.

 Spatial imputation completed!
 Total remaining missing values: 0
 All missing values have been successfully imputed!

 Final dataset shape: (4719, 16)
 Final dataset info:
```

```
LSOA_CODE                  object
AvgPrice                  float64
MEAN_PTAL_2023            float64
Population                float64
Area_km2                  float64
MeanSentiment             float64
SentimentSD               float64
ReviewCount               float64
NearestStation_m          float64
StationsWithin500m          int64
NearestRail_m             float64
StreetLength_m            float64
StreetDensity_m_per_m2    float64
StreetSegments              int64
LandUse_Diversity           int64
LandUse_Area              float64
dtype: object
```

```python
# ============================================================================
# STANDARDIZE SPATIALLY-IMPUTED FEATURES AND SAVE FINAL GCN MATRIX
# ============================================================================

print("  Standardizing spatially-imputed features...")

# Separate identifier column from features
feature_columns = [col for col in gcn_custom_spatial_imputed.columns if col != 'LSOA_CODE']
identifier_column = gcn_custom_spatial_imputed[['LSOA_CODE']]

# Standardize the features (mean=0, std=1)
scaler_spatial = StandardScaler()
features_scaled = scaler_spatial.fit_transform(gcn_custom_spatial_imputed[feature_columns])

# Create the standardized dataframe
gcn_custom_spatial_scaled = pd.DataFrame(
    features_scaled,
    columns=feature_columns,
    index=gcn_custom_spatial_imputed.index
)

# Add back the identifier column
gcn_custom_spatial_scaled = pd.concat([identifier_column, gcn_custom_spatial_scaled], axis=1)
```

```
print(f"  Standardization completed!")
print(f"  Standardized feature statistics:")
print(gcn_custom_spatial_scaled[feature_columns].describe())

# Verify standardization (should be close to mean=0, std=1)
means = gcn_custom_spatial_scaled[feature_columns].mean()
stds = gcn_custom_spatial_scaled[feature_columns].std()

print(f"\n  Standardization verification:")
print(f"  Mean of features (should be ~0): {means.abs().max():.6f}")
print(f"  Std of features (should be ~1): {stds.min():.6f} to {stds.max():.6f}")

# ============================================================================
# SAVE SPATIALLY-IMPUTED AND STANDARDIZED GCN MATRICES
# ============================================================================

print(f"\n  Saving spatially-imputed GCN matrices...")

# Define output paths
output_file_spatial_imputed = "/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_mat
output_file_spatial_scaled = "/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matr
output_file_spatial_with_geometry = "/Users/goffy/Desktop/CASA0004/data-preparation/gcn_featu

# Save the spatially-imputed (unscaled) matrix
gcn_custom_spatial_imputed.to_csv(output_file_spatial_imputed, index=False)
print(f"  Saved spatially-imputed matrix: {output_file_spatial_imputed}")

# Save the spatially-imputed and scaled matrix
gcn_custom_spatial_scaled.to_csv(output_file_spatial_scaled, index=False)
print(f"  Saved spatially-imputed & scaled matrix: {output_file_spatial_scaled}")

# Create and save matrix with geometry for spatial analysis
gcn_spatial_with_geometry = pd.merge(
    gcn_custom_spatial_scaled,
    lsoa_gdf[['code', 'geometry']],
    left_on='LSOA_CODE',
    right_on='code',
    how='left'
).drop('code', axis=1)

# Convert geometry to WKT for CSV export
gcn_spatial_with_geometry['geometry_wkt'] = gcn_spatial_with_geometry['geometry'].apply(
```

```
    lambda x: x.wkt if x is not None else None
)
gcn_spatial_export = gcn_spatial_with_geometry.drop('geometry', axis=1)
gcn_spatial_export.to_csv(output_file_spatial_with_geometry, index=False)
print(f" Saved spatial matrix with geometry: {output_file_spatial_with_geometry}")

print(f"\n Summary of saved files:")
print(f"  1. Spatially-imputed matrix: {output_file_spatial_imputed}")
print(f"     - Shape: {gcn_custom_spatial_imputed.shape}")
print(f"     - Features: standardized=No, imputed=Yes")
print(f"  2. Spatially-imputed & scaled matrix: {output_file_spatial_scaled}")
print(f"     - Shape: {gcn_custom_spatial_scaled.shape}")
print(f"     - Features: standardized=Yes, imputed=Yes")
print(f"  3. Spatial matrix with geometry: {output_file_spatial_with_geometry}")
print(f"     - Shape: {gcn_spatial_export.shape}")
print(f"     - Features: standardized=Yes, imputed=Yes, geometry=WKT")
```

```
 Standardizing spatially-imputed features...
 Standardization completed!
 Standardized feature statistics:
           AvgPrice  MEAN_PTAL_2023    Population      Area_km2  \
count  4.719000e+03    4.719000e+03  4.719000e+03  4.719000e+03
mean   1.144337e-16    9.636519e-17  5.149515e-16  1.204565e-17
std    1.000106e+00    1.000106e+00  1.000106e+00  1.000106e+00
min   -1.161581e+00   -1.109434e+00 -6.525909e+00 -6.936297e-01
25%   -4.890309e-01   -1.109434e+00 -2.499702e-01 -3.959233e-01
50%   -2.409056e-01    1.058443e-01  6.037845e-02 -2.214084e-01
75%    1.371589e-01    1.058443e-01  3.577959e-01  5.825543e-02
max    1.561299e+01    2.536401e+00  1.937958e+01  2.478565e+01

       MeanSentiment   SentimentSD   ReviewCount  NearestStation_m  \
count   4.719000e+03  4.719000e+03  4.719000e+03      4.719000e+03
mean   -8.842259e-16  3.199626e-17  3.011412e-18      2.107989e-16
std     1.000106e+00  1.000106e+00  1.000106e+00      1.000106e+00
min    -5.061611e+00 -1.916535e+00 -1.028478e-01     -1.551074e+00
25%    -1.788546e-02 -1.611503e-01 -9.689534e-02     -7.001013e-01
50%     1.141716e-01 -1.611503e-01 -9.491118e-02     -1.655972e-01
75%     2.377088e-01 -1.611503e-01 -9.156945e-02      4.555854e-01
max     3.304839e+00  6.321332e+00  4.418620e+01      8.275955e+00

       StationsWithin500m  NearestRail_m  StreetLength_m  \
count        4.719000e+03   4.719000e+03    4.719000e+03
```

```
mean        -6.022825e-17  -4.818260e-17    1.806847e-16
std          1.000106e+00   1.000106e+00    1.000106e+00
min         -2.009234e+00  -1.243591e+00   -1.919367e+00
25%         -7.511745e-01  -6.965832e-01   -5.772423e-01
50%         -5.225253e-02  -2.548129e-01   -1.415846e-01
75%          6.466694e-01   4.091435e-01    3.921599e-01
max          6.098261e+00   8.730737e+00    1.962265e+01

       StreetDensity_m_per_m2  StreetSegments  LandUse_Diversity  LandUse_Area
count            4.719000e+03    4.719000e+03       4.719000e+03  4.719000e+03
mean             6.022825e-17    7.528531e-18       1.445478e-16  4.215977e-17
std              1.000106e+00    1.000106e+00       1.000106e+00  1.000106e+00
min             -2.527152e+00   -2.034934e+00      -1.983594e+00 -6.596514e-01
25%             -6.859315e-01   -5.966231e-01      -8.417118e-01 -3.708117e-01
50%             -6.486071e-02   -1.363637e-01      -2.707706e-01 -2.017019e-01
75%              6.331294e-01    3.814282e-01       3.001706e-01  6.627842e-02
max              4.731996e+00    2.615596e+01       4.867700e+00  2.226282e+01

 Standardization verification:
  Mean of features (should be ~0): 0.000000
  Std of features (should be ~1): 1.000106 to 1.000106

 Saving spatially-imputed GCN matrices...
 Saved spatially-imputed matrix: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_
 Saved spatially-imputed & scaled matrix: /Users/goffy/Desktop/CASA0004/data-preparation/gcn
 Saved spatial matrix with geometry: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feat

 Summary of saved files:
  1. Spatially-imputed matrix: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_mat
     - Shape: (4719, 16)
     - Features: standardized=No, imputed=Yes
  2. Spatially-imputed & scaled matrix: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_fe
     - Shape: (4719, 16)
     - Features: standardized=Yes, imputed=Yes
  3. Spatial matrix with geometry: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature
     - Shape: (4719, 17)
     - Features: standardized=Yes, imputed=Yes, geometry=WKT
```

```
# ============================================================================
# GENERATE DOCUMENTATION AND SUMMARY STATISTICS FOR SPATIAL MATRICES
# ============================================================================
```

```
print("  Generating documentation and summary statistics for spatially-imputed matrices...")

# Create documentation for the spatially-imputed features
feature_documentation_spatial = {
    'Feature_Name': [],
    'Description': [],
    'Data_Type': [],
    'Unit': [],
    'Source': [],
    'Processing': [],
    'Missing_Values_Handled': [],
    'Standardized': []
}

# Feature descriptions and metadata
feature_info_spatial = {
    'LSOA_CODE': {
        'description': 'Lower Layer Super Output Area unique identifier',
        'unit': 'Categorical',
        'source': 'ONS LSOA boundaries',
        'processing': 'No processing - identifier field',
        'missing_handled': 'None (identifier)',
        'standardized': 'No'
    },
    'AvgPrice': {
        'description': 'Average housing price in the LSOA',
        'unit': 'British Pounds (£)',
        'source': 'HM Land Registry',
        'processing': 'Averaged by LSOA, spatial imputation applied',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'MEAN_PTAL_2023': {
        'description': 'Public Transport Accessibility Level (PTAL) mean score',
        'unit': 'Score (0-6, higher = better access)',
        'source': 'Transport for London',
        'processing': 'Mean PTAL score by LSOA, spatial imputation applied',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'Population': {
        'description': 'Total population count in the LSOA',
```

```
        'unit': 'Number of people',
        'source': 'ONS Census/Population estimates',
        'processing': 'Population count by LSOA, spatial imputation applied',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'Area_km2': {
        'description': 'Area of the LSOA in square kilometers',
        'unit': 'Square kilometers (km²)',
        'source': 'ONS LSOA boundaries',
        'processing': 'Calculated from polygon geometry, converted m² to km²',
        'missing_handled': 'Calculated from spatial geometry',
        'standardized': 'Yes'
    },
    'MeanSentiment': {
        'description': 'Mean sentiment score from social media/reviews',
        'unit': 'Sentiment score (-1 to 1, higher = more positive)',
        'source': 'Social media sentiment analysis',
        'processing': 'Mean sentiment by LSOA, spatial imputation applied',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'SentimentSD': {
        'description': 'Standard deviation of sentiment scores',
        'unit': 'Standard deviation (0-1)',
        'source': 'Social media sentiment analysis',
        'processing': 'Standard deviation of sentiment by LSOA, spatial imputation applied',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'ReviewCount': {
        'description': 'Number of reviews/social media posts',
        'unit': 'Count',
        'source': 'Social media sentiment analysis',
        'processing': 'Count of reviews by LSOA, spatial imputation applied',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'NearestStation_m': {
        'description': 'Distance to nearest railway/underground station',
        'unit': 'Meters',
        'source': 'Transport for London station locations',
```

```
        'processing': 'Euclidean distance from LSOA centroid to nearest station',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'StationsWithin500m': {
        'description': 'Number of stations within 500m radius',
        'unit': 'Count',
        'source': 'Transport for London station locations',
        'processing': 'Count of stations within 500m buffer of LSOA centroid',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'NearestRail_m': {
        'description': 'Distance to nearest rail network',
        'unit': 'Meters',
        'source': 'Transport for London rail network',
        'processing': 'Euclidean distance from LSOA centroid to nearest rail line',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'StreetLength_m': {
        'description': 'Total length of street network within LSOA',
        'unit': 'Meters',
        'source': 'OpenStreetMap/Transport for London',
        'processing': 'Sum of street lengths intersecting LSOA polygon',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'StreetDensity_m_per_m2': {
        'description': 'Street network density (street length per unit area)',
        'unit': 'Meters per square meter',
        'source': 'Calculated from street network and LSOA area',
        'processing': 'StreetLength_m / Area_m2',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'StreetSegments': {
        'description': 'Number of street segments within LSOA',
        'unit': 'Count',
        'source': 'OpenStreetMap/Transport for London',
        'processing': 'Count of street segments intersecting LSOA polygon',
        'missing_handled': 'Spatial correlation-based imputation',
```

```
        'standardized': 'Yes'
    },
    'LandUse_Diversity': {
        'description': 'Number of different land use types in LSOA',
        'unit': 'Count',
        'source': 'London Land Use Survey',
        'processing': 'Count of unique land use categories intersecting LSOA',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    },
    'LandUse_Area': {
        'description': 'Total area covered by different land uses',
        'unit': 'Square meters',
        'source': 'London Land Use Survey',
        'processing': 'Sum of land use polygon areas intersecting LSOA',
        'missing_handled': 'Spatial correlation-based imputation',
        'standardized': 'Yes'
    }
}

# Build documentation dataframe
for feature in gcn_custom_spatial_scaled.columns:
    info = feature_info_spatial.get(feature, {})

    feature_documentation_spatial['Feature_Name'].append(feature)
    feature_documentation_spatial['Description'].append(info.get('description', 'No descripti
    feature_documentation_spatial['Data_Type'].append(str(gcn_custom_spatial_scaled[feature]
    feature_documentation_spatial['Unit'].append(info.get('unit', 'Unknown'))
    feature_documentation_spatial['Source'].append(info.get('source', 'Unknown'))
    feature_documentation_spatial['Processing'].append(info.get('processing', 'Standard proce
    feature_documentation_spatial['Missing_Values_Handled'].append(info.get('missing_handled
    feature_documentation_spatial['Standardized'].append(info.get('standardized', 'Yes'))

# Create documentation DataFrame
feature_doc_spatial_df = pd.DataFrame(feature_documentation_spatial)

# Generate summary statistics for both versions
summary_stats_spatial_unscaled = gcn_custom_spatial_imputed.describe().round(4)
summary_stats_spatial_scaled = gcn_custom_spatial_scaled.describe().round(4)

# Save documentation and statistics
doc_file_spatial = "/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix_spatial
```

```
summary_file_spatial_unscaled = "/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_m
summary_file_spatial_scaled = "/Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_mat

# Save files
feature_doc_spatial_df.to_csv(doc_file_spatial, index=False)
summary_stats_spatial_unscaled.to_csv(summary_file_spatial_unscaled)
summary_stats_spatial_scaled.to_csv(summary_file_spatial_scaled)

print(f" Documentation and statistics saved:")
print(f"   Documentation: {doc_file_spatial}")
print(f"   Summary stats (unscaled): {summary_file_spatial_unscaled}")
print(f"   Summary stats (scaled): {summary_file_spatial_scaled}")

# ============================================================================
# FINAL SUMMARY OF ALL CREATED FILES
# ============================================================================

print(f"\n SPATIAL CORRELATION-BASED IMPUTATION COMPLETED!")
print(f"="*80)
print(f" All generated files:")
print(f"")
print(f" CUSTOM GCN FEATURE MATRICES (Spatially Imputed):")
print(f"  1. gcn_feature_matrix_custom_spatial_imputed.csv")
print(f"     - Raw features with spatial imputation (no standardization)")
print(f"     - Shape: {gcn_custom_spatial_imputed.shape}")
print(f"     - Missing values: 0")
print(f"")
print(f"  2. gcn_feature_matrix_custom_spatial_imputed_scaled.csv")
print(f"     - Spatially imputed + standardized features (mean 0, std 1)")
print(f"     - Shape: {gcn_custom_spatial_scaled.shape}")
print(f"     - Missing values: 0")
print(f"")
print(f"  3. gcn_feature_matrix_custom_spatial_imputed_with_geometry.csv")
print(f"     - Spatially imputed + standardized + geometry (WKT format)")
print(f"     - Shape: {gcn_spatial_export.shape}")
print(f"     - For spatial analysis and visualization")
print(f"")
print(f" DOCUMENTATION:")
print(f"  4. gcn_feature_matrix_spatial_imputed_documentation.csv")
print(f"     - Complete feature documentation with imputation methods")
print(f"")
print(f" SUMMARY STATISTICS:")
```

```
print(f"  5. gcn_feature_matrix_spatial_imputed_summary_stats.csv")
print(f"     - Statistical summary of unscaled spatially-imputed features")
print(f"")
print(f"  6. gcn_feature_matrix_spatial_imputed_scaled_summary_stats.csv")
print(f"     - Statistical summary of scaled spatially-imputed features")
print(f"")
print(f"  RECOMMENDED FOR GCN:")
print(f"   Use: gcn_feature_matrix_custom_spatial_imputed_scaled.csv")
print(f"   - Complete (no missing values)")
print(f"   - Standardized (mean 0, std 1)")
print(f"   - Spatially-imputed using neighbor correlation")
print(f"   - Optimized feature set: {len(feature_columns)} features")
print(f"="*80)
```

```
  Generating documentation and summary statistics for spatially-imputed matrices...
  Documentation and statistics saved:
    Documentation: /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix_spatial_
    Summary stats (unscaled): /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matr
    Summary stats (scaled): /Users/goffy/Desktop/CASA0004/data-preparation/gcn_feature_matrix

  SPATIAL CORRELATION-BASED IMPUTATION COMPLETED!
================================================================================
  All generated files:

  CUSTOM GCN FEATURE MATRICES (Spatially Imputed):
   1. gcn_feature_matrix_custom_spatial_imputed.csv
      - Raw features with spatial imputation (no standardization)
      - Shape: (4719, 16)
      - Missing values: 0

   2. gcn_feature_matrix_custom_spatial_imputed_scaled.csv
      - Spatially imputed + standardized features (mean 0, std 1)
      - Shape: (4719, 16)
      - Missing values: 0

   3. gcn_feature_matrix_custom_spatial_imputed_with_geometry.csv
      - Spatially imputed + standardized + geometry (WKT format)
      - Shape: (4719, 17)
      - For spatial analysis and visualization

  DOCUMENTATION:
   4. gcn_feature_matrix_spatial_imputed_documentation.csv
```

```
    - Complete feature documentation with imputation methods

 SUMMARY STATISTICS:
  5. gcn_feature_matrix_spatial_imputed_summary_stats.csv
     - Statistical summary of unscaled spatially-imputed features

  6. gcn_feature_matrix_spatial_imputed_scaled_summary_stats.csv
     - Statistical summary of scaled spatially-imputed features

 RECOMMENDED FOR GCN:
   Use: gcn_feature_matrix_custom_spatial_imputed_scaled.csv
   - Complete (no missing values)
   - Standardized (mean 0, std 1)
   - Spatially-imputed using neighbor correlation
   - Optimized feature set: 15 features
================================================================================
```

```python
# Check current working directory and verify files were created
import os
import glob

print(" Current working directory:", os.getcwd())
print("\n Files in current directory:")
current_files = [f for f in os.listdir('.') if f.endswith('.csv')]
for f in sorted(current_files):
    print(f"  - {f}")

print("\n Looking for GCN files specifically:")
gcn_files = glob.glob("*gcn*.csv")
for f in sorted(gcn_files):
    file_size = os.path.getsize(f) / (1024*1024)  # Size in MB
    print(f"    {f} ({file_size:.2f} MB)")

# Check if our spatial files exist
spatial_files = [
    "gcn_feature_matrix_custom_spatial_imputed.csv",
    "gcn_feature_matrix_custom_spatial_imputed_scaled.csv",
    "gcn_feature_matrix_custom_spatial_imputed_with_geometry.csv",
    "gcn_feature_matrix_spatial_imputed_documentation.csv",
    "gcn_feature_matrix_spatial_imputed_summary_stats.csv",
    "gcn_feature_matrix_spatial_imputed_scaled_summary_stats.csv"
]
```

```python
print(f"\n Spatial imputation files created:")
for filename in spatial_files:
    if os.path.exists(filename):
        file_size = os.path.getsize(filename) / (1024*1024)  # Size in MB
        print(f"   {filename} ({file_size:.2f} MB)")
    else:
        print(f"   {filename} (missing)")

print(f"\n FINAL RECOMMENDATION:")
print(f"For your GCN model, use: gcn_feature_matrix_custom_spatial_imputed_scaled.csv")
print(f"This file contains:")
print(f"  - All {len(feature_columns)} optimized features")
print(f"  - No missing values (spatially imputed)")
print(f"  - Standardized features (mean 0, std 1)")
print(f"  - Ready for Graph Convolutional Network training")
```

```
Current working directory: /Users/goffy/Desktop/CASA0004/data-preparation

Files in current directory:
 - gcn_feature_matrix_custom_spatial_imputed.csv
 - gcn_feature_matrix_custom_spatial_imputed_scaled.csv
 - gcn_feature_matrix_custom_spatial_imputed_with_geometry.csv
 - gcn_feature_matrix_spatial_imputed_documentation.csv
 - gcn_feature_matrix_spatial_imputed_scaled_summary_stats.csv
 - gcn_feature_matrix_spatial_imputed_summary_stats.csv

Looking for GCN files specifically:
  gcn_feature_matrix_custom_spatial_imputed.csv (0.74 MB)
  gcn_feature_matrix_custom_spatial_imputed_scaled.csv (1.39 MB)
  gcn_feature_matrix_custom_spatial_imputed_with_geometry.csv (58.08 MB)
  gcn_feature_matrix_spatial_imputed_documentation.csv (0.00 MB)
  gcn_feature_matrix_spatial_imputed_scaled_summary_stats.csv (0.00 MB)
  gcn_feature_matrix_spatial_imputed_summary_stats.csv (0.00 MB)

Spatial imputation files created:
  gcn_feature_matrix_custom_spatial_imputed.csv (0.74 MB)
  gcn_feature_matrix_custom_spatial_imputed_scaled.csv (1.39 MB)
  gcn_feature_matrix_custom_spatial_imputed_with_geometry.csv (58.08 MB)
  gcn_feature_matrix_spatial_imputed_documentation.csv (0.00 MB)
  gcn_feature_matrix_spatial_imputed_summary_stats.csv (0.00 MB)
  gcn_feature_matrix_spatial_imputed_scaled_summary_stats.csv (0.00 MB)
```

```
 FINAL RECOMMENDATION:
For your GCN model, use: gcn_feature_matrix_custom_spatial_imputed_scaled.csv
This file contains:
  - All 15 optimized features
  - No missing values (spatially imputed)
  - Standardized features (mean 0, std 1)
  - Ready for Graph Convolutional Network training
```

```python
# ================================================================================
# FINAL VERIFICATION OF THE RECOMMENDED GCN FEATURE MATRIX
# ================================================================================

print("  Final verification of the recommended GCN feature matrix...")

# Load the recommended file to verify
recommended_file = "gcn_feature_matrix_custom_spatial_imputed_scaled.csv"
gcn_final = pd.read_csv(recommended_file)

print(f"\n  Final GCN Feature Matrix Summary:")
print(f"    File: {recommended_file}")
print(f"    Shape: {gcn_final.shape}")
print(f"     Features: {list(gcn_final.columns)}")

print(f"\n  Missing values check:")
missing_check = gcn_final.isnull().sum()
if missing_check.sum() == 0:
    print("    NO missing values - perfect!")
else:
    print(f"     Found missing values:")
    for col, count in missing_check[missing_check > 0].items():
        print(f"    - {col}: {count} missing")

print(f"\n  Feature standardization check (excluding LSOA_CODE):")
numeric_features = [col for col in gcn_final.columns if col != 'LSOA_CODE']
feature_stats = gcn_final[numeric_features].agg(['mean', 'std']).round(6)

print("  Feature means (should be  0):")
for col in numeric_features[:5]:  # Show first 5 features
    mean_val = feature_stats.loc['mean', col]
    print(f"    - {col}: {mean_val}")
```

```
print("  Feature standard deviations (should be   1):")
for col in numeric_features[:5]:  # Show first 5 features
    std_val = feature_stats.loc['std', col]
    print(f"    - {col}: {std_val}")

print(f"\n  Sample data (first 3 rows):")
print(gcn_final.head(3))

print(f"\n  SUCCESS! Your optimal GCN feature matrix is ready!")
print(f"="*60)
print(f"  RECOMMENDED FILE: {recommended_file}")
print(f"  DIMENSIONS: {gcn_final.shape[0]} LSOAs × {gcn_final.shape[1]-1} features")
print(f"  PREPROCESSING: Spatially imputed + Standardized")
print(f"  READY FOR: Graph Convolutional Network training")
print(f"="*60)
```

```
Final verification of the recommended GCN feature matrix...

Final GCN Feature Matrix Summary:
  File: gcn_feature_matrix_custom_spatial_imputed_scaled.csv
  Shape: (4719, 16)
   Features: ['LSOA_CODE', 'AvgPrice', 'MEAN_PTAL_2023', 'Population', 'Area_km2', 'MeanSen

Missing values check:
  NO missing values - perfect!

Feature standardization check (excluding LSOA_CODE):
 Feature means (should be   0):
   - AvgPrice: 0.0
   - MEAN_PTAL_2023: 0.0
   - Population: 0.0
   - Area_km2: 0.0
   - MeanSentiment: -0.0
 Feature standard deviations (should be   1):
   - AvgPrice: 1.000106
   - MEAN_PTAL_2023: 1.000106
   - Population: 1.000106
   - Area_km2: 1.000106
   - MeanSentiment: 1.000106

Sample data (first 3 rows):
  LSOA_CODE  AvgPrice  MEAN_PTAL_2023  Population  Area_km2  MeanSentiment  \
```

```
0  E01000001  0.562237        0.105844    0.435383 -0.407067      -1.423654
1  E01000002  0.594885        0.105844   -0.090485 -0.153783      -0.597232
2  E01000003 -0.214787        0.105844    0.254346 -0.589051      -2.360832

   SentimentSD  ReviewCount  NearestStation_m  StationsWithin500m  \
0     2.395906     0.054214         -1.240113            2.044513
1     0.681987     0.504721          0.217765            2.184298
2    -0.161150    -0.102639         -0.467001            1.904729

   NearestRail_m  StreetLength_m  StreetDensity_m_per_m2  StreetSegments  \
0      -0.798986       -0.333851                0.966407        0.438961
1      -0.960413        0.541895                0.556147        1.129350
2      -0.868722       -1.482122               -0.408466       -1.402077

   LandUse_Diversity  LandUse_Area
0           0.300171     -0.554583
1           0.871112     -0.415063
2          -0.270771     -0.581698

 SUCCESS! Your optimal GCN feature matrix is ready!
============================================================
 RECOMMENDED FILE: gcn_feature_matrix_custom_spatial_imputed_scaled.csv
 DIMENSIONS: 4719 LSOAs × 15 features
 PREPROCESSING: Spatially imputed + Standardized
 READY FOR: Graph Convolutional Network training
============================================================
```