Data cleaning and preprocessing

In [7]:
```
import pandas as pd

# Specify the path to your CSV file
file_path = 'Group_2_Raw_Data.csv'

# Load the CSV file into a DataFrame
df = pd.read_csv(file_path)

df.head()
```

Out[7]:

	communityname	State	countyCode	communityCode	fold	pop	perHoush	pctBlack	pctWhite	pctAsian	...	burg
0	BerkeleyHeightstownship	NJ	39	5320	1	11980	3.10	1.37	91.78	6.50	...	
1	Marpletownship	PA	45	47616	1	23123	2.82	0.80	95.57	3.44	...	
2	Tigardcity	OR	?	?	1	29344	2.43	0.74	94.33	3.43	...	
3	Gloversvillecity	NY	35	29443	1	16656	2.40	1.70	97.35	0.50	...	
4	Bemidjicity	MN	7	5068	1	11245	2.76	0.53	89.16	1.17	...	

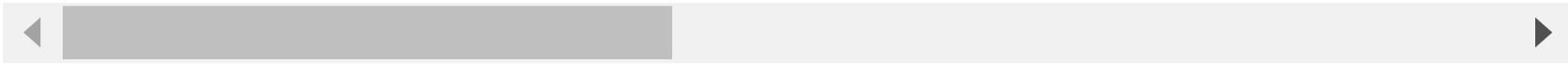
5 rows × 147 columns

In [8]:
```
df.describe()
```

Out[8]:

	fold	pop	perHoush	pctBlack	pctWhite	pctAsian	pctHisp	pct12-21	pct12-29
count	2215.000000	2.215000e+03	2215.000000	2215.000000	2215.000000	2215.000000	2215.000000	2215.000000	2215.000000
mean	5.494357	5.311798e+04	2.707327	9.335102	83.979819	2.670203	7.950176	14.445837	27.644840
std	2.872924	2.046203e+05	0.334120	14.247156	16.419080	4.473843	14.589832	4.518623	6.181517
min	1.000000	1.000500e+04	1.600000	0.000000	2.680000	0.030000	0.120000	4.580000	9.380000
25%	3.000000	1.436600e+04	2.500000	0.860000	76.320000	0.620000	0.930000	12.250000	24.415000
50%	5.000000	2.279200e+04	2.660000	2.870000	90.350000	1.230000	2.180000	13.620000	26.780000
75%	8.000000	4.302400e+04	2.850000	11.145000	96.225000	2.670000	7.810000	15.360000	29.205000
max	10.000000	7.322564e+06	5.280000	96.670000	99.630000	57.460000	95.290000	54.400000	70.510000

8 rows × 102 columns



In [9]:
```
import numpy as np

# Replace '?' with NaN
df.replace('?', np.nan, inplace=True)
```

In [10]:
```
# Summary table of columns with missing data
missing_summary = df.isnull().sum()[df.isnull().sum() > 0].to_frame('Missing Count')
missing_summary['Missing Percentage'] = (df.isnull().mean() * 100)[df.isnull().sum() > 0]
print(missing_summary)
```

	Missing Count	Missing Percentage
countyCode	1221	55.124153
communityCode	1224	55.259594
otherPerCap	1	0.045147
numPolice	1872	84.514673
policePerPop	1871	84.469526
policeField	1871	84.469526
policeFieldPerPop	1871	84.469526
policeCalls	1871	84.469526
policCallPerPop	1871	84.469526
policCallPerOffic	1871	84.469526
policePerPop2	1871	84.469526
racialMatch	1871	84.469526
pctPolicWhite	1871	84.469526
pctPolicBlack	1871	84.469526
pctPolicHisp	1871	84.469526
pctPolicAsian	1871	84.469526
pctPolicMinority	1871	84.469526
officDrugUnits	1871	84.469526
numDiffDrugsSeiz	1871	84.469526
policAveOT	1871	84.469526
policCarsAvail	1871	84.469526
policOperBudget	1871	84.469526
pctPolicPatrol	1871	84.469526
gangUnit	1871	84.469526
policBudgetPerPop	1871	84.469526
rapes	208	9.390519
rapesPerPop	208	9.390519
robberies	1	0.045147
robbbPerPop	1	0.045147
assaults	13	0.586907
assaultPerPop	13	0.586907
burglaries	3	0.135440
burglPerPop	3	0.135440
larcenies	3	0.135440
larcPerPop	3	0.135440
autoTheft	3	0.135440
autoTheftPerPop	3	0.135440
arsons	91	4.108352
arsonsPerPop	91	4.108352
violentPerPop	221	9.977427
nonViolPerPop	97	4.379233

In [11]:
```
# Define the list of ID columns
id_columns = ['communityname', 'State', 'countyCode', 'communityCode', 'fold']

# Separate ID columns from the DataFrame
df_id = df[id_columns]

# Remove ID columns to leave only features and targets
df_features_targets = df.drop(columns=id_columns)

# If needed, separate features and targets further
# Assuming targets start from column index 124
feature_columns = df_features_targets.columns[:124]  # Adjust if necessary
target_columns = df_features_targets.columns[124:]

# Separate features and targets
df_features = df_features_targets[feature_columns]
df_targets = df_features_targets[target_columns]
```

In [12]:
```
# Select the first 124 columns as feature columns
feature_columns = df_features_targets.columns[:124]
df_features = df_features_targets[feature_columns]

# Calculate the percentage of missing data in these feature columns
missing_percentage = df_features.isnull().mean() * 100

# Identify columns with more than 80% missing data
columns_to_drop = missing_percentage[missing_percentage > 80].index
print("Columns to drop due to high missing data:", columns_to_drop)

# Drop these columns from the feature DataFrame
df_features_cleaned = df_features.drop(columns=columns_to_drop)

# Display the remaining feature columns after dropping
print("Remaining feature columns:", df_features_cleaned.columns)
```

```
Columns to drop due to high missing data: Index(['numPolice', 'policePerPop', 'policeField', 'policeFieldPerPop',
       'policeCalls', 'policCallPerPop', 'policCallPerOffic', 'policePerPop2',
       'racialMatch', 'pctPolicWhite', 'pctPolicBlack', 'pctPolicHisp',
       'pctPolicAsian', 'pctPolicMinority', 'officDrugUnits',
       'numDiffDrugsSeiz', 'policAveOT', 'policCarsAvail', 'policOperBudget',
       'pctPolicPatrol', 'gangUnit', 'policBudgetPerPop'],
      dtype='object')
Remaining feature columns: Index(['pop', 'perHoush', 'pctBlack', 'pctWhite', 'pctAsian', 'pctHisp',
       'pct12-21', 'pct12-29', 'pct16-24', 'pct65up',
       ...
       'persHomeless', 'pctForeignBorn', 'pctBornStateResid', 'pctSameHouse-5',
       'pctSameCounty-5', 'pctSameState-5', 'landArea', 'popDensity',
       'pctUsePubTrans', 'pctOfficDrugUnit'],
      dtype='object', length=102)
```

In [13]:
```python
# Identify non-numeric columns in df_features_cleaned
non_numeric_features = df_features_cleaned.select_dtypes(exclude=['float', 'int']).columns

# Display the non-numeric columns and their data types
print("Non-numeric features and their data types:")
print(df_features_cleaned[non_numeric_features].dtypes)
```

```
Non-numeric features and their data types:
otherPerCap       object
pctNotSpeakEng    object
landArea          object
dtype: object
```

In [14]:
```python
# Check the unique values in each column to understand the data
for column in non_numeric_features:
    print(f"Unique values in {column}:")
    print(df_features_cleaned[column].unique())
    print("\n")

# Attempt to convert to numeric, replacing non-convertible values with NaN to identify issues
for column in non_numeric_features:
    df_features_cleaned[column] = pd.to_numeric(df_features_cleaned[column], errors='coerce')

# Display the columns again to confirm if conversion was successful
print("Data types after conversion attempt:")
print(df_features_cleaned[non_numeric_features].dtypes)
```

```
Unique values in otherPerCap:
['5115' '5250' '5954' ... '8532' '4436' '7540']


Unique values in pctNotSpeakEng:
['1.37.1' '1.81' '1.14' '0.56' '0.39' '0.6' '0.28' '0.43' '2.51' '0.81'
 '10.8' '0.59' '0.54' '0.42' '0.69' '0.92' '15.46' '0.53' '2.2' '9.84'
 '4.31' '1.23' '0.75' '1.21' '3.17' '2.94' '0.18' '2.4' '21.64' '2.49'
 '0.24' '0.61' '3.08' '0.68' '2.25' '0.21' '1.64' '3.89' '0.45' '1.35'
 '0.31' '0.51' '0.85' '2.03' '14.64' '0.95' '4.86' '0.8' '2.63' '1.63'
 '0.7' '0.35' '0.44' '1.19' '0.67' '0.47' '0.49' '0.52' '0.48' '0.71'
 '1.53' '0.55' '1.15' '0.23' '0.29' '4.14' '1.0' '1.76' '1.02' '0.93'
 '0.13' '1.06' '3.21' '2.17' '2.59' '5.83' '0.84' '7.86' '0.76' '0.26'
 '0.3' '3.93' '1.1' '0.83' '1.26' '0.57' '3.07' '8.21' '1.99' '1.58'
 '1.48' '3.36' '0.15' '2.43' '0.25' '1.24' '5.43' '3.33' '0.73' '1.78'
 '0.19' '17.45' '16.23' '1.46' '0.17' '5.54' '1.01' '1.57' '1.28' '2.19'
 '1.31' '0.74' '1.41' '2.64' '30.91' '3.38' '0.64' '7.03' '1.04' '14.8'
 '2.02' '1.77' '0.32' '0.38' '0.08' '0.94' '0.33' '6.13' '1.45' '5.4'
 '5.07' '0.78' '5.0' '2.85' '0.36' '18.63' '0.9' '18.2' '3.06' '8.08'
 '1.42' '1.27' '2.75' '0.4' '6.29' '1.73' '9.81' '0.89' '8.33' '1.82'
 '4.76' '0.5' '5.62' '9.11' '0.66' '1.29' '0.58' '6.41' '16.08' '1.44'
 '0.41' '1.03' '0.34' '4.15' '0.46' '2.28' '0.27' '2.88' '0.98' '2.61'
 '1.08' '0.37' '4.38' '0.77' '4.69' '7.02' '1.6' '0.2' '2.08' '0.07'
 '5.16' '0.63' '2.7' '0.65' '1.43' '3.25' '3.51' '1.52' '17.59' '1.75'
 '2.47' '12.14' '3.01' '0.87' '6.26' '7.46' '1.32' '9.26' '2.01' '0.09'
 '0.86' '1.22' '0.91' '0.0' '2.72' '3.84' '1.17' '2.57' '1.67' '3.48'
 '0.62' '2.27' '0.79' '1.91' '1.16' '1.38' '6.36' '4.53' '6.8' '5.93'
 '0.99' '10.48' '1.95' '3.64' '6.72' '2.6' '6.07' '2.22' '5.75' '25.71'
 '4.82' '2.93' '1.88' '1.51' '3.77' '1.69' '2.92' '15.02' '4.42' '2.48'
 '1.13' '5.15' '4.65' '8.8' '11.13' '2.65' '1.09' '6.06' '4.41' '3.16'
 '1.86' '3.75' '1.55' '0.06' '12.52' '4.09' '1.68' '10.35' '1.33' '1.11'
 '1.18' '0.82' '2.16' '8.73' '3.0' '9.62' '3.41' '4.83' '6.21' '21.27'
 '0.96' '0.88' '7.18' '7.84' '3.83' '17.72' '1.07' '3.43' '3.27' '2.55'
 '17.17' '14.47' '0.12' '6.43' '13.84' '2.74' '2.83' '2.32' '15.14' '0.97'
 '6.27' '14.53' '18.35' '3.79' '5.21' '14.03' '4.34' '0.04' '1.49' '3.15'
 '3.18' '4.32' '4.43' '1.7' '0.1' '2.96' '1.34' '17.57' '5.88' '1.89'
 '1.93' '2.95' '3.78' '2.97' '8.9' '0.72' '14.45' '6.88' '3.99' '0.22'
 '5.04' '6.95' '7.78' '14.73' '3.39' '6.09' '6.47' '1.47' '8.34' '7.53'
 '1.61' '6.99' '3.86' '7.12' '29.27' '3.55' '3.59' '2.09' '1.83' '0.16'
 '10.84' '5.82' '2.98' '38.33' '0.11' '5.55' '6.4' '1.94' '1.36' '22.32'
 '7.97' '1.05' '1.65' '1.4' '22.46' '18.11' '4.94' '8.38' '4.21' '7.68'
 '4.29' '6.98' '18.51' '3.95' '4.75' '16.67' '2.62' '2.1' '1.39' '3.29'
```

```
 '2.67' '1.71' '1.79' '13.15' '22.83' '13.98' '3.28' '3.5' '19.9' '6.37'
 '8.1' '2.56' '3.63' '2.38' '1.87' '2.14' '0.14' '9.22' '10.63' '9.66'
 '6.15' '1.37' '4.9' '0.05' '5.64' '7.29' '1.62' '12.31' '14.19' '3.4'
 '2.15' '1.5' '6.31' '9.85' '24.08' '8.75' '3.74' '7.24' '2.87' '3.54'
 '14.81' '6.64' '5.5' '4.06' '2.07' '2.89' '3.94' '3.97' '15.25' '2.46'
 '6.05' '3.44' '1.12' '3.87' '2.04' '3.65' '3.7' '7.92' '2.82' '12.98'
 '10.39' '11.7' '2.71' '6.01' '7.07' '7.52' '1.56' '8.74' '2.69' '5.91'
 '16.85' '1.3' '1.66' '5.81' '2.5' '1.8' '22.13' '11.03' '4.62' '8.58'
 '3.58' '2.06' '26.89' '31.26' '4.52' '2.34' '6.54' '14.76' '3.52' '4.88'
 '1.25' '1.92' '6.93' '17.14' '2.39' '7.56' '8.17' '2.11' '9.09' '1.97'
 '3.45' '30.37' '2.23' '7.91' '10.01' '1.2' '1.74' '10.34' '18.13' '15.0'
 '3.53' '1.9' '4.19' '6.6' '10.36' '36.78' '6.2' '3.49' '13.83' '4.08'
 '2.24' '6.53' '7.13' '7.42' '9.4' '6.42' '7.6' '0.01' '29.78' '8.54'
 '3.22' '8.24' '21.03' '9.74' '4.39' '2.3' '2.05' '5.25' '3.8' '7.98'
 '13.11' '3.12' '8.48' '7.33' '5.41' '4.8' '5.08' '10.37' '18.56' '2.29'
 '3.9' '25.73' '10.41' '11.91' '7.9' '3.04' '5.45' '1.96' '19.62' '2.84'
 '14.38' '3.32' '2.0' '5.69' '9.23' '5.87' '2.13' '15.56' '3.92' '16.07'
 '9.08' '6.74' '3.47' '4.96' '8.13' '6.18' '2.21' '5.53' '3.85' '2.81'
 '4.6' '7.72' '3.13' '4.98' '2.36' '20.26' '2.31' '24.51' '4.07' '4.61'
 '6.55' '13.0' '4.1' '3.23' '2.42' '7.75' '13.28' '29.07' '11.42' '24.15'
 '7.66' '20.54' '8.81' '34.27' '2.52' '7.0' '9.2' '5.1' '4.55' '1.59'
 '4.04' '23.31' '4.74' '5.35' '20.56' '5.97' '1.54' '31.34' '4.87' '12.03'
 '5.65' '1.84' '17.82' '4.18' '11.27' '16.42' '10.7' '4.49' '18.4' '2.54'
 '18.15' '3.26' '6.83' '2.58' '3.91' '1.98' '1.72' '5.24' '31.63' '7.48'
 '36.42' '4.72' '2.37' '2.18' '3.11' '25.41' '3.09' '5.57' '4.12' '4.56'
 '16.81' '5.71' '12.58' '11.87' '6.66' '7.76']


Unique values in landArea:
['6.5.1' '10.6' '5.2' '11.5' '70.4' '10.9' '39.2' '30.9' '78.5' '38.7'
 '2.7' '30.0' '43.9' '8.9' '17.9' '34.6' '31.7' '5.8' '96.4' '34.1'
 '320.1' '9.1' '41.8' '10.1' '23.0' '3.7' '14.9' '9.4' '4.8' '6.2' '52.5'
 '19.1' '4.2' '7.3' '14.2' '27.0' '49.6' '7.9' '20.7' '414.5' '11.4'
 '27.2' '9.0' '9.6' '25.2' '30.1' '5.9' '1.9' '8.3' '83.5' '140.0' '18.7'
 '26.1' '51.2' '24.7' '12.8' '44.4' '5.1' '10.2' '26.7' '38.6' '36.1'
 '1.2' '80.0' '7.5' '36.8' '13.7' '10.5' '34.8' '161.9' '93.6' '16.2'
 '3.1' '4.1' '29.1' '11.7' '10.3' '3.8' '62.2' '2.9' '129.1' '10.4' '21.3'
 '20.9' '19.7' '6.3' '5.4' '13.0' '15.0' '19.5' '3.3' '4.7' '7.8' '6.5'
 '22.9' '7.1' '18.6' '8.5' '17.0' '65.6' '5.3' '42.2' '13.9' '9.5' '12.5'
 '20.2' '21.6' '5.5' '12.2' '486.2' '1.3' '23.9' '38.3' '9.3' '46.7'
 '32.4' '45.9' '5.0' '6.1' '11.6' '57.9' '4.9' '11.0' '23.6' '57.6' '26.8'
 '37.1' '3.9' '35.1' '14.0' '2.6' '53.3' '2.3' '35.7' '112.6' '354.7'
```

```
'14.1' '22.8' '44.6' '4.3' '30.8' '23.2' '14.4' '20.8' '73.8' '22.6'
'29.4' '16.1' '3.0' '13.6' '11.3' '18.1' '51.8' '70.0' '29.5' '16.0'
'24.3' '25.8' '30.5' '6.7' '27.4' '630.0' '6.6' '12.1' '5.6' '8.0' '20.6'
'36.6' '12.4' '2.8' '24.9' '28.6' '10.7' '23.4' '6.4' '28.2' '15.9' '4.5'
'3.5' '22.4' '15.7' '8.2' '15.2' '36.9' '4.4' '1.8' '33.4' '23.3' '17.5'
'4.0' '11.2' '25.5' '7.0' '3.6' '8.7' '33.5' '22.5' '2.0' '7.4' '49.5'
'9.9' '59.6' '24.5' '46.6' '20.0' '14.6' '8.1' '28.9' '17.4' '34.0'
'24.0' '7.2' '119.3' '112.9' '16.7' '10.0' '40.1' '88.6' '5.7' '28.8'
'7.7' '13.3' '3.2' '10.8' '21.9' '6.9' '50.9' '13.5' '9.7' '19.3' '28.5'
'2.1' '31.5' '12.3' '26.0' '24.6' '8.4' '42.1' '57.5' '30.2' '108.9'
'143.7' '82.6' '15.4' '27.6' '147.5' '64.8' '33.6' '99.8' '39.8' '14.8'
'12.9' '7.6' '291.2' '48.1' '30.7' '27.3' '13.2' '14.5' '8.8' '21.1'
'38.9' '19.2' '16.5' '6.0' '22.1' '1.7' '74.1' '2.2' '37.7' '4.6' '2.4'
'17.8' '15.3' '22.0' '27.9' '31.0' '15.8' '16.6' '12.6' '8.6' '19.0'
'32.1' '48.4' '13.4' '59.8' '31.1' '65.9' '40.0' '35.3' '24.1' '11.1'
'33.1' '21.2' '1.0' '47.8' '35.2' '158.8' '24.2' '22.3' '40.4' '39.4'
'33.2' '17.6' '18.8' '3.4' '70.8' '40.2' '37.2' '40.7' '13.8' '50.2'
'53.2' '14.3' '75.7' '29.8' '6.8' '17.2' '12.0' '18.0' '1.6' '21.8'
'18.2' '76.6' '11.8' '34.4' '29.9' '21.7' '19.6' '61.9' '15.1' '45.4'
'190.1' '11.9' '34.2' '30.6' '9.2' '35.9' '106.6' '25.7' '27.5' '57.0'
'44.2' '27.1' '42.6' '254.2' '82.5' '28.7' '31.4' '57.1' '183.4' '2.5'
'31.2' '23.5' '25.1' '63.6' '91.3' '9.8' '29.3' '225.6' '13.1' '322.7'
'51.3' '177.4' '22.2' '24.4' '38.2' '180.5' '122.7' '12.7' '23.1' '20.3'
'17.3' '50.4' '187.2' '26.3' '88.3' '122.3' '35.4' '18.3' '121.4' '43.6'
'37.0' '196.1' '34.5' '21.5' '25.3' '559.3' '28.4' '83.7' '22.7' '25.9'
'26.4' '19.4' '18.9' '23.8' '26.2' '20.1' '265.3' '1.5' '63.7' '33.8'
'33.7' '27.8' '16.4' '81.0' '23.7' '43.8' '78.0' '50.0' '112.5' '53.7'
'25.6' '1758.8' '70.1' '42.4' '16.9' '103.7' '30.4' '20.4' '16.3' '102.7'
'18.4' '52.0' '21.4' '26.5' '87.0' '17.1' '40.9' '43.7' '64.3' '44.8'
'15.5' '71.8' '69.6' '59.2' '39.1' '86.3' '54.6' '29.2' '37.5' '107.9'
'71.2' '69.7' '56.6' '58.1' '36.0' '16.8' '65.5' '95.1' '41.2' '48.3'
'1.1' '61.7' '34.7' '33.3' '224.1' '28.1' '29.6' '54.2' '55.7' '153.8'
'19.8' '26.9' '45.6' '64.9' '32.5' '21.0' '2686.9' '59.4' '36.2' '29.0'
'56.9' '80.4' '43.3' '335.7' '345.0' '233.0' '34.9' '62.3' '36.3' '137.3'
'3569.8' '47.1' '27.7' '49.8' '75.9' '31.3' '48.5' '45.0' '37.3' '82.9'
'56.0' '99.4' '20.5' '39.9' '79.8' '32.3' '0.9' '79.3' '36.5' '51.9'
'60.2' '40.5' '136.5' '35.0' '38.1' '49.3' '85.6' '38.0' '43.0' '64.4'
'18.5' '54.5' '188.2' '29.7' '49.4' '801.7' '53.0' '46.0' '61.6' '39.0'
'15.6' '53.1' '44.5' '14.7' '68.6' '25.4' '48.9' '100.0' '42.7' '44.3'
'91.1' '77.6' '31.6' '42.3' '137.0' '24.8' '44.7' '191.0' '68.2' '57.4'
'170.3' '62.7' '28.3' '140.9' '139.8' '33.9' '80.5' '64.0' '44.9' '352.9'
'17.7' '34.3' '92.0' '41.1' '102.2' '32.0' '82.7' '380.0' '26.6' '54.1'
```

```
 '257.3' '435.0' '66.6' '32.7' '74.8' '41.9' '86.9' '56.2' '45.3' '48.8'
 '78.2']


Data types after conversion attempt:
otherPerCap       float64
pctNotSpeakEng    float64
landArea          float64
dtype: object
```

In [15]:
```
# Calculate the percentage of missing data in non-numeric feature columns
missing_percentage_non_numeric = df_features[non_numeric_features].isnull().mean() * 100

# Display the results
print("Percentage of missing data in non-numeric feature columns:")
print(missing_percentage_non_numeric)
```

```
Percentage of missing data in non-numeric feature columns:
otherPerCap       0.045147
pctNotSpeakEng    0.000000
landArea          0.000000
dtype: float64
```

In [16]:
```
# Identify non-numeric columns in df_targets
non_numeric_targets = df_targets.select_dtypes(exclude=['float', 'int']).columns
print("Non-numeric target columns:", non_numeric_targets)
# Convert non-numeric columns to numeric, forcing errors to NaN
"""
for column in non_numeric_targets:
    df_targets[column] = pd.to_numeric(df_targets[column], errors='coerce')
    """
```

```
Non-numeric target columns: Index(['rapes', 'rapesPerPop', 'robberies', 'robbbPerPop', 'assaults',
       'assaultPerPop', 'burglaries', 'burglPerPop', 'larcenies', 'larcPerPop',
       'autoTheft', 'autoTheftPerPop', 'arsons', 'arsonsPerPop',
       'violentPerPop', 'nonViolPerPop'],
      dtype='object')
```

Out[16]:
```
"\nfor column in non_numeric_targets:\n    df_targets[column] = pd.to_numeric(df_targets[column], errors='coerce')\n
"
```

Impute missing df_id

In [17]:
```
# Calculate the percentage of missing values for each column in df_id
missing_percentage = df_id.isnull().mean() * 100

# Display the missing percentage for each ID column
print("Percentage of missing values in each ID column:\n", missing_percentage)
```

```
Percentage of missing values in each ID column:
 communityname     0.000000
State              0.000000
countyCode        55.124153
communityCode     55.259594
fold               0.000000
dtype: float64
```

In [18]:
```
from sklearn.preprocessing import LabelEncoder

# Apply label encoding to categorical columns in df_id
for col in df_id.columns:
    if df_id[col].dtype == 'object':
        le = LabelEncoder()
        df_id.loc[:, col] = le.fit_transform(df_id[col].astype(str))
```

In [19]:
```
from sklearn.impute import KNNImputer
import pandas as pd

# Initialize the KNN imputer with the number of neighbors
knn_imputer = KNNImputer(n_neighbors=3)

# Perform KNN imputation
df_id_imputed = pd.DataFrame(knn_imputer.fit_transform(df_id), columns=df_id.columns)

# Verify that the missing values have been imputed
print("Missing values after KNN imputation:\n", df_id_imputed.isnull().sum())
```

```
Missing values after KNN imputation:
 communityname    0
State             0
countyCode        0
communityCode     0
fold              0
dtype: int64
```

Inpute df_features_cleaned and df_targets

In [20]:
```
from sklearn.impute import SimpleImputer


# Separate numerical and categorical columns in df_features_cleaned
num_features = df_features_cleaned.select_dtypes(include=['float', 'int']).columns


# Mean imputation for numerical features
num_imputer = SimpleImputer(strategy='median')
df_features_cleaned[num_features] = num_imputer.fit_transform(df_features_cleaned[num_features])


# Check if there are any remaining missing values
print("Remaining missing values in df_features_cleaned:\n", df_features_cleaned.isnull().sum())
```

```
Remaining missing values in df_features_cleaned:
 pop                 0
perHoush             0
pctBlack             0
pctWhite             0
pctAsian             0
                    ..
pctSameState-5       0
landArea             0
popDensity           0
pctUsePubTrans       0
pctOfficDrugUnit     0
Length: 102, dtype: int64
```

In [21]:
```
# Impute missing values in the target variables using mean strategy
target_imputer = SimpleImputer(strategy='median')
df_targets= pd.DataFrame(target_imputer.fit_transform(df_targets), columns=df_targets.columns)

# Check if there are any remaining missing values
print("Remaining missing values in df_targets:\n", df_targets.isnull().sum())
```

```
Remaining missing values in df_targets:
 murders            0
murdPerPop          0
rapes               0
rapesPerPop         0
robberies           0
robbbPerPop         0
assaults            0
assaultPerPop       0
burglaries          0
burglPerPop         0
larcenies           0
larcPerPop          0
autoTheft           0
autoTheftPerPop     0
arsons              0
arsonsPerPop        0
violentPerPop       0
nonViolPerPop       0
dtype: int64
```

Output the cleaned dataset

In [22]:
```
# Combine the DataFrames along columns
df_combined = pd.concat([df_id_imputed, df_features_cleaned, df_targets], axis=1)

# Output the combined DataFrame to a CSV file
df_combined.to_csv("Group_2_clean_Data..csv", index=False)

print("Data successfully saved to combined_output.csv")
```

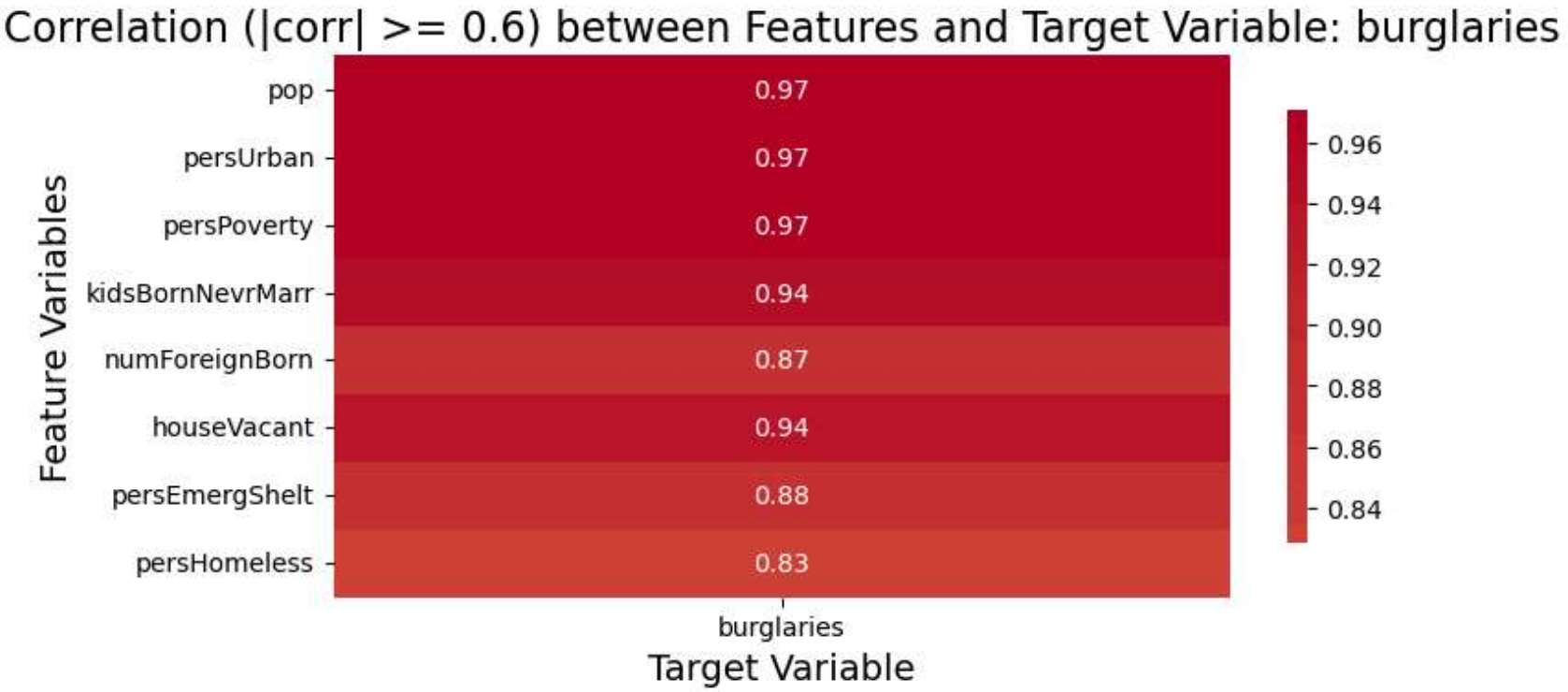
```
Data successfully saved to combined_output.csv
```

Exploratory data analysis

In [23]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
```

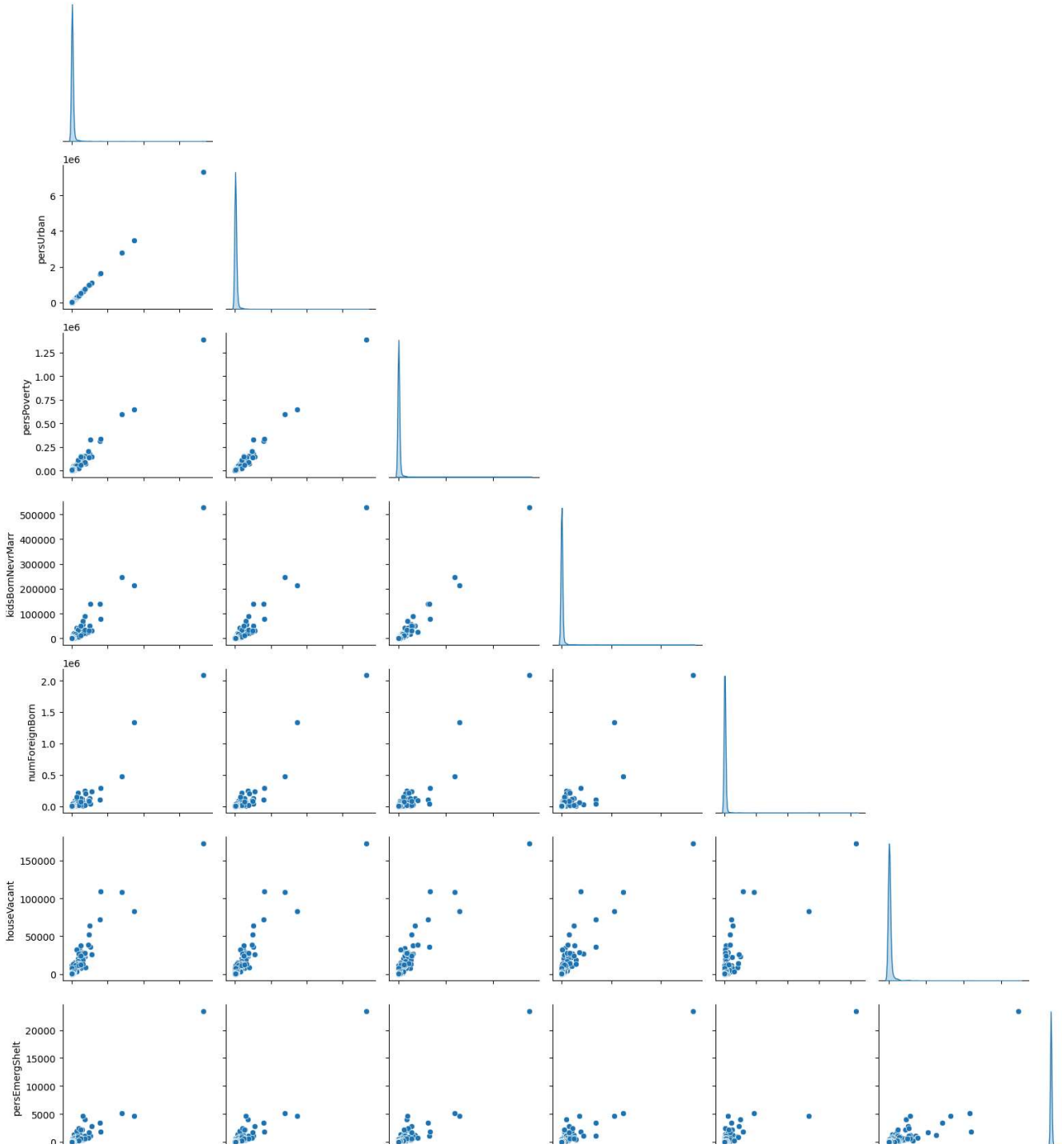
```
# Combine features and targets into one DataFrame to calculate correlations
df_combined = pd.concat([df_features_cleaned, df_targets], axis=1)

# Ensure 'violentPerPop' exists in the targets DataFrame
if 'burglaries' in df_targets.columns:
    # Calculate the correlation matrix for the combined DataFrame
    correlation_matrix = df_combined.corr()

    # Extract correlations between features and the specific target variable
    correlation_with_target = correlation_matrix['burglaries'][df_features_cleaned.columns]

    # Define a correlation threshold
    threshold = 0.6

    # Filter features based on the threshold
    filtered_correlations = correlation_with_target[correlation_with_target.abs() >= threshold]

    # Check if any features remain after filtering
    if filtered_correlations.empty:
        print(f"No features have correlation >= {threshold} with 'burglaries'.")
    else:
        # Plot the heatmap for filtered correlations
        plt.figure(figsize=(8, len(filtered_correlations) * 0.5))  # Adjust figure size dynamically

        sns.heatmap(
            filtered_correlations.to_frame(),
            annot=True,
            cmap="coolwarm",
            center=0,
            fmt=".2f",
            cbar_kws={"shrink": 0.8},
        )

        # Set titles and labels
        plt.title(
            f"Correlation (|corr| >= {threshold}) between Features and Target Variable: burglaries",
            fontsize=16,
        )
        plt.xlabel("Target Variable", fontsize=14)
        plt.ylabel("Feature Variables", fontsize=14)
```

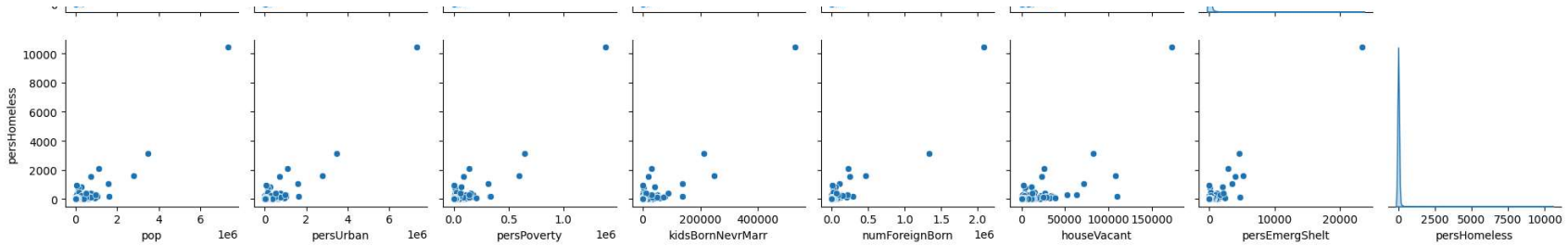
```
        # Display the heatmap
        plt.tight_layout()
        plt.show()
else:
    print("The target variable 'burglaries' is not in df_targets. Please check your input data.")
```



In [24]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define the selected features
select_features = [
    'pop',
    'persUrban',
    'persPoverty',
    'kidsBornNevrMarr',
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
```

```
]

# Ensure `df_features_cleaned` contains the selected features
if all(feature in df_features_cleaned.columns for feature in select_features):
    # Create a pairplot
    pairplot = sns.pairplot(df_features_cleaned[select_features], diag_kind="kde", corner=True)

    # Adjust layout for better visibility
    plt.tight_layout()
    plt.show()
else:
    print("Some selected features are not in the dataset. Please verify the feature names.")
```


1e6
persUrban
1e6
persPoverty
kidsBornNevrMarr
1e6
numForeignBorn
houseVacant
persEmergShelt



In [25]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

select_features = [
    'pop',
    'persUrban',
    'persPoverty',
    'kidsBornNevrMarr',
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
]


try:
    # Compute the correlation matrix for the selected features
    correlation_matrix = df_features_cleaned[selected_features].corr()

    # Plot the correlation matrix
    plt.figure(figsize=(10, 8))
    sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", cbar=True)
    plt.title("Correlation Matrix for Selected Features", fontsize=16)
    plt.show()
except NameError as e:
    print("Error:", e)
    print("Please ensure the dataset `df_features_cleaned` is loaded and contains the selected features.")
```

```
Error: name 'selected_features' is not defined
Please ensure the dataset `df_features_cleaned` is loaded and contains the selected features.
```

Correlation Matrix Summary

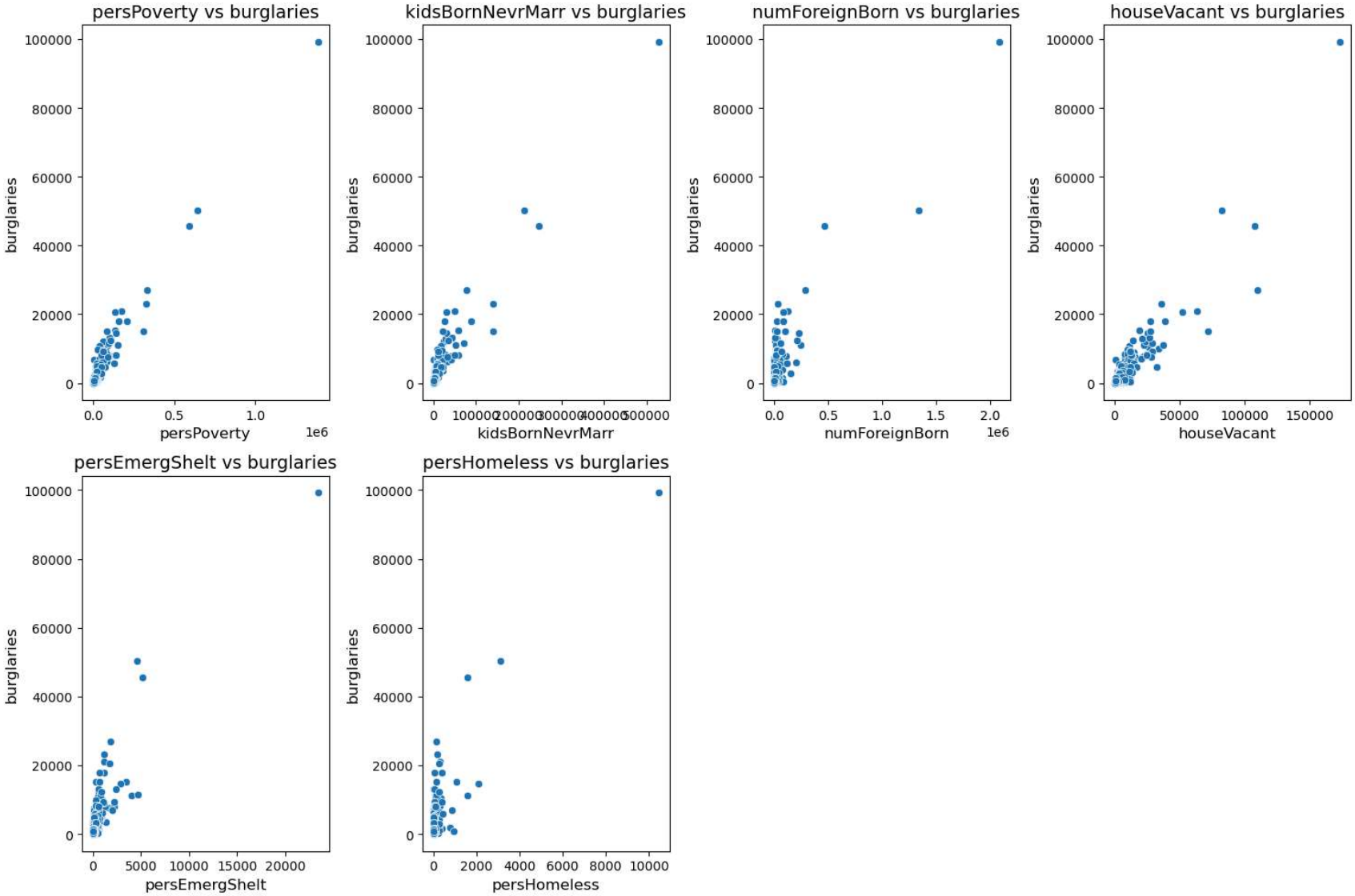
- **Strong Positive Correlations**:
  - Features like `pct2Par`, `pctKids2Par`, `pctKids-4w2Par`, and `pct12-17w2Par` are highly correlated (>0.9), indicating redundancy in family structure metrics.
- **Strong Negative Correlations**:
  - `pctKidsBornNevrMarr` shows strong negative correlations (~-0.85) with features like `pct2Par` and `pctKids2Par`, suggesting an inverse relationship with two-parent household metrics.
- **Moderate Correlations**:
  - `pctWhite` is moderately correlated with both family structure metrics (~0.7) and `pctKidsBornNevrMarr` (~-0.8), highlighting potential demographic trends.

Recommendation:

- Address multicollinearity by applying **dimensionality reduction** (e.g., PCA) or **feature selection** to improve model performance.

In [26]:
```
import matplotlib.pyplot as plt
import seaborn as sns


selected_features = [
    'persPoverty',
    'kidsBornNevrMarr',
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
]

target = 'burglaries'

# Set up the figure and axes
```

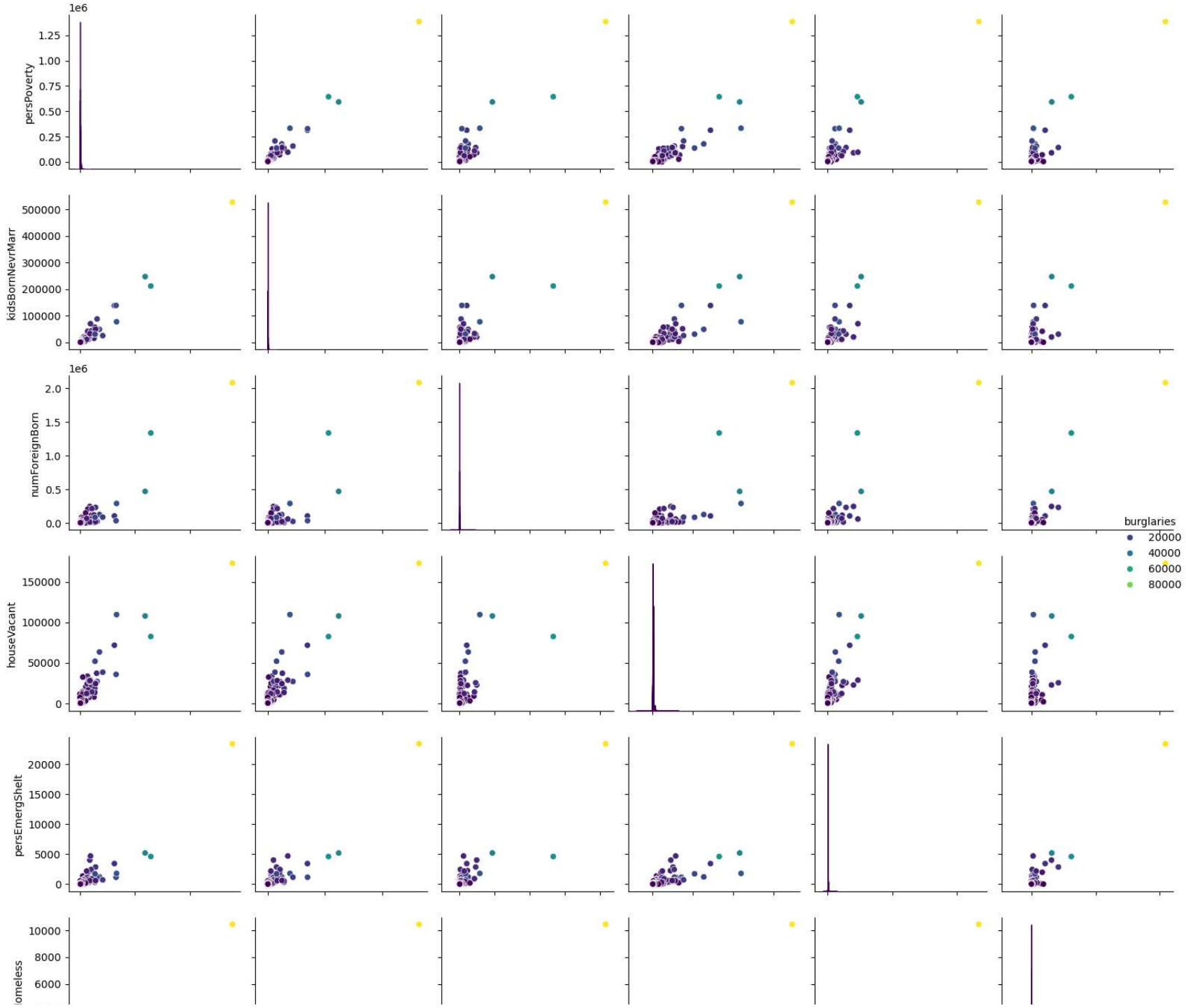
```
n_features = len(selected_features)
n_cols = 4  # Number of columns in the subplot grid
n_rows = (n_features + n_cols - 1) // n_cols  # Calculate rows needed to fit all features

fig, axes = plt.subplots(n_rows, n_cols, figsize=(15, n_rows * 5))  # Adjust figure size

# Flatten axes for easy indexing
axes = axes.flatten()

# Iterate over features and create scatter plots using sns
for i, feature in enumerate(selected_features):
    if feature in df_features_cleaned.columns:
        sns.scatterplot(
            x=df_features_cleaned[feature],
            y=df_targets[target],
            ax=axes[i]
        )
        axes[i].set_title(f"{feature} vs {target}", fontsize=14)
        axes[i].set_xlabel(feature, fontsize=12)
        axes[i].set_ylabel(target, fontsize=12)
    else:
        axes[i].text(0.5, 0.5, "Feature not found",
                     fontsize=12, ha='center', va='center')
        axes[i].set_title(f"{feature} (Error)", fontsize=14)

# Remove any unused subplots
for j in range(len(selected_features), len(axes)):
    fig.delaxes(axes[j])

# Adjust layout
plt.tight_layout()

# Save the figure as a PNG file
plt.savefig('scatter_plots.png', dpi=300)

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

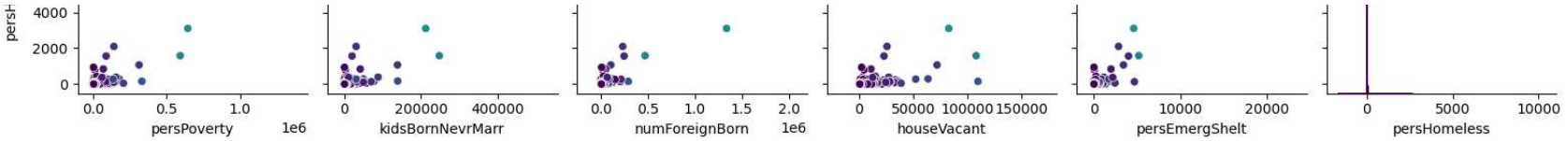


In [27]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define the selected features
selected_features = [
    'persPoverty',
    'kidsBornNevrMarr',
```

```
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
]
target = 'burglaries'
# Ensure you select the target variable as well for hue
df_selected = df_features_cleaned[selected_features]
df_selected[target] = df_targets[target]  # Add target column to the dataframe

# Create the pairplot with hue based on the target variable
sns.pairplot(df_selected, hue=target, palette='viridis', diag_kind='kde')

# Show the plot
plt.tight_layout()
plt.show()
```

```
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\1565295081.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy
  df_selected[target] = df_targets[target]  # Add target column to the dataframe
```

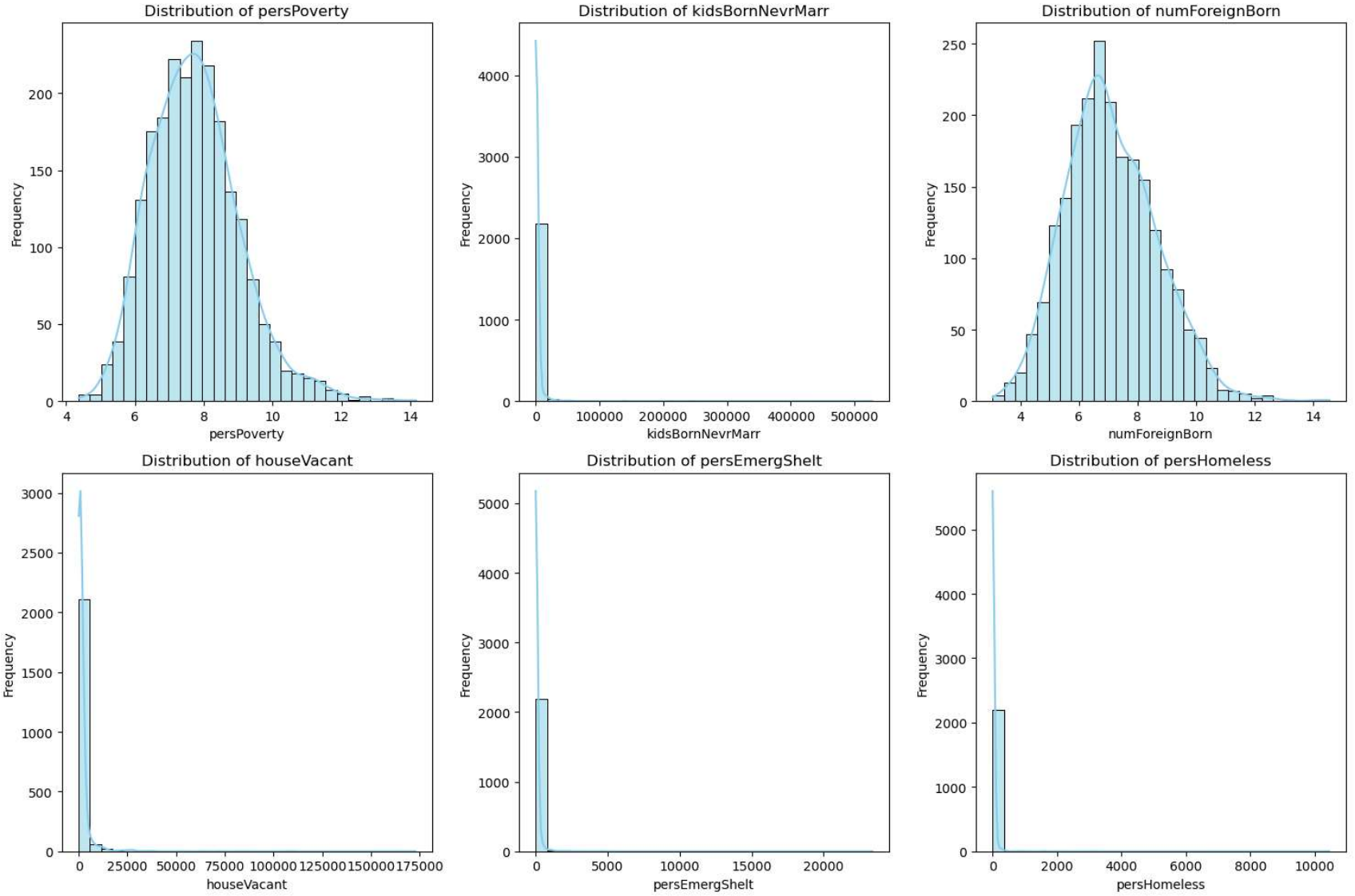

1e6
persPoverty
kidsBornNevrMarr
1e6
numForeignBorn
houseVacant
persEmergShelt
omeless
burglaries
20000
40000
60000
80000



In [ ]:

In [36]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define your selected features
selected_features = [
    'persPoverty',
    'kidsBornNevrMarr',
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
]

# Select only numerical columns from the selected features
numerical_selected_features = df[selected_features]

# Determine the number of rows needed (for 3 columns per row)
n_rows = len(selected_features) // 3 + (len(selected_features) % 3 > 0)

# Create subplots with 3 columns per row
fig, axes = plt.subplots(n_rows, 3, figsize=(15, n_rows * 5))

# Flatten the axes array to iterate over it easily
axes = axes.flatten()

# Plot histogram and KDE for each selected numerical feature
for i, column in enumerate(numerical_selected_features.columns):
    sns.histplot(df[column], kde=True, bins=30, color='skyblue', edgecolor='black', ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')

# Hide any empty subplots (if the number of features isn't a multiple of 3)
for i in range(len(numerical_selected_features.columns), len(axes)):
```

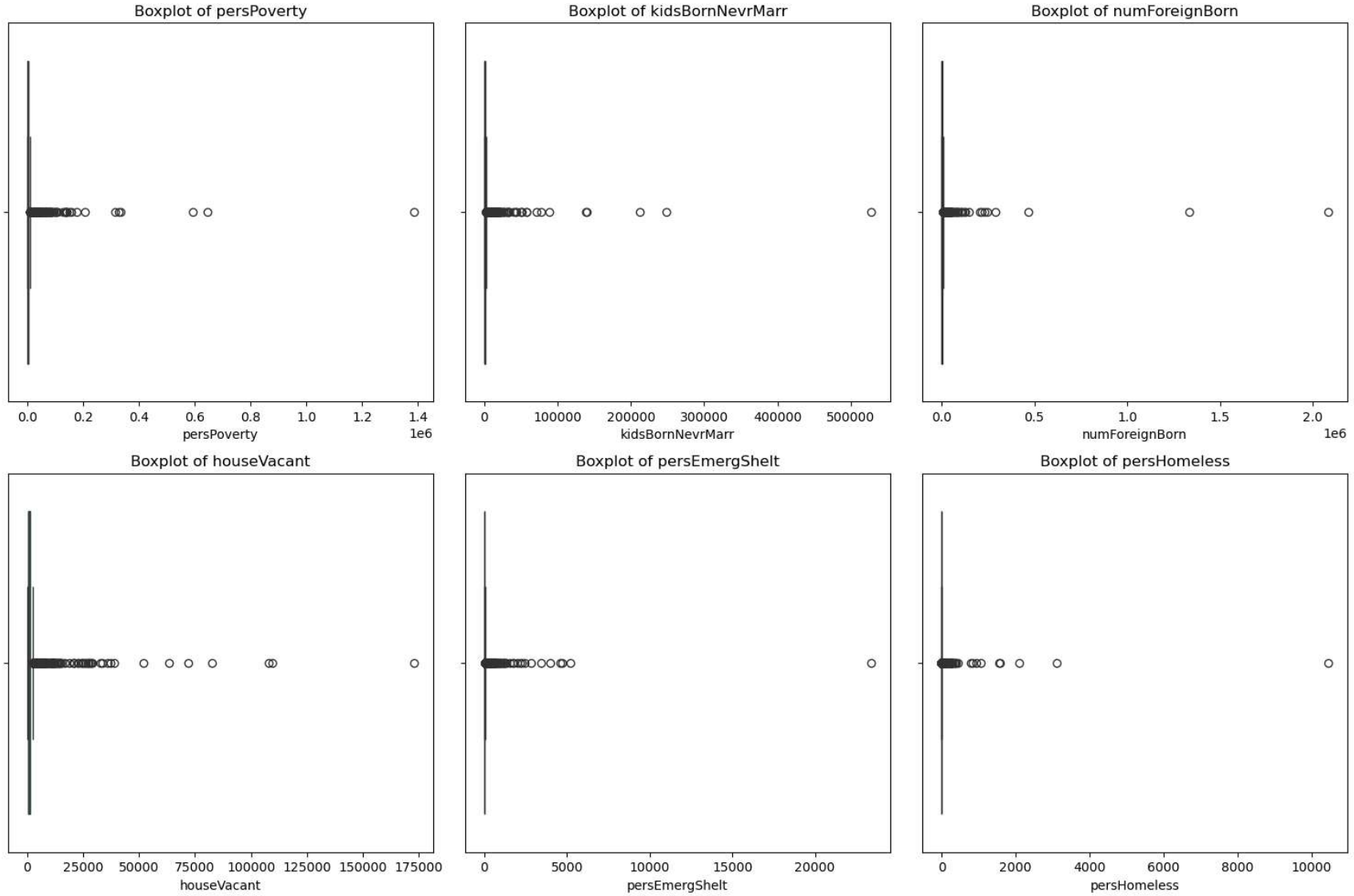
```
    axes[i].axis('off')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.show()
```



In [30]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

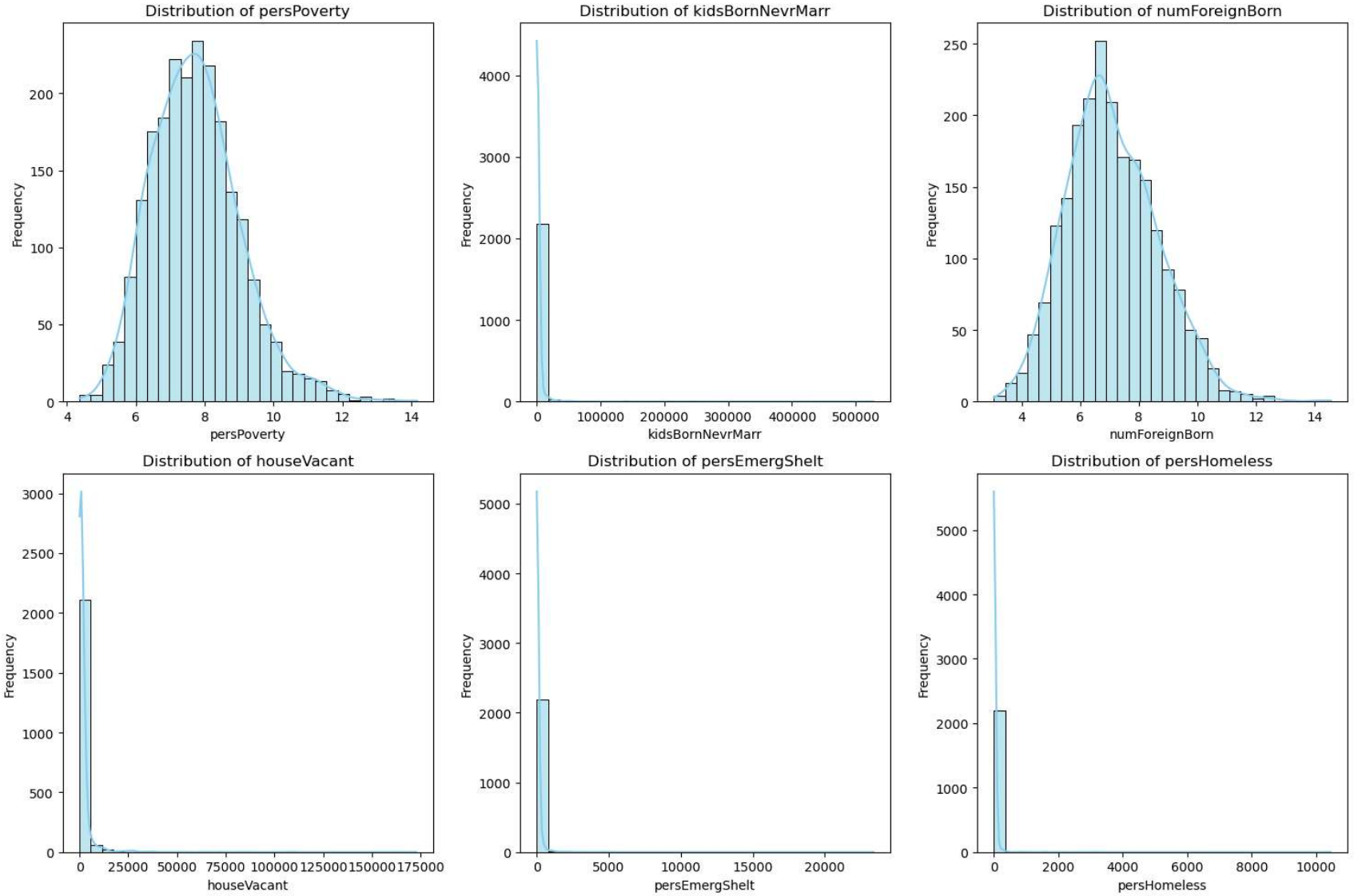
```
# Define the selected features
selected_features = [
    'persPoverty',
    'kidsBornNevrMarr',
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
]

# Ensure the dataframe only contains the selected features
df_selected = df_features_cleaned[selected_features]

# Create boxplots for each selected feature
plt.figure(figsize=(15, 10))  # Adjust the figure size as needed

# Loop through each feature and create a boxplot
for i, feature in enumerate(selected_features):
    plt.subplot(2, 3, i + 1)  # Arrange in a 2x3 grid of plots
    sns.boxplot(x=df_selected[feature], palette='viridis')
    plt.title(f'Boxplot of {feature}')

# Adjust layout for clarity
plt.tight_layout()
plt.show()
```

```
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\525490335.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x=df_selected[feature], palette='viridis')
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\525490335.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x=df_selected[feature], palette='viridis')
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\525490335.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x=df_selected[feature], palette='viridis')
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\525490335.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x=df_selected[feature], palette='viridis')
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\525490335.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x=df_selected[feature], palette='viridis')
C:\Users\hyz20\AppData\Local\Temp\ipykernel_7612\525490335.py:23: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `h
ue` and set `legend=False` for the same effect.

  sns.boxplot(x=df_selected[feature], palette='viridis')
```



In [39]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Define your selected features
selected_features = [
    'persPoverty',
    'kidsBornNevrMarr',
```

```
    'numForeignBorn',
    'houseVacant',
    'persEmergShelt',
    'persHomeless'
]

# Select only numerical columns from the selected features
numerical_selected_features = df[selected_features]

# Determine the number of rows needed (for 3 columns per row)
n_rows = len(selected_features) // 3 + (len(selected_features) % 3 > 0)

# Create subplots with 3 columns per row
fig, axes = plt.subplots(n_rows, 3, figsize=(15, n_rows * 5))

# Flatten the axes array to iterate over it easily
axes = axes.flatten()

# Plot histogram and KDE for each selected numerical feature
for i, column in enumerate(numerical_selected_features.columns):
    sns.histplot(df[column], kde=True, bins=30, color='skyblue', edgecolor='black', ax=axes[i])
    axes[i].set_title(f'Distribution of {column}')
    axes[i].set_xlabel(column)
    axes[i].set_ylabel('Frequency')

# Hide any empty subplots (if the number of features isn't a multiple of 3)
for i in range(len(numerical_selected_features.columns), len(axes)):
    axes[i].axis('off')

# Adjust layout to prevent overlap
plt.tight_layout()
plt.savefig('feature_distribution.png', dpi=300)
plt.show()
```



Feature Scaling and Normalization

From the feature plots, we can observe that the **feature scale ranges vary significantly**. To address this, we need to apply **Min-Max Scaling** or **StandardScaler** to normalize the data.

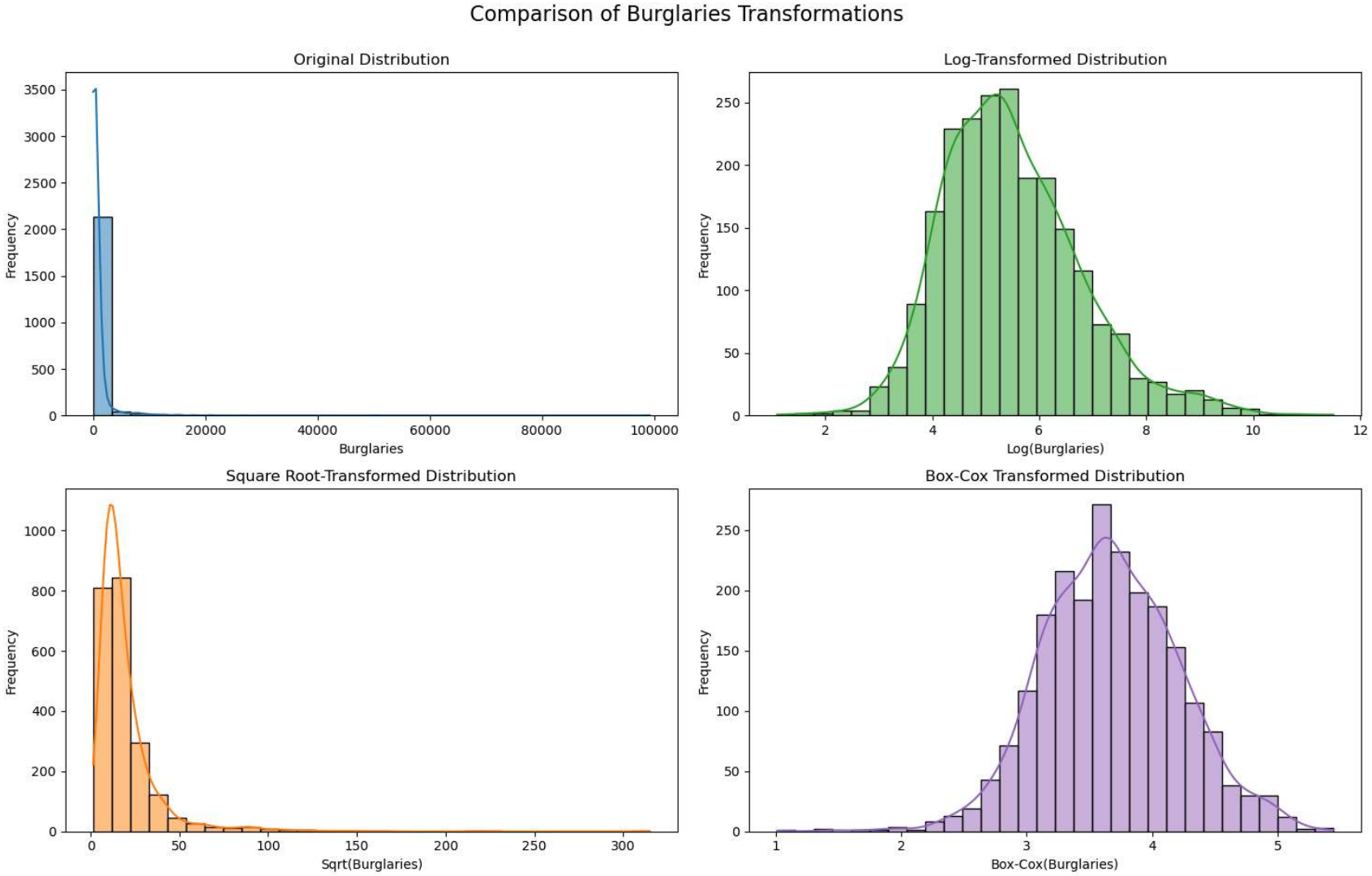
Normalization is crucial for **scale-sensitive algorithms** such as:

- **KNN (k-nearest neighbors)**
- **SVM (Support Vector Machines)**
- **Linear Regression**
- **Advanced learning algorithms like deep learning (e.g., neural networks)**

This ensures all features contribute equally, improving model performance and convergence.

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats

# Apply transformations to the target variable
original = df_targets['burglaries']
log_transformed = np.log1p(original)  # Log transformation (log(1 + x) to handle zeroes)
sqrt_transformed = np.sqrt(original)  # Square root transformation
boxcox_transformed, _ = scipy.stats.boxcox(original + 1)  # Box-Cox requires positive values

# Define conference paper-friendly colors
colors = {
    'original': '#1f77b4',  # Blue (for original)
    'log': '#2ca02c',       # Green (for log-transformed)
    'sqrt': '#ff7f0e',      # Orange (for sqrt-transformed)
    'boxcox': '#9467bd'     # Purple (for box-cox transformed)
}

# Create subplots to compare transformations
fig, axes = plt.subplots(2, 2, figsize=(15, 10))
fig.suptitle("Comparison of Burglaries Transformations", fontsize=16, y=0.95)

# Original distribution
sns.histplot(original, kde=True, color=colors['original'], ax=axes[0, 0], bins=30, edgecolor='black')
axes[0, 0].set_title('Original Distribution')
axes[0, 0].set_xlabel('Burglaries')
axes[0, 0].set_ylabel('Frequency')

# Log-transformed distribution
sns.histplot(log_transformed, kde=True, color=colors['log'], ax=axes[0, 1], bins=30, edgecolor='black')
```

```
axes[0, 1].set_title('Log-Transformed Distribution')
axes[0, 1].set_xlabel('Log(Burglaries)')
axes[0, 1].set_ylabel('Frequency')

# Square root-transformed distribution
sns.histplot(sqrt_transformed, kde=True, color=colors['sqrt'], ax=axes[1, 0], bins=30, edgecolor='black')
axes[1, 0].set_title('Square Root-Transformed Distribution')
axes[1, 0].set_xlabel('Sqrt(Burglaries)')
axes[1, 0].set_ylabel('Frequency')

# Box-Cox transformed distribution
sns.histplot(boxcox_transformed, kde=True, color=colors['boxcox'], ax=axes[1, 1], bins=30, edgecolor='black')
axes[1, 1].set_title('Box-Cox Transformed Distribution')
axes[1, 1].set_xlabel('Box-Cox(Burglaries)')
axes[1, 1].set_ylabel('Frequency')

# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.95])  # Leave space for the main title
plt.savefig('transformation.png', dpi=300)
plt.show()
```



The target variable is highly skewed, which can negatively affect the performance of machine learning models. To address this, two effective transformations can be applied: log transformation and box-cox transformation

In [ ]: