```
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression



import os

# Define the path to your folder
folder_path = '/content/drive/My Drive/Crime data'

# List all files in the folder to verify the folder path
files = os.listdir(folder_path)
print("Files in the 'crime data' folder:", files)
```

```
Files in the 'crime data' folder: ['2024-01-cleveland-street.csv', '2024-01-bedfordshire-street.csv', '2024-01-cambridgeshire-street.csv', '2024-01-avon-and-somerset-street.csv', '202
```

```
# Find the West Yorkshire street file
WestYorkshire_file = '2024-01-west-yorkshire-street.csv'

# Check if the Cleveland street file exists in the folder
if WestYorkshire_file in files:
    # Load the Cleveland street file into a DataFrame
    WestYorkshire_df = pd.read_csv(os.path.join(folder_path, WestYorkshire_file))

    # Display the first few rows of the Cleveland DataFrame
    print("Data from WestYorkshire File:")
    print(WestYorkshire_df.head())  # Display the first few rows
else:
    print(f"File {WestYorkshire_file} not found in the folder.")
```

```
Data from WestYorkshire File:
                                          Crime ID    Month  \
0  f12c8aea9112d5ea05990b630506b7b318f44e6fc37d40...  2024-01
1  9065faeb5a1a0a2a1b75de849f0674562397dd0ba1b9dc...  2024-01
2  b1604107cdec688fc0d4d2a1dab188cc75526e4cb8359f...  2024-01
3  e22a3152ae18583d5ad6cebae73721db6f1a421ecc71ea...  2024-01
4                                                NaN  2024-01

             Reported by           Falls within  Longitude   Latitude  \
0  West Yorkshire Police  West Yorkshire Police  -1.732999  53.545103
1  West Yorkshire Police  West Yorkshire Police  -1.878940  53.943744
2  West Yorkshire Police  West Yorkshire Police  -1.863081  53.938891
3  West Yorkshire Police  West Yorkshire Police  -1.884698  53.943938
4  West Yorkshire Police  West Yorkshire Police  -1.889080  53.946054

                         Location  LSOA code      LSOA name  \
0  On or near Wood Royd Hill Lane  E01007426  Barnsley 027D
1       On or near Cross End Fold  E01010646  Bradford 001A
2             On or near Millfold  E01010646  Bradford 001A
3         On or near Ridleys Fold  E01010647  Bradford 001B
4          On or near School Lane  E01010648  Bradford 001C

                      Crime type  \
```

```
0  Violence and sexual offences
1                   Shoplifting
2  Violence and sexual offences
3                      Burglary
4         Anti-social behaviour

                            Last outcome category  Context
0                    Unable to prosecute suspect      NaN
1  Investigation complete; no suspect identified      NaN
2                    Unable to prosecute suspect      NaN
3  Investigation complete; no suspect identified      NaN
4                                            NaN      NaN
```

```
# Drop the unnecessary columns
df_cleaned = WestYorkshire_df.drop(columns=['Crime ID','Context'])


# Find unique values in the 'Last outcome category' column
unique_outcomes = df_cleaned['Last outcome category'].unique()


# Print unique values
print("Unique values in 'Last outcome category':")
for outcome in unique_outcomes:
    print(outcome)
```

```
Unique values in 'Last outcome category':
Unable to prosecute suspect
Investigation complete; no suspect identified
nan
Status update unavailable
Awaiting court outcome
Further action is not in the public interest
Formal action is not in the public interest
Further investigation is not in the public interest
Offender given a caution
Local resolution
Action to be taken by another organisation
Suspect charged as part of another case
```

```
import pandas as pd

# Sample DataFrame (replace this with your actual DataFrame)
data = {'Last outcome category': [
    'Unable to prosecute suspect', 'Investigation complete; no suspect identified',
    None, 'Status update unavailable', 'Awaiting court outcome',
    'Further action is not in the public interest', 'Formal action is not in the public interest',
    'Further investigation is not in the public interest', 'Offender given a caution',
    'Local resolution', 'Action to be taken by another organisation',
    'Suspect charged as part of another case'
]}

df = pd.DataFrame(data)

# Define a mapping dictionary for outcomes
outcome_mapping = {
    'Unable to prosecute suspect': 'Unprosecuted',
    'Investigation complete; no suspect identified': 'Unresolved',
    'Status update unavailable': 'Pending',
```

```
    'Awaiting court outcome': 'Pending',
    'Further action is not in the public interest': 'No Further Action',
    'Formal action is not in the public interest': 'No Formal Action',
    'Further investigation is not in the public interest': 'No Investigation',
    'Offender given a caution': 'Cautioned',
    'Local resolution': 'Resolved',
    'Action to be taken by another organisation': 'Transferred',
    'Suspect charged as part of another case': 'Charged',
    None: 'Unknown'  # Handle NaN values or missing data
}

# Apply the mapping to the 'Last outcome category' column
df_cleaned['Simplified Outcome'] = df_cleaned['Last outcome category'].replace(outcome_mapping)
df_cleaned = df_cleaned.drop(columns=['Last outcome category'])
# Display the DataFrame with simplified outcomes
print(df_cleaned)
```

```
          Month            Reported by           Falls within  Longitude  \
0       2024-01  West Yorkshire Police  West Yorkshire Police  -1.732999
1       2024-01  West Yorkshire Police  West Yorkshire Police  -1.878940
2       2024-01  West Yorkshire Police  West Yorkshire Police  -1.863081
3       2024-01  West Yorkshire Police  West Yorkshire Police  -1.884698
4       2024-01  West Yorkshire Police  West Yorkshire Police  -1.889080
...         ...                    ...                    ...        ...
23805   2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23806   2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23807   2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23808   2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23809   2024-01  West Yorkshire Police  West Yorkshire Police        NaN

         Latitude                      Location  LSOA code      LSOA name  \
0       53.545103  On or near Wood Royd Hill Lane  E01007426  Barnsley 027D
1       53.943744       On or near Cross End Fold  E01010646  Bradford 001A
2       53.938891            On or near Millfold  E01010646  Bradford 001A
3       53.943938         On or near Ridleys Fold  E01010647  Bradford 001B
4       53.946054          On or near School Lane  E01010648  Bradford 001C
...           ...                             ...        ...            ...
23805         NaN                     No Location        NaN            NaN
23806         NaN                     No Location        NaN            NaN
23807         NaN                     No Location        NaN            NaN
23808         NaN                     No Location        NaN            NaN
23809         NaN                     No Location        NaN            NaN

                        Crime type Simplified Outcome
0      Violence and sexual offences       Unprosecuted
1                      Shoplifting         Unresolved
2      Violence and sexual offences       Unprosecuted
3                         Burglary         Unresolved
4             Anti-social behaviour            Unknown
...                             ...                ...
23805                 Other crime            Pending
23806                 Other crime            Pending
23807                 Other crime       Unprosecuted
23808                 Other crime       Unprosecuted
23809                 Other crime         Unresolved

[23810 rows x 10 columns]
```

```
df
```

	Last outcome category	Simplified Outcome
0	Unable to prosecute suspect	Unprosecuted
1	Investigation complete; no suspect identified	Unresolved
2	None	Unprosecuted
3	Status update unavailable	Unresolved
4	Awaiting court outcome	Unknown
5	Further action is not in the public interest	Unresolved
6	Formal action is not in the public interest	Unresolved
7	Further investigation is not in the public int...	Unknown
8	Offender given a caution	Unknown
9	Local resolution	Unknown
10	Action to be taken by another organisation	Unknown
11	Suspect charged as part of another case	Unresolved

```
df = df.drop(columns=['Last outcome category'])
concatenated_df = pd.concat([df, df_cleaned], ignore_index=True)
```

```
# Drop the unnecessary columns
df_clean = WestYorkshire_df.drop(columns=['Crime ID','Context'])
```

```
df_clean
```

	Month	Reported by	Falls within	Longitude	Latitude	Location	LSOA code	LSOA name	Crime type	Last outcome category
0	2024-01	West Yorkshire Police	West Yorkshire Police	-1.732999	53.545103	On or near Wood Royd Hill Lane	E01007426	Barnsley 027D	Violence and sexual offences	Unable to prosecute suspect
1	2024-01	West Yorkshire Police	West Yorkshire Police	-1.878940	53.943744	On or near Cross End Fold	E01010646	Bradford 001A	Shoplifting	Investigation complete; no suspect identified
2	2024-01	West Yorkshire Police	West Yorkshire Police	-1.863081	53.938891	On or near Millfold	E01010646	Bradford 001A	Violence and sexual offences	Unable to prosecute suspect
3	2024-01	West Yorkshire Police	West Yorkshire Police	-1.884698	53.943938	On or near Ridleys Fold	E01010647	Bradford 001B	Burglary	Investigation complete; no suspect identified
4	2024-01	West Yorkshire Police	West Yorkshire Police	-1.889080	53.946054	On or near School Lane	E01010648	Bradford 001C	Anti-social behaviour	NaN
...	...	...	...	...	...	...	...	...	...	...
23805	2024-01	West Yorkshire Police	West Yorkshire Police	NaN	NaN	No Location	NaN	NaN	Other crime	Status update unavailable
23806	2024-01	West Yorkshire Police	West Yorkshire Police	NaN	NaN	No Location	NaN	NaN	Other crime	Status update unavailable
23807	2024-01	West Yorkshire Police	West Yorkshire Police	NaN	NaN	No Location	NaN	NaN	Other crime	Unable to prosecute suspect
23808	2024-01	West Yorkshire Police	West Yorkshire Police	NaN	NaN	No Location	NaN	NaN	Other crime	Unable to prosecute suspect
23809	2024-01	West Yorkshire Police	West Yorkshire Police	NaN	NaN	No Location	NaN	NaN	Other crime	Investigation complete; no suspect identified

23810 rows × 10 columns

```
# Importing regular expression module
import re

# Function to remove unwanted phrases
def clean_location(text):
    # Use regex to remove phrases like "On or near"
    return re.sub(r'^On or near\s+', '', text)

# Apply the cleaning function to the Location column
df_clean['Location'] = df_clean['Location'].apply(clean_location)

# Display the cleaned DataFrame
print(df_clean)
```

```
         Month            Reported by           Falls within  Longitude  \
0      2024-01  West Yorkshire Police  West Yorkshire Police  -1.732999
1      2024-01  West Yorkshire Police  West Yorkshire Police  -1.878940
2      2024-01  West Yorkshire Police  West Yorkshire Police  -1.863081
3      2024-01  West Yorkshire Police  West Yorkshire Police  -1.884698
4      2024-01  West Yorkshire Police  West Yorkshire Police  -1.889080
...        ...                    ...                    ...        ...
23805  2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23806  2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23807  2024-01  West Yorkshire Police  West Yorkshire Police        NaN
23808  2024-01  West Yorkshire Police  West Yorkshire Police        NaN
```

```
23809  2024-01  West Yorkshire Police  West Yorkshire Police        NaN

       Latitude             Location  LSOA code      LSOA name  \
0      53.545103  Wood Royd Hill Lane  E01007426  Barnsley 027D
1      53.943744       Cross End Fold  E01010646  Bradford 001A
2      53.938891             Millfold  E01010646  Bradford 001A
3      53.943938         Ridleys Fold  E01010647  Bradford 001B
4      53.946054          School Lane  E01010648  Bradford 001C
...          ...                  ...        ...            ...
23805        NaN          No Location        NaN            NaN
23806        NaN          No Location        NaN            NaN
23807        NaN          No Location        NaN            NaN
23808        NaN          No Location        NaN            NaN
23809        NaN          No Location        NaN            NaN

                        Crime type  \
0      Violence and sexual offences
1                      Shoplifting
2      Violence and sexual offences
3                        Burglary
4              Anti-social behaviour
...                              ...
23805                  Other crime
23806                  Other crime
23807                  Other crime
23808                  Other crime
23809                  Other crime

                             Last outcome category
0                         Unable to prosecute suspect
1      Investigation complete; no suspect identified
2                         Unable to prosecute suspect
3      Investigation complete; no suspect identified
4                                                NaN
...                                              ...
23805                       Status update unavailable
23806                       Status update unavailable
23807                     Unable to prosecute suspect
23808                     Unable to prosecute suspect
23809  Investigation complete; no suspect identified

[23810 rows x 10 columns]
```

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23810 entries, 0 to 23809
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Month               23810 non-null  object
 1   Reported by         23810 non-null  object
 2   Falls within        23810 non-null  object
 3   Longitude           23494 non-null  float64
 4   Latitude            23494 non-null  float64
 5   Location            23810 non-null  object
 6   LSOA code           23494 non-null  object
 7   LSOA name           23494 non-null  object
 8   Crime type          23810 non-null  object
 9   Simplified Outcome  23810 non-null  object
dtypes: float64(2), object(8)
memory usage: 1.8+ MB
```

```
# Replace 'Unknown' with NaN
df_clean['Longitude'].replace('Unknown', np.nan, inplace=True)
df_clean['Latitude'].replace('Unknown', np.nan, inplace=True)



# Summary of NaN values
nan_summary = df_clean[['LSOA code', 'LSOA name']].isna().sum()
print("Number of NaN values in each column:")
print(nan_summary)
```

```
Number of NaN values in each column:
LSOA code    316
LSOA name    316
dtype: int64
```

```
# Fill missing values in 'LSOA code' with 'Unknown'
df_clean['LSOA code'] = df_clean['LSOA code'].fillna('Unknown')

# Fill missing values in 'LSOA name' with 'Unknown'
df_clean['LSOA name'] = df_clean['LSOA name'].fillna('Unknown')

# Verify that there are no missing values left
print(df_clean[['LSOA code', 'LSOA name']].isna().sum())
```

```
LSOA code    0
LSOA name    0
dtype: int64
```

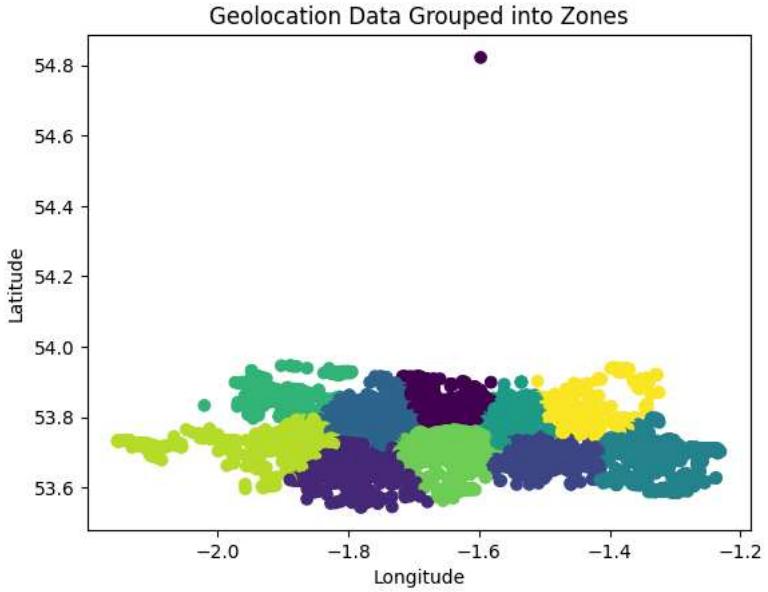
```
import pandas as pd

# Assuming df is your DataFrame

# Fill missing values in 'Longitude' with 0
df_clean['Longitude'] = df_clean['Longitude'].fillna(0)

# Fill missing values in 'Latitude' with 0
df_clean['Latitude'] = df_clean['Latitude'].fillna(0)



from sklearn.cluster import KMeans

# Prepare data for clustering
coords = df_clean[['Longitude', 'Latitude']].dropna()

# Define the number of clusters (zones)
n_clusters = 10
kmeans = KMeans(n_clusters=n_clusters)

# Fit the model
df_clean['Zone'] = kmeans.fit_predict(coords)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` e
  super()._check_params_vs_input(X, default_n_init=10)
```

```
# Drop rows with missing geolocation data
df_cleaned.dropna(subset=['Longitude', 'Latitude'], inplace=True)

# Ensure Latitude and Longitude are floats
df_cleaned['Longitude'] = df_cleaned['Longitude'].astype(float)
df_cleaned['Latitude'] = df_cleaned['Latitude'].astype(float)

# Prepare the data for clustering
geolocation_data = df_cleaned[['Latitude', 'Longitude']]

# Apply K-Means Clustering
# Choose the number of clusters (zones)
kmeans = KMeans(n_clusters=10, random_state=42)  # You can adjust the number of clusters
df_cleaned['Zone'] = kmeans.fit_predict(geolocation_data)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init` e
  super()._check_params_vs_input(X, default_n_init=10)
```

```
from sklearn.cluster import KMeans

# Define the number of clusters
n_clusters = 10

# Explicitly set the `n_init` parameter
kmeans = KMeans(n_clusters=n_clusters, n_init=10)  # You can set this to 10 or any other number you prefer

# Fit the model
df_clean['Zone'] = kmeans.fit_predict(df_clean[['Longitude', 'Latitude']].dropna())



import matplotlib.pyplot as plt

# (Optional) Plot the clusters to visualize the zones
plt.scatter(geolocation_data['Longitude'], geolocation_data['Latitude'], c=df_cleaned['Zone'], cmap='viridis')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Geolocation Data Grouped into Zones')
plt.show()
```




```
df_clean
```

```
# Replace Latitude and Longitude with Zone IDs in your main data
df_clean.drop(columns=['Latitude', 'Longitude'], inplace=True)

# Now, df_cleaned contains the Zone column instead of Latitude and Longitude
print(df_clean.head())
```

```
     Month            Reported by           Falls within             Location  \
0  2024-01  West Yorkshire Police  West Yorkshire Police  Wood Royd Hill Lane
1  2024-01  West Yorkshire Police  West Yorkshire Police       Cross End Fold
2  2024-01  West Yorkshire Police  West Yorkshire Police             Millfold
3  2024-01  West Yorkshire Police  West Yorkshire Police         Ridleys Fold
4  2024-01  West Yorkshire Police  West Yorkshire Police          School Lane

   LSOA code      LSOA name                    Crime type  \
0  E01007426  Barnsley 027D  Violence and sexual offences
1  E01010646  Bradford 001A                   Shoplifting
2  E01010646  Bradford 001A  Violence and sexual offences
3  E01010647  Bradford 001B                      Burglary
4  E01010648  Bradford 001C         Anti-social behaviour

                          Last outcome category  Zone
0                    Unable to prosecute suspect     8
1  Investigation complete; no suspect identified     7
2                    Unable to prosecute suspect     7
3  Investigation complete; no suspect identified     7
4                                           NaN     7
```

```
# Filter out the outlier data based on a defined latitude range
df_filtered = df_cleaned[(df_cleaned['Latitude'] > 53.6) & (df_cleaned['Latitude'] < 54.4)]
```

```
# Re-run K-means clustering on the filtered data
kmeans = KMeans(n_clusters=n_clusters, n_init=10)
df_filtered['Zone'] = kmeans.fit_predict(df_filtered[['Longitude', 'Latitude']])
```

```
<ipython-input-24-32431f9c212a>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered['Zone'] = kmeans.fit_predict(df_filtered[['Longitude', 'Latitude']])
```

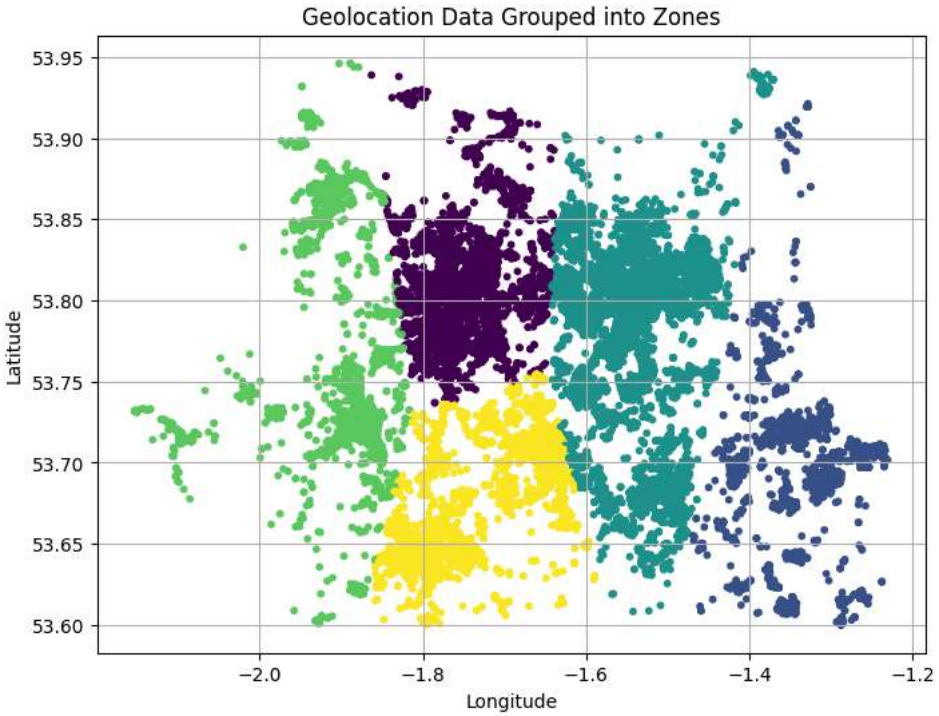
```
from sklearn.preprocessing import StandardScaler

# Normalize the data
scaler = StandardScaler()
df_cleaned[['Longitude', 'Latitude']] = scaler.fit_transform(df_cleaned[['Longitude', 'Latitude']])
```

```
from sklearn.metrics import silhouette_score

# Example: Finding optimal number of clusters using silhouette score
range_n_clusters = [2, 3, 4, 5, 6, 7, 8, 9, 10]
for n_clusters in range_n_clusters:
    kmeans = KMeans(n_clusters=n_clusters, n_init=10)
    cluster_labels = kmeans.fit_predict(df_cleaned[['Longitude', 'Latitude']])
    silhouette_avg = silhouette_score(df_cleaned[['Longitude', 'Latitude']], cluster_labels)
    print(f"For n_clusters = {n_clusters}, the average silhouette_score is: {silhouette_avg}")
```

```
For n_clusters = 2, the average silhouette_score is: 0.3887164310554597
For n_clusters = 3, the average silhouette_score is: 0.4212077534679748
For n_clusters = 4, the average silhouette_score is: 0.46808834384368136
For n_clusters = 5, the average silhouette_score is: 0.4623274284356523
For n_clusters = 6, the average silhouette_score is: 0.4647847614941451
For n_clusters = 7, the average silhouette_score is: 0.4761279896475356
For n_clusters = 8, the average silhouette_score is: 0.48448614203071805
For n_clusters = 9, the average silhouette_score is: 0.4975857782051197
For n_clusters = 10, the average silhouette_score is: 0.4442333324710481
```

```
# Define the number of clusters
n_clusters = 5  # You can choose the optimal number based on your analysis

# Perform K-means clustering
kmeans = KMeans(n_clusters=n_clusters, n_init=10)
df_filtered['Zone'] = kmeans.fit_predict(df_filtered[['Longitude', 'Latitude']])

# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(df_filtered['Longitude'], df_filtered['Latitude'], c=df_filtered['Zone'], cmap='viridis', s=10)
plt.title('Geolocation Data Grouped into Zones')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
plt.show()
```

```
<ipython-input-27-08b1163b5312>:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_filtered['Zone'] = kmeans.fit_predict(df_filtered[['Longitude', 'Latitude']])
```

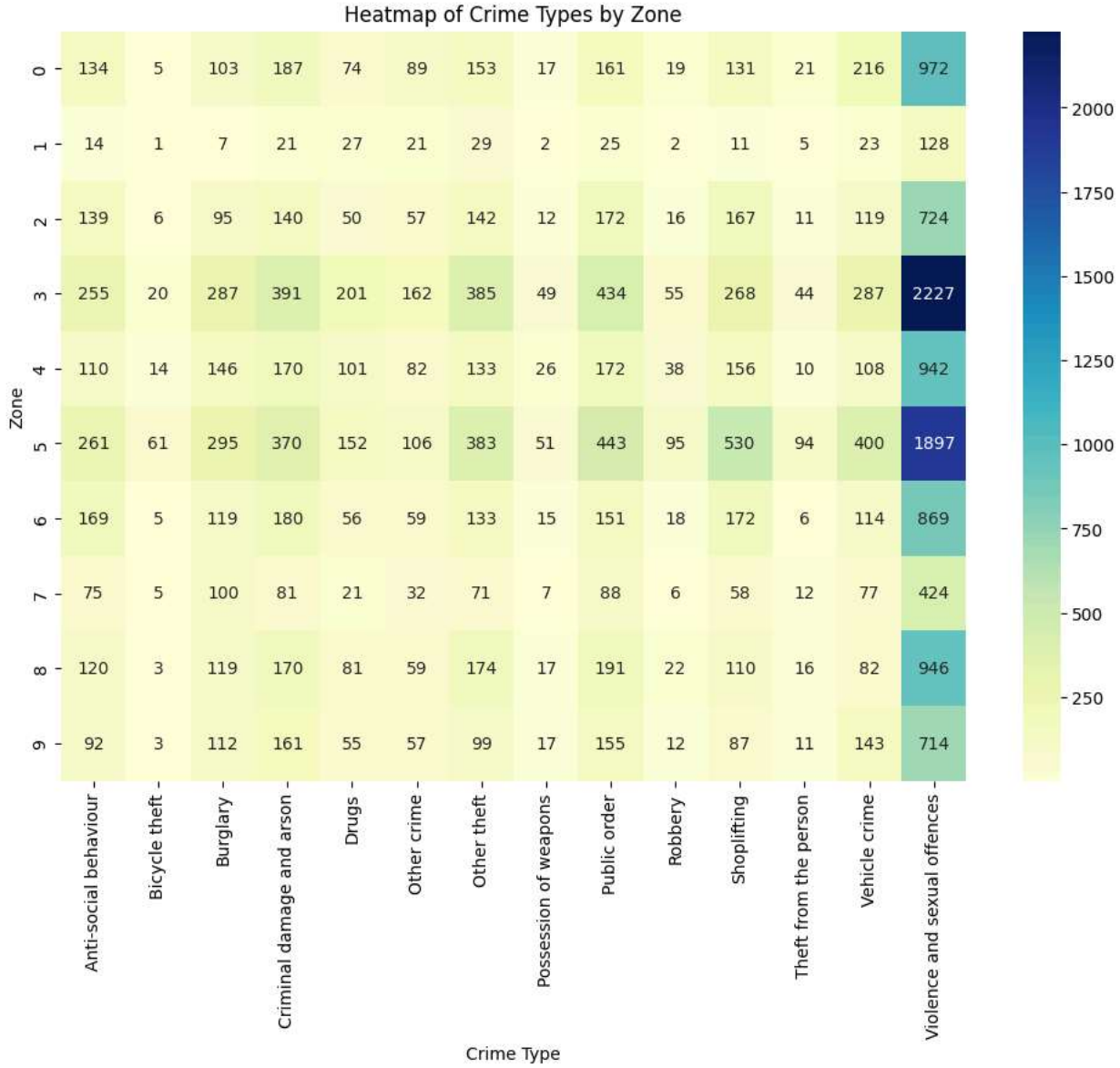



```
import pandas as pd

# Assuming df is your DataFrame

# Group the data by the 'Zone' column
zones_grouped = df_clean.groupby('Zone').agg({
    'Location': lambda x: list(x.unique()),      # Get unique locations in each zone
    'LSOA code': lambda x: list(x.unique()),     # Get unique LSOA codes in each zone
    'LSOA name': lambda x: list(x.unique()),     # Get unique LSOA names in each zone
    'Crime type': lambda x: list(x.unique())     # Get unique crime types in each zone
})

# Display the information for each zone
for zone, details in zones_grouped.iterrows():
    print(f"Zone {zone}:")
    print(f"  Locations: {details['Location']}")
    print(f"  LSOA Codes: {details['LSOA code']}")
    print(f"  LSOA Names: {details['LSOA name']}")
    print(f"  Crime Types: {details['Crime type']}")
    print("-" * 40)
```

```
  LSOA Names: ['Bradford 057A', 'Bradford 057C', 'Kirklees 001A', 'Kirklees 001B', 'Kirklees 001C', 'Kirklees 001D', 'Kirklees 001E', 'Kirklees 002A', 'Kirklees 002B', 'Kirklees 0
  Crime Types: ['Violence and sexual offences', 'Other theft', 'Vehicle crime', 'Other crime', 'Possession of weapons', 'Anti-social behaviour', 'Criminal damage and arson', 'Drug
----------------------------------------
Zone 1:
  Locations: ['No Location']
  LSOA Codes: ['Unknown']
  LSOA Names: ['Unknown']
  Crime Types: ['Anti-social behaviour', 'Bicycle theft', 'Burglary', 'Criminal damage and arson', 'Drugs', 'Other theft', 'Possession of weapons', 'Public order', 'Robbery', 'Sho
----------------------------------------
Zone 2:
  Locations: ['Wakefield Road', 'Heritage Court', 'The Bungalows', 'Spring Grove', 'Springfield Avenue', 'Grove House Drive', 'Station Court', 'Sunnymead', 'Willow Gardens', 'Park
  LSOA Codes: ['E01011121', 'E01011109', 'E01011111', 'E01011112', 'E01035049', 'E01011496', 'E01011634', 'E01011636', 'E01011492', 'E01011500', 'E01032496', 'E01035050', 'E010350
  LSOA Names: ['Kirklees 018A', 'Kirklees 054A', 'Kirklees 057B', 'Kirklees 057C', 'Leeds 099H', 'Leeds 100A', 'Leeds 100C', 'Leeds 100D', 'Leeds 101B', 'Leeds 105D', 'Leeds 105F'
  Crime Types: ['Criminal damage and arson', 'Violence and sexual offences', 'Anti-social behaviour', 'Other theft', 'Vehicle crime', 'Other crime', 'Public order', 'Burglary', 'D
----------------------------------------
Zone 3:
  Locations: ['Langford Court', 'Midgley Road', 'Langford Mews', 'Station Road', 'Manse Road', 'Norwood Avenue', 'Sun Lane', 'Park Row', 'Main Street', 'Victoria Road', 'Lawn Aven
  LSOA Codes: ['E01010767', 'E01010768', 'E01010769', 'E01010771', 'E01010770', 'E01010772', 'E01010773', 'E01010774', 'E01010573', 'E01010574', 'E01010575', 'E01010577', 'E010105
  LSOA Names: ['Bradford 003A', 'Bradford 003B', 'Bradford 003C', 'Bradford 003D', 'Bradford 005A', 'Bradford 005B', 'Bradford 005C', 'Bradford 005D', 'Bradford 013A', 'Bradford 0
  Crime Types: ['Violence and sexual offences', 'Criminal damage and arson', 'Public order', 'Anti-social behaviour', 'Burglary', 'Other theft', 'Vehicle crime', 'Other crime', 'B
----------------------------------------
Zone 4:
  Locations: ['Ainsty Drive', 'Maple Drive', 'Supermarket', 'Leven Gardens', 'Derwent Rise', 'Kings Meadow Close', 'Northfield Place', 'Rudgate Green', 'Tow Bank Close', 'Wood Lan
  LSOA Codes: ['E01011698', 'E01011699', 'E01011701', 'E01011697', 'E01011700', 'E01011704', 'E01011705', 'E01011707', 'E01011709', 'E01011710', 'E01011711', 'E01011712', 'E010115
  LSOA Names: ['Leeds 001A', 'Leeds 001B', 'Leeds 001C', 'Leeds 002A', 'Leeds 002B', 'Leeds 002C', 'Leeds 002D', 'Leeds 005A', 'Leeds 005B', 'Leeds 005C', 'Leeds 005D', 'Leeds 005
  Crime Types: ['Violence and sexual offences', 'Criminal damage and arson', 'Public order', 'Other crime', 'Anti-social behaviour', 'Drugs', 'Other theft', 'Burglary', 'Shoplifti
----------------------------------------
Zone 5:
  Locations: ['Parking Area', 'The Beeches', 'Leeds Road', 'Park Mount', 'Mulberry Chase', 'Swallow Close', 'Creskeld Lane', 'Bedlam Lane', 'Hall Rise Close', 'Breary Lane East',
  LSOA Codes: ['E01011569', 'E01011570', 'E01011576', 'E01032503', 'E01032504', 'E01011554', 'E01011563', 'E01011564', 'E01011567', 'E01011376', 'E01011377', 'E01011378', 'E010113
  LSOA Names: ['Leeds 004A', 'Leeds 007A', 'Leeds 007C', 'Leeds 007E', 'Leeds 007F', 'Leeds 012A', 'Leeds 012C', 'Leeds 012D', 'Leeds 012E', 'Leeds 013A', 'Leeds 013B', 'Leeds 013
  Crime Types: ['Violence and sexual offences', 'Burglary', 'Public order', 'Anti-social behaviour', 'Other theft', 'Vehicle crime', 'Drugs', 'Robbery', 'Criminal damage and arson
----------------------------------------
Zone 6:
  Locations: ['Manor Chase', 'Waddle Road', 'Newfield Crescent', 'Green Lane', 'Church Road', 'Jubilee Gardens', 'Glencoe Gardens', 'Glencoe Croft', 'Well Close', 'Garden Village'
  LSOA Codes: ['E01011297', 'E01011404', 'E01011392', 'E01011298', 'E01011299', 'E01011302', 'E01011303', 'E01011300', 'E01011301', 'E01011304', 'E01011305', 'E01011306', 'E010113
  LSOA Names: ['Leeds 030B', 'Leeds 077D', 'Leeds 087C', 'Leeds 088A', 'Leeds 088B', 'Leeds 088C', 'Leeds 088D', 'Leeds 089A', 'Leeds 089B', 'Leeds 089C', 'Leeds 089D', 'Leeds 089
  Crime Types: ['Burglary', 'Vehicle crime', 'Violence and sexual offences', 'Anti-social behaviour', 'Criminal damage and arson', 'Drugs', 'Other theft', 'Public order', 'Other c
----------------------------------------
Zone 7:
  Locations: ['Cross End Fold', 'Millfold', 'Ridleys Fold', 'School Lane', 'Moor Croft', 'Bus/Coach Station', 'Brook Street', 'Whitton Croft Road', 'Sedbergh Drive', 'Crossbeck Ro
  LSOA Codes: ['E01010646', 'E01010647', 'E01010648', 'E01010692', 'E01010691', 'E01010694', 'E01010695', 'E01010696', 'E01010697', 'E01010698', 'E01010699', 'E01010706', 'E010107
  LSOA Names: ['Bradford 001A', 'Bradford 001B', 'Bradford 001C', 'Bradford 001D', 'Bradford 002A', 'Bradford 002C', 'Bradford 002D', 'Bradford 002E', 'Bradford 002F', 'Bradford 0
  Crime Types: ['Shoplifting', 'Violence and sexual offences', 'Burglary', 'Anti-social behaviour', 'Vehicle crime', 'Criminal damage and arson', 'Other theft', 'Public order', 'O
----------------------------------------
Zone 8:
  Locations: ['Wood Royd Hill Lane', 'Bolehill Park', 'St Giles Road', 'Pond Farm Drive', 'Bramley View', 'Knowle Top Drive', 'Flowerlands', 'Upper Sutherland Road', 'The Grove',
  LSOA Codes: ['E01007426', 'E01010904', 'E01010906', 'E01010907', 'E01010909', 'E01010878', 'E01010879', 'E01010905', 'E01010908', 'E01010910', 'E01010990', 'E01010875', 'E010108
  LSOA Names: ['Barnsley 027D', 'Calderdale 011A', 'Calderdale 011B', 'Calderdale 011C', 'Calderdale 011D', 'Calderdale 015A', 'Calderdale 015B', 'Calderdale 015C', 'Calderdale 01
  Crime Types: ['Violence and sexual offences', 'Other theft', 'Anti-social behaviour', 'Vehicle crime', 'Criminal damage and arson', 'Other crime', 'Burglary', 'Possession of wea
----------------------------------------
Zone 9:
  Locations: ['Foreside Bottom Lane', 'Deep Lane', 'Weavers Court', 'Fountain Street', 'High Street', 'Gothic Street', 'Mount Pleasant Street', 'Spring Garden Street', 'High Croft
  LSOA Codes: ['E01010595', 'E01010795', 'E01010796', 'E01010752', 'E01010753', 'E01010754', 'E01010756', 'E01010758', 'E01010759', 'E01010760', 'E01010911', 'E01010912', 'E010109
  LSOA Names: ['Bradford 031D', 'Bradford 043B', 'Bradford 043C', 'Bradford 058A', 'Bradford 058B', 'Bradford 058C', 'Bradford 058D', 'Bradford 058E', 'Bradford 058F', 'Bradford 0
  Crime Types: ['Burglary', 'Anti-social behaviour', 'Violence and sexual offences', 'Drugs', 'Public order', 'Vehicle crime', 'Other theft', 'Possession of weapons', 'Other crime
----------------------------------------
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Assuming df_clean is your DataFrame
# Replace 'Crime type' and 'Zone' with your actual column names if different

# 1. Generate the Heatmap

# Create a pivot table to count occurrences of each crime type by zone
crime_type_counts = df_clean.pivot_table(index='Zone', columns='Crime type', aggfunc='size', fill_value=0)

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(crime_type_counts, cmap='YlGnBu', annot=True, fmt='d')
plt.title('Heatmap of Crime Types by Zone')
plt.xlabel('Crime Type')
plt.ylabel('Zone')
plt.show()

# 2. Group and Display Zone Details

# Group the data by the 'Zone' column and aggregate unique values
zones_grouped = df_clean.groupby('Zone').agg({
    'Location': lambda x: list(x.unique()),      # Get unique locations in each zone
    'LSOA code': lambda x: list(x.unique()),     # Get unique LSOA codes in each zone
    'LSOA name': lambda x: list(x.unique()),     # Get unique LSOA names in each zone
    'Crime type': lambda x: list(x.unique())     # Get unique crime types in each zone
})

# Display the information for each zone
for zone, details in zones_grouped.iterrows():
    print(f"Zone {zone}:")
    print(f"  Locations: {', '.join(details['Location'])}")
    print(f"  LSOA Codes: {', '.join(details['LSOA code'])}")
    print(f"  LSOA Names: {', '.join(details['LSOA name'])}")
    print(f"  Crime Types: {', '.join(details['Crime type'])}")
    print("-" * 40)
```



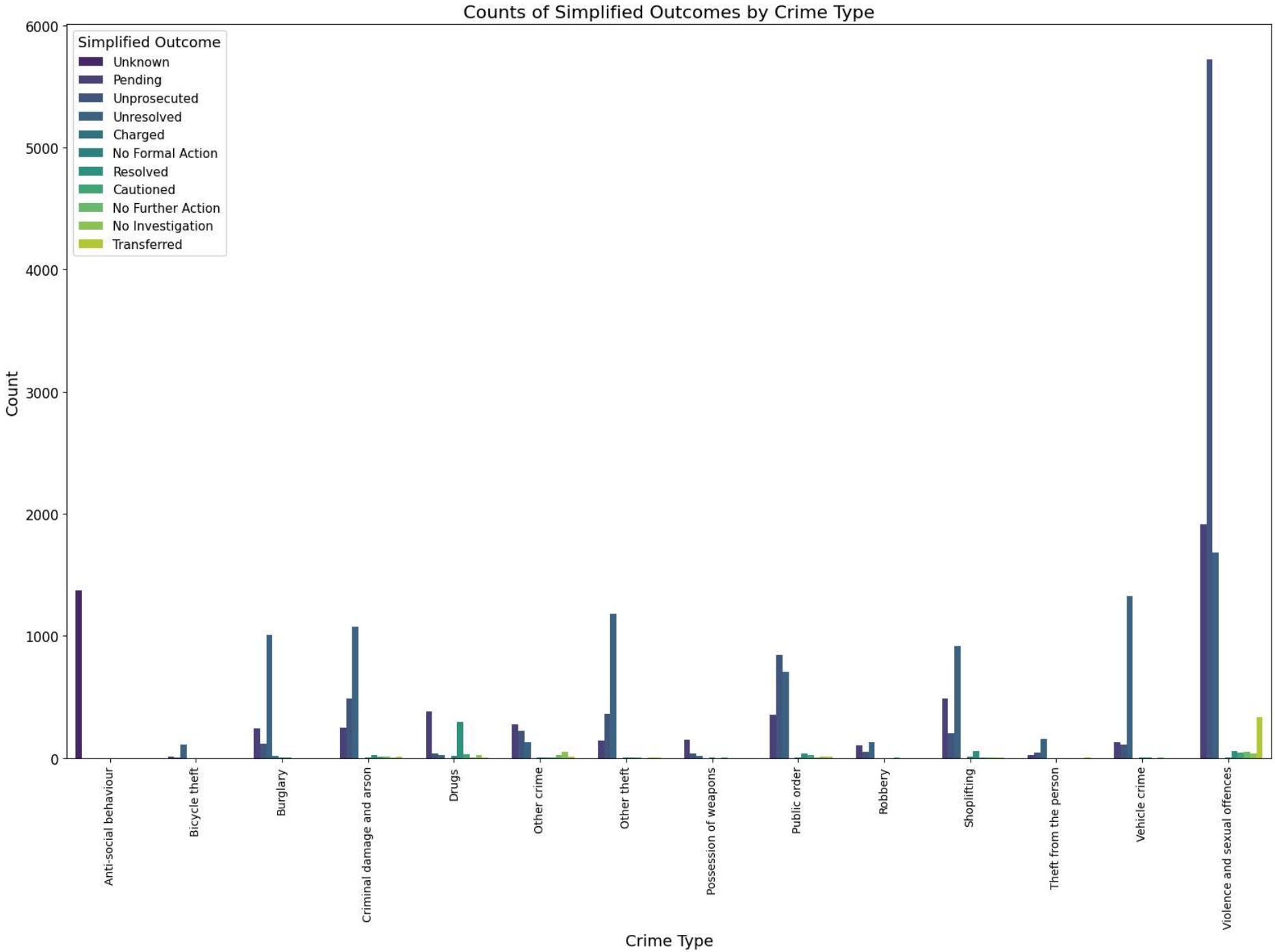

```
Zone 0:
  Locations: Montserrat Road, Moorcroft Terrace, Petrol Station, Bradford And Wakefield Road, New Lane, Eggleston Drive, Church Farm Close, Westgate Terrace, Cross Lane, Dawson Lane,
  LSOA Codes: E01010814, E01010816, E01011051, E01011055, E01011056, E01011057, E01011058, E01011048, E01011049, E01011050, E01011052, E01011053, E01011059, E01011062, E01011068, E016
  LSOA Names: Bradford 057A, Bradford 057C, Kirklees 001A, Kirklees 001B, Kirklees 001C, Kirklees 001D, Kirklees 001E, Kirklees 002A, Kirklees 002B, Kirklees 002C, Kirklees 002D, Kirk
  Crime Types: Violence and sexual offences, Other theft, Vehicle crime, Other crime, Possession of weapons, Anti-social behaviour, Criminal damage and arson, Drugs, Public order, Sho
----------------------------------------
Zone 1:
  Locations: No Location
  LSOA Codes: Unknown
  LSOA Names: Unknown
  Crime Types: Anti-social behaviour, Bicycle theft, Burglary, Criminal damage and arson, Drugs, Other theft, Possession of weapons, Public order, Robbery, Shoplifting, Theft from the
----------------------------------------
```

```
Zone 2:
  Locations: Wakefield Road, Heritage Court, The Bungalows, Spring Grove, Springfield Avenue, Grove House Drive, Station Court, Sunnymead, Willow Gardens, Park Avenue, Long Lane, Newl
  LSOA Codes: E01011121, E01011109, E01011111, E01011112, E01035049, E01011496, E01011634, E01011636, E01011492, E01011500, E01032496, E01035050, E01035051, E01035052, E01035053, E016
  LSOA Names: Kirklees 018A, Kirklees 054A, Kirklees 057B, Kirklees 057C, Leeds 099H, Leeds 100A, Leeds 100C, Leeds 100D, Leeds 101B, Leeds 105D, Leeds 105F, Leeds 105G, Leeds 105H, L
  Crime Types: Criminal damage and arson, Violence and sexual offences, Anti-social behaviour, Other theft, Vehicle crime, Other crime, Public order, Burglary, Drugs, Robbery, Theft f
----------------------------------------
Zone 3:
  Locations: Langford Court, Midgley Road, Langford Mews, Station Road, Manse Road, Norwood Avenue, Sun Lane, Park Row, Main Street, Victoria Road, Lawn Avenue, Corn Mill Lane, Richar
  LSOA Codes: E01010767, E01010768, E01010769, E01010771, E01010770, E01010772, E01010773, E01010774, E01010573, E01010574, E01010575, E01010577, E01010568, E01010569, E01010571, E016
  LSOA Names: Bradford 003A, Bradford 003B, Bradford 003C, Bradford 003D, Bradford 005A, Bradford 005B, Bradford 005C, Bradford 005D, Bradford 013A, Bradford 013B, Bradford 013C, Brac
  Crime Types: Violence and sexual offences, Criminal damage and arson, Public order, Anti-social behaviour, Burglary, Other theft, Vehicle crime, Other crime, Bicycle theft, Shoplift
----------------------------------------
Zone 4:
  Locations: Ainsty Drive, Maple Drive, Supermarket, Leven Gardens, Derwent Rise, Kings Meadow Close, Northfield Place, Rudgate Green, Tow Bank Close, Wood Lane, Northfields, Hampole
  LSOA Codes: E01011698, E01011699, E01011701, E01011697, E01011700, E01011704, E01011705, E01011707, E01011709, E01011710, E01011711, E01011712, E01011561, E01011696, E01011706, E016
  LSOA Names: Leeds 001A, Leeds 001B, Leeds 001C, Leeds 002A, Leeds 002B, Leeds 002C, Leeds 002D, Leeds 005A, Leeds 005B, Leeds 005C, Leeds 005D, Leeds 005E, Leeds 006A, Leeds 006B, L
  Crime Types: Violence and sexual offences, Criminal damage and arson, Public order, Other crime, Anti-social behaviour, Drugs, Other theft, Burglary, Shoplifting, Vehicle crime, Rob
----------------------------------------
Zone 5:
  Locations: Parking Area, The Beeches, Leeds Road, Park Mount, Mulberry Chase, Swallow Close, Creskeld Lane, Bedlam Lane, Hall Rise Close, Breary Lane East, Sports/Recreation Area, H
  LSOA Codes: E01011569, E01011570, E01011576, E01032503, E01032504, E01011554, E01011563, E01011564, E01011567, E01011376, E01011377, E01011378, E01011382, E01011556, E01011557, E016
  LSOA Names: Leeds 004A, Leeds 007A, Leeds 007C, Leeds 007E, Leeds 007F, Leeds 012A, Leeds 012C, Leeds 012D, Leeds 012E, Leeds 013A, Leeds 013B, Leeds 013C, Leeds 013D, Leeds 015A, L
  Crime Types: Violence and sexual offences, Burglary, Public order, Anti-social behaviour, Other theft, Vehicle crime, Drugs, Robbery, Criminal damage and arson, Other crime, Possess
----------------------------------------
Zone 6:
  Locations: Manor Chase, Waddle Road, Newfield Crescent, Green Lane, Church Road, Jubilee Gardens, Glencoe Gardens, Glencoe Croft, Well Close, Garden Village, Churchville Avenue, Pit
  LSOA Codes: E01011297, E01011404, E01011392, E01011298, E01011299, E01011302, E01011303, E01011300, E01011301, E01011304, E01011305, E01011306, E01011307, E01011308, E01011393, E016
  LSOA Names: Leeds 030B, Leeds 077D, Leeds 087C, Leeds 088A, Leeds 088B, Leeds 088C, Leeds 088D, Leeds 089A, Leeds 089B, Leeds 089C, Leeds 089D, Leeds 089E, Leeds 103A, Leeds 103B, L
  Crime Types: Burglary, Vehicle crime, Violence and sexual offences, Anti-social behaviour, Criminal damage and arson, Drugs, Other theft, Public order, Other crime, Robbery, Bicycle
----------------------------------------
Zone 7:
  Locations: Cross End Fold, Millfold, Ridleys Fold, School Lane, Moor Croft, Bus/Coach Station, Brook Street, Whitton Croft Road, Sedbergh Drive, Crossbeck Road, Wheatley Road, Wharf
  LSOA Codes: E01010646, E01010647, E01010648, E01010692, E01010691, E01010694, E01010695, E01010696, E01010697, E01010698, E01010699, E01010706, E01010707, E01010708, E01010709, E016
  LSOA Names: Bradford 001A, Bradford 001B, Bradford 001C, Bradford 001D, Bradford 002A, Bradford 002C, Bradford 002D, Bradford 002E, Bradford 002F, Bradford 002G, Bradford 002H, Brac
  Crime Types: Shoplifting, Violence and sexual offences, Burglary, Anti-social behaviour, Vehicle crime, Criminal damage and arson, Other theft, Public order, Other crime, Possessior
----------------------------------------
Zone 8:
  Locations: Wood Royd Hill Lane, Bolehill Park, St Giles Road, Pond Farm Drive, Bramley View, Knowle Top Drive, Flowerlands, Upper Sutherland Road, The Grove, George Street, Whitehal
  LSOA Codes: E01007426, E01010904, E01010906, E01010907, E01010909, E01010878, E01010879, E01010905, E01010908, E01010910, E01010990, E01010875, E01010876, E01010877, E01010880, E016
  LSOA Names: Barnsley 027D, Calderdale 011A, Calderdale 011B, Calderdale 011C, Calderdale 011D, Calderdale 015A, Calderdale 015B, Calderdale 015C, Calderdale 015D, Calderdale 015E, C
  Crime Types: Violence and sexual offences, Other theft, Anti-social behaviour, Vehicle crime, Criminal damage and arson, Other crime, Burglary, Possession of weapons, Public order,
----------------------------------------
Zone 9:
  Locations: Foreside Bottom Lane, Deep Lane, Weavers Court, Fountain Street, High Street, Gothic Street, Mount Pleasant Street, Spring Garden Street, High Croft, Granby Street, Unior
  LSOA Codes: E01010595, E01010795, E01010796, E01010752, E01010753, E01010754, E01010756, E01010758, E01010759, E01010760, E01010911, E01010912, E01010913, E01010915, E01010914, E016
  LSOA Names: Bradford 031D, Bradford 043B, Bradford 043C, Bradford 058A, Bradford 058B, Bradford 058C, Bradford 058D, Bradford 058E, Bradford 058F, Bradford 058G, Calderdale 001A, Ca
  Crime Types: Burglary, Anti-social behaviour, Violence and sexual offences, Drugs, Public order, Vehicle crime, Other theft, Possession of weapons, Other crime, Theft from the persc
----------------------------------------
```

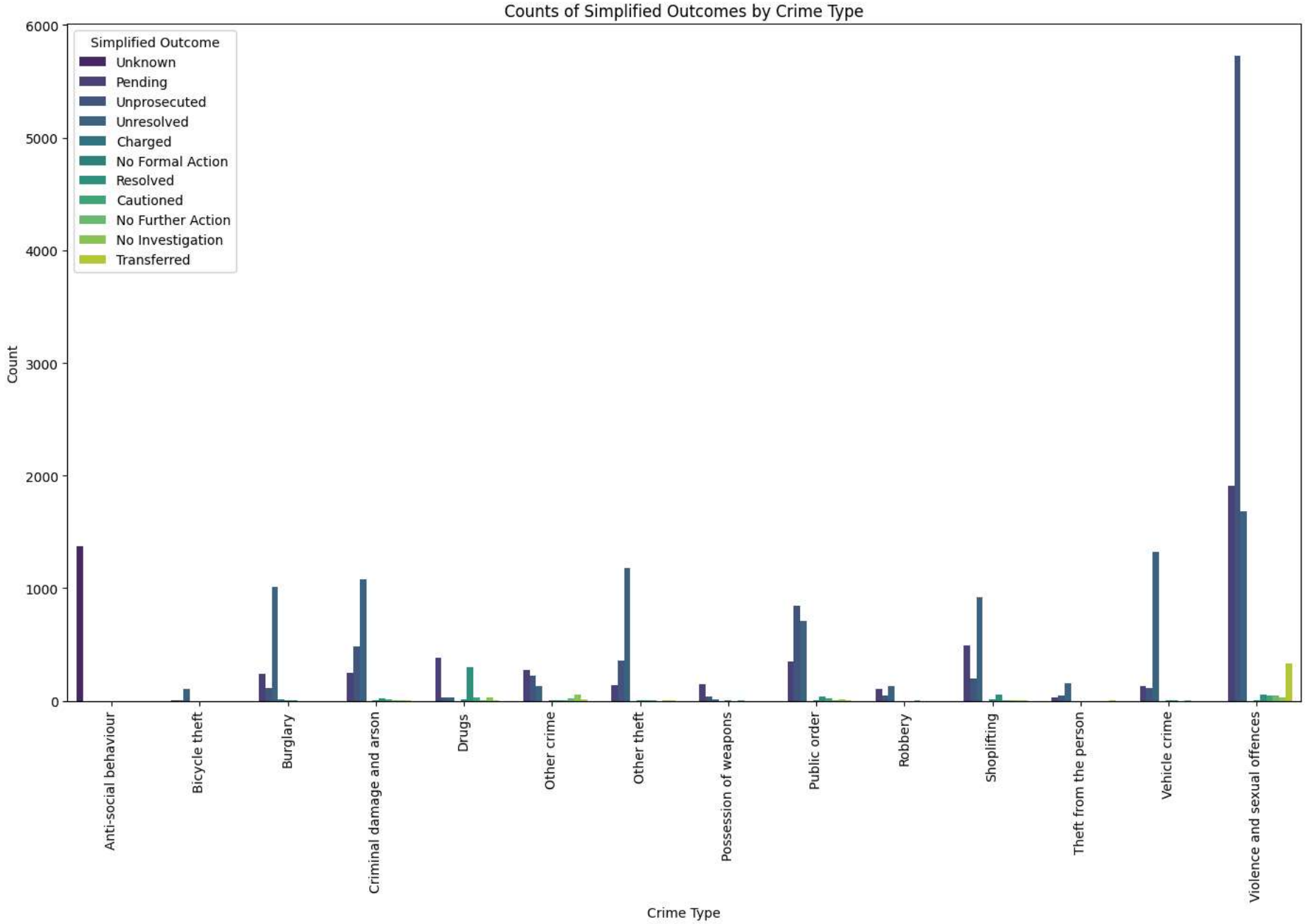
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame
# Replace 'Simplified Outcome' with the actual column name if different

# Count occurrences of each outcome in the 'Simplified Outcome' column
outcome_counts = df_clean['Simplified Outcome'].value_counts()

# Create a DataFrame for plotting
outcome_df = outcome_counts.reset_index()
outcome_df.columns = ['Simplified Outcome', 'Count']

# Set up the plot
plt.figure(figsize=(12, 8))
sns.barplot(data=outcome_df, x='Simplified Outcome', y='Count', palette='viridis')

# Customize the plot
plt.title('Counts of Simplified Outcomes')
plt.xlabel('Simplified Outcome')
plt.ylabel('Count')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()

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

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3790         try:
-> 3791             return self._engine.get_loc(casted_key)
   3792         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'Simplified Outcome'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                          2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3796             ):
   3797                 raise InvalidIndexError(key)
-> 3798             raise KeyError(key) from err
   3799         except TypeError:
   3800             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'Simplified Outcome'
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create a DataFrame with counts of each simplified outcome for each crime type
outcome_crime_counts = df_clean.groupby(['Crime type', 'Simplified Outcome']).size().reset_index(name='Count')

# Set up the plot with increased size
plt.figure(figsize=(16, 12))  # Larger figure size

# Create the bar plot
sns.barplot(data=outcome_crime_counts, x='Crime type', y='Count', hue='Simplified Outcome', palette='viridis')

# Customize the plot
plt.title('Counts of Simplified Outcomes by Crime Type', fontsize=16)
plt.xlabel('Crime Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=90, fontsize=10)  # Rotate x-axis labels for better readability
plt.yticks(fontsize=12)
plt.legend(title='Simplified Outcome', title_fontsize='13', fontsize='11')
plt.tight_layout()

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


Counts of Simplified Outcomes by Crime Type
Simplified Outcome
Unknown
Pending
Unprosecuted
Unresolved
Charged
No Formal Action
Resolved
Cautioned
No Further Action
No Investigation
Transferred
Count
0
1000
2000
3000
4000
5000
6000
Anti-social behaviour
Bicycle theft
Burglary
Criminal damage and arson
Drugs
Other crime
Other theft
Possession of weapons
Public order
Robbery
Shoplifting
Theft from the person
Vehicle crime
Violence and sexual offences
Crime Type

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df is your DataFrame
# Replace 'Simplified Outcome' and 'Crime type' with actual column names if different

# Create a DataFrame with counts of each simplified outcome for each crime type
outcome_crime_counts = df_clean.groupby(['Crime type', 'Simplified Outcome']).size().reset_index(name='Count')

# Set up the plot
plt.figure(figsize=(14, 10))
sns.barplot(data=outcome_crime_counts, x='Crime type', y='Count', hue='Simplified Outcome', palette='viridis')

# Customize the plot
plt.title('Counts of Simplified Outcomes by Crime Type')
plt.xlabel('Crime Type')
plt.ylabel('Count')
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.legend(title='Simplified Outcome')
plt.tight_layout()

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


Counts of Simplified Outcomes by Crime Type
Simplified Outcome
Unknown
Pending
Unprosecuted
Unresolved
Charged
No Formal Action
Resolved
Cautioned
No Further Action
No Investigation
Transferred
Count
0
1000
2000
3000
4000
5000
6000
Anti-social behaviour
Bicycle theft
Burglary
Criminal damage and arson
Drugs
Other crime
Other theft
Possession of weapons
Public order
Robbery
Shoplifting
Theft from the person
Vehicle crime
Violence and sexual offences
Crime Type

```
df_clean
```

	Month	Reported by	Falls within	Location	LSOA code	LSOA name	Crime type	Simplified Outcome	Zone
0	2024-01	West Yorkshire Police	West Yorkshire Police	Wood Royd Hill Lane	E01007426	Barnsley 027D	Violence and sexual offences	Unprosecuted	6
1	2024-01	West Yorkshire Police	West Yorkshire Police	Cross End Fold	E01010646	Bradford 001A	Shoplifting	Unresolved	9
2	2024-01	West Yorkshire Police	West Yorkshire Police	Millfold	E01010646	Bradford 001A	Violence and sexual offences	Unprosecuted	9
3	2024-01	West Yorkshire Police	West Yorkshire Police	Ridleys Fold	E01010647	Bradford 001B	Burglary	Unresolved	9
4	2024-01	West Yorkshire Police	West Yorkshire Police	School Lane	E01010648	Bradford 001C	Anti-social behaviour	Unknown	9
...	...	...	...	...	...	...	...	...	...
23805	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Pending	1
23806	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Pending	1
23807	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Unprosecuted	1
23808	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Unprosecuted	1
23809	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Unresolved	1

23810 rows × 9 columns

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Define the zones of interest
zones_of_interest = ['Zone 0', 'Zone 1', 'Zone 2', 'Zone 3', 'Zone 4', 'Zone 5', 'Zone 6', 'Zone 7', 'Zone 8', 'Zone 9']

# Filter the DataFrame for these zones
filtered_df = df_clean[df_clean['Zone'].isin(zones_of_interest)]

# Print the shape and first few rows of filtered_df to debug
print(f"Filtered DataFrame shape: {filtered_df.shape}")
print(filtered_df.head())

# Create a DataFrame for crime types count per zone
crime_columns = [col for col in df.columns if col.startswith('Crime type_')]
print(f"Crime columns: {crime_columns}")

# Check if crime_columns is not empty
if crime_columns:
    crime_type_counts = filtered_df.groupby('Zone')[crime_columns].sum()

    # Print the shape and first few rows of crime_type_counts to debug
    print(f"Crime type counts shape: {crime_type_counts.shape}")
    print(crime_type_counts.head())

    # Transpose for better heatmap representation
    crime_type_counts = crime_type_counts.T

    # Plot heatmap if crime_type_counts is not empty
    if not crime_type_counts.empty:
        plt.figure(figsize=(12, 8))
        sns.heatmap(crime_type_counts, cmap='YlGnBu', annot=True, fmt='d')
        plt.title('Heatmap of Crime Types by Zone')
        plt.xlabel('Zone')
```

```
            plt.ylabel('Crime Type')
            plt.show()
        else:
            print("Error: Crime type counts DataFrame is empty.")
    else:
        print("Error: No crime type columns found in the DataFrame.")
```

```
Filtered DataFrame shape: (0, 9)
Empty DataFrame
Columns: [Month, Reported by, Falls within, Location, LSOA code, LSOA name, Crime type, Simplified Outcome, Zone]
Index: []
Crime columns: []
Error: No crime type columns found in the DataFrame.
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming df is your DataFrame
# Define the zones of interest
zones_of_interest = ['Zone 0', 'Zone 1', 'Zone 2', 'Zone 3', 'Zone 4', 'Zone 5', 'Zone 6', 'Zone 7', 'Zone 8', 'Zone 9']

# Filter the DataFrame for these zones
filtered_df = df_clean[df_clean['Zone'].isin(zones_of_interest)]

# Create a DataFrame for crime types count per zone
# This assumes the columns are in a format where each crime type column is a boolean indicating presence
crime_columns = [col for col in df.columns if col.startswith('Crime type_')]
crime_type_counts = filtered_df.groupby('Zone')[crime_columns].sum()

# Transpose for better heatmap representation
crime_type_counts = crime_type_counts.T

# Plot heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(crime_type_counts, cmap='YlGnBu', annot=True, fmt='d')
plt.title('Heatmap of Crime Types by Zone')
plt.xlabel('Zone')
plt.ylabel('Crime Type')
plt.show()
```

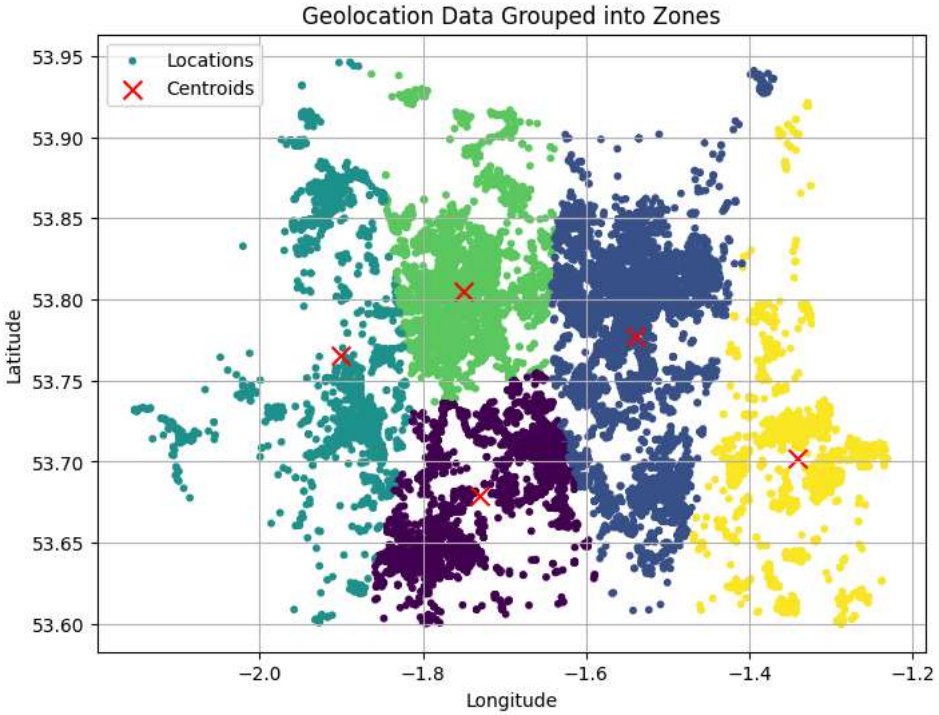
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-26-456ac754f9e0> in <cell line: 22>()
     20 # Plot heatmap
     21 plt.figure(figsize=(12, 8))
---> 22 sns.heatmap(crime_type_counts, cmap='YlGnBu', annot=True, fmt='d')
     23 plt.title('Heatmap of Crime Types by Zone')
     24 plt.xlabel('Zone')

                                 3 frames
/usr/local/lib/python3.10/dist-packages/numpy/lib/nanfunctions.py in nanmin(a, axis, out, keepdims, initial, where)
    341         # Fast, but not safe for subclasses of ndarray, or object arrays,
    342         # which do not implement isnan (gh-9009), or fmin correctly (gh-8975)
--> 343         res = np.fmin.reduce(a, axis=axis, out=out, **kwargs)
    344         if np.isnan(res).any():
    345             warnings.warn("All-NaN slice encountered", RuntimeWarning,

ValueError: zero-size array to reduction operation fmin which has no identity

<Figure size 1200x800 with 0 Axes>
```

कोडिंग शुरू करें या एआई की मदद से कोड जनरेट करें.

```
import pandas as pd

# Assuming df is your DataFrame

# Find unique zone values
unique_zones = df_filtered['Zone'].unique()
print(f"Unique zones: {unique_zones}")
```

```
Unique zones: [2 3 0 1 4]
```

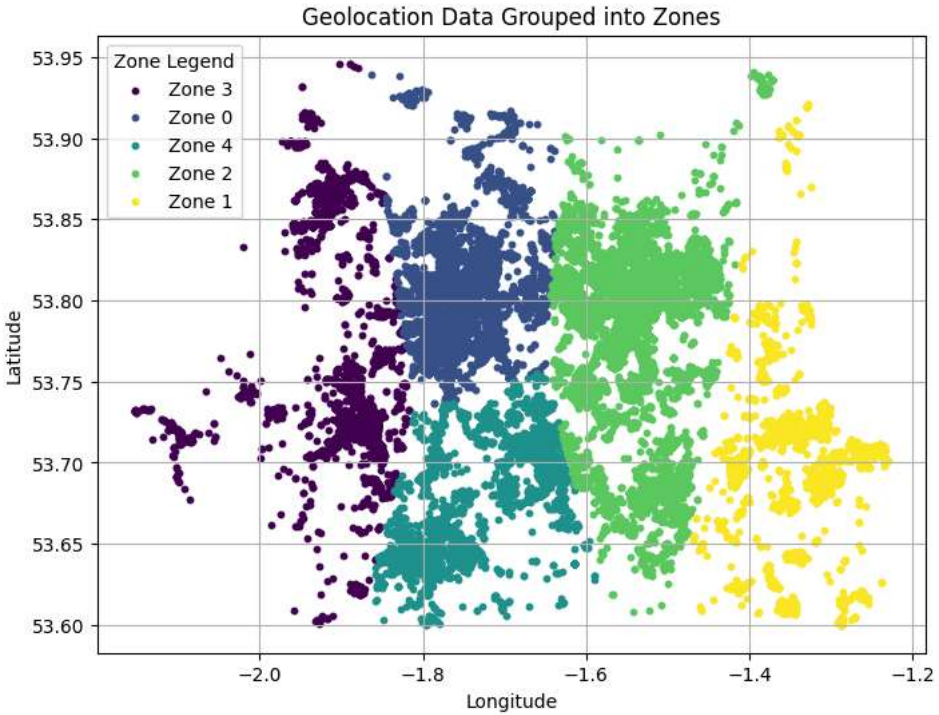
```
# Plot the clusters
plt.figure(figsize=(8, 6))
plt.scatter(df_filtered['Longitude'], df_filtered['Latitude'], c=df_filtered['Zone'], cmap='viridis', s=10, label='Locations')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', color='red', s=100, label='Centroids')  # Mark centroids

plt.title('Geolocation Data Grouped into Zones')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
plt.legend()
plt.show()
```




```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import matplotlib.cm as cm


# Extract unique zones from the dataset
unique_zones = df_filtered['Zone'].unique()

# Dynamically assign colors to each unique zone
colors = cm.get_cmap('viridis', len(unique_zones))  # Using a colormap for consistent colors
zone_colors = {zone: colors(i) for i, zone in enumerate(unique_zones)}

# Plot the clusters
plt.figure(figsize=(8, 6))

# Plot each zone with its dynamically assigned color
for zone, color in zone_colors.items():
    zone_data = df_filtered[df_filtered['Zone'] == zone]
    plt.scatter(zone_data['Longitude'], zone_data['Latitude'], color=color, label=f'Zone {zone}', s=10)



plt.title('Geolocation Data Grouped into Zones')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.grid(True)
```

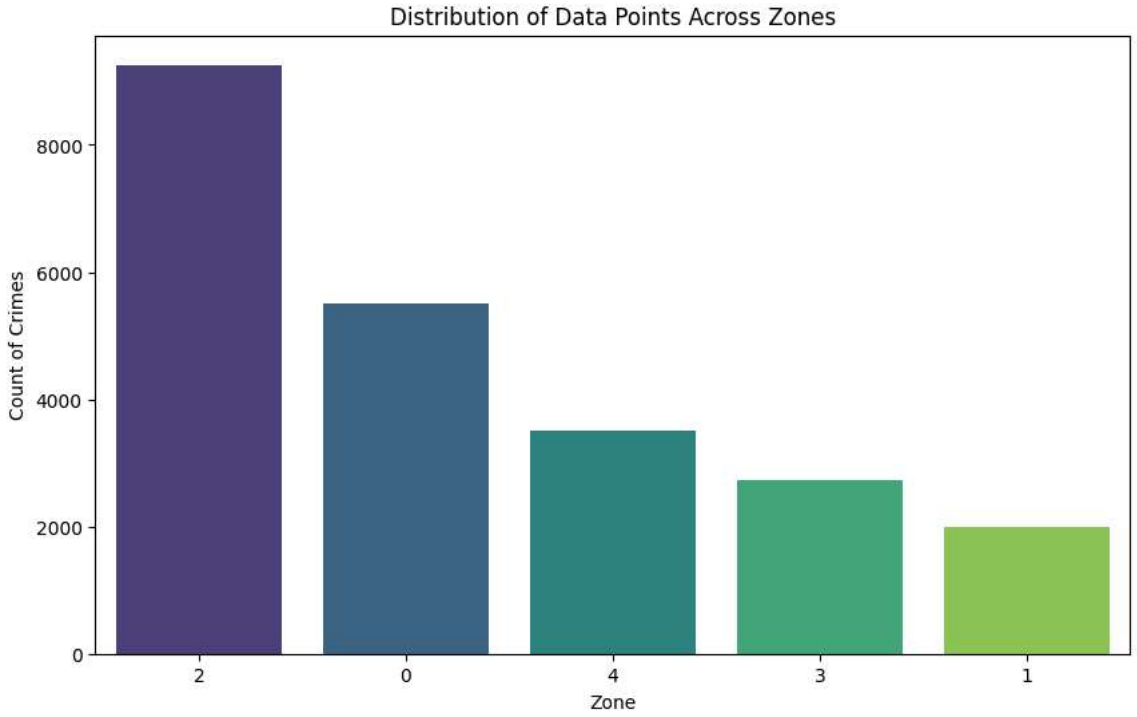
```
plt.legend(title='Zone Legend')
plt.show()
```

```
<ipython-input-31-09b1c53a5ed9>:11: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib 3.7 and will be removed two minor releases later. Use ``matplotlib
  colors = cm.get_cmap('viridis', len(unique_zones))  # Using a colormap for consistent colors
```




```
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming 'df' is your DataFrame containing 'Zone' and 'Crime Type' columns
plt.figure(figsize=(10, 6))
sns.countplot(data=df_filtered, x='Zone', order=df_filtered['Zone'].value_counts().index, palette='viridis')
plt.title('Distribution of Data Points Across Zones')
plt.xlabel('Zone')
plt.ylabel('Count of Crimes')
plt.show()
```

```
<ipython-input-32-3701c849d5b5>:6: FutureWarning:

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

  sns.countplot(data=df_filtered, x='Zone', order=df_filtered['Zone'].value_counts().index, palette='viridis')
```




```
# Import necessary libraries
import pandas as pd

# Calculate summary statistics for Latitude and Longitude within each zone
zone_stats = df_filtered.groupby('Zone')[['Latitude', 'Longitude']].agg(['mean', 'median', 'std'])
zone_stats.columns = ['Mean Latitude', 'Median Latitude', 'Std Latitude', 'Mean Longitude', 'Median Longitude', 'Std Longitude']
print(zone_stats)
```

```
       Mean Latitude  Median Latitude  Std Latitude  Mean Longitude  \
Zone
0          53.804642        53.797665      0.033216       -1.750717
1          53.702083        53.702691      0.054890       -1.341234
2          53.777020        53.796525      0.055257       -1.539263
3          53.765235        53.734158      0.068682       -1.900381
4          53.678779        53.680834      0.034709       -1.730840

       Median Longitude  Std Longitude
Zone
0             -1.754121       0.042846
1             -1.344330       0.048355
2             -1.540619       0.048883
3             -1.888479       0.057791
4             -1.748160       0.067663
```

```
pip install pysal
```

```
Collecting pysal
  Downloading pysal-24.7-py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: beautifulsoup4>=4.10 in /usr/local/lib/python3.10/dist-packages (from pysal) (4.12.3)
Requirement already satisfied: geopandas>=0.10.0 in /usr/local/lib/python3.10/dist-packages (from pysal) (0.14.4)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.10/dist-packages (from pysal) (1.26.4)
Requirement already satisfied: packaging>=22 in /usr/local/lib/python3.10/dist-packages (from pysal) (24.1)
Requirement already satisfied: pandas>=1.4 in /usr/local/lib/python3.10/dist-packages (from pysal) (2.1.4)
Requirement already satisfied: platformdirs>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from pysal) (4.3.2)
Requirement already satisfied: requests>=2.27 in /usr/local/lib/python3.10/dist-packages (from pysal) (2.32.3)
Requirement already satisfied: scipy>=1.8 in /usr/local/lib/python3.10/dist-packages (from pysal) (1.13.1)
Requirement already satisfied: shapely>=2.0.1 in /usr/local/lib/python3.10/dist-packages (from pysal) (2.0.6)
Requirement already satisfied: scikit-learn>=1.1 in /usr/local/lib/python3.10/dist-packages (from pysal) (1.3.2)
Collecting libpysal>=4.12.0 (from pysal)
  Downloading libpysal-4.12.1-py3-none-any.whl.metadata (4.8 kB)
Collecting access>=1.1.9 (from pysal)
  Downloading access-1.1.9-py3-none-any.whl.metadata (2.4 kB)
Collecting esda>=2.6.0 (from pysal)
  Downloading esda-2.6.0-py3-none-any.whl.metadata (2.0 kB)
Collecting giddy>=2.3.5 (from pysal)
  Downloading giddy-2.3.5-py3-none-any.whl.metadata (6.4 kB)
Collecting inequality>=1.0.1 (from pysal)
  Downloading inequality-1.0.1-py3-none-any.whl.metadata (3.3 kB)
Collecting pointpats>=2.5.0 (from pysal)
  Downloading pointpats-2.5.0-py3-none-any.whl.metadata (4.7 kB)
Collecting segregation>=2.5 (from pysal)
  Downloading segregation-2.5-py3-none-any.whl.metadata (2.2 kB)
Collecting spaghetti>=1.7.6 (from pysal)
  Downloading spaghetti-1.7.6-py3-none-any.whl.metadata (12 kB)
Collecting mgwr>=2.2.1 (from pysal)
  Downloading mgwr-2.2.1-py3-none-any.whl.metadata (1.5 kB)
Collecting momepy>=0.7.2 (from pysal)
  Downloading momepy-0.8.0-py3-none-any.whl.metadata (1.4 kB)
Collecting spglm>=1.1.0 (from pysal)
  Downloading spglm-1.1.0-py3-none-any.whl.metadata (3.9 kB)
Collecting spint>=1.0.7 (from pysal)
  Downloading spint-1.0.7.tar.gz (28 kB)
  Preparing metadata (setup.py) ... done
Collecting spreg>=1.5.0 (from pysal)
  Downloading spreg-1.6.1-py3-none-any.whl.metadata (1.7 kB)
Collecting spvcm>=0.3.0 (from pysal)
  Downloading spvcm-0.3.0.tar.gz (5.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.7/5.7 MB 38.8 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting tobler>=0.11.2 (from pysal)
  Downloading tobler-0.11.3-py3-none-any.whl.metadata (1.9 kB)
Collecting mapclassify>=2.7.0 (from pysal)
  Downloading mapclassify-2.8.0-py3-none-any.whl.metadata (2.8 kB)
Collecting splot>=1.1.5.post1 (from pysal)
  Downloading splot-1.1.7-py3-none-any.whl.metadata (8.9 kB)
Collecting spopt>=0.6.1 (from pysal)
  Downloading spopt-0.6.1-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.10->pysal) (2.6)
Requirement already satisfied: fiona>=1.8.21 in /usr/local/lib/python3.10/dist-packages (from geopandas>=0.10.0->pysal) (1.10.0)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from geopandas>=0.10.0->pysal) (3.6.1)
Collecting quantecon>=0.4.7 (from giddy>=2.3.5->pysal)
  Downloading quantecon-0.7.2-py3-none-any.whl.metadata (4.9 kB)
Requirement already satisfied: networkx>=2.7 in /usr/local/lib/python3.10/dist-packages (from mapclassify>=2.7.0->pysal) (3.3)
```

```
df_filtered
```

	Month	Reported by	Falls within	Longitude	Latitude	Location	LSOA code	LSOA name	Crime type	Simplified Outcome	Zone
1	2024-01	West Yorkshire Police	West Yorkshire Police	-1.878940	53.943744	On or near Cross End Fold	E01010646	Bradford 001A	Shoplifting	Unresolved	3
2	2024-01	West Yorkshire Police	West Yorkshire Police	-1.863081	53.938891	On or near Millfold	E01010646	Bradford 001A	Violence and sexual offences	Unprosecuted	0
3	2024-01	West Yorkshire Police	West Yorkshire Police	-1.884698	53.943938	On or near Ridleys Fold	E01010647	Bradford 001B	Burglary	Unresolved	3
4	2024-01	West Yorkshire Police	West Yorkshire Police	-1.889080	53.946054	On or near School Lane	E01010648	Bradford 001C	Anti-social behaviour	Unknown	3
5	2024-01	West Yorkshire Police	West Yorkshire Police	-1.902777	53.945985	On or near Moor Croft	E01010648	Bradford 001C	Vehicle crime	Unresolved	3
...	...	...	...	...	...	...	...	...	...	...	...
23358	2024-01	West Yorkshire Police	West Yorkshire Police	-1.270903	53.604787	On or near A638	E01011860	Wakefield 044A	Vehicle crime	Unresolved	1
23359	2024-01	West Yorkshire Police	West Yorkshire Police	-1.287178	53.604022	On or near Sandford Road	E01011860	Wakefield 044A	Violence and sexual offences	Unprosecuted	1
23360	2024-01	West Yorkshire Police	West Yorkshire Police	-1.286335	53.604691	On or near Melton Close	E01011860	Wakefield 044A	Violence and sexual offences	Unprosecuted	1
23363	2024-01	West Yorkshire Police	West Yorkshire Police	-1.287108	53.604588	On or near Newbury Drive	E01011860	Wakefield 044A	Violence and sexual offences	Unprosecuted	1
23364	2024-01	West Yorkshire Police	West Yorkshire Police	-1.287178	53.604022	On or near Sandford Road	E01011860	Wakefield 044A	Violence and sexual offences	Unprosecuted	1

23013 rows × 11 columns

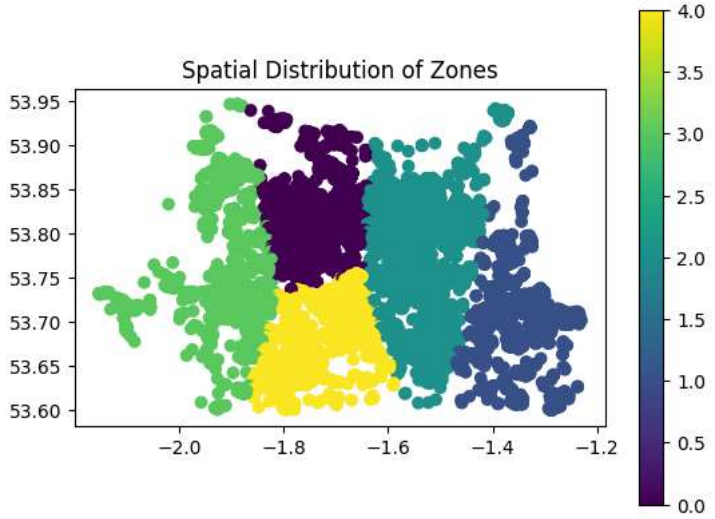
```
pip install geopandas esda libpysal
```

```
Requirement already satisfied: geopandas in /usr/local/lib/python3.10/dist-packages (0.14.4)
Requirement already satisfied: esda in /usr/local/lib/python3.10/dist-packages (2.6.0)
Requirement already satisfied: libpysal in /usr/local/lib/python3.10/dist-packages (4.12.1)
Requirement already satisfied: fiona>=1.8.21 in /usr/local/lib/python3.10/dist-packages (from geopandas) (1.10.0)
Requirement already satisfied: numpy>=1.22 in /usr/local/lib/python3.10/dist-packages (from geopandas) (1.26.4)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from geopandas) (24.1)
Requirement already satisfied: pandas>=1.4.0 in /usr/local/lib/python3.10/dist-packages (from geopandas) (2.1.4)
Requirement already satisfied: pyproj>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from geopandas) (3.6.1)
Requirement already satisfied: shapely>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from geopandas) (2.0.6)
Requirement already satisfied: scikit-learn>=1.2 in /usr/local/lib/python3.10/dist-packages (from esda) (1.3.2)
Requirement already satisfied: scipy>=1.9 in /usr/local/lib/python3.10/dist-packages (from esda) (1.13.1)
Requirement already satisfied: beautifulsoup4>=4.10 in /usr/local/lib/python3.10/dist-packages (from libpysal) (4.12.3)
Requirement already satisfied: platformdirs>=2.0.2 in /usr/local/lib/python3.10/dist-packages (from libpysal) (4.3.2)
Requirement already satisfied: requests>=2.27 in /usr/local/lib/python3.10/dist-packages (from libpysal) (2.32.3)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4>=4.10->libpysal) (2.6)
Requirement already satisfied: attrs>=19.2.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.21->geopandas) (24.2.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.21->geopandas) (2024.8.30)
Requirement already satisfied: click~=8.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.21->geopandas) (8.1.7)
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.21->geopandas) (1.1.1)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.10/dist-packages (from fiona>=1.8.21->geopandas) (0.7.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.4.0->geopandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.4.0->geopandas) (2024.2)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.4.0->geopandas) (2024.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->libpysal) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->libpysal) (3.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.27->libpysal) (2.0.7)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2->esda) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.2->esda) (3.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>=1.4.0->geopandas) (1.16.0)
```

```
import geopandas as gpd
from esda.moran import Moran
```

```
from libpysal.weights import KNN
import matplotlib.pyplot as plt

# Assuming you have a DataFrame 'df' with Latitude, Longitude, and Zone columns
# Convert the DataFrame to a GeoDataFrame
gdf = gpd.GeoDataFrame(df_filtered, geometry=gpd.points_from_xy(df_filtered['Longitude'], df_filtered['Latitude']))

# Ensure the correct coordinate reference system (CRS), using EPSG 4326 for WGS84
gdf.crs = "EPSG:4326"

# Create spatial weights (K-nearest neighbors here, you can adjust K)
w = KNN.from_dataframe(gdf, k=8)

# Calculate Moran's I for 'Zone'
moran = Moran(gdf['Zone'], w)

# Moran's I results
print(f"Moran's I: {moran.I}")
print(f"P-value: {moran.p_sim}")

# Visualize results (optional: plot geolocation with cluster information)
gdf.plot(column='Zone', legend=True, cmap='viridis')
plt.title("Spatial Distribution of Zones")
plt.show()
```

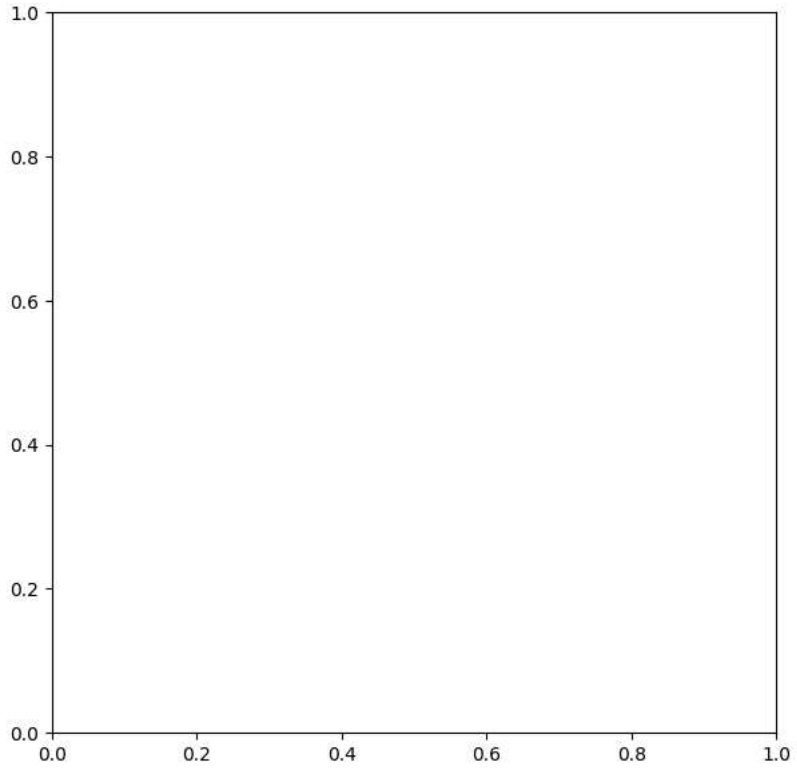
```
/usr/local/lib/python3.10/dist-packages/libpysal/weights/distance.py:153: UserWarning: The weights matrix is not fully connected:
 There are 65 disconnected components.
  W.__init__(self, neighbors, id_order=ids, **kwargs)
Moran's I: 0.9910651098792923
P-value: 0.001
```




```
import geopandas as gpd
import numpy as np
from libpysal.weights import KNN
import matplotlib.pyplot as plt
```

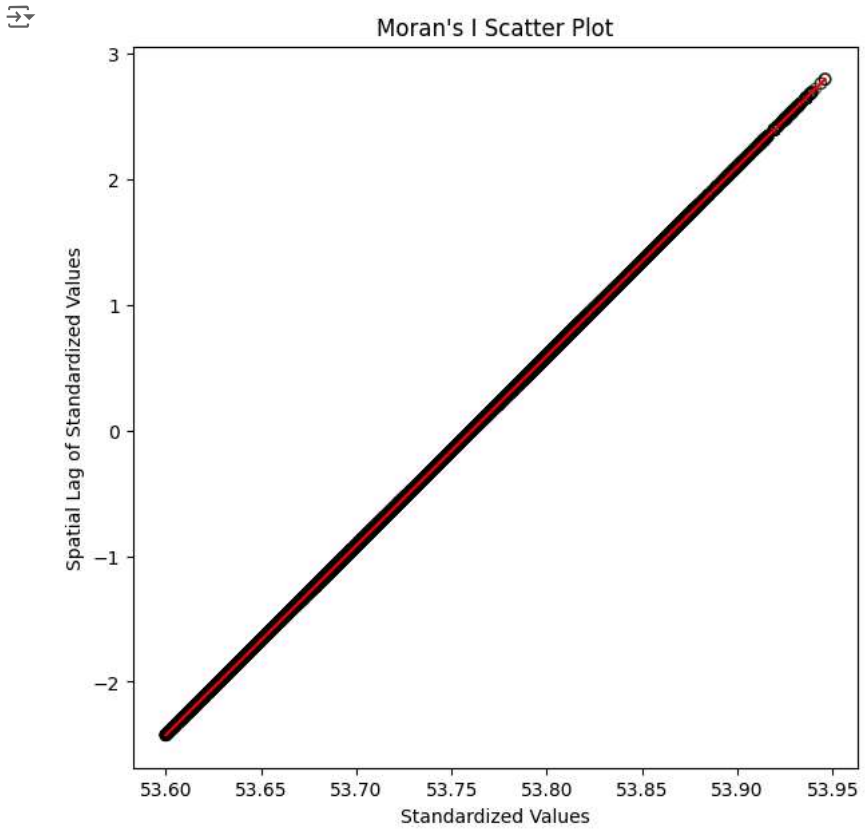
```
# Assuming 'df' is a DataFrame with 'Latitude' and 'Longitude' columns
# Convert your DataFrame to a GeoDataFrame
gdf = gpd.GeoDataFrame(df_filtered, geometry=gpd.points_from_xy(df_filtered['Longitude'], df_filtered['Latitude']))

# Create weights using K-Nearest Neighbors (KNN)
# 5 Nearest Neighbors (this can be tuned based on the dataset's density)
knn_weights = KNN.from_dataframe(gdf, k=5)

# Calculate Moran's I using Latitude and Longitude or any other metric of interest (e.g., crime intensity)
# Example using Latitude for Moran's I calculation
moran = Moran(gdf['Latitude'], knn_weights)

# Moran's I statistic
print(f"Moran's I: {moran.I}")

# P-value
print(f"P-value: {moran.p_sim}")

# Optional: Plot Moran's I scatter plot
fig, ax = plt.subplots(figsize=(7, 7))
ax.scatter(moran.y, moran.EI_sim, edgecolor='black', facecolor='none', alpha=0.5)
ax.plot([min(moran.y), max(moran.y)], [min(moran.EI_sim), max(moran.EI_sim)], color="r")
ax.set_title("Moran's I Scatter Plot")
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/libpysal/weights/distance.py:153: UserWarning: The weights matrix is not fully connected:
 There are 393 disconnected components.
  W.__init__(self, neighbors, id_order=ids, **kwargs)
Moran's I: 0.9992071317119485
P-value: 0.001
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-43-456a5aea406c> in <cell line: 26>()
     24 # Optional: Plot Moran's I scatter plot
     25 fig, ax = plt.subplots(figsize=(7, 7))
---> 26 ax.scatter(moran.y, moran.EI_sim, edgecolor='black', facecolor='none', alpha=0.5)
     27 ax.plot([min(moran.y), max(moran.y)], [min(moran.EI_sim), max(moran.EI_sim)], color="r")
     28 ax.set_title("Moran's I Scatter Plot")

                                 1 frames
/usr/local/lib/python3.10/dist-packages/matplotlib/axes/_axes.py in scatter(self, x, y, s, c, marker, cmap, norm, vmin, vmax, alpha, linewidths, edgecolors, plotnonfinite, **kwargs)
   4582         y = np.ma.ravel(y)
   4583         if x.size != y.size:
-> 4584             raise ValueError("x and y must be the same size")
   4585
   4586         if s is None:

ValueError: x and y must be the same size
```




```
import matplotlib.pyplot as plt
```

```
# Moran's I scatter plot with standardized values and spatial lag
fig, ax = plt.subplots(figsize=(7, 7))

# x-axis is the standardized values (moran.y), and y-axis is the spatial lag (moran.z)
ax.scatter(moran.y, moran.z, edgecolor='black', facecolor='none', alpha=0.5)
ax.plot([min(moran.y), max(moran.y)], [min(moran.z), max(moran.z)], color="r")

# Add labels and title
ax.set_xlabel('Standardized Values')
ax.set_ylabel('Spatial Lag of Standardized Values')
ax.set_title("Moran's I Scatter Plot")

# Show plot
plt.show()
```



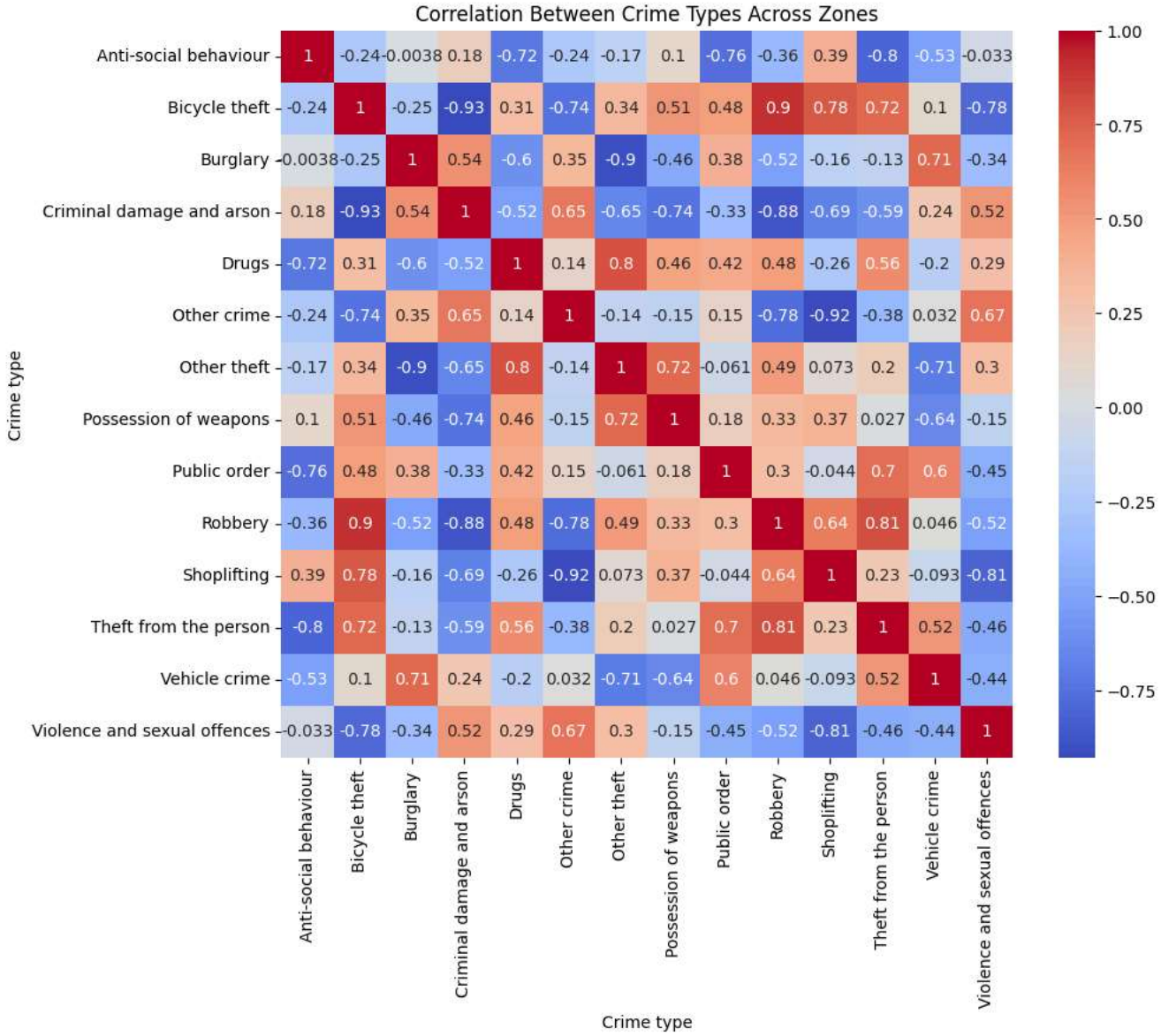

```
from pysal.explore.esda.moran import Moran
import numpy as np
from scipy.spatial import distance_matrix

# Example: Calculating Moran's I for Latitude and Longitude
coords = df_filtered[['Latitude', 'Longitude']].values
dist_matrix = distance_matrix(coords, coords)
```

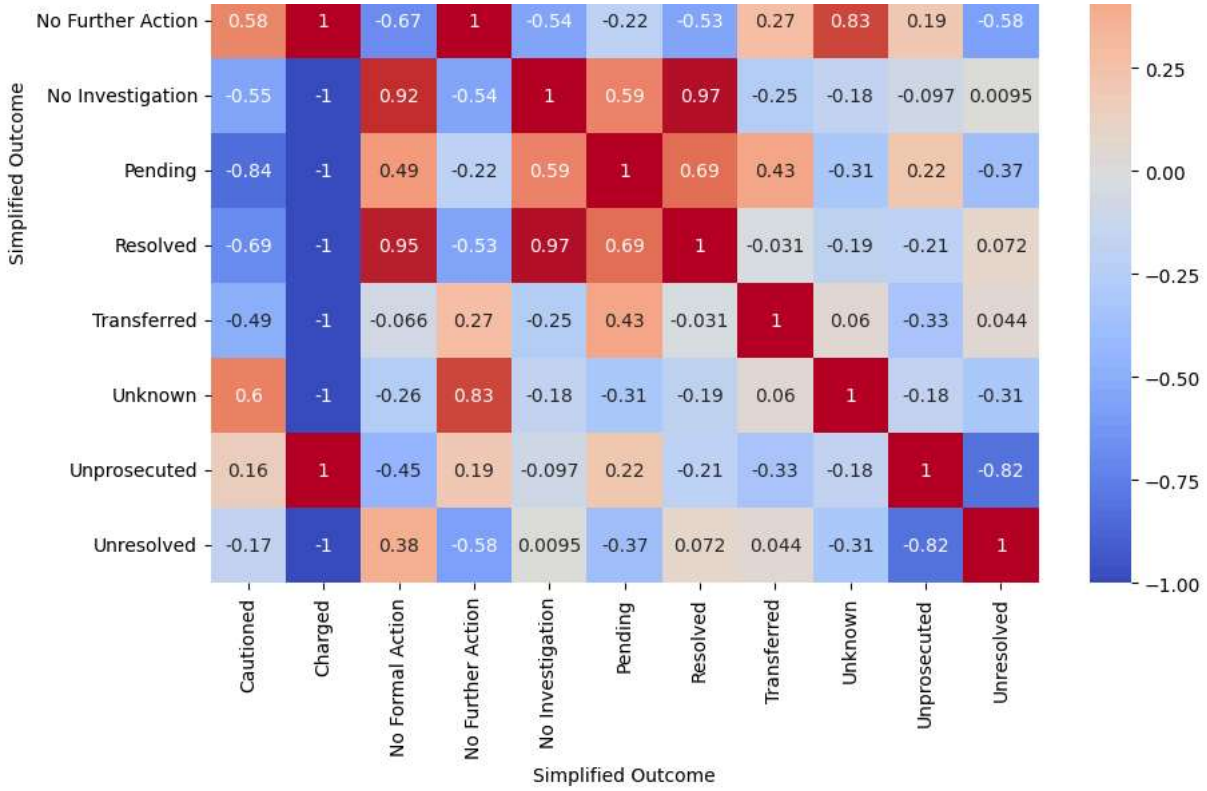
```
# Creating a weight matrix by inverting the distance (you may need to add a small epsilon to avoid division by zero)
epsilon = 1e-10
weights = 1 / (dist_matrix + epsilon)
np.fill_diagonal(weights, 0)

# Moran's I Calculation
moran_i = Moran(df['Zone'], weights)
print(f"Moran's I: {moran_i.I}")
```

```
/usr/local/lib/python3.10/dist-packages/spaghetti/network.py:41: FutureWarning: The next major release of pysal/spaghetti (2.0.0) will drop support for all ``libpysal.cg`` geometries
  warnings.warn(dep_msg, FutureWarning, stacklevel=1)
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-35-bcb18907edff> in <cell line: 1>()
----> 1 from pysal.explore.esda.moran import Moran
      2 import numpy as np
      3 from scipy.spatial import distance_matrix
      4
      5 # Example: Calculating Moran's I for Latitude and Longitude

/usr/local/lib/python3.10/dist-packages/pysal/explore/esda/__init__.py in <module>
     14 from esda.util import fdr
     15 from esda.smaup import Smaup
---> 16 from esda.lee import Spatial_Pearson, Local_Spatial_Pearson
     17 from esda.silhouettes import (path_silhouette, boundary_silhouette,
     18                               silhouette_alist, nearest_label)

ImportError: cannot import name 'Local_Spatial_Pearson' from 'esda.lee' (/usr/local/lib/python3.10/dist-packages/esda/lee.py)

---------------------------------------------------------------------------
NOTE: If your import is failing due to a missing package, you can
manually install dependencies using either !pip or !apt.

To view examples of installing some common dependencies, click the
"Open Examples" button below.
---------------------------------------------------------------------------
```

OPEN EXAMPLES

```
# Import necessary libraries
import seaborn as sns
import matplotlib.pyplot as plt

# Correlation between Zones and different crime types
plt.figure(figsize=(10, 8))
sns.heatmap(df_filtered.groupby('Zone')['Crime type'].value_counts(normalize=True).unstack().corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Between Crime Types Across Zones')
plt.show()

# Correlation between Zones and Simplified Outcome
plt.figure(figsize=(10, 8))
sns.heatmap(df_filtered.groupby('Zone')['Simplified Outcome'].value_counts(normalize=True).unstack().corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Between Crime Outcomes Across Zones')
plt.show()
```


Correlation Between Crime Types Across Zones
Crime type
Anti-social behaviour 1 -0.24 -0.0038 0.18 -0.72 -0.24 -0.17 0.1 -0.76 -0.36 0.39 -0.8 -0.53 -0.033
Bicycle theft -0.24 1 -0.25 -0.93 0.31 -0.74 0.34 0.51 0.48 0.9 0.78 0.72 0.1 -0.78
Burglary -0.0038 -0.25 1 0.54 -0.6 0.35 -0.9 -0.46 0.38 -0.52 -0.16 -0.13 0.71 -0.34
Criminal damage and arson 0.18 -0.93 0.54 1 -0.52 0.65 -0.65 -0.74 -0.33 -0.88 -0.69 -0.59 0.24 0.52
Drugs -0.72 0.31 -0.6 -0.52 1 0.14 0.8 0.46 0.42 0.48 -0.26 0.56 -0.2 0.29
Other crime -0.24 -0.74 0.35 0.65 0.14 1 -0.14 -0.15 0.15 -0.78 -0.92 -0.38 0.032 0.67
Other theft -0.17 0.34 -0.9 -0.65 0.8 -0.14 1 0.72 -0.061 0.49 0.073 0.2 -0.71 0.3
Possession of weapons 0.1 0.51 -0.46 -0.74 0.46 -0.15 0.72 1 0.18 0.33 0.37 0.027 -0.64 -0.15
Public order -0.76 0.48 0.38 -0.33 0.42 0.15 -0.061 0.18 1 0.3 -0.044 0.7 0.6 -0.45
Robbery -0.36 0.9 -0.52 -0.88 0.48 -0.78 0.49 0.33 0.3 1 0.64 0.81 0.046 -0.52
Shoplifting 0.39 0.78 -0.16 -0.69 -0.26 -0.92 0.073 0.37 -0.044 0.64 1 0.23 -0.093 -0.81
Theft from the person -0.8 0.72 -0.13 -0.59 0.56 -0.38 0.2 0.027 0.7 0.81 0.23 1 0.52 -0.46
Vehicle crime -0.53 0.1 0.71 0.24 -0.2 0.032 -0.71 -0.64 0.6 0.046 -0.093 0.52 1 -0.44
Violence and sexual offences -0.033 -0.78 -0.34 0.52 0.29 0.67 0.3 -0.15 -0.45 -0.52 -0.81 -0.46 -0.44 1
Anti-social behaviour
Bicycle theft
Burglary
Criminal damage and arson
Drugs
Other crime
Other theft
Possession of weapons
Public order
Robbery
Shoplifting
Theft from the person
Vehicle crime
Violence and sexual offences
Crime type
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
Correlation Between Crime Outcomes Across Zones
Cautioned 1 1 -0.65 0.58 -0.55 -0.84 -0.69 -0.49 0.6 0.16 -0.17
Charged 1 1 -1 1 -1 -1 -1 -1 -1 1 -1
No Formal Action -0.65 -1 1 -0.67 0.92 0.49 0.95 -0.066 -0.26 -0.45 0.38
1.00
0.75
0.50


Simplified Outcome
No Further Action
No Investigation
Pending
Resolved
Transferred
Unknown
Unprosecuted
Unresolved
0.58 1 -0.67 1 -0.54 -0.22 -0.53 0.27 0.83 0.19 -0.58
-0.55 -1 0.92 -0.54 1 0.59 0.97 -0.25 -0.18 -0.097 0.0095
-0.84 -1 0.49 -0.22 0.59 1 0.69 0.43 -0.31 0.22 -0.37
-0.69 -1 0.95 -0.53 0.97 0.69 1 -0.031 -0.19 -0.21 0.072
-0.49 -1 -0.066 0.27 -0.25 0.43 -0.031 1 0.06 -0.33 0.044
0.6 -1 -0.26 0.83 -0.18 -0.31 -0.19 0.06 1 -0.18 -0.31
0.16 1 -0.45 0.19 -0.097 0.22 -0.21 -0.33 -0.18 1 -0.82
-0.17 -1 0.38 -0.58 0.0095 -0.37 0.072 0.044 -0.31 -0.82 1
Cautioned
Charged
No Formal Action
No Further Action
No Investigation
Pending
Resolved
Transferred
Unknown
Unprosecuted
Unresolved
Simplified Outcome
0.25
0.00
−0.25
−0.50
−0.75
−1.00

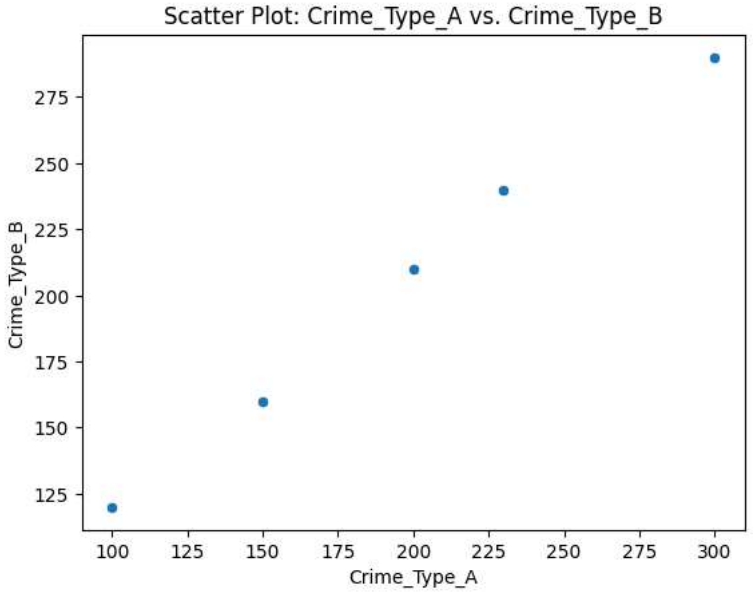
```
from scipy.stats import pearsonr
import pandas as pd

# Example: Testing correlation significance between 'Crime_Type_A' and 'Crime_Type_B'
crime_data = pd.DataFrame({
    'Crime_Type_A': [100, 200, 300, 150, 230],
    'Crime_Type_B': [120, 210, 290, 160, 240]
})

# Pearson correlation coefficient and p-value
corr, p_value = pearsonr(crime_data['Crime_Type_A'], crime_data['Crime_Type_B'])

print(f"Correlation: {corr}")
print(f"P-value: {p_value}")

if p_value < 0.05:
    print("The correlation is statistically significant.")
else:
    print("The correlation is not statistically significant.")
```

```
Correlation: 0.9976353256510933
P-value: 0.0001379865521983261
The correlation is statistically significant.
```

```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'crime_corr_matrix' is the correlation matrix of crime types
plt.figure(figsize=(10, 8))
sns.heatmap(crime_corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix Heatmap of Crime Types')
plt.show()
```

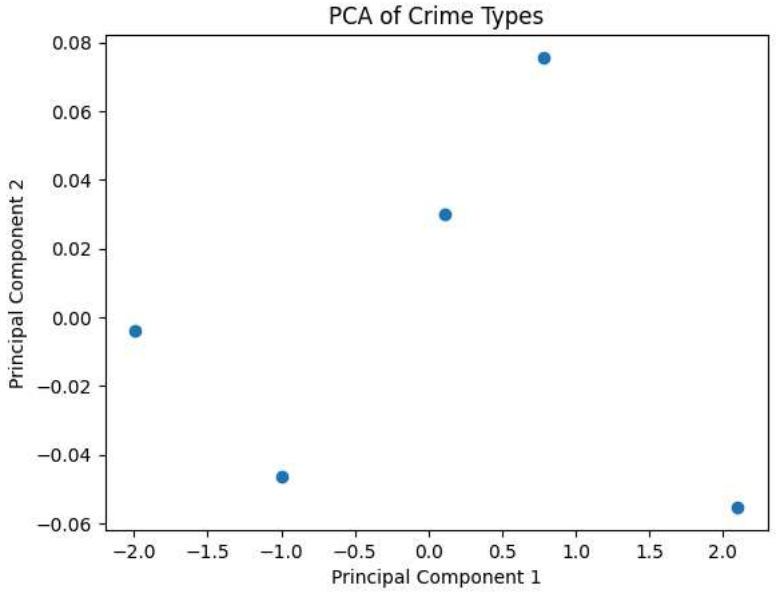
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-46-d3ba878343b7> in <cell line: 6>()
      4 # Assuming 'crime_corr_matrix' is the correlation matrix of crime types
      5 plt.figure(figsize=(10, 8))
----> 6 sns.heatmap(crime_corr_matrix, annot=True, cmap='coolwarm', center=0)
      7 plt.title('Correlation Matrix Heatmap of Crime Types')
      8 plt.show()

NameError: name 'crime_corr_matrix' is not defined

<Figure size 1000x800 with 0 Axes>
```

```
sns.scatterplot(x=crime_data['Crime_Type_A'], y=crime_data['Crime_Type_B'])
plt.title("Scatter Plot: Crime_Type_A vs. Crime_Type_B")
plt.show()
```




```
from sklearn.decomposition import PCA
import numpy as np

# Standardize the data
from sklearn.preprocessing import StandardScaler
crime_data_standardized = StandardScaler().fit_transform(crime_data)

# Perform PCA
pca = PCA(n_components=2)  # Reduce to 2 dimensions (for example)
principal_components = pca.fit_transform(crime_data_standardized)

# Explained variance
print("Explained variance by each component:", pca.explained_variance_ratio_)

# Visualize the components
plt.scatter(principal_components[:, 0], principal_components[:, 1])
plt.title("PCA of Crime Types")
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```

```
Explained variance by each component: [0.99881766 0.00118234]
```




```
df_clean
```

	Month	Reported by	Falls within	Location	LSOA code	LSOA name	Crime type	Simplified Outcome	Zone
0	2024-01	West Yorkshire Police	West Yorkshire Police	Wood Royd Hill Lane	E01007426	Barnsley 027D	Violence and sexual offences	Unprosecuted	2
1	2024-01	West Yorkshire Police	West Yorkshire Police	Cross End Fold	E01010646	Bradford 001A	Shoplifting	Unresolved	8
2	2024-01	West Yorkshire Police	West Yorkshire Police	Millfold	E01010646	Bradford 001A	Violence and sexual offences	Unprosecuted	8
3	2024-01	West Yorkshire Police	West Yorkshire Police	Ridleys Fold	E01010647	Bradford 001B	Burglary	Unresolved	8
4	2024-01	West Yorkshire Police	West Yorkshire Police	School Lane	E01010648	Bradford 001C	Anti-social behaviour	Unknown	8
...	...	...	...	...	...	...	...	...	...
23805	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Pending	1
23806	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Pending	1
23807	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Unprosecuted	1
23808	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Unprosecuted	1
23809	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Unknown	Other crime	Unresolved	1

23810 rows × 9 columns

```
# Extract year and month as separate features
df_encoded['Year'] = df_filtered['Month'].dt.year
df_encoded['Month'] = df_filtered['Month'].dt.month

# Drop original datetime column if not needed
df_encoded = df_filtered.drop(columns=['Month'])
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-49-66b99a9c72d1> in <cell line: 2>()
      1 # Extract year and month as separate features
----> 2 df_encoded['Year'] = df_filtered['Month'].dt.year
      3 df_encoded['Month'] = df_filtered['Month'].dt.month
      4
      5 # Drop original datetime column if not needed

                                 2 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/accessors.py in __new__(cls, data)
    606             return PeriodProperties(data, orig)
    607
--> 608         raise AttributeError("Can only use .dt accessor with datetimelike values")

AttributeError: Can only use .dt accessor with datetimelike values
```

```
# Drop the 'LSOA code' column
df_clean = df_clean.drop(columns=['LSOA code'])
```

```
df_clean
```

	Location	CrimeCount	CrimeType	SocioEconomicFactor	RiskLevel	PredictedRiskLevel
0	A	10	2	0.5	Medium	Medium
1	B	5	1	0.7	Low	Medium
2	C	20	3	0.3	High	High
3	D	8	1	0.6	Medium	Medium
4	E	15	2	0.4	Medium	Medium

```
# Define risk levels based on Simplified Outcome
def categorize_risk(outcome):
    if outcome in ['Unprosecuted', 'Unresolved']:
        return 'High Risk'
    elif outcome == 'Investigation complete; no suspect identified':
        return 'Medium Risk'
    else:
        return 'Low Risk'
df_kk = WestYorkshire_df
# Apply the risk categorization
df_kk['Risk Level'] = df_encoded['Simplified Outcome'].apply(categorize_risk)

# Count occurrences of each risk level by location
risk_counts = df_kk.groupby(['Location', 'Risk Level']).size().unstack(fill_value=0)

# Plot heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(risk_counts, cmap='YlGnBu', annot=True, fmt='d', linewidths=0.5)
plt.title('Heatmap of Risk Levels by Location')
plt.xlabel('Risk Level')
plt.ylabel('Location')
plt.xticks(rotation=45)
```

```
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# Prepare data for prediction
# Feature Engineering: Example features based on your data
df_features = df_clean[['Zone']]  # Example feature; you may include more features based on your dataset
df_target = df_clean['Risk Level']  # Target variable

# Convert categorical variables to numeric
df_features = pd.get_dummies(df_features, drop_first=True)
df_target = pd.get_dummies(df_target, drop_first=True)

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(df_features, df_target, test_size=0.3, random_state=42)

# Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict on test set
y_pred = model.predict(X_test)

# Evaluate model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-54-38d5663d82f4> in <cell line: 9>()
      7     else:
      8         return 'Low Risk'
----> 9 df_kk
     10 # Apply the risk categorization
     11 df_kk['Risk Level'] = df_encoded['Simplified Outcome'].apply(categorize_risk)

NameError: name 'df_kk' is not defined
```

```
# Print the column names to check for discrepancies
print(df_clean.columns)
```

```
Index(['Location', 'CrimeCount', 'CrimeType', 'SocioEconomicFactor',
       'RiskLevel', 'PredictedRiskLevel'],
      dtype='object')
```

```
df_clean.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23810 entries, 0 to 23809
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Month               23810 non-null  object
 1   Reported by         23810 non-null  object
```

```
 2   Falls within        23810 non-null  object
 3   Location            23810 non-null  object
 4   LSOA name           23810 non-null  object
 5   Crime type          23810 non-null  object
 6   Simplified Outcome  23810 non-null  object
 7   Zone                23810 non-null  int32
dtypes: int32(1), object(7)
memory usage: 1.4+ MB
```

```
# Drop the original 'Crime type' column (it is now redundant)
df_encoded = df_encoded.drop(columns=['Month','Falls within'])
df_encoded
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-65-2af99f8dcbf7> in <cell line: 2>()
      1 # Drop the original 'Crime type' column (it is now redundant)
----> 2 df_encoded = df_encoded.drop(columns=['Month','Falls within'])
      3 df_encoded

3 frames
/usr/local/lib/python3.10/dist-packages/pandas/core/indexes/base.py in drop(self, labels, errors)
   6998         if mask.any():
   6999             if errors != "ignore":
-> 7000                 raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7001             indexer = indexer[~mask]
   7002         return self.delete(indexer)

KeyError: "['Month', 'Falls within'] not found in axis"
```

```
df_encoded
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-40-a0235eeb254f> in <cell line: 1>()
----> 1 df_encoded

NameError: name 'df_encoded' is not defined
```

```
df_encoded = pd.get_dummies(df_clean, columns=['Crime type','Simplified Outcome', 'Location','LSOA name'], drop_first=True)
df_encoded
```

	Month	Reported by	Falls within	Zone	Crime type_encoded	Crime type_Bicycle theft	Crime type_Burglary	Crime type_Criminal damage and arson	Crime type_Drugs	Crime type_Other crime	...	LSOA name_Wakefield 043C	LSOA name_Wakefield 043D	LSOA name_Wakefield 044A	name_Wake
0	2024-01	West Yorkshire Police	West Yorkshire Police	2	13	False	False	False	False	False	...	False	False	False	
1	2024-01	West Yorkshire Police	West Yorkshire Police	8	10	False	False	False	False	False	...	False	False	False	
2	2024-01	West Yorkshire Police	West Yorkshire Police	8	13	False	False	False	False	False	...	False	False	False	
3	2024-01	West Yorkshire Police	West Yorkshire Police	8	2	False	True	False	False	False	...	False	False	False	
4	2024-01	West Yorkshire Police	West Yorkshire Police	8	0	False	False	False	False	False	...	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
23805	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23806	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23807	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23808	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23809	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	

23810 rows × 10407 columns

```
df_encoded
```

	Reported by	Crime type	Zone	Crime type_encoded	Simplified Outcome_Charged	Simplified Outcome_No Formal Action	Simplified Outcome_No Further Action	Simplified Outcome_No Investigation	Simplified Outcome_Pending	Simplified Outcome_Resolved	...	LSOA name_Wakefield 043C	LSOA name_Wakefield 043D	name_Wak
0	West Yorkshire Police	Violence and sexual offences	2	13	False	False	False	False	False	False	...	False	False	
1	West Yorkshire Police	Shoplifting	8	10	False	False	False	False	False	False	...	False	False	
2	West Yorkshire Police	Violence and sexual offences	8	13	False	False	False	False	False	False	...	False	False	
3	West Yorkshire Police	Burglary	8	2	False	False	False	False	False	False	...	False	False	
4	West Yorkshire Police	Anti-social behaviour	8	0	False	False	False	False	False	False	...	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
23805	West Yorkshire Police	Other crime	1	5	False	False	False	False	True	False	...	False	False	
23806	West Yorkshire Police	Other crime	1	5	False	False	False	False	True	False	...	False	False	
23807	West Yorkshire Police	Other crime	1	5	False	False	False	False	False	False	...	False	False	
23808	West Yorkshire Police	Other crime	1	5	False	False	False	False	False	False	...	False	False	
23809	West Yorkshire Police	Other crime	1	5	False	False	False	False	False	False	...	False	False	

23810 rows × 10393 columns

```
# List of columns to encode
columns_to_encode = [ 'Crime type']


# Apply One-Hot Encoding
df_encoded = pd.get_dummies(df_clean, columns=columns_to_encode)


df_encoded
```

	Month	Reported by	Falls within	Zone	Crime type_encoded	Crime type_Bicycle theft	Crime type_Burglary	Crime type_Criminal damage and arson	Crime type_Drugs	Crime type_Other crime	...	LSOA name_Wakefield 043C	LSOA name_Wakefield 043D	LSOA name_Wakefield 044A	name_Wake
0	2024-01	West Yorkshire Police	West Yorkshire Police	2	13	False	False	False	False	False	...	False	False	False	
1	2024-01	West Yorkshire Police	West Yorkshire Police	8	10	False	False	False	False	False	...	False	False	False	
2	2024-01	West Yorkshire Police	West Yorkshire Police	8	13	False	False	False	False	False	...	False	False	False	
3	2024-01	West Yorkshire Police	West Yorkshire Police	8	2	False	True	False	False	False	...	False	False	False	
4	2024-01	West Yorkshire Police	West Yorkshire Police	8	0	False	False	False	False	False	...	False	False	False	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
23805	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23806	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23807	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23808	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	
23809	2024-01	West Yorkshire Police	West Yorkshire Police	1	5	False	False	False	False	True	...	False	False	False	

23810 rows × 10407 columns

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Assuming df_encoded is your DataFrame with encoded features and target
# Assuming `df_encoded` is your dataframe and 'Crime type_encoded' is the target
X = df_encoded.drop(columns=['Zone','Reported by','Falls within','Reported by','Month'])
y = df_encoded['Zone']



# Split the data into training and test sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(random_state=42)

# Train the model
clf.fit(X_train, y_train)

# Predict on the test set
y_pred = clf.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)

conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

```
Accuracy: 0.99
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        47
           1       1.00      1.00      1.00        65
           2       0.98      0.98      0.98       430
           3       0.98      0.99      0.99       906
           4       0.98      0.99      0.99       982
           5       0.99      0.98      0.99       440
           6       1.00      0.99      0.99       599
           7       0.97      0.99      0.98       341
           8       0.98      0.98      0.98       288
           9       1.00      0.97      0.98       664

    accuracy                           0.99      4762
   macro avg       0.99      0.99      0.99      4762
weighted avg       0.99      0.99      0.99      4762

Confusion Matrix:
[[ 46   0   0   0   0   0   1   0   0   0]
 [  0  65   0   0   0   0   0   0   0   0]
 [  0   0 421   0   4   0   1   4   0   0]
 [  0   0   0 894   1   4   0   5   0   2]
 [  0   0   3   0 976   0   0   0   3   0]
 [  0   0   0   0   5 433   0   2   0   0]
 [  0   0   2   0   2   0 593   0   2   0]
 [  0   0   0   2   2   0   0 337   0   0]
 [  1   0   0   0   4   0   0   0 283   0]
 [  0   0   2  13   5   0   0   0   0 644]]
```

```
from sklearn.tree import DecisionTreeClassifier

# Initialize and train the model with the best parameters
best_model = DecisionTreeClassifier(max_depth=15, min_samples_split=2)
best_model.fit(X_train, y_train)
```

```
# Evaluate the model
y_pred = best_model.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix

print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00        77
           1       1.00      1.00      1.00        93
           2       1.00      0.06      0.11       660
           3       1.00      0.16      0.27      1381
           4       0.22      1.00      0.36      1446
           5       0.00      0.00      0.00       620
           6       1.00      0.14      0.25       938
           7       1.00      0.16      0.28       516
           8       1.00      0.10      0.18       430
           9       1.00      0.03      0.05       982

    accuracy                           0.29      7143
   macro avg       0.72      0.26      0.25      7143
weighted avg       0.74      0.29      0.22      7143

Confusion Matrix:
 [[   0    0    0    0   77    0    0    0    0    0]
 [   0   93    0    0    0    0    0    0    0    0]
 [   0    0   39    0  621    0    0    0    0    0]
 [   0    0    0  217 1164    0    0    0    0    0]
 [   0    0    0    0 1446    0    0    0    0    0]
 [   0    0    0    0  620    0    0    0    0    0]
 [   0    0    0    0  804    0  134    0    0    0]
 [   0    0    0    0  431    0    0   85    0    0]
 [   0    0    0    0  387    0    0    0   43    0]
 [   0    0    0    0  957    0    0    0    0   25]]
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
```

```
df_clean
```

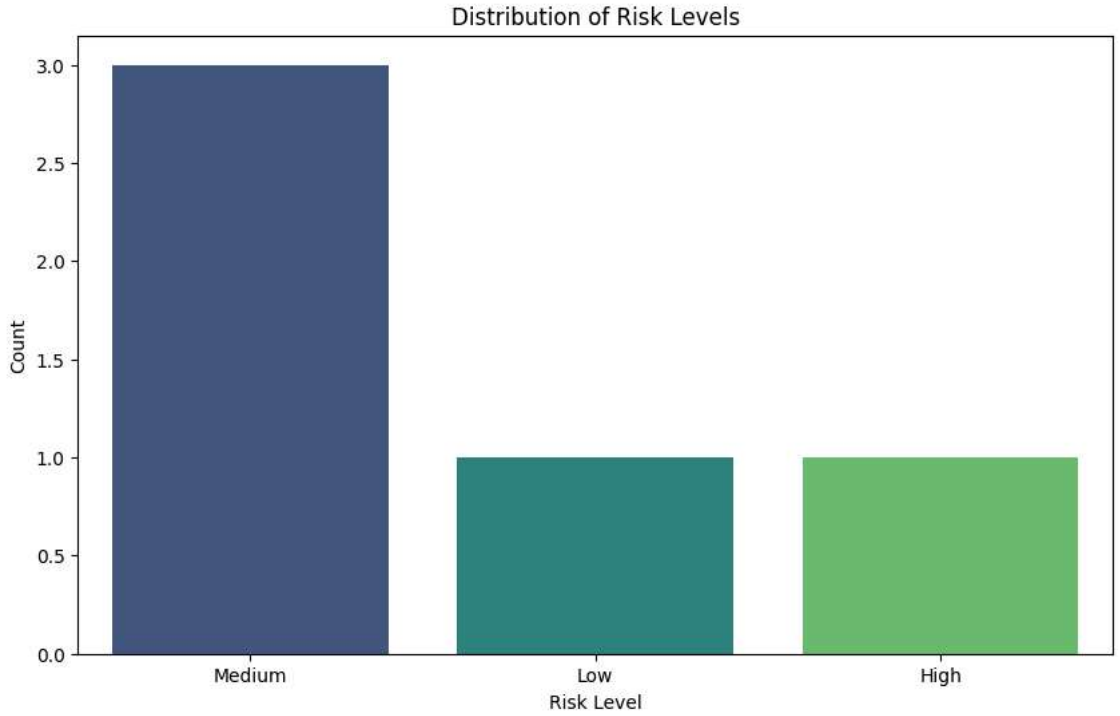
	Location	CrimeCount	CrimeType	SocioEconomicFactor	RiskLevel	PredictedRiskLevel
0	A	10	2	0.5	Medium	Medium
1	B	5	1	0.7	Low	Medium
2	C	20	3	0.3	High	High
3	D	8	1	0.6	Medium	Medium
4	E	15	2	0.4	Medium	Medium

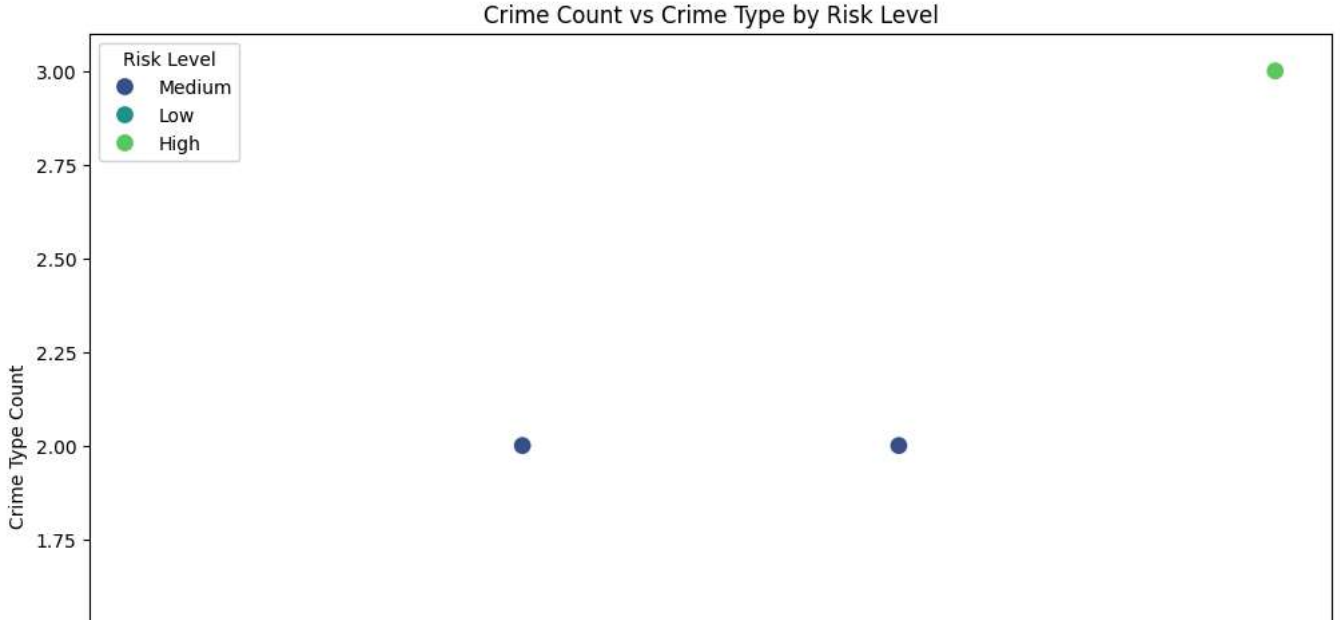
```
import pandas as pd

# Example structure of df_clean
df_clean = pd.DataFrame({
    'Location': ['A', 'B', 'C', 'D', 'E'],
    'CrimeCount': [10, 5, 20, 8, 15],
    'CrimeType': [2, 1, 3, 1, 2],
    'SocioEconomicFactor': [0.5, 0.7, 0.3, 0.6, 0.4]  # Example feature
})

print(df_clean.head())
```

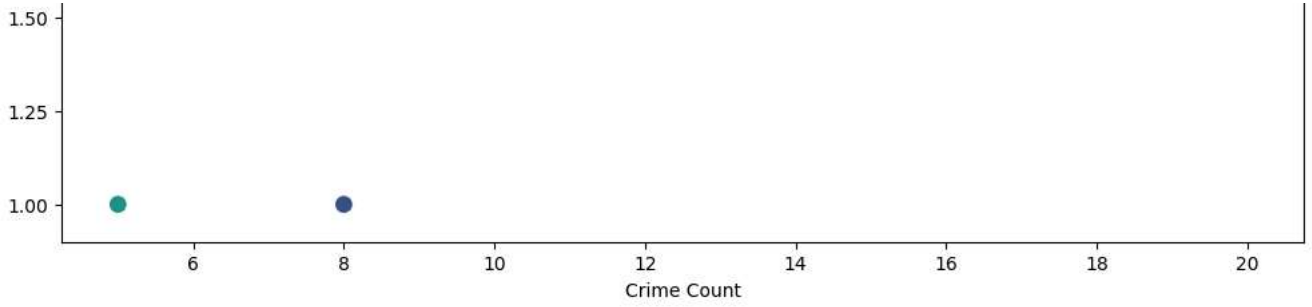
```
  Location  CrimeCount  CrimeType  SocioEconomicFactor
0        A          10          2                  0.5
1        B           5          1                  0.7
2        C          20          3                  0.3
3        D           8          1                  0.6
4        E          15          2                  0.4
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Plot the distribution of risk levels
plt.figure(figsize=(10, 6))
sns.countplot(data=df_clean, x='RiskLevel', palette='viridis')
plt.title('Distribution of Risk Levels')
plt.xlabel('Risk Level')
plt.ylabel('Count')
plt.show()

# Optionally, plot risk levels by location
plt.figure(figsize=(12, 8))
sns.scatterplot(data=df_clean, x='CrimeCount', y='CrimeType', hue='RiskLevel', palette='viridis', s=100)
plt.title('Crime Count vs Crime Type by Risk Level')
plt.xlabel('Crime Count')
plt.ylabel('Crime Type Count')
plt.legend(title='Risk Level')
plt.show()
```

```
<ipython-input-47-684c2e0b37ea>:6: FutureWarning:

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

  sns.countplot(data=df_clean, x='RiskLevel', palette='viridis')
```








```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Assuming `df_encoded` is your dataframe and 'Crime type_encoded' is the target
X = df_encoded.drop(columns=['Zone','Reported by','Falls within','Reported by','Month'])
y = df_encoded['Zone']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred))
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)

conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Assuming `df_encoded` is your dataframe and 'Crime type_encoded' is the target
X = df_encoded.drop(columns=['Zone','Reported by','Falls within','Reported by','Month'])
y = df_encoded['Zone']

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
```

```
print(classification_report(y_test, y_pred))
report = classification_report(y_test, y_pred)
print('Classification Report:')
print(report)

conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:')
print(conf_matrix)
```

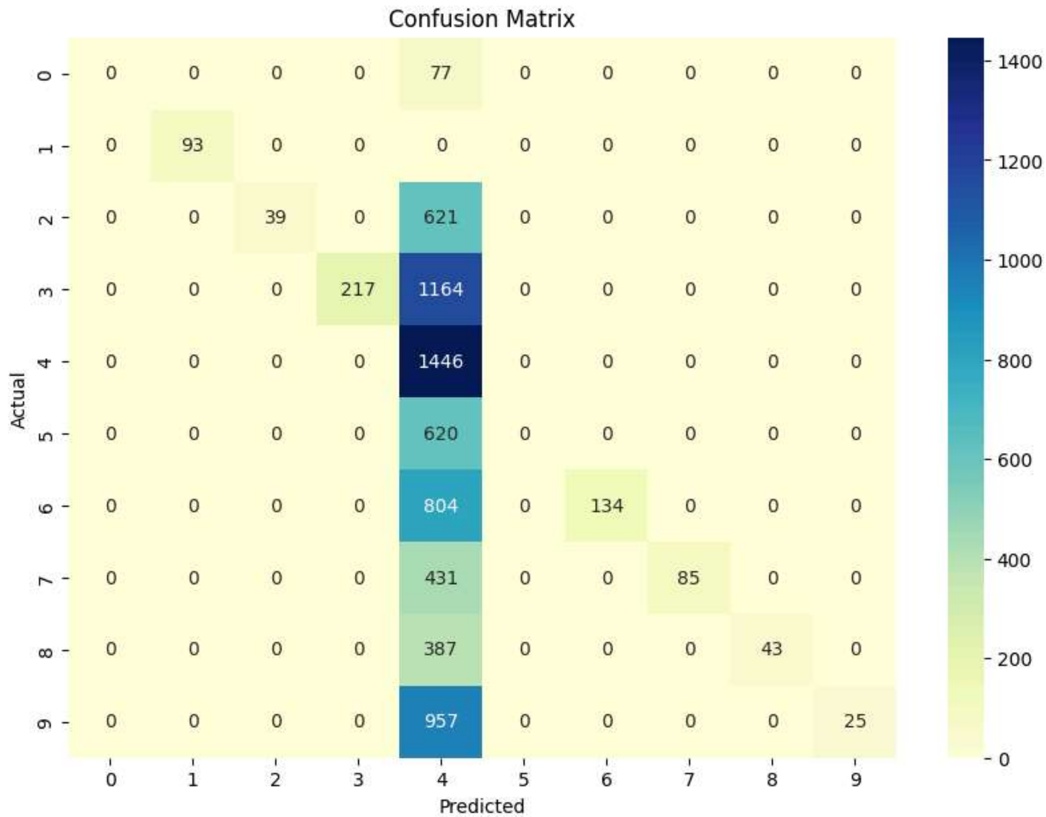
```
              precision    recall  f1-score   support

           0       0.99      0.88      0.93        77
           1       1.00      1.00      1.00        93
           2       0.99      0.94      0.96       660
           3       0.99      0.98      0.98      1381
           4       0.91      1.00      0.95      1446
           5       0.99      0.97      0.98       620
           6       0.99      0.96      0.98       938
           7       0.97      0.96      0.96       516
           8       0.99      0.95      0.97       430
           9       0.99      0.96      0.98       982

    accuracy                           0.97      7143
   macro avg       0.98      0.96      0.97      7143
weighted avg       0.97      0.97      0.97      7143
```

```
# For comparison of actual vs. predicted values
comparison_df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(comparison_df.head())  # Display first few rows of the comparison
```

```
       Actual  Predicted
11474       4          4
21928       5          4
11985       3          4
3142        4          4
1202        4          4
```

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='YlGnBu', xticklabels=sorted(y.unique()), yticklabels=sorted(y.unique()))
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        77
           1       1.00      1.00      1.00        93
           2       1.00      0.06      0.11       660
           3       1.00      0.16      0.27      1381
           4       0.22      1.00      0.36      1446
           5       0.00      0.00      0.00       620
           6       1.00      0.14      0.25       938
           7       1.00      0.16      0.28       516
           8       1.00      0.10      0.18       430
           9       1.00      0.03      0.05       982

    accuracy                           0.29      7143
   macro avg       0.72      0.26      0.25      7143
weighted avg       0.74      0.29      0.22      7143

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
```



```
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split

# Assuming X and y are your feature set and target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Apply SMOTE to training data
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)



from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Use 'balanced' mode for class weights
model = DecisionTreeClassifier(class_weight='balanced', max_depth=15, min_samples_split=2, random_state=42)

# Fit the model on the resampled dataset
model.fit(X_train_resampled, y_train_resampled)

# Predict on test data
y_pred = model.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.32      0.48        47
           1       1.00      1.00      1.00        65
           2       1.00      0.07      0.13       430
           3       1.00      0.17      0.28       906
           4       1.00      0.07      0.12       982
           5       0.10      1.00      0.19       440
           6       1.00      0.13      0.23       599
           7       1.00      0.17      0.29       341
           8       1.00      0.10      0.18       288
           9       0.00      0.00      0.00       664

    accuracy                           0.20      4762
   macro avg       0.81      0.30      0.29      4762
weighted avg       0.78      0.20      0.19      4762

Confusion Matrix:
[[ 15   0   0   0   0  32   0   0   0   0]
 [  0  65   0   0   0   0   0   0   0   0]
 [  0   0  31   0   0 399   0   0   0   0]
 [  0   0   0 150   0 756   0   0   0   0]
 [  0   0   0   0  65 917   0   0   0   0]
 [  0   0   0   0   0 440   0   0   0   0]
 [  0   0   0   0   0 521  78   0   0   0]
 [  0   0   0   0   0 284   0  57   0   0]
 [  0   0   0   0   0 259   0   0  29   0]
 [  0   0   0   0   0 664   0   0   0   0]]
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no p
  _warn_prf(average, modifier, msg_start, len(result))
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

# Initialize RandomForestClassifier with balanced class weights
rf_model = RandomForestClassifier(class_weight='balanced', n_estimators=100, random_state=42)

# Fit the model on the resampled dataset
rf_model.fit(X_train_resampled, y_train_resampled)

# Predict on test data
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
print("Random Forest Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.09      0.94      0.16        47
           1       1.00      1.00      1.00        65
           2       0.81      0.80      0.81       430
           3       0.98      0.85      0.91       906
           4       0.99      0.79      0.88       982
           5       0.86      0.84      0.85       440
           6       0.94      0.79      0.86       599
           7       0.81      0.86      0.83       341
           8       0.77      0.89      0.83       288
           9       0.95      0.81      0.88       664

    accuracy                           0.83      4762
   macro avg       0.82      0.86      0.80      4762
weighted avg       0.91      0.83      0.86      4762

Random Forest Confusion Matrix:
[[ 44   0   1   0   0   0   0   1   0   1]
 [  0  65   0   0   0   0   0   0   0   0]
 [ 48   0 344   2   1   9   7  11   6   2]
 [ 65   0  16 772   2   8   1  12  17  13]
 [118   0  28   0 774  16   6  10  23   7]
 [ 39   0   5   1   0 368   4  10   9   4]
 [ 75   0  10   0   0  15 474  13  11   1]
 [ 31   0   5   1   0   3   5 293   2   1]
 [ 21   0   1   0   1   4   1   3 256   1]
 [ 74   0  14  10   2   5   4   8   7 540]]
```

```
df_clean
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-8442b2128281> in <cell line: 1>()
----> 1 df_clean

NameError: name 'df_clean' is not defined
```

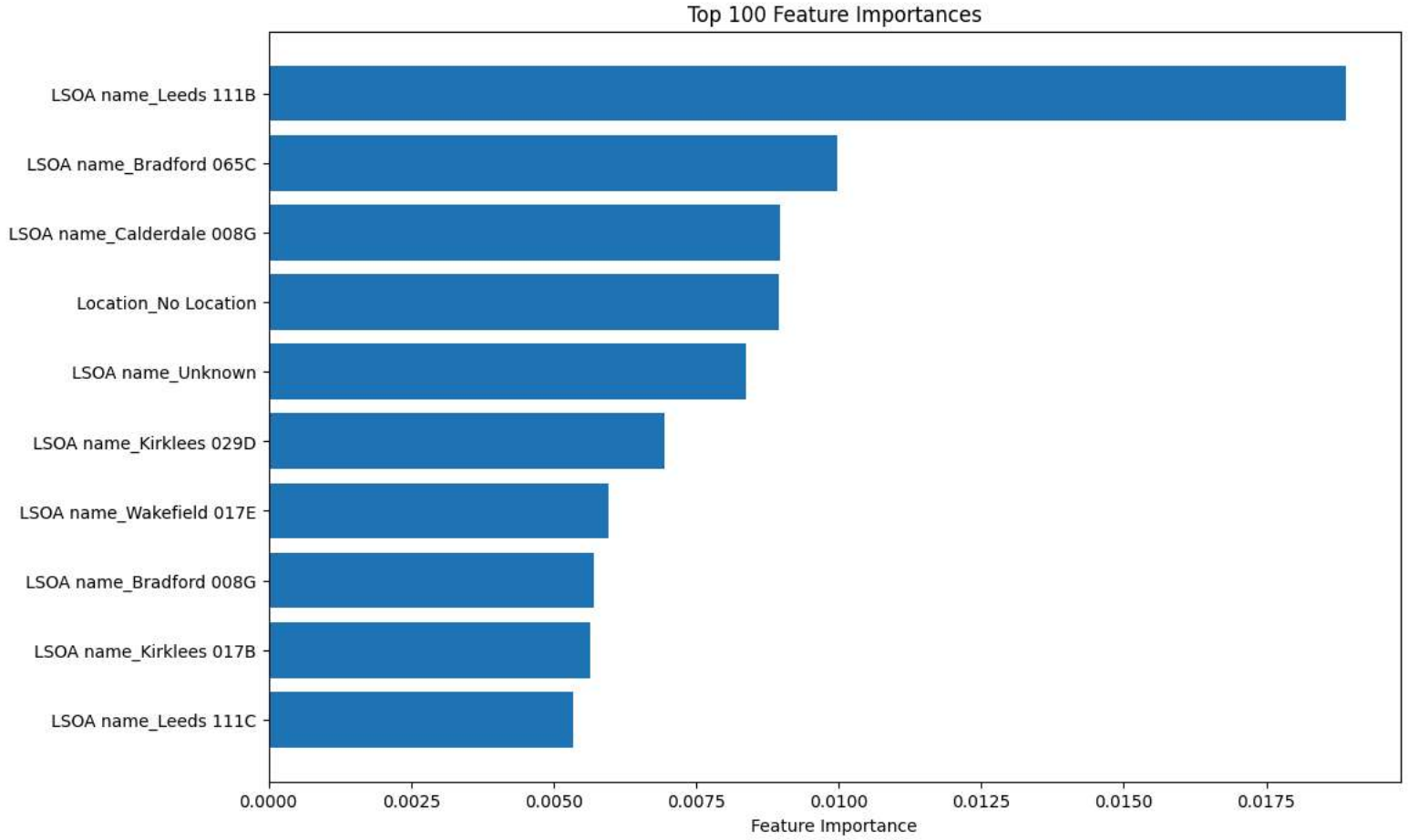
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid_rf = {
    'n_estimators': [100, 200, 300],
    'max_depth': [10, 15, 20, None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize the RandomForestClassifier
rf = RandomForestClassifier(class_weight='balanced', random_state=42)

# Use GridSearchCV for hyperparameter tuning
grid_search_rf = GridSearchCV(estimator=rf, param_grid=param_grid_rf, cv=5, scoring='f1_weighted', verbose=2, n_jobs=-1)
grid_search_rf.fit(X_train_resampled, y_train_resampled)

# Display the best parameters found by GridSearchCV
print("Best Parameters found by GridSearchCV for RandomForest:")
print(grid_search_rf.best_params_)

# Use the best estimator to predict on the test set
best_rf_model = grid_search_rf.best_estimator_
y_pred_rf_optimized = best_rf_model.predict(X_test)

# Evaluate the optimized model
print("Optimized Random Forest Model Classification Report:")
print(classification_report(y_test, y_pred_rf_optimized))
print("Confusion Matrix of Optimized Random Forest Model:")
print(confusion_matrix(y_test, y_pred_rf_optimized))
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

```
features_df.head(10)
```

	Feature	Importance
10180	LSOA name_Leeds 111B	0.018903
9319	LSOA name_Bradford 065C	0.009979
9357	LSOA name_Calderdale 008G	0.008968
5727	Location_No Location	0.008952
10189	LSOA name_Unknown	0.008382
9579	LSOA name_Kirklees 029D	0.006952
10268	LSOA name_Wakefield 017E	0.005971
9043	LSOA name_Bradford 008G	0.005715
9523	LSOA name_Kirklees 017B	0.005634
10181	LSOA name_Leeds 111C	0.005347

```
# Select the top 100 features
top_features = features_df.head(10)

# Plot feature importances
plt.figure(figsize=(12, 8))
plt.barh(top_features['Feature'], top_features['Importance'])
plt.xlabel('Feature Importance')
plt.title('Top 100 Feature Importances')
plt.gca().invert_yaxis()  # Invert y-axis to have the most important feature at the top
plt.show()
```


Top 100 Feature Importances
LSOA name_Leeds 111B
LSOA name_Bradford 065C
LSOA name_Calderdale 008G
Location_No Location
LSOA name_Unknown
LSOA name_Kirklees 029D
LSOA name_Wakefield 017E
LSOA name_Bradford 008G
LSOA name_Kirklees 017B
LSOA name_Leeds 111C
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
Feature Importance

```
df_clean
```

	Month	Reported by	Falls within	Location	LSOA name	Crime type	Simplified Outcome	Zone	Crime type_encoded	Unique Crime Types
0	2024-01	West Yorkshire Police	West Yorkshire Police	Wood Royd Hill Lane	Barnsley 027D	Violence and sexual offences	Unprosecuted	2	13	1
1	2024-01	West Yorkshire Police	West Yorkshire Police	Cross End Fold	Bradford 001A	Shoplifting	Unresolved	8	10	1
2	2024-01	West Yorkshire Police	West Yorkshire Police	Millfold	Bradford 001A	Violence and sexual offences	Unprosecuted	8	13	1
3	2024-01	West Yorkshire Police	West Yorkshire Police	Ridleys Fold	Bradford 001B	Burglary	Unresolved	8	2	1
4	2024-01	West Yorkshire Police	West Yorkshire Police	School Lane	Bradford 001C	Anti-social behaviour	Unknown	8	0	5
...	...	...	...	...	...	...	...	...	...	...
23805	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Other crime	Pending	1	5	14
23806	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Other crime	Pending	1	5	14
23807	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Other crime	Unprosecuted	1	5	14
23808	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Other crime	Unprosecuted	1	5	14
23809	2024-01	West Yorkshire Police	West Yorkshire Police	No Location	Unknown	Other crime	Unresolved	1	5	14

23810 rows × 10 columns

```
df_encoded
```