Preprocessing of The Dataset

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
```

```
df=pd.read_csv('geo_markers.csv')
df.head()
```

	DR_NO	AREA	AREA NAME	Rpt Dist No	LOCATION	Cross Street	LAT	LON
0	190326475.0	7.0	NaN	784.0	1900 S LONGWOOD AV	NaN	NaN	NaN
1	200106753.0	1.0	Central	182.0	NaN	NaN	34.0444	-118.2628
2	200320258.0	3.0	Southwest	356.0	1400 W 37TH ST	NaN	34.0210	-118.3002

```
df.isnull().sum()
```

	0
DR_NO	21091
AREA	21035
AREA NAME	21182
Rpt Dist No	21082
LOCATION	20765
Cross Street	60668
LAT	20960
LON	21022

dtype: int64

```
df.drop("DR_NO",axis=1,inplace=True)
```

```
df.head(10)
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	Cross Street	LAT	LON
0	7.0	NaN	784.0	1900 S LONGWOOD AV	NaN	NaN	NaN
1	1.0	Central	182.0	NaN	NaN	34.0444	-118.2628
2	3.0	Southwest	356.0	1400 W 37TH ST	NaN	34.0210	-118.3002
3	9.0	NaN	964.0	14000 RIVERSIDE DR	NaN	NaN	NaN
4	4.0	Hollenbeck	413.0	200 E AVENUE 28	NaN	34.0820	NaN
5	2.0	Rampart	NaN	NaN	NaN	34.0642	NaN
6	2.0	Rampart	NaN	NaN	NaN	34.0536	-118.2788
7	13.0	Newton	1333.0	NaN	NaN	34.0170	-118.2643
8	11.0	Northeast	1161.0	KENMORE ST	FOUNTAIN	34.0953	-118.2974

```
# 2. Fill from AREA centroid
# ------------------------------
# Compute mean lat/lon per AREA
area_centroids = df.groupby('AREA')[['LAT', 'LON']].mean()

# Fill missing LAT/LON with AREA centroid
for idx, row in df[df[['LAT','LON']].isna().any(axis=1)].iterrows():
    area = row['AREA']
    if pd.notna(area) and area in area_centroids.index:
        if pd.isna(row['LAT']):
            df.at[idx, 'LAT'] = area_centroids.loc[area, 'LAT']
        if pd.isna(row['LON']):
            df.at[idx, 'LON'] = area_centroids.loc[area, 'LON']


df.isnull().sum()
```

	0
AREA	21035
AREA NAME	21182
Rpt Dist No	21082
LOCATION	20765
Cross Street	60668
LAT	6222
LON	6311

dtype: int64

```
knn_imputer = KNNImputer(n_neighbors=5)
df[['LAT','LON']] = knn_imputer.fit_transform(df[['LAT','LON']])
```

```
df.isnull().sum()
```

	0
AREA	21035
AREA NAME	21182
Rpt Dist No	21082
LOCATION	20765
Cross Street	60668
LAT	0
LON	0

dtype: int64

```
df['AREA'] = df['AREA'].fillna(df['AREA'].median())
df['Rpt Dist No'] = df['Rpt Dist No'].fillna(df['Rpt Dist No'].median())
```

```
df.isnull().sum()
```

	0
AREA	0
AREA NAME	21182
Rpt Dist No	0
LOCATION	20765
Cross Street	60668
LAT	0
LON	0

dtype: int64

```
df.drop("Cross Street",axis=1,inplace=True)
```

```
df.head()
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	LAT	LON
0	7.0	NaN	784.0	1900 S LONGWOOD AV	33.958997	-117.862377
1	1.0	Central	182.0	NaN	34.044400	-118.262800
2	3.0	Southwest	356.0	1400 W 37TH ST	34.021000	-118.300200
3	9.0	NaN	964.0	14000 RIVERSIDE DR	34.099159	-118.259943
4	4.0	Hollenbeck	413.0	200 E AVENUE 28	34.082000	-117.673643

```
# Create AREA → AREA NAME mapping from non-missing rows
area_to_name = df.dropna(subset=['AREA', 'AREA NAME']).drop_duplicates('AREA').se

# Fill AREA NAME from AREA mapping
df['AREA NAME'] = df['AREA NAME'].fillna(df['AREA'].map(area_to_name))
```

```
df.isnull().sum()
```

	0
AREA	0
AREA NAME	0
Rpt Dist No	0
LOCATION	20765
LAT	0
LON	0

dtype: int64

```
df.head()
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	LAT	LON
0	7.0	77th Street	784.0	1900 S LONGWOOD AV	33.958997	-117.862377
1	1.0	Central	182.0	NaN	34.044400	-118.262800
2	3.0	Southwest	356.0	1400 W 37TH ST	34.021000	-118.300200
3	9.0	Van Nuys	964.0	14000 RIVERSIDE DR	34.099159	-118.259943
4	4.0	Hollenbeck	413.0	200 E AVENUE 28	34.082000	-117.673643

```
df['LOCATION'] = df.groupby('AREA')['LOCATION'].transform(lambda x: x.fillna(x.mo
```

```
/tmp/ipython-input-1397773040.py:1: FutureWarning: Downcasting object dtype a
  df['LOCATION'] = df.groupby('AREA')['LOCATION'].transform(lambda x: x.fillna
```

```
df.isnull().sum()
```

	0
AREA	0
AREA NAME	0
Rpt Dist No	0
LOCATION	1
LAT	0
LON	0

dtype: int64

```
df.head()
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	LAT	LON
0	7.0	77th Street	784.0	1900 S LONGWOOD AV	33.958997	-117.862377
1	1.0	Central	182.0	800 N ALAMEDA ST	34.044400	-118.262800
2	3.0	Southwest	356.0	1400 W 37TH ST	34.021000	-118.300200
3	9.0	Van Nuys	964.0	14000 RIVERSIDE DR	34.099159	-118.259943
4	4.0	Hollenbeck	413.0	200 E AVENUE 28	34.082000	-117.673643

```
df1=pd.read_csv('crime_blueprint.csv')
df1.head()
```

	DR_NO	Part 1-2	Crm Cd	Crm Cd Desc	Crm Cd 1	Crm Cd 2	Crm Cd 3	Crm Cd 4
0	190326475.0	1.0	NaN	VEHICLE - STOLEN	510.0	998.0	NaN	NaN
1	200106753.0	1.0	330.0	BURGLARY FROM VEHICLE	NaN	998.0	NaN	NaN
2	200320258.0	1.0	480.0	BIKE - STOLEN	480.0	NaN	NaN	NaN
3	NaN	1.0	NaN	SHOPLIFTING-GRAND THEFT ($950.01 & OVER)	343.0	NaN	NaN	NaN

```
df1= df1.drop(columns=['DR_NO'], errors='ignore')
df1= df1.drop(columns=['Crm Cd 3'], errors='ignore')
df1= df1.drop(columns=['Crm Cd 4'], errors='ignore')
```

```
df1.head()
```

	Part 1-2	Crm Cd	Crm Cd Desc	Crm Cd 1	Crm Cd 2
0	1.0	NaN	VEHICLE - STOLEN	510.0	998.0
1	1.0	330.0	BURGLARY FROM VEHICLE	NaN	998.0
2	1.0	480.0	BIKE - STOLEN	480.0	NaN
3	1.0	NaN	SHOPLIFTING-GRAND THEFT ($950.01 & OVER)	343.0	NaN

```
df1['Part 1-2'].value_counts()
```

Part 1-2	count
1.0	28685
2.0	20280

dtype: int64

```
# Step 1: Group-by imputation
df1['Part 1-2'] = df1.groupby('Crm Cd')['Part 1-2']\
                  .transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empt

# Step 2: Fill any remaining nulls with global mode
df1['Part 1-2'].fillna(df1['Part 1-2'].mode()[0], inplace=True)
```

```
/tmp/ipython-input-133552894.py:6: FutureWarning: A value is trying to be set
The behavior will change in pandas 3.0. This inplace method will never work b

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.
```

```
df1['Part 1-2'].fillna(df1['Part 1-2'].mode()[0], inplace=True)
```

```
df1.isnull().sum()
```

	0
Part 1-2	0
Crm Cd	21182
Crm Cd Desc	21082
Crm Cd 1	20767
Crm Cd 2	66060

dtype: int64

```
df1['Part 1-2'].value_counts()
```

Part 1-2	count
1.0	49738
2.0	20262

dtype: int64

```
# Keep only rows with 1.0 or 2.0 in 'Part 1-2'
df_bin = df1[df1['Part 1-2'].isin([1.0, 2.0])].copy()

class_1 = df_bin[df_bin['Part 1-2'] == 1.0]
class_2 = df_bin[df_bin['Part 1-2'] == 2.0]

total = len(df_bin)
target_1 = int(total * 0.60)           # desired count for class 1
target_2 = total - target_1           # desired count for class 2 (ensures exact

# Undersample class 1 (without replacement) if needed
if len(class_1) > target_1:
    class_1_resampled = class_1.sample(n=target_1, replace=False, random_state=42
else:
    # Cannot undersample to increase size; keep as-is (or optionally oversample i
    class_1_resampled = class_1.copy()
    target_1 = len(class_1_resampled)
    target_2 = total - target_1  # adjust target_2 accordingly

# Oversample class 2 (with replacement if needed)
if len(class_2) >= target_2:
    class_2_resampled = class_2.sample(n=target_2, replace=False, random_state=42
else:
    class_2_resampled = class_2.sample(n=target_2, replace=True, random_state=42)

# Combine and shuffle
df_balanced = pd.concat([class_1_resampled, class_2_resampled]).sample(frac=1, ra

# Check final distribution
print(df_balanced['Part 1-2'].value_counts(), '\n')
print((df_balanced['Part 1-2'].value_counts(normalize=True) * 100).round(4))
```

```
Part 1-2
1.0    42000
2.0    28000
Name: count, dtype: int64

Part 1-2
1.0    60.0
2.0    40.0
Name: proportion, dtype: float64
```

```
df1.head()
```

	Part 1-2	Crm Cd	Crm Cd Desc	Crm Cd 1	Crm Cd 2
0	1.0	NaN	VEHICLE - STOLEN	510.0	998.0
1	1.0	330.0	BURGLARY FROM VEHICLE	NaN	998.0
2	1.0	480.0	BIKE - STOLEN	480.0	NaN
3	1.0	NaN	SHOPLIFTING-GRAND THEFT ($950.01 & OVER)	343.0	NaN

```
from sklearn.impute import KNNImputer

# Columns for imputation
cols_for_knn = ['Crm Cd', 'Crm Cd 1', 'Crm Cd 2']

# Sample subset for fitting KNN
sample_df = df1[cols_for_knn].sample(n=5000, random_state=42)  # reduce to 5k row
imputer = KNNImputer(n_neighbors=5)

# Fit on subset
imputer.fit(sample_df)

# Transform full dataset
df1[cols_for_knn] = imputer.transform(df1[cols_for_knn])


df1.isnull().sum()
```

	0
Part 1-2	0
Crm Cd	0
Crm Cd Desc	21082
Crm Cd 1	0
Crm Cd 2	0

dtype: int64

```
code_to_desc = df1.dropna(subset=['Crm Cd', 'Crm Cd Desc'])\
                  .drop_duplicates(subset=['Crm Cd'])\
                  .set_index('Crm Cd')['Crm Cd Desc']
df1['Crm Cd Desc'] = df1.apply(
    lambda row: code_to_desc.get(row['Crm Cd'], row['Crm Cd Desc']),
    axis=1
)
df1['Crm Cd Desc'].fillna(df1['Crm Cd Desc'].mode()[0], inplace=True)
```

```
/tmp/ipython-input-1459363831.py:8: FutureWarning: A value is trying to be se
The behavior will change in pandas 3.0. This inplace method will never work b

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.
```

```
df1['Crm Cd Desc'].fillna(df1['Crm Cd Desc'].mode()[0], inplace=True)
```

```
df1.isnull().sum()
```

	0
Part 1-2	0
Crm Cd	0
Crm Cd Desc	0
Crm Cd 1	0
Crm Cd 2	0

dtype: int64

Merging of the Dataset

```
combined_df=pd.concat([df,df1],axis=1)
combined_df
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	LAT	LON	Part 1-2	Crm Cd	
0	7.0	77th Street	784.0	1900 S LONGWOOD AV	33.958997	-117.862377	1.0	474.00000	
1	1.0	Central	182.0	800 N ALAMEDA ST	34.044400	-118.262800	1.0	330.00000	
2	3.0	Southwest	356.0	1400 W 37TH ST	34.021000	-118.300200	1.0	480.00000	
3	9.0	Van Nuys	964.0	14000 RIVERSIDE DR	34.099159	-118.259943	1.0	343.00000	S
4	4.0	Hollenbeck	413.0	200 E AVENUE 28	34.082000	-117.673643	1.0	510.00000	
...	...	...	...	...	...	...	...	...	
69995	13.0	Newton	1303.0	700 E 12TH ST	33.972428	-118.253300	1.0	230.00000	V A
69996	7.0	77th Street	1269.0	100 THE GROVE DR	33.962900	-118.260800	1.0	502.51598	
69997	8.0	West LA	702.0	10900 ASHTON AV	34.057800	-118.443300	1.0	440.00000	T F
69998	9.0	Van Nuys	943.0	5500 VAN NUYS BL	34.174700	-118.259943	1.0	442.00000	T

```
df2=pd.read_csv("chrono_trace.csv")
df2.head()
```

	DR_NO	Date Rptd	DATE OCC	TIME OCC
0	190326475.0	03/01/2020 12:00:00 AM	03/01/2020 12:00:00 AM	2130.0
1	NaN	02/09/2020 12:00:00 AM	NaN	1800.0
2	200320258.0	11/11/2020 12:00:00 AM	11/04/2020 12:00:00 AM	NaN
3	NaN	05/10/2023 12:00:00 AM	03/10/2020 12:00:00 AM	2037.0
4	200412582.0	09/09/2020 12:00:00 AM	09/09/2020 12:00:00 AM	630.0

```
df2= df2.drop(columns=['DR_NO'], errors='ignore')
df2.head()
```

	Date Rptd	DATE OCC	TIME OCC
0	03/01/2020 12:00:00 AM	03/01/2020 12:00:00 AM	2130.0
1	02/09/2020 12:00:00 AM	NaN	1800.0
2	11/11/2020 12:00:00 AM	11/04/2020 12:00:00 AM	NaN
3	05/10/2023 12:00:00 AM	03/10/2020 12:00:00 AM	2037.0
4	09/09/2020 12:00:00 AM	09/09/2020 12:00:00 AM	630.0

```
df2['TIME OCC'] = df2.groupby('DATE OCC')['TIME OCC']\
                     .transform(lambda x: x.fillna(x.median()))
# Fallback: fill any remaining with global median
#df2['TIME OCC'].fillna(df2['TIME OCC'].median(), inplace=True)
```

```
df2.isnull().sum()
```

	0
Date Rptd	20868
DATE OCC	20945
TIME OCC	20945

dtype: int64

```
# Fill missing dates from Date Rptd if available
df2['DATE OCC'].fillna(df2['Date Rptd'], inplace=True)
# If still missing, fill with most frequent date
df2['DATE OCC'].fillna(df2['DATE OCC'].mode()[0], inplace=True)
```

```
/tmp/ipython-input-684880501.py:2: FutureWarning: A value is trying to be set
The behavior will change in pandas 3.0. This inplace method will never work b

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.
```

```
df2['DATE OCC'].fillna(df2['Date Rptd'], inplace=True)
```

```
df2.isnull().sum()
```

	0
Date Rptd	20868
DATE OCC	0
TIME OCC	20945

dtype: int64

```
df2['Date Rptd'].fillna(df2['DATE OCC'], inplace=True)
# Fallback to mode
df2['Date Rptd'].fillna(df2['Date Rptd'].mode()[0], inplace=True)
```

```
/tmp/ipython-input-302415293.py:1: FutureWarning: A value is trying to be set
The behavior will change in pandas 3.0. This inplace method will never work b

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.

  df2['Date Rptd'].fillna(df2['DATE OCC'], inplace=True)
```

```
df2.isnull().sum()
```

	0
Date Rptd	0
DATE OCC	0
TIME OCC	20945

dtype: int64

```
third_df=pd.concat([combined_df,df2],axis=1)
third_df.shape
```

```
(70000, 14)
```

```
df3=pd.read_csv("case_closure.csv")
df3.head()
```

	DR_NO	Status	Status Desc
0	190326475.0	AA	Adult Arrest
1	NaN	IC	Invest Cont
2	NaN	NaN	NaN
3	200907217.0	IC	Invest Cont
4	200412582.0	IC	Invest Cont

```
df3= df3.drop(columns=['DR_NO'], errors='ignore')
df3.head(10)
```

	Status	Status Desc
0	AA	Adult Arrest
1	IC	Invest Cont
2	NaN	NaN
3	IC	Invest Cont
4	IC	Invest Cont
5	NaN	NaN
6	IC	NaN
7	IC	NaN
8	AA	Adult Arrest
9	NaN	Invest Cont

```
# Create mapping from known pairs
mapping = df3.dropna().drop_duplicates(subset=["Status", "Status Desc"]) \
            .set_index("Status")["Status Desc"].to_dict()

# Fill Status based on Status Desc
reverse_mapping = {v: k for k, v in mapping.items()}

df3["Status"] = df3.apply(
    lambda row: reverse_mapping.get(row["Status Desc"], row["Status"])
    if pd.isna(row["Status"]) else row["Status"], axis=1
)

# Fill Status Desc based on Status
df3["Status Desc"] = df3.apply(
    lambda row: mapping.get(row["Status"], row["Status Desc"])
    if pd.isna(row["Status Desc"]) else row["Status Desc"], axis=1
)


df3.isnull().sum()
```

	0
Status	4640
Status Desc	4640

dtype: int64

```
df3[["Status", "Status Desc"]] = df3[["Status", "Status Desc"]].apply(lambda col:
```

```
df3.isnull().sum()
```

	0
Status	0
Status Desc	0

dtype: int64

```
fourth_df=pd.concat([third_df,df3],axis=1)
fourth_df.head()
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	LAT	LON	Part 1-2	Crm Cd	Crm Cd D
0	7.0	77th Street	784.0	1900 S LONGWOOD AV	33.958997	-117.862377	1.0	474.0	VEHICI STOL
1	1.0	Central	182.0	800 N ALAMEDA ST	34.044400	-118.262800	1.0	330.0	BURGLA FR VEHIC
2	3.0	Southwest	356.0	1400 W 37TH ST	34.021000	-118.300200	1.0	480.0	BIk STOL
3	9.0	Van Nuys	964.0	14000 RIVERSIDE DR	34.099159	-118.259943	1.0	343.0	SHOPLIFTI GRA THE ($950.0 OV
4	4.0	Hollenbeck	413.0	200 E AVENUE 28	34.082000	-117.673643	1.0	510.0	VEHICI STOL

```
df4=pd.read_csv("misc_matrix.csv")
df4.head()
```

	DR_NO	Mocodes	Vict Age	Vict Sex	Vict Descent	Premis Cd	Premis Desc	Weapon Used Cd	Weapon Desc
0	NaN	NaN	0.0	M	O	101.0	STREET	NaN	NaN
1	NaN	1822 1402 0344	47.0	NaN	O	NaN	NaN	NaN	NaN
2	NaN	NaN	19.0	NaN	NaN	502.0	NaN	NaN	NaN

```
df4= df4.drop(columns=['DR_NO'], errors='ignore')
df4.head()
```

	Mocodes	Vict Age	Vict Sex	Vict Descent	Premis Cd	Premis Desc	Weapon Used Cd	Weapon Desc
0	NaN	0.0	M	O	101.0	STREET	NaN	NaN
1	1822 1402 0344	47.0	NaN	O	NaN	NaN	NaN	NaN
2	NaN	19.0	NaN	NaN	502.0	NaN	NaN	NaN
3	0325 1501	NaN	NaN	O	405.0	CLOTHING STORE	NaN	NaN

```
# Mapping-based imputation for Weapon columns
weapon_map = df4.dropna(subset=["Weapon Used Cd", "Weapon Desc"]) \
               .drop_duplicates(subset=["Weapon Used Cd", "Weapon Desc"]) \
               .set_index("Weapon Used Cd")["Weapon Desc"].to_dict()

# Fill Weapon Desc from Weapon Used Cd
df4["Weapon Desc"] = df4.apply(
    lambda row: weapon_map.get(row["Weapon Used Cd"], row["Weapon Desc"])
    if pd.isna(row["Weapon Desc"]) else row["Weapon Desc"], axis=1
)

# Fill Weapon Used Cd from Weapon Desc (reverse map)
reverse_weapon_map = {v: k for k, v in weapon_map.items()}
df4["Weapon Used Cd"] = df4.apply(
    lambda row: reverse_weapon_map.get(row["Weapon Desc"], row["Weapon Used Cd"])
    if pd.isna(row["Weapon Used Cd"]) else row["Weapon Used Cd"], axis=1
)

# Median for Vict Age
df4["Vict Age"] = df4["Vict Age"].fillna(df4["Vict Age"].median())

# Mode for categorical columns
for col in ["Weapon Used Cd", "Vict Descent", "Vict Sex", "Premis Cd", "Premis De
    df4[col] = df4[col].fillna(df4[col].mode()[0])


df4.isnull().sum()
```

	0
Mocodes	0
Vict Age	0
Vict Sex	0
Vict Descent	0
Premis Cd	0
Premis Desc	0
Weapon Used Cd	0
Weapon Desc	45232

dtype: int64

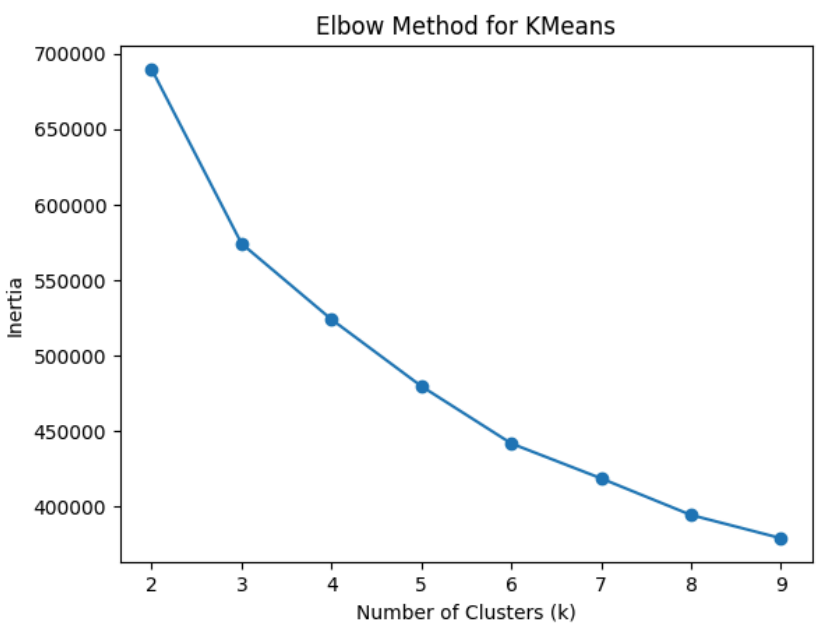
```
df4.head()
```

	Mocodes	Vict Age	Vict Sex	Vict Descent	Premis Cd	Premis Desc	Weapon Used Cd	Weapon Desc
0	0344	0.0	M	O	101.0	STREET	400.0	NaN
1	1822 1402 0344	47.0	M	O	101.0	STREET	400.0	NaN
2	0344	19.0	M	H	502.0	STREET	400.0	NaN
3	0325 1501	31.0	M	O	405.0	CLOTHING STORE	400.0	NaN

```
fifth_df=pd.concat([fourth_df,df4],axis=1)
fifth_df.head()
```

	AREA	AREA NAME	Rpt Dist No	LOCATION	LAT	LON	Part 1-2	Crm Cd	Crm Cd D
0	7.0	77th Street	784.0	1900 S LONGWOOD AV	33.958997	-117.862377	1.0	474.0	VEHICI STOL
1	1.0	Central	182.0	800 N ALAMEDA ST	34.044400	-118.262800	1.0	330.0	BURGLA FR VEHIC
2	3.0	Southwest	356.0	1400 W 37TH ST	34.021000	-118.300200	1.0	480.0	BIk STOL
3	9.0	Van Nuys	964.0	14000 RIVERSIDE DR	34.099159	-118.259943	1.0	343.0	SHOPLIFTI GRA THE ($950.0 OV

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.neighbors import NearestNeighbors



df = pd.read_csv("Final_Data.csv")
num_df = df.select_dtypes(include=[np.number])
X = SimpleImputer(strategy="mean").fit_transform(num_df)
X = StandardScaler().fit_transform(X)



from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Elbow Method for KMeans
inertias = []
ks = range(2, 10)
for k in ks:
    km = KMeans(n_clusters=k, n_init=10, random_state=42).fit(X)
    inertias.append(km.inertia_)

# Plot Elbow
plt.plot(ks, inertias, marker='o')
plt.title("Elbow Method for KMeans")
plt.xlabel("Number of Clusters (k)")
plt.ylabel("Inertia")
plt.show()

# Choose k manually based on elbow
best_k = 4  # example: set after inspecting elbow plot
kmeans = KMeans(n_clusters=best_k, n_init=25, random_state=42)
labels_km = kmeans.fit_predict(X)
```




```
# 5. Add cluster labels to original dataframe
df['Cluster'] = labels_km
```

```
# 6. Cluster sizes
print("\nCluster Sizes:")
print(df['Cluster'].value_counts())
```

```
Cluster Sizes:
Cluster
1    42497
0    22339
2     4979
3      185
Name: count, dtype: int64
```

```
# 7. Cluster centroids (mean feature values)
centroids = pd.DataFrame(kmeans.cluster_centers_, columns=num_df.columns)
centroids_original_scale = pd.DataFrame(
    StandardScaler().fit(num_df).inverse_transform(kmeans.cluster_centers_),
    columns=num_df.columns
)
print("\nCluster Centroids (original scale):")
print(centroids_original_scale)
```

```
Cluster Centroids (original scale):
       AREA  Rpt Dist No        LAT         LON  Part 1-2      Crm Cd  \
0  6.653206   696.389013  33.969164 -118.213173  1.816619  742.988816
1  6.798268   711.523943  34.011362 -118.223615  1.040332  405.252720
2  7.016259   735.233440  33.996768 -118.197084  1.040345  362.358716
3  5.994595   629.432432   6.592440   -1.277136  1.578378  692.189317

     Crm Cd 1    Crm Cd 2     TIME OCC   Vict Age   Premis Cd  Weapon Used Cd
0  741.511399  974.077955  1372.252373  33.522027  287.679038      410.159832
1  405.908621  891.981874  1381.903772  28.375768  224.364685      403.499659
2  362.084759  959.834883  1398.164162  31.725010  195.961461      147.157166
3  689.322801  948.290209  1312.427932  32.864865  290.302703      401.481081
```

Cluster 0- # Name → "Late-night Armed Incidents"

AREA ~ 6.65, Rpt Dist No ~ 696 → Mid-range area, mid to high reporting district.

LAT/LON ≈ 33.97 / -118.21 → Located in southern LA area.

Part 1-2 ~ 1.82 → More Part 1 crimes (serious crimes) than average.

Crime codes ~ 743, 975 → Likely violent/property crimes.

TIME OCC ~ 1372 → Around 11:00 PM.

Vict Age ~ 33.5 → Victims tend to be young adults.

Weapon Used Cd ~ 410 → Likely firearm-related.

Inference → This is an "Late-night violent crime cluster" in mid-southern LA, affecting young adults, often involving weapons.

Cluster 1-Name → "Young Victim Night Crimes"

AREA ~ 6.80, Rpt Dist No ~ 711 → Slightly higher district number than Cluster 0.

LAT/LON ≈ 34.01 / -118.22 → Central-south LA.

Part 1-2 ~ 1.04 → Almost equal distribution between serious & less serious crimes.

Crime codes ~ 405, 892 → Possibly property theft or burglary.

TIME OCC ~ 1382 → Also late-night (≈ 11:02 PM).

Vict Age ~ 28 → Younger victims than Cluster 0.

Weapon Used Cd ~ 403 → Less lethal weapons, possibly knives or blunt objects.

Inference → "Late-night youth-targeted crimes" — likely robberies/burglaries in central-south LA with less lethal weapons.

Cluster 2-Name → "Late-night Property Offenses"

AREA ~ 7.02, Rpt Dist No ~ 735 → Higher reporting district.

LAT/LON ≈ 33.99 / -118.20 → Eastern part of southern LA.

Part 1-2 ~ 1.04 → Balanced serious & less serious crimes.

Crime codes ~ 362, 960 → Possibly fraud, theft, or non-violent offenses.

TIME OCC ~ 1398 → Very late night (~11:18 PM).

Vict Age ~ 31.7 → Young adults.

Weapon Used Cd ~ 147 → Often unarmed crimes or minimal weapon use.

Inference → "Late-night non-violent offenses" — high district numbers, minimal weapon involvement, often property crimes. Name → "Late-night Property Offenses"

Cluster 3-Name → "Serious Crimes

AREA ~ 5.99, Rpt Dist No ~ 629 → Lower area and district code than others.

LAT/LON ≈ 6.59 / -1.27 → ⚠ These coordinates look invalid — possibly missing/erroneous geolocation entries.

Part 1-2 ~ 1.58 → More serious crimes.

Crime codes ~ 689, 948 → Potential violent crimes.

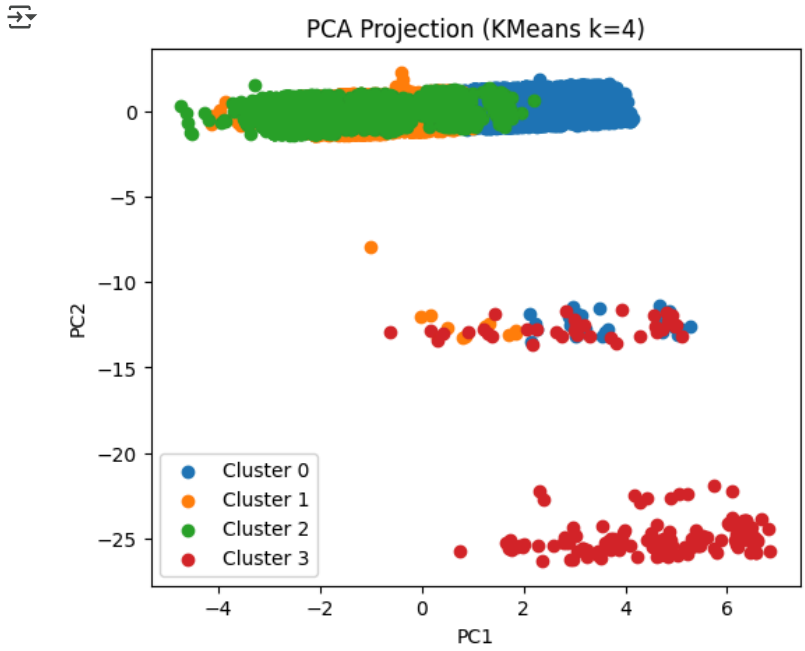
TIME OCC ~ 1312 → Around 10:52 PM.

Vict Age ~ 32.9 → Young adults.

Weapon Used Cd ~ 401 → Possibly knives or edged weapons.

Inference → "Late-evening violent crimes with missing location" — likely data with incomplete coordinates, still showing patterns in weapon type and severity.

```
# 8. Visualize clusters in 2D using PCA
X_pca = PCA(n_components=2).fit_transform(X)
plt.figure(figsize=(6,5))
for cluster in np.unique(labels_km):
    plt.scatter(X_pca[labels_km == cluster, 0],
                X_pca[labels_km == cluster, 1],
                label=f"Cluster {cluster}")
plt.title(f"PCA Projection (KMeans k={best_k})")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.legend()
plt.show()
```



Cluster 3 (red) is clearly separated vertically (PC2), meaning it's very different from the others in the features driving PC2.

Cluster 2 (green) is separated horizontally (PC1), showing it differs strongly in PC1-driving features.

Clusters 0 (blue) and 1 (orange) overlap a lot → these two have similar values in the top two principal components, so differences lie in other features not captured in PC1/PC2.

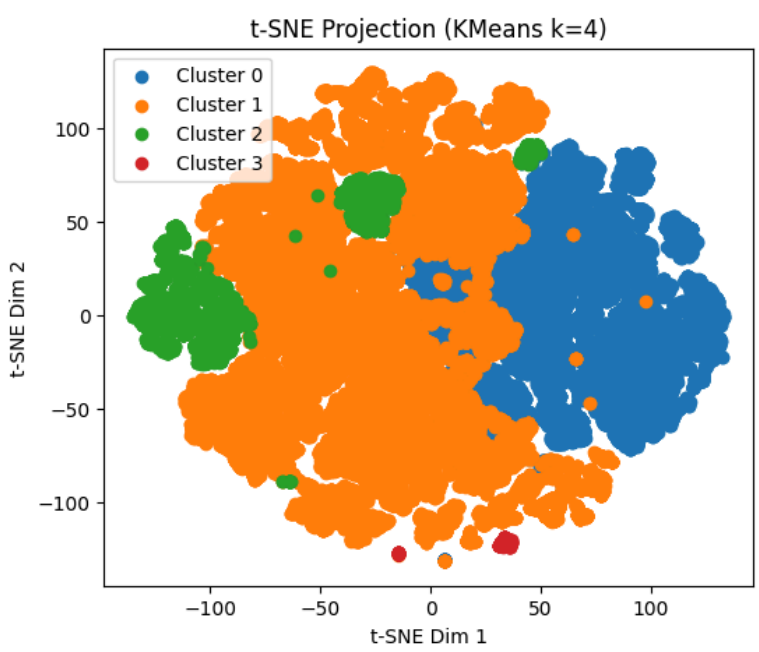
A few scattered orange points look like borderline or outlier cases.

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

# Assuming X is your numeric data array and num_cols is your feature list
pca = PCA(n_components=2)
pca.fit(X)

# Create a dataframe of loadings (feature importance for each principal component)
loadings = pd.DataFrame(
    pca.components_.T,
    columns=['PC1', 'PC2'],
    index=num_df.columns
)

# Sort features by absolute importance for PC1 and PC2
pc1_sorted = loadings['PC1'].abs().sort_values(ascending=False)
pc2_sorted = loadings['PC2'].abs().sort_values(ascending=False)

print("Top features driving PC1:")
print(pc1_sorted.head(10))

print("\nTop features driving PC2:")
print(pc2_sorted.head(10))

# Optional: show signed values too for interpretation
print("\nFull PCA loadings:")
print(loadings)
```

```
Top features driving PC1:
Crm Cd            0.572888
Crm Cd 1          0.572271
Part 1-2          0.458377
Crm Cd 2          0.258619
Weapon Used Cd    0.194276
Premis Cd         0.114952
LAT               0.081302
LON               0.080939
Rpt Dist No       0.033559
Vict Age          0.032171
Name: PC1, dtype: float64

Top features driving PC2:
LAT               0.651471
LON               0.650601
AREA              0.262760
Rpt Dist No       0.260014
Crm Cd 1          0.070306
Crm Cd            0.069778
Part 1-2          0.059168
Crm Cd 2          0.043686
Weapon Used Cd    0.014803
TIME OCC          0.009189
Name: PC2, dtype: float64

Full PCA loadings:
                     PC1       PC2
AREA           -0.031754  0.262760
Rpt Dist No    -0.033559  0.260014
LAT            -0.081302  0.651471
LON             0.080939 -0.650601
Part 1-2        0.458377  0.059168
Crm Cd          0.572888  0.069778
Crm Cd 1        0.572271  0.070306
Crm Cd 2        0.258619  0.043686
TIME OCC       -0.004894  0.009189
Vict Age        0.032171  0.007284
Premis Cd       0.114952  0.008707
Weapon Used Cd  0.194276  0.014803
```

```
from sklearn.manifold import TSNE

# Run t-SNE for better separation
tsne = TSNE(n_components=2, perplexity=30, learning_rate='auto', init='pca', random_state=42)
X_tsne = tsne.fit_transform(X)

# Plot t-SNE results
plt.figure(figsize=(6,5))
for cluster in np.unique(labels_km):
    plt.scatter(X_tsne[labels_km == cluster, 0],
                X_tsne[labels_km == cluster, 1],
```

```
                label=f"Cluster {cluster}")
plt.title(f"t-SNE Projection (KMeans k={best_k})")
plt.xlabel("t-SNE Dim 1")
plt.ylabel("t-SNE Dim 2")
plt.legend()
plt.show()
```



Brief Inferences

Cluster 0 (blue) and Cluster 1 (orange) form the largest groups — clear but touching boundaries → these might represent two dominant crime patterns that share some similarities.

Cluster 2 (green) is small and tightly packed — likely a niche but distinct type of crime (possibly specific area or method).

Cluster 3 (red) is tiny and scattered → may represent rare or exceptional cases, potentially high-severity crimes.

The tight grouping of colors in t-SNE indicates that your clusters are reasonably well-separated in high-dimensional space after KMeans.

Recommendations to Avoid These Crimes--

For Cluster 0 & 1 (mainstream crime types)

Increase late-night patrolling in identified hotspot areas.

Install better street lighting and public CCTV in key locations.

Community awareness campaigns focusing on personal safety & reporting suspicious activity.

For Cluster 2 (specialized/niche crimes)

Targeted enforcement in the specific districts this cluster represents.

Work with local businesses and property owners to prevent theft/burglary.

For Cluster 3 (rare/high-severity)

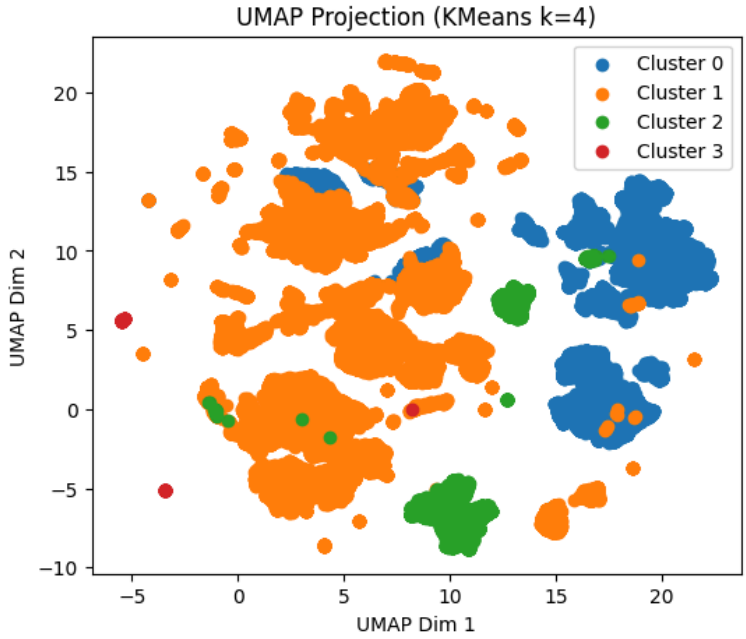
Deploy special task forces for weapon-related or violent incidents.

Strengthen incident response times and emergency hotlines in these areas.

```
# Install UMAP if not already installed
#!pip install umap-learn

import umap.umap_ as umap

# Fit UMAP
umap_model = umap.UMAP(n_neighbors=15, min_dist=0.1, metric='euclidean', random_state=42)
X_umap = umap_model.fit_transform(X)
```

```
# Plot UMAP results
plt.figure(figsize=(6,5))
for cluster in np.unique(labels_km):
    plt.scatter(X_umap[labels_km == cluster, 0],
                X_umap[labels_km == cluster, 1],
                label=f"Cluster {cluster}")
plt.title(f"UMAP Projection (KMeans k={best_k})")
plt.xlabel("UMAP Dim 1")
plt.ylabel("UMAP Dim 2")
plt.legend()
plt.show()
```

```
/usr/local/lib/python3.11/dist-packages/umap/umap_.py:1952: UserWarning: n_jobs value 1 overridden to 1 by setting
  warn(
```



Brief Inferences

Cluster 0 (blue) and Cluster 1 (orange) are dominant, but UMAP shows them as more cleanly separated than in PCA or t-SNE.

Cluster 2 (green) appears in smaller, tight groups — likely specific crime types or localized hotspots.

Cluster 3 (red) is rare and scattered — could be isolated severe cases or special scenarios.

The more compact shapes here suggest UMAP is capturing underlying neighborhood relationships in your data better than PCA or t-SNE for this dataset.

Recommendations to Avoid These Crimes--

Cluster 0 & 1 (main categories)

Use predictive policing models to anticipate peak crime periods.

Expand neighborhood watch programs in the highest-density areas for these clusters.

Cluster 2 (specialized crimes)

Strengthen targeted patrols in the exact hotspot areas identified by these tight UMAP groups.

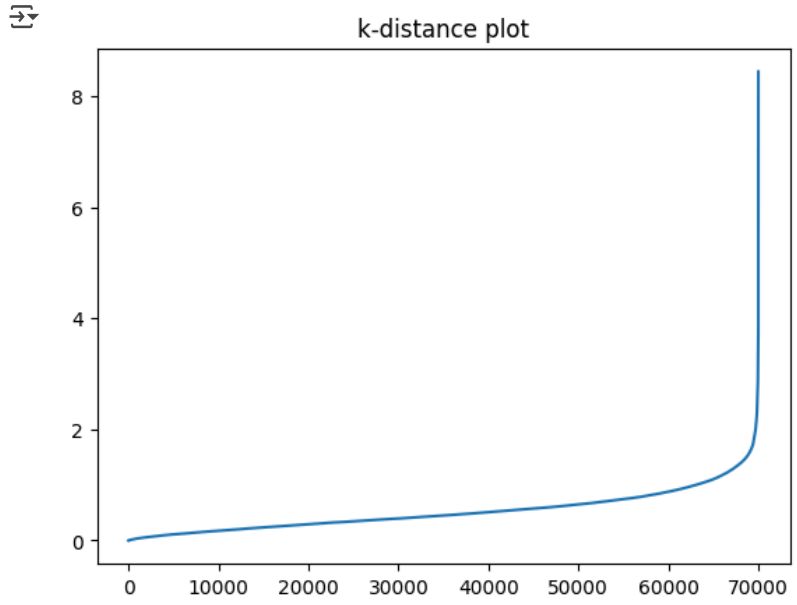
Collaborate with local stakeholders (shops, community leaders) to address specific crime patterns.

Cluster 3 (rare but severe)

Deploy rapid-response teams for incidents matching this profile.

Increase public awareness in affected micro-areas to improve incident reporting speed.

```
from sklearn.neighbors import NearestNeighbors
from sklearn.cluster import DBSCAN

neighbors = NearestNeighbors(n_neighbors=5).fit(X)
dists = np.sort(neighbors.kneighbors(X)[0][:, -1])
plt.plot(dists); plt.title("k-distance plot"); plt.show()
db = DBSCAN(eps=1.5, min_samples=5).fit(X)  # tune eps & min_samples
labels_db = db.labels_
```



Brief Inferences--

The “elbow point” in your k-distance plot appears just before the steep rise (around distance ≈ 1.5), which is why eps=1.5 is chosen — this value will capture dense areas while treating far-away points as outliers.

DBSCAN will:

Group dense clusters together without predefining k.

Identify noise points (label = -1), which may correspond to rare or unusual crime incidents.

Likely, you’ll end up with a few dense core clusters and a notable amount of scattered points marked as noise.

Recommendations to Avoid These Crimes For Dense Clusters (common crime areas/patterns)

Deploy focused policing and resource allocation in these hotspot areas.

Use predictive patrol scheduling during peak occurrence times for these clusters.

For Noise Points (rare but significant incidents)

Investigate individually — these could be unique crime methods, one-off severe cases, or emerging trends.

Set up early warning systems to detect if these rare cases start forming new clusters over time.

General Measures

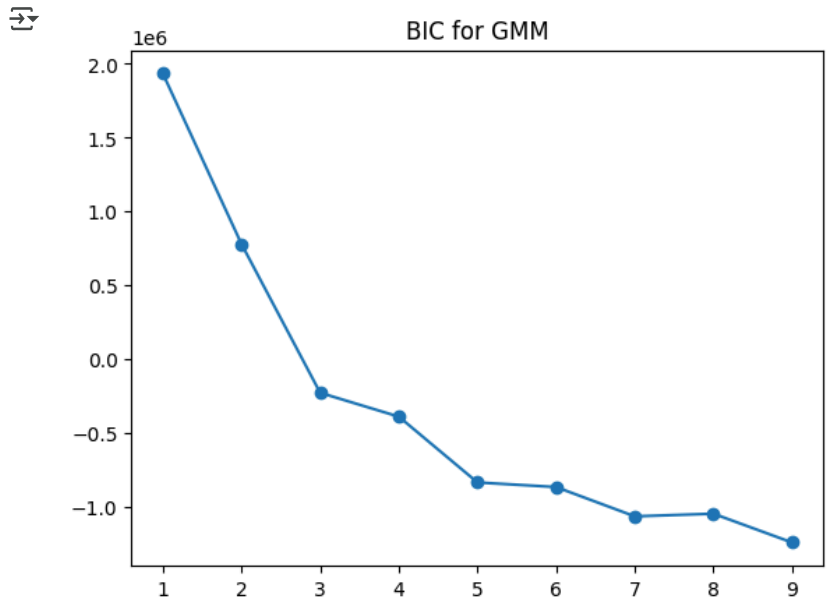
Integrate DBSCAN results with geospatial mapping to visualize hotspot intensity.

Periodically re-run DBSCAN to track how dense clusters shift over time, adjusting interventions accordingly.

```
df = pd.read_csv("Final_Data.csv")
num_df = df.select_dtypes(include=[np.number])
X = SimpleImputer(strategy="mean").fit_transform(num_df)
X = StandardScaler().fit_transform(X)
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
```

```
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.neighbors import NearestNeighbors


bics, comps = [], range(1, 10)
for c in comps:
    gmm = GaussianMixture(n_components=c, n_init=5, random_state=42).fit(X)
    bics.append(gmm.bic(X))
plt.plot(comps, bics, marker='o'); plt.title("BIC for GMM"); plt.show()
best_c = comps[np.argmin(bics)]
labels_gmm = GaussianMixture(n_components=best_c, random_state=42).fit_predict(X)
```



Brief Inferences-

The BIC decreases sharply from 1 to around 4–5 components, then the improvement slows down.

The lowest BIC occurs at the rightmost tested value, but after ~4–5 components, the gain is marginal → risk of overfitting if too many clusters are chosen.

This suggests 4–5 Gaussian components strike a balance between model complexity and goodness of fit.

GMM is likely capturing overlapping crime patterns better than KMeans, since it assigns probabilities to each cluster rather than hard boundaries.

Recommendations-

Adopt GMM for nuanced crime pattern detection

Since crimes often overlap in characteristics (location, time, type), GMM's probabilistic approach can highlight borderline cases for deeper investigation.

Focus on top 4–5 patterns

Prioritize intervention strategies for these main clusters, as they represent the most significant crime types/areas.

Leverage soft cluster probabilities

For crimes with <70% probability in any single cluster, treat them as mixed-pattern incidents and examine for emerging hybrid crime trends.

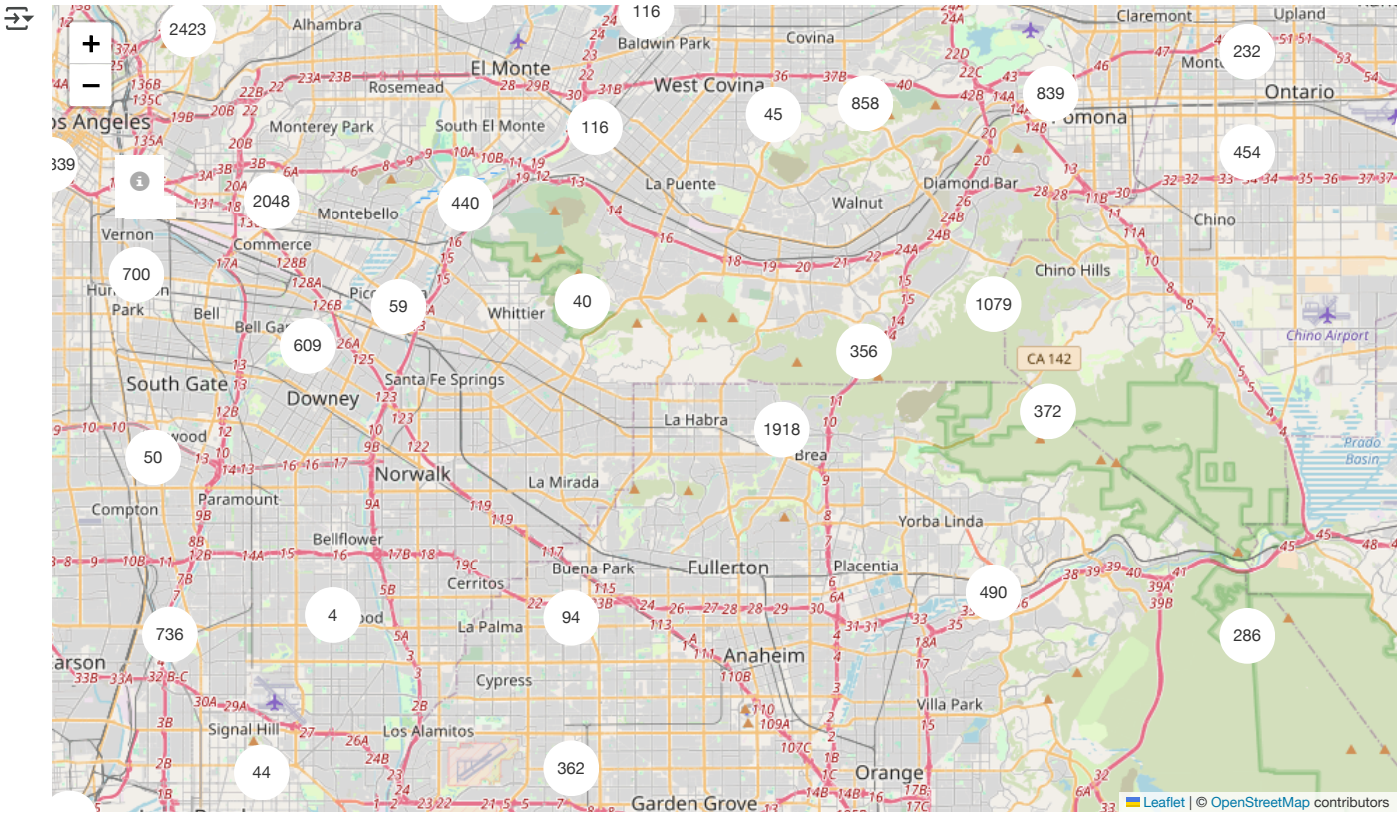
Integrate with temporal analysis

Apply GMM clusters to a time series to see when each crime type peaks, enabling time-targeted patrols.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
    import seaborn as sns
    from scipy.stats import f_oneway
```

```
# Install Folium if not already installed
#!pip install folium

import pandas as pd
from sklearn.cluster import KMeans
import folium
from folium.plugins import MarkerCluster
from IPython.display import display

# --- Load your dataset ---
df = pd.read_csv("Final_Data.csv")  # Upload or mount your CSV

# --- Identify latitude & longitude columns ---
lat_col = [col for col in df.columns if 'lat' in col.lower()][0]
lon_col = [col for col in df.columns if 'lon' in col.lower()][0]

# --- Drop rows with missing coordinates ---
coords = df[[lat_col, lon_col]].dropna()

# --- Apply K-Means clustering ---
k = 5  # number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
coords['Cluster'] = kmeans.fit_predict(coords)

# --- Create Folium map ---
m = folium.Map(location=[coords[lat_col].mean(), coords[lon_col].mean()], zoom_start=11)
marker_cluster = MarkerCluster().add_to(m)

# Color palette
colors = ['red', 'blue', 'green', 'purple', 'orange',
          'darkred', 'lightred', 'beige', 'darkblue', 'darkgreen']

# --- Add crime points ---
for _, row in coords.iterrows():
    folium.CircleMarker(
        location=[row[lat_col], row[lon_col]],
        radius=3,
        color=colors[int(row['Cluster']) % len(colors)],
        fill=True,
        fill_color=colors[int(row['Cluster']) % len(colors)],
        fill_opacity=0.6
    ).add_to(marker_cluster)

# --- Add cluster centers ---
centers = kmeans.cluster_centers_
for idx, center in enumerate(centers):
    folium.Marker(
        location=[center[0], center[1]],
        popup=f"Cluster {idx} Center",
        icon=folium.Icon(color='black', icon='info-sign')
    ).add_to(m)

# --- Display map in Colab ---
display(m)
```



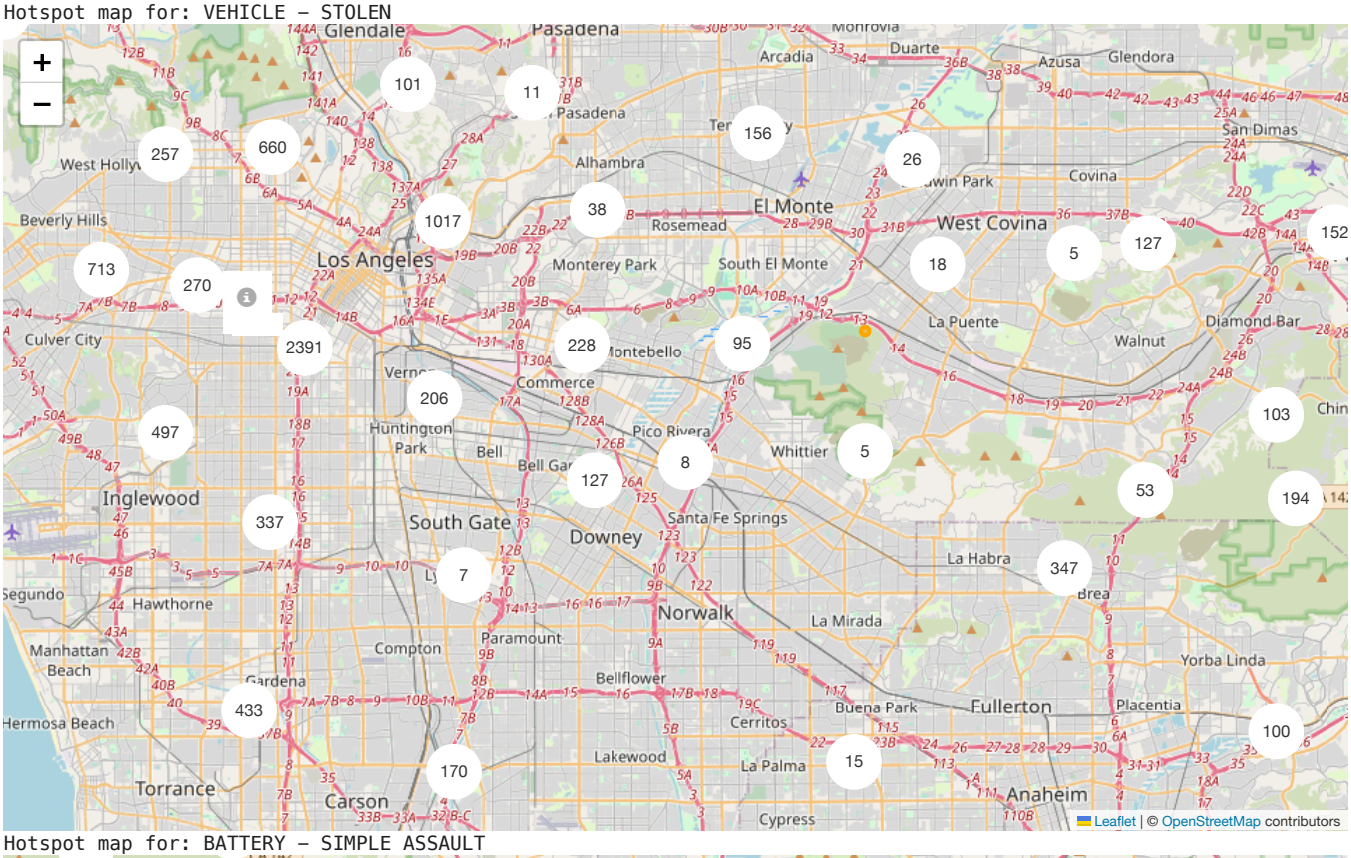
Inferences:

1. Crime incidents are highly concentrated in central Los Angeles, particularly around downtown, with several clusters exceeding 2,000+ incidents, indicating it's a primary hotspot for criminal activity.
2. Peripheral regions such as Inglewood, Torrance, and Pomona also show significant crime clusters, suggesting that high crime is not only a central LA issue but also affects certain suburban zones. Recommendations:
3. Focused Patrol Deployment: Allocate more law enforcement presence in central LA high-density clusters, especially during peak crime hours, to deter criminal activity.
4. Community Engagement & Surveillance: Install additional CCTV cameras and encourage community watch programs in hotspot areas like Inglewood and Torrance to enhance monitoring and early intervention.

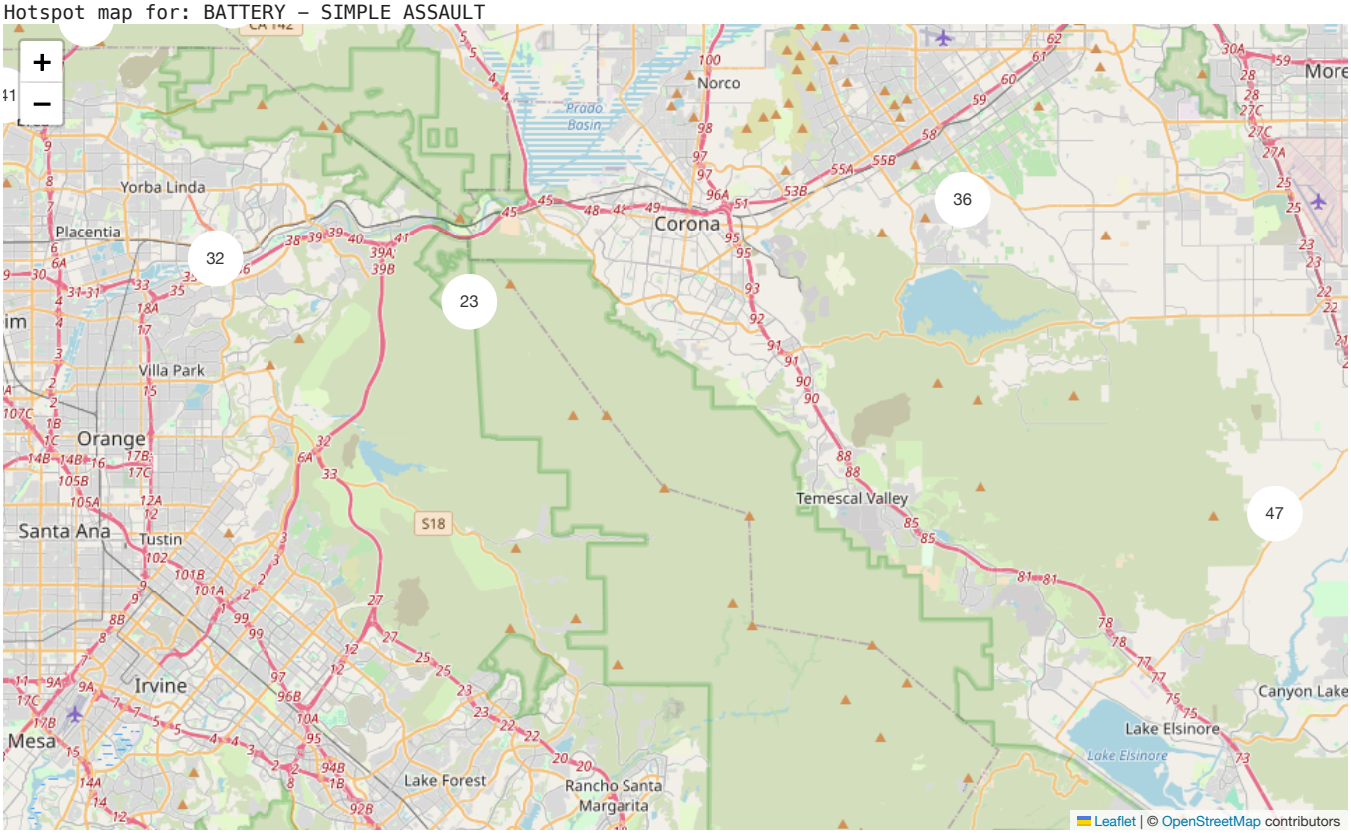
```
import pandas as pd
from sklearn.cluster import KMeans
import folium
from folium.plugins import MarkerCluster
from geopy.geocoders import Nominatim
from IPython.display import display

# Load dataset
df = pd.read_csv("Final_Data.csv")

# Identify latitude & longitude columns
lat_col = [col for col in df.columns if 'lat' in col.lower()][0]
lon_col = [col for col in df.columns if 'lon' in col.lower()][0]

# Set your column names
crime_type_col = "Crm Cd Desc"  # Change if different
victim_col = "Vict Descent"     # Change if different
time_col = "TIME OCC"           # Change if different

# Initialize geocoder
geolocator = Nominatim(user_agent="crime_hotspot_analysis")

# Function to get place name from coordinates
def get_place_name(lat, lon):
    try:
        location = geolocator.reverse((lat, lon), exactly_one=True, timeout=10)
        return location.address if location else "Unknown Location"
    except:
        return "Unknown Location"
```

```
# Get top 3 crimes by frequency
top_3_crimes = df[crime_type_col].value_counts().head(3).index.tolist()

# Loop over top 3 crime types
for crime in top_3_crimes:
    subset = df[df[crime_type_col] == crime][[lat_col, lon_col, victim_col, time_col]].dropna(subset=[lat_col, lon_col

    if len(subset) < 10:
        continue

    # K-Means clustering
    k = min(5, len(subset))
    kmeans = KMeans(n_clusters=k, random_state=42)
    subset['Cluster'] = kmeans.fit_predict(subset[[lat_col, lon_col]])

    # Create map
    m = folium.Map(location=[subset[lat_col].mean(), subset[lon_col].mean()], zoom_start=11)
    marker_cluster = MarkerCluster().add_to(m)

    colors = ['red', 'blue', 'green', 'purple', 'orange',
              'darkred', 'lightred', 'beige', 'darkblue', 'darkgreen']

    # Add crime points
    for _, row in subset.iterrows():
        folium.CircleMarker(
            location=[row[lat_col], row[lon_col]],
            radius=3,
            color=colors[int(row['Cluster']) % len(colors)],
            fill=True,
            fill_color=colors[int(row['Cluster']) % len(colors)],
            fill_opacity=0.6
        ).add_to(marker_cluster)

    # Add cluster centers with place name
    for idx, center in enumerate(kmeans.cluster_centers_):
        cluster_data = subset[subset['Cluster'] == idx]
        crime_count = len(cluster_data)
        most_common_victim = cluster_data[victim_col].mode()[0] if not cluster_data[victim_col].mode().empty else "N/A
        most_common_time = cluster_data[time_col].mode()[0] if not cluster_data[time_col].mode().empty else "N/A"
        place_name = get_place_name(center[0], center[1])

        popup_text = (
            f"<b>Crime Type:</b> {crime}<br>"
            f"<b>Cluster ID:</b> {idx}<br>"
            f"<b>Location:</b> {place_name}<br>"
            f"<b>Number of Crimes:</b> {crime_count}<br>"
            f"<b>Most Common Victim:</b> {most_common_victim}<br>"
            f"<b>Most Common Time:</b> {most_common_time}"
        )

        folium.Marker(
            location=[center[0], center[1]],
            popup=popup_text,
            icon=folium.Icon(color='black', icon='info-sign')
        ).add_to(m)

    print(f"Hotspot map for: {crime}")
    display(m)
```

Hotspot map for: VEHICLE – STOLEN



Hotspot map for: BATTERY – SIMPLE ASSAULT



Hotspot map for: VANDALISM – FELONY ($400 & OVER, ALL CHURCH VANDALISMS)

For VECHILE STOLEN- Inferences (Vehicle – Stolen Hotspots):

1. Vehicle theft incidents are highly concentrated in central Los Angeles, with one cluster exceeding 2,300+ cases, indicating it as a major hotspot.
2. Significant secondary clusters are visible in Inglewood, Torrance, and West LA, suggesting theft activity extends beyond downtown into residential and commercial zones. Recommendations:
3. Enhanced Surveillance & Patrols: Deploy targeted police patrols and install ANPR (Automatic Number Plate Recognition) cameras in major hotspots like central LA and Inglewood.
4. Public Awareness Campaigns: Educate residents on theft prevention (steering locks, alarms, secure parking) in vulnerable areas, especially during late-night hours.

For SIMPLE ASSAULT-

Inferences (Simple Assault Hotspots):

1. Higher concentrations of simple assault cases are observed near Brea, Riverside, and Norco, indicating localized aggression-prone zones.
2. Incidents are more dispersed compared to vehicle theft, suggesting assaults occur across smaller clusters rather than one central hotspot. Recommendations:
3. Increase police foot patrols and community mediation programs in Brea, Riverside, and Norco to address frequent altercations.
4. Implement conflict resolution workshops and public awareness campaigns to reduce interpersonal disputes in identified clusters.

```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load dataset
df = pd.read_csv("Final_Data.csv")

# Detect latitude/longitude columns automatically
lat_col = [c for c in df.columns if 'lat' in c.lower()][0]
lon_col = [c for c in df.columns if 'lon' in c.lower()][0]

# Select coordinates and scale them
coords = df[[lat_col, lon_col]].dropna()
scaler = StandardScaler()
coords_scaled = scaler.fit_transform(coords)

# Run K-Means clustering
k = 5  # choose number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
df.loc[coords.index, 'cluster'] = kmeans.fit_predict(coords_scaled)

# Identify column names for crime, victim, and time
crime_col = 'Crm_Cd_Desc' if 'Crm_Cd_Desc' in df.columns else None
victim_col = 'Vict Descent' if 'Vict Descent' in df.columns else None
time_col = 'TIME OCC' if 'TIME OCC' in df.columns else None

# Build cluster profiles
profiles = []
for c_id, grp in df.groupby('cluster'):
    profile = {
        'Cluster': c_id,
```

```
            'Crime Count': len(grp),
            'Top Crime Type': grp[crime_col].mode()[0] if crime_col and not grp[crime_col].mode().empty else None,
            'Top Victim Type': grp[victim_col].mode()[0] if victim_col and not grp[victim_col].mode().empty else None,
            'Peak Time': grp[time_col].mode()[0] if time_col and not grp[time_col].mode().empty else None
        }
        profiles.append(profile)

    # Convert to DataFrame and sort by crime count
    profiles_df = pd.DataFrame(profiles).sort_values('Crime Count', ascending=False)

    print("\nCluster Profiles:")
    print(profiles_df)
```

```
Cluster Profiles:
   Cluster  Crime Count Top Crime Type Top Victim Type  Peak Time
0      0.0        45326           None               H     1500.0
4      4.0        24449           None               H     1500.0
1      1.0          149           None               H     1500.0
2      2.0           39           None               H     1215.0
3      3.0           37           None               H     1515.0
```

```
    crime_col = 'Crm Cd Desc'  # exact match from your CSV
    victim_col = 'Vict Descent'

    profiles = []
    for c_id, grp in df.groupby('spatio_temp_cluster'):
        profile = {
            'Cluster': c_id,
            'Crime Count': len(grp),
            'Top Crime Type': grp[crime_col].mode()[0] if crime_col in df.columns and not grp[crime_col].mode().empty else
            'Top Victim Type': grp[victim_col].mode()[0] if victim_col in df.columns and not grp[victim_col].mode().empty
            'Peak Time': grp['hour'].mode()[0] if not grp['hour'].mode().empty else None
        }
        profiles.append(profile)

    profiles_df = pd.DataFrame(profiles).sort_values('Crime Count', ascending=False)
    print(profiles_df)
```

```
   Cluster  Crime Count            Top Crime Type Top Victim Type  Peak Time
1      1.0         9782          VEHICLE - STOLEN               H       15.0
0      0.0         8542          VEHICLE - STOLEN               H       14.0
2      2.0         8522          VEHICLE - STOLEN               H       12.0
4      4.0         7872          VEHICLE - STOLEN               H       14.0
5      5.0         7601          VEHICLE - STOLEN               H       14.0
3      3.0          119  BATTERY - SIMPLE ASSAULT               H       13.0
```

```
    from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_harabasz_score
    print("Silhouette:", silhouette_score(coords_scaled, df['cluster']))
    print("DB Index:", davies_bouldin_score(coords_scaled, df['cluster']))
    print("CH Index:", calinski_harabasz_score(coords_scaled, df['cluster']))
```

```
Silhouette: 0.44192997718012517
DB Index: 0.4112758640750972
CH Index: 5633157.696434791
```

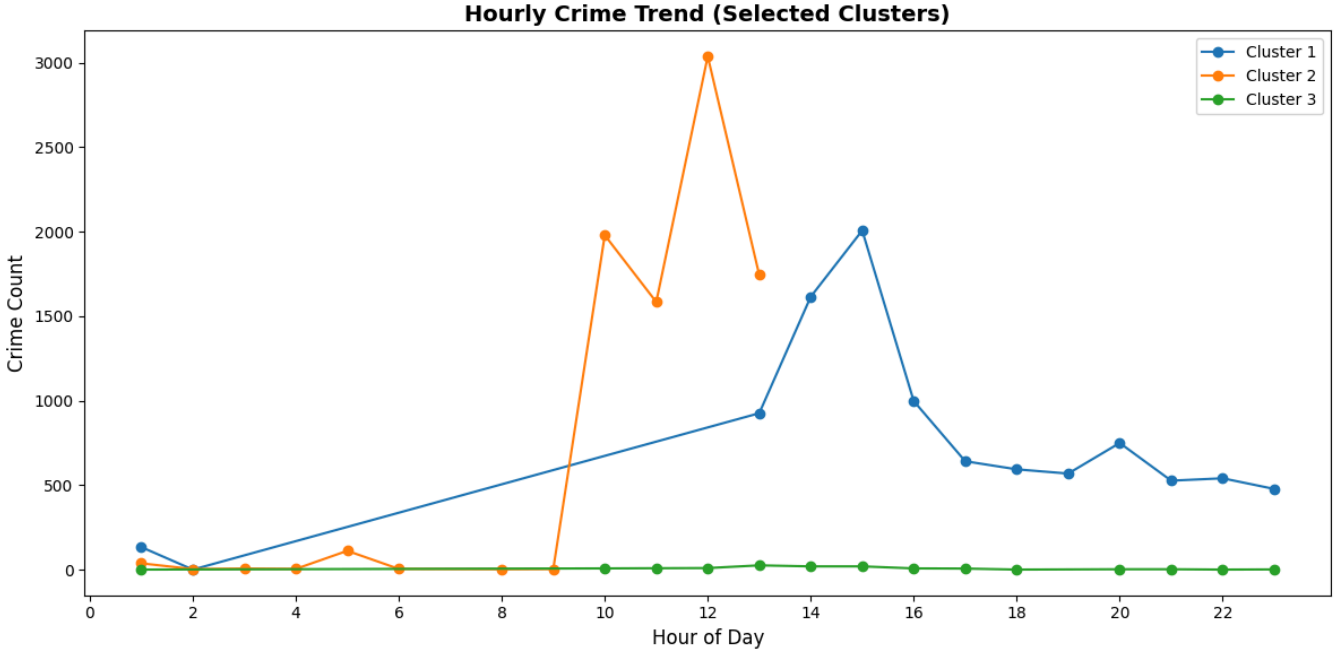
```
    # Ensure TIME OCC is string and 4 digits (HHMM)
    df[time_col] = df[time_col].astype(str).str.zfill(4)

    # Extract hour safely
    df['hour'] = pd.to_numeric(df[time_col].str[:2], errors='coerce')

    # Keep only valid hours (0–23)
    df.loc[~df['hour'].between(0, 23), 'hour'] = np.nan
```

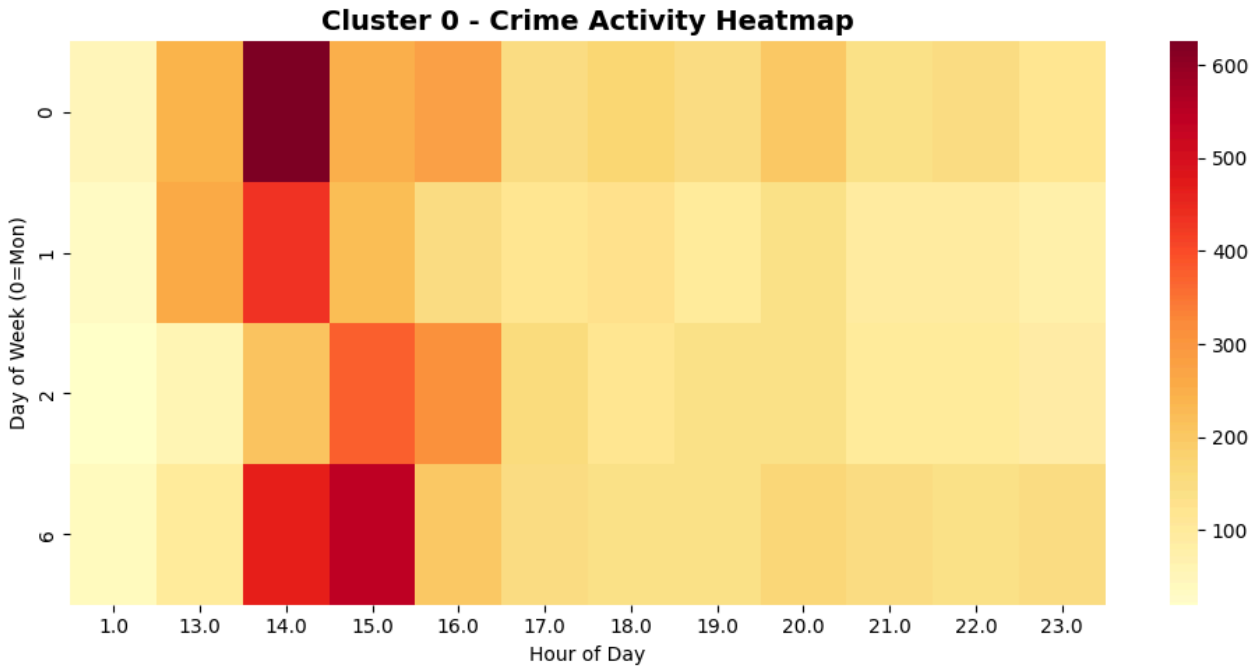
```
    import pandas as pd
    import numpy as np
    from sklearn.preprocessing import StandardScaler
    from sklearn.cluster import KMeans
    import matplotlib.pyplot as plt
    import seaborn as sns

    # ------------------ Load Data ------------------
    df = pd.read_csv("Final_Data.csv")
```

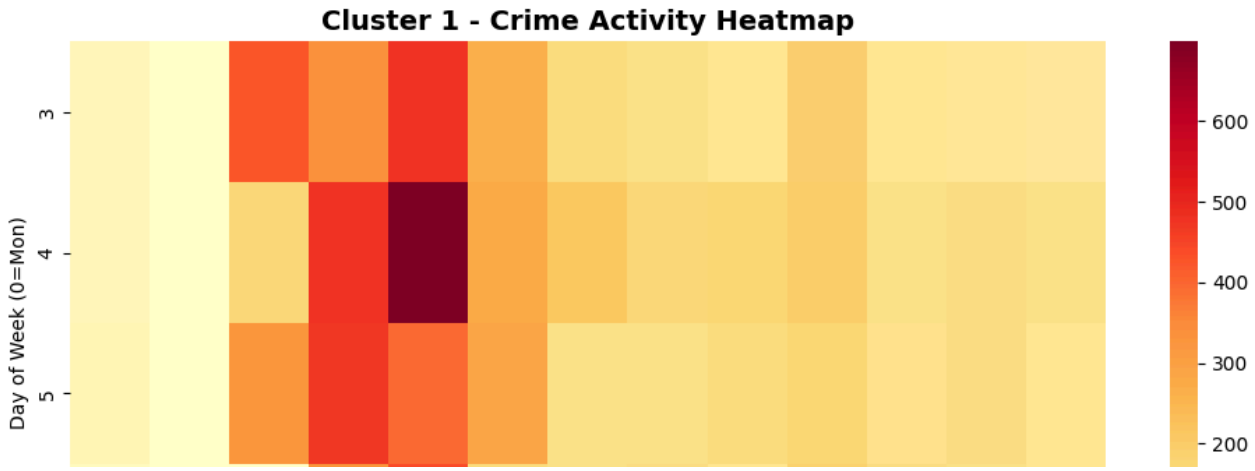
```
# Detect latitude/longitude
lat_col = [c for c in df.columns if 'lat' in c.lower()][0]
lon_col = [c for c in df.columns if 'lon' in c.lower()][0]

# Date & Time column names
date_col = 'DATE OCC' if 'DATE OCC' in df.columns else None
time_col = 'TIME OCC' if 'TIME OCC' in df.columns else None

# Convert date
if date_col:
    df[date_col] = pd.to_datetime(df[date_col], errors='coerce')

# --- FIX TIME OCC ---
if time_col:
    df[time_col] = df[time_col].astype(str).str.zfill(4)  # Ensure 4-digit HHMM
    df['hour'] = pd.to_numeric(df[time_col].str[:2], errors='coerce')  # Extract HH
    df.loc[~df['hour'].between(0, 23), 'hour'] = np.nan
else:
    raise ValueError("TIME OCC column not found")

# Extract day & month
df['day_of_week'] = df[date_col].dt.dayofweek
df['month'] = df[date_col].dt.month

# Cyclical encoding
df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)
df['dow_sin'] = np.sin(2 * np.pi * df['day_of_week'] / 7)
df['dow_cos'] = np.cos(2 * np.pi * df['day_of_week'] / 7)
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)

# ------------------ Spatio-Temporal Clustering ------------------
features = df[[lat_col, lon_col,
               'hour_sin', 'hour_cos',
               'dow_sin', 'dow_cos',
               'month_sin', 'month_cos']].dropna()

scaler = StandardScaler()
features_scaled = scaler.fit_transform(features)

k = 6
kmeans = KMeans(n_clusters=k, random_state=42)
df.loc[features.index, 'spatio_temp_cluster'] = kmeans.fit_predict(features_scaled)

# ------------------ Cluster Profiles ------------------
crime_col = 'Crm Cd Desc' if 'Crm Cd Desc' in df.columns else None
victim_col = 'Vict Descent' if 'Vict Descent' in df.columns else None

profiles = []
for c_id, grp in df.groupby('spatio_temp_cluster'):
    profile = {
        'Cluster': c_id,
        'Crime Count': len(grp),
        'Top Crime Type': grp[crime_col].mode()[0] if crime_col and not grp[crime_col].mode().empty else None,
        'Top Victim Type': grp[victim_col].mode()[0] if victim_col and not grp[victim_col].mode().empty else None,
        'Peak Hour': grp['hour'].mode()[0] if not grp['hour'].mode().empty else None
    }
    profiles.append(profile)

profiles_df = pd.DataFrame(profiles).sort_values('Crime Count', ascending=False)

print("\nCluster Profiles:")
print(profiles_df)

# ------------------ Hourly Trend Plot (exclude clusters 0, 4, 5) ------------------
exclude_clusters = {0, 4, 5}

plt.figure(figsize=(12,6))
for cluster_id in sorted(df['spatio_temp_cluster'].dropna().unique()):
    if int(cluster_id) in exclude_clusters:
        continue
    cluster_data = df[df['spatio_temp_cluster'] == cluster_id]
    hourly_counts = cluster_data['hour'].value_counts().sort_index()
    plt.plot(hourly_counts.index, hourly_counts.values, marker='o', label=f"Cluster {int(cluster_id)}")

plt.title("Hourly Crime Trend (Selected Clusters)", fontsize=14, fontweight='bold')
plt.xlabel("Hour of Day", fontsize=12)
```

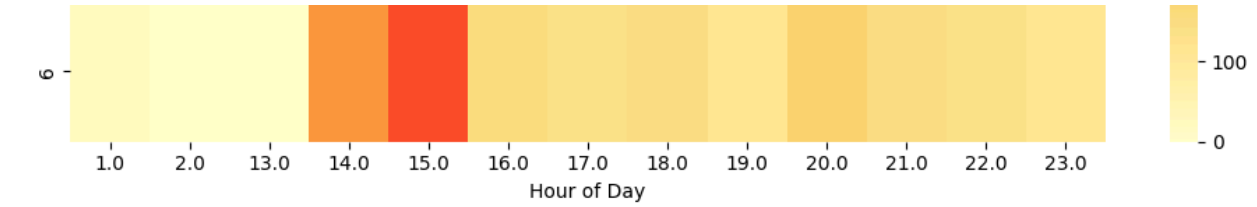
```
    plt.ylabel("Crime Count", fontsize=12)
    plt.xticks(ticks=range(0, 24, 2))
    plt.legend()
    plt.tight_layout()
    plt.show()

    # ------------------ Heatmaps ------------------
    for cluster_id in sorted(df['spatio_temp_cluster'].dropna().unique()):
        cluster_data = df[df['spatio_temp_cluster'] == cluster_id]
        pivot_table = cluster_data.pivot_table(index='day_of_week', columns='hour',
                                               values=lat_col, aggfunc='count').fillna(0)
        plt.figure(figsize=(10,5))
        sns.heatmap(pivot_table, cmap='YlOrRd')
        plt.title(f"Cluster {int(cluster_id)} - Crime Activity Heatmap", fontsize=14, fontweight='bold')
        plt.xlabel("Hour of Day")
        plt.ylabel("Day of Week (0=Mon)")
        plt.tight_layout()
        plt.show()
```

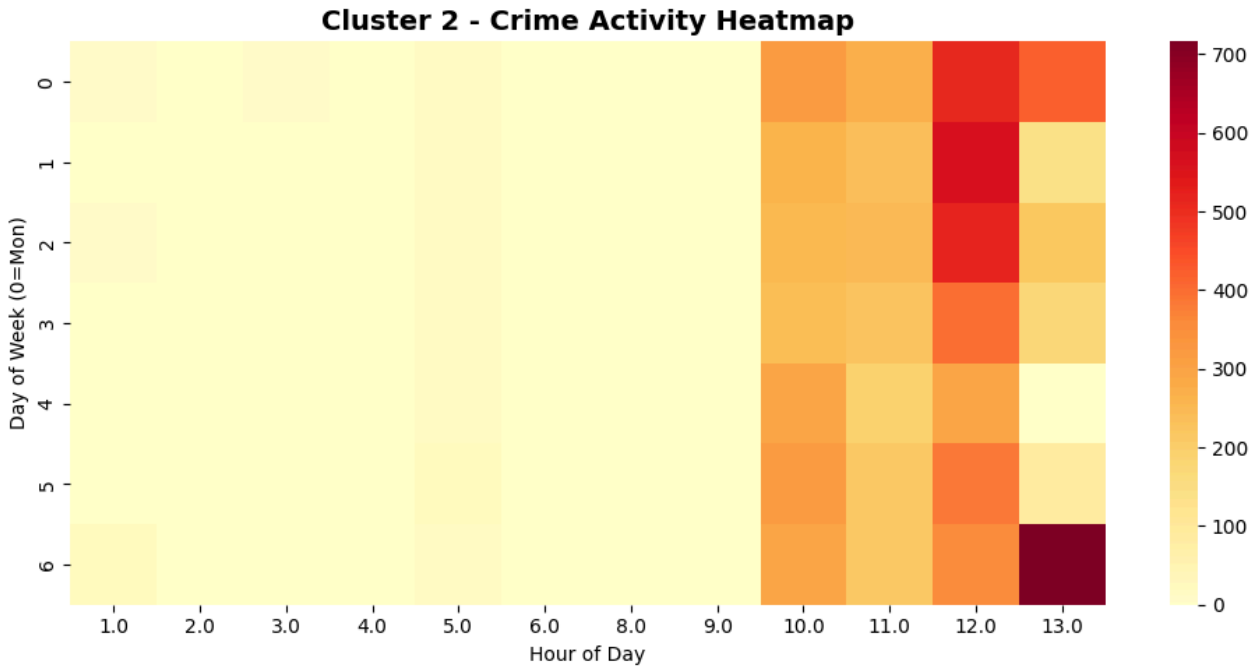
```
/tmp/ipython-input-2405335716.py:21: UserWarning: Could not infer format, so each element will be parsed individual
  df[date_col] = pd.to_datetime(df[date_col], errors='coerce')

Cluster Profiles:
   Cluster  Crime Count            Top Crime Type Top Victim Type  Peak Hour
1      1.0         9782          VEHICLE - STOLEN                H       15.0
0      0.0         8542          VEHICLE - STOLEN                H       14.0
2      2.0         8522          VEHICLE - STOLEN                H       12.0
4      4.0         7872          VEHICLE - STOLEN                H       14.0
5      5.0         7601          VEHICLE - STOLEN                H       14.0
3      3.0          119  BATTERY - SIMPLE ASSAULT                H       13.0
```

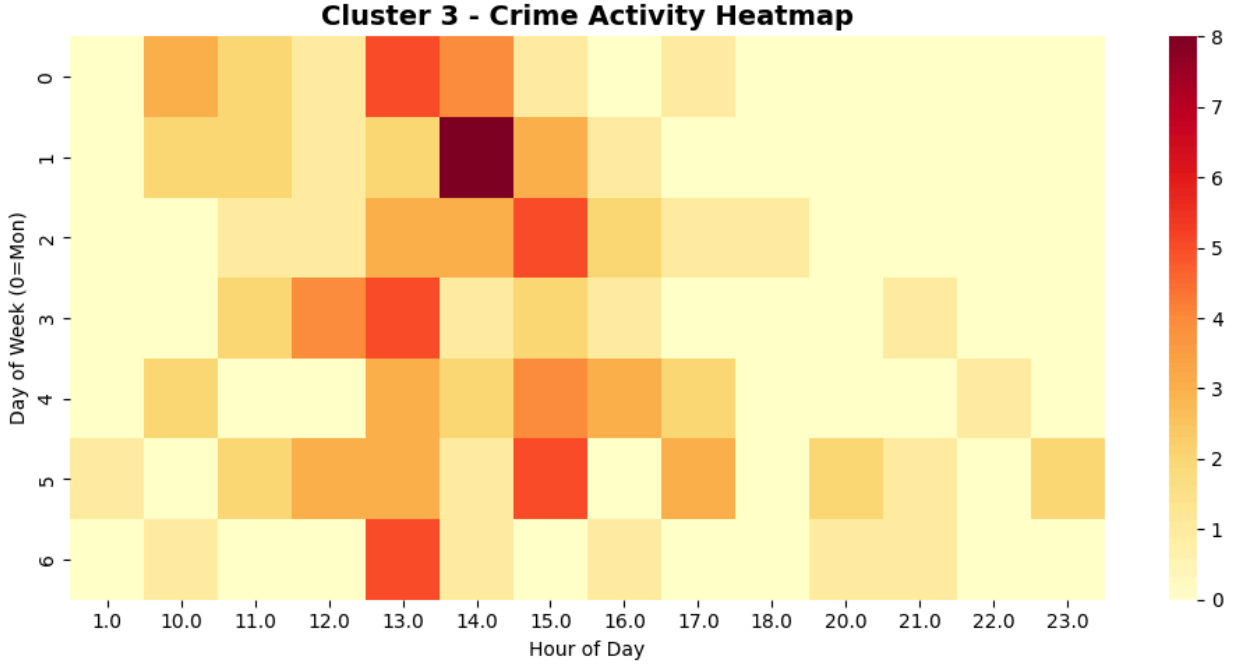


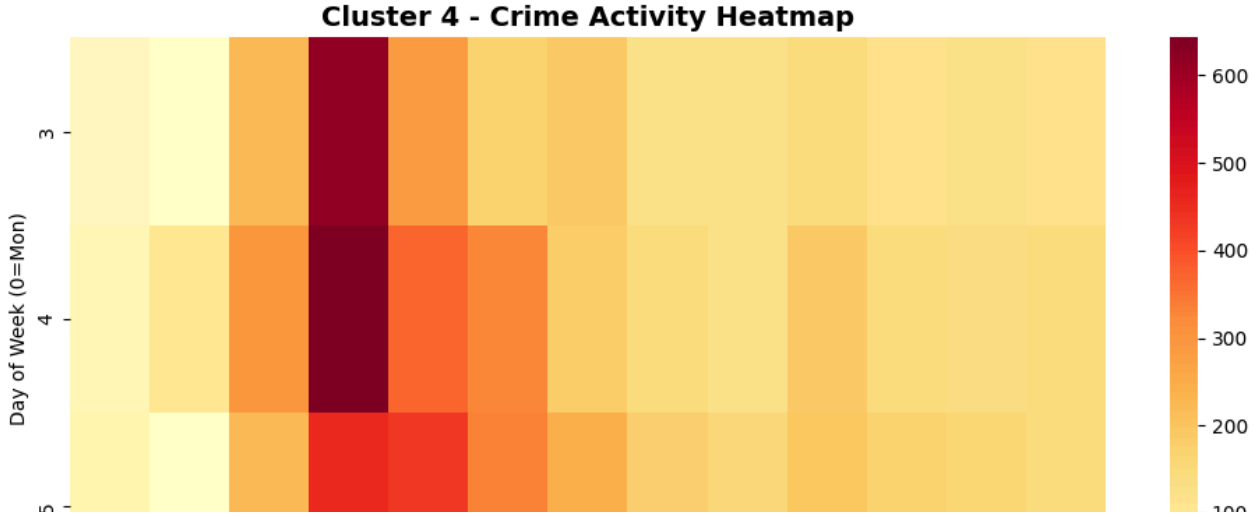


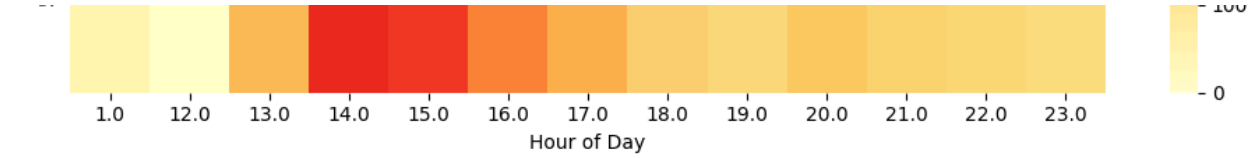


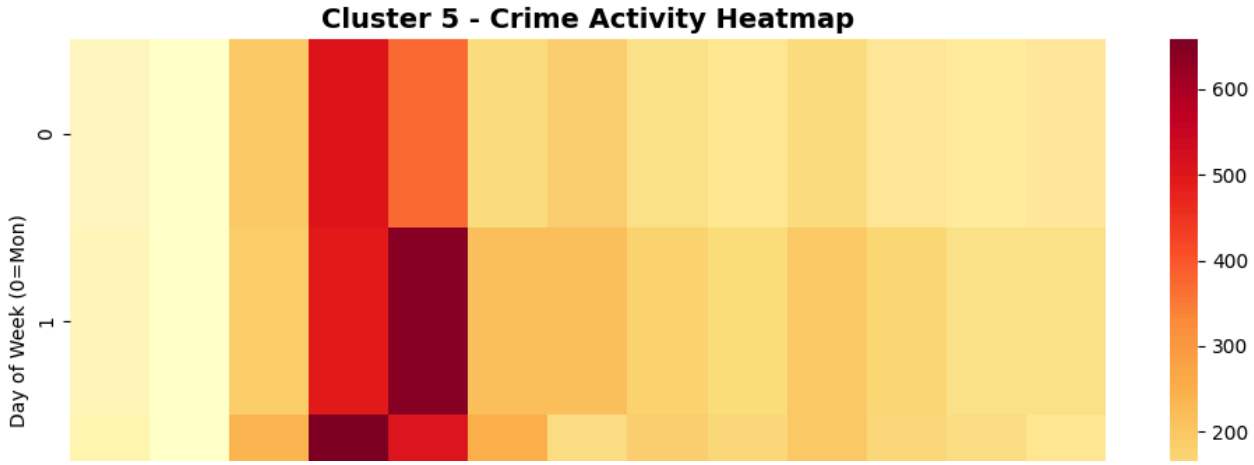

6
1.0
2.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
Hour of Day
100
0


Cluster 2 - Crime Activity Heatmap
Day of Week (0=Mon)
0
1
2
3
4
5
6
1.0
2.0
3.0
4.0
5.0
6.0
8.0
9.0
10.0
11.0
12.0
13.0
Hour of Day
700
600
500
400
300
200
100
0


Cluster 3 - Crime Activity Heatmap
Day of Week (0=Mon)
0
1
2
3
4
5
6
1.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
20.0
21.0
22.0
23.0
Hour of Day
8
7
6
5
4
3
2
1
0


Cluster 4 - Crime Activity Heatmap
Day of Week (0=Mon)
3
4
5
600
500
400
300
200


100
0
1.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
Hour of Day


Cluster 5 - Crime Activity Heatmap
Day of Week (0=Mon)
0
1
600
500
400
300
200

1st-Image

Inferences:

1.Cluster 2 shows a sharp midday spike around 12 PM, indicating a high concentration of crimes during late morning to noon hours. 2.Cluster 1 has a more gradual rise, peaking in the afternoon (~3 PM), while Cluster 3 remains consistently low across all hours.

Recommendations:

1.Deploy targeted midday patrols in Cluster 2 hotspots to deter high-volume crimes during peak hours. 2.Increase afternoon surveillance in Cluster 1 areas, particularly between 1–4 PM, to address sustained crime activity.

2nd -Image Inferences:

1.2.Crime activity in Cluster 0 peaks on Mondays around 2–3 PM and again on Sundays around 2 PM, indicating specific weekday and weekend spikes. Late afternoon to evening hours show relatively moderate but consistent activity across the week.

Recommendations: 1.Increase patrol presence in Cluster 0 zones during early afternoon hours on Mondays and Sundays. 2.Implement preventive policing strategies (e.g., community engagement, CCTV monitoring) in the hours leading to peak times to deter crime.

3rd IMAGE- Inferences:

1.Cluster 1 crime activity peaks sharply on Thursdays around 3 PM, with elevated incidents also on Wednesdays and Fridays in early afternoons. 2.Crime intensity drops significantly during evening and early morning hours across the week.

Recommendations: 1.Strengthen midday patrols in hotspot areas on Thursdays, focusing on the 2–4 PM window. 2.Use predictive policing tools to anticipate and prevent afternoon surges mid-to-late week.

4th IMAGE- Inferences:

1.Cluster 2 crime activity peaks between 11 AM and 1 PM, with the highest spike on Sundays around 1 PM. 2.Activity is minimal during early morning hours, indicating daytime-driven incidents.

Recommendations: 1.Deploy daytime patrols in hotspot zones, especially late mornings on weekends. 2.Increase surveillance and community presence during Sunday midday hours to deter high-activity spikes.

5th Image- Inferences:

1.Crime activity in Cluster 3 peaks around early afternoon (13:00–14:00), especially on Tuesdays (day 1), indicating a consistent high-risk time slot. 2.Monday mornings and mid-week afternoons also show moderate activity, with evenings and late nights generally low.

Recommendations:

1.Increase police patrols and surveillance between 12:00–15:00 on Tuesdays, with some coverage on Monday mornings. 2.Install real-time monitoring systems (CCTV, AI analytics) in high-incident areas to proactively detect and deter crime during peak hours.

6th Image-

Inferences:

1.Crime incidents in Cluster 4 are highly concentrated between 14:00–15:00, especially on Thursdays (day 3) and Fridays (day 4). 2.Activity remains moderately high in the late afternoon until around 18:00, with relatively low occurrences in early mornings and late nights.

Recommendations:

1.Deploy maximum patrol and community policing efforts from 13:00–16:00 on Thursdays and Fridays. 2.Implement targeted deterrence measures (public awareness, CCTV, quick-response units) in hotspot areas during the afternoon peak.

7th Image-

Inferences:

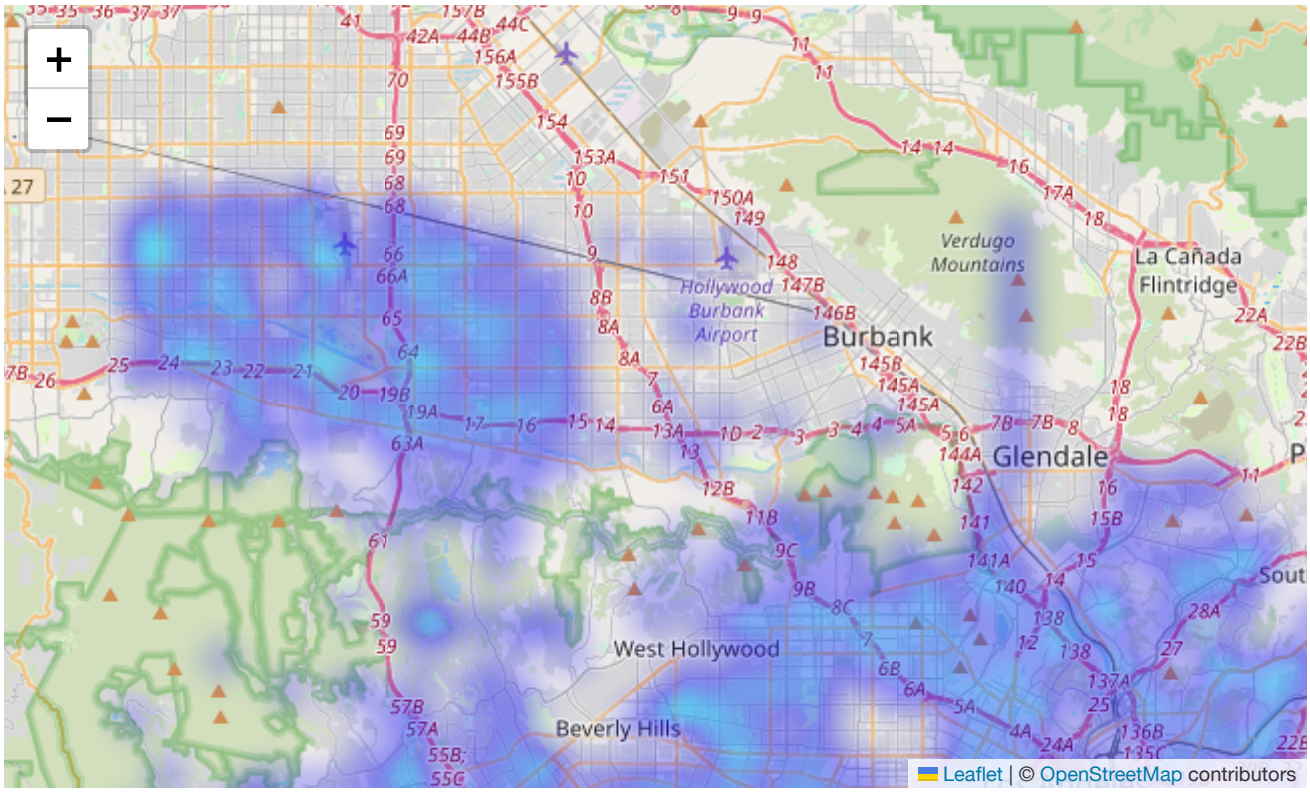
1.Cluster 5 shows a strong crime peak between 14:00–15:00, especially on Wednesdays (day 2) and moderately high on Mondays (day 0) and Tuesdays (day 1). 2.Crime intensity drops significantly after 16:00 and is minimal during early mornings and late nights.
Recommendations:

1.Concentrate policing, CCTV surveillance, and rapid response units between 13:00–16:00 on Monday to Wednesday.

2.Launch targeted community safety programs in hotspot zones to deter mid-afternoon criminal activity.

```
# !pip install folium

import pandas as pd
import folium
from folium.plugins import HeatMap

# --- Load dataset ---
df = pd.read_csv("Final_Data.csv")
```

```
# --- Identify required columns ---
lat_col = [col for col in df.columns if 'lat' in col.lower()][0]
lon_col = [col for col in df.columns if 'lon' in col.lower()][0]
severity_col = [col for col in df.columns if 'part' in col.lower()][0]  # "Part 1

# --- Drop missing values ---
df = df.dropna(subset=[lat_col, lon_col, severity_col])

# --- Convert severity to numeric weight ---
# Assuming Part 1 = more severe (weight 2), Part 2 = less severe (weight 1)
df['severity_weight'] = df[severity_col].apply(lambda x: 2 if x == 1 else 1)

# --- Prepare weighted data for HeatMap ---
heat_data = df[[lat_col, lon_col, 'severity_weight']].values.tolist()

# --- Create base map ---
m = folium.Map(location=[df[lat_col].mean(), df[lon_col].mean()], zoom_start=11)

# --- Add severity heatmap ---
HeatMap(
    heat_data,
    radius=10,
    blur=15,
    max_zoom=1,
    min_opacity=0.3
).add_to(m)

# --- Display map ---
m
```

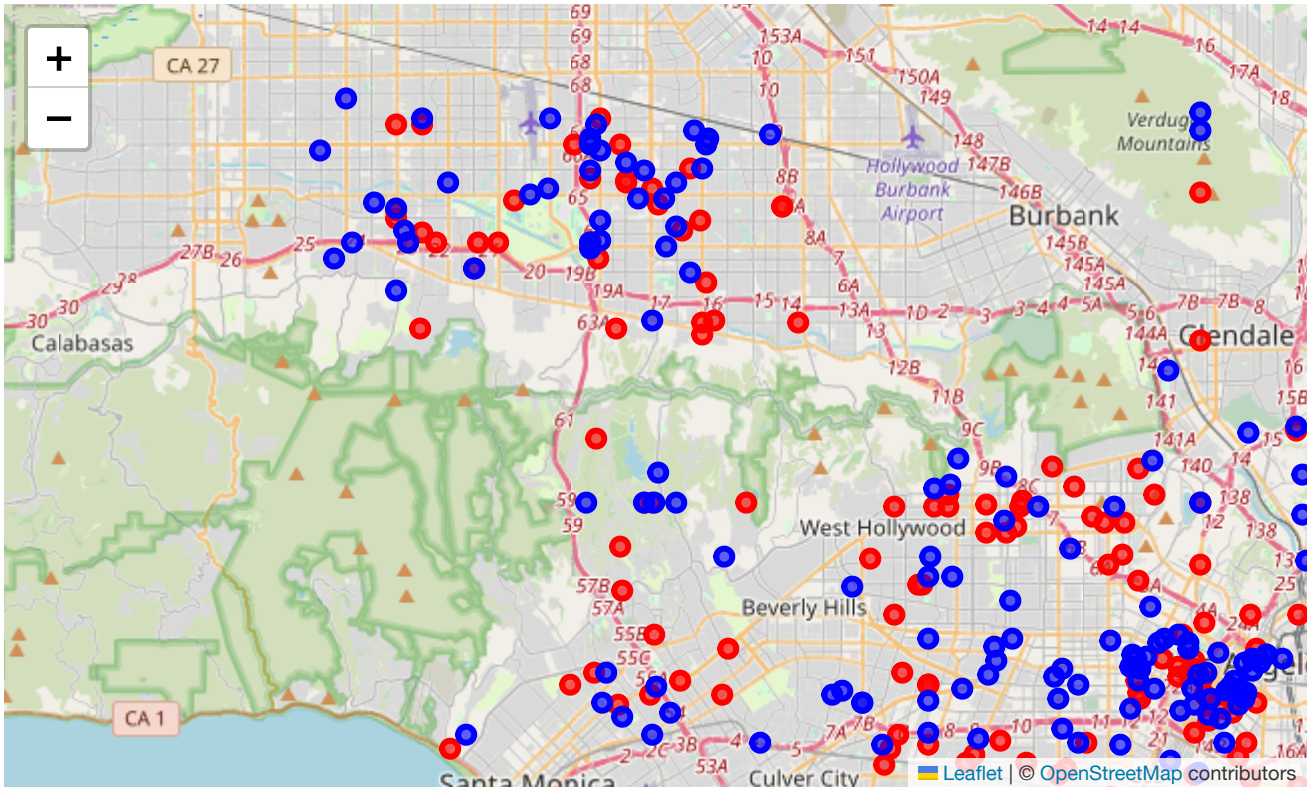


Inferences:

1. Major crime hotspots are concentrated in central Los Angeles, particularly around Downtown, Inglewood, and West Hollywood, with moderate zones extending toward Glendale and Pasadena.
2. Minor crime areas are scattered in the outskirts such as Santa Monica, Beverly Hills, and parts of South Gate, showing lower but notable activity. Recommendations:
3. Deploy more intensive law enforcement presence and surveillance in central high-density hotspots, especially during identified peak hours.
4. Maintain community policing and preventive measures in minor crime areas to prevent escalation and keep crime rates low.

```
import pandas as pd
import folium
```

```
# --- Load dataset ---
df = pd.read_csv("Final_Data.csv")

# --- Identify columns ---
lat_col = [col for col in df.columns if 'lat' in col.lower()][0]
lon_col = [col for col in df.columns if 'lon' in col.lower()][0]
severity_col = [col for col in df.columns if 'part' in col.lower()][0]

# --- Drop missing values ---
df = df.dropna(subset=[lat_col, lon_col, severity_col])

# --- Filter datasets ---
severe_df = df[df[severity_col] == 1]  # Severe crimes
minor_df = df[df[severity_col] == 2]   # Minor crimes

# --- Sample to reduce congestion ---
severe_df = severe_df.sample(n=300, random_state=42) if len(severe_df) > 300 else
minor_df = minor_df.sample(n=300, random_state=42) if len(minor_df) > 300 else mi

# --- Create base map ---
m = folium.Map(location=[df[lat_col].mean(), df[lon_col].mean()], zoom_start=11)

# --- Add severe crimes in red ---
for _, row in severe_df.iterrows():
    folium.CircleMarker(
        location=[row[lat_col], row[lon_col]],
        radius=4,
        color='red',
        fill=True,
        fill_color='red',
        fill_opacity=0.6,
        popup="Severe Crime"
    ).add_to(m)

# --- Add minor crimes in blue ---
for _, row in minor_df.iterrows():
    folium.CircleMarker(
        location=[row[lat_col], row[lon_col]],
        radius=4,
        color='blue',
        fill=True,
        fill_color='blue',
        fill_opacity=0.6,
        popup="Minor Crime"
    ).add_to(m)

# --- Display map ---
m
```



Inferences:

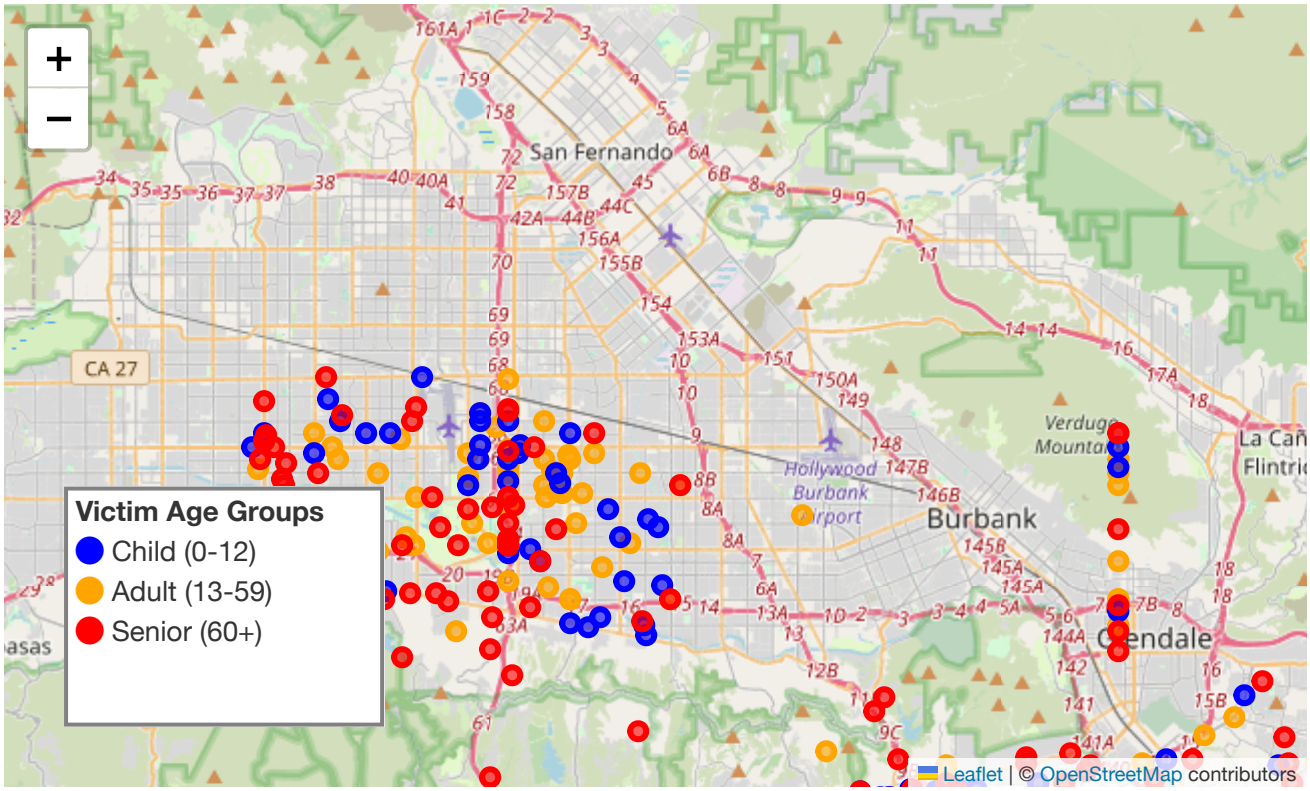
1. Severe crimes (red) are densely concentrated in central Los Angeles, with notable clusters extending toward East LA and parts of South LA.
2. Less severe crimes (blue) are more evenly distributed but still show higher density in downtown and surrounding neighborhoods. Recommendations:
3. Allocate more police resources and rapid-response units to severe crime hotspots, especially in central and east LA.
4. Use preventive policing strategies and community programs in mixed-density areas to reduce both severe and minor crimes simultaneously.

```
# !pip install folium

import pandas as pd
import folium
```

```
from branca.element import Template, MacroElement

# --- Load dataset ---
df = pd.read_csv("Final_Data.csv")

# --- Identify columns ---
lat_col = [col for col in df.columns if 'lat' in col.lower()][0]
lon_col = [col for col in df.columns if 'lon' in col.lower()][0]
age_col = [col for col in df.columns if 'age' in col.lower()][0]  # Vict Age

# --- Drop missing or invalid ages ---
df = df.dropna(subset=[lat_col, lon_col, age_col])
df = df[(df[age_col] >= 0) & (df[age_col] <= 120)]  # realistic ages only

# --- Classify into age groups (excluding Teen) ---
def classify_age(age):
    if age <= 12:
        return "Child (0-12)"
    elif age <= 59:
        return "Adult (13-59)"  # merged teens into adults
    else:
        return "Senior (60+)"

df["Age_Group"] = df[age_col].apply(classify_age)

# --- Color mapping ---
color_map = {
    "Child (0-12)": "blue",
    "Adult (13-59)": "orange",
    "Senior (60+)": "red"
}

# --- Reduce number of points (sampling for each group) ---
sampled_df = df.groupby("Age_Group").apply(lambda x: x.sample(n=min(500, len(x)),

# --- Create map ---
m = folium.Map(location=[df[lat_col].mean(), df[lon_col].mean()], zoom_start=11)

# --- Add points with color ---
for _, row in sampled_df.iterrows():
    folium.CircleMarker(
        location=[row[lat_col], row[lon_col]],
        radius=4,
        color=color_map[row["Age_Group"]],
        fill=True,
        fill_color=color_map[row["Age_Group"]],
        fill_opacity=0.6,
        popup=f"Age: {row[age_col]}, Group: {row['Age_Group']}"
    ).add_to(m)

# --- Add Legend ---
legend_html = """
{% macro html(this, kwargs) %}
<div style="
    position: fixed;
```

```
    bottom: 30px; left: 30px; width: 160px; height: 120px;
    background-color: white;
    border:2px solid grey; z-index:9999; font-size:14px;
    ">
     <b>Victim Age Groups</b><br>
     <i class="fa fa-circle" style="color:blue"></i> Child (0-12)<br>
     <i class="fa fa-circle" style="color:orange"></i> Adult (13-59)<br
     <i class="fa fa-circle" style="color:red"></i> Senior (60+)
</div>
{% endmacro %}
"""
legend = MacroElement()
legend._template = Template(legend_html)
m.get_root().add_child(legend)

# --- Display map ---
m
```

```
/tmp/ipython-input-3686694838.py:38: DeprecationWarning: DataFrameGroupBy.app
  sampled_df = df.groupby("Age_Group").apply(lambda x: x.sample(n=min(500, le
```



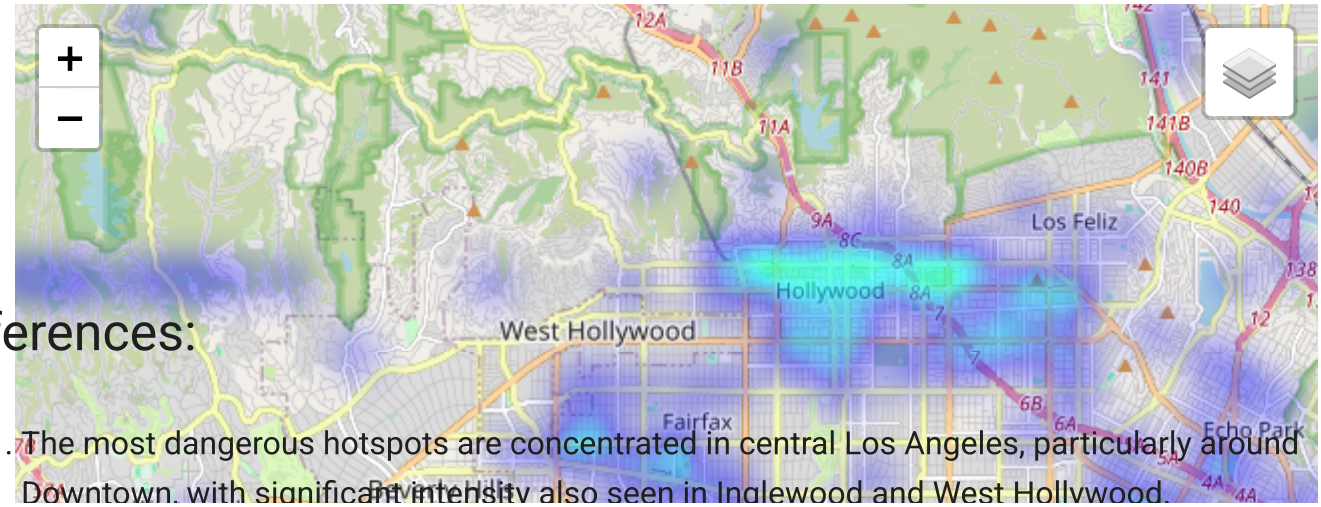
Inferences:

1. Seniors (red) are disproportionately targeted in central Los Angeles and surrounding areas, with notable clusters also in eastern neighborhoods.
2. Adults (orange) and children (blue) are affected across the region, but adult victim cases are more dispersed, while child cases are relatively fewer and scattered.
   Recommendations:
3. Strengthen safety measures for seniors in central and eastern LA through neighborhood watch programs, targeted patrols, and awareness campaigns.
4. Implement child safety education and protective infrastructure in scattered high-risk areas to reduce vulnerability among minors.)

```
# !pip install folium

import pandas as pd
import folium
from folium.plugins import HeatMap

# --- Load dataset ---
df = pd.read_csv("Final_Data.csv")

# --- Identify columns ---
lat_col = [col for col in df.columns if 'lat' in col.lower()][0]
lon_col = [col for col in df.columns if 'lon' in col.lower()][0]
date_col = [col for col in df.columns if 'date' in col.lower()][0]

# --- Convert to datetime ---
df[date_col] = pd.to_datetime(df[date_col], errors='coerce')

# --- Drop missing values ---
df = df.dropna(subset=[lat_col, lon_col, date_col])

# --- Extract hour ---
df["Hour"] = df[date_col].dt.hour

# --- Define time slots ---
def get_time_slot(hour):
    if 5 <= hour < 12:
        return "Morning (5–11)"
    elif 12 <= hour < 17:
        return "Afternoon (12–16)"
    elif 17 <= hour < 21:
        return "Evening (17–20)"
    else:
        return "Night (21–4)"

df["Time_Slot"] = df["Hour"].apply(get_time_slot)

# --- Identify extremely dangerous locations (top 5% by crime count) ---
```

```
location_counts = df.groupby([lat_col, lon_col]).size().reset_index(name="Crime_C
threshold = location_counts["Crime_Count"].quantile(0.95)  # top 5%
dangerous_locs = location_counts[location_counts["Crime_Count"] >= threshold]

# --- Merge back to get only dangerous crimes ---
df = df.merge(dangerous_locs[[lat_col, lon_col]], on=[lat_col, lon_col], how="inn

# --- Create base map ---
m = folium.Map(location=[df[lat_col].mean(), df[lon_col].mean()], zoom_start=13)

# --- Add heatmap layer for each time slot ---
for slot in df["Time_Slot"].unique():
    slot_df = df[df["Time_Slot"] == slot]
    heat_data = slot_df[[lat_col, lon_col]].values.tolist()
    HeatMap(
        heat_data,
        radius=12,
        blur=18,
        min_opacity=0.4
    ).add_to(folium.FeatureGroup(name=f"{slot} - Dangerous Hotspots").add_to(m))

# --- Add layer control ---
folium.LayerControl().add_to(m)

# --- Show map ---
m
```

```
/tmp/ipython-input-2020041788.py:16: UserWarning: Could not infer format, so e
  df[date_col] = pd.to_datetime(df[date_col], errors='coerce')
```



Inferences:

1. The most dangerous hotspots are concentrated in central Los Angeles, particularly around Downtown, with significant intensity also seen in Inglewood and West Hollywood.

Inferences:

1.The most dangerous hotspots are concentrated in central Los Angeles, particularly around Downtown, with significant intensity also seen in Inglewood and West Hollywood.

2.Smaller but notable danger zones extend toward Pasadena, Glendale, and parts of Santa Monica, indicating a spread into surrounding urban areas.

Recommendations:

1.Prioritize increased patrols, undercover operations, and CCTV coverage in central LA hotspots to disrupt high-risk criminal activity.

2.Implement neighborhood safety initiatives and rapid response networks in secondary hotspots (Pasadena, Glendale, Santa Monica) to prevent escalation.

Start coding or generate with AI.