



BUSINESS UNDERSTANDING Provided with the crime dataset the analysis aims to examine the different crime rates in different cities and states. The data present is both categorical and numerical describing the different types of crime ie violentcrime, rape, muder,robbery,Aggravatedassault,propertycrime,burglary,theft and motorvehicletheft

DATA LOADING

```
#loading dataset
import pandas as pd
df=pd.read_csv("/content/crime.csv")
df
```




	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burglary	
0	412.5	5.3	56.0	78.4	272.8	3,609.00	852	2,4
1	238.4	5.1	38.2	75.2	119.8	2,552.40	575.3	1,8
2	667.9	7.8	30.4	157.9	471.8	3,894.10	1,099.60	2,6
3	114.3	2.5	28.2	20.7	63.0	3,208.40	484.6	2,4
4	792.6	6.1	63.8	206.7	516.0	4,607.80	883.4	3,0
...
373	251.6	11.4	6.3	74.6	159.3	823.2	265.5	
374	237.5	11.5	5.2	82.3	138.6	1,320.00	377.1	



i have successfully loaded the file


```
#head
df.head()
```



	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burglary	Theft
0	412.5	5.3	56.0	78.4	272.8	3,609.00	852	2,493
1	238.4	5.1	38.2	75.2	119.8	2,552.40	575.3	1,853
2	667.9	7.8	30.4	157.9	471.8	3,894.10	1,099.60	2,652
3	114.3	2.5	28.2	20.7	63.0	3,208.40	484.6	2,476
4	792.6	6.1	63.8	206.7	516.0	4,607.80	883.4	3,047

shows first five rows there are no missing values in the head


```
#tail
df.tail()
```



	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burglary	Theft
373	251.6	11.4	6.3	74.6	159.3	823.2	265.5	53
374	237.5	11.5	5.2	82.3	138.6	1,320.00	377.1	86

shows last five rows. There are no missing values in the tail.

```
#information
df.info()
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 378 entries, 0 to 377
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   ViolentCrime          377 non-null    object
1   Murder                378 non-null    float64
2   Rape                  378 non-null    float64
3   Robbery               378 non-null    float64
4   AggravatedAssault     377 non-null    float64
5   PropertyCrime         372 non-null    object
6   Burglary              374 non-null    object
7   Theft                 375 non-null    object
8   MotorVehicleTheft     378 non-null    float64
9   State                 378 non-null    object
```

```

10 City          373 non-null    object
dtypes: float64(5), object(6)
memory usage: 32.6+ KB

```

there are 11 columns and 378 rows. type of datatypes present are float and object the size of the dataset is 32.6+ KB

```

#statistics
df.describe()

```



	Murder	Rape	Robbery	AggravatedAssault	MotorVehicleTheft
count	378.000000	378.000000	378.000000	377.000000	378.000000
mean	4.574868	41.306614	77.702646	241.319098	191.081746
std	3.383652	19.506386	53.977181	137.267225	134.532918
min	0.000000	2.700000	2.300000	25.500000	15.700000
25%	2.200000	29.000000	36.625000	144.200000	97.550000
50%	3.800000	37.550000	67.350000	215.700000	154.450000
75%	6.275000	51.550000	106.050000	304.900000	249.650000
max	20.600000	165.600000	351.900000	981.300000	718.600000

shows summary statistics before cleaning. mean murder is lowest with 4.57 indicating that murder is the crime least committed while mean aggravated assault is highest indicating that it is the crime most committed. murder standard deviation is small indicating a stable crime pattern while the std deviation for aggravated assault is very large indicating variability in the crime pattern

```

#column list
from tabulate import tabulate

# Get all column names
column_names = df.columns.tolist()

# Convert column names to a list of lists for tabulate
column_names_table = [[i+1, name] for i, name in enumerate(column_names)]

# Print column names as a table
print(tabulate(column_names_table, headers=["Index", "Column Name"]))

```



Index	Column Name
1	ViolentCrime
2	Murder

```

3 Rape
4 Robbery
5 AggravatedAssault
6 PropertyCrime
7 Burglary
8 Theft
9 MotorVehicleTheft
10 State
11 City

```

shows columns in a list

```

#changing datatype to numeric
#identifying numerical cols with object dtype
object_cols=df.select_dtypes(include='object').columns

```

```

#converting relevant cols to strings,replace commas and convert to float
for col in object_cols:
    try:
        df[col]=df[col].str.replace(',','').astype(float)
    except ValueError:
        continue
df

```



	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burglary	Th
0	412.5	5.3	56.0	78.4	272.8	3609.0	852.0	24
1	238.4	5.1	38.2	75.2	119.8	2552.4	575.3	18
2	667.9	7.8	30.4	157.9	471.8	3894.1	1099.6	26
3	114.3	2.5	28.2	20.7	63.0	3208.4	484.6	24
4	792.6	6.1	63.8	206.7	516.0	4607.8	883.4	30
...
373	251.6	11.4	6.3	74.6	159.3	823.2	265.5	5
374	237.5	11.5	5.2	82.3	138.6	1320.0	377.1	8



datatypes changed to numeric

```
#checking datatype
df.dtypes
```

```

ViolentCrime    float64
Murder          float64
Rape            float64
Robbery         float64
AggravatedAssault float64
PropertyCrime   float64
Burglary        float64
Theft           float64
MotorVehicleTheft float64
State           object
City            object
dtype: object

```

datatypes successfully changed to numeric except state and city which are still categorical and will be used in the analysis

DATA CLEANING

```
#checking for missing values
missing_values=df.isnull().sum()
missing_values
```

```

ViolentCrime    1
Murder          0
Rape            0
Robbery         0
AggravatedAssault 1
PropertyCrime   6
Burglary        4
Theft           3
MotorVehicleTheft 0
State           0
City            5
dtype: int64

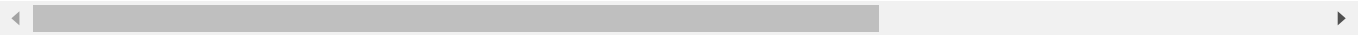
```

this shows total number of missing values in each column

```
#dropping missing values
df_cleaned=df.dropna()
df_cleaned
```



	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burglary	Th
0	412.5	5.3	56.0	78.4	272.8	3609.0	852.0	24
1	238.4	5.1	38.2	75.2	119.8	2552.4	575.3	18
2	667.9	7.8	30.4	157.9	471.8	3894.1	1099.6	26
3	114.3	2.5	28.2	20.7	63.0	3208.4	484.6	24
4	792.6	6.1	63.8	206.7	516.0	4607.8	883.4	30
...
373	251.6	11.4	6.3	74.6	159.3	823.2	265.5	5
374	237.5	11.5	5.2	82.3	138.6	1320.0	377.1	8



after successfully removing the missing values the cleaned dataset consists of 366 rows and 11 cols

```
#checking for duplicates
duplicates=df_cleaned.duplicated().sum()
duplicates
```

0

there are no duplicated present in the cleaned dataset

```
#checking for outliers
#introducing variable numeric columns
numeric_columns=df_cleaned.select_dtypes(include=['float64']).columns
numeric_columns
```

Index(['ViolentCrime', 'Murder', 'Rape', 'Robbery', 'AggravatedAssault',
'PropertyCrime', 'Burglary', 'Theft', 'MotorVehicleTheft'],
dtype='object')

shows the numerical columns

```

outliers=df_cleaned[numeric_columns].apply(lambda x: (x - x.mean()).abs() > 3 * x.std()).sum
print("\nOutliers:")
print(outliers)

```



```

Outliers:
ViolentCrime      5
Murder            6
Rape              5
Robbery           7
AggravatedAssault 5
PropertyCrime     1
Burglary          5
Theft             1
MotorVehicleTheft 7
dtype: int64

```

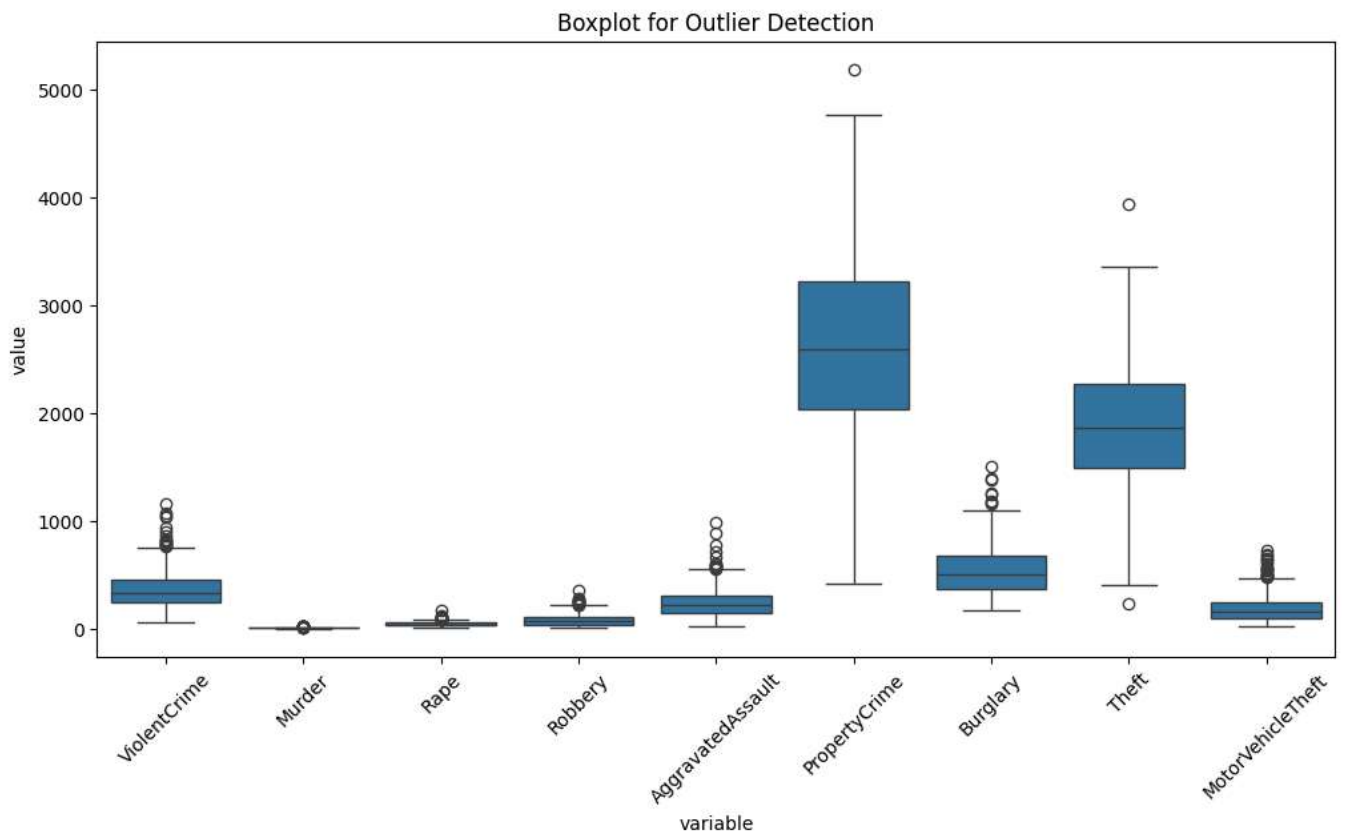
this shows the total number of outliers in each numerical column

```

#visualizing outliers in box plots
import matplotlib.pyplot as plt
import seaborn as sns
melted_df=df_cleaned[numeric_columns].melt()
#creating boxplot
plt.figure(figsize=(12, 6))
sns.boxplot(x='variable', y='value', data=melted_df)
plt.title('Boxplot for Outlier Detection')
plt.xlabel('variable')
plt.ylabel('value')

#rotating x axis labels to visualize clearly
plt.xticks(rotation=45)
plt.show()

```



the boxplot displays outliers present in the dataset

```
#removing outliers using z score
threshold=3
z_scores=df_cleaned[numeric_columns].apply(lambda x: (x - x.mean())/x.std())

outlier_rows=z_scores[(z_scores.abs().gt(threshold)).any(axis=1)]
outlier_rows
```




	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burg
4	2.352197	0.430315	1.166822	2.393407	1.994221	2.404828	1.49
5	3.143972	-0.039645	-0.288218	0.780823	3.885237	2.353823	2.73
9	3.711650	1.047137	6.400851	2.251887	3.088294	1.701587	0.06
18	1.051109	0.782785	-0.457887	0.968896	1.054162	0.982399	1.80
23	1.067627	-0.480232	3.233698	-0.321544	1.089033	0.251159	0.88
84	2.284472	-1.361407	4.128316	0.829238	2.140969	1.256903	3.07
94	2.951810	4.307483	0.827484	3.143836	2.443910	0.369482	0.69
140	2.224456	-0.215880	0.133384	1.186762	2.459166	2.603372	3.63
177	1.449199	-0.069017	1.130831	0.319021	1.630257	1.929955	4.20
184	2.475533	0.665295	1.141114	3.627984	1.673119	0.225839	1.43
211	3.703391	2.016429	0.575551	3.384048	3.434096	1.615401	2.21
220	1.732762	1.517097	-0.231662	3.493912	0.918311	0.154261	-0.29
224	1.166737	0.782785	-0.571000	1.384146	1.060700	1.290257	0.80
225	4.375133	3.073839	0.009988	0.819927	5.374512	2.519377	2.78
237	0.930526	3.426309	0.339043	1.672772	0.443196	0.484396	-0.03
243	3.880136	0.870902	0.734937	0.901860	4.642225	1.949675	1.13
260	2.729364	2.985721	1.305642	5.097186	1.354196	0.237039	-0.39
263	2.084051	2.398272	0.637249	0.717511	2.320409	1.406876	3.13
272	1.378170	0.988392	3.279971	0.680269	1.065785	3.114278	3.71
275	0.606217	0.459687	3.269688	-0.518927	0.528193	-0.046230	-0.41
290	0.428371	2.691996	-0.221379	0.929792	0.167135	-0.534737	-0.54
296	0.657974	0.195335	-0.524726	2.924108	-0.203368	1.178630	-0.10
297	0.692112	0.518432	-0.596707	3.007903	-0.188838	0.888057	-0.07
298	0.855092	-0.157135	-0.329350	3.318874	-0.116190	1.980960	-0.07
305	0.193261	3.044466	-0.961752	1.482837	-0.263665	0.829626	0.62
339	0.308338	-0.039645	0.585834	0.847859	-0.005040	2.576104	0.06
375	-0.737819	3.925641	-1.856370	-0.222852	-0.721345	-2.126982	-1.42
377	-0.569332	4.689325	-1.861512	1.482837	-1.182657	-1.644683	-1.13

this shows the extreme values in the dataset

```
#dropping rows with outliers
df_filtered=df_cleaned.drop(outlier_rows.index)
df_filtered
```



	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime	Burglary	Th
0	412.5	5.3	56.0	78.4	272.8	3609.0	852.0	24
1	238.4	5.1	38.2	75.2	119.8	2552.4	575.3	18
2	667.9	7.8	30.4	157.9	471.8	3894.1	1099.6	26
3	114.3	2.5	28.2	20.7	63.0	3208.4	484.6	24
6	216.5	0.8	28.7	25.5	161.6	1430.2	218.9	11
...
371	108.7	4.8	3.5	40.5	59.9	710.2	284.5	4
372	99.3	6.9	2.7	53.1	36.6	1031.9	287.3	6

the filtered dataset consist of 338 rows and 11 cols

```
#summary statistics after cleaning
df_cleaned.describe()
```



	ViolentCrime	Murder	Rape	Robbery	AggravatedAssault	PropertyCrime
count	366.000000	366.000000	366.000000	366.000000	366.000000	366.000000
mean	365.400546	4.634973	41.105738	78.167760	241.493716	2632.277049
std	181.617205	3.404547	19.449643	53.702617	137.650865	821.481927
min	61.600000	0.000000	2.700000	2.300000	25.500000	420.000000
25%	236.750000	2.225000	29.000000	37.025000	144.300000	2027.950000
50%	332.750000	3.900000	37.400000	67.900000	215.600000	2591.550000
75%	447.625000	6.400000	50.625000	106.975000	305.800000	3224.375000
max	1160.000000	20.600000	165.600000	351.900000	981.300000	5190.600000

this shows the summary statistics after cleaning.

```
#checking for completeness
#checking for missing values
missing_values=df_filtered.isnull().sum()
#total number of entries in the dataframe
total_entries=df_filtered.shape[0] * df_cleaned.shape[1]
#total number of complete entries
complete_entries=total_entries-missing_values.sum()
#completeness percentage
completeness_percentage=(complete_entries/total_entries)*100
print(f"Completeness Percentage: {completeness_percentage:.2f}%")
```



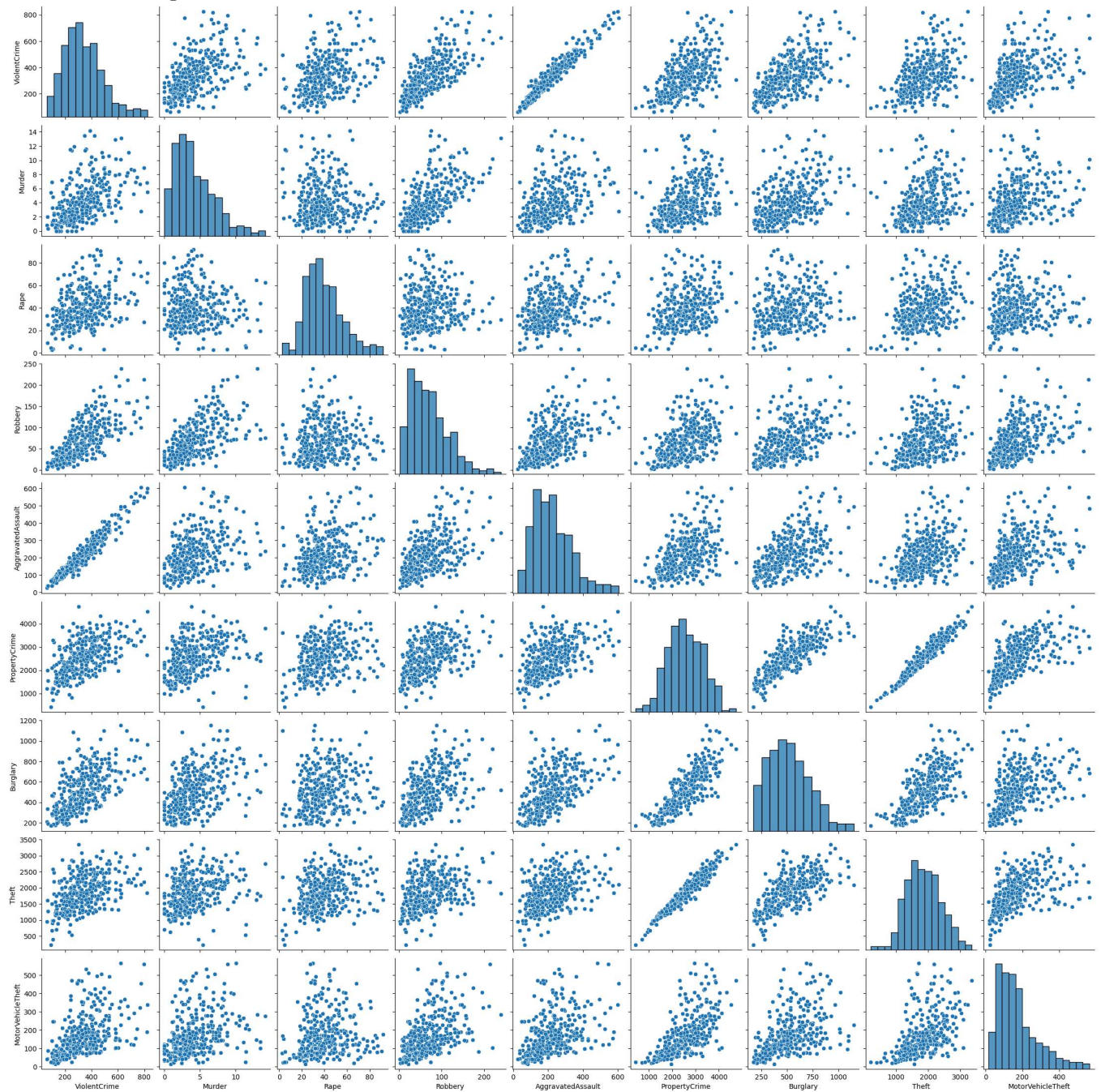
Completeness Percentage: 100.00%

EXPLORATORY DATA ANALYSIS

```
import seaborn as sns
import matplotlib.pyplot as plt
# Pairplot to visualize pairwise relationships and distributions
sns.pairplot(df_filtered)
```

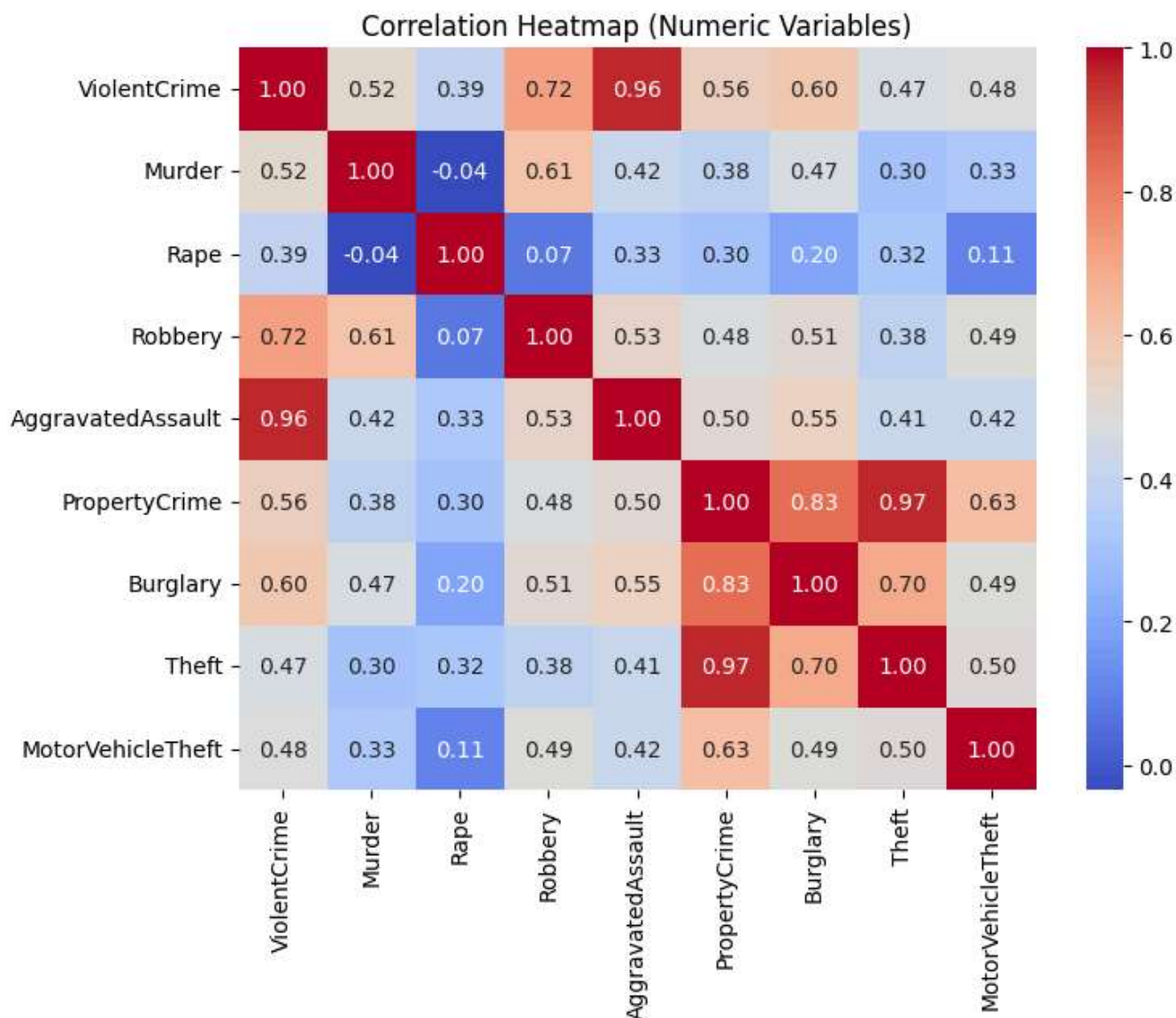


<seaborn.axisgrid.PairGrid at 0x797e49b784c0>



the above diagram shows different distributions between different variables. The distributions are presented as scatter plots and bar plots

```
# Plot the correlation heatmap
correlation_matrix = df_filtered[numeric_columns].corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap (Numeric Variables)')
plt.show()
```



the correlation heat map shows correlation strengths between variables. All variables present a positive correlation except for murder against rape which shows -0.04 implying that there is no correlation between them

MODELLING USING DIMENSIONALITY REDUCTION ALGORITHMS

STEP1:SCALE/NORMALIZE NUMERIC DATA

```
from sklearn.preprocessing import StandardScaler
```

```
#extract numeric columns for scaling
X_numeric=df_filtered[numeric_columns]
#scale the data
scaler=StandardScaler()
X_scaled=scaler.fit_transform(X_numeric)
print("Numeric Data after Scaling:")
print(X_scaled)
```

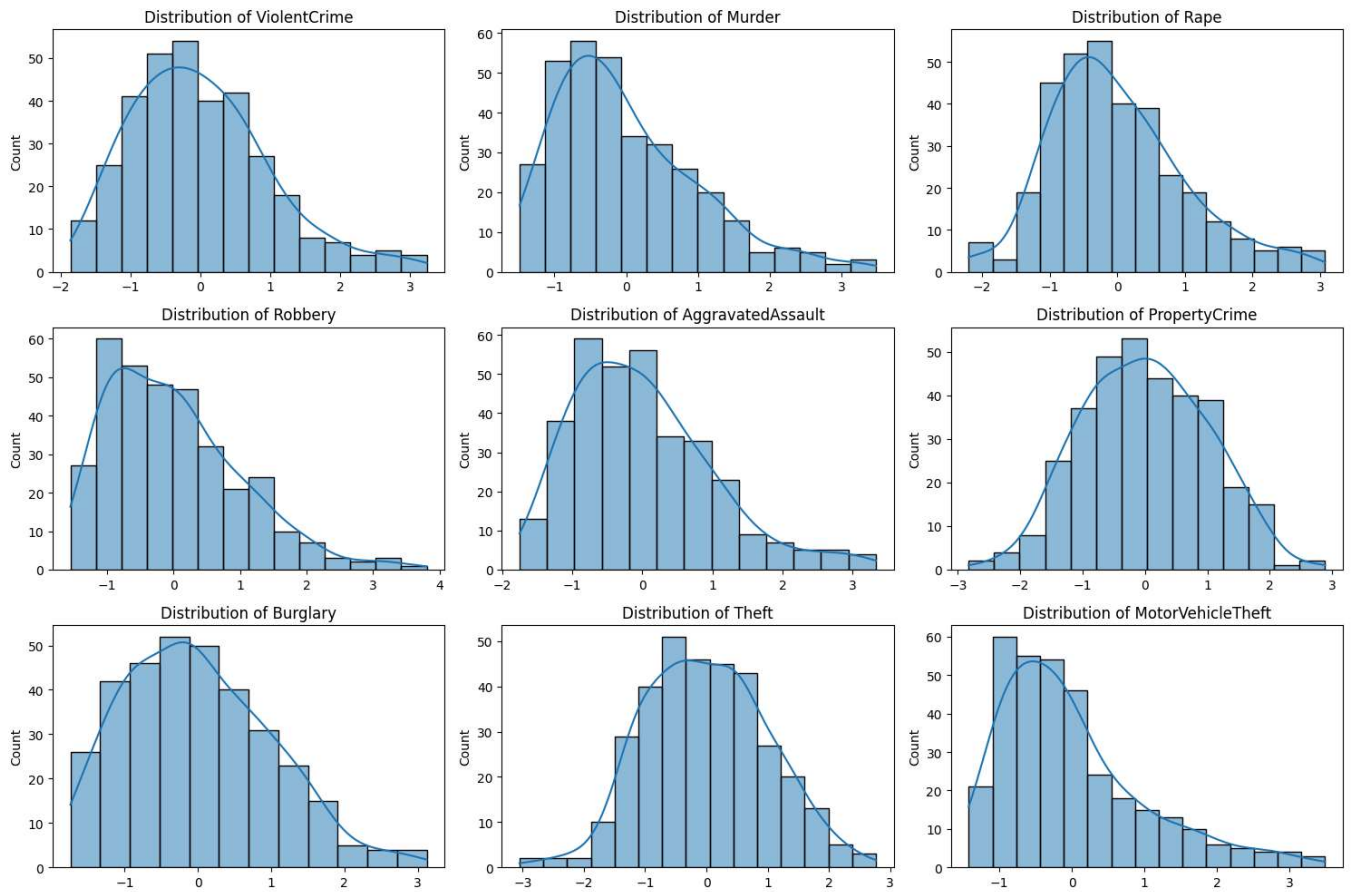


Numeric Data after Scaling:

```
[[ 0.48355514  0.36792729  0.95058662 ...  1.63959062  1.17295524
  0.79325645]
 [-0.67889652  0.29800352 -0.10019872 ...  0.27082441 -0.02090497
 -0.45267661]
 [ 2.18884035  1.2419744  -0.56065521 ...  2.86440635  1.46964982
 -0.29525793]
 ...
 [-0.59076118  2.50060223 -1.98334772 ... -1.26167928 -2.48447632
 -1.32474031]
 [-0.68490575  2.53556412 -2.0482839  ... -0.70962178 -1.86853689
 -0.83549021]
 [-1.65572982  0.40288917 -2.08370363 ... -1.7395355  -3.05214697
 -1.34262879]]
```

It is important to scale data by standardizing and normalizing the data inorder to improve the models performance

```
#visualizing scaled data
#plot the distributions of the numeric data
plt.figure(figsize=(15,10))
for i in range(X_scaled.shape[1]):
    plt.subplot(3,3,i+1)
    sns.histplot(X_scaled[:,i],kde=True)
    plt.title(f"Distribution of {numeric_columns[i]}")
plt.tight_layout()
plt.show()
```

this shows the scaled distributions of the different types of crimes

STEP 2:PRINCIPAL COMPONENT ANALYSIS(PCA)


```
from sklearn.decomposition import PCA
```

```
#Apply PCA
```

```
pca=PCA(n_components=5)
```

```
X_pca=pca.fit_transform(X_scaled)
```

```
print("PCA applied")
```

```
X_pca
```

```

PCA applied
array([[ 2.45851715, -1.20904625, -0.31251196, -0.29495372, -0.09047526],
       [-0.54168366, -0.06995562, -0.70096106, -0.71838693, -0.47842597],
       [ 4.71629688,  0.90802976, -0.17700884, -1.36187643,  1.71280671],
       ...,
       [-2.68747477,  3.96833671, -0.25475934, -1.18318518, -0.49152127],
       [-1.90030077,  3.47969358, -0.99344337, -1.12713728, -0.56241737],
       [-5.01436369,  2.58044975, -0.33350136, -0.18804866,  0.02317186]])

```

this shows the X_scaled data after transforming it using PCA.

```
#visualizing pca components
```

```
plt.figure(figsize=(15,6))
```

```
plt.scatter(X_pca[:,0], X_pca[:,1], c='blue',edgecolor='k',s=50)
```

```
plt.xlabel('PC1')
```

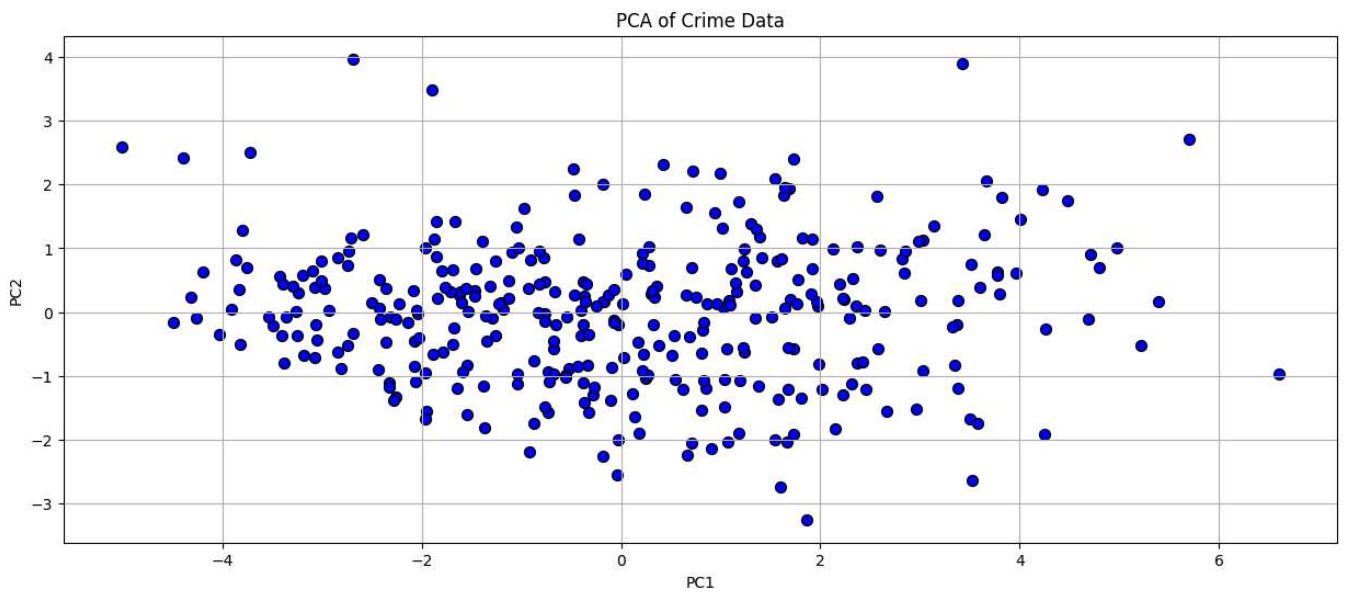
```
plt.ylabel('PC2')
```

```
plt.title('PCA of Crime Data')
```

```
plt.grid(True)
```

```
plt.show()
```

```
print("PCA components plotted")
```



STEP 3:PRINCIPAL COMPONENT REGRESSION(PCR)

```

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# Use PCA transformed data for regression (assuming some target variable, `y`)
# Here, we create a dummy target variable for demonstration purposes
import numpy as np
y = np.random.rand(X_pca.shape[0])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2, random_state=42)

# Train the regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict and evaluate
y_pred = regressor.predict(X_test)

print("Principal Component Regression (PCR) applied.")
print("First few predictions:", y_pred[:5])

```

⇒ Principal Component Regression (PCR) applied.
 First few predictions: [0.49854718 0.53905605 0.50548246 0.46584472 0.43688532]

STEP 8: Partial Least Square Regression (PLSR)

```

from sklearn.cross_decomposition import PLSRegression

# Apply PLS Regression
pls = PLSRegression(n_components=2)
pls.fit(X_train, y_train)
y_pred_pls = pls.predict(X_test)

print("Partial Least Squares Regression (PLS) applied.")
print("First few PLS predictions:", y_pred_pls[:5])

```

⇒ Partial Least Squares Regression (PLS) applied.
 First few PLS predictions: [[0.49858832]
 [0.53902211]
 [0.50549316]
 [0.46576528]
 [0.43700367]]

Step 9: Multidimensional Scaling (MDS)

```
from sklearn.manifold import MDS
```

```
# Apply MDS
```

```
mds = MDS(n_components=2)
```

```
X_mds = mds.fit_transform(X_scaled)
```

```
print("Multidimensional Scaling (MDS) applied.")
```

```
print("First few rows of the MDS transformed data:")
```

```
print(X_mds[:5])
```

```
➞ /usr/local/lib/python3.10/dist-packages/sklearn/manifold/_mds.py:299: FutureWarning: The
  warnings.warn(
    Multidimensional Scaling (MDS) applied.
    First few rows of the MDS transformed data:
    [[-2.74518881  0.3945899 ]
     [ 0.51457314  0.25723657]
     [-4.90176486 -2.47410249]
     [ 0.38376731  3.00052762]
     [ 3.43531813  0.71990405]]
```

```
#visualizing mds components
```

```
# Plot the MDS components
```

```
plt.figure(figsize=(10, 7))
```

```
plt.scatter(X_mds[:, 0], X_mds[:, 1], c='green', edgecolor='k', s=50)
```

```
plt.xlabel('MDS Component 1')
```

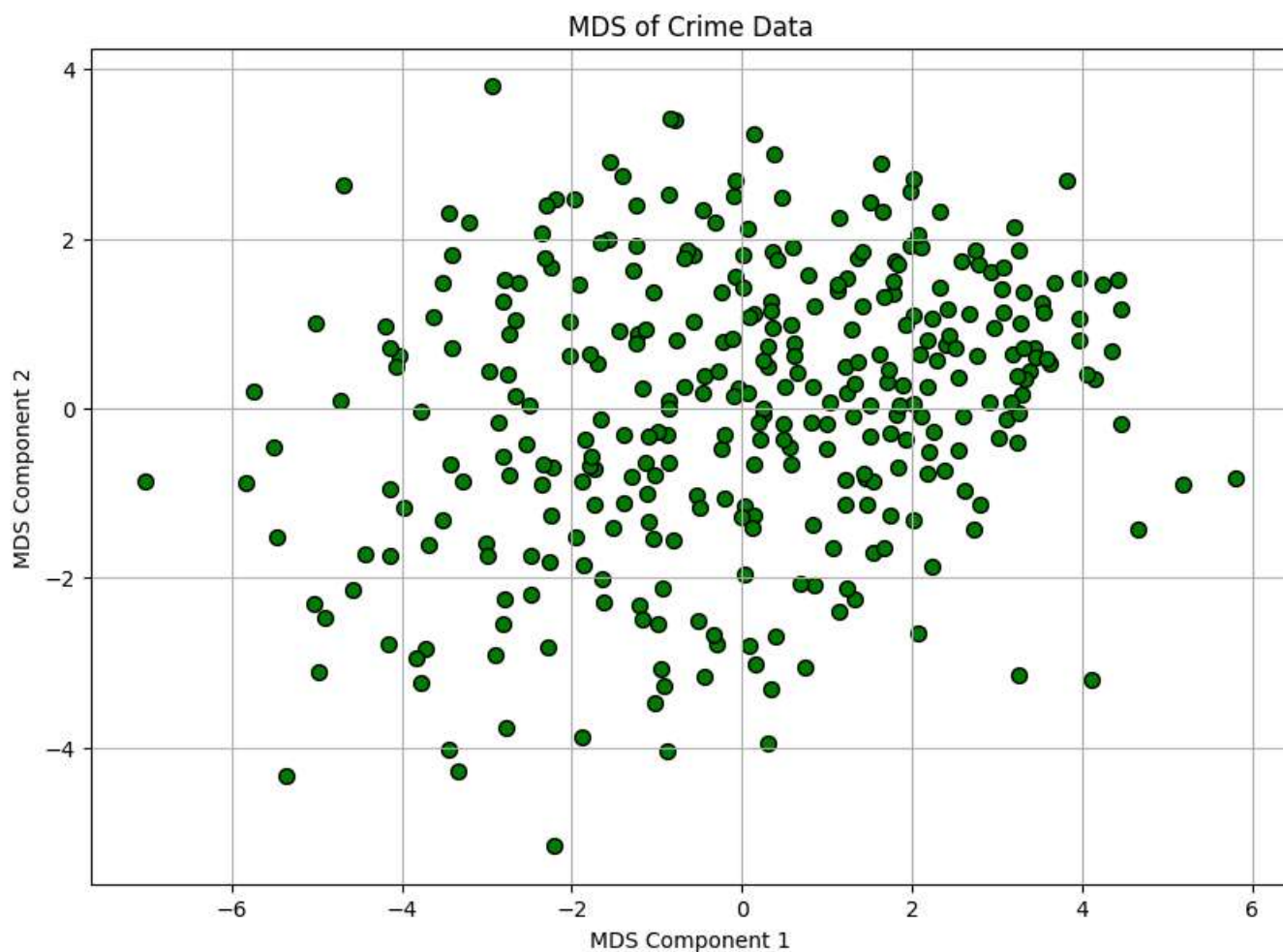
```
plt.ylabel('MDS Component 2')
```

```
plt.title('MDS of Crime Data')
```

```
plt.grid(True)
```

```
plt.show()
```

```
print("MDS components plotted.")
```



MDS components plotted.

Step 10: Mixture Discriminant Analysis (MDA)

```
from sklearn.mixture import GaussianMixture

# Apply Gaussian Mixture Model (as a proxy for MDA)
gmm = GaussianMixture(n_components=2)
gmm.fit(X_scaled)
labels = gmm.predict(X_scaled)

print("Mixture Discriminant Analysis (MDA) applied.")
print("First few cluster labels:", labels[:5])
```

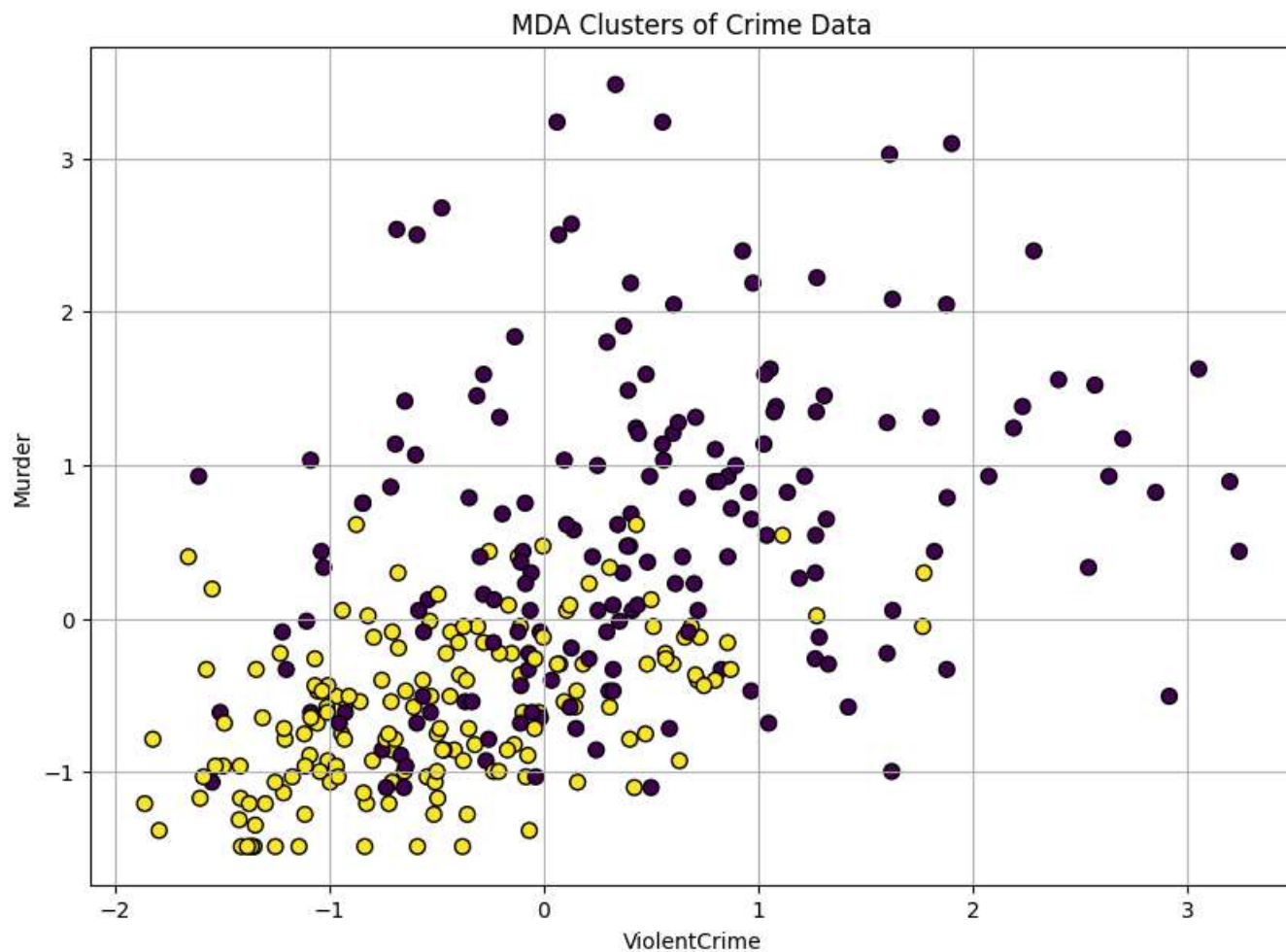


Mixture Discriminant Analysis (MDA) applied.
First few cluster labels: [0 1 0 0 1]

```

#visualizing mda components
plt.figure(figsize=(10, 7))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='viridis', edgecolor='k', s=50)
plt.xlabel(df.columns[0])
plt.ylabel(df.columns[1])
plt.title('MDA Clusters of Crime Data')
plt.grid(True)
plt.show()
print("MDA clusters plotted.")

```



MDA clusters plotted.