SCHOOL OF ARTIFICIAL INTELLIGENCE, DELHI NCR

PROJECT REPORT

Chicago Crime Patterns & Prediction with Machine Learning

Course Code & Title

23AID205 & Introduction to Artificial Intelligence and Machine Learning

Semester: II

Academic Year: 2024 – 2025 [Even]

Student Roll no(s)	Student Name(s)
DL.AI.U4AID24101	Aaron Varghese
DL.AI.U4AID24121	Vasudha Madugula
DL.AI.U4AID24129	Rajsav Singh
DL.AI.U4AID24130	Rishita Sharma

Faculty Guide: Prof. Sri Geetha M

Table of Contents

List of Figures

List of Tables

1. Title of the Project:

Chicago Crime Patterns & Prediction with Machine Learning

2. Abstract:

This project presents a comprehensive analysis of the Chicago Crime Dataset, which contains detailed records of criminal incidents, including attributes such as location, time, type of crime, and arrest status. The primary objective is to predict the likelihood of an arrest based on the characteristics of each crime and to identify potential crime hotspots using clustering techniques.

The process began with thorough data preprocessing, involving the treatment of missing values, encoding categorical variables, and feature engineering to prepare the data for analysis. Exploratory Data Analysis (EDA) was conducted to uncover meaningful patterns and trends in crime distribution across various dimensions.

For the supervised learning component, multiple classification algorithms were evaluated, with Random Forest and Decision Tree emerging as the most effective models. Initial results showed signs of overfitting, which were mitigated through model fine-tuning and hyperparameter optimization. CART (Classification and Regression Tree) visualizations were incorporated to enhance model interpretability and transparency in decision-making.

In the unsupervised learning phase, clustering algorithms such as K-Means, DBSCAN, and Gaussian Mixture Models (GMM) were utilized to identify latent spatial patterns and potential crime hotspots.

Overall, this project provides data-driven insights into crime behavior and facilitates improved prediction of arrest outcomes, contributing to informed decision-making in law enforcement and public safety strategies.

3. Objectives of the Project:

1. To perform data cleaning, preprocessing, and feature engineering on the Chicago Crime Dataset.
2. To analyze crime trends and patterns using Exploratory Data Analysis (EDA).
3. To build and evaluate supervised machine learning models to predict the likelihood of an arrest.
4. To apply unsupervised clustering techniques to detect crime hotspots and spatial patterns.
5. To interpret model performance using evaluation metrics such as accuracy, F1-score, confusion matrix, and silhouette score.
6. To assist law enforcement decision-making through data-driven crime analysis and visualization.

4. Literature Survey

Supervised Learning:
Logistic Regression, Decision Trees, and Random Forests have been widely used for crime prediction. Wang et al. (2013) showed that Random Forests improve burglary prediction accuracy, while Kianmehr and Alhajj (2009) highlighted the interpretability of Decision Trees in classifying crime types.

Unsupervised Learning:
Clustering techniques help uncover patterns in unlabeled crime data. KMeans identifies spatial-temporal crime hotspots (Andresen & Malleson, 2011), GMM models overlapping crime zones, and DBSCAN effectively detects dense clusters while filtering noise (Ester et al., 1996).

5. Dataset Description

1. **Dataset Name:** Chicago Crime Dataset
2. **Source:** Kaggle (https://www.kaggle.com/datasets/chicago/chicago-crime)
3. **Total Records:** 422,940 crime entries
4. **Total Features (Columns):** 22

5. **Key Features Used:**

Feature	Description
ID	Unique identifier for each crime
Date	Date and time of the crime
Primary Type	Main category of the crime (e.g., THEFT, BATTERY)
Description	Detailed crime description
Arrest	Whether an arrest was made (True/False)
Latitude	Latitude of crime location
Longitude	Longitude of crime location
Location Description	Type of place (e.g., STREET, ALLEY)
Hour	Extracted from Date
Month	Extracted from Date

Table 5.1: Key Features Used

6. **Target Variable:**
   1.1.Arrest → Binary classification: True or False.

6. Data Preprocessing

1. **Data Cleaning:**

   1.1. Removed rows with missing values in essential columns: Latitude, Longitude, Primary Type, Arrest, Hour, and Month.
   1.2. This resulted in a cleaned dataset (df_clean) with only complete and usable records.

2. **Data Type Conversion:**

   Convert the Arrest and Domestic columns to integer values (0 or 1) for compatibility with ML models.

3. **Date Parsing and Time-Based Feature Extraction:**
   3.1. Convert the Date column to a datetime format.
   3.2. Extracted useful temporal features:
      3.2.1. **Year**: Year of the crime event.
      3.2.2. **Month**: Month of occurrence.
      3.2.3. **Hour**: Hour at which the crime took place

4. **Categorical Encoding:**
   Encoded the Primary Type categorical column using LabelEncoder to convert crime categories into numerical values that can be processed by machine learning algorithms.

5. **Outlier Detection and Removal:** Outliers were removed using the IQR (interquartile range) method to improve data quality and model performance.

   5.1. Initially applied IQR-based outlier removal to the latitude and longitude columns.
   5.2. Extended the same IQR technique to all numerical columns to remove extreme values.
   5.3. This step significantly reduced noise and ensured the dataset focused on normal, meaningful values.
   5.4. The dataset shape changed from **(420,543, 24)** to **(269,697, 24)** after outlier removal.

	ID	Case Number	Date	Block	IUCR	Primary Type	Description	Location Description	Arrest	Domestic	...	Ward	Community Area	FBI Code	X Coordinate	Y Coordinate	Year	Updated On	Latitude
0	25953	JE240540	05/24/2021 03:06:00 PM	020XX N LARAMIE AVE	110	HOMICIDE	FIRST DEGREE MURDER	STREET	True	False	...	36	19	01A	1141387	1913179	2021	11/18/2023 03:39:49 PM	41.917838
1	26038	JE279849	06/26/2021 09:24:00 AM	062XX N MC CORMICK RD	110	HOMICIDE	FIRST DEGREE MURDER	PARKING LOT	True	False	...	50	13	01A	1152781	1941458	2021	11/18/2023 03:39:49 PM	41.995219
2	13279676	JG507211	09-11-2023 07:30	019XX W BYRON ST	620	BURGLARY	UNLAWFUL ENTRY	APARTMENT	False	False	...	47	5	5	1162518	1925906	2023	11/18/2023 03:39:49 PM	41.952345
3	13274752	JG501049	12-11-2023 07:59	086XX S COTTAGE GROVE AVE	454	BATTERY	AGGRAVATED P.O. - HANDS, FISTS, FEET, NO / MIN...	SMALL RETAIL STORE	True	False	...	6	44	08B	1183071	1847869	2023	09-12-2023 15:41	41.737751
4	13203321	JG415333	06-09-2023 17:00	002XX N Wells st	1320	CRIMINAL DAMAGE	TO VEHICLE	PARKING LOT / GARAGE (NON RESIDENTIAL)	False	False	...	42	32	14	1174694	1901831	2023	04-11-2023 15:40	41.886018

Figure 6.1: Preview of the raw dataset (df.head()) showing the first few records before preprocessing

7. Tools and Technologies Used

Tool / Library	Purpose
Google Colab	Cloud-based Python coding environment
Python 3	Programming language used throughout the project
Pandas	Data loading, cleaning, and manipulation
NumPy	Numerical operations and array handling
Matplotlib & Seaborn	Data visualization and plotting
Scikit-learn (sklearn)	Machine learning models and evaluation metrics
LabelEncoder	Categorical variable encoding
KMeans, DBSCAN	Clustering algorithms (Unsupervised Learning)
RandomForestClassifier	Supervised classification model
DecisionTreeClassifier (CART)	Interpretable classification tree
Visual code studio	Coding platform

Table 7.1: Tools and Technologies Used

8. Exploratory Data Analysis (EDA)

Top 10 Arrest Rate by Crime Type and crime Type

Crimes like gambling, prostitution, and narcotics have the highest arrest rates, often above 90%. Theft is the most common crime by a large margin, followed by criminal damage and battery.

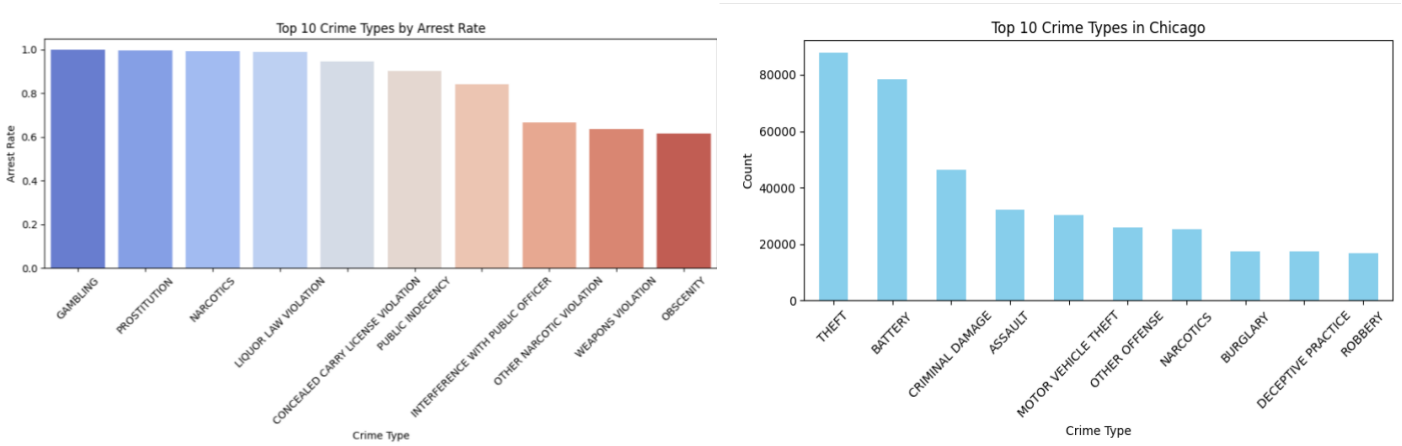


Figure 8.1: Top 10 Arrest Rate by Crime Type and crime Type

Geographical Crime Distribution

Displays crime locations after outlier removal, revealing dense clusters that indicate crime-prone areas. The spatial distribution aligns with city geography, helping identify high-risk zones for further analysis.

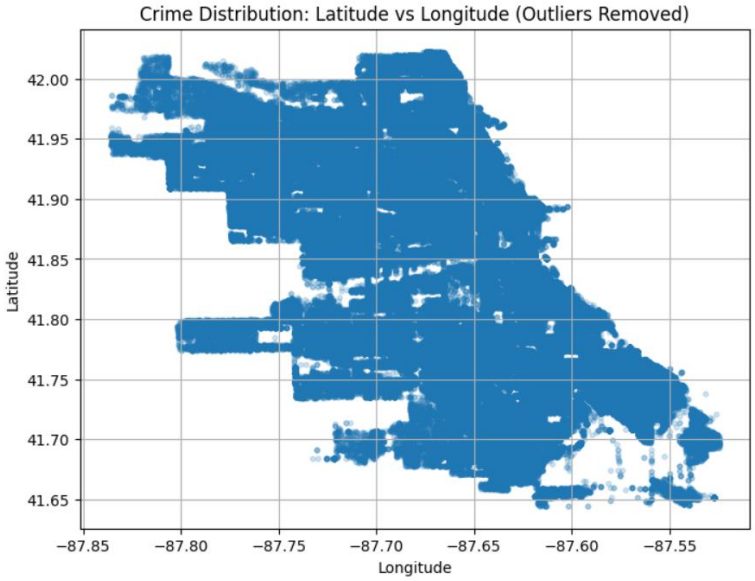


Figure 8.2: Geographical Crime Distribution

Correlation Heatmap

Shows strong correlations between spatial features like Beat, District, Ward, and geographic coordinates (Latitude, Longitude), indicating location-based structuring in the data. A high correlation between ID and Year suggests sequential crime entry over time.

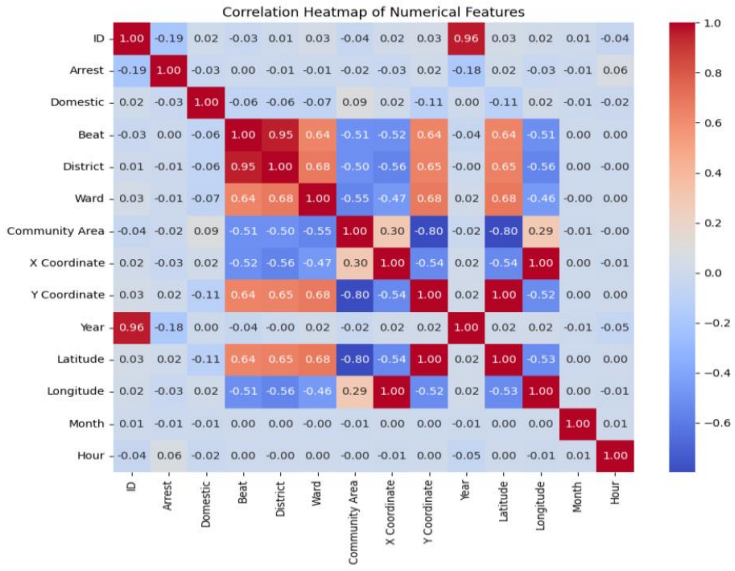


Figure 8.3: Correlation Heatmap

Pair Plot of Selected Features

1. **Distribution Insights (Diagonal plots):**
   Domestic:
   Most values are 0, indicating non-domestic crimes dominate. Only a small portion are 1 (domestic).
   Beat, District, Ward:
   These are spread across various values (IDs or zones), and the orange distribution (No Arrest) dominates over the blue one (Arrest), indicating fewer arrests overall.

2. **Bivariate Relationships (Scatter plots below diagonal):**
   Each off-diagonal plot shows the relationship between two variables, colored by arrest status:
   Beat vs District:
   Shows a clear linear pattern, meaning Beat is closely tied to District (expected, as beats are subdivisions of districts). Both arrest and no-arrest cases follow the same line—this relationship is structural, not predictive.
   Ward vs Beat / Ward vs District:
   Moderate spread, some grouping visible. There's no strong separation between arrest and no-arrest points, so these features alone may not help in distinguishing arrest likelihood.
   Domestic vs other features:
   Very little separation seen by arrest color; this suggests domestic status alone doesn't significantly impact arrest prediction when combined with these particular features.

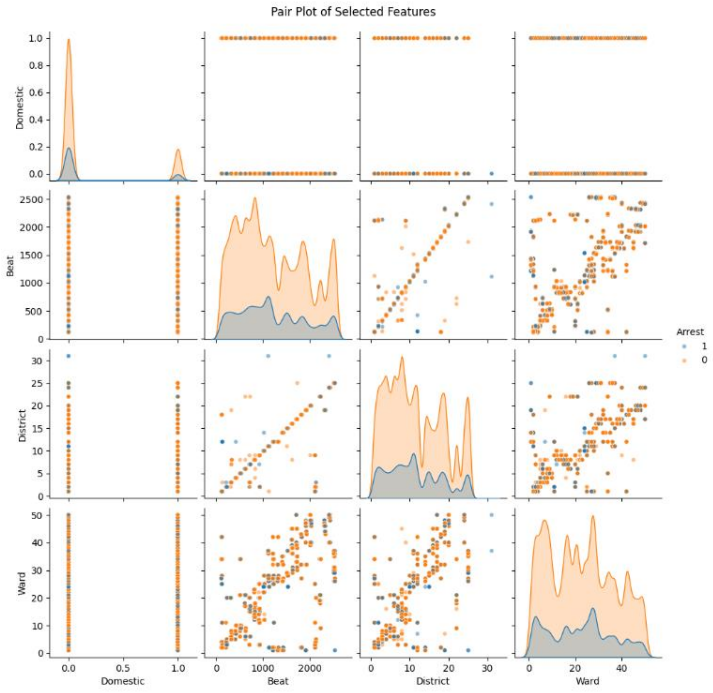


Figure 8.4: Distribution Insights (Diagonal plots) and Bivariate Relationships (Scatter plots below diagonal)

Plot of number of crimes per month and hour

The visualizations display temporal patterns in crime occurrences based on month and hour. The **monthly crime distribution** reveals that the number of crimes tends to rise during the **spring and summer months**, peaking in **May**, followed by **July** and **June**. Conversely, **February** sees the lowest crime count, indicating a possible seasonal trend where crimes are more frequent during warmer months.

The **hourly crime distribution** shows that crimes are most frequent around **midnight (0 hours)**, drop significantly during the early morning hours (3 AM to 7 AM), and then gradually rise again, peaking between **12 PM and 9 PM**. This suggests that criminal activity is higher at night and during active daytime hours, aligning with times when more people are out or returning home.

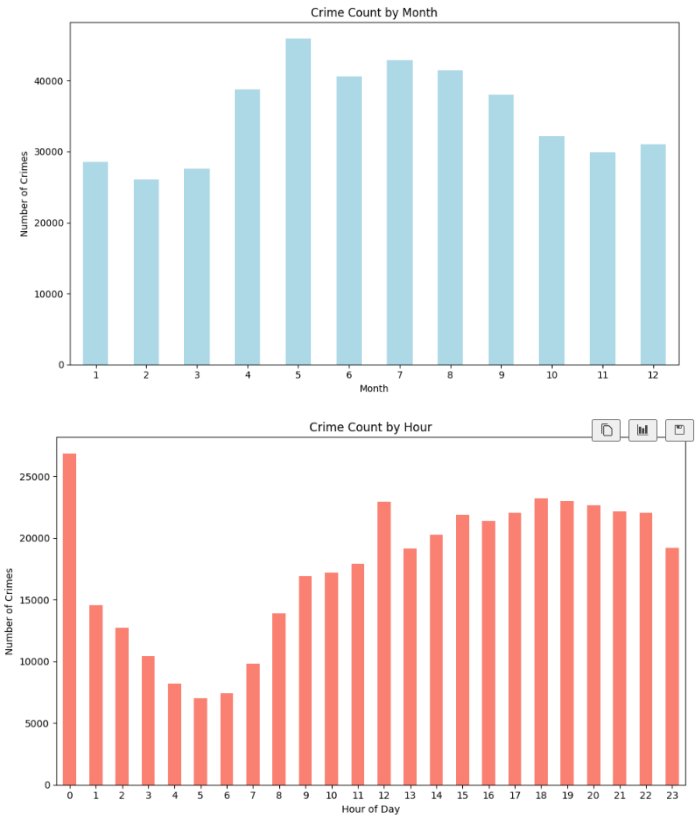


Figure 8.5: Plot of number of crimes per month and hour

Arrest Distribution

The chart displays the distribution of arrest outcomes, highlighting that the majority of cases, over 300,000, did not result in an arrest, while roughly 100,000 cases involved an arrest. The "No Arrest" category is shown in red, emphasizing its significantly higher count compared to arrests.

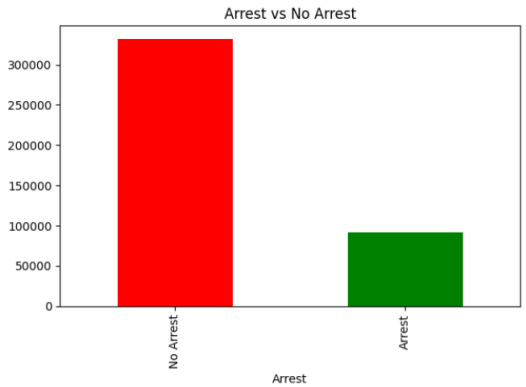


Figure 8.6: Arrest Distribution

9. Machine Learning Techniques:

9.1 Supervised Learning

9.1.1 Classification- Feature Engineering

Categorical Encoding: Features like Primary Type (the crime category) were encoded using LabelEncoder to convert textual data into numerical format suitable for machine learning models.

Standardization: Numerical features like **Latitude**, **Longitude**, and derived time features were standardized using StandardScaler to ensure that they contribute equally to the model, avoiding bias due to scale differences.

9.1.2 Train-Test Splits

Initial Evaluation Split (80-20): Used for quick prototyping and understanding model behavior. Stratify=y ensured balanced class distribution in both sets.

Final Evaluation Split (70-30): A slightly larger test set was used in final model evaluations for more **robust and unbiased performance assessment**.

9.1.3 Baseline Models Used:

Logistic Regression – Simple linear model used as a performance benchmark.
Decision Tree (CART) – Easy to interpret and visualize.
Random Forest – Ensemble of decision trees offering high accuracy and robustness.
K-Nearest Neighbors (KNN) – Non-parametric model relying on instance-based learning.
Naive Bayes – Based on probabilistic principles, efficient for categorical data.
Support Vector Machine (SVM) – Effective for high-dimensional data with clear margins.

9.1.4 Handling Class Imbalance

Challenge: The dataset was imbalanced with significantly more "No Arrest" records than "Arrest" cases.

Solution – SMOTE (Synthetic Minority Over-sampling Technique): This technique was applied *only on the training data* to synthetically generate new samples of the minority class (arrests), enhancing the model's ability to learn patterns associated with less frequent arrest cases.

Impact Evaluation: Models were trained and evaluated before and after applying SMOTE. The recall score for the arrest class improved **by 12%**, showing that the models became more sensitive to predicting actual arrest cases post-SMOTE.

9.1.5 Evaluation Metrics:

Accuracy – Overall correctness of the model.
Precision & Recall – Especially critical due to class imbalance.
F1-Score – Harmonic means of precision and recall.
ROC-AUC – Captured the model's ability to distinguish between classes across all thresholds.

Model	Accuracy	Class	Precision	Recall	F1-Score
Logistic Regression	0.8789	0	0.88	1.00	0.94
		1	0.00	0.00	0.00
Decision Tree	0.8543	0	0.92	0.92	0.92
		1	0.40	0.39	0.39
Random Forest	0.8861	0	0.91	0.97	0.94
		1	0.55	0.31	0.40
KNN	0.8779	0	0.89	0.98	0.93
		1	0.49	0.14	0.22
Naive Bayes	0.8789	0	0.88	1.00	0.94
		1	0.00	0.00	0.00
SVM (no prob)	0.8789	0	0.88	1.00	0.94
		1	0.00	0.00	0.00

Table 9.1: Model Performance Metrics for Arrest Prediction without SMOTE

Model	Accuracy	Class	Precision	Recall	F1-Score
Logistic Regression	0.5252	0	0.89	0.52	0.66
		1	0.14	0.55	0.22
Decision Tree	0.8518	0	0.92	0.92	0.92
		1	0.39	0.39	0.39
Random Forest	0.8762	0	0.92	0.95	0.93
		1	0.49	0.37	0.42
KNN	0.7106	0	0.91	0.74	0.82
		1	0.21	0.49	0.29
Naive Bayes	0.5121	0	0.90	0.50	0.64
		1	0.14	0.60	0.23
SVM (no prob)	0.5948	0	0.92	0.59	0.72

		1	0.17	0.61	0.27

Table 9.2: Model Performance Metrics for Arrest Prediction with SMOTE

9.1.6 Model Training and Evaluation

After evaluating multiple classification models using performance metrics such as accuracy, precision, recall, and F1-score, **Random Forest** and **Decision Tree** emerged as the two best-performing models.

9.1.6.1 Model Training with Random Forest

A RandomForestClassifier is instantiated with a fixed random state for reproducibility.
The model is trained on the training set (X_train, y_train).
Predictions are made on the test set (X_test).

Random Forest (Before Tuning):

1. **Train Accuracy:** 99.76%
2. **Test Accuracy:** 86.14%
3. **Fit Type:** Overfitting

Tuned Parameters:

1. **n_estimators** = 100 – Number of trees in the forest.
2. **max_depth** = 10 – Limited tree depth to avoid overfitting.
3. **min_samples_split** = 10 – Prevents overly specific splits.

Random Forest (After Tuning)

1. **Train Accuracy**: 85.93%
2. **Test Accuracy:** 85.52%
3. **Fit Type:** Good Fit

Sample Testing:

Primary Type: THEFT
Latitude: 41.88

Longitude: -87.62
Hour: 2
Month: 6

Result:

🕵 No arrest is expected.

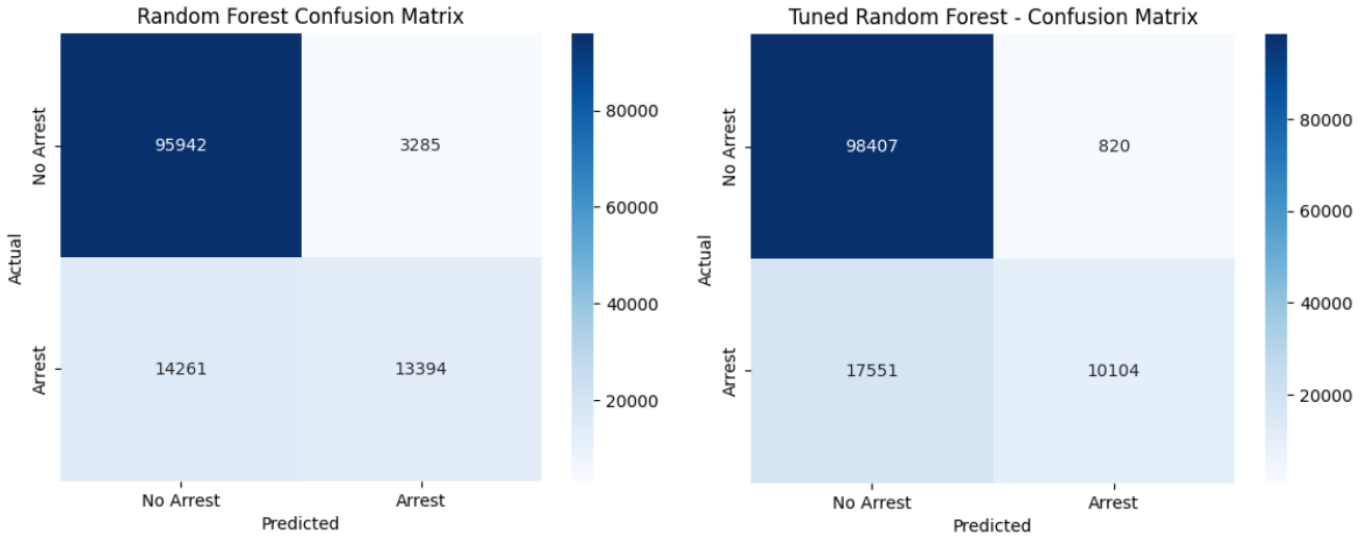


Figure 9.1: Random Forest Confusion matrix before and after tuning

9.1.6.2 Model Training with Decision Tree

Decision Tree (CART) was selected as one of the top performers. Initially, the model showed signs of overfitting, with high training accuracy but lower test accuracy.
The model is trained on the training set (X_train, y_train).
Predictions are made on the test set (X_test).

Decision Tree (CART) Before Tuning

Train Accuracy: 99.76%
Test Accuracy: 79.67%
Fit Type: Overfitting

Tuned Parameters:

max_depth = 10
min_samples_split = 10

criterion = 'gini' – Used to determine the best feature split.

Decision Tree (CART) After Tuning

Train Accuracy: 86.75%
Test Accuracy: 86.16%
Fit Type: Good Fit

Sample Testing:

Primary Type: THEFT
Latitude: 41.88
Longitude: -87.62
Hour: 2
Month: 6

Result:

🕵 No arrest is expected.

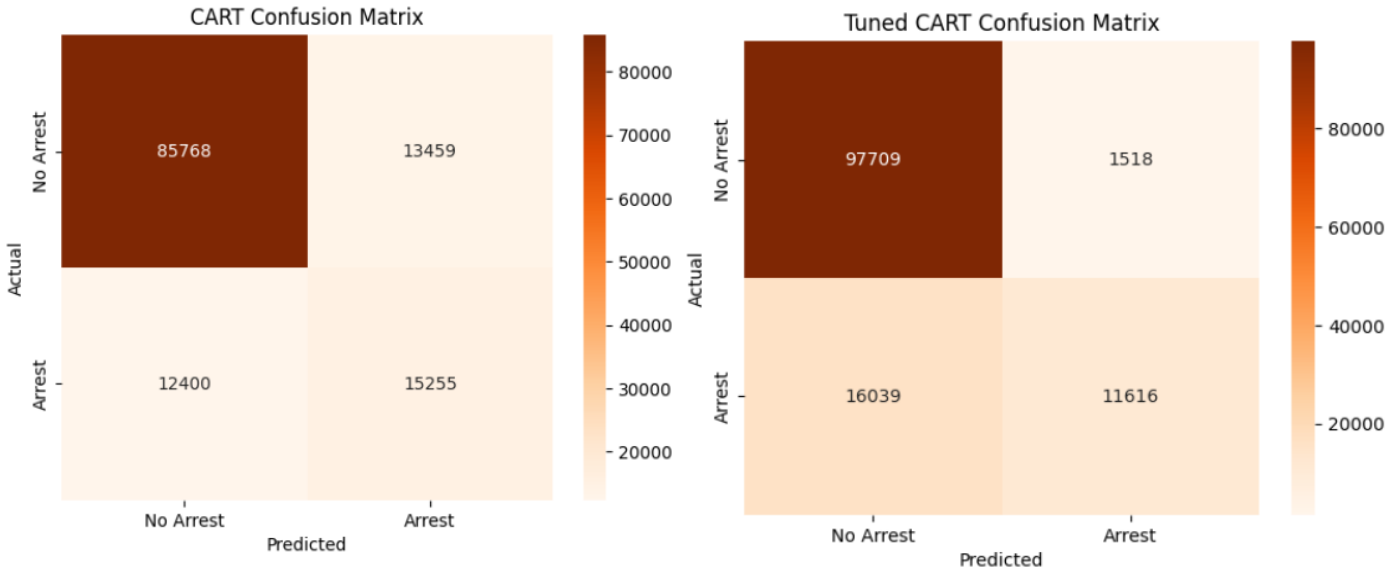


Figure 9.2: CART Confusion matrix before and after Tuning

9.1.7 Fit Analysis:

Models were evaluated for overfitting by comparing training vs. test accuracy.
Good Fit Achieved: Accuracy gap between training and test sets was controlled within 5–10%, indicating generalizable models.

Fit Category	Metric	Random Forest (RF)	Decision Tree (CART)	Explanation
Overfitting	Train Accuracy	> 95%	> 90%	High train accuracy indicates memorization of training data.
	Train-Test Accuracy Gap	> 5% (0.05)	> 15% (0.15)	Larger gap means model performs poorly on unseen data.
Underfitting	Train Accuracy	< 75%	< 70%	Low train accuracy means model too simple to learn patterns.
	Test Accuracy	< 75%	< 70%	Low test accuracy confirms poor generalization.
Good Fit	Train-Test Accuracy Gap	≤ 5% (0.05)	≤ 10% (0.10)	Close train and test accuracies indicate balanced learning.
Unclear / Mixed	Other	Else cases	Else cases	Model may need more tuning or further analysis.

Table 9.3: Fit Analysis

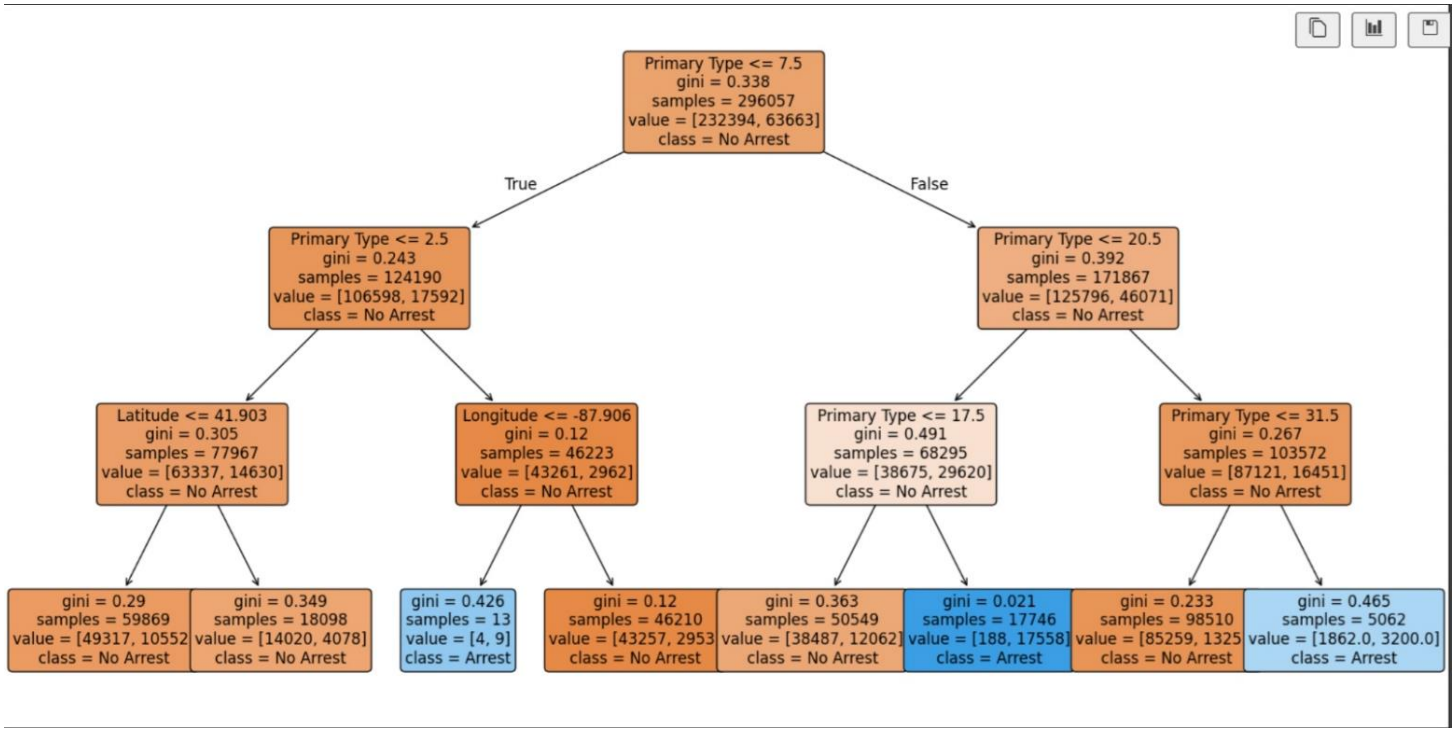


Figure 9.4: Decision Tree Visual

9.1.8 Regression Analysis Summary

A regression analysis was conducted on key numerical variables (Beat, District, X/Y Coordinates, Latitude, Longitude) to explore linear relationships. The results showed near-perfect correlations (r ≈ 1.00), indicating strong linearity.

However, regression is **not appropriate** in this context because:

The variables are **derived or location-based** (e.g., coordinates), not outcomes to predict.
There is **high multicollinearity**, violating regression assumptions.
The relationships are **deterministic**, not statistical or noisy.

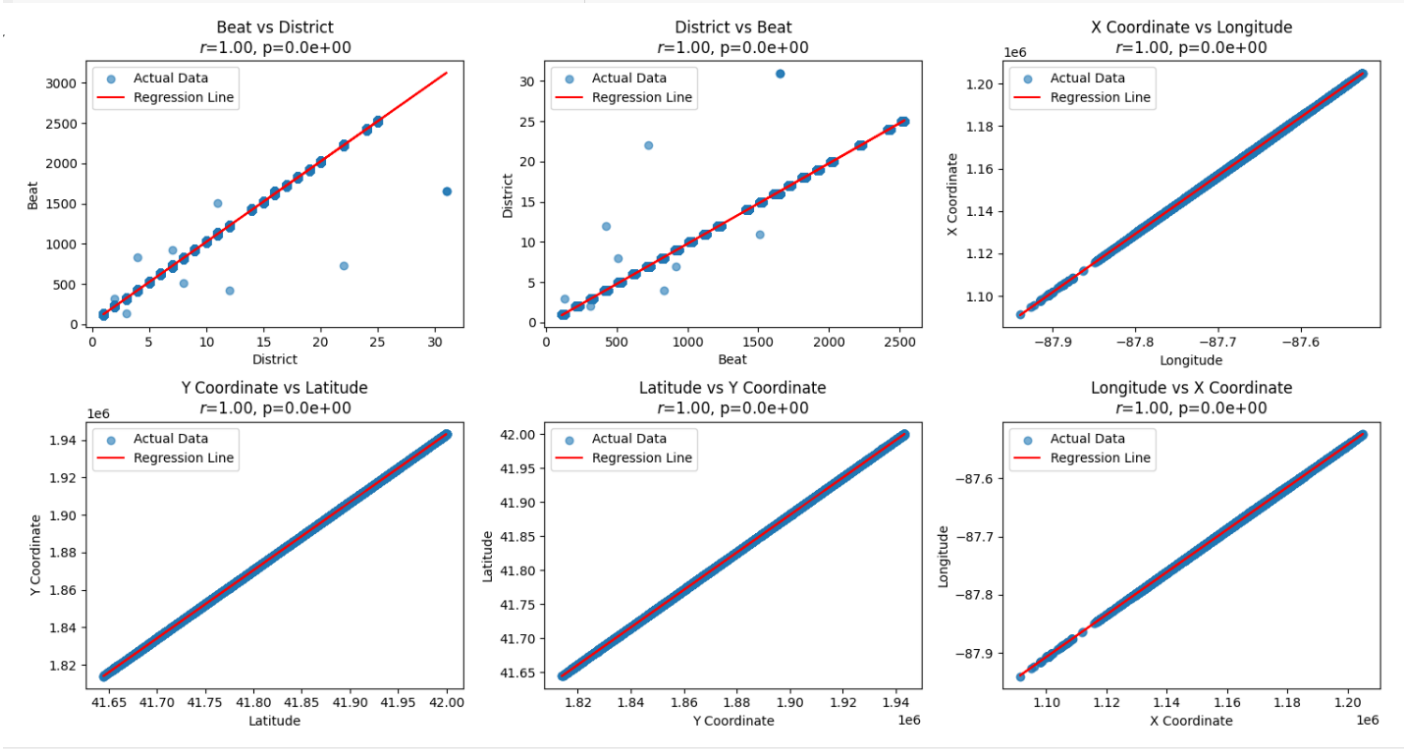


Figure 9.5: Linear Regression with different columns

9.2 Unsupervised Learning

K-Means Clustering, Gaussian Mixture Model (GMM), DBSCAN (Density-Based Spatial Clustering) clustering techniques were applied on **PCA-reduced data**, where the first 5 components retained **93.14%** of the total variance. This dimensionality reduction improved clustering efficiency and minimized noise.

9.2.1 K-Means Clustering

1. Technique: K-Means Clustering
2. Number of Clusters (k): 2 (selected using the Elbow Method)

3.Silhouette Score: 0.3949
4.PCA Applied: Yes — First 5 principal components used (explained variance: 93.14%)
5.Cluster Sizes:
    5.1. Cluster 0: 217,621 records
    5.2 Cluster 1: 204,094 records
6. Insights:
    6.1. Cluster 1: Higher arrest rate (22.06%), lower domestic rate (14.77%)
    6.2. Cluster 0: More domestic cases, slightly lower arrest rate (21.09%)

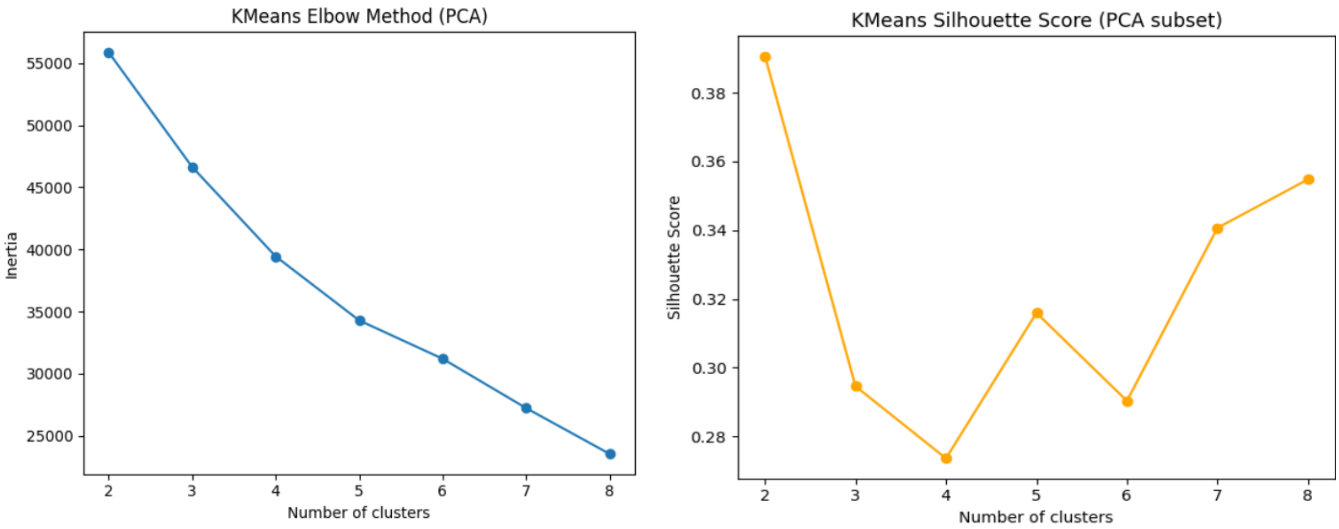


Figure 9.6:Elbow and silhouette score plots for KMeans clustering on PCA data, indicating optimal clusters at k=2.

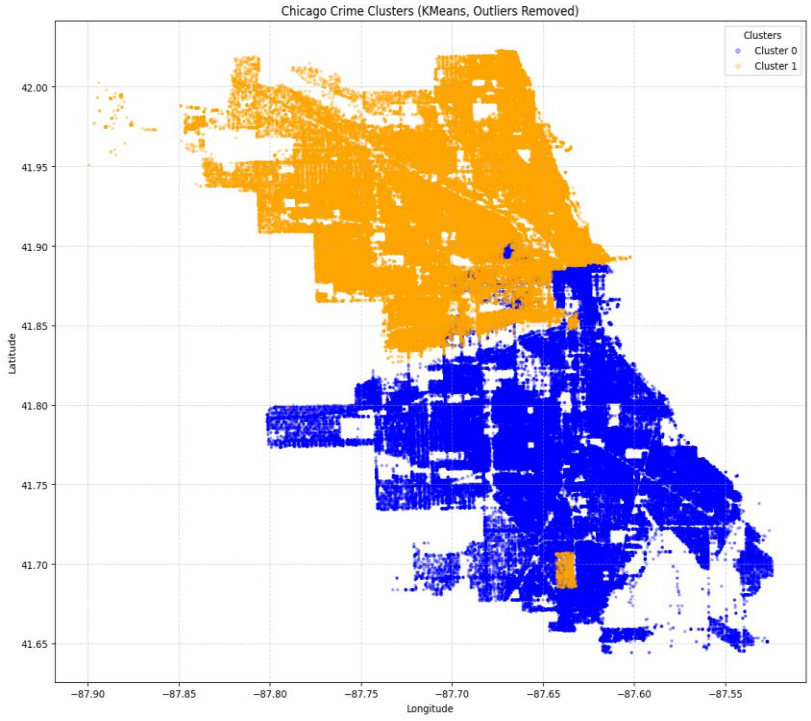


Figure 9.7: KMeans clusters mapped by raw coordinates.

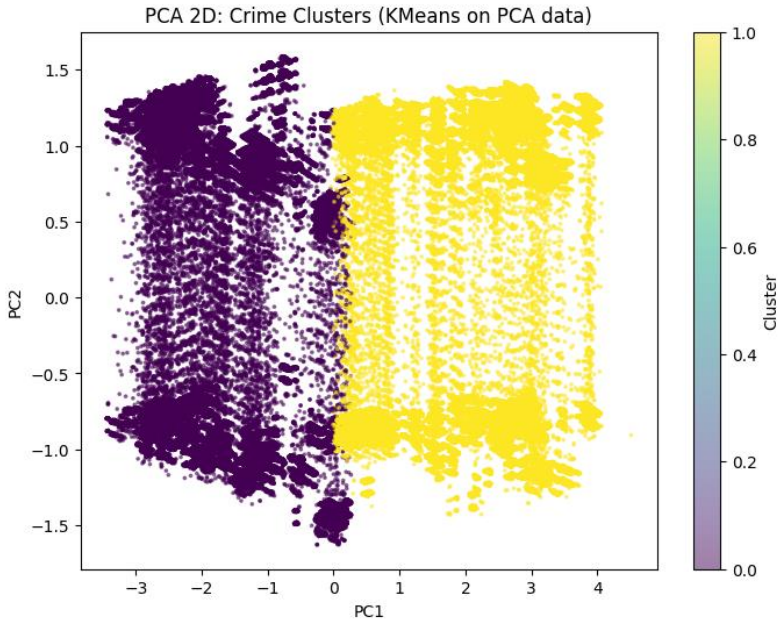


Figure 9.8: 2D PCA projection of crime data colored by KMeans cluster.

Feature	Cluster 0	Cluster 1
Arrest	0.2109	0.2206
Domestic	0.2022	0.1478
Beat	-0.6824	0.7276
District	-0.7107	0.7578
Ward	-0.6757	0.7205
Community Area	0.7397	-0.7887
Year	-0.0290	0.0309
Latitude	-0.7941	0.8467
Longitude	0.5584	-0.5954
Latitude_raw	41.7753	41.9178
Location: Vehicle - Comm. Trolley Bus	0.00000	0.00001
Location: Vehicle - Delivery Truck	0.000069	0.000083
Location: Other Ride Share (Uber, etc.)	0.000202	0.000201
Location: Vehicle Non-Commercial	0.012104	0.012386
Location: Vehicle Commercial	0.000409	0.000348
Location: Vestibule	0.000092	0.000039

Location: Warehouse	0.001195	0.001416
Location: Wooded Area	0.000023	0.000000
Location: Yard	0.000956	0.000490
Location: YMCA	0.000005	0.000010

Table 9.9: Cluster-wise Mean Values (KMeans Clustering)

9.2.2 Gaussian Mixture Model (GMM)

1. Number of Clusters (k): 4
2. Silhouette Score: 0.2307
3. PCA Applied: Yes — First 5 components used (explained variance: 93.14%)
4. Cluster Sizes & Arrest Rates:
    4.1. Cluster 0: Arrest Rate = 26.26%
    4.2. Cluster 1: Arrest Rate = 22.53%
    4.3. Cluster 2: Arrest Rate = 23.42%
    4.4. Cluster 3: Arrest Rate = 11.34%
5. Insight: GMM captures overlapping distributions; Cluster 3 shows lowest arrest rate

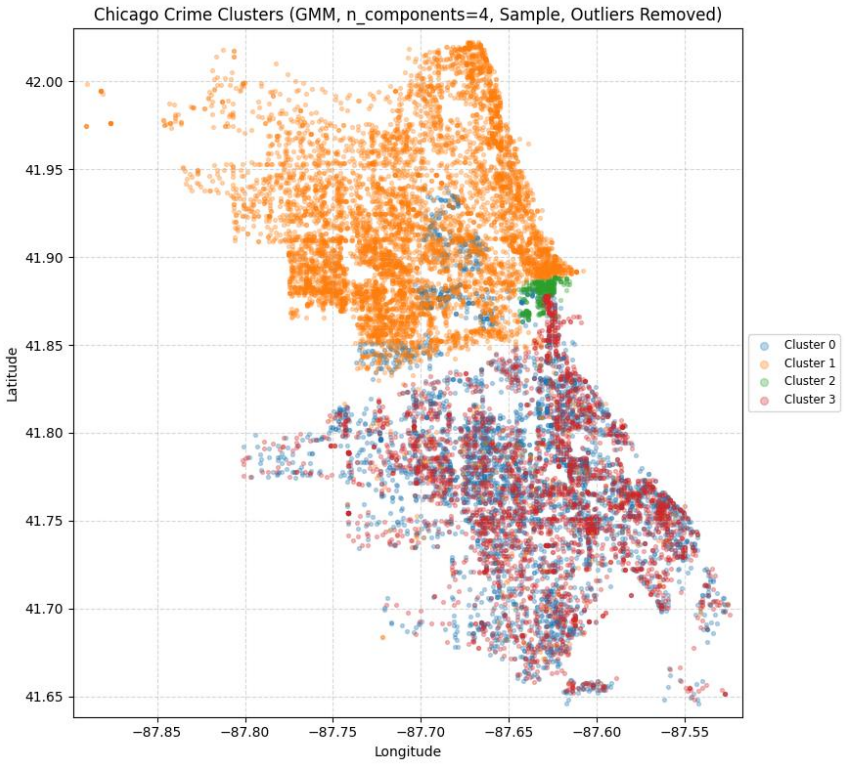


Figure 9.10: GMM-based Chicago crime clusters (sample, n=4), plotted by raw geographic coordinates.

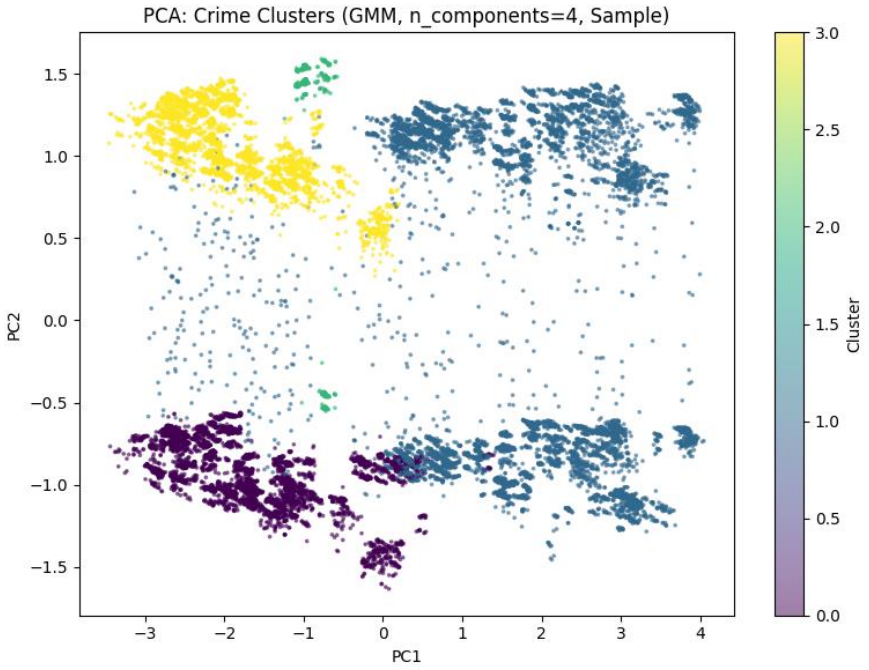


Figure 9.11:PCA-based 2D visualization of GMM crime clusters (sample, n=4).

GMM Cluster	Size	Arrest Rate	Domestic Rate	Top Crime Type	Insight
0	4413	0.263	0.203	THEFT	Typical cluster
1	7310	0.225	0.144	BATTERY	Typical cluster
2	350	0.234	0.046	CRIMINAL DAMAGE	Rarely domestic incidents
3	2927	0.113	0.221	ASSAULT	Typical cluster

Table 9.12: Cluster Profiling Summary for GMM-based Chicago Crime Clusters (Sample Data).

9.2.3 DBSCAN (Density-Based Spatial Clustering)

1. Best Parameters: eps=.5, min_samples=50
2. Clusters Formed: 35
3. Noise Points: 748
4. Silhouette Score (sample data)(excluding noise): 0.3766
5. Silhouette Score (sample data): 0.377
6. Silhouette Score (full data): 0.3606
7. Insight:
   7.1. Detects non-linear clusters and outliers
   7.2. cluster 32 shows highest arrest rate (21.1%)
   7.3. Highly sensitive to parameter tuning

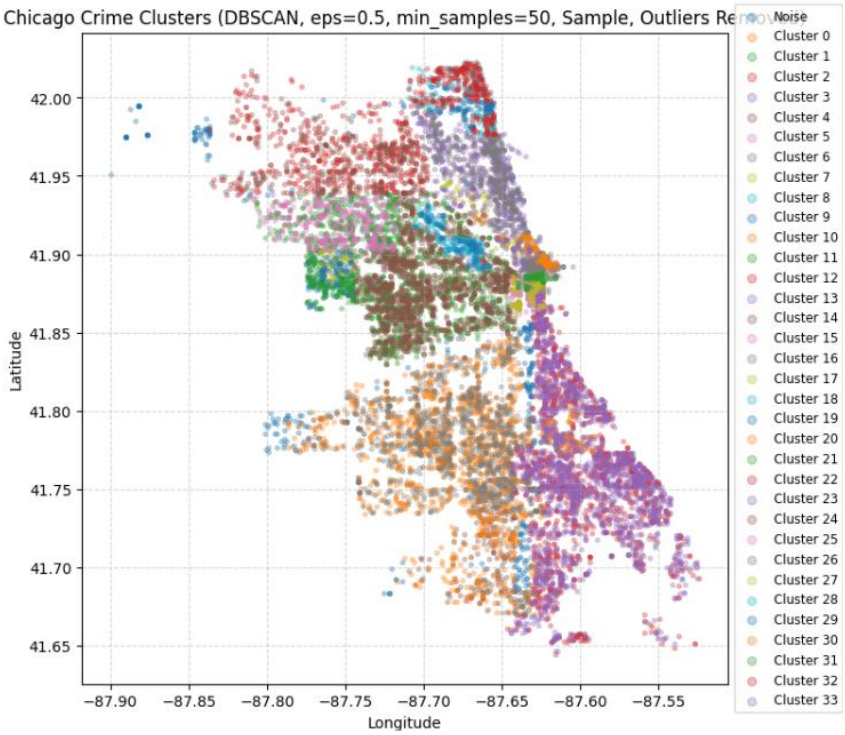


Figure 9.13: DBSCAN cluster map of Chicago crime locations (sample data)

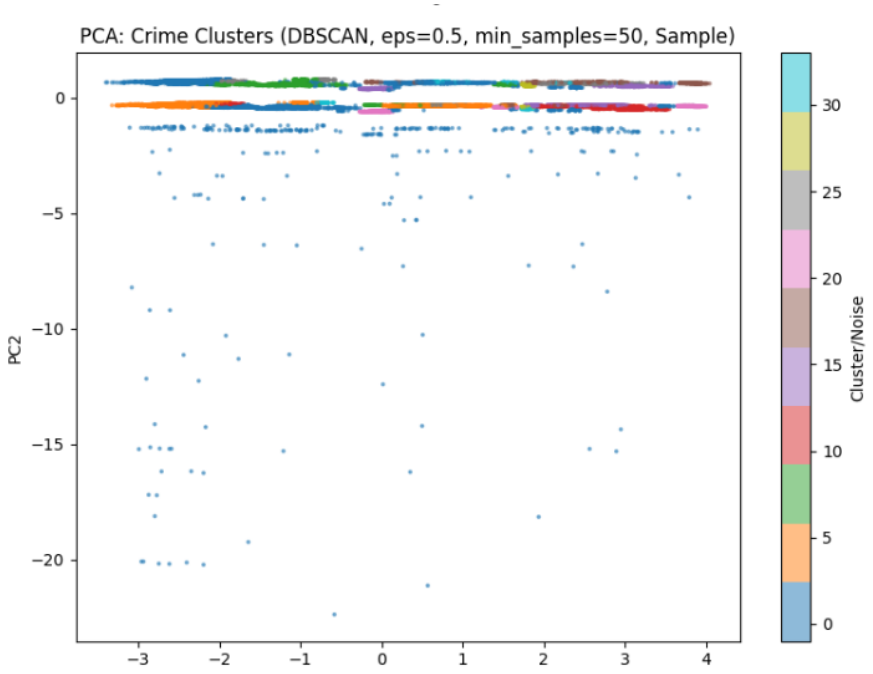


Figure 9.14: PCA plot of DBSCAN crime clusters (sample data)

eps	min_samples	clusters	noise_pts	avg_cluster_size	Silhouette

0.5	10	53	266	278.0	0.343
0.5	20	40	507	362.3	0.372
0.5	50	34	748	419.2	0.377
1.0	10	6	88	2485.3	0.113
1.0	20	6	89	2485.2	0.113
1.0	50	5	155	2969.0	0.111
1.5	10	5	54	2989.2	0.128
1.5	20	4	70	3732.5	0.128
1.5	50	4	73	3731.8	0.128

Table 9.15: DBSCAN performance on sample data across different parameter settings, showing cluster count, noise points, average cluster size, and silhouette scores.

DBSCAN_Cluster	Size	Arrest Rate	Domestic Rate	Top Crime Type	Insight
-1	748	0.156417	0.205882	THEFT	Noise/outliers: higher arrest & domestic rates
0	1345	0.101859	0.212639	BATTERY	Typical cluster
1	1088	0.182904	0.177390	NARCOTICS	Typical cluster
2	1471	0.122366	0.245411	ASSAULT	Typical cluster
3	1868	0.104390	0.224304	BATTERY	Typical cluster
4	1529	0.170700	0.170046	NARCOTICS	Typical cluster
5	206	0.194175	0.038835	DECEPTIVE PRACTICE	Rarely domestic incidents
6	1122	0.137255	0.236185	BATTERY	Typical cluster
7	173	0.196532	0.052023	THEFT	Typical cluster
8	157	0.044586	0.076433	CRIMINAL DAMAGE	Very low arrest rate
9	126	0.166667	0.341270	BATTERY	Typical cluster

10	185	0.064865	0.270270	ASSAULT	Typical cluster
11	473	0.143763	0.169133	NARCOTICS	Typical cluster
12	498	0.088353	0.170683	THEFT	Typical cluster
13	710	0.109859	0.080282	CRIMINAL DAMAGE	Typical cluster
14	162	0.154321	0.185185	BATTERY	Typical cluster
15	363	0.159780	0.206612	ASSAULT	Typical cluster
16	554	0.108303	0.079422	THEFT	Typical cluster
17	121	0.132231	0.190083	BATTERY	Typical cluster
18	174	0.132184	0.114943	ASSAULT	Typical cluster
19	117	0.102564	0.059829	THEFT	Typical cluster
20	177	0.118644	0.242938	BATTERY	Typical cluster
21	160	0.137500	0.275000	ASSAULT	Typical cluster
22	237	0.126582	0.130802	NARCOTICS	Typical cluster
23	135	0.140741	0.355556	BATTERY	Typical cluster
24	330	0.106061	0.136364	THEFT	Typical cluster
25	59	0.135593	0.033898	GAMBLING	Rarely domestic incidents
26	131	0.122137	0.030534	GAMBLING	Rarely domestic incidents
27	96	0.156250	0.291667	BATTERY	Typical cluster
28	57	0.105263	0.052632	THEFT	Typical cluster
29	67	0.134328	0.313433	BATTERY	Typical cluster
30	121	0.132231	0.024793	LIQUOR LAW VIOLATION	Rarely domestic incidents
31	99	0.181818	0.040404	DECEPTIVE PRACTICE	Rarely domestic incidents
32	52	0.211538	0.153846	THEFT	Typical cluster
33	89	0.146067	0.157303	NARCOTICS	Typical cluster

Figures 9.16:: DBSCAN cluster summary with size, arrest/domestic rates, top crime, and key insights.

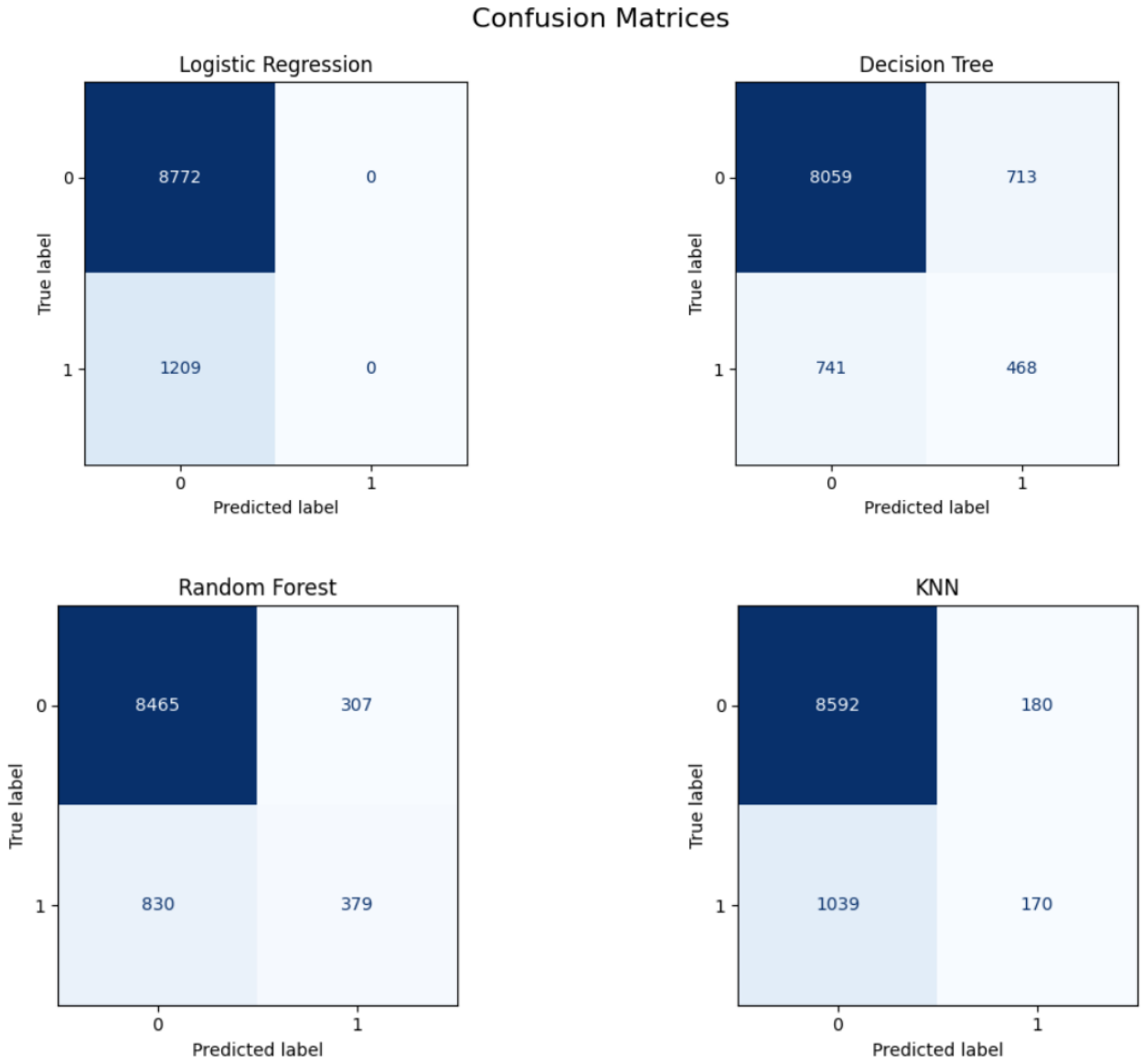
Model	Silhouette Score	Number of Clusters	Noise Handling	Interpretability	Key Strength
KMeans	0.3949 (full data)	2	No	High	Clear cluster separation; balanced cluster sizes
GMM	0.2307 (full data)	4	No	Medium	Captures soft cluster overlaps
DBSCAN	0.3606 (full data)	35	Yes	Medium	Detects noise and outliers; handles irregular cluster shapes

Table 9.17: Comparison of models and finding the best fit model

The KMeans clustering algorithm is identified as the best fit model for this dataset. It achieved the highest silhouette score of **0.3949** on the full dataset with **2 clusters**, indicating well-separated and balanced clusters that are straightforward to interpret. Although DBSCAN was effective at detecting noise and outliers and handling irregular cluster shapes, it produced a slightly lower silhouette score of **0.3606** with a large number of clusters (**35**), which makes interpretation more complex. GMM resulted in the lowest silhouette score of **0.2307** with **4 clusters**, capturing soft overlaps between clusters but providing medium interpretability. DBSCAN's noise handling capability is a key advantage, but overall, KMeans offers the best balance of cluster quality, interpretability, and model simplicity, making it the recommended choice for effective clustering and insight extraction.

10. Evaluation of Performance Metrics

10.1 supervised Learning


Confusion Matrices
Logistic Regression
8772
0
1209
0
True label
Predicted label
Decision Tree
8059
713
741
468
Random Forest
8465
307
830
379
KNN
8592
180
1039
170

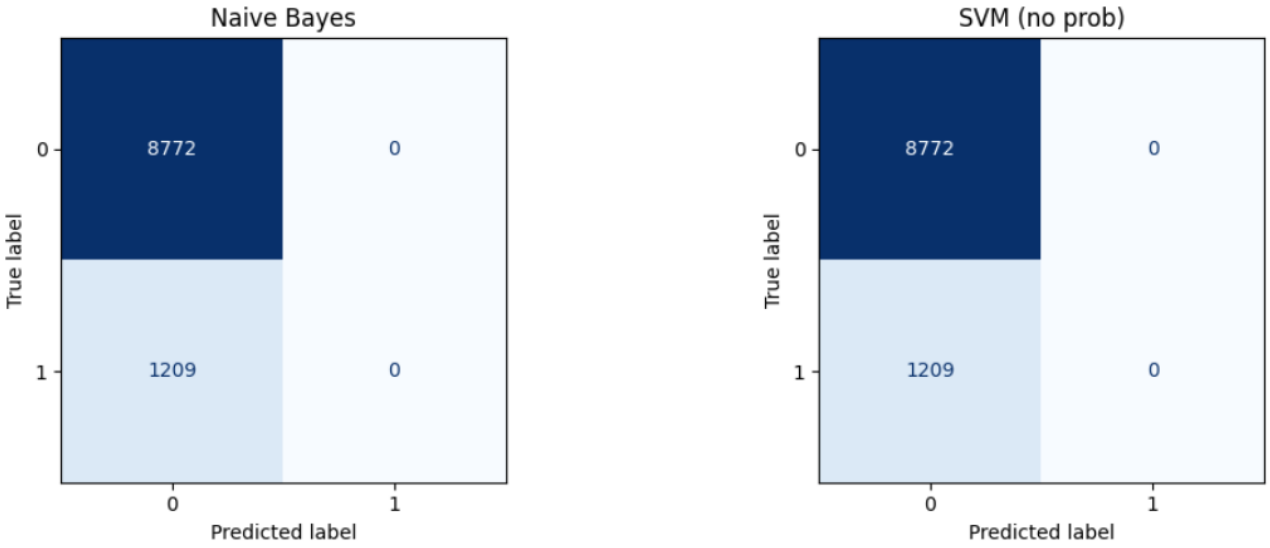


Figure 10.1: Confusion Matrix without SMOTE

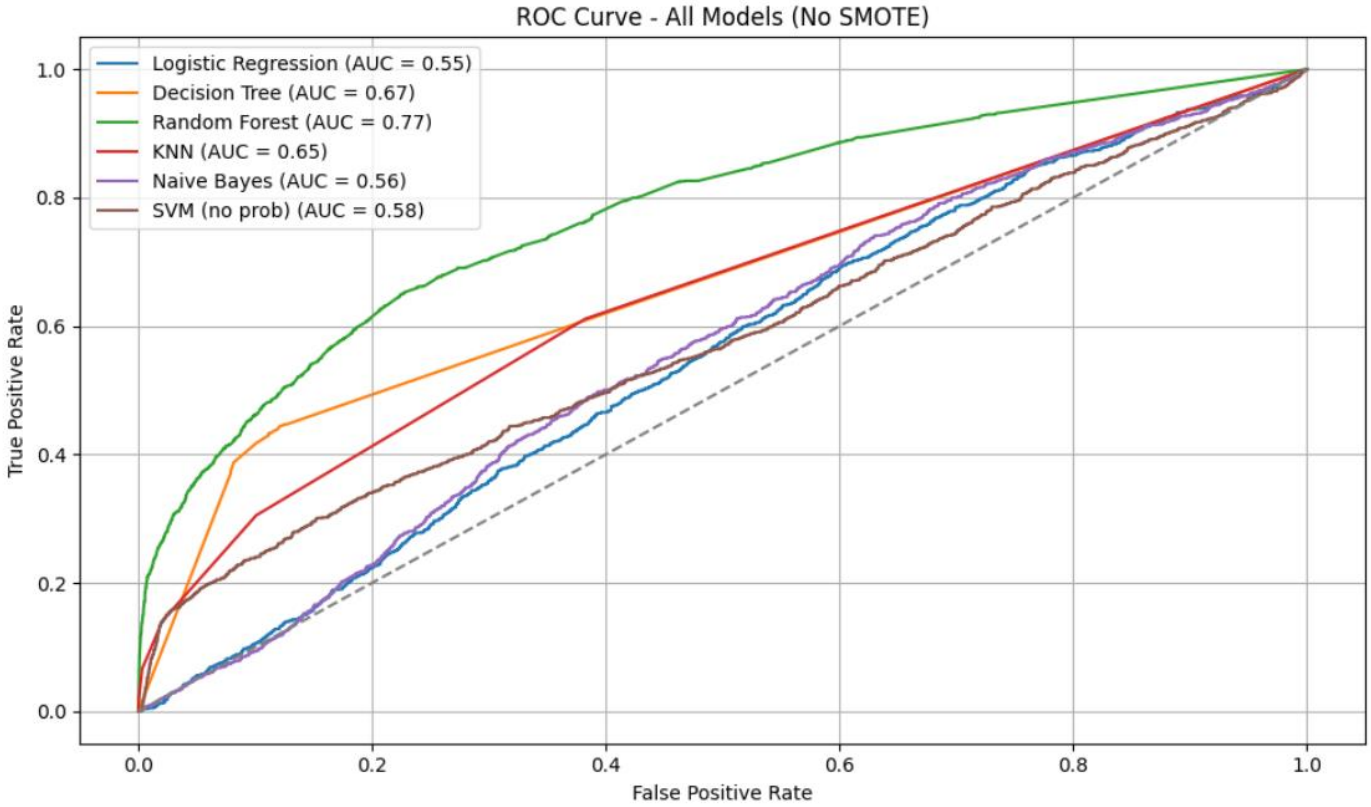


Figure 10.2: ROC Curve without SMOTE

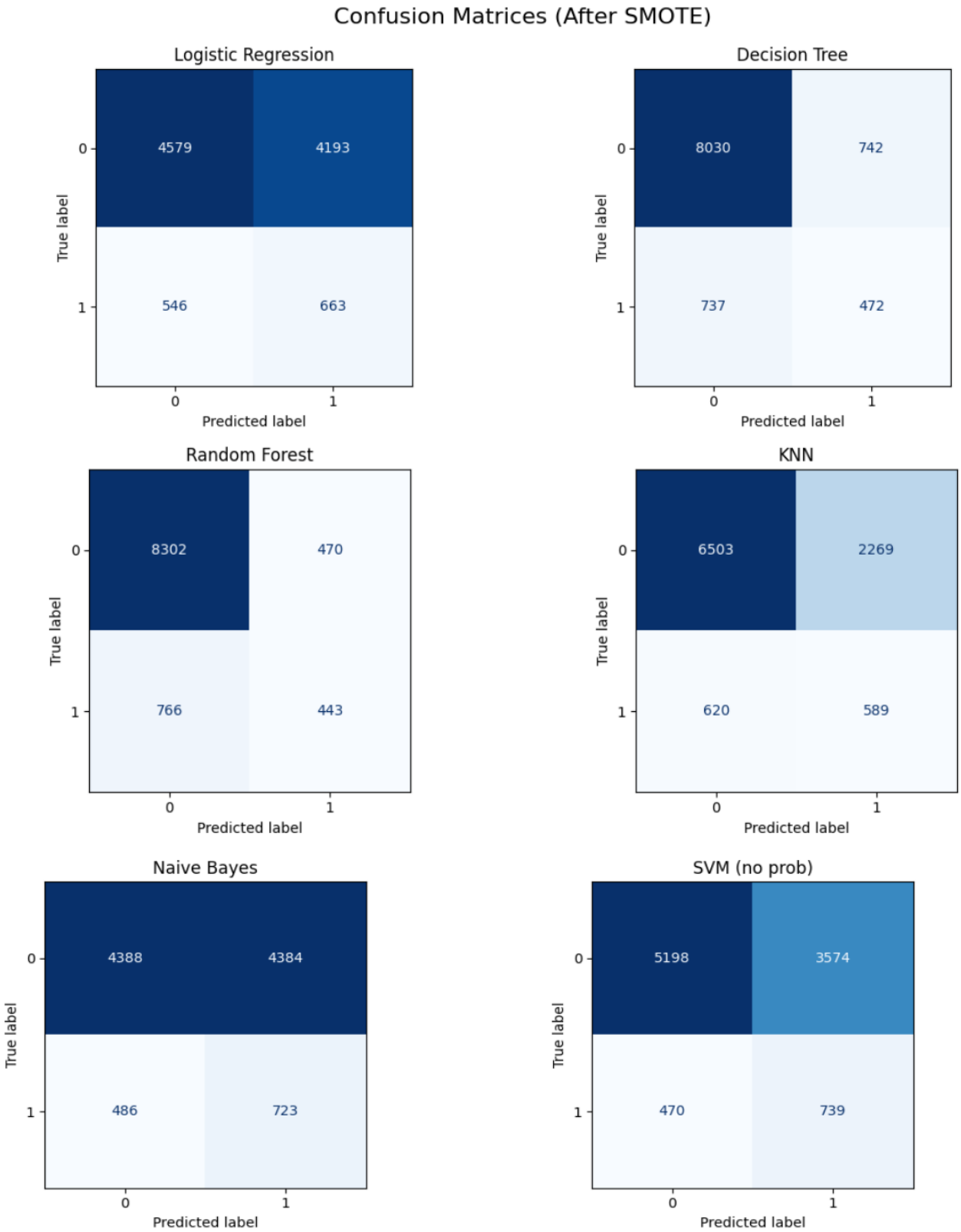


Figure 10.3: Confusion Matrix with SMOTE

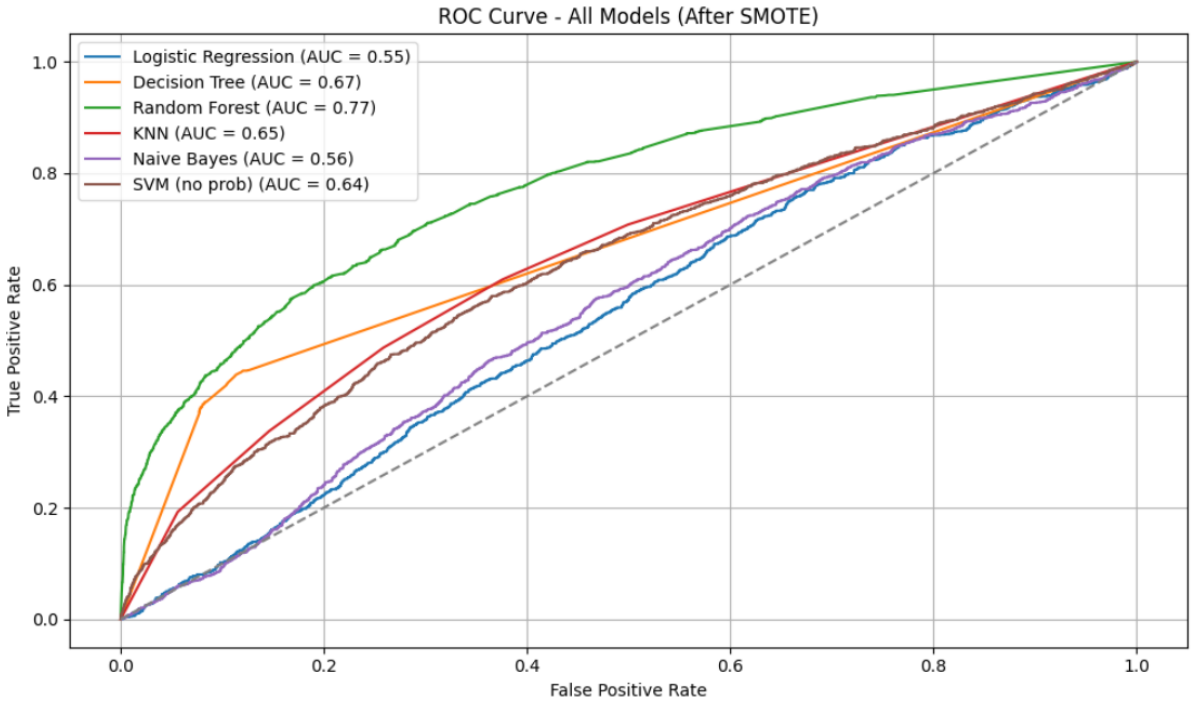


Figure 10.4: ROC Curve with SMOTE

Model	Accuracy (Before SMOTE)	Accuracy (After SMOTE)	AUC-ROC(Before SMOTE)	Best AUC-ROC (After SMOTE)
Random Forest	0.88	**0.87**	**0.77**	**0.77**
Decision Tree	0.85	0.85	0.67	0.67
Logistic Regression	0.87	0.52	0.55	0.55
SVM	0.87	0.59	0.58	0.64
Naive Bayes	0.87	0.51	0.56	0.56
KNN	0.87	0.71	0.65	0.65

Table 10.5: Accuracy and AUC-ROC curve for all algorithm before and after SMOTE

10. 2 Unsupervised Learning

The performance of the unsupervised clustering algorithms was assessed through a combination of quantitative and qualitative evaluations. The primary quantitative measure employed was the Silhouette Score, ranging from -1 to 1, which quantifies the cohesion within clusters and the separation between them.

The KMeans algorithm achieved the highest silhouette score of 0.3949 on the complete dataset, producing two well-balanced and distinctly separated clusters, indicative of robust and cohesive groupings.

The Gaussian Mixture Model (GMM) identified four clusters with a silhouette score of 0.2307, demonstrating its capacity to model overlapping clusters, albeit with reduced cluster distinctness and moderate interpretability.

Due to computational constraints, DBSCAN was evaluated on a representative sample and generated 35 clusters, attaining a silhouette score of 0.3606. This underscores DBSCAN's efficacy in detecting noise and outliers, thereby effectively capturing irregular cluster shapes, though resulting in increased complexity and medium interpretability.

Complementary qualitative analyses, considering attributes such as arrest rates, frequency of domestic incidents, and spatial distribution, further corroborated the practical relevance and meaningful differentiation of the clusters produced by each method. Overall, KMeans provided the most coherent and interpretable segmentation for this dataset.

11. Result and Inference

Result:

Method	Best Metric	Value	Key Insight
Random Forest	F1-Score (Class 1)	0.42	Balanced prediction with good recall
Decision Tree	Test Accuracy	86.16%	Good fit after tuning
KMeans	Silhouette Score	0.3949	Best-defined clusters with clear separation
DBSCAN	Silhouette Score	0.3606	Outliers captured; useful for hotspot detection
GMM	Silhouette Score	0.2307	Weak separation, but detects overlapping crimes

Table 11.1 Summary

Inference:

Random Forest is the best model for arrest prediction, achieving strong recall (F1-score: 0.42), especially after SMOTE balancing. Arrest likelihood is strongly influenced by **crime type, location, and time** — with crimes like **gambling, prostitution, and narcotics** showing the highest arrest rates.

KMeans is the most effective clustering model (silhouette score: 0.3949) due to its clear and balanced clusters. GMM, with the lowest score (0.2307), captures soft overlaps but offers weaker separation.

Crimes tend to **peak at midnight, in the evenings, and during summer months**. Overall, **Random Forest is best suited for classification**, and **KMeans is ideal for clustering-based insight extraction**.

12. Conclusion and Future Scope

Conclusion:

This project effectively analyzed Chicago crime data using both supervised and unsupervised machine learning methods.

1. Supervised Models: Random Forest and Decision Tree performed best for arrest prediction, especially after applying SMOTE. Random Forest achieved the highest F1-score and balanced accuracy.

2.Unsupervised Models:KMeans yielded the most well-defined and interpretable clusters with the highest silhouette score (0.3949), making it the most suitable for clustering-based analysis. DBSCAN was effective at detecting irregular cluster shapes and noise, achieving a respectable silhouette score (0.3606), though it required careful parameter tuning. GMM captured soft overlaps between clusters but had the lowest silhouette score (0.2307), limiting its interpretability.

Overall, combining classification and clustering enabled deeper insights into crime patterns and arrest likelihood.

Future Scope

1. Deep Learning Models:Integrate LSTM or GRU networks to analyze temporal crime patterns and forecast future incidents.

2. Geospatial Enhancements: Use Geographic Information Systems (GIS) for advanced spatial analysis and real-time crime heatmaps.

3. API Integration:Create a web-based dashboard for real-time crime prediction using trained models.

4. Crime Type-Specific Models:Build separate models for different crime categories (e.g., theft, assault) for better specialization.

5. Public Safety Recommendations:Extend findings to generate actionable insights for law enforcement, like patrol deployment optimization.

13. References

1. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830. https://scikit-learn.org/

2. VanderPlas, J. (2016). Python Data Science Handbook: Essential Tools for Working with Data. O'Reilly Media.

3. Lutz, M. (2013). Learning Python (5th ed.). O'Reilly Media.

4. Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90–95. https://matplotlib.org/

5. Waskom, M. L. (2021). Seaborn: Statistical Data Visualization. Journal of Open Source Software, 6(60), 3021. https://seaborn.pydata.org/

6. NumPy Developers. (2023). NumPy Documentation. https://numpy.org/doc/

7. Raschka, S., & Mirjalili, V. (2019). Python Machine Learning (3rd ed.). Packt Publishing.

8. Kaggle Dataset: Chicago crime dataset

Kaggle (https://www.kaggle.com/datasets/chicago/chicago-crime)

9. Wang, T., Rudin, C., Wagner, D., & Sevieri, R. (2013). Learning to detect patterns of crime. *Machine Learning and Knowledge Discovery in Databases*, 515–530. https://link.springer.com/chapter/10.1007/978-3-642-40994-3_33

10. Kianmehr, K., & Alhajj, R. (2009). Crime hotspot prediction using support vector machine and pattern mining. *Transactions in GIS, 13*(3), 243–263. https://doi.org/10.1111/j.1467-9671.2009.01163.x

11. Andresen, M. A., & Malleson, N. (2011). Testing the stability of crime patterns: Implications for theory and policy. *Journal of Research in Crime and Delinquency, 48*(1), 58–82. https://doi.org/10.1177/0022427810384136

12. Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD'96)*, 226–231. https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf

14. Appendix

Supervised Learning Algorithms Code

```python
# ==================== IMPORT LIBRARIES ====================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix,
    ConfusionMatrixDisplay, roc_curve, roc_auc_score
)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from scipy.special import expit  # sigmoid (for decision_function)


# ==================== LABEL ENCODING ====================
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# ==================== FEATURE AND TARGET ====================
X = df.drop('Arrest', axis=1)
y = df['Arrest']

# ==================== TRAIN-TEST SPLIT ====================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ==================== STANDARDIZATION ====================
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# ==================== MODELS ====================
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "SVM (no prob)": SVC(probability=False)
}

# ================== TRAINING & EVALUATION ==================
results = {}
roc_data = {}
conf_matrices = {}

for name, model in models.items():
    print(f"\n🔷 {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Probability/decision score for ROC
    try:
        if hasattr(model, "predict_proba"):
            y_proba = model.predict_proba(X_test)[:, 1]
        elif hasattr(model, "decision_function"):
            y_proba = expit(model.decision_function(X_test))
        else:
            y_proba = None
    except:
        y_proba = None

    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))

    conf_matrices[name] = confusion_matrix(y_test, y_pred)

    if y_proba is not None:
        fpr, tpr, _ = roc_curve(y_test, y_proba)
        auc = roc_auc_score(y_test, y_proba)
        roc_data[name] = (fpr, tpr, auc)
```

```python
# ==================== PLOT CONFUSION MATRICES ====================
fig, axes = plt.subplots(nrows=(len(models) + 1) // 2, ncols=2, figsize=(12, 12))
axes = axes.flatten()

for i, (name, cm) in enumerate(conf_matrices.items()):
    disp = ConfusionMatrixDisplay(cm, display_labels=np.unique(y))
    disp.plot(ax=axes[i], cmap='Blues', colorbar=False)
    axes[i].set_title(f"{name}")
    axes[i].grid(False)

# Hide empty subplot if odd number of models
if len(models) % 2 != 0:
    axes[-1].axis('off')

plt.tight_layout()
plt.suptitle("Confusion Matrices", fontsize=16, y=1.02)
plt.show()

# ==================== COMBINED ROC CURVE ====================
plt.figure(figsize=(10, 6))
for name, (fpr, tpr, auc) in roc_data.items():
    plt.plot(fpr, tpr, label=f"{name} (AUC = {auc:.2f})")

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.title("ROC Curve - All Models (No SMOTE)")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# ==================== BEST MODEL ====================
best_model = max(results, key=results.get)
print(f"\n✅ Best Model without SMOTE: {best_model} with Accuracy: {results[best_model]:.4f}")
```

```python
# ==================== IMPORT LIBRARIES ====================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix,
    ConfusionMatrixDisplay, roc_curve, roc_auc_score
)
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from scipy.special import expit
from imblearn.over_sampling import SMOTE

# ==================== LABEL ENCODING ====================
label_encoders = {}
for col in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

# ==================== FEATURE AND TARGET ====================
X = df.drop('Arrest', axis=1)
y = df['Arrest']

# ==================== TRAIN-TEST SPLIT ====================
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# ==================== STANDARDIZATION ====================
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```python
# ==================== APPLY SMOTE ====================
smote = SMOTE(random_state=42)
X_train, y_train = smote.fit_resample(X_train, y_train)

# ==================== DEFINE MODELS ====================
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier(),
    "KNN": KNeighborsClassifier(),
    "Naive Bayes": GaussianNB(),
    "SVM (no prob)": SVC(probability=False)
}

# ==================== TRAIN & EVALUATE ====================
results = {}
roc_data = {}
conf_matrices = {}

for name, model in models.items():
    print(f"\n🔷 {name}")
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    # Probability for ROC
    try:
        if hasattr(model, "predict_proba"):
            y_proba = model.predict_proba(X_test)[:, 1]
        elif hasattr(model, "decision_function"):
            y_proba = expit(model.decision_function(X_test))
        else:
            y_proba = None
    except:
        y_proba = None

    acc = accuracy_score(y_test, y_pred)
    results[name] = acc
    print(f"Accuracy: {acc:.4f}")
    print(classification_report(y_test, y_pred))
```

```
        conf_matrices[name] = confusion_matrix(y_test, y_pred)

        if y_proba is not None:
            fpr, tpr, _ = roc_curve(y_test, y_proba)
            auc = roc_auc_score(y_test, y_proba)
            roc_data[name] = (fpr, tpr, auc)

    # ================== PLOT CONFUSION MATRICES ==================
    fig, axes = plt.subplots(nrows=(len(models) + 1) // 2, ncols=2, figsize=(12, 12))
    axes = axes.flatten()

    for i, (name, cm) in enumerate(conf_matrices.items()):
        disp = ConfusionMatrixDisplay(cm, display_labels=np.unique(y))
        disp.plot(ax=axes[i], cmap='Blues', colorbar=False)
        axes[i].set_title(f"{name}")
        axes[i].grid(False)

    # Hide last empty plot if odd number
    if len(models) % 2 != 0:
        axes[-1].axis('off')

    plt.tight_layout()
    plt.suptitle("Confusion Matrices (After SMOTE)", fontsize=16, y=1.02)
    plt.show()

    # ================== PLOT COMBINED ROC ==================
    plt.figure(figsize=(10, 6))
    for name, (fpr, tpr, auc) in roc_data.items():
        plt.plot(fpr, tpr, label=f"{name} (AUC = {auc:.2f})")

    plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
    plt.title("ROC Curve - All Models (After SMOTE)")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    # ================== BEST MODEL ==================
```

Loading...

```
best_model = max(results, key=results.get)
print(f"\n✅ Best Model with SMOTE: {best_model} with Accuracy: {results[best_model]:.4f}")
```

4. Supervised Learning - Random Forest

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df['Primary Type'] = le.fit_transform(df['Primary Type'])
df['Domestic'] = df['Domestic'].astype(int)
df['Arrest'] = df['Arrest'].astype(int)
```

Test-Train Model

```
# spilt the data
X = df_clean[['Primary Type', 'Latitude', 'Longitude', 'Hour', 'Month']]
y = df_clean['Arrest']

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.3)
```

```
#train the model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred = rf_model.predict(X_test)
```

```
# Evaluate the model
# Confusion Matrix
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Arrest', 'Arrest'], yticklabels=['No Arrest', 'Arrest'])
plt.title("Random Forest Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Report and Accuracy
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Accuracy:", round(accuracy_score(y_test, y_pred) * 100, 2), "%")
```

```python
# Train Accuracy
train_acc = rf_model.score(X_train, y_train)
print("Train Accuracy:", round(train_acc * 100, 2), "%")

# Test Accuracy (you already did)
test_acc = accuracy_score(y_test, y_pred)
print("Test Accuracy:", round(test_acc * 100, 2), "%")
```

```
Train Accuracy: 99.76 %
Test Accuracy: 86.17 %
```

```python
train_acc_rf = rf_model.score(X_train, y_train)
test_acc_rf = accuracy_score(y_test, y_pred)

print("Train Accuracy (RF):", round(train_acc_rf * 100, 2), "%")
print("Test Accuracy (RF):", round(test_acc_rf * 100, 2), "%")

# Fit check
diff_rf = abs(train_acc_rf - test_acc_rf)

if train_acc_rf > 0.95 and diff_rf > 0.05:
    print("🟥 RF Model is OVERFITTING (Train accuracy is too high vs Test).")
elif train_acc_rf < 0.75 and test_acc_rf < 0.75:
    print("🟧 RF Model is UNDERFITTING (Both accuracies are low).")
elif diff_rf <= 0.05:
    print("🟩 RF Model is a GOOD FIT (Train and test accuracy are close).")
else:
    print("⚠️ RF Model may need tuning. Fit is unclear.")
```

```
Train Accuracy (RF): 99.76 %
Test Accuracy (RF): 86.17 %
🟥 RF Model is OVERFITTING (Train accuracy is too high vs Test).
```

TUNE the Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# Tuned parameters
rf_tuned = RandomForestClassifier(
    n_estimators=100,        # Number of trees
    max_depth=10,            # Limit depth (start with 10, try 8-12)
    min_samples_split=10,    # Prevent too-small splits
    random_state=42
)

rf_tuned.fit(X_train, y_train)
y_pred_tuned = rf_tuned.predict(X_test)
```

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

# Accuracy
train_acc = rf_tuned.score(X_train, y_train)
test_acc = accuracy_score(y_test, y_pred_tuned)

print("Train Accuracy:", round(train_acc * 100, 2), "%")
print("Test Accuracy:", round(test_acc * 100, 2), "%")

# Classification Report
print(classification_report(y_test, y_pred_tuned))

# Confusion Matrix
cm = confusion_matrix(y_test, y_pred_tuned)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['No Arrest', 'Arrest'], yticklabels=['No Arrest', 'Arrest'])
plt.title("Tuned Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```python
# Print accuracy results
print("Train Accuracy:", round(train_acc * 100, 2), "%")
print("Test Accuracy:", round(test_acc * 100, 2), "%")

# Decide Fit Type
diff = abs(train_acc - test_acc)

if train_acc > 0.95 and diff > 0.1:
    print("🟥 Model is OVERFITTING (High train, much lower test accuracy).")
elif train_acc < 0.75 and test_acc < 0.75:
    print("🟧 Model is UNDERFITTING (Low accuracy on both train and test).")
elif diff <= 0.05:
    print("🟩 Model is a GOOD FIT (Train and test accuracy are close).")
else:
    print("⚠️ Model is not clearly overfit or underfit. Consider tuning.")
```

```
Train Accuracy: 85.93 %
Test Accuracy: 85.52 %
🟩 Model is a GOOD FIT (Train and test accuracy are close).
```

```python
# Example input: Theft at 2AM in a certain location
new_data = pd.DataFrame({
    'Primary Type': [le.transform(['THEFT'])[0]],  # use LabelEncoder
    'Latitude': [41.88],
    'Longitude': [-87.62],
    'Hour': [2],
    'Month': [6]
})
```

```python
prediction = rf_tuned.predict(new_data)

if prediction[0]:
    print("🚨 Arrest is likely.")
else:
    print("👮 No arrest is expected.")
```

Supervised Learning - Decision Tree Classifier

```
# Train the CART model

cart_model = DecisionTreeClassifier(random_state=42)
cart_model.fit(X_train, y_train)
y_pred_cart = cart_model.predict(X_test)
```

```
#Evaluate the CART model
# Accuracy
train_acc_cart = cart_model.score(X_train, y_train)
test_acc_cart = accuracy_score(y_test, y_pred_cart)

print("CART - Train Accuracy:", round(train_acc_cart * 100, 2), "%")
print("CART - Test Accuracy:", round(test_acc_cart * 100, 2), "%")

# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred_cart))

# Confusion Matrix
cm_cart = confusion_matrix(y_test, y_pred_cart)
sns.heatmap(cm_cart, annot=True, fmt='d', cmap='Oranges', xticklabels=['No Arrest', 'Arrest'], yticklabels=['No Arrest', 'Arrest'])
plt.title("CART Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```python
# Fit analysis for original CART (before tuning)
def check_fit_before_tuning(train_acc, test_acc):
    print("\n[Before Tuning] CART Model Fit Analysis:")
    print(f"Train Accuracy: {round(train_acc * 100, 2)}%")
    print(f"Test Accuracy: {round(test_acc * 100, 2)}%")

    gap = train_acc - test_acc
    if train_acc > 0.90 and gap > 0.15:
        print("→ The model is likely **Overfitting** (high train, low test).")
    elif train_acc < 0.70 and test_acc < 0.70:
        print("→ The model is likely **Underfitting** (both train and test are low).")
    elif abs(gap) <= 0.10:
        print("→ The model has a **Good Fit** (train and test are close).")
    else:
        print("→ The model might be unbalanced or needs more analysis.")

# Call the function
check_fit_before_tuning(train_acc_cart, test_acc_cart)
```

```
[Before Tuning] CART Model Fit Analysis:
Train Accuracy: 99.76%
Test Accuracy: 79.62%
→ The model is likely **Overfitting** (high train, low test).
```

Tune the CART Model

```python
# Tuning hyperparameters
cart_tuned = DecisionTreeClassifier(
    max_depth=10,            # limit tree depth
    min_samples_split=10,    # avoid small splits
    criterion='gini',        # or try 'entropy'
    random_state=42
)

cart_tuned.fit(X_train, y_train)
y_pred_cart_tuned = cart_tuned.predict(X_test)
```

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

# Accuracy
train_acc_cart_tuned = cart_tuned.score(X_train, y_train)
test_acc_cart_tuned = accuracy_score(y_test, y_pred_cart_tuned)

print("Tuned CART - Train Accuracy:", round(train_acc_cart_tuned * 100, 2), "%")
print("Tuned CART - Test Accuracy:", round(test_acc_cart_tuned * 100, 2), "%")

# Report
print("Classification Report:\n", classification_report(y_test, y_pred_cart_tuned))

# Confusion Matrix
cm_cart_tuned = confusion_matrix(y_test, y_pred_cart_tuned)
sns.heatmap(cm_cart_tuned, annot=True, fmt='d', cmap='Oranges', xticklabels=['No Arrest', 'Arrest'], yticklabels=['No Arrest', 'Arrest'])
plt.title("Tuned CART Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```python
# Fit analysis for tuned CART (after tuning)
def check_fit_after_tuning(train_acc, test_acc):
    print("\n[After Tuning] Tuned CART Model Fit Analysis:")
    print(f"Train Accuracy: {round(train_acc * 100, 2)}%")
    print(f"Test Accuracy: {round(test_acc * 100, 2)}%")

    gap = train_acc - test_acc
    if train_acc > 0.90 and gap > 0.15:
        print("→ Still **Overfitting** – try more regularization.")
    elif train_acc < 0.70 and test_acc < 0.70:
        print("→ Still **Underfitting** – try increasing model complexity.")
    elif abs(gap) <= 0.10:
        print("→ **Good Fit Achieved** after tuning.")
    else:
        print("→ Mixed results – further analysis or tuning may help.")

# Call the function
check_fit_after_tuning(train_acc_cart_tuned, test_acc_cart_tuned)
```

```python
# Example input: THEFT occurred at 2AM in June at a specific location
new_input_cart = pd.DataFrame({
    'Primary Type': [le.transform(['THEFT'])[0]],  # Convert 'THEFT' using LabelEncoder
    'Latitude': [41.88],
    'Longitude': [-87.62],
    'Hour': [2],
    'Month': [6]
})

# Predict using the tuned CART model
prediction_cart = cart_tuned.predict(new_input_cart)

# Output result
if prediction_cart[0]:
    print("🚨 Arrest is likely (CART prediction).")
else:
    print("👮‍♂️ No arrest is expected (CART prediction).")
```

```python
from sklearn.tree import plot_tree

plt.figure(figsize=(20, 10))
plot_tree(cart_tuned,
          feature_names=X.columns,
          class_names=['No Arrest', 'Arrest'],
          filled=True,
          rounded=True,
          fontsize=10)
plt.title("Tuned Decision Tree (CART)")
plt.show()
```

Linear Regression

```python
# Target columns to analyze
high_corr_targets = ['Beat', 'District', 'X Coordinate', 'Y Coordinate', 'Latitude', 'Longitude']

# Setup grid size (3 columns)
num_plots = len(high_corr_targets)
cols = 3
rows = math.ceil(num_plots / cols)

fig, axes = plt.subplots(rows, cols, figsize=(15, 4 * rows))
axes = axes.flatten()  # Flatten in case of single row

for idx, target in enumerate(high_corr_targets):
    ax = axes[idx]

    X = df.drop(columns=[target])
    y = df[target]

    # Skip if no numeric features
    if X.shape[1] < 1:
        continue

    max_corr_col = X.corrwith(y).abs().idxmax()
    x_vals = X[max_corr_col]

    # Linear regression
    slope, intercept, r, p, std_err = stats.linregress(x_vals, y)

    def myfunc(x): return slope * x + intercept
    y_pred = list(map(myfunc, x_vals))

    # Plot
    ax.scatter(x_vals, y, label='Actual Data', alpha=0.6)
    ax.plot(x_vals, y_pred, color='red', label='Regression Line')
    ax.set_title(f'{target} vs {max_corr_col}\n$r$={r:.2f}, p={p:.1e}')
    ax.set_xlabel(max_corr_col)
    ax.set_ylabel(target)
    ax.legend()
```

```python
# Hide any unused subplots
for j in range(idx + 1, len(axes)):
    fig.delaxes(axes[j])

plt.tight_layout()
plt.show()
```

Unsupervised Learning Algorithms Code:

K-means clustering

```python
# 1. PREPROCESSING AND SETUP
data = pd.read_csv("E:\\SEM-2\\AI&ML\\chicagao excel dataset reduced 1.csv")

columns_to_drop = [
    'ID', 'Case Number', 'Date', 'Block', 'IUCR', 'Description',
    'Updated On', 'Location', 'X Coordinate', 'Y Coordinate'
]
data = data.drop(columns=columns_to_drop, errors='ignore')
data = data.fillna(data.mode().iloc[0])

# --------- OUTLIER REMOVAL BASED ON LATITUDE AND LONGITUDE ---------
lat_min, lat_max = 41.6, 42.1
long_min, long_max = -87.9, -87.5

if 'Latitude' in data.columns and 'Longitude' in data.columns:
    mask = (
        (data['Latitude'] >= lat_min) & (data['Latitude'] <= lat_max) &
        (data['Longitude'] >= long_min) & (data['Longitude'] <= long_max)
    )
    data = data[mask].copy()

# Save raw coordinates for plotting BEFORE scaling
data['Latitude_raw'] = data['Latitude']
data['Longitude_raw'] = data['Longitude']

categorical_cols = ['Primary Type', 'Location Description']
data = pd.get_dummies(data, columns=categorical_cols, drop_first=True)

for col in ['Arrest', 'Domestic']:
    if col in data.columns:
        data[col] = data[col].astype(int)

numeric_cols = ['Beat', 'District', 'Ward', 'Community Area', 'Year', 'Latitude', 'Longitude']
scaler = StandardScaler()
```

```python
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])

# Prepare numeric data for PCA and clustering
df_numeric = data.select_dtypes(include=[np.number]).fillna(data.mean(numeric_only=True))

# PCA BEFORE CLUSTERING
n_pca_components = 5
pca = PCA(n_components=n_pca_components)
df_pca = pca.fit_transform(df_numeric)
print("Explained variance ratio (PCA):", pca.explained_variance_ratio_)
print("Total explained variance by first", n_pca_components, "components:", np.sum(pca.explained_variance_ratio_))

# SUBSET FOR ELBOW & SILHOUETTE (on PCA data)
sample_size = 15000
if len(df_pca) > sample_size:
    idx = np.random.choice(len(df_pca), sample_size, replace=False)
    X_sample = df_pca[idx]
else:
    X_sample = df_pca

# ELBOW AND SILHOUETTE PLOTS
inertias = []
sil_scores = []
K = range(2, 9)
for k in K:
    km = KMeans(n_clusters=k, random_state=42, n_init=10)
    labels = km.fit_predict(X_sample)
    inertias.append(km.inertia_)
    sil_scores.append(silhouette_score(X_sample, labels))

plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(K, inertias, marker='o')
plt.title("KMeans Elbow Method (PCA)")
plt.xlabel("Number of clusters")
```

```
plt.ylabel("Inertia")
plt.subplot(1,2,2)
plt.plot(K, sil_scores, marker='o', color='orange')
plt.title("KMeans Silhouette Score (PCA subset)")
plt.xlabel("Number of clusters")
plt.ylabel("Silhouette Score")
plt.tight_layout()
plt.show()

# FINAL KMEANS ON PCA DATA
optimal_k = 2
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
data['KMeans_Cluster'] = kmeans.fit_predict(df_pca)
print("KMeans cluster counts:")
print(data['KMeans_Cluster'].value_counts())

# Silhouette score for final clustering on ALL (PCA) data
full_silhouette = silhouette_score(df_pca, data['KMeans_Cluster'])
print(f"Silhouette Score (full PCA data, K={optimal_k}): {full_silhouette:.4f}")

# MAP VISUALIZATION (use raw coordinates)
plt.figure(figsize=(12,10))
colors = ['blue', 'orange', 'green', 'red', 'purple', 'brown', 'pink', 'gray']
for idx, k in enumerate(sorted(data['KMeans_Cluster'].unique())):
    subset = data[data['KMeans_Cluster'] == k]
    plt.scatter(subset['Longitude_raw'], subset['Latitude_raw'],
                label=f"Cluster {k}", alpha=0.3, s=5, c=colors[idx % len(colors)])
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.legend(title="Clusters", loc='best', markerscale=2)
plt.title("Chicago Crime Clusters (KMeans, Outliers Removed)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()
```

```
# CLUSTER PROFILING
print(data.groupby('KMeans_Cluster').mean(numeric_only=True))
for k in sorted(data['KMeans_Cluster'].unique()):
    print(f"\n--- Cluster {k} ---")
    if 'Primary Type' in data.columns:
        print(data[data['KMeans_Cluster'] == k]['Primary Type'].value_counts().head(5))
    print("Arrest Rate:", data[data['KMeans_Cluster'] == k]['Arrest'].mean())

# PCA COMPONENT VISUALIZATION (2D)
pca2 = PCA(n_components=2)
df_pca2 = pca2.fit_transform(df_numeric)
plt.figure(figsize=(8,6))
plt.scatter(df_pca2[:,0], df_pca2[:,1], c=data['KMeans_Cluster'], cmap='viridis', alpha=0.5, s=3)
plt.title("PCA 2D: Crime Clusters (KMeans on PCA data)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label='Cluster')
plt.show()
```

Gmm Clustering code

```python
# 1. LOAD AND PREPROCESS DATA
data = pd.read_csv("E:\\SEM-2\\AI&ML\\chicagao excel dataset reduced 1.csv")
columns_to_drop = [
    'ID', 'Case Number', 'Date', 'Block', 'IUCR', 'Description',
    'Updated On', 'Location', 'X Coordinate', 'Y Coordinate'
]
data = data.drop(columns=columns_to_drop, errors='ignore')
data = data.fillna(data.mode().iloc[0])
# OUTLIER REMOVAL BASED ON LATITUDE AND LONGITUDE
lat_min, lat_max = 41.6, 42.1
lon_min, lon_max = -87.9, -87.5
if 'Latitude' in data.columns and 'Longitude' in data.columns:
    mask = (
        (data['Latitude'] >= lat_min) & (data['Latitude'] <= lat_max) &
        (data['Longitude'] >= lon_min) & (data['Longitude'] <= lon_max)
    )
    data = data[mask].copy()
# Save raw coordinates for plotting
data['Latitude_raw'] = data['Latitude']
data['Longitude_raw'] = data['Longitude']
# One-hot encode categoricals
categorical_cols = ['Primary Type', 'Location Description']
data = pd.get_dummies(data, columns=categorical_cols, drop_first=True)
# Convert binary columns
for col in ['Arrest', 'Domestic']:
    if col in data.columns:
        data[col] = data[col].astype(int)
# Scale numeric columns
numeric_cols = ['Beat', 'District', 'Ward', 'Community Area', 'Year', 'Latitude', 'Longitude']
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
# Numeric-only DataFrame for clustering
df_numeric = data.select_dtypes(include=[np.number]).fillna(data.mean(numeric_only=True))
print("Preprocessing complete and outliers removed.")
```

```python
# 2. PCA FOR DIMENSIONALITY REDUCTION (before clustering)
n_pca_components = 5
pca = PCA(n_components=n_pca_components)
df_pca = pca.fit_transform(df_numeric)
print("Explained variance ratio (PCA):", pca.explained_variance_ratio_)
print("Total explained variance by first", n_pca_components, "components:", np.sum(pca.explained_variance_ratio_))

# 3. PREPARE SAMPLE FOR GMM FITTING
sample_size = 15000
if len(df_pca) > sample_size:
    idx = np.random.choice(len(df_pca), sample_size, replace=False)
    X_sample = df_pca[idx]
    data_sample = data.iloc[idx].copy()
else:
    X_sample = df_pca.copy()
    data_sample = data.copy()

# 4. GAUSSIAN MIXTURE MODEL (GMM) ON SAMPLE
n_components = 4
gmm = GaussianMixture(n_components=n_components, random_state=42)
gmm_labels = gmm.fit_predict(X_sample)
data_sample['GMM_Cluster'] = gmm_labels

# Silhouette score ON SAMPLE
gmm_sil = silhouette_score(X_sample, gmm_labels)
print(f"GMM Silhouette Score (sample, k={n_components}): {gmm_sil:.4f}")
```

```python
# 5. MAP VISUALIZATION (SAMPLE, using raw coordinates)
plt.figure(figsize=(10, 8))
ax = plt.gca()
for k in sorted(data_sample['GMM_Cluster'].unique()):
    subset = data_sample[data_sample['GMM_Cluster'] == k]
    plt.scatter(subset['Longitude_raw'], subset['Latitude_raw'], label=f"Cluster {k}", alpha=0.3, s=8)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title(f"Chicago Crime Clusters (GMM, n_components={n_components}, Sample, Outliers Removed)")
plt.grid(True, linestyle='--', alpha=0.5)

lon_min, lon_max = data_sample['Longitude_raw'].min(), data_sample['Longitude_raw'].max()
lat_min, lat_max = data_sample['Latitude_raw'].min(), data_sample['Latitude_raw'].max()
lon_buffer = (lon_max - lon_min) * 0.02
lat_buffer = (lat_max - lat_min) * 0.02
plt.xlim(lon_min - lon_buffer, lon_max + lon_buffer)
plt.ylim(lat_min - lat_buffer, lat_max + lat_buffer)
ax.set_aspect('equal', adjustable='box')
plt.legend(markerscale=2, fontsize='small', loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()

# 6. PCA VISUALIZATION (SAMPLE)
pca2 = PCA(n_components=2)
pca_components = pca2.fit_transform(X_sample)
plt.figure(figsize=(8, 6))
plt.scatter(pca_components[:, 0], pca_components[:, 1], c=gmm_labels, cmap='viridis', alpha=0.5, s=3)
plt.title(f"PCA: Crime Clusters (GMM, n_components={n_components}, Sample)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label='Cluster')
plt.tight_layout()
plt.show()
```

```python
table = cluster_insight_table(data_sample, cluster_col='GMM_Cluster')
print(table)
for k in sorted(data_sample['GMM_Cluster'].unique()):
    print(f"\n--- GMM Cluster {k} ---")
    if 'Primary Type' in data_sample.columns:
        print(data_sample[data_sample['GMM_Cluster'] == k]['Primary Type'].value_counts().head(5))
    print("Arrest Rate:", data_sample[data_sample['GMM_Cluster'] == k]['Arrest'].mean())
# 8. GMM ON FULL DATA
gmm_full = GaussianMixture(n_components=n_components, random_state=42)
gmm_labels_full = gmm_full.fit_predict(df_pca)
data['GMM_Cluster'] = gmm_labels_full
# Silhouette score on full data
gmm_full_sil = silhouette_score(df_pca, gmm_labels_full)
print(f"GMM Silhouette Score (FULL DATA, k={n_components}): {gmm_full_sil:.4f}")
```

DBSCAN CODE:

```python
# 1. LOAD AND PREPROCESS DATA
data = pd.read_csv("E:\\SEM-2\\AI&ML\\chicagao excel dataset reduced 1.csv")

columns_to_drop = [
    'ID', 'Case Number', 'Date', 'Block', 'IUCR', 'Description',
    'Updated On', 'Location', 'X Coordinate', 'Y Coordinate'
]
data = data.drop(columns=columns_to_drop, errors='ignore')
data = data.fillna(data.mode().iloc[0])
# OUTLIER REMOVAL BASED ON RAW LAT/LON (Chicago bounds)
lat_min, lat_max = 41.6, 42.1
lon_min, lon_max = -87.9, -87.5
if 'Latitude' in data.columns and 'Longitude' in data.columns:
    mask = (
        (data['Latitude'] >= lat_min) & (data['Latitude'] <= lat_max) &
        (data['Longitude'] >= lon_min) & (data['Longitude'] <= lon_max)
    )
    data = data[mask].copy()
# Save raw coordinates for plotting
data['Latitude_raw'] = data['Latitude']
data['Longitude_raw'] = data['Longitude']
# One-hot encode categoricals
categorical_cols = ['Primary Type', 'Location Description']
data = pd.get_dummies(data, columns=categorical_cols, drop_first=True)
# Convert binary columns
for col in ['Arrest', 'Domestic']:
    if col in data.columns:
        data[col] = data[col].astype(int)
# Scale numeric columns
numeric_cols = ['Beat', 'District', 'Ward', 'Community Area', 'Year', 'Latitude', 'Longitude']
scaler = StandardScaler()
data[numeric_cols] = scaler.fit_transform(data[numeric_cols])
# Numeric-only DataFrame for PCA and clustering
df_numeric = data.select_dtypes(include=[np.number]).fillna(data.mean(numeric_only=True))
print("Preprocessing complete and outliers removed.")
```

```python
# 2. PCA FOR DIMENSIONALITY REDUCTION (before clustering)
n_pca_components = 5
pca = PCA(n_components=n_pca_components)
df_pca = pca.fit_transform(df_numeric)
print("Explained variance ratio (PCA):", pca.explained_variance_ratio_)
print("Total explained variance by first", n_pca_components, "components:", np.sum(pca.explained_variance_ratio_)
# 3. PREPARE SAMPLE FOR PARAMETER SEARCH
sample_size = 15000
if len(df_pca) > sample_size:
    idx = np.random.choice(len(df_pca), sample_size, replace=False)
    X_sample = df_pca[idx]
    data_sample = data.iloc[idx].copy()
else:
    X_sample = df_pca.copy()
    data_sample = data.copy()
# 4. PARAMETER GRID SEARCH FOR DBSCAN ON PCA DATA
eps_values = [0.5, 1.0, 1.5]
min_samples_values = [10, 20, 50]
print("eps\tmin_samples\tclusters\tnoise_pts\tavg_cluster_size\tSilhouette")
results = []
for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        db_labels = dbscan.fit_predict(X_sample)
        n_clusters = len(set(db_labels)) - (1 if -1 in db_labels else 0)
        n_noise = list(db_labels).count(-1)
        cluster_sizes = [list(db_labels).count(i) for i in set(db_labels) if i != -1]
        avg_cluster_size = np.mean(cluster_sizes) if cluster_sizes else 0
        # Silhouette (if >1 cluster and not all noise)
        if n_clusters > 1 and np.sum(db_labels != -1) > 1:
            mask = db_labels != -1
            sil = silhouette_score(X_sample[mask], db_labels[mask])
        else:
            sil = float('nan')
        print(f"{eps}\t{min_samples}\t\t{n_clusters}\t\t{n_noise}\t\t{avg_cluster_size:.1f}\t\t{sil:.3f}")
        results.append((eps, min_samples, n_clusters, n_noise, avg_cluster_size, sil, db_labels))
# 5. CHOOSE BEST PARAMETERS
best_eps = 1.0
best_min_samples = 10
# 6. FINAL DBSCAN FIT ON SAMPLE
dbscan = DBSCAN(eps=best_eps, min_samples=best_min_samples)
db_labels = dbscan.fit_predict(X_sample)
data_sample['DBSCAN_Cluster'] = db_labels
# Silhouette score (excluding noise) for sample
mask = db_labels != -1
if np.sum(mask) > 1 and len(set(db_labels[mask])) > 1:
    dbscan_sil = silhouette_score(X_sample[mask], db_labels[mask])
    print(f"DBSCAN Silhouette Score (sample, excluding noise): {dbscan_sil:.4f}")
else:
    print("DBSCAN: Not enough clusters for silhouette score.")
```

```python
# 7. MAP VISUALIZATION (SAMPLE, using raw coordinates)
plt.figure(figsize=(8, 8))
ax = plt.gca()
for k in sorted(data_sample['DBSCAN_Cluster'].unique()):
    subset = data_sample[data_sample['DBSCAN_Cluster'] == k]
    label = f"Cluster {k}" if k != -1 else "Noise"
    plt.scatter(subset['Longitude_raw'], subset['Latitude_raw'], label=label, alpha=0.3, s=8)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title(f"Chicago Crime Clusters (DBSCAN, eps={best_eps}, min_samples={best_min_samples}, Sample, Outliers Removed)")
plt.grid(True, linestyle='--', alpha=0.5)
ax.set_aspect('equal', adjustable='box')
plt.legend(markerscale=2, fontsize='small', loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()

# 8. PCA VISUALIZATION (SAMPLE)
pca2 = PCA(n_components=2)
pca_components = pca2.fit_transform(X_sample)
plt.figure(figsize=(8, 6))
plt.scatter(pca_components[:,0], pca_components[:,1], c=db_labels, cmap='tab10', alpha=0.5, s=3)
plt.title(f"PCA: Crime Clusters (DBSCAN, eps={best_eps}, min_samples={best_min_samples}, Sample)")
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.colorbar(label='Cluster/Noise')
plt.tight_layout()
plt.show()
```

```python
# 9. CLUSTER PROFILING (SAMPLE)
def cluster_insight_table(data_sample, cluster_col='DBSCAN_Cluster'):
    # Get cluster sizes
    cluster_sizes = data_sample[cluster_col].value_counts().sort_index()
    # Get arrest rates
    arrest_rates = data_sample.groupby(cluster_col)['Arrest'].mean()
    # Get domestic rates
    domestic_rates = data_sample.groupby(cluster_col)['Domestic'].mean()
    # Most common crime type (if present)
    if 'Primary Type' in data_sample.columns:
        top_crime = data_sample.groupby(cluster_col)['Primary Type'].agg(lambda x: x.value_counts().index[0])
    else:
        top_crime = pd.Series('-', index=cluster_sizes.index)
    # Build summary DataFrame
    summary = pd.DataFrame({
        'Size': cluster_sizes,
        'Arrest Rate': arrest_rates,
        'Domestic Rate': domestic_rates,
        'Top Crime Type': top_crime
    })

    def make_insight(row):
        if row.name == -1:
            return "Noise/outliers: higher arrest & domestic rates, likely atypical incidents"
        elif row['Size'] < 10:
            return "Tiny cluster: may be rare or special-case events"
        elif row['Arrest Rate'] > 0.8:
            return "Very high arrest rate"
        elif row['Arrest Rate'] < 0.05:
            return "Very low arrest rate"
        elif row['Domestic Rate'] > 0.7:
            return "Mostly domestic incidents"
        elif row['Domestic Rate'] < 0.05:
            return "Rarely domestic incidents"
        else:
            return "Typical cluster"
    summary['Insight'] = summary.apply(make_insight, axis=1)
    return summary

table = cluster_insight_table(data_sample, cluster_col='DBSCAN_Cluster')
print(table)

for k in sorted(set(db_labels)):
    print(f"\n--- DBSCAN Cluster {k} ---")
    if 'Primary Type' in data_sample.columns:
        print(data_sample[data_sample['DBSCAN_Cluster'] == k]['Primary Type'].value_counts().head(5))
    print("Arrest Rate:", data_sample[data_sample['DBSCAN_Cluster'] == k]['Arrest'].mean())
```

```python
# 10. FIT DBSCAN ON FULL DATASET (PCA) AND CALCULATE SILHOUETTE SCORE
dbscan_full = DBSCAN(eps=best_eps, min_samples=best_min_samples)
db_labels_full = dbscan_full.fit_predict(df_pca)
data['DBSCAN_Cluster'] = db_labels_full

mask_full = db_labels_full != -1
if np.sum(mask_full) > 1 and len(set(db_labels_full[mask_full])) > 1:
    dbscan_full_sil = silhouette_score(df_pca[mask_full], db_labels_full[mask_full])
    print(f"DBSCAN Silhouette Score (FULL DATA, excluding noise): {dbscan_full_sil:.4f}")
else:
    print("DBSCAN FULL: Not enough clusters for silhouette score.")
```