DATA MINING PROJECT

Teams IDs

1. ZENAB OSAMA 2305289
2. MALAK MOHAMED 2305303
3. MENNA WALID 2305304
4. LOGINA MAHMOUD 2305532
5. MARIAM AHMED 2305300
6. AHMED SADEK 2305355

```python
# Importing necessary libraries for data manipulation, visualization, and model building
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, LabelEncoder ,StandardScaler
from sklearn_extra.cluster import KMedoids
import skfuzzy as fuzz
from skfuzzy import control as ctrl
from sklearn.datasets import make_blobs
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report ,pairwise_distances_argmin_min
from deap import base, creator, tools, algorithms
from sklearn.tree import DecisionTreeClassifier
from sklearn.cluster import AgglomerativeClustering
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from scipy.cluster.hierarchy import dendrogram, linkage
```

Purpose: Imports essential libraries
each library for an algorithm in this code

```python
# Loading the dataset
df = pd.read_csv(r"d:\Semester Four\Data Mining\train.csv\train.csv")
```

Generate + Code + Markdown

```python
df.head(10) # Displaying the first few rows of the dataset to understand its structure
```

```python
df.shape # Checking the dimensions of the dataset (rows, columns)
```

```
(878049, 9)
```

```python
df.describe()
```

```python
df.info()  # Checking dataset information such as column names, data types, and non-null counts
```

The Output

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 878049 entries, 0 to 878048
Data columns (total 9 columns):
 #   Column      Non-Null Count   Dtype  
---  ------      --------------   -----  
 0   Dates       878049 non-null  object 
 1   Category    878049 non-null  object 
 2   Descript    878049 non-null  object 
 3   DayOfWeek   878049 non-null  object 
 4   PdDistrict  878049 non-null  object 
 5   Resolution  878049 non-null  object 
 6   Address     878049 non-null  object 
 7   X           878049 non-null  float64
 8   Y           878049 non-null  float64
dtypes: float64(2), object(7)
memory usage: 60.3+ MB
```

This code begins by loading a CSV file containing the dataset using pandas.read_csv() and stores it in a DataFrame named df. It then uses df.head(10) to display the first 10 rows, giving a quick preview of the dataset's structure. The command df.shape returns the dimensions of the dataset, showing that it has 878,049 rows and 9 columns. The df.describe() function provides statistical summaries (like mean, min, max, and standard deviation) for the numeric columns. Finally, df.info() prints detailed information about each column, including data types and the number of non-null values, helping to assess data quality and identify any missing data.

```
df.isnull().sum()  # Check for missing values
```

```
Dates         0
Category      0
Descript      0
DayOfWeek     0
PdDistrict    0
Resolution    0
Address       0
X             0
Y             0
dtype: int64
```

```
df.duplicated().sum()
```

```
np.int64(2323)
```

Generate

```
df.drop_duplicates(inplace=True) # Removing duplicate rows from the dataset
```

```
# Checking if there are any remaining duplicate rows
df.duplicated().sum()
```

```
np.int64(0)
```

For cleaning the data

- firstly count the null and duplicates
- then remove them

```python
df.head() #re-check for data
```

```python
numerical_features = ['X', 'Y', 'Hour', 'Month', 'Year', 'DayOfYear']
print("Quantile Analysis (5%, 25%, 50%, 75%, 95%):")
```

```
Quantile Analysis (5%, 25%, 50%, 75%, 95%):
```

```python
df.shape #re-check for dimensions
```

```
(875726, 9)
```

```python
# Investigate the unique values in the categorical columns
# Check unique values in 'Category' (target variable)
print("\nUnique Categories:")
print(df['Category'].unique())
```

```
Unique Categories:
['WARRANTS' 'OTHER OFFENSES' 'LARCENY/THEFT' 'VEHICLE THEFT' 'VANDALISM'
 'NON-CRIMINAL' 'ROBBERY' 'ASSAULT' 'WEAPON LAWS' 'BURGLARY'
 'SUSPICIOUS OCC' 'DRUNKENNESS' 'FORGERY/COUNTERFEITING' 'DRUG/NARCOTIC'
 'STOLEN PROPERTY' 'SECONDARY CODES' 'TRESPASS' 'MISSING PERSON' 'FRAUD'
 'KIDNAPPING' 'RUNAWAY' 'DRIVING UNDER THE INFLUENCE'
 'SEX OFFENSES FORCIBLE' 'PROSTITUTION' 'DISORDERLY CONDUCT' 'ARSON'
 'FAMILY OFFENSES' 'LIQUOR LAWS' 'BRIBERY' 'EMBEZZLEMENT' 'SUICIDE'
 'LOITERING' 'SEX OFFENSES NON FORCIBLE' 'EXTORTION' 'GAMBLING'
 'BAD CHECKS' 'TREA' 'RECOVERED VEHICLE' 'PORNOGRAPHY/OBSCENE MAT']
```

This block of code performs a detailed exploratory data analysis on the dataset. It starts by using df.head() to re-display the top rows of the DataFrame for verification. A list of numerical features (X, Y, Hour, Month, Year, DayOfYear) is defined, likely for future statistical or modeling use. The df.shape command is repeated to confirm the current dimensions of the data (875,726 rows and 9 columns). Then, the script investigates the categorical target variable Category by printing all unique crime types recorded in the dataset using df['Category'].unique(). This helps to understand the variety of crime labels present, which are essential for classification tasks in data mining.

```python
# Check unique values in 'DayOfWeek'
print("\nUnique Days of the Week:")
print(df['DayOfWeek'].unique())
```

```
Unique Days of the Week:
['Wednesday' 'Tuesday' 'Monday' 'Sunday' 'Saturday' 'Friday' 'Thursday']
```

```python
# Check unique values in 'PdDistrict'
print("\nUnique Police Districts:")
print(df['PdDistrict'].unique())
```

```
Unique Police Districts:
['NORTHERN' 'PARK' 'INGLESIDE' 'BAYVIEW' 'RICHMOND' 'CENTRAL' 'TARAVAL'
 'TENDERLOIN' 'MISSION' 'SOUTHERN']
```

```python
# Check if geospatial data (X, Y) is valid or within expected ranges
print("\nGeospatial Data Summary (X, Y coordinates):")
print(df[['X', 'Y']].describe())
```

```
Geospatial Data Summary (X, Y coordinates):
                X              Y
count  875726.000000  875726.000000
mean     -122.422623      37.771032
std         0.030363       0.457497
min      -122.513642      37.707879
25%      -122.432952      37.752427
50%      -122.416446      37.775421
75%      -122.406959      37.784380
max      -120.500000      90.000000
```

The code summarizes data by listing unique days and police districts, and provides statistical details of geospatial coordinates, revealing a likely outlier in the Y (latitude) values.

```python
# Normalization (Scaling) of numerical features (X and Y)
scaler = MinMaxScaler()
df[['X', 'Y']] = scaler.fit_transform(df[['X', 'Y']])
```

```python
# Display the dataframe after scaling
df.head()
```

```python
# Split the data into features (X) and target (y)
X = df.drop(columns=['Category'])  # All features except 'Category'
y = df['Category']  # The target variable is 'Category'

# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Check the shape of the resulting datasets
print(f"Training data (X_train) shape: {X_train.shape}")
print(f"Testing data (X_test) shape: {X_test.shape}")
print(f"Training labels (y_train) shape: {y_train.shape}")
print(f"Testing labels (y_test) shape: {y_test.shape}")
```

```
Training data (X_train) shape: (700580, 8)
Testing data (X_test) shape: (175146, 8)
Training labels (y_train) shape: (700580,)
Testing labels (y_test) shape: (175146,)
```

This code normalizes the geospatial features (X, Y) using MinMaxScaler, then splits the dataset into features (X) and target labels (y, the Category column). It further divides the data into training (80%) and testing (20%) sets, confirming that each set contains 8 features per sample and a total of 875,726 records.

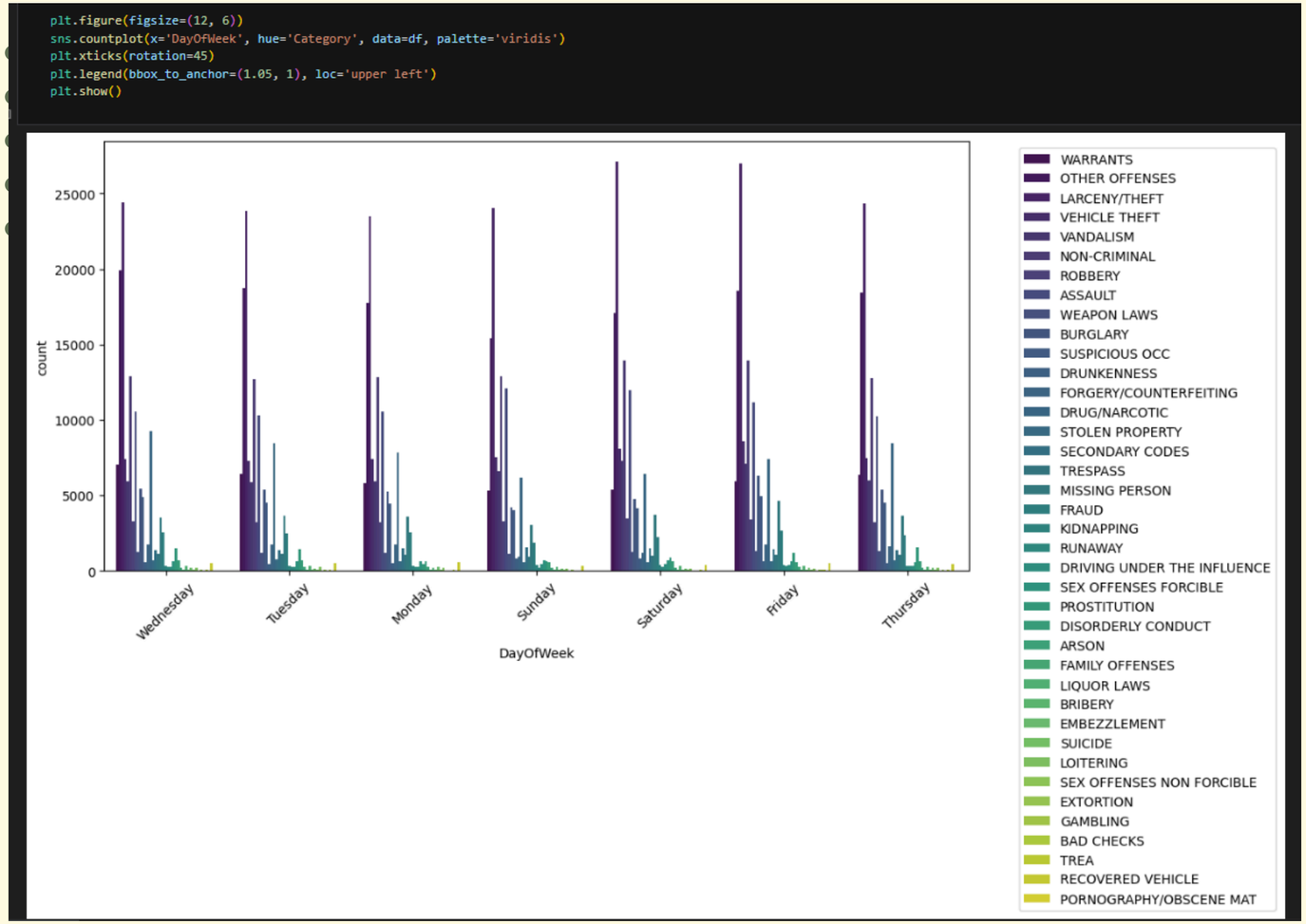
4o

This code converts the Dates column to datetime format, extracts the hour into a new Hour column, and then previews the result. Finally, it uses a crosstab to visualize the frequency of each crime Category by DayOfWeek using a color gradient for better readability.

```
# Convert 'Dates' column to datetime
df['Dates'] = pd.to_datetime(df['Dates'])

# Extract the hour
df['Hour'] = df['Dates'].dt.hour

# Check
df[['Dates', 'Hour']].head()
```

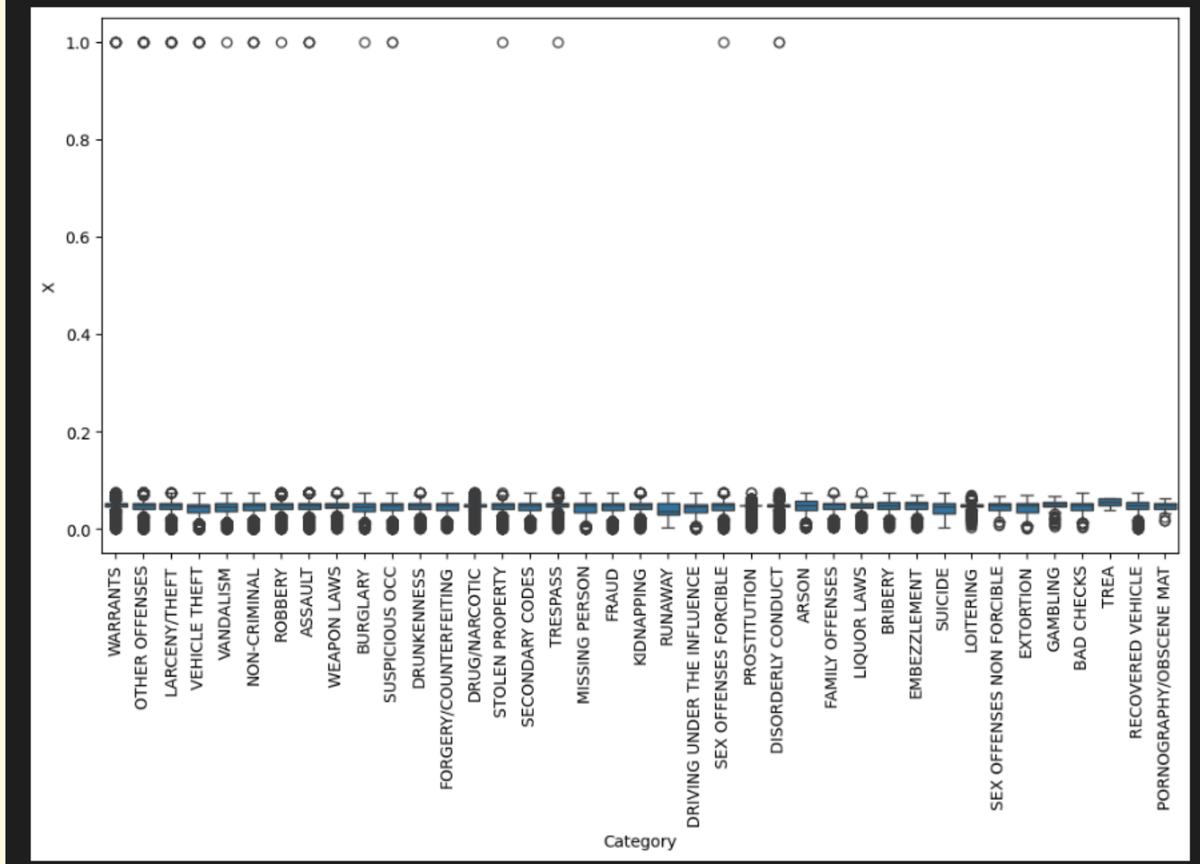
```
# Example: Crime Category vs. Day of Week
pd.crosstab(df['Category'], df['DayOfWeek']).style.background_gradient(cmap='Oranges')
```

```
plt.figure(figsize=(12, 6))
sns.countplot(x='DayOfWeek', hue='Category', data=df, palette='viridis')
plt.xticks(rotation=45)
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.show()
```



- Purpose: Shows the distribution of different crime categories across the days of the week.
- Insight: Helps identify which types of crimes are more frequent on specific days.

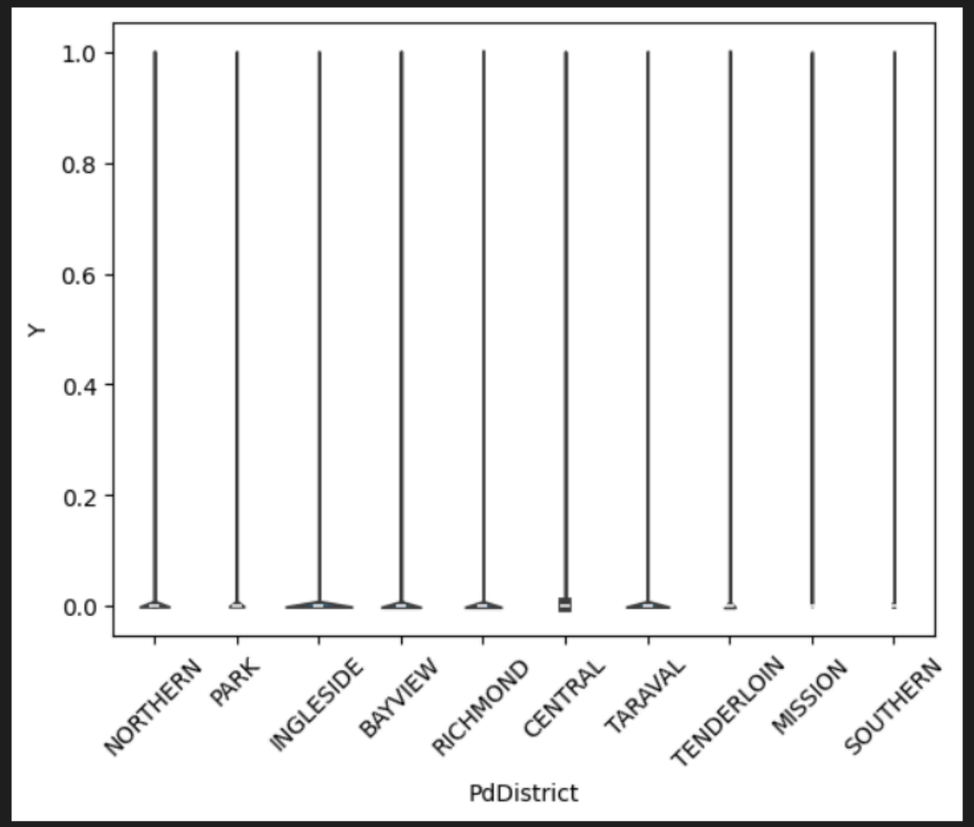
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='Category', y='X', data=df)  # Longitude distribution per crime type
plt.xticks(rotation=90)
plt.show()
```



- Purpose: Visualizes the spread of crime locations (longitude) per category.
- Insight: Useful for detecting location-based crime trends.

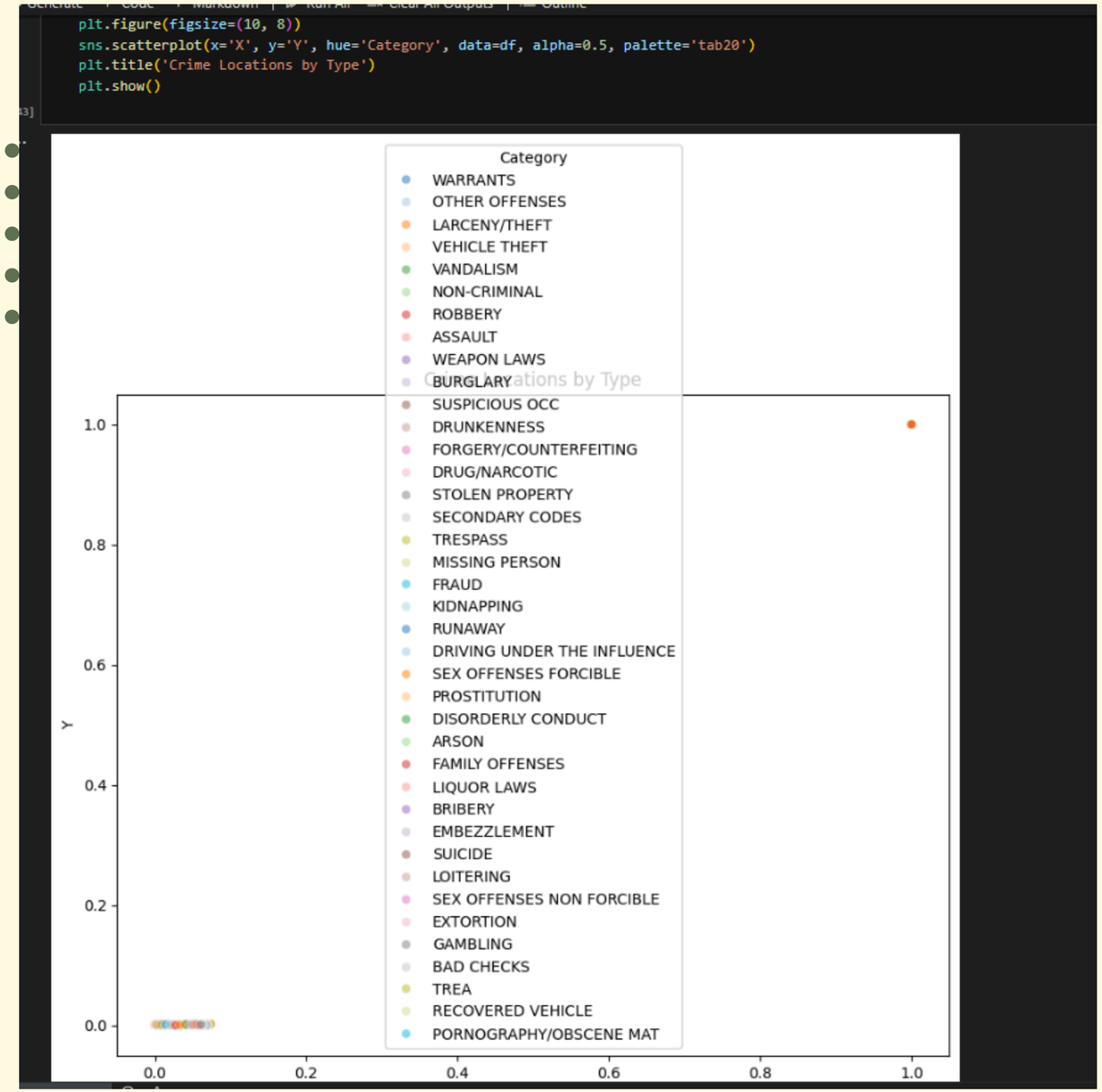
```
sns.violinplot(x='PdDistrict', y='Y', data=df)  # Latitude distribution per district
plt.xticks(rotation=45)
plt.show()
```

42]



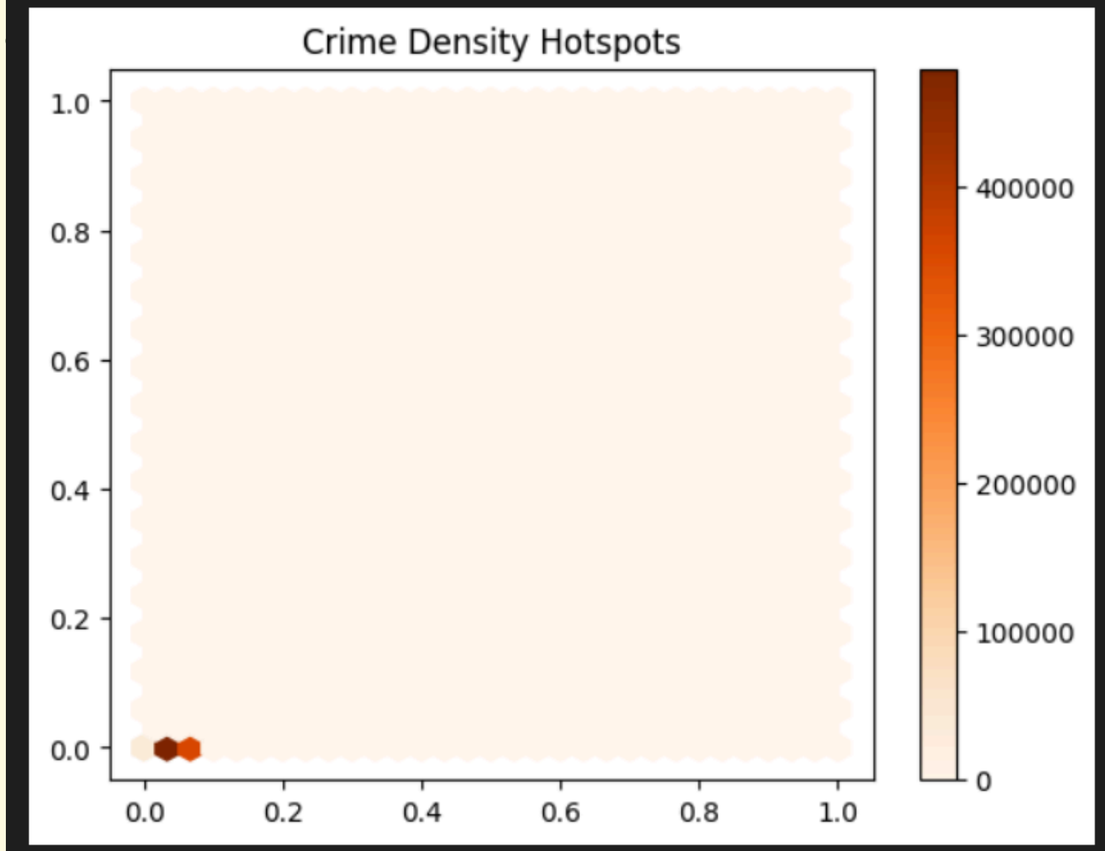
- Purpose: Shows the distribution of latitude per district.
- Insight: Identifies how spatial crime patterns differ across districts.

```
plt.figure(figsize=(10, 8))
sns.scatterplot(x='X', y='Y', hue='Category', data=df, alpha=0.5, palette='tab20')
plt.title('Crime Locations by Type')
plt.show()
```



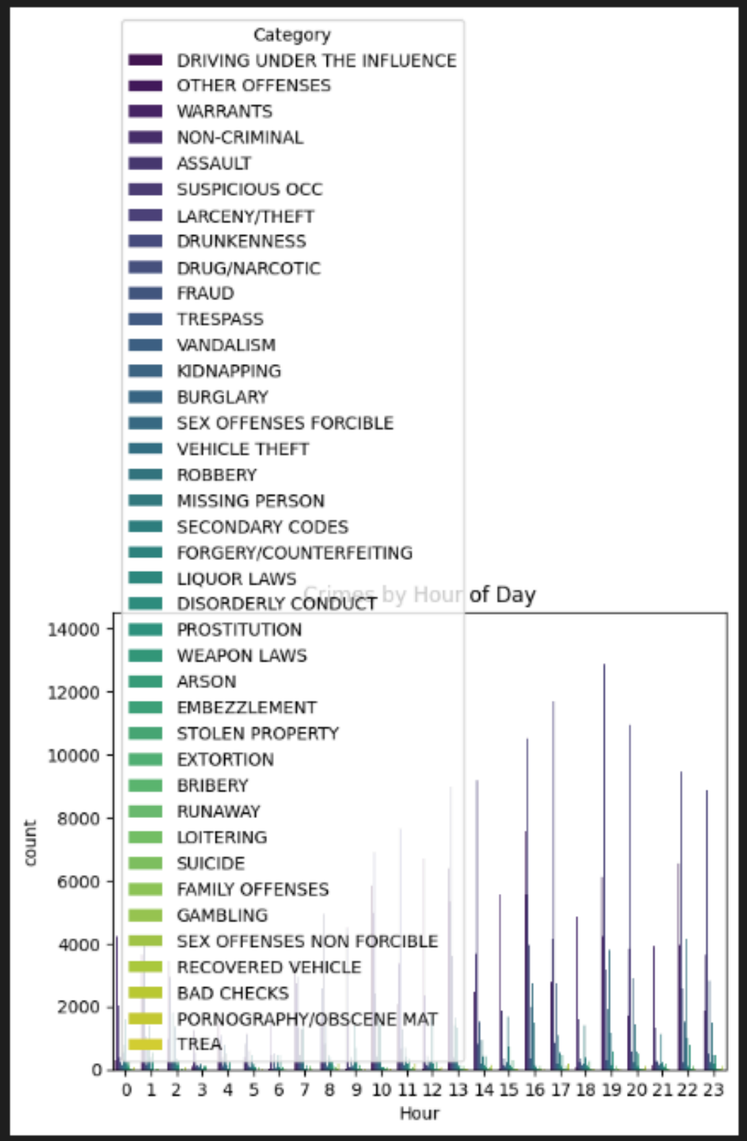
- Purpose: Plots all crime incidents spatially with color indicating type.
- Insight: Helps identify clusters or spatial trends for different crimes.

```
plt.hexbin(df['X'], df['Y'], gridsize=30, cmap='Oranges')
plt.colorbar()
plt.title('Crime Density Hotspots')
plt.show()
```



- Purpose: Highlights dense areas of crime activity.
- Insight: Identifies crime “hotspots.”

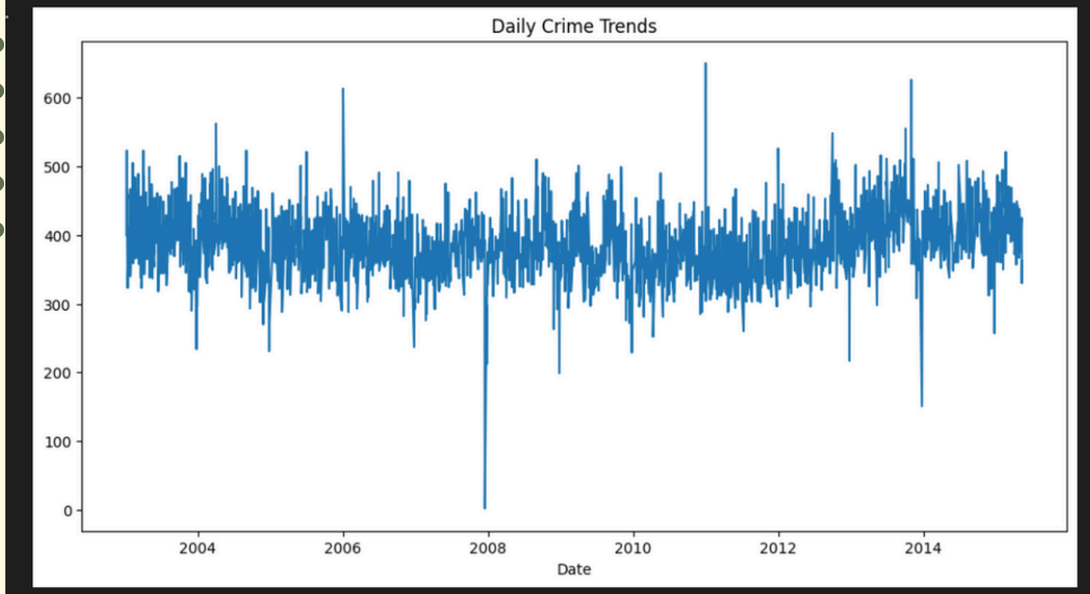
```
df['Hour'] = pd.to_datetime(df['Dates']).dt.hour
sns.countplot(x='Hour', hue='Category', data=df, palette='viridis')
plt.title('Crimes by Hour of Day')
plt.show()
```



- Purpose: Examines crime frequency by time of day.
- Insight: Detects patterns such as more crimes occurring at night or in the afternoon.

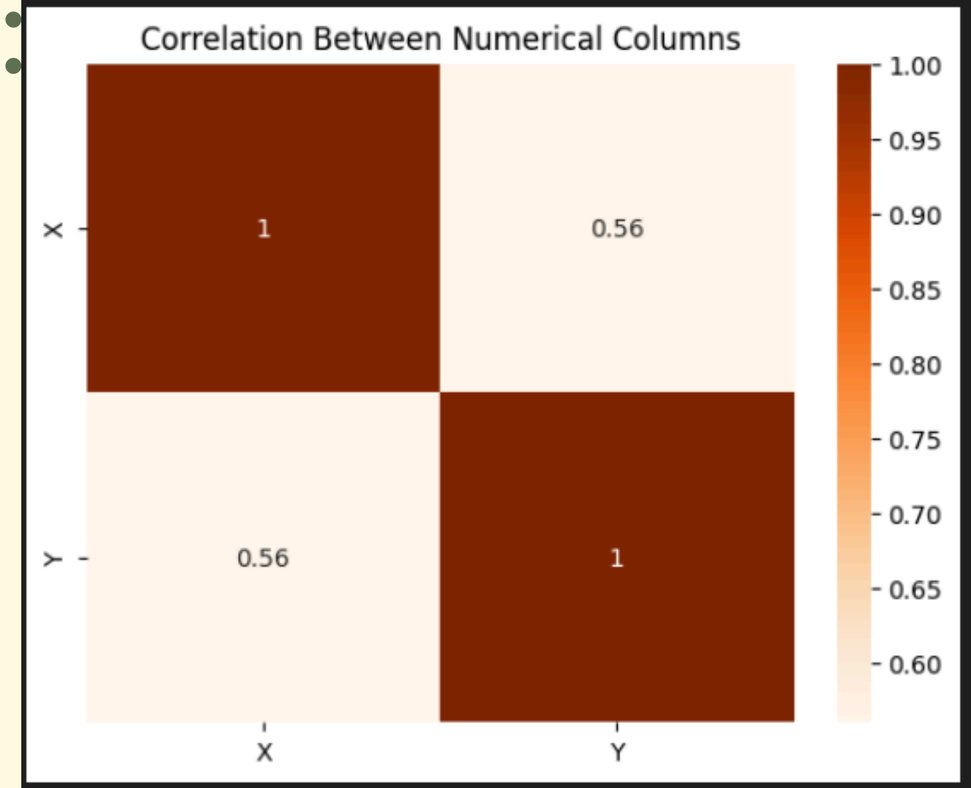
```
df['Date'] = pd.to_datetime(df['Dates']).dt.date
daily_crimes = df.groupby('Date').size()
daily_crimes.plot(figsize=(12, 6), title='Daily Crime Trends')
```

```
<Axes: title={'center': 'Daily Crime Trends'}, xlabel='Date'>
```



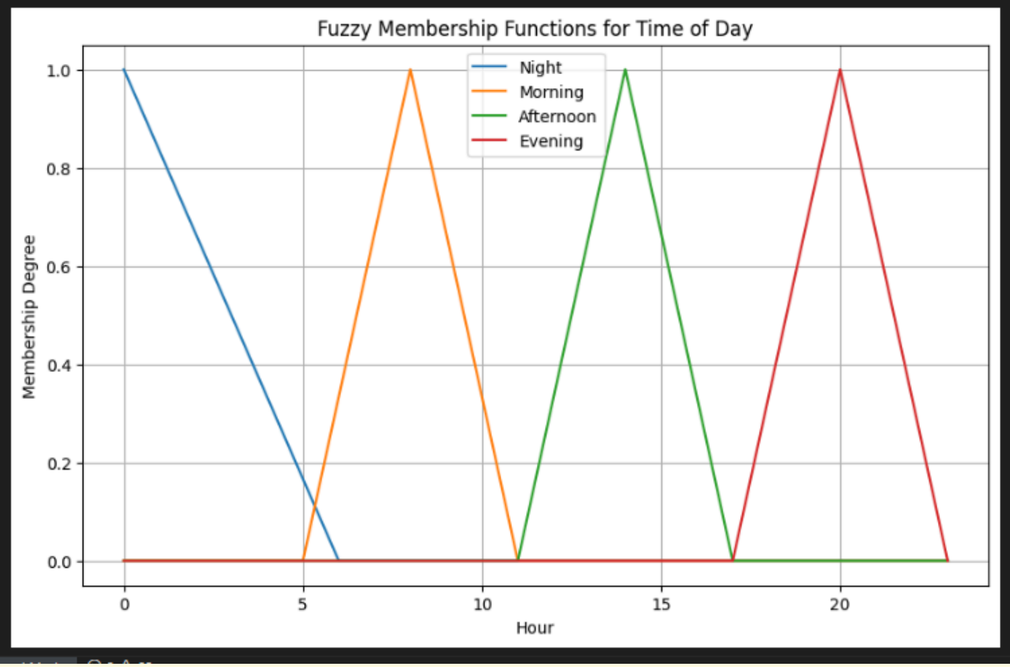
- The "Daily Crime Trends" plot shows the number of reported crimes per day from 2003 to 2015. While there are daily ups and downs, the overall trend remains fairly stable with most days reporting between 300 and 500 crimes. Occasional sharp drops or spikes may indicate data anomalies or special events.

```python
# Select numerical columns
numerical_data = df[['X', 'Y']]  # Add other numerical columns if present
sns.heatmap(numerical_data.corr(), annot=True, cmap='Oranges')
plt.title('Correlation Between Numerical Columns')
plt.show()
```



- Purpose: Shows correlation between variables like location coordinates.
- Insight: Identifies whether variables move together.

```python
# Define the universe of discourse for time (0-23 hours)
x_time = np.arange(0, 24, 1)

# Define fuzzy membership functions
night = fuzz.trimf(x_time, [0, 0, 6])
morning = fuzz.trimf(x_time, [5, 8, 11])
afternoon = fuzz.trimf(x_time, [11, 14, 17])
evening = fuzz.trimf(x_time, [17, 20, 23])

# Plot the fuzzy sets
plt.figure(figsize=(10,6))
plt.plot(x_time, night, label='Night')
plt.plot(x_time, morning, label='Morning')
plt.plot(x_time, afternoon, label='Afternoon')
plt.plot(x_time, evening, label='Evening')
plt.title('Fuzzy Membership Functions for Time of Day')
plt.xlabel('Hour')
plt.ylabel('Membership Degree')
plt.legend()
plt.grid(True)
plt.show()
```

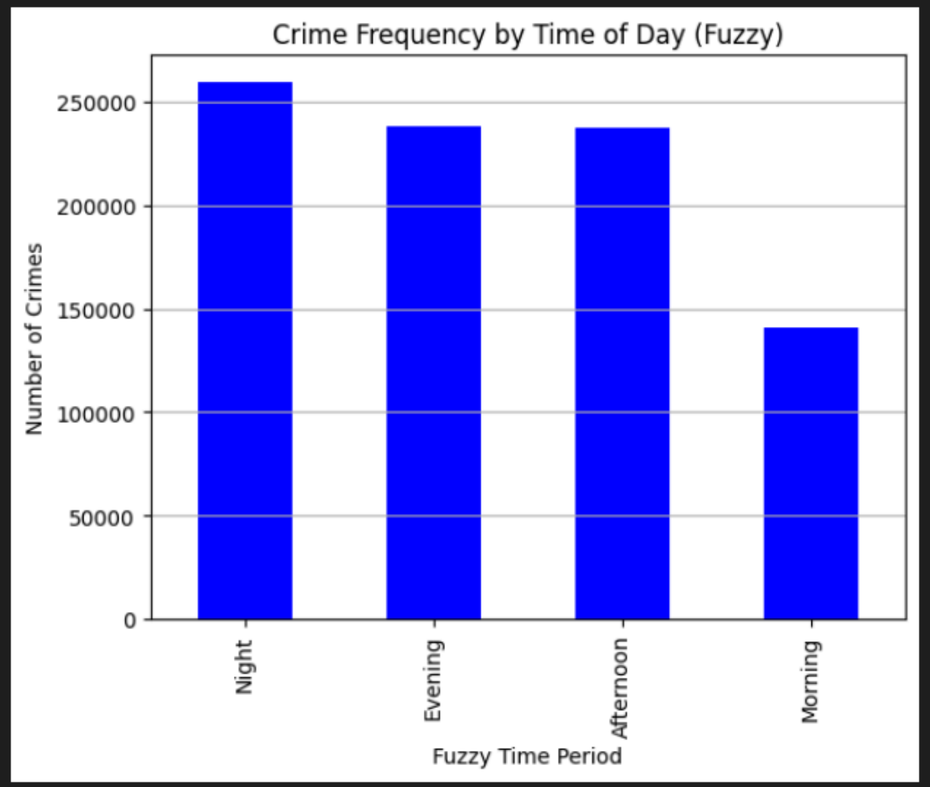


- Purpose: Visualizes how fuzzy sets categorize hours into morning, night, etc.
- Insight: Basis for fuzzifying time into qualitative periods.

```python
# Define a function to classify hour into fuzzy period
def classify_time_of_day(hour):
    night_level = fuzz.interp_membership(x_time, night, hour)
    morning_level = fuzz.interp_membership(x_time, morning, hour)
    afternoon_level = fuzz.interp_membership(x_time, afternoon, hour)
    evening_level = fuzz.interp_membership(x_time, evening, hour)

    levels = {
        'Night': night_level,
        'Morning': morning_level,
        'Afternoon': afternoon_level,
        'Evening': evening_level
    }

    # Return the period with highest membership
    return max(levels, key=levels.get)

# Apply to whole dataframe
df['FuzzyPeriod'] = df['Hour'].apply(classify_time_of_day)

# Check the results
df[['Dates', 'Hour', 'FuzzyPeriod']].head()
```

- The code classifies each crime into a fuzzy time period (Night, Morning, Afternoon, or Evening) based on the hour it occurred. It uses fuzzy logic to assign each hour a degree of belonging to each time period, then labels it with the strongest match. This creates a new column FuzzyPeriod, allowing for more natural, flexible time-based analysis.

```python
# Simple plot
df['FuzzyPeriod'].value_counts().plot(kind='bar', color='blue')
plt.title('Crime Frequency by Time of Day (Fuzzy)')
plt.xlabel('Fuzzy Time Period')
plt.ylabel('Number of Crimes')
plt.grid(axis='y')
plt.show()
```



- Purpose: Counts crimes in fuzzy time periods.
- Insight: Shows when most crimes happen (e.g., "Night" vs "Morning").

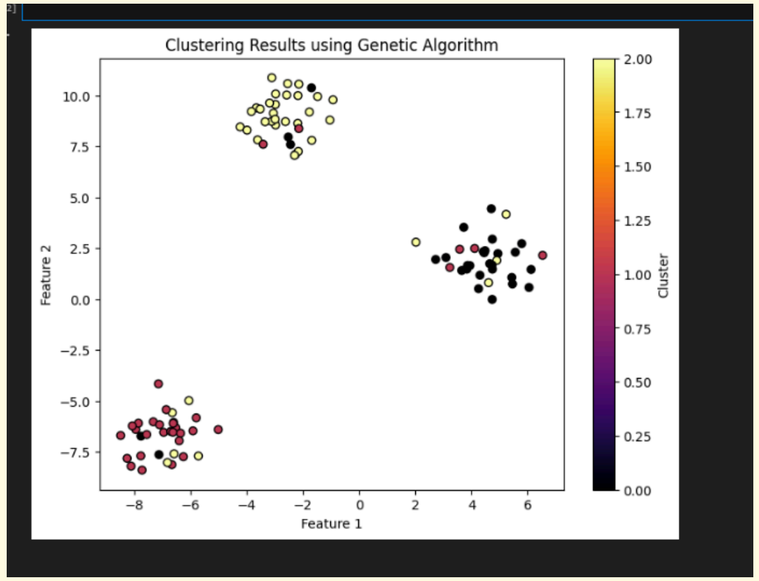
```python
# Create synthetic data
X, _ = make_blobs(n_samples=150, centers=3, random_state=42)

# Define the fitness function
def evaluate(individual):
    # Assign each point to a cluster based on the individual's chromosome
    clusters = [individual[i] for i in range(len(X))]

    # Calculate the within-cluster sum of squared distances (WSS)
    _, distances = pairwise_distances_argmin_min(X, X)
    wss = np.sum(distances**2)

    # Return the fitness score
    return (wss,)

# Define genetic algorithm parameters
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))  # Minimize WSS
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("attr_int", np.random.randint, 0, 3)  # Cluster labels (0, 1, or 2)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_int, n=len(X))
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate)

# Create the population
population = toolbox.population(n=50)

# Run the genetic algorithm
algorithms.eaSimple(population, toolbox, cxpb=0.7, mutpb=0.2, ngen=50, verbose=True)

# Get the best solution
best_individual = tools.selBest(population, 1)[0]
print(f"Best clustering: {best_individual}")
```

The Output

```
gen     nevals
0       50
1       41
2       37
3       36
4       38
5       38
6       40
7       43
8       38
9       43
10      37
11      40
12      41
13      37
14      37
15      39
16      43
17      40
18      39
19      41
20      42
21      39
22      36
23      40
...
48      41
49      41
50      43
Best clustering: [1, 2, 2, 0, 1, 0, 0, 2, 1, 1, 0, 2, 0, 1, 0, 2, 1, 0, 0, 0, 0, 1, 0, 2, 2, 0, 2, 2, 2, 2, 1, 2, 0, 1, 2, 1, 1, 1, 0, 2, 2, 0, 1, 2, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 2, 2, 1, 1, 0, 1, 1, 0, 0, 1, 2, 0, 1,
```

- This code uses a Genetic Algorithm (GA) to solve a clustering problem with data generated by make_blobs. Each individual in the population represents a possible way to assign points to 3 clusters. The GA evolves these cluster assignments by minimizing the within-cluster sum of squared distances (WSS). Over 50 generations, it uses selection, crossover, and mutation to find the most compact clustering. The result is the best individual (i.e., the optimal cluster assignment).

```python
# Generate synthetic data
X, y = make_blobs(n_samples=100, centers=3, random_state=42)

# Create the DEAP classes for Genetic Algorithm
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

# Define a function to create an individual (a potential solution)
def create_individual():
    return [np.random.randint(0, 3) for _ in range(len(X))]

# Define the evaluation function (fitness function)
def evaluate(individual):
    # Assign clusters to individuals
    clusters = np.array(individual)
    # Compute the distance between points and their assigned centroids
    dist = pairwise_distances_argmin_min(X, np.array([X[clusters == i].mean(axis=0) for i in range(3)])[clusters])[1]
    return (dist.sum(),)  # Return the sum of distances as the fitness

# Set up the toolbox
toolbox = base.Toolbox()
toolbox.register("individual", tools.initIterate, creator.Individual, create_individual)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("evaluate", evaluate)

# Generate the population
population = toolbox.population(n=50)

# Run the genetic algorithm
algorithms.eaSimple(population, toolbox, cxpb=0.7, mutpb=0.2, ngen=50, verbose=False)

# Extract the best individual
best_individual = tools.selBest(population, 1)[0]

# Assign the final clusters
final_clusters = np.array(best_individual)

# Visualize the clustering results
plt.figure(figsize=(8, 6))
plt.scatter(X[:, 0], X[:, 1], c=final_clusters, cmap='inferno', marker='o', edgecolor='k')
plt.title("Clustering Results using Genetic Algorithm")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.colorbar(label="Cluster")
plt.show()
```

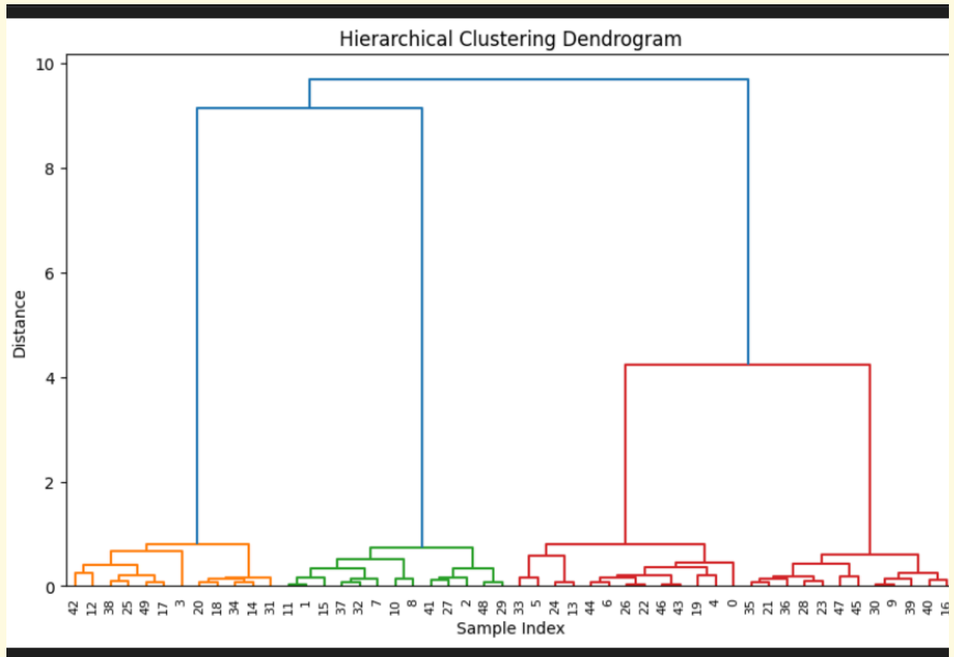


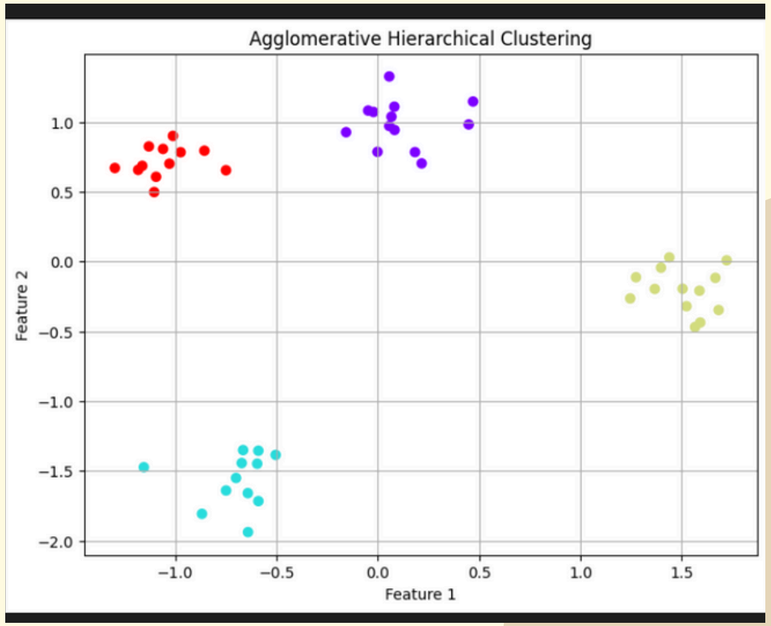
This code applies a Genetic Algorithm (GA) to cluster data points from a synthetic dataset with 3 natural groupings. Each individual represents a potential clustering (assigning each point to cluster 0, 1, or 2). The fitness function evaluates how well points are grouped by calculating the total distance from each point to its cluster's centroid. Over 50 generations, the GA evolves better cluster assignments. Finally, the best result is visualized, showing the GA-based clustering.

```python
# Replace this with your real data
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples=50, centers=4, random_state=42)

# Scale the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Plot the dendrogram to decide number of clusters
linked = linkage(X_scaled, method='ward')

plt.figure(figsize=(10, 6))
dendrogram(linked, orientation='top', distance_sort='descending', show_leaf_counts=True)
plt.title("Hierarchical Clustering Dendrogram")
plt.xlabel("Sample Index")
plt.ylabel("Distance")
plt.show()

# Apply Agglomerative Clustering
agglo = AgglomerativeClustering(n_clusters=4)
labels = agglo.fit_predict(X_scaled)

# Plot result
plt.figure(figsize=(8, 6))
plt.scatter(X_scaled[:, 0], X_scaled[:, 1], c=labels, cmap='rainbow')
plt.title("Agglomerative Hierarchical Clustering")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.grid(True)
plt.show()
```

This code performs Agglomerative Hierarchical Clustering on scaled synthetic data with 4 centers. It first visualizes a dendrogram to help decide the number of clusters, then applies clustering using AgglomerativeClustering with n_clusters=4. Finally, it plots the clustered data, showing how points are grouped based on their similarity using hierarchical merging.

4o



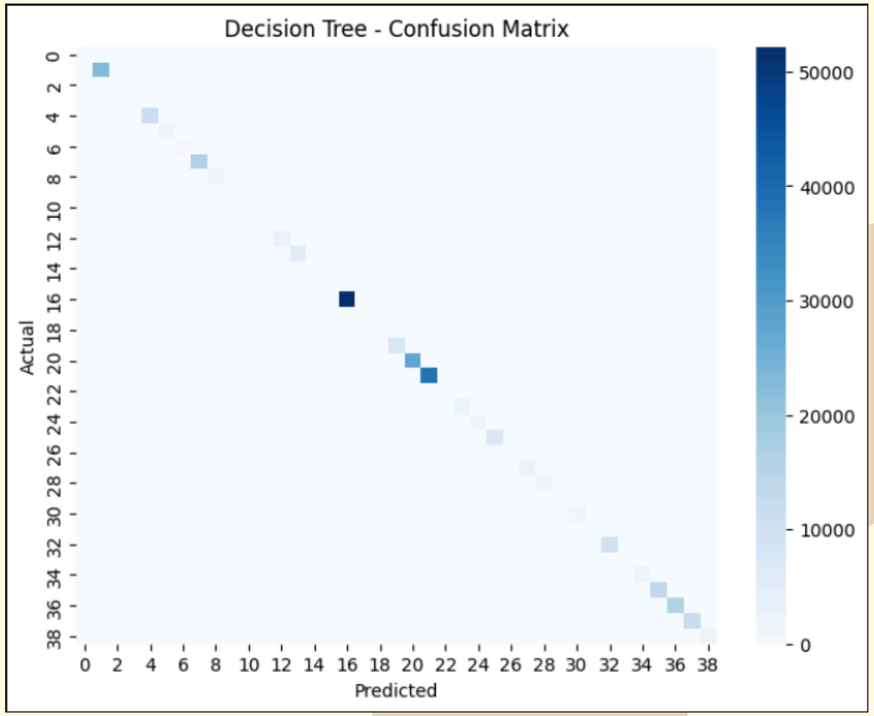


```python
# Handle missing values by filling or dropping
df = df.fillna('Unknown')

# Identify columns that are categorical
categorical_cols = df.select_dtypes(include=['object']).columns

# Convert all categorical columns to strings to avoid issues with encoding
for col in categorical_cols:
    df[col] = df[col].astype(str)

# Encode categorical features using LabelEncoder
label_encoders = {}  # To store encoders for each column if needed later
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le

X = df.drop(columns=['Category', 'Dates'])  # Features (excluding target and date columns)
y = df['Category']  # Target (the category)

# Train-test split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=42)

# Decision Tree Classifier
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)  # Fit on training data
y_pred_dt = dt_model.predict(X_val)  # Predict on validation data

# Evaluation
print("Model: Decision Tree")
print(f"Accuracy: {accuracy_score(y_val, y_pred_dt):.4f}")
print("Classification Report:")
print(classification_report(y_val, y_pred_dt))
print("Confusion Matrix:")
cm_dt = confusion_matrix(y_val, y_pred_dt)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_dt, annot=False, cmap="Blues")
plt.title("Decision Tree - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

This code trains a Decision Tree Classifier to predict crime categories using a dataset with categorical features. It fills missing values, label-encodes categorical columns, splits the data into training and validation sets, and fits the model. The performance is evaluated using accuracy, a classification report, and a confusion matrix heatmap to visualize prediction errors.

The Output

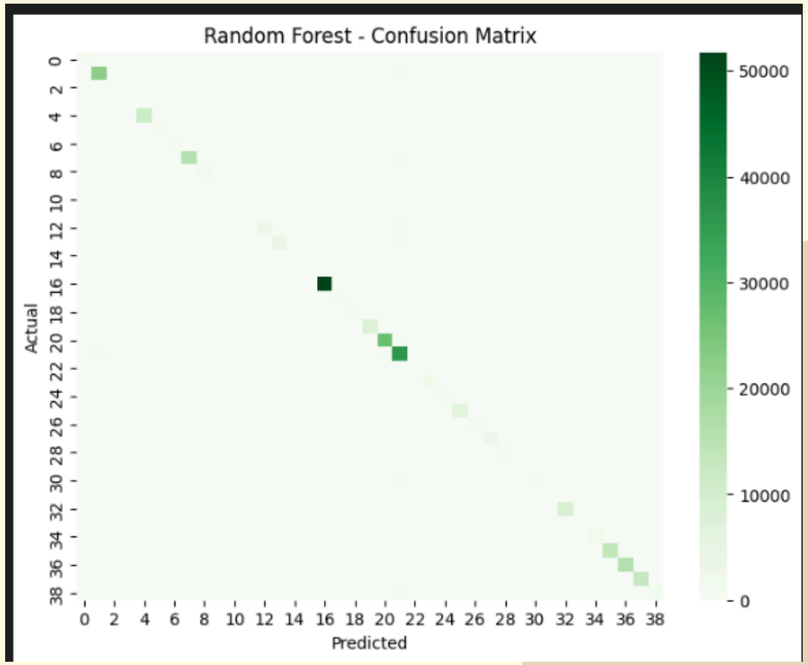
```
Model: Decision Tree
Accuracy: 0.9993
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       462
           1       1.00      1.00      1.00     23200
           2       1.00      1.00      1.00       117
           3       1.00      1.00      1.00        96
           4       1.00      1.00      1.00     10958
           5       1.00      1.00      1.00      1295
           6       1.00      1.00      1.00       680
           7       1.00      1.00      1.00     16230
           8       1.00      1.00      1.00      1297
           9       1.00      1.00      1.00       337
          10       1.00      1.00      1.00        71
          11       1.00      1.00      1.00       154
          12       1.00      1.00      1.00      3151
          13       1.00      1.00      1.00      5100
          14       1.00      0.96      0.98        47
          15       0.94      0.90      0.92       673
          16       1.00      1.00      1.00     52161
          17       0.99      0.99      0.99       584
          18       0.99      0.99      0.99       373
          19       1.00      1.00      1.00      7701
...
   macro avg       1.00      0.99      0.99    262718
weighted avg       1.00      1.00      1.00    262718

Confusion Matrix:
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



```python
# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_val)

# Evaluation
print("Model: Random Forest")
print(f"Accuracy: {accuracy_score(y_val, y_pred_rf):.4f}")
print("Classification Report:")
print(classification_report(y_val, y_pred_rf))
print("Confusion Matrix:")
cm_rf = confusion_matrix(y_val, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_rf, annot=False, cmap="Greens")
plt.title("Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Model: Random Forest
Accuracy: 0.9547
Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.74      0.81       462
           1       0.94      0.96      0.95     23200
           2       0.71      0.17      0.28       117
           3       0.85      0.29      0.43        96
           4       1.00      0.99      0.99     10958
           5       0.89      0.72      0.80      1295
           6       0.93      0.85      0.89       680
           7       0.98      0.96      0.97     16230
           8       0.87      0.83      0.85      1297
           9       0.78      0.53      0.63       337
          10       0.22      0.06      0.09        71
          11       0.84      0.17      0.28       154
          12       0.84      0.85      0.85      3151
          13       0.88      0.85      0.87      5100
          14       0.80      0.09      0.15        47
          15       0.55      0.30      0.39       673
          16       0.99      0.99      0.99     52161
          17       0.89      0.55      0.68       584
          18       0.93      0.67      0.78       373
          19       0.98      0.97      0.98      7701
...
   macro avg       0.82      0.68      0.72    262718
weighted avg       0.95      0.95      0.95    262718

Confusion Matrix:
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

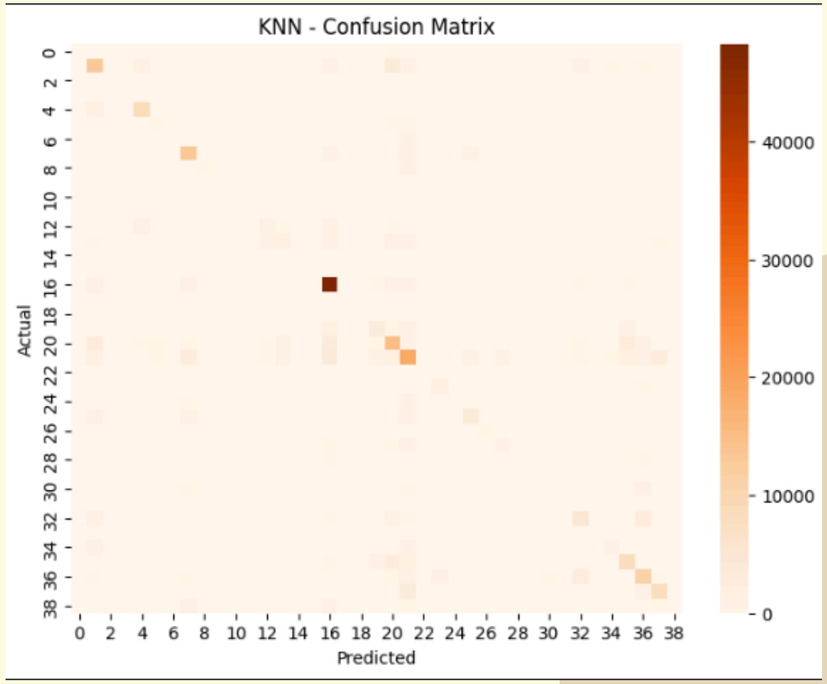
This code uses a Random Forest Classifier with 100 trees to predict crime categories. It trains the model on labeled data, makes predictions on a validation set, and evaluates performance using accuracy, a classification report, and a confusion matrix heatmap. The heatmap visualizes how well the model distinguishes between crime types, helping identify patterns and misclassifications.



```python
# K-Nearest Neighbors Classifier
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_val)

# Evaluation
print("Model: K-Nearest Neighbors (KNN)")
print(f"Accuracy: {accuracy_score(y_val, y_pred_knn):.4f}")
print("Classification Report:")
print(classification_report(y_val, y_pred_knn, zero_division=0))
print("Confusion Matrix:")
cm_knn = confusion_matrix(y_val, y_pred_knn)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_knn, annot=False, cmap="Oranges")
plt.title("KNN - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

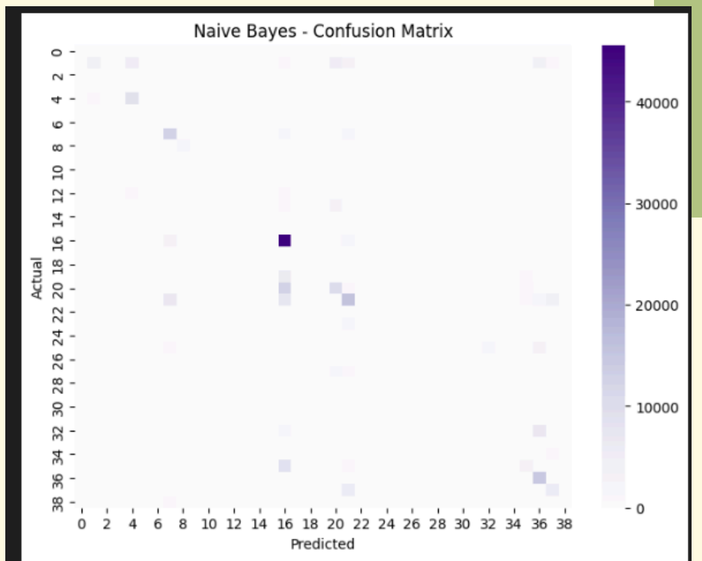
```
Model: K-Nearest Neighbors (KNN)
Accuracy: 0.6155
Classification Report:
              precision    recall  f1-score   support

           0       0.06      0.01      0.02       462
           1       0.55      0.57      0.56     23200
           2       0.02      0.01      0.01       117
           3       0.03      0.01      0.02        96
           4       0.69      0.78      0.73     10958
           5       0.23      0.24      0.23      1295
           6       0.06      0.03      0.04       680
           7       0.69      0.81      0.75     16230
           8       0.24      0.15      0.19      1297
           9       0.06      0.04      0.05       337
          10       0.12      0.04      0.06        71
          11       0.09      0.05      0.06       154
          12       0.34      0.25      0.28      3151
          13       0.36      0.27      0.31      5100
          14       0.00      0.00      0.00        47
          15       0.10      0.03      0.04       673
          16       0.77      0.93      0.84     52161
          17       0.21      0.05      0.08       584
          18       0.49      0.32      0.39       373
          19       0.53      0.43      0.48      7701
...
   macro avg       0.32      0.28      0.29    262718
weighted avg       0.59      0.62      0.60    262718

Confusion Matrix:
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

This code applies a K-Nearest Neighbors (KNN) classifier to predict crime categories. It fits the model on training data, predicts on the validation set, and evaluates performance using accuracy, a detailed classification report, and a confusion matrix heatmap. The heatmap helps visualize how well the model classifies each crime type and where it makes mistakes.



```python
# Naive Bayes Classifier
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_val)

# Evaluation
print("Model: Naive Bayes")
print(f"Accuracy: {accuracy_score(y_val, y_pred_nb):.4f}")
print("Classification Report:")
print(classification_report(y_val, y_pred_nb, zero_division=0))
print("Confusion Matrix:")
cm_nb = confusion_matrix(y_val, y_pred_nb)
plt.figure(figsize=(8, 6))
sns.heatmap(cm_nb, annot=False, cmap="Purples")
plt.title("Naive Bayes - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

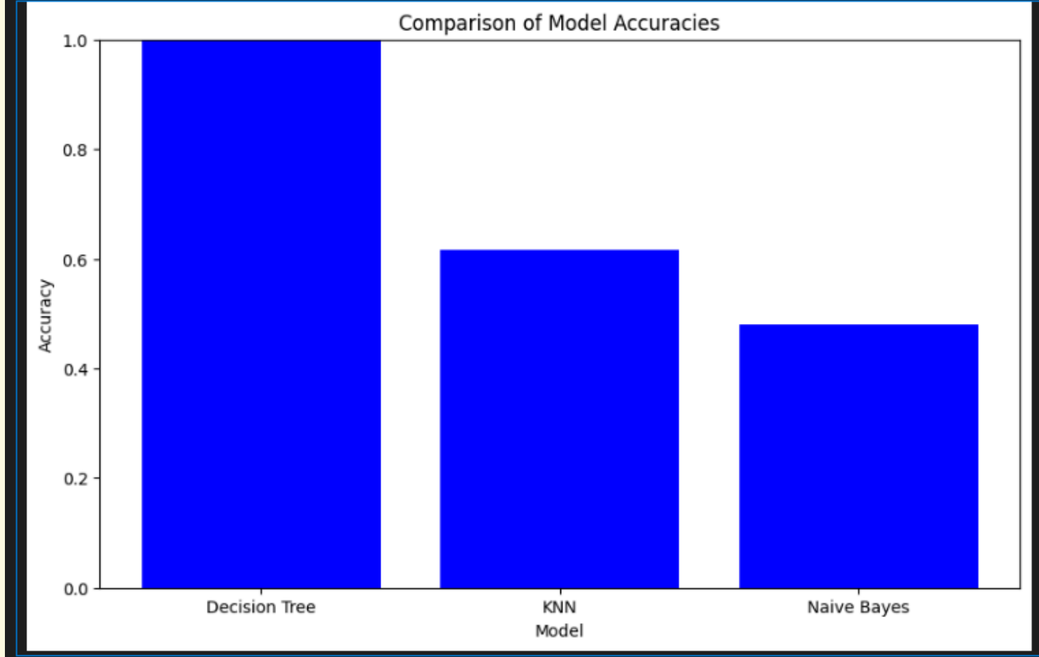
```
Model: Naive Bayes
Accuracy: 0.4804
Classification Report:
              precision    recall  f1-score   support

           0       0.36      0.03      0.06       462
           1       0.62      0.20      0.30     23200
           2       1.00      0.72      0.84       117
           3       0.00      0.00      0.00        96
           4       0.53      0.84      0.65     10958
           5       0.00      0.00      0.00      1295
           6       0.99      0.88      0.93       680
           7       0.50      0.79      0.61     16230
           8       1.00      1.00      1.00      1297
           9       0.85      0.23      0.37       337
          10       0.00      0.00      0.00        71
          11       0.00      0.00      0.00       154
          12       0.00      0.00      0.00      3151
          13       0.00      0.00      0.00      5100
          14       0.00      0.00      0.00        47
          15       0.00      0.00      0.00       673
          16       0.51      0.87      0.65     52161
          17       0.00      0.00      0.00       584
          18       0.00      0.00      0.00       373
          19       0.16      0.01      0.02      7701
...
   macro avg       0.28      0.24      0.23    262718
weighted avg       0.43      0.48      0.41    262718

Confusion Matrix:
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```



he code you provided trains a Naive Bayes classifier (GaussianNB) on the training data, makes predictions on the validation data, and then evaluates its performance using the following steps:

1. Accuracy: Calculates the proportion of correct predictions using accuracy_score.
2. Classification Report: Displays precision, recall, f1-score, and support for each class using classification_report.
3. Confusion Matrix: Visualizes the true vs. predicted class values in a heatmap using Seaborn (sns.heatmap).

```python
# Compare model accuracies (optional summary chart)
accuracies = {
    "Decision Tree": accuracy_score(y_val, y_pred_dt),
    # "Random Forest": accuracy_score(y_test, y_pred_rf),
    "KNN": accuracy_score(y_val, y_pred_knn),
    "Naive Bayes": accuracy_score(y_val, y_pred_nb)
}

plt.figure(figsize=(10, 6))
plt.bar(accuracies.keys(), accuracies.values(), color='blue')
plt.title("Comparison of Model Accuracies")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
plt.show()
```



The code compares the accuracy of different machine learning models (Decision Tree, KNN, and Naive Bayes) by calculating their accuracy scores and displaying them in a bar chart. This visualization allows for a quick comparison of model performance, helping to identify which model achieved the highest accuracy.

```
    print("Accuracy:", accuracy_score(y_val, y_pred_dt))
    print("Precision:", precision_score(y_val, y_pred_dt, average='weighted', zero_division=0))
    print("Recall:", recall_score(y_val, y_pred_dt, average='weighted'))
    print("F1-score:", f1_score(y_val, y_pred_dt, average='weighted'))
    print("Confusion Matrix:\n", confusion_matrix(y_val, y_pred_dt))
    print("Classification Report:\n", classification_report(y_val, y_pred_dt, zero_division=0))
```

60]

```
Accuracy: 0.9993414992501466
Precision: 0.9993352884235487
Recall: 0.9993414992501466
F1-score: 0.9993368190355474
Confusion Matrix:
 [[  462     0     0 ...     0     0     0]
 [    0 23157     0 ...     0     0     0]
 [    0     0   117 ...     0     0     0]
 ...
 [    0     0     0 ... 15879     0     0]
 [    0     0     0 ...     0 12600     0]
 [    0     0     0 ...     0     0  2463]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       462
           1       1.00      1.00      1.00     23200
           2       1.00      1.00      1.00       117
           3       1.00      1.00      1.00        96
           4       1.00      1.00      1.00     10958
           5       1.00      1.00      1.00      1295
           6       1.00      1.00      1.00       680
           7       1.00      1.00      1.00     16230
           8       1.00      1.00      1.00      1297
           9       1.00      1.00      1.00       337
...
    accuracy                           1.00    262718
   macro avg       1.00      0.99      0.99    262718
weighted avg       1.00      1.00      1.00    262718

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

```
    plt.figure(figsize=(12, 6))
    crime_counts = df['Category'].value_counts().head(10)
```

61]

```
<Figure size 1200x600 with 0 Axes>
```

the first block of code

1. calculate the accuracy
2. precision
3. recall
4. f1 score
5. conf matrix
6. classification report

the second block

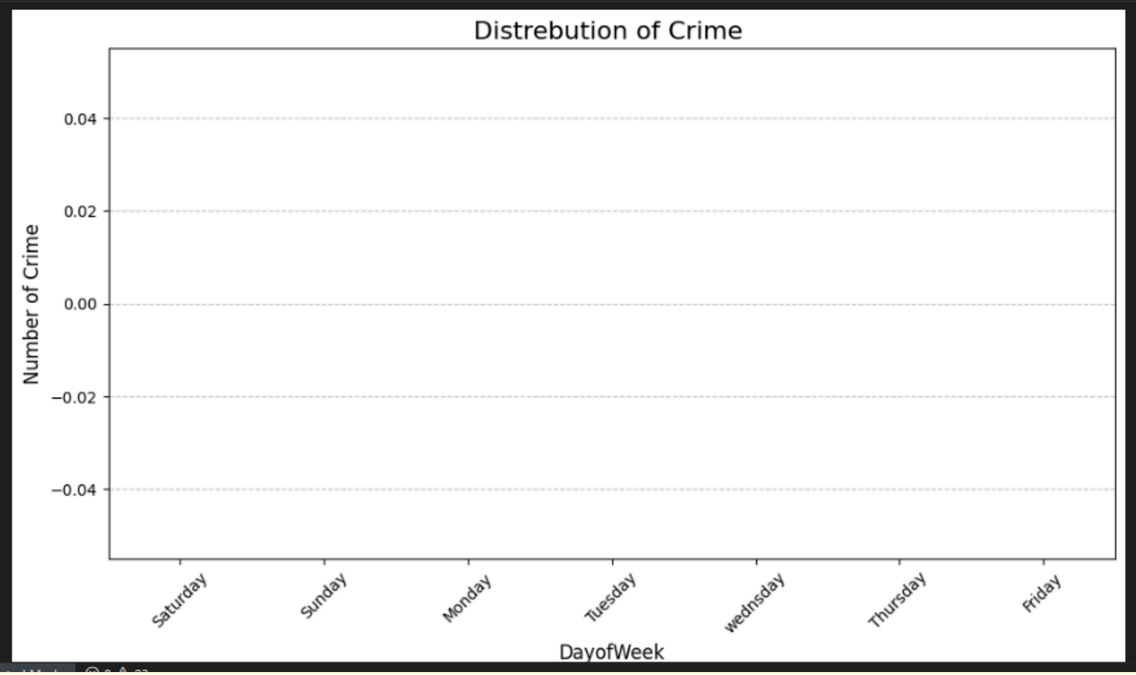
- represent the size of the plot figure

```python
# Create the barplot
sns.barplot(x=crime_counts.values, y=crime_counts.index, palette='viridis')

# Plot styling
plt.title('The Most Common Types of Crime', fontsize=16)
plt.xlabel('Number of Incidents (Log Scale)', fontsize=12)
plt.ylabel('Type of Crime', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Apply log scale to x-axis
plt.xscale('log')

# Add value labels to each bar
for i, v in enumerate(crime_counts.values):
    plt.text(v * 1.05, i, str(v), color='black', ha='left', va='center')

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

The code creates a horizontal bar plot showing the most common crime types, with crime counts on a logarithmic scale. It includes value labels on each bar, a grid for the x-axis, and is styled with the viridis color palette for clear visualization.

4o mini

```python
day_order = ['Saturday', 'Sunday', 'Monday', 'Tuesday', 'wednsday', 'Thursday', 'Friday']

plt.figure(figsize=(10, 6))
sns.countplot(data=df, x='DayOfWeek', order=day_order, palette='coolwarm')

plt.title('Distrebution of Crime ', fontsize=16)
plt.xlabel('DayofWeek', fontsize=12)
plt.ylabel('Number of Crime', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)

for p in plt.gca().patches:
    plt.gca().annotate(f'{int(p.get_height())}',
                       (p.get_x() + p.get_width() / 2., p.get_height()),
                       ha='center', va='center',
                       xytext=(0, 5),
                       textcoords='offset points')

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



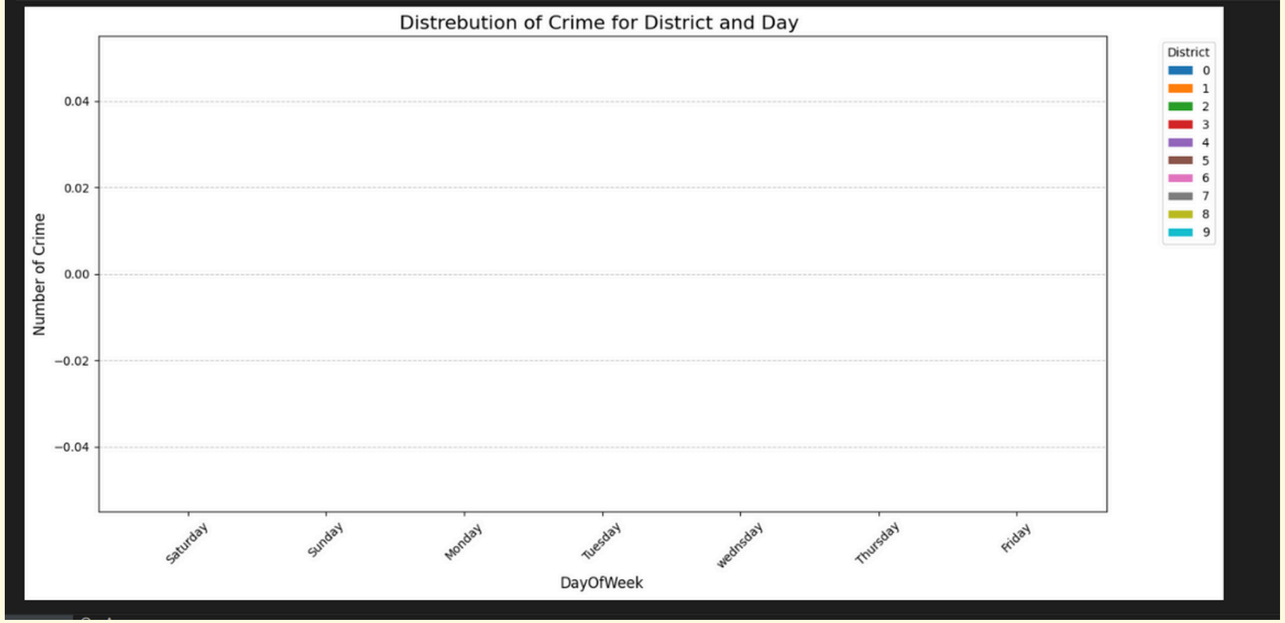
This code creates a bar plot to show the distribution of crimes by day of the week. It uses a custom order for the days, applies the coolwarm color palette, and displays the number of crimes for each day. Value labels are added on top of each bar, and the x-axis labels are rotated for better readability. The plot includes a grid for the y-axis and proper styling for clarity.

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

# Count crimes per district
district_counts = df['PdDistrict'].value_counts()

# Set figure size
plt.figure(figsize=(12, 6))

# Create barplot
sns.barplot(x=district_counts.values, y=district_counts.index, palette='magma')

# Title and labels
plt.title('Distribution of Crime by District', fontsize=16)
plt.xlabel('Number of Crimes (Log Scale)', fontsize=12)
plt.ylabel('District', fontsize=12)
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Apply log scale to x-axis
plt.xscale('log')

# Add value labels with spacing suitable for log scale
for i, v in enumerate(district_counts.values):
    plt.text(v * 1.05, i, str(v), color='black', ha='left', va='center')

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

Python

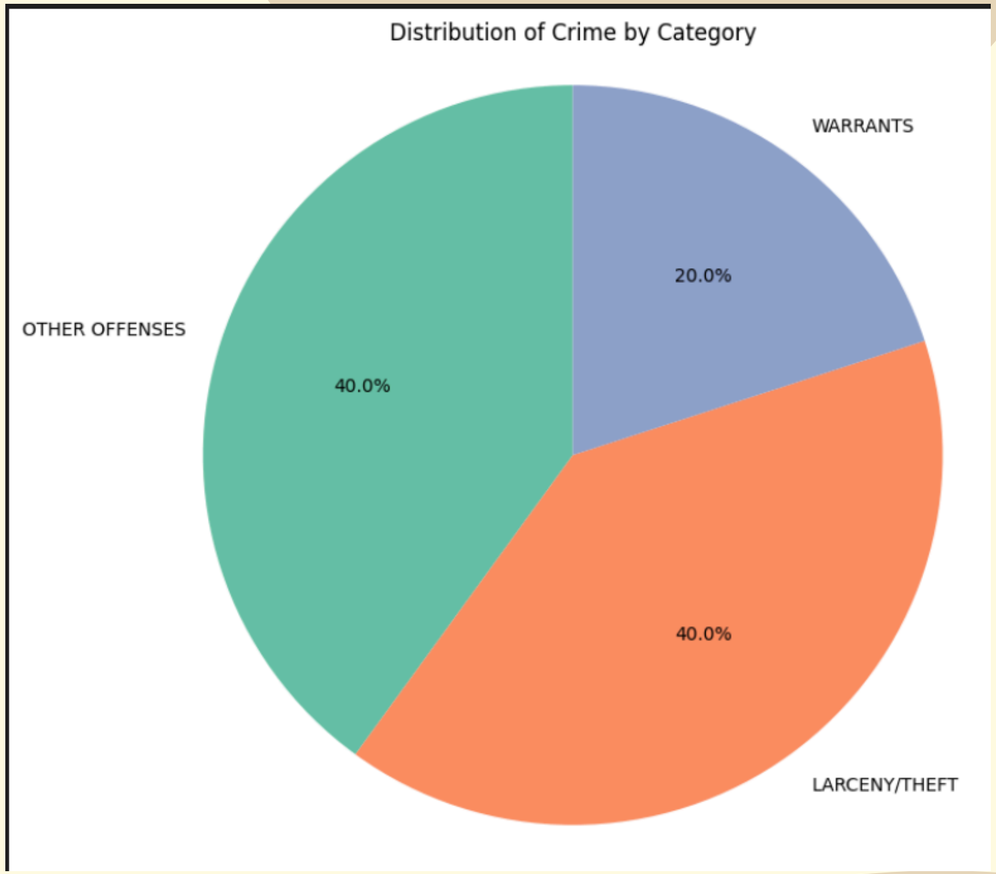
This code creates a bar plot to show the distribution of crimes by district, using a logarithmic scale for the x-axis. The plot uses the magma color palette, includes value labels on each bar, and has a grid on the x-axis for better clarity. The x-axis is scaled logarithmically to handle varying crime counts across districts.

```
cross_tab = pd.crosstab(df['DayOfWeek'], df['PdDistrict']).reindex(day_order)
cross_tab.plot(kind='bar', figsize=(14, 7), width=0.8)

plt.title('Distrebution of Crime for District and Day ', fontsize=16)
plt.xlabel('DayOfWeek', fontsize=12)
plt.ylabel('Number of Crime', fontsize=12)
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.legend(title='District', bbox_to_anchor=(1.05, 1), loc='upper left')

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



This code creates a grouped bar plot to show the distribution of crimes across different districts for each day of the week. It uses a cross-tabulation of DayOfWeek and PdDistrict, and visualizes the counts in a bar chart. The plot includes a title, axis labels, rotated x-axis labels, a legend for districts, and a grid on the y-axis for clarity.

```python
from matplotlib import cm
# data
data = {
    'Dates': ['2015-05-13 23:53:00', '2015-05-13 23:53:00', '2015-05-13 23:33:00', '2015-05-13 23:30:00', '2015-05-13 23:30:00'],
    'Category': ['WARRANTS', 'OTHER OFFENSES', 'OTHER OFFENSES', 'LARCENY/THEFT', 'LARCENY/THEFT'],
    'Descript': ['WARRANT ARREST', 'TRAFFIC VIOLATION ARREST', 'TRAFFIC VIOLATION ARREST',
                 'GRAND THEFT FROM LOCKED AUTO', 'GRAND THEFT FROM LOCKED AUTO'],
    'DayOfWeek': ['Wednesday', 'Wednesday', 'Wednesday', 'Wednesday', 'Wednesday'],
    'PdDistrict': ['NORTHERN', 'NORTHERN', 'NORTHERN', 'NORTHERN', 'PARK'],
    'Resolution': ['ARREST, BOOKED', 'ARREST, BOOKED', 'ARREST, BOOKED', 'NONE', 'NONE'],
    'Address': ['OAK ST / LAGUNA ST', 'OAK ST / LAGUNA ST', 'VANNESS AV / GREENWICH ST',
                '1500 Block of LOMBARD ST', '100 Block of BRODERICK ST'],
    'X': [-122.425892, -122.425892, -122.424363, -122.426995, -122.438738],
    'Y': [37.774599, 37.774599, 37.800414, 37.800873, 37.771541]
}

df = pd.DataFrame(data)

# Count categories
category_counts = df['Category'].value_counts()

# Choose a colormap with enough colors
colors = cm.Set2.colors[:len(category_counts)]  # You can try Set3, tab20, Pastel1, etc.

# Plot
plt.figure(figsize=(8, 8))
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%', startangle=90, colors=colors)
plt.title('Distribution of Crime by Category')
plt.axis('equal')  # Equal aspect ratio for a perfect circle

plt.show()
```



This code creates a pie chart to visualize the distribution of crimes by category. It counts the occurrences of each crime category, selects a colormap with enough distinct colors, and then plots the pie chart with percentage labels. The chart is displayed with an equal aspect ratio for a perfect circle.

THANK YOU