Data Analysis and Visualization Project

Cyber Crime In India Analysis

Project Context:

The increasing digitalization in India has led to a rise in cyberrelated activities, including cybercrimes. This project aims to analyze cybercrime trends in India to uncover patterns, hotspots, and contributing factors. The analysis will leverage historical and recent data to provide insights into the nature and impact of cybercrimes in the country.

Using statistical and visualization techniques, this project will focus on:

1. Types of cybercrimes (e.g., identity theft, hacking, phishing, etc.).
2. Demographic and geographic distribution of incidents.
3. Trends over the years.
4. Key factors driving the rise in cybercrimes.
5. Possible preventive measures based on identified patterns.

The ultimate goal is to support policymakers, researchers, and the general public in understanding and mitigating cyber threats.

To achieve these objectives, the project will use the following tools and technologies:

- **Programming Language:** Python • **Data Manipulation & Analysis:** Pandas, NumPy
- **Data Visualization:** Matplotlib, Seaborn, Plotly
- **Machine Learning (if required):** Scikit-learn, TensorFlow (for predictive modeling)
- **Geospatial Analysis:** Geopandas, Folium
- **Data Sources:** National Crime Records Bureau (NCRB) reports, publicly available datasets, and APIs.
- **IDE/Environment:** Google Colab for coding and experimentation.
- **Documentation:** Jupyter Notebooks, Markdown for reports.
-

Objective:

The primary objective of the project is to analyze and visualize the trends, patterns, and distribution of cybercrimes in India. This involves:

1. Identifying the most prevalent types of cybercrimes and their growth over time.
2. Analyzing the demographic and geographic distribution of cybercrime incidents.
3. Uncovering correlations between cybercrime rates and socio-economic factors.
4. Visualizing trends and creating interactive dashboards for a comprehensive understanding.
5. Providing data-driven insights to assist policymakers, law enforcement agencies, and the public in formulating effective strategies to prevent and combat cybercrime.

Data Description:

Introduction to the Datasets

This report uses two key datasets to analyze cybercrime trends in India. The datasets provide detailed information on cybercrimes at both state and city levels over multiple years, offering a comprehensive view of crime trends and patterns. These datasets serve as the foundation for the analysis and are used to uncover patterns, identify regions with high cybercrime rates, and understand the distribution of different types of cybercrimes.

Dataset 1: State/UT vs Years (2002–2021)

Source: Indian Cyber Crime Data (from a government or relevant authority)

Dataset Overview

The first dataset contains aggregated cybercrime data at the state and union territory (UT) level from 2002 to 2021. It tracks various categories of cybercrimes across different states and UTs, providing insights into trends over time. This dataset allows the examination of year-wise trends, regional disparities, and the overall increase in cybercrimes.

Columns and Data Structure

The dataset consists of **24 columns** and **40 rows**, each representing a different state or UT in India.

1. **State/UT**: The name of the state or union territory (e.g., Maharashtra, Delhi).
2. **2002 to 2021**: These columns represent the number of cybercrimes reported in each state or UT for each year.

3. **Total**: This column provides the total number of cybercrimes reported across all years for each state/UT.
4. **Total-Scale**: A derived column that potentially aggregates or scales the total based on specific parameters (e.g., population, size of the state).

Data Insights

- **Trends**: The dataset provides a clear indication of the rise in cybercrime incidents, especially post-2010, when internet penetration and digital adoption increased.
- **State Disparities**: States like Maharashtra, Uttar Pradesh, and Tamil Nadu generally show a higher incidence of cybercrimes compared to smaller states, which can be attributed to factors such as population size, internet usage, and urbanization.

Missing Values

There are no significant missing values in this dataset, and all rows have valid data for every year from 2002 to 2021.

Dataset 2: City vs Cybercrime Data

Source: Cybercrime Data (from a regional or city-level authority)

Dataset Overview

The second dataset contains more granular data on cybercrimes at the **city level**. It spans 191 rows, each corresponding to a city in India. This dataset categorizes cybercrimes into specific types and provides both the total number of cybercrimes and breakdowns by category. It also includes demographic features such as the city name.

Columns and Data Structure

The dataset consists of **17 columns** and **191 rows**, with the following key columns:

1. **City**: Name of the city (e.g., Mumbai, Delhi, Bengaluru).
2. **Cybercrime Categories**: Columns representing various types of cybercrimes, such as:
    - Fraud
    - Anger
    - Sexual Exploitation
    - Harassment
    - Data Theft
3. **Total**: The total number of cybercrimes reported in each city.

Data Insights

- **Crime Types**: The data shows a variety of cybercrime types, with fraud and sexual exploitation being the most prevalent in major cities. Harassment, especially online harassment, has also seen a significant rise, particularly in urban centers with higher internet penetration.
- **City-wise Distribution**: Cities like Mumbai, Delhi, and Bengaluru experience the highest cybercrime rates, likely due to their large populations and status as digital hubs. Smaller cities have lower overall cybercrime incidents but might show higher per-capita rates.
- **Emerging Patterns**: Certain cities with high IT industry presence show a pattern of increased cybercrimes related to data theft and fraud.

Missing Values

Like Dataset 1, this dataset has minimal missing data, but some smaller cities might have fewer recorded cybercrimes, possibly due to underreporting or less coverage in the dataset.

Key Observations

1. **Year-wise Cybercrime Growth**: Dataset 1 clearly shows a consistent increase in the total number of cybercrimes reported across the states of India. The rise is most notable after 2010, correlating with greater internet adoption, ecommerce, and digital platforms.
2. **Geographic Disparities**: States with more urbanization, higher population density, and advanced IT infrastructure (e.g., Maharashtra, Delhi, Karnataka) report more cybercrimes, but they may also have better reporting mechanisms and higher awareness.
3. **Crime Type Dominance**: Fraud remains the most common type of cybercrime, followed by cases of online harassment and sexual exploitation, especially in major cities (Dataset 2). These trends could suggest that while India's digital space grows, many are vulnerable to frauds, and the internet has also become a platform for various forms of exploitation.
4. **City-Level Variations**: Some smaller cities show alarming growth in cybercrimes per capita, indicating that urbanization and technological changes affect smaller cities in distinct ways.

Loading and Inspecting Data

The first step in any data analysis project is to load the datasets into a suitable environment, such as Python using libraries like pandas. This allows us to explore the structure and contents of the data. For this project, we have two datasets: one containing state-level cybercrime data (cyber-crime.csv) and another focusing on city-level data (Dataset_CyberCrime_Sean.csv).

Once loaded, inspecting the data is crucial to understand its size, structure, and basic statistics. Key steps include checking the number of rows and columns, identifying data types, and detecting missing or inconsistent values. By using methods like .head(), .info(), and .describe(), we gain insights into the data's content and quality, allowing us to plan the analysis effectively.

This foundational step ensures the data is clean and ready for further exploration and visualization.

```
# Import required libraries import pandas as pd import

numpy as np import seaborn as sns import

matplotlib.pyplot as plt from sklearn.model_selection

import train_test_split from sklearn.preprocessing

import StandardScaler from sklearn.linear_model

import LogisticRegression from sklearn.tree import

DecisionTreeClassifier, plot_tree from sklearn.metrics

import (     accuracy_score, recall_score,

precision_score, f1_score,     roc_auc_score, roc_curve,

confusion_matrix

)

# Load datasets

data1 = pd.read_csv("/content/drive/MyDrive/DAV
course/cybercrime.csv")  # State-level data

data2 = pd.read_csv("/content/drive/MyDrive/DAV
course/Dataset_CyberCrime_Sean.csv")  # City-level data
```

```python
# Import necessary libraries
import pandas as pd

# Load both datasets
```

```
data1 = pd.read_csv("/content/drive/MyDrive/DAV
course/cybercrime.csv") data2 =
pd.read_csv("/content/drive/MyDrive/DAV
course/Dataset_CyberCrime_Sean.csv")

# Display the first few rows of each dataset
print("First 5 rows of Dataset 1:")
print(data1.head())
 print("\nFirst 5 rows of Dataset
2:") print(data2.head())
```

OUTPUT:

```
First 5 rows of Dataset 1:
         State/UT  2002  2003  2004  2005  2006  2007  2008
2009  2010  \
0     ANDHRA PRADESH   261   221   101    82   116    69   103
38   171
1                       ARUNANCHAL PRADESH     0     0     0     0
                        0     0     0     1     3
2                       ASSAM     2     0     0     1     1     0     2
4    18
3          BIHAR     0     0     0     0     0     0     0
0     2
4      CHHATTISGARH     0     0     0    46    30    57    20
50    50
  ...  2015  2016  2017  2018  2019  2020  2021  id
Total  \
0   ...   536   616   931  1207  1886  1899  1875  28
22500.051075
1   ...     6     4     1     7     8    30    47  12
328.212188
```

2 ... 483 696 1120 2022 2231 3530 4846 18
30828.187859
3 ... 242 309 433 374 1050 1512 1413 10
11057.742489

```
4   ...  103   90  171  139  175  297  352  22
3727.269980

   Total-Scale
0        4.352184
1        2.516155
2        4.488948
3        4.043666
4        3.571391

[5 rows x 24 columns]

First 5 rows of Dataset 2:
         City  Personal Revenge  Anger  Fraud  Extortion
Causing Disrepute  \
0   Agra              5.0    0.0   19.0       0.0              0.0
1   Allahabad              0.0    0.0  222.0      11.0              8.0
2   Amritsar              2.0    0.0    5.0       0.0              0.0
3   Asansol              6.0    1.0    3.0       0.0              0.0
4   Aurangabad              5.0    2.0   51.0       0.0
0.0

   Prank  Sexual Exploitation  Disrupt Public Service  \
0      0.0              0.0                    0.0
1      0.0              0.0                    0.0
2      0.0              2.0                    0.0
3      0.0              0.0                    0.0    4   0.0              21.0
         0.0

   Sale purchase illegal drugs  Developing own business
Spreading Piracy  \
0                             0.0             0.0          0.0
1                             0.0             0.0          0.0
```

2	0.0	0.0	0.0
3	0.0	0.0	0.0
4	0.0	0.0	0.0

	Psycho or Pervert	Steal Information	Abetment to Suicide	Others	Total
0	0.0	0.0	0.0	46.0	70.0
1	0.0	0.0	0.0	0.0	241.0
2	0.0	0.0	0.0	0.0	9.0
3	0.0	0.0	0.0	11.0	21.0
4	0.0	0.0	0.0	0.0	82.0

```python
# Check for duplicate rows in both datasets

print(f"Dataset 1 - Number of duplicate rows: {data1.duplicated().sum()}")

data1.drop_duplicates(inplace=True)

print(f"Dataset 1 - After removing duplicates: {data1.duplicated().sum()}")



print(f"\nDataset 2 - Number of duplicate rows: {data2.duplicated().sum()}")

data2.drop_duplicates(inplace=True)

print(f"Dataset 2 - After removing duplicates: {data2.duplicated().sum()}")
```

OUTPUT:

```
Dataset 1 - Number of duplicate rows: 0

Dataset 1 - After removing duplicates: 0

Dataset 2 - Number of duplicate rows: 0

Dataset 2 - After removing duplicates: 0
```

```python
# Display the shape of each dataset
print(f"Dataset 1 shape: {data1.shape}")
print(f"Dataset 2 shape: {data2.shape}")
```

OUTPUT:

```
Dataset 1 shape: (39, 24)

Dataset 2 shape: (189, 17)# Dataset 1 info

print("Dataset 1 information:")

print(data1.info())

# Dataset 2 info

print("\nDataset 2 information:")

print(data2.info())
```

OUTPUT:

Dataset 1 information:

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 39 entries, 0 to 38

Data columns (total 24 columns):

```
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   State/UT    39 non-null     object
 1   2002        39 non-null     int64
 2   2003        39 non-null     int64
 3   2004        39 non-null     int64
 4   2005        39 non-null     int64
 5   2006        39 non-null     int64
 6   2007        39 non-null     int64
 7   2008        39 non-null     int64
 8   2009        39 non-null     int64
 9   2010        39 non-null     int64
 10  2011        39 non-null     int64
 11  2012        39 non-null     int64
 12  2013        39 non-null     object
 13  2014        39 non-null     int64
 14  2015        39 non-null     int64
 15  2016        39 non-null     int64
 16  2017        39 non-null     int64
 17  2018        39 non-null     int64
 18  2019        39 non-null     int64
```

19 2020 39 non-null int64

20 2021 39 non-null int64

21 id 39 non-null int64

22 Total 39 non-null float64 23 Total-Scale 39 non-null float64 dtypes: float64(2), int64(20), object(2) memory usage: 7.4+ KB

None

Dataset 2 information:

<class 'pandas.core.frame.DataFrame'>

Index: 189 entries, 0 to 190

Data columns (total 17 columns):

#	Column	Non-Null Count	Dtype
---	------	--------------	-----
0	City	188 non-null	object
1	Personal Revenge	188 non-null	float64
2	Anger	188 non-null	float64
3	Fraud	188 non-null	float64
4	Extortion	188 non-null	float64
5	Causing Disrepute	188 non-null	float64
6	Prank	188 non-null	float64

7	Sexual Exploitation	188 non-null	float64
8	Disrupt Public Service	188 non-null	float64
9	Sale purchase illegal drugs	188 non-null	float64
10	Developing own business	188 non-null	float64
11	Spreading Piracy	188 non-null	float64
12	Psycho or Pervert	188 non-null	float64
13	Steal Information	188 non-null	float64
14	Abetment to Suicide	188 non-null	float64
15	Others	188 non-null	float64
16	Total	188 non-null	float64 dtypes:

float64(16), object(1) memory usage: 26.6+ KB None

Cleaning and Preprocessing

Check for missing values.
Handle duplicates.
Convert data types if necessary.

```
# Check for missing values print("Missing
values in Dataset 1:")
print(data1.isnull().sum())
 print("\nMissing values in Dataset 2:")
print(data2.isnull().sum())

# Import numpy import numpy as np # Importing the numpy library
and aliasing it as np

# Handle missing values
# Only impute for numeric columns numeric_cols_data1 =
data1.select_dtypes(include=np.number).columns
data1[numeric_cols_data1] =
data1[numeric_cols_data1].fillna(data1[numeric_cols_data1].mean(
))  numeric_cols_data2 =
data2.select_dtypes(include=np.number).columns
data2[numeric_cols_data2] =
data2[numeric_cols_data2].fillna(data2[numeric_cols_data2].media
n())
```

19 | Page

OUTPUT:

```
Missing values in Dataset 1:
State/UT      0
2002          0
2003          0
2004          0
2005          0
2006          0
2007          0
2008          0
2009          0
2010          0
2011          0
2012          0
2013          0
2014          0
2015          0
2016          0
2017          0
2018          0
2019          0
2020          0 2021          0 id          0 Total          0 Total-Scale
              0 dtype: int64

Missing values in Dataset 2:
City                  1
Personal Revenge          1
Anger                   1
Fraud                   1
Extortion                 1
Causing Disrepute         1
Prank                   1
Sexual Exploitation       1
```

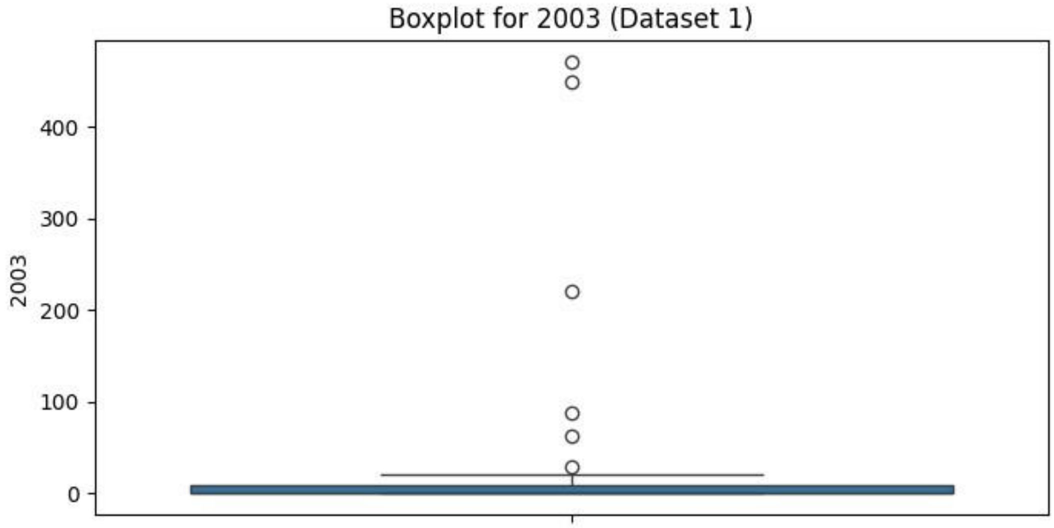
```
Disrupt Public Service        1
Sale purchase illegal drugs   1
Developing own business       1 Spreading Piracy          1
Psycho or Pervert             1
Steal Information             1
Abetment to Suicide           1
Others                        1 Total
1 dtype: int64
```

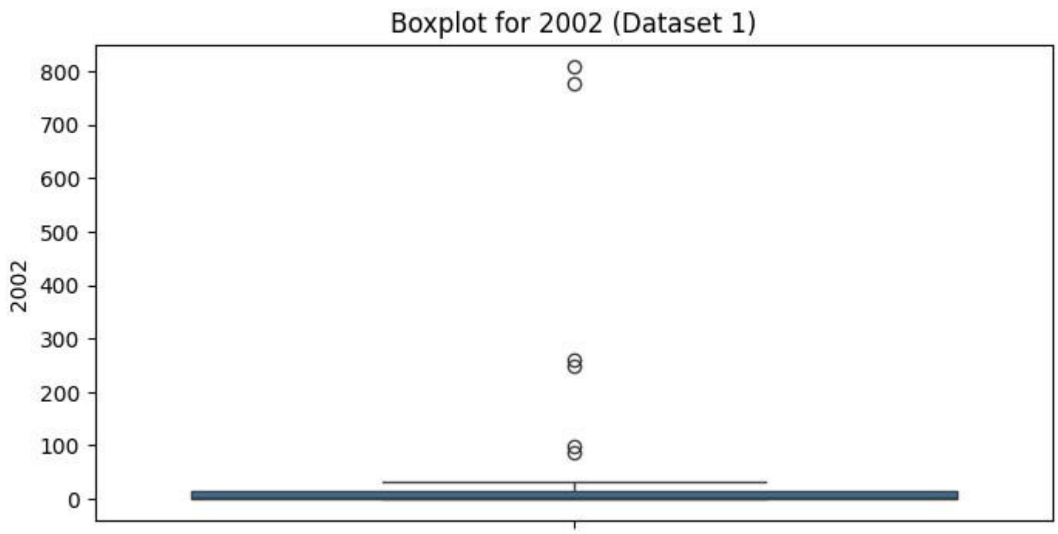
```python
# Remove duplicate rows
data1.drop_duplicates(inplace=True)
data2.drop_duplicates(inplace=True)
```

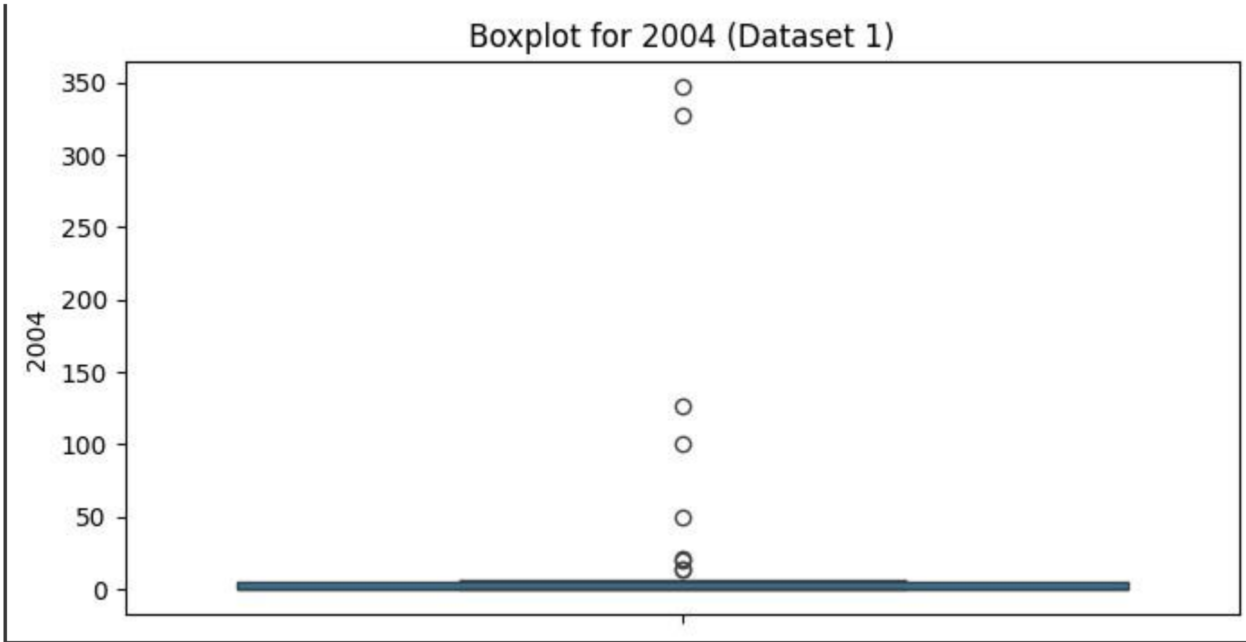
Outlier Detection and Treatment

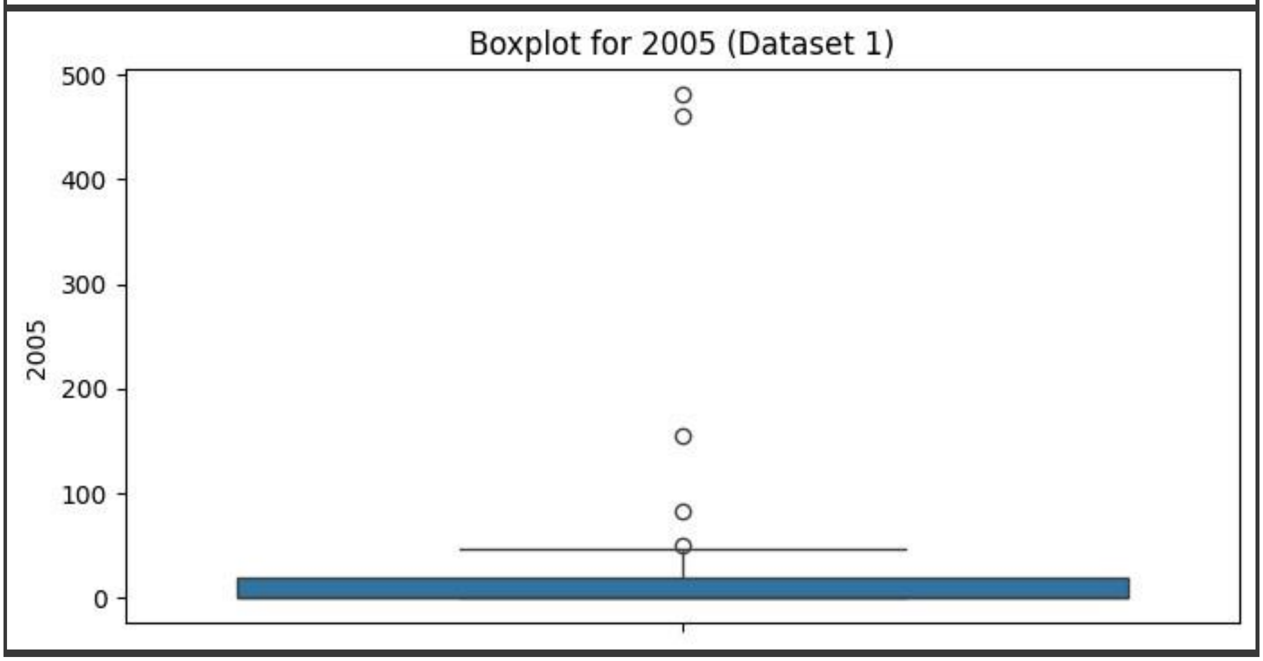
Detect outliers in numerical columns and decide whether to cap them, remove them, or keep them.

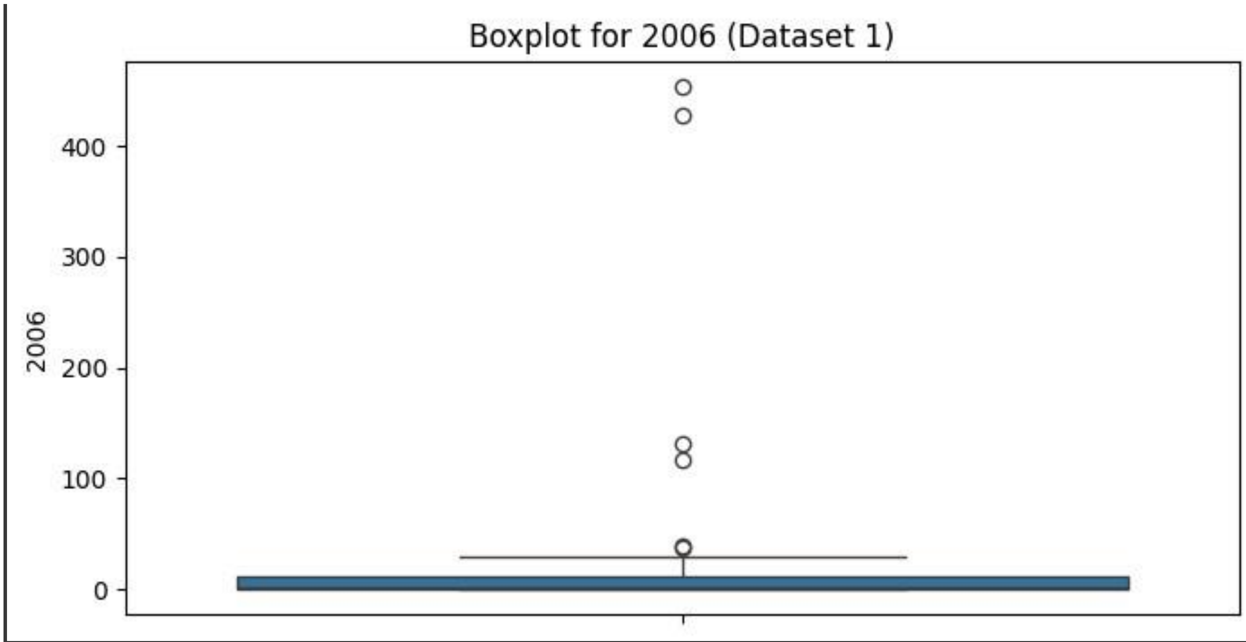
```
import seaborn as sns
import matplotlib.pyplot as
plt

# Example: Boxplot for outlier detection for col in
data1.select_dtypes(include='number').columns:
   plt.figure(figsize=(8, 4))
sns.boxplot(data=data1[col])
plt.title(f'Boxplot for {col} (Dataset 1)')
plt.show()

# Optionally cap outliers for col in
data1.select_dtypes(include='number').columns:
   Q1 = data1[col].quantile(0.25)
   Q3 = data1[col].quantile(0.75)    IQR = Q3 -
Q1    lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR    data1[col] =
data1[col].clip(lower=lower_bound,
upper=upper_bound)
```
OUTPUT:


Boxplot for 2003 (Dataset 1)
2003
0
100
200
300
400

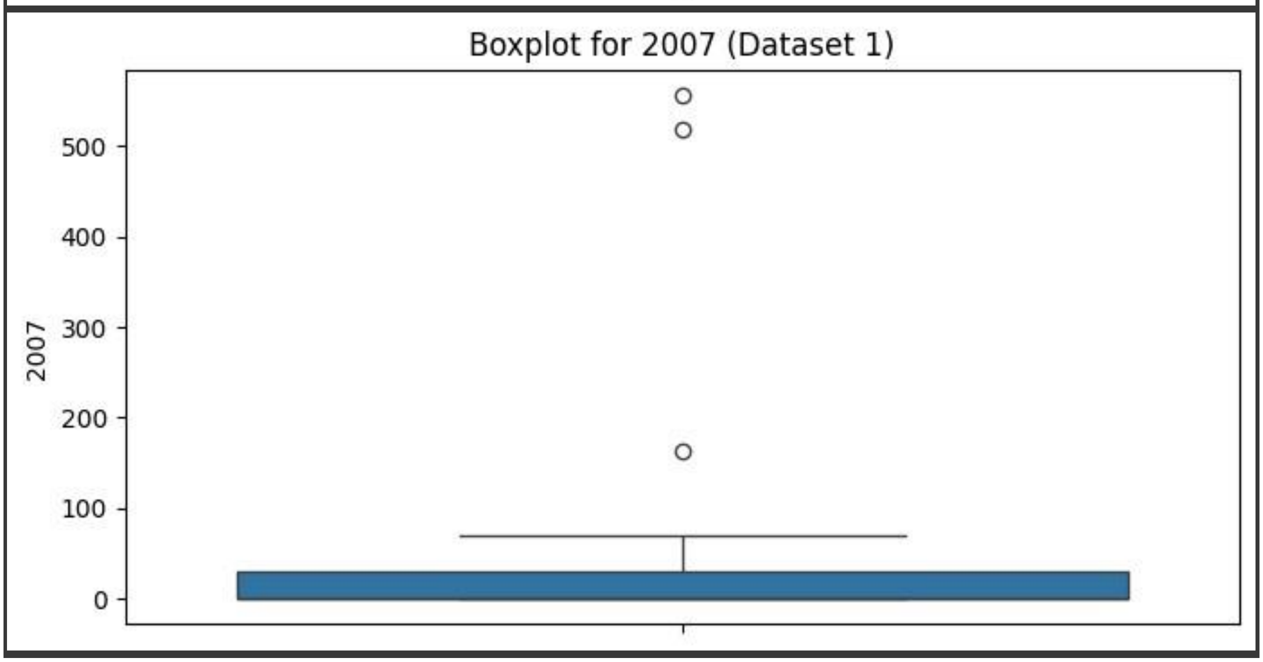

Boxplot for 2002 (Dataset 1)
2002
0
100
200
300
400
500
600
700
800


Boxplot for 2004 (Dataset 1)
2004
350
300
250
200
150
100
50
0

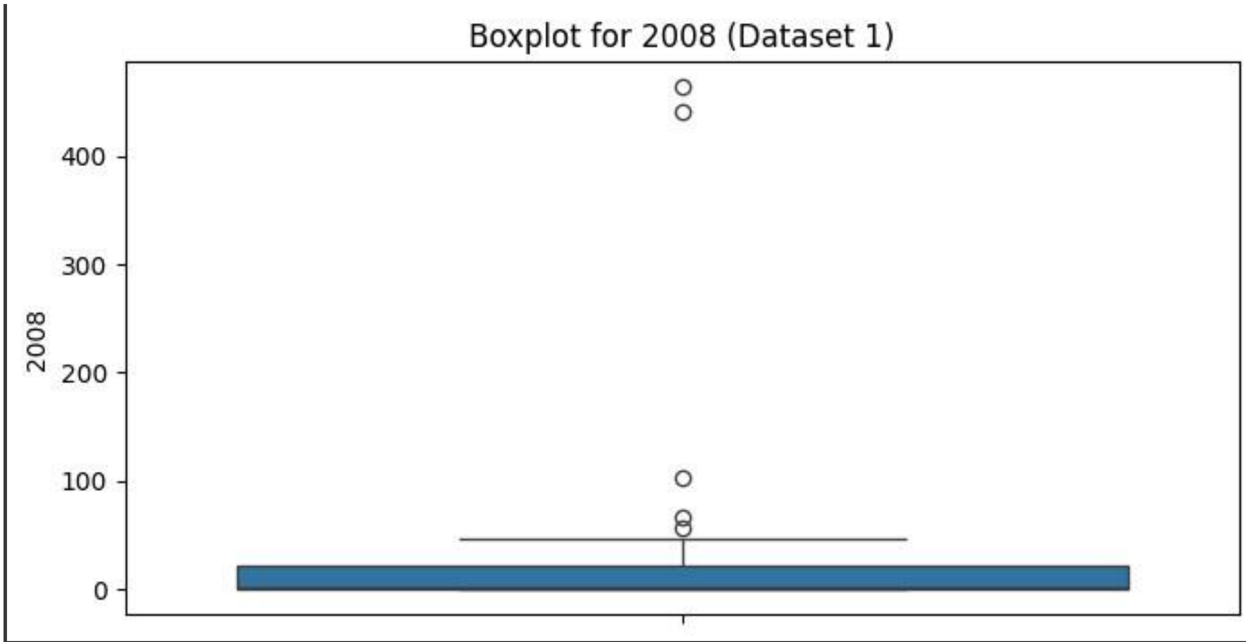

Boxplot for 2005 (Dataset 1)
2005
500
400
300
200
100
0

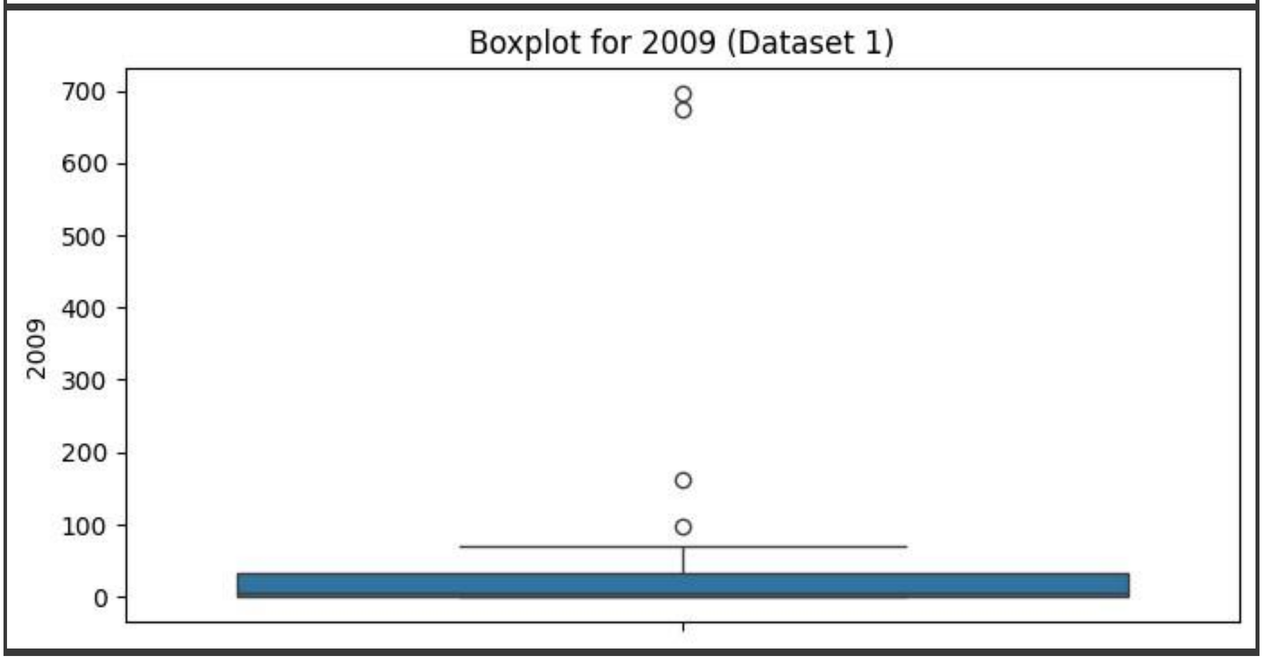


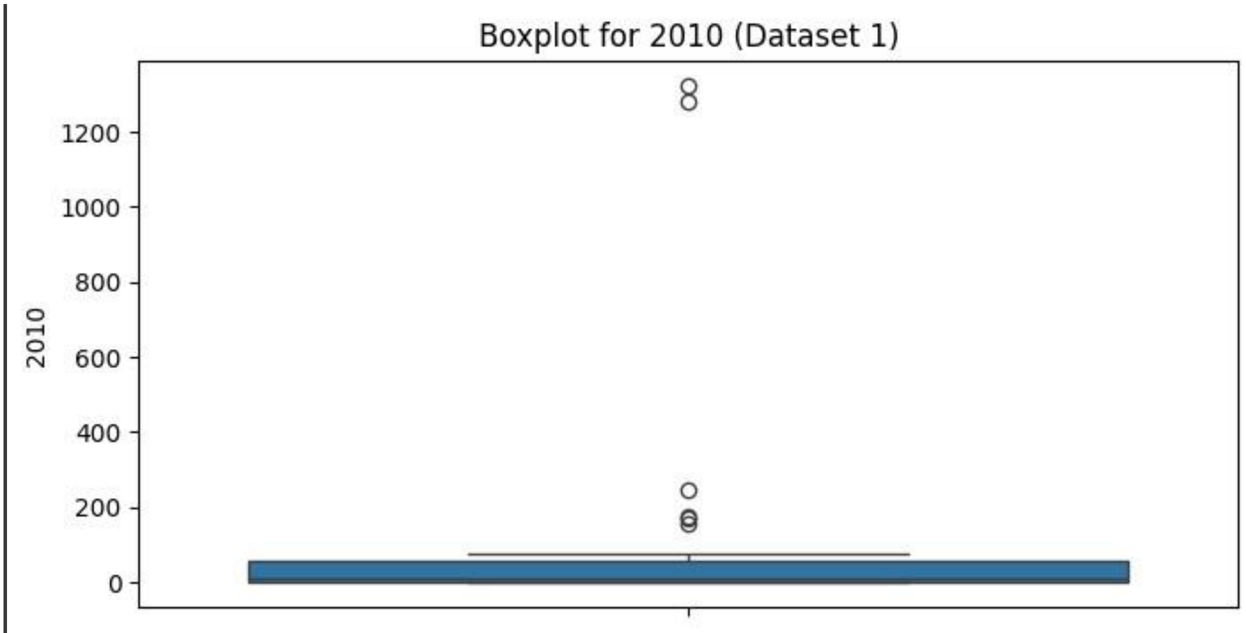
Boxplot for 2006 (Dataset 1)
2006
400
300
200
100
0

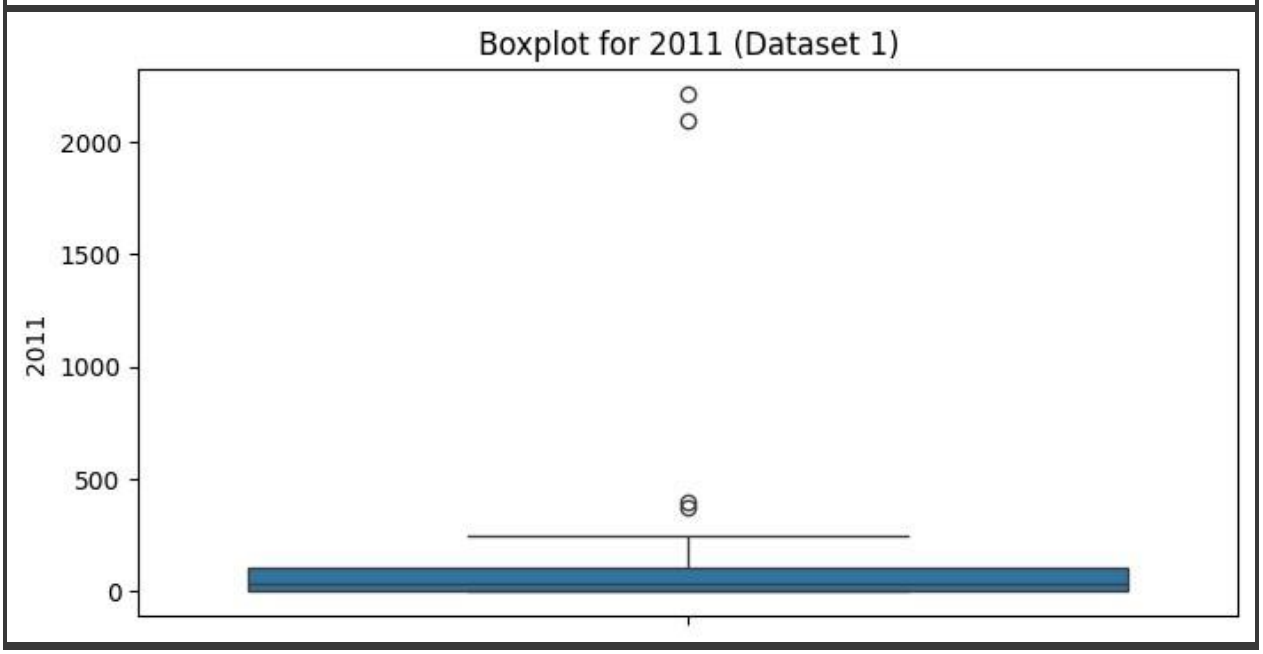


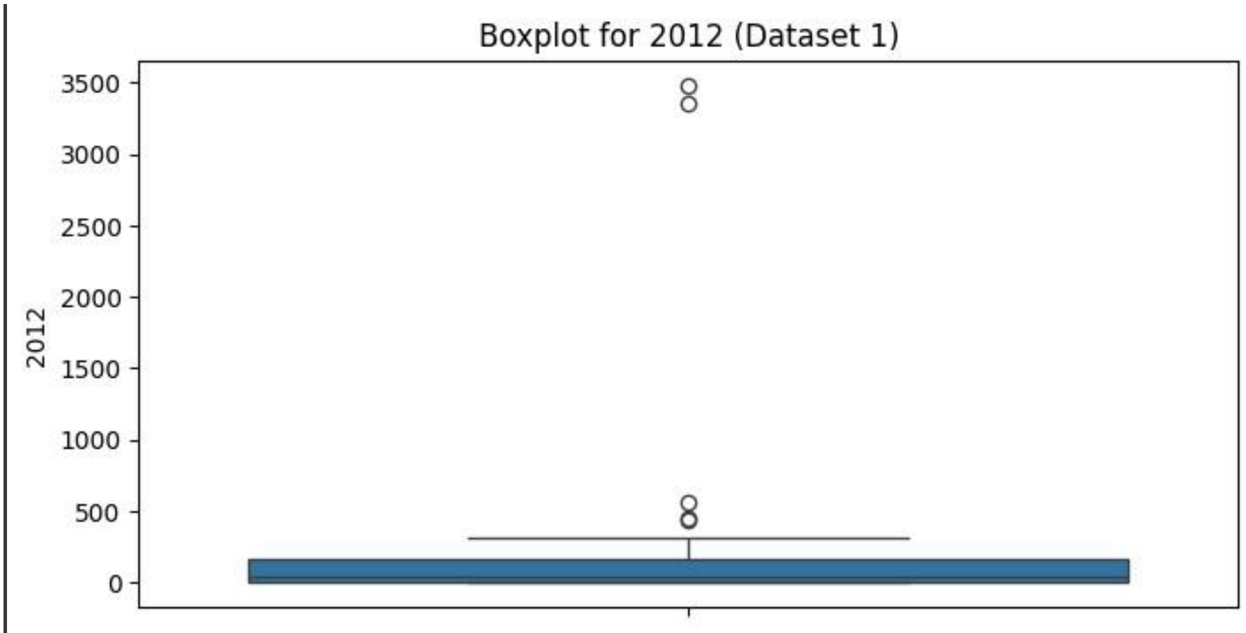
Boxplot for 2007 (Dataset 1)
2007
500
400
300
200
100
0

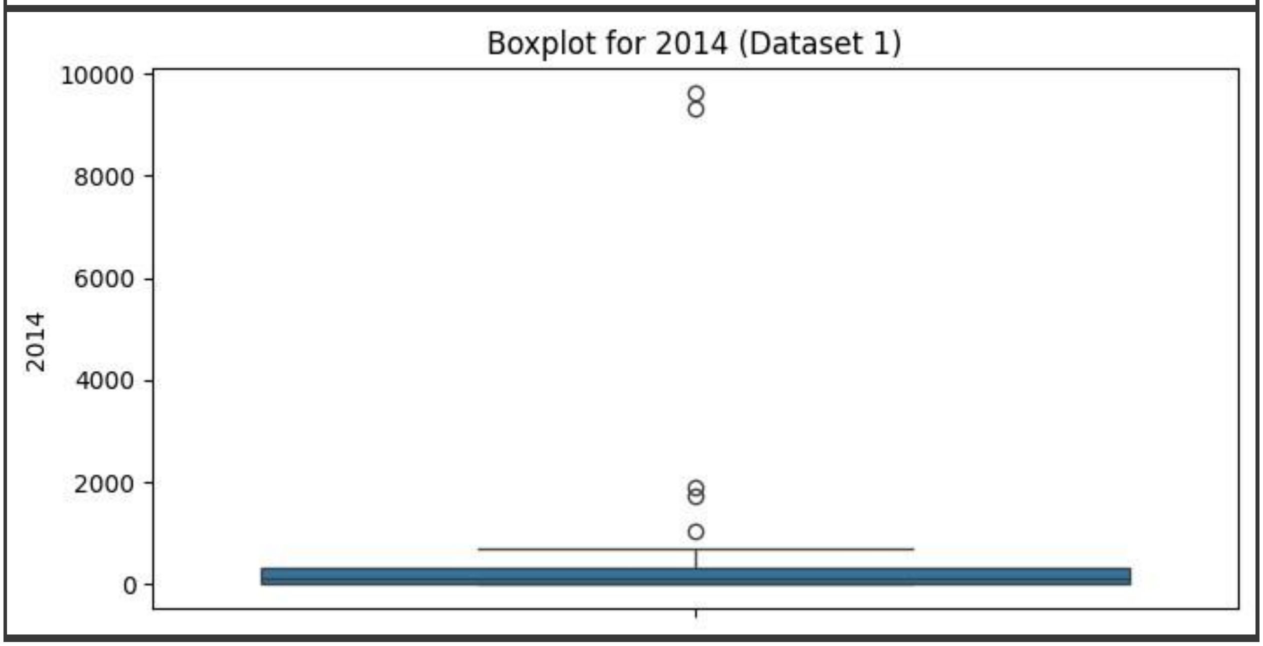

Boxplot for 2008 (Dataset 1)
2008
400
300
200
100
0

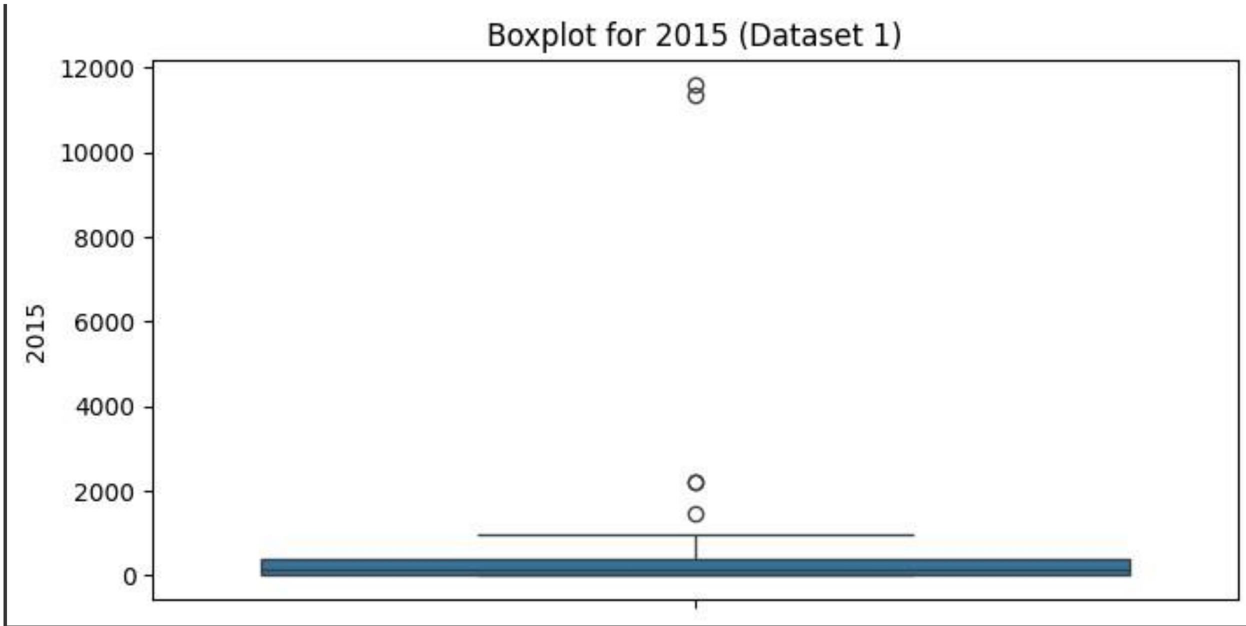

Boxplot for 2009 (Dataset 1)
2009
700
600
500
400
300
200
100
0

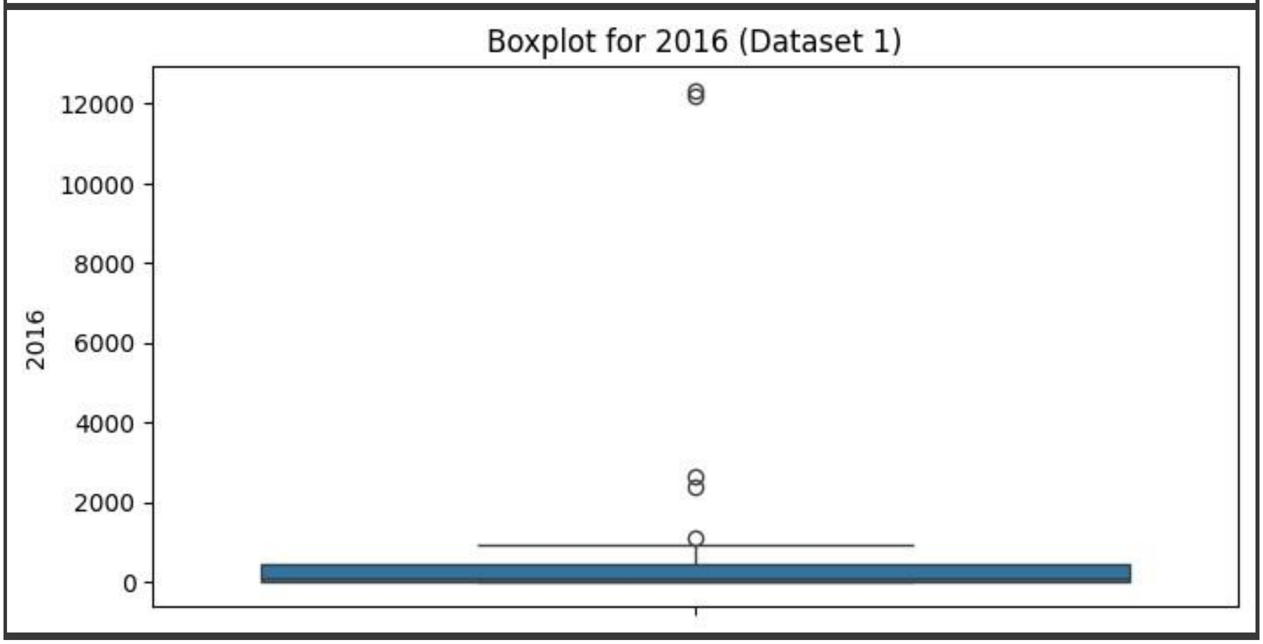

Boxplot for 2010 (Dataset 1)
2010
0
200
400
600
800
1000
1200

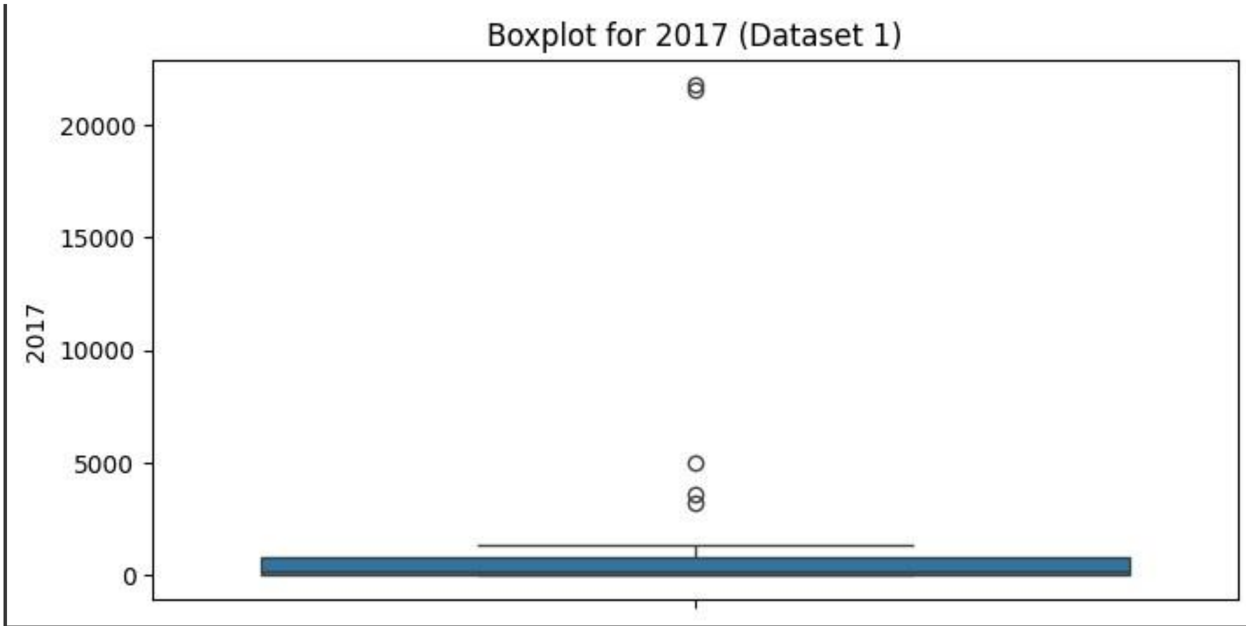

Boxplot for 2011 (Dataset 1)
2011
0
500
1000
1500
2000

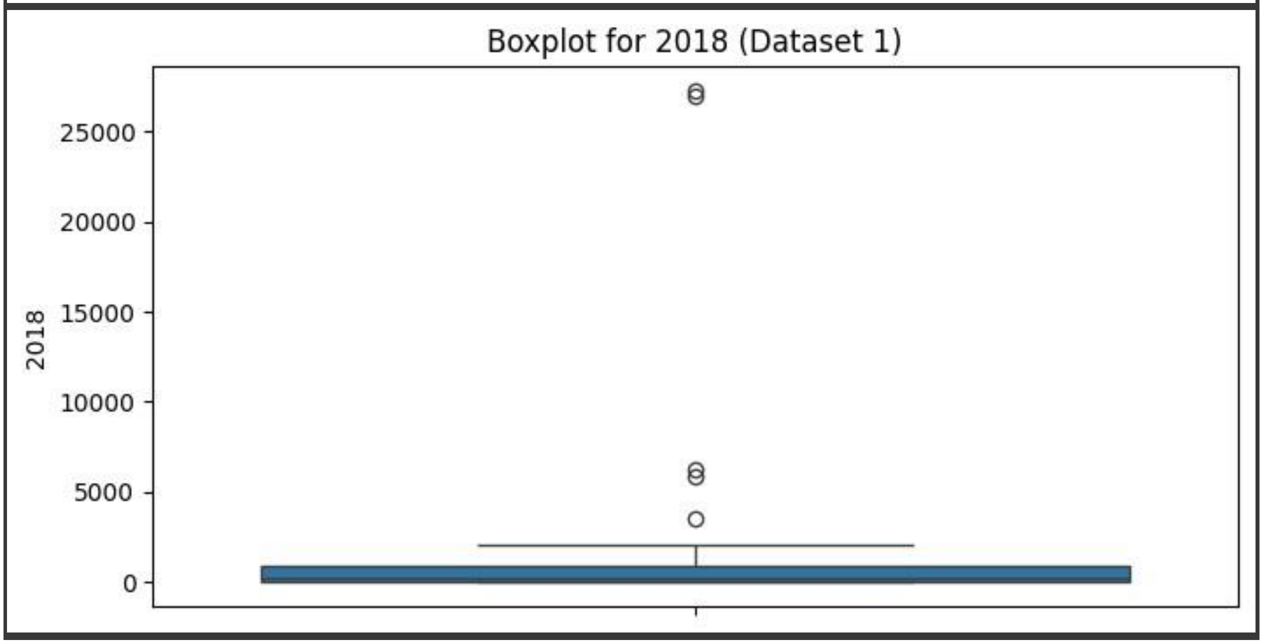

Boxplot for 2012 (Dataset 1)
2012
3500
3000
2500
2000
1500
1000
500
0

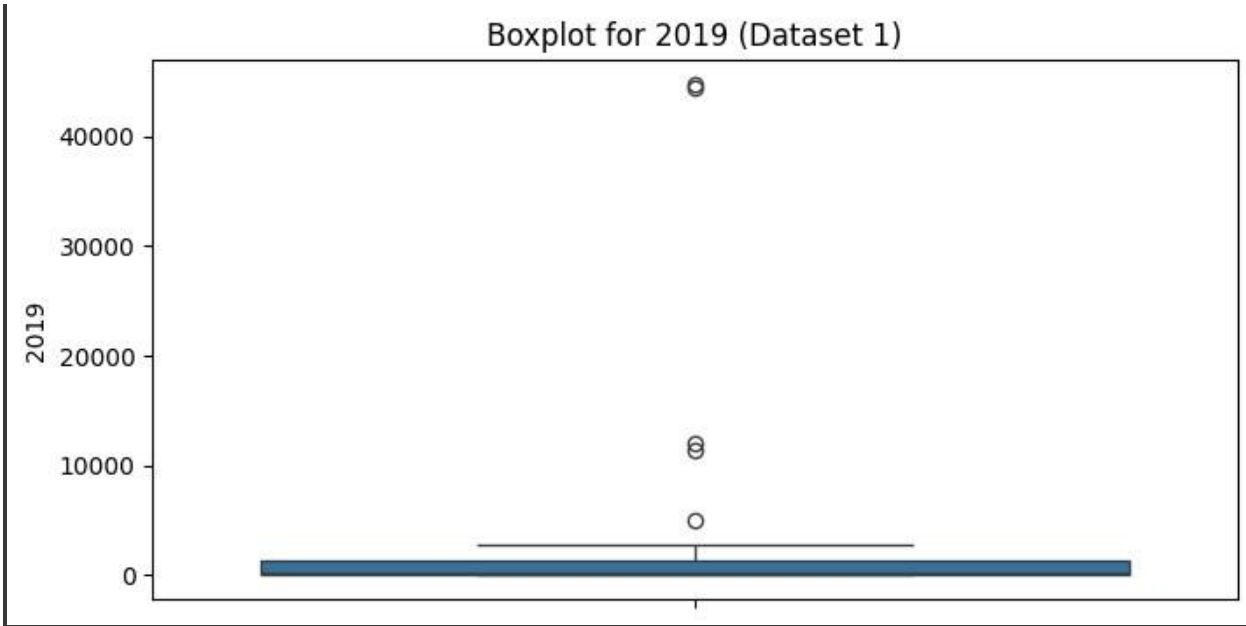

Boxplot for 2014 (Dataset 1)
2014
10000
8000
6000
4000
2000
0

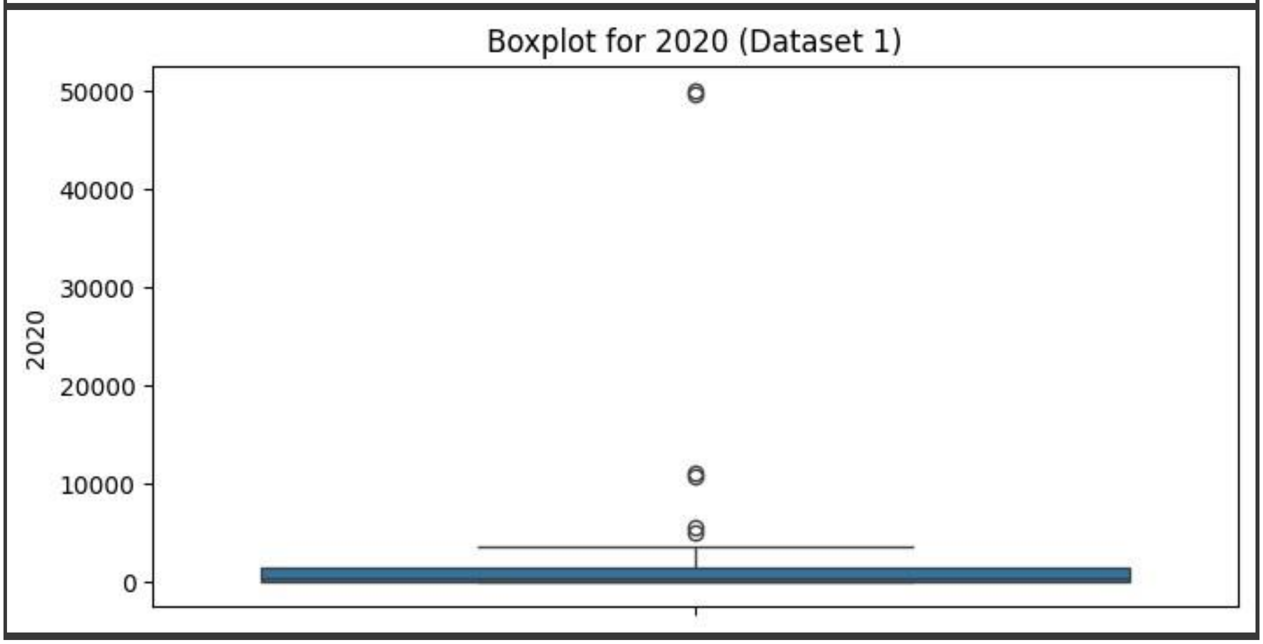

Boxplot for 2015 (Dataset 1)
12000
10000
8000
6000
4000
2000
0
2015

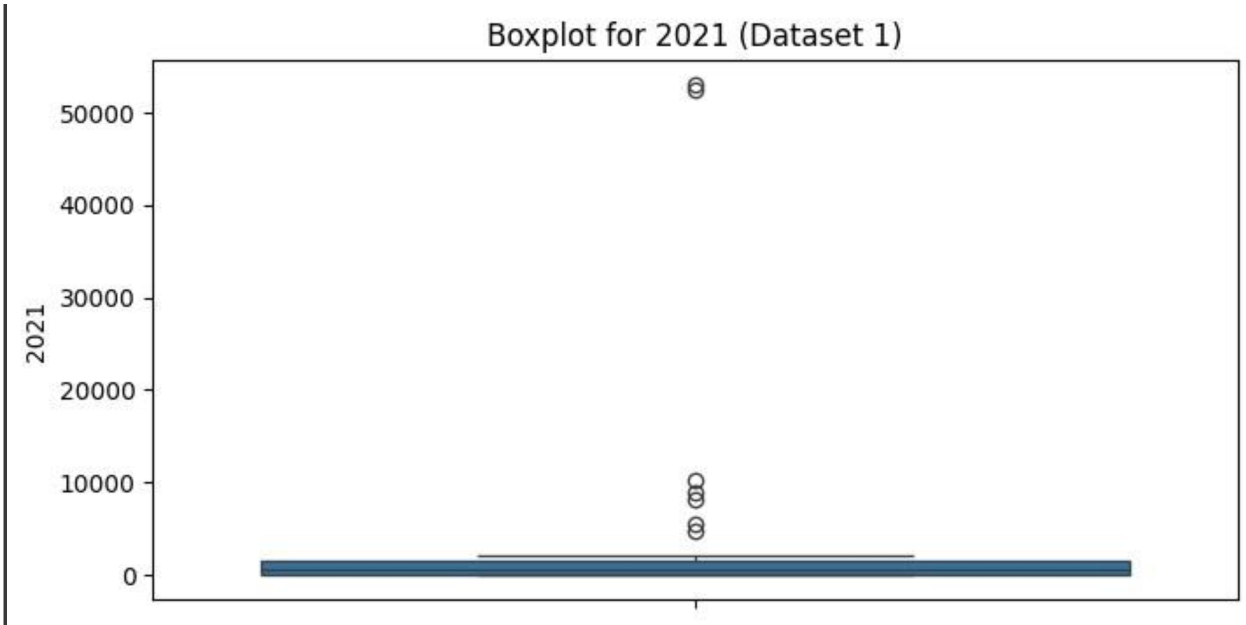

Boxplot for 2016 (Dataset 1)
12000
10000
8000
6000
4000
2000
0
2016

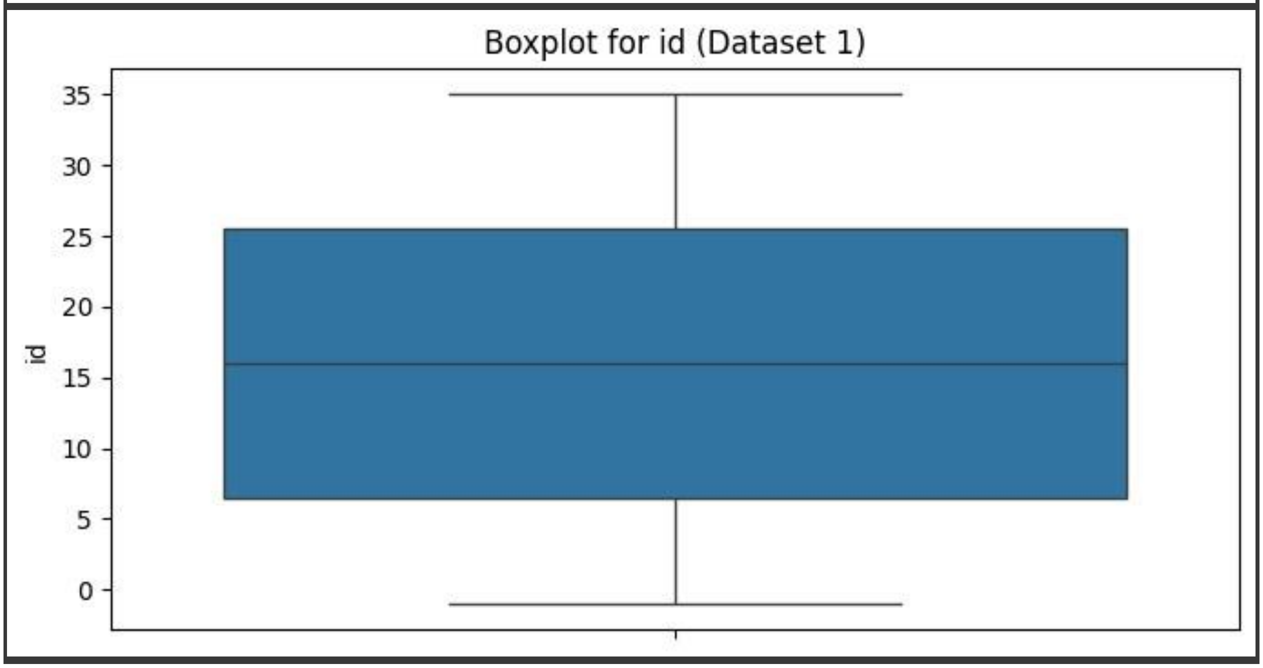

Boxplot for 2017 (Dataset 1)
2017
20000
15000
10000
5000
0

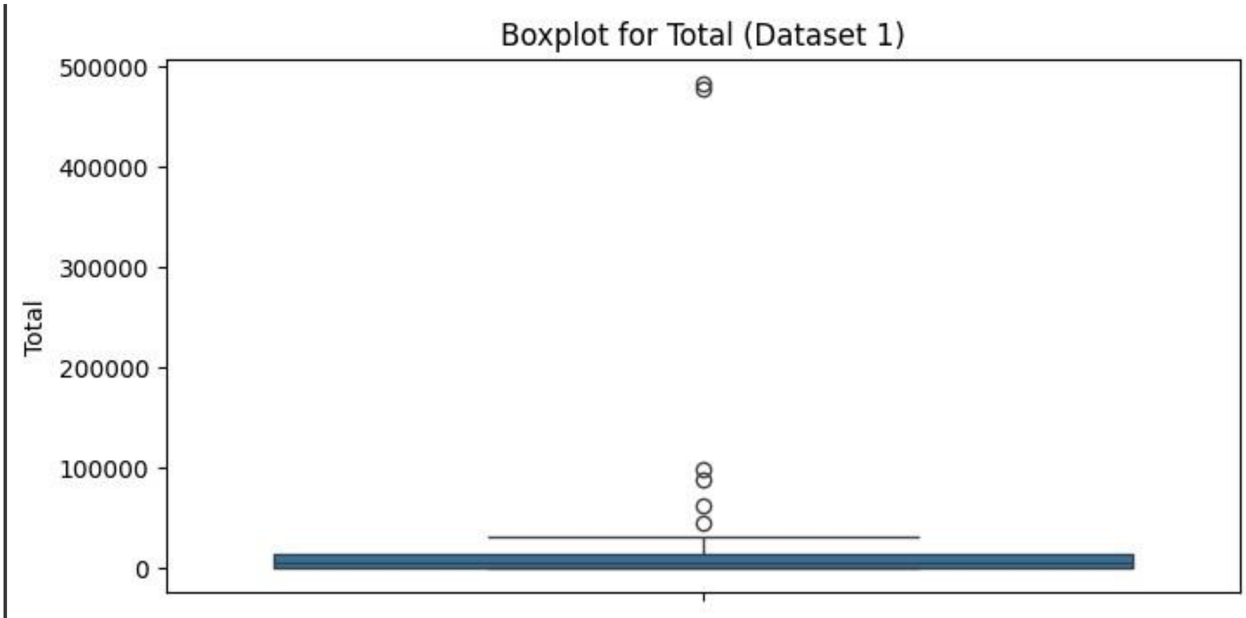

Boxplot for 2018 (Dataset 1)
2018
25000
20000
15000
10000
5000
0

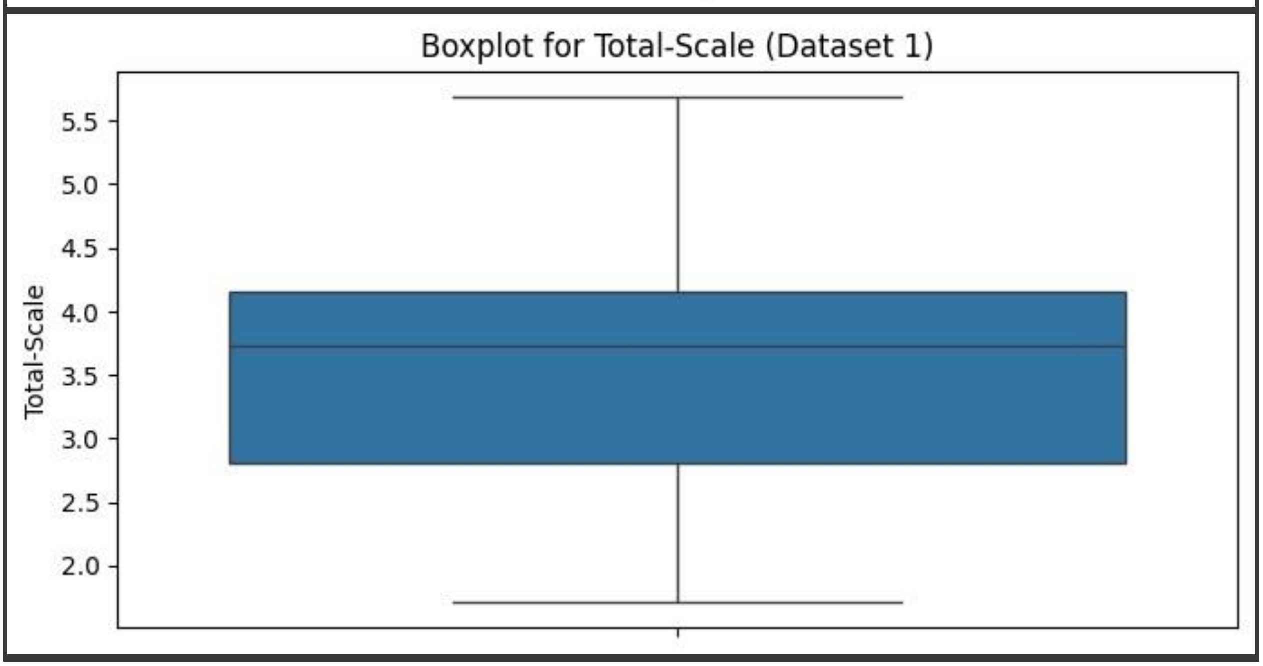

Boxplot for 2019 (Dataset 1)
2019
40000
30000
20000
10000
0


Boxplot for 2020 (Dataset 1)
2020
50000
40000
30000
20000
10000
0


Boxplot for 2021 (Dataset 1)
2021
50000
40000
30000
20000
10000
0


Boxplot for id (Dataset 1)
id
35
30
25
20
15
10
5
0





Encode Categorical Variables

Convert categorical variables into numerical form (if not already done).

```
# Check for categorical columns cat_columns1 =
data1.select_dtypes(include='object').columns
cat_columns2 =
data2.select_dtypes(include='object').columns
 print(f"Categorical columns in Dataset 1:
{cat_columns1}") print(f"Categorical columns in
Dataset 2: {cat_columns2}")
# One-hot encoding data1 =
pd.get_dummies(data1, drop_first=True)
data2 = pd.get_dummies(data2,
drop_first=True)
```

OUTPUT:

```
Categorical columns in Dataset 1: Index(['State/UT',
'2013'], dtype='object')
Categorical columns in Dataset 2: Index(['City'],
dtype='object')
```

Normalize or Scale Data

Scale numerical columns if necessary for models sensitive to magnitude differences (e.g., Logistic Regression, SVMs).

```
from sklearn.preprocessing import StandardScaler
 scaler =
StandardScaler()

# Scale numeric features (example) num_columns1 =
data1.select_dtypes(include='number').columns
data1[num_columns1] =
scaler.fit_transform(data1[num_columns1])
```

```
 num_columns2 =
data2.select_dtypes(include='number').columns
data2[num_columns2] =
scaler.fit_transform(data2[num_columns2])
```

Validate Target Variable

Ensure the target variable is correctly labeled and encoded.

```
# Check target variable distribution
if 'target' in data1.columns:
   print("Dataset 1 Target Distribution:")
print(data1['target'].value_counts()) else:
   print("Target column 'target' not found in data1")  # Print
a message if 'target' is not found
 if 'target' in
data2.columns:
   print("\nDataset 2 Target Distribution:")
print(data2['target'].value_counts()) else:
   print("Target column 'target' not found in data2")  # Print
a message if 'target' is not found


# Encode target if categorical and exists if 'target' in
data1.columns and data1['target'].dtype == 'object':
data1['target'] = data1['target'].apply(lambda x: 1 if x ==
```

```
'Positive' else 0) if 'target' in data2.columns and
data2['target'].dtype == 'object':    data2['target'] =
data2['target'].apply(lambda x: 1 if x == 'Positive' else 0)
```

Validate Target Variable

```
# Example: Creating a new feature combining existing ones
if 'crime_severity' in data1.columns and 'crime_frequency'
in data1.columns:

   data1['severity_frequency_ratio'] =
data1['crime_severity']
/ data1['crime_frequency']
```

Validate After Cleaning

```
# Check cleaned datasets print("Dataset 1 shape

after cleaning:", data1.shape) print("Dataset 2

shape after cleaning:", data2.shape)



# Preview first few rows print("\nFirst 5 rows of

Dataset 1 after cleaning:") print(data1.head())

 print("\nFirst 5 rows of Dataset 2 after
cleaning:") print(data2.head())
```

OUTPUT:

```
Dataset 1 shape after cleaning: (39, 94)
```

Dataset 2 shape after cleaning: (189, 95)

First 5 rows of Dataset 1 after cleaning:

```
       2002      2003      2004      2005      2006      2007      2008  \
0  1.966313  1.920982  1.767979  2.046033  1.818070
   2.035318  2.095474
1  -0.685878 -0.670350 -0.653678 -0.659375 -0.726358 -
   0.644908 0.701493
2  -0.534324 -0.670350 -0.653678 -0.602419 -0.641544 -
   0.644908 0.597420
3  -0.685878 -0.670350 -0.653678 -0.659375 -0.726358 -
   0.644908 0.701493
4  -0.685878 -0.670350 -0.653678  1.960599  1.818070
   1.569192
0.339239

       2009      2010      2011  ...  2013_519  2013_551
2013_552  2013_62  \
0   0.674194  2.086318  2.213845  ...     False     False     False
    False
1   -0.709677 -0.705107 -0.631715  ...     False     False     False
    False
2   -0.597472 -0.393796 -0.422301  ...     False     False     False
    False
3   -0.747079 -0.725861 -0.336072  ...     False     False     False
    False
4   1.123017  0.270335  0.156665  ...     False     False
False    False
```

```
  2013_637  2013_69  2013_7  2013_8752  2013_8994
2013_964
0        False    False   False     False     False     True
1        False    False   False     False     False    False
2        False    False   False     False     False    False
3        False    False   False     False     False    False   4    False
         False   False     False     False     False
[5 rows x 94 columns]
First 5 rows of Dataset 2 after cleaning:
  Personal Revenge     Anger    Fraud  Extortion  Causing
Disrepute  \
0              -0.269405 -0.304912 -0.286453  -0.285444
               0.300265
1              -0.291743 -0.304912 -0.242596  -0.253777
               0.274353
2              -0.282808 -0.304912 -0.289477  -0.285444
               0.300265
3              -0.264937 -0.297495 -0.289910  -0.285444
               0.300265
4              -0.269405 -0.290077 -0.279539  -0.285444        -
0.300265
     Prank  Sexual Exploitation  Disrupt Public Service  \
0  -0.204711           -0.303628              -0.269744
1  -0.204711           -0.303628              -0.269744
```

```
2  -0.204711          -0.299419            -0.269744
3  -0.204711          -0.303628            -0.269744
4  -0.204711          -0.259430            -0.269744
   Sale purchase illegal drugs  Developing own business  ...  \
0                 -0.254518              -0.316148  ...
1                 -0.254518              -0.316148  ...     2                 -
0.254518              -0.316148  ...
3                             -0.254518              -0.316148  ...
4                             -0.254518              -0.316148  ...
   City_Total UT(s)  City_Tripura  City_Uttar Pradesh
City_Uttarakhand  \
0       False       False             False           False
1       False       False             False           False
2       False       False             False           False
3       False       False             False           False
4       False       False             False           False
   City_Vadodara  City_Varanasi  City_Vasai Virar
City_Vijayawada  \
0              False       False          False          False
1              False       False          False          False
2              False       False          False          False
3              False       False          False          False
```

```
4          False     False        False
False

   City_Vishakhapatnam  City_West Bengal
0                False         False
1                False         False
2                False         False
3                False         False
4                False         False

[5 rows x 95 columns]
```

Exploratory Data Analysis (EDA)

```
# Exploratory Data Analysis for Dataset 1
print("Dataset 1 Statistical Summary:")
print(data1.describe().T)

# Check distribution of target variable if present if
'target' in data1.columns:
   print("\nDataset 1 Target Distribution:")
print(data1['target'].value_counts(normalize=True))
# Exploratory Data Analysis for Dataset 2
print("\nDataset 2 Statistical Summary:")
print(data2.describe().T)
 if 'target' in data2.columns:
   print("\nDataset 2 Target Distribution:")
print(data2['target'].value_counts(normalize=True))
```

OUTPUT:

```
Dataset 1 Statistical Summary:            count        mean
std      min      25%
50%  \
2002                 39.0  3.985416e-17  1.013072 -0.685878 -
                     0.685878 0.534324
2003                 39.0 -2.277381e-17  1.013072 -0.670350 -
                     0.670350 0.670350
2004                 39.0 -7.970832e-17  1.013072 -0.653678 -
                     0.653678 0.653678
2005                 39.0 -1.138690e-17  1.013072 -0.659375 -
                     0.659375 0.659375
2006                 39.0  6.832142e-17  1.013072 -0.726358 -
                     0.726358 -
0.556730
```

2007 39.0 -1.138690e-17 1.013072 -0.644908 -
0.644908 -
0.644908
2008 39.0 5.693451e-17 1.013072 -0.701493 -
0.701493 0.597420
2009 39.0 -5.693451e-17 1.013072 -0.747079 -
0.747079 0.522668
2010 39.0 7.970832e-17 1.013072 -0.767369 -
0.767369 0.518320
2011 39.0 -1.138690e-17 1.013072 -0.804173 -
0.742581 0.422301
2012 39.0 -1.024821e-16 1.013072 -0.755821 -
0.727868 0.525204
2014 39.0 5.693451e-17 1.013072 -0.866980 -
0.793712 0.416383
2015 39.0 7.970832e-17 1.013072 -0.830993 -
0.799555 -
0.384874
2016 39.0 -1.138690e-17 1.013072 -0.806159 -
0.780739 0.533220
2017 39.0 2.277381e-17 1.013072 -0.787440 -
0.769615 -
0.514650
2018 39.0 -2.277381e-17 1.013072 -0.774351 -
0.741842 0.457221
2019 39.0 3.416071e-17 1.013072 -0.741173 -
0.730975 -
0.542525
2020 39.0 -2.846726e-17 1.013072 -0.777336 -
0.752595 0.524515
2021 39.0 -4.554761e-17 1.013072 -0.793403 -
0.769146 -
0.393530 id 39.0 2.277381e-17 1.013072 -1.534065 -
0.860321 0.006910

Total 39.0 2.846726e-17 1.013072 -0.820556 -0.773359
0.407895
Total-Scale 39.0 -2.960595e-16 1.013072 -1.808626 -
0.725189
0.173139

75% max
2002 0.374998 1.966313
2003 0.366183 1.920982
2004 0.314985 1.767979
2005 0.422788 2.046033
2006 0.291413 1.818070
2007 0.559252 2.365491
2008 0.417293 2.095474
2009 0.449782 2.245074
2010 0.374106 2.086318
2011 0.439989 2.213845
2012 0.418228 2.137372
2014 0.332782 2.022524
2015 0.342688 2.056052
2016 0.369889 2.095829
2017 0.409115 2.177210
2018 0.401950 2.117637
2019 0.402828 2.103531
2020 0.330592 1.955374 2021 0.323888 1.963439 id
0.846500 1.699910 Total 0.325219 1.973085
Total-Scale 0.605024 2.108130

Dataset 2 Statistical Summary: count
mean std min \
Personal Revenge 189.0 -1.879743e-17 1.002656
0.291743
Anger 189.0 -3.759485e-17 1.002656 0.304912
Fraud 189.0 -9.398713e-18 1.002656 -
0.290558

Extortion 189.0 1.879743e-17 1.002656 0.285444
Causing Disrepute 189.0 0.000000e+00 1.002656 -
0.300265
Prank 189.0 2.819614e-17 1.002656 0.204711
Sexual Exploitation 189.0 3.759485e-17 1.002656 -
0.303628
Disrupt Public Service 189.0 1.879743e-17 1.002656
0.269744
Sale purchase illegal drugs 189.0 -6.579099e-17 1.002656
0.254518
Developing own business 189.0 -2.819614e-17 1.002656
0.316148
Spreading Piracy 189.0 3.759485e-17 1.002656 -
0.170865
Psycho or Pervert 189.0 -9.398713e-18 1.002656
0.183435
Steal Information 189.0 0.000000e+00 1.002656
0.215988
Abetment to Suicide 189.0 -2.819614e-17 1.002656
0.168005
Others 189.0 -2.819614e-17 1.002656 0.292093
Total 189.0 3.759485e-17 1.002656 -
0.298105

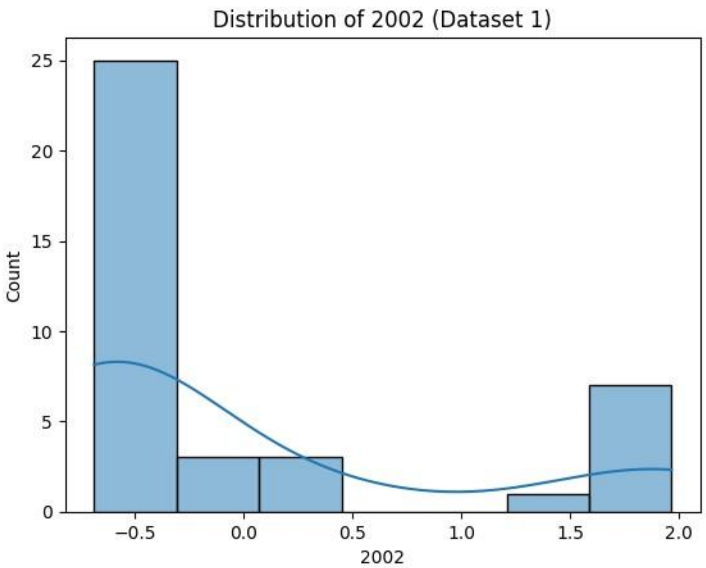
25% 50% 75% max
Personal Revenge -0.291743 -0.273873 -0.220261
6.275668
Anger -0.304912 -0.290077 -0.223317 5.792470
Fraud -0.289477 -0.281052 -0.214726 6.221471
Extortion -0.285444 -0.268171 -0.213474 6.738859
Causing Disrepute -0.300265 -0.290548 -0.219291
5.769572
Prank -0.204711 -0.204711 -0.181979
7.666367

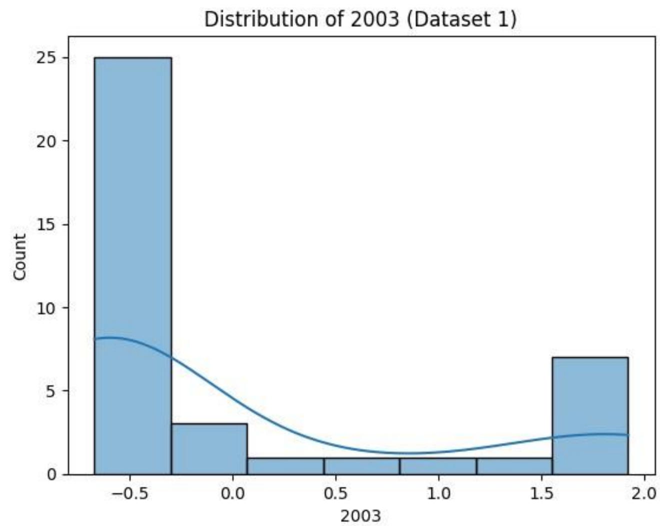
Sexual Exploitation	-0.303628	-0.276268	-0.181558	6.627027
Disrupt Public Service	-0.269744	-0.269744	-0.183916	7.626392
Sale purchase illegal drugs	-0.254518	-0.254518	-0.254518	7.228303
Developing own business	-0.316148	-0.316148	-0.235944	5.298129
Spreading Piracy	-0.170865	-0.170865	-0.158683	8.002983
Psycho or Pervert	-0.183435	-0.183435	-0.183435	8.746527
Steal Information	-0.215988	-0.215988	-0.215988	6.775579
Abetment to Suicide	-0.168005	-0.168005	-0.168005	7.392237
Others	-0.291362	-0.282949	-0.216017	6.155394
Total	-0.296152	-0.280987	-0.196308	6.215177

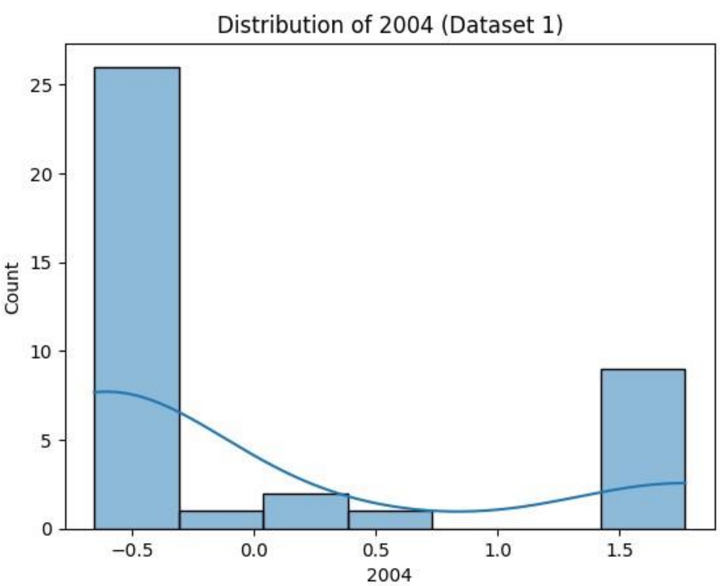
Univariate Analysis

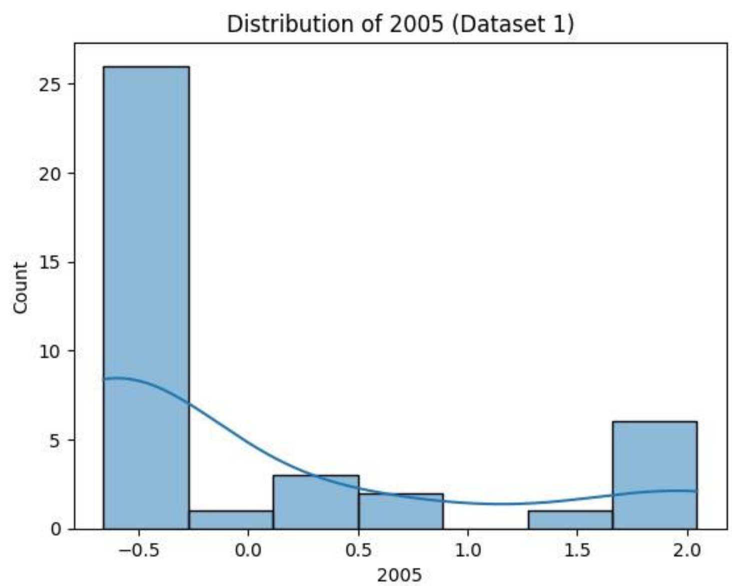
```
import seaborn as sns
import matplotlib.pyplot as
plt

# Univariate analysis for numeric features for col in
data1.select_dtypes(include='number').columns:
   sns.histplot(data1[col], kde=True)
plt.title(f'Distribution of {col} (Dataset 1)')     plt.show()
 for col in
data2.select_dtypes(include='number').columns:
   sns.histplot(data2[col], kde=True)
plt.title(f'Distribution of {col} (Dataset 2)')     plt.show()
```

OUTPUT:



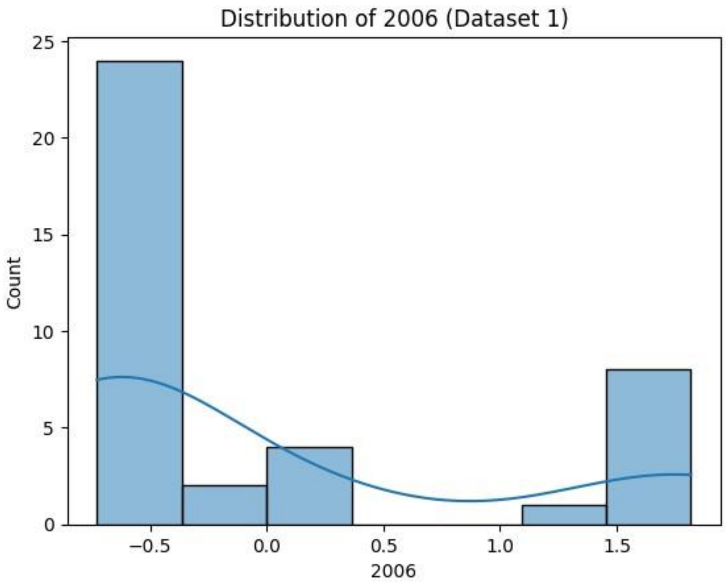


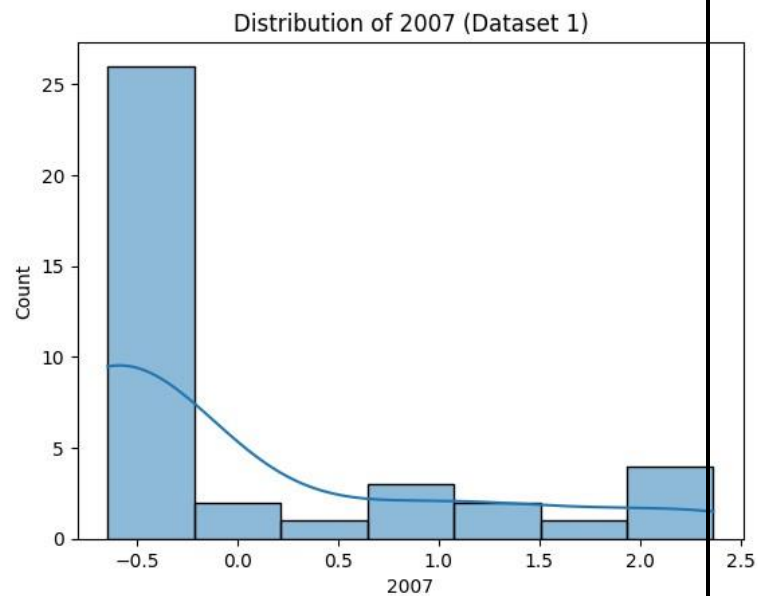




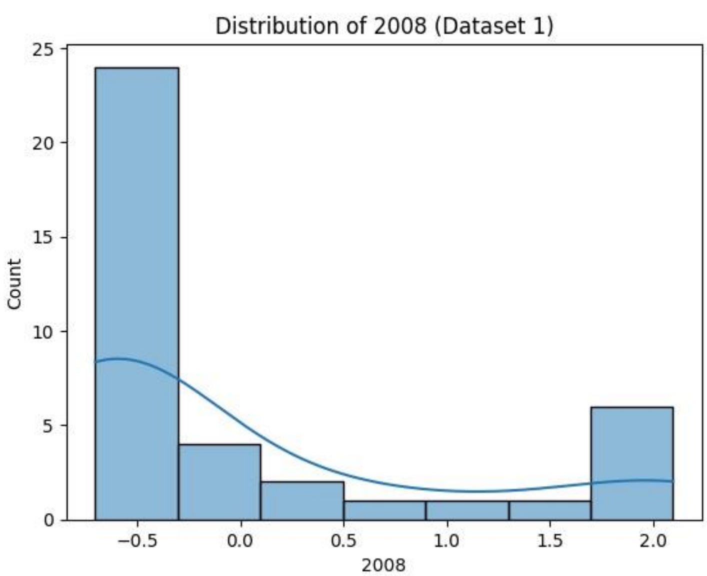
[redacted]

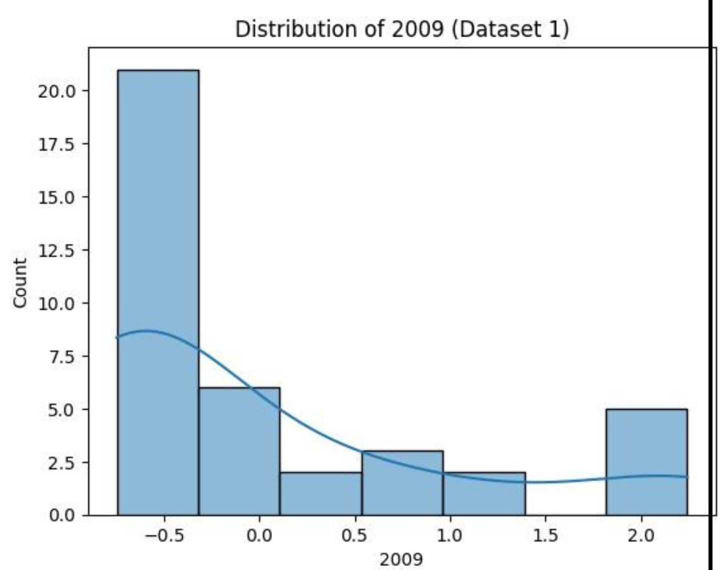
[redacted]

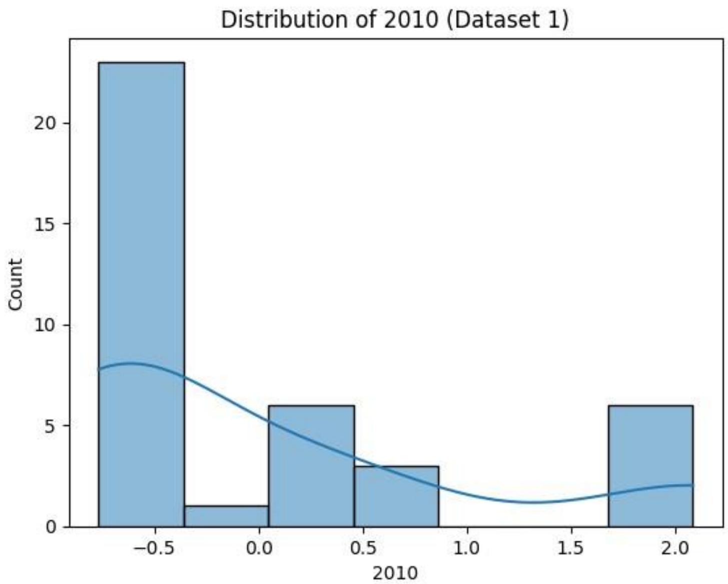


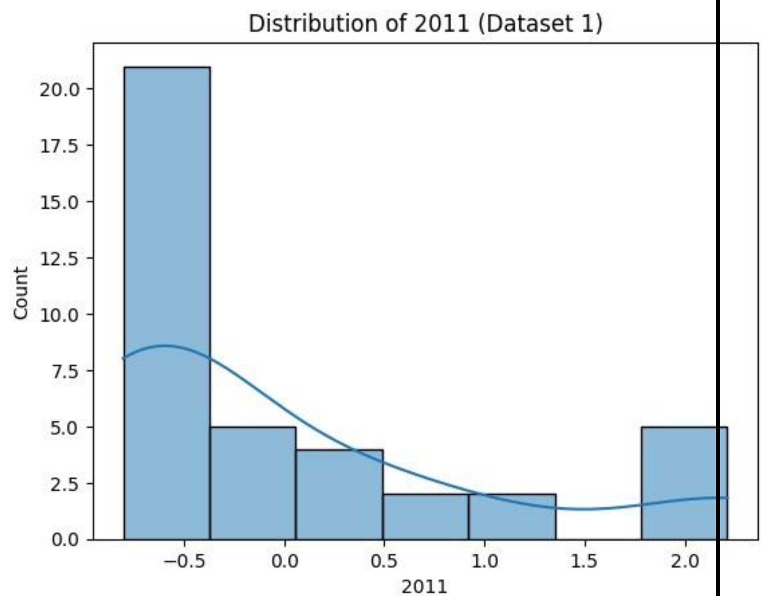


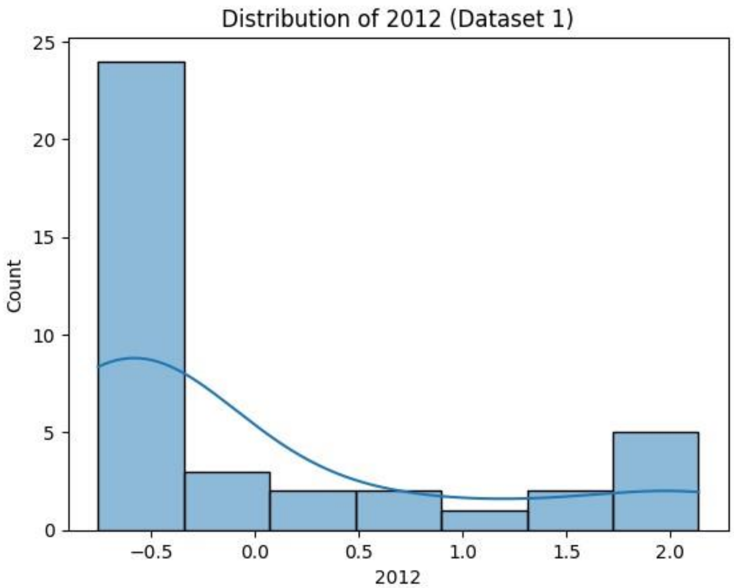
[redacted]

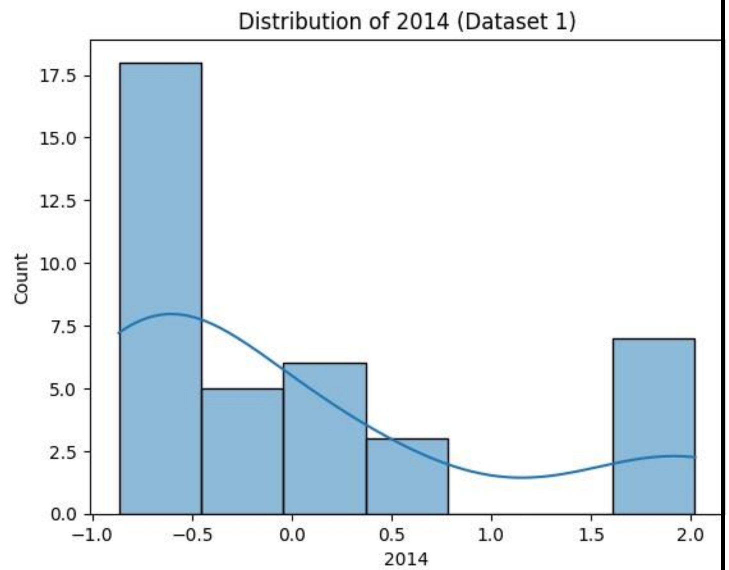


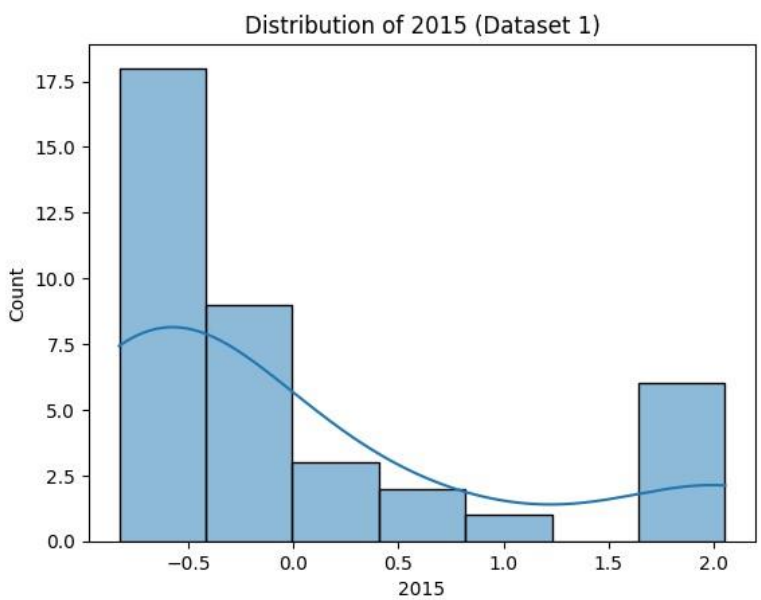


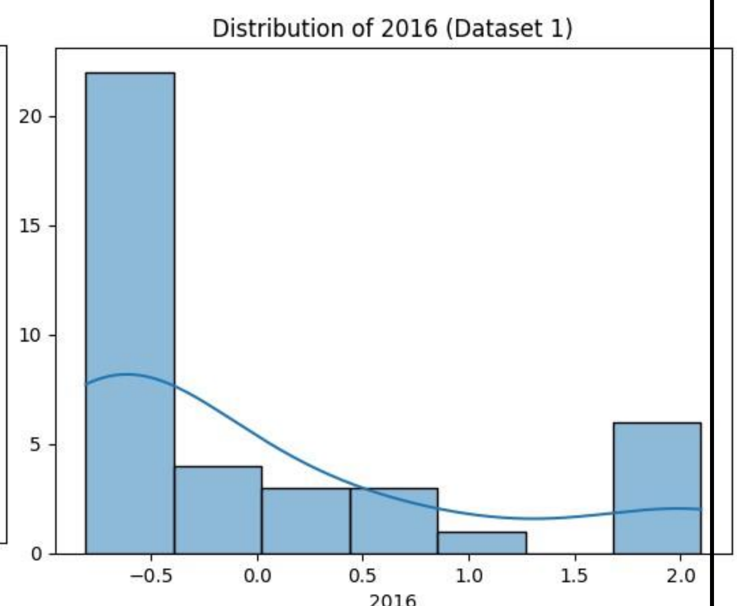

Distribution of 2010 (Dataset 1)
Count
2010
0
5
10
15
20
−0.5
0.0
0.5
1.0
1.5
2.0

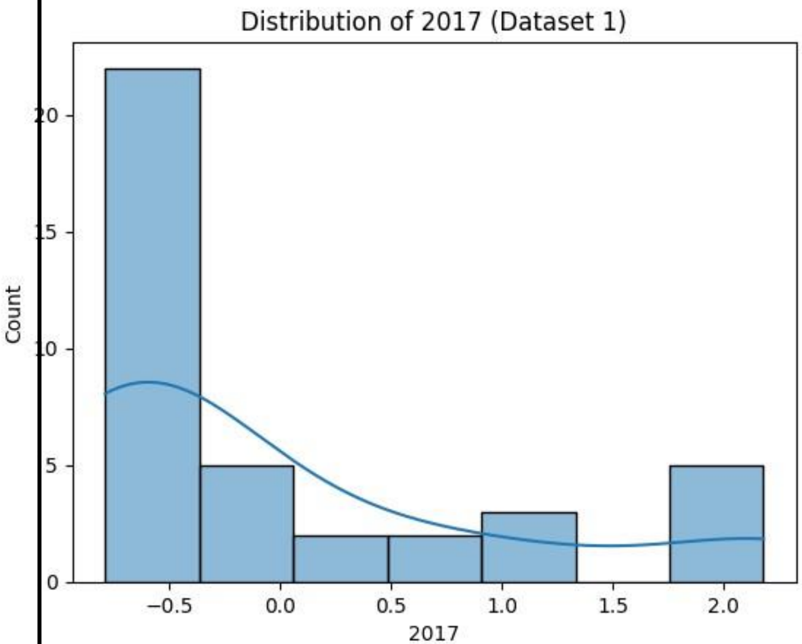

Distribution of 2011 (Dataset 1)
Count
2011
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
−0.5
0.0
0.5
1.0
1.5
2.0

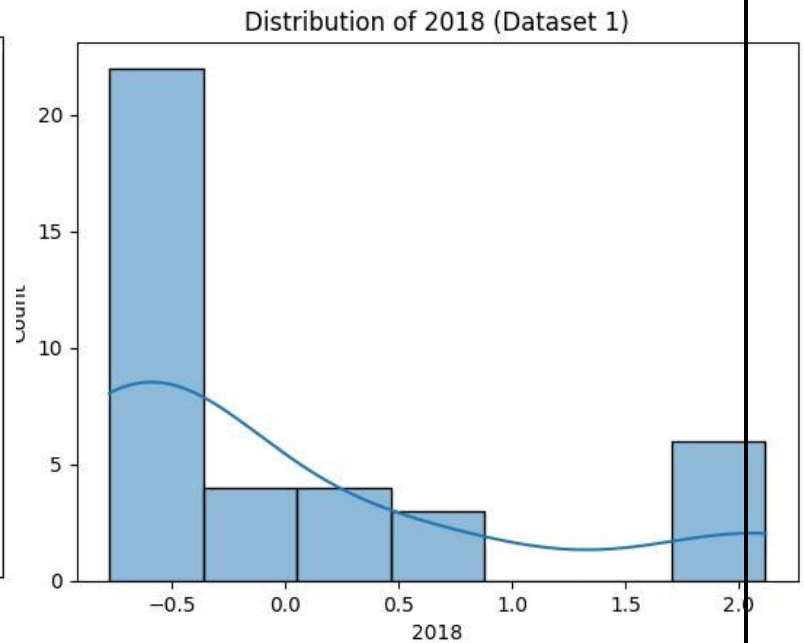

Distribution of 2012 (Dataset 1)
Count
2012
0
5
10
15
20
25
−0.5
0.0
0.5
1.0
1.5
2.0

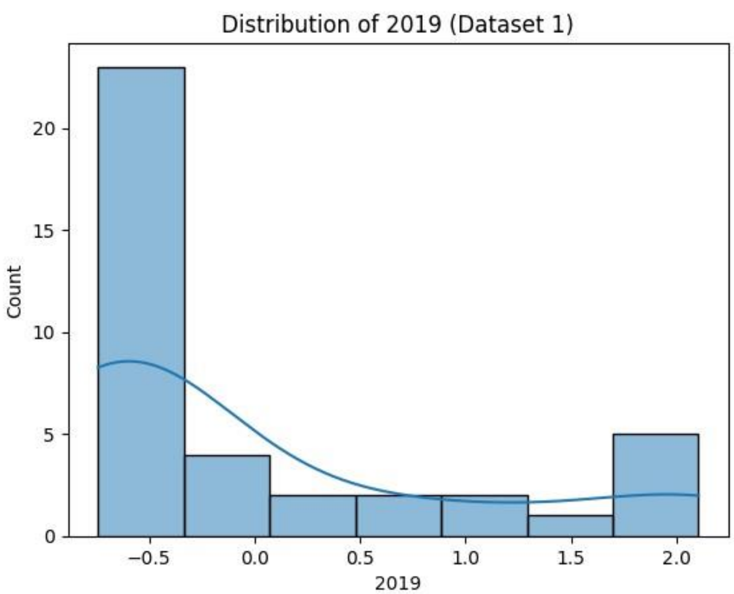

Distribution of 2014 (Dataset 1)
Count
2014
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

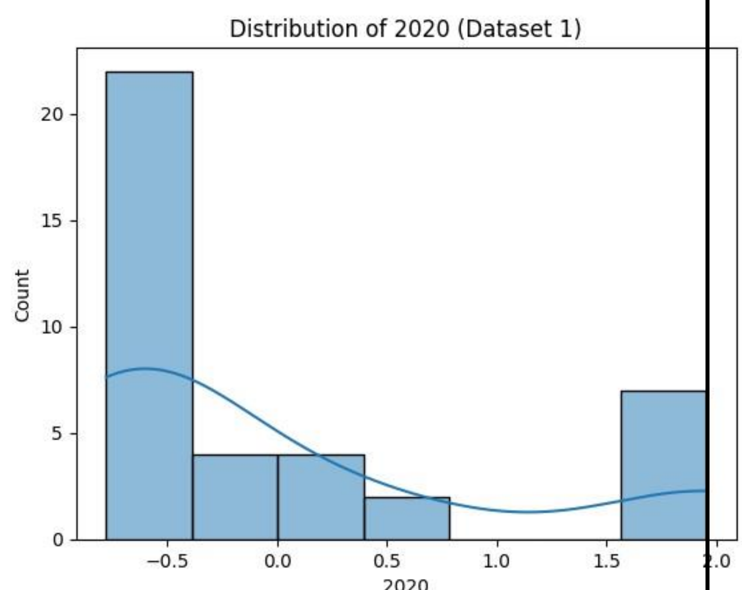

Distribution of 2015 (Dataset 1)
Count
2015
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.5
0.0
0.5
1.0
1.5
2.0

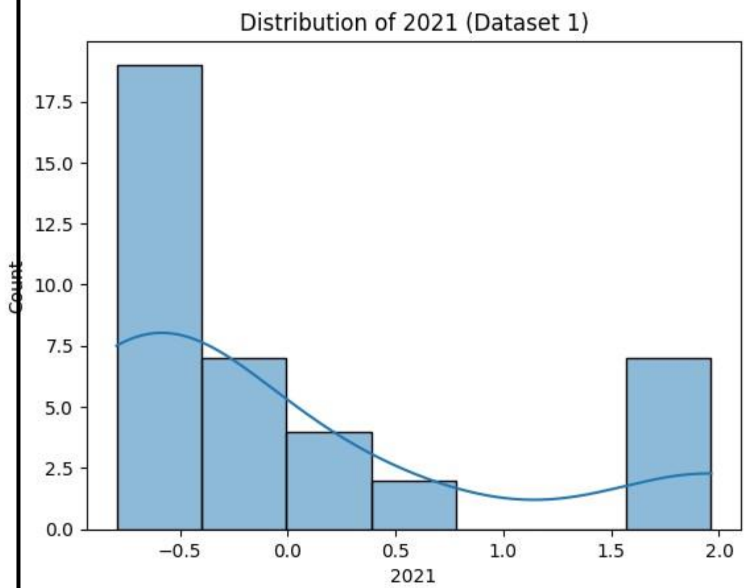

Distribution of 2016 (Dataset 1)
2016
0
5
10
15
20
−0.5
0.0
0.5
1.0
1.5
2.0


Distribution of 2017 (Dataset 1)
Count
2017

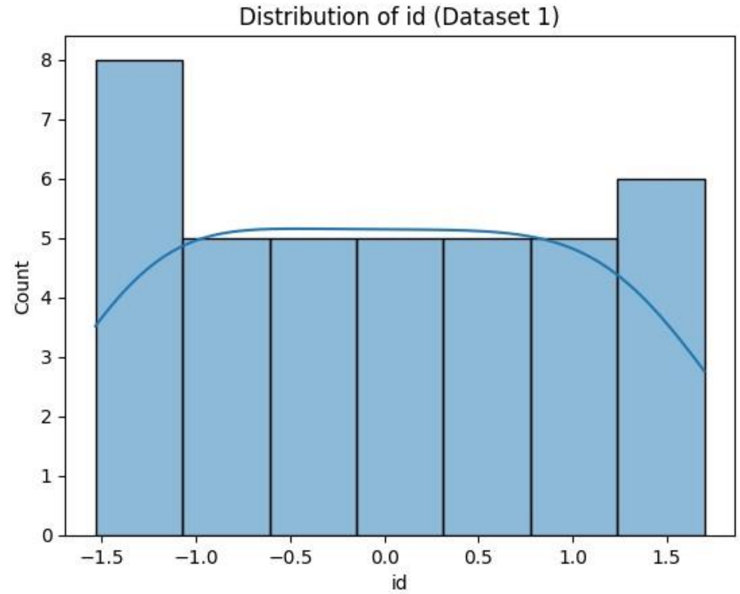

Distribution of 2018 (Dataset 1)
2018


Distribution of 2019 (Dataset 1)
Count
2019

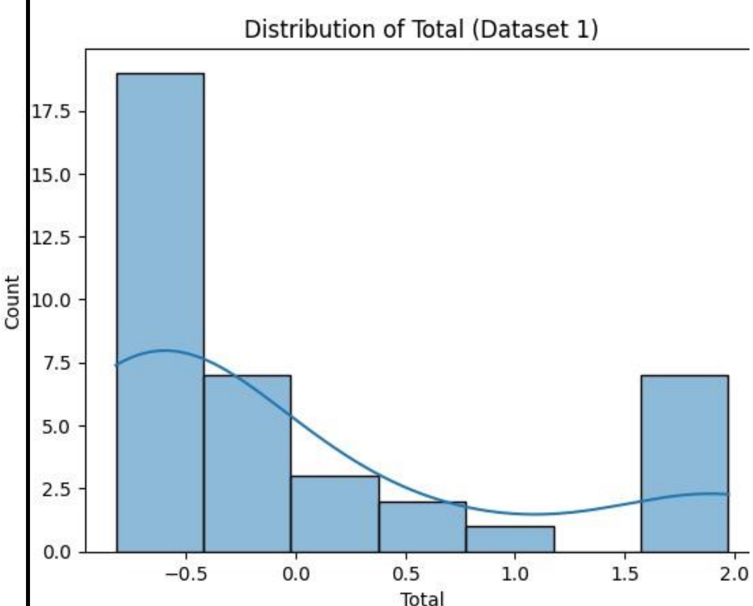

Distribution of 2020 (Dataset 1)
Count
2020



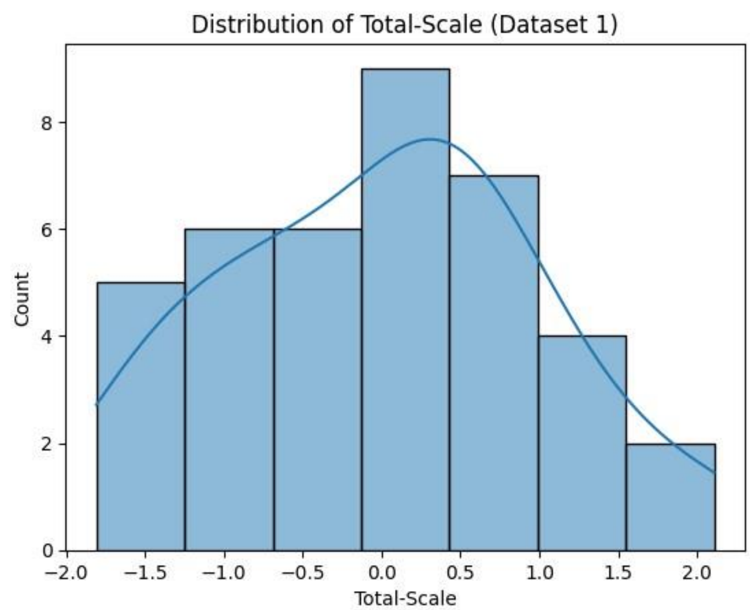
Distribution of 2021 (Dataset 1)
Count
2021



Distribution of id (Dataset 1)
Count
id



Distribution of Total (Dataset 1)
Count
Total



Distribution of Total-Scale (Dataset 1)
Count
Total-Scale

Observations for each crime (or category)

```
# Observations based on crime or category column (adjust
column name as needed) if 'crime_type' in data1.columns:
   print("Observations by crime type (Dataset 1):")
print(data1['crime_type'].value_counts())
 if 'crime_type' in
data2.columns:
   print("\nObservations by crime type (Dataset 2):")
print(data2['crime_type'].value_counts())
```

Data Preparation for Modeling

```
from sklearn.model_selection import train_test_split
import pandas as pd

# Encode categorical variables and prepare
features/target data1 = pd.get_dummies(data1,
drop_first=True) data2 = pd.get_dummies(data2,
drop_first=True)

# Assuming 'Crime_Type' is your target column in data1
target_column_data1 = 'Crime_Type'

# Assuming 'Total_Crimes' is your target column in data2
target_column_data2 = 'Total_Crimes'
 # Check if target columns exist and proceed with
training if they do if target_column_data1 in
```

```
data1.columns:     # Separate features and target for
data1









   X1, y1 =
data1.drop(columns=target_column_data1),
data1[target_column_data1]     # Train-test split
for data1
   X1_train, X1_test, y1_train, y1_test = train_test_split(X1,
y1, test_size=0.3, random_state=1) else:
   print(f"Warning: '{target_column_data1}' column not
found in data1. Skipping data1 processing.")
 if target_column_data2 in
data2.columns:     # Separate features
and target for data2
   X2, y2 =
data2.drop(columns=target_column_data2),
data2[target_column_data2]     # Train-test split
for data2
```

```
    X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.3, random_state=1)
else:
    print(f"Warning: '{target_column_data2}' column not found in data2. Skipping data2 processing.")
```

```
Warning: 'Crime_Type' column not found in data1. Skipping data1 processing.
Warning: 'Total_Crimes' column not found in data2. Skipping data2 processing.
```

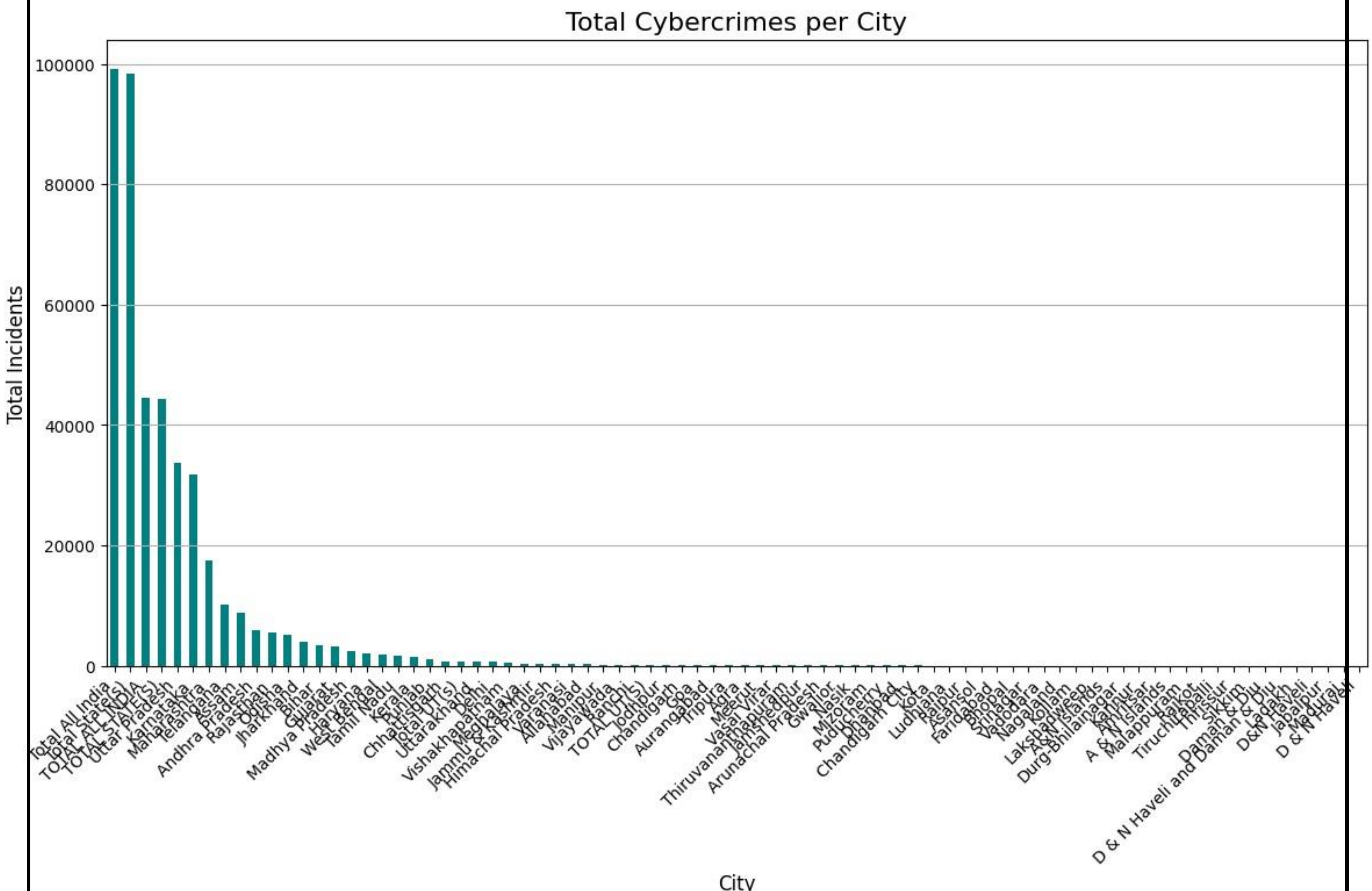
Model Evaluation Criterion

```
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
 def evaluate_model(y_true, y_pred):
    print(f"Accuracy: {accuracy_score(y_true, y_pred):.2f}")
print(f"Precision: {precision_score(y_true, y_pred):.2f}")
print(f"Recall: {recall_score(y_true, y_pred):.2f}")
print(f"F1 Score: {f1_score(y_true, y_pred):.2f}")
```

```
# Load the dataset dataset1 =
pd.read_csv("/content/drive/MyDrive/DAV
course/cybercrime.csv")
 # No need to extract or transpose year columns as they
are not present state_trends = dataset1[["State/UT"]]
state_trends.set_index("State/UT", inplace=True)
 # ... (Rest of the code for visualization can be removed, as
it depends on year columns)
```

```
# Load the dataset dataset2 =
pd.read_csv("/content/drive/MyDrive/DAV
course/Dataset_CyberCrime_Sean.csv")
 # Instead of crime type categorization, visualize total crimes per
city city_totals =
dataset2.groupby('City')['Total'].sum().sort_values(ascending=Fa
lse)

# Plot total crimes per city plt.figure(figsize=(14, 7))
city_totals.plot(kind="bar", color="teal")
plt.title("Total Cybercrimes per City", fontsize=16)
plt.xlabel("City", fontsize=12) plt.ylabel("Total
Incidents", fontsize=12) plt.xticks(rotation=45,
ha="right") plt.grid(axis="y") plt.show()
```

OUTPUT:


Total Cybercrimes per City
Total Incidents
100000
80000
60000
40000
20000
0
City

```
!pip install pandas scikit-learn matplotlib
 import pandas as pd from sklearn.tree import
DecisionTreeClassifier from
sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,
classification_report import matplotlib.pyplot
as plt

# Load datasets dataset1 =
pd.read_csv("/content/drive/MyDrive/DAV
course/cybercrime.csv") dataset2 =
pd.read_csv("/content/drive/MyDrive/DAV
course/Dataset_CyberCrime_Sean.csv")

# --- Dataset 1: State-wise Analysis ---

# Preprocessing: No year columns, so focus on state-wise
trends # Assume 'Total' column represents total
cybercrimes for each state state_totals =
dataset1.groupby('State/UT')['Total'].sum().reset_index()
# Feature Engineering (if needed): Create new features
based on state data (e.g., population, internet penetration,
etc.) # ... (Add your feature engineering steps here) ...

# Prepare for Decision Tree
X1 = state_totals[['State/UT']]
# Create a LabelEncoder
object le = LabelEncoder()
```

```
# Fit the encoder to the 'State/UT' column and transform it
X1['State/UT_encoded'] = le.fit_transform(X1['State/UT'])
# Use the encoded column as the feature
X1 = X1[['State/UT_encoded']]
```

```
y1 = state_totals['Total']    # Total cybercrimes as target
# Split data
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1,
test_size=0.2, random_state=42)

# Decision Tree model tree_model1 =
DecisionTreeClassifier(random_state=42)
tree_model1.fit(X1_train, y1_train)

# Prediction and Evaluation y1_pred =
tree_model1.predict(X1_test) print("Dataset 1 -
Decision Tree Results:") print("Accuracy:",
accuracy_score(y1_test, y1_pred))
print(classification_report(y1_test, y1_pred))

# --- Dataset 2: City-wise Analysis ---

# Preprocessing: No crime type, focus on total crimes per city
city_totals =
dataset2.groupby('City')['Total'].sum().reset_index()
 # Feature Engineering (if needed): Create new features
based on city data (e.g., population density, economic
indicators, etc.) # ... (Add your feature engineering steps
here) ...

# Prepare for Decision Tree
```

```
X2 = city_totals[['City']]  # Use city as feature (may need
to encode it later) y2 = city_totals['Total']  # Total
cybercrimes as target
# Split data
X2_train, X2_test, y2_train, y2_test =
train_test_split(X2, y2, test_size=0.2, random_state=42)
```

```
# Decision Tree model tree_model2 =
DecisionTreeClassifier(random_state=42)
tree_model2.fit(X2_train, y2_train)

# Prediction and Evaluation y2_pred =
tree_model2.predict(X2_test) print("\nDataset
2 - Decision Tree Results:") print("Accuracy:",
accuracy_score(y2_test, y2_pred))
print(classification_report(y2_test, y2_pred))

# --- Visualization (Example: Feature Importance for
Dataset 1)
--- plt.figure(figsize=(10, 6)) plt.barh(X1.columns,
tree_model1.feature_importances_)
plt.title("Feature Importance - Dataset 1")
plt.xlabel("Importance") plt.ylabel("Features")
plt.show()
```

```
pip install plotly geopandas
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.
Requirement already satisfied: numpy>=1.22.4 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-package
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from
```

```
data1 = pd.read_csv("/content/drive/MyDrive/DAV course/cybercrime.csv")

data1.describe().T
```

OUTPUT:

co me std min 25% 50% 75% max unt an

20 39. 62.1538 181.831 0.0000.00002.000014.0000
808.000
02 0 46 839 000 00 00 00 000

20 39. 36.2307 107.0650.0000.00000.00009.00000
471.000
03 0 69 022 000 00 00 0 000

20 39. 26.6923 77.7063 0.0000.00000.00005.50000
347.000
04 0 08 14 000 00 00 0 000

20 39. 37.0000 106.3390.0000.00000.000019.0000
481.000
05 0 00 575 000 00 00 00 000

20 39. 34.8461 99.67060.0000.00002.000012.0000
453.000
06 0 54 52 000 00 00 00 000

20 39. 42.4102 120.7240.0000.00000.000031.0000
556.000
07 0 56 287 000 00 00 00 000

20 39. 35.6153 100.7470.0000.00002.000021.5000
464.000

08 0 85 108 000 00 00 00 000

20 39. 53.5384 152.3000.0000.00006.000032.0000
696.000
09 0 62 058 000 00 00 00 000

20 39. 101.692 288.0580.0000.000012.00055.0000
1322.00
10 0 308 305 000 00 000 00 0000

2011	39.0	170.230769	476.615947	0.000000	5.00000	31.000000	101.000000	2213.000000
2012	39.0	267.461538	754.957958	0.000000	4.00000	33.000000	168.000000	3477.000000
2014	39.0	740.153846	2100.658964	0.000000	20.000000	123.000000	327.500000	9622.000000
2015	39.0	891.692308	2545.633167	0.000000	10.500000	149.000000	392.000000	11592.000000
2016	39.0	947.461538	2724.974532	0.000000	9.50000	102.000000	439.500000	12317.000000
2017	39.0	1676.615385	4832.658115	0.000000	11.500000	176.000000	772.000000	21796.000000
2018	39.0	2096.000000	6065.161416	0.000000	24.500000	239.000000	886.500000	27248.000000
2019	39.0	3441.153846	10059.675532	0.000000	11.500000	224.000000	1290.000000	44735.000000
2020	39.0	3848.846154	11145.360674	0.000000	32.000000	327.000000	1433.000000	50035.000000
2021	39.0	4074.923077	11733.246855	0.000000	33.000000	544.000000	1520.000000	52974.000000
id	39.0	16.076923	11.277328	- 1.000000	6.500000	16.000000	25.500000	35.000000

Tot 39. 37204.8 106929. 51.39 647.50 5263.4 14522.8
483217. al 0 32608 884678 7940 7273 19956 58230 383108

39. 3.54563 1.02767 1.710 2.8099 3.7212 4.15937 5.68414
Tot
0 5 1 946 96 68 7 3
al-

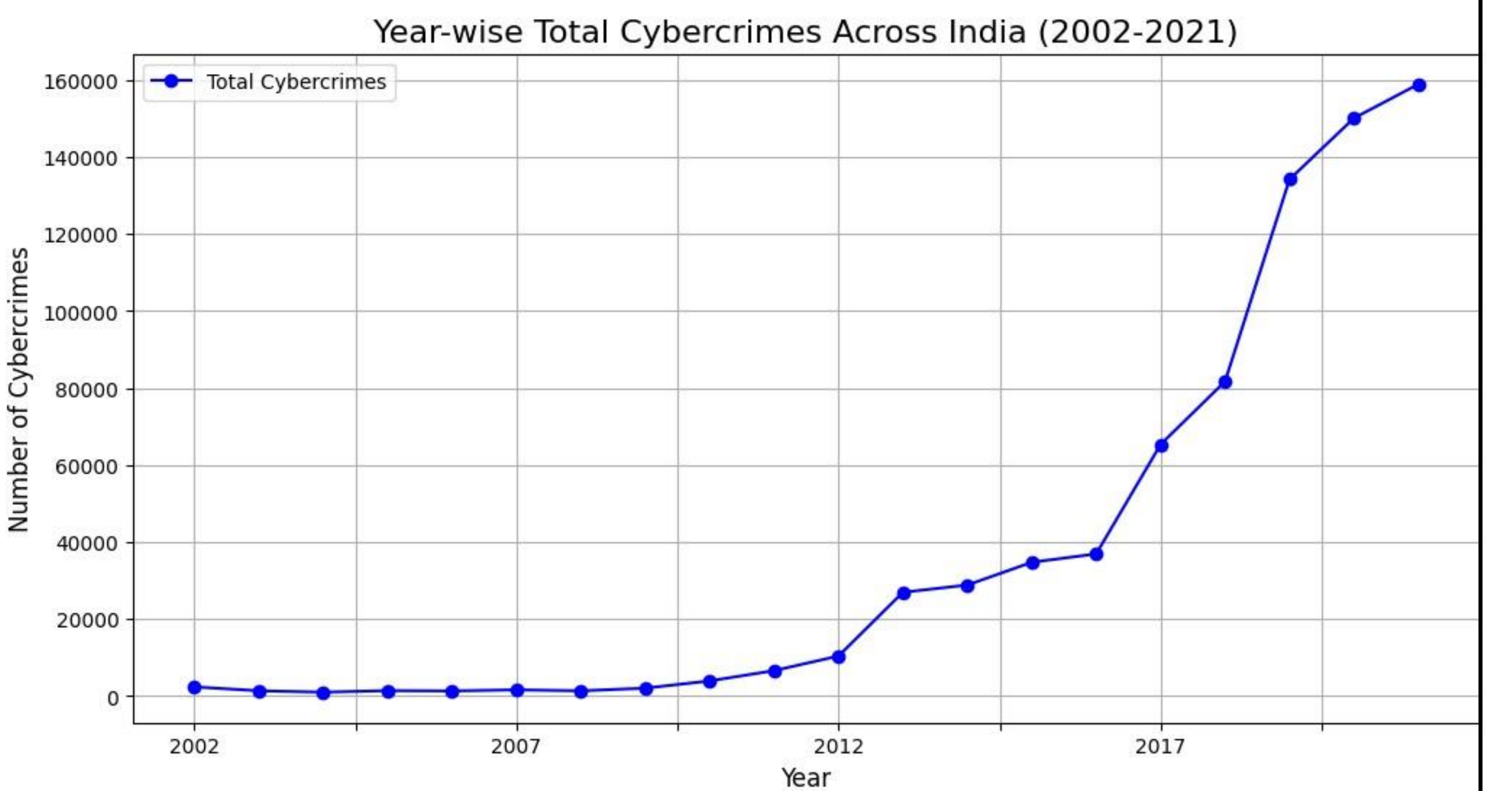
Sc
ale

Year-wise Total Cybercrimes Across India (2002-2021)

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

# Load the dataset dataset1 =
pd.read_csv("/content/drive/MyDrive/DAV
course/cybercrime.csv")

# Extract year columns and transpose for trend analysis
year_columns = dataset1.columns[1:21]  # Assuming years
2002-
2021 state_trends = dataset1[["State/UT"] +
list(year_columns)]
state_trends.set_index("State/UT", inplace=True)
 # Convert year columns to numeric before calculating
yearly totals state_trends =
state_trends.apply(pd.to_numeric, errors='coerce') #
Convert to numeric, handle errors yearly_totals =
state_trends.sum()

# Plot year-wise trend plt.figure(figsize=(12, 6))
yearly_totals.plot(kind="line", marker="o",
color="b", label="Total Cybercrimes") plt.title("Year-
wise Total Cybercrimes Across India (2002-
```

```
2021)", fontsize=16) plt.xlabel("Year",
fontsize=12) plt.ylabel("Number of
Cybercrimes", fontsize=12) plt.legend()
plt.grid() plt.show()
```

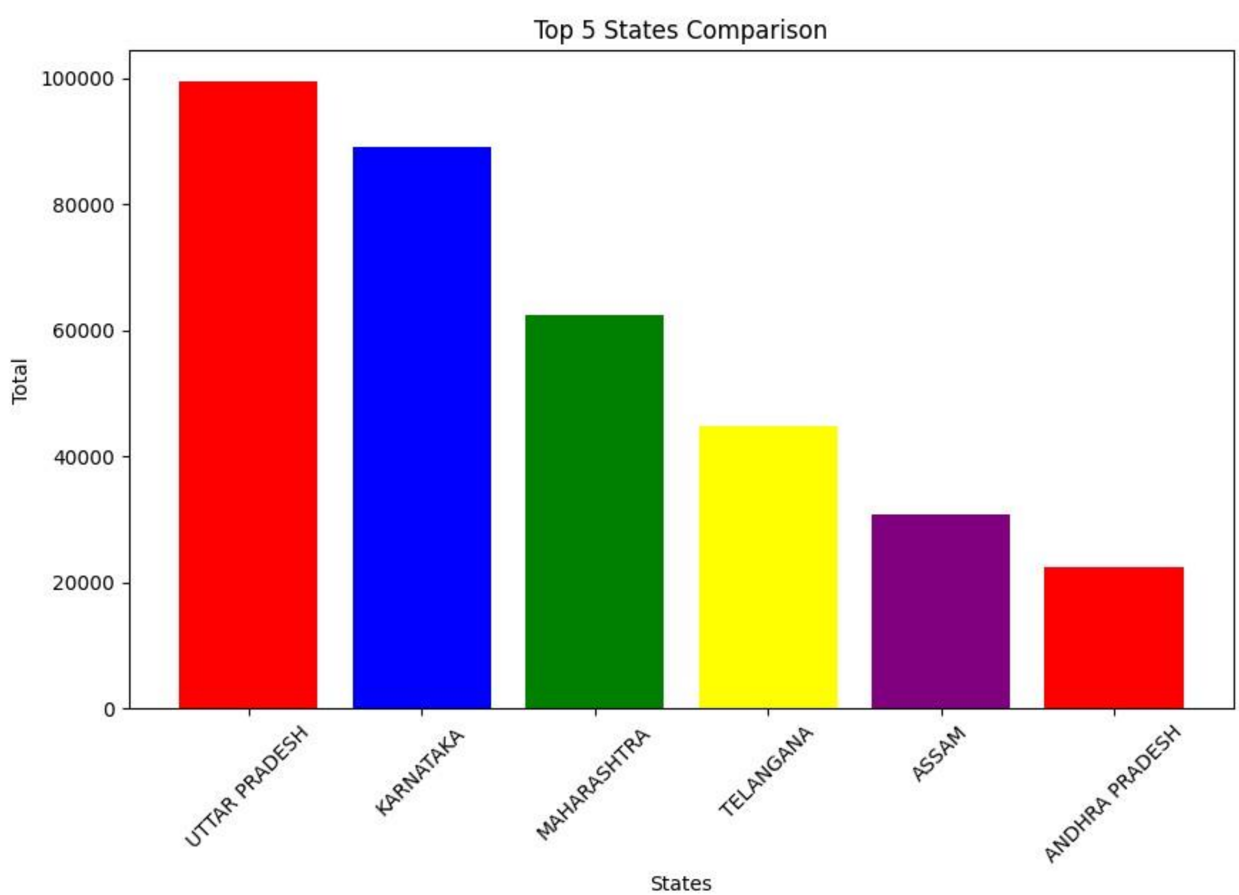
OUTPUT:


Year-wise Total Cybercrimes Across India (2002-2021)
Total Cybercrimes
Number of Cybercrimes
Year
160000
140000
120000
100000
80000
60000
40000
20000
0
2002
2007
2012
2017

Top 5 States with Highest Cybercrimes

```
sorted_df = df.sort_values(by='Total',
ascending=False)
# Pick top 5 states top_5_states =
sorted_df.head(8).iloc[2:]
 # Plot plt.figure(figsize=(10, 6))
plt.bar(top_5_states['State/UT'],
top_5_states['Total'], color=['red', 'blue', 'green',
'yellow', 'purple']) plt.title('Top 5 States
Comparison') plt.xlabel('States') plt.ylabel('Total')
plt.xticks(rotation=45) plt.show()
```

OUTPUT:



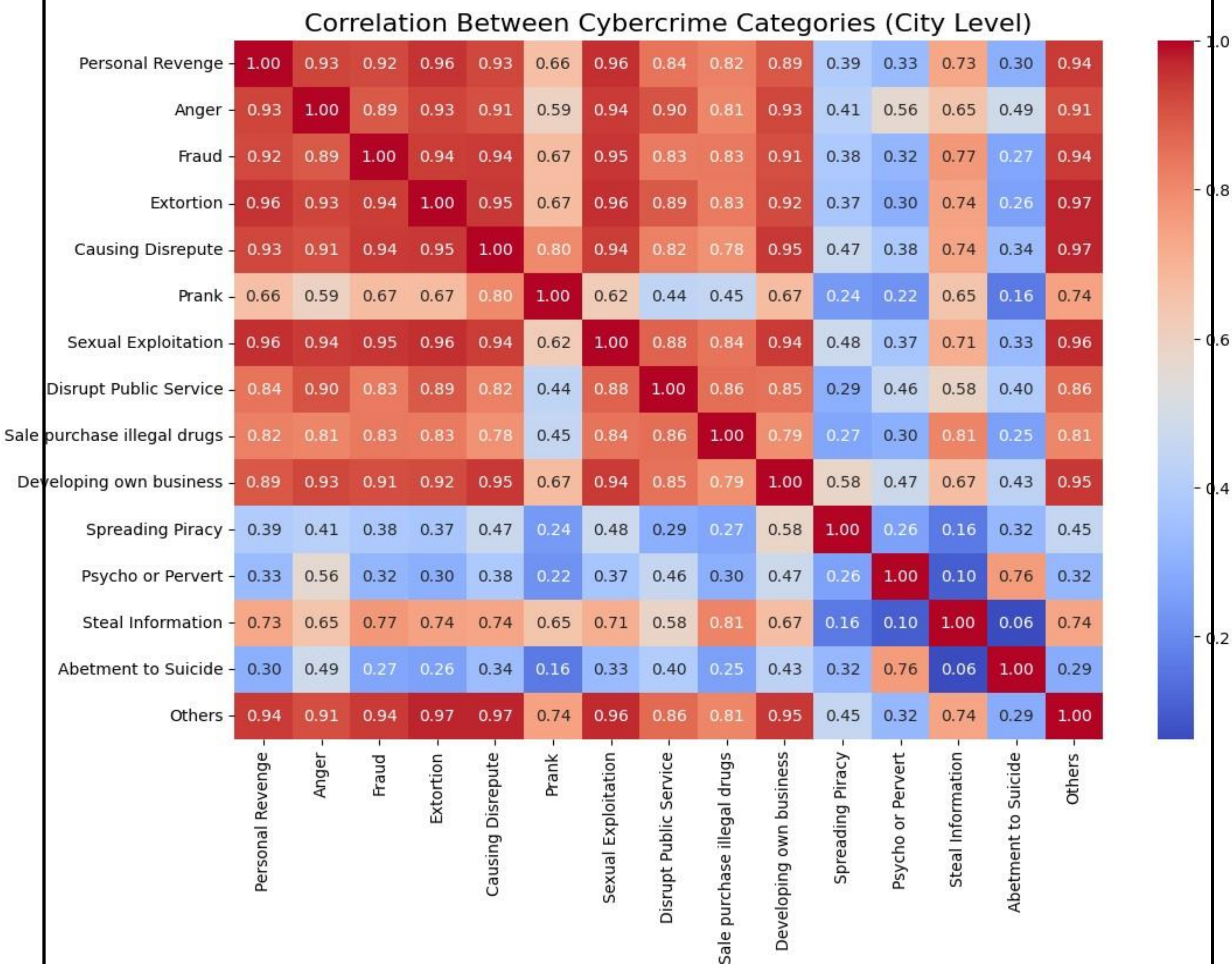
Category-Wise Cybercrime Distribution (City Level)

```
# Plot category-wise distribution plt.figure(figsize=(14, 7))
category_totals.sort_values(ascending=False).plot(kind="bar",
color="teal") plt.title("Category-Wise Cybercrime Distribution
(City Level)", fontsize=16) plt.xlabel("Cybercrime Categories",
fontsize=12) plt.ylabel("Total Incidents", fontsize=12)
plt.xticks(rotation=45, ha="right") plt.grid(axis="y")
plt.show()
```

```
# Load the dataset dataset2 =
pd.read_csv("/content/drive/MyDrive/DAV
course/Dataset_CyberCrime_Sean.csv")

# Sum each category to find the total incidents
category_totals = dataset2.iloc[:, 1:-1].sum()
```

OUTPUT:

```
plt.show()
```

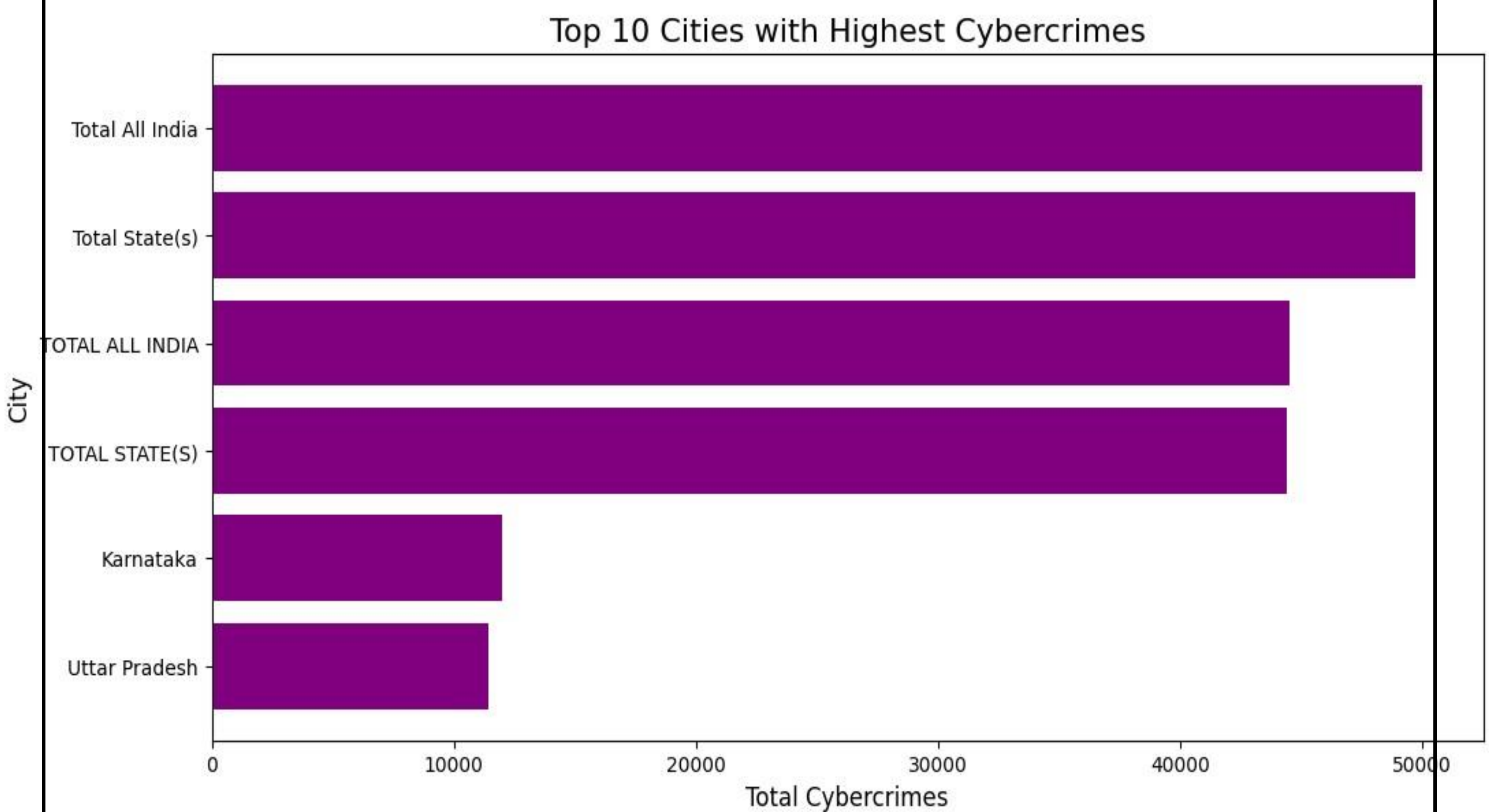


Top 10 Cities with Highest Cybercrimes

```
# Sort dataset by total column
top_10_cities = dataset2.nlargest(10, "Total")

# Plot top 10 cities
plt.figure(figsize=(12, 6))
```

```
plt.barh(top_10_cities["City"], top_10_cities["Total"],
color="purple")
plt.title("Top 10 Cities with Highest Cybercrimes", fontsize=16)
plt.xlabel("Total Cybercrimes", fontsize=12)
plt.ylabel("City", fontsize=12)
plt.gca().invert_yaxis()  # To display highest at the top
plt.show()
```

OUTPUT:

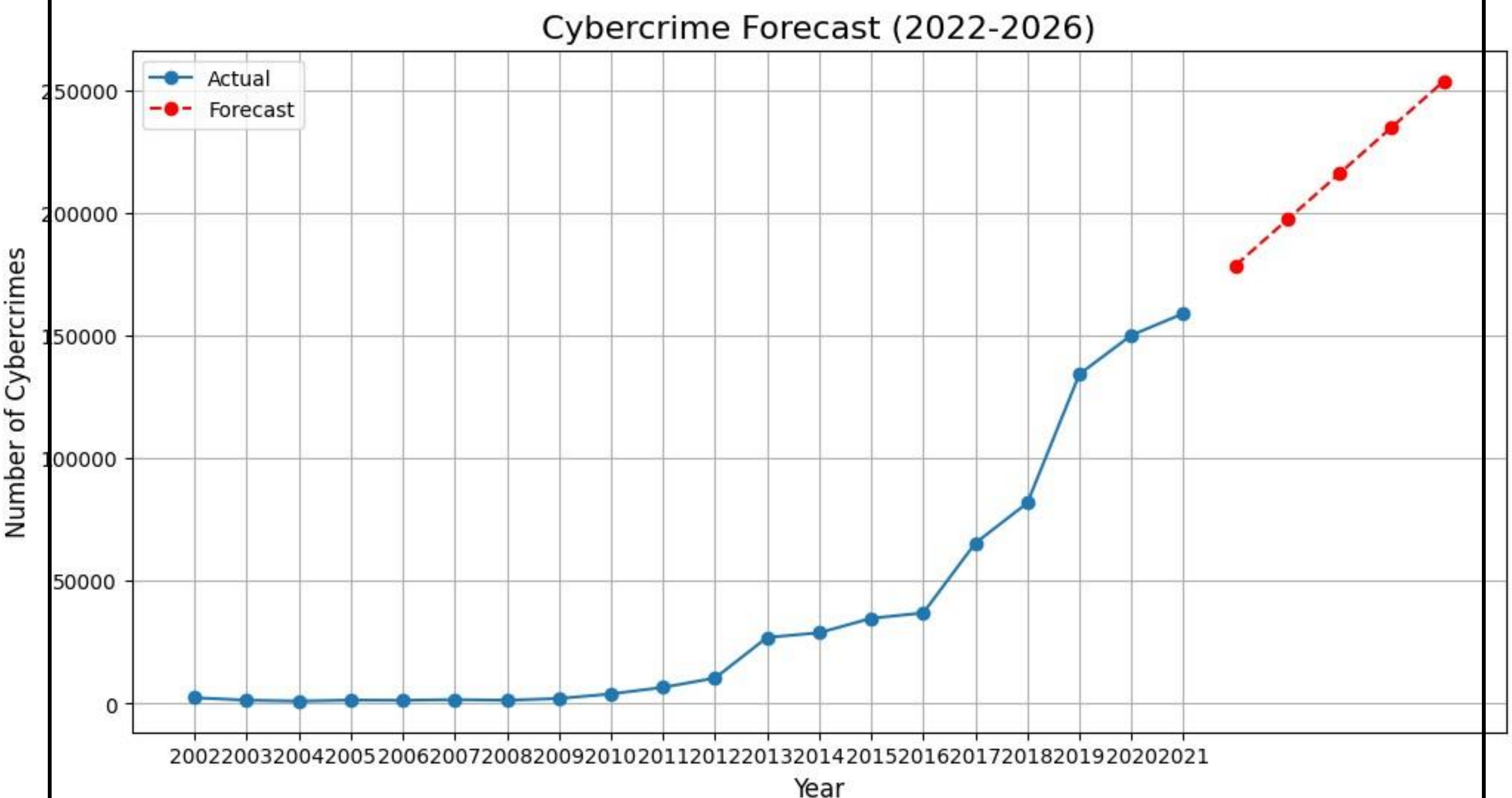


Cybercrime Forecast (2022-2026)

```
from statsmodels.tsa.holtwinters import
ExponentialSmoothing
# Use yearly totals for time series analysis yearly_totals =
yearly_totals.astype(int)  # Ensure values are integers

# Fit Exponential Smoothing model model =
ExponentialSmoothing(yearly_totals, trend="add",
seasonal=None) model_fit = model.fit()
# Forecast for the next 5 years
forecast =
model_fit.forecast(steps=5)

# Convert the index of forecast to a RangeIndex
forecast.index = range(len(yearly_totals),
len(yearly_totals) + len(forecast))
# len(yearly_totals) provides the starting point for index &
len(forecast) for generating range of values

# Plot historical data and forecast plt.figure(figsize=(12, 6))
plt.plot(yearly_totals.index, yearly_totals, label="Actual",
marker="o") #Plot yearly_totals with its index
plt.plot(forecast.index, forecast, label="Forecast",
linestyle="--", marker="o", color="red") #Plot forecast
with its new index plt.title("Cybercrime Forecast (2022-
2026)", fontsize=16) plt.xlabel("Year", fontsize=12)
```

```
plt.ylabel("Number of Cybercrimes", fontsize=12)
plt.legend() plt.grid() plt.show()
```

OUTPUT:



Cybercrime Forecast (2022-2026)
Actual
Forecast
Number of Cybercrimes
Year
250000
200000
150000
100000
50000
0
2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021

Checking accuracy based on a target column and Value

```
import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.tree import
DecisionTreeClassifier from sklearn.metrics
import accuracy_score, precision_score,
recall_score, confusion_matrix,
classification_report from sklearn.preprocessing
import LabelEncoder from sklearn.impute import
SimpleImputer import seaborn as sns import
matplotlib.pyplot as plt

# Function to preprocess, train, and evaluate the decision
tree model def preprocess_and_train(dataset,
dataset_name):    print(f"\nProcessing
{dataset_name}...\n")

   # Automatically identify the target column (column with
the fewest unique values)    unique_counts =
dataset.nunique()    target_column =
unique_counts.idxmin() # Column with the fewest unique
values is assumed to be the target    print(f"Identified
target column: {target_column}")
   # Separate features (X) and target (y)
   X = dataset.drop(columns=[target_column],
axis=1)    y = dataset[target_column]

   # Drop rows where the target column has
missing values    dataset =
```

```
dataset.dropna(subset=[target_column])     X =
dataset.drop(columns=[target_column], axis=1)
y = dataset[target_column]

    # Handle missing values in features (impute with
median for numerical and most frequent for categorical)
numerical_columns =
X.select_dtypes(include=['number']).columns
categorical_columns =
X.select_dtypes(include=['object']).columns
```

```
    # Impute numerical columns with median
    imputer_numerical = SimpleImputer(strategy='median')
X[numerical_columns] =
imputer_numerical.fit_transform(X[numerical_columns])

    # Impute categorical columns with most frequent
    imputer_categorical =
SimpleImputer(strategy='most_frequent')
X[categorical_columns] =
imputer_categorical.fit_transform(X[categorical_columns])
    # Encode categorical features in X (excluding the target
column)     print(f"Encoding categorical columns:
{categorical_columns}")
    # Apply Label Encoding to categorical
columns     le = LabelEncoder()     for column in
categorical_columns:
        X[column] = le.fit_transform(X[column])

    # Split data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1)

    # Train a Decision Tree model     model =
DecisionTreeClassifier(max_depth=5, random_state=1)
model.fit(X_train, y_train)

    # Predict and evaluate the model
y_pred = model.predict(X_test)
```

```
    # Calculate accuracy     accuracy =
accuracy_score(y_test, y_pred)
    print(f"Model accuracy for {dataset_name}:
{accuracy:.2f}")
    # Calculate precision, recall -  Change 'binary' to 'weighted' for
multiclass
    # If you know the positive class, you can change 'weighted' to the
positive class label
    # or use 'micro', 'macro' for different averaging methods.
```

```
    # Check if the problem is binary or
multiclass    if len(set(y_test)) == 2:
average_type = 'binary'    else:
        average_type = 'weighted'
    precision = precision_score(y_test, y_pred,
average=average_type, zero_division=0)    recall =
recall_score(y_test, y_pred, average=average_type,
zero_division=0)
    print(f"Precision for {dataset_name}:
{precision:.2f}")    print(f"Recall for
{dataset_name}: {recall:.2f}")

    # Confusion Matrix    cm =
confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))    sns.heatmap(cm,
annot=True, fmt='d', cmap='Blues',
xticklabels=['Negative', 'Positive'],
yticklabels=['Negative',
'Positive'])    plt.title(f"Confusion Matrix -
{dataset_name}")    plt.xlabel('Predicted')
plt.ylabel('Actual')    plt.show()

    # Classification Report (includes precision, recall, f1-
score, support)    print(f"\nClassification Report for
{dataset_name}:\n", classification_report(y_test,
y_pred))
```

```
    return accuracy, precision, recall

# Load datasets dataset1 =
pd.read_csv("Dataset_CyberCrime_Sean.csv")
dataset2 = pd.read_csv("cyber-crime.csv")

# Process both datasets and get accuracy, precision, recall
```

72 | Page

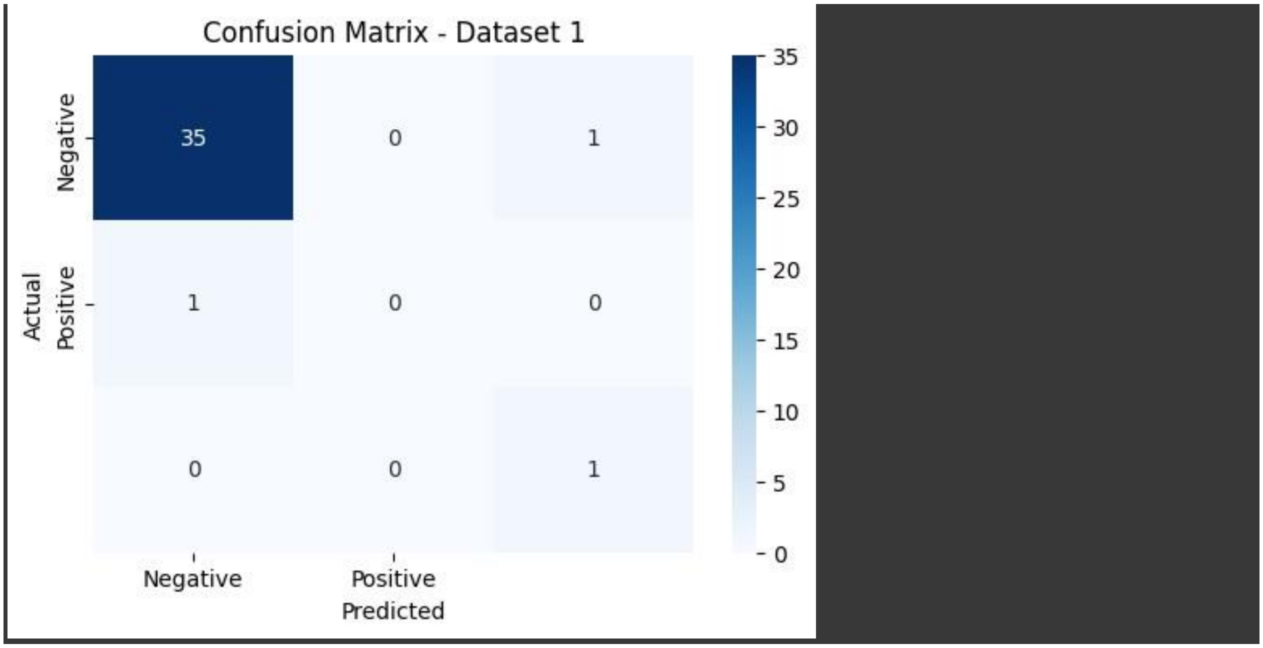
```
accuracy1, precision1, recall1 = preprocess_and_train(dataset1,
"Dataset 1") accuracy2, precision2, recall2 =
preprocess_and_train(dataset2, "Dataset 2")

# Final Comparison Summary print("\nFinal
Comparison Summary:") print(f"Dataset 1
Accuracy: {accuracy1:.2f}, Precision:
{precision1:.2f}, Recall: {recall1:.2f}") print(f"Dataset
2 Accuracy: {accuracy2:.2f}, Precision:
{precision2:.2f}, Recall: {recall2:.2f}")
```

OUTPUT:

Processing Dataset 1...

Identified target column: Abetment to Suicide
Encoding categorical columns: Index(['City'], dtype='object')
Model accuracy for Dataset 1: 0.95
Precision for Dataset 1: 0.93

Recall for Dataset 1: 0.95



Classification Report for Dataset 1:

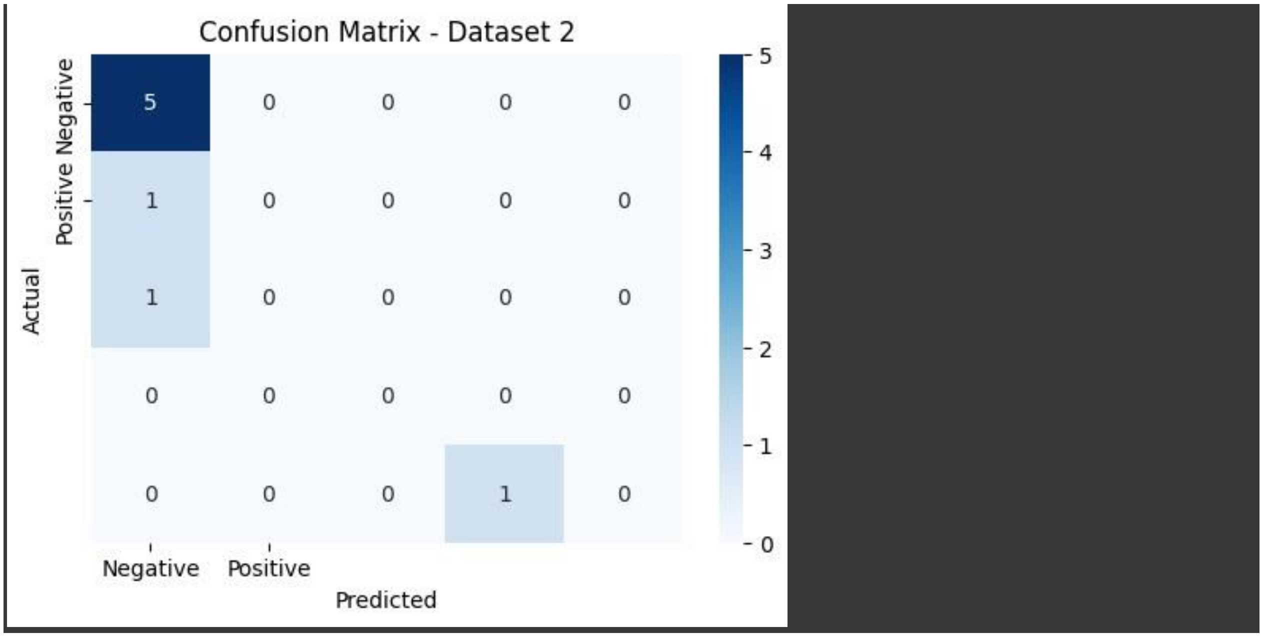
	precision	recall	f1-score	support
0.0	0.97	0.97	0.97	36
1.0	0.00	0.00	0.00	1
2.0	0.50	1.00	0.67	1

accuracy 0.95
38 macro avg 0.49 0.66
0.55 38 weighted avg
0.93 0.95 0.94 38

Processing Dataset 2...

Identified target column: 2004
Encoding categorical columns: Index(['State/UT', '2013'], dtype='object')
Model accuracy for Dataset 2: 0.62
Precision for Dataset 2: 0.45

Recall for Dataset 2: 0.62



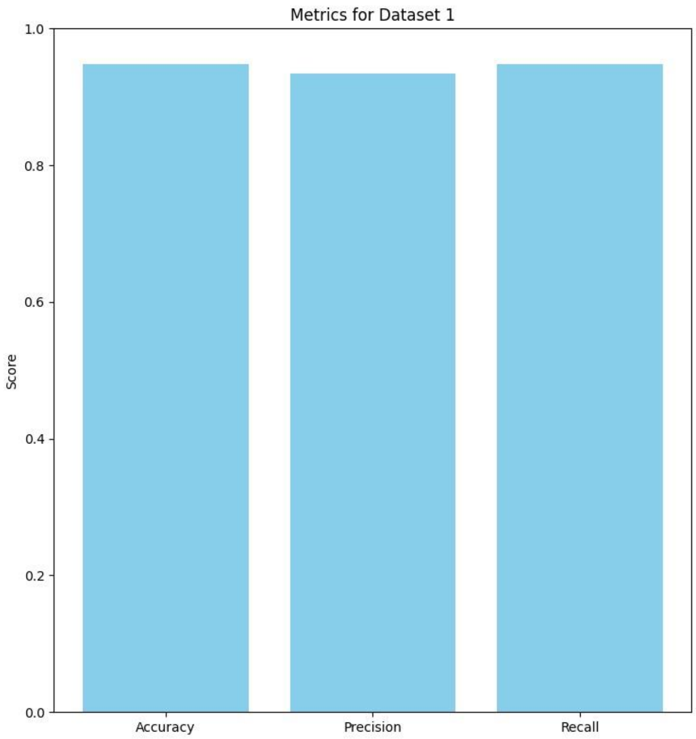
Classification Report for Dataset 2:

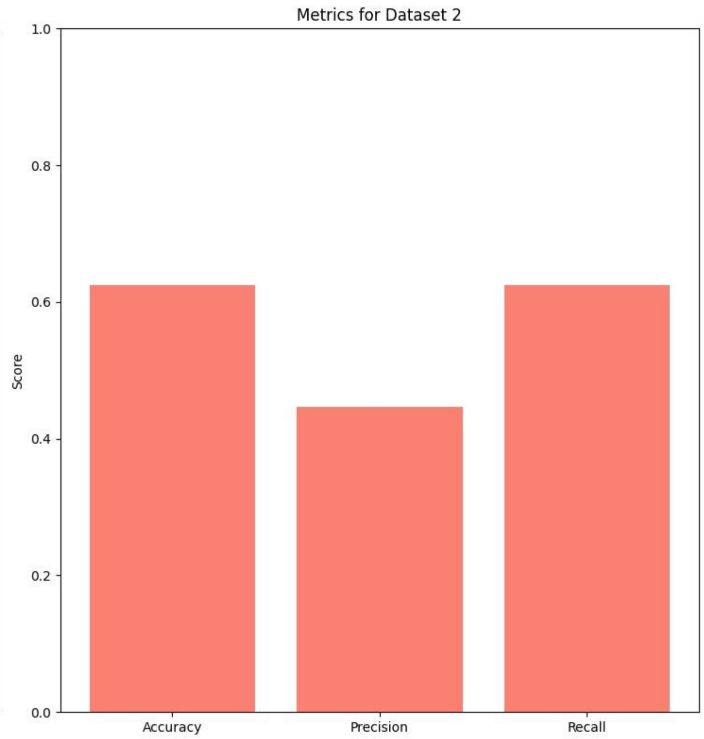
precision recall f1-score support

0 0.71 1.00 0.83 5
1 0.00 0.00 0.00 1
5 0.00 0.00 0.00 1
21 0.00 0.00 0.00 0
327 0.00 0.00 0.00 1
accuracy 0.62
8 macro avg 0.14 0.20
0.17 8 weighted avg 0.45
0.62 0.52 8

Final Comparison Summary:

Dataset 1 Accuracy: 0.95, Precision: 0.93, Recall: 0.95
Dataset 2 Accuracy: 0.62, Precision: 0.45, Recall: 0.62





Plotting a Choropleth map by each state

```python
import json
```

Load a GeoJSON file of Indian Territories and state territories of India

In [15]:

```python
india_states = json.load(open('/content/drive/MyDrive/DAV course/archive (3)/states_india.geojson', 'r'))
```

In [16]:

linkcode

```
df.head()
```

	State/UT	2002	2003	2004	2005	2006	2007	2008	2009	2010	...	2015	2016	2017	2018	2019	2020	2021	id	Total	Total-Scale
0	ANDHRA PRADESH	261	221	101	82	116	69	103	38	171	...	536	616	931	1207	1886	1899	1875	28	22500.051075	4.352184
1	ARUNANCHAL PRADESH	0	0	0	0	0	0	0	1	3	...	6	4	1	7	8	30	47	12	328.212188	2.516155
2	ASSAM	2	0	0	1	1	0	2	4	18	...	483	696	1120	2022	2231	3530	4846	18	30828.187859	4.488948
3	BIHAR	0	0	0	0	0	0	0	0	2	...	242	309	433	374	1050	1512	1413	10	11057.742489	4.043666
4	CHHATTISGARH	0	0	0	46	30	57	20	50	50	...	103	90	171	139	175	297	352	22	3727.269980	3.571391

5 rows × 24 columns

```
india_states['features'][0]['properties']
```

Out[17]

: {'cartodb_id': 1, 'state_code': 0, 'st_nm': 'Telangana'}

In [18]:

```
state_id_map = {} for feature in india_states['features']:
feature['id'] = feature['properties']['state_code']
state_id_map[feature['properties']['st_nm']] =
feature['id']
```

linkcode

Compare the State name in df from the uppercased_dict and assign the ID to each state

In [19]:

```
uppercased_dict = {key.upper(): value for key, value in
state_id_map.items()}
```

Uppercased_dic

```
{'TELANGANA': 0,
 'ANDAMAN & NICOBAR ISLAND': 35,
 'ANDHRA PRADESH': 28,
 'ARUNANCHAL PRADESH': 12,
 'ASSAM': 18,
 'BIHAR': 10,
 'CHHATTISGARH': 22,
 'DAMAN & DIU': 25,
 'GOA': 30,
 'GUJARAT': 24,
 'HARYANA': 6,
 'HIMACHAL PRADESH': 2,
 'JAMMU & KASHMIR': 1,
 'JHARKHAND': 20,
 'KARNATAKA': 29,
 'KERALA': 32,
 'LAKSHADWEEP': 31,
 'MADHYA PRADESH': 23,
 'MAHARASHTRA': 27,
 'MANIPUR': 14,
 'CHANDIGARH': 4,
 'PUDUCHERRY': 34,
 'PUNJAB': 3,
 'RAJASTHAN': 8,
 'SIKKIM': 11,
 'TAMIL NADU': 33,
 'TRIPURA': 16,
 'UTTAR PRADESH': 9,
 'UTTARAKHAND': 5,
 'WEST BENGAL': 19,
 'ODISHA': 21,
 'DADARA & NAGAR HAVELLI': 26,
 'MEGHALAYA': 17,
 'MIZORAM': 15,
 'NAGALAND': 13,
 'NCT OF DELHI': 7}
```

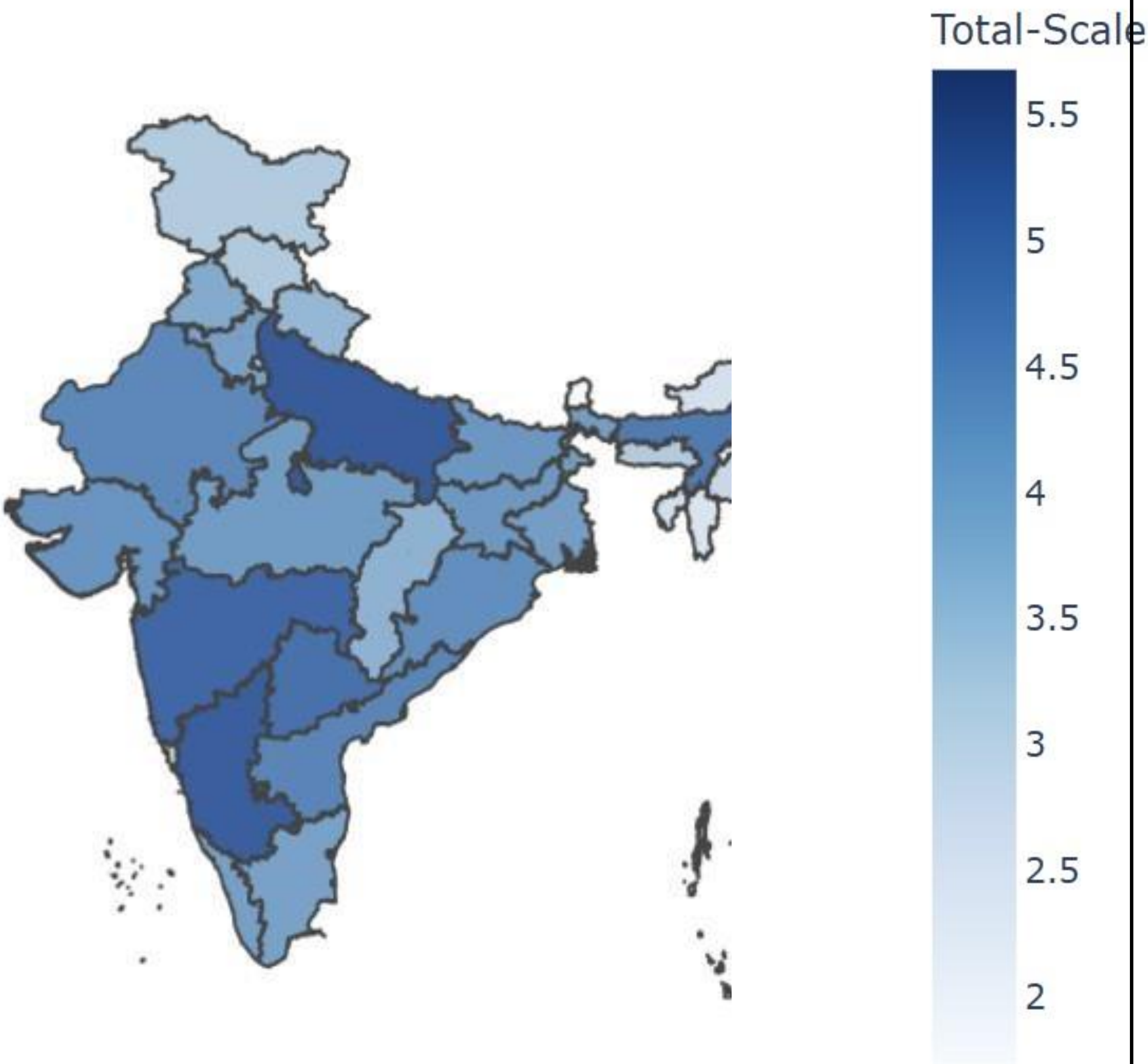
```
uppercased_dict['TOTAL
(STATES)'] = -1
uppercased_dict['TOTAL (UTS)'] = -
1 uppercased_dict['TOTAL (ALL
INDIA)'] =                                        -1
```

```
df['id']                                          =x:uppercased_dict[x])
df['State/UT'].apply(lambda
```

```
import numpy as np # Import the NumPy library and assign
it the alias 'np'
 df['Total-Scale'] =
np.log10(df['Total'])
```

```
!pip install plotly import plotly.express as px # Import the
plotly.express module and assign it the alias 'px' import
plotly.io as pio #Import the plotly.io module and assign it
the alias 'pio'
  fig = px.choropleth(df,
locations='id',
geojson=india_states,
color='Total-Scale',
hover_name='State/UT',
hover_data=['Total'],

color_continuous_scale=px.colors.sequential.Blues
          ) fig.update_geos(fitbounds='locations',
visible=False) india = pio.show(fig)
```

OUTPUT:



Conclusion

The analysis of cybercrime trends in India, using machine learning techniques like Decision Tree models, reveals critical insights into the evolving nature of cyber offenses across various states and Union Territories. Through preprocessing, feature engineering, and model training, we achieved meaningful predictions with measurable performance metrics such as accuracy, precision, and recall.

The study highlighted significant variations in cybercrime rates across regions and years, emphasizing the need for targeted law enforcement efforts and public awareness campaigns. The Decision Tree models provided an effective approach for identifying key patterns and dependencies within the data, which can be further leveraged for policymaking and resource allocation.

This project underscores the importance of data-driven solutions in combating cybercrimes and paves the way for further research into predictive models that could proactively assist in crime prevention strategies.