**COde:**

import pandas as pd

# Load the dataset

data\_path = 'FINAL DATASET.csv'

df = pd.read\_csv(data\_path)

# Display the first few rows of the dataset

print("Initial Dataset Preview:")

print(df.head())

# Step 1: Handle Missing Values and Replace 0s with Mean

print("\nHandling Missing Values and Replacing 0s with Mean...")

# Check for missing values and 0s before imputation

print("Missing values and 0s before imputation:")

print((df.isnull() | (df == 0)).sum()) # Check for both NaN and 0

# Replace null values and 0s in numerical columns with the mean

numerical\_df = df.select\_dtypes(include=['number'])

for col in numerical\_df.columns:

mean\_val = numerical\_df[col][numerical\_df[col] != 0].mean() # Calculate mean excluding 0s

df[col] = df[col].replace(0, mean\_val) # Replace 0s with mean

df[col] = df[col].fillna(mean\_val) # Fill NaNs with mean

# Check for missing values and 0s after imputation

print("Missing values and 0s after imputation:")

print((df.isnull() | (df == 0)).sum())

# Step 2: Save the Cleaned Data

output\_path = 'CLEANED\_FINAL\_DATASET.csv'

df.to\_csv(output\_path, index=False)

print(f"\nCleaned dataset saved to: {output\_path}")

# Final Preview of Cleaned Data

print("\nCleaned Dataset Preview:")

print(df.head())

!pip install matplotlib

import pandas as pd

import matplotlib.pyplot as plt

# Load the dataset

data\_path = 'FINAL DATASET.csv'

df = pd.read\_csv(data\_path)

# Display the first few rows of the dataset

print("Initial Dataset Preview:")

print(df.head())

# ... (rest of your data cleaning code from the first cell) ...

# Crime Analysis

try:

crime\_by\_state = df.groupby("States/UTs").sum(numeric\_only=True) # Corrected column name

except KeyError:

print("Error: 'States/UTs' column not found in the DataFrame. Please check the column name.")

else: # Execute if no KeyError

crime\_types = [

"Rape other than Custodial",

"Rape\_Gang Rape",

"Kidnapping & Abduction\_Total",

"Extortion",

"Incidence of Rash Driving"

]

crime\_by\_state\_filtered = crime\_by\_state[crime\_types]

fig, axes = plt.subplots(len(crime\_types), 1, figsize=(12, 18), sharex=True)

for i, crime in enumerate(crime\_types):

crime\_by\_state\_filtered[crime].sort\_values(ascending=False).plot(

kind="bar", ax=axes[i], color="skyblue", edgecolor="black"

)

axes[i].set\_title(f'{crime} by States/UT', fontsize=14)

axes[i].set\_ylabel('Incidents', fontsize=12)

axes[i].tick\_params(axis='x', rotation=90)

plt.tight\_layout()

plt.show()

cr='Rape other than Custodial'

crime\_dist=df.groupby('States/UTs')[cr].sum()

plt.figure(figsize=(20,10))

crime\_dist.plot(kind='pie',autopct='%1.1f%%',startangle=140, cmap='tab20',textprops={'fontsize' : 10})

plt.title(f'Distribution of {cr} Across States/UTs, fontsize=14')

plt.ylabel('')

plt.tight\_layout()

plt.show()

!pip install xgboost scikit-learn imbalanced-learn

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.cluster import KMeans

from imblearn.over\_sampling import SMOTE

import xgboost as xgb

from sklearn.metrics import accuracy\_score, classification\_report

# Load the dataset

data = pd.read\_csv("FINAL DATASET.csv")

# Encode categorical columns

label\_encoders = {}

for col in ['States/UTs', 'District']:

le = LabelEncoder()

data[col] = le.fit\_transform(data[col])

label\_encoders[col] = le

# Normalize numerical columns (excluding 'Year')

numerical\_columns = data.drop(columns=['States/UTs', 'District', 'Year']).columns

scaler = StandardScaler()

data[numerical\_columns] = scaler.fit\_transform(data[numerical\_columns])

# Apply KMeans clustering

n\_clusters = 5 # Adjust the number of clusters as needed

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

data['Cluster'] = kmeans.fit\_predict(data[numerical\_columns])

# Filter out classes with fewer than 2 samples

filtered\_data = data[data['Cluster'].map(data['Cluster'].value\_counts()) > 1]

# Redefine X and y

X = filtered\_data.drop(columns=['Cluster', 'Year'])

y = filtered\_data['Cluster']

# Apply SMOTE for oversampling the minority class

smote = SMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(X, y)

# Map class labels to [0, 1], handling NaN values

y\_resampled = y\_resampled.map({0: 0, 4: 1}).fillna(0)

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_resampled, y\_resampled, test\_size=0.2, random\_state=42

)

# Initialize and train the XGBoost model

xgb\_model = xgb.XGBClassifier(

use\_label\_encoder=False,

eval\_metric='mlogloss',

random\_state=42

)

xgb\_model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = xgb\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Assuming your data is in a CSV file named 'crime\_data.csv'

# Replace 'crime\_data.csv' with the actual file path if needed

data = pd.read\_csv('FINAL DATASET.csv')

# Step 1: Preprocessing

# Encode categorical features

le\_states = LabelEncoder()

le\_districts = LabelEncoder()

data['States/UTs'] = le\_states.fit\_transform(data['States/UTs'])

data['District'] = le\_districts.fit\_transform(data['District'])

# Step 2: Define features and target

X = data.drop(columns=['Total Cognizable IPC crimes']) # Features

y = data['Total Cognizable IPC crimes'] # Target

# Step 3: Train/Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Step 4: Train a regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Step 5: Predictions

y\_pred = model.predict(X\_test)

# Calculate performance metrics

mse = mean\_squared\_error(y\_test, y\_pred)

# Add predictions to the test dataset

test\_results = X\_test.copy()

test\_results['Actual'] = y\_test

test\_results['Predicted'] = y\_pred

test\_results.head(), mse