Github ReponLink https://colab.research.google.com/drive/1OjUr94Kd752apyGrR-6k\_O6yogX0Nsxf?usp=sharing

# Import necessary libraries

import pandas as pd

import numpy as np

# Load the dataset

data\_path = 'FINAL DATASET (1).csv' # Specify the path to your dataset

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

# Display the first few rows of the dataset

print("Initial Dataset Preview:")

print(df.head())

# Display basic dataset information

print("\nDataset Information:")

df.info()

# Display basic statistics for numerical columns

print("\nBasic Statistical Summary:")

print(df.describe(include='all'))

# Display missing values and zeros before processing

print("\nCount of Missing Values and Zeros (Before Processing):")

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

import pandas as pd

# Display column-wise non-null counts and percentages

non\_null\_counts = df.notnull().sum()

non\_null\_percentages = (non\_null\_counts / len(df)) \* 100

print("Non-null counts and percentages:")

print(pd.DataFrame({

"Non-Null Count": non\_null\_counts,

"Percentage": non\_null\_percentages

}))

print(df['States/UTs'].value\_counts()) # Count of each State/UT

print(df[df['Rape other than Custodial'] > 0].count()) # Count rows where values > 0

print(df['Rape\_Gang Rape'].unique()) # Ensure only valid numerical values exist

print(df.describe())

import pandas as pd

# Load the dataset

file\_path = '/content/FINAL DATASET (1).csv' # Replace with the correct dataset path

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

# Display the dataset before preprocessing

print("Dataset Before Preprocessing:")

print(df.head())

print("\nMissing Values and Zero Counts (Before Preprocessing):")

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

# Preprocessing: Replace NaN and zeros with the column mean

for col in df.columns:

if df[col].dtype in ['float64', 'int64']: # Process numeric columns only

mean\_value = df.loc[df[col] != 0, col].mean() # Exclude zeros from mean calculation

df[col] = df[col].fillna(mean\_value) # Replace NaN with mean

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

# Generate crime codes (mapping crime types to unique codes)

crime\_columns = df.columns[3:] # All columns after 'Year' (adjust if needed)

crime\_codes = {crime: f"Code\_{i+1}" for i, crime in enumerate(crime\_columns)}

# Create a DataFrame to store the crime codes

crime\_code\_df = pd.DataFrame()

# Assign crime codes for each crime type and add them to the new DataFrame

for crime in crime\_columns:

crime\_code\_df[f'{crime}\_code'] = df[crime].apply(lambda x: crime\_codes.get(crime, 'Unknown'))

# Concatenate the crime code DataFrame to the original dataset

df = pd.concat([df, crime\_code\_df], axis=1)

# Display the dataset after preprocessing (with crime codes added)

print("\nDataset After Preprocessing (with crime codes added):")

print(df.head())

# Display the first 5 crime codes for display purposes

print("\nFirst 5 crime codes:")

for i in range(5):

crime = crime\_columns[i]

crime\_code = crime\_codes.get(crime, 'Unknown')

print(f"Code\_{i+1}: {crime} -> {crime\_code}")

# Save the cleaned dataset with crime codes

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

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

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

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load cleaned data

df = pd.read\_csv('CLEANED\_FINAL\_DATASET.csv')

# Frequency plot for categorical variable (States/UTs)

if 'States/UTs' in df.columns:

plt.figure(figsize=(14, 8))

df['States/UTs'].value\_counts().plot(kind='bar', color='skyblue', edgecolor='black')

plt.title("State-wise Distribution", fontsize=14)

plt.xlabel("States/UTs", fontsize=12)

plt.ylabel("Frequency", fontsize=12)

plt.xticks(rotation=45, fontsize=10)

plt.tight\_layout()

plt.show()

# Univariate Analysis: Histogram

if 'Total Cognizable IPC crimes' in df.columns:

plt.figure(figsize=(10, 6))

sns.histplot(df['Total Cognizable IPC crimes'], bins=30, kde=True, color='skyblue', edgecolor='black')

plt.title("Distribution of Total Cognizable IPC Crimes", fontsize=14)

plt.xlabel("Total Cognizable IPC Crimes", fontsize=12)

plt.ylabel("Frequency", fontsize=12)

plt.tight\_layout()

plt.show()

# Boxplot for outlier detection

if 'Year' in df.columns and 'Total Cognizable IPC crimes' in df.columns:

plt.figure(figsize=(14, 8))

sns.boxplot(data=df, x='Year', y='Total Cognizable IPC crimes', palette='Set3')

plt.title("Total Cognizable IPC Crimes by Year", fontsize=16)

plt.xlabel("Year", fontsize=12)

plt.ylabel("Total Cognizable IPC Crimes", fontsize=12)

plt.xticks(fontsize=10)

plt.yticks(fontsize=10)

plt.tight\_layout()

plt.show()

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Load cleaned data

df = pd.read\_csv('CLEANED\_FINAL\_DATASET.csv')

# Ensure the dataset includes the column for cities/states

if 'States/UTs' in df.columns:

# Select numeric columns for crimes

crime\_columns = df.select\_dtypes(include=['number']).columns

# Aggregate crime data by state

heatmap\_data = df[['States/UTs'] + list(crime\_columns)].groupby('States/UTs').sum()

# Create a simple mapping for crime codes (e.g., CC01, CC02, etc.)

crime\_codes = {crime: f"CC{str(i+1).zfill(2)}" for i, crime in enumerate(crime\_columns)}

# Rename the columns of heatmap\_data to use short codes

heatmap\_data.columns = [crime\_codes.get(crime, crime) for crime in heatmap\_data.columns]

# Plot heatmap with simple crime codes for the x-axis

plt.figure(figsize=(14, 10))

sns.heatmap(heatmap\_data, cmap='coolwarm', annot=False, linewidths=0.5, xticklabels=heatmap\_data.columns, yticklabels=heatmap\_data.index)

# Title and labels

plt.title("Crime Trends Across States", fontsize=16)

plt.xlabel("Crime Types (Codes)", fontsize=12)

plt.ylabel("States/UTs", fontsize=12)

# Rotate x and y axis labels for better readability

plt.xticks(rotation=45, fontsize=10)

plt.yticks(rotation=0, fontsize=10)

plt.tight\_layout()

plt.show()

else:

print("Column 'States/UTs' not found in the dataset.")

import pandas as pd

from sklearn.preprocessing import StandardScaler

# Load the dataset (ensure it's correctly loaded)

df = pd.read\_csv('CLEANED\_FINAL\_DATASET.csv')

# Feature Engineering

# 1. Scaling numerical features using StandardScaler

scaler = StandardScaler()

# List of numerical columns to scale (can add more columns as needed)

numerical\_columns = ['Rape other than Custodial', 'Rape\_Gang Rape', 'Kidnapping & Abduction\_Total']

# Apply scaling to the selected columns

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

# 2. One-Hot Encoding categorical variables

# 'States/UTs' and 'District' columns are categorical variables, so we'll apply one-hot encoding to them

df = pd.get\_dummies(df, columns=['States/UTs', 'District'], drop\_first=True)

# Check the dataset after feature engineering

print("Dataset after Feature Engineering:\n", df.head())

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

# Define features (X) and target variable (y)

X = df.drop('Total Cognizable IPC crimes', axis=1) # Independent variables (features)

y = df['Total Cognizable IPC crimes'] # Dependent variable (target)

# Split the data into training and testing sets (80% training, 20% testing)

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

# Display the shape of the resulting data splits

print(f"Training Data Shape: {X\_train.shape}")

print(f"Testing Data Shape: {X\_test.shape}")

import pandas as pd

from sklearn.preprocessing import LabelEncoder

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

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from xgboost import XGBRegressor

from sklearn.metrics import mean\_squared\_error

# Load the dataset (replace with your actual file path)

df = pd.read\_csv('CLEANED\_FINAL\_DATASET.csv')

# Check and clean column names (remove extra spaces or hidden characters)

df.columns = df.columns.str.strip() # Clean up column names

# Check if there are any non-numeric values that should be numeric

print(df.dtypes) # Check column data types

# Encode categorical columns ('States/UTs' and 'District') using LabelEncoder

label\_encoder = LabelEncoder()

# Encode 'States/UTs' and 'District' columns if they exist in the dataset

if 'States/UTs' in df.columns:

df['States/UTs'] = label\_encoder.fit\_transform(df['States/UTs'])

if 'District' in df.columns:

df['District'] = label\_encoder.fit\_transform(df['District'])

# Ensure all remaining categorical columns are encoded or converted to numeric

# Apply LabelEncoder or pd.get\_dummies to other potential categorical columns, if any

for column in df.select\_dtypes(include=['object']).columns:

if column not in ['States/UTs', 'District']: # Skip the already processed ones

df[column] = label\_encoder.fit\_transform(df[column])

# Define features (X) and target variable (y)

X = df.drop('Total Cognizable IPC crimes', axis=1) # Independent variables

y = df['Total Cognizable IPC crimes'] # Dependent variable

# Check if there are any non-numeric columns in the features

print(X.dtypes) # Make sure all columns are numeric

# Split the data into training and testing sets (80% training, 20% testing)

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

# Initialize the models

lr\_model = LinearRegression()

rf\_model = RandomForestRegressor(n\_estimators=100, random\_state=42)

xgb\_model = XGBRegressor(n\_estimators=100, random\_state=42)

# Train the models

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

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

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

# Make predictions on the test set

lr\_predictions = lr\_model.predict(X\_test)

rf\_predictions = rf\_model.predict(X\_test)

xgb\_predictions = xgb\_model.predict(X\_test)

# Evaluate the models using Mean Squared Error (MSE)

lr\_mse = mean\_squared\_error(y\_test, lr\_predictions)

rf\_mse = mean\_squared\_error(y\_test, rf\_predictions)

xgb\_mse = mean\_squared\_error(y\_test, xgb\_predictions)

# Print model training completion and MSE values

print("Model training complete!")

print(f"Linear Regression MSE: {lr\_mse}")

print(f"Random Forest MSE: {rf\_mse}")

print(f"XGBoost MSE: {xgb\_mse}")

import pandas as pd

from sklearn.preprocessing import LabelEncoder

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

from xgboost import XGBClassifier

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

# Load the dataset (replace with your actual file path)

df = pd.read\_csv('CLEANED\_FINAL\_DATASET.csv')

# Check and clean column names (remove extra spaces or hidden characters)

df.columns = df.columns.str.strip() # Clean up column names

# Check if there are any non-numeric values that should be numeric

print(df.dtypes) # Check column data types

# Encode categorical columns ('States/UTs' and 'District') using LabelEncoder

label\_encoder = LabelEncoder()

# Encode 'States/UTs' and 'District' columns if they exist in the dataset

if 'States/UTs' in df.columns:

df['States/UTs'] = label\_encoder.fit\_transform(df['States/UTs'])

if 'District' in df.columns:

df['District'] = label\_encoder.fit\_transform(df['District'])

# Ensure all remaining categorical columns are encoded or converted to numeric

for column in df.select\_dtypes(include=['object']).columns:

if column not in ['States/UTs', 'District']: # Skip the already processed ones

df[column] = label\_encoder.fit\_transform(df[column])

# Define features (X) and target variable (y)

X = df.drop('Total Cognizable IPC crimes', axis=1) # Independent variables

y = df['Total Cognizable IPC crimes'] # Dependent variable

# Convert target variable into binary class (you can modify the threshold)

y = (y > y.median()).astype(int) # Classifying based on median value of 'Total Cognizable IPC crimes'

# Check if there are any non-numeric columns in the features

print(X.dtypes) # Make sure all columns are numeric

# Split the data into training and testing sets (80% training, 20% testing)

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

# Initialize the XGBoost classifier

xgb\_model = XGBClassifier(n\_estimators=100, random\_state=42)

# Train the model

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

# Make predictions on the test set

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

# Evaluate the model using accuracy and classification report

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

print(f"Accuracy: {accuracy}")

print("\nClassification Report:")

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

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

# Assuming you have already trained one of the models, e.g., `lr\_model`

y\_pred = lr\_model.predict(X\_test) # Replace with your model variable

# Evaluate the model's performance

mae = mean\_absolute\_error(y\_test, y\_pred)

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

rmse = mse \*\* 0.5

r2 = r2\_score(y\_test, y\_pred)

# Output the evaluation metrics

print(f"Mean Absolute Error (MAE): {mae}")

print(f"Mean Squared Error (MSE): {mse}")

print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"R-squared (R2): {r2}")

from sklearn.model\_selection import GridSearchCV

from sklearn.ensemble import RandomForestRegressor

# Define parameter grid

param\_grid = {

'n\_estimators': [50, 100, 150],

'max\_depth': [10, 20, 30],

'min\_samples\_split': [2, 5, 10],

}

# Grid Search

grid\_search = GridSearchCV(RandomForestRegressor(random\_state=42), param\_grid, cv=5, scoring='r2')

grid\_search.fit(X\_train, y\_train)

# Get the best model from the grid search

best\_model = grid\_search.best\_estimator\_

# Print the best parameters

print("Best Parameters:", grid\_search.best\_params\_)