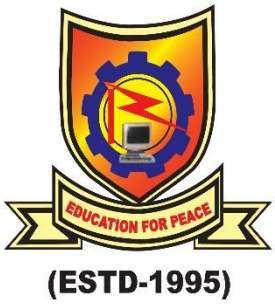

RAJEEV GANDHI MEMORIAL COLLEGE OF ENGINEERING & TECHNOLOGY (AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CRIME PREDICTION USING A MACHINE LEARNING ALGORITHM

BACKEND SOURCE CODE (USING PYTHON)

```
from flask import Flask, render_template, request
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
import joblib
from xgboost import XGBClassifier
from sklearn.neural_network import MLPClassifier


app = Flask(__name__)


# --- Machine Learning Model Training ---
```

```
try:
    df = pd.read_csv('crime_data.csv')
    df.columns = df.columns.str.strip()
    df.dropna(inplace=True)
except FileNotFoundError:
    print ("Error: 'crime_data.csv' not found. Please ensure the file is in the same directory.")
    exit ()

# Identify features and target
features = ['Year', 'City']
target = 'Type'

# Encode categorical features
le_location = LabelEncoder()
df['Location_encoded'] = le_location.fit_transform(df['City'])
le_target = LabelEncoder()
df['Type_encoded'] = le_target.fit_transform(df['Type'])

X = df[['Year', 'Location_encoded']]
y = df['Type_encoded']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

models = {
```

```
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100,
random_state=42),
    "Support Vector Machine": SVC(kernel='linear', C=1.0, random_state=42),
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=5),
    "Naive Bayes": GaussianNB(),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "XGBoost": XGBClassifier(use_label_encoder=False, eval_metric='mlogloss',
random_state=42),
    "MLP Classifier": MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000,
random_state=42)
}

models_loaded = {}
for name, model in models.items():
    print(f"Training {name}...")
    model.fit(X_train, y_train)
    models_loaded[name] = model
    print(f"{name} trained.")

joblib.dump(le_location, 'le_location.pkl')
joblib.dump(le_target, 'le_target.pkl')

@app.route('/')
def home():
    locations = sorted(df['City'].unique())
```

```
    return render_template('index.html', locations=locations)


@app.route('/predict', methods=['POST'])
def predict():
    try:
        date_str = request.form['date']
        location_name = request.form['location']
        prediction_year = int(date_str.split('-')[0])


        location_encoded = le_location.transform([location_name])[0]
        input_data = pd.DataFrame([[prediction_year, location_encoded]],
columns=['Year', 'Location_encoded'])


        all_predictions = {}
        for name, model in models_loaded.items():
            encoded_prediction = model.predict(input_data)[0]
            original_prediction =
le_target.inverse_transform([encoded_prediction])[0]
            all_predictions[name] = original_prediction


        filtered_data = df[(df['City'] == location_name) & (df['Year'] ==
prediction_year)]
        filtered_table = filtered_data.to_html(classes='table table-striped table-
bordered mt-4')
```

```
        return render_template('result.html', predictions=all_predictions,
table=filtered_table)


    except Exception as e:
        return f"An error occurred: {e}"


if __name__ == '__main__':
    app.run(debug=True)
```