1. **Code for data cleaning, testing, training and model selection**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import OneHotEncoder, StandardScaler

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

from sklearn.pipeline import Pipeline

from sklearn.linear\_model import LinearRegression

from sklearn.neighbors import KNeighborsRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoostingRegressor

from xgboost import XGBRegressor

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

from sklearn.metrics import classification\_report, confusion\_matrix

from sklearn.metrics import f1\_score

df = pd.read\_csv('/content/IndicatorS.csv')

df = df.dropna()

X = df[['CountryName', 'CountryCode', 'IndicatorName', 'IndicatorCode', 'Year']]

y = df['Value']

categorical\_features = ['CountryName', 'CountryCode', 'IndicatorName', 'IndicatorCode']

numerical\_features = ['Year']

preprocessor = ColumnTransformer(transformers=[

('cat', OneHotEncoder(handle\_unknown='ignore'), categorical\_features),

('num', StandardScaler(), numerical\_features)

])

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

model\_1 = Pipeline(steps=[('pre', preprocessor), ('reg', LinearRegression())])

model\_2 = Pipeline(steps=[('pre', preprocessor), ('reg', KNeighborsRegressor())])

model\_3 = Pipeline(steps=[('pre', preprocessor), ('reg', DecisionTreeRegressor(random\_state=42))])

model\_4 = Pipeline(steps=[('pre', preprocessor), ('reg', RandomForestRegressor(n\_estimators=100, random\_state=42))])

model\_5 = Pipeline(steps=[('pre', preprocessor), ('reg', XGBRegressor(n\_estimators=100, random\_state=42))])

model\_6 = Pipeline(steps=[('pre', preprocessor), ('reg', AdaBoostRegressor(n\_estimators=100, random\_state=42))])

model\_7 = Pipeline(steps=[('pre', preprocessor), ('reg', GradientBoostingRegressor(n\_estimators=100, random\_state=42))])

model\_1.fit(X\_train, y\_train)

model\_2.fit(X\_train, y\_train)

model\_3.fit(X\_train, y\_train)

model\_4.fit(X\_train, y\_train)

model\_5.fit(X\_train, y\_train)

model\_6.fit(X\_train, y\_train)

model\_7.fit(X\_train, y\_train)

pred\_1 = model\_1.predict(X\_test)

pred\_2 = model\_2.predict(X\_test)

pred\_3 = model\_3.predict(X\_test)

pred\_4 = model\_4.predict(X\_test)

pred\_5 = model\_5.predict(X\_test)

pred\_6 = model\_6.predict(X\_test)

pred\_7 = model\_7.predict(X\_test)

def evaluate\_model(name, y\_true, y\_pred):

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

rmse = np.sqrt(mse)

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

print(f"\n{name}")

print(f"R² Score: {r2:.4f}")

print(f"RMSE : {rmse:.4f}")

evaluate\_model("Model 1: Linear Regression", y\_test, pred\_1)

evaluate\_model("Model 2: KNN", y\_test, pred\_2)

evaluate\_model("Model 3: Decision Tree", y\_test, pred\_3)

evaluate\_model("Model 4: Random Forest", y\_test, pred\_4)

evaluate\_model("Model 5: XGBoost", y\_test, pred\_5)

evaluate\_model("Model 6: AdaBoost", y\_test, pred\_6)

evaluate\_model("Model 7: Gradient Boost", y\_test, pred\_7)

# Define models dictionary BEFORE the loop

models = {

"Linear Regression": LinearRegression(),

"KNN": KNeighborsRegressor(),

"Decision Tree": DecisionTreeRegressor(random\_state=42),

"Random Forest": RandomForestRegressor(n\_estimators=100, random\_state=42),

"XGBoost": XGBRegressor(n\_estimators=100, random\_state=42, verbosity=0),

"AdaBoost": AdaBoostRegressor(n\_estimators=100, random\_state=42),

"Gradient Boost": GradientBoostingRegressor(n\_estimators=100, random\_state=42)

}

bins = [0, 100000, 500000, float('inf')]

labels = ['Low', 'Medium', 'High']

regression\_results = []

classification\_reports = {}

confusion\_matrices = {}

for name, regressor in models.items():

print(f"\n======= {name} =======")

# Build pipeline

model = Pipeline([('pre', preprocessor), ('reg', regressor)])

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

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

# Regression evaluation

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

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"R² Score: {r2:.4f}")

print(f"RMSE : {rmse:.2f}")

regression\_results.append((name, r2, rmse))

# Simulated classification

y\_true\_binned = pd.cut(y\_test, bins=bins, labels=labels)

y\_pred\_binned = pd.cut(y\_pred, bins=bins, labels=labels)

# 🔒 Filter out any NaNs (due to values outside bin ranges)

mask = (~y\_true\_binned.isna()) & (~y\_pred\_binned.isna())

y\_true\_binned\_clean = y\_true\_binned[mask]

y\_pred\_binned\_clean = y\_pred\_binned[mask]

# Classification report

report = classification\_report(

y\_true\_binned\_clean,

y\_pred\_binned\_clean,

labels=labels,

output\_dict=False,

zero\_division=0

)

print("\nClassification Report:")

print(report)

# Confusion matrix

cm = confusion\_matrix(y\_true\_binned\_clean, y\_pred\_binned\_clean, labels=labels)

print("\nConfusion Matrix:")

print(cm)

classification\_reports[name] = report

confusion\_matrices[name] = cm

# Plot confusion matrix

plt.figure(figsize=(5, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)

plt.title(f"Confusion Matrix - {name}")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.tight\_layout()

plt.show()

# Filter original df to find CO2-related indicators

co2\_indicators = df[df['IndicatorName'].str.contains('CO2', case=False)][['IndicatorName', 'IndicatorCode']].drop\_duplicates()

print(co2\_indicators)

# Use a CO2-specific indicator (modify if you prefer another)

selected\_indicator\_name = "CO2 emissions (metric tons per capita)"

selected\_indicator\_code = "EN.ATM.CO2E.PC"

df = df[df['IndicatorName'].str.contains('CO2', case=False)]

def predict\_co2\_emission(country\_name, year):

# Get the country code

try:

country\_code = df[df['CountryName'] == country\_name]['CountryCode'].iloc[0]

except IndexError:

return f"❌ Country '{country\_name}' not found in the dataset."

# Create input row

input\_data = pd.DataFrame([{

'CountryName': country\_name,

'CountryCode': country\_code,

'IndicatorName': selected\_indicator\_name,

'IndicatorCode': selected\_indicator\_code,

'Year': year

}])

# Predict using trained XGBoost model

prediction = model\_5.predict(input\_data)[0]

returnf" Predicted CO₂ emission for {country\_name} in {year}: {prediction:.4f} metric tons per capita"

1. **Code for creating pickle file and ngrok installation with HTML code in it with app.run().**

from flask import Flask, request, render\_template\_string

from flask\_ngrok import run\_with\_ngrok

import pickle

import numpy as np

# Load your model and encoder

with open("co2\_rf\_model.pkl", "rb") as f:

model = pickle.load(f)

with open("label\_encoder.pkl", "rb") as f:

le = pickle.load(f)

# HTML template

html = '''

<!DOCTYPE html>

<html>

<head><title>CO₂ Predictor</title></head>

<body>

<h2>Predict CO₂ Emissions (kt)</h2>

<form method="POST">

Country: <input name="country"><br><br>

Year: <input name="year" type="number"><br><br>

<input type="submit" value="Predict">

</form>

{% if prediction %}

<p><strong>Prediction:</strong> {{ prediction | round(2) }} kt</p>

{% elif error %}

<p style="color:red">{{ error }}</p>

{% endif %}

</body>

</html>

'''

# Setup Flask

app = Flask(\_name\_)

run\_with\_ngrok(app)

@app.route("/", methods=["GET", "POST"])

def index():

prediction = None

error = None

if request.method == "POST":

country = request.form["country"]

year = request.form["year"]

try:

year = int(year)

if country not in le.classes\_:

error = f"'{country}' not found in training data."

else:

encoded\_country = le.transform([country])[0]

x = np.array([[encoded\_country, year]])

prediction = model.predict(x)[0]

except Exception as e:

error = str(e)

return render\_template\_string(html, prediction=prediction, error=error)

app.run()