# Import libraries

import pandas as pd

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.pipeline import Pipeline

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

import joblib

import gradio as gr

# Step 1: Load data

data = pd.read\_csv("adult 3.csv") # Adjust path if needed

# Step 2: Data cleaning

data.workclass.replace({'?': 'Others'}, inplace=True)

data.occupation.replace({'?': 'Others'}, inplace=True)

data = data[~data['workclass'].isin(['Without-pay', 'Never-worked'])]

# Remove outliers for numerical columns

data = data[(data['age'] >= 17) & (data['age'] <= 75)]

data = data[(data['educational-num'] >= 5) & (data['educational-num'] <= 16)]

# Drop redundant column

data.drop(columns=['education'], inplace=True)

# Step 3: Encode categorical columns

categorical\_cols = ['workclass', 'marital-status', 'occupation', 'relationship', 'race', 'gender', 'native-country']

encoders = {}

for col in categorical\_cols:

le = LabelEncoder()

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

encoders[col] = le

# Save encoders for deployment (changed file name here)

joblib.dump(encoders, 'employee\_encoders.pkl')

# Step 4: Train-test split

X = data.drop(columns=['income'])

y = data['income']

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

# Step 5: Define and train models

models = {

"LogisticRegression": LogisticRegression(max\_iter=1000),

"RandomForest": RandomForestClassifier(),

"KNN": KNeighborsClassifier(),

"SVM": SVC(),

"GradientBoosting": GradientBoostingClassifier()

}

results = {}

for name, model in models.items():

pipe = Pipeline([

('scaler', StandardScaler()),

('model', model)

])

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

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

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

results[name] = acc

print(f"{name} Accuracy: {acc:.4f}")

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

# Step 6: Plot model accuracies for comparison

plt.figure(figsize=(8, 5))

plt.bar(results.keys(), results.values(), color='skyblue')

plt.ylabel('Accuracy Score')

plt.title('Model Comparison')

plt.xticks(rotation=45)

plt.grid(True)

plt.show()

# Step 7: Save the best model (changed file name here)

best\_model\_name = max(results, key=results.get)

print(f"\n✅ Best model: {best\_model\_name} with accuracy {results[best\_model\_name]:.4f}")

best\_model = models[best\_model\_name]

best\_pipe = Pipeline([

('scaler', StandardScaler()),

('model', best\_model)

])

best\_pipe.fit(X\_train, y\_train)

joblib.dump(best\_pipe, "employee\_income\_model.pkl")

print("✅ Saved best model as employee\_income\_model.pkl")

# Step 8: Build Gradio app for prediction

model = joblib.load("employee\_income\_model.pkl")

encoders = joblib.load("employee\_encoders.pkl")

def predict\_income(age, workclass, fnlwgt, educational\_num, marital\_status, occupation, relationship, race, gender, hours\_per\_week, capital\_gain, capital\_loss, native\_country):

# Prepare input data dictionary

input\_data = {

"age": int(age),

"workclass": workclass,

"fnlwgt": int(fnlwgt),

"educational-num": int(educational\_num),

"marital-status": marital\_status,

"occupation": occupation,

"relationship": relationship,

"race": race,

"gender": gender,

"capital-gain": float(capital\_gain),

"capital-loss": float(capital\_loss),

"hours-per-week": int(hours\_per\_week),

"native-country": native\_country

}

# Encode categorical inputs

categorical\_cols = ['workclass', 'marital-status', 'occupation', 'relationship', 'race', 'gender', 'native-country']

for col in categorical\_cols:

le = encoders[col]

try:

input\_data[col] = le.transform([input\_data[col]])[0]

except:

# If unseen label, fallback to most common class

input\_data[col] = le.transform([le.classes\_[0]])[0]

# Create DataFrame with columns ordered as training set

df = pd.DataFrame([input\_data])

df = df[X\_train.columns] # Ensure correct column order

# Predict using loaded model

prediction = model.predict(df)[0]

return ">50K" if prediction == 1 else "<=50K"

# Define Gradio interface inputs and outputs

inputs = [

gr.Slider(17, 75, value=30, label="Age"),

gr.Dropdown(choices=list(encoders['workclass'].classes\_), label="Workclass"),

gr.Number(value=50000, label="Fnlwgt"),

gr.Slider(5, 16, value=10, label="Education Number"),

gr.Dropdown(choices=list(encoders['marital-status'].classes\_), label="Marital Status"),

gr.Dropdown(choices=list(encoders['occupation'].classes\_), label="Occupation"),

gr.Dropdown(choices=list(encoders['relationship'].classes\_), label="Relationship"),

gr.Dropdown(choices=list(encoders['race'].classes\_), label="Race"),

gr.Dropdown(choices=list(encoders['gender'].classes\_), label="Gender"),

gr.Slider(1, 100, value=40, label="Hours per Week"),

gr.Number(value=0, label="Capital Gain"),

gr.Number(value=0, label="Capital Loss"),

gr.Dropdown(choices=list(encoders['native-country'].classes\_), label="Native Country")

]

output = gr.Textbox(label="Predicted Income Category")

# Launch the Gradio app

gr.Interface(

fn=predict\_income,

inputs=inputs,

outputs=output,

title="Income Prediction App",

description="Enter personal and work details to predict income category"

).launch()