**#Data Loading and Preprocessing**

import pandas as pd

# Load the dataset

data = pd.read\_csv("/content/adult 3.csv")

# View first 10 and last 3 rows (data exploration)

data.head(10)

data.tail(3)

# Check the dataset shape

data.shape

# Check for missing values

data.isna().sum()

**#Cleaning Categorical Data**

# Display frequency count of 'workclass' values

print(data.workclass.value\_counts())

# Replace missing value '?' with 'Others' in workclass

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

print(data['workclass'].value\_counts())

# Replace '?' in occupation as well

print(data['occupation'].value\_counts())

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

print(data['occupation'].value\_counts())

# Remove rows with irrelevant workclass values

data = data[data['workclass'] != 'Without-pay']

data = data[data['workclass'] != 'Never-worked']

print(data['workclass'].value\_counts())

# Print class distribution for relationship and gender

print(data.relationship.value\_counts())

print(data.gender.value\_counts())

# Final shape after cleaning

data.shape

**#Outlier Detection and Handling**

import matplotlib.pyplot as plt

# Detect and visualize outliers in age

plt.boxplot(data['age'])

plt.show()

# Remove age outliers (<17 and >75)

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

plt.boxplot(data['age'])

plt.show()

data.shape

# Capital gain (2 boxplots shown, likely a duplicate)

plt.boxplot(data['capital-gain'])

plt.show()

# Educational number outliers

plt.boxplot(data['educational-num'])

plt.show()

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

plt.boxplot(data['educational-num'])

plt.show()

# Hours-per-week boxplot (no filtering here, just inspection)

plt.boxplot(data['hours-per-week'])

plt.show()

**#Feature Selection**

# Drop redundant 'education' feature as 'educational-num' is numerical representation

data = data.drop(columns=['education'])

**#Pipeline for Encoding, Scaling & Modeling**

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

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import Pipeline

import joblib

# Label encode categorical features

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

# 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)

# Create pipeline with scaling and Random Forest model

pipe = Pipeline([

('scaler', StandardScaler()),

('model', RandomForestClassifier())

])

# Train the pipeline

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

# Save the model pipeline and encoders

joblib.dump(pipe, 'salary\_prediction\_pipeline.pkl')

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

**#Comparing ML Models**

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

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

# Split again for comparison (can be optimized by reusing earlier split)

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

# Define models to compare

models = {

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

"RandomForest": RandomForestClassifier(),

"KNN": KNeighborsClassifier(),

"SVM": SVC(),

"GradientBoosting": GradientBoostingClassifier()

}

# Evaluate models

results = {}

for name, model in models.items():

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

preds = model.predict(X\_test)

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

results[name] = acc

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

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

# Identify and save the best performing model

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

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

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

joblib.dump(best\_model, "best\_model.pkl")

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

**#Visualization of Model Performance**

import matplotlib.pyplot as plt

# Bar plot to compare model accuracies

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()

**#Deploying Model using Gradio**

# Install gradio

pip install gradio

import gradio as gr

import numpy as np

import joblib

# Load best model and label encoders

model = joblib.load("best\_model.pkl")

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

# Get dropdown options from label encoders

workclass\_values = list(encoders['workclass'].classes\_)

marital\_status\_values = list(encoders['marital-status'].classes\_)

occupation\_values = list(encoders['occupation'].classes\_)

relationship\_values = list(encoders['relationship'].classes\_)

race\_values = list(encoders['race'].classes\_)

gender\_values = list(encoders['gender'].classes\_)

native\_country\_values = list(encoders['native-country'].classes\_)

# Define prediction function

def predict(age, workclass, fnlwgt, education\_num, marital\_status, occupation, relationship, race,

gender, capital\_gain, capital\_loss, hours\_per\_week, native\_country):

try:

# Encode categorical inputs

workclass\_enc = encoders['workclass'].transform([workclass])[0]

marital\_enc = encoders['marital-status'].transform([marital\_status])[0]

occupation\_enc = encoders['occupation'].transform([occupation])[0]

relationship\_enc = encoders['relationship'].transform([relationship])[0]

race\_enc = encoders['race'].transform([race])[0]

gender\_enc = encoders['gender'].transform([gender])[0]

native\_country\_enc = encoders['native-country'].transform([native\_country])[0]

except Exception as e:

return f"Encoding Error: {str(e)}"

# Prepare input for model

input\_data = np.array([

age, workclass\_enc, fnlwgt, education\_num, marital\_enc,

occupation\_enc, relationship\_enc, race\_enc, gender\_enc,

capital\_gain, capital\_loss, hours\_per\_week, native\_country\_enc

]).reshape(1, -1)

# Predict

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

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

# Gradio UI setup

demo = gr.Interface(

fn=predict,

inputs=[

gr.Number(label="Age"),

gr.Dropdown(workclass\_values, label="Workclass"),

gr.Number(label="Fnlwgt"),

gr.Number(label="Education-num"),

gr.Dropdown(marital\_status\_values, label="Marital Status"),

gr.Dropdown(occupation\_values, label="Occupation"),

gr.Dropdown(relationship\_values, label="Relationship"),

gr.Dropdown(race\_values, label="Race"),

gr.Dropdown(gender\_values, label="Gender"),

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

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

gr.Number(label="Hours per Week"),

gr.Dropdown(native\_country\_values, label="Native Country")

],

outputs=gr.Textbox(label="Predicted Salary Category"),

title="Employee Salary Prediction",

description="Enter employee information. Categorical features are accepted as readable strings."

)

# Run Gradio app

if \_\_name\_\_ == "\_\_main\_\_":

demo.launch()