#import libraries

import joblib

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

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy\_score

#load data set

import pandas as pd

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

print(df)

#view first 10 rows and last 5 row

print(df.head(10))

print(df.tail(3))

#check dataset

df.shape

#null values

df.isna().sum() #mean mdeian mode arbitrary

#display frequency count

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

print("\n")

print(df.occupation.value\_counts())

print("\n")

print(df['native-country'].value\_counts())

print("\n") print(df.relationship.value\_counts())

print("\n")

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

df.shape

#replace missing value

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

print("\n")

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

print("\n")

print(df.workclass.value\_counts()) print("\n") print(df.occupation.value\_counts())

#remove irrevalent data

df = df[(df['workclass'] != 'Without-pay') & (df['workclass'] != 'Never-worked')] print(df.workclass.value\_counts())

df.shape

#outlier detection in age

import matplotlib.pyplot as plt #visualization

plt.boxplot(df['age']) plt.show()

#remove age outlires (<17 and >75)

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

plt.boxplot(df['age']) plt.show()

df.shape

#education number outliers

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

print(df)

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

plt.show()

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

plt.show()

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

plt.show()

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

plt.show()

#redundant features removal

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

df

#label encoding

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

label\_encoders = {}

for col in categorical\_cols: l

e = LabelEncoder()

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

label\_encoders[col] = le

# Save encoders

joblib.dump(label\_encoders, "label\_encoders.pkl")

# Prepare features and target

X = df.drop(columns=["income"])

y = df["income"].apply(lambda x: 1 if x.strip() == ">50K" else 0)

feature\_columns = X.columns.tolist()

joblib.dump(feature\_columns, "feature\_columns.pkl")

#comparing ML ALGORITHMS

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

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

import joblib

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

models = {

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

"RandomForest": RandomForestClassifier(),

"KNN": KNeighborsClassifier(),

"SVM": SVC(),

"GradientBoosting": GradientBoostingClassifier()

}

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))

# Select best 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

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()

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

model = GradientBoostingClassifier()

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

joblib.dump(model, "salary\_model.pkl")

!pip install gradio --quiet

!pip install gradio --quiet

import gradio as gr

import os

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

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

feature\_columns = joblib.load("feature\_columns.pkl")

history = []

def predict\_salary(age, workclass, marital\_status, occupation, relationship, race, gender,

hours\_per\_week, native\_country, capital\_gain, capital\_loss, educational\_num, fnlwgt):

try:

data = pd.DataFrame([[age, workclass, marital\_status, occupation, relationship, race, gender,

hours\_per\_week, native\_country, capital\_gain, capital\_loss, educational\_num, fnlwgt]],

columns=['age', 'workclass', 'marital-status', 'occupation', 'relationship', 'race', 'gender',

'hours-per-week', 'native-country', 'capital-gain', 'capital-loss', 'educational-num', 'fnlwgt'])

for col in ['workclass', 'marital-status', 'occupation', 'relationship', 'race', 'gender', 'native-country']:

data[col] = label\_encoders[col].transform(data[col])

data = data[feature\_columns]

pred = model.predict(data)[0]

result = ">50K" if pred == 1 else "<=50K"

history.append({

"Age": age,

"Hours": hours\_per\_week,

"Capital Gain": capital\_gain,

"Capital Loss": capital\_loss,

"Education Num": educational\_num,

"fnlwgt": fnlwgt,

"Income": result

})

return f"Predicted Income: {result}"

except Exception as e:

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

def show\_history():

if not history:

return pd.DataFrame()

return pd.DataFrame(history)

def draw\_chart():

if not history:

return None

latest = pd.DataFrame([list(history[-1].values())[:-1]], columns=list(history[-1].keys())[:-1])

avg = pd.DataFrame([{

"Age": df["age"].mean(),

"Hours": df["hours-per-week"].mean(),

"Capital Gain": df["capital-gain"].mean(),

"Capital Loss": df["capital-loss"].mean(),

"Education Num": df["educational-num"].mean(),

"fnlwgt": df["fnlwgt"].mean()

}])

plt.figure(figsize=(8, 4))

plt.bar(latest.columns, latest.iloc[0], label="Latest", alpha=0.7, color='skyblue')

plt.bar(avg.columns, avg.iloc[0], label="Average", alpha=0.7, color='orange')

plt.xticks(rotation=45)

plt.legend()

plt.title("Latest Input vs Average Profile")

plt.tight\_layout()

chart\_path = "/tmp/chart.png"

plt.savefig(chart\_path)

plt.close()

return chart\_path

with gr.Blocks(theme=gr.themes.Soft()) as demo:

gr.Markdown("""

<div style="text-align:center">

<h1 style="color:#4CAF50; font-size: 32px;">💼 Employee Salary Predictor</h1>

<p style="font-size: 16px;">🔍 Estimate if a person earns more than 50K using demographic and job details</p>

</div>

""")

with gr.Tab("FORM"):

with gr.Row():

with gr.Column():

age = gr.Slider(18, 70, value=30, step=1, label="Age")

fnlwgt = gr.Number(value=100000, label="fnlwgt")

educational\_num = gr.Slider(5, 16, value=10, step=1, label="Education Number")

hours\_per\_week = gr.Slider(1, 80, value=40, step=1, label="Hours per Week")

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

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

workclass = gr.Dropdown(label\_encoders['workclass'].classes\_.tolist(), label="Workclass")

with gr.Column():

marital\_status = gr.Dropdown(label\_encoders['marital-status'].classes\_.tolist(), label="Marital Status")

occupation = gr.Dropdown(label\_encoders['occupation'].classes\_.tolist(), label="Occupation")

relationship = gr.Dropdown(label\_encoders['relationship'].classes\_.tolist(), label="Relationship")

race = gr.Dropdown(label\_encoders['race'].classes\_.tolist(), label="Race")

gender = gr.Dropdown(label\_encoders['gender'].classes\_.tolist(), label="Gender")

native\_country = gr.Dropdown(label\_encoders['native-country'].classes\_.tolist(), label="Native Country")

experience= gr.Slider(0, 100, value=50, step=1, label="experience")

predict\_btn = gr.Button("🎯 Predict Income")

output = gr.Textbox(label="Prediction")

predict\_btn.click(

fn=predict\_salary,

inputs=[age, workclass, marital\_status, occupation, relationship, race, gender,

hours\_per\_week, native\_country, capital\_gain, capital\_loss, educational\_num, fnlwgt],

outputs=output

)

with gr.Tab("📜 History"):

show\_btn = gr.Button("📋 Show History")

history\_table = gr.Dataframe()

show\_btn.click(fn=show\_history, inputs=[], outputs=history\_table)

with gr.Tab("📈 Comparison Chart"):

chart\_btn = gr.Button("📊 Show Comparison")

chart\_output = gr.Image()

chart\_btn.click(fn=draw\_chart, inputs=[], outputs=chart\_output)

os.environ["GRADIO\_SERVER\_NAME"] = "0.0.0.0"

os.environ["GRADIO\_SERVER\_PORT"] = "7860"

demo.launch(share=True)