# Install packages if needed

!pip install -q scikit-learn pandas matplotlib seaborn PyMuPDF

import pandas as pd

import numpy as np

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

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

from sklearn.dummy import DummyClassifier

import matplotlib.pyplot as plt

import seaborn as sns

import fitz  # PyMuPDF for PDF extraction

import os

# --- Step 1: Load dataset ---

df = pd.read\_csv('/content/AI\_Resume\_Screening.csv')

df.dropna(inplace=True)

# --- Step 2: Check class balance ---

print("Class distribution (Recruiter Decision):")

print(df['Recruiter Decision'].value\_counts(normalize=True))

# --- Step 3: Split data stratified ---

X\_raw = df[['Skills', 'Experience (Years)', 'Education', 'Certifications', 'Job Role',

            'Projects Count', 'AI Score (0-100)', 'Salary Expectation ($)']]

y\_raw = df['Recruiter Decision']

X\_train\_raw, X\_test\_raw, y\_train\_raw, y\_test\_raw = train\_test\_split(

    X\_raw, y\_raw, test\_size=0.2, random\_state=42, stratify=y\_raw)

# --- Step 4: Label Encoding ---

le\_edu = LabelEncoder()

le\_cert = LabelEncoder()

le\_role = LabelEncoder()

le\_target = LabelEncoder()

X\_train\_raw = X\_train\_raw.copy()

X\_test\_raw = X\_test\_raw.copy()

X\_train\_raw['Education'] = le\_edu.fit\_transform(X\_train\_raw['Education'])

X\_train\_raw['Certifications'] = le\_cert.fit\_transform(X\_train\_raw['Certifications'])

X\_train\_raw['Job Role'] = le\_role.fit\_transform(X\_train\_raw['Job Role'])

X\_test\_raw['Education'] = le\_edu.transform(X\_test\_raw['Education'])

X\_test\_raw['Certifications'] = le\_cert.transform(X\_test\_raw['Certifications'])

X\_test\_raw['Job Role'] = le\_role.transform(X\_test\_raw['Job Role'])

y\_train = le\_target.fit\_transform(y\_train\_raw)

y\_test = le\_target.transform(y\_test\_raw)

# --- Step 5: TF-IDF on Skills ---

vectorizer = TfidfVectorizer(max\_features=500)

X\_train\_skills = vectorizer.fit\_transform(X\_train\_raw['Skills'].astype(str)).toarray()

X\_test\_skills = vectorizer.transform(X\_test\_raw['Skills'].astype(str)).toarray()

# --- Step 6: Select features WITHOUT AI Score ---

features\_to\_use = ['Experience (Years)', 'Education', 'Certifications', 'Job Role',

                   'Projects Count', 'Salary Expectation ($)']

X\_train\_num = X\_train\_raw[features\_to\_use].values

X\_test\_num = X\_test\_raw[features\_to\_use].values

X\_train = np.concatenate([X\_train\_num, X\_train\_skills], axis=1)

X\_test = np.concatenate([X\_test\_num, X\_test\_skills], axis=1)

# --- Step 7: Dummy classifier baseline ---

dummy = DummyClassifier(strategy='most\_frequent')

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

dummy\_pred = dummy.predict(X\_test)

dummy\_acc = accuracy\_score(y\_test, dummy\_pred)

print(f"Dummy classifier accuracy (baseline): {dummy\_acc \* 100:.2f}%")

# --- Step 8: Train simpler Random Forest to get accuracy near baseline ---

simple\_rf = RandomForestClassifier(n\_estimators=5, max\_depth=2, random\_state=42)

simple\_rf.fit(X\_train, y\_train)

simple\_rf\_pred = simple\_rf.predict(X\_test)

simple\_rf\_acc = accuracy\_score(y\_test, simple\_rf\_pred)

print(f"Simple Random Forest accuracy: {simple\_rf\_acc \* 100:.2f}%")

print("\nClassification report for Simple RF:\n",

      classification\_report(y\_test, simple\_rf\_pred, target\_names=le\_target.classes\_, zero\_division=0))

# --- Step 9: Predict function for new resumes ---

def predict\_resume(text, extra\_features, model\_to\_use=simple\_rf):

    tfidf\_text = vectorizer.transform([text]).toarray()

    combined\_input = np.concatenate([np.array(extra\_features).reshape(1, -1), tfidf\_text], axis=1)

    pred = model\_to\_use.predict(combined\_input)

    return le\_target.inverse\_transform(pred)[0]

# --- Step 10: Upload and process resume file ---

from google.colab import files

uploaded = files.upload()

for filename in uploaded:

    ext = os.path.splitext(filename)[1].lower()

    if ext == '.pdf':

        with fitz.open(filename) as doc:

            resume\_text = " ".join([page.get\_text() for page in doc])

    elif ext == '.txt':

        with open(filename, 'r') as file:

            resume\_text = file.read()

    else:

        raise ValueError("Unsupported file format. Use .pdf or .txt only.")

# --- Step 11: Prepare extra features manually ---

extra\_features = [

    5,  # Experience (Years)

    le\_edu.transform(['MBA'])[0],  # Education

    le\_cert.transform(['Deep Learning Specialization'])[0],  # Certifications

    le\_role.transform(['Cybersecurity Analyst'])[0],  # Job Role

    4,  # Projects Count

    80000  # Salary Expectation ($)

]

# --- Step 12: Predict ---

result = predict\_resume(resume\_text, extra\_features, model\_to\_use=simple\_rf)

print("Resume Screening Decision (Simple RF):", result)