import pandas as pd

import re

import tkinter as tk

from tkinter import messagebox

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.preprocessing import LabelEncoder

from sklearn.naive\_bayes import MultinomialNB

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

from collections import Counter

# 🔹 Load dataset

file\_path = "large\_mental\_health\_dataset.csv"  # Update with actual path

df = pd.read\_csv(file\_path, encoding="utf-8")

# 🔹 Clean column names

df.columns = df.columns.str.strip().str.lower()

# 🔹 Ensure necessary columns exist

required\_columns = {"label", "text", "treatment"}

if not required\_columns.issubset(df.columns):

    raise KeyError(f"Missing columns: {required\_columns - set(df.columns)}")

# 🔹 Define stopwords

manual\_stopwords = set([

    "i", "me", "my", "myself", "we", "our", "ours", "ourselves", "you", "your",

    "yours", "yourself", "he", "him", "his", "she", "her", "hers", "it", "its",

    "they", "them", "their", "this", "that", "these", "those", "am", "is", "are",

    "was", "were", "be", "been", "have", "has", "had", "a", "an", "the", "and",

    "but", "if", "or", "because", "as", "until", "while", "of", "at", "by",

    "for", "with", "about", "against", "between", "into", "through", "before",

    "after", "above", "below", "to", "from", "up", "down", "in", "out", "on",

    "off", "over", "under", "again", "further", "then", "once", "here", "there",

    "when", "where", "why", "how", "all", "any", "both", "each", "few", "more",

    "most", "other", "some", "such", "only", "own", "same", "so", "than", "too",

    "very", "can", "will", "just", "don", "should", "now"

])

# 🔹 Clean text

def clean\_text(text):

    text = text.lower()

    text = re.sub(r"[^\w\s]", "", text)

    text = " ".join([word for word in text.split() if word not in manual\_stopwords])

    return text

# 🔹 Preprocess text

df["cleaned\_text"] = df["text"].apply(clean\_text)

# 🔹 Encode labels

label\_encoder = LabelEncoder()

df["label\_encoded"] = label\_encoder.fit\_transform(df["label"])

# 🔹 Create dictionary for treatments

treatment\_dict = dict(zip(df["label"], df["treatment"]))

# 🔹 Train/test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df["cleaned\_text"], df["label\_encoded"], test\_size=0.2, random\_state=42)

# 🔹 TF-IDF

vectorizer = TfidfVectorizer(max\_features=5000)

X\_train\_tfidf = vectorizer.fit\_transform(X\_train)

X\_test\_tfidf = vectorizer.transform(X\_test)

# 🔹 Model training

model = MultinomialNB()

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

# 🔹 Prediction for test

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

print("✅ Accuracy:", accuracy\_score(y\_test, y\_pred))

print("📊 Classification Report:\n", classification\_report(y\_test, y\_pred))

# 🔹 Visualization function

def visualize():

    plt.figure(figsize=(10, 5))

    df["label"].value\_counts().plot(kind="bar", color="skyblue")

    plt.title("Disorder Distribution")

    plt.xticks(rotation=75)

    plt.show()

    if "severity" in df.columns:

        plt.figure(figsize=(8, 5))

        sns.countplot(data=df, x="severity", palette="coolwarm")

        plt.title("Severity Levels")

        plt.show()

    plt.figure(figsize=(6, 6))

    sns.heatmap(confusion\_matrix(y\_test, y\_pred), annot=True, fmt="d", cmap="Blues",

                xticklabels=label\_encoder.classes\_, yticklabels=label\_encoder.classes\_)

    plt.title("Confusion Matrix")

    plt.show()

    df["word\_count"] = df["cleaned\_text"].apply(lambda x: len(x.split()))

    plt.figure(figsize=(8, 5))

    sns.histplot(df["word\_count"], bins=30, kde=True, color="purple")

    plt.title("Word Count in Symptoms")

    plt.show()

    all\_words = " ".join(df["cleaned\_text"]).split()

    common\_words = Counter(all\_words).most\_common(20)

    words, counts = zip(\*common\_words)

    plt.figure(figsize=(10, 5))

    sns.barplot(x=list(words), y=list(counts), palette="magma")

    plt.title("Top 20 Common Words")

    plt.xticks(rotation=75)

    plt.show()

# 🔹 GUI

def launch\_gui():

    def predict\_disorder():

        user\_input = input\_entry.get("1.0", tk.END).strip().lower()

        if not user\_input:

            messagebox.showwarning("Input Error", "Please enter your symptoms.")

            return

        cleaned\_input = clean\_text(user\_input)

        input\_vectorized = vectorizer.transform([cleaned\_input])

        predicted\_label = model.predict(input\_vectorized)[0]

        disorder\_name = label\_encoder.inverse\_transform([predicted\_label])[0]

        result\_label.config(text=f"🤔 Prediction: You might have \*{disorder\_name}\*")

        result\_label.disorder\_name = disorder\_name

        treatment\_button.pack(pady=5)

    def show\_treatment():

        disorder = getattr(result\_label, 'disorder\_name', None)

        if not disorder:

            messagebox.showinfo("Info", "Please predict a disorder first.")

            return

        treatment = treatment\_dict.get(disorder, "❌ No treatment info available.")

        result\_label.config(text=f"💡 Treatment for {disorder}: {treatment}")

    def show\_visuals():

        visualize()

    window = tk.Tk()

    window.title("🧠 Mental Health Assistant")

    window.geometry("700x550")

    window.config(bg="#f0f8ff")

    tk.Label(window, text="Mental Health Assistant", font=("Arial", 18, "bold"), bg="#f0f8ff").pack(pady=10)

    tk.Label(window, text="Describe your symptoms below:", font=("Arial", 14), bg="#f0f8ff").pack()

    input\_entry = tk.Text(window, height=6, width=70, font=("Arial", 12))

    input\_entry.pack(pady=10)

    tk.Button(window, text="Predict Disorder", command=predict\_disorder, bg="#4caf50", fg="white", font=("Arial", 12)).pack(pady=5)

    treatment\_button = tk.Button(window, text="Show Treatment", command=show\_treatment, bg="#f57c00", fg="white", font=("Arial", 12))

    treatment\_button.pack\_forget()

    tk.Button(window, text="Show Visualizations", command=show\_visuals, bg="#2196f3", fg="white", font=("Arial", 12)).pack(pady=5)

    result\_label = tk.Label(window, text="", bg="#f0f8ff", wraplength=600, font=("Arial", 13), fg="#333")

    result\_label.pack(pady=20)

    window.mainloop()

# 🔹 Run

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

    launch\_gui()