import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.ensemble import VotingClassifier

import numpy as np

# Load the dataset

file\_path = r'C:\Users\neha irfan\OneDrive\Desktop\datafile.csv'

df = pd.read\_csv(file\_path)

# Define the feature columns and target columns

feature\_columns = df.columns.difference(["feeling sad", "Irritable feel", "Trouble sleeping", "Problems concentrating", "Overeating", "Feeling anxious", "Feeling of guilt", "Problems of bonding with baby", "Suicide attempt"])

target\_columns = ["feeling sad", "Irritable feel", "Trouble sleeping", "Problems concentrating", "Overeating", "Feeling anxious", "Feeling of guilt", "Problems of bonding with baby", "Suicide attempt"]

# Initialize classifiers

rf = RandomForestClassifier(random\_state=42)

dt = DecisionTreeClassifier(random\_state=42)

svm = SVC(probability=True, random\_state=42)

lr = LogisticRegression(random\_state=42, max\_iter=10000)

gnb = GaussianNB()

mlp = MLPClassifier(random\_state=42)

models = {

'Random Forest': rf,

'Decision Tree': dt,

'SVM': svm,

'Logistic Regression': lr,

'Gaussian Naive Bayes': gnb,

'MLP': mlp

}

ensembles = {

'Ensemble (RF + DT)': VotingClassifier(estimators=[('rf', rf), ('dt', dt)], voting='soft'),

'Ensemble (RF + DT + SVM + LR)': VotingClassifier(estimators=[('rf', rf), ('dt', dt), ('svm', svm), ('lr', lr)], voting='soft'),

'Ensemble (RF + DT + SVM + LR + GaussianNB + MLP)': VotingClassifier(estimators=[('rf', rf), ('dt', dt), ('svm', svm), ('lr', lr), ('gnb', gnb), ('mlp', mlp)], voting='soft')

}

# Function to evaluate models

def evaluate\_models(X\_train, X\_test, y\_train, y\_test):

results = {}

for name, model in models.items():

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

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

results[name] = {

'Accuracy': accuracy\_score(y\_test, y\_pred),

'Precision': precision\_score(y\_test, y\_pred, average='weighted', zero\_division=0),

'Recall': recall\_score(y\_test, y\_pred, average='weighted', zero\_division=0),

'F1 Score': f1\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

}

for name, ensemble in ensembles.items():

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

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

results[name] = {

'Accuracy': accuracy\_score(y\_test, y\_pred),

'Precision': precision\_score(y\_test, y\_pred, average='weighted', zero\_division=0),

'Recall': recall\_score(y\_test, y\_pred, average='weighted', zero\_division=0),

'F1 Score': f1\_score(y\_test, y\_pred, average='weighted', zero\_division=0)

}

return results

# Loop through each target column and evaluate models

for target in target\_columns:

X = df[feature\_columns]

y = df[target]

# Using StratifiedKFold for better distribution of classes in training and test sets

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

for train\_index, test\_index in skf.split(X, y):

X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

print(f'Evaluating for target: {target}')

results = evaluate\_models(X\_train, X\_test, y\_train, y\_test)

# Print the results

for name, metrics in results.items():

print(f'{name}:')

print(f' Accuracy: {metrics["Accuracy"]:.4f}')

print(f' Precision: {metrics["Precision"]:.4f}')

print(f' Recall: {metrics["Recall"]:.4f}')

print(f' F1 Score: {metrics["F1 Score"]:.4f}')

print()