import pandas as pd  
from matplotlib import pyplot as plt  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
from sklearn import tree  
from sklearn.ensemble import RandomForestClassifier  
import warnings  
warnings.filterwarnings('ignore')  
  
  
def getData(year, path):  
 df = pd.read\_csv(path)  
 df = df[df['Year'] == year]  
 X = df[['Mean Return', 'Volatility']].values  
 Y = df['Label'].values  
 return X, Y  
  
  
def getTable(FP, TN, TP, FN):  
 TPR = TP / (TP + FN)  
 TNR = TN / (TN + FP)  
 ACC = (TP + TN) / (TP + TN + FP + FN)  
 d = {'Accuracy': [ACC], 'True positive rate': [TPR], 'True negative rate': [TNR]}  
 dfx = pd.DataFrame(data=d)  
 return dfx  
  
  
def decisionTrees():  
 *# 1. implement a decision tree and compute its accuracy for year 2* path = "TMO\_weekly\_label.csv"  
 x\_train, y\_train = getData(2021, path)  
 x\_test, y\_test = getData(2022, path)  
 clf = tree.DecisionTreeClassifier(criterion='entropy')  
 clf = clf.fit(x\_train, y\_train)  
 prediction = clf.predict(x\_test)  
 year2ACU = accuracy\_score(y\_test, prediction)  
 print("-" \* 50)  
 print("Implement decision trees.")  
 print("\nTask 1:")  
 print("The accuracy for year 2 is", year2ACU)  
  
 *# 2. compute the confusion matrix for year 2* cm = confusion\_matrix(y\_test, prediction)  
 print("\nTask 2:")  
 print('the confusion matrix is\n', cm)  
  
 *# 3. what is true positive rate and true negative rate for year 2?* TP = cm[0][0]  
 FP = cm[0][1]  
 FN = cm[1][0]  
 TN = cm[1][1]  
 dfx = getTable(FP, TN, TP, FN)  
 print("\nTask 3:")  
 print(dfx)  
  
 *# 4. implement a trading strategy based on your labels for year 2 and compare the performance  
 # with the ”buy-and-hold” strategy. Which strategy results in a larger amount at the end of the year?* df3 = pd.read\_csv("TMO\_weekly\_label.csv")  
 df3 = df3[df3['Year'] == 2022]  
 meanReturn = df3['Mean Return']  
 print("\nTask 4:")  
 print('Money earned based on buy-and-hold strategy for Year2:')  
 print("-2.2672499999999984")  
  
 meanReturn = list(meanReturn)  
 moneyEarned = 0  
 for i in range(52):  
 if prediction[i] == 'g':  
 moneyEarned = moneyEarned + meanReturn[i]  
 print('\nNew strategy: only buy when the predicted label is green.')  
 print('Money earned based on this strategy for Year2:')  
 print(moneyEarned)  
  
 print('\nStrategy based on decision trees has the larger amount at the end of the year.')  
  
  
def randomForest():  
 *# 1. take N = 1,...,10 and d = 1,2,...,5. For each value of N and d construct a random tree  
 # classifier (use ”entropy” as splitting criteria - this is the default) use your year 1 labels  
 # as training set and compute the error rate for year 2. Plot your error rates and find the best  
 # combination of N and d.* path = "TMO\_weekly\_label.csv"  
 x\_train, y\_train = getData(2021, path)  
 x\_test, y\_test = getData(2022, path)  
  
 error\_rate = []  
 result\_table = pd.DataFrame(columns=['n\_estimators', 'max\_depth', 'accuracy'])  
 for i in range(1, 11):  
 for j in range(1, 6):  
 rf = RandomForestClassifier(n\_estimators=i, max\_depth=j)  
 rf.fit(x\_train, y\_train)  
 error\_rate.append(1 - accuracy\_score(y\_test, rf.predict(x\_test)))  
 ACC = accuracy\_score(y\_test, rf.predict(x\_test))  
 result\_table.loc[len(result\_table.index)] = [i, j, ACC]  
 *#print(result\_table)  
  
 # plot the error rate* plt.plot(range(1, 11), error\_rate[:10], label="max\_depth=1")  
 plt.plot(range(1, 11), error\_rate[10:20], label="max\_depth=2")  
 plt.plot(range(1, 11), error\_rate[20:30], label="max\_depth=3")  
 plt.plot(range(1, 11), error\_rate[30:40], label="max\_depth=4")  
 plt.plot(range(1, 11), error\_rate[40:50], label="max\_depth=5")  
 plt.legend()  
 plt.xlabel("n\_estimators")  
 plt.ylabel("error rate")  
 plt.show()  
  
 best\_n = error\_rate.index(min(error\_rate)) % 10+1  
 best\_max = error\_rate.index(min(error\_rate)) % 5+1  
  
 print("-" \* 50)  
 print("Implement random forest.")  
 print("\nTask 1:")  
 print("the best n\_estimators and max\_depth are", best\_n, "and", best\_max)  
  
 *# 2. using the optimal values from year 1, compute the confusion matrix for year 2* rf = RandomForestClassifier(n\_estimators=best\_n, max\_depth=best\_max)  
 rf.fit(x\_train, y\_train)  
 cm = confusion\_matrix(y\_test, rf.predict(x\_test))  
 print("\nTask 2:")  
 print('the confusion matrix is\n', cm)  
  
 *# 3. what is true positive rate and true negative rate for year 2?* TP = cm[0][0]  
 FP = cm[0][1]  
 FN = cm[1][0]  
 TN = cm[1][1]  
 dfx = getTable(FP, TN, TP, FN)  
 print("\nTask 3:")  
 print(dfx)  
  
  
def Tips():  
 df = pd.read\_csv("tips.csv")  
 *# 1. what is the average tip (as a percentage of meal cost) for for lunch and for dinner?* print("-" \* 50)  
 print("Tips")  
 print("\nTask 1:")  
 df['tip\_percentage'] = df['tip'] / df['total\_bill'] \* 100  
 print(df.groupby('time')['tip\_percentage'].mean())  
  
 *# 2. what is average tip for each day of the week (as a percentage of meal cost)?* print("\nTask 2:")  
 print(df.groupby('day')['tip\_percentage'].mean())  
  
 *# 3. when are tips highest (which day and time)?* print("\nTask 3:")  
 print(df.groupby(['day', 'time'])['tip'].max())  
 print("When are tips highest (which day and time)?")  
 print("Sunday and dinner.")  
  
 *# 4. compute the correlation between meal prices and tips* print("\nTask 4:")  
 corr = df['total\_bill'].corr(df['tip'])  
 print("The correlation between meal prices and tips is", corr)  
  
 *# 5. is there any relationship between tips and size of the group?* print("\nTask 5:")  
 print("the correlation between tips and size of the group is", df["size"].corr(df['tip']))  
  
 *# 6. what percentage of people are smoking?* print("\nTask 6:")  
 print(df['smoker'].value\_counts(normalize=True) \* 100)  
 print("The answer is about 38.5%.")  
  
 *# 7. assume that rows in the tips.csv file are arranged in time. Are tips increasing with time in each day?* print("\nTask 7:")  
 tip\_list = []  
 for i in range(len(df) - 1):  
 tip\_list.append(df["tip"][i])  
 if df["day"][i] != df["day"][i + 1]:  
 plt.plot(tip\_list)  
 plt.show()  
 tip\_list = []  
 print("No, tips are not increasing.")  
  
 *# 8. is there any difference in correlation between tip amounts from smokers and non-smokers?* print("\nTask 8:")  
 print("The correlation between meal prices and tips for smokers is",  
 df[df['smoker'] == 'Yes']["total\_bill"].corr(df[df['smoker'] == 'Yes']['tip']))  
 print("The correlation between meal prices and tips for non-smokers is",  
 df[df['smoker'] == 'No']["total\_bill"].corr(df[df['smoker'] == 'No']['tip']))  
  
  
decisionTrees()  
randomForest()  
Tips()