# K Means:

import pandas as pd  
from matplotlib import pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import datasets  
import warnings  
warnings.filterwarnings('ignore')  
  
  
def KCluster():  
 *# 1. take k = 3 and use k-means sklearn library routing for k- means  
 # (random initialization and use the defaults). Take k = 1,2,...7,8 and compute  
 # the distortion vs. k. Use the ”knee” method to find out the best k.* data = pd.read\_csv("TMO\_weekly\_label.csv")  
 temp2021 = data[data['Year'] == 2021]  
 temp2022 = data[data['Year'] == 2022]  
 data2021 = temp2021[['Mean Return', 'Volatility']].values  
 label2021 = temp2021[['Mean Return', 'Volatility', 'Label']]  
 data2022 = temp2022[['Mean Return', 'Volatility']].values  
 label2022 = temp2022[['Mean Return', 'Volatility', 'Label']]  
  
 inertia\_list\_2022 = []  
 for i in range(2):  
 if i == 0:  
 df = data2021  
 else:  
 df = data2022  
 inertia\_list\_2021 = inertia\_list\_2022.copy()  
 inertia\_list\_2022 = []  
  
 for k in range(1, 9):  
 kmeans\_classifier = KMeans(n\_clusters=k)  
 *y\_kmeans* = kmeans\_classifier.fit\_predict(df)  
 inertia = kmeans\_classifier.inertia\_  
 inertia\_list\_2022.append(inertia)  
  
  
 *fig*, *ax* = plt.subplots(1, figsize=(7, 5))  
 plt.plot(range(1, 9), inertia\_list\_2021, marker='o', color ='green')  
 plt.plot(range(1, 9), inertia\_list\_2022, marker='o', color='red')  
 plt.legend()  
 plt.xlabel('number of clusters: k')  
 plt.ylabel('inertia')  
 plt.tight\_layout()  
 *#plt.show()* x = data2021  
 kmeans\_classifier = KMeans(n\_clusters=3)  
 y\_kmeans = kmeans\_classifier.fit\_predict(x)  
 centroids = kmeans\_classifier.cluster\_centers\_  
  
 print(y\_kmeans)  
  
 *fig*, *ax* = plt.subplots(1, figsize=(7, 5))  
 plt.scatter(x[y\_kmeans == 0, 0], x[y\_kmeans == 0, 1], s=75, c='red')  
 plt.scatter(x[y\_kmeans == 1, 0], x[y\_kmeans == 1, 1], s=75, c='green')  
 plt.scatter(x[y\_kmeans == 2, 0], x[y\_kmeans == 2, 1], s=75, c='blue')  
 plt.scatter(centroids[:, 0], centroids[:, 1], s=200, c='black', label='Centroids')  
  
 plt.xlabel('Mean Return')  
 plt.ylabel('Volatility')  
 plt.legend()  
 plt.tight\_layout()  
 plt.show()  
  
 label2021.plot.scatter(x='Mean Return', y='Volatility', c='Label')  
 label2022.plot.scatter(x='Mean Return', y='Volatility', c='Label')  
 plt.show()  
  
  
KCluster()

# SVM & Clustering

import pandas as pd  
import numpy as np  
import random  
from matplotlib import pyplot as plt  
import sklearn.metrics  
from sklearn.cluster import KMeans  
from sklearn.metrics import accuracy\_score, confusion\_matrix  
from sklearn.model\_selection import train\_test\_split  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.preprocessing import StandardScaler  
from sklearn import svm, metrics  
import warnings  
warnings.filterwarnings('ignore')  
  
  
def getTable(cm, i, all=False):  
 TP = cm[i][0][0]  
 FP = cm[i][0][1]  
 FN = cm[i][1][0]  
 TN = cm[i][1][1]  
 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)  
 if all:  
 return TP, FP, FN, TN, TPR, TNR, ACC  
 return dfx  
  
  
def Q1\_Q2():  
 df = pd.read\_csv("seeds\_dataset.csv")  
 *# Take the subset of the dataset containing your two class labels.  
 # You will use random 50/50 splits for training and testing data.* df = df[df['L'] > 1] *# R = 1: class L = 2 (negative) and L = 3 (positive)* X = df.drop('L', axis=1).values  
 scaler = StandardScaler()  
 scaler.fit(X)  
 X = scaler.transform(X)  
 Y = df['L'].values  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, Y, train\_size=0.5)  
  
 *# 1. implement a linear kernel SVM. What is your accuracy and confusion matrix?  
 # 2. implement a Gaussian kernel SVM. What is your accuracy and confusion matrix?  
 # 3. implement a polynomial kernel SVM of degree 3. What is your accuracy and confusion matrix?* result\_table = pd.DataFrame(columns=['Method', 'TP', 'FP', 'FN', 'TN', 'Accuracy', 'TPR', 'TNR'])  
 method = ['Linear SVM', 'Gaussian SVM', 'Polynomial SVM', 'Naive Bayesian NB']  
 cm = [0] \* 4  
  
 for i in range(3):  
 if i == 0:  
 kern = 'linear'  
 elif i == 1:  
 kern = 'rbf'  
 elif i == 2:  
 kern = 'poly'  
  
 svm\_classifier = svm.SVC(kernel=kern)  
 svm\_classifier.fit(x\_train, y\_train)  
 predicted = svm\_classifier.predict(x\_test)  
 accuracy = svm\_classifier.score(x\_test, y\_test)  
 cm[i] = confusion\_matrix(y\_test, predicted)  
 dfx = getTable(cm, i)  
  
 print("\nQ1 - Task", i+1, ":")  
 print('The ACC of', kern, 'SVC is', accuracy)  
 print('Confusion matrix:')  
 print(dfx)  
  
 TP, FP, FN, TN, TPR, TNR, ACC = getTable(cm, i, True)  
 result\_table.loc[len(result\_table.index)] = [method[i], TP, FP, FN, TN, ACC, TPR, TNR]  
  
 NB\_classifier = GaussianNB().fit(x\_train, y\_train)  
 accuracy = accuracy\_score(y\_test, NB\_classifier.predict(x\_test))  
 cm[3] = confusion\_matrix(y\_test, NB\_classifier.predict(x\_test))  
 dfx = getTable(cm, 3)  
  
 print("\nQ2 - Task 1:")  
 print('The ACC of Naive Bayesian is', accuracy)  
 print('Confusion matrix:')  
 print(dfx)  
  
 TP, FP, FN, TN, TPR, TNR, ACC = getTable(cm, 3, True)  
 result\_table.loc[len(result\_table.index)] = [method[3], TP, FP, FN, TN, ACC, TPR, TNR]  
  
 print('\nQ2 - Task 2:')  
 print(result\_table)  
  
  
def Q3\_Task1():  
 df = pd.read\_csv("seeds\_dataset.csv")  
 *# for k = 1 - 8 use k-means clustering with random initialization and defaults.  
 # Compute and plot distortion vs k. Use the ”knee” method to find the best k.* inertia\_list = []  
 for k in range(1, 9):  
 kmeans\_classifier = KMeans(n\_clusters=k)  
 *y\_kmeans* = kmeans\_classifier.fit\_predict(df)  
 inertia = kmeans\_classifier.inertia\_  
 inertia\_list.append(inertia)  
  
 *# print(inertia\_list)* plt.plot(range(1, 9), inertia\_list, marker='o', color='green')  
 plt.legend()  
 plt.xlabel('number of clusters: k')  
 plt.ylabel('inertia')  
 plt.tight\_layout()  
 plt.show()  
  
  
  
def Q3\_Task2\_Task3():  
 df = pd.read\_csv("seeds\_dataset.csv")  
 *# 2. rerun your clustering with best k clusters. Pick two features fi and fj at random  
 # (using python, of course) and plot your datapoints (different color for each class and  
 # centroids) using fi and fj as axis. Examine your plot. Are there any interesting patterns?* num\_list = random.sample(range(1, 7), 2)  
 num1, num2 = num\_list[0], num\_list[1]  
 *# num1, num2 = 1, 4* df1 = df[['f1', 'f2', 'f3', 'f4', 'f5', 'f6', 'f7', 'L']].values  
  
 kmeans\_classifier = KMeans(n\_clusters=3)  
 y\_kmeans = kmeans\_classifier.fit\_predict(df1)  
 centroids = kmeans\_classifier.cluster\_centers\_  
  
 percent1 = pd.DataFrame(df1[y\_kmeans == 0, 7])  
 percent2 = pd.DataFrame(df1[y\_kmeans == 1, 7])  
 percent3 = pd.DataFrame(df1[y\_kmeans == 2, 7])  
  
 *# Task 3  
 # print percentage of each cluster* print(percent1.value\_counts(normalize=True) \* 100)  
 print(percent2.value\_counts(normalize=True) \* 100)  
 print(percent3.value\_counts(normalize=True) \* 100)  
  
 print(centroids)  
  
 plt.scatter(df1[y\_kmeans == 0, num1], df1[y\_kmeans == 0, num2], s=75, c='red', label='Canadian')  
 plt.scatter(df1[y\_kmeans == 1, num1], df1[y\_kmeans == 1, num2], s=75, c='green', label='Kama')  
 plt.scatter(df1[y\_kmeans == 2, num1], df1[y\_kmeans == 2, num2], s=75, c='blue', label='Rosa')  
 plt.scatter(centroids[:, num1], centroids[:, num2], s=200, c='black', label='Centroids')  
  
 plt.xlabel('f' + str(num1))  
 plt.ylabel('f' + str(num2))  
 plt.legend()  
 plt.tight\_layout()  
 plt.show()  
  
  
Q1\_Q2()  
Q3\_Task1()  
Q3\_Task2\_Task3()