import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
import seaborn as sns  
from matplotlib import ticker  
from scipy.stats import chi2\_contingency  
from statsmodels.graphics.mosaicplot import mosaic  
from scipy.stats import multivariate\_normal  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.tree import plot\_tree  
from sklearn.model\_selection import GridSearchCV  
from sklearn.model\_selection import KFold  
from sklearn.model\_selection import RandomizedSearchCV  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import confusion\_matrix  
from sklearn.neural\_network import MLPClassifier  
from sklearn.preprocessing import StandardScaler, MinMaxScaler  
from sklearn.cluster import KMeans  
from sklearn.decomposition import PCA  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import silhouette\_score, davies\_bouldin\_score, calinski\_harabasz\_score  
from tqdm import tqdm  
from sklearn.svm import SVC  
  
###-------------------Create the data frame-------------------------------###  
  
data=pd.read\_csv('A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 1\Xy\_train.csv') #read the excel file from location on pc  
results=pd.read\_csv('A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 2\Results.csv') #read the excel file from location on pc  
results\_ann=pd.read\_csv('A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 2\Results\_ann.csv') #read the excel file from location on pc  
  
  
  
###-------------------Change invalid values -------------------------------###  
  
np.random.seed(356) #seed from normal distribution  
age120=data[data['age']<120] #subset age under 120  
agemean =np.round\_(np.mean(age120['age'])) #mean to all age under 120  
agestd = np.std(age120['age']) #std to all age under 120  
for i in data.index: #changing ages over 120 to random number fro normal distribution with agemean, agestd  
 if (data['age'].values[i] > 120):  
 data['age'].values[i] = np.round\_(np.random.normal(agemean, agestd, 1))  
  
###-------------------Changing missing values -------------------------------###  
  
#CA  
data.loc[data['ca'] == 4,'ca'] = 0  
  
#Thal  
data.loc[data['thal'] == 0,'thal'] = 2  
  
###-------------------Changing to dummies -------------------------------###  
  
#Chol changing to categorical variable  
data.loc[data['chol'] <= 200, 'chol'] = 0 #change chol to 0\1 , 0= normal 1=high  
data.loc[data['chol'] > 300, 'chol'] = 2 #change chol to 0\1 , 0= normal 1=high  
data.loc[data['chol'] > 200,'chol'] = 1 #change chol to 0\1 , 0= normal 1=high  
  
dataDummies = pd.get\_dummies(data, columns=['cp', 'restecg', 'slope', 'ca', 'thal' , 'chol'], drop\_first=True)  
dataDummies=dataDummies.drop(['id'], axis=1)  
  
#for col in data.columns:  
# print(col)  
  
#for col in dataDummies.columns:  
# print(col)  
  
###---------------------------- Validation set & Train set -----------------------------###  
  
X\_train = dataDummies.drop(['y'], axis=1).values  
y\_train = dataDummies['y'].values  
X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.2, random\_state=123)  
  
###---------------------------- Kfold 9 - Full Max Depth Tree -----------------------------###  
  
#X\_train = dataDummies.drop(['y'], axis=1).values  
#y\_train = dataDummies['y'].values  
  
kfold = KFold(n\_splits=9, shuffle=True, random\_state=123)  
DT\_res = pd.DataFrame()  
for train\_idx, val\_idx in kfold.split(X\_train):  
 modelDT = DecisionTreeClassifier(criterion='entropy', random\_state=123)  
 modelDT.fit(X\_train[train\_idx], y\_train[train\_idx])  
 accTrain=accuracy\_score(y\_true=y\_train[train\_idx], y\_pred=modelDT.predict(X\_train[train\_idx]))  
 accVal = accuracy\_score(y\_train[val\_idx], modelDT.predict(X\_train[val\_idx]))  
 DT\_res = DT\_res.append({'accVal': accVal , 'accTrain' : accTrain}, ignore\_index=True)  
  
print("Max Depth Tree Performances:")  
print(round(DT\_res,3))  
print(round(DT\_res.mean(),3))  
  
preds\_DT = modelDT.predict(X\_val)  
print("Max Depth Tree- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_DT), 3))  
print()  
  
  
###--------------------------------------Hyperparameter tuning - Tree-------------------------------###  
  
DTGrid= {'max\_depth' : np.arange(1, 11, 1),  
 'criterion' : ['gini', 'entropy'],  
 'splitter' : ['best', 'random']}  
  
grid\_search = GridSearchCV(estimator=DecisionTreeClassifier(random\_state=42), param\_grid=DTGrid, refit=True, cv=9)  
grid\_search.fit(X\_train, y\_train)  
  
  
#Show Results  
#results\_dt=pd.DataFrame(grid\_search.cv\_results\_)  
#results\_dt=results\_dt.drop(['mean\_fit\_time','std\_fit\_time','mean\_score\_time','std\_score\_time','rank\_test\_score'],1)  
#results\_dt.to\_csv("A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 2\Results.csv")  
  
print("Best DT Hyper Parameters: ", grid\_search.best\_params\_)  
print("Best DT Score: ", round(grid\_search.best\_score\_,3))  
  
best\_model = grid\_search.best\_estimator\_  
preds\_DT = best\_model.predict(X\_val)  
print("DT- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_DT), 3))  
print()  
  
DTGrid\_df = pd.DataFrame(columns=['index','max\_depth', 'criterion', 'splitter', 'train ac', 'val ac'])  
criterion\_list = {'gini', 'entropy'}  
splitter\_list={'best', 'random'}  
j=0  
'''  
for c in criterion\_list:  
 for s in splitter\_list:  
 for i in np.arange(1,11,1):  
 model\_csi=DecisionTreeClassifier(criterion=c, splitter=s, max\_depth=i, random\_state=42)  
 model\_csi.fit(X\_train, y\_train)  
 preds\_csi = model\_csi.predict(X\_val)  
 DTGrid\_df.at[j,'index'] = j  
 DTGrid\_df.at[j,'max\_depth'] = i  
 DTGrid\_df.at[j,'criterion'] = c  
 DTGrid\_df.at[j,'splitter'] = s  
 DTGrid\_df.at[j,'train ac'] = round(results['mean\_test\_score'][j],3)  
 DTGrid\_df.at[j,'val ac'] = round(accuracy\_score(y\_val,preds\_csi),3)  
 j=j+1  
'''  
for j in np.arange(0,40,1):  
 model\_csi = DecisionTreeClassifier(criterion=results['param\_criterion'][j], splitter=results['param\_splitter'][j], max\_depth=results['param\_max\_depth'][j], random\_state=42)  
 model\_csi.fit(X\_train, y\_train)  
 preds\_csi = model\_csi.predict(X\_val)  
 DTGrid\_df.at[j, 'index'] = j  
 DTGrid\_df.at[j, 'max\_depth'] = results['param\_max\_depth'][j]  
 DTGrid\_df.at[j, 'criterion'] = results['param\_criterion'][j]  
 DTGrid\_df.at[j, 'splitter'] = results['param\_splitter'][j]  
 DTGrid\_df.at[j, 'train ac'] = round(results['mean\_test\_score'][j], 3)  
 DTGrid\_df.at[j, 'val ac'] = round(results['val ac'][j], 3)  
 #DTGrid\_df.at[j, 'val ac'] = round(accuracy\_score(y\_val, preds\_csi), 3)  
 #results.at['val ac'][j]= round(accuracy\_score(y\_val, preds\_csi), 3)  
  
#results.to\_csv("C:/Users/shoha/Desktop/results.csv")  
  
print(DTGrid\_df)  
plt.figure(figsize=(13, 4))  
plt.plot(DTGrid\_df['index'], DTGrid\_df['train ac'], marker='o', markersize=4)  
plt.plot(DTGrid\_df['index'], DTGrid\_df['val ac'], marker='o', markersize=4)  
plt.legend(['Train accuracy', 'Validation accuracy'])  
#plt.show()  
  
  
'''  
res = pd.DataFrame()  
max\_depth\_list = np.arange(1, 10, 1)  
criterion\_list = {'gini', 'entropy'}  
splitter\_list={'best', 'random'}  
#min\_samples\_leaf\_list = np.arange(1,5,1)  
#max\_features\_list= {'auto', 'sqrt', 'log2' }  
bestmodel = DecisionTreeClassifier(criterion='gini', splitter='random', max\_depth=7, random\_state=123)  
for train\_idx, val\_idx in kfold.split(X\_train):  
 for criterion in criterion\_list:  
 for splitter in splitter\_list:  
 for max\_depth in max\_depth\_list:  
 model = DecisionTreeClassifier(criterion=criterion,max\_depth=max\_depth ,splitter=splitter, random\_state=123)  
 model.fit(X\_train[train\_idx], y\_train[train\_idx])  
 acc = accuracy\_score(y\_train[val\_idx], model.predict(X\_train[val\_idx]))  
 res = res.append({ 'criterion' : criterion,'splitter':splitter,'max\_depth': max\_depth,'acc': acc ,  
 'train\_acc': accuracy\_score(y\_train, model.predict(X\_train))}, ignore\_index=True)  
 bestmodel.fit(X\_train[train\_idx], y\_train[train\_idx])  
  
  
print(res[['max\_depth', 'criterion','splitter','acc' ]].groupby(['max\_depth' , 'splitter' ,'criterion']).mean().reset\_index().sort\_values('acc', ascending=False).head(3))  
#print(res[['max\_depth', 'criterion','acc' ]].groupby(['max\_depth' , 'criterion']).std().reset\_index().sort\_values('acc', ascending=False).head(10))  
  
###------------------------create res2 in order to get max\_depth plot ---------------------------------------------##  
  
res2 = res.drop(['criterion' ,'splitter' ] ,axis= 1)  
res2 = res2[['max\_depth','acc','train\_acc']].groupby(['max\_depth']).mean().reset\_index()  
  
plt.figure(figsize=(13, 4))  
plt.plot(res2['max\_depth'], res2['train\_acc'], marker='o', markersize=4)  
plt.plot(res2['max\_depth'], res2['acc'], marker='o', markersize=4)  
plt.legend(['Train accuracy', 'Validation accuracy'])  
plt.show()  
  
###---------------------------------------------------print trees--------------------------------------------------------------##  
  
plt.figure(figsize=(12, 10))  
plot\_tree(bestmodel, filled=True, max\_depth=3,class\_names=['no Heart attak', 'Heart attak'], feature\_names=['age', 'gender', 'trestbps','fbs','thalach','exang' ,  
 'oldpeak','cp\_1','cp\_2','cp\_3','restecg\_1','restecg\_2',  
 'slope\_1','slope\_2','ca\_1','ca\_2','ca\_3', 'thal\_2','thal\_3',  
 'chol\_1','chol\_2'])  
plt.show()  
  
plt.figure(figsize=(12, 10))  
plot\_tree(bestmodel, filled=True, class\_names=['no Heart attak', 'Heart attak'], feature\_names=['age', 'gender', 'trestbps','fbs','thalach','exang' ,  
 'oldpeak','cp\_1','cp\_2','cp\_3','restecg\_1','restecg\_2',  
 'slope\_1','slope\_2','ca\_1','ca\_2','ca\_3', 'thal\_2','thal\_3',  
 'chol\_1','chol\_2'])  
plt.show()  
  
###---------------------------------------------------features importances---------------------------------------------##  
print('features importances:')  
print(bestmodel.feature\_importances\_)  
  
'''  
###---------------------------------------------------ANN-scale values--------------------------------------------------------------##  
'''  
#X\_train = data.drop(['y'], axis=1)  
y\_train = data['y'].values  
  
scaled\_features1 = data.copy()  
scaled\_features = pd.get\_dummies(scaled\_features1, columns=['cp', 'restecg', 'slope', 'ca', 'thal' , 'chol'], drop\_first=True)  
col\_names = ['age', 'trestbps' , 'oldpeak' , 'thalach']  
features = scaled\_features[col\_names]  
scaler = StandardScaler().fit(features.values)  
features = scaler.transform(features.values)  
scaled\_features[col\_names] = features  
X\_train\_SK = scaled\_features.drop(['y'], axis=1).values  
print(X\_train\_SK)  
'''  
  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_val\_scaled = scaler.fit\_transform(X\_val)  
  
###---------------------------------------------------ANN-default network--------------------------------------------------------------##  
  
ANNres = pd.DataFrame()  
for train\_idx, val\_idx in kfold.split(X\_train\_scaled):  
 modelANN= MLPClassifier(activation='relu', alpha=0.0001, batch\_size='auto', beta\_1=0.9,  
 beta\_2=0.999, early\_stopping=False, epsilon=1e-08,  
 hidden\_layer\_sizes=(100,) , learning\_rate='constant',  
 learning\_rate\_init=0.001, max\_fun=15000, max\_iter=200,  
 momentum=0.9, n\_iter\_no\_change=10, nesterovs\_momentum=True,  
 power\_t=0.5, random\_state=None, shuffle=True, solver='adam',  
 tol=0.0001, validation\_fraction=0.1, verbose=False,  
 warm\_start=False)  
 modelANN.fit(X\_train\_scaled[train\_idx], y\_train[train\_idx])  
 accTrain2=accuracy\_score(y\_true=y\_train[train\_idx], y\_pred=modelANN.predict(X\_train\_scaled[train\_idx]))  
 accVal2 = accuracy\_score(y\_train[val\_idx], modelANN.predict(X\_train\_scaled[val\_idx]))  
 ANNres = ANNres.append({'accVal': accVal2 , 'accTrain' : accTrain2}, ignore\_index=True)  
  
print("Default ANN Performances:")  
print(round(ANNres,3))  
print(round(ANNres.mean(),3))  
  
preds\_ANN = modelANN.predict(X\_val\_scaled)  
print("Default ANN- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_ANN), 3))  
print()  
  
###---------------------------------------------------ANN-Best network--------------------------------------------------------------##  
  
AnnGrid= {'activation' : ['logistic', 'tanh', 'relu'],  
 'max\_iter' : np.arange(100,1000,50),  
 'learning\_rate\_init' : np.arange(0.001 ,0.050, 0.005),  
 'solver' : ['lbfgs' , 'sgd' , 'adam' ],  
 'hidden\_layer\_sizes' : [(10,),(10,10,),(10,10,10,),(10,10,10,10,),(20,),(20,20,),(20,20,20,),(20,20,20,20,) ]}  
  
random\_search = RandomizedSearchCV(MLPClassifier(random\_state=42), param\_distributions=AnnGrid, cv=9, random\_state=123, n\_iter=10, refit=True)  
random\_search.fit(X\_train\_scaled, y\_train)  
print("Best ANN Hyper Parameters: ", random\_search.best\_params\_)  
print("Best ANN Score: ", round(random\_search.best\_score\_,3))  
  
best\_model\_ANN = random\_search.best\_estimator\_  
preds\_ANN = best\_model\_ANN.predict(X\_val\_scaled)  
print("ANN- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_ANN), 3))  
  
#Show Results  
#results\_ann=pd.DataFrame(random\_search.cv\_results\_)  
#results\_ann=results\_ann.drop(['mean\_fit\_time','std\_fit\_time','mean\_score\_time','std\_score\_time','rank\_test\_score'],1)  
#results\_ann.to\_csv("A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 2\Results.csv")  
  
ANNGrid\_df = pd.DataFrame(columns=['index','activation', 'max\_iter', 'learning\_rate\_init', 'solver', 'hidden\_layer\_sizes','train ac', 'val ac'])  
activation\_list = {'idntity', 'logistic', 'tanh', 'relu'}  
max\_iter\_list= np.arange(100,1000,50),  
learning\_rate\_init\_list=np.arange(0.001 ,0.050, 0.005)  
solver\_list = {'lbfgs' , 'sgd' , 'adam'}  
hidden\_layer\_sizes\_list= {(10,),(10,10,),(10,10,10,),(10,10,10,10,),(20,),(20,20,),(20,20,20,),(20,20,20,20,)}  
'''  
for j in np.arange(0,40,1):  
 model\_csi = DecisionTreeClassifier(criterion=results['param\_criterion'][j], splitter=results['param\_splitter'][j], max\_depth=results['param\_max\_depth'][j], random\_state=42)  
 model\_csi.fit(X\_train, y\_train)  
 preds\_csi = model\_csi.predict(X\_val)  
 DTGrid\_df.at[j, 'index'] = j  
 DTGrid\_df.at[j, 'max\_depth'] = results['param\_max\_depth'][j]  
 DTGrid\_df.at[j, 'criterion'] = results['param\_criterion'][j]  
 DTGrid\_df.at[j, 'splitter'] = results['param\_splitter'][j]  
 DTGrid\_df.at[j, 'train ac'] = round(results['mean\_test\_score'][j], 3)  
 DTGrid\_df.at[j, 'val ac'] = round(results['val ac'][j], 3)  
 #DTGrid\_df.at[j, 'val ac'] = round(accuracy\_score(y\_val, preds\_csi), 3)  
 #results.at['val ac'][j]= round(accuracy\_score(y\_val, preds\_csi), 3)  
  
#results.to\_csv("A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 2\Results.csv")  
'''  
#model2 = MLPClassifier(activation=results\_ann['param\_activation'][0],hidden\_layer\_sizes=(results\_ann['param\_hidden\_layer\_sizes'][0]),  
 # learning\_rate\_init=results\_ann['param\_learning\_rate\_init'][0],max\_iter=results\_ann['param\_max\_iter'][0],  
 # random\_state=42, solver=results\_ann['param\_solver'][0])  
#model2.fit(X\_train\_scaled, y\_train)  
#print(model2.get\_params())  
  
  
for j in np.arange(0,100,1):  
 str = results\_ann['param\_hidden\_layer\_sizes'][j]  
 s2 = ","  
 number = str[1] + str[2]  
 numberof = str.count(s2)  
 if (numberof == 1):  
 x = (int(number),)  
 if (numberof == 2):  
 x = (int(number), int(number),)  
 if (numberof == 3):  
 x = (int(number), int(number), int(number),)  
 if (numberof == 4):  
 x = (int(number), int(number), int(number), int(number),)  
  
 model\_i = MLPClassifier(activation=results\_ann['param\_activation'][j], max\_iter=results\_ann['param\_max\_iter'][j],  
 learning\_rate\_init=results\_ann['param\_learning\_rate\_init'][j], solver=results\_ann['param\_solver'][j],  
 hidden\_layer\_sizes=x, random\_state=42)  
 model\_i.fit(X\_train\_scaled,y\_train)  
 preds\_i = model\_i.predict(X\_val\_scaled)  
 ANNGrid\_df.at[j, 'index'] = j  
 ANNGrid\_df.at[j, 'activation'] = results\_ann['param\_activation'][j]  
 ANNGrid\_df.at[j, 'max\_iter'] = results\_ann['param\_max\_iter'][j]  
 ANNGrid\_df.at[j, 'learning\_rate\_init'] = results\_ann['param\_learning\_rate\_init'][j]  
 ANNGrid\_df.at[j, 'solver'] = results\_ann['param\_solver'][j]  
 ANNGrid\_df.at[j, 'hidden\_layer\_sizes'] = results\_ann['param\_hidden\_layer\_sizes'][j]  
 ANNGrid\_df.at[j, 'train ac'] = round(results\_ann['mean\_test\_score'][j], 3)  
 #ANNGrid\_df.at[j, 'val ac'] = round(results\_ann['val ac'][j], 3)  
 ANNGrid\_df.at[j, 'val ac'] = round(accuracy\_score(y\_val, preds\_i), 3)  
 results\_ann.at[j,'val ac']= round(accuracy\_score(y\_val, preds\_i), 3)  
  
results\_ann.to\_csv("A:\אוניברסיטה\שנה ג\סמסטר ב\משין\פרויקט\חלק 2\Results\_ann.csv")  
  
  
  
'''  
print(DTGrid\_df)  
plt.figure(figsize=(13, 4))  
plt.plot(DTGrid\_df['index'], DTGrid\_df['train ac'], marker='o', markersize=4)  
plt.plot(DTGrid\_df['index'], DTGrid\_df['val ac'], marker='o', markersize=4)  
plt.legend(['Train accuracy', 'Validation accuracy'])  
plt.show()  
'''  
  
###--------------------------------------------------- PCA --------------------------------------------------------------##  
'''  
pca = PCA(n\_components=2)  
pca.fit(X\_train\_scaled)  
X\_train\_pca = pca.transform(X\_train\_scaled)  
X\_train\_pca = pd.DataFrame(X\_train\_pca, columns=['PC1', 'PC2'])  
  
pca.fit(X\_val\_scaled)  
X\_val\_pca = pca.transform(X\_val\_scaled)  
X\_val\_pca = pd.DataFrame(X\_val\_pca, columns=['PC1', 'PC2'])  
  
#X\_train\_pca['y'] = y\_train  
#sns.scatterplot(x='PC1', y='PC2', hue='y', data=X\_train\_pca)  
#plt.show()  
  
###--------------------------------------------------- KMeans- default clustring ------------------------------------##  
  
Kres = pd.DataFrame()  
for train\_idx, val\_idx in kfold.split(X\_train\_pca.values):  
 kmeans = KMeans(n\_clusters=2,init= 'k-means++',n\_init = 10,max\_iter= 300, tol= 1e-4,  
 precompute\_distances= 'deprecated',verbose= 0, random\_state= None,  
 copy\_x= True,n\_jobs= 'deprecated', algorithm= 'auto')  
 modelK=kmeans.fit(X\_train\_pca.values[train\_idx])  
 accVal3 = accuracy\_score(y\_train[val\_idx], modelK.predict(X\_train\_pca.values[val\_idx]))  
 accTrain3 = accuracy\_score(y\_true=y\_train[train\_idx], y\_pred=modelK.predict(X\_train\_pca.values[train\_idx]))  
 Kres = Kres.append({'accVal': accVal3, 'accTrain': accTrain3}, ignore\_index=True)  
  
print("Default KMeans Performances:")  
print(round(Kres,3))  
print(round(Kres.mean(),3))  
  
preds\_KMeans = modelK.predict(X\_val\_pca)  
print("Default KMeans- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_KMeans), 3))  
print()  
  
###--------------------------------------------------- KMeans- K clustres ------------------------------------##  
  
fig, axes = plt.subplots(nrows=2, ncols=4, figsize=(20, 10))  
for n\_clusters in range(2, 10, 1):  
 kmeans = KMeans(n\_clusters=n\_clusters, max\_iter=300, n\_init=10, random\_state=42)  
 kmeans.fit(X\_train\_pca)  
 assignments = kmeans.predict(X\_train\_pca)  
 scheme = X\_train\_pca.copy()  
 scheme = pd.DataFrame(scheme, columns=['PC1', 'PC2'])  
 scheme['cluster'] = assignments  
 i = 0 if n\_clusters in [2, 3, 4, 5] else 1  
 j = 0  
 j = 1 if n\_clusters in [3, 7] else j  
 j = 2 if n\_clusters in [4, 8] else j  
 j = 3 if n\_clusters in [5, 9] else j  
 sns.scatterplot(x='PC1', y='PC2', data=scheme, hue='cluster', ax=axes[i, j], palette='Accent\_r', legend=False)  
#plt.show()  
  
iner\_list = []  
dbi\_list = []  
sil\_list = []  
ch\_list = []  
for n\_clusters in tqdm(range(2, 10, 1)):  
 kmeans = KMeans(n\_clusters=n\_clusters, max\_iter=300, n\_init=10, random\_state=42)  
 kmeans.fit(X\_train\_pca)  
 assignment = kmeans.predict(X\_train\_pca)  
 iner = kmeans.inertia\_  
 sil = silhouette\_score(X\_train\_pca, assignment)  
 dbi = davies\_bouldin\_score(X\_train\_pca, assignment)  
 ch = calinski\_harabasz\_score(X\_train\_pca, assignment)  
 dbi\_list.append(dbi)  
 sil\_list.append(sil)  
 ch\_list.append(ch)  
 iner\_list.append(iner)  
  
plt.plot(range(2, 10, 1), iner\_list, marker='o')  
plt.title("Inertia")  
plt.xlabel("Number of clusters")  
#plt.show()  
  
plt.plot(range(2, 10, 1), sil\_list, marker='o')  
plt.title("Silhouette")  
plt.xlabel("Number of clusters")  
#plt.show()  
  
plt.plot(range(2, 10, 1), dbi\_list, marker='o')  
plt.title("Davies-Bouldin")  
plt.xlabel("Number of clusters")  
#plt.show()  
  
plt.plot(range(2, 10, 1), ch\_list, marker='o')  
plt.title("Calinski-Harabasz")  
plt.xlabel("Number of clusters")  
#plt.show()  
  
###--------------------------------------------------- SVM ------------------------------------##  
  
print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")  
  
SVMGrid= {'kernel' : ['linear', 'poly', 'rbf', 'sigmoid'], 'C' : np.arange(0.5, 3.5, 0.5), 'gamma' : ['scale', 'auto']}  
  
grid\_searchSVM = GridSearchCV(estimator=SVC(random\_state=42), param\_grid=SVMGrid, refit=True, cv=9)  
grid\_searchSVM.fit(X\_train\_pca, y\_train)  
print("Best SVM Hyper Parameters: ", grid\_searchSVM.best\_params\_)  
print("Best SVM Score: ", round(grid\_searchSVM.best\_score\_,3))  
  
best\_modelSVM = grid\_searchSVM.best\_estimator\_  
preds\_SVM = best\_modelSVM.predict(X\_val\_pca)  
print("before")  
print("SVM- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_SVM), 3))  
print(confusion\_matrix(y\_true=y\_val, y\_pred=preds\_SVM))  
  
for i in np.arange(0,len(preds\_SVM),1):  
 if(preds\_SVM[i]==1):  
 preds\_SVM[i]=0  
 else: preds\_SVM[i]=1  
  
print("SVM- Validation accuracy: ", round(accuracy\_score(y\_val, preds\_SVM), 3))  
print("after")  
print(confusion\_matrix(y\_true=y\_val, y\_pred=preds\_SVM))  
print()  
'''