```
#import numpy as np
#import pandas as pd
#from scipy import stats
#from pandas.core.dtypes.common import is_numeric_dtype

#from sklearn.decomposition import PCA
#from sklearn.preprocessing import StandardScaler
#from factor_analyzer import FactorAnalyzer,calculate_bartlett_sphericity,calculate_kmo
#from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
#from sklearn.model_selection import train_test_split

#import scipy.cluster.hierarchy as hclust
#from sklearn.metrics import silhouette_score
#from sklearn.metrics import silhouette_samples

#import matplotlib.pyplot as plt
#import seaborn as sb

#from sklearn.metrics import confusion_matrix,accuracy_score

# df_mortalitate = pd.read_csv('Mortalitate.csv',index_col=0)
# df_coduri = pd.read_csv('CoduriTariExtins.csv',index_col=0)

# # ---- Analiza Componentelor Principale
#
# def nan_replace(t):
#     assert isinstance(t,pd.DataFrame)
#     for i in t.columns:
```

```
#         if any(t[i].isna()):
#             t[i].fillna(t[i].mean(),inplace=True)
#         else:
#             t[i].fillna(t[i].mode()[0],inplace=True)
#
# nan_replace(df_mortalitate)
# nan_replace(df_coduri)
#
# set_date = list(df_mortalitate.columns)[1:]
# x = df_mortalitate[set_date].values
#
# # Standardizam datele
# scaler = StandardScaler()
# date_standardizate = scaler.fit_transform(df_mortalitate)
#
# # Model ACP
# modelACP = PCA()
# C = modelACP.fit_transform(date_standardizate) # C - componente principale
#
# # Varianta componentelor principale
# varianta = modelACP.explained_variance_
# # print(varianta)
# # Scoruri
# scoruri = C / np.sqrt(varianta)
# etichetari = ['C'+str(i+1) for i in range(len(varianta))]
# df_scoruri = pd.DataFrame(data=scoruri,index=df_mortalitate.index,columns=etichetari)
# print(df_scoruri)
#
# plt.figure("Plot scoruri in cele 2 axe principale", figsize=(9,9))
# plt.subplot(1,1,1)
# plt.xlabel('C1')
# plt.ylabel('C2')
# plt.title('Plot scoruri in cele doua axe principale')
# plt.scatter(df_scoruri['C1'], df_scoruri['C2'],color='r')
# for index, (x,y) in df_scoruri[['C1','C2']].iterrows():
#     plt.text(x,y,index)
# # plt.show()
# #
# # Corelatii factoriale
# r_x_c = np.corrcoef(date_standardizate,C, rowvar=False)[:len(varianta),len(varianta):]
# eticheta_corel = ['C'+str(i+1) for i in range( len(varianta))]
# df_corelatii = pd.DataFrame(data=r_x_c,index=df_mortalitate.columns,columns=[eticheta_corel])
# # print(df_corelatii)
#
# # Corelograma
# plt.figure('Corelograma',figsize=(9,9))
# plt.subplot(1,1,1)
# sb.heatmap(data=r_x_c,vmin=-1,vmax=1,cmap='bwr',annot=True)
# plt.title("Corelograma factoriala")
# # plt.show()
#
# # Calcul Contributii
# C_patrat = C * C
# contributii = C_patrat / np.sum(C_patrat,axis=0)
# df_contributii = pd.DataFrame(data=contributii,index=df_mortalitate.index,columns=eticheta_corel)
# # print(df_contributii)
#
# # Calcul Cosinusuri
# cosinusuri = np.transpose(C_patrat.T / np.sum(C_patrat,axis=1))
# df_cosinusuri = pd.DataFrame(data=cosinusuri,index=df_mortalitate.index,columns=eticheta_corel)
# print(df_cosinusuri)
#
# # Comunalitati
# comunalitati = np.cumsum(r_x_c * r_x_c,axis=1)
# df_comunalitati = pd.DataFrame(data=comunalitati,index=df_mortalitate.columns,columns=eticheta_corel)
# print(df_comunalitati)
#
# #Corelograma Comunitati
# plt.figure("Corelegrama comunalitati",figsize=(9,9))
# plt.subplot(1,1,1)
# sb.heatmap(data=df_comunalitati,vmin=-1,vmax=1,cmap='bwr',annot=True)
# plt.title("Corelograma comunalitati")
# plt.show()

# # ----------  Analiza Factoriala
# df_voturi = pd.read_csv("VotBUN.csv",index_col=0)
# df_coduri = pd.read_csv("Coduri_Localitati.csv",index_col=0)
#
# def nan_replace(t):
#     assert isinstance(t,pd.DataFrame)
#     for i in t.columns:
```

```
#         if any(t[i].isna()):
#             if is_numeric_dtype(t[i]):
#                 t[i].fillna(t[i].mean(),inplace=True)
#             else:
#                 t[i].fillna(t[i].mode()[0],inplace=True)
# nan_replace(df_voturi)
# nan_replace(df_coduri)
# set_date = list(df_voturi.columns[1:])
# df_voturiFaraLoc = df_voturi[set_date]
#
# # Standardizare
# scaler = StandardScaler()
# date_standardizate = scaler.fit_transform(df_voturiFaraLoc)
# df_standardizat = pd.DataFrame(data=date_standardizate,index=df_voturiFaraLoc.index,columns=df_voturiFaraLoc.columns)
# # print(df_standardizat)
#
#
# x = df_standardizat[set_date]
# nr_var = len(x.columns)
#
# # Model AF
# modelAF = FactorAnalyzer(n_factors=nr_var, rotation=None) #sau 'varimax'
# F = modelAF.fit(x)
#
# # Scoruri AF
# scoruri = modelAF.transform(x)
# etichete_scoruri = ['F'+str(i+1) for i in range(len(set_date))]
# df_scoruri = pd.DataFrame(data=scoruri,index=df_voturiFaraLoc.index, columns=etichete_scoruri)
# # print(df_scoruri)
#
# # Plot Scoruri
# plt.figure("Plot scoruri",figsize=(9,9))
# plt.subplot(1,1,1)
# plt.xlabel = 'F1'
# plt.ylabel= 'F2'
# plt.scatter(df_scoruri['F1'],df_scoruri['F2'],color='y')
# plt.title("Plot scoruri")
# for index,(X,Y) in df_scoruri[['F1','F2']].iterrows():
#     plt.text(X,Y,index)
#
# # Testul Bartlett
# bartlett_test = calculate_bartlett_sphericity(x)
# print("P-Value: ",bartlett_test[1]) #0.01
#
# # Testul KMO
# kmo_test = calculate_kmo(x)
# print("KMO Value: ",kmo_test[1]) #0.05
#
#
# # Varianta factori
# varianta = modelAF.get_factor_variance()[0]
# print("Varianta: ",varianta)
#
# #Corelatii Factoriale
# corelatii = modelAF.loadings_
# df_corelatii = pd.DataFrame(data=corelatii, index=df_voturiFaraLoc.columns,columns=etichete_scoruri)
# print(df_corelatii)
#
# plt.figure("Corelograma Factoriala", figsize=(9,9))
# plt.subplot(1,1,1)
# plt.title("Corelograma Factoriala")
# sb.heatmap(data=df_corelatii,vmax=1,vmin=-1,annot=True)
#
# #Comunalitati
# comunalitati = modelAF.get_communalities()
# df_comunalitati = pd.DataFrame(data=comunalitati,index=set_date,columns=['Comunalitati'])
# print(df_comunalitati)
#
# plt.figure("Corelograma Comunalitati", figsize=(9,9))
# plt.subplot(1,1,1)
# plt.title("Corelograma Comunalitati")
# sb.heatmap(data=df_comunalitati,vmin=0,annot=True)
#
# plt.show()

# ---------------- Analiza de clusteri
# df_acohol = pd.read_csv("alcohol.csv",index_col=0)
#
# def nan_replace(t):
#     assert isinstance(t,pd.DataFrame)
#     for i in t.columns:
```

```
#         if any(t[i].isna()):
#             t[i].fillna(t[i].mean(),inplace=True)
#         else:
#             t[i].fillna(t[i].mode()[0],inplace=True)
#
# nan_replace(df_acohol)
# print(df_acohol)
# set_date = list(df_acohol.columns[1:])
#
# # Standard
# scalar = StandardScaler()
# date_standardizate = scalar.fit_transform(df_acohol[set_date])
# df_dateStandard = pd.DataFrame(data=date_standardizate,index=df_acohol[set_date].index,columns=df_acohol[set_date].columns)
# print(df_dateStandard)
# # Model Cluster, metoda "WARD"
# # --- construim ierarhia
# x = df_dateStandard[set_date].values
# print(x)
# metoda = 'ward'
# h = hclust.linkage(x,method=metoda)
# print("Type h: ",type(h))
# print("h",h)
#
# # Numar de pasi/jonctiuni, avem nevoie pentru diferenta dintre 2 clusteri
# instante = list(df_acohol.index)
# n = len(instante)
# p = n - 1
# print(p);
# # Distanta maxima intre 2 clusteri
# k_dif_max = np.argmax(h[1:,2] - h[:(p-1),2])
# print("Distanta maxima: ",k_dif_max)
# nr_clusteri = p - k_dif_max
# print("Numarul clusterilor: ",nr_clusteri)
#
# # Trasare plot dendograma (se bazeaza pe partitia optimala + calcul partitie)
# fig = plt.figure("Dendograma",figsize=(9,9))
# plt.subplot(1,1,1)
# plt.title("Dendograma")
# hclust.dendrogram(h,labels=instante)
#
# # Calcul partitie optimala prin metoda Elbow
# distanta = h[:,2]
# diferente = np.diff(distanta,2)
# elbow_pct = np.argmax(diferente) + 1
# partitie_optimala = hclust.fcluster(h,t=distanta[elbow_pct - 1],criterion='distance')
# # print(partitie_optimala)
#
# # Pentru un numar dorit de la tastatura
# # num_cluster = int(input("Introduceti numarul de clusteri dorit: "))
# num_cluster = 4 #test
# custom = hclust.fcluster(h,t=num_cluster,criterion='maxclust')
# # print(custom)
#
# # Calcul indecsi Silhouette
# silhouette_optimal = silhouette_score(x,partitie_optimala)
# #Trasare plot Silhouette
# plt.figure("Plot silhouette",figsize=(9,9))
# plt.subplot(1,1,1)
# plt.scatter(range(len(x)), silhouette_samples(x,partitie_optimala), c=partitie_optimala, cmap='viridis')
# plt.title("Silhouette pentru partitie optimala")
# plt.show()
#
# #Trasare plot partitie in axe principale
# modelACP = PCA(2)
# z = modelACP.fit_transform(x)
#
# plt.figure("Plot partitie in axe principale",figsize=(9,9))
# plt.subplot(1,1,1)
# sb.scatterplot(x=z[:,0], y=z[:,1],hue=partitie_optimala,hue_order=np.unique(partitie_optimala))
# plt.show()

# ------------- Analiza discriminanta
# df_hernia = pd.read_csv("hernia.csv",index_col=0)
# print(df_hernia)
#
# def nan_replace(t):
#     assert isinstance(t,pd.DataFrame)
#     for i in t.columns:
#         if any(t[i].isna()):
#             t[i].fillna(t[i].mean(),inplace=True)
#         else:
```

```
#           t[i].fillna(t[i].mode()[0],inplace=True)
#
# nan_replace(df_hernia)
#
# variabile_numerice = list(df_hernia.columns[:-1])
# print(variabile_numerice)
#
# standard = StandardScaler()
# date_standardizate = standard.fit_transform(df_hernia[variabile_numerice])
# print(date_standardizate)
#
# # PAS 1 : Antrenare model
# variabile = list(df_hernia)
# predictori = variabile[:-1]
# tinta = variabile[len(predictori)]
#
# X_train,X_test,Y_train,Y_test = train_test_split(
#     df_hernia[predictori],
#     df_hernia[tinta],
#     test_size=0.4
# )
# # PAS 2 : Implementare Model
# model_LDA = LinearDiscriminantAnalysis()
# model_LDA.fit(X_train,Y_train)
#
# # Calcul pe axe discriminante
# clase = model_LDA.classes_
# q = len(clase)
# m = q - 1
# print("m= ",m)
#
# # Calculam scorurile discriminante ->LDA(z)
# z = model_LDA.transform(X_test)
# etichete_z = ["z"+ str(i+1) for i in range(m)]
# df_z = pd.DataFrame(data=z,index=X_test.index, columns=etichete_z)
# print(df_z)
#
# #Predictie in set de testare
# predictie_lda_test = model_LDA.predict(X_test)
# # print(predictie_lda_test)
#
# # Plot distributii in axa discriminanta
# def plot_distributie(z,y,k=0):
#     fig = plt.figure("Plot Distributie",figsize=(9,9))
#     ax = fig.add_subplot(1,1,1)
#     ax.set_title("Distributie in axa discriminanta "+ str(k+1))
#     sb.kdeplot(x=z[:,k], hue=y, fill=True)
# for i in range(m):
#     plot_distributie(z,Y_test,i)
#
# matrice_confuzie = confusion_matrix(Y_test,predictie_lda_test)
# print(matrice_confuzie)
# acuratetea = accuracy_score(Y_test,predictie_lda_test)
# print(acuratetea)

# ---------------- Analiza Canonica
# import numpy as np
# import pandas as pd
# from sklearn.cross_decomposition import CCA
# from utils import *
#
# def execute():
#     t1 = pd.read_csv("C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\mortalitate.csv", index_col=1)
#     variabile1 = list(t1.columns)[1:]
#
#     t2 = pd.read_csv("C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\Teritorial.csv",
#                      index_col=1)
#     variabile2 = list(t2.columns)[3:]
#
#     nan_replace(t1)
#     nan_replace(t2)
#
#     t = t1[variabile1].merge(t2[variabile2], left_index=True, right_index=True)
#     instante = list(t.index)
#
#     x = t[variabile1].values
#     y = t[variabile2].values
#
#     n = len(t)
#     p = len(variabile1)
#     q = len(variabile2)
#     m = min(p, q) # m = nr radacini posibile (nr coloane)
#
#     et_z = ["z" + str(i+1) for i in range(m)]
#     et_u = ["u" + str(i+1) for i in range(m)]
#     etichete_radacini = ["rad" + str(i+1) for i in range(m)]
#
#     # construire model CCA
#     model_cca = CCA(n_components=m)
#     model_cca.fit(x, y)
#
#     # Calcul variabile/scoruri/radacini canonice : z, u
#     z, u = model_cca.transform(x, y)
#     tabel_z = tabelare_matrice(z, instante, et_z,
#                          "C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\XScores.csv")
#
#     tabel_u = tabelare_matrice(u, instante, et_u,
#                          "C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\YScores.csv")
#
#     # Calcul corelatii dintre radacini canonice
#     r = np.diag(np.corrcoef(z, u, rowvar=False)[:m, m:])
#
#     # Determinare relevanta radacini oarecare - test Bartlet
#     r2 = r*r
#     p_values = test_bartlett(r2, n, p, q, m)
#     print("p_values test_bartlett ", p_values)
#
#     tabel_radacini = pd.DataFrame(
#         data={
#             'R': np.round(r, 3),
#             'R2': np.round(r2, 3),
#             "p_values": np.round(p_values, 5)
#         },
#         index=etichete_radacini
#     )
#
#     tabel_radacini.to_csv("C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\radacini.csv")
#
#     criteriu_radacini = np.where(p_values > 0.05)
#     print("Criteriu alegere: ", criteriu_radacini)
#
#     nr_rad_semnificative = criteriu_radacini[0][0]
#     nr_rad_semnificative = nr_rad_semnificative + 1 if nr_rad_semnificative == 1 else nr_rad_semnificative
#     print("Nr radacini semnificative: ", nr_rad_semnificative)
#
#     # Calcul corelatii var observate - var canonice
#     r_xz = np.corrcoef(x, z, rowvar=False)[:p, p:]
#     tabel_r_xz = tabelare_matrice(r_xz, variabile1, et_z,
"C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\r_xz.csv")
#
#     r_yu = np.corrcoef(y, u, rowvar=False)[:q, q:]
#     tabel_r_yu = tabelare_matrice(r_yu, variabile2, et_u,
"C:\\Users\\TonyMontana\\Desktop\\AN3SEM1\\ANALIZA_DATELOR\\seminar\\Seminar13\\r_yu.csv")
#
#     # Corelograma corelatii var obs - var canonice
#     corelograma(tabel_r_xz, titlu="Corelatii X-Z")
#     corelograma(tabel_r_yu, titlu="Corelatii Y-U")
#
#     # Plot corelatii var obs - var canonice (cerc corelatii)
#     for i in range(1, nr_rad_semnificative):
#         plot_corelatii(tabel_r_xz, "z1", et_z[i], tabel_r_yu, "u1", et_u[i])
#
#     # Calcul varianta comuna/explicata si redundanta informationala
#     vx = np.sum(r_xz[:, :nr_rad_semnificative] * r_xz[:, :nr_rad_semnificative], axis=0)
#     vy = np.sum(r_yu[:, :nr_rad_semnificative] * r_yu[:, :nr_rad_semnificative], axis=0)
#     sx = vx * r2[:nr_rad_semnificative]
#     sy = vy * r2[:nr_rad_semnificative]
#
#     # Tabelare varianta/redundanta
#     tabel_varianta_red = pd.DataFrame(data={
#         'vx': vx, 'vy': vy, 'vx(%)': vx * 100 / p, 'vy(%)': vy * 100 / q,
#         'sx': sx, 'sy': sy, 'sx(%)': sx * 100 / p, 'sy(%)': sy * 100 / q
#     }, index=etichete_radacini[:nr_rad_semnificative])
#     tabel_varianta_red.to_csv("var_red.csv")
#
#     show()
#
# if __name__== "__main__":
#     execute()
#
#
#
```

```
#
#
####################################################################################################
# ######################
#
####################################################################################################
#
# import numpy as np
# import pandas as pd
# import matplotlib.pyplot as plt
# import scipy.stats as stats
# import seaborn as sb
#
# def nan_replace(t):
#     assert isinstance(t, pd.DataFrame)
#     nume_variabile = list(t.columns)
#
#     for each in nume_variabile:
#         if any(t[each].isna()):
#             if pd.api.types.is_numeric_dtype(t[each]):
#                 t[each].fillna(t[each].mean(), inplace=True)
#             else:
#                 modul = t[each].mode()[0]
#                 t[each].fillna(modul, inplace=True)
#
# def tabelare_matrice(x, nume_linii=None, nume_coloane=None, out=None):
#     t = pd.DataFrame(x, nume_linii, nume_coloane)
#     if out is not None:
#         t.to_csv(out)
#
#     return t
#
# def corelograma(x, valMin=-1, valMax=1, titlu="Corelatii factoriale"):
#     fig = plt.figure(titlu, figsize=(9,9))
#     ax = fig.add_subplot(1, 1, 1)
#
#     ax.set_title(titlu, fontdict={"fontsize":16, "color": "b"})
#
#     ax_ = sb.heatmap(data=x,
#                  vmin=valMin, vmax=valMax,
#                  cmap="bwr", annot=True, ax=ax)
#     ax_.set_xticklabels(x.columns, ha="right", rotation=30)
#
# def plot_corelatii(t_z, var_z1, var_z2, t_u, var_u1, var_u2, titlu="Plot corelatii", aspect="auto"):
#     fig = plt.figure(figsize=(9, 9))
#     ax = fig.add_subplot(1, 1, 1)
#
#     ax.set_title(titlu, fontdict={"fontsize": 16, "color": "b"})
#     ax.set_xlabel(var_z1+"/"+var_z2, fontdict={"fontsize": 12, "color":"b"})
#     ax.set_ylabel(var_u1+"/"+var_u2, fontdict={"fontsize": 12, "color":"b"})
#
#     ax.set_aspect(aspect)
#
#     theta = np.arange(0, 2*np.pi, 0.01) # arange da elementele spatiate egal intre ele
#     ax.plot(np.cos(theta), np.sin(theta), color="b")
#
#     ax.axhline(0)
#     ax.axvline(0)
#
#     ax.scatter(t_z[var_z1], t_z[var_z2], color="r", label="Spatiul X")
#     ax.scatter(t_u[var_u1], t_u[var_u2], color="b", label="Spatiul Y")
#
#     for i in range(len(t_z)):
#         ax.text(t_z[var_z1].iloc[i], t_z[var_z2].iloc[i], t_z.index[i])
#     for i in range(len(t_u)):
#         ax.text(t_u[var_u1].iloc[i], t_u[var_u2].iloc[i], t_u.index[i])
#
#     ax.legend()
#
#
# def test_bartlett(r2, n, p, q, m):
#     v = 1 - r2
#     chi2_ = (-n + 1 + (p + q + 1)/2) * np.log(np.flip(np.cumprod(np.flip(v))))
#     nlib = [(p - k + 1) * (q - k + 1) for k in range(1, m + 1)]
#
#     p_values = 1 - stats.chi2.cdf(chi2_, nlib)
#     return p_values
#
# def show():
#     plt.show()
```