Наборы данных с известными метками классов (кластеров) объектов.

import pandas as pd

crx\_data = pd.read\_csv("crx.data", header=None)

print(crx\_data.info())

print(crx\_data.head())

**<class 'pandas.core.frame.DataFrame'>**

**RangeIndex: 690 entries, 0 to 689**

**Data columns (total 16 columns):**

**# Column Non-Null Count Dtype**

**--- ------ -------------- -----**

**0 0 690 non-null object**

**1 1 690 non-null object**

**2 2 690 non-null float64**

**3 3 690 non-null object**

**4 4 690 non-null object**

**5 5 690 non-null object**

**6 6 690 non-null object**

**7 7 690 non-null float64**

**8 8 690 non-null object**

**9 9 690 non-null object**

**10 10 690 non-null int64**

**11 11 690 non-null object**

**12 12 690 non-null object**

**13 13 690 non-null object**

**14 14 690 non-null int64**

**15 15 690 non-null object**

**dtypes: float64(2), int64(2), object(12)**

**memory usage: 86.4+ KB**

**None**

**0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15**

**...**

**1 a 58.67 4.460 u g q h 3.04 t t 6 f g 00043 560 +**

**2 a 24.50 0.500 u g q h 1.50 t f 0 f g 00280 824 +**

**3 b 27.83 1.540 u g w v 3.75 t t 5 t g 00100 3 +**

**4 b 20.17 5.625 u g w v 1.71 t f 0 f s 00120 0 + cluster\_agg cluster\_kmeans cluster\_fuzzy\_cmeans cluster\_dbscan**

**0 1 1 1 -1**

**1 0 0 1 -1**

**2 1 1 1 -1**

**3 0 0 1 -1**

**4 0 1 1 -1**

print(crx\_data.describe())

**2 7 10 14**

**count 690.000000 690.000000 690.00000 690.000000**

**mean 4.758725 2.223406 2.40000 1017.385507**

**std 4.978163 3.346513 4.86294 5210.102598**

**min 0.000000 0.000000 0.00000 0.000000**

**25% 1.000000 0.165000 0.00000 0.000000**

**50% 2.750000 1.000000 0.00000 5.000000**

**75% 7.207500 2.625000 3.00000 395.500000**

**max 28.000000 28.500000 67.00000 100000.000000**

corr\_matrix = crx\_data\_encoded.corr()

print(corr\_matrix)

import numpy as np

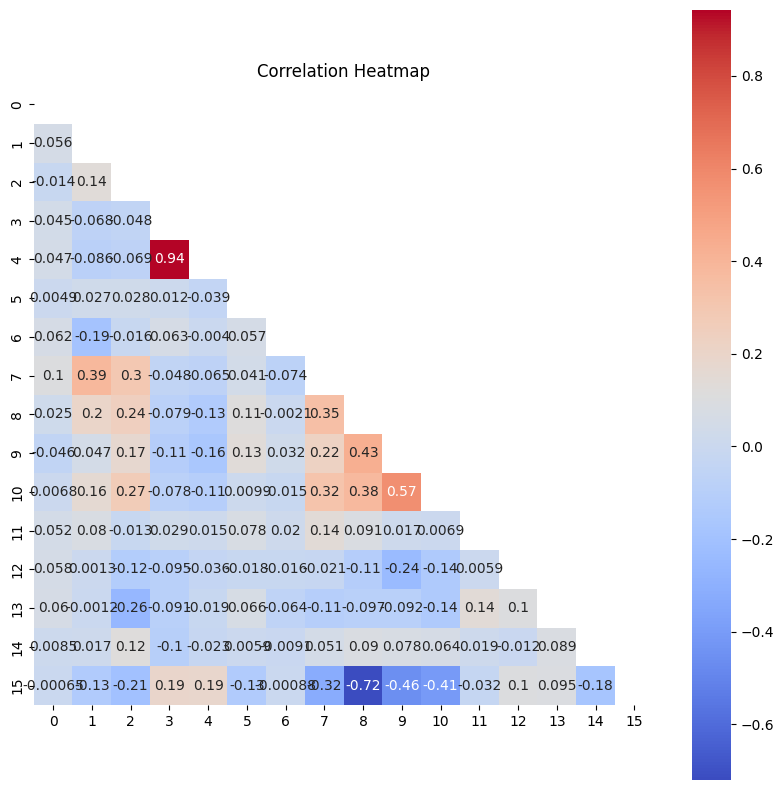
mask = np.triu(np.ones\_like(corr\_matrix, dtype=bool))

plt.figure(figsize=(10, 10))

sns.heatmap(corr\_matrix, mask=mask, annot=True, cmap="coolwarm", square=True)

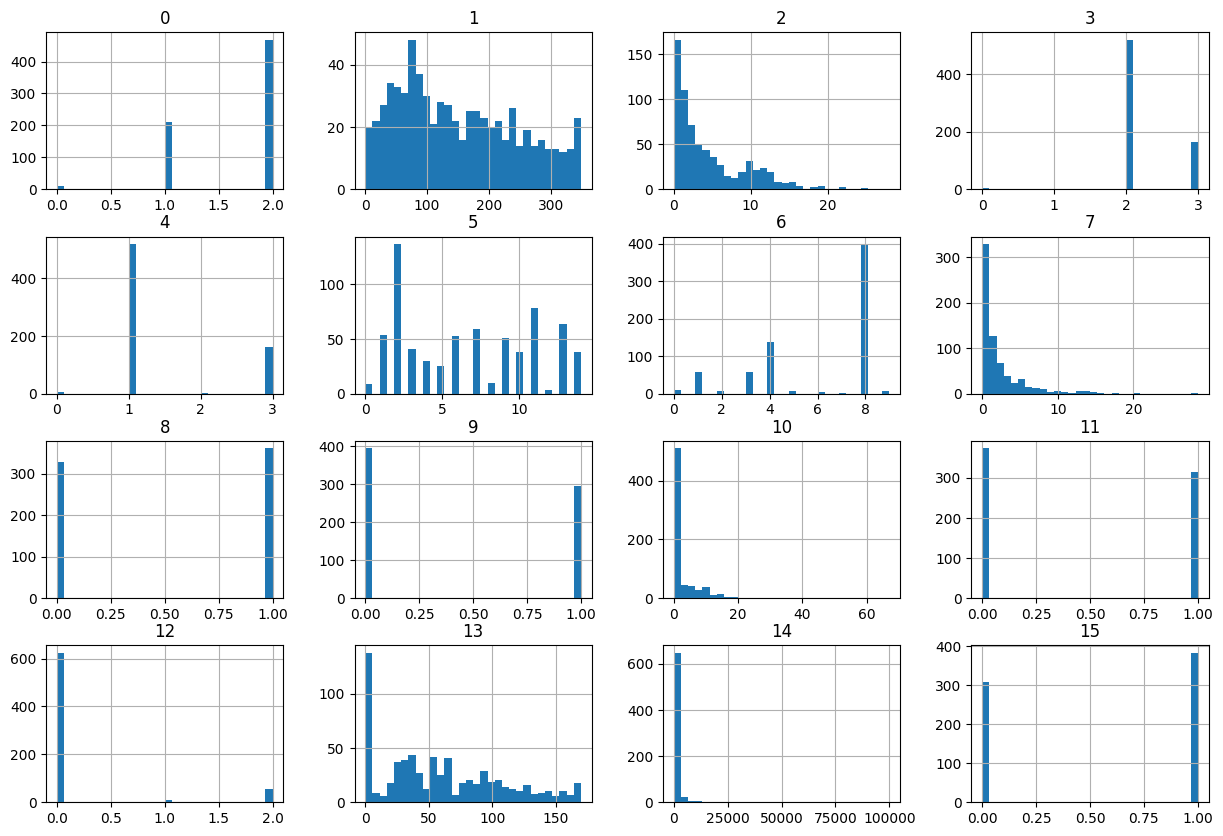
plt.title("Correlation Heatmap")

plt.show()

****

crx\_data\_encoded.hist(bins=30, figsize=(15, 10))

plt.show()



import pandas as pd

import numpy as np

from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN

from sklearn.preprocessing import LabelEncoder, StandardScaler

import skfuzzy as fuzz

crx\_data = pd.read\_csv("crx.data")

le = LabelEncoder()

for column in crx\_data.columns:

    if crx\_data[column].dtype == type(object):

        crx\_data[column] = le.fit\_transform(crx\_data[column])

crx\_data.replace([np.inf, -np.inf], np.nan, inplace=True)

crx\_data.fillna(crx\_data.mean(), inplace=True)

scaler = StandardScaler()

crx\_data\_scaled = pd.DataFrame(scaler.fit\_transform(crx\_data), columns=crx\_data.columns)

agg\_clustering = AgglomerativeClustering(n\_clusters=3)

crx\_data["cluster\_agg"] = agg\_clustering.fit\_predict(crx\_data\_scaled)

kmeans = KMeans(n\_clusters=3)

crx\_data["cluster\_kmeans"] = kmeans.fit\_predict(crx\_data\_scaled)

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

    crx\_data\_scaled.T, 3, 2, error=0.005, maxiter=1000, init=None)

crx\_data["cluster\_fuzzy\_cmeans"] = np.argmax(u, axis=0)

dbscan = DBSCAN(eps=0.3, min\_samples=5)

crx\_data["cluster\_dbscan"] = dbscan.fit\_predict(crx\_data\_scaled)

print(crx\_data.head())

b 30.83 0 u g w v 1.25 t t.1 01 f g.1 00202 0.1 + \

0 1 327 4.460 2 1 11 4 3.04 1 1 6 0 0 11 560 0

1 1 89 0.500 2 1 11 4 1.50 1 0 0 0 0 95 824 0

2 2 125 1.540 2 1 13 8 3.75 1 1 5 1 0 31 3 0

3 2 43 5.625 2 1 13 8 1.71 1 0 0 0 2 37 0 0

4 2 167 4.000 2 1 10 8 2.50 1 0 0 1 0 114 0 0

cluster\_agg cluster\_kmeans cluster\_fuzzy\_cmeans cluster\_dbscan

0 1 1 1 -1

1 0 0 1 -1

2 1 1 1 -1

3 0 0 1 -1

4 0 1 1 -1

from sklearn.metrics import silhouette\_score

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn.preprocessing import StandardScaler

import skfuzzy as fuzz

import numpy as np

import matplotlib.pyplot as plt

# Chuẩn hóa dữ liệu

scaler = StandardScaler()

crx\_data\_scaled = pd.DataFrame(scaler.fit\_transform(crx\_data), columns=crx\_data.columns)

methods = {

    "KMeans": KMeans,

    "Agglomerative Clustering": AgglomerativeClustering,

    # Fuzzy C-means và DBSCAN cần một cách tiếp cận khác nhau

}

for method\_name, method in methods.items():

    silhouette\_scores = []

    for k in range(2, 11):

        clusterer = method(n\_clusters=k)

        labels = clusterer.fit\_predict(crx\_data\_scaled)

        silhouette\_scores.append(silhouette\_score(crx\_data\_scaled, labels))

    optimal\_k = silhouette\_scores.index(max(silhouette\_scores)) + 2

    print(f"Số lượng cụm tối ưu theo điểm số Silhouette cho {method\_name}: {optimal\_k}")

    # Vẽ biểu đồ Silhouette scores

    plt.figure(figsize=(10, 5))

    plt.plot(range(2, 11), silhouette\_scores, "bx-")

    plt.title(f"Silhouette scores for {method\_name}")

    plt.xlabel("Number of clusters")

    plt.ylabel("Silhouette score")

    plt.show()

    wcss = []

# Tính WCSS cho KMeans với số lượng cụm từ 2 đến 10

for k in range(2, 11):

    kmeans = KMeans(n\_clusters=k).fit(crx\_data\_scaled)

    wcss.append(kmeans.inertia\_)

# Vẽ biểu đồ Elbow

plt.figure(figsize=(10, 5))

plt.plot(range(2, 11), wcss, "bx-")

plt.title("Biểu đồ Elbow")

plt.xlabel("Số lượng cụm")

plt.ylabel("WCSS")

plt.show()

# Danhsách để lưu trữ giá trị hàm mục tiêu

fcm\_obj\_values = []

# Tính giá trị hàm mục tiêu cho Fuzzy C-means với số lượng cụm từ 2 đến 10

for k in range(2, 11):

    fcm = fuzz.cluster.cmeans(crx\_data\_scaled.T, k, 2, error=0.005, maxiter=1000)

    fcm\_obj\_values.append(fcm[1][-1])  # chỉ lấy giá trị cuối cùng

plt.figure(figsize=(10, 5))

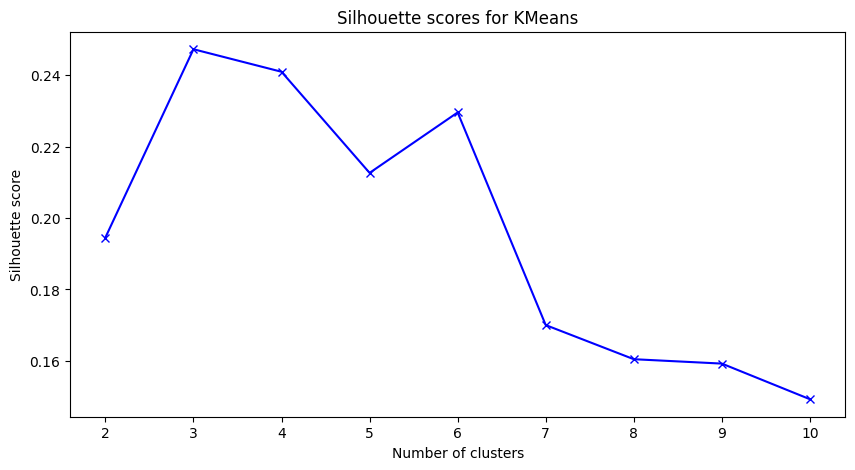
plt.plot(range(2, 11), fcm\_obj\_values, "bx-")

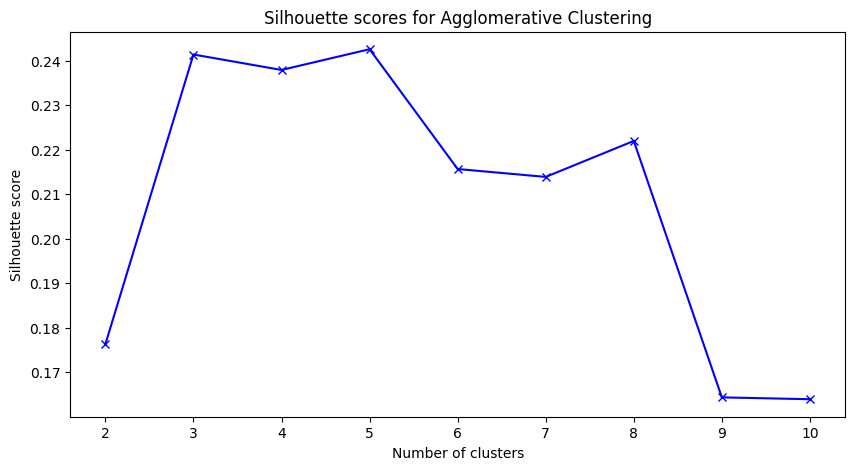
plt.title("Biểu đồ giá trị hàm mục tiêu Fuzzy C-means")

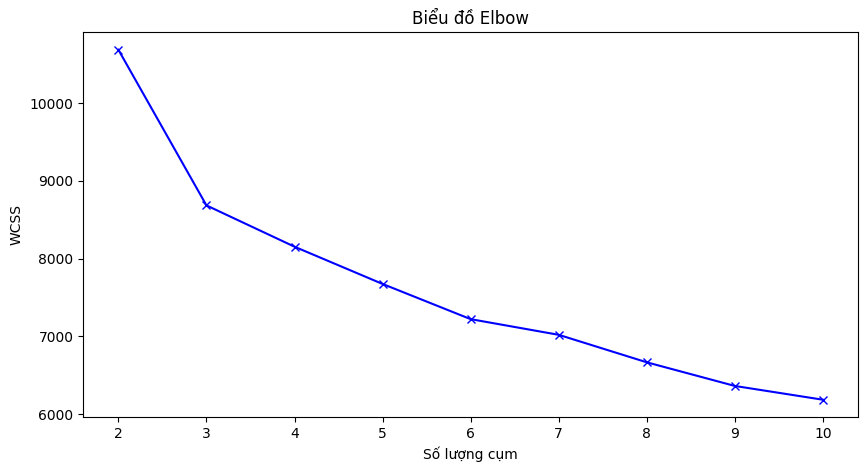
plt.xlabel("Số lượng cụm")

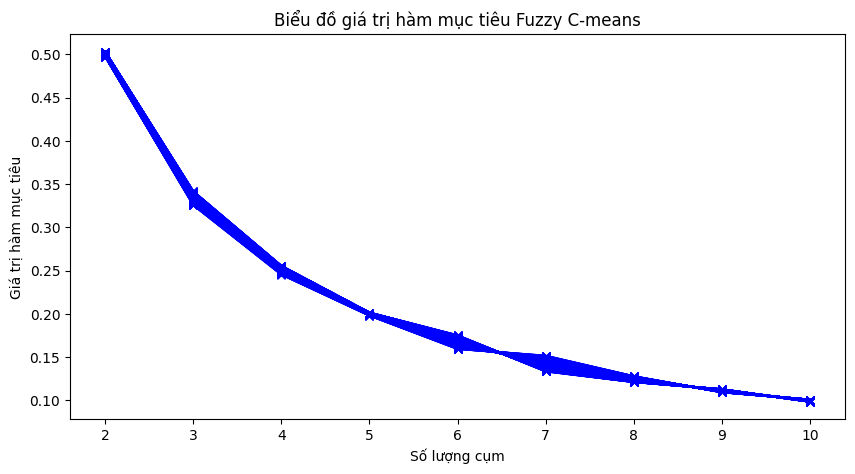
plt.ylabel("Giá trị hàm mục tiêu")

plt.show()









На локтевой диаграмме сложно определить оптимальное количество кластеров, поэтому мы используем индекс Эльхуэте и определяем оптимальное количество кластеров, равное 3.

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.metrics import (

    adjusted\_rand\_score,

    normalized\_mutual\_info\_score,

    fowlkes\_mallows\_score,

)

import skfuzzy as fuzz

import matplotlib.pyplot as plt

# Đọc dữ liệu

crx\_data = pd.read\_csv("crx.data", header=None)

# Lấy nhãn thực sự và chuyển đổi thành dạng số

true\_labels = crx\_data.iloc[:, -1].values

true\_labels = [1 if label == "+" else 0 for label in true\_labels]

# Loại bỏ cột nhãn khỏi dữ liệu

crx\_data = crx\_data.drop(crx\_data.columns[-1], axis=1)

# Mã hóa tất cả các cột dạng chuỗi

le = LabelEncoder()

for column in crx\_data.columns:

    if crx\_data[column].dtype == type(object):

        crx\_data[column] = le.fit\_transform(crx\_data[column])

# Xử lý giá trị vô cùng

crx\_data.replace([np.inf, -np.inf], np.nan, inplace=True)

crx\_data.fillna(crx\_data.mean(), inplace=True)

# Chuẩn hóa dữ liệu

scaler = StandardScaler()

crx\_data\_scaled = pd.DataFrame(scaler.fit\_transform(crx\_data), columns=crx\_data.columns)

optimal\_k = 3

# Phân cụm dữ liệu bằng KMeans với số lượng cụm tối ưu

kmeans = KMeans(n\_clusters=optimal\_k)

kmeans.fit(crx\_data\_scaled)

predicted\_labels = kmeans.labels\_

# Tính và in ra ARI, NMI, FMI cho KMeans

ari = adjusted\_rand\_score(true\_labels, predicted\_labels)

nmi = normalized\_mutual\_info\_score(true\_labels, predicted\_labels)

fmi = fowlkes\_mallows\_score(true\_labels, predicted\_labels)

print("Adjusted Rand Index for KMeans: ", ari)

print("Normalized Mutual Information for KMeans: ", nmi)

print("Fowlkes-Mallows Index for KMeans: ", fmi)

# Phân cụm dữ liệu bằng Agglomerative Clustering

agg\_clustering = AgglomerativeClustering(n\_clusters=optimal\_k)

agg\_labels = agg\_clustering.fit\_predict(crx\_data\_scaled)

# Tính và in ra ARI, NMI, FMI cho Agglomerative Clustering

ari\_agg = adjusted\_rand\_score(true\_labels, agg\_labels)

nmi\_agg = normalized\_mutual\_info\_score(true\_labels, agg\_labels)

fmi\_agg = fowlkes\_mallows\_score(true\_labels, agg\_labels)

print("Adjusted Rand Index for Agglomerative Clustering: ", ari\_agg)

print("Normalized Mutual Information for Agglomerative Clustering: ", fmi\_agg)

# Phân cụm dữ liệu bằng Fuzzy C-means

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

    crx\_data\_scaled.T, optimal\_k, 2, error=0.005, maxiter=1000, init=None

)

fuzzy\_labels = np.argmax(u, axis=0)

# Tính và in ra ARI, NMI, FMI cho Fuzzy C-means

ari\_fuzzy = adjusted\_rand\_score(true\_labels, fuzzy\_labels)

nmi\_fuzzy = normalized\_mutual\_info\_score(true\_labels, fuzzy\_labels)

fmi\_fuzzy = fowlkes\_mallows\_score(true\_labels, fuzzy\_labels)

print("Adjusted Rand Index for Fuzzy C-means: ", ari\_fuzzy)

print("Normalized Mutual Information for Fuzzy C-means: ", nmi\_fuzzy)

print("Fowlkes-Mallows Index for Fuzzy C-means: ", fmi\_fuzzy)

# Phân cụm dữ liệu bằng DBSCAN

dbscan = DBSCAN(eps=0.2, min\_samples=6)

dbscan\_labels = dbscan.fit\_predict(crx\_data\_scaled)

# Tính và in ra ARI, NMI, FMI cho DBSCAN

ari\_dbscan = adjusted\_rand\_score(true\_labels, dbscan\_labels)

nmi\_dbscan = normalized\_mutual\_info\_score(true\_labels, dbscan\_labels)

fmi\_dbscan = fowlkes\_mallows\_score(true\_labels, dbscan\_labels)

print("Adjusted Rand Index for DBSCAN: ", ari\_dbscan)

print("Normalized Mutual Information for DBSCAN: ", nmi\_dbscan)

print("Fowlkes-Mallows Index for DBSCAN: ", fmi\_dbscan)

Adjusted Rand Index for KMeans: 0.26475590178864183

Normalized Mutual Information for KMeans: 0.23942911288858024

Fowlkes-Mallows Index for KMeans: 0.5812483514920085

Adjusted Rand Index for Agglomerative Clustering: 0.15924398415311974

Normalized Mutual Information for Agglomerative Clustering: 0.5178328265842079

Adjusted Rand Index for Fuzzy C-means: 0.36747570693742043

Normalized Mutual Information for Fuzzy C-means: 0.2744388445778863

Fowlkes-Mallows Index for Fuzzy C-means: 0.6750737600247656

Adjusted Rand Index for DBSCAN: 0.0

Normalized Mutual Information for DBSCAN: 0.0

Fowlkes-Mallows Index for DBSCAN: 0.7108790805200357

Основываясь на рассчитанных индексах, мы можем оценить эффективность алгоритмов кластеризации следующим образом:

1. \*\*KMeans\*\*: Имеет относительно низкие значения ARI, NMI и FMI, что указывает на не очень высокую эффективность кластеризации.

2. \*\*Agglomerative Clustering\*\*: Имеет низкий ARI, но NMI выше, чем у KMeans, что указывает на то, что он может лучше кластеризовать на основе взаимной информации между кластерами.

3. \*\*Fuzzy C-means\*\*: Имеет наивысшие значения ARI и FMI среди всех алгоритмов, что указывает на наилучшую эффективность кластеризации.

4. \*\*DBSCAN\*\*: Имеет ARI и NMI равные 0, что указывает на очень низкую эффективность кластеризации. Однако высокий FMI указывает на то, что сходство между кластерами и истинными метками довольно хорошее.

На основе этих индексов, \*Fuzzy C-means\* кажется наилучшим алгоритмом кластеризации для этого набора данных, так как он имеет наивысшие значения ARI И FMI.

from sklearn.cluster import KMeans

from sklearn.manifold import TSNE

import umap.umap\_ as umap

import matplotlib.pyplot as plt

optimal\_k = 3

kmeans = KMeans(n\_clusters=optimal\_k)

kmeans.fit(crx\_data\_scaled)

predicted\_labels = kmeans.labels\_

predicted\_labels\_list = [predicted\_labels, agg\_labels, fuzzy\_labels, dbscan\_labels]

titles = ["KMeans", "Agglomerative Clustering", "Fuzzy C-means", "DBSCAN"]

kmeans\_centers = kmeans.cluster\_centers\_

fuzzy\_centers = cntr

for perplexity in [5, 30, 50, 100]:

    tsne = TSNE(n\_components=2, perplexity=perplexity)

    crx\_data\_tsne = tsne.fit\_transform(crx\_data\_scaled)

    if kmeans\_centers.shape[0] > perplexity:

        kmeans\_centers\_tsne = tsne.fit\_transform(kmeans\_centers)

    if fuzzy\_centers.shape[0] > perplexity:

        fuzzy\_centers\_tsne = tsne.fit\_transform(fuzzy\_centers)

    plt.figure(figsize=(20, 5))

    for i, predicted\_labels in enumerate(predicted\_labels\_list):

        plt.subplot(1, 4, i + 1)

        plt.scatter(crx\_data\_tsne[:, 0], crx\_data\_tsne[:, 1], c=predicted\_labels)

        if titles[i] == "KMeans" and kmeans\_centers.shape[0] > perplexity:

            plt.scatter(

                kmeans\_centers\_tsne[:, 0],

                kmeans\_centers\_tsne[:, 1],

                c="red",

                marker="x",)

        elif titles[i] == "Fuzzy C-means" and fuzzy\_centers.shape[0] > perplexity:

            plt.scatter(

                fuzzy\_centers\_tsne[:, 0], fuzzy\_centers\_tsne[:, 1], c="blue", marker="x")

        plt.title(

            f"t-SNE visualization of {titles[i]} clusters with perplexity={perplexity}"

        )

    plt.show()

for n\_neighbors in [5, 15, 30, 50]:

    reducer = umap.UMAP(n\_neighbors=n\_neighbors)

    crx\_data\_umap = reducer.fit\_transform(crx\_data\_scaled)

    kmeans\_centers\_umap = reducer.transform(kmeans\_centers)

    fuzzy\_centers\_umap = reducer.transform(fuzzy\_centers)

    plt.figure(figsize=(20, 5))

    for i, predicted\_labels in enumerate(predicted\_labels\_list):

        plt.subplot(1, 4, i + 1)

        plt.scatter(crx\_data\_umap[:, 0], crx\_data\_umap[:, 1], c=predicted\_labels)

        if titles[i] == "KMeans":

            plt.scatter(

                kmeans\_centers\_umap[:, 0],

                kmeans\_centers\_umap[:, 1],

                c="red",

                marker="x",

            )

        elif titles[i] == "Fuzzy C-means":

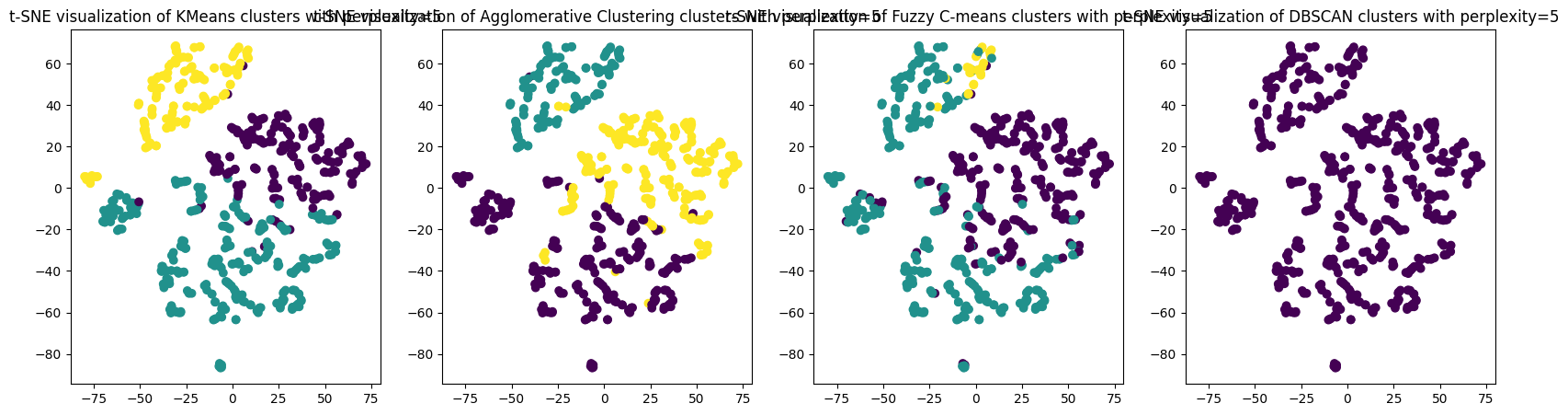
            plt.scatter(

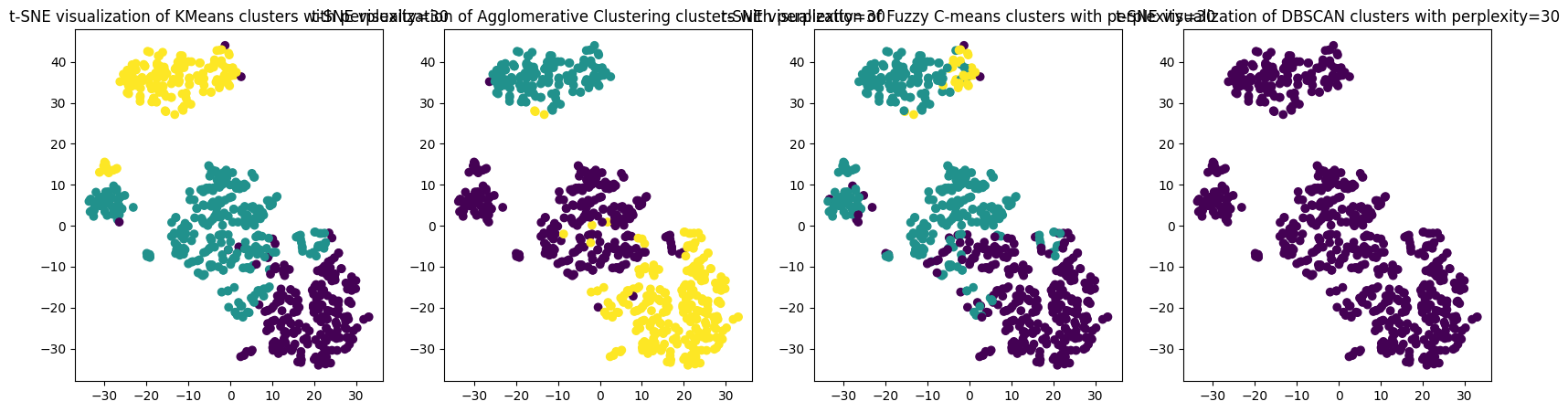
                fuzzy\_centers\_umap[:, 0], fuzzy\_centers\_umap[:, 1], c="blue", marker="x")

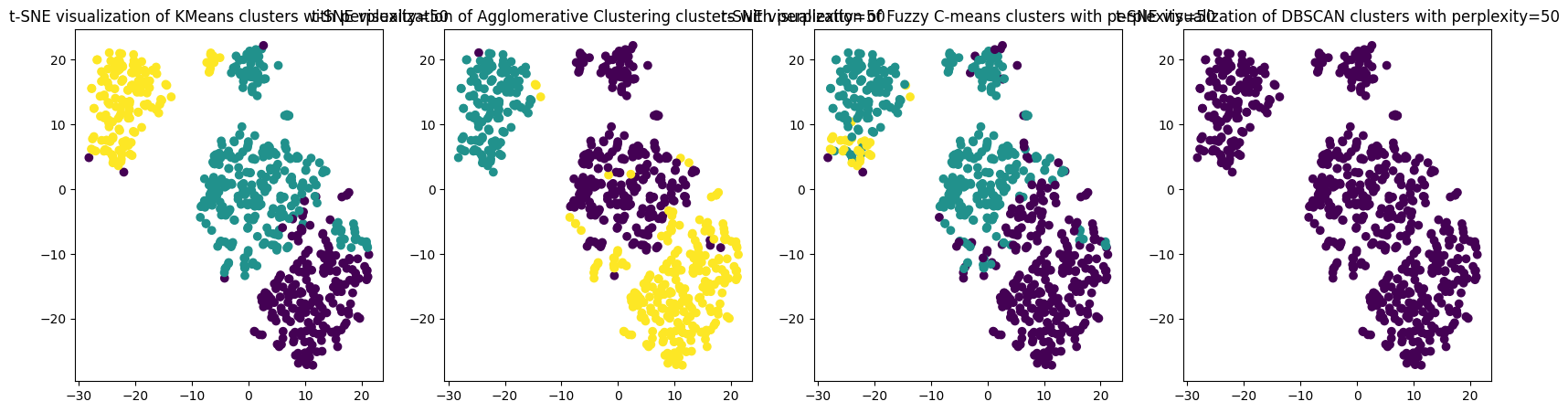
        plt.title(

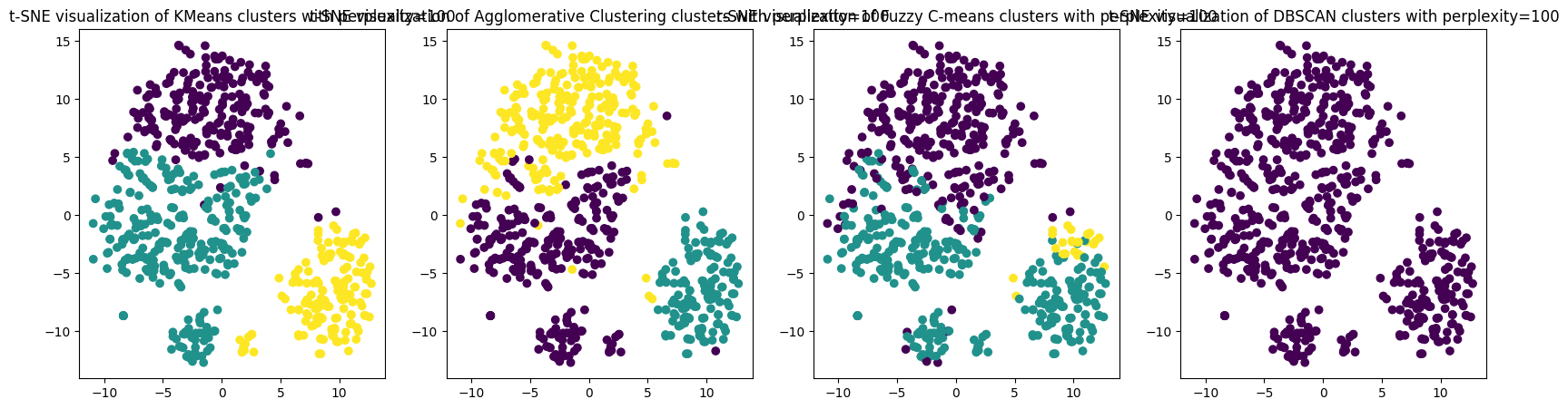
            f"UMAP visualization of {titles[i]} clusters with n\_neighbors={n\_neighbors}")

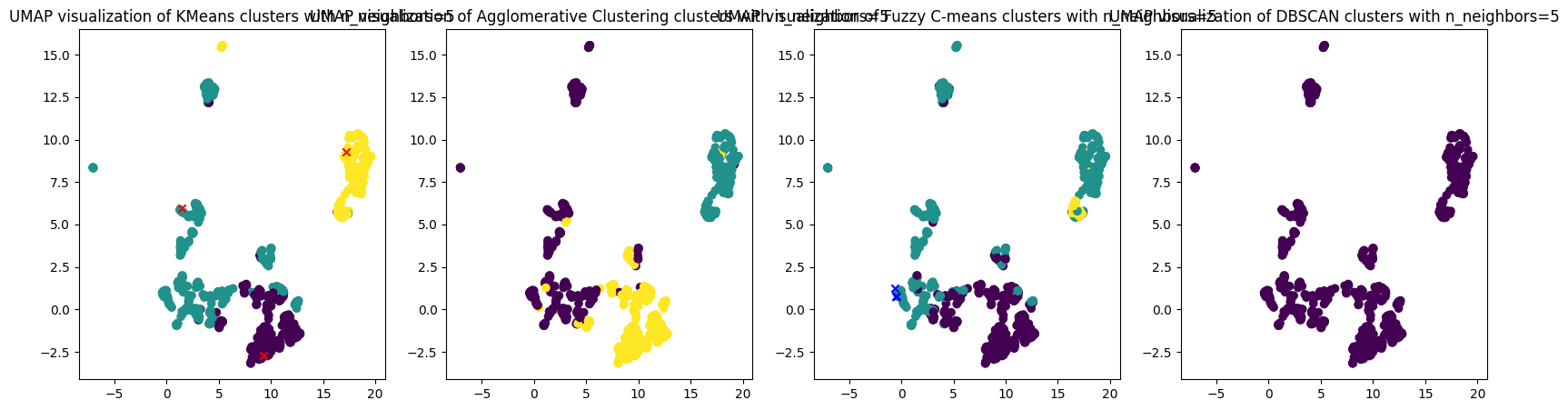
    plt.show()

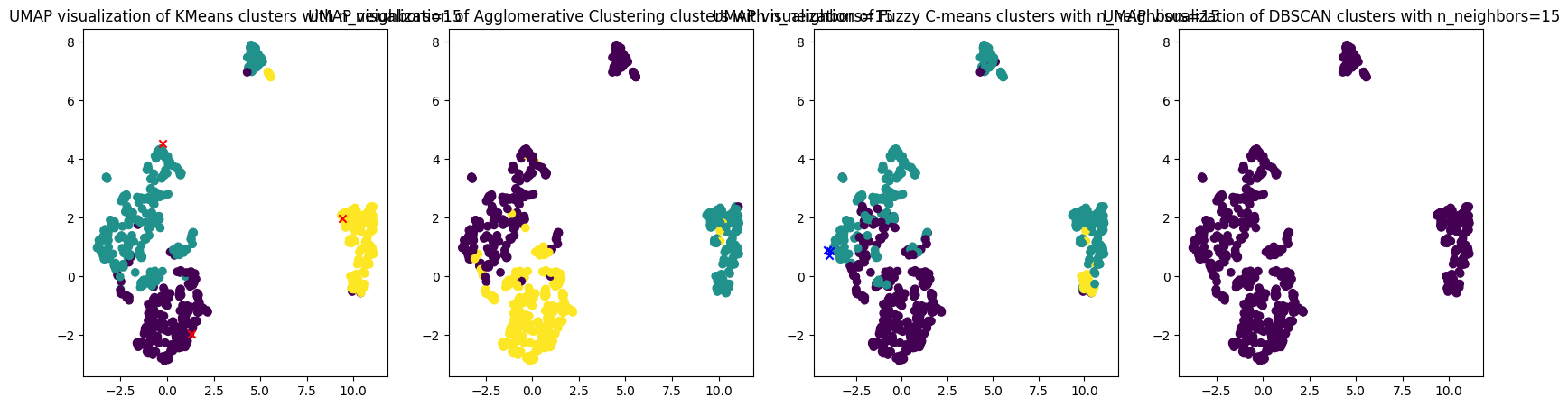


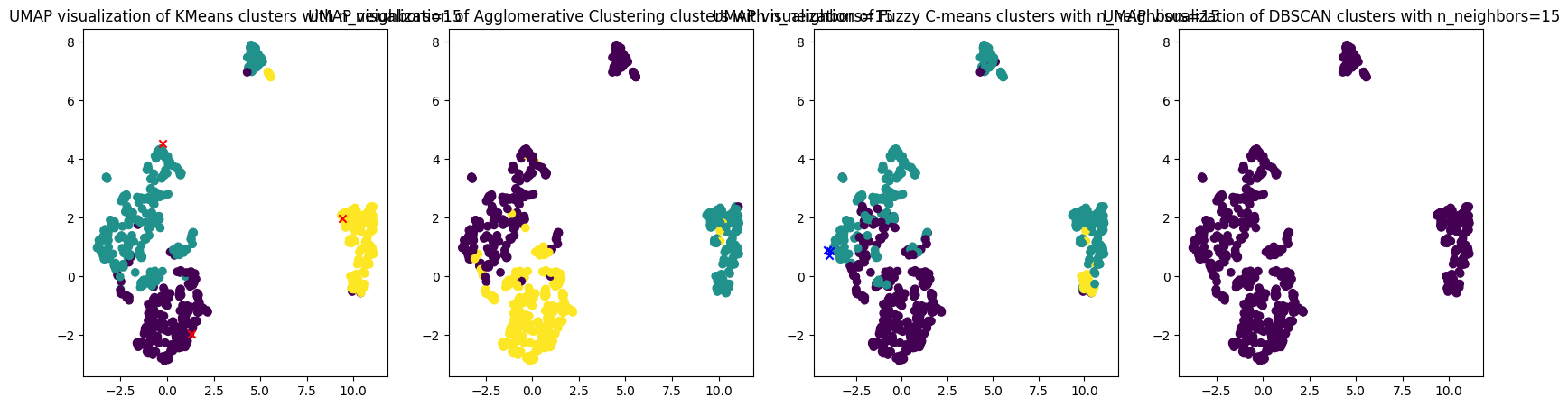


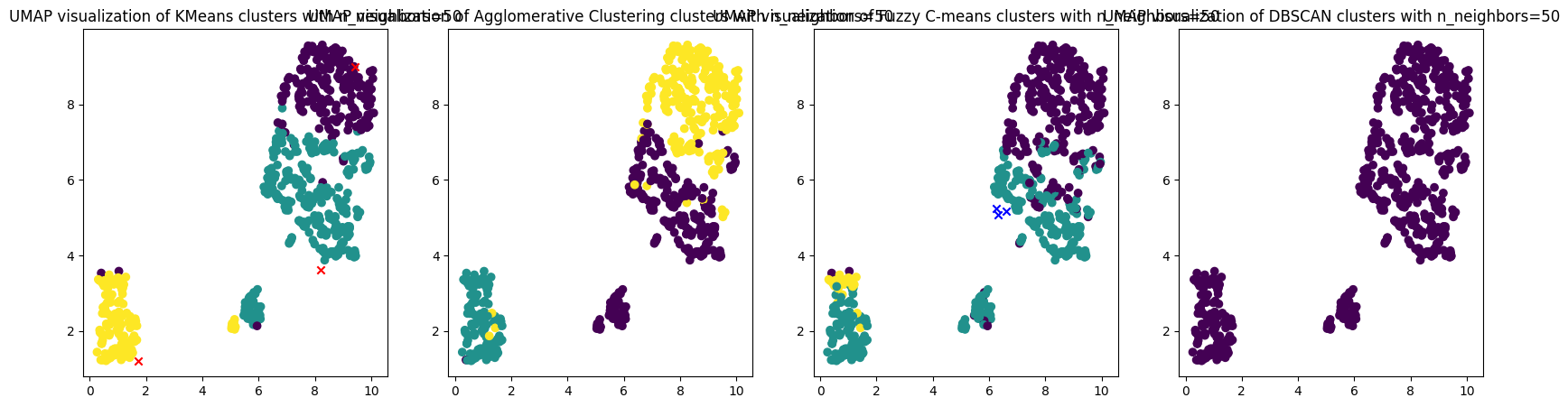












Наборы данных с неизвестными метками классов (кластеров) объектов.

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

df = pd.read\_csv("country\_wise\_latest.csv")

print(df.head())

print(df.describe())

for column in df.select\_dtypes(include=[np.number]).columns:

    plt.figure()

    sns.histplot(df[column])

    plt.title(f"Distribution of {column}")

    plt.show()

for column in df.select\_dtypes(include=[np.number]).columns:

    plt.figure()

    sns.boxplot(x=df[column])

    plt.title(f"Box plot of {column}")

    plt.show()

df\_numeric = df.select\_dtypes(include=[np.number])

corr = df\_numeric.corr()

sns.heatmap(corr, annot=True, cmap="coolwarm")

plt.title("Correlation Matrix")

plt.show()

**Country/Region Confirmed Deaths Recovered Active New cases New deaths \**

**0 Afghanistan 36263 1269 25198 9796 106 10**

**1 Albania 4880 144 2745 1991 117 6**

**2 Algeria 27973 1163 18837 7973 616 8**

**3 Andorra 907 52 803 52 10 0**

**4 Angola 950 41 242 667 18 1**

**New recovered Deaths / 100 Cases Recovered / 100 Cases \**

**0 18 3.50 69.49**

**1 63 2.95 56.25**

**2 749 4.16 67.34**

**3 0 5.73 88.53**

**4 0 4.32 25.47**

**Deaths / 100 Recovered Confirmed last week 1 week change \**

**0 5.04 35526 737**

**1 5.25 4171 709**

**2 6.17 23691 4282**

**3 6.48 884 23**

**4 16.94 749 201**

**1 week % increase WHO Region**

**0 2.07 Eastern Mediterranean**

**1 17.00 Europe**

**2 18.07 Africa**

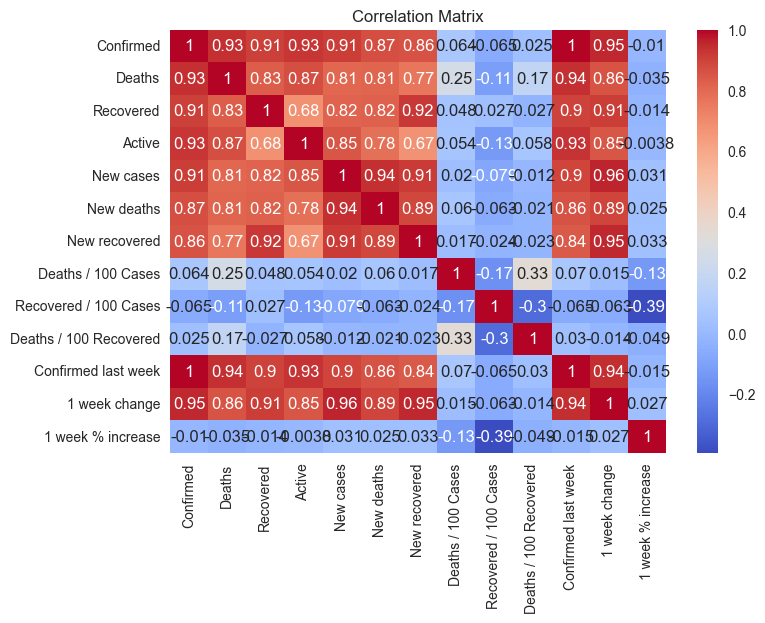
**...**

**25% 2.775000**

**50% 6.890000**

**75% 16.855000**

**max 226.320000**

****

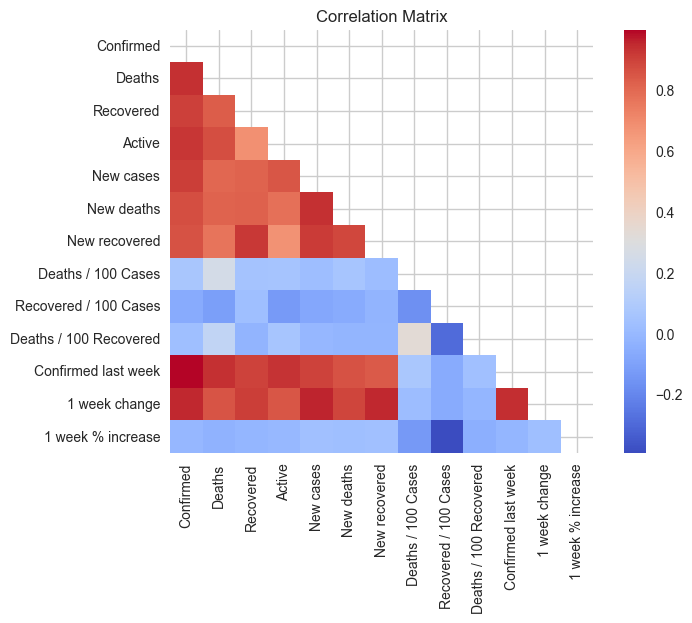
corr = df\_numeric.corr()

mask = np.triu(np.ones\_like(corr, dtype=bool))

sns.heatmap(corr, mask=mask, cmap="coolwarm", annot=False, cbar=True, square=True)

plt.title("Correlation Matrix")

plt.show()

****

import pandas as pd

import numpy as np

from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN

from sklearn.preprocessing import LabelEncoder, StandardScaler

import skfuzzy as fuzz

data = pd.read\_csv("country\_wise\_latest.csv")

le = LabelEncoder()

for column in data.columns:

    if data[column].dtype == type(object):

        data[column] = le.fit\_transform(data[column])

data.replace([np.inf, -np.inf], np.nan, inplace=True)

data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()

data\_scaled = pd.DataFrame(scaler.fit\_transform(data), columns=data.columns)

agg\_clustering = AgglomerativeClustering(n\_clusters=3)

data["cluster\_agg"] = agg\_clustering.fit\_predict(data\_scaled)

kmeans = KMeans(n\_clusters=3)

data["cluster\_kmeans"] = kmeans.fit\_predict(data\_scaled)

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

    data\_scaled.T, 3, 2, error=0.005, maxiter=1000, init=None

)

data["cluster\_fuzzy\_cmeans"] = np.argmax(u, axis=0)

dbscan = DBSCAN(eps=0.5, min\_samples=10)

data["cluster\_dbscan"] = dbscan.fit\_predict(data\_scaled)

print(data.head())

**Country/Region Confirmed Deaths Recovered Active New cases \**

**0 0 36263 1269 25198 9796 106**

**1 1 4880 144 2745 1991 117**

**2 2 27973 1163 18837 7973 616**

**3 3 907 52 803 52 10**

**4 4 950 41 242 667 18**

**New deaths New recovered Deaths / 100 Cases Recovered / 100 Cases \**

**0 10 18 3.50 69.49**

**1 6 63 2.95 56.25**

**2 8 749 4.16 67.34**

**3 0 0 5.73 88.53**

**4 1 0 4.32 25.47**

**Deaths / 100 Recovered Confirmed last week 1 week change \**

**0 5.04 35526 737**

**1 5.25 4171 709**

**2 6.17 23691 4282**

**3 6.48 884 23**

**4 16.94 749 201**

**1 week % increase WHO Region cluster\_agg cluster\_kmeans \**

**0 2.07 2 1 0**

**1 17.00 3 1 0**

**2 18.07 0 1 0**

**...**

**1 0 -1**

**2 0 -1**

**3 1 -1**

**4 0 -1**

from sklearn.metrics import silhouette\_score

from sklearn.cluster import KMeans, AgglomerativeClustering

from sklearn.preprocessing import StandardScaler, LabelEncoder

import skfuzzy as fuzz

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

data = pd.read\_csv("country\_wise\_latest.csv")

true\_labels = data["Country/Region"].values

le = LabelEncoder()

true\_labels = le.fit\_transform(true\_labels)

data = data.drop("Country/Region", axis=1)

for column in data.columns:

    if data[column].dtype == type(object):

        data[column] = le.fit\_transform(data[column])

data.replace([np.inf, -np.inf], np.nan, inplace=True)

data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()

data\_scaled = pd.DataFrame(scaler.fit\_transform(data), columns=data.columns)

methods = {

    "KMeans": KMeans,

    "Agglomerative Clustering": AgglomerativeClustering,

}

for method\_name, method in methods.items():

    silhouette\_scores = []

    for k in range(2, 11):

        clusterer = method(n\_clusters=k)

        labels = clusterer.fit\_predict(data\_scaled)

        silhouette\_scores.append(silhouette\_score(data\_scaled, labels))

    optimal\_k = silhouette\_scores.index(max(silhouette\_scores)) + 2

    print(f"Số lượng cụm tối ưu theo điểm số Silhouette cho {method\_name}: {optimal\_k}")

    plt.figure(figsize=(10, 5))

    plt.plot(range(2, 11), silhouette\_scores, "bx-")

    plt.title(f"Silhouette scores for {method\_name}")

    plt.xlabel("Number of clusters")

    plt.ylabel("Silhouette score")

    plt.show()

wcss = []

for k in range(2, 11):

    kmeans = KMeans(n\_clusters=k).fit(data\_scaled)

    wcss.append(kmeans.inertia\_)

plt.figure(figsize=(10, 5))

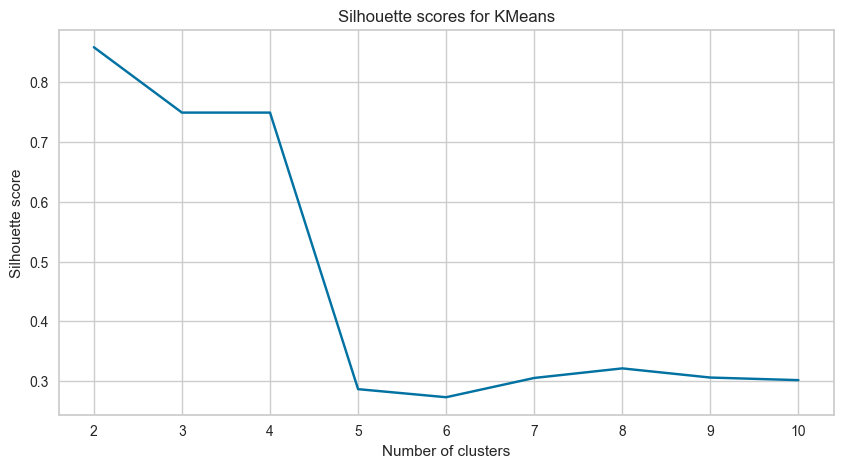
plt.plot(range(2, 11), wcss, "bx-")

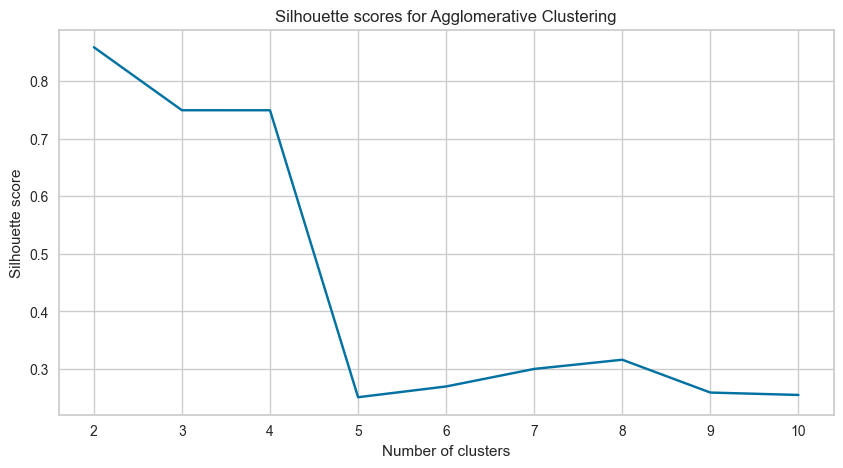
plt.title("Biểu đồ Elbow")

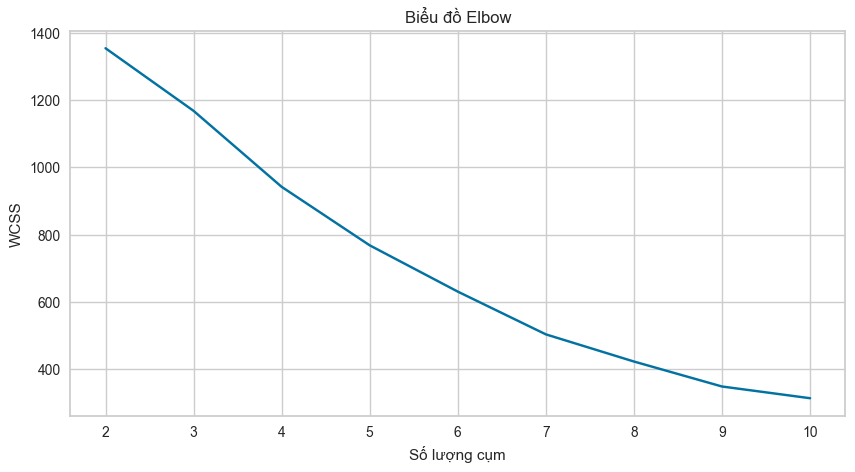
plt.xlabel("Số lượng cụm")

plt.ylabel("WCSS")

plt.show()

****

****

****

from sklearn.metrics import (

    silhouette\_score,

    davies\_bouldin\_score,

    calinski\_harabasz\_score,

)

from yellowbrick.cluster import KElbowVisualizer

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

kmeans = KMeans(n\_clusters=optimal\_k)

kmeans.fit(data\_scaled)

silhouette\_kmeans = silhouette\_score(data\_scaled, kmeans.labels\_)

db\_kmeans = davies\_bouldin\_score(data\_scaled, kmeans.labels\_)

ch\_kmeans = calinski\_harabasz\_score(data\_scaled, kmeans.labels\_)

print("Индекс Silhouette для KMeans: ", silhouette\_kmeans)

print("Индекс Davies-Bouldin для KMeans: ", db\_kmeans)

print("Индекс Calinski-Harabasz для KMeans: ", ch\_kmeans)

agg\_clustering = AgglomerativeClustering(n\_clusters=optimal\_k)

agg\_clustering.fit(data\_scaled)

silhouette\_agg = silhouette\_score(data\_scaled, agg\_clustering.labels\_)

db\_agg = davies\_bouldin\_score(data\_scaled, agg\_clustering.labels\_)

ch\_agg = calinski\_harabasz\_score(data\_scaled, agg\_clustering.labels\_)

print("Индекс Silhouette для Agglomerative Clustering: ", silhouette\_agg)

print("Индекс Davies-Bouldin для Agglomerative Clustering: ", db\_agg)

print("Индекс Calinski-Harabasz для Agglomerative Clustering: ", ch\_agg)

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

    data\_scaled.T, optimal\_k, 2, error=0.005, maxiter=1000, init=None

)

silhouette\_fuzzy\_cmeans = silhouette\_score(data\_scaled, np.argmax(u, axis=0))

db\_fuzzy\_cmeans = davies\_bouldin\_score(data\_scaled, np.argmax(u, axis=0))

ch\_fuzzy\_cmeans = calinski\_harabasz\_score(data\_scaled, np.argmax(u, axis=0))

print("Индекс Silhouette для Fuzzy C-means: ", silhouette\_fuzzy\_cmeans)

print("Индекс Davies-Bouldin для Fuzzy C-means: ", db\_fuzzy\_cmeans)

print("Индекс Calinski-Harabasz для Fuzzy C-means: ", ch\_fuzzy\_cmeans)

dbscan = DBSCAN(eps=0.5, min\_samples=10)

data["cluster\_dbscan"] = dbscan.fit\_predict(data\_scaled)

n\_clusters = len(set(dbscan.labels\_)) - (1 if -1 in dbscan.labels\_ else 0)

print("Количество кластеров: ", n\_clusters)

if n\_clusters > 1:

    silhouette\_dbscan = silhouette\_score(data\_scaled, dbscan.labels\_)

    db\_dbscan = davies\_bouldin\_score(data\_scaled, dbscan.labels\_)

    ch\_dbscan = calinski\_harabasz\_score(data\_scaled, dbscan.labels\_)

    print("Индекс Silhouette для DBSCAN: ", silhouette\_dbscan)

    print("Индекс Davies-Bouldin для DBSCAN: ", db\_dbscan)

    print("Индекс Calinski-Harabasz для DBSCAN: ", ch\_dbscan)

else:

    print(

        "Невозможно рассчитать индексы Silhouette, Davies-Bouldin, Calinski-Harabasz для DBSCAN, так как создан только один кластер."

    )

**На основе индексов Silhouette, Davies-Bouldin и Calinski-Harabasz, методы KMeans и Agglomerative Clustering показывают наилучшие результаты с индексом Silhouette 0.8422532370133237, что значительно выше, чем у Fuzzy C-means (0.24031844392256416). Индекс Davies-Bouldin для KMeans и Agglomerative Clustering также ниже (0.492991904674355), что указывает на лучшую сегментацию кластеров, по сравнению с Fuzzy C-means (1.781475874068944).**

**Метод DBSCAN не создает более одного кластера, поэтому невозможно вычислить индексы Silhouette, Davies-Bouldin и Calinski-Harabasz.**

**Таким образом, на основе этих индексов, KMeans или Agglomerative Clustering могут быть лучшим выбором для этого набора данных. Однако, KMeans обычно предпочтительнее, поскольку он проще и понятнее, чем Agglomerative Clustering.**

from sklearn.cluster import AgglomerativeClustering, DBSCAN, KMeans

from scipy.cluster.hierarchy import dendrogram

from scipy.spatial.distance import cdist

import skfuzzy as fuzz

from sklearn.manifold import TSNE

import umap.umap\_ as umap

import matplotlib.pyplot as plt

import matplotlib.cm as cm

optimal\_k = 2

kmeans = KMeans(n\_clusters=optimal\_k)

kmeans.fit(data\_scaled)

agg\_clustering = AgglomerativeClustering(n\_clusters=optimal\_k)

agg\_clustering.fit(data\_scaled)

cntr, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(

    data\_scaled.T, optimal\_k, 2, error=0.005, maxiter=1000, init=None

)

dbscan = DBSCAN(eps=0.5, min\_samples=10)

dbscan.fit(data\_scaled)

colors = cm.rainbow(np.linspace(0, 1, optimal\_k))

for perplexity in [5, 30, 50, 100]:

    tsne = TSNE(n\_components=2, perplexity=perplexity)

    data\_tsne = tsne.fit\_transform(data\_scaled)

    plt.figure(figsize=(20, 5))

    for i, (model, name) in enumerate(

        [

            (kmeans, "KMeans"),

            (agg\_clustering, "Agglomerative Clustering"),

            (dbscan, "DBSCAN"),

            (fuzz.cluster.cmeans, "Fuzzy C-means"),

        ]

    ):

        plt.subplot(1, 4, i + 1)

        if name != "Fuzzy C-means":

            plt.scatter(data\_tsne[:, 0], data\_tsne[:, 1], c=colors[model.labels\_])

            if hasattr(model, "cluster\_centers\_"):

                plt.scatter(

                    model.cluster\_centers\_[:, 0],

                    model.cluster\_centers\_[:, 1],

                    c="red",

                    marker="x",

                )

        else:

            plt.scatter(

                data\_tsne[:, 0], data\_tsne[:, 1], c=colors[np.argmax(u, axis=0)]

            )

            plt.scatter(cntr[:, 0], cntr[:, 1], c="red", marker="x")

        plt.title(f"t-SNE visualization for {name} with perplexity={perplexity}")

    plt.show()

for n\_neighbors in [5, 15, 30, 50]:

    reducer = umap.UMAP(n\_neighbors=n\_neighbors)

    data\_umap = reducer.fit\_transform(data\_scaled)

    plt.figure(figsize=(20, 5))

    for i, (model, name) in enumerate(

        [

            (kmeans, "KMeans"),

            (agg\_clustering, "Agglomerative Clustering"),

            (dbscan, "DBSCAN"),

            (fuzz.cluster.cmeans, "Fuzzy C-means"),

        ]

    ):

        plt.subplot(1, 4, i + 1)

        if name != "Fuzzy C-means":

            plt.scatter(data\_umap[:, 0], data\_umap[:, 1], c=colors[model.labels\_])

            if hasattr(model, "cluster\_centers\_"):

                plt.scatter(

                    model.cluster\_centers\_[:, 0],

                    model.cluster\_centers\_[:, 1],

                    c="red",

                    marker="x",

                )

        else:

            plt.scatter(

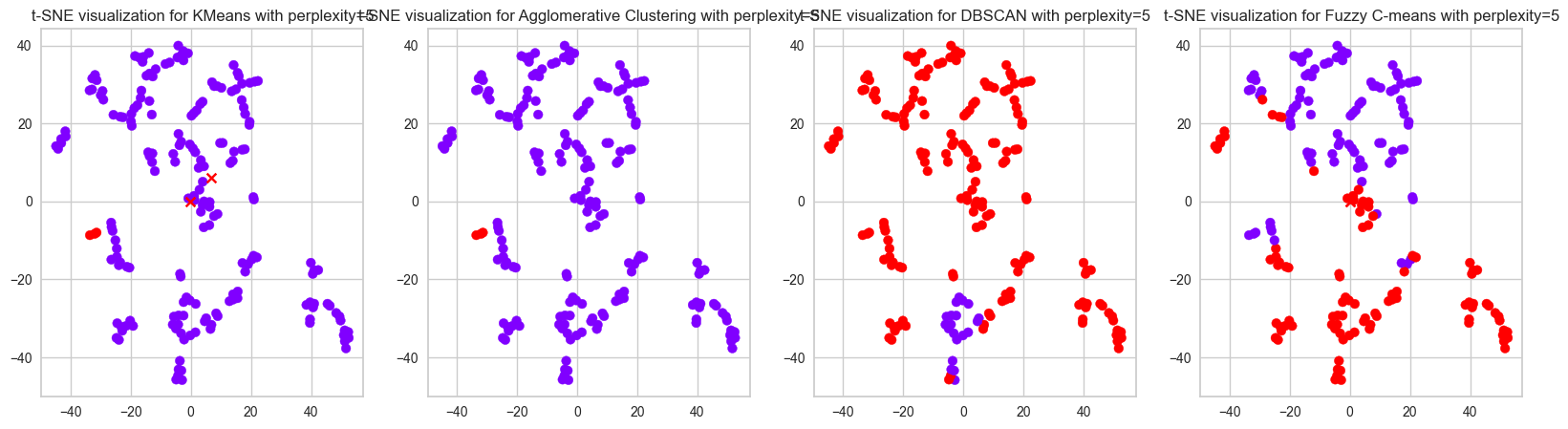
                data\_umap[:, 0], data\_umap[:, 1], c=colors[np.argmax(u, axis=0)]

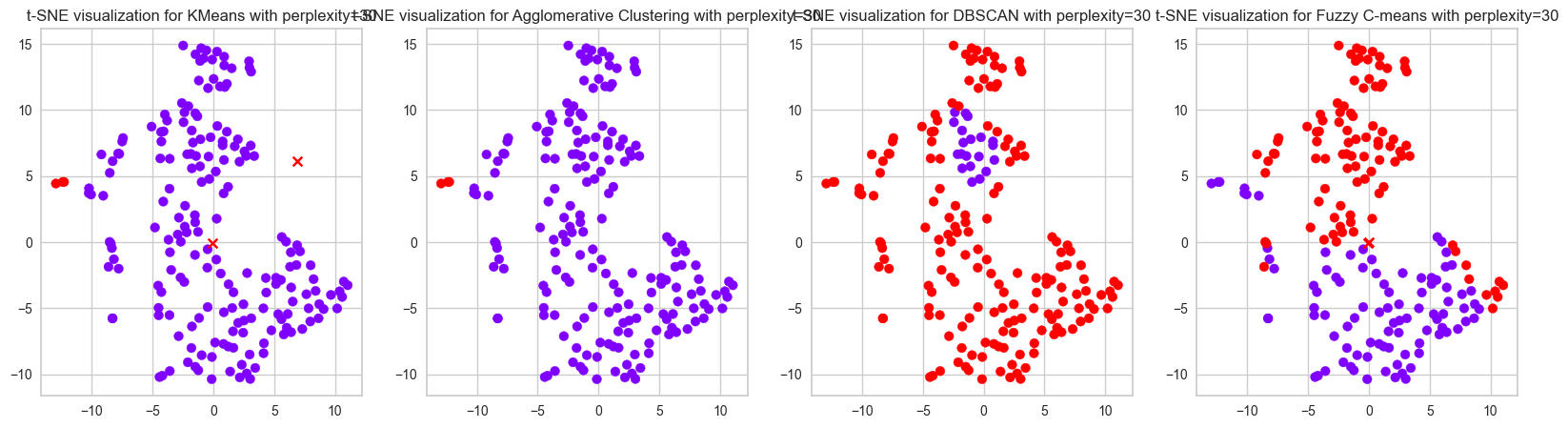
            )

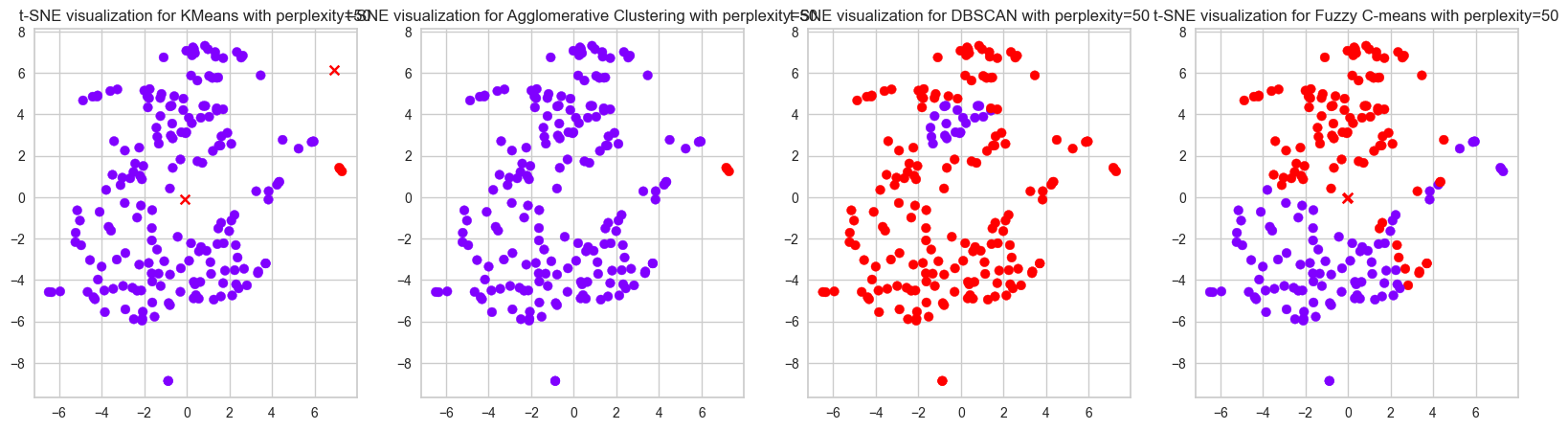
            plt.scatter(cntr[:, 0], cntr[:, 1], c="red", marker="x")

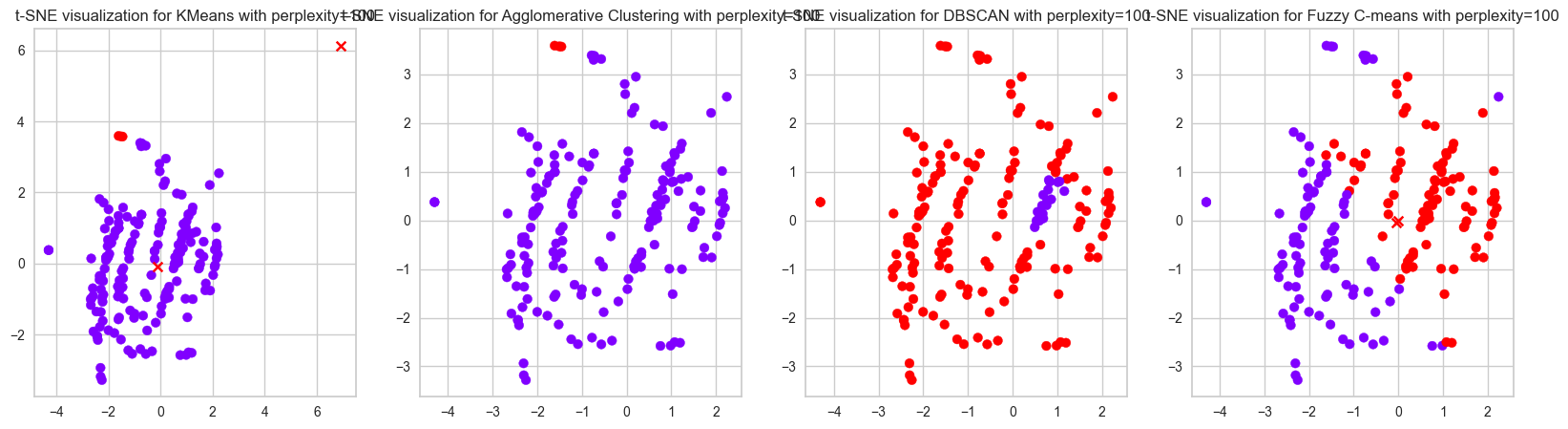
        plt.title(f"UMAP visualization for {name} with n\_neighbors={n\_neighbors}")

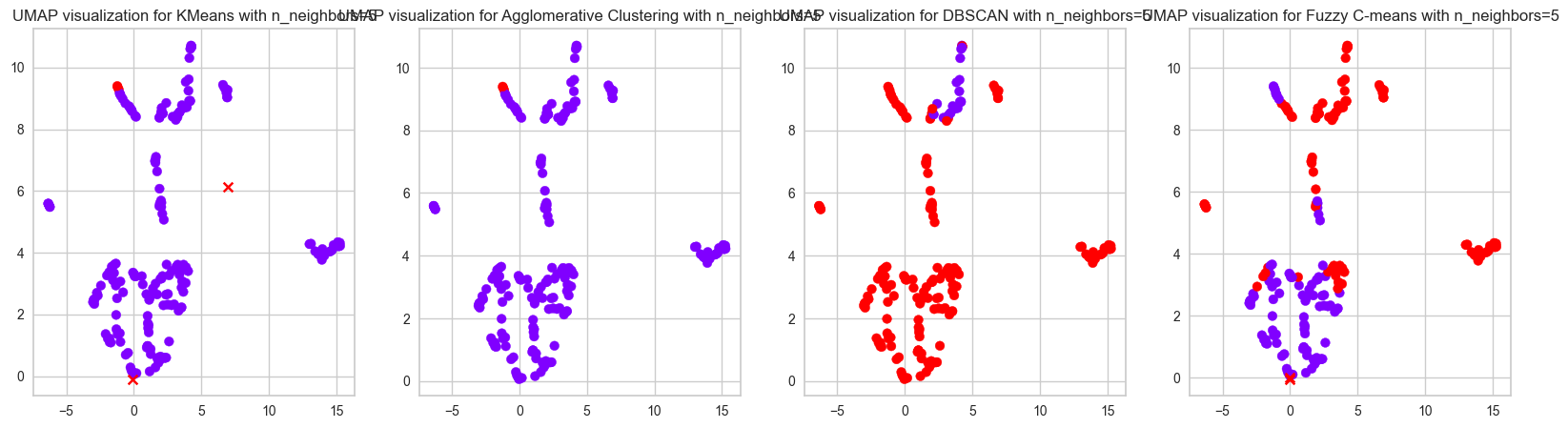
    plt.show()

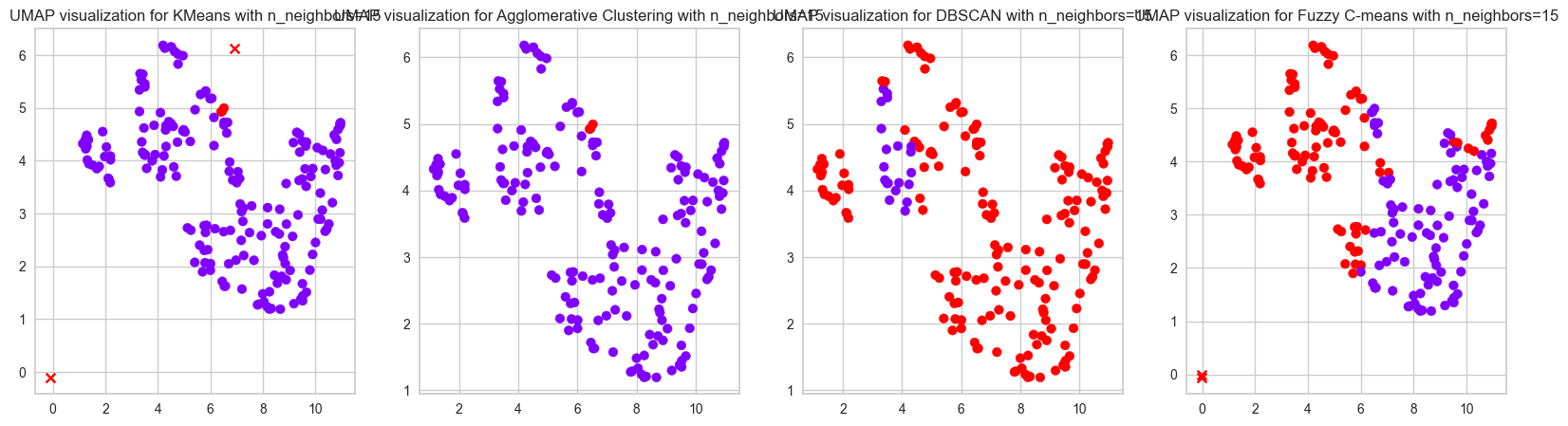
****

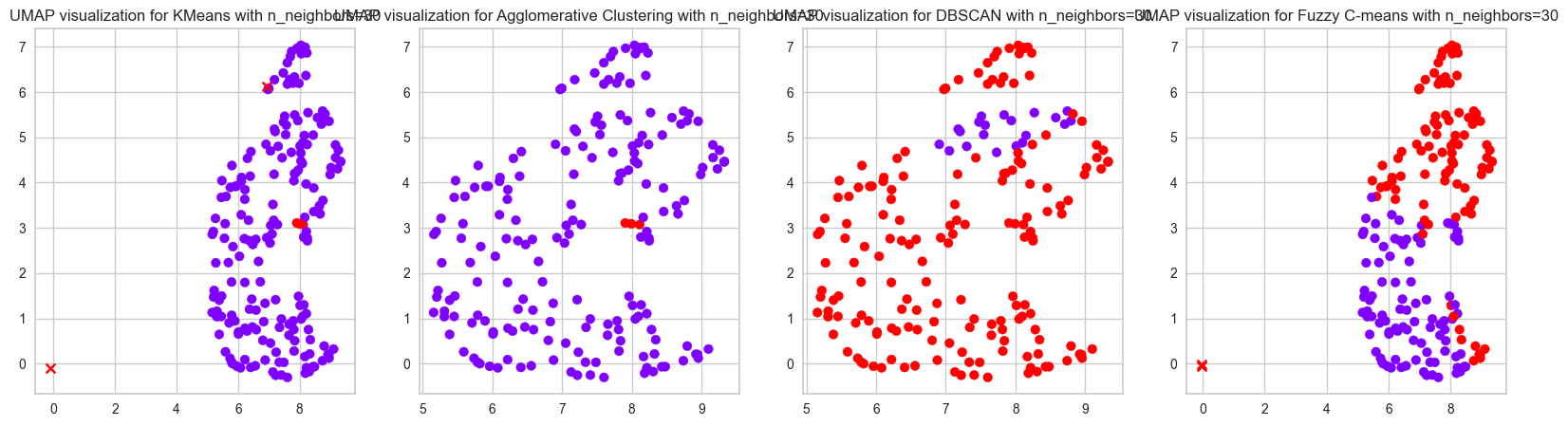
****

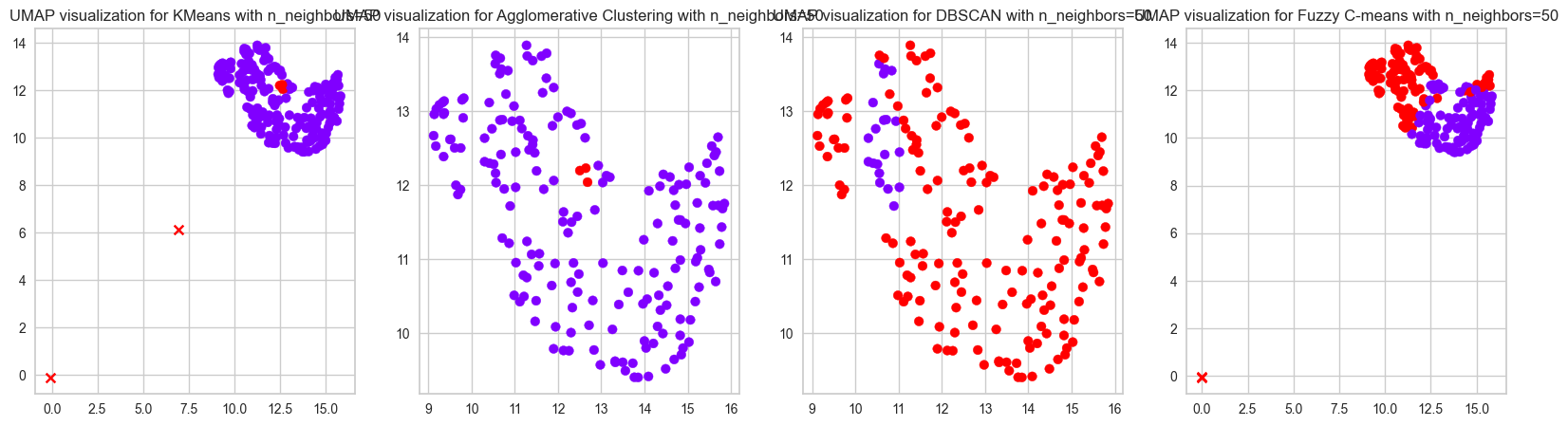
****

****

****

****

****

****