**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи №2 з дисципліни

«Програмування інтелектуальних інформаційних систем»

„Метрики кластеризації/класифікації”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-11 Головня Олександр Ростиславович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Баришич Л. М.*

Київ 2023

**Завдання 1**import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

from scipy.stats import norm

from sklearn.preprocessing import StandardScaler

from scipy import stats

data = '1.csv'

df = pd.read\_csv(data, header=None, sep=', ', engine='python')

col\_names = ['age', 'workclass', 'fnlwgt', 'education', 'education\_num', 'marital\_status', 'occupation', 'relationship',

             'race', 'sex', 'capital\_gain', 'capital\_loss', 'hours\_per\_week', 'native\_country', 'income']

df.columns = col\_names

df.head(5)

X = df.drop(['income'], axis=1)

y = df['income']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.3, random\_state = 0)

for df2 in [X\_train, X\_test]:

    df2['workclass'].fillna(X\_train['workclass'].mode()[0], inplace=True)

    df2['occupation'].fillna(X\_train['occupation'].mode()[0], inplace=True)

    df2['native\_country'].fillna(X\_train['native\_country'].mode()[0], inplace=True)

import category\_encoders as ce

encoder = ce.OneHotEncoder(cols=['workclass', 'education', 'marital\_status', 'occupation', 'relationship',

                                 'race', 'sex', 'native\_country'])

X\_train = encoder.fit\_transform(X\_train)

X\_test = encoder.transform(X\_test)

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

# Gaussian Naive Bayes

gnb = GaussianNB()

gnb.fit(X\_train, y\_train)

y\_pred\_gnb = gnb.predict(X\_test)

# Support Vector Machine

svm = SVC()

svm.fit(X\_train, y\_train)

y\_pred\_svm = svm.predict(X\_test)

from sklearn.metrics import recall\_score

recall\_gnb = recall\_score(y\_test, y\_pred\_gnb, pos\_label='>50K')

recall\_svm = recall\_score(y\_test, y\_pred\_svm, pos\_label='>50K')

print('Gaussian Naive Bayes Recall:', recall\_gnb)

print('Support Vector Machine Recall:', recall\_svm)from sklearn.metrics import f1\_score

f1\_gnb = f1\_score(y\_test, y\_pred\_gnb, pos\_label='>50K')

f1\_svm = f1\_score(y\_test, y\_pred\_svm, pos\_label='>50K')

print('Gaussian Naive Bayes f1-score:', f1\_gnb)

print('Support Vector Machine f1-score:', f1\_svm)

from sklearn.metrics import confusion\_matrix

cm\_gnb = confusion\_matrix(y\_test, y\_pred\_gnb)

cm\_svm = confusion\_matrix(y\_test, y\_pred\_svm)

cm\_matrix\_gnb = pd.DataFrame(data=cm\_gnb, columns=['Actual Positive:1', 'Actual Negative:0'],

                                 index=['Predict Positive:1', 'Predict Negative:0'])

cm\_matrix\_svm = pd.DataFrame(data=cm\_svm, columns=['Actual Positive:1', 'Actual Negative:0'],

                                 index=['Predict Positive:1', 'Predict Negative:0'])

plt.figure(figsize=(10, 4))

plt.subplot(1, 2, 1)

sns.heatmap(cm\_matrix\_gnb, annot=True, fmt='d', cmap='YlGnBu')

plt.title('Confusion Matrix - GNB')

plt.subplot(1, 2, 2)

plt.title('Confusion Matrix - SVM')

sns.heatmap(cm\_matrix\_svm, annot=True, fmt='d', cmap='YlGnBu')

plt.show()

from sklearn.metrics import accuracy\_score

print(f'Gaussian Naive Bayes accuracy score: {(accuracy\_score(y\_test, y\_pred\_gnb))}')

print(f'Support Vector Machine accuracy score: {(accuracy\_score(y\_test, y\_pred\_svm))}')

null\_accuracy = (7407/(7407+2362))

print(f'Null accuracy score: {null\_accuracy}')

print(f'Training set score: {gnb.score(X\_train, y\_train)}')

print(f'Test set score: {gnb.score(X\_test, y\_test)}')

print(f'Training set score: {svm.score(X\_train, y\_train)}')

print(f'Test set score: {svm.score(X\_test, y\_test)}')

**Recall metric**

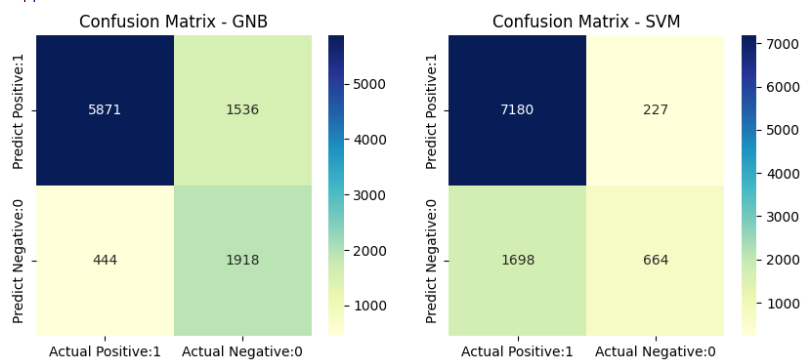
****

(У SVM Recall набагато нижчий, а це значить, що SVM позначає багато істинних значень як хибні)

**F1 metric**

  
(У SVM нижчий – тому модель є менш збалансованою щодо точності та повноти)

**Matrix**



(GNM краще вгадує негативні, а SVM позитивні.)

**Accuracy score**



**null гіпотеза  
**

(Точність обох вище за нуль гіпотезу)

**Over/Under fitting for SVM**



**ЗАВДАННЯ 2**

import itertools

import numpy as np

import matplotlib.pyplot as plt

from matplotlib.ticker import NullFormatter

import pandas as pd

import matplotlib.ticker as ticker

from sklearn import preprocessing

df = pd.read\_csv('2.csv')

df.head()

X = df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',

       'employ', 'retire', 'gender', 'reside']].values

X[0:5]

y = df['custcat'].values

y[0:5]

X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))

X[0:5]

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split( X, y, test\_size=0.25, random\_state=4)

print ('Train set:', X\_train.shape,  y\_train.shape)

print ('Test set:', X\_test.shape,  y\_test.shape)

from sklearn.neighbors import KNeighborsClassifier

k = 4

#Train Model

euclidean = KNeighborsClassifier(n\_neighbors = k, metric='euclidean').fit(X\_train,y\_train)

#Train Model

manhattan = KNeighborsClassifier(n\_neighbors = k, metric='manhattan').fit(X\_train,y\_train)

#Train Model

minkowski = KNeighborsClassifier(n\_neighbors = k, metric='minkowski').fit(X\_train,y\_train)

yhat\_euclidean = euclidean.predict(X\_test)

yhat\_manhattan = manhattan.predict(X\_test)

yhat\_minkowski = minkowski.predict(X\_test)

yhat\_euclidean[0:10], yhat\_manhattan[0:10], yhat\_minkowski[0:10]

from sklearn.metrics import recall\_score

# Calculate and compare recall scores

recall\_euclidean = recall\_score(y\_test, yhat\_euclidean, average='weighted')

recall\_manhattan = recall\_score(y\_test, yhat\_manhattan, average='weighted')

recall\_minkowski = recall\_score(y\_test, yhat\_minkowski, average='weighted')

print('Euclodean metric Recall:', recall\_euclidean)

print('Manhattan metric Recall:', recall\_manhattan)

print('Minkowski metric Recall:', recall\_minkowski)

from sklearn.metrics import f1\_score

f1\_euclidean = f1\_score(y\_test, yhat\_euclidean, average='weighted')

f1\_manhattan = f1\_score(y\_test, yhat\_manhattan, average='weighted')

f1\_minkowski = f1\_score(y\_test, yhat\_minkowski, average='weighted')

print('Euclodean metric f1-score:', f1\_euclidean)

print('Manhattan metric f1-score:', f1\_manhattan)

print('Minkowski metric f1-score:', f1\_minkowski)

from sklearn.metrics import confusion\_matrix

import seaborn as sns

cm\_euclidean = confusion\_matrix(y\_test, yhat\_euclidean)

cm\_manhattan = confusion\_matrix(y\_test, yhat\_manhattan)

cm\_minkowski = confusion\_matrix(y\_test, yhat\_minkowski)

plt.figure(figsize=(9, 3))

plt.subplot(1, 3, 1)

sns.heatmap(cm\_euclidean, annot=True, fmt='d', cmap='Reds', cbar=False)

plt.title('Euclidean metric')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.subplot(1, 3, 2)

sns.heatmap(cm\_manhattan, annot=True, fmt='d', cmap='Purples', cbar=False)

plt.title('Manhattan metric')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.subplot(1, 3, 3)

sns.heatmap(cm\_minkowski, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Minkowski metric')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.tight\_layout()

plt.show()

from sklearn import metrics

print("Test set Accuracy for Euclodean metric: ", metrics.accuracy\_score(y\_test, yhat\_euclidean))

print("Test set Accuracy for Manhattan metric: ", metrics.accuracy\_score(y\_test, yhat\_manhattan))

print("Test set Accuracy for Minkowski metric: ", metrics.accuracy\_score(y\_test, yhat\_minkowski))

pd.Series(y\_test).value\_counts()

null\_accuracy = (70/(70+65+64+51))

print(f'Null accuracy score: {null\_accuracy}')

print(f'Training set score: {euclidean.score(X\_train, y\_train)}')

print(f'Test set score: {euclidean.score(X\_test, y\_test)}')

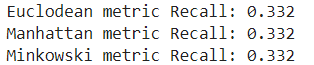
print(f'Training set score: {manhattan.score(X\_train, y\_train)}')

print(f'Test set score: {manhattan.score(X\_test, y\_test)}')

print(f'Training set score: {minkowski.score(X\_train, y\_train)}')

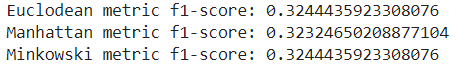
print(f'Test set score: {minkowski.score(X\_test, y\_test)}')

**Recall metric**

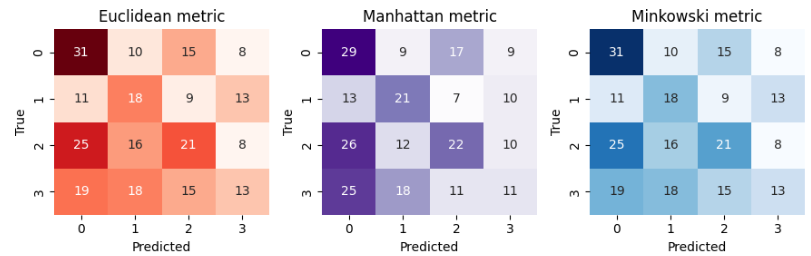
****

(Усі три мають низький)

**F1 metric**

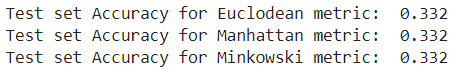
  
(Ф1 досить низький – невисока точність та повнота)

**Matrix**



(Моделі часто помиляються.)

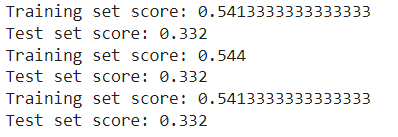
**Accuracy score**

****

(Низька точність)  
**null гіпотеза  
**

(Точність усіх вище за нуль гіпотезу)

**Over/Under fitting for SVM**



(Бачимо, що у всіх є Under fitting)

**ЗАВДАННЯ 3**

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

import matplotlib.pyplot as plt #Data Visualization

import seaborn as sns  #Python library for Vidualization

import os

np.random.seed(10)

from sklearn import cluster, datasets, mixture

X1,Y1 = datasets.make\_moons(n\_samples=2000, noise=.09,random\_state=10)

plt.scatter(X1[:, 0], X1[:, 1], s=10, c=Y1)

plt.title('DATASET 1')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.show()

from sklearn.datasets import make\_blobs

X3,Y3  = make\_blobs(n\_samples=2000,cluster\_std=3.5,centers=2, n\_features=2,random\_state=10)

plt.title('DATASET 2')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.scatter(X3[:, 0], X3[:, 1], s=10, c=Y3)

plt.show()

from sklearn.cluster import AgglomerativeClustering

agnes1 = AgglomerativeClustering(n\_clusters=2)

y\_agnes1 = agnes1.fit\_predict(X1)

print(y\_agnes1, y\_agnes1.shape)

agnes2 = AgglomerativeClustering(n\_clusters=2)

y\_agnes2 = agnes2.fit\_predict(X3)

print(y\_agnes2, y\_agnes2.shape)

from sklearn.cluster import Birch

birch1 = Birch(n\_clusters=2,threshold=0.5,branching\_factor=100)

y\_birch1 = birch1.fit\_predict(X1)

print(y\_birch1, y\_birch1.shape)

birch2 = Birch(n\_clusters=2,threshold=0.1,branching\_factor=100)

y\_birch2 = birch2.fit\_predict(X3)

print(y\_birch2, y\_birch2.shape)

from sklearn.cluster import DBSCAN

dbscan1 = DBSCAN(eps=.2, min\_samples=70)

y\_dbscan1 = dbscan1.fit\_predict(X1)

print(y\_dbscan1, y\_dbscan1.shape)

dbscan2 = DBSCAN(eps=1, min\_samples=10)

y\_dbscan2 = dbscan2.fit\_predict(X3)

print(y\_dbscan2, y\_dbscan2.shape)

plt.figure(figsize=(14,5))

plt.subplot(1,2,1)

plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y\_agnes1)

plt.title('Agnes Dataset1')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.subplot(1,2,2)

plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y\_agnes2)

plt.title('Agnes Dataset2')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.show()

plt.figure(figsize=(14,5))

plt.subplot(1,2,1)

plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y\_birch1)

plt.title('Birch Dataset1')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.subplot(1,2,2)

plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y\_birch2)

plt.title('Birch Dataset2')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.show()

plt.figure(figsize=(14,5))

plt.subplot(1,2,1)

plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y\_dbscan1)

plt.title('DBSCAN Dataset1')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.subplot(1,2,2)

plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y\_dbscan2)

plt.title('DBSCAN Dataset2')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.show()

from sklearn.metrics import silhouette\_score

print("Dataset1:")

sil\_birch=silhouette\_score(X1,y\_birch1)

sil\_dbscan=silhouette\_score(X1,y\_dbscan1)

sil\_agnes=silhouette\_score(X1,y\_agnes1)

print("Silhouette Coefficient with Birch :"+ str(sil\_birch))

print("Silhouette Coefficient with Dbscan : "+ str(sil\_dbscan))

print("Silhouette Coefficient with Agnes : "+ str(sil\_agnes))

print("Dataset2:")

sil\_birch=silhouette\_score(X3,y\_birch2)

sil\_dbscan=silhouette\_score(X3,y\_dbscan2)

sil\_agnes=silhouette\_score(X3,y\_agnes2)

print("Silhouette Coefficient with Birch :"+ str(sil\_birch))

print("Silhouette Coefficient with Dbscan : "+ str(sil\_dbscan))

print("Silhouette Coefficient with Agnes : "+ str(sil\_agnes))

from sklearn.metrics.cluster import adjusted\_rand\_score

print("DATASET1:")

ari\_birch=adjusted\_rand\_score(Y1,y\_birch1)

ari\_dbscan=adjusted\_rand\_score(Y1,y\_dbscan1)

ari\_agnes=adjusted\_rand\_score(Y1,y\_agnes1)

print("ARI of Birch :"+ str(ari\_birch))

print("ARI of Dbscan: "+ str(ari\_dbscan))

print("ARI of Agnes: "+ str(ari\_agnes))

print("DATASET2:")

ari\_birch=adjusted\_rand\_score(Y3,y\_birch2)

ari\_dbscan=adjusted\_rand\_score(Y3,y\_dbscan2)

ari\_agnes=adjusted\_rand\_score(Y3,y\_agnes2)

print("ARI of Birch :"+ str(ari\_birch))

print("ARI of Dbscan: "+ str(ari\_dbscan))

print("ARI of Agnes: "+ str(ari\_agnes))

from sklearn.metrics.cluster import normalized\_mutual\_info\_score

print("DATASET1:")

nmi\_birch=normalized\_mutual\_info\_score(Y1,y\_birch1)

nmi\_dbscan=normalized\_mutual\_info\_score(Y1,y\_dbscan1)

nmi\_agnes=normalized\_mutual\_info\_score(Y1,y\_agnes1)

print("NMI of Birch :"+ str(nmi\_birch))

print("NMI of Dbscan: "+ str(nmi\_dbscan))

print("NMI of Agnes: "+ str(nmi\_agnes))

print("DATASET2:")

nmi\_birch=normalized\_mutual\_info\_score(Y3,y\_birch2)

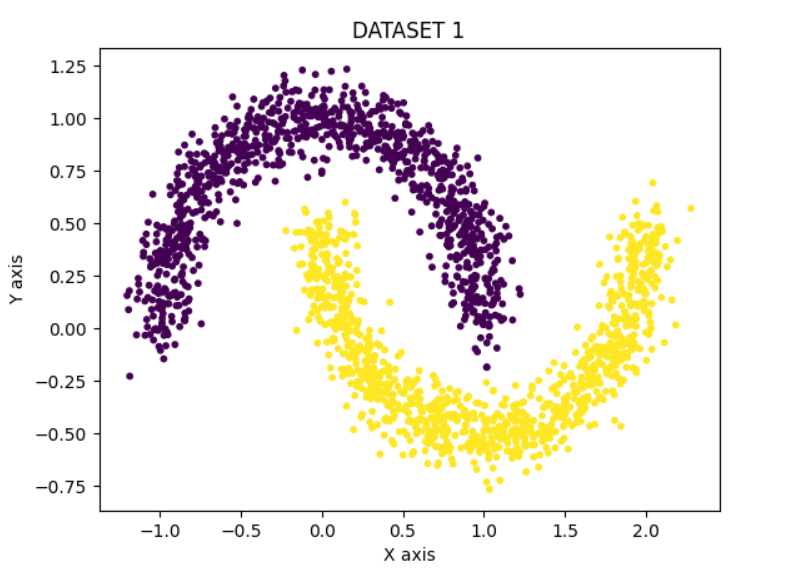
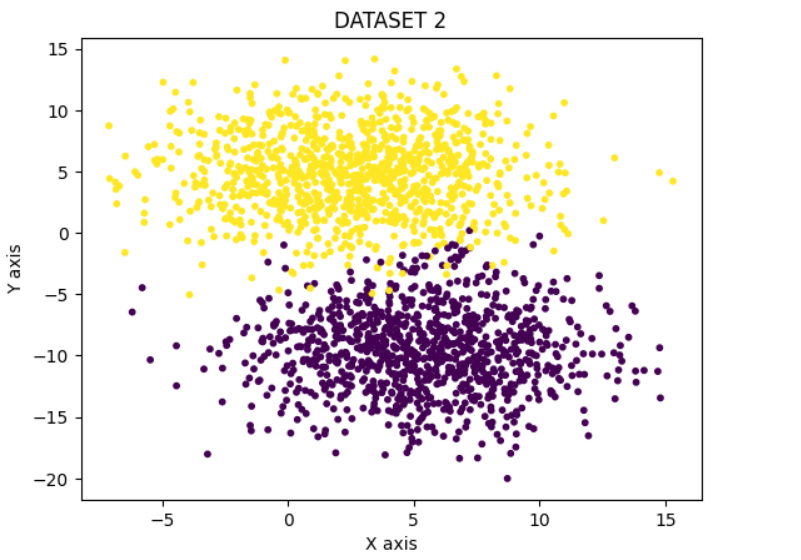
nmi\_dbscan=normalized\_mutual\_info\_score(Y3,y\_dbscan1)

nmi\_agnes=normalized\_mutual\_info\_score(Y3,y\_agnes2)

print("NMI of Birch :"+ str(nmi\_birch))

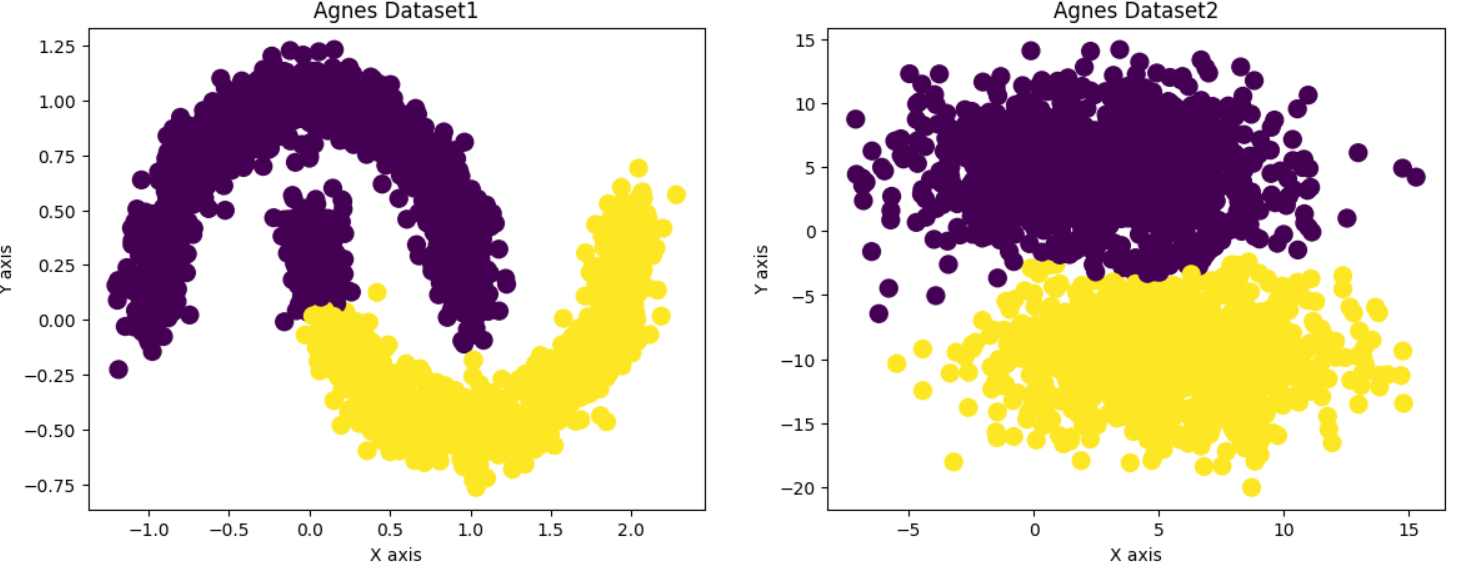
print("NMI of Dbscan: "+ str(nmi\_dbscan))

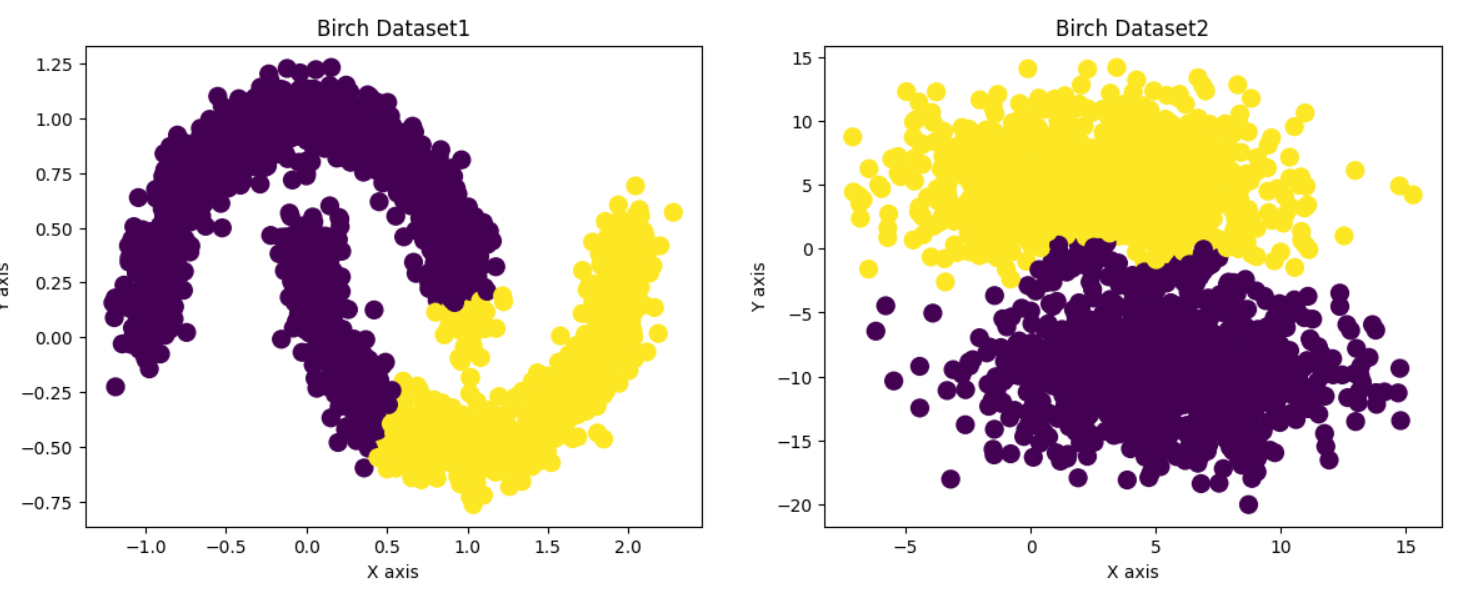
print("NMI of Agnes: "+ str(nmi\_agnes))

**** ****

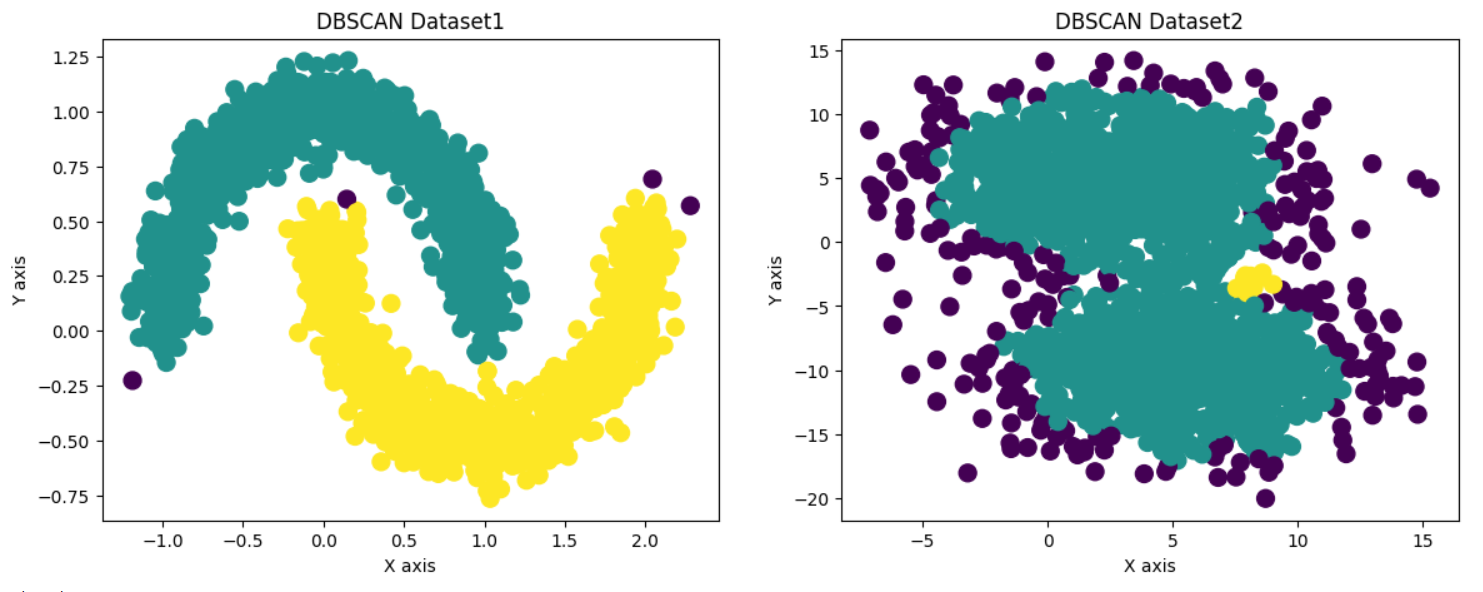
(Два датасети, розміром в 2000 екземплярів)

**AGNES**

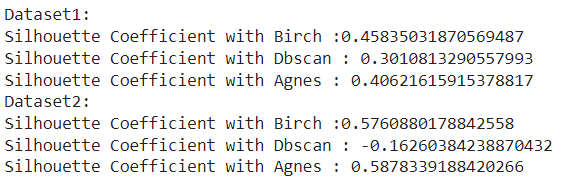
**  
Birch**

****

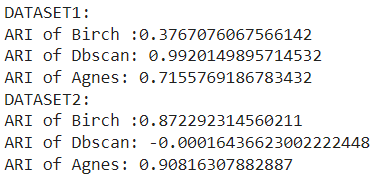
**DBSCAN**

****

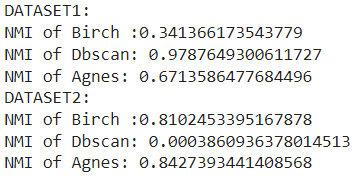
**Silhouette Coefficient**

****

**ARI**



**NMI**



(Бачимо різницю в якості кластеризації для кожного датасету)

**ЗАВДАННЯ 4**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

from scipy import stats

mall\_data = pd.read\_csv('4.csv')

print(mall\_data.head(5))

from sklearn.cluster import KMeans

X\_numerics = mall\_data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

KM\_6\_clusters = KMeans(n\_clusters=6, init='k-means++').fit(X\_numerics) # initialise and fit K-Means model

KM6\_clustered = X\_numerics.copy()

KM6\_clustered.loc[:,'Cluster'] = KM\_6\_clusters.labels\_ # append labels to points

KM6\_clust\_sizes = KM6\_clustered.groupby('Cluster').size().to\_frame()

KM6\_clust\_sizes.columns = ["KM\_size"]

KM6\_clust\_sizes

from sklearn.cluster import AffinityPropagation

AF = AffinityPropagation(preference=-11800).fit(X\_numerics)

AF\_clustered = X\_numerics.copy()

AF\_clustered.loc[:,'Cluster'] = AF.labels\_ # append labels to points

AF\_clust\_sizes = AF\_clustered.groupby('Cluster').size().to\_frame()

AF\_clust\_sizes.columns = ["AF\_size"]

AF\_clust\_sizes

import plotly as py

import plotly.graph\_objs as go

def tracer(db, n, name):

    '''

    This function returns trace object for Plotly

    '''

    return go.Scatter3d(

        x = db[db['Cluster']==n]['Age'],

        y = db[db['Cluster']==n]['Spending Score (1-100)'],

        z = db[db['Cluster']==n]['Annual Income (k$)'],

        mode = 'markers',

        name = name,

        marker = dict(

            size = 5

        )

     )

# Plotly interactive 3D plot

trace0 = tracer(KM6\_clustered, 0, 'Cluster 0')

trace1 = tracer(KM6\_clustered, 1, 'Cluster 1')

trace2 = tracer(KM6\_clustered, 2, 'Cluster 2')

trace3 = tracer(KM6\_clustered, 3, 'Cluster 3')

trace4 = tracer(KM6\_clustered, 4, 'Cluster 4')

trace5 = tracer(KM6\_clustered, 5, 'Cluster 5')

data = [trace0, trace1, trace2, trace3, trace4, trace5]

layout = go.Layout(

    title = 'Clusters by K-Means',

    scene = dict(

            xaxis = dict(title = 'Age'),

            yaxis = dict(title = 'Spending Score'),

            zaxis = dict(title = 'Annual Income')

        )

)

fig = go.Figure(data=data, layout=layout)

py.offline.iplot(fig)

# Plotly interactive 3D plot

trace0 = tracer(AF\_clustered, 0, 'Cluster 0')

trace1 = tracer(AF\_clustered, 1, 'Cluster 1')

trace2 = tracer(AF\_clustered, 2, 'Cluster 2')

trace3 = tracer(AF\_clustered, 3, 'Cluster 3')

trace4 = tracer(AF\_clustered, 4, 'Cluster 4')

trace5 = tracer(AF\_clustered, 5, 'Cluster 5')

data = [trace0, trace1, trace2, trace3, trace4, trace5]

layout = go.Layout(

    title = 'Clusters by Affinity Propagation',

    scene = dict(

            xaxis = dict(title = 'Age'),

            yaxis = dict(title = 'Spending Score'),

            zaxis = dict(title = 'Annual Income')

        )

)

fig = go.Figure(data=data, layout=layout)

py.offline.iplot(fig)

from sklearn.cluster import DBSCAN

DBS\_clustering = DBSCAN(eps=12.5, min\_samples=4).fit(X\_numerics)

DBSCAN\_clustered = X\_numerics.copy()

DBSCAN\_clustered.loc[:,'Cluster'] = DBS\_clustering.labels\_ # append labels to points

DBSCAN\_clust\_sizes = DBSCAN\_clustered.groupby('Cluster').size().to\_frame()

DBSCAN\_clust\_sizes.columns = ["DBSCAN\_size"]

DBSCAN\_clust\_sizes

from sklearn.metrics import silhouette\_score

sil\_kmeans=silhouette\_score(X\_numerics,KM\_6\_clusters.labels\_)

sil\_af=silhouette\_score(X\_numerics,AF.labels\_)

print("Silhouette Coefficient with K-Means :"+ str(sil\_kmeans))

print("Silhouette Coefficient with Affinity Propagation : "+ str(sil\_af))

from sklearn.metrics.cluster import adjusted\_rand\_score

ari\_kmeans=adjusted\_rand\_score(DBSCAN\_clustered.loc[:,'Cluster'],KM6\_clustered.loc[:,'Cluster'])

ari\_af=adjusted\_rand\_score(DBSCAN\_clustered.loc[:,'Cluster'],AF\_clustered.loc[:,'Cluster'])

print("ARI of K-Means :"+ str(ari\_kmeans))

print("ARI of Affinity Propagation: "+ str(ari\_af))

from sklearn.metrics.cluster import normalized\_mutual\_info\_score

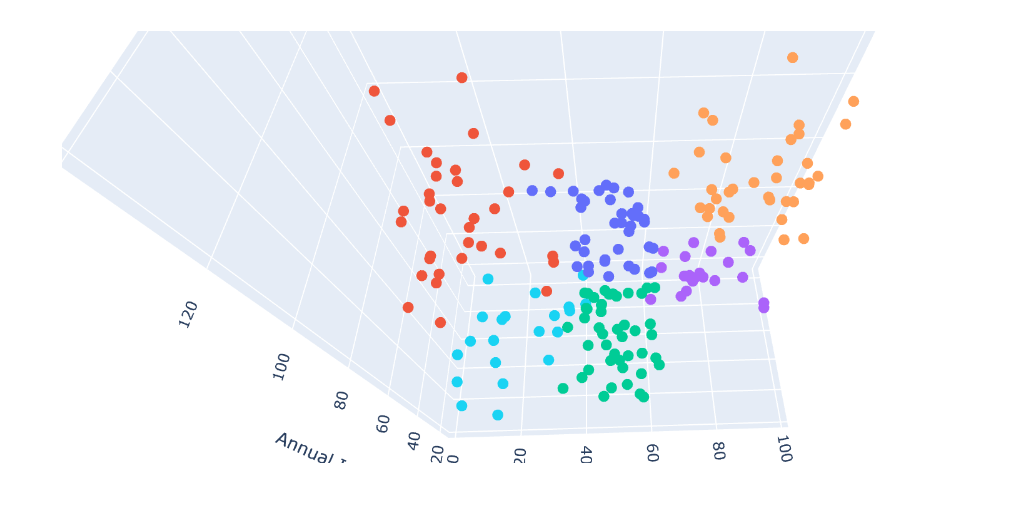
nmi\_kmeans=normalized\_mutual\_info\_score(DBSCAN\_clustered.loc[:,'Cluster'],KM6\_clustered.loc[:,'Cluster'])

nmi\_af=normalized\_mutual\_info\_score(DBSCAN\_clustered.loc[:,'Cluster'],AF\_clustered.loc[:,'Cluster'])

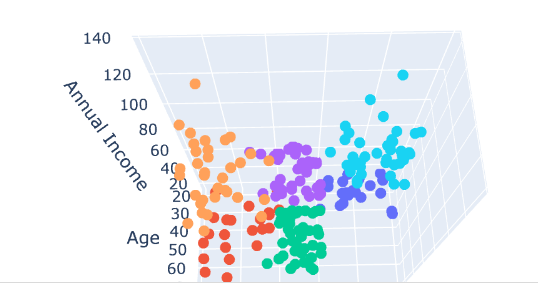
print("NMI of K-Means :"+ str(nmi\_kmeans))

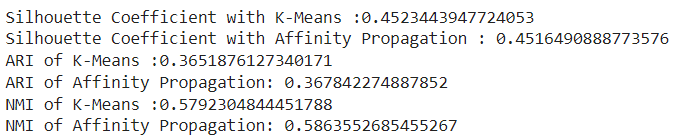
print("NMI of Affinity Propagation: "+ str(nmi\_af))

**K-means**

****

**Affinity Propagation**

****

**Metrics:  
**