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»

„**Методи класифікації і кластеризації**”

Виконав(ла) *ІП-11 Прищепа В.С.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Баришич Л. М.* _____
(прізвище, ім'я, по батькові)

Київ 2023

ЗАВДАННЯ:

1.Dataset1: /kaggle/input/adult-dataset/adult.csv'

Bayesian Classification + Support Vector Machine

Зробити предікшн двома вищезгаданими алгоритмами. Порівняти наступні метрики:

Recall, f1-score, Confusion matrix, accuracy score. Порівняти з нуль-гіпотезою і перевірити на оверфітинг. Пояснити результати.

2.Dataset2: https://www.kaggle.com/code/stieranka/k-nearest-neighbors

K nearest neighbours.

Те саме що і в 1 завданні, але порівнюємо між собою метрики. Euclidean, Manhattan, Minkowski. Кластери потрібно візуалізувати. Метрики аналогічно п.1

3.Dataset3: https://www.kaggle.com/code/nuhashafnan/cluster-analysis-kmeans-kmediod-agnes-birch-dbscan

Agnes,Birch,DBSCAN

Інші методи можна ігнорувати. Зняти метрики (Silhouette Coefficient, ARI, NMI. Можна з п.1-2), пояснити.

4.Dataset4: https://www.kaggle.com/code/datark1/customers-clustering-k-means-dbscan-and-ap

Affinity propagation.

Порівняти з k-means. Метрики - Silhouette Coefficient, ARI, NMI

Хід роботи:

1.**Bayesian Classification + Support Vector Machine**

Код програми:

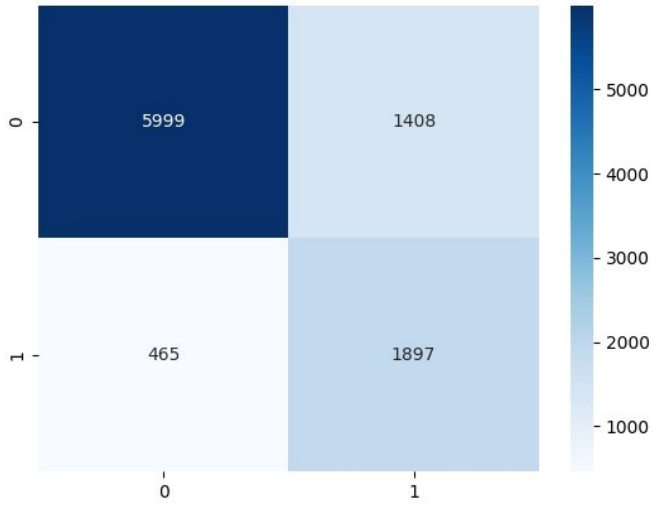
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
import category_encoders as ce
```

```
from sklearn.preprocessing import RobustScaler
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

def prediction(classifier, X_train, y_train, X_test, y_test):
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))
    y_pred_train = classifier.predict(X_train)
    print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train,
y_pred_train)))
    cm = confusion_matrix(y_test, y_pred)
    null_hpt=y_test.value_counts()
    null_accuracy = (max(null_hpt) / sum(null_hpt))
    print('Null accuracy score: {0:0.4f}'.format(null_accuracy))
    print(classification_report(y_test, y_pred))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.show()

#Prepearing
df = pd.read_csv("adult.csv", header=None, sep=',\s')
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
categorical = [var for var in df.columns if df[var].dtype=='O']
df['workclass'].replace('?', np.NaN, inplace=True)
df['occupation'].replace('?', np.NaN, inplace=True)
df['native_country'].replace('?', np.NaN, inplace=True)
X = df.drop(['income'], axis=1)
y = df['income']

#Training
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)
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)
cols = X_train.columns
scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])

#Results
gnb = GaussianNB()
svc = SVC()
print("Gaussian Naive Bayes:")
prediction(gnb, X_train, y_train, X_test, y_test)
print("Support Vector Machine:")
prediction(svc, X_train, y_train, X_test, y_test)
```

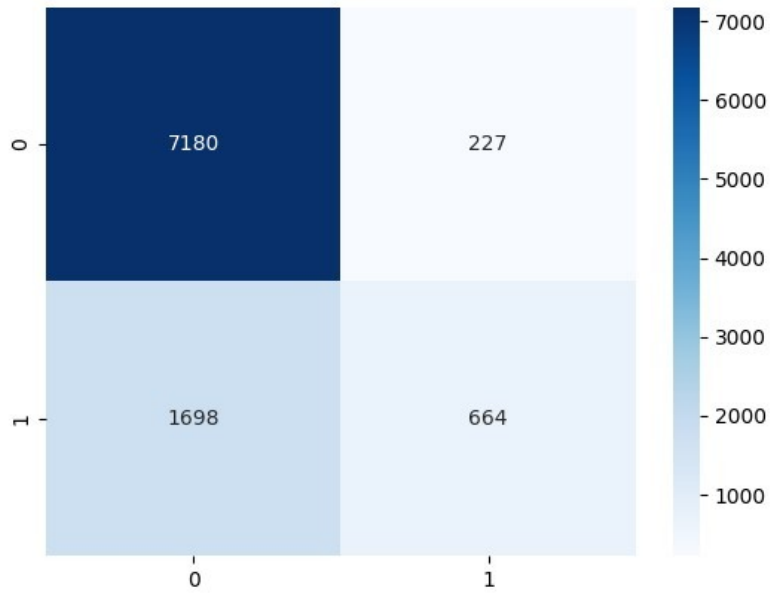
Результат:

Gaussian Naive Bayes:




```
Gaussian Naive Bayes:
Model accuracy score: 0.8083
Training-set accuracy score: 0.8067
Null accuracy score: 0.7582
              precision    recall  f1-score   support

       <=50K       0.93      0.81      0.86      7407
        >50K       0.57      0.80      0.67      2362

    accuracy                           0.81      9769
   macro avg       0.75      0.81      0.77      9769
weighted avg       0.84      0.81      0.82      9769
```

Support Vector Machine:




```
Support Vector Machine:
Model accuracy score: 0.8029
Training-set accuracy score: 0.8021
Null accuracy score: 0.7582
              precision    recall  f1-score   support

       <=50K       0.81      0.97      0.88      7407
        >50K       0.75      0.28      0.41      2362

    accuracy                           0.80      9769
   macro avg       0.78      0.63      0.65      9769
weighted avg       0.79      0.80      0.77      9769
```

Отже, бачимо, що в обох випадках точності на тренувальних та тестових наборах приблизно рівні, а точності моделей на тестових наборах більші за null-accuracy. Це значить, що оверфіт відсутній і моделі гарно справляються із прогнозуванням класів. В загальному, метрики у Gaussian Naive Bayes трішки кращі, ніж у Support Vector Machine і застосовує ресурсів перша модель менше, ніж друга, тому Gaussian Naive Bayes краща.

2.**K nearest neighbours.**

Код програми:

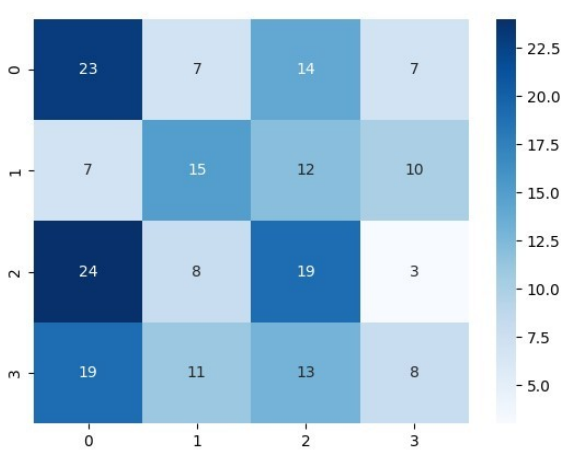
```
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

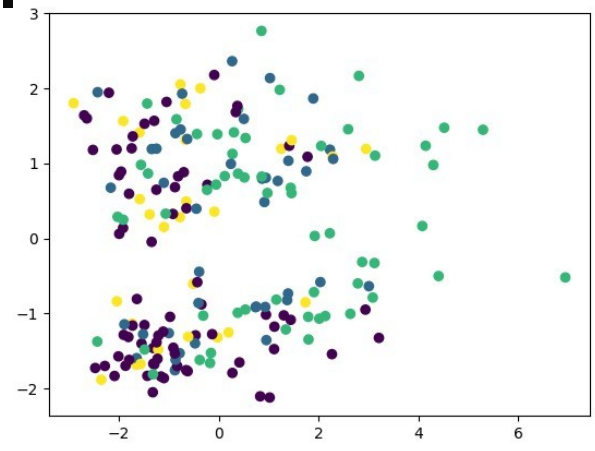
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA

def prediction(classifier, X_train, y_train, X_test, y_test):
    classifier.fit(X_train, y_train)
    y_pred = classifier.predict(X_test)
    print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))
    y_pred_train = classifier.predict(X_train)
    print('Training-set accuracy score: {0:0.4f}'.format(accuracy_score(y_train,
y_pred_train)))
    cm = confusion_matrix(y_test, y_pred)
    null_hpt=y_test.value_counts()
    null_accuracy = (max(null_hpt) / sum(null_hpt))
    print('Null accuracy score: {0:0.4f}'.format(null_accuracy))
    print(classification_report(y_test, y_pred))

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.show()

    pca = PCA(n_components=2)
    pca.fit(X_train)
    X_test_pca = pca.transform(X_test)
    plt.scatter(X_test_pca[:, 0], y=X_test_pca[:, 1], c=y_pred)
    plt.show()

df = pd.read_csv('teleCust1000t.csv')
X = df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed', 'employ', 'retire',
'gender', 'reside']]
y = df['custcat']
standardizer=StandardScaler()
X = standardizer.fit(X).transform(X.astype(float))
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.2,
random_state=4)
for j in ['minkowski', 'euclidean', 'manhattan']:
      knn=KNeighborsClassifier(n_neighbors=8, metric=j, n_jobs=-1)
      print(j)
      prediction(knn, X_train,y_train,X_test,y_test)
```

Результат:

minkowski:

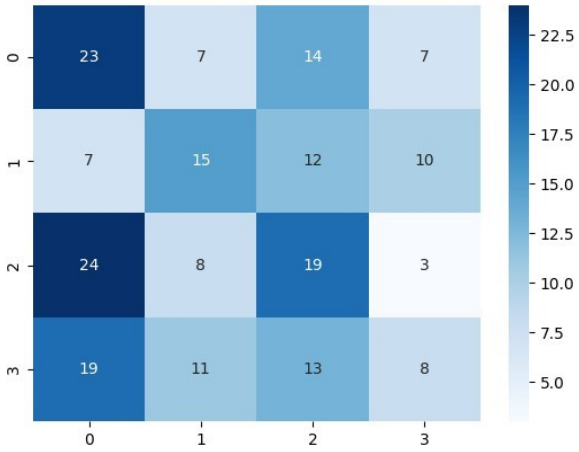
```
minkowski
Model accuracy score: 0.3250
Training-set accuracy score: 0.4925
Null accuracy score: 0.2700
              precision    recall  f1-score   support

           1       0.32      0.45      0.37        51
           2       0.37      0.34      0.35        44
           3       0.33      0.35      0.34        54
           4       0.29      0.16      0.20        51

    accuracy                           0.33       200
   macro avg       0.32      0.33      0.32       200
weighted avg       0.32      0.33      0.32       200
```

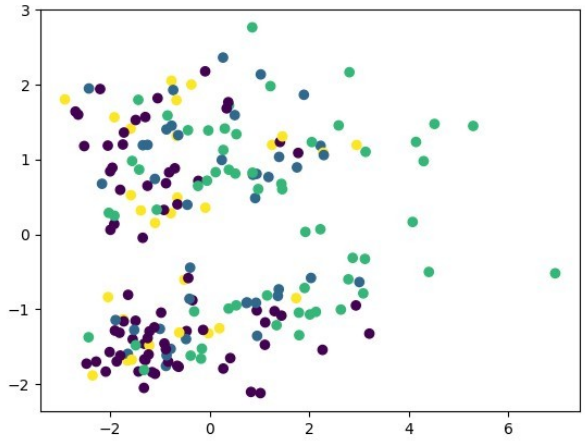




euclidean:

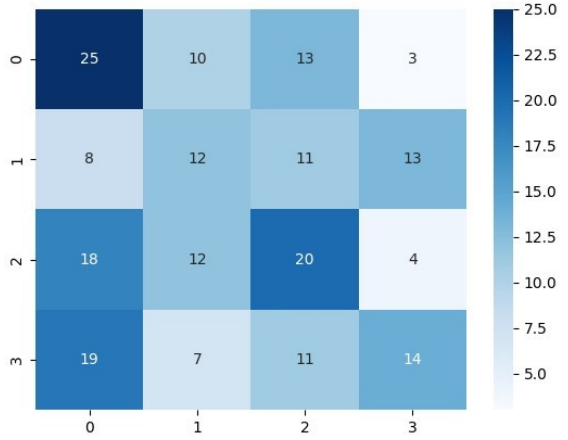
```
euclidean
Model accuracy score: 0.3250
Training-set accuracy score: 0.4925
Null accuracy score: 0.2700
              precision    recall  f1-score   support

           1       0.32      0.45      0.37        51
           2       0.37      0.34      0.35        44
           3       0.33      0.35      0.34        54
           4       0.29      0.16      0.20        51

    accuracy                           0.33       200
   macro avg       0.32      0.33      0.32       200
weighted avg       0.32      0.33      0.32       200
```

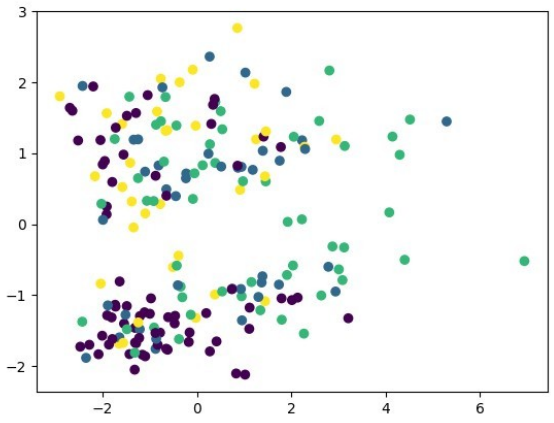




manhattan:

```
manhattan
Model accuracy score: 0.3550
Training-set accuracy score: 0.5038
Null accuracy score: 0.2700
              precision    recall  f1-score   support

           1       0.36      0.49      0.41        51
           2       0.29      0.27      0.28        44
           3       0.36      0.37      0.37        54
           4       0.41      0.27      0.33        51

    accuracy                           0.36       200
   macro avg       0.36      0.35      0.35       200
weighted avg       0.36      0.35      0.35       200
```





У всіх трьох моделях велика різниця між точностями на тренувальних та тестових вибірках, що свідчить про наявність оверфіту. До того ж, точності всіх моделей занизькі, через що усі моделі є неточними, але модель з метрикою Мангетенська відстань показала себе трохи краще.

3.**Agnes,Birch,DBSCAN**

Код програми:

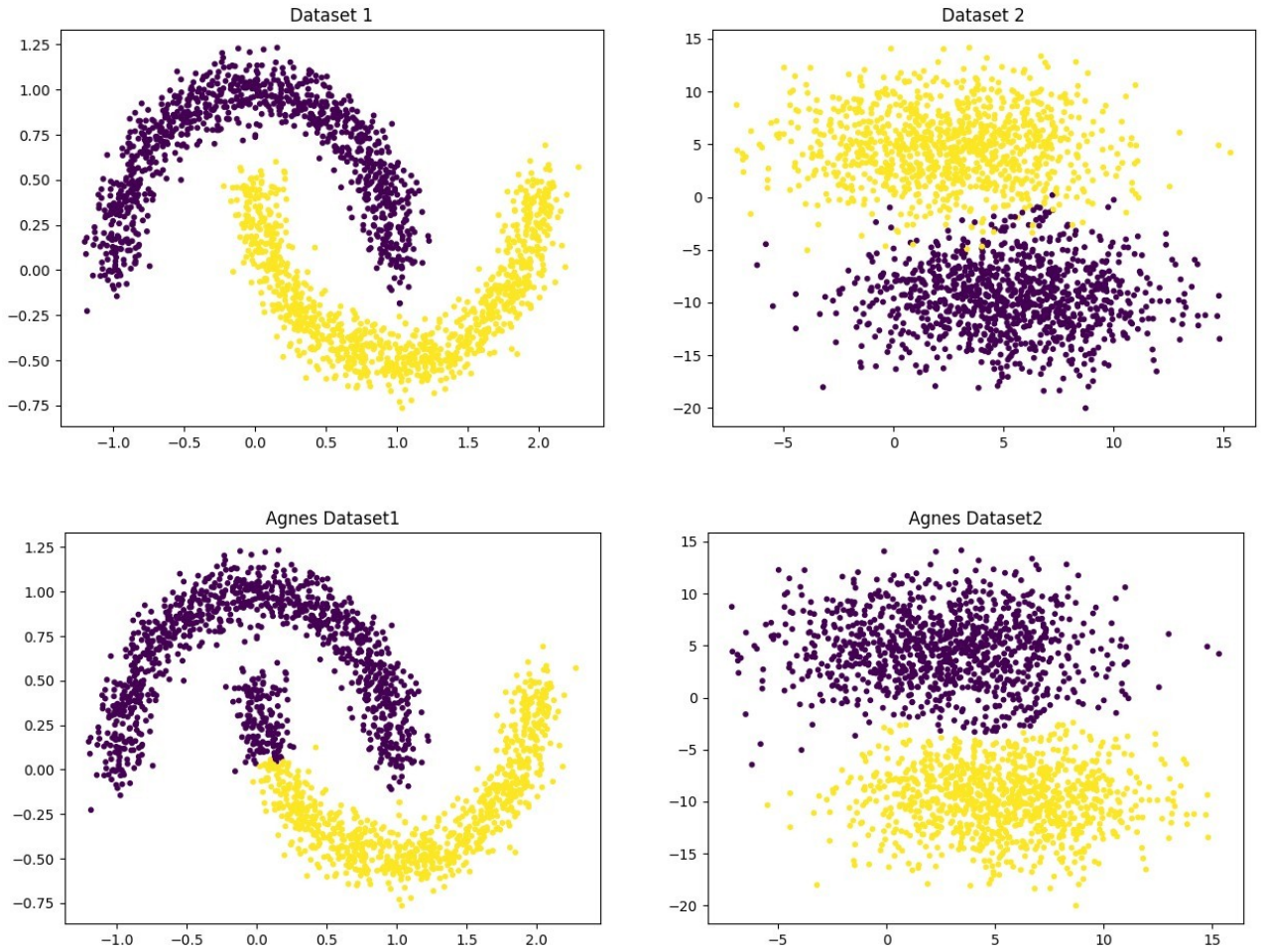
import numpy as np

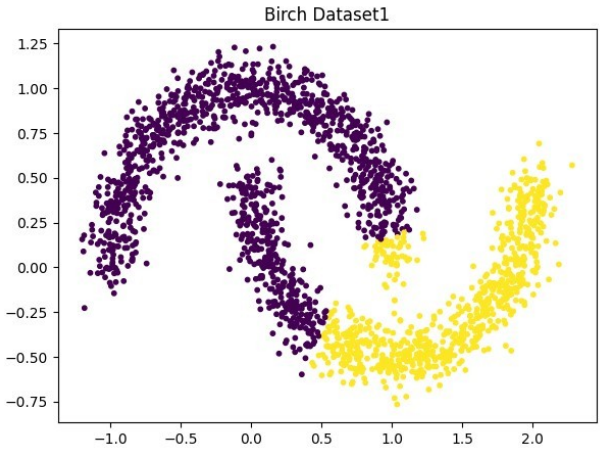
```
import matplotlib.pyplot as plt
from sklearn.cluster import Birch, AgglomerativeClustering, DBSCAN
from sklearn import datasets
from sklearn.datasets import make_blobs
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.cluster import normalized_mutual_info_score
from sklearn.metrics import silhouette_score
import pandas as pd

np.random.seed(10)

#Datasets
X1,Y1 = datasets.make_moons(n_samples=2000, noise=.09,random_state=10)
X2,Y2  = make_blobs(n_samples=2000,cluster_std=3.5,centers=2,
n_features=2,random_state=10)
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=Y1)
plt.title('Dataset 1')
plt.subplot(1,2,2)
plt.scatter(X2[:, 0], X2[:, 1], s=10, c=Y2)
plt.title('Dataset 2')
plt.show()

#Agnes
agnesmodel = AgglomerativeClustering(n_clusters=2)
y_agnes=agnesmodel.fit_predict(X1)
agnesmodel2 = AgglomerativeClustering(n_clusters=2)
y_agnes2=agnesmodel2.fit_predict(X2)
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=y_agnes)
plt.title('Agnes Dataset1')
plt.subplot(1,2,2)
plt.scatter(X2[:, 0], X2[:, 1], s=10, c=y_agnes2)
plt.title('Agnes Dataset2')
plt.show()

#Birch
birchmodel=Birch(n_clusters=2,threshold=0.5,branching_factor=100)
y_birch=birchmodel.fit_predict(X1)
birchmodel2=Birch(n_clusters=2,threshold=0.1,branching_factor=100)
y_birch2=birchmodel2.fit_predict(X2)
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
```

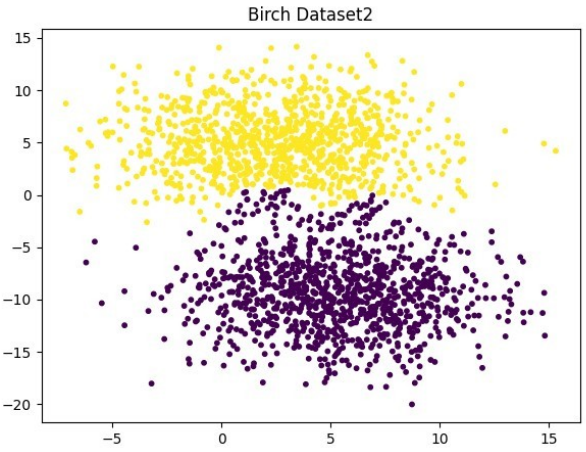
```
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=y_birch)
plt.title('Birch Dataset1')
plt.subplot(1,2,2)
plt.scatter(X2[:, 0], X2[:, 1], s=10, c=y_birch2)
plt.title('Birch Dataset2')
plt.show()

#DBSCAN
dbscanmodel=DBSCAN(eps=.2, min_samples=70)
y_dbscan=dbscanmodel.fit_predict(X1)
dbscanmodel2=DBSCAN(eps=1,min_samples=10)
y_dbscan2=dbscanmodel2.fit_predict(X2)
plt.figure(figsize=(14,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=y_dbscan)
plt.title('DBSCAN Dataset1')
plt.subplot(1,2,2)
plt.scatter(X2[:, 0], X2[:, 1], s=10, c=y_dbscan2)
plt.title('DBSCAN Dataset2')
plt.show()

# Metrics
data1 = {
    'Model': ['Agnes', 'Birch', 'DBSCAN'],
    'Silhouette Coefficient': [
        silhouette_score(X1, y_agnes),
        silhouette_score(X1, y_birch),
        silhouette_score(X1, y_dbscan)
    ],
    'ARI': [
        adjusted_rand_score(Y1, y_agnes),
        adjusted_rand_score(Y1, y_birch),
        adjusted_rand_score(Y1, y_dbscan)
    ],
    'NMI': [
        normalized_mutual_info_score(Y1, y_agnes),
        normalized_mutual_info_score(Y1, y_birch),
        normalized_mutual_info_score(Y1, y_dbscan)
    ]
}
data2 = {
    'Model': ['Agnes', 'Birch', 'DBSCAN'],
    'Silhouette Coefficient': [
        silhouette_score(X2, y_agnes2),
        silhouette_score(X2, y_birch2),
```

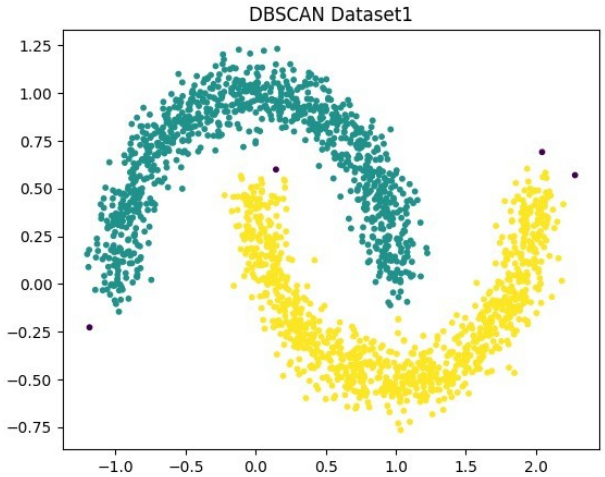
```
        silhouette_score(X2, y_dbscan2)
    ],
    'ARI': [
        adjusted_rand_score(Y2, y_agnes2),
        adjusted_rand_score(Y2, y_birch2),
        adjusted_rand_score(Y2, y_dbscan2)
    ],
    'NMI': [
        normalized_mutual_info_score(Y2, y_agnes2),
        normalized_mutual_info_score(Y2, y_birch2),
        normalized_mutual_info_score(Y2, y_dbscan2)
    ]
}
print('Dataset 1\n', pd.DataFrame(data1).set_index('Model').T, sep='', end='\n\n')
print('Dataset 2\n', pd.DataFrame(data2).set_index('Model').T, sep='')
```

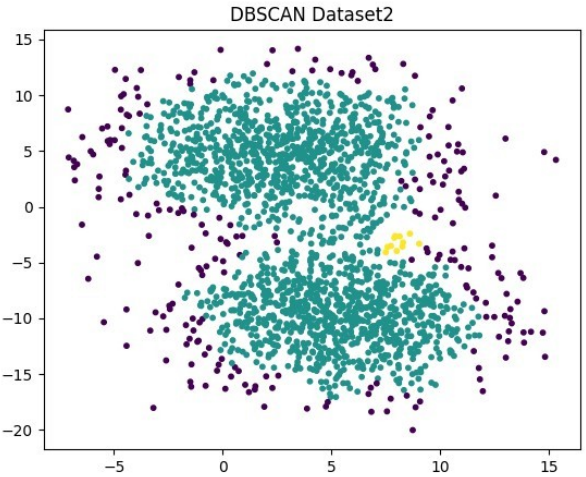
Результат:












```
Dataset 1
Model                        Agnes     Birch    DBSCAN
Silhouette Coefficient   0.406216  0.458350  0.301081
ARI                      0.715577  0.376708  0.992015
NMI                      0.671359  0.341366  0.978765

Dataset 2
Model                        Agnes     Birch    DBSCAN
Silhouette Coefficient   0.587834  0.576088 -0.162604
ARI                      0.908163  0.872292 -0.000164
NMI                      0.842739  0.810245  0.002236
Press any key to continue . . .
```

Виходячи з отриманого, не можна сказати, що можна вибрати найкращу модель, бо якість кластеризації залежить від датасету. AGNES має найкращі показники на 2 датасеті і посередні на першому, Birch - найгірші на датасеті 1 і посередні на датасеті 2, DBSCAN - найкращі на датасеті 1 та найгірші на датасеті 2.

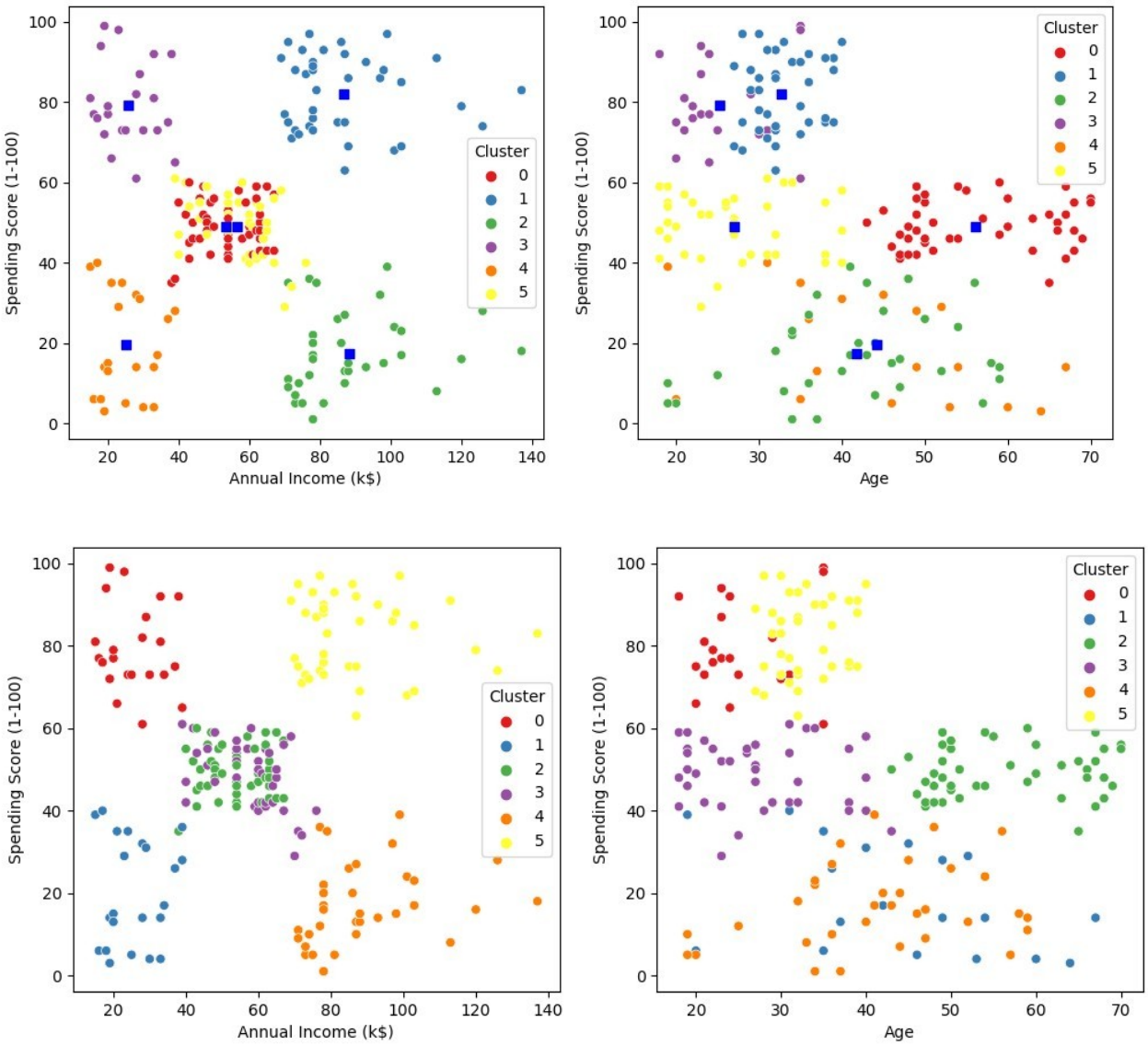
4.**Affinity propagation**.

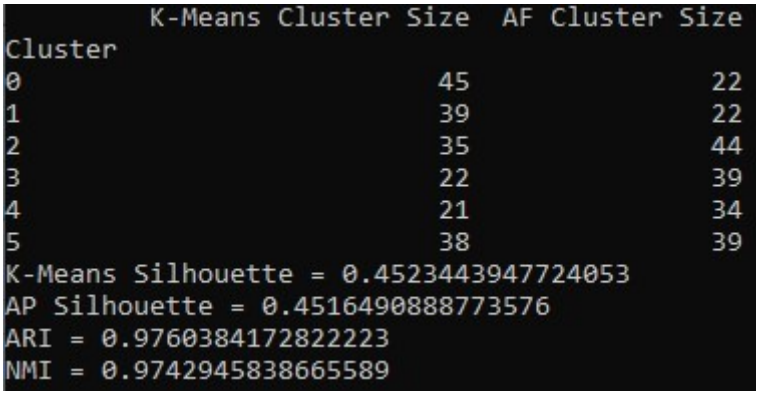
Код програми:

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.cluster import AffinityPropagation
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.cluster import normalized_mutual_info_score
from sklearn.metrics import silhouette_score


mall_data = pd.read_csv("Mall_Customers.csv")
X_numerics = mall_data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]

#KMeans
KM_6_clusters = KMeans(n_clusters=6, init='k-means++').fit(X_numerics)
KM6_clustered = X_numerics.copy()
KM6_clustered.loc[:,'Cluster'] = KM_6_clusters.labels_
fig1, axes = plt.subplots(1,2,figsize=(12,5))
scat_1 = sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)',
data=KM6_clustered,
            hue='Cluster', ax=axes[0], palette='Set1', legend='full')
scat2=sns.scatterplot(x='Age', y='Spending Score (1-100)', data=KM6_clustered,
            hue='Cluster', palette='Set1', ax=axes[1], legend='full')
axes[0].scatter(KM_6_clusters.cluster_centers_[:,1],KM_6_clusters.cluster_centers_[
:,2], marker='s', s=40, c="blue")
axes[1].scatter(KM_6_clusters.cluster_centers_[:,0],KM_6_clusters.cluster_centers_[
:,2], marker='s', s=40, c="blue")
plt.show()

#AP
AP = AffinityPropagation(preference=-11800).fit(X_numerics)
AP_clustered = X_numerics.copy()
AP_clustered.loc[:,'Cluster'] = AP.labels_
fig3, (ax_af) = plt.subplots(1,2,figsize=(12,5))
scat_1 = sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)',
data=AP_clustered,
            hue='Cluster', ax=ax_af[0], palette='Set1', legend='full')
sns.scatterplot(x='Age', y='Spending Score (1-100)', data=AP_clustered,
            hue='Cluster', palette='Set1', ax=ax_af[1], legend='full')
plt.setp(ax_af[0].get_legend().get_texts(), fontsize='10')
plt.setp(ax_af[1].get_legend().get_texts(), fontsize='10')
plt.show()
```

```
#Metrics
KM_sizes=KM6_clustered.groupby('Cluster').size().to_frame()
KM_sizes.columns=["K-Means Cluster Size"]
AP_sizes=AP_clustered.groupby('Cluster').size().to_frame()
AP_sizes.columns=["AF Cluster Size"]
clusters=pd.concat([KM_sizes,AP_sizes],axis=1, sort=False)
print(clusters)

print("K-Means Silhouette =",silhouette_score(X_numerics, KM_6_clusters.labels_))
print("AP Silhouette =",silhouette_score(X_numerics, AP.labels_))
print("ARI =", adjusted_rand_score(KM_6_clusters.labels_, AP.labels_))
print("NMI =", normalized_mutual_info_score(KM_6_clusters.labels_, AP.labels_))
```

Результат:





Cluster	K-Means Cluster Size	AF Cluster Size
0	45	22
1	39	22
2	35	44
3	22	39
4	21	34
5	38	39

K-Means Silhouette = 0.4523443947724053
AP Silhouette = 0.4516490888773576
ARI = 0.9760384172822223
NMI = 0.9742945838665589

Моделі K-Means і Affinity Propagation мають дуже близькі значення метрик Silhouette, а значення ARI та NMI (в цих випадках моделі порівнювалися між собою) наближений до 1, що свідчить про рівність ефективностей цих моделей. Тому, в даному випадку, я б використав Affinity Propagation, бо там не треба вручну визначати оптимальну кількість кластерів.

Висновок:

Отже, у цій роботі я ознайомився з різними методами кластеризації і класифікації. Я дослідив різні метрики, порівняв моделі, та візуалізував кластери. Код програм та результати їх виконання наведені вище.