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.

Імпортуємо бібліотеки

```
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 # for data visualization purposes
import seaborn as sns # for statistical data visualization
%matplotlib inline
```

Завантажуємо датасет

```
data = '/content/1.csv'
df = pd.read_csv(data, header=None, sep=',\s',engine='python')
```

```
pip install category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.2-py2.py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.8/81.8 kB 2.0 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.23.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.11.3)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.14.0)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/dist-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_enco
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.0.5->category_encoders) (202
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_encode
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.20.0->category_
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.10/dist-packages (from statsmodels>=0.9.0->category_encode
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.2
```

Підготовка даних

```
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
# replace '?' values with `NaN`
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']
# split X and y into training and testing sets
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)
# impute missing categorical variables with most frequent value
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
import category_encoders as ce
# encode remaining variables with one-hot encoding
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
from sklearn.preprocessing import RobustScaler
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])
```

Побудова Gaussian Naive Bayes classifier

```
# train a Gaussian Naive Bayes classifier on the training set
from sklearn.naive_bayes import GaussianNB
# instantiate the model
gnb = GaussianNB()
# fit the model
gnb.fit(X_train, y_train)
gnb_pred = gnb.predict(X_test)
gnb_pred
```

```
array(['<=50K', '<=50K', '>50K', ..., '>50K', '<=50K', '<=50K'],
      dtype='<U5')
```

Побудова Support Vector Machine Classifier

```
#Import svm model
from sklearn import svm
#Create a svm Classifier
svm = svm.SVC()
#Train the model using the training sets
svm.fit(X_train, y_train)
#Predict the response for test dataset
svm_pred = svm.predict(X_test)
svm_pred
```

```
array(['<=50K', '<=50K', '<=50K', ..., '>50K', '<=50K', '<=50K'],
      dtype=object)
```

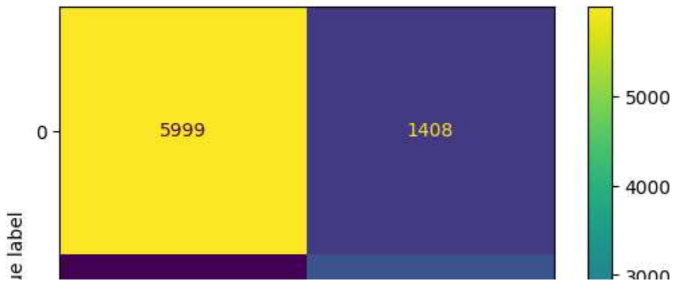
Обчислення метрик Recall, f1-score та accuracy score для моделей

```
from sklearn.metrics import f1_score
f1_gnb=f1_score(y_test,gnb_pred,pos_label='>50K')
f1_svm=f1_score(y_test,svm_pred,pos_label='>50K')
from sklearn.metrics import recall_score
recall_gnb=recall_score(y_test,gnb_pred,pos_label='>50K')
recall_svm=recall_score(y_test,svm_pred,pos_label='>50K')
from sklearn.metrics import accuracy_score
accuracy_gnb=accuracy_score(y_test,gnb_pred)
accuracy_svm=accuracy_score(y_test,svm_pred)
print(f"Bayesian Classification F1 score:{f1_gnb}")
print(f"Support Vector Machine F1 score:{f1_svm}")
print(f"Bayesian Classification Recall score:{recall_gnb}")
print(f"Support Vector Machine Recall score:{recall_svm}")
print(f"Bayesian Classification Accuracy score:{accuracy_gnb}")
print(f"Support Vector Machine Accuracy score:{accuracy_svm}")
```

```
Bayesian Classification F1 score:0.6694900299982354
Support Vector Machine F1 score:0.4082385490316631
Bayesian Classification Recall score:0.8031329381879763
Support Vector Machine Recall score:0.28111769686706184
Bayesian Classification Accuracy score:0.8082710615211383
Support Vector Machine Accuracy score:0.8029481011362473
```

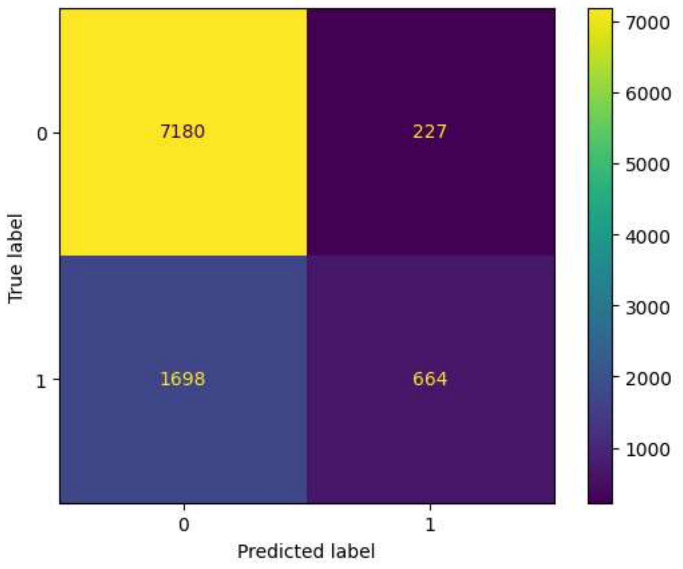
Візуалізація Confusion matrix для Gaussian Naive Bayes classifier

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
confusion_matrix_gnb=confusion_matrix(y_test,gnb_pred)
disp_gnb = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_gnb)
disp_gnb.plot()
plt.show()
```



Візуалізація Confusion matrix для Support Vector Machine Classifier

```
confusion_matrix_svm=confusion_matrix(y_test,svm_pred)
disp_svm = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_svm)
disp_svm.plot()
plt.show()
```



Перевірка на оверфітинг Gaussian Naive Bayes classifier

```
pred_train=gnb.predict(X_train)
print('Train accuracy:'+str(accuracy_score(y_train,pred_train)))
print('Test accuracy:'+str(accuracy_gnb))
```

```
    Train accuracy:0.8067304317304317
    Test accuracy:0.8082710615211383
```

Перевірка на оверфітинг Support Vector Machine Classifier

```
pred_train_svm=svm.predict(X_train)
print('Train accuracy:'+str(accuracy_score(y_train,pred_train_svm)))
print('Test accuracy:'+str(accuracy_svm))
```

```
    Train accuracy:0.8020796770796771
    Test accuracy:0.8029481011362473
```

Порівняння з нуль-гіпотезою

```
from sklearn.dummy import DummyClassifier
# Створення та навчання DummyClassifier зі стратегією "most_frequent"
dummy_model = DummyClassifier(strategy="most_frequent")
dummy_model.fit(X_train, y_train)
dummy_predictions = dummy_model.predict(X_test)
dummy_accuracy = accuracy_score(y_test, dummy_predictions)
print("Точність:", dummy_accuracy)
```

```
    Точність: 0.7582147609786057
```

Висновок

За обчисленими метриками видно,що Gaussian Naive Bayes classifier є кращою моделлю.Точність на тренувальних даних не більша,ніж на тестових в обох моделей,тому оверфітингу немає.

Завдання 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

Імпортуємо бібліотеки

```
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import numpy as np
import matplotlib.ticker as ticker
from sklearn import preprocessing
%matplotlib inline
```

Завантаження,підготовка та розділення даних на тренувальні та тестові

```
df = pd.read_csv('/content/2.csv')
X = df[['region', 'tenure','age', 'marital', 'address', 'income', 'ed', 'employ','retire', 'gender', 'reside']] .values
y = df['custcat'].values
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))
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=4)
```

Побудова моделей K nearest neighbours з метриками Euclidean, Manhattan, Minkowski

```
from sklearn.neighbors import KNeighborsClassifier
k = 4
knn_euclidean = KNeighborsClassifier(n_neighbors = k,metric='euclidean').fit(X_train,y_train)
knn_manhattan=KNeighborsClassifier(n_neighbors = k,metric='manhattan').fit(X_train,y_train)
knn_minkowski=KNeighborsClassifier(n_neighbors = k,metric='minkowski').fit(X_train,y_train)
euc_pred=knn_euclidean.predict(X_test)
manhattan_pred=knn_manhattan.predict(X_test)
minkowski_pred=knn_minkowski.predict(X_test)
```

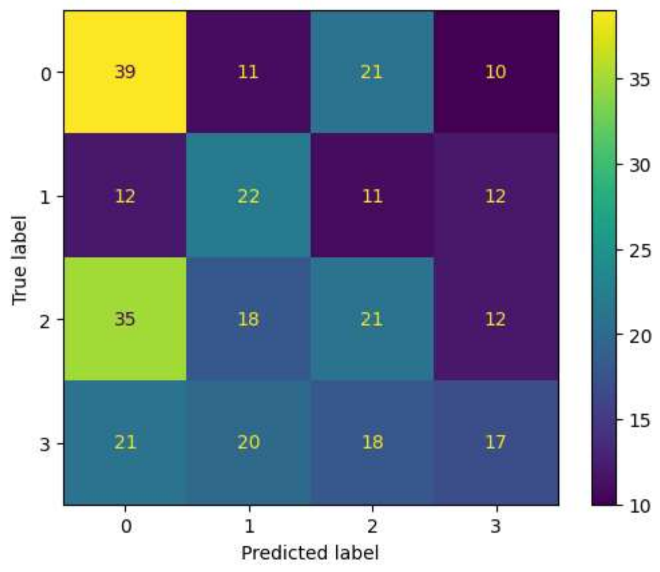
Обчислення метрик Recall, f1-score та accuracy score для моделей

```
from sklearn.metrics import f1_score
f1_euc=f1_score(y_test,euc_pred,average='weighted')
f1_manhattan=f1_score(y_test,manhattan_pred,average='weighted')
f1_minkowski=f1_score(y_test,minkowski_pred,average='weighted')

from sklearn.metrics import recall_score
recall_euc=recall_score(y_test,euc_pred,average='weighted')
recall_manhattan=recall_score(y_test,manhattan_pred,average='weighted')
recall_minkowski=recall_score(y_test,minkowski_pred,average='weighted')

from sklearn.metrics import accuracy_score
accuracy_euc=accuracy_score(y_test,euc_pred)
accuracy_manhattan=accuracy_score(y_test,manhattan_pred)
accuracy_minkowski=accuracy_score(y_test,minkowski_pred)

print(f"Euclidean F1 score:{f1_euc}")
print(f"Manhattan F1 score:{f1_manhattan}")
print(f"Minkowski F1 score:{f1_minkowski}")
print(f"Euclidean Recall score:{recall_euc}")
print(f"Manhattan Recall score:{recall_manhattan}")
print(f"Minkowski Recall score:{recall_minkowski}")
print(f"Euclidean Accuracy score:{accuracy_euc}")
print(f"Manhattan Accuracy score:{accuracy_manhattan}")
print(f"Minkowski Accuracy score:{accuracy_minkowski}")
```

```
    Euclidean F1 score:0.32184319699462793
    Manhattan F1 score:0.2997066229843059
    Minkowski F1 score:0.32184319699462793
    Euclidean Recall score:0.33
    Manhattan Recall score:0.30666666666666664
    Minkowski Recall score:0.33
    Euclidean Accuracy score:0.33
    Manhattan Accuracy score:0.30666666666666664
    Minkowski Accuracy score:0.33
```

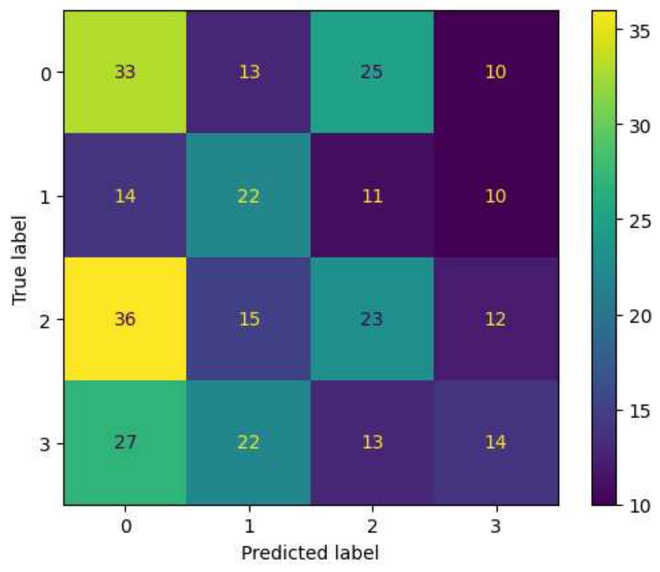
Візуалізація Confusion matrix для Euclidean

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
confusion_matrix_euc=confusion_matrix(y_test,euc_pred)
disp_euc = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_euc)
disp_euc.plot()
plt.show()
```



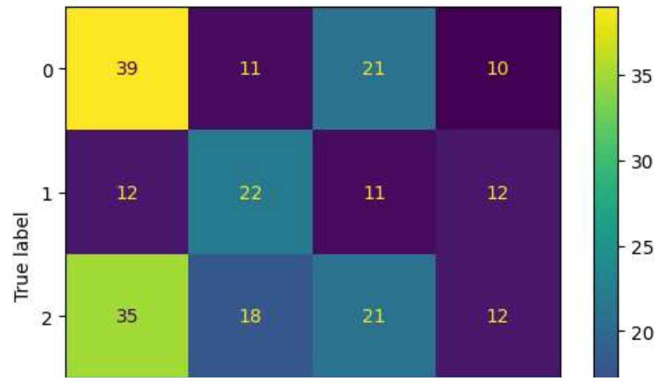
Візуалізація Confusion matrix для Manhattan

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
confusion_matrix_manhattan=confusion_matrix(y_test,manhattan_pred)
disp_manhattan = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_manhattan)
disp_manhattan.plot()
plt.show()
```



Візуалізація Confusion matrix для Minkowski

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
confusion_matrix_minkowski=confusion_matrix(y_test,minkowski_pred)
disp_minkowski = ConfusionMatrixDisplay(confusion_matrix=confusion_matrix_minkowski)
disp_minkowski.plot()
plt.show()
```



Перевірка на оверфітинг Euclidean

```
pred_train_euc=knn_euclidean.predict(X_train)
print('Train accuracy:'+str(accuracy_score(y_train,pred_train_euc)))
print('Test accuracy:'+str(accuracy_euc))
```

```
Train accuracy:0.5371428571428571
Test accuracy:0.33
```

Перевірка на оверфітинг Manhattan

```
pred_train_manhattan=knn_manhattan.predict(X_train)
print('Train accuracy:'+str(accuracy_score(y_train,pred_train_manhattan)))
print('Test accuracy:'+str(accuracy_manhattan))
```

```
Train accuracy:0.54
Test accuracy:0.30666666666666664
```

Перевірка на оверфітинг Minkowski

```
pred_train_minkowski=knn_minkowski.predict(X_train)
print('Train accuracy:'+str(accuracy_score(y_train,pred_train_minkowski)))
print('Test accuracy:'+str(accuracy_minkowski))
```

```
Train accuracy:0.5371428571428571
Test accuracy:0.33
```

Висновок

За обчисленими метриками моделі з Euclidean і Minkowski distance виявилися кращими,їхня точність однакова.Також є оверфітинг.

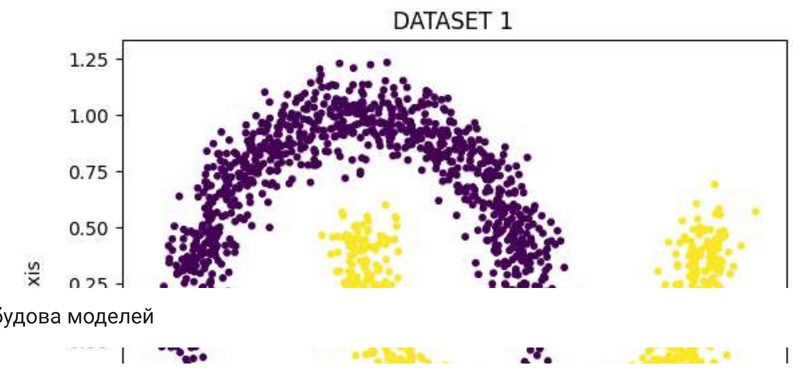
Завдання 3

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.

Імпортуємо бібліотеки і генеруємо дані

```
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], marker='o', c=Y1,s=25, edgecolor='r')
print(X1.shape)
print(Y1.shape)
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=Y1)
plt.title('DATASET 1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
#plt.savefig('Dataset1')

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)
print(X3.shape)
print(Y3.shape)
plt.title('DATASET 2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.scatter(X3[:, 0], X3[:, 1], s=10, c=Y3)
#plt.savefig('Dataset2')
plt.show()
```

```
(2000, 2)
(2000,)
```



Побудова моделей

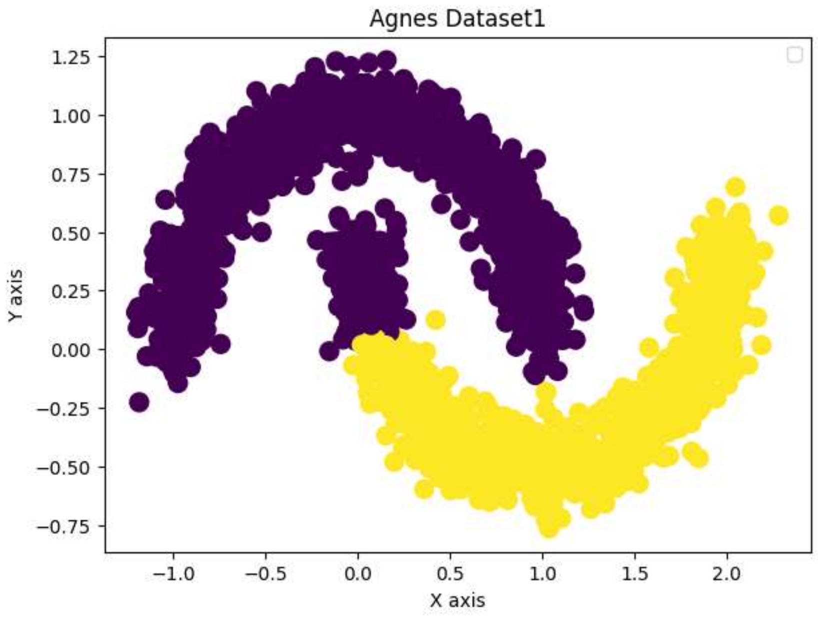
```
from sklearn.cluster import DBSCAN
from sklearn.cluster import Birch
from sklearn.cluster import AgglomerativeClustering
#Model Build dataset1
dbscanmodel=DBSCAN(eps=0.1)
y_dbscan=dbscanmodel.fit_predict(X1)
print(y_dbscan.shape)

birchmodel=Birch(n_clusters=2,threshold=0.5,branching_factor=100)
y_birch=birchmodel.fit_predict(X1)
print(y_birch.shape)

agnesmodel = AgglomerativeClustering(n_clusters=2)
y_agnes=agnesmodel.fit_predict(X1)
print(y_agnes.shape)

#Model Build dataset 2
dbscanmodel2=DBSCAN(eps=0.1)
y_dbscan2=dbscanmodel2.fit_predict(X3)
print(y_dbscan2.shape)

birchmodel2=Birch(n_clusters=2,threshold=0.1,branching_factor=100)
y_birch2=birchmodel2.fit_predict(X3)
print(y_birch2.shape)

agnesmodel2 = AgglomerativeClustering(n_clusters=2)
y_agnes2=agnesmodel2.fit_predict(X3)
print(y_agnes2.shape)
```

```
(2000,)
(2000,)
(2000,)
(2000,)
(2000,)
(2000,)
```

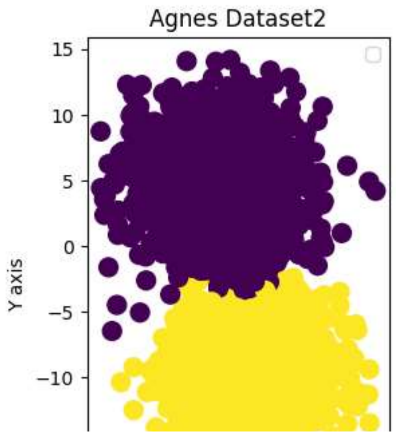
Візуалізація кластерів

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_agnes)
plt.title('Agnes Dataset1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('Kmeansd1',dpi=300)
plt.show()

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.legend()
#plt.savefig('Kmeansd1d2',dpi=300)
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```

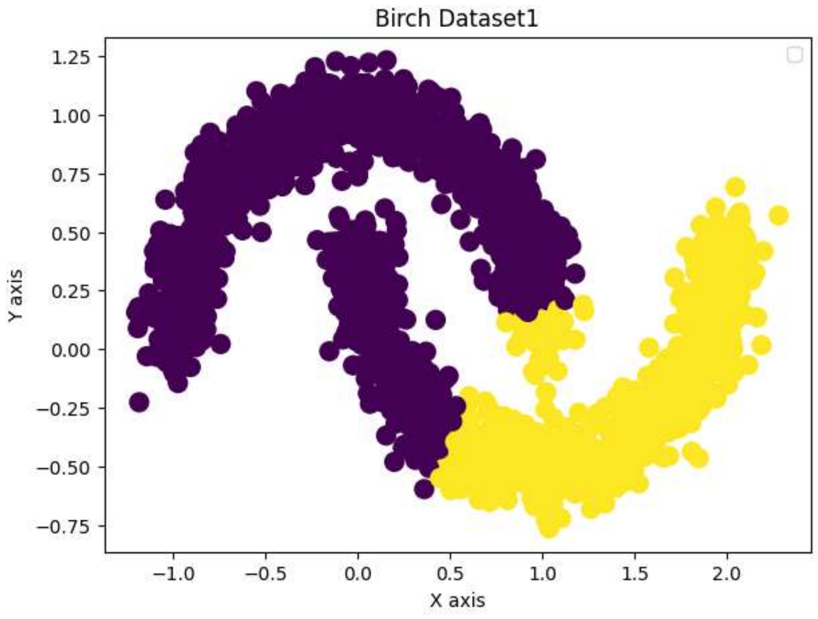


```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```

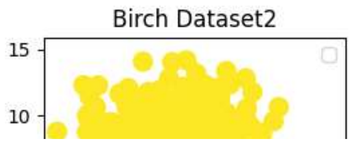


```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_birch)
plt.title('Birch Dataset1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend()
#plt.savefig('Kmeansd1',dpi=300)
plt.show()

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.legend()
#plt.savefig('birchd1d2',dpi=300)
plt.show()
```

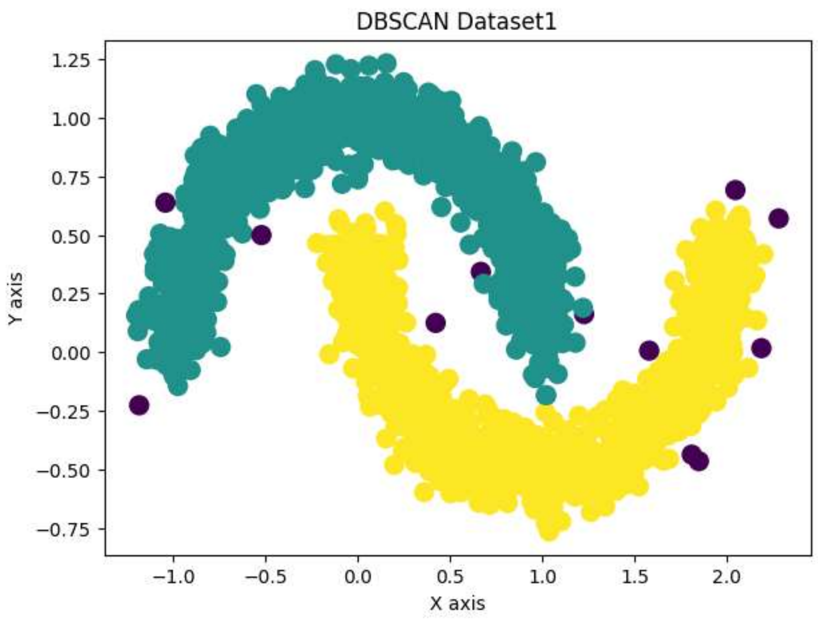
```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```

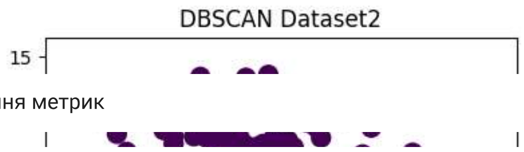


```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists whose label start with an underscore are
```



```
plt.figure(figsize=(15,5))
plt.subplot(1, 2, 1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_dbscan)
plt.title('DBSCAN Dataset1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
plt.figure(figsize=(10,8))
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()
```





Обчислення метрик

```
from sklearn.metrics.cluster import adjusted_rand_score
from sklearn.metrics.cluster import normalized_mutual_info_score
from sklearn.metrics import silhouette_score
#Dataset1
ari_birch=adjusted_rand_score(Y1,y_birch)
ari_dbscan=adjusted_rand_score(Y1,y_dbscan)
ari_agnes=adjusted_rand_score(Y1,y_agnes)
print("DATASET1:")
print("ARI of Birch :"+ str(ari_birch))
print("ARI of Dbscan: "+ str(ari_dbscan))
print("ARI of Agnes: "+ str(ari_agnes))

nmi_birch=normalized_mutual_info_score(Y1,y_birch)
nmi_dbscan=normalized_mutual_info_score(Y1,y_dbscan)
nmi_agnes=normalized_mutual_info_score(Y1,y_agnes)
print("NMI of Birch :"+ str(nmi_birch))
print("NMI of Dbscan: "+ str(nmi_dbscan))
print("NMI of Agnes: "+ str(nmi_agnes))

sil_birch=silhouette_score(X1,y_birch)
sil_dbscan=silhouette_score(X1,y_dbscan)
sil_agnes=silhouette_score(X1,y_agnes)
print("Silhouette Coefficient with Birch :"+ str(sil_birch))
print("Silhouette Coefficient with Dbscan : "+ str(sil_dbscan))
print("Silhouette Coefficient with Agnes : "+ str(sil_agnes))

#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("DATASET2:")
print("ARI of Birch :"+ str(ari_birch))
print("ARI of Dbscan: "+ str(ari_dbscan))
print("ARI of Agnes: "+ str(ari_agnes))

nmi_birch=normalized_mutual_info_score(Y3,y_birch2)
nmi_dbscan=normalized_mutual_info_score(Y3,y_dbscan2)
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))

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))
```

```
DATASET1:
ARI of Birch :0.3767076067566142
ARI of Dbscan: 0.9880329422921506
ARI of Agnes: 0.7155769186783432
NMI of Birch :0.341366173543779
NMI of Dbscan: 0.9713359869748723
NMI of Agnes: 0.6713586477684496
Silhouette Coefficient with Birch :0.45835031870569487
Silhouette Coefficient with Dbscan : 0.2563888544482361
Silhouette Coefficient with Agnes : 0.40621615915378817
DATASET2:
ARI of Birch :0.872292314560211
ARI of Dbscan: 2.0010104852824367e-05
ARI of Agnes: 0.90816307882887
NMI of Birch :0.8102453395167878
NMI of Dbscan: 0.004885857858962106
NMI of Agnes: 0.8427393441408568
Silhouette Coefficient with Birch :0.5760880178842558
Silhouette Coefficient with Dbscan : -0.17834294700481815
Silhouette Coefficient with Agnes : 0.5878339188420266
```

Висновок

За обчисленими метриками можна стверджувати, що якість моделі залежить від датасету. Birch та Agnes набагато краще спрацювали на другому датасеті,DBSCAN - на першому.

Завдання 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

Імпортуємо бібліотеки

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

Завантажуємо датасет

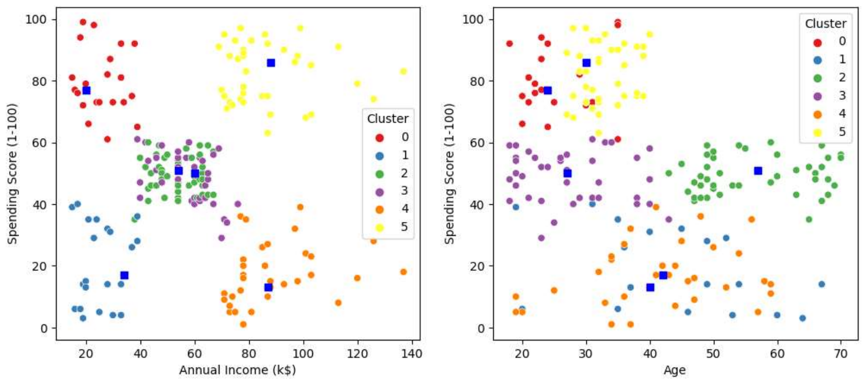
```
mall_data = pd.read_csv('/content/4.csv')
```

Кластеризація Affinity propagation

```
from sklearn.cluster import AffinityPropagation
X_numerics = mall_data[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']] # subset with numeric variables only
AF = AffinityPropagation(preference=-11800).fit(X_numerics)
AF_clustered = X_numerics.copy()
AF_clustered.loc[:,'Cluster'] = AF.labels_ # append labels to points
```

Візуалізація кластерів

```
fig11, (axes) = plt.subplots(1,2,figsize=(12,5))

sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', data=AF_clustered,hue='Cluster', ax=axes[0], palette='Set1', legend=

sns.scatterplot(x='Age', y='Spending Score (1-100)', data=AF_clustered,hue='Cluster', palette='Set1', ax=axes[1], legend='full')

# plotting centroids
axes[0].scatter(AF.cluster_centers_[:,1], AF.cluster_centers_[:,2], marker='s', s=40, c="blue")
axes[1].scatter(AF.cluster_centers_[:,0], AF.cluster_centers_[:,2], marker='s', s=40, c="blue")
plt.show()
```

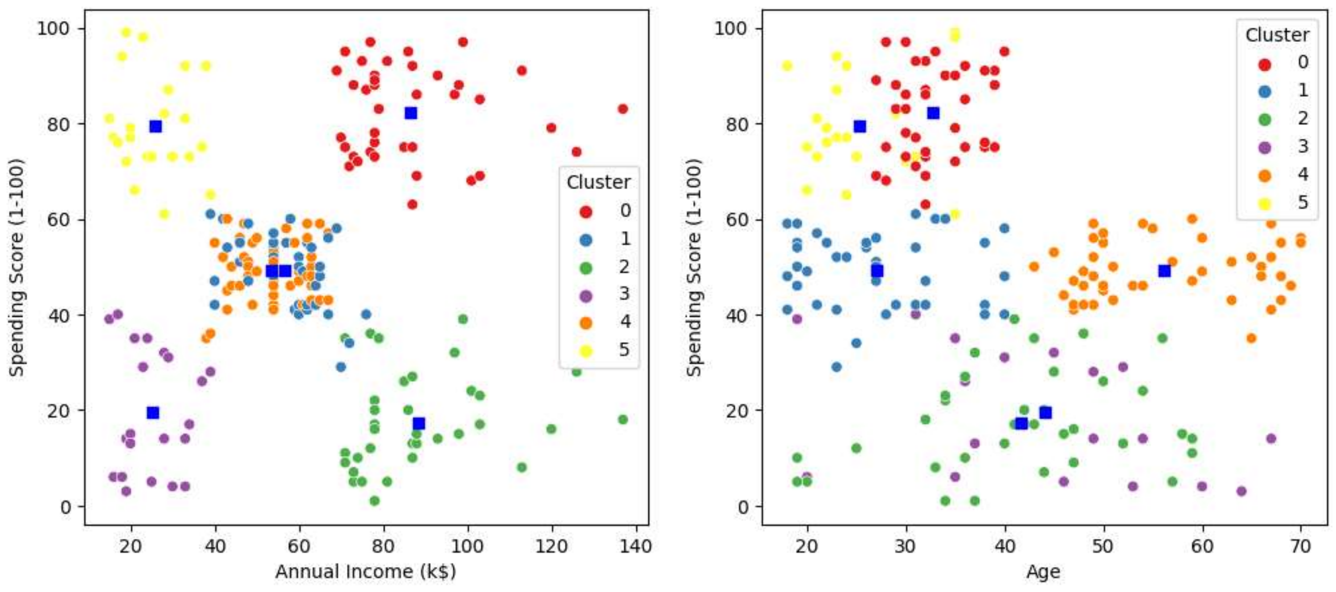


Кластеризація k-means

```
from sklearn.cluster import KMeans
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
```

Візуалізація кластерів

```
fig11, (axes) = plt.subplots(1,2,figsize=(12,5))

sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', data=KM6_clustered,hue='Cluster', ax=axes[0], palette='Set1', legend=

sns.scatterplot(x='Age', y='Spending Score (1-100)', data=KM6_clustered,hue='Cluster', palette='Set1', ax=axes[1], legend='full')

# plotting centroids
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()
```



Обчислення ARI, NMI для Affinity propagation

```
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics import normalized_mutual_info_score

ari=adjusted_rand_score(KM_6_clusters.labels_, AF.labels_)
nmi=normalized_mutual_info_score(KM_6_clusters.labels_, AF.labels_)
print("ARI of Affinity propagation :"+ str(ari))
print("NMI of Affinity propagation : "+ str(nmi))
```

```
ARI of Affinity propagation :0.9760384172822223
NMI of Affinity propagation : 0.974294583866559
```

Обчислення Silhouette Coefficient для Affinity propagation та K-means

```
from sklearn.metrics import silhouette_score

af_sil=silhouette_score(X_numerics, AF.labels_)
km_sil=silhouette_score(X_numerics, KM_6_clusters.labels_)

print("Silhouette Coefficient with Affinity propagation :"+ str(af_sil))
print("Silhouette Coefficient with K-means : "+ str(km_sil))
```

```
Silhouette Coefficient with Affinity propagation :0.4516490888773576
Silhouette Coefficient with K-means : 0.4523443947724053
```

Висновок

Обидві моделі майже однаково добре спрацювали на датасеті.