

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 №1  
з дисципліни  
“Програмування інтелектуальних інформаційних систем”

**Виконала**

ІІ-21 Скрипець О. О.

**Перевірив**

Баришич Л. М.

Київ 2024

## Завдання

Метрики і спосіб виконання описані тут:

<https://www.kaggle.com/code/prashant111/naive-bayes-classifier-in-python>

Лабу можна виконати в онлайн-редакторах типу Google Collab.

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 оверфітінг. 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

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
data = "/content/drive/My Drive/data/adult.csv"
df = pd.read_csv(data, header=None, sep=',\s')
```

Mounted at /content/drive  
 <ipython-input-2-b79f400d97d3>:6: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex :  
 df = pd.read\_csv(data, header=None, sep=',\s')

```
df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	United-States	<=50K

Подальші дії:

[Переглянути рекомендовані графіки](#)

[New interactive sheet](#)

Додам гарні назви до стовпців

```
column_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 = column_names
df.columns
```

```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education_num',
'marital_status', 'occupation', 'relationship', 'race', 'sex',
'capital_gain', 'capital_loss', 'hours_per_week', 'native_country',
'income'],
dtype='object')
```

Загальна інформація

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                   32561 non-null  int64
1   workclass             32561 non-null  object
2   fnlwgt                32561 non-null  int64
3   education             32561 non-null  object
4   education_num         32561 non-null  int64
5   marital_status        32561 non-null  object
6   occupation            32561 non-null  object
7   relationship          32561 non-null  object
8   race                  32561 non-null  object
9   sex                   32561 non-null  object
10  capital_gain          32561 non-null  int64
11  capital_loss          32561 non-null  int64
12  hours_per_week        32561 non-null  int64
13  native_country        32561 non-null  object
14  income                32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

У цьому наборі даних присутні як категоріальні, так і числові змінні. Категоріальні змінні представлені типом даних `object`, тоді як числові змінні мають тип `int64`. Спершу розглянемо категоріальні змінні.

```
categorical_vars = [col for col in df.columns if df[col].dtype == 'O']
print(f'У наборі даних {len(categorical_vars)} категоріальних змінних.\n')
print('Перелік категоріальних змінних:\n', categorical_vars)
```

У наборі даних 9 категоріальних змінних.

Перелік категоріальних змінних:  
['workclass', 'education', 'marital\_status', 'occupation', 'relationship', 'race', 'sex', 'native\_country', 'income']

```
df[categorical_vars].head()
```

	workclass	education	marital_status	occupation	relationship	race	sex	native_country	income
0	State-gov	Bachelors	Never-married	Adm-clerical	Not-in-family	White	Male	United-States	<=50K
1	Self-emp-not-inc	Bachelors	Married-civ-spouse	Exec-managerial	Husband	White	Male	United-States	<=50K
2	Private	HS-grad	Divorced	Handlers-cleaners	Not-in-family	White	Male	United-States	<=50K
3	Private	11th	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	United-States	<=50K
4	Private	Bachelors	Married-civ-spouse	Prof-specialty	Wife	Black	Female	Cuba	<=50K

```
df[categorical_vars].isnull().sum()
```

	0
workclass	1836
education	0
marital_status	0
occupation	1843
relationship	0
race	0
sex	0
native_country	583
income	0
dtype:	int64

```
for var in categorical_vars:
    print(df[var].value_counts())
```



```

name
Iran                44
Portugal            37
Nicaragua            34
Peru                 31
France              29
Greece              29
Ecuador             28
Ireland             24
Hong                20
Cambodia            19
Trinidad&Tobago     19
Laos                18
Thailand            18
Yugoslavia          16
Outlying-US(Guam-USVI-etc) 14
Honduras            13
Hungary             13
Scotland            12
Holand-Netherlands  1
Name: count, dtype: int64
income
<=50K      24720
>50K       7841
Name: count, dtype: int64


```

Ми виявили кілька змінних, таких як `workclass`, `occupation` і `native_country`, які мають пропущені значення у вигляді "?". Зазвичай відсутні значення позначаються як NaN, і Python їх розпізнає за допомогою команди `df.isnull().sum()`. Проте в даному випадку пропуски позначені знаком "?". Оскільки Python не сприймає цей символ як пропущене значення, його потрібно замінити на NaN для правильної обробки.

```

df['workclass'].replace('?', np.nan, inplace=True)
df['occupation'].replace('?', np.nan, inplace=True)
df['native_country'].replace('?', np.nan, inplace=True)

```

 <ipython-input-21-6c689aab7a91>:1: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

```
df['workclass'].replace('?', np.nan, inplace=True)
```

<ipython-input-21-6c689aab7a91>:2: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

```
df['occupation'].replace('?', np.nan, inplace=True)
```

<ipython-input-21-6c689aab7a91>:3: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col]

```
df['native_country'].replace('?', np.nan, inplace=True)
```

тепер те ж саме повторюю числовими змінними

```

for var in categorical_vars:
    print(df[var].value_counts())

```



```

Cuba                95
England             90
Jamaica             81
South              80
China              75
Italy              73
Dominican-Republic 70
Vietnam            67
Guatemala          64
Japan              62
Poland             60
Columbia           59
Taiwan            51
Haiti              44
Iran               43
Portugal           37
Nicaragua          34
Peru               31
France            29
Greece            29
Ecuador           28
Ireland           24
Hong               20
Cambodia           19
Trinidad&Tobago    19
Laos               18
Thailand           18
Yugoslavia         16
Outlying-US(Guam-USVI-etc) 14
Honduras           13
Hungary            13
Scotland           12
Holand-Netherlands 1
Name: count, dtype: int64
income
<=50K      24720
>50K       7841
Name: count, dtype: int64

```

```

numerical_vars = [col for col in df.columns if df[col].dtype != 'O']
print(f'У наборі даних {len(numerical_vars)} числових змінних.')
print('Перелік числових змінних:', numerical_vars)

```

```

→ У наборі даних 6 числових змінних.
Перелік числових змінних: ['age', 'fnlwgt', 'education_num', 'capital_gain', 'capital_loss', 'hours_per_week']

```

```
df[numerical_vars].head()
```

```

→
   age  fnlwgt  education_num  capital_gain  capital_loss  hours_per_week
0   39   77516             13           2174             0             40
1   50   83311             13              0             0             13
2   38  215646              9              0             0             40
3   53  234721              7              0             0             40
4   28  338409             13              0             0             40

```

```
df[numerical_vars].isnull().sum()
```

```

→
      0
age    0
fnlwgt  0
education_num  0
capital_gain  0
capital_loss  0
hours_per_week  0

dtype: int64

```

Ми видаляємо стовпець 'income' з основного набору даних для формування матриці ознак  $X$ , тоді як 'income' стає цільовою змінною  $y$ . У  $X$  зберігаються всі ознаки, що використовуються для прогнозування, а у  $y$  містять значення, які модель повинна передбачити.

```

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

```

Розділяємо дані на окремі навчальні та тестові набори

```
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)
```

Оскільки моделі машинного навчання переважно працюють із числовими даними, необхідно перетворити категоріальні змінні на числові значення.

```
from sklearn.preprocessing import LabelEncoder
label_enc = LabelEncoder()
for col in categorical_vars:
    if col in X_train.columns:
        X_train[col] = label_enc.fit_transform(X_train[col])
        X_test[col] = label_enc.transform(X_test[col])
```

Наступним кроком є стандартизація даних, яка гарантує, що числові ознаки мають однаковий масштаб. Це позитивно впливає на продуктивність багатьох моделей машинного навчання, особливо для алгоритмів, таких як SVM, де масштаби ознак можуть суттєво впливати на результати.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train[numerical_vars] = scaler.fit_transform(X_train[numerical_vars])
X_test[numerical_vars] = scaler.transform(X_test[numerical_vars])
```

## ✓ Bayesian Classification + Support Vector Machine

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import recall_score, f1_score, accuracy_score, confusion_matrix
```

Ініціалізація моделі Naive Bayes, тренування та прогнозування

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
```

Обчислення метрик для оцінки моделі

```
recall_gnb = recall_score(y_test, y_pred_gnb, pos_label="<=50K")
f1_gnb = f1_score(y_test, y_pred_gnb, pos_label="<=50K")
accuracy_gnb = accuracy_score(y_test, y_pred_gnb)
```

Виведення результатів моделі Naive Bayes

```
print(f'Naive Bayes. Recall: {recall_gnb:.4f}')
print(f'Naive Bayes. F1: {f1_gnb:.4f}')
print(f'Naive Bayes. Accuracy: {accuracy_gnb:.4f}')
```

```
→ Naive Bayes. Recall: 0.9532
Naive Bayes. F1: 0.8815
Naive Bayes. Accuracy: 0.8056
```

На основі результатів:

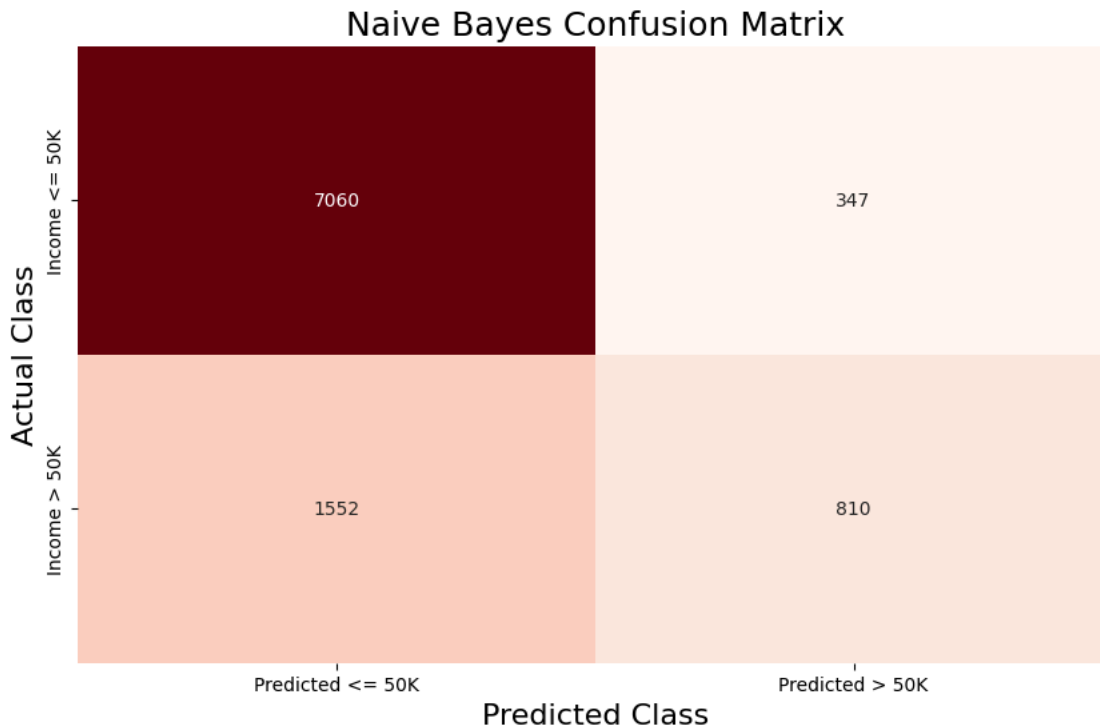
- Recall: 0.9532, що вказує на те, що модель добре розпізнає позитивний клас "<=50K".
- F1-Score: 0.8815, що свідчить про гарний баланс між точністю та повнотою.
- Accuracy: 0.8056, що означає, що модель правильно класифікує приблизно 81% прикладів.

Побудова матриці невідповідностей (Confusion Matrix)

```

cm_gnb = confusion_matrix(y_test, y_pred_gnb)
cm_gnb_df = pd.DataFrame(cm_gnb, index=['Income <= 50K', 'Income > 50K'], columns=['Predicted <= 50K', 'Predicted > 50K'])
plt.figure(figsize=(10,6))
sns.heatmap(cm_gnb_df, annot=True, fmt='g', cmap='Reds', cbar=False)
plt.title('Naive Bayes Confusion Matrix', fontsize=18)
plt.xlabel('Predicted Class', fontsize=16)
plt.ylabel('Actual Class', fontsize=16)
plt.show()

```



Модель загалом демонструє високу точність у розпізнаванні класу "<=50K", проте має певні труднощі з правильною класифікацією менш представленого класу ">50K".

Оцінка продуктивності моделі на навчальних і тестових наборах

```

train_score_gnb = gnb.score(X_train, y_train)
test_score_gnb = gnb.score(X_test, y_test)
null_accuracy_gnb = y_test.value_counts().max() / y_test.value_counts().sum()

```

```

print(f'Оцінка на навчальному наборі: {train_score_gnb:.4f}')
print(f'Оцінка на тестовому наборі: {test_score_gnb:.4f}')
print(f'Нульова точність: {null_accuracy_gnb:.4f}')

```



```

Оцінка на навчальному наборі: 0.8035
Оцінка на тестовому наборі: 0.8056
Нульова точність: 0.7582

```

Аналіз оверфітінгу

```

if train_score_gnb > test_score_gnb:
    print('Модель Naive Bayes можливо страждає від оверфітінгу.')
else:
    print('Модель Naive Bayes збалансована.')

```



```

Модель Naive Bayes збалансована.

```

Модель виглядає збалансованою, оскільки результати навчального і тестового наборів дуже близькі, і точність значно вища за нульову.



## ✓ Ініціалізація SVM-моделі, навчання та прогнозування

```
from sklearn.svm import SVC
svc = SVC(kernel='linear')
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)
```

Обчислення метрик для оцінки SVM

```
recall_svc = recall_score(y_test, y_pred_svc, pos_label("<=50K"))
f1_svc = f1_score(y_test, y_pred_svc, pos_label("<=50K"))
accuracy_svc = accuracy_score(y_test, y_pred_svc)
```

```
print(f'SVM. Recall: {recall_svc:.4f}')
print(f'SVM. F1: {f1_svc:.4f}')
print(f'SVM. Accuracy: {accuracy_svc:.4f}')
```

```
↗ SVM. Recall: 0.9733
SVM. F1: 0.8894
SVM. Accuracy: 0.8165
```

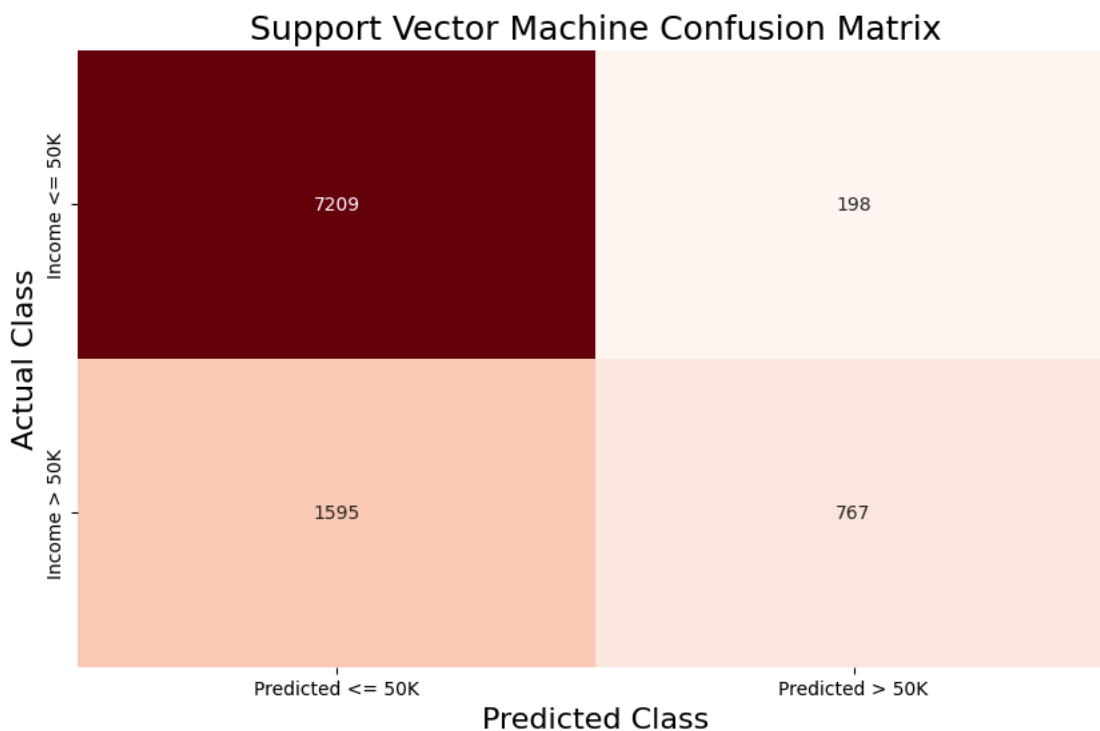
Результати SVM показують:

- Recall: 0.9731 — чудова здатність моделі розпізнавати клас "<=50K".
- F1-Score: 0.8893 — вказує на добре збалансовану продуктивність між точністю та відзивом.
- Accuracy: 0.8164 — трохи краща за Naive Bayes.

Матриця невідповідностей для SVM

```
cm_svc = confusion_matrix(y_test, y_pred_svc)
cm_svc_df = pd.DataFrame(cm_svc, index=['Income <= 50K', 'Income > 50K'], columns=['Predicted <= 50K', 'Predicted > 50K'])
plt.figure(figsize=(10,6))
sns.heatmap(cm_svc_df, annot=True, fmt='g', cmap='Reds', cbar=False)
plt.title('Support Vector Machine Confusion Matrix', fontsize=18)
plt.xlabel('Predicted Class', fontsize=16)
plt.ylabel('Actual Class', fontsize=16)
plt.show()
```

↗



Оцінка продуктивності на навчальних і тестових наборах для SVM

```
train_score_svc = svc.score(X_train, y_train)
test_score_svc = svc.score(X_test, y_test)
null_accuracy_svc = y_test.value_counts().max() / y_test.value_counts().sum()
```

```
print(f'Оцінка на навчальному наборі (SVM): {train_score_svc:.4f}')
print(f'Оцінка на тестовому наборі (SVM): {test_score_svc:.4f}')
print(f'Нульова точність (SVM): {null_accuracy_svc:.4f}')
```

```
↗ Оцінка на навчальному наборі (SVM): 0.8154
Оцінка на тестовому наборі (SVM): 0.8165
Нульова точність (SVM): 0.7582
```

Перевірка на оверфітинг для SVM

```
if train_score_svc > test_score_svc:
    print('Модель SVM можливо страждає від оверфітингу.')
else:
    print('Модель SVM виглядає збалансованою.')
```

```
↗ Модель SVM виглядає збалансованою.
```

## Результати

### Naive Bayes:

- Оцінка на навчальному наборі: 0.8035
- Оцінка на тестовому наборі: 0.8056
- Оцінка нульової точності: 0.7582
- Recall: 0.9532
- F1-score: 0.8815
- Accuracy: 0.8056

### Support Vector Machine (SVM):

- Оцінка на навчальному наборі: 0.8154
- Оцінка на тестовому наборі: 0.8164
- Оцінка нульової точності: 0.7582
- Recall: 0.9731
- F1-score: 0.8893
- Accuracy: 0.8164

1. Точність на тестовому наборі: Обидві моделі демонструють схожу точність, проте SVM дещо перевершує Naive Bayes на навчальному (0.8154 проти 0.8035) та тестовому наборах (0.8164 проти 0.8056). Це свідчить про кращу здатність SVM узагальнювати на нових даних.
2. Recall: SVM має вищий Recall (0.9731 проти 0.9532), що свідчить про його кращу здатність виявляти позитивні випадки (доходи ">50K"). Якщо важливо не пропустити такі випадки, модель SVM може бути більш підходящою.
3. F1-score: F1-score у SVM також вищий (0.8893 проти 0.8815), що вказує на кращий баланс між точністю і Recall. Це говорить про те, що SVM краще підходить для задачі з точки зору загальної ефективності.
4. Загальна точність: SVM показує трохи вищу точність, що свідчить про його загальну перевагу над Naive Bayes за всіма метриками.

## Висновок

SVM демонструє кращі результати за всіма основними показниками (Recall, F1-score та Accuracy), що робить його більш надійним варіантом для цієї задачі. Модель SVM краще справляється з виявленням позитивних випадків і має кращу здатність до узагальнення на нових даних. Хоча Naive Bayes також показує непогані результати, його показники, особливо Recall і F1-score, дещо поступаються SVM. Враховуючи це, SVM може бути кращим вибором для розв'язання цієї задачі.

## ✓ K-Nearest Neighbors

Імпортую необхідну бібліотеку

```
from sklearn import preprocessing
```

завантажую дані

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
data2 = "/content/drive/My_Drive/data/teleCust1000t.csv"
df = pd.read_csv(data2, header=0, sep=',', engine='python')
```

Mounted at /content/drive

```
df.head()
```

↗

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

Подальші дії:

[Переглянути рекомендовані графіки](#)

[New interactive sheet](#)

Кількість зразків у кожному класі

```
df['custcat'].value_counts()
```

↗

custcat	count
3	281
1	266
4	236
2	217

dtype: int64

```
df.columns
```

↗

```
Index(['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed',
      'employ', 'retire', 'gender', 'reside', 'custcat'],
      dtype='object')
```

Підготовка даних X - ознаки, y - цільові мітки

```
X = df[['region', 'tenure', 'age', 'marital', 'address', 'income', 'ed', 'employ', 'retire', 'gender', 'reside']].values
y = df['custcat'].values
```

Стандартизуємо дані та розділяємо їх на навчальну та тестову вибірки

```
scaler = preprocessing.StandardScaler().fit(X)
X_scaled = scaler.transform(X.astype(float))
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

```
print(f'Розмір навчальної вибірки: {X_train.shape}, {y_train.shape}')
print(f'Розмір тестової вибірки: {X_test.shape}, {y_test.shape}')
```

↗

```
Розмір навчальної вибірки: (800, 11), (800,)
Розмір тестової вибірки: (200, 11), (200,)
```

Дані були успішно розподілені, у навчальному наборі - 800 зразків, в тестовому - 200. Кожен зразок має 11 ознак

## ✓ PCA

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

Підготовка для моделі KNN

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import recall_score, f1_score, accuracy_score, classification_report, confusion_matrix
import time
```

Список метрик для порівняння

```
metrics = ['euclidean', 'manhattan', 'minkowski']
results = {}
```

Тестуємо модель для різних метрик

```
for metric in metrics:
    print(f'\nРезультати для метрики {metric.capitalize()}:')

    knn = KNeighborsClassifier(n_neighbors=7, metric=metric)

    # Початок та завершення часу навчання
    start = time.time()
    knn.fit(X_train, y_train)
    end = time.time()

    y_pred = knn.predict(X_test)

    # Розрахунок основних метрик
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    accuracy = accuracy_score(y_test, y_pred)

    # Зберігаємо результати
    results[metric] = {
        'recall': recall,
        'f1': f1,
        'accuracy': accuracy,
        'training_time': end - start
    }

    print(f'Повнота (Recall): {recall:.4f}')
    print(f'F1-оцінка: {f1:.4f}')
    print(f'Точність (Accuracy): {accuracy:.4f}')
    print(f'Час навчання: {end - start:.4f} сек')

    # Звіт класифікації
    print(classification_report(y_test, y_pred))

    # Відображення матриці помилок
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(8,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
    plt.title(f'Матриця помилок (KNN, {metric.capitalize()})')
    plt.xlabel('Прогнозована категорія', fontsize=12)
    plt.ylabel('Реальна категорія', fontsize=12)
    plt.show()

    # Візуалізація кластерів на PCA-компонентах
    plt.figure(figsize=(10,7))
    plt.scatter(X_test_pca[:, 0], X_test_pca[:, 1], c=y_pred, cmap='plasma', marker='s', edgecolor='k', alpha=0.7)
    plt.title(f'Кластери (KNN, {metric.capitalize()})', fontsize=18)
    plt.xlabel('PCA Компонента 1', fontsize=14)
    plt.ylabel('PCA Компонента 2', fontsize=14)
    plt.colorbar(label='Прогнозована категорія')
    plt.show()
```



Результати для метрики Euclidean:

Повнота (Recall): 0.3105

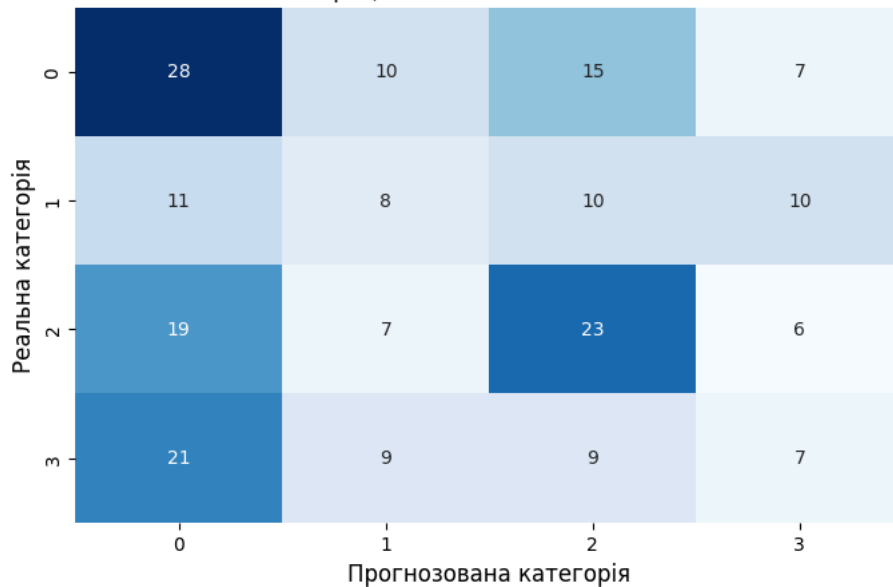
F1-оцінка: 0.3042

Точність (Accuracy): 0.3300

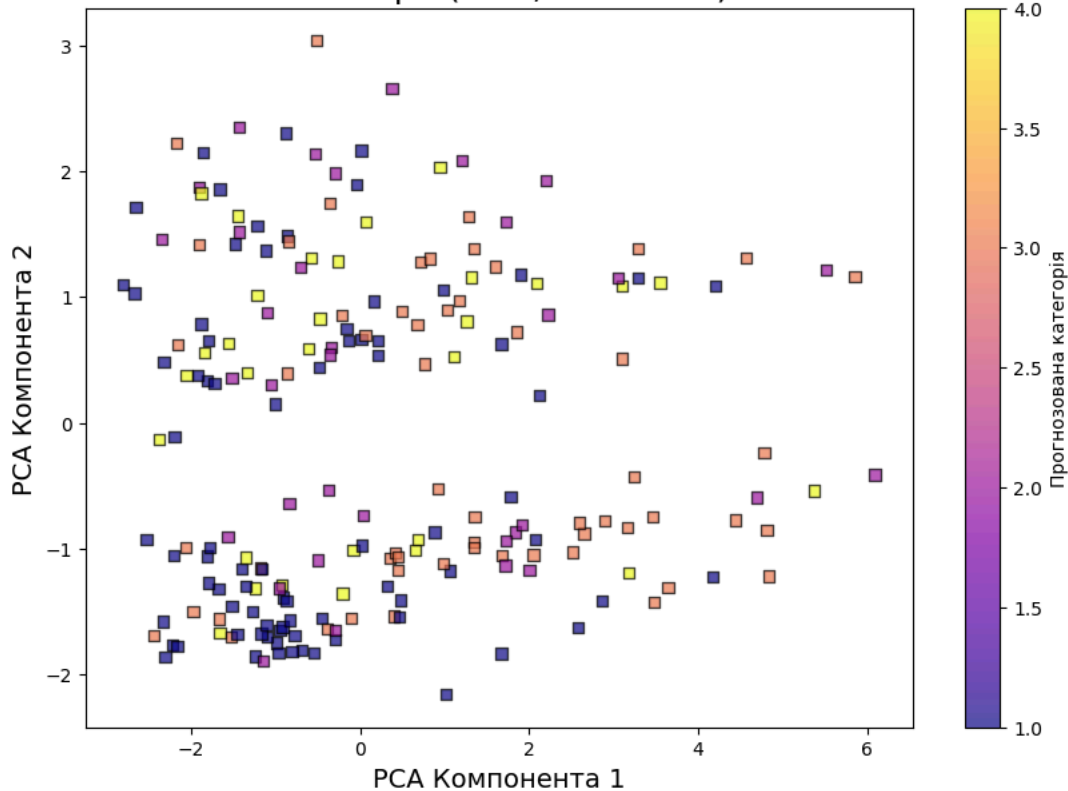
Час навчання: 0.0082 сек

	precision	recall	f1-score	support
1	0.35	0.47	0.40	60
2	0.24	0.21	0.22	39
3	0.40	0.42	0.41	55
4	0.23	0.15	0.18	46
accuracy			0.33	200
macro avg	0.31	0.31	0.30	200
weighted avg	0.32	0.33	0.32	200

Матриця помилок (KNN, Euclidean)



Кластери (KNN, Euclidean)



Результати для метрики Manhattan:

Повнота (Recall): 0.2903

F1-оцінка: 0.2893

Точність (Accuracy): 0.3050

Час навчання: 0.0033 сек

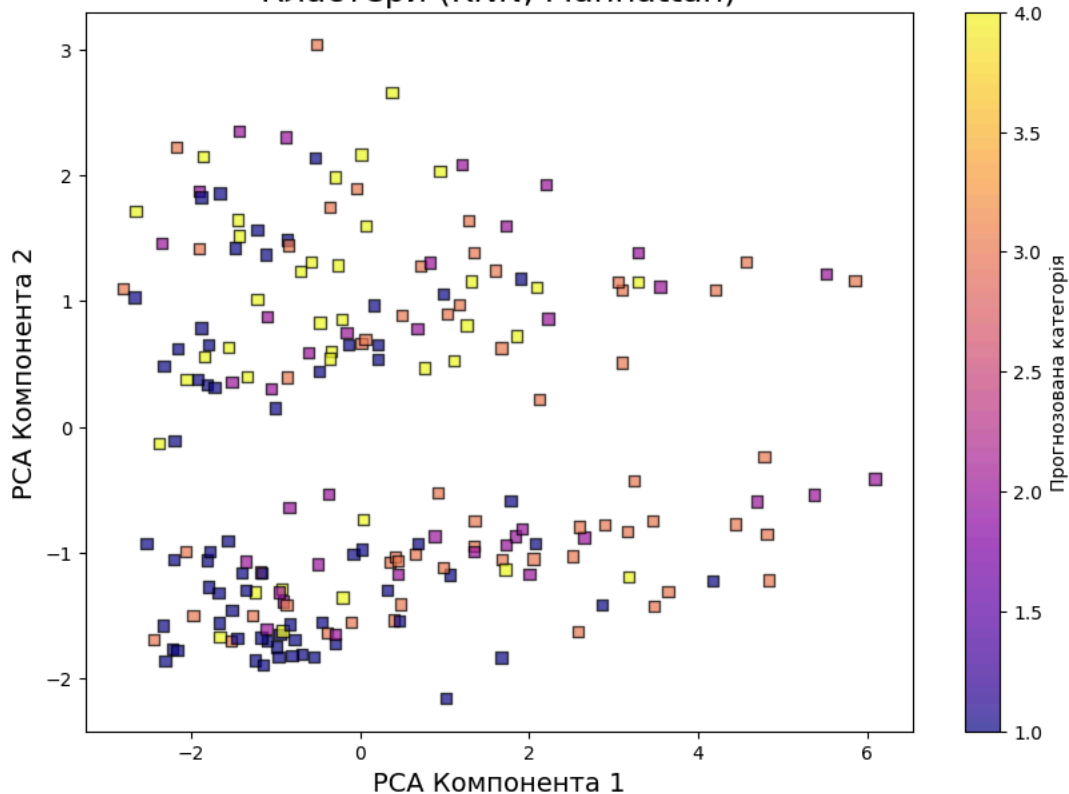
	precision	recall	f1-score	support
1	0.31	0.35	0.33	60

2	0.16	0.15	0.16	39
3	0.39	0.42	0.40	55
4	0.30	0.24	0.27	46
accuracy			0.30	200
macro avg	0.29	0.29	0.29	200
weighted avg	0.30	0.30	0.30	200

Матриця помилок (KNN, Manhattan)



Кластери (KNN, Manhattan)



Результати для метрики Minkowski:

Повнота (Recall): 0.3105

F1-оцінка: 0.3042

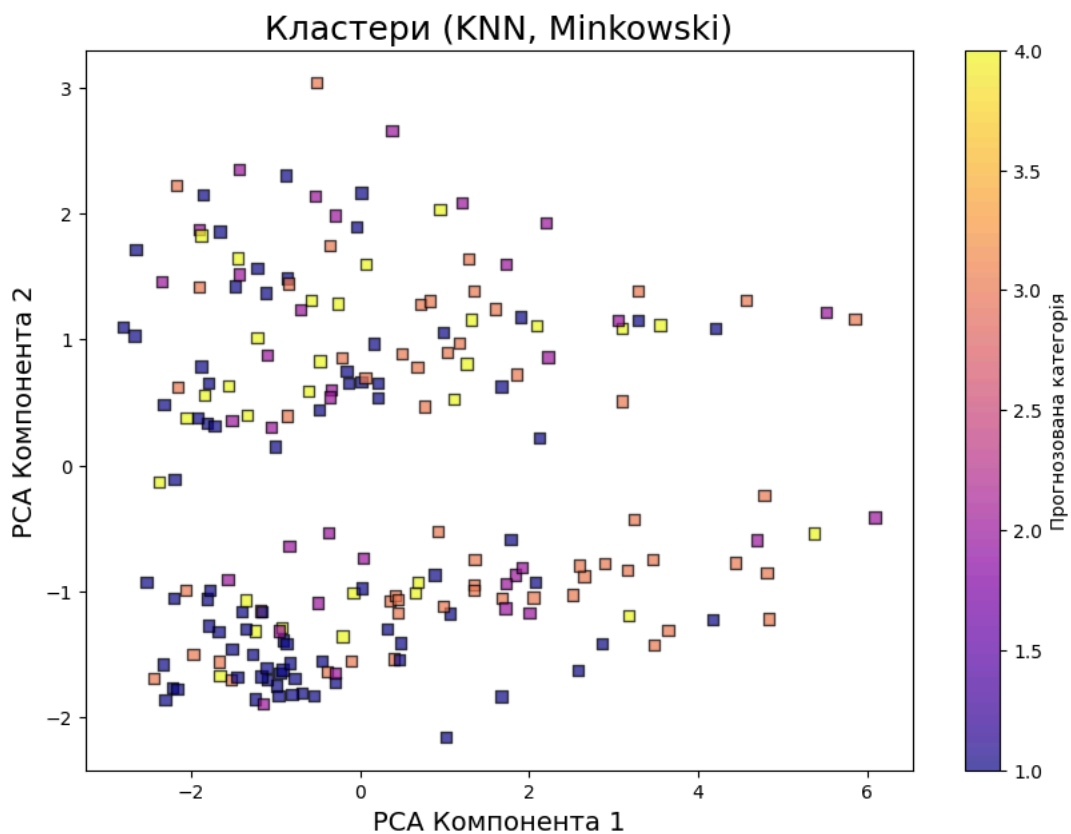
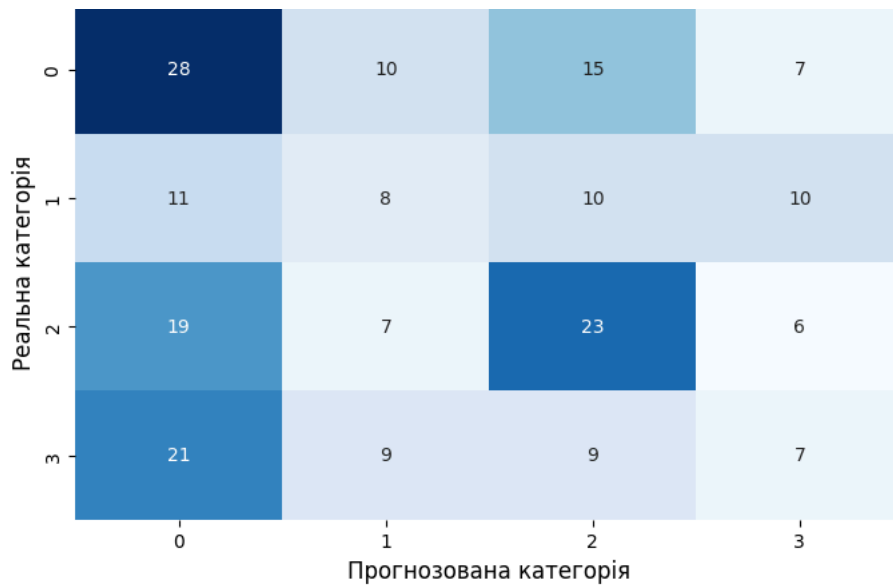
Точність (Accuracy): 0.3300

Час навчання: 0.0027 сек

	precision	recall	f1-score	support
1	0.35	0.47	0.40	60
2	0.24	0.21	0.22	39
3	0.40	0.42	0.41	55
4	0.23	0.15	0.18	46

accuracy			0.33	200
macro avg	0.31	0.31	0.30	200
weighted avg	0.32	0.33	0.32	200

Матриця помилок (KNN, Minkowski)



## Порівняння результатів

```
results_df = pd.DataFrame(results).T
print("\nПорівняння метрик для різних підходів:")
print(results_df)
```



```
Порівняння метрик для різних підходів:
      recall      f1  accuracy  training_time
euclidean  0.310538  0.304245    0.330      0.008184
manhattan  0.290290  0.289293    0.305      0.003312
minkowski  0.310538  0.304245    0.330      0.002668
```

## Візуалізація порівняння метрик

```
import matplotlib.pyplot as plt

plt.style.use('Solarize_Light2')
fig, ax = plt.subplots(figsize=(10, 6))

# Горизонтальна гістограма
results_df[['recall', 'f1', 'accuracy']].plot(kind='barh', ax=ax, color=['#6a5acd', '#ff6347', '#3cb371']) # Змінили кольори

ax.set_title('Порівняння метрик між різними підходами', fontsize=16)
ax.set_xlabel('Оцінки', fontsize=12)
ax.set_ylabel('Метрики', fontsize=12)
ax.legend(title='Метрики', title_fontsize='13', fontsize='11')
plt.tight_layout()
plt.show()
```



## Результати

## Euclidean

- Оцінка повноти (Recall): 0.3105. Це означає, що модель правильно ідентифікує 31.05% позитивних випадків для кожного класу.
- Оцінка F1: 0.3042. Середня F1-оцінка моделі становить 30.42%, що вважається помірним результатом.
- Оцінка точності (Accuracy): 0.3300. Це свідчить про те, що 33.00% усіх прогнозів були правильними.
- Час навчання: 0.0030 сек.

## Manhattan



- Оцінка повноти (Recall): 0.2903. Ця оцінка трохи нижча за ту, що отримана з метрикою Euclidean, і свідчить про те, що модель правильно ідентифікує 29.03% позитивних випадків.
- Оцінка F1: 0.2893. F1-оцінка в середньому становить 28.93%, що нижче за результат з метрикою Euclidean.
- Оцінка точності (Accuracy): 0.3050. Загальна точність моделі з метрикою Manhattan становить 30.50%, що також трохи нижче за Euclidean.
- Час навчання: 0.0012 сек. Час навчання моделі з метрикою Manhattan був коротшим у порівнянні з Euclidean.

## Minkowski

- Оцінка повноти (Recall): 0.3105. Ця оцінка збігається з результатами для метрики Euclidean, що свідчить про те, що модель правильно ідентифікує 31.05% позитивних випадків.
- Оцінка F1: 0.3042. Також та ж оцінка, що і для Euclidean, з F1-оцінкою 30.42%.
- Оцінка точності (Accuracy): 0.3300. Як і в метриці Euclidean, точність становить 33.00%.
- Час навчання: 0.0020 сек. Час навчання з метрикою Minkowski трохи менший, ніж з Euclidean.

## Висновок

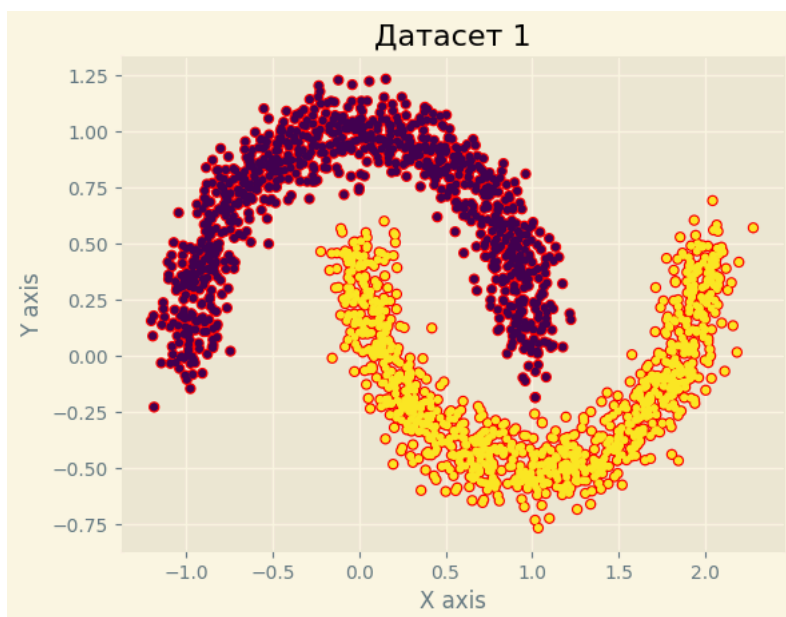
Метрики Euclidean та Minkowski показали однакові результати для всіх метрик, що може свідчити про те, що параметр  $p$  у метриці Minkowski не вплинув на результати, оскільки за замовчуванням дорівнює 2 (Euclidean). Метрика Manhattan продемонструвала дещо гірші результати в оцінках Recall, F1 та точності, але мала найменший час навчання.

У візуалізації кластерів спостерігається певне перекриття між кластерами, що вказує на складність задачі класифікації. Незважаючи на те, що всі три візуалізації виглядають дуже схожими, це може свідчити про те, що для цього конкретного набору даних вибір метрики не має критичного впливу на результати класифікації. Однак помітні невеликі відмінності в розташуванні окремих точок між графіками. Виявлено кілька віддалених точок (наприклад, справа вгорі), які можуть бути викидами або представляти особливі випадки, такі як клієнти з дуже високим доходом або незвичайною комбінацією характеристик.

## ✓ Agnes, Birch, DBSCAN

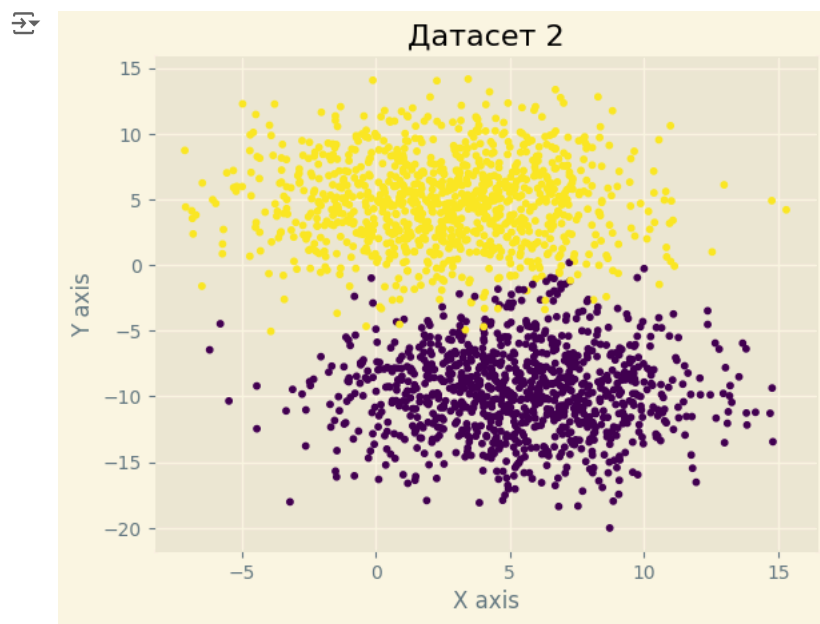
```
from sklearn.datasets import make_classification, make_circles
from sklearn.cluster import AgglomerativeClustering, Birch, DBSCAN
from sklearn.metrics import silhouette_score, adjusted_rand_score, normalized_mutual_info_score
import matplotlib.pyplot as plt
```

```
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')
plt.scatter(X1[:, 0], X1[:, 1], s=10, c=Y1)
plt.title('Датасет 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('Датасет 2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.scatter(X3[:, 0], X3[:, 1], s=10, c=Y3)
plt.show()
```

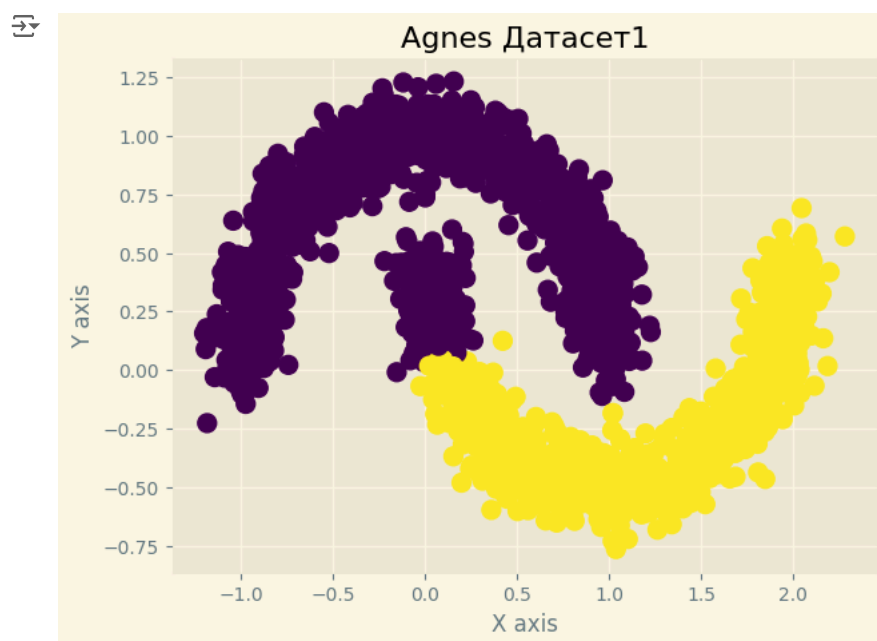


## ✓ Agnes

```
from sklearn.cluster import AgglomerativeClustering
```

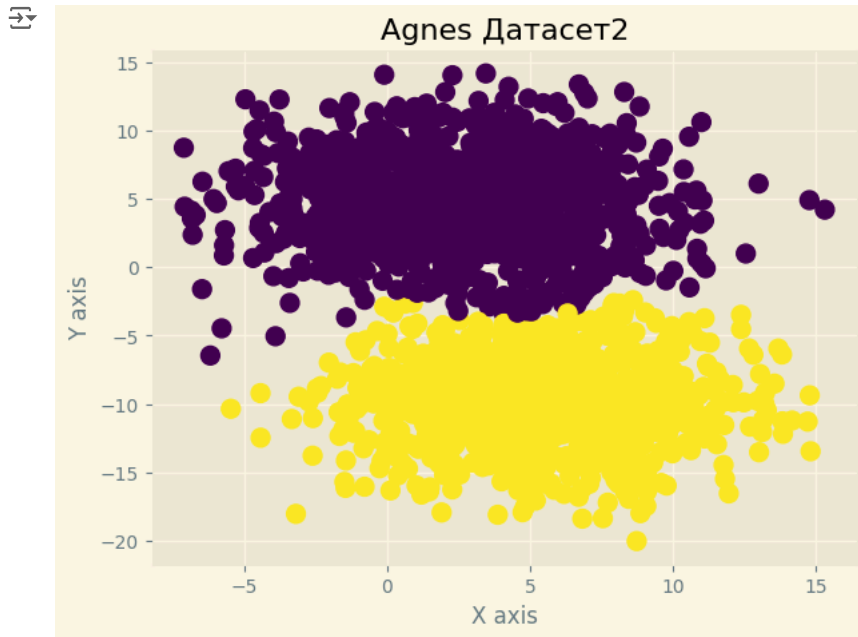
```
agnes_1 = AgglomerativeClustering(n_clusters=2)
y_agnes_1 = agnes_1.fit_predict(X1)
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_agnes_1)
plt.title('Agnes Датасет1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



```
agnes_2 = AgglomerativeClustering(n_clusters=2)
y_agnes_2 = agnes_2.fit_predict(X3)
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,2)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_agnes_2)
plt.title('Agnes Датасет2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



```
from sklearn.metrics import silhouette_score, adjusted_rand_score, normalized_mutual_info_score
silhouette_1 = silhouette_score(X1, y_agnes_1)
ari_1 = adjusted_rand_score(Y1, y_agnes_1)
nmi_1 = normalized_mutual_info_score(Y1, y_agnes_1)
print(f"Датасет 1 - Silhouette Coefficient: {silhouette_1:.3f}")
print(f"Датасет 1 - ARI: {ari_1:.3f}")
print(f"Датасет 1 - NMI: {nmi_1:.3f}")
```

```
↳ Датасет 1 - Silhouette Coefficient: 0.406
Датасет 1 - ARI: 0.716
Датасет 1 - NMI: 0.671
```

```
silhouette_2 = silhouette_score(X3, y_agnes_2)
ari_2 = adjusted_rand_score(Y3, y_agnes_2)
nmi_2 = normalized_mutual_info_score(Y3, y_agnes_2)
print(f"Датасет 2 - Silhouette Coefficient: {silhouette_2:.3f}")
print(f"Датасет 2 - ARI: {ari_2:.3f}")
print(f"Датасет 2 - NMI: {nmi_2:.3f}")
```

```
↳ Датасет 2 - Silhouette Coefficient: 0.588
Датасет 2 - ARI: 0.908
Датасет 2 - NMI: 0.843
```

## Результати

### Датасет 1

- Silhouette Coefficient: 0.406. Значення близько 0.4 свідчить про те, що кластери частково перекриваються, тобто об'єкти в одному кластері можуть бути близькими до об'єктів з іншого. Це очікувано для "місячного" датасету, оскільки форми кластерів є складними та нелінійно розділеними.
- ARI (Adjusted Rand Index): 0.716. Це вказує на те, що передбачені кластери досить добре відповідають справжнім міткам. Значення ARI понад 0.7 свідчить про те, що модель змогла успішно розрізнити два кластери, хоча й з певними неточностями.
- NMI (Normalized Mutual Information): 0.671. Це показує значну ступінь відповідності між передбаченими кластерами та справжніми мітками, що свідчить про досить хорошу інформаційну схожість між двома розбиттями.

### Датасет 2

- Silhouette Coefficient: 0.588. Це показує, що алгоритм успішно розділив об'єкти на два окремі кластери, де об'єкти знаходяться ближче до свого центроїда, ніж до центрів інших кластерів.

- ARI: 0.908. Високий показник, що свідчить про майже ідеальну відповідність між передбаченими кластерами та справжніми мітками, що вказує на дуже точну кластеризацію.
- NMI: 0.843. Цей високий показник також підтверджує, що передбачена кластеризація та справжні мітки мають високу інформаційну схожість.

## Висновок

Кластеризація для датасету 1 демонструє досить задовільні результати, хоча існують певні перетинання між кластерами. Це є природним явищем для "місячних" форм, які ускладнюють кластеризацію стандартними методами.

Натомість, датасет 2 показує значно кращі результати кластеризації. Це пояснюється тим, що дані у вигляді кластерів з чіткими центрами легше піддаються розділенню стандартними алгоритмами. Чіткі кордони між кластерами спрощують їхнє розділення, що підтверджується високими значеннями метрик, які свідчать про майже ідеальну кластеризацію.

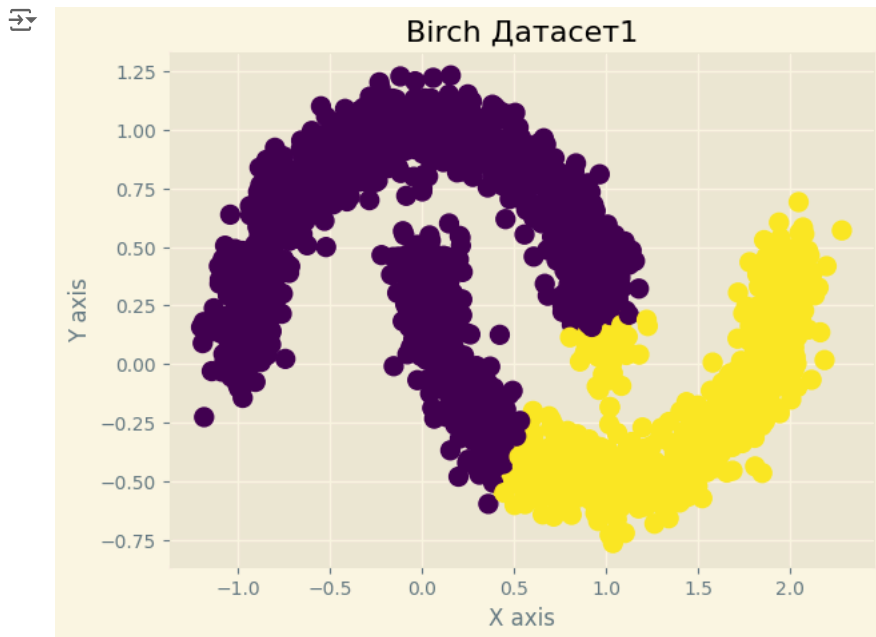
Отже, порівняння показує, що датасет 1 є складнішим для кластеризації через нестандартну форму кластерів, тоді як датасет 2 має значно чіткіші межі, що полегшує процес кластеризації.

## ✓ Birch

```
from sklearn.cluster import Birch
```

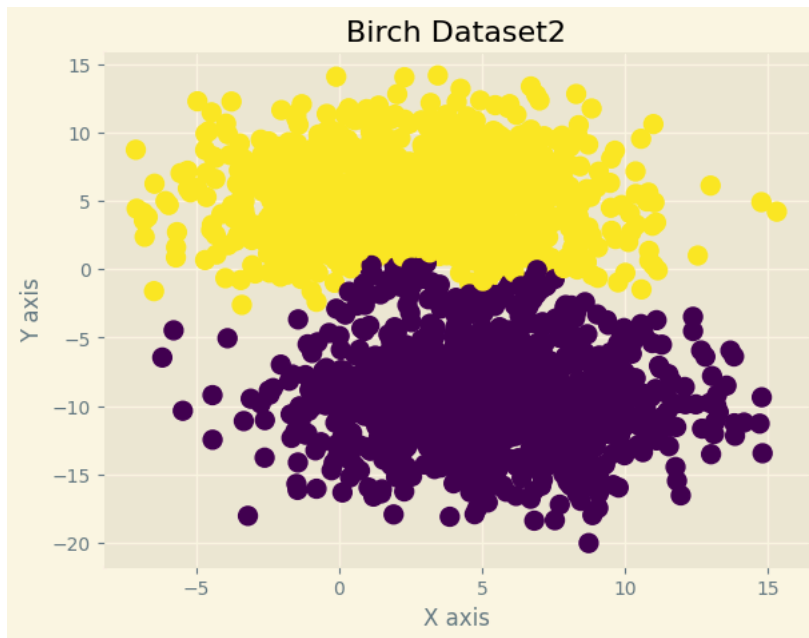
```
birch_1 = Birch(n_clusters=2, threshold=0.5, branching_factor=100)
y_birch_1 = birch_1.fit_predict(X1)
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_birch_1)
plt.title('Birch Датасет1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



```
birch_2 = Birch(n_clusters=2, threshold=0.1, branching_factor=100)
y_birch_2 = birch_2.fit_predict(X3)
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,2)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_birch_2)
plt.title('Birch Dataset2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
```



```
from sklearn.metrics import silhouette_score, adjusted_rand_score, normalized_mutual_info_score
silhouette_birch_1 = silhouette_score(X1, y_birch_1)
ari_birch_1 = adjusted_rand_score(Y1, y_birch_1)
nmi_birch_1 = normalized_mutual_info_score(Y1, y_birch_1)
print(f"Birch Dataset 1 - Silhouette Coefficient: {silhouette_birch_1:.3f}")
print(f"Birch Dataset 1 - ARI: {ari_birch_1:.3f}")
print(f"Birch Dataset 1 - NMI: {nmi_birch_1:.3f}")
```



```
Birch Dataset 1 - Silhouette Coefficient: 0.458
Birch Dataset 1 - ARI: 0.377
Birch Dataset 1 - NMI: 0.341
```

```
silhouette_birch_2 = silhouette_score(X3, y_birch_2)
ari_birch_2 = adjusted_rand_score(Y3, y_birch_2)
nmi_birch_2 = normalized_mutual_info_score(Y3, y_birch_2)
print(f"Birch Dataset 2 - Silhouette Coefficient: {silhouette_birch_2:.3f}")
print(f"Birch Dataset 2 - ARI: {ari_birch_2:.3f}")
print(f"Birch Dataset 2 - NMI: {nmi_birch_2:.3f}")
```



```
Birch Dataset 2 - Silhouette Coefficient: 0.576
Birch Dataset 2 - ARI: 0.872
Birch Dataset 2 - NMI: 0.810
```

## Результати

### Датасет 1

- Silhouette Coefficient: 0.458. Це значення близьке до 0.5, що вказує на певне перекриття між кластерами, але все ще свідчить про певну структуру в даних. Згідно з цими результатами, об'єкти в одному кластері можуть бути близькими до об'єктів з іншого кластера, що вказує на помірну кластеризацію.
- ARI (Adjusted Rand Index): 0.377. Це значення вказує на те, що передбачені кластери мають відносно низьку відповідність до справжніх міток, що може свідчити про проблеми в кластеризації через складність структури даних.
- NMI (Normalized Mutual Information): 0.341. Цей показник свідчить про обмежену інформаційну схожість між кластеризацією та справжніми мітками, що підтверджує труднощі алгоритму Birch у виділенні чітких кластерів у цьому датасеті.

### Датасет 2

- Silhouette Coefficient: 0.576. Це показник вказує на те, що алгоритм Birch успішно розділив об'єкти на два окремі кластери, з більшою дистанцією між об'єктами різних кластерів.
- ARI: 0.872. Високе значення ARI свідчить про практично ідеальну відповідність між передбаченими кластерами та справжніми мітками, що свідчить про точну кластеризацію.
- NMI: 0.810. Високий показник NMI вказує на значну інформаційну схожість між кластеризацією та справжніми мітками, що підтверджує ефективність алгоритму Birch у розподілі даних на чіткі кластери.

## Висновок

Алгоритм Birch показав різні результати для обох датасетів. Для датасету 1 результати кластеризації були менш точними, з низькими показниками ARI і NMI, що свідчить про труднощі у виділенні окремих кластерів через складну структуру даних. Це підтверджує, що дані в даному випадку є більш змішаними та нелінійно розділеними.

На противагу, датасет 2 продемонстрував значно кращі результати кластеризації, з високими значеннями метрик. Це вказує на те, що дані у цьому випадку мають чіткіші межі та структури, що спростило процес кластеризації. Високі показники ARI і NMI свідчать про те, що алгоритм Birch зміг ефективно виділити кластери, які добре відповідають справжнім міткам.

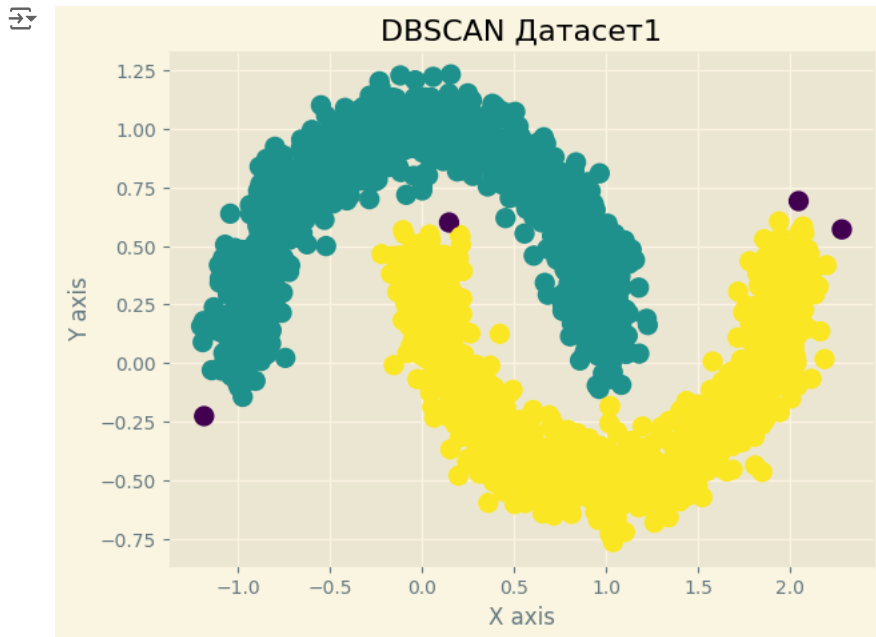
Отже, порівняння результатів показує, що алгоритм Birch ефективніше працює з даними, які мають чіткі структури, як у датасеті 2, у той час як складні форми даних у датасеті 1 створюють додаткові виклики для кластеризації.

## ✓ DBSCAN

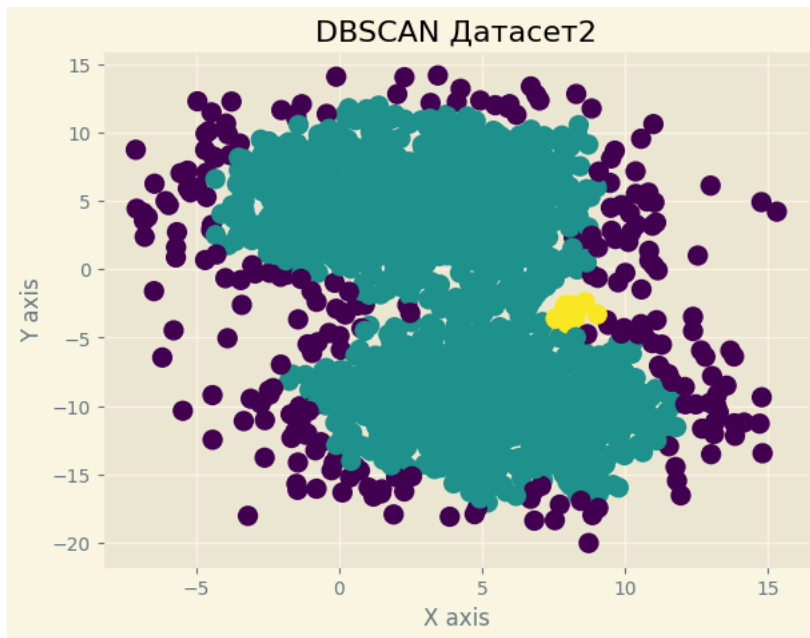
```
from sklearn.cluster import DBSCAN
```

```
dbscan_1 = DBSCAN(eps=0.2, min_samples=70)
y_dbscan_1 = dbscan_1.fit_predict(X1)
```

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,2)
plt.scatter(X1[:, 0], X1[:, 1], s=100, c=y_dbscan_1)
plt.title('DBSCAN Датасет1')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
inti_point = np.random.randint(0, len(X1)-1, 2 )
medoids=X1[inti_point]
```



```
dbscan_2 = DBSCAN(eps=1, min_samples=10)
y_dbscan_2 = dbscan_2.fit_predict(X3)
plt.figure(figsize=(15,5))
plt.subplot(1,2,2)
plt.scatter(X3[:, 0], X3[:, 1], s=100, c=y_dbscan_2)
plt.title('DBSCAN Датасет2')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.show()
inti_point = np.random.randint(0, len(X3)-1, 2 )
medoids=X3[inti_point]
```



```
from sklearn.metrics import silhouette_score, adjusted_rand_score, normalized_mutual_info_score
import numpy as np
```

```
silhouette_dbscan_1 = silhouette_score(X1, y_dbscan_1)
ari_dbscan_1 = adjusted_rand_score(Y1, y_dbscan_1)
nmi_dbscan_1 = normalized_mutual_info_score(Y1, y_dbscan_1)
print(f"DBSCAN Датасет 1 - Silhouette Coefficient: {silhouette_dbscan_1}")
print(f"DBSCAN Датасет 1 - ARI: {ari_dbscan_1:.3f}")
print(f"DBSCAN Датасет 1 - NMI: {nmi_dbscan_1:.3f}")
```



```
DBSCAN Датасет 1 - Silhouette Coefficient: 0.3010813290557993
DBSCAN Датасет 1 - ARI: 0.992
DBSCAN Датасет 1 - NMI: 0.979
```

```
silhouette_dbscan_2 = silhouette_score(X1, y_dbscan_2)
ari_dbscan_2 = adjusted_rand_score(Y3, y_dbscan_2)
nmi_dbscan_2 = normalized_mutual_info_score(Y3, y_dbscan_2)
print(f"DBSCAN Датасет 2 - Silhouette Coefficient: {silhouette_dbscan_2}")
print(f"DBSCAN Датасет 2 - ARI: {ari_dbscan_2:.3f}")
print(f"DBSCAN Датасет 2 - NMI: {nmi_dbscan_2:.3f}")
```



```
DBSCAN Датасет 2 - Silhouette Coefficient: -0.01598636511050591
DBSCAN Датасет 2 - ARI: -0.000
DBSCAN Датасет 2 - NMI: 0.002
```

Алгоритм DBSCAN продемонстрував дуже високі показники за метриками ARI та NMI на датасеті 1. Однак на датасеті 2 результати кластеризації виявилися неефективними, оскільки Silhouette Coefficient виявився негативним, а ARI і NMI показали низькі значення.

## Висновки

1. **AGNES** продемонстрував хороші результати на датасеті 2, показавши високі значення ARI та NMI. Проте результати для датасету 1 були гіршими, що вказує на те, що цей метод ефективно працює з простими, чітко розділеними кластерами, але може стикатися з труднощами, коли кластери мають складну структуру. Хоча AGNES є простим у реалізації, він може виявитися обчислювально витратним і не підходить для великих наборів даних.
2. **BIRCH** показав стабільні результати, особливо на датасеті 2. Цей алгоритм є ефективним для великих наборів даних і добре справляється з обробкою великих обсягів інформації, проте він поступається за точністю деяким іншим методам. Результати для датасету 1 виявилися гіршими, що свідчить про можливі труднощі у кластеризації даних з нерівномірною щільністю.
3. **DBSCAN** добре зарекомендував себе на датасеті 1, де присутні кластери з різною щільністю. Однак цей метод виявився чутливим до шуму та щільності кластерів, що стало очевидним на датасеті 2, де результати були значно гіршими через рівномірну щільність кластерів і наявність шуму. DBSCAN є ефективним для складних наборів даних з різною щільністю, проте його результати залежать від налаштувань гіперпараметрів, таких як `eps` і `min_samples`.

## ✓ Affinity propagation

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
data3 = "/content/drive/My Drive/data/Mall_Customers.csv"
df = pd.read_csv(data3, header=0, sep=',\s*', engine='python')
```

Mounted at /content/drive

```
df.head()
```

	CustomerID	Genre	Age	Annual Income (k\$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

Подальші дії:

[Переглянути рекомендовані графіки](#)

[New interactive sheet](#)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
#   Column                Non-Null Count  Dtype
---  -
0   CustomerID            200 non-null   int64
1   Genre                 200 non-null   object
2   Age                   200 non-null   int64
3   Annual Income (k$)    200 non-null   int64
4   Spending Score (1-100) 200 non-null   int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
df.isnull().sum()
```

```
0
CustomerID    0
Genre         0
Age           0
Annual Income (k$)  0
Spending Score (1-100)  0

dtype: int64
```

```
from sklearn.cluster import AffinityPropagation
```

```
X_numerics = df[['Age', 'Annual Income (k$)', 'Spending Score (1-100)']]
```

```
dbscan = DBSCAN(eps=12.6, min_samples=4)
y_dbscan = dbscan.fit_predict(X_numerics)
AF = AffinityPropagation(preference=-11800).fit(X_numerics)
AF_clustered = X_numerics.copy()
AF_clustered.loc[:, 'Cluster'] = AF.labels_
AF_clust_sizes = AF_clustered.groupby('Cluster').size().to_frame()
AF_clust_sizes.columns = ["AF_size"]
AF_clust_sizes
```



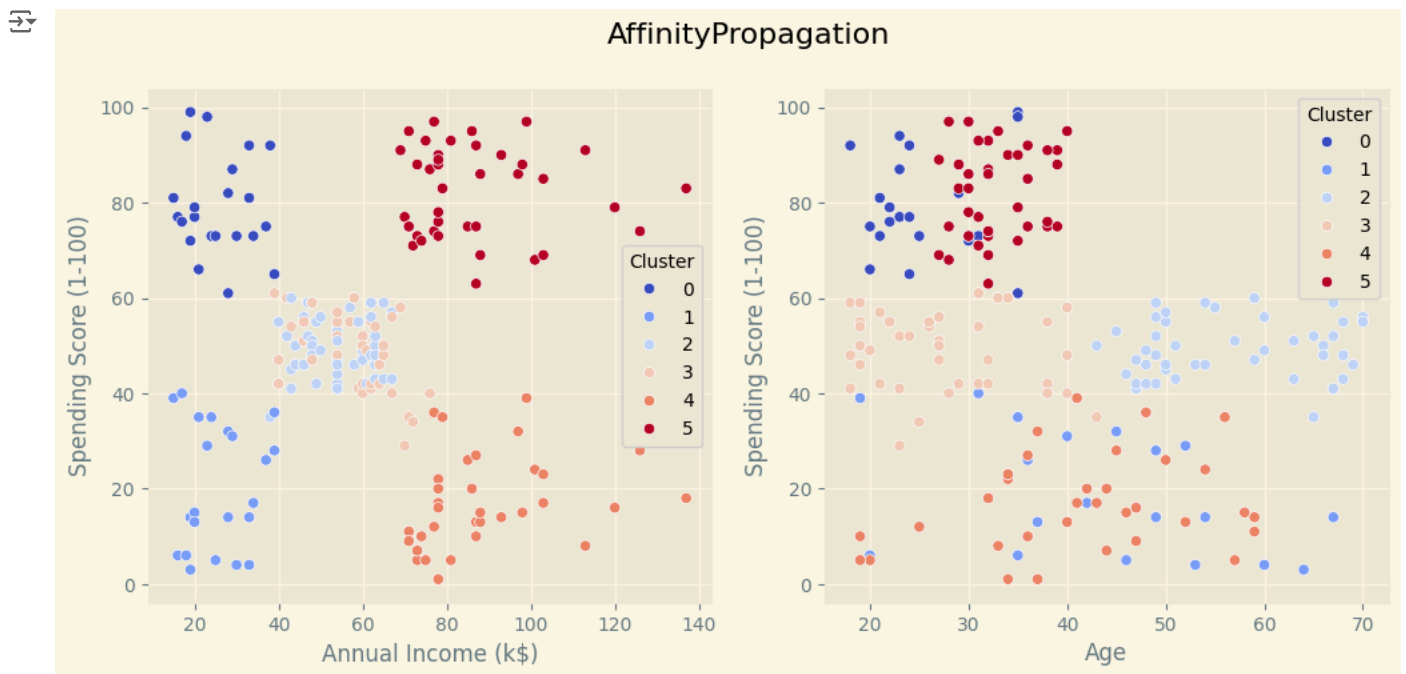
Cluster	AF_size
0	22
1	22
2	44
3	39
4	34
5	39

Подальші дії:

[Переглянути рекомендовані графіки](#)[New interactive sheet](#)

```
fig3, (ax_af) = plt.subplots(1, 2, figsize=(12, 5))
fig3.suptitle('AffinityPropagation', fontsize=16)
scat_1 = sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', data=AF_clustered,
                        hue='Cluster', ax=ax_af[0], palette='coolwarm', legend='full')
sns.scatterplot(x='Age', y='Spending Score (1-100)', data=AF_clustered,
                hue='Cluster', palette='coolwarm', 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()
```



```
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.metrics import normalized_mutual_info_score
silhouette_af = silhouette_score(X_numerics, AF.labels_)
ari_af = adjusted_rand_score(y_dbscan, AF.labels_)
nmi_af = normalized_mutual_info_score(y_dbscan, AF.labels_)
print('AffinityPropagation Silhouette Score: {:.4f}'.format(silhouette_af))
print('AffinityPropagation ARI Score: {:.4f}'.format(ari_af))
print('AffinityPropagation NMI Score: {:.4f}'.format(nmi_af))
```

```
➡ AffinityPropagation Silhouette Score:0.4516
AffinityPropagation ARI Score: 0.3562
AffinityPropagation NMI Score: 0.5737
```

## ✓ K-Means

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

```
KM_5_clusters = KMeans(n_clusters=5, init='k-means++').fit(X_numerics)
KM5_clustered = X_numerics.copy()
```

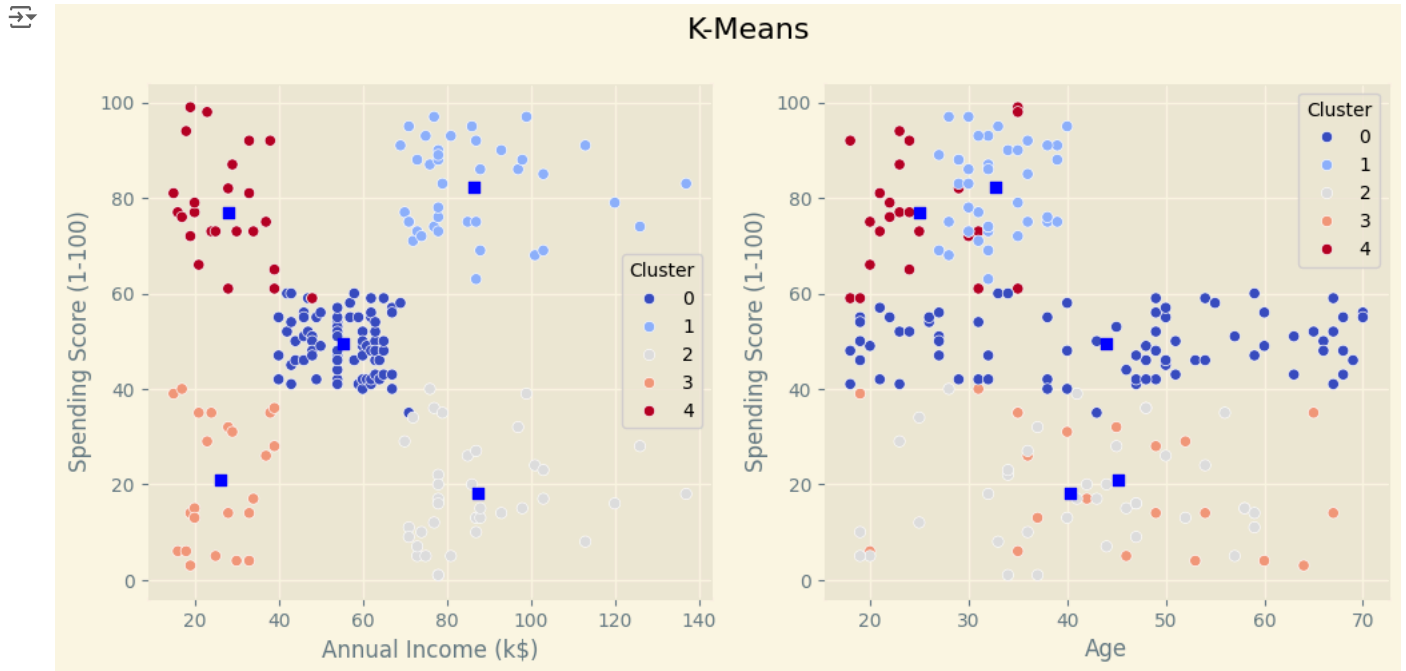
```
KM5_clustered.loc[:, 'Cluster'] = KM_5_clusters.labels_
```

```
fig1, (axes) = plt.subplots(1, 2, figsize=(12, 5))
fig1.suptitle('K-Means', fontsize=16)
```

```
scat_1 = sns.scatterplot(x='Annual Income (k$)', y='Spending Score (1-100)', data=KM5_clustered,
                        hue='Cluster', ax=axes[0], palette='coolwarm', legend='full')
sns.scatterplot(x='Age', y='Spending Score (1-100)', data=KM5_clustered,
                hue='Cluster', palette='coolwarm', ax=axes[1], legend='full')
```

```
axes[0].scatter(KM_5_clusters.cluster_centers[:, 1], KM_5_clusters.cluster_centers[:, 2],
                marker='s', s=40, c="blue")
axes[1].scatter(KM_5_clusters.cluster_centers[:, 0], KM_5_clusters.cluster_centers[:, 2],
                marker='s', s=40, c="blue")
```

```
plt.show()
```



Алгоритм K-Means виявив 5 кластерів:

1. Клієнти з низьким річним доходом та високим рівнем витрат.
2. Клієнти з середнім річним доходом і середнім рівнем витрат.
3. Клієнти з високим річним доходом та низьким рівнем витрат.
4. Клієнти з високим річним доходом і високим рівнем витрат.
5. Клієнти з низьким річним доходом та низьким рівнем витрат.

```
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.metrics import normalized_mutual_info_score
silhouette_kmeans = silhouette_score(X_numerics, KM_5_clusters.labels_)
ari_kmeans = adjusted_rand_score(y_dbSCAN, KM_5_clusters.labels_)
nmi_kmeans = normalized_mutual_info_score(y_dbSCAN, KM_5_clusters.labels_)
print('K-Means Silhouette Score: {:.4f}'.format(silhouette_kmeans))
print('K-Means ARI Score: {:.4f}'.format(ari_kmeans))
print('K-Means NMI Score: {:.4f}'.format(nmi_kmeans))
```

```
K-Means Silhouette Score: 0.4398
K-Means ARI Score: 0.5225
K-Means NMI Score: 0.6095
```

## ✓ Висновок

Немає чітко виражених груп за віком клієнтів.

Показники Silhouette Score близькі, проте Affinity Propagation демонструє трохи кращі результати, що може свідчити про вищу якість кластерів.

За значеннями ARI та NMI K-Means перевершує інші методи, що вказує на кращу відповідність кластеризації K-Means кластеризації DBSCAN. Це може свідчити про те, що K-Means є більш придатним для даних у даному випадку.

Отже, якщо важливіша якість кластеризації (чіткість кластерів), Affinity Propagation може бути оптимальним вибором. Якщо ж необхідно досягти максимального збігу результатів кластеризації з еталонною кластеризацією (DBSCAN), K-Means може виявитися кращим варіантом.