**ЛАБОРАТОРНА РОБОТА № 7**

**ДОСЛІДЖЕННЯ МЕТОДІВ НЕКОНТРОЬОВАНОГО НАВЧАННЯ.**

***Мета:*** використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи неконтрольованої класифікації даних у машинному навчанні.

**Хід роботи**

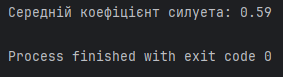
Завдання 2.1. Кластеризація даних за допомогою методу k-середніх

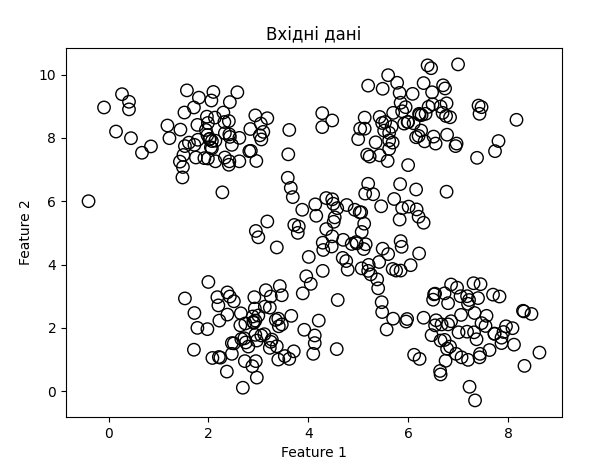
Лістинг програми:

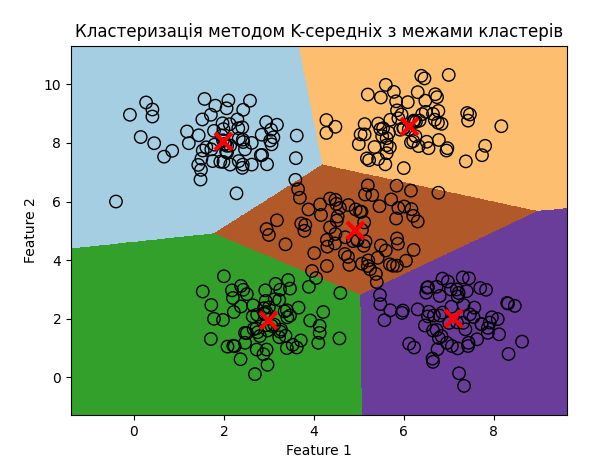
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn.metrics import silhouette\_score  
  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
  
# Кількість кластерів  
num\_clusters = 5  
  
# Візуалізація вхідних даних  
plt.figure()  
plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none', edgecolors='black', s=80)  
plt.title('Вхідні дані')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.show()  
  
# Ініціалізація та навчання моделі KMeans  
kmeans = KMeans(init='k-means++', n\_clusters=num\_clusters, n\_init=10, random\_state=42)  
kmeans.fit(X)  
  
# Виведення оцінки силуета для оцінки якості кластеризації  
silhouette\_avg = silhouette\_score(X, kmeans.labels\_)  
print(f"Середній коефіцієнт силуета: {silhouette\_avg:.2f}")  
  
# Встановлення параметрів сітки  
step\_size = 0.01  
x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1  
y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1  
  
# Побудова сітки координат для відображення кордонів кластерів  
x\_vals, y\_vals = np.meshgrid(np.arange(x\_min, x\_max, step\_size), np.arange(y\_min, y\_max, step\_size))  
  
# Передбачення кластерних міток для кожної точки сітки  
output = kmeans.predict(np.c\_[x\_vals.ravel(), y\_vals.ravel()])  
output = output.reshape(x\_vals.shape)

# Візуалізація меж кластерів  
plt.figure()  
plt.imshow(output, interpolation='nearest', extent=(x\_vals.min(), x\_vals.max(), y\_vals.min(), y\_vals.max()),  
 cmap=plt.cm.Paired, aspect='auto', origin='lower')  
  
# Відображення вхідних даних  
plt.scatter(X[:, 0], X[:, 1], marker='o', facecolors='none', edgecolors='black', s=80)  
  
# Відображення центрів кластерів  
cluster\_centers = kmeans.cluster\_centers\_  
plt.scatter(cluster\_centers[:, 0], cluster\_centers[:, 1], marker='x', s=150, linewidths=3, color='red', zorder=10)  
  
# Настроювання меж графіку  
plt.title('Кластеризація методом K-середніх з межами кластерів')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.xlim(x\_min, x\_max)  
plt.ylim(y\_min, y\_max)  
plt.show()

Виконання програми:





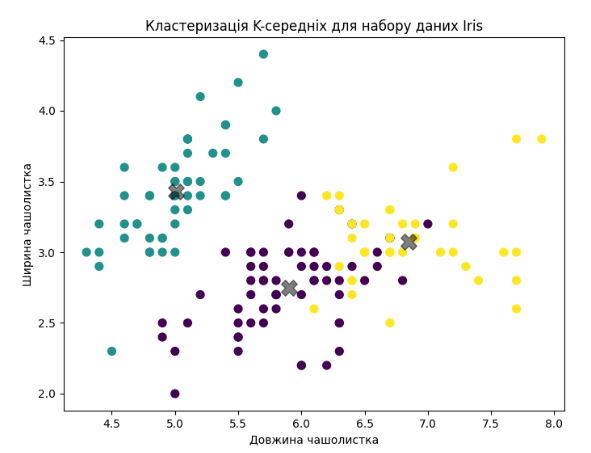


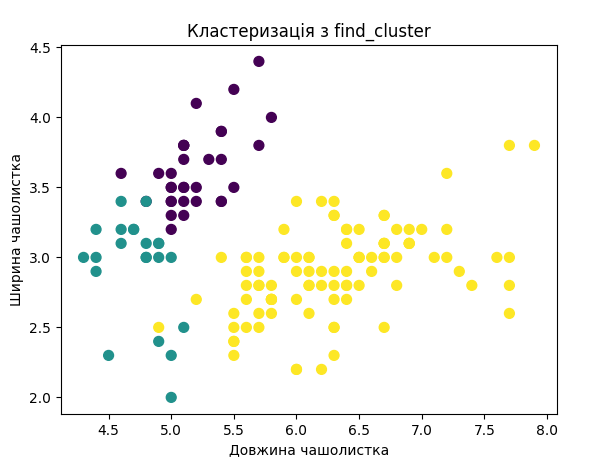
Завдання 2.2. Кластеризація K-середніх для набору даних Iris

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn.datasets import load\_iris  
from sklearn.metrics import pairwise\_distances\_argmin  
  
iris = load\_iris()  
X = iris['data'] # Дані про атрибути квіток Iris  
y = iris['target'] # Класи квіток (Setosa, Versicolour, Virginica)  
  
# Ініціалізація моделі  
kmeans = KMeans(  
 n\_clusters=3, # Кількість кластерів дорівнює 3 (оскільки є 3 класи квітів)  
 init='k-means++', # Використання методу ініціалізації k-means++ для швидшої збіжності  
 n\_init=10, # Кількість запусків алгоритму з різними початковими центроїдами  
 max\_iter=300, # Максимальна кількість ітерацій для одного запуску алгоритму  
 tol=0.0001, # Допустима похибка для зупинки алгоритму  
 random\_state=42 # Встановлюємо random\_state для відтворюваності результатів  
)  
  
# Навчання моделі  
kmeans.fit(X)  
# Отримання передбачених міток для кожної точки даних  
y\_kmeans = kmeans.predict(X)  
  
# Візуалізація кластерів  
plt.figure(figsize=(8, 6))  
plt.scatter(X[:, 0], X[:, 1], c=y\_kmeans, s=50, cmap='viridis') # Відображення точок з кольорами  
  
# Отримання центрів кластерів  
centers = kmeans.cluster\_centers\_  
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5, marker='X') # Відображення центрів кластерів  
  
plt.title('Кластеризація K-середніх для набору даних Iris')  
plt.xlabel('Довжина чашолистка')  
plt.ylabel('Ширина чашолистка')  
plt.show()  
  
# Функція для ручної кластеризації, що знаходить центри кластерів  
def find\_cluster(X, n\_clusters, rseed=2):  
 rng = np.random.RandomState(rseed) # Ініціалізація випадкових чисел для відтворюваності  
 i = rng.permutation(X.shape[0])[:n\_clusters] # Вибір початкових центрів випадковим чином  
 centers = X[i] # Обрані центри  
  
 while True:  
 # Призначення кожної точки до найближчого центру кластера  
 labels = pairwise\_distances\_argmin(X, centers)  
  
 # Обчислення нових центрів кластерів як середнє значення точок у кожному кластері  
 new\_centers = np.array([X[labels == i].mean(0) for i in range(n\_clusters)])  
  
 # Якщо центри не змінилися, виходимо з циклу  
 if np.all(centers == new\_centers):  
 break  
  
 centers = new\_centers  
  
 return centers, labels  
  
# Використання функції find\_cluster з 3 кластерами  
centers, labels = find\_cluster(X, 3)  
plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')  
plt.title("Кластеризація з find\_cluster")  
plt.xlabel("Довжина чашолистка")  
plt.ylabel("Ширина чашолистка")  
plt.show()

Виконання програми:



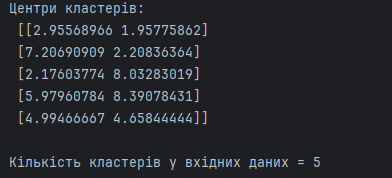


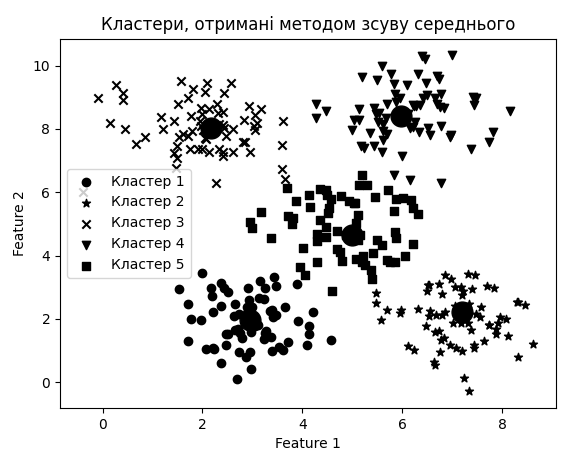
Завдання 2.3. Оцінка кількості кластерів з використанням методу зсуву середнього

Лістинг програми:

import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import MeanShift, estimate\_bandwidth  
from itertools import cycle  
  
# Завантаження вхідних даних з файлу  
X = np.loadtxt('data\_clustering.txt', delimiter=',')  
  
# Оцінка ширини вікна (bandwidth) для набору даних X  
# Ширина вікна визначає, наскільки великими будуть кластери  
# Параметр quantile впливає на ширину вікна: при більшому значенні кількість кластерів зменшується  
bandwidth\_X = estimate\_bandwidth(X, quantile=0.1, n\_samples=len(X))  
  
# Ініціалізація та навчання моделі MeanShift з використанням обчисленої ширини вікна  
meanshift\_model = MeanShift(bandwidth=bandwidth\_X, bin\_seeding=True)  
meanshift\_model.fit(X)  
  
# Отримання центрів кластерів після навчання  
cluster\_centers = meanshift\_model.cluster\_centers\_  
print('\nЦентри кластерів:\n', cluster\_centers)  
  
# Отримання міток кластерів для кожної точки в наборі даних  
labels = meanshift\_model.labels\_  
  
# Оцінка кількості кластерів (кількість унікальних міток)  
num\_clusters = len(np.unique(labels))  
print("\nКількість кластерів у вхідних даних =", num\_clusters)  
  
# Візуалізація точок даних з кольорами для кожного кластера  
plt.figure()  
markers = cycle('o\*xvs') # Цикл для маркерів, щоб кожен кластер мав свій стиль точки  
  
for i, marker in zip(range(num\_clusters), markers):  
 # Відображення точок, що належать до поточного кластера  
 plt.scatter(X[labels == i, 0], X[labels == i, 1], marker=marker, color='black', label=f'Кластер {i+1}')  
  
 # Відображення центру поточного кластера  
 center = cluster\_centers[i]  
 plt.plot(center[0], center[1], marker='o', markerfacecolor='black',  
 markeredgecolor='black', markersize=15)  
  
plt.title('Кластери, отримані методом зсуву середнього')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.legend()  
plt.show()

Виконання програми:



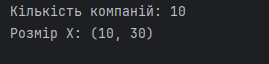


Завдання 2.4. Знаходження підгруп на фондовому ринку з використанням моделі поширення подібності

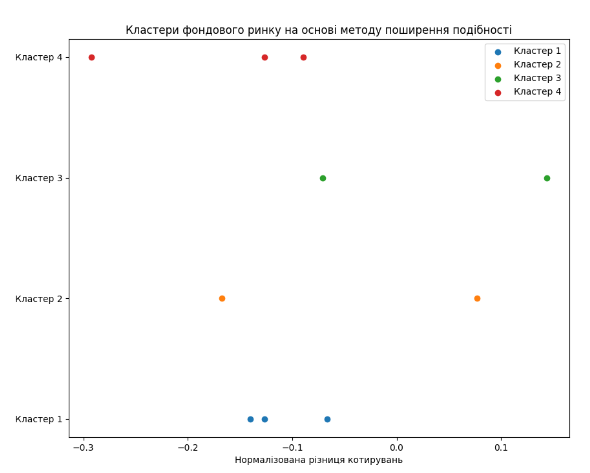
Лістинг програми:

import json  
import numpy as np  
from sklearn import covariance, cluster  
import matplotlib.pyplot as plt  
from sklearn.model\_selection import KFold  
  
# Завантаження даних із JSON  
json\_path = "generated\_stock\_data\_large.json"  
with open(json\_path, "r") as file:  
 data = json.load(file)  
  
# Отримання символів компаній та їхніх назв  
symbols = [item["Symbol"] for item in data]  
names = [item["Name"] for item in data]  
  
# Замість завантаження даних із yfinance  
quotes = []  
valid\_symbols = []  
valid\_names = []  
  
# Отримання історичних даних з JSON  
for item in data:  
 historical\_data = item.get("HistoricalData", [])  
 if historical\_data:  
 # Збереження котирувань (тільки цін закриття та відкриття)  
 quotes.append(historical\_data)  
 valid\_symbols.append(item["Symbol"])  
 valid\_names.append(item["Name"])  
 else:  
 print(f"Немає історичних даних для {item['Symbol']}")  
  
# Перевірка, чи отримано дані  
if not quotes:  
 print("Не вдалося отримати дані для жодної компанії.")  
 exit()  
  
# Обчислення нормалізованих змін цін  
closing\_prices = []  
opening\_prices = []  
  
# Отримання даних цін закриття та відкриття  
for stock\_data in quotes:  
 closing\_prices.append([day['Close'] for day in stock\_data])  
 opening\_prices.append([day['Open'] for day in stock\_data])  
  
closing\_prices = np.array(closing\_prices)  
opening\_prices = np.array(opening\_prices)  
quotes\_diff = closing\_prices - opening\_prices  
  
# Нормалізація даних (кожен рядок відповідає окремій компанії)  
X = quotes\_diff / quotes\_diff.std(axis=1, keepdims=True)  
  
# Перевірка розмірів масивів  
print(f"Кількість компаній: {len(valid\_names)}")  
print(f"Розмір X: {X.shape}")  
  
# Побудова графової моделі залежностей з меншою кількістю сплітів  
edge\_model = covariance.GraphicalLassoCV(cv=KFold(n\_splits=3))  
  
# Навчання графової моделі  
with np.errstate(invalid='ignore'):  
 edge\_model.fit(X)  
  
# Виконання кластеризації  
affinity\_model = cluster.AffinityPropagation(affinity="euclidean", damping=0.9)  
affinity\_model.fit(X) # Використання X для кластеризації, а не подібностей  
  
# Отримання міток кластерів  
labels = affinity\_model.labels\_  
num\_clusters = len(np.unique(labels))  
print(f"\nКількість кластерів: {num\_clusters}\n")  
  
# Додаткове виведення для перевірки розмірів  
print(f"Розмір labels: {len(labels)}")  
  
# Перевірка відповідності міток компаніям  
if len(labels) != len(valid\_names):  
 print("Невідповідність між кількістю міток та компаній.")  
else:  
 # Виведення компаній у кожному кластері  
 for i in range(num\_clusters):  
 cluster\_members = np.array(valid\_names)[labels == i]  
 if cluster\_members.size > 0:  
 print(f"Кластер {i + 1} =>", ', '.join(cluster\_members))  
 else:  
 print(f"Кластер {i + 1} порожній.")  
  
# Візуалізація кластерів  
plt.figure(figsize=(10, 8))  
  
# Перевірка, чи розмір міток відповідає розміру даних  
if X.shape[0] == len(labels):  
 for i in range(num\_clusters):  
 members = X[labels == i]  
 if members.size > 0:  
 plt.scatter(members.mean(axis=1), np.zeros\_like(members[:, 0]) + i, label=f"Кластер {i + 1}")  
else:  
 print("Невідповідність між даними та мітками кластерів. Неможливо побудувати графік.")  
  
plt.title("Кластери фондового ринку на основі методу поширення подібності")  
plt.xlabel("Нормалізована різниця котирувань")  
plt.yticks(range(num\_clusters), [f"Кластер {i + 1}" for i in range(num\_clusters)])  
plt.legend(loc="best")  
plt.show()

Виконання програми:







Посилання на ГітХаб: <https://github.com/KaidanovychBohdan/SystemOfAI>

**Висновок**: в ході виконання лабораторної роботи опрацював спеціалізовані бібліотеки та мову програмування Python дослідити методи неконтрольованої класифікації даних у машинному навчанні.