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

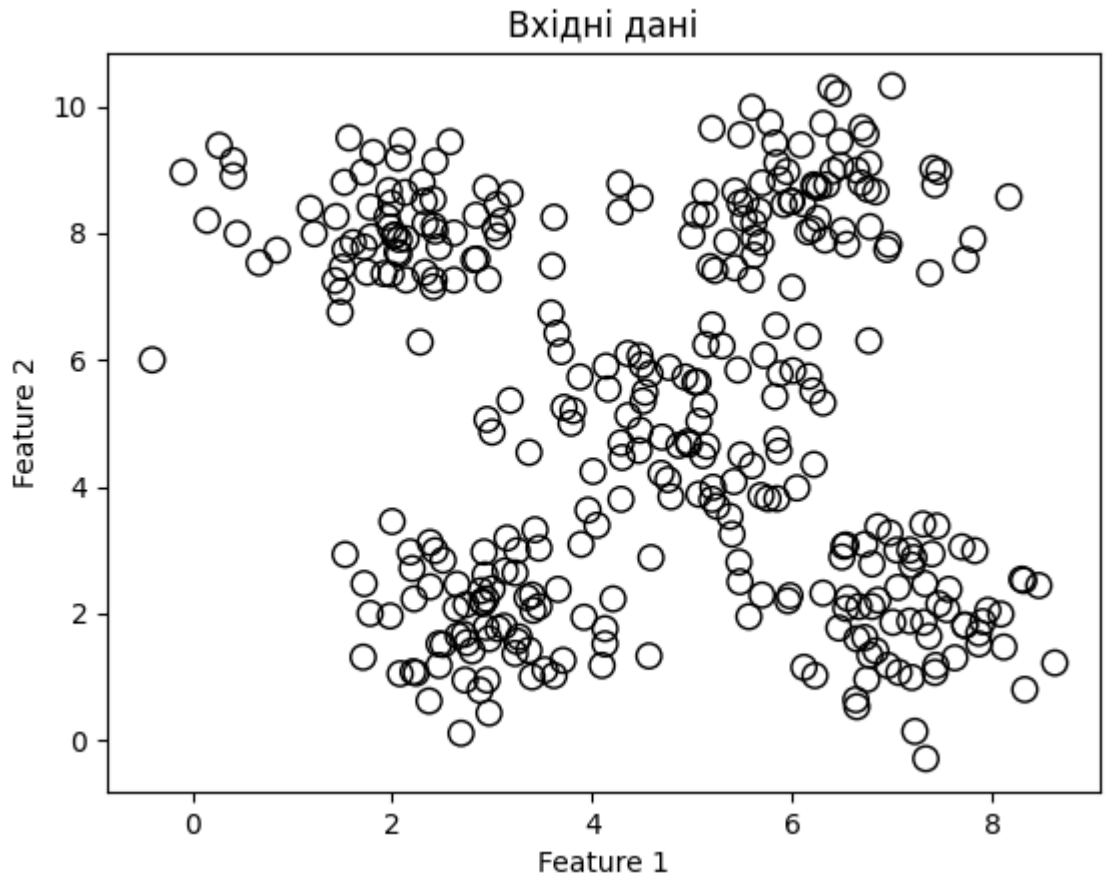
Лістинг програми:

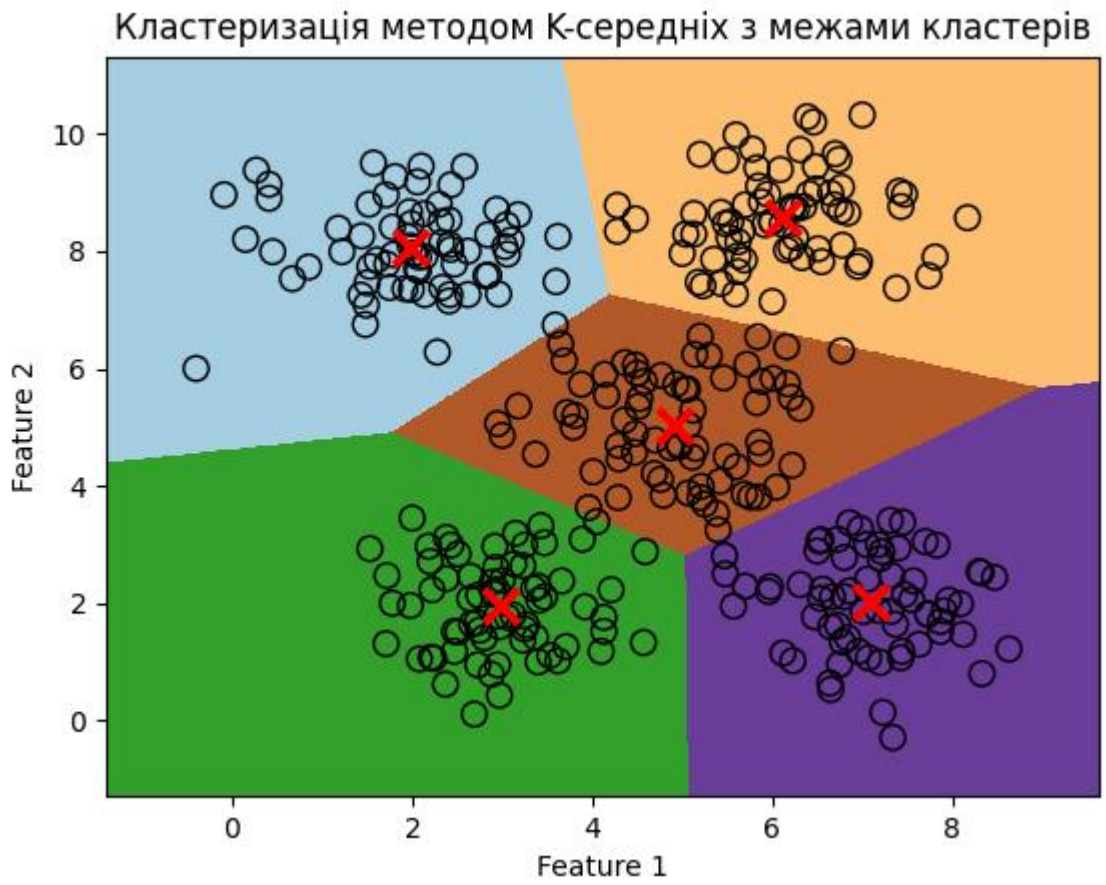
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score

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 К-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:

```
Середній коефіцієнт силуета: 0.59

Process finished with exit code 0
```





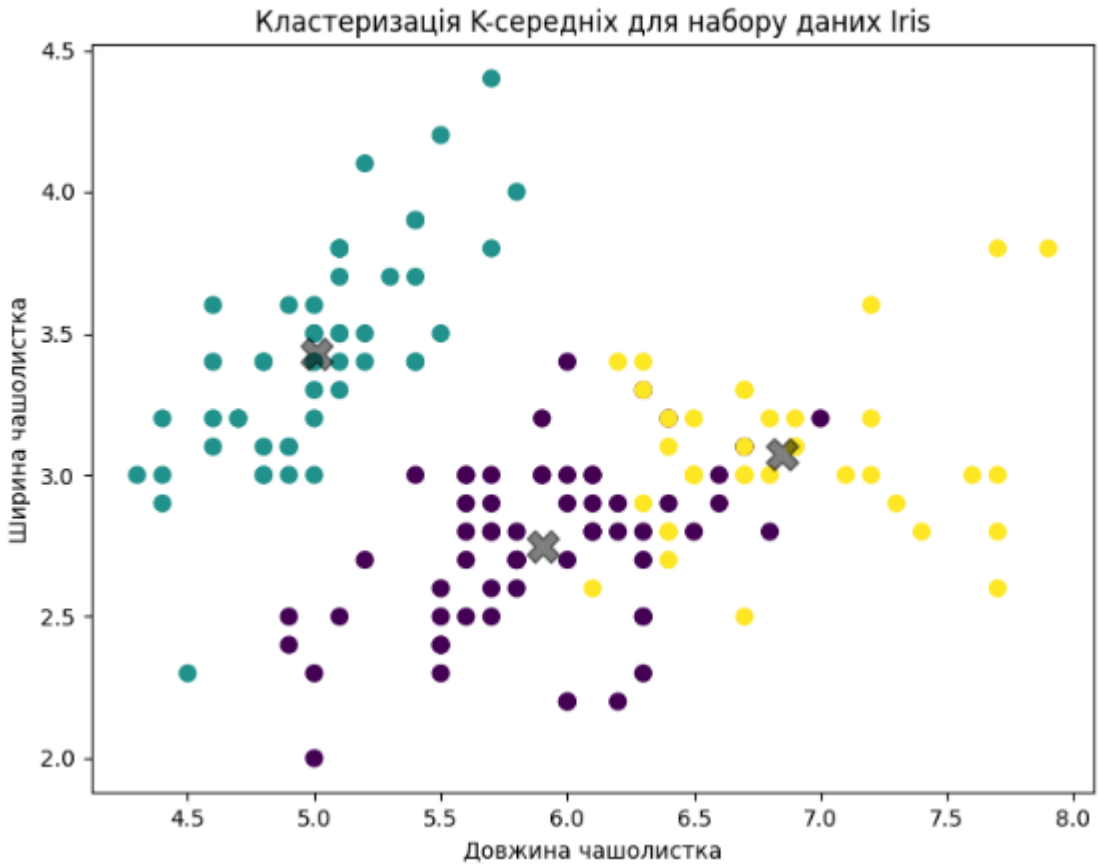
Завдання 2.2. Кластеризація K-середніх для набору даних Iris

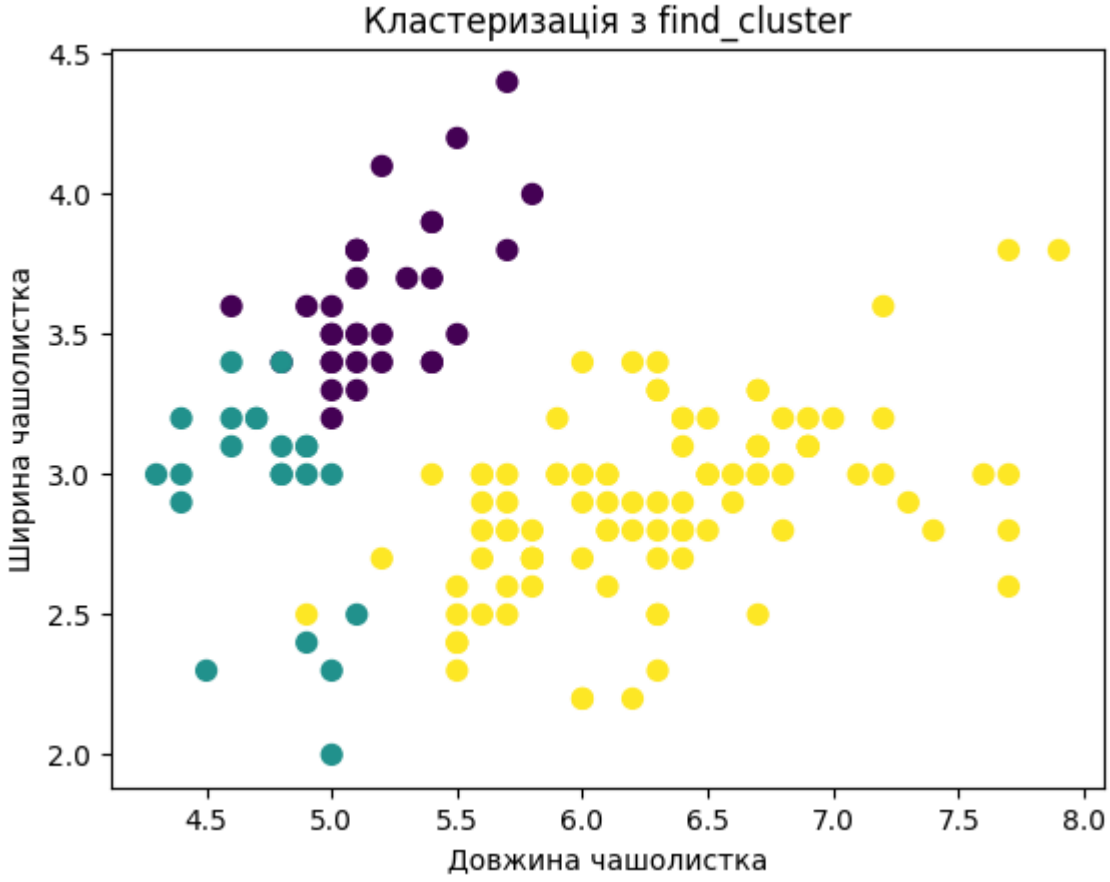
Лістинг програми:

```
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 квіті
тів)
    init='k-means++',    # Використання методу ініціалізації k-means++ для швидшої
збіжності
    n_init=10,           # Кількість запусків алгоритму з різними початковими цент
роїдами
    max_iter=300,        # Максимальна кількість ітерацій для одного запуску алго
ритму
    tol=0.0001,          # Допустима похибка для зупинки алгоритму
    random_state=42      # Встановлюємо random_state для відтворюваності результа
тів
)

# 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 К-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:


Кластеризація K-середніх для набору даних Iris
Ширина чашолистка
Довжина чашолистка
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0


Кластеризація з find_cluster
Ширина чашолистка
Довжина чашолистка
4.5
4.0
3.5
3.0
2.5
2.0
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0

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:

```
Центри кластерів:
 [[2.95568966 1.95775862]
 [7.20690909 2.20836364]
 [2.17603774 8.03283019]
 [5.97960784 8.39078431]
 [4.99466667 4.65844444]]

Кількість кластерів у вхідних даних = 5
```

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

Лістинг програми:

```python
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]
```

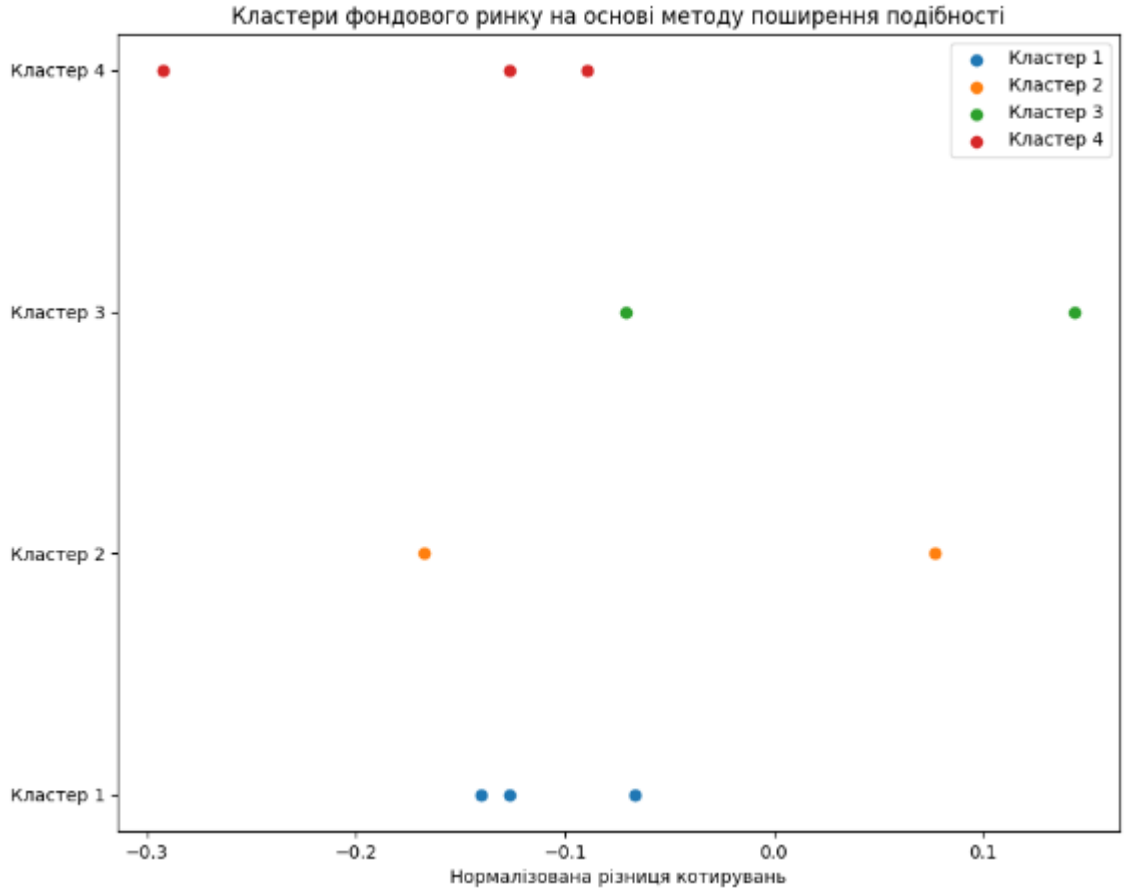
```
names = [item["Name"] for item in data]

# 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:

```
Кількість компаній: 10
Розмір X: (10, 30)
```

```
Кількість кластерів: 4

Розмір labels: 10
Кластер 1 => Microsoft Corp., Alphabet Inc., Meta Platforms Inc.
Кластер 2 => Apple Inc., Amazon.com Inc.
Кластер 3 => JPMorgan Chase & Co., Visa Inc.
Кластер 4 => Tesla Inc., NVIDIA Corp., Walmart Inc.
```



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.