МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра «Системний аналіз та обчислювальна математика»
(повне найменування кафедри)

Пояснювальна записка

до дипломної роботи
магістра
(ступінь вищої освіти)

на тему Система побудови та аналізу взаємозв'язків багатопараметричних даних за допомогою графічних моделей
(назва теми)

Виконав: студент 2 курсу, групи 813м

Спеціальності 124 Системний аналіз
(код і найменування спеціальності)

Освітня програма (спеціалізація)
«Інтелектуальні технології та прийняття рішень в складних системах»

СІРЕНКО Р.В.
(ПРІЗВИЩЕ та ініціали)

Керівник ТЕРЕЩЕНКО Е.В.
(ПРІЗВИЩЕ та ініціали)

Рецензент ЛОЗОВСЬКА Л.І.
(ПРІЗВИЩЕ та ініціали)

2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут, факультет Факультет комп'ютерних наук і технологій
Кафедра «Системний аналіз та обчислювальна математика»
Ступінь вищої освіти магістра
Спеціальність 124 Системний аналіз
(код і найменування)
Освітня програма (спеціалізація) «Інтелектуальні технології та прийняття рішень в складних системах»
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
Еліна ТЕРЕЩЕНКО
«16» грудня 2024 року

ЗАВДАННЯ
НА ДИПЛОМНУ РОБОТУ СТУДЕНТА

СІРЕНКУ Руслану Володимировичу
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема роботи Система побудови та аналізу взаємозв'язків багатопараметричних даних за допомогою графічних моделей, System for building and analyzing relationships in multi-parameter data using graph models

керівник роботи Терещенко Еліна Валентинівна
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від «20» листопада 2024 року №480
2. Строк подання студентом роботи 17грудня 2024
3. Вихідні дані до роботи статистичні дані польового експерименту, що агреговані в електронних таблицях MS Excel та надані лабораторією генетичних досліджень Інституту олійних культур НААН України
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Обґрунтування актуальності проблеми; аналіз методів кластеризації, нормалізації та візуалізації багатопараметричних даних; розробка програмного застосунку; реалізація алгоритмів; тестування розробленого застосунку на реальних даних та формулювання висновків.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) презентація

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Прийняв виконане завдання
Розділ 1, Розділ 2	ТЕРЕЩЕНКО Е.В., в.о. завідувача кафедри	21.10.2024	17.12.2014
Нормоконтроль	ШИРОКОРАД Д.В., доцент	20.12.2024	20.12.2024

7. Дата видачі завдання «21» жовтня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір методів класифікації та візуалізації.	1 тиждень	Розділ 1
4	Робота з Excel файлами та базою даних.	2 тиждень	Розділ 2
5	Реалізація алгоритмів та методів.	3-6 тиждень	Розділ 2
6	Реалізація візуалізації даних.	6-8 тиждень	Розділ 2
7	Аналіз на основі візуалізації	9 тиждень	Розділ 2
8	Оформлення пояснювальної записки та відповідної документації.	10 тиждень	Додатки
9	Нормоконтроль та рецензування.	11 тиждень	
10	Захист дипломної роботи.	12 тиждень	

Студент _____ СІРЕНКО Р.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ ТЕРЕЩЕНКО Е.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Дипломна робота: 100 с., 30 рис., 20 джерел, 1 додаток.

Об’єкт дослідження – багатопараметричні дані, що містять числові та строкові характеристики.

Предмет дослідження – практична реалізація застосунку для аналізу та візуалізації даних.

Мета роботи – створення програмного продукту, здатного ефективно працювати з багатопараметричними даними, забезпечуючи їх аналіз, візуалізацію та надання інструментів для розуміння взаємозв’язків між об’єктами.

Методи дослідження – методи класифікації, методи теорії графів, алгоритми кластеризації, методи вимірювання відстаней, методів зниження розмірності, методів попередньої обробки даних.

Актуальність проблеми – існує потреба у створенні легковісного застосунку, здатного працювати з таблицями MS Excel, що відповідають необхідному формату, і забезпечувати швидкий та якісний аналіз даних. Такий інструмент може значно підвищити ефективність роботи з великими масивами інформації у наукових та практичних дослідженнях.

Задачі дослідження: аналіз вимог до програмного забезпечення для роботи з багатопараметричними даними; розробка інтерфейсу для налаштувань параметрів; реалізація алгоритмів класифікації об'єктів, алгоритмів нормалізації числових та строкових даних, алгоритмів кластеризації з налаштуваннями параметрів; створення функціоналу для візуалізації кластеризованих даних; тестування застосунку на реальних даних.

Результати роботи та їх новизна – розроблено програмний продукт, який дозволяє аналізувати дані, застостовуючи кластеризацію та візуалізацію за допомогою графів та графіків, з можливістю гнучкого налаштування

параметрів. Впроваджено алгоритми нормалізації для різнотипних даних та створено ефективний механізм виявлення зв’язків між об’єктами.

Основні висновки – система є дієвим інструментом для роботи з багатопараметричними даними. Вона забезпечує ефективну кластеризацію, надає зручні засоби візуалізації та дозволяє швидко отримувати зрозумілі результати. Система може застосовуватись у дослідженнях різних галузей і адаптуватись під конкретні вимоги користувача, що робить її універсальним рішенням для аналізу даних.

БАГАТОПАРАМЕТРИЧНІ ДАНІ, КЛАСТЕРИЗАЦІЯ, АНАЛІЗ ДАНИХ, НОРМАЛІЗАЦІЯ, ВІЗУАЛІЗАЦІЯ, C#, K-MEANS, DBSCAN, HAC.

ЗМІСТ

ЗАВДАННЯ 2

РЕФЕРАТ 4

ЗМІСТ 6

ВСТУП 8

1 ТЕОРЕТИЧНА ЧАСТИНА 10

1.1 Характеристика та особливості багатопараметричних даних 10

1.2 Аналіз вимог до системи та особливості роботи з багатопараметричними даними 11

1.3 Кластеризація як метод виявлення закономірностей у багатопараметричних даних 13

1.4 Метрики для вимірювання відстаней у кластеризації 15

1.4.1 Евклідова відстань 15

1.4.2 Манхеттенська відстань 16

1.4.3 Косинусна відстань 17

1.4.4 Відстань Хеммінга 19

1.5 Алгоритми кластеризації 20

1.5.1 Алгоритм K-середніх 20

1.5.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise) .. 22

1.5.3 Агломеративний ієрархічний алгоритм (HAC) 24

1.6 Алгоритми зменшення розмірності 26

1.7 Графічні моделі для візуалізації взаємозв’язків 28

1.8 Вибір технологій розробки 30

2 ПРАКТИЧНА ЧАСТИНА 33
2.1 Постановка задачі 33
2.2 Структура розробленого застосунку 34
2.2.1 Вхідні дані 34
2.2.2 Інтеграція Excel-файлів як джерела вхідних даних 35
2.2.3 Структура бази даних 37
2.2.4 Відображення завантажених даних у застосунку 39
2.3 Порівняння об’єктів у датасеті 41
2.4 Візуалізація результатів порівняння 43
2.5 Реалізація алгоритмів кластеризації 46
2.6 Візуалізація результатів кластеризації 48
2.7 Тестування застосунку на основі даних Fragile States Index 51
2.8 Тестування застосунку на основі даних про сорти сої 62
ВИСНОВКИ 69
ПЕРЕЛІК ПОСИЛАНЬ 71
ДОДАТОК А. ПРОГРАМНІ КОДИ 73

ВСТУП

Аналіз багатопараметричних даних є ключовим інструментом у сучасній науці та аграрній галузі, дозволяючи отримувати нові знання про взаємозв’язки між об’єктами, їхні характеристики та закономірності. Особливого значення такі методи набувають у селекції сільськогосподарських культур, де ефективне використання даних сприяє розробці нових сортів із покращеними властивостями. У цьому контексті важливим завданням є створення інструментів, які спрощують обробку великих масивів інформації, забезпечуючи її структурування, аналіз і візуалізацію.

Робота з великими обсягами даних потребує спеціалізованих програмних засобів, здатних швидко й точно опрацьовувати інформацію, виділяти ключові закономірності та представляти їх у зрозумілій формі. Традиційні методи обчислень, що використовуються в аграрній науці, можуть бути недостатньо ефективними через обмеженість у масштабуванні та обробці складних багатопараметричних структур. Тому застосування сучасних цифрових інструментів, які об'єднують алгоритми машинного навчання, методи кластеризації та графічну візуалізацію, є важливою складовою сучасного наукового підходу.

Теоретична основа такого підходу включає принципи математичної статистики, аналізу даних та інформаційного моделювання. Вибір ефективних способів групування даних, пошуку подібностей і виявлення відхилень залежить від коректності обраної моделі та її параметрів. Окрім цього, візуалізація даних відіграє важливу роль у представленні результатів, адже саме вона дозволяє вченим і практикам інтерпретувати отримані висновки.

Практична частина цього дослідження спрямована на розробку програмного застосунку для роботи з багатопараметричними даними, представленими у форматі Excel. Метою є побудова системи, здатної

аналізувати багатопараметричні дані, виявляти схожості між ними, а також представляти результати у вигляді графічних моделей. Такий підхід дозволяє наочно продемонструвати взаємозв’язки між об’єктами, що спрощує подальше дослідження отриманих даних і прийняття рішень на їхній основі.

Одним із ключових елементів аналізу є кластеризація, що дозволяє групувати схожі об’єкти. Використання цього методу дає змогу виділяти класи або групи об’єктів із подібними параметрами, що є надзвичайно важливим для селекційної роботи. Важливість інтеграції кластеризації у програмний інструмент зумовлена необхідністю забезпечити гнучкість і точність аналізу залежно від поставлених завдань.

Графічна візуалізація результатів є ще одним важливим аспектом роботи програми. Зв’язки між об’єктами, представлені у графічному вигляді, дозволяють швидко й інтуїтивно оцінити ступінь подібності між ними, виокремити групи або індивідуальні об’єкти з унікальними характеристиками. Такий підхід сприяє не лише науковому аналізу, а й практичному застосуванню результатів, наприклад, у плануванні подальших досліджень чи виробничих процесів.

Розробка цього застосунку є не лише частиною дослідницької роботи, а й практичним інструментом, створеним спеціально для вирішення поставлених завдань. Програма дозволяє автоматизувати обробку даних, значно скорочуючи час і зусилля, необхідні для аналізу великих таблиць, а також забезпечує інтерактивність і зручність у роботі.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Характеристика та особливості багатопараметричних даних

Багатопараметричні дані — це дані, які містять кілька вимірюваних характеристик або параметрів для кожного об’єкта або спостереження. Такі дані можуть бути як числовими, так і строковими, і часто використовуються для аналізу складних явищ, де кожен об’єкт може бути описаний за допомогою кількох різних ознак. Наприклад, в контексті біологічних досліджень це можуть бути параметри фенотипу рослин, у маркетингу — характеристики поведінки споживачів, а в географії — просторові координати та інші фактори. Особливістю багатопараметричних даних є їх складність, оскільки велика кількість параметрів може ускладнювати розуміння взаємозв’язків між об’єктами та вимагати застосування спеціальних методів аналізу.

Один із важливих аспектів багатопараметричних даних — це наявність різних типів параметрів. Вони можуть бути числовими (наприклад, вимірювання температури, маси, часу), строковими (наприклад, кольори, класи, статуси) або змішаними. Числові дані, як правило, піддаються математичному аналізу та обробці, що дозволяє застосовувати різноманітні методи статистичного аналізу та кластеризації. Строкові дані, в свою чергу, вимагають спеціальних підходів для обробки, таких як методи перетворення строкових значень у числові (наприклад, через one-hot encoding), що дозволяє інтегрувати їх у загальні алгоритми аналізу [1].

Іншою характерною рисою багатопараметричних даних є їх високий рівень вимірювальної складності. Кількість параметрів у таких наборах даних може бути дуже великою, що призводить до так званої "прокляття вимірювальної розмірності". Це означає, що зі збільшенням кількості параметрів зростає складність обробки та аналізу даних [2]. Оскільки багато з цих параметрів можуть бути взаємно залежними, виникає необхідність у

застосуванні спеціальних методів, таких як зниження розмірності, що дозволяють зменшити кількість змінних без значних втрат інформації.

Ще однією особливістю багатопараметричних даних є їх гетерогенність. В одному наборі даних можуть співіснувати різні типи значень, що потребує спеціальних методів для нормалізації та стандартизації. Числові дані можуть мати різні одиниці виміру та масштаби, що може ускладнювати їх порівняння. Строкові ж дані можуть бути неструктурованими і мати різні рівні значень (наприклад, "високий", "середній", "низький"). Для коректного аналізу таких даних необхідно застосовувати техніки попередньої обробки, які забезпечують їх узгодженість та дозволяють правильно інтегрувати різні типи даних в одну аналітичну модель [3].

1.2 Аналіз вимог до системи та особливості роботи з багатопараметричними даними

Створення системи для аналізу та візуалізації багатопараметричних даних потребує врахування специфічних вимог до програмного забезпечення. Однією з основних вимог є зручність роботи користувача із системою. Інтерфейс повинен бути інтуїтивно зрозумілим, з чіткою навігацією та налаштуваннями, які дозволяють виконувати завдання без попереднього навчання чи знань у галузі програмування. Особливу увагу слід приділити можливості завантаження даних у форматі MS Excel, що є одним із найпоширеніших форматів для зберігання багатопараметричних таблиць.

Важливим аспектом є підтримка різноманітних алгоритмів кластеризації. Користувач має мати можливість обрати необхідний алгоритм залежно від своїх потреб і налаштувати його параметри. Це дозволить забезпечити точність аналізу для даних із різними структурами та характером розподілу.

Окрім алгоритмів кластеризації, система повинна підтримувати методи для побудови графів і візуалізації результатів. Графи забезпечують наочне представлення взаємозв’язків між об’єктами, що особливо корисно для виявлення схожих груп або окремих аномальних об'єктів. Разом із графами система має забезпечувати побудову графіків, які дозволяють візуалізувати багатопараметричні дані у двовимірному просторі. Графіки дозволяють краще зрозуміти розподіл об’єктів у вибірці, їхню приналежність до кластерів. Подібні візуалізації є важливим доповненням до графів, оскільки вони надають альтернативний погляд на структуру даних, допомагаючи користувачеві приймати обґрунтовані рішення.

Система також має враховувати вимоги до обробки великих обсягів даних. Оптимізація обчислювальних процесів і зниження витрат ресурсів є критичними для роботи з багатопараметричними таблицями, які можуть містити тисячі об'єктів і десятки параметрів. Для цього можуть використовуватись спеціальні методи зниження розмірності, які дозволяють зменшити обсяг даних без втрати важливих закономірностей.

Окремою вимогою є забезпечення універсальності системи, тобто її адаптивності до роботи з різними типами даних. Це включає підтримку як числових, так і строкових параметрів, а також можливість використання різних метрик для порівняння об’єктів. Наприклад, для числових даних це може бути евклідова або манхеттенська відстань, тоді як для строкових — коефіцієнт Жаккара чи відстань Левенштейна.

Таким чином, система для роботи з багатопараметричними даними повинна бути багатофункціональною, зручною у використанні та ефективною з точки зору обробки великих обсягів інформації. Врахування цих вимог дозволить створити програмний продукт, який буде корисним як для дослідників, так і для користувачів із мінімальним досвідом роботи з аналізом даних.

1.3 Кластеризація як метод виявлення закономірностей у багатопараметричних даних

Кластеризація є одним із основних методів аналізу багатопараметричних даних, що дозволяє виявляти приховані структури в наборі даних. Цей метод об'єднує об'єкти, що мають подібні характеристики, в окремі групи або кластери, які відрізняються один від одного. Кластеризація не передбачає наявність міток класів, тому вона відноситься до методів навчання без учителя [4]. За її допомогою можна виділити схожі групи даних, що дає змогу краще зрозуміти структуру та взаємозв'язки між об'єктами.

Основний принцип кластеризації полягає в тому, щоб об'єкти, що належать до одного кластеру, були якомога більш схожими між собою за певними параметрами, а між об'єктами з різних кластерів була велика різниця [5]. Вибір методу кластеризації залежить від типу даних та задачі, яку необхідно вирішити. Кластеризація широко використовується в різних галузях, зокрема в аграрних дослідженнях, де важливо групувати об'єкти (наприклад, сорти рослин) за схожістю їхніх характеристик для подальшого аналізу та прийняття рішень.

Існує кілька основних підходів до кластеризації. Один із найпоширеніших — це кластеризація на основі відстані, коли об'єкти групуються за мірою їхньої схожості, що визначається через певні метрики відстані (наприклад, евклідову відстань або манхеттенську відстань). Вибір метрики відстані має велике значення, оскільки різні метрики можуть по-різному оцінювати схожість об'єктів.

Вибір метрики відстані значно впливає на результат кластеризації. Для числових даних зазвичай використовуються евклідова або манхеттенська відстань, оскільки ці метрики дозволяють ефективно вимірювати схожість між об'єктами з різними числовими характеристиками. У випадку текстових даних або даних із високою дисперсією параметрів більше підходить косинусна

відстань, оскільки вона фокусується на напрямку векторів, а не на їхніх величинах. Для роботи з бінарними даними або текстами, де важлива кількість відмінностей між об'єктами, застосовується відстань Хеммінга.

Ще один популярний підхід — це ієрархічна кластеризація, яка передбачає побудову ієрархії кластерів, де на кожному етапі об'єкти або кластери об'єднуються на основі певної міри подібності. Такий підхід дає можливість отримати не тільки кластери, але й візуалізувати відносну схожість між об'єктами за допомогою дендрограм, що є корисним для глибшого розуміння структури даних [6].

Крім того, існують алгоритми, які орієнтовані на виявлення кластерів з різною формою і розмірами. Наприклад, метод DBSCAN (Density-Based Spatial Clustering of Applications with Noise) дозволяє знаходити кластери, що мають різну густину, і виділяти «шум» (непідходящі або аномальні об'єкти). Це особливо корисно для роботи з великими і складними наборами даних, де є багато шуму або незвичайних об'єктів.

Кластеризація може бути використана для виявлення закономірностей у даних, які не завжди очевидні на перший погляд. Наприклад, в аграрних дослідженнях вона дозволяє виділити групи схожих сортів рослин або різних станів ґрунту, що мають спільні характеристики. Це може допомогти у прогнозуванні результатів, таких як врожайність, або в оптимізації аграрних процесів, таких як вибір сортів для певних умов [7].

У випадку багатопараметричних даних кластеризація також дозволяє виявити структури в даних, які мають велику кількість змінних, що ускладнюють традиційні методи аналізу. Завдяки кластеризації можна обробляти й аналізувати складні масиви даних і знаходити зв'язки між об'єктами, які не були б очевидними при простому порівнянні.

У цілому, кластеризація є потужним інструментом для виявлення закономірностей у даних, що робить її важливою складовою частиною аналізу багатопараметричних даних у різних галузях, включаючи агрономію та інші

сфери, де важливо ефективно обробляти і аналізувати великі обсяги різноманітних даних.

1.4 Метрики для вимірювання відстаней у кластеризації

1.4.1 Евклідова відстань

Евклідова відстань є однією з найбільш поширених метрик для вимірювання відстаней між двома точками в евклідовому просторі. Вона визначає прямолінійну відстань між двома точками як довжину найкоротшого шляху, що з'єднує ці точки. Для двох точок у двовимірному просторі (x_1, y_1) і (x_2, y_2) евклідова відстань обчислюється за формулою [8]:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

У загальному випадку для n-вимірного простору евклідова відстань між точками $P_1(x_1, x_2, \dots , \ x_n)$ і $P_2(y_1, y_2, \dots , \ y_n)$ визначається за формулою:

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Така метрика відображає геометричну відстань між точками в просторі характеристик і є найпоширенішою для числових даних.

Евклідова відстань є природною метрикою для багатьох задач кластеризації, оскільки вона пропонує просту та інтуїтивно зрозумілу модель для визначення подібності між об'єктами. Вона добре працює, коли дані мають природну геометричну інтерпретацію і коли важливе саме відстань між

точками у просторі ознак. Наприклад, у задачах кластеризації, що передбачають аналіз числових характеристик, евклідова відстань часто є оптимальним вибором.

Однак, евклідова відстань може мати деякі обмеження, зокрема, коли дані мають різні масштаби або одиниці вимірювання. В такому випадку одні ознаки можуть значно домінувати в розрахунках відстані, що спотворює результати кластеризації. Для уникнення цієї проблеми перед кластеризацією часто проводиться нормалізація або стандартизація даних.

Крім того, евклідова відстань втрачає свою ефективність при роботі зі строковими або змішаними типами даних, де не має чіткого геометричного значення для відстані між точками. У таких випадках можуть використовуватись інші метрики відстаней, що краще підходять для дискретних або категоріальних даних.

1.4.2 Манхеттенська відстань

Манхеттенська відстань, також відома як відстань таксі або L1-норма, є альтернативною метрикою для вимірювання відстаней між об'єктами в просторі ознак. Вона визначає суму абсолютних різниць між координатами двох точок. Ця метрика отримала свою назву завдяки аналогії з пересуванням по прямокутній міській мережі вулиць, де найкоротший шлях між двома точками проходить лише по горизонталі та вертикалі.

Для двох точок у двовимірному просторі (x_1, y_1) і (x_2, y_2) манхеттенська відстань обчислюється за формулою [8]:

$$d = |x_2 - x_1| + |y_2 - y_1|$$

У загальному випадку для n-вимірного простору манхеттенська відстань між точками $P_1(x_1, x_2, \dots, x_n)$ і $P_2(y_1, y_2, \dots, y_n)$ визначається за формулою:

$$d = \sum_{i=1}^{n} |x_i - y_i|$$

На відміну від евклідової відстані, манхеттенська не враховує діагональні напрями, вимірюючи відстань тільки вздовж осей. Це робить її особливо корисною у задачах, де переміщення по діагоналі не має фізичного сенсу або є обмеженим.

Манхеттенська відстань менш чутлива до великих відхилень в окремих ознаках у порівнянні з евклідовою, оскільки вона враховує лише абсолютні значення різниць між координатами. Це дозволяє уникнути значного впливу аномальних значень на результат. Вона також підходить для роботи з даними різних масштабів, хоча нормалізація може бути необхідною для забезпечення коректних результатів.

Ця метрика широко використовується у задачах кластеризації, зокрема для аналізу даних у міських умовах, оптимізації маршрутів або задачах, де параметри змінюються незалежно один від одного. Манхеттенська відстань також ефективно працює з числовими та строковими характеристиками, після попередньої обробки даних.

1.4.3 Косинусна відстань

Косинусна відстань — це метрика, яка використовується для вимірювання схожості між двома об'єктами, представленими векторами в

багатовимірному просторі. Її особливість полягає в тому, що вона враховує лише напрямок векторів, ігноруючи їх довжину. Це робить косинусну відстань ефективною у випадках, коли важлива відносна структура даних, а не абсолютні значення.

Косинусна відстань ґрунтується на косинусі кута між двома векторами. Якщо θ — це кут між векторами A і B, косинусна схожість обчислюється за формулою [8]:

$$sim = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Косинусна відстань визначається як доповнення косинусної схожості до 1:

$$d = sim - 1$$

Головною перевагою косинусної відстані є її здатність оцінювати схожість об'єктів, навіть якщо їхні значення мають значну різницю в масштабах. Наприклад, якщо два об'єкти мають однакові співвідношення між характеристиками, але різні абсолютні значення, косинусна відстань покаже, що вони схожі.

Косинусна відстань широко використовується у задачах кластеризації, пов’язаних з аналізом текстових даних, обробкою зображень, або рекомендаційними системами. Вона дозволяє ефективно групувати об'єкти на основі їхньої орієнтації у просторі ознак, що особливо важливо у випадках, коли абсолютні значення даних є несуттєвими.

Наприклад, у рекомендаційних системах косинусна відстань застосовується для порівняння профілів користувачів або продуктів. Це

дозволяє визначати спільні інтереси або подібність між продуктами, навіть якщо кількість оцінок або переглядів суттєво різниться.

Попри свої переваги, косинусна відстань має деякі обмеження. Вона не враховує абсолютні різниці між значеннями векторів, що може бути критичним для задач, де важлива відстань у просторі ознак. Крім того, для обчислення метрики дані повинні бути представлені у вигляді числових векторів, що вимагає додаткової підготовки у випадках роботи зі строковими ознаками.

Також слід враховувати, що косинусна відстань може давати некоректні результати, якщо вектори містять нульові значення в усіх координатах.

1.4.4 Відстань Хеммінга

Відстань Хеммінга є однією з популярних метрик, що використовується для вимірювання відмінностей між двома об'єктами однакової довжини. Вона визначається як кількість позицій, у яких відповідні елементи двох векторів відрізняються. Ця метрика найбільше підходить для роботи з векторами, що містять строкові або дискретні дані, такі як біти, символи або категорії.

Формула для обчислення відстані Хеммінга між двома векторами A і B [8]:

$$d = \sum_{i=1}^{n} \delta(A_i, B_i),$$

де

$$\delta(A_i, B_i) = \begin{cases} 0, & \text{якщо } A_i = B_i \\ 1, & \text{якщо } A_i \neq B_i \end{cases}.$$

Метрика відстані Хеммінга відзначається простотою обчислення, оскільки вона потребує лише порівняння відповідних елементів векторів. Її особливість полягає в тому, що вона є чутливою до позицій елементів: якщо хоча б один елемент у певній позиції відрізняється, це одразу впливає на результат. Важливою умовою використання цієї метрики є те, що вектори повинні мати однакову довжину.

Застосування цієї метрики є широким у багатьох сферах, наприклад, для порівняння строкових даних, виявлення помилок у двійкових кодах чи аналізу генетичних послідовностей. Її простота й ефективність роблять її корисним інструментом у задачах, пов’язаних з аналізом дискретних або строкових даних.

1.5 Алгоритми кластеризації

1.5.1 Алгоритм K-середніх

Алгоритм K-середніх є одним із найпоширеніших методів кластеризації, який використовується для поділу набору багатопараметричних даних на K кластерів. Основна ідея алгоритму полягає у мінімізації сумарної квадратичної відстані між кожним об'єктом і центром його кластера. Центри кластерів, які називаються центроїдами, представляють середні значення об’єктів у відповідному кластері [9].

Загальний принцип роботи алгоритму:

1. Ініціалізація: Вибирається K початкових центроїдів випадковим чином або за певною стратегією.

2. Призначення: Кожен об’єкт даних призначається до найближчого центроїда на основі обраної метрики відстані (зазвичай евклідової).

3. Оновлення: Для кожного кластера обчислюється новий центроїд як середнє значення параметрів усіх об’єктів у цьому кластері.

4. Повторення: Процес призначення та оновлення повторюється до досягнення зупинки — коли центроїди перестають змінюватися, або досягнуто максимальної кількості ітерацій.

Для реалізації алгоритму з використанням обчислювальної формули $O(i \cdot n \cdot k \cdot d)$ часову складність можна пояснити таким чином, n — кількість об'єктів у наборі даних, k — кількість кластерів, d — кількість параметрів (розмірність даних), i — максимальна кількість ітерацій [10].

Переваги алгоритму:

– простота реалізації: Алгоритм зрозумілий і легкий у використанні.

– Висока швидкість: Завдяки порівняно невеликій обчислювальній складності алгоритм є досить швидким навіть для великих наборів даних.

– Широка сфера застосування: Використовується у багатьох практичних задачах, таких як сегментація зображень, аналіз клієнтів, класифікація документів.

Недоліки:

– залежність від початкових умов: Початковий вибір центроїдів впливає на кінцевий результат, що може призвести до локальних мінімумів.

– Фіксована кількість кластерів: Користувач повинен заздалегідь вказати K, що може бути проблематичним, якщо оптимальна кількість кластерів невідома.

– Чутливість до шуму: Алгоритм погано працює з даними, що містять шуми або аномальні значення, оскільки ці об'єкти можуть суттєво впливати на положення центроїдів.

1.5.2 DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

DBSCAN є алгоритмом кластеризації, який класифікує об'єкти в залежності від їх щільності у просторі. Він дозволяє знаходити групи об'єктів, що мають високу щільність, та виділяти шуми, тобто об'єкти, які не належать жодному кластеру[11]. На відміну від алгоритму K-середніх, DBSCAN не потребує визначення кількості кластерів заздалегідь. Це робить його корисним для аналізу даних, де неможливо точно передбачити кількість груп.

Основні параметри алгоритму DBSCAN: ε (eps): максимальна відстань, на якій два об'єкти можуть бути взаємно сусідніми для формування одного кластера, minPts: мінімальна кількість точок, які повинні бути в околі об'єкта для того, щоб він міг бути визнаний центральним об'єктом кластера.

Алгоритм працює наступним чином:

1. Для кожного об'єкта визначається його «ε-окружність», тобто набір об'єктів, що знаходяться в межах відстані ε.

2. Якщо кількість об'єктів у цій окружності перевищує значення minPts, то об'єкт вважається «ядром» кластера.

3. Інші об'єкти, що знаходяться в межах цього ядра, додаються до кластера.

4. Об'єкти, які не можуть бути включені в жоден кластер через недостатню кількість сусідів (тобто менше ніж minPts в межах їх ε-окружності), позначаються як шуми.

5. Процес продовжується до тих пір, поки не будуть перевірені всі об'єкти.

Часова складність алгоритму DBSCAN в найгіршому випадку складає $O(n^2 \cdot d)$, n — кількість об'єктів, d — кількість параметрів.

Це пов'язано з необхідністю обчислювати відстані між кожною парою точок для визначення їх відносин у просторі. Оскільки алгоритм порівнює

кожен об'єкт з іншими об'єктами для визначення його сусідів, а також враховує всі виміри для кожного об'єкта, то складність обчислення відстаней між точками залежить як від їх кількості, так і від кількості параметрів. Висока складність може стати значним обмеженням при роботі з великими наборами даних [11].

Переваги:

– не потрібно вказувати кількість кластерів**:** DBSCAN самостійно визначає кількість кластерів на основі щільності, що дає перевагу в задачах з невідомою кількістю груп.

– Виявлення аномалій: Алгоритм може виділяти шуми та аномальні об'єкти, що є корисним при роботі з реальними даними, де частина елементів може не належати до жодної групи.

– Гнучкість в обробці форм кластера: На відміну від K-середніх, який обмежений до круглих або овальних кластерів, DBSCAN може знаходити кластери будь-якої форми, оскільки він базується на щільності точок.

Недоліки:

– вибір параметрів: Точний вибір значень ε та minPts може бути складним і залежить від характеру даних. Невірний вибір цих параметрів може призвести до поганої класифікації.

– Часова складність: У випадку великого об'єму даних та високої кількості вимірів (параметрів), часова складність алгоритму може бути дуже великою, що обмежує його застосування в великих наборах даних.

– Чутливість до масштабу даних: Якщо дані мають різні масштаби або одиниці вимірювання, це може вплинути на точність класифікації, оскільки відстань ε та minPts залежать від масштабу.

1.5.3 Агломеративний ієрархічний алгоритм (HAC)

Агломеративна ієрархічна кластеризація є методом кластеризації, який працює за принципом знизу-вгору, починаючи з окремих точок (кожна точка розглядається як окремий кластер) і поступово об'єднуючи найближчі кластери. Один з популярних підходів до агломеративної ієрархічної кластеризації — це метод UPGMA (Unweighted Pair Group Method with Arithmetic Mean), який використовується для обчислення відстаней між кластерами. Цей метод визначає відстань між двома кластерами як середнє арифметичне відстаней між усіма парами точок з різних кластерів [12]. Такий підхід дозволяє ефективно будувати ієрархію кластерів на основі схожості між ними.

Процес роботи алгоритму:

1. Ініціалізація: Спочатку кожен об'єкт є окремим кластером, тобто на першому етапі кожна точка вважається кластером сама по собі.

2. Обчислення відстаней між кластерами: Для кожної пари кластерів обчислюється відстань за допомогою середнього арифметичного відстаней між усіма парами точок з різних кластерів.

3. Об'єднання найближчих кластерів: Після того як всі відстані між кластерами обчислені, вибираються два найближчі кластери, які об'єднуються в новий кластер.

4. Повторення кроків: Після кожного об'єднання відстані між новим кластером та іншими кластерами обчислюються знову. Процес об'єднання триває до тих пір, поки не буде досягнуто бажаного числа кластерів або поки не буде виконано порогове значення (threshold).

5. Формування ієрархії: Кроки об'єднання та обчислення відстаней повторюються, що дозволяє створити ієрархічне дерево кластерів.

Часова складність алгоритму становить $O(n^3 \cdot d)$, n — кількість об'єктів, d — кількість параметрів.

Це пояснюється тим, що на кожному кроці обчислюється відстань між усіма парами точок $\left(O(n^2)\right)$, і кожне об'єднання вимагає перерахунку відстаней для нових кластерів. Крім того, процес об'єднання триває до тих пір, поки всі точки не будуть згруповані в один кластер, що також вимагає повторного обчислення відстаней.

Переваги:

– простота реалізації: Алгоритм досить простим для розуміння та реалізації, що робить його доступним для використання в різних прикладних задачах.

– Не потребує попереднього визначення кількості кластерів: Алгоритм автоматично побудує ієрархічну структуру кластерів, дозволяючи користувачеві вибрати оптимальне число кластерів в залежності від потрібного порогового значення.

– Гнучкість: Алгоритм дозволяє використовувати різні метрики для обчислення відстаней між точками, що робить його більш гнучким у різних контекстах.

Недоліки:

– висока обчислювальна складність: Враховуючи необхідність обчислення відстаней між всіма парами точок і порівняння кожного з нових кластерів, алгоритм може стати дуже повільним при великій кількості об'єктів. Це може обмежувати його застосування на великих наборах даних.

– Чутливість до шуму та аномалій: Як і інші методи кластеризації, UPGMA може бути чутливим до шуму, і аномальні об'єкти можуть вплинути на результати класифікації.

– Лінійні форми кластерів: Хоча цей метод може використовуватись для класифікації кластерів різної форми, він може мати труднощі з класифікацією складних форм кластерів, де враховуються не тільки відстані між точками.

1.6 Алгоритми зменшення розмірності

Зменшення розмірності є важливою технікою для роботи з великими наборами даних, де кількість параметрів може бути занадто великою для ефективного аналізу чи візуалізації. Цей процес дозволяє скоротити кількість змінних, зберігаючи при цьому основну інформацію про структуру даних. Зменшення розмірності часто застосовується перед кластеризацією чи іншими методами аналізу, оскільки дозволяє покращити ефективність обчислень і знизити ризик перенавчання моделей.

Зазвичай зменшення розмірності здійснюється за допомогою лінійних або нелінійних методів. Одним з найпоширеніших підходів є метод головних компонент (PCA), який дозволяє зменшити розмірність, зберігаючи основну варіативність даних.

Метод головних компонент (PCA) використовує лінійне перетворення, щоб проєктувати дані в нову просторову систему, де нові змінні (компоненти) є лінійними комбінаціями початкових. Основною ідеєю є те, що перші кілька компонент часто пояснюють більшу частину варіації в даних, тоді як решта компонент можуть містити лише незначну частину інформації.

Алгоритм PCA складається з кількох основних етапів:

1. Спочатку необхідно відцентрувати дані, віднявши середнє значення кожної ознаки від усіх значень цієї ознаки.

2. Далі обчислюється коваріаційна матриця для всіх змінних, яка показує, як варіація кожної пари ознак корелює між собою.

3. Після цього знаходяться власні вектори та власні значення цієї матриці. Власні вектори визначають напрямки, в яких відбувається найбільша варіація даних, а власні значення — це величини, що вказують на важливість кожного з напрямків.

4. Вибираються перші кілька головних компонент, що мають найбільші власні значення, і на їх основі формується новий простір для даних.

Переваги:

– збереження варіації: Вибір головних компонент дозволяє зберегти найбільшу частину інформації про варіацію в даних, що робить метод ефективним для зменшення розмірності.

– Зниження складності: Оскільки кількість компонент зменшується, це знижує складність подальших етапів аналізу, таких як кластеризація або класифікація.

– Простота реалізації: PCA є добре вивченим і широко використаним методом, з наявністю численних бібліотек для його застосування.

Недоліки:

– лінійність: Оскільки PCA є лінійним методом, він може не ефективно працювати в ситуаціях, де існують складні нелінійні залежності між ознаками.

– Втрати інтерпретації: Нові компоненти не завжди мають чітке інтерпретаційне значення, що може ускладнити розуміння результатів.

– Чутливість до масштабу: PCA чутливий до масштабу даних, тому перед застосуванням необхідно масштабувати або нормалізувати ознаки.

Алгоритм PCA має часову складність $O(n^3)$, n — кількість ознак (параметрів). Це зумовлено необхідністю обчислення коваріаційної матриці та виконання лінійної алгебри для пошуку власних значень і векторів. Тому, хоча PCA є потужним інструментом для зменшення розмірності, його застосування на великих наборах даних з багатьма параметрами може вимагати значних обчислювальних ресурсів.

В результаті, PCA є дуже ефективним методом для зменшення розмірності, особливо в задачах, де варіація в даних може бути описана лінійними комбінаціями параметрів. Однак для задач з нелінійними залежностями можуть бути корисні інші методи.

1.7 Графічні моделі для візуалізації взаємозв’язків

Використання графів для візуалізації взаємозв'язків між об'єктами є потужним інструментом для представлення складних даних у зручному та наочному вигляді. Графи дозволяють абстрагуватися від деталей, зберігаючи при цьому важливу інформацію про взаємозв'язки між елементами, що робить їх ідеальними для аналізу та інтерпретації багатопараметричних даних. Зв'язки між об'єктами можуть бути відображені як ребра між вузлами, що дозволяє швидко визначати залежності та шукати закономірності.

У контексті нашого застосунку графічні моделі використовуються для відображення зв'язків між різними об'єктами на основі їхніх характеристик та параметрів. Кожен об'єкт, що аналізується, представлений вузлом графу, а зв'язки між об'єктами — ребрами, що з'єднують відповідні вузли [13]. Визначення і відображення цих зв'язків дає можливість візуально спостерігати за структурою даних і на основі цього робити висновки про їхню схожість або відмінності.

Переваги графічного представлення даних:

– інтуїтивність: Графи є зрозумілими навіть для користувачів без глибоких технічних знань. Залежності та зв'язки між об'єктами чітко видимі, що полегшує інтерпретацію результатів.

– Покращена візуалізація: Графи дозволяють представити складні взаємозв'язки в компактному вигляді, де кожен вузол і ребро відображають важливу інформацію. Це особливо корисно, коли кількість параметрів і об'єктів велика.

– Гнучкість: Графічні моделі можна адаптувати під різні типи даних і завдання, а також використовувати для інтерактивної візуалізації, де користувач може самостійно досліджувати і маніпулювати даними.

– Аналіз зв'язків: За допомогою графів можна виявляти найбільш значущі зв'язки між об'єктами, що дозволяє проводити кластеризацію, виявляти групи схожих об'єктів або визначати важливі параметри для подальшого аналізу.

У випадку, коли кластеризація вже здійснена і мітки кластерів присвоєні кожному об'єкту, можна побудувати граф, в якому кожен вузол буде позначений відповідною кластерною міткою. Це дозволяє чітко відобразити, як об'єкти групуються в різні кластери і які зв'язки існують між ними на основі схожості. Візуалізація таких графів допомагає користувачам і дослідникам наочно бачити розподіл об'єктів за кластерами та оцінювати якість кластеризації.

Цей тип графів може бути корисним для виявлення певних тенденцій у даних, таких як класи з найбільшою кількістю об'єктів, а також для аналізу взаємозв'язків між об'єктами з різних кластерів [14]. Графи з кластерними мітками можуть бути додатково підсвічені за допомогою різних кольорів або розмірів вузлів, що ще більше полегшує сприйняття та інтерпретацію результатів кластеризації.

Після зниження розмірності даних за допомогою методу головних компонент (PCA), можна побудувати графіки, що відображають розташування об'єктів у новому просторі з меншою кількістю параметрів. Оскільки PCA дозволяє зберегти основні компоненти варіації в даних, графіки, побудовані в результаті зниження розмірності, допомагають наочно продемонструвати структуру даних на основі перших кількох компонент.

Зазвичай зниження розмірності здійснюється до двох або трьох компонент, що дозволяє візуалізувати дані в двовимірному або тривимірному просторі. Кожен об'єкт на графіку буде представлений точкою, і її розташування залежатиме від значень компонент, які були отримані в результаті PCA. Візуалізація таких графіків дозволяє дослідникам зрозуміти, як об'єкти згруповані в новому просторі, а також виявити можливі аномалії

або недосконалості в даних, які не були помічені в оригінальному просторі з високою розмірністю.

Для графіків після зниження розмірності можна також використовувати кольорові мітки або кластерні мітки, що додають додаткову інтерпретацію до отриманих результатів. Ці графіки особливо корисні для візуалізації складних наборів даних, де зменшення розмірності дозволяє зрозуміти їх структуру, що може бути недоступним в просторі з великою кількістю параметрів.

Графічні моделі є незамінним інструментом для візуалізації багатопараметричних даних і результатів кластеризації. Вони допомагають наочно побачити структуру даних, взаємозв'язки між об'єктами, а також оцінити ефективність кластеризації. Завдяки таким візуалізаціям, можна з легкістю виявляти закономірності, що спрощує подальший аналіз даних і прийняття рішень у різних областях досліджень та застосувань.

1.8 Вибір технологій розробки

Для реалізації програмного продукту обрано сучасні та надійні інструменти, які забезпечують ефективну розробку та впровадження програмного рішення. Вибір технологій залежав від специфіки завдання, вимог до інтеграції з різними компонентами та зручності в роботі з даними, а також необхідності створення зручного та функціонального інтерфейсу користувача. Нижче описано кожну з обраних технологій.

.NET Framework обрано для розробки програмного продукту, оскільки це одна з найбільш стабільних і широко використовуваних платформ для створення Windows-додатків [15]. .NET надає розробникам великий набір бібліотек і компонентів, що дозволяє швидко реалізувати функціональність програми, зокрема для роботи з даними, графічними елементами та логікою

додатку. Платформа має чудову підтримку як для великих, так і для малих проєктів, що дозволяє забезпечити високу продуктивність та масштабованість програми [16].

Для реалізації користувацького інтерфейсу обрано Windows Forms, що є однією з найбільш зручних технологій для створення класичних віконних додатків на платформі Windows. Windows Forms дозволяє швидко та ефективно створювати інтерфейс з використанням стандартних елементів управління, таких як кнопки, текстові поля, таблиці та графічні компоненти. Це дає змогу створити інтуїтивно зрозумілий і функціональний UI, що відповідає вимогам користувача.

Entity Framework Core (EF Core) вибрано для роботи з базами даних завдяки своїй гнучкості та високому рівню абстракції. EF Core дозволяє працювати з даними в об'єктно-орієнтованому стилі, що значно спрощує процес обробки даних та їх збереження в базах даних без необхідності писати складні SQL-запити. Це дозволяє ефективно працювати з великими обсягами даних, зберігаючи при цьому високу продуктивність і зрозумілість коду.

ADO.NET використовується для більш низькорівневого доступу до даних і виконання специфічних запитів, що потребують високої точності або складної логіки. Ця технологія забезпечує гнучкість у взаємодії з базою даних, дозволяючи виконувати прямі SQL-запити та працювати з даними в реальному часі, що важливо для обробки і аналізу великих наборів даних.

Для побудови графів і візуалізації взаємозв'язків між об'єктами в системі обрано бібліотеку Microsoft Msagl. Msagl дозволяє створювати та візуалізувати графи, що є необхідним для виявлення зв'язків між багатопараметричними даними [17]. Ця технологія забезпечує автоматичне розташування вузлів і ребер графу, що допомагає створити зрозумілі та ефективні візуалізації, необхідні для аналізу і кластеризації даних.

Для зберігання і обробки даних вибрано SQL як мову запитів та SQL Server Management Studio (SSMS) як інструмент для управління базою даних. SQL є стандартною мовою для роботи з реляційними базами даних і дозволяє

ефективно виконувати складні запити, що необхідні для обробки багатопараметричних даних [18]. SSMS є потужним інструментом для адміністрування баз даних, виконання запитів, моніторингу стану сервера та керування даними, що робить його ідеальним вибором для ефективної роботи з великими обсягами інформації.

Для зниження розмірності даних у проєкті було обрано ML.NET, оскільки цей фреймворк забезпечує зручні інструменти для виконання задач з машинного навчання, включаючи методи зменшення розмірності, зокрема PCA [19]. ML.NET дозволяє інтегрувати алгоритми машинного навчання безпосередньо в середовище .NET [20], що робить його ідеальним вибором для застосування в проєктах на платформі .NET.

Для побудови точкових діаграм було обрано бібліотеку ScottPlot, оскільки вона надає потужні та гнучкі інструменти для візуалізації даних, зокрема для створення різноманітних типів діаграм, включаючи точкові. ScottPlot є зручною бібліотекою для .NET [21], яка дозволяє швидко створювати інтерактивні та високоякісні графіки, що є необхідним для візуалізації багатопараметричних даних.

Обрані технології забезпечують надійність, високу продуктивність та зручність у розробці програмного продукту. .NET Framework і Windows Forms забезпечують ефективне створення додатка з інтуїтивно зрозумілим інтерфейсом, а Entity Framework Core та ADO.NET — потужні можливості для роботи з даними. Microsoft Msagl надає необхідні інструменти для побудови графів і візуалізації результатів кластеризації, а SQL і SQL Server Management Studio дозволяють ефективно управляти базами даних. Всі ці інструменти разом створюють потужне середовище для розробки та впровадження програмного продукту, що відповідає вимогам щодо аналізу багатопараметричних даних.

2 ПРАКТИЧНА ЧАСТИНА

2.1 Постановка задачі

Метою розробки є створення програмного продукту, здатного ефективно працювати з багатопараметричними даними, забезпечуючи їх аналіз, візуалізацію та надання інструментів для розуміння взаємозв’язків між об’єктами. Така система повинна не лише спрощувати обробку складних наборів інформації, але й допомагати користувачеві у виявленні прихованих закономірностей і структур у даних.

Для досягнення цієї мети передбачається виконання наступних завдань:

– завантаження даних. Застосунок повинен дозволяти користувачеві завантажувати багатопараметричні дані у зручному форматі, а також забезпечувати їхнє налаштування для подальшого аналізу. Цей процес включає розпізнавання структури даних, їхнє збереження в базі даних та забезпечення доступу до них у подальшій роботі.

– Обробка даних. Необхідно реалізувати механізми роботи з параметрами, які можуть мати різну природу (числові, текстові тощо). Це забезпечить універсальність застосунку та його здатність адаптуватися до різних типів інформації.

– Порівняння об’єктів. Важливим завданням є розробка інструментів для аналізу схожості між об’єктами, враховуючи різні налаштування, що можуть змінюватися залежно від потреб користувача. Порівняння повинно бути точним і гнучким, дозволяючи враховувати специфіку кожного параметра.

– Візуалізація даних. Окремий акцент робиться на графічному представленні даних, адже саме візуалізація є ключем до швидкого та інтуїтивного розуміння інформації. Візуалізація має відображати зв’язки між

об’єктами, виділяти групи схожих об’єктів та дозволяти користувачеві легко інтерпретувати результати.

– Кластеризація даних. Для виділення груп об’єктів із подібними характеристиками необхідно інтегрувати алгоритми кластеризації, які дозволять автоматизувати процес аналізу даних і зробити його більш інформативним.

2.2 Структура розробленого застосунку

2.2.1 Вхідні дані

Основними вхідними даними для застосунку є таблиці, надані Інститутом олійних культур Національної академії аграрних наук України (ІОК НААН), що містять інформацію про сорти сої. Дані були представлені у форматі Excel, що дозволяє зручно зберігати та обробляти структуровану інформацію.

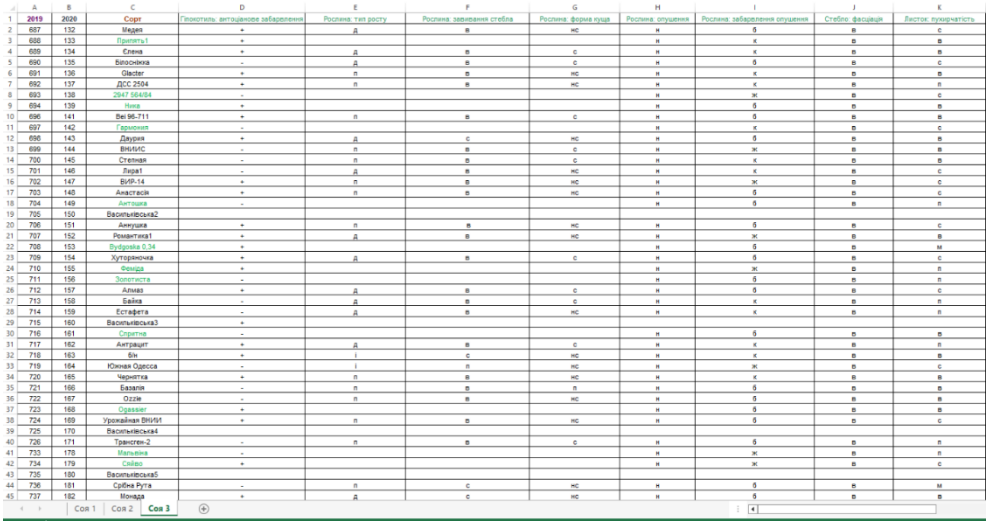
Вхідні дані складаються з трьох таблиць. Перша таблиця (рис. 2.1) містить 139 об'єктів і 12 числових параметрів, друга таблиця включає 80 об'єктів та також 12 числових параметрів.

	A	B	C	D	E	F	G	H	I
1	№ делянки	Рослина: за висотою	Стебло: за товщиною	Стебло: кількість пилок	Стебло: за висотою прикріплення нижнього бобу	Стебло кількість вузлів	Біб: за довжиною	Біб: за шириною	Біб: кількість бобів
2	Васильківська	55	0.76	5.2	5	5.2	4.38	0.94	80.8
3	Медея	60	0.64	4	11.6	4	3.6	0.74	72.8
4	Прип'ять1	74	0.82	1.4	7.4	4	5.2	0.84	45.8
5	Елена	128	1	1.8	6	1.8	3.8	0.9	90
6	Білоскіжка	102.4	0.98	6	11.4	6	4.56	0.8	122.8
7	Glacier	115.8	0.98	5	11.2	5	4.74	0.84	82.4
8	ДС 3504	128.2	0.92	4	15	4	4.56	0.92	91.4
9	2947 564/84	118	0.76	2	20.8	2	4.5	0.74	54.8
10	Ніка	122.2	1.04	4.4	9.2	4.4	4.14	0.88	135.8
11	Bei 96-711	82	0.96	3.6	191	3.6	4.2	0.8	136.4
12	Гармонія	60	1.08	3.2	5	3.2	5	1.2	95.6
13	Даурія	101.4	0.94	5.4	6.2	5.4	3.36	0.76	103
14	ВНИИС 1	93.2	0.76	3.8	6.8	3.8	4.4	0.8	79.4
15	Степная-90	101	1.08	3.2	6.6	3.2	3.98	0.76	103.4
16	ВИР-14	98	0.96	3.8	6.4	3.8	5.06	0.86	81
17	Анастасія	99.4	0.86	4.2	9.6	4.2	4.04	0.84	91.2
18	Антошка	84.4	0.92	2	5	2	4.32	0.82	92.6
19	Bydgoska 0,34	96.4	0.98	3	12.6	3.4	4.62	0.9	81.6
20	Кудряночка	96	1.06	5.8	9	9	4.86	1	87
21	Фемідв	77	0.9	3.6	9.8	3.6	4.92	0.9	56.6
22	Золотиста	69.25	0.875	2.75	3.75	2.75	4.3	0.9	59
23	Естафета	66.2	0.76	3.2	9.2	3.2	3.74	0.8	54.8
24	Спритна	89	0.8	2.6	6.4	5.64	4.8	0.92	65.8
25	Антрацит	122	0.86	1.333333333	13.6	1.333333333	4.36	0.78	60.4
26	б/н	133	0.88	5.8	9.4	5.6	4.18	0.92	72.4
27	Южная Одесса	135.2	0.98	2.2	8.2	2.2	4.26	0.76	61.4
28	Чернятка	124	0.76	3.6	13.8	3.6	4.56	0.84	71
29	Базалія	117	0.9	6	16.2	6	4.5	0.9	132.6
30	Ozzie	105	1.02	2.8	12.4	2.8	4.84	0.94	97.6
31	Ogassier	107	0.92	5.6	5.6	5.6	3.86	0.76	93.6
32	Трансген-2	87	1.06	3.4	14	3.4	4.4	0.86	58.4
33	Лира	94.5	1.025	3.75	9	5	3.95	0.8	178.5
34	Несь-1	93.8	0.64	4	14.6	4	3.38	0.82	58.2
35	Луч Надежды	100.4	1.12	5.4	13.2	5.4	4.6	0.8	103.8
36	Октябрь 70	96.8	1.1	5.6	8.6	5.6	4.38	0.84	158.6
37	Соната	147	1.12	2.2	12	2.2	4.28	1	123.2
38	Славия	59.6	1.02	4.8	5.333333333	4.8	3.76	0.86	95.6
39	Мальвіна	97	1.38	7	7.8	7	4.04	0.78	220
40	Сяйво	96	0.92	2.8	6.6	2.8	4.32	0.78	92.6
41	Криниця	81.2	0.94	3.4	11.2	3.4	4.3	0.76	94.4
42	Горлиця	85	0.85	5.5	8.75	5.5	4.075	0.925	95
43	Юг-30	124.6	1.02	3.6	5	3.6	4.36	0.82	111
44	Устя	86	0.9	2.8	13	2.8	4.2	0.8	82.2
45	Особлива	78	1.12	3.2	11	3.2	3.4	0.94	130.4

Сої 1 | Сої 2 | Сої 3

Рисунок 2.1 – Частина даних першої таблиці MS Excel

Третя таблиця (рис. 2.2) містить 151 об'єкт і 24 строкових параметри. З цих параметрів 22 будуть використовуватися для порівняння, а два додаткові параметри (унікальні ідентифікатори 2019 та 2020 років) не є значущими для аналізу і будуть відключені.



2019	2020	Сорт	Глюкотиль: антоціанове забарвлення	Рослина: тип росту	Рослина: забарвлення стебла	Рослина: форма куща	Рослина: опушення	Рослина: забарвлення опушення	Стебло: фасціація	Листок: пухирчатість
687	132	Медея	+	д	в	нс	н	б	в	с
688	133	Притять1	+				н	к	в	в
689	134	Елена	+	д	в	с	н	к	в	в
690	135	Білосніжка	-	д	в	с	н	б	в	с
691	136	Glacter	+	п	в	нс	н	к	в	в
692	137	ДСС 2504	+	п	в	нс	н	к	в	п
693	138	2947 58/494	-				н	ж	в	с
694	139	Нива	+				н	б	в	в
696	141	Bel 96-711	+	п	в	с	н	б	в	в
697	142	Гармония	-				н	к	в	с
698	143	Даурия	+	д	с	нс	н	б	в	в
699	144	ВНИИС	-	п	в	с	н	ж	в	в
700	145	Степная	-	п	в	с	н	к	в	в
701	146	Лира1	-	д	в	нс	н	к	в	с
702	147	ВИР-14	+	п	в	нс	н	ж	в	с
703	148	Анастасія	+	п	в	нс	н	б	в	с
704	149	Антацка	-				н	б	в	п
705	150	Васильківська2								
706	151	Аннушка	+	п	в	нс	н	б	в	с
707	152	Романтика1	+	д	в	нс	н	ж	в	в
708	153	Вуфорка 0.34	+				н	б	в	м
709	154	Хуторяночка	+	д	в	с	н	б	в	с
710	155	Фокуд	+				н	ж	в	п
711	156	Золотиста	-				н	б	в	п
712	157	Алмаз	+	д	в	с	н	б	в	с
713	158	Байка	-	д	в	с	н	к	в	п
714	159	Естафета	-	д	в	нс	н	к	в	п
715	160	Васильківська3	+							
716	161	Спутник	-				н	б	в	в
717	162	Антрацит	+	д	в	с	н	к	в	п
718	163	Бін	+	і	с	нс	н	к	в	в
719	164	Южная Одесса	-	і	п	нс	н	ж	в	с
720	165	Чернятка	+	п	в	нс	н	к	в	в
721	166	Базалія	-	п	в	п	н	б	в	в
722	167	Ozzie	-	п	в	нс	н	б	в	в
723	168	Opassier	+				н	б	в	в
724	169	Урожайная ВНИИ	+	п	в	нс	н	б	в	с
725	170	Васильківська4								
726	171	Трансген-2	-	п	в	с	н	б	в	п
733	178	Мальвіна	-				н	ж	в	п
734	179	Сяйво	+				н	ж	в	с
735	180	Васильківська5								
736	181	Срібна Рута	-	п	с	нс	н	б	в	м
737	182	Монада	+	д	с	нс	н	б	в	в

Соя 1 | Соя 2 | Соя 3

Рисунок 2.2 – Частина даних третьої таблиці MS Excel

Усі таблиці були приведені до стандартизованого вигляду. Перший рядок кожної таблиці містить заголовки стовпців, серед яких обов'язково є стовпець з унікальними назвами об'єктів. Кожен рядок таблиці представляє один об'єкт з його характеристиками. Таким чином, дані були структуровані, що дозволяє застосунку коректно їх обробляти. Ці дані будуть використовуватися в застосунку для подальшого аналізу і побудови графів, кластеризації та порівняння параметрів різних об'єктів.

2.2.2 Інтеграція Excel-файлів як джерела вхідних даних

Однією з ключових функцій розробленого застосунку є забезпечення зручного процесу імпорту багатопараметричних даних із зовнішніх джерел. Для цієї мети було обрано формат Excel, оскільки дані для аналізу надходили

від Інституту олійних культур Національної академії аграрних наук України (ІОК НААН) та були представлені у вигляді таблиць MS Excel. Цей формат є універсальним для зберігання та обробки структурованої інформації і широко використовується у різних галузях.

Інтеграція Excel-файлів у застосунок дозволяє користувачам швидко завантажувати необхідну інформацію для подальшого аналізу. Застосунок автоматично розпізнає структуру файлу, включаючи рядок із назвами параметрів, який може знаходитися у довільній позиції таблиці. Під час імпорту підтримується коректне розпізнавання як числових, так і строкових параметрів, забезпечуючи їх правильну обробку. Окрім цього, програма ефективно працює з великими обсягами даних, що дозволяє проводити аналіз масштабних наборів інформації без втрати продуктивності.

Користувач має можливість обрати стовпець, який містить назви об'єктів, тоді як інші параметри обробляються після завантаження завдяки налаштуванням у застосунку. На етапі імпорту реалізовано перевірку даних для виявлення можливих помилок, таких як відсутність необхідних параметрів, некоректні значення або дублювання назв об'єктів. Особлива увага приділяється перевірці стовпця з назвами об'єктів, який повинен містити унікальні значення. У разі виявлення помилок користувач отримує відповідне повідомлення, що забезпечує високу якість і достовірність завантажених даних.

Для завантаження вхідних даних у програмний застосунок використовується головне меню, у якому доступний пункт меню "Data". Один із пунктів цього меню — "Load Excel" (рис. 2.3).

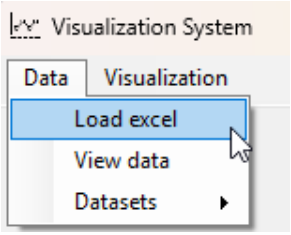


Рисунок 2.3 – Зовнішній вигляд меню “Data”

При натисканні на цей пункт відкривається вікно, що дозволяє обрати Excel-файл для завантаження. Після обрання файлу застосунок починає обробку кожного аркуша окремо, перетворюючи кожний аркуш в окремий датасет, назвою якого є назва листа. Перший заповнений рядок кожного аркуша вважається рядком заголовків. Кожен стовпець зчитується, і далі користувачу пропонується обрати стовпець, який містить унікальні назви об'єктів. Якщо обраний стовпець містить повторювані значення, система показує користувачу повідомлення про помилку з зазначенням назв повторюваних об'єктів. У випадках, коли структура таблиці не відповідає вимогам (наприклад, відсутні заголовки або дані неструктуровані), користувач отримує повідомлення про необхідність форматування файлу (рис. 2.4).

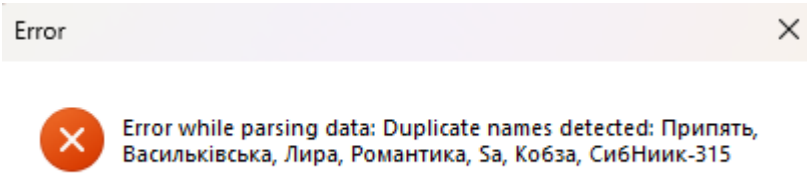


Рисунок 2.4 – Вікно, що повідомляє про помилку зчитування даних

Після успішної обробки дані з кожного аркуша, що відповідають встановленим вимогам, зберігаються в базу даних, а останній завантажений датасет автоматично встановлюється як поточний для подальшого використання.

2.2.3 Структура бази даних

Для забезпечення ефективного зберігання, обробки та аналізу багатопараметричних даних застосунок використовує реляційну базу даних. Її структура розроблена з урахуванням специфіки роботи з

багатопараметричними наборами даних. База даних складається з кількох основних таблиць:

– NodeTables (Таблиці): містить інформацію про завантажені таблиці, включаючи їх унікальні ідентифікатори, назви та мета-дані.

– NodeObjects (Об’єкти): зберігає основні характеристики об’єктів (вузлів), що відповідають записам у таблицях.

– NodeParameters (Параметри об’єктів): містить значення параметрів для кожного об’єкта.

– ParameterTypes (Назви параметрів): зберігає назви параметрів (заголовки) для кожної таблиці.

– Крім основних таблиць, передбачено таблиці для зберігання налаштувань користувача, що забезпечують гнучкість у роботі із датасетами:

– UserSettings (Головні налаштування користувача): містить загальні параметри, такі як допустима похибка для числових даних, мінімальний поріг схожості між об’єктами для створення зв’язків та тип візуалізації (граф, кластеризація тощо).

– ParameterStates (Налаштування параметрів): зберігає інформацію про вагу, статус активності та інші параметри кожного поля.

– ClusterAlgorithmSettings (Налаштування алгоритмів кластеризації): містить параметри алгоритмів кластеризації, які обрав користувач.

Для забезпечення можливості кластеризації даних, вони мають бути приведені до відповідного вигляду. Щоб уникнути повторного перетворення даних під час кожного аналізу, нормалізовані значення зберігаються у базі даних у наступних таблицях:

– NormalizedParameterStates: містить загальну інформацію про параметри, наприклад, тип (числовий або текстовий) та кількість можливих категорій.

– NormalizedParameters: зберігає нормалізовані параметрів.

Структура бази даних (рис. 2.5) розроблена для забезпечення ефективності роботи з великими обсягами даних, їх цілісності та надійного зберігання. Зберігання нормалізованих даних додатково зменшує навантаження на обчислювальні ресурси, що є критично важливим для обробки великих наборів інформації.

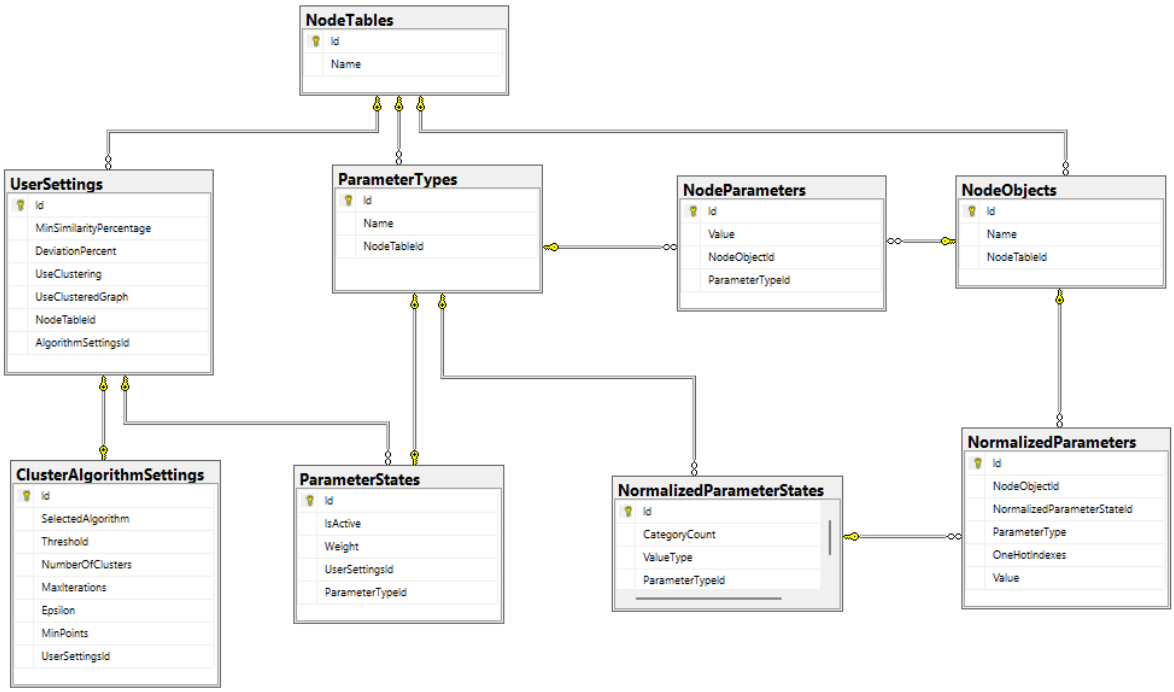
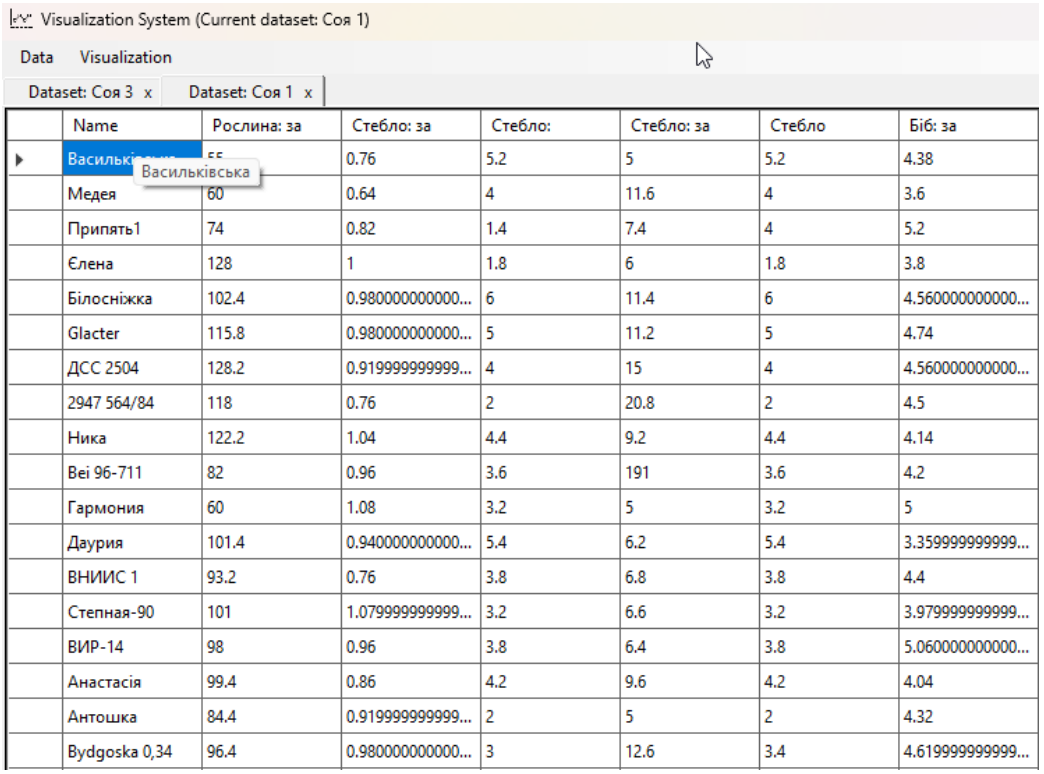


Рисунок 2.5 – Діаграма структури бази даних

2.2.4 Відображення завантажених даних у застосунку

Розділ "Data" в головному меню застосунку надає користувачеві додаткові можливості для роботи з даними, що були завантажені через Excel-файли. Пункт "View data" відкриває нову вкладку в головному вікні програми, де відображаються дані поточного датасету у вигляді таблиці (рис. 2.6). Це дозволяє користувачам переглядати та аналізувати дані безпосередньо в інтерфейсі застосунку.



	Name	Рослина: за	Стебло: за	Стебло:	Стебло: за	Стебло	Біб: за
▶	Василькі...	55	0.76	5.2	5	5.2	4.38
	Медея	60	0.64	4	11.6	4	3.6
	Припять1	74	0.82	1.4	7.4	4	5.2
	Єлена	128	1	1.8	6	1.8	3.8
	Білосніжка	102.4	0.980000000000...	6	11.4	6	4.560000000000...
	Glacter	115.8	0.980000000000...	5	11.2	5	4.74
	ДСС 2504	128.2	0.919999999999...	4	15	4	4.560000000000...
	2947 564/84	118	0.76	2	20.8	2	4.5
	Ника	122.2	1.04	4.4	9.2	4.4	4.14
	Bei 96-711	82	0.96	3.6	191	3.6	4.2
	Гармония	60	1.08	3.2	5	3.2	5
	Даурия	101.4	0.940000000000...	5.4	6.2	5.4	3.359999999999...
	ВНИИС 1	93.2	0.76	3.8	6.8	3.8	4.4
	Степная-90	101	1.079999999999...	3.2	6.6	3.2	3.979999999999...
	ВИР-14	98	0.96	3.8	6.4	3.8	5.060000000000...
	Анастасія	99.4	0.86	4.2	9.6	4.2	4.04
	Антошка	84.4	0.919999999999...	2	5	2	4.32
	Bydgoska 0,34	96.4	0.980000000000...	3	12.6	3.4	4.619999999999...

Рисунок 2.6 – Вкладка з даними датасету “Соя 1”

У таблиці надається можливість фільтрувати інформацію за кожним доступним параметром, що суттєво полегшує процес аналізу, даючи змогу працювати з потрібними даними на основі конкретних критеріїв. Фільтрація підтримує роботу з різними типами даних, як числовими, так і текстовими, що дозволяє користувачеві швидко знаходити необхідну інформацію.

Назва вкладки, в яку відкриваються дані, містить інформацію про її вміст, зокрема вказує на те, що вона відображає дані, а також включає назву датасету. Це дозволяє користувачеві візуально ідентифікувати дані, з якими він працює в поточний момент.

Для більш зручного керування завантаженими наборами даних у меню є пункт "Datasets", що надає доступ до списку всіх датасетів , завантажених у застосунок (рис. 2.7). Це дозволяє користувачеві переглядати всі доступні набори даних та ефективно управляти ними. Для кожного датасету передбачено дві основні дії: обрати набір даних як поточний для подальшого використання в аналізі або видалити набір даних з бази даних застосунку.

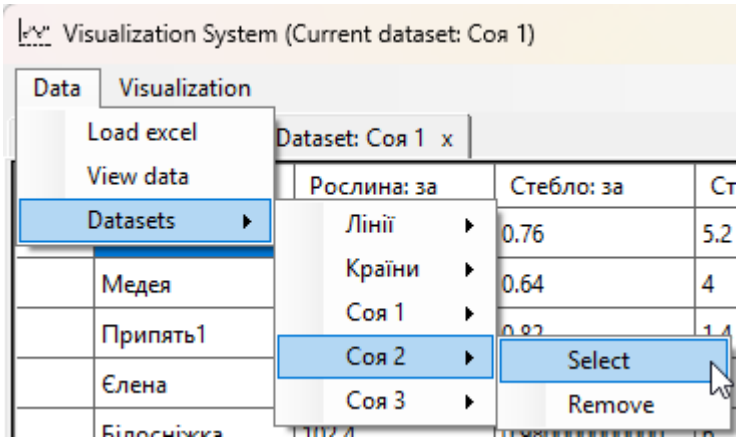


Рисунок 2.7 – Пункт меню «Datasets»

Ці дії забезпечують простоту в управлінні даними, дозволяючи користувачеві швидко перемикатися між різними наборами або видаляти неактуальні.

2.3 Порівняння об’єктів у датасеті

Порівняння об’єктів у датасеті здійснюється за допомогою алгоритму, який виконує повний перебір можливих пар об’єктів та оцінює їх схожість за різними параметрами. Цей процес допомагає виявити родинні зв’язки між об’єктами на основі заданих критеріїв схожості. Основними аспектами порівняння є числові та строкові параметри, а також інші налаштування, такі як ваги параметрів і їх активність.

Числові параметри порівнюються з урахуванням допустимої похибки, що дає можливість визначити межі, в межах яких різниця між значеннями числових параметрів вважається прийнятною. За замовчуванням допустима похибка для числових параметрів становить 10%, але користувач має можливість змінювати цей параметр у налаштуваннях. Це дозволяє більш гнучко налаштовувати критерії порівняння залежно від точності, необхідної для аналізу.

Строкові параметри оцінюються на основі повного збігу значень. Це означає, що якщо значення в обох об’єктах для певного параметра є однаковими, то параметр вважається співпадаючим. Для строкових даних важливо, щоб вони були в однаковому форматі та коректно введені, що забезпечує точність порівняння.

Родинний зв’язок між об’єктами визначається, коли відсоток схожості між об’єктами перевищує мінімальний поріг. За замовчуванням мінімальний поріг схожості для створення родинного зв’язку становить 55%. Цей поріг можна налаштувати в меню параметрів порівняння, що дозволяє в залежності від задачі коригувати рівень схожості для того, щоб визначити, коли об’єкти є достатньо схожими для встановлення зв’язку.

Ваги параметрів відіграють важливу роль у визначенні впливу кожного параметра на результат порівняння. За замовчуванням вага кожного параметра дорівнює 1, але користувач має можливість змінювати ці значення в налаштуваннях, в залежності від важливості конкретного параметра для аналізу. Вага визначає важливість параметра під час порівняння об'єктів. Параметри з вищою вагою мають більший вплив на результат схожості, тобто якщо параметр співпав і має високу вагу, це суттєво покращує загальний результат порівняння. Вага застосовується до кожного активного параметра, і в кінці загальна схожість обчислюється з урахуванням цих ваг.

Цей підхід є визначенням зваженого середнього, де кожному значенню приписується своя вага, і середнє значення обчислюється таким чином, щоб значення з більшою вагою мали більший вплив на кінцевий результат. Це дозволяє акцентувати увагу на найбільш важливих параметрах при порівнянні об'єктів.

Активність параметра дозволяє включати або виключати певні параметри з розрахунків при порівнянні. Це особливо корисно, коли є необхідність зосередитись на певних параметрах і тимчасово ігнорувати інші. За замовчуванням всі параметри активні, але користувач може змінити їх

активність через меню налаштувань, вказавши, які параметри повинні брати участь у порівнянні.

Меню налаштувань (рис. 2.8) у програмі дозволяє зручно керувати всіма цими параметрами, включаючи зміну допустимої похибки для числових параметрів, встановлення мінімального відсотка схожості для визначення родинних зв’язків, а також налаштування ваги та активності кожного параметра. Це дає користувачеві повний контроль над процесом порівняння об’єктів, дозволяючи адаптувати алгоритм під конкретні вимоги та забезпечувати точний аналіз даних.

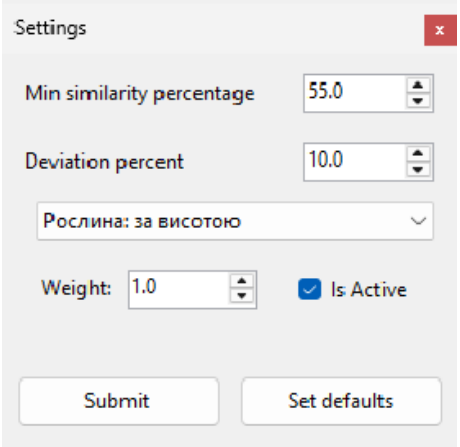


Рисунок 2.8 – Вікно налаштувань

2.4 Візуалізація результатів порівняння

Розділ "Visualization" у головному меню застосунку надає потужні інструменти для візуалізації результатів аналізу даних, що містяться в поточному датасеті. Цей розділ складається з трьох основних підпунктів: "Build graph", "Build plot" та "Settings". Пункт "Build graph" відповідає за створення графа на основі поточного датасету та налаштувань, що були визначені користувачем, тоді як "Settings" відкриває вікно налаштувань.

Коли користувач вибирає пункт "Build graph", застосунок порівнює об'єкти поточного набору даних і на основі результатів створює граф за допомогою бібліотеки Microsoft.Msagl [17]. Граф додається до головного вікна програми як нова вкладка (рис. 2.9). У назві вкладки вказано, що це граф, а також назву датасету, для якого він був побудований. Вершини графа представляють окремі об'єкти датасету. Зв'язки між вершинами відображають родинні зв’язки, що визначаються на основі схожості їх параметрів. Колір ребер варіюється залежно від відсотка схожості між об'єктами: чим більша схожість, тим зеленіше ребро, в той час як низька схожість позначається червоними відтінками. Якщо зв'язок між двома об'єктами не існує (схожість нижча за мінімальний поріг), то таке ребро не будується.

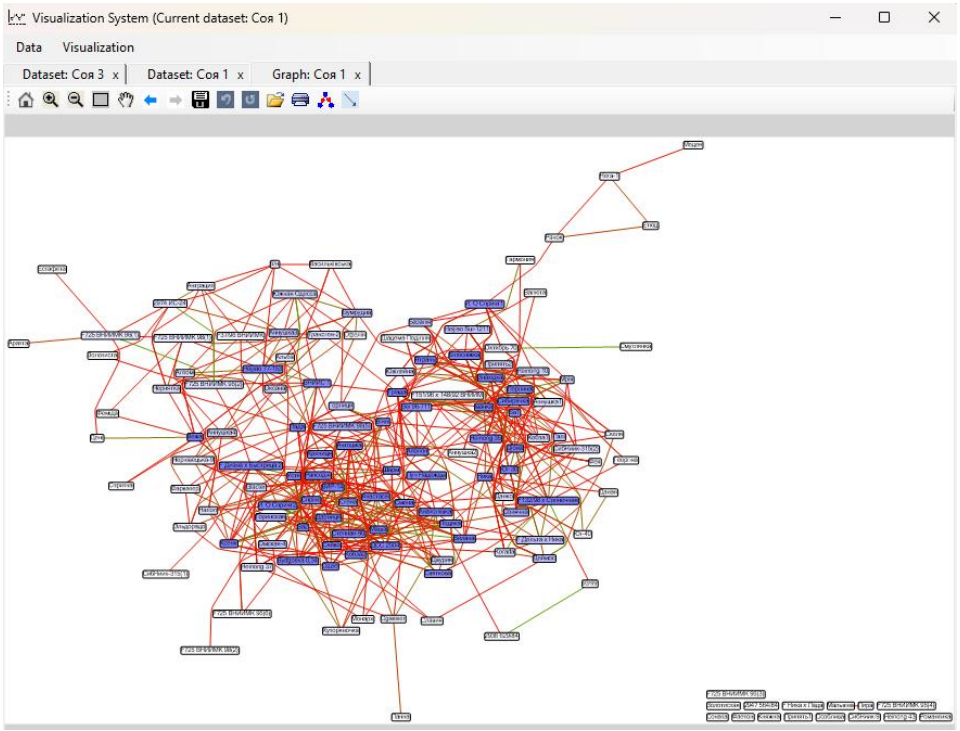


Рисунок 2.9 – Вкладка з побудованим графом для датасету «Соя 1»

Вершини графа можуть змінювати відтінок залежно від кількості зв'язків, що виходять з цієї вершини. Чим більше зв’язків має об'єкт, тим більш фіолетовий відтінок він отримує, що дозволяє легко визначити, які об’єкти мають найбільшу кількість родинних зв’язків. Це полегшує візуальну оцінку

структури зв'язків у графі та дає можливість визначити найбільш важливі або центральні об'єкти.

Застосунок надає користувачеві можливість переглядати деталі кожної вершини. Для цього достатньо подвійно натиснути на будь-яку вершину, що відкриває вікно з детальною інформацією про об’єкт (рис. 2.10).

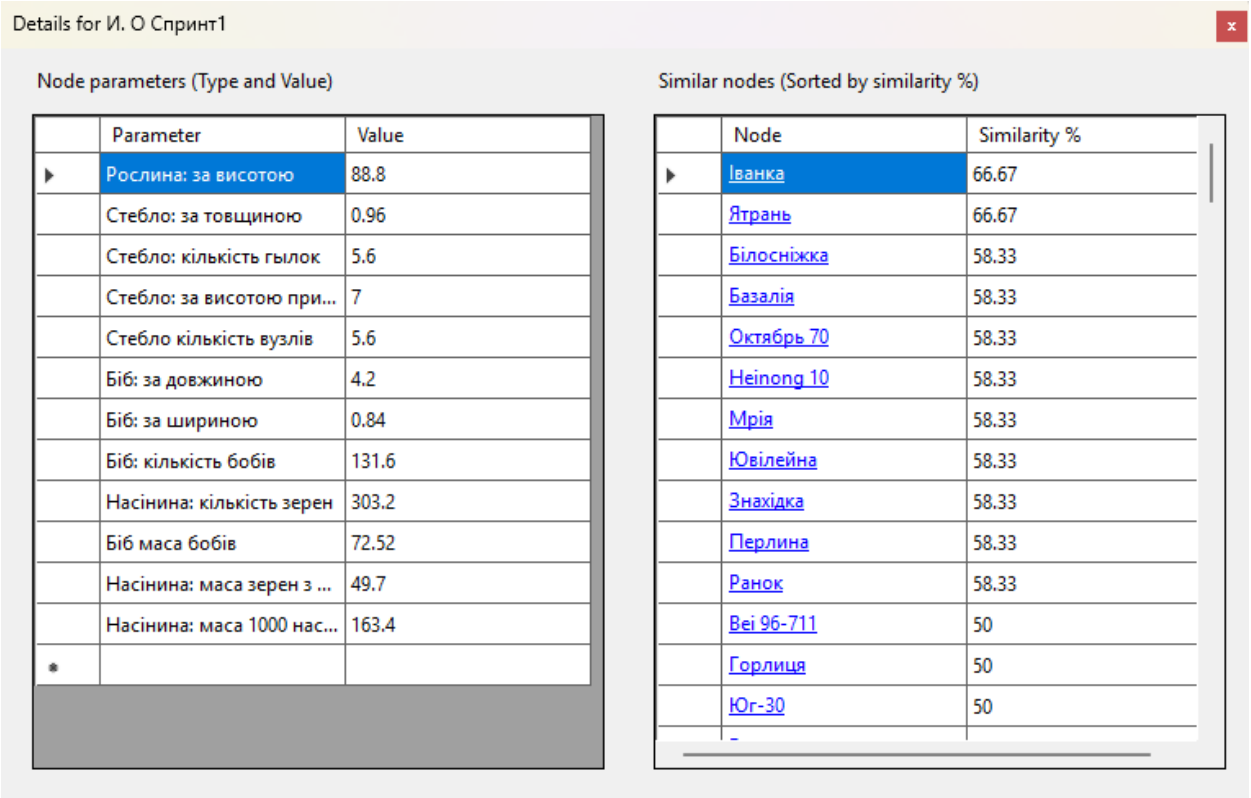


Рисунок 2.10 – Вікно деталей

Вікно містить дві таблиці: перша — з повним списком параметрів об'єкта, а друга — з відсотками схожості між обраним об'єктом та іншими. Ці таблиці підтримують сортування, що дозволяє детально аналізувати зв’язки між об’єктами. Користувач може відкрити кілька таких вікон одночасно для порівняння параметрів різних об'єктів.

2.5 Реалізація алгоритмів кластеризації

У застосунку реалізовано три основні алгоритми кластеризації, кожен з яких можна налаштувати за допомогою параметрів: K-середніх, DBSCAN та агломеративна ієрархічна кластеризація (HAC). Користувач має можливість вибрати один із цих алгоритмів кластеризації через налаштування в інтерфейсі програми (рис. 2.11).

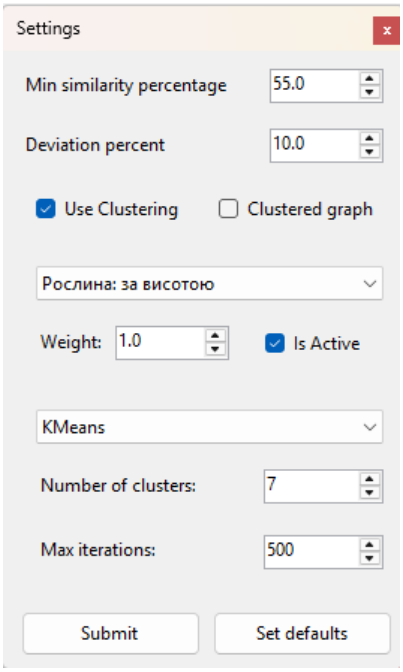


Рисунок 2.11 – Вікно налаштувань з увімкненою опцією кластеризації.

Для кожного методу кластеризації доступні параметри, які можуть бути змінені відповідно до потреб користувача. Для алгоритму K-середніх користувач може визначити кількість кластерів, які необхідно знайти, а також максимальну кількість ітерацій, які алгоритм може виконати для досягнення конвергенції. Алгоритм DBSCAN дозволяє налаштувати два ключові параметри: eps, що визначає максимальну відстань між двома точками, щоб вони потрапили в один кластер, та minPts, що вказує мінімальну кількість точок для утворення кластера. У випадку ієрархічної агломеративної

кластеризації (HAC) користувач може налаштувати поріг (Threshold), який визначає максимальну відстань, на якій зливатимуться кластери під час ієрархічного об'єднання. Ці параметри доступні через інтерфейс налаштувань, і користувач може змінювати їх у реальному часі для адаптації до конкретних даних і вимог. Всі ці алгоритми працюють із нормалізованими даними, що забезпечує точність і коректність кластеризації, незалежно від початкових масштабів і одиниць виміру різних параметрів.

Перед тим, як застосовувати ці алгоритми, всі дані піддаються попередній обробці для забезпечення коректної роботи кластеризації. Для числових даних використовується нормалізація методом MinMax, що дозволяє привести всі значення до діапазону від 0 до 1, що особливо важливо для коректного порівняння числових величин між собою. Для строкових даних застосовується One-Hot кодування, де кожен параметр перетворюється на набір бінарних значень, що також дозволяє зберегти всю інформацію про категорії і використовувати її для кластеризації.

Для вимірювання відстаней між об'єктами в застосунку реалізовано кілька метрик відстаней, що дозволяють точніше оцінити схожість між об'єктами. Усі метрики повертають значення відстані в діапазоні від 0 до 1, де 0 означає повну схожість, а 1 — максимальну різницю між об'єктами. Це дає змогу стандартизувати вимірювання та коректно порівнювати різні об'єкти за допомогою різних методів кластеризації. Евклідова відстань вимірює відстань між двома точками в просторі і є однією з найпоширеніших метрик для числових даних. Мантехенівська відстань вимірює суму абсолютних різниць між координатами точок, і є альтернативою євклідовій відстані, яка краще працює з дискретними даними. Косинусна відстань оцінює схожість між двома векторами на основі кута між ними, що особливо корисно для даних з великою кількістю вимірів, які не є чітко числовими. Хеммінгова відстань застосовується для строкових даних і визначає кількість різних елементів у двох рядках одного й того ж розміру. Усі ці метрики розроблені для забезпечення порівняння об'єктів у різних типах даних, як числових, так і

строкових, і використовуються для обчислення схожості між об'єктами, що дозволяє алгоритмам кластеризації коректно групувати їх.

2.6 Візуалізація результатів кластеризації

У застосунку реалізовані розширені можливості для побудови візуальних моделей, які демонструють результати кластеризації даних. Ці можливості включають візуалізацію у вигляд графів з кластерними мітками, точкових діаграм та кластерних блоків. Кожен з цих підходів відповідає різним потребам і завданням аналізу, дозволяючи більш точно і наочно інтерпретувати дані.

Граф з кластерними мітками є інструментом для відображення об'єктів у вигляді вершин, що згруповані за кластерами (рис. 2.12).

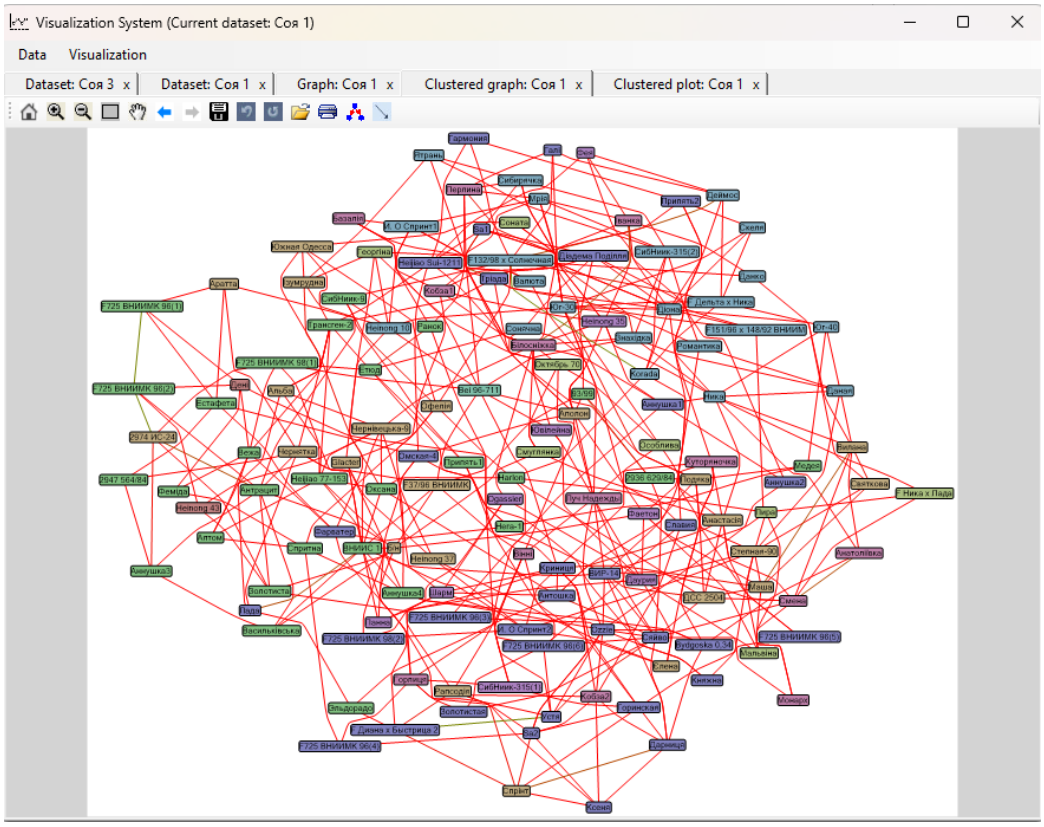


Рисунок 2.12 – Граф з кластерними мітками

Для наочності вершини, які належать до одного кластера, мають однаковий колір. Цей тип візуалізації забезпечує чітке уявлення про структуру даних і їхню кластерну організацію. Вершини графа з'єднуються між собою мінімум трьома зв'язками з іншими об'єктами, які демонструють найбільший відсоток схожості. Якщо відсоток схожості с кожним іншим об'єктом не перевищує нуль, то і зв'язків вершина не має. Цей підхід дозволяє не лише ідентифікувати об'єкти, які тісно пов’язані між собою, але й зосередити увагу на ключових елементах кластерів. Крім того, користувач може налаштовувати параметри побудови графа, зокрема мінімальний відсоток схожості, допустиму похибку, а також вагу та статус активності параметрів, що дозволяє адаптувати візуалізацію до специфіки аналізованих даних.

Точкова діаграма представляє ще один підхід до візуалізації, який дозволяє зменшити складність багатовимірних даних і представити їх у двовимірному просторі (рис. 2.13). Для побудови цієї діаграми використовується бібліотека ScottPlot [21].

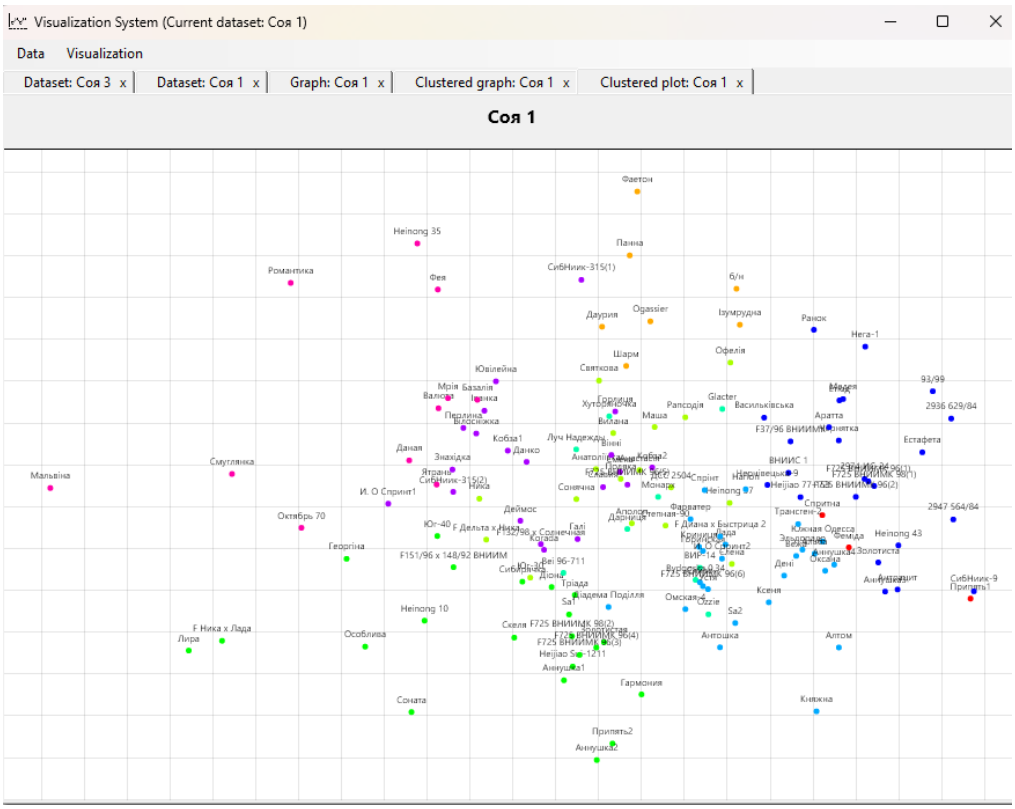


Рисунок 2.13 – Точкова діаграма

Кожна точка на графіку відповідає окремому об'єкту, а її колір вказує на належність до певного кластера. Для створення такого графіка використовується метод зниження розмірності PCA, реалізований у бібліотеці ML.NET [19]. Метод дозволяє перетворити багатовимірні дані в двовимірний простір. Це дає змогу візуально оцінити взаємне розташування кластерів, щільність об'єктів у них, а також виявити можливі відхилення або незвичайні об'єкти. Завдяки цьому підходу навіть складні структури даних стають доступними для швидкого аналізу та прийняття рішень.

Додатково застосунок підтримує побудову кластерних блоків, які представляють групи об'єктів, що належать до одного кластера (рис. 2.14).

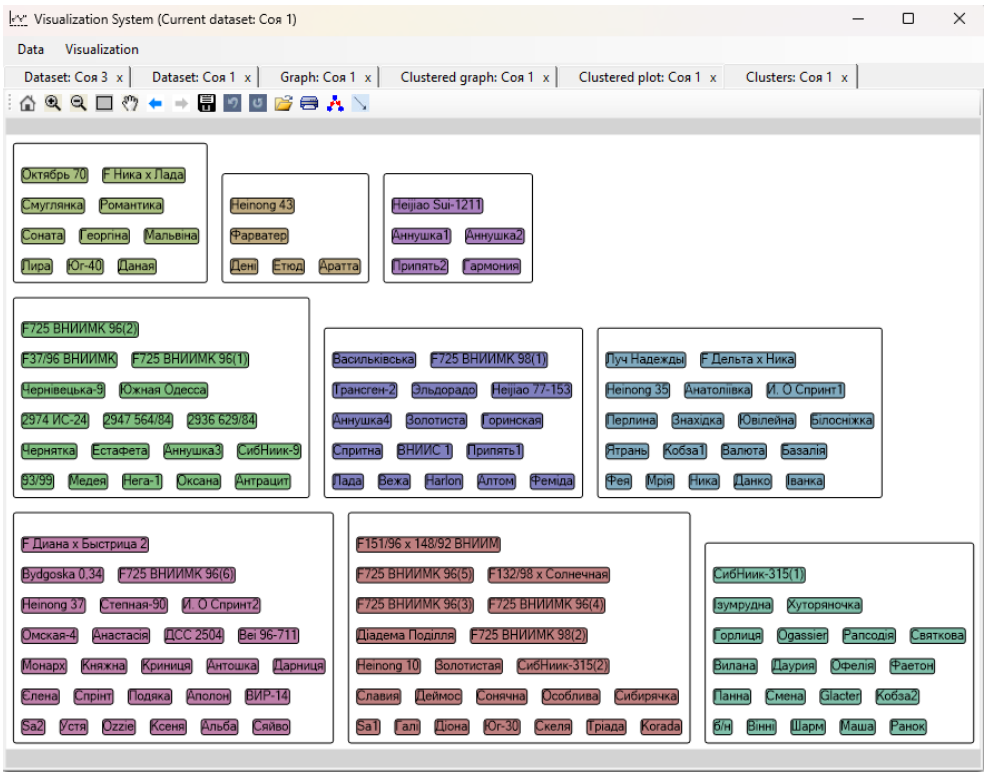


Рисунок 2.14 – Візуалізація кластерних блоків

Цей метод дозволяє візуально згрупувати об'єкти, щоб показати їхню кластерну приналежність. Позиції блоків у цьому випадку не мають зв'язку з реальним розташуванням об'єктів у багатовимірному просторі, проте вони дозволяють отримати чітке уявлення про те, які об'єкти належать до певного кластера, та оцінити розподіл об'єктів між кластерами.

Усі ці підходи до візуалізації спрямовані на створення інтуїтивно зрозумілого інструменту для аналізу результатів кластеризації. Вони дозволяють користувачам не лише глибше зрозуміти структуру даних, але й з легкістю знаходити ключові особливості кластерів, досліджувати їхні взаємозв'язки та оцінювати якість кластеризації. Це робить процес аналізу даних доступним навіть для користувачів без глибоких знань у галузі машинного навчання чи статистики.

2.7 Тестування застосунку на основі даних Fragile States Index

Для перевірки роботи розробленого застосунку було обрано набір даних із таблиці Fragile States Index [22]. Цей набір використовується для оцінки стабільності країн за широким спектром соціальних, економічних, політичних та військових показників. Його структурованість та багатопараметричний характер роблять його ідеальним для тестування функціоналу застосунку, включаючи імпорт даних, їх обробку, кластеризацію та візуалізацію.

Таблиця Fragile States Index наступні параметри:

– S1: Demographic Pressures (Демографічний тиск) — оцінює проблеми, пов'язані з перенаселенням, нестачею ресурсів та природними катастрофами.

– S2: Refugees and IDPs (Біженці та внутрішньо переміщені особи) — враховує масштаби міграції через конфлікти чи інші причини.

– C3: Group Grievance (Групові скарги) — аналізує соціальну напруженість, етнічні конфлікти та дискримінацію.

– E3: Human Flight and Brain Drain (Відтік людського капіталу) — оцінює рівень міграції кваліфікованих працівників та фахівців.

– E2: Economic Inequality (Економічна нерівність) — вимірює розрив між багатими та бідними, доступ до основних послуг.

– E1: Economy (Економіка) — характеризує загальний стан економіки, включаючи інфляцію та безробіття.

– P1: State Legitimacy (Легітимність держави) — оцінює рівень корупції, прозорість уряду та довіру населення.

– P2: Public Services (Державні послуги) — аналізує доступність і якість освіти, медицини та інфраструктури.

– P3: Human Rights (Права людини) — враховує порушення прав людини, репресії та свободи громадян.

– C1: Security Apparatus (Силові структури) — аналізує військову активність, рівень насильства та внутрішні конфлікти.

– C2: Factionalized Elites (Роздроблені еліти) — вимірює внутрішньополітичні конфлікти та боротьбу між політичними угрупованнями.

– X1: External Intervention (Зовнішнє втручання) — враховує вплив іноземних держав через військове втручання, економічний тиск чи дипломатію.

Побудуємо граф на основі алгоритму порівняння (рис. 2.15).

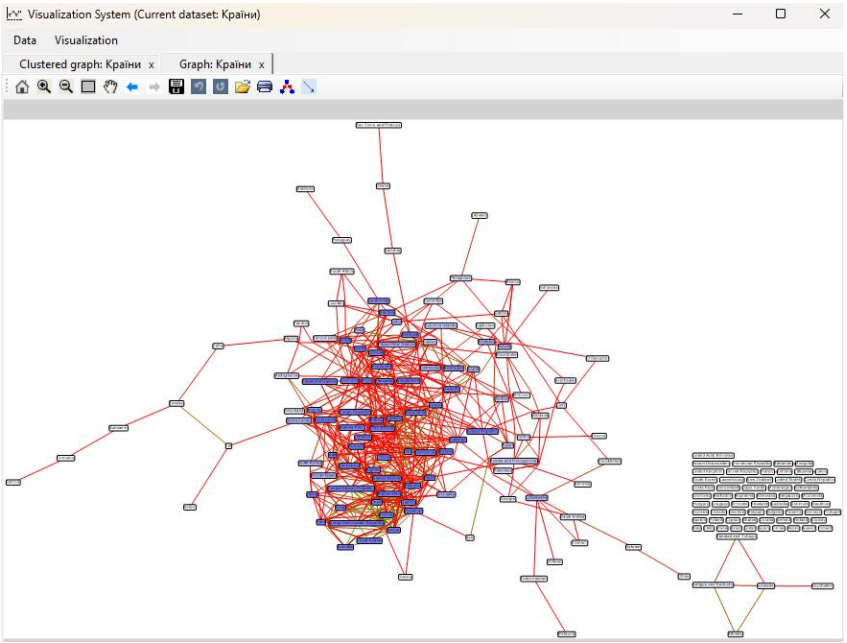


Рисунок 2.15 – Граф на основі порівняння об'єктів для датасету “FSI”

Граф містить велику кількість вершин, які пов’язані між собою численними зв’язками, та значну кількість вершин, що взагалі не мають зв’язків з іншими. Зміна параметрів для порівняння не призвела до суттєвих змін. У будь-якому випадку, при малих значеннях мінімального необхідного відсотка схожості виникає надто щільна область, де вершини сильно переплетені між собою, що ускладнює оцінку зв’язків. Натомість, при великих значеннях цього параметра вершини майже не утворюють зв’язків одна з одною. Через це стає очевидним, що аналіз результатів кластеризації може забезпечити більш інформативну й зрозумілу структуру для подальшого дослідження. Тому перейдемо до їх розгляду.

Спочатку виконаємо візуалізацію у вигляді кластерних блоків. Проведемо перевірку роботи алгоритму К-середніх (рис. 2.16) із такими параметрами: кількість кластерів — 11, максимальна кількість ітерацій — 500.

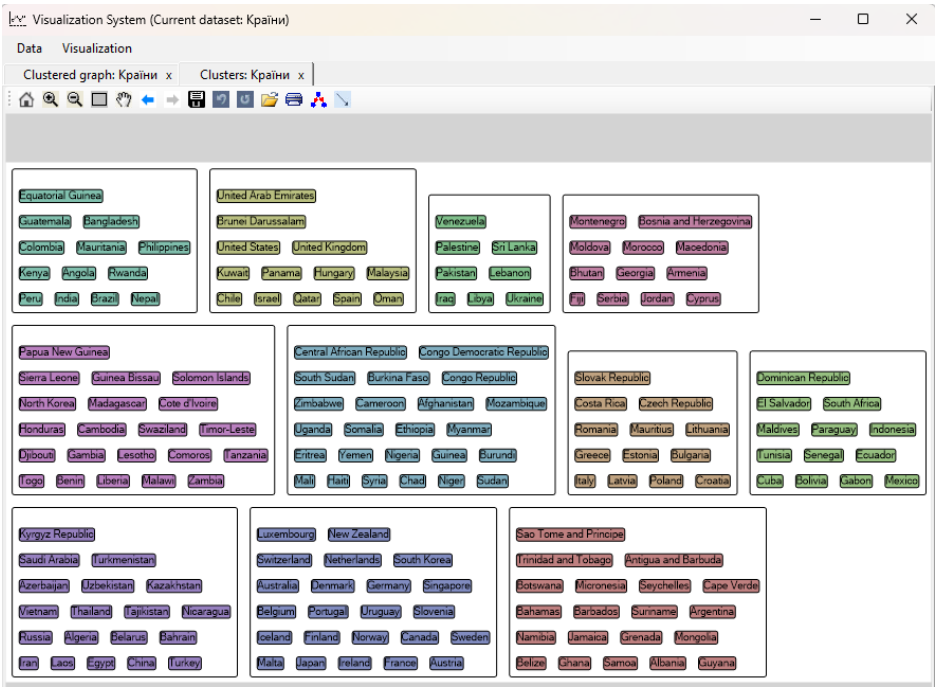


Рисунок 2.16 – Візуалізація кластерних блоків за алгоритмом К-середніх для датасету «FSI»

Цей метод візуалізації дозволяє оцінити коректність роботи алгоритму та виявити можливі аномалії, такі як надто великі кластери або повністю

порожні. Результати показують, що кластери мають збалансовані розміри. Можна помітити певну географічну логіку: країни Європи групуються разом, країни Африки формують окремий кластер, а азійські країни здебільшого також належать до спільних кластерів. Загалом алгоритм K-середніх продемонстрував адекватну роботу, створивши логічну структуру без очевидних помилок.

Далі проводимо перевірку алгоритму DBSCAN (рис. 2.17) із параметрами: eps = 0.115 та minPts = 2.

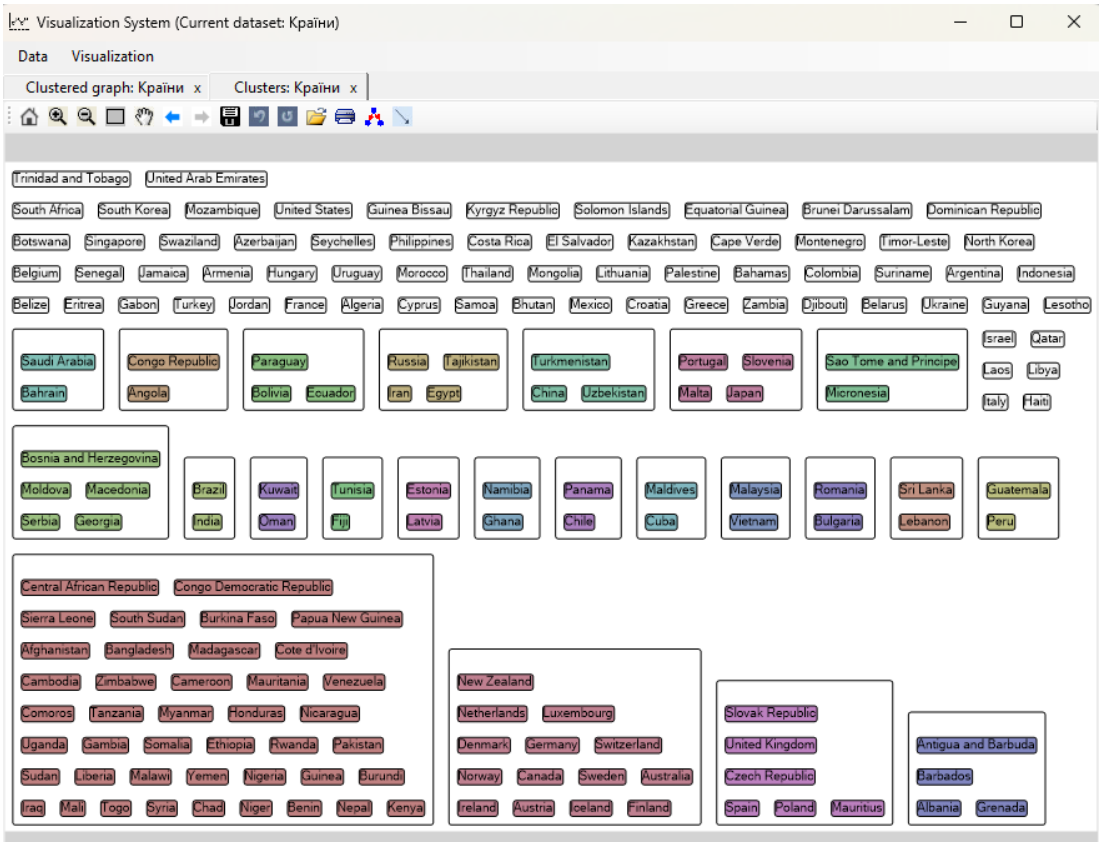


Рисунок 2.17 – Візуалізація кластерних блоків за алгоритмом DBSCAN для датасету «FSI»

Цей метод створив більшу кількість дрібних кластерів. Значна частина країн залишилася поодиноко або у дуже малих групах. DBSCAN показав меншу "примусовість" у формуванні кластерів порівняно з K-середніх, що дозволило краще виявити природні групи країн, дійсно схожих за своїми параметрами. Алгоритм виділив кілька великих стабільних кластерів, зокрема

серед африканських країн, що підтверджує його здатність до тоншого аналізу даних.

Переходимо до перевірки алгоритму агломеративної ієрархічної кластеризації (HAC (рис. 2.18)) із параметром Threshold = 0.22.

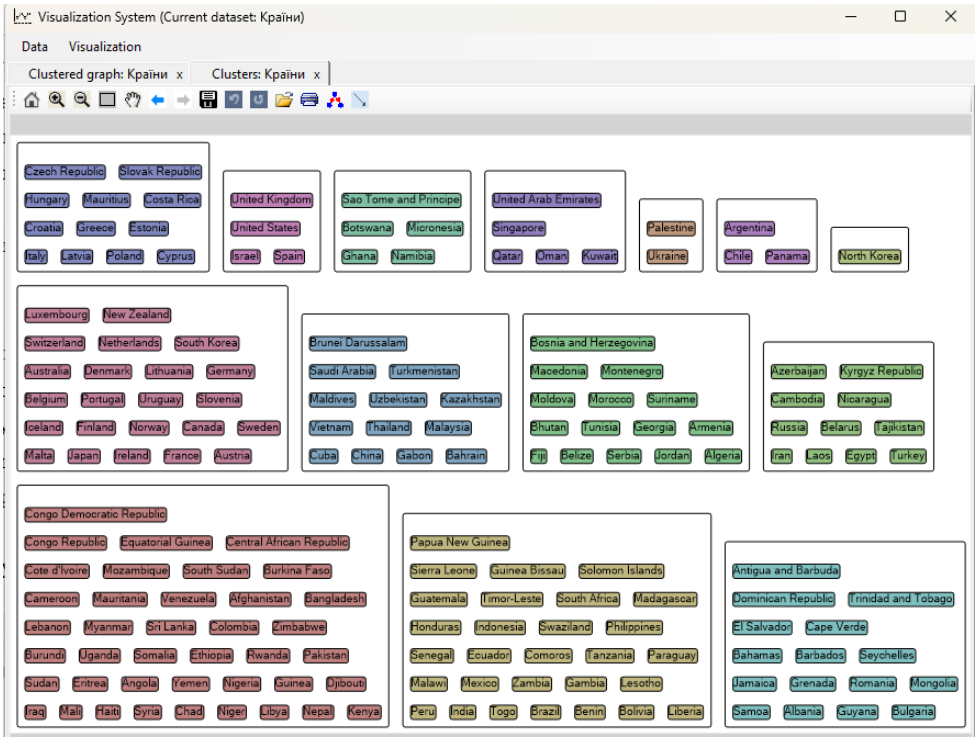


Рисунок 2.18 – Візуалізація кластерних блоків за алгоритмом HAC для датасету «FSI»

Цей метод створив найбільш збалансовану кластерну структуру. Чітко проглядається географічна логіка: окремий кластер східноєвропейських країн, окремий кластер західноєвропейських і скандинавських країн, групування країн Близького Сходу, а африканські країни поділені на кілька логічних підгруп. Агломеративна кластеризація також зберігає ієрархічну структуру, що дозволяє побачити як більші об’єднання країн, так і дрібніші підгрупи.

Однак такий спосіб візуалізації не дозволяє детально оцінити взаємне розташування кластерів та об’єктів відносно один одного. Для отримання більш детальної картини перейдемо до наступного методу візуалізації.

Загальні висновки: HAC показав найбільш інтуїтивно зрозумілі результати, дозволяючи глибше аналізувати групування країн відповідно до вказаних параметрів. DBSCAN виявився найчутливішим до відмінностей між країнами, створивши велику кількість малих кластерів. Натомість K-середніх забезпечив більш збалансовані за розміром кластери, але менш природні. Усі три методи підтвердили існування стійких груп країн, які постійно опиняються разом, наприклад, скандинавські країни або країни Західної Європи. Водночас деякі країни, такі як Північна Корея, часто формують окремі кластери, що вказує на їх унікальні характеристики.

Проведемо побудову графів із кластерними мітками, використовуючи алгоритми K-середніх (рис. 2.19), DBSCAN (рис. 2.20) та HAC (рис. 2.21) із тими самими параметрами.

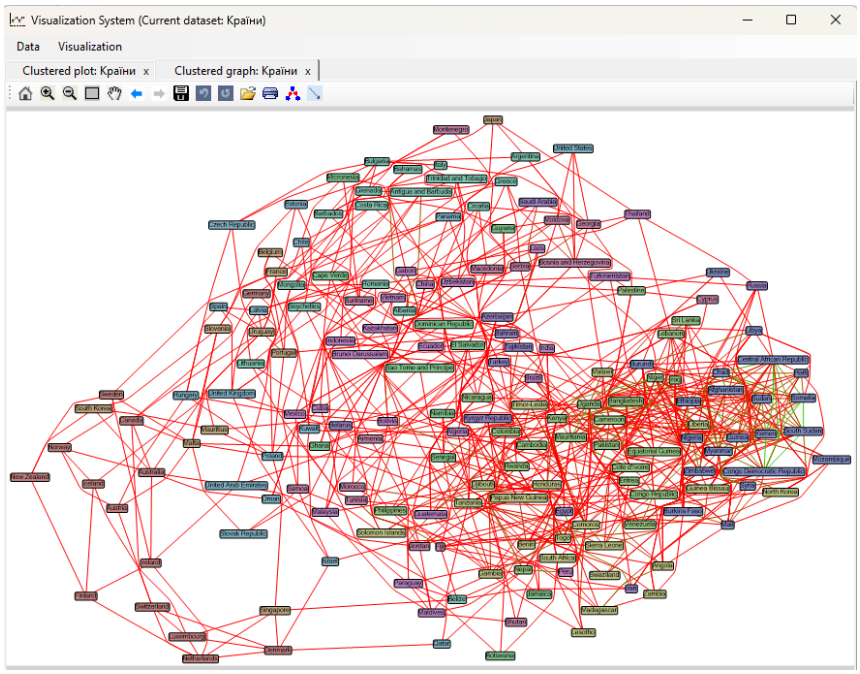


Рисунок 2.19 – Граф із кластерними мітками побудований за алгоритмом K-середніх для датасету «FSI»

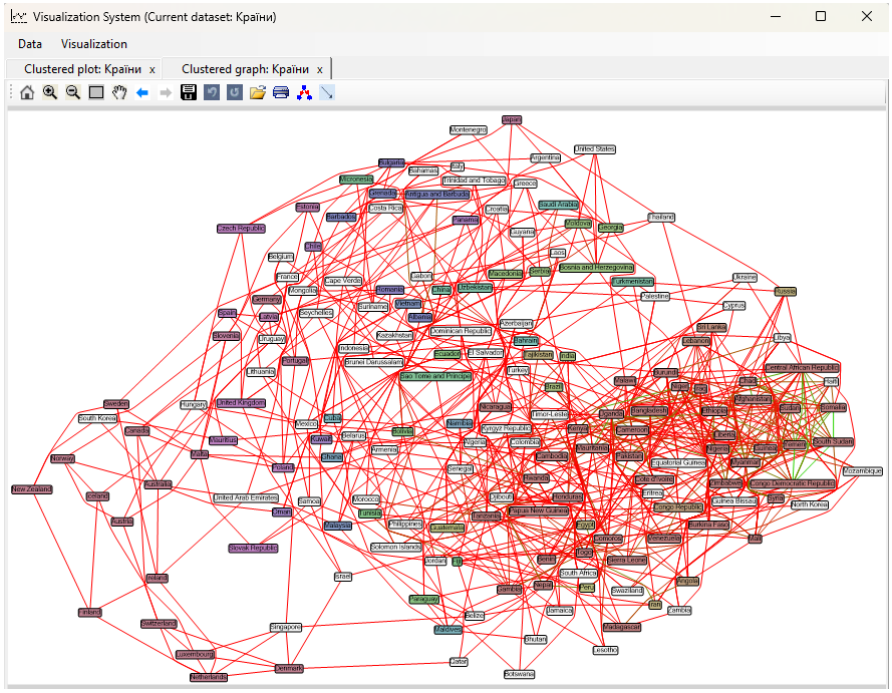


Рисунок 2.20 – Граф із кластерними мітками побудований за алгоритмом DBSCAN для датасету «FSI»

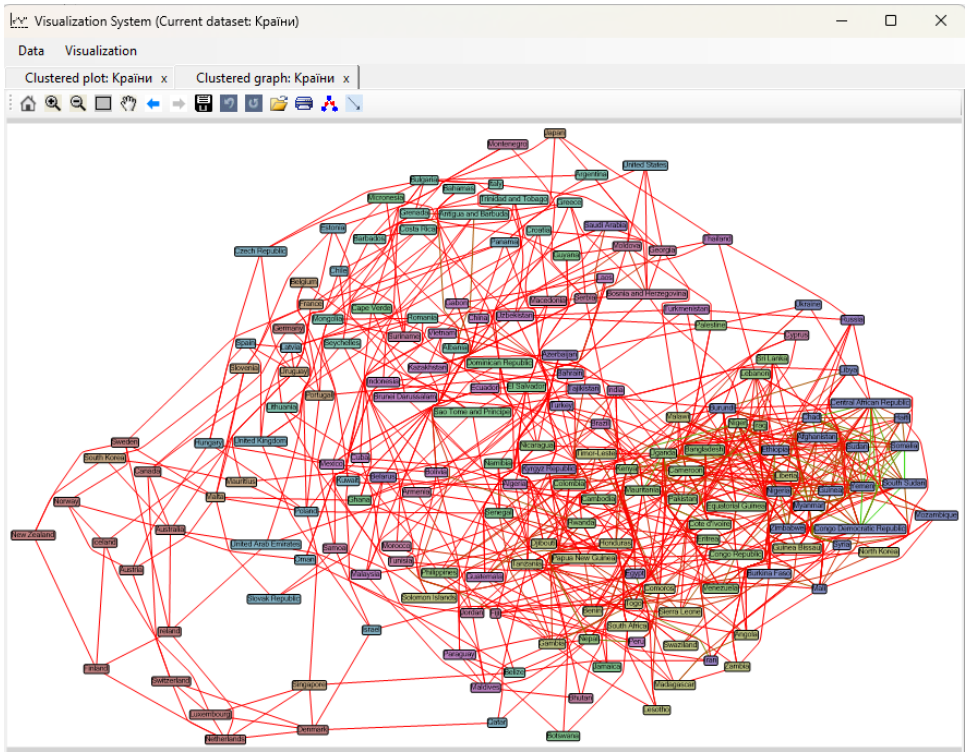


Рисунок 2.21 – Граф із кластерними мітками побудований за алгоритмом HAC для датасету «FSI»

Усі графи демонструють зв’язки між країнами на основі їхньої схожості, причому кожна вершина має щонайменше три зв’язки, якщо схожість з іншими об’єктами є достатньою. Кольорове кодування відображає належність країн до кластерів, що дозволяє візуально розрізняти різні групи.

Варто зазначити, що алгоритм DBSCAN створює меншу кількість вершин із кластерними мітками в графі порівняно з іншими алгоритмами. Мітки кластерів є досить схожими між усіма графами, оскільки схожі об'єкти потрапляють до спільних кластерів за всіма методами кластеризації. Однак ребра між вершинами залишаються однаковими, оскільки для побудови графів використовується єдиний алгоритм порівняння, спільний для всіх методів кластеризації.

Проведемо побудову точкових діаграм, використовуючи алгоритми К-середніх (рис. 2.22), DBSCAN (рис. 2.23) та НАС (рис. 2.24) із тими самими параметрами.

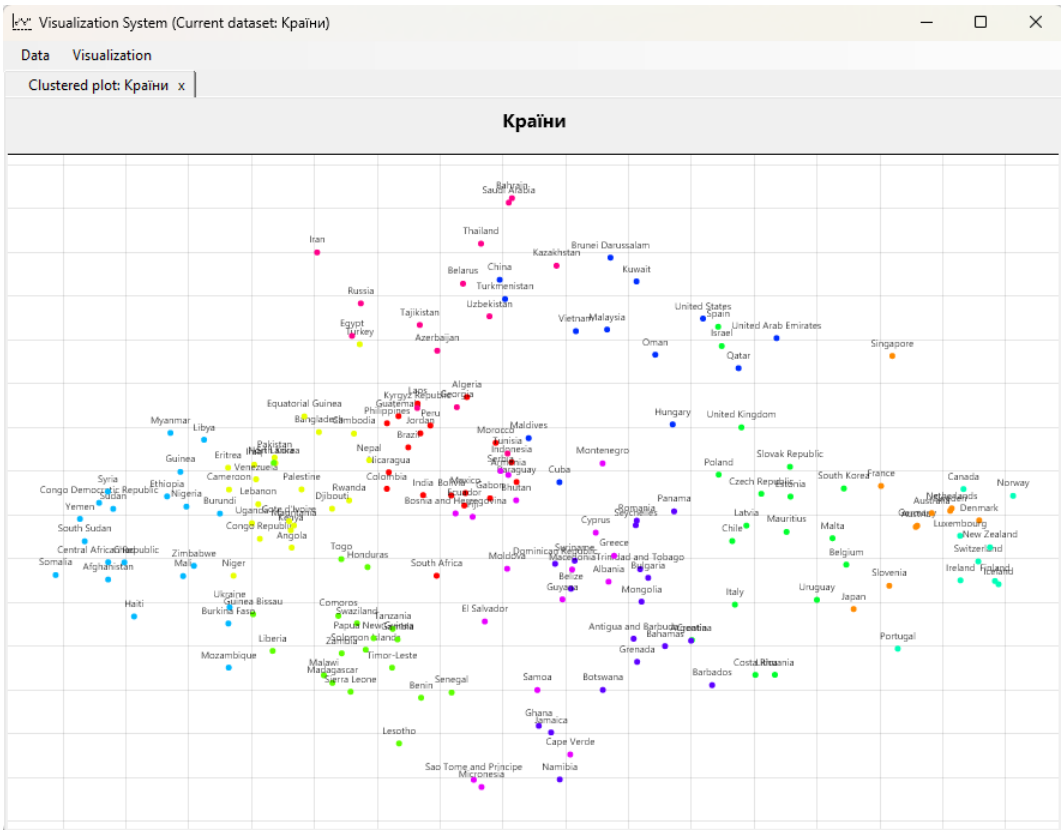


Рисунок 2.22 – Точкова діаграма побудована за алгоритмом К-середніх для датасету «FSI»

Алгоритм К-середніх чітко розділяє країни на кластери різних кольорів, що дозволяє легко візуально ідентифікувати групи країн. Кластери в цілому мають відносно рівномірний розмір, що свідчить про збалансованість кластерів. Також можна помітити певну географічну логіку групування країн. Однак через випадковий вибір початкових центроїдів результати можуть варіюватися при кожному запуску алгоритму, що може призвести до незначних відмінностей у кластеризації.

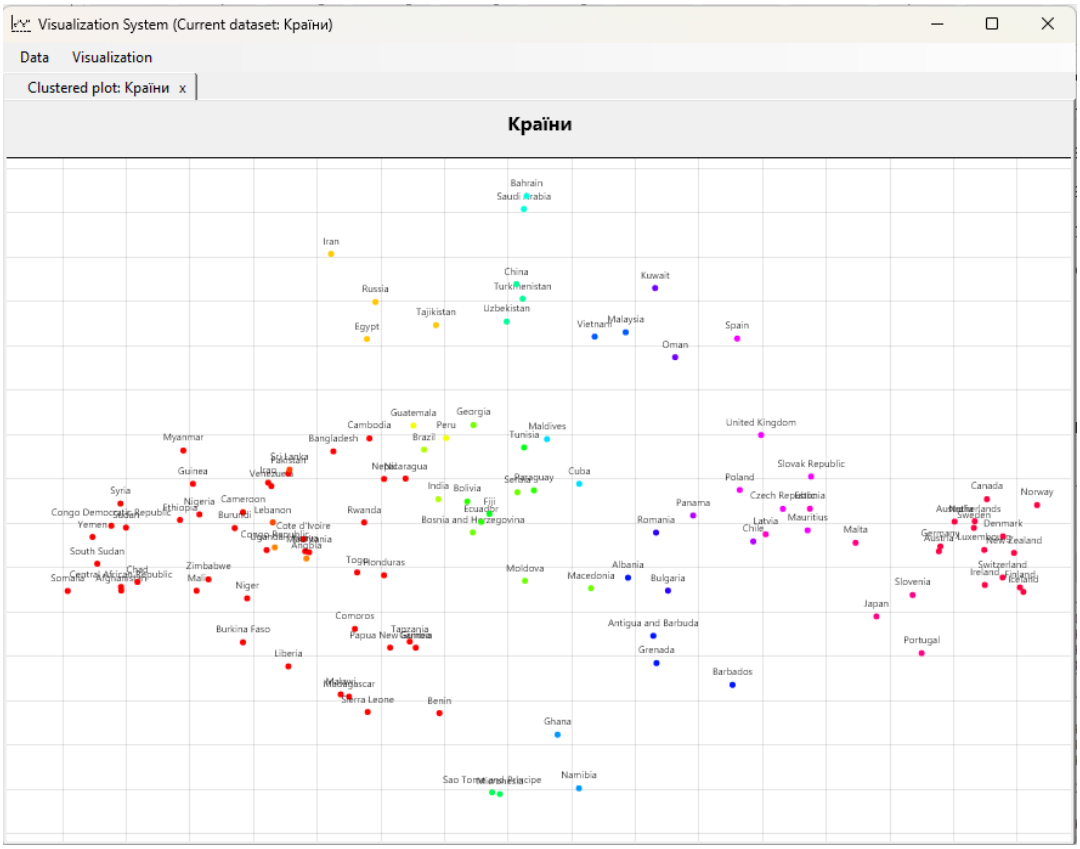


Рисунок 2.23 – Точкова діаграма побудована за алгоритмом DBSCAN для датасету «FSI»

DBSCAN працює дещо інакше, оскільки він виключає шумові об'єкти, внаслідок чого відображає менше точок на діаграмі. Кластери, сформовані цим алгоритмом, мають різну щільність та розмір, що дозволяє краще відобразити природні групи країн, які схожі за параметрами. Водночас результати DBSCAN є більш стабільними порівняно з К-середніх, оскільки цей алгоритм не залежить від випадкових початкових значень.

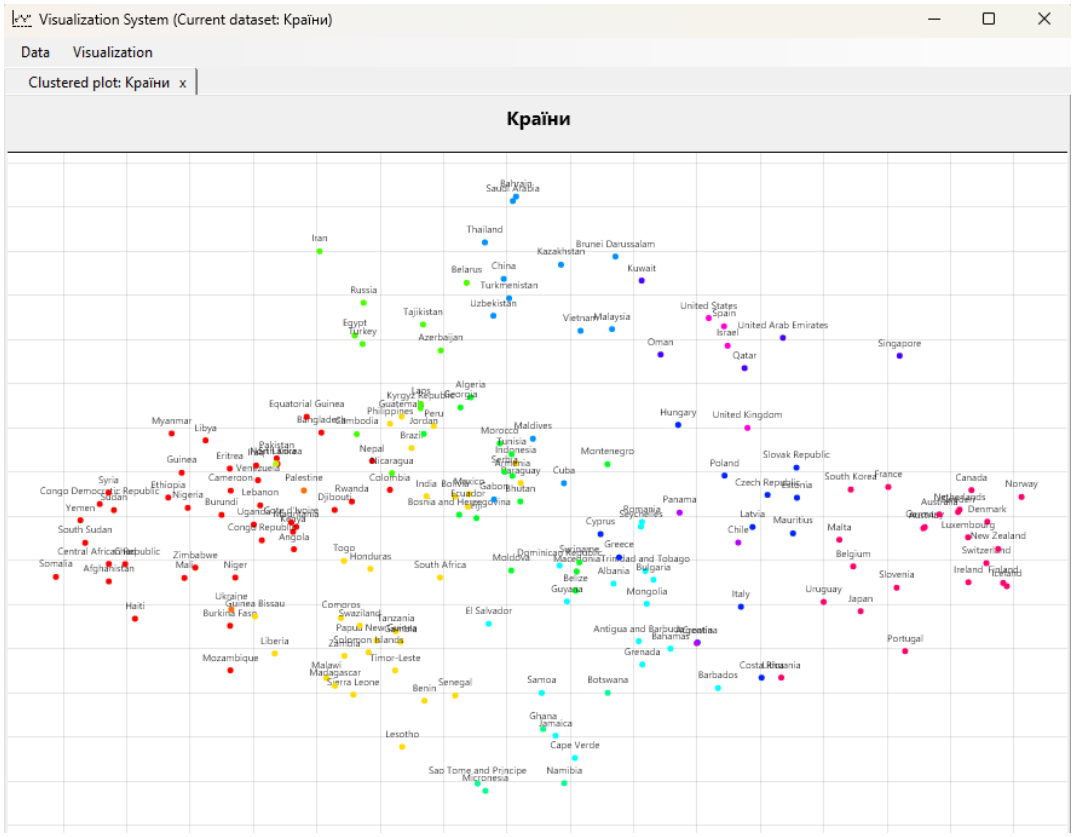


Рисунок 2.24 – Точкова діаграма побудована за алгоритмом HAC для датасету «FSI»

Алгоритм HAC демонструє більш ієрархічну структуру кластерів, що дає змогу чітко розмежувати різні групи країн, це проявляється у вигляді чітко визначених і відокремлених груп, що мають зрозумілу структуру. Кластери мають чіткі межі, що дозволяє легко зрозуміти, які країни належать до того чи іншого кластеру. Явно помітна географічна логіка групування, зокрема між східноєвропейськими та західноєвропейськими країнами. Хоча цей алгоритм працює повільніше за інші, він дає стабільні та передбачувані результати, що робить його ефективним для виявлення ієрархічних зв'язків між країнами.

Аналіз результатів кластеризації та візуалізації показує, що точкові діаграми є дієвим інструментом для швидкого огляду загальної структури даних. Вони дозволяють побачити відмінності між кластерами та їх розподіл у двовимірному просторі. Проте ці діаграми не завжди здатні детально відобразити складні зв'язки між об'єктами, що є важливим для більш

глибокого аналізу. В цьому випадку графи з мітками, навпаки, забезпечують чітке уявлення про взаємозв'язки між об'єктами, дозволяючи краще зрозуміти, як вони взаємодіють на основі схожості. Поєднання обох типів візуалізації дає найповніше розуміння структури даних.

Що стосується практичного застосування алгоритмів кластеризації, то для швидкого та ефективного аналізу, коли потрібно швидко виявити основні групи, алгоритм К-середніх є оптимальним вибором. Його здатність розподіляти дані рівномірно та забезпечувати чітке розділення кластерів робить його ідеальним для таких випадків. Однак для виявлення аномалій, таких як ізольовані об'єкти або дрібні кластерні структури, алгоритм DBSCAN показує себе як більш чутливий та ефективний інструмент, оскільки він дозволяє виявляти менш очевидні зв'язки та не вимагає попереднього визначення кількості кластерів.

Агломеративна ієрархічна кластеризація (HAC) забезпечує чітке групування об'єктів, що дозволяє виявити не лише великі групи, але й більш дрібні підгрупи, демонструючи їхні взаємозв'язки. Також HAC дає більш глибоке уявлення про структуру даних, порівняно з іншими методами кластеризації. Кожен з цих методів має свої сильні та слабкі сторони, тому важливо комбінувати їх для досягнення більш повного та точного аналізу.

Використання різних методів візуалізації та кластеризації не тільки покращує точність аналізу, але й дозволяє адаптувати підхід залежно від специфічних потреб дослідження. Зокрема, комбінування точкових діаграм, графів та кластерних блоків дає змогу отримати більш комплексне уявлення про структуру даних, виявити аномалії, а також зрозуміти зв’язки та залежності між об’єктами на різних рівнях кластеризації.

2.8 Тестування застосунку на основі даних про сорти сої

Перша таблиця містить числові дані про сорти сої, зокрема вимірювання таких характеристик, як висота рослин, товщина стебла, кількість гілок та інші кількісні параметри. Дані добре підготовлені для кластеризації, оскільки майже немає пропусків, а значення параметрів знаходяться в допустимих межах.

Побудуємо точкову діаграму, використовуючи алгоритм кластеризації методом К-середніх (рис. 2.25) з оптимальними параметрами: кількість кластерів — 9, максимальна кількість ітерацій — 500.

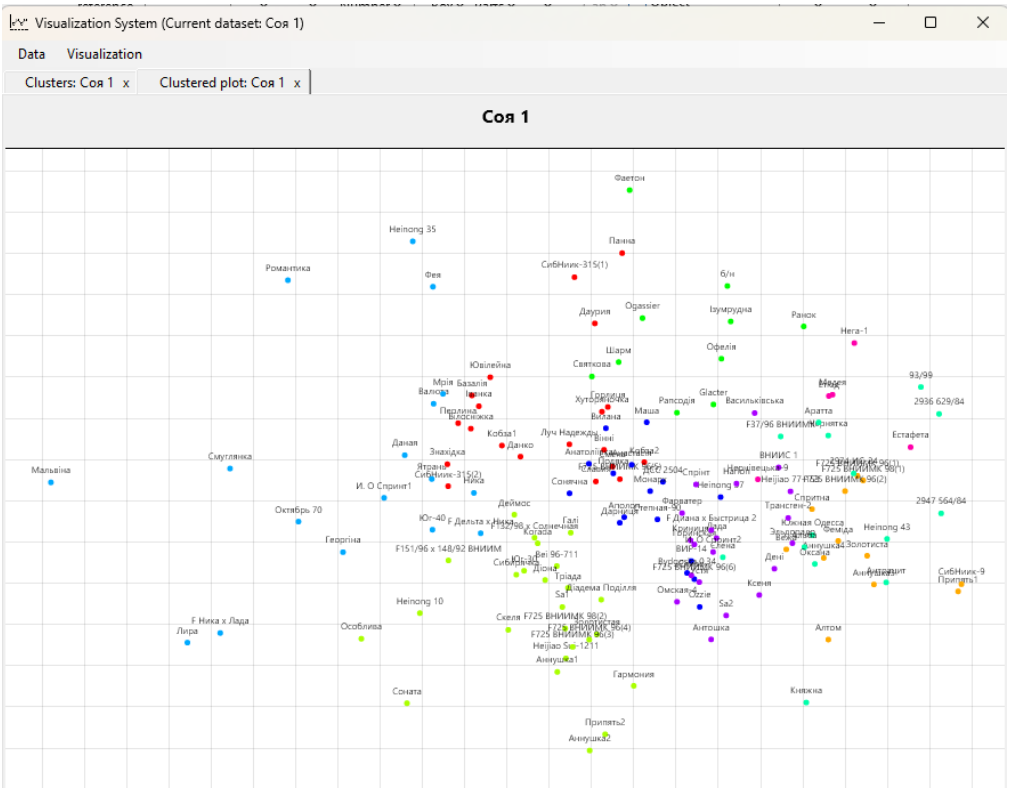


Рисунок 2.25 – Точкова діаграма побудована за алгоритмом К-середніх для датасету «Соя 1»

Можна зробити висновок, що кольори візуалізації рівномірно розподілені, що свідчить про добре розподілення кластерів. Точкова діаграма

показує гарну когезію кластерів з чіткими групами. Хоча спостерігається невелике перекриття між кластерами, загальний поділ виглядає змістовно і інтерпретовано.

Побудуємо точкову діаграму, використовуючи алгоритм кластеризації методом DBSCAN (рис. 2.26) з оптимальними параметрами: eps = 0.083 та minPts = 2

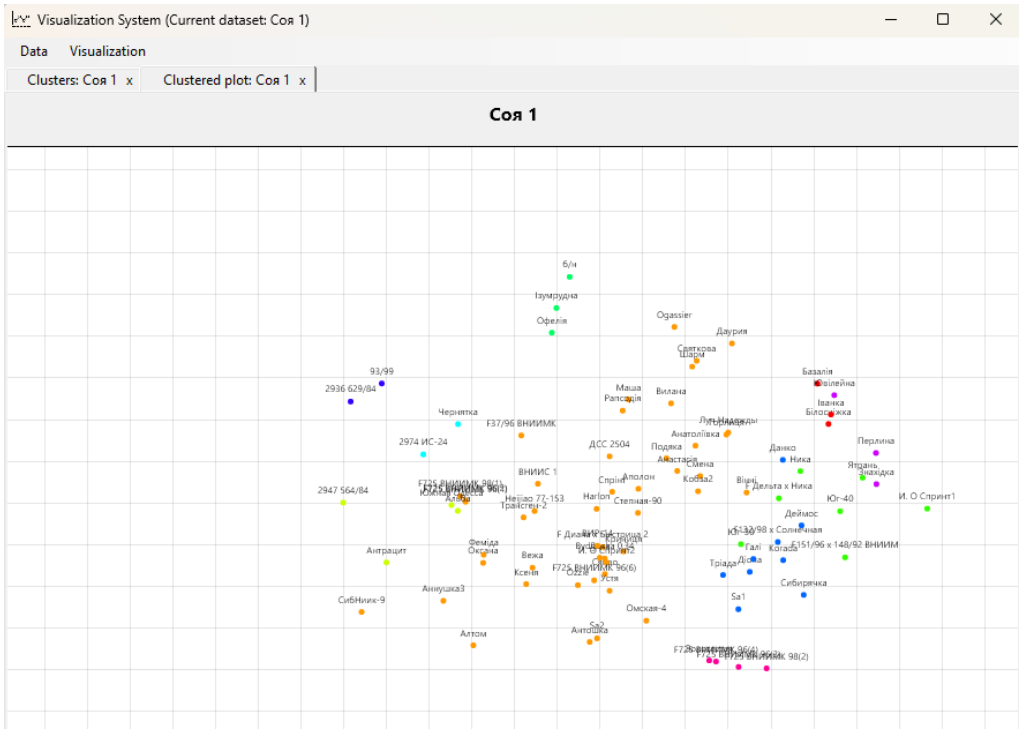


Рисунок 2.26 – Точкова діаграма побудована за алгоритмом DBSCAN для датасету «Соя 1»

DBSCAN виявляє більш фрагментовані групування в порівнянні з K-середніми. Деякі сорти ізольовані в окремі кластери, а також є багато шумових точок. Розподіл точок більш розсіяний з чіткими щільними регіонами. Цей метод ефективно виявляє аномалії і добре обробляє не сферичні форми кластерів.

Побудуємо точкову діаграму, використовуючи алгоритм кластеризації методом HAC (рис. 2.27) з оптимальними параметром Threshold – 0.19.

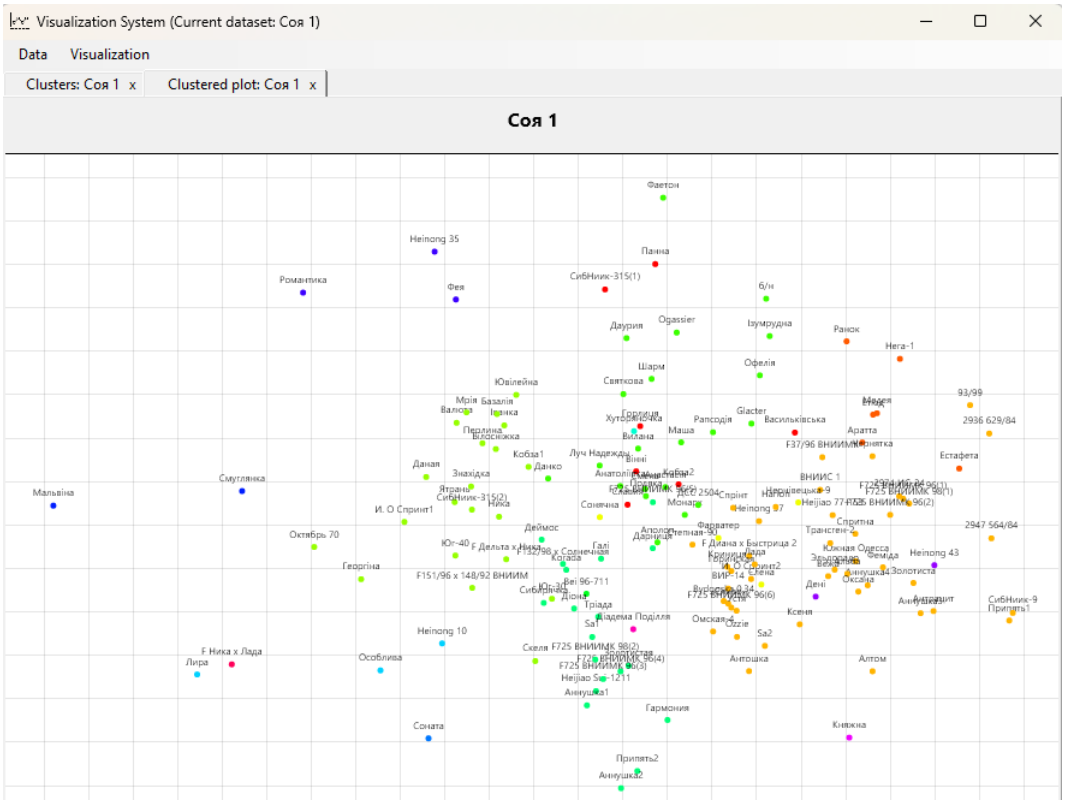


Рисунок 2.27 – Точкова діаграма побудована за алгоритмом HAC для датасету «Соя 1»

HAC демонструє ієрархічну структуру з чіткими основними групами. Кластери більш збалансовані за розмірами порівняно з DBSCAN, а точкова діаграма показує гарну роздільність між основними групами. Цей метод ефективно захоплює природні ієрархічні взаємозв'язки між сортами.

Загалом метод К-середніх надає найбільш збалансовану та інтерпретовану кластеризацію з чітким розподілом сортів. DBSCAN ефективно виявляє аномалії, але може бути занадто чутливим до варіацій щільності, що призводить до надмірного фрагментування. HAC пропонує хороший середній варіант, де є інтерпретовані ієрархічні зв'язки та більш збалансовані групи, ніж у DBSCAN.

Деякі сорти постійно кластеризуються разом за різними алгоритмами, що свідчить про їхню схожість. Також у наборі є чітко виражені аномалії та окремі групи сортів, що мають унікальні характеристики. Точкові діаграми ефективно показують просторові зв'язки між сортами та допомагають краще розуміти їх взаємозв'язки.

Далі перейдемо до третьої таблиці, вона містить категоріальні дані про сорти сої, такі як глікотиль, антоціанове забарвлення, тип росту рослини, форма куща та інші характеристики. Дані добре структуровані з послідовним кодуванням для кожного сорту, але є деякі пропуски, причому кількість пропущених значень варіюється серед сортів.

Побудуємо точкову діаграму, використовуючи алгоритм кластеризації методом К-середніх (рис. 2.28) з оптимальними параметрами: кількість кластерів — 6, максимальна кількість ітерацій — 500.

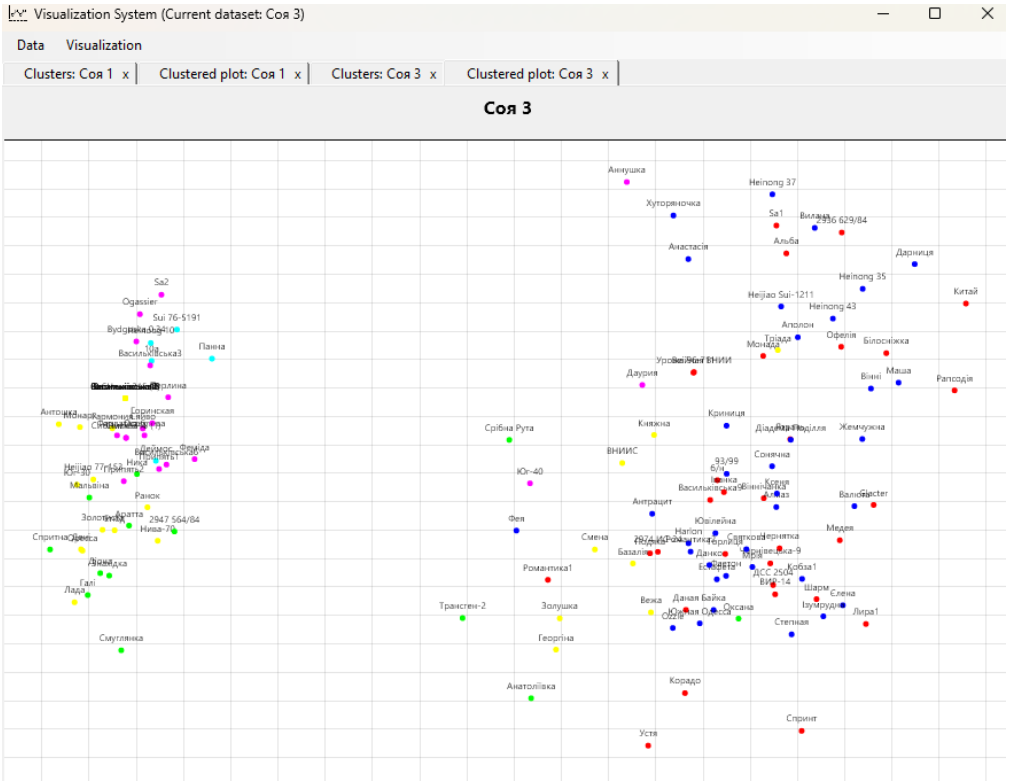


Рисунок 2.28 – Точкова діаграма побудована за алгоритмом К-середніх для датасету «Соя 3»

Результати кластеризації за допомогою методу К-середніх показують, що групування сортів не є надто чітким, хоча окремі групи можна виділити. Ці групи можуть пересікатися та входити одна в одну. Точкова діаграма демонструє допустиме розділення, при цьому основні групи можна чітко ідентифікувати.

Побудуємо точкову діаграму, використовуючи алгоритм кластеризації методом DBSCAN (рис. 2.29) з оптимальними параметрами: eps = 0.145 та minPts = 2

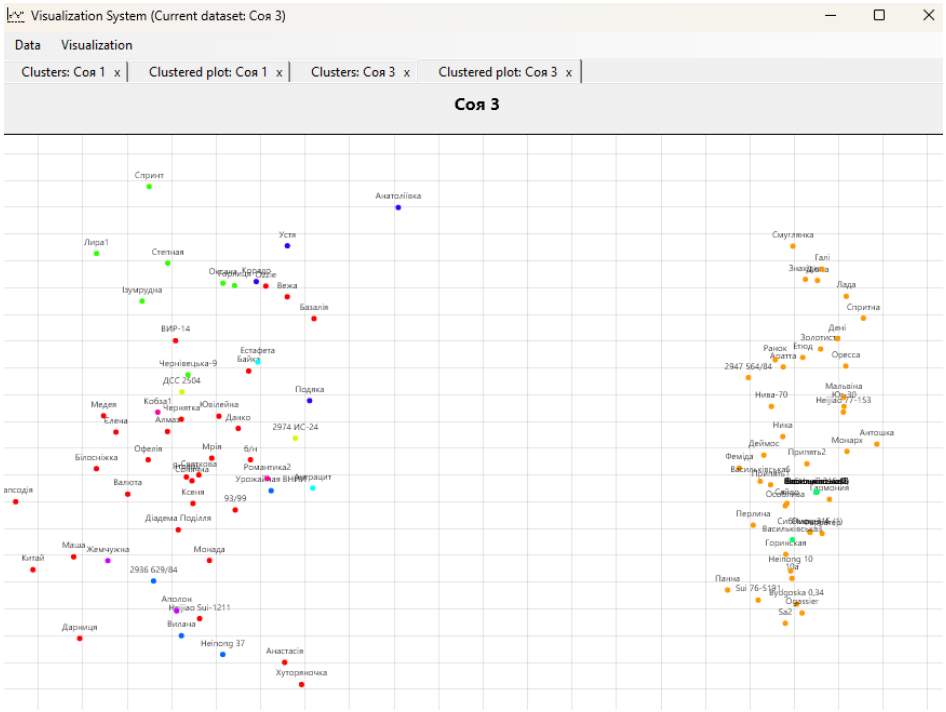


Рисунок 2.29 – Точкова діаграма побудована за алгоритмом DBSCAN для датасету «Соя 3»

Метод DBSCAN ефективно ідентифікує сорти з унікальними комбінаціями категоріальних ознак і виявляє значну кількість шумових точок. Групи більш чіткі та збалансовані в порівнянні з К-середніми. Цей метод краще виявляє аномалії у категоріальному просторі, оскільки здатен правильно обробляти нечислові дані.

Побудуємо точкову діаграму, використовуючи алгоритм кластеризації методом HAC (рис. 2.30) з оптимальними параметром Threshold – 0.26.

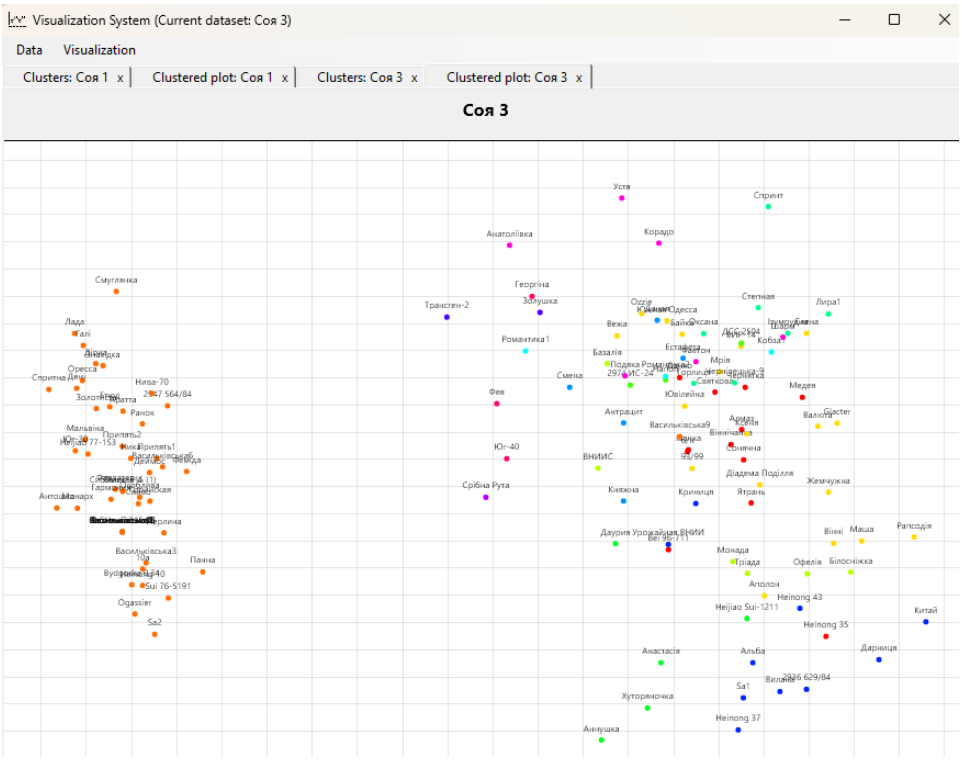


Рисунок 2.30 – Точкова діаграма побудована за алгоритмом HAC для датасету «Соя 3»

Агломеративна ієрархічна кластеризація (HAC) чітко виражає основні групи, хоча інколи може спостерігатися їхнє перекриття. Вона ефективно захоплює взаємозв'язки між схожими сортами, демонструючи більш збалансовану кластеризацію.

Загалом незважаючи на певні обмеження методу К-середніх при роботі з нечисловими даними, він забезпечує хороше загальне групування для категоріальних характеристик. DBSCAN, в свою чергу, добре справляється з ідентифікацією унікальних сортів та маленьких груп з чіткими комбінаціями ознак. Найбільш інтерпретовані результати для категоріальних даних надає метод HAC, оскільки він показує найбільш чіткі групи, враховуючи всі значення.

При аналізі категоріальних даних виникають проблеми з відстанями між значеннями. У числових даних відстань між значеннями є природною, тоді як у строкових даних складно визначити відстань між категоріями. Числові дані мають широкий діапазон значень і можуть бути коректно

агреговані для обчислення середнього, в той час як для строкових даних це неможливо, що ускладнює точне визначення центроїда кластера в методі К-середніх. Числові дані надають точну кількісну інформацію, тоді як строкові дані часто узагальнюють характеристики.

Кластеризація строкових даних є менш чіткою та стабільною, що призводить до більшого шуму в результатах та складніших для інтерпретації результатів. Однак строкові дані також можуть бути ефективно проаналізовані, оскільки точкові діаграми показують відносну подібність між сортами, а комбіновані візуалізації дозволяють виявити як належність до кластерів, так і взаємозв'язки між ними.

ВИСНОВКИ

Була реалізована система для аналізу багатопараметричних даних з використанням кластеризації. Основною метою розробки було створення застосунку, який дозволяє обробляти та візуалізувати взаємозв'язки між даними, зокрема з використанням алгоритмів кластеризації. У результаті роботи було створено ефективний інструмент для аналізу даних, який успішно справляється з вхідними даними, їх інтеграцією, кластеризацією та візуалізацією результатів. Також було необхідно створити гнучкий інтерфейс для роботи з різними наборами даних, що дасть можливість обробляти як числові, так і текстові дані.

Структура розробленого застосунку побудована на основі модульного підходу, де кожен етап роботи з даними був реалізований як окремий компонент. Ця структура забезпечує простоту розширення та адаптації застосунку для роботи з новими типами даних. Застосунок включає в себе чітке розмежування між етапами завантаження даних, їх обробки, кластеризації та візуалізації.

Система дозволяє легко інтегрувати файли у форматі MS Excel, що робить її зручною для роботи з реальними даними, які часто зберігаються в такому форматі. Структура бази даних була розроблена таким чином, щоб забезпечити ефективне збереження та доступ до даних. База даних використовує таблиці для збереження різних параметрів, що дає змогу швидко виконувати пошук і фільтрацію даних. Крім того, база даних була оптимізована для роботи з великими обсягами інформації.

Було реалізовано алгоритми кластеризації K-середніх, DBSCAN та агломеративний ієрархічний алгоритм (HAC). Кожен з цих алгоритмів був реалізований із можливістю налаштування параметрів. Алгоритм K-середніх

виявився ефективним для роботи з числовими даними, DBSCAN — для кластеризації даних з шумами, а HAC — для побудови ієрархічних кластерів.

Візуалізація результатів кластеризації була реалізована через графічне представлення кластерів на графах за допомогою бібліотеки Msagl, а також через точкові діаграми, використовуючи бібліотеку ScottPlot. Кожен кластер виділяється окремим кольором, що дозволяє користувачам наочно побачити структуру даних та взаємозв'язки між об'єктами.

Тестування застосунку на основі даних Fragile States Index показало, що система здатна обробляти реальні дані, що містять різноманітні параметри та можуть бути складними для традиційних методів аналізу. Тестування підтвердило ефективність алгоритмів кластеризації, які здатні виявляти закономірності навіть у складних наборах даних.

Тестування на основі даних про сорти сої дозволило перевірити працездатність системи в аграрному контексті. В результаті тестування застосунок успішно класифікував сорти сої за різними параметрами, що підтвердило його здатність працювати з сільськогосподарськими даними та використовувати кластеризацію для виявлення оптимальних груп для подальшого дослідження.

Таким чином, розроблений застосунок виявився потужним інструментом для обробки та аналізу багатопараметричних даних, зокрема через використання кластеризації, візуалізації та інтеграції з різними джерелами даних. Це дозволяє ефективно працювати з даними, виявляти закономірності та використовувати ці результати для прийняття рішень у різних сферах.

ПЕРЕЛІК ПОСИЛАНЬ

1. **Moreira, J.** A General Introduction to Data Analytics [Текст] / J. Moreira, A. Carvalho, T. Horvath. – Wiley-Interscience, 2018. – 352 p.

2. **Serdobolskii, V.** Multiparametric Statistics [Текст] / V. Serdobolskii. – Elsevier Science, 2007. – 334 p.

3. **Rocca, M.** Advanced Algorithms and Data Structures [Текст] / M. Rocca, J. Brierley. – Manning Publications, 2021. – 245 p.

4. **Aljarah, I.** Evolutionary Data Clustering: Algorithms and Applications [Текст] / I. Aljarah, F. Hossam. – Springer, 2021. – 448 p.

5. **Fienberg, H.** High-Dimensional Single Cell Analysis [Текст] / H. Fienberg, G. Nolan. – Springer, 2014. – 370 p.

6. **Aggarwal, C.** Data Clustering: Algorithms and Applications [Текст] / C. Aggarwal, C. Reddy. – Chapman and Hall/CRC, 2018. – 652 p.

7. **Simovici, D.** Clustering: Theoretical And Practical Aspects [Текст] / D. Simovici. – World Scientific, 2021. – 1141 p.

8. **Перепелиця В. О.** Задачі класифікації: підходи, методи, алгоритми [Текст] / В. О. Перепелиця, І. В. Козін, Е. В. Терещенко. – Запоріжжя: Поліграф, 2008. – 188 с.

9. **Wu, J.** Advances in K-means Clustering: A Data Mining Thinking [Текст] / J. Wu. – Springer, 2012. – 198 p.

10. **Maarseveen, H.** Mastering K-Means Clustering [Текст] / H. Maarseveen. – Manning Publications, 2023. – 73 p.

11. **Celebi, E.** Partitional Clustering Algorithms [Текст] / E. Celebi. – Springer, 2014. – 708 p.

12. **Wilson, C.** Mathematical Modeling, Clustering Algorithms and Applications [Текст] / C. Wilson. – Nova Science, 2011. – 288 p.

13. **Yau, N.** Data Points: Visualization That Means Something [Текст] / N. Yau. – Wiley, 2013. – 320 p.

14. **Knaflic, C.** Storytelling with Data: A Data Visualization Guide for Business Professionals [Текст] / C. Knaflic. – Wiley, 2015. – 288 p.

15. Microsoft [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/ru-ru/dotnet/csharp/ (дата звернення: 16.12.2024).

16. Microsoft [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/ru-ru/ef/core// (дата звернення: 16.12.2024).

17. GitHub [Електронний ресурс]. – Режим доступу: https://github.com/microsoft/automatic-graph-layout (дата звернення: 16.12.2024).

18. Microsoft [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/ru-ru/sql (дата звернення: 16.12.2024).

19. Microsoft [Електронний ресурс]. – Режим доступу: https://dotnet.microsoft.com/ru-ru/apps/ai/ml-dotnet (дата звернення: 16.12.2024).

20. **Capellman, J.** Hands-On Machine Learning with ML.NET [Текст] / J. Capellman. – Packt Publishing, 2020. – 451 p.

21. ScottPlot [Електронний ресурс]. – Режим доступу: https://scottplot.net/ (дата звернення: 16.12.2024).

22. Fragile States Index [Електронний ресурс]. – Режим доступу: https://fragilestatesindex.org/excel/ (дата звернення: 16.12.2024).

ДОДАТОК А

Програмні коди

```
public abstract class BaseClusterer : ISettingsObserver
{
    protected readonly DataNormalizer dataNormalizer;
    protected readonly IDistanceCalculator distanceCalculator;

    protected abstract ClusterAlgorithm Algorithm { get; }

    protected UserSettings settings = default!;

    protected BaseClusterer(
        DataNormalizer dataNormalizer,
        DistanceCalculatorFactory distanceCalculatorFactory,
        ISettingsSubject settingsSubject
        )
    {
        this.dataNormalizer = dataNormalizer;
        distanceCalculator = distanceCalculatorFactory.Create(Algorithm);

        settingsSubject.Attach(this);
    }

    public abstract Task<List<Cluster>> ClusterAsync(NodeTable
nodeTable);

    public void Update(UserSettings settings) => this.settings = settings;
}
public class KMeansClusterer : BaseClusterer
```

```
    {
        private readonly Random random = new();
        private List<KMeansCluster> kMeansClusters;

        protected override ClusterAlgorithm Algorithm =>
ClusterAlgorithm.KMeans;

        public KMeansClusterer(
            DataNormalizer dataNormalizer,
            DistanceCalculatorFactory distanceCalculatorFactory,
            ISettingsSubject settingsSubject
            )
            : base(dataNormalizer, distanceCalculatorFactory, settingsSubject)
        { }

        public override async Task<List<Cluster>> ClusterAsync(NodeTable
nodeTable)
        {
            var nodes = await
dataNormalizer.GetCalculationNodesAsync(nodeTable, settings.ParameterStates);
            var parametersCount = nodes.FirstOrDefault()?.Parameters.Count ??
0;

            if (nodes.Count < settings.AlgorithmSettings.NumberOfClusters)
                throw new InvalidOperationException("Nodes amount is less than
the number of clusters");

            kMeansClusters = new
List<KMeansCluster>(settings.AlgorithmSettings.NumberOfClusters);
            InitializeClusters(nodes, parametersCount);
```

```
            for (int iteration = 0; iteration <
settings.AlgorithmSettings.MaxIterations; ++iteration)
            {
                var assignmentsChanged = false;

                foreach (var kMeansCluster in kMeansClusters)
                    kMeansCluster.Nodes.Clear();

                for (int i = 0; i < nodes.Count; ++i)
                {
                    var clusterIndex = GetNearestClusterIndex(nodes[i]);

                    if (kMeansClusters[clusterIndex].Nodes.Contains(nodes[i]))
                        continue;

                    assignmentsChanged = true;
                    kMeansClusters[clusterIndex].AddNode(nodes[i]);
                }

                if (!assignmentsChanged)
                    break;

                RecalculateClusters(nodes);
            }

            return kMeansClusters.Cast<Cluster>().ToList();
        }
```

```
        private void InitializeClusters(List<CalculationNode> nodes, int
parametersCount)
        {
            var selectedIndices = new HashSet<int>();

            for (int i = 0; i < settings.AlgorithmSettings.NumberOfClusters; ++i)
            {
                int randomIndex;
                do
                {
                    randomIndex = random.Next(nodes.Count);
                } while (!selectedIndices.Add(randomIndex));

                var cluster = new KMeansCluster(nodes[randomIndex]);
                kMeansClusters.Add(cluster);
            }
        }

        private int GetNearestClusterIndex(CalculationNode node)
        {
            var clusterIndex = 0;
            var minDistance = double.MaxValue;

            for (int i = 0; i < kMeansClusters.Count; ++i)
            {
                var distance = distanceCalculator.Calculate(node,
kMeansClusters[i].Centroid);

                if (distance >= minDistance)
                    continue;
```

```
                minDistance = distance;
                clusterIndex = i;
            }

            return clusterIndex;
        }

        private void RecalculateClusters(List<CalculationNode> data)
        {
            foreach (var kMeansCluster in kMeansClusters)
            {
                kMeansCluster.RecalculateCentroid();
            }
        }
    }
    public class KMeansCluster : Cluster
    {
        public Centroid Centroid { get; set; }

        public KMeansCluster(CalculationNode node)
        {
            Centroid = new Centroid(node);
        }

        public void RecalculateCentroid()
        {
            if (Nodes.Any(n => n.Parameters.Count !=
Centroid.Parameters.Count))
```

```
                    throw new InvalidOperationException("Node parameter count does
not match centroid's");

                Centroid.Merge(Nodes);
            }
        }
        public class DBSCANClusterer : BaseClusterer
        {
            private HashSet<CalculationNode> visitedNodes;

            protected override ClusterAlgorithm Algorithm =>
ClusterAlgorithm.DBSCAN;

            public DBSCANClusterer(
                DataNormalizer dataNormalizer,
                DistanceCalculatorFactory distanceCalculatorFactory,
                ISettingsSubject settingsSubject
                )
                : base(dataNormalizer, distanceCalculatorFactory, settingsSubject)
            { }

            public override async Task<List<Cluster>> ClusterAsync(NodeTable
nodeTable)
            {
                var nodes = await
dataNormalizer.GetCalculationNodesAsync(nodeTable, settings.ParameterStates);

                var clusters = new List<Cluster>();
                visitedNodes = new HashSet<CalculationNode>();
```

```
            foreach (var node in nodes)
            {
                if (visitedNodes.Contains(node))
                    continue;

                visitedNodes.Add(node);
                var neighbors = GetNeighbors(node, nodes);

                if (neighbors.Count < settings.AlgorithmSettings.MinPoints)
                    continue;

                var cluster = new Cluster();
                clusters.Add(cluster);

                ExpandCluster(cluster, node, neighbors, nodes);
            }

            return clusters;
        }

        private List<CalculationNode> GetNeighbors(CalculationNode node,
List<CalculationNode> nodes)
        {
            return nodes
                .Where(other => IsNeighbor(node, other))
                .ToList();
        }

        private bool IsNeighbor(CalculationNode node, CalculationNode other)
        {
```

```
            var distance = distanceCalculator.Calculate(node, other);
            return distance <= settings.AlgorithmSettings.Epsilon;
        }

        private void ExpandCluster(
            Cluster cluster,
            CalculationNode node,
            List<CalculationNode> neighbors,
            List<CalculationNode> nodes
            )
        {
            cluster.AddNode(node);

            var connectedNodes = new Queue<CalculationNode>(neighbors);

            while (connectedNodes.Count > 0)
            {
                var neighbor = connectedNodes.Dequeue();

                if (visitedNodes.Contains(neighbor))
                    continue;

                visitedNodes.Add(neighbor);
                var neighborNeighbors = GetNeighbors(neighbor, nodes);

                if (neighborNeighbors.Count >=
settings.AlgorithmSettings.MinPoints)
                {
                    foreach (var newNeighbor in neighborNeighbors)
                    {
```

```
                    if (!cluster.Nodes.Contains(newNeighbor) &&
!connectedNodes.Contains(newNeighbor))
                    {
                        connectedNodes.Enqueue(newNeighbor);
                    }
                }
            }

            if (!cluster.Nodes.Contains(neighbor))
            {
                cluster.AddNode(neighbor);
            }
        }
    }
}
public class AgglomerativeClusterer : BaseClusterer
{
    private List<AgglomerativeCluster> agglomerativeClusters;

    protected override ClusterAlgorithm Algorithm =>
ClusterAlgorithm.HierarchicalAgglomerative;

    public AgglomerativeClusterer(
        DataNormalizer dataNormalizer,
        DistanceCalculatorFactory distanceCalculatorFactory,
        ISettingsSubject settingsSubject
        )
        : base(dataNormalizer, distanceCalculatorFactory, settingsSubject)
    { }
```

```
        public override async Task<List<Cluster>> ClusterAsync(NodeTable
nodeTable)
        {
            var nodes = await
dataNormalizer.GetCalculationNodesAsync(nodeTable, settings.ParameterStates);

            agglomerativeClusters = nodes
                .ConvertAll(n => new AgglomerativeCluster(n));

            while (agglomerativeClusters.Count(c => !c.IsMerged) > 1)
            {
                var similarCluster = FindMostSimilarClusters();

                if (similarCluster.Similarity >
settings.AlgorithmSettings.Threshold)
                    break;

                agglomerativeClusters[similarCluster.FirstClusterId]
                    .Merge(agglomerativeClusters[similarCluster.SecondClusterId]);
            }

            return agglomerativeClusters
                .Where(c => !c.IsMerged)
                .Cast<Cluster>()
                .ToList();
        }

        private ClusterSimilarityResult FindMostSimilarClusters()
        {
            var clusterSimilarity = new ClusterSimilarityResult();
```

```
            for (int i = 0; i < agglomerativeClusters.Count; ++i)
            {
                if (agglomerativeClusters[i].IsMerged)
                    continue;

                for (int j = i + 1; j < agglomerativeClusters.Count; ++j)
                {
                    if (agglomerativeClusters[j].IsMerged)
                        continue;

                    var similarity = GetAverageDistance(
                        agglomerativeClusters[i],
                        agglomerativeClusters[j]
                        );

                    if (similarity > clusterSimilarity.Similarity)
                        continue;

                    clusterSimilarity.Update(i, j, similarity);
                }
            }

            return clusterSimilarity;
        }

        private double GetAverageDistance(AgglomerativeCluster first,
AgglomerativeCluster second)
        {
            return first.Nodes
```

```
                .SelectMany(firstNode => second.Nodes,
distanceCalculator.Calculate)
                .Average();
        }
    }
    public abstract class BaseGraphBuilder<TGraph> :
IGraphBuilder<TGraph>, ISettingsObserver
    {
        protected readonly ColorHelper colorHelper;

        protected UserSettings settings = default!;

        protected BaseGraphBuilder(
            ColorHelper colorHelper,
            ISettingsSubject settingsSubject
            )
        {
            this.colorHelper = colorHelper;

            settingsSubject.Attach(this);
        }

        public abstract TGraph Build(string name, List<NodeSimilarityResult>
similarityResults);

        public abstract TGraph Build(string name, List<NodeObject> nodes,
List<Cluster> clusters);

        public abstract TGraph Build(string name, List<NodeSimilarityResult>
similarityResults, List<Cluster> clusters);
```

```
        public void Update(UserSettings settings) => this.settings = settings;

        protected virtual void AddNodes(TGraph graph,
List<NodeSimilarityResult> similarityResults)
        {
            var minSimilarity = similarityResults.Min(result =>
result.SimilarNodes.Min(s => s.SimilarityPercentage));
            var maxSimilarity = similarityResults.Max(result =>
result.SimilarNodes.Max(s => s.SimilarityPercentage));
            var maxSimilarNodesAboveThreshold = similarityResults
                .Max(result => result.SimilarNodes
                    .Count(sn => sn.SimilarityPercentage >
settings.MinSimilarityPercentage)
                );

            foreach (var similarityResult in similarityResults)
            {
                var currentNodeName = similarityResult.Node.Name;

                var nodeColor =
colorHelper.GetNodeColorFromDensityWithSigmoid(
                    similarityResult,
                    maxSimilarNodesAboveThreshold,
                    settings.MinSimilarityPercentage
                    );

                AddNode(
                    graph,
                    currentNodeName,
```

```
                nodeColor
                );
            }
        }

        protected virtual void AddNodes(TGraph graph, List<NodeObject>
nodes, List<Cluster> clusters)
        {
            var clustersIds = clusters.ConvertAll(cluster => cluster.Id);
            var colors = colorHelper.GetDistinctColors(clustersIds);

            foreach (var node in nodes)
            {
                var currentNodeName = node.Name;
                var cluster = clusters
                    .FirstOrDefault(c => c.Nodes.Any(n => n.Name ==
currentNodeName));

                var nodeColor = cluster != null ? colors[cluster.Id] :
Color.WhiteSmoke;

                AddNode(
                    graph,
                    currentNodeName,
                    nodeColor
                );
            }
        }
```

```
        protected virtual void AddEdges(TGraph graph,
List<NodeSimilarityResult> similarityResults)
        {
            var addedEdges = new HashSet<CustomEdge>();

            foreach (var similarityResult in similarityResults)
            {
                var sourceNode = similarityResult.Node;

                foreach (var similarNode in similarityResult.SimilarNodes)
                {
                    if (IsNeighbor(similarNode))
                        continue;

                    var targetNode = similarNode.Node;
                    var edge = new CustomEdge(sourceNode, targetNode);

                    if (addedEdges.Any(e => e.Equals(edge)))
                        continue;

                    AddEdge(graph, sourceNode.Name, targetNode.Name,
similarNode.SimilarityPercentage);
                    addedEdges.Add(edge);
                }
            }
        }

        private bool IsNeighbor(SimilarNode similarNode) =>
similarNode.SimilarityPercentage < settings.MinSimilarityPercentage;
```

```
        protected double GetNodeSize(int connectionCount, int
maxConnections)
        {
            return 1 + connectionCount / (double)maxConnections * 20;
        }

        protected abstract void AddClusters(TGraph graph, List<Cluster>
clusters);
        protected abstract void AddNode(TGraph graph, string nodeName, Color
nodeColor);
        protected abstract void AddEdge(TGraph graph, string sourceName,
string targetName, float similarityPercentage);
    }
    public class MsaglGraphBuilder : BaseGraphBuilder<ExtendedGraph>
    {
        private const int MinEdgesPerNodeClusteredGraph = 3;

        public MsaglGraphBuilder(
            ColorHelper colorHelper,
            ISettingsSubject settingsSubject
            )
            : base(colorHelper, settingsSubject)
        { }

        public override ExtendedGraph Build(string name,
List<NodeSimilarityResult> similarityResults)
        {
            var graph = new ExtendedGraph(name, "Graph")
            {
                LayoutAlgorithmSettings = new MdsLayoutSettings(),
```

```
            };

            AddNodes(graph, similarityResults);
            AddEdges(graph, similarityResults);

            return graph;
        }

        public override ExtendedGraph Build(string name, List<NodeObject>
nodes, List<Cluster> clusters)
        {
            var graph = new ExtendedGraph(name, "Clusters")
            {
                LayoutAlgorithmSettings = new SugiyamaLayoutSettings(),
            };

            AddNodes(graph, nodes, clusters);
            AddClusters(graph, clusters);

            return graph;
        }

        public override ExtendedGraph Build(string name,
List<NodeSimilarityResult> similarityResults, List<Cluster> clusters)
        {
            var graph = new ExtendedGraph(name, "Clustered graph")
            {
                LayoutAlgorithmSettings = new MdsLayoutSettings(),
            };
```

```
            AddNodes(graph, similarityResults, clusters);
            AddEdges(graph, similarityResults);
            AddExtraEdges(graph, similarityResults);

            return graph;
        }

        protected override void AddClusters(ExtendedGraph graph,
List<Cluster> clusters)
        {
            graph.NodeDataMap.Clear();

            foreach (var cluster in clusters)
            {
                var subgraph = new Subgraph(cluster.Id.ToString())
                {
                    LabelText = string.Empty,
                };

                foreach (var node in cluster.Nodes)
                {
                    var existingNode = graph.FindNode(node.Name);

                    if (existingNode == null)
                        continue;

                    subgraph.AddNode(existingNode);
                }

                graph.RootSubgraph.AddSubgraph(subgraph);
```

```
            }
        }

        protected override void AddNodes(ExtendedGraph graph,
List<NodeSimilarityResult> similarityResults)
        {
            base.AddNodes(graph, similarityResults);

            graph.NodeDataMap = similarityResults
                .ToDictionary(sr => sr.Node.Name, sr => sr);
        }

        protected void AddNodes(ExtendedGraph graph,
List<NodeSimilarityResult> similarityResults, List<Cluster> clusters)
        {
            var nodes = similarityResults
                .ConvertAll(result => result.Node);

            base.AddNodes(graph, nodes, clusters);

            graph.NodeDataMap = similarityResults
                .ToDictionary(sr => sr.Node.Name, sr => sr);
        }

        protected void AddExtraEdges(ExtendedGraph graph,
List<NodeSimilarityResult> similarityResults)
        {
            foreach (var similarityResult in similarityResults)
            {
                var sourceNode = graph.FindNode(similarityResult.Node.Name);
```

```
                if (sourceNode == null)
                    continue;

                var edgesCount = sourceNode.InEdges.Count() +
                    sourceNode.OutEdges.Count();

                if (edgesCount > MinEdgesPerNodeClusteredGraph)
                    continue;

                var similarNodes = similarityResult.SimilarNodes
                    .Where(sn => sn.SimilarityPercentage > 0)
                    .OrderByDescending(sr => sr.SimilarityPercentage)
                    .Take(edgesCount + MinEdgesPerNodeClusteredGraph)
                    .ToList();

                foreach (var similarNode in similarNodes)
                {
                    var targetNode = graph.FindNode(similarNode.Node.Name);

                    if (graph.Edges.Any(e => HasEdge(e, sourceNode, targetNode)))
                        continue;

                    AddEdge(graph, sourceNode.Id, targetNode.Id,
similarNode.SimilarityPercentage);
                }
            }
        }
```

```
        protected override void AddNode(ExtendedGraph graph, string
nodeName, SystemColor nodeColor)
        {
            var node = new Node(nodeName)
            {
                LabelText = nodeName,
                Attr =
                {
                    FillColor = new MsaglColor(nodeColor.R, nodeColor.G,
nodeColor.B),
                }
            };

            graph.AddNode(node);
        }

        protected override void AddEdge(ExtendedGraph graph, string
sourceName, string targetName, float similarityPercentage)
        {
            var edge = graph.AddEdge(sourceName, targetName);

            edge.Attr.ArrowheadAtSource = ArrowStyle.None;
            edge.Attr.ArrowheadAtTarget = ArrowStyle.None;

            var minSimilarityPercentage =
Math.Min(settings.MinSimilarityPercentage, similarityPercentage);
            var edgeColor =
colorHelper.CalculateEdgeColor(similarityPercentage, minSimilarityPercentage);
            edge.Attr.Color = new MsaglColor(edgeColor.A, edgeColor.R,
edgeColor.G, edgeColor.B);
```

```
    }

    protected bool HasEdge(Edge edge, Node source, Node target)
    {
        return (edge.SourceNode == source && edge.TargetNode == target) ||
               (edge.SourceNode == target && edge.TargetNode == source);
    }
}
public class GraphCreationManager<TGraph> : ISettingsObserver
{
    protected readonly IGraphBuilder<TGraph> graphBuilder;
    protected readonly SimilarityComparer similarityComparer;
    protected readonly ClustererFactory clustererFactory;

    private UserSettings settings = default!;

    public GraphCreationManager(
        IGraphBuilder<TGraph> graphBuilder,
        SimilarityComparer similarityComparer,
        ClustererFactory clustererFactory,
        ISettingsSubject settingsSubject
        )
    {
        this.graphBuilder = graphBuilder;
        this.similarityComparer = similarityComparer;
        this.clustererFactory = clustererFactory;

        settingsSubject.Attach(this);
    }
```

```
        public void Update(UserSettings settings) => this.settings = settings;

        public async Task<TGraph> BuildGraphAsync(NodeTable nodeTable)
        {
            if (settings.UseClustering)
            {
                return await BuildClusteredGraphAsync(nodeTable);
            }

            return BuildGraph(nodeTable);
        }

        protected TGraph BuildGraph(NodeTable nodeTable)
        {
            var similarityResults =
similarityComparer.CalculateSimilarNodes(nodeTable.NodeObjects);

            return graphBuilder.Build(nodeTable.Name, similarityResults);
        }

        protected async Task<TGraph> BuildClusteredGraphAsync(NodeTable
nodeTable)
        {
            var clusterer =
clustererFactory.Create(settings.AlgorithmSettings.SelectedAlgorithm);
            var clusters = await clusterer.ClusterAsync(nodeTable);

            if (settings.UseClusteredGraph)
            {
```

```
                var similarityResults =
similarityComparer.CalculateSimilarNodes(nodeTable.NodeObjects);
                return graphBuilder.Build(nodeTable.Name, similarityResults,
clusters);
            }

            return graphBuilder.Build(nodeTable.Name, nodeTable.NodeObjects,
clusters);
        }
    }
    public class PlotCreationManager : ISettingsObserver
    {
        protected readonly ClustererFactory clustererFactory;
        protected readonly DataNormalizer dataNormalizer;
        protected readonly IDimensionReducer dimensionReducer;

        private UserSettings settings = default!;

        public PlotCreationManager(
            ClustererFactory clustererFactory,
            DataNormalizer dataNormalizer,
            IDimensionReducer dimensionReducer,
            ISettingsSubject settingsSubject
            )
        {
            this.clustererFactory = clustererFactory;
            this.dataNormalizer = dataNormalizer;
            this.dimensionReducer = dimensionReducer;

            settingsSubject.Attach(this);
```

```
        }

        public void Update(UserSettings settings) => this.settings = settings;

        public async Task<ClusteredPlot> BuildClusteredPlotAsync(NodeTable
nodeTable)
        {
            if (!settings.UseClustering)
                throw new InvalidOperationException("Clustering has not been
selected. Enable clustering to generate the plot.");

            var clusterer =
clustererFactory.Create(settings.AlgorithmSettings.SelectedAlgorithm);
            var clusters = await clusterer.ClusterAsync(nodeTable);

            var plotData = dimensionReducer.ReduceDimension(clusters);

            return new ClusteredPlot(nodeTable.Name, plotData);
        }
    }
    public class PlotConfigurator
    {
        private const int DefaultMarkerSize = 5;
        private const int DefaultLineWidth = 0;
        private const int DefaultLabelFontSize = 8;
        private const float DefaultLabelOffsetY = -10;

        private readonly ColorHelper colorHelper;

        private Plot plot = default!;
```

```
        public PlotConfigurator(ColorHelper colorHelper)
        {
            this.colorHelper = colorHelper;
        }

        public void UpdatePlot(FormsPlot formsPlot, ClusteredPlot
clusteredPlot)
        {
            plot = formsPlot.Plot;

            ConfigurePlot(clusteredPlot.Name);

            var clusterIds = clusteredPlot.Points
                .Select(p => p.ClusterId)
                .Distinct()
                .Order()
                .ToList();

            var colors = colorHelper.GetDistinctColors(clusterIds,
ColorBrightness.All);

            AddPointsToPlot(clusteredPlot.Points, colors);

            formsPlot.Plot.Legend.IsVisible = false;
            formsPlot.Refresh();
        }

        private void ConfigurePlot(string title)
        {
```

```
            plot.Clear();
            plot.Title(title);
            plot.XLabel("");
            plot.YLabel("");
            plot.Axes.Bottom.IsVisible = false;
            plot.Axes.Right.IsVisible = false;
            plot.Axes.Left.IsVisible = false;
        }

        private void AddPointsToPlot(List<ReducedDataPoint> points,
Dictionary<int, SystemColor> colorMap)
        {
            foreach (var point in points)
            {
                var color = point.ClusterId >= 0
                    ? colorMap[point.ClusterId]
                    : SystemColor.Gray;

                var scatter = plot.Add.Scatter(
                    new[] { (double)point.X },
                    new[] { (double)point.Y }
                );
                ConfigureScatter(scatter, point.Name, color);

                var label = plot.Add.Text(point.Name, point.X, point.Y);
                ConfigureLabel(label, point.Name, point.X, point.Y,
SystemColor.Black);
            }
        }
```

```
        private static void ConfigureScatter(Scatter scatter, string legendText,
SystemColor color)
        {
            scatter.Color = ScottColor.FromColor(color);
            scatter.MarkerStyle.Size = DefaultMarkerSize;
            scatter.LineStyle.Width = DefaultLineWidth;
            scatter.LegendText = legendText;
        }

        private static void ConfigureLabel(Text label, string text, double x,
double y, SystemColor color)
        {
            label.Alignment = Alignment.MiddleCenter;
            label.OffsetY = DefaultLabelOffsetY;
            label.LabelFontSize = DefaultLabelFontSize;
            label.LabelFontColor = ScottColor.FromColor(color);
        }
    }
```