Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавра

за спеціальності 121 – Інженерія програмного забезпечення

На тему: Порівняльний аналіз застосування рекомендаційних систем для оцінки переваги користувача

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.
Студент гр. ІПЗ-20-2
_____/М.А Ацеховський/

Керівник
кваліфікаційної роботи _____ /А.В Козиков/

Завідувач кафедри _____ /А.М.Стрюк/

Кривий Ріг
2024

Криворізький національний університет

Факультет: Інформаційних технологій

Кафедра: Моделювання та програмного забезпечення

Ступінь вищої освіти: бакалавр

Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:

Зав. кафедри

_____ А.М.Стрюк

«__» _____ 20__р.

ЗАВДАННЯ

на кваліфікаційну роботу

студенту групи ІПЗ-20-2 Ацеховському Микиті Андрійовичу

1. Тема: Порівняльний аналіз застосування рекомендаційних систем для оцінки переваги користувача. Затверджено наказом по КНУ №__ від «__» _____ 2024р.

2. Термін подання студентом закінченої роботи : «_» _____ 2024р.

3. Вихідні дані по роботі: датасети, методи реалізації рекомендаційних систем, алгоритми рекомендаційних систем

4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): потрібно провести огляд сучасних методів та алгоритмів рекомендаційних систем, проаналізувати варіанти оцінки переваг користувача в контексті рекомендаційних систем, також порівняти ефективність різних методів реалізації рекомендаційних систем, реалізувати програмні засоби для дослідження теми, зробити висновки та підбити підсумки.

5. Перелік ілюстративного матеріалу: графіки, блок-схеми, скріншоти екранних форм.

Календарний план**:**

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Огляд літератури за тематикою та збір даних	03.03.2024 – 08.03.2024
2	Проведення порівняльного аналізу різних методів реалізації рекомендаційних систем	10.03.2024 – 13.03.2024
3	Підготовка матеріалів першого розділу роботи	14.03.2024 – 19.03.2024
4	Розроблення програмного забезпечення для реалізації різних типів рекомендаційних систем	20.03.2024 – 28.03.2024
5	Підготовка матеріалів другого розділу роботи	01.04.2024 – 06.04.2024
6	Підготовка матеріалів третього розділу роботи	07.04.2024 – 13.04.2024
7	Розроблення програмного забезпечення для порівняння ефективності різних методів реалізації рекомендаційних систем для різних задач	14.04.2024 – 29.04.2024
8	Оформлення висновків	03.05.2024 – 04.05.2024
9	Оформлення появснювальної записки	06.05.2024 – 12.05.2024

Дата видачі завдання: «02» березня 2024р.

Студент _____/ М.А.Ацеховський

Керівник роботи _____/А.В.Козиков

РЕФЕРАТ

ПОРІВНЯЛЬНИЙ АНАЛІЗ ЗАСТОСУВАННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ ДЛЯ ОЦІНКИ ПЕРЕВАГИ КОРИСТУВАЧА, ДОСЛІДЖЕННЯ СПЕЦИФІКИ ВИКОРИСТАННЯ РІЗНИХ СПОСОБІВ ЇХ РЕАЛІЗАЦІЇ

Пояснювальна записка: 71 с., 7 табл., 38 рис., 1 дод., 11 джерел.

Рекомендаційна система – це реалізована за допомогою штучного інтелекту система, зазвичай також пов'язана з машинним навчанням, яка використовує великі набори даних, щоб пропонувати або рекомендувати користувачам різні продукти.

Рекомендаційні системи можуть ґрунтуватися на різних критеріях, включаючи минулі покупки, історію пошуку, демографічну інформацію та інші фактори. Рекомендаційні системи є дуже корисними, оскільки допомагають користувачам знаходити продукти та послуги, які вони могли б не знайти самостійно.

Технологія рекомендаційних систем широко використовується в різних галузях, зокрема в електронній комерції, стрімінгових платформах, новинах і медіа та цифровому маркетингу, щоб підвищити залученість користувачів, зміцнити довіру користувачів, збільшити продажі та покращити загальний рівень задоволеності клієнтів.

Ця кваліфікаційна робота спрямована на проведення порівняльного аналізу різних методів та підходів до визначення переваг користувачів у контексті рекомендаційних систем. Основний акцент буде зроблено на оцінці ефективності цих методів та їх можливості забезпечити персоналізовані та релевантні рекомендації.

ABSTRACT

COMPARATIVE ANALYSIS OF THE USE OF RECOMMENDER SYSTEMS FOR ASSESSING USER PREFERENCES, STUDY OF THE SPECIFICS OF USING DIFFERENT METHODS OF THEIR IMPLEMENTATION

Thesis in 71 p., 7 tables, 38 figure, 1 appendix, 11 sources.

A recommender system is an artificial intelligence-powered system, usually also related to machine learning, that uses large data sets to suggest or recommend different products to users.

Recommender systems can be based on various criteria, including past purchases, search history, demographic information, and other factors. Recommender systems are very useful because they help users find products and services that they might not have found on their own.

Recommender system technology is widely used in various industries, including e-commerce, streaming platforms, news and media, and digital marketing, to increase user engagement, build user trust, increase sales, and improve overall customer satisfaction.

This qualification work aims to conduct a comparative analysis of different methods and approaches to determining user preferences in the context of recommender systems. The main focus will be on evaluating the effectiveness of these methods and their ability to provide personalized and relevant recommendations.

ЗМІСТ

ВИКОРИСТАНІ ТЕРМІНИ ТА СКОРОЧЕННЯ

1. РС – рекомендаційна система.

2. Проблема холодного старту - потенційна проблема в комп'ютерних інформаційних системах, яка передбачає певний ступінь автоматизованого моделювання даних. Зокрема, йдеться про те, що система не може робити висновки для користувачів або об'єктів, про які вона ще не зібрала достатньо інформації.

3. Датасет або набір даних - це сукупність даних, пов'язаних з певною темою, предметом або галуззю. Набори даних включають різні типи інформації, такі як числа, текст, зображення, відео та аудіо, і можуть зберігатися в різних форматах, таких як CSV, JSON або SQL.

ВСТУП

Застосування рекомендаційних систем є ключовим аспектом в сучасному інформаційному середовищі де великий обсяг даних неможливо обробити вручну. Ці системи не лише допомагають користувачам знаходити інформацію та товари, що відповідають їхнім інтересам, а й мають значний вплив на різні сфери діяльності, від електронної комерції до соціальних медіа та культурної сфери.

Хороший механізм рекомендацій передбачає, що може зацікавити відвідувача, і спрямувати його до найбільш релевантного контенту - будь то продукти, послуги чи інформація. Основний плюс цього полягає в тому, що відвідувач отримує доступ до контенту, про який він, можливо, не знав, але завдяки штучному інтелекту (ШІ) може про нього дізнатися.

Існує велика кількість способів реалізації рекомендаційних систем, і кожен з них відрізняється за своєю складністю, ефективністю та застосовністю в конкретних сферах. Враховуючи різноманіття завдань та контекстів, в яких можуть використовуватися рекомендаційні системи, важливо визначити оптимальний метод для кожного конкретного випадку.

Деякі з найпоширеніших методів включають у себе коллаборативний та контентний підходи, гібридні системи, фільтрацію на основі змісту, збір і аналіз Big Data, машинне навчання та штучний інтелект. Кожен з цих методів має свої переваги та недоліки.

Наприклад, коллаборативні методи рекомендацій використовують інформацію про спільну взаємодію користувачів з продуктами або контентом, тоді як контентні методи враховують сам характер контенту та його атрибути. Гібридні системи поєднують у собі обидва підходи для отримання кращої точності та роботи в різних ситуаціях.

Навіть у межах кожного з цих методів існує ряд підвидів та варіацій, які можуть бути оптимізовані для конкретних завдань.

Різноманітність способів реалізації рекомендаційних систем відображається у їхній архітектурі, алгоритмах та методологіях використання даних.

У цій кваліфікаційній роботі буде проведено порівняльний аналіз різних методів та підходів до реалізації рекомендаційних систем на різних типах даних таких як музика, фільми, товари і т.д.

Дослідження охопить аналіз технологічних підходів, алгоритмів та стратегій, які використовуються для збору, обробки та аналізу даних користувачів з метою створення персоналізованих рекомендацій.

Крім того, буде звернута увага на особливості впровадження та ефективність рекомендаційних систем на різних платформах з урахуванням їхніх специфічних вимог і контексту використання.

Аналіз та реалізація різних рекомендаційних систем дозволить виявити переваги та недоліки кожного підходу, а також зробити висновки щодо оптимальних стратегій реалізації рекомендаційних систем для різних типів даних з метою підвищення задоволеності користувачів та покращення їхнього досвіду використання програмних продуктів.

1 СУЧАСНИЙ СТАН І АКТУАЛЬНІСТЬ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1.1 Актуальність теми кваліфікаційної роботи

З ростом обсягів інформації та контенту в інтернеті, користувачам стає складніше знаходити та відбирати те, що відповідає їхнім інтересам та потребам.

Рисунок 1.1 – Кількість людей які регулярно користуються інтернетом в Україні (дослідження Factum Group)

Рекомендаційні системи стають важливим інструментом для забезпечення персоналізованого досвіду користувачів та підвищення їхньої задоволеності від використання цифрових платформ, таких як інтернет-магазини, соціальні медіа, стрімінгові сервіси і т.д.

Основними плюсами рекомендаційних систем можна назвати:

1) те, що існує досить мало способів досягти збільшення продажів без посилення маркетингових зусиль, і система рекомендацій - один з них. Налаштувавши автоматизовану систему рекомендацій

компанія/магазин/сервіс отримує постійні додаткові продажі без жодних зусиль, оскільки вона набагато швидше знаходить потрібний товар для покупця;

2) рекомендаційні системи дозволяють скоротити шлях клієнта до продажу, рекомендуючи йому відповідний варіант, іноді навіть до того, як він сам його шукатиме;

3) листи з рекомендаційною системою - один з найкращих способів повторного залучення клієнтів. Знижки або купони - інші ефективні, але дорогі способи повторного залучення клієнтів, і їх можна поєднувати з рекомендаціями, щоб підвищити ймовірність конверсії клієнтів.

Зростаюча конкуренція у цифрових середовищах змушує компанії шукати нові способи підвищення залученості користувачів та забезпечення їхнього задоволення від використання сервісів.

Ефективні рекомендаційні системи можуть забезпечити значний конкурентний перевагу, тому дослідження їхнього застосування та порівняльний аналіз різних підходів є актуальними та важливими для практичного застосування в будь-якій сфері.

1.2 Цілі та завдання кваліфікаційної роботи

Цілі та завдання кваліфікаційної роботи полягають у проведенні порівняльного аналізу різних методів реалізації рекомендаційних систем на мові Python з метою визначення їхньої ефективності та придатності для оцінки переваг користувача.

Крім того, необхідно дослідити які типи рекомендаційних систем найбільш ефективно вирішують завдання оцінки переваг користувача в різних контекстах.

Завдання включають огляд наукової літератури, реалізацію та налаштування різних типів рекомендаційних систем, збір та підготовку даних для тестування, проведення експериментальних досліджень та аналіз результатів з метою визначення найбільш ефективних методів.

Також в ході дослідження теми потрібно детально ознайомитися з принципами роботи та особливостями різних типів РС (на основі фільтрів, на основі сусідства, collaborative filtering).

Розробити програмне забезпечення для реалізації вибраних методів РС на мові Python. Також провести експерименти з різними методами РС на тестових наборах даних. Та в кінці порівняти ефективність різних методів РС та зробити висновки щодо їх доцільності для різних задач.

1.3 Критичний аналіз літературних джерел за темою

Я проаналізував багато літературних та наукових джерел, присвячених рекомендаційним системам. Дослідження включали:

1) огляд методів рекомендаційних систем, таких як колаборативна фільтрація, контентна фільтрація та гібридні методи;

2) аналіз алгоритмів, що використовуються в кожному методі, з акцентом на їхні переваги та недоліки;

3) порівняння результатів досліджень, що оцінюють ефективність різних методів в різних контекстах;

4) вивчення бібліотек та фреймворків, які використовуються для розробки рекомендаційних систем.

За результатами аналізу вдалося визначити кілька ключових висновків. Перш за все, найефективніші методи реалізації рекомендаційних систем базуються на комбінації колаборативного та контентного підходів. Це дозволяє отримати більш точні рекомендації, враховуючи як особисті вподобання користувачів, так і характеристики самого контенту.

Також дуже часто в реалізації сучасним рекомендаційних систем використовується фільтрування на основі вмісту.

Фільтрація на основі вмісту використовує властивості елементів, щоб рекомендувати інші елементи, схожі на той, що сподобався користувачеві, на основі його попередніх дій або явних відгуків.

Рисунок 1.2 - Принцип фільтрування на основі вмісту

Деякі з найбільш часто використовуваних бібліотек для реалізації рекомендаційних систем включають:

Таблиця 1.1 – Бібліотеки які частіше за все використовуються при реалізації РС.

Назва бібліотеки	Опис
Surprise	Python-бібліотека для побудови та оцінювання рекомендаційних систем на основі колаборативного фільтрування та інших методів.
LightFM	Python-бібліотека, яка надає реалізацію різних алгоритмів рекомендаційних систем, включаючи колаборативне та контентне фільтрування, з використанням моделі факторизації матриць.

Продовження таблиці 1.1.

TensorFlow Recommenders	Фреймворк TensorFlow, який надає інструменти для реалізації різноманітних алгоритмів рекомендаційних систем, включаючи глибокі нейронні мережі та моделі з використанням векторних представлень.
scikit-surprise	Python-бібліотека для реалізації різних алгоритмів рекомендаційних систем, яка надає простий та зручний інтерфейс для розробки та оцінювання моделей на основі колаборативного фільтрування.

Важливою частиною аналізу була також критична оцінка достовірності та авторитетності джерел. Дослідження всі були актуальні, останні 5 років, адже за цей час методи реалізації РС дуже сильно покращилися.

У цілому, результати аналізу літературних джерел підтверджують важливість поєднання різних підходів та використання сучасних бібліотек для досягнення оптимальних результатів у сфері рекомендаційних систем.

2 РОЗРОБКА ФУНКЦІОНАЛЬНОЇ СХЕМИ І АЛГОРИТМУ

2.1 Алгоритми реалізації рекомендаційних систем

У ландшафті сучасних рекомендаційних систем використовуються різні алгоритми для ефективного прогнозування та пропонування елементів, що цікавлять користувачів.

Структура найпопулярніших алгоритмів рекомендаційних систем зображена на рисунку 2.1.

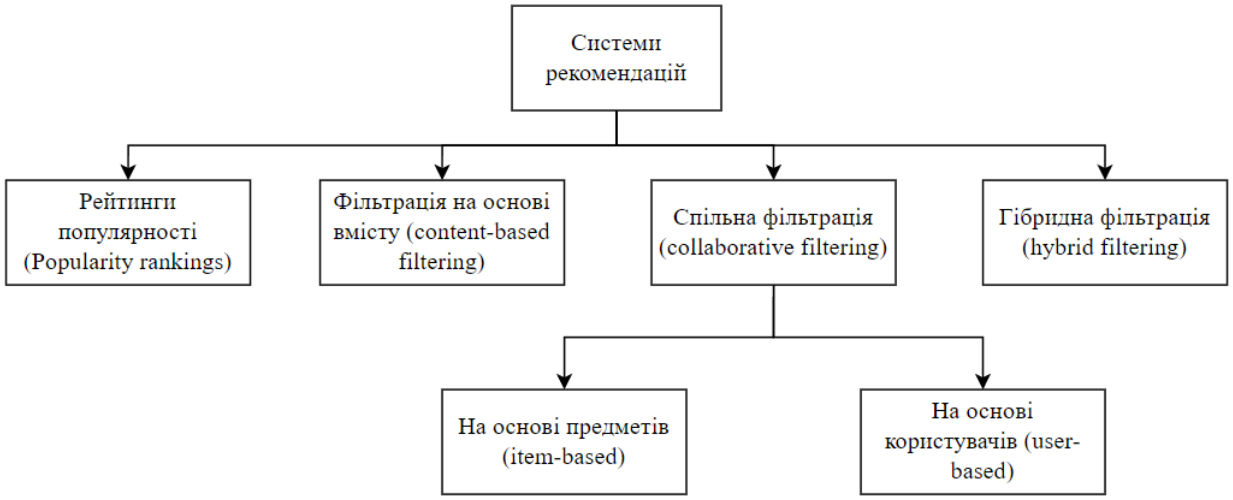


Рисунок 2.1 - Основні алгоритми реалізації рекомендаційних систем

Рейтинги популярності - Рекомендації генеруються на основі популярності товарів/послуг, не використовує інформацію про користувачів. Простий, але не завжди дає точні рекомендації.

Фільтрація на основі вмісту - рекомендації генеруються на основі характеристик товарів/послуг, які сподобалися користувачу раніше. Враховує опис товару, жанр, категорію тощо. Може бути неточним для нових користувачів.

Спільна фільтрація - рекомендації генеруються на основі поведінки схожих користувачів. Враховує рейтинги, покупки, перегляди інших людей. Ефективна, але потребує багато даних.

Гібридна фільтрація - поєднує два або більше алгоритмів для кращої точності. Може використовувати рейтинги, характеристики, поведінку. Найефективніший тип, але й найскладніший.

2.1.1 Спільна фільтрація на основі користувачів (User-based Collaborative Filtering)

Спільна фільтрація на основі користувачів (User-based Collaborative Filtering) - це метод, який використовується для передбачення об'єктів, які можуть сподобатися користувачеві, на основі оцінок, виставлених цим об'єктам іншими користувачами, які мають схожі смаки зі смаками цільового користувача. Чимало веб-сайтів використовують колаборативну фільтрацію для побудови своїх рекомендаційних систем.

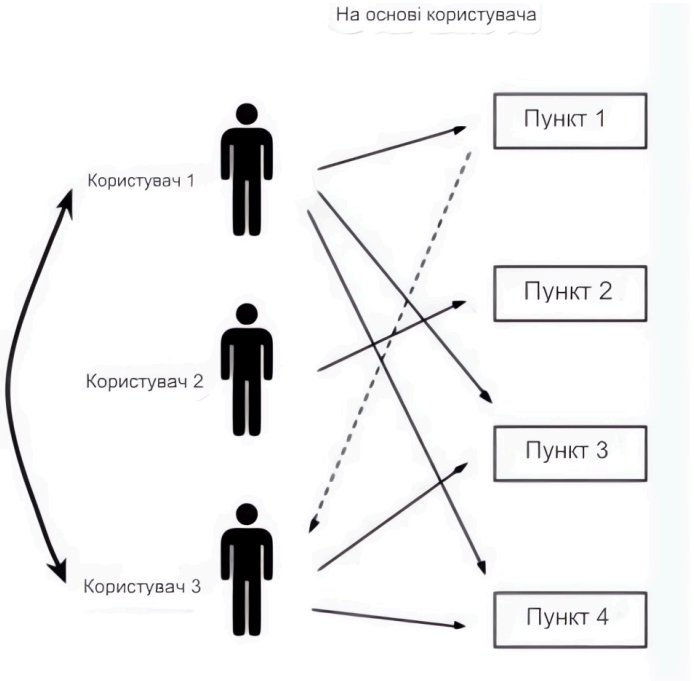


Рисунок 2.2 - Алгоритм роботи спільної фільтрації на основі користувача

Розглянемо на конкретному прикладі зображеному на рисунку 2.3.

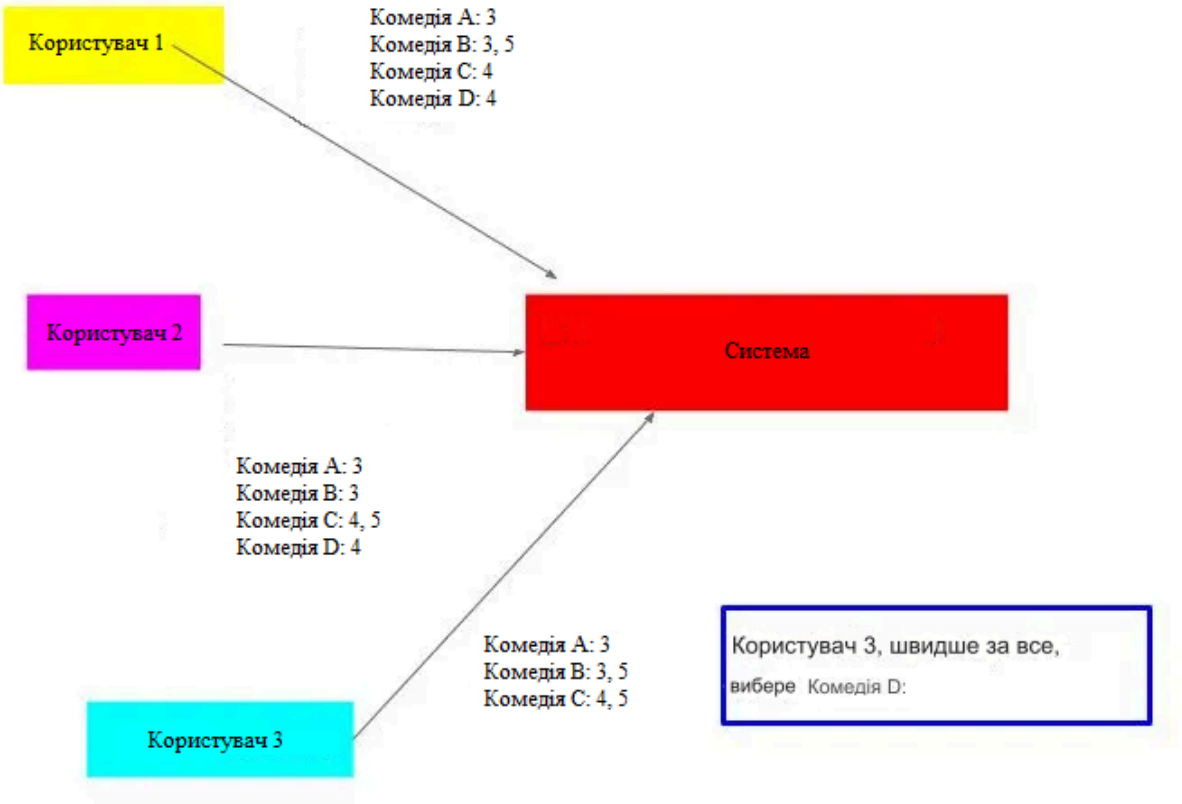


Рисунок 2.3 - Приклад роботи спільної фільтрації на основі користувача

Користувач 1 подивився комедії A, B, C, D і поставив оцінку відповідно до своїх інтересів.

Користувач 2, скоріш за все, також подивився ті ж самі комедії і надав свої вподобання в системі оцінювання.

Користувач 3 переглядає рекомендації, і на основі історії його вибору система порівнює оцінки цього користувача з оцінками інших користувачів і знаходить людей з найбільш "схожими" смаками, тобто в цьому сценарії Користувача 1 і Користувача 2.

Відповідно, система рекомендує "Комедію D" Користувачеві 3, оскільки він, швидше за все, подивиться його і йому сподобається.

Тобто як можна побачити цей алгоритм базується на припущенні, що користувачі, які мають схожі вподобання або історії взаємодії з системою, будуть мати схожі вподобання у майбутньому. Він використовує схожість між користувачами для рекомендації предметів.

Основними його перевагами є простота реалізації, ефективність для невеликих наборів даних, здатність до персоналізації.

Недоліками є проблема холодного старту (cold start problem[1]), неефективність для великих наборів даних, проблема шкалювання зі зростанням користувачів і предметів.

2.1.2 Спільна фільтрація на основі предметів (Item-based Collaborative Filtering)

Спільна фільтрація на основі предметів - це такий вид РС який шукає схожі товари на основі товарів, які вже сподобалися користувачам або з якими вони позитивно взаємодіяли.

Принцип роботи IBCF (Item-based Collaborative Filtering) полягає в тому, що він пропонує позиції на основі позицій, які користувач споживав раніше. Вона шукає продукти, які користувач споживав, потім знаходить інші продукти, схожі на спожиті, і рекомендує їх.

Можна розглянути це на прикладі рисунку 2.4.

[1] Cold start problem - це потенційна проблема в комп'ютерних інформаційних системах, які передбачають певний ступінь автоматизованого моделювання даних. Зокрема, йдеться про те, що система не може зробити жодних висновків для користувачів або об'єктів, про які вона ще не зібрала достатньо інформації.

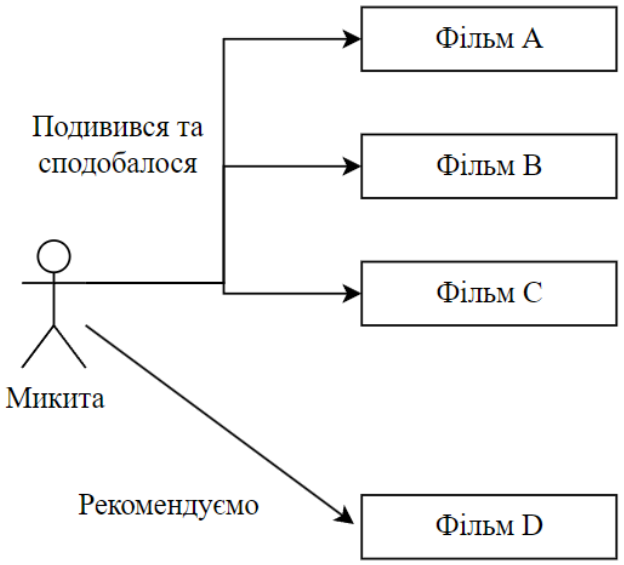


Рисунок 2.4 - Приклад роботи спільної фільтрації на основі предмету

Припустимо, наш користувач Микита хоче придбати DVD з фільмом. Наше завдання - рекомендувати йому фільм на основі його минулих уподобань. Спочатку ми шукатимемо фільми, які Микита переглядав або які йому сподобалися, назвемо їх фільмами «A», «B» і «C». Потім ми будемо шукати інші фільми, схожі на ці три фільми. Припустимо, ми з'ясували, що фільм 'D' дуже схожий на 'C', отже, ймовірність того, що Микита також сподобається фільм 'D', дуже велика, оскільки він схожий на той, який вже сподобався Микиті. Таким чином, ми запропонуємо фільм 'D'.

Основними перевагами такого алгоритму є ефективність для рекомендацій у ситуаціях з великою кількістю предметів та невеликою кількістю користувачів, здатність працювати з новими користувачами, оскільки модель базується на схожості предметів, а не користувачів, та звичайно висока точність рекомендацій у випадку, коли існують чіткі патерни в споживанні.

Недоліками є наявність Cold start problem, велика обчислювальна складність при обробці великої кількості предметів та вразливість до шуму

або випадкових оцінок, оскільки модель може реагувати на неправильні оцінки користувачів.

2.1.3 Рейтинги популярності (Popularity ranking)

Система рекомендацій за рейтингом популярності - це тип рекомендаційної системи, яка пропонує користувачам елементи на основі їхньої популярності серед інших користувачів. Замість того, щоб надавати персоналізовані рекомендації на основі індивідуальних уподобань або поведінки користувача, системи рейтингу популярності покладаються на загальну популярність або тенденції елементів у певному наборі даних.

Хоча системи рекомендацій за рейтингом популярності відносно прості в реалізації та обчислювально ефективні, вони мають певні обмеження:

1) по-перше - це відсутність персоналізації. Оскільки рекомендації базуються виключно на показниках популярності, а не на індивідуальних уподобаннях користувача, система не може надавати персоналізовані рекомендації;

2) системи, засновані на популярності, як правило, рекомендують лише найпопулярніші товари, що може призвести до недостатньої різноманітності рекомендацій. Користувачам може бути представлений вузький спектр товарів, і вони можуть пропустити менш відомі, але релевантні варіанти;

3) проблема "холодного старту": новим об'єктам може бути важко отримати видимість і популярність у системі, особливо якщо вони ще не отримали достатньої взаємодії з користувачами. Це може стати проблемою для рекомендацій нещодавно доданих або нішевих товарів.

Але звичайно системи рекомендацій за рейтингом популярності мають і декілька переваг:

1) вони відносно прості в реалізації та ефективні з точки зору обчислень. Не вимагають складних алгоритмів або обробки великої кількості даних користувачів. Ця простота та ефективність робить popularity ranking

систем особливо придатними для застосувань, де потрібні рекомендації в режимі реального часу, наприклад, для платформ електронної комерції та сервісів потокового передавання контенту;

2) рейтинг популярності ґрунтується на колективній поведінці та вподобаннях користувачів, що може вселяти довіру та впевненість у користувачів. Користувачі можуть відчувати себе комфортніше, взаємодіючи з рекомендованими товарами, які вже довели свою популярність серед інших користувачів, оскільки вони сприймають їх як надійні або якісні.

2.1.4 Фільтрація на основі вмісту (Content-based filtering)

Фільтрація на основі вмісту в рекомендаційних системах використовує алгоритми машинного навчання, щоб передбачити і рекомендувати користувачеві нові, але схожі товари. Рекомендувати товари на основі їхніх характеристик можна лише за наявності чіткого набору характеристик товару та переліку варіантів вибору користувача.

Така система рекомендацій зберігає попередні дані користувача, такі як кліки, рейтинги та вподобання, щоб створити профіль користувача. Чим більше клієнт взаємодіє з системою, тим точнішими будуть майбутні рекомендації.

Наприклад, якщо ми розглядаємо рекомендаційну систему для рекомендації статей, алгоритм фільтрації на основі вмісту може аналізувати текст кожної статті. Потім він може порівнювати цей текст з іншими статтями на основі ряду критеріїв, таких як тема, ключові слова, стиль письма тощо. Якщо користувач прочитав та оцінив певну статтю, система може рекомендувати інші статті зі схожим вмістом або тематикою.

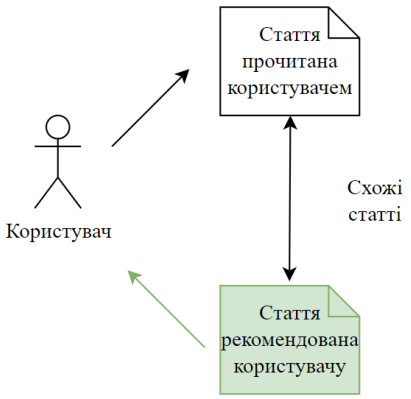


Рисунок 2.5 - Приклад роботи фільтрації на основі вмісту

Наприклад, якщо користувач зацікавлений у статтях про космос, система може рекомендувати йому інші статті з космічною тематикою, навіть якщо вони публікуються різними виданнями або не мають багато спільних читачів. Це дає можливість забезпечити персоналізовані рекомендації, які враховують індивідуальні інтереси кожного користувача, незалежно від того, як вони відрізняються від інших користувачів.

Плюси фільтрації на основі вмісту:

1) забезпечує персоналізовані рекомендації, орієнтовані на індивідуальні інтереси користувачів;

2) рекомендації можуть бути надані навіть для об'єктів, які не мають багато спільних користувачів;

3) може надавати рекомендації навіть для нових об'єктів, оскільки аналізує вміст, а не історію взаємодії користувачів.

Мінуси фільтрації на основі вмісту:

1) може бути складно рекомендувати об'єкти, які мають складну структуру або де недостатньо інформації для аналізу вмісту;

2) для точних рекомендацій потрібно провести аналіз великого обсягу даних, що може бути затратним в сенсі часу та ресурсів.

-

2.1.5 Порівняння ефективності алгоритмів рекомендаційних систем для різних типів контенту

Для різного контенту може бути різна ефективність алгоритмів, залежно від особливостей самого контенту, вподобань користувачів, обраної метрики оцінки тощо.

Наприклад, алгоритми на основі колаборативного фільтрування (User-based та Item-based CF) можуть бути дуже ефективними для рекомендацій фільмів або музики, коли взаємодія між користувачами та об'єктами велика. Однак для менш популярних або специфічних типів контенту, таких як книги або маловідомі музичні жанри, ці алгоритми можуть не давати задовільних результатів через обмежену кількість оцінок користувачів.

В таблиці 2.1 описано ефективність використання різних типів алгоритмів з використанням таких параметрів як точність, покриття, ресурсні витрати, та можливості адаптації.

Таблиця 2.1 – Оцінка ефективності різних алгоритмів рекомендаційних систем

Алгоритм	Типи контенту	Точність	Обсяги даних	Обчислювальна складність	Гнучкість (здатність адаптуватися до нових даних)
User-based CF	Різноманітний	Висока	Середня	Висока	Висока
Item-based CF	Різноманітний	Висока	Висока	Висока	Висока

Продовження таблиці 2.1.

Popularity ranking	Універсальний	Низька	Висока	Низька	Висока
Content-based filtering	Залежить від типу	Середня	Висока	Низька	Висока
Hybrid filtering	Залежить від компонентів	Висока	Висока	Висока	Висока

Звичайно різні алгоритми будуть ефективні для різних типів контенту, наприклад:

Музика:

1) user-based Collaborative Filtering. Для музичних рекомендацій цей метод може бути ефективним, оскільки схожість музичних смаків між користувачами може бути відносно стійкою. Точність може бути високою для користувачів зі схожими уподобаннями;

2) item-based Collaborative Filtering. Схожий на попередній, але базується на схожості між музичними треками. Цей метод може бути ефективним, оскільки музичні треки можуть мати явні зв'язки між собою, такі як спільні жанри або виконавці;

3) popularity Ranking. Цей метод може бути ефективним для новачків або для користувачів, які шукають популярні треки, але він може бути менш ефективним для користувачів з унікальними смаками;

4) content-based Filtering. Музичні рекомендації на основі вмісту можуть враховувати атрибути музики, такі як жанр, темп, інструментальність тощо. Це може бути ефективним для користувачів зі специфічними уподобаннями;

5) hybrid Filtering. Комбінація різних методів може дати найкращі результати, враховуючи як інформацію про користувачів, так і про вміст.

Фільми:

Фільми мають різні аспекти, такі як жанр, актори, режисери, сюжетні лінії тощо. Hybrid Filtering, поєднуючи наприклад User-based Collaborative Filtering і Content-based Filtering, може бути найефективнішим, оскільки це дозволяє враховувати як історію переглядів користувачів, так і характеристики самого фільму.

Книги:

1) user-based і Item-based Collaborative Filtering. Можуть бути ефективними, оскільки читаці часто мають стійкі уподобання до жанрів чи авторів;

2) popularity Ranking. Може бути ефективним для відновлення популярних книг, але менш ефективним для специфічних або нішевих жанрів;

3) content-based Filtering. Може бути ефективним для рекомендацій на основі схожості вмісту книг. Але цей метод буде дуже ресурсно затратним адже потребує аналізу вмісту всіх кних;

4) hybrid Filtering. Знову, комбінація методів може забезпечити кращі результати.

Ігри:

1) user-based і Item-based Collaborative Filtering. Для онлайн-ігор ці методи можуть бути ефективними, враховуючи історію гравців та їхніх уподобань;

2) popularity Ranking. Може бути ефективним для широко відомих або популярних ігор, але не для менш відомих або нішевих творів;

3) content-based Filtering. Може бути ефективним для ігор на основі атрибутів гри, таких як жанр, графіка, механіка гри тощо;

4) hybrid Filtering. Використання комбінації методів може забезпечити більш персоналізовані рекомендації для гравців.

Можна зробити деякі попередні висновки які в подальшому будуть прикріплені дослідженнями.

Для музики найкращим алгоритмом є Content-based Filtering, оскільки він базується на властивостях самого контенту, таких як жанр, виконавець, текст пісні тощо. Для фільмів найкращим буде Hybrid Filtering, який поєднує User-based Collaborative Filtering і Content-based Filtering для кращої персоналізованої рекомендації. Для книг ефективним методом є Content-based Filtering. У світі ігор найкращим варіантом є Item-based Collaborative Filtering, який рекомендує ігри на основі їхньої схожості між собою, що зазвичай відповідає уподобанням гравців.

2.2 Опис функціональних та нефункціональних вимог до системи

Функціональні вимоги - це те, що система повинна робити (наприклад, аналізувати методи рекомендаційних систем), а нефункціональні вимоги - це як система повинна працювати (наприклад, ефективно та зручно для користувачів).

Функціональні вимоги до фінальної системи наступні:

1) сучасні методи реалізації РС. Система повинна забезпечувати можливість детального огляду та аналізу різних методів рекомендаційних систем, таких як колаборативне фільтрування, контентне фільтрування, гібридні підходи тощо;

2) система повинна досліджувати різні способи оцінки вподобань користувачів, включаючи рейтингові системи, звичайні відповіді "так" або "ні", інтерактивні оцінки, і т.д;

3) проведення порівняльного аналізу ефективності. Система повинна забезпечувати можливість проведення експериментів з різними

методами рекомендаційних систем та оцінювати їх ефективність за допомогою метрик, таких як точність, відновлення тощо.

Також система повинна включати в себе програмні інструменти для проведення досліджень та експериментів з рекомендаційними системами.

Нефункціональні вимоги до системи наступні:

1) ефективність та продуктивність. Система повинна працювати ефективно та швидко, навіть при обробці великих обсягів даних;

2) масштабованість. Система повинна бути здатна масштабуватися для обробки ростучого обсягу даних та користувачів;

3) система повинна бути надійною та стійкою до помилок;

4) система повинна бути побудована з урахуванням модульної архітектури, щоб дозволити легке розширення та модифікацію функціональності в майбутньому.

2.3 Розробка та детальний опис функціональної схеми програми

Фінальне реалізоване програмне забезпечення буде складатися з декількох блоків, різні методи РС будуть реалізовані під різні типи даних (книги, комп’ютерні ігри, музика). Для кожного методу буде проведена оцінка ефективності.

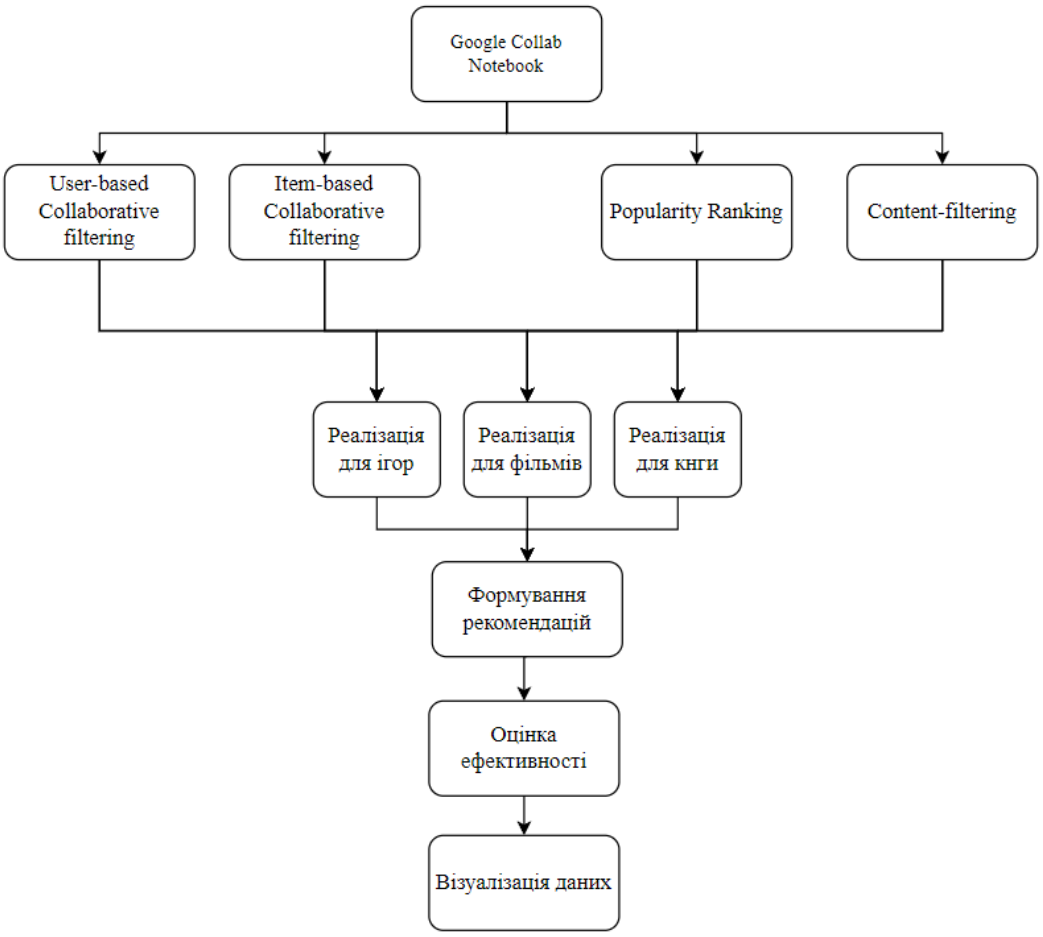


Рисунок 2.6 - Загальна структурна схема ПЗ

Блок-схема на рисунку 2.6 описує 4 основні методи реалізації РС які будуть використані для порівняння. Всі вони будуть реалізовані для різних типів даних, після цього будуть сформовані рекомендації і буде оцінена ефективність. Для оцінки може використовуватися ряд метрик, таких як точність, accuracy та F1-міра.

Діаграма компонентів надає концептуальну картину взаємодії між різними системами. Можуть бути присутні як аспекти логічного, так і фізичного моделювання.

На рисунку 2.7 зображена діаграма компонентів яка складається з наступних елементів:

Google Collab. Це хмарна платформа для машинного навчання. Вона містить в собі файл розширення .ipynb та завантажені необхідні бібліотеки.

Контент: Фільми, ігри та книги. Це елементи які будуть використані для реалізації рекомендаційних систем.

Датасети films.csv, games.csv, books.csv необхідні при навчанні та реалізації РС.

База даних в якій будуть зберігатися необхідні дані.

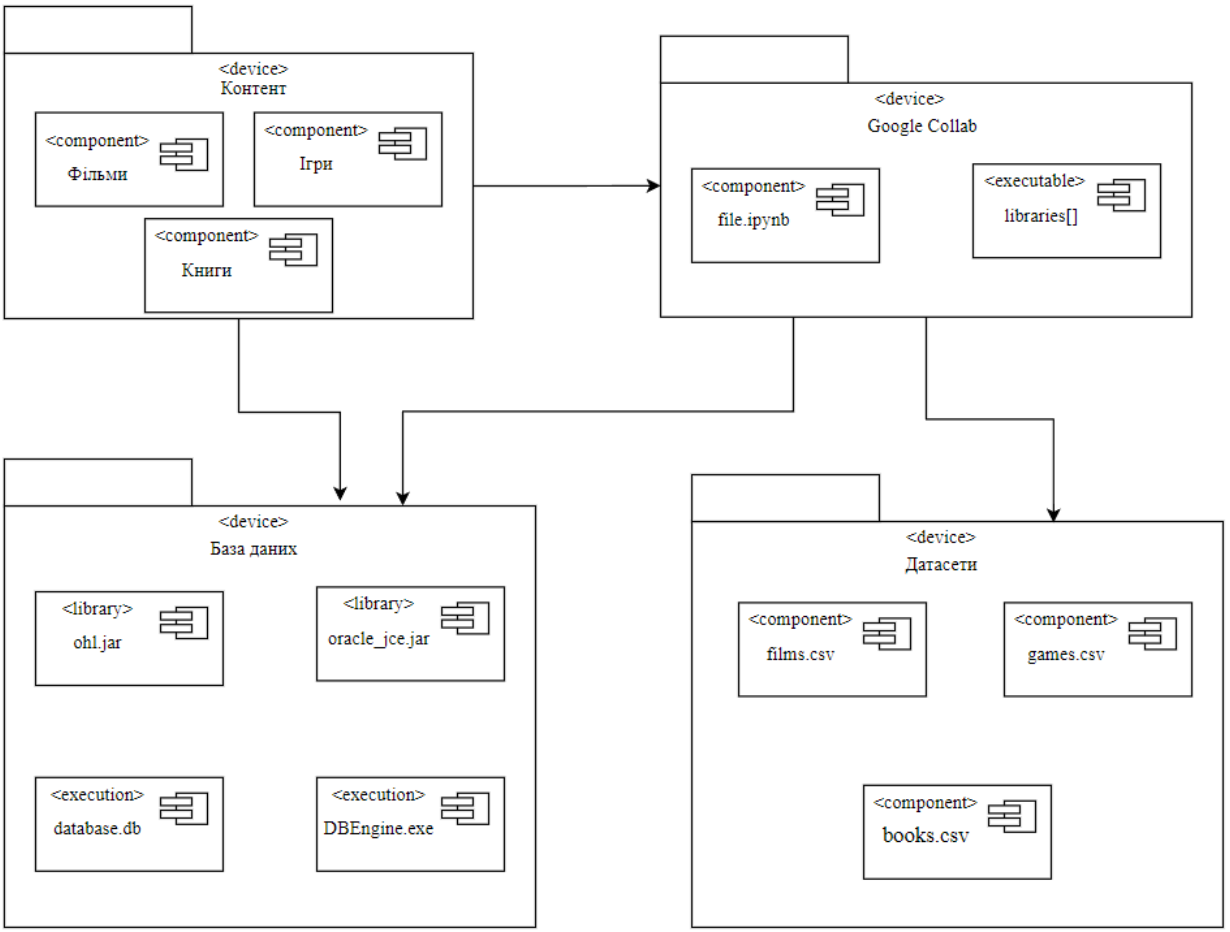


Рисунок 2.7 - Діаграма компонентів

Порівняння різних методів реалізації РС для різного типу контенту відбувається на основі наступних метрик: Точність (наскільки точно метод рекомендує об'єкти які користувачеві ймовірно сподобаються), ефективність (наскільки ефективно метод може генерувати рекомендації), масштабованість (наскільки добре метод може масштабуватися до великих наборів даних).

Тобто, як і зображено на рисунку 2.8, система приймає на вхід декілька датасетів, приймає до них основні методи реалізації РС, та рекомендує об’єкт.

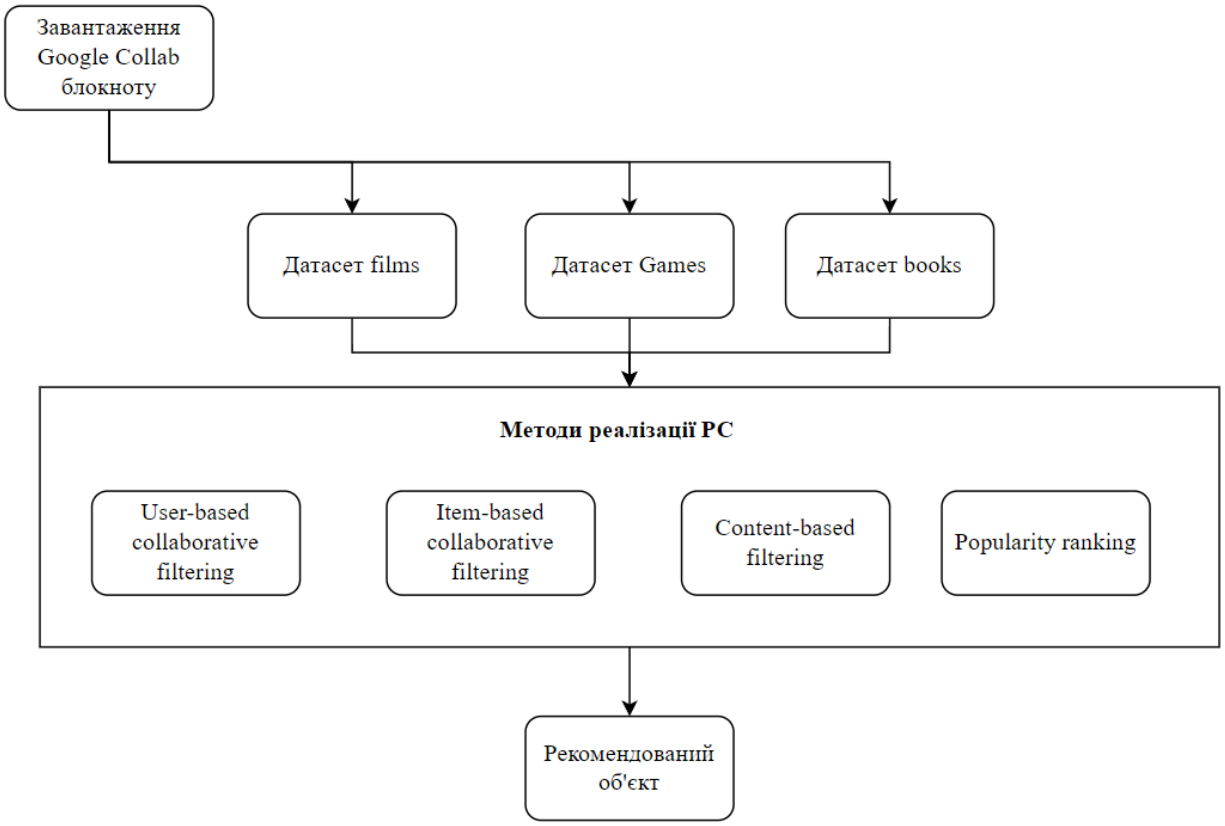


Рисунок 2.8 - Загальна схема рекомендації об’єктів

Більш детально розглянемо процес генерації рекомендацій на основі методу User-based. На вхід система отримує ідентифікатори користувачів, ідентифікатори об'єктів (наприклад, фільми, книги, товари), самі об’єкти та іцінки, відгуки або інші форми взаємодії користувачів з об'єктами (наприклад, оцінки фільмів, перегляди сторінок товарів).

На виході система видає список рекомендацій та рейтинг об’єктів (рейтинг користувачів для кожного рекомендованого об'єкта)

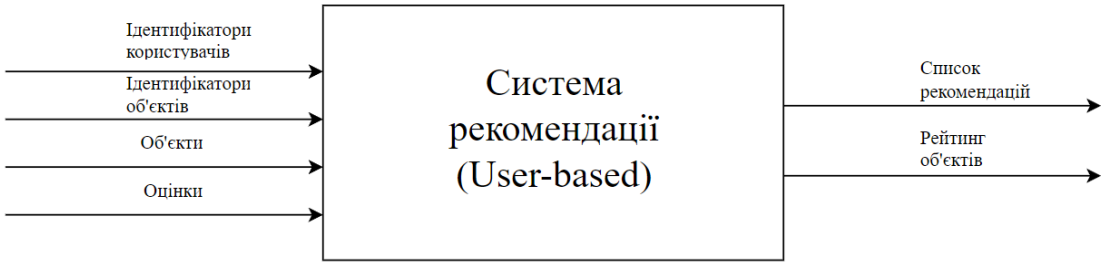


Рисунок 2.9 - Контекстна діаграма

На рисунку 2.10 зображена більш детальна функціональна діаграма.

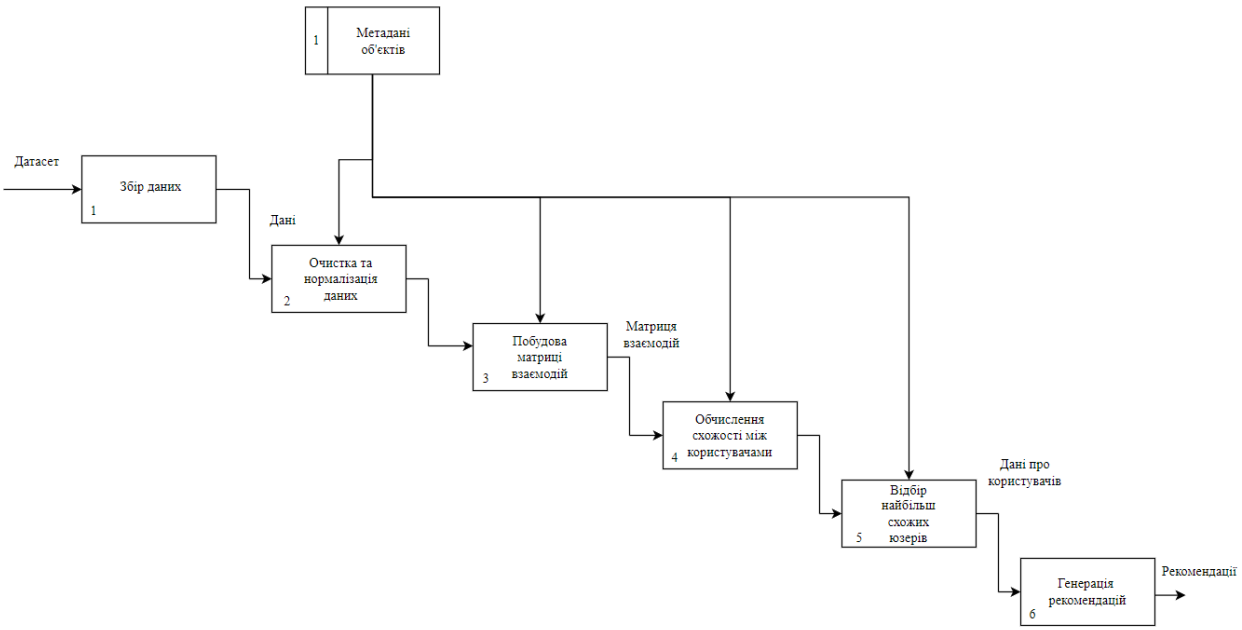


Рисунок 2.10 - Функціональна діаграма

Система на вхід отримує датасет (наприклад список фільмів, книг, будь яких об’єктів). Після цього дані очищуються та нормалізуються. Після чого система створює матрицю взаємодій, де рядки відповідають користувачам, а стовпці - об'єктам. Кожен елемент матриці представляє взаємодію користувача з об'єктом (наприклад, оцінка фільму користувачем).

Обчислюється схожість між користувачами, відбираються найбільш схожі юзери та після цього рекомендації генеруються. Також присутній блок с БД (базою даних) метаданих об’єктів (для фільмів це наприклад може бути жанр, актори, час, назва і т.д).

Більш детально процес побудови матриці взаємодії зображено на рисунку 2.11.

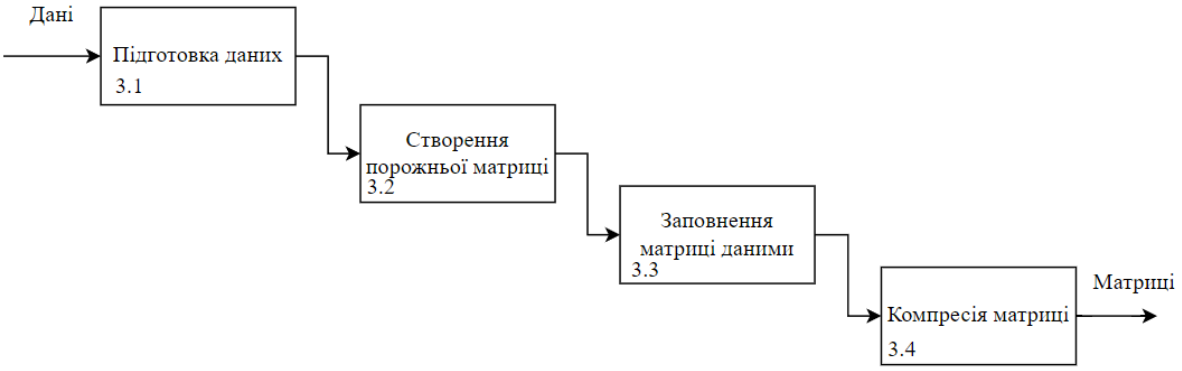


Рисунок 2.11 - Деталізована блок-схема побудови матриці взаємодії

Дані очищаються від помилок та дублікатів, форматуються у відповідний тип. Після цього ініціалізується матриця з нулів. Потім отримуються індекси та дані про взаємодію (наприклад оцінки), дані записуються у відповідні елементи матриці та потім компресуються.

2.4 Опис середи розробки

Google Colab - сервіс, створений Google, який надає можливість працювати з кодом мовою Python через Jupyter Notebook, не встановлюючи на свій комп'ютер додаткових програм. У Google Colab можна застосовувати різні бібліотеки на Python, завантажувати та запускати файли, аналізувати дані та отримувати результати в браузері.

Colab базується на хмарному обчисленні та надає доступ до великого обсягу обчислювальних ресурсів, включаючи графічні процесори (GPU) та тензорні процесори (TPU), для виконання складних обчислень та навчання моделей машинного навчання.

Інтерфейс Google Colab зображено на рисунку 2.12.

Рисунок 2.12 - Інтерфейс середи Google Colab

Де:

1 – головне меню (робота з файлами, налаштування);

2 – кнопка для під’єднання до середи розробки;

3 – поле для вводу кода та виводу результатів компіляції;

4 – кнопка початку компіляції;

5 – бокове меню.

Меню яке необхідне для завантаження файлів та даних має наступний вигляд та зображене на рисунку 2.10:

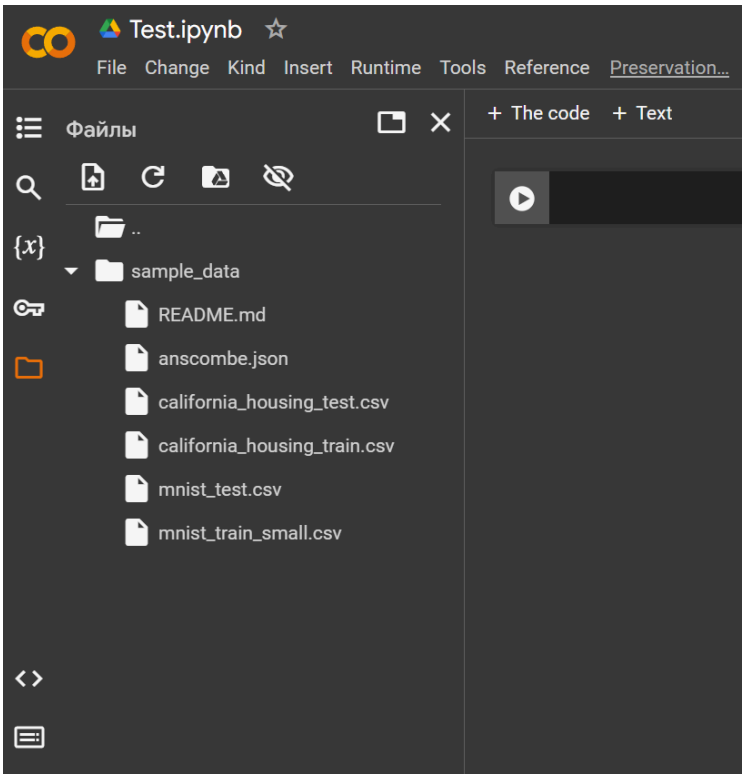


Рисунок 2.13 - Бокове меню Google Colab з доданими файлами

Google Colab підтримує велику кількість бібліотек, досить швидко працює та безкоштовно надає весь необхідний інструментарій для реалізації мети кваліфікаційної роботи.

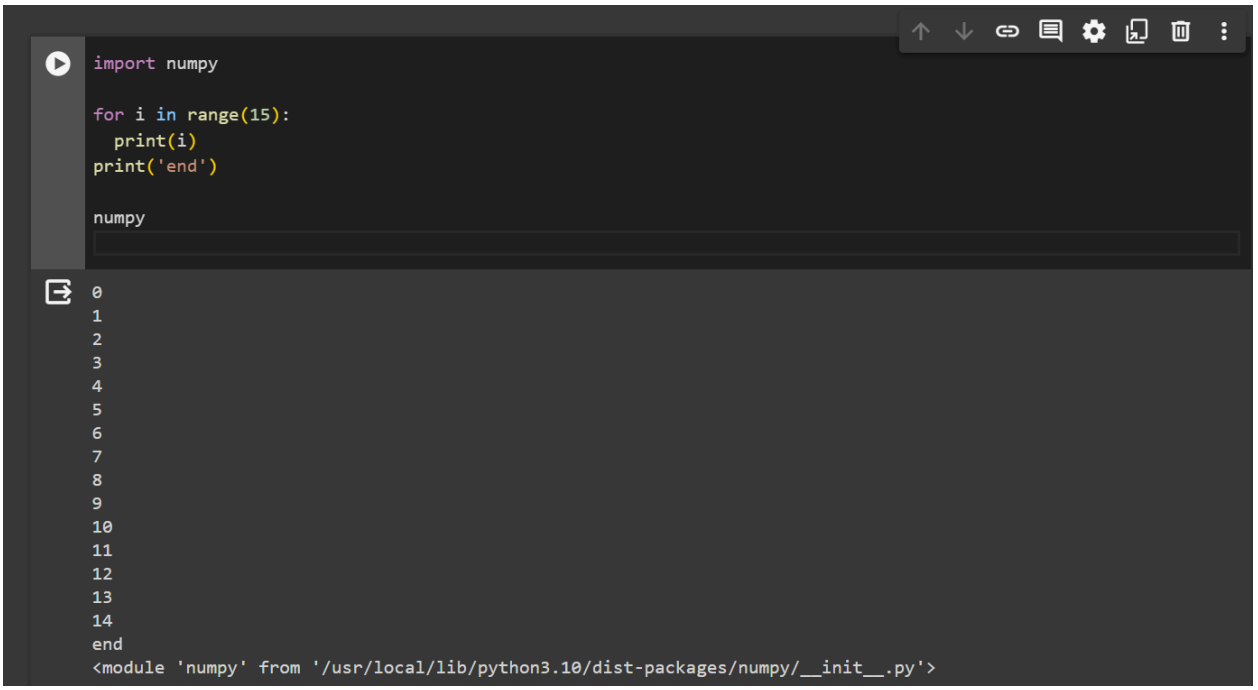


Рисунок 2.14 - Приклад імпортування бібліотеки та компіляції коду в Google Colab

Основні плюси та мінуси використання платформи Google Colab описано в таблиці 2.2.

Таблиця 2.2 – Основні плюси та мінуси використання Google Colab.

Плюси	Мінуси
Безкоштовність	Обмежені ресурси (особливо для великих обсягів даних)
Легка доступність через веб-браузер	Залежність від Інтернет-з'єднання
Можливість використання графічних процесорів та тензорних процесорів	Обмежений час виконання безкоштовних сеансів
Інтеграція з Google Drive	Обмежена підтримка середовища (наприклад, встановлення певних бібліотек)
Можливість спільного доступу та співпраці	Відсутність підтримки для довгострокового зберігання даних

Популярність та велика спільнота користувачів	Ризик приватності даних через використання хмарних сервісів Google
Підтримка відомих бібліотек для машинного навчання та обробки даних	Відсутність повного контролю над середовищем виконання
Можливість використання наочних засобів (наприклад, візуалізація)	Відсутність можливості використання власних обчислювальних ресурсів

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз обраної середи програмування

Для розробки програмного забезпевчення було обрано мову програмування Python та дві популярні середи програмування PyCharm та платформу Google Colab.

Python - популярний вибір для створення систем рекомендацій завдяки своїй простоті, універсальності та великим бібліотекам. Однак, як і будь-який інструмент, Python має свої переваги та недоліки при використанні для реалізації РС:

Плюси:

1) мова Python відома своєю читабельністю та простотою, що робить її доступною. Ця простота використання є перевагою при розробці та підтримці рекомендаційних систем, оскільки вона скорочує час навчання та дозволяє швидко створювати прототипи;

2) python може похвалитися великою екосистемою бібліотек та фреймворків, спеціально розроблених для задач машинного навчання та науки про дані. Такі популярні бібліотеки, як scikit-learn, TensorFlow та PyTorch, надають надійні функціональні можливості для побудови та навчання моделей рекомендацій;

3) гнучкість Python дозволяє реалізовувати різні типи рекомендаційних систем, включаючи спільну фільтрацію, фільтрацію на основі контенту та гібридні підходи. Ця універсальність дозволяє налаштовувати систему відповідно до конкретних сценаріїв використання та характеристик даних.

Мінуси:

1) хоча Python є чудовим за продуктивністю для розробників, він не завжди може запропонувати найкращу продуктивність порівняно з мовами

нижчого рівня, такими як C++ або Java, особливо для завдань з інтенсивними обчисленнями. Це обмеження можна до певної міри пом'якшити, використовуючи оптимізовані бібліотеки та методи, але це залишається проблемою для великомасштабних систем рекомендацій або систем, що працюють у реальному часі;

2) глобальне блокування інтерпретатора (GIL) у Python може створювати проблеми для масштабування багатопотокових додатків, обмежуючи паралелізм у певних сценаріях. Хоча такі рішення, як багатопроцесорність та асинхронне програмування, можуть допомогти вирішити цю проблему, масштабування рекомендаційних систем на основі Python для роботи з великими обсягами користувачів і даних може вимагати ретельного проектування та оптимізації архітектури;

3) Накладні витрати під час виконання та використання пам'яті Python можуть бути вищими порівняно з більш легкими мовами, особливо у середовищах з обмеженими ресурсами.

PyCharm - це інтегроване середовище розробки (IDE) для програмування на Python. Забезпечує аналіз коду, графічний відладчик, інтегрований тестер модулів, інтеграцію з системами контролю версій, а також підтримує веб-розробку за допомогою Django. PyCharm розробляється чеською компанією JetBrains.

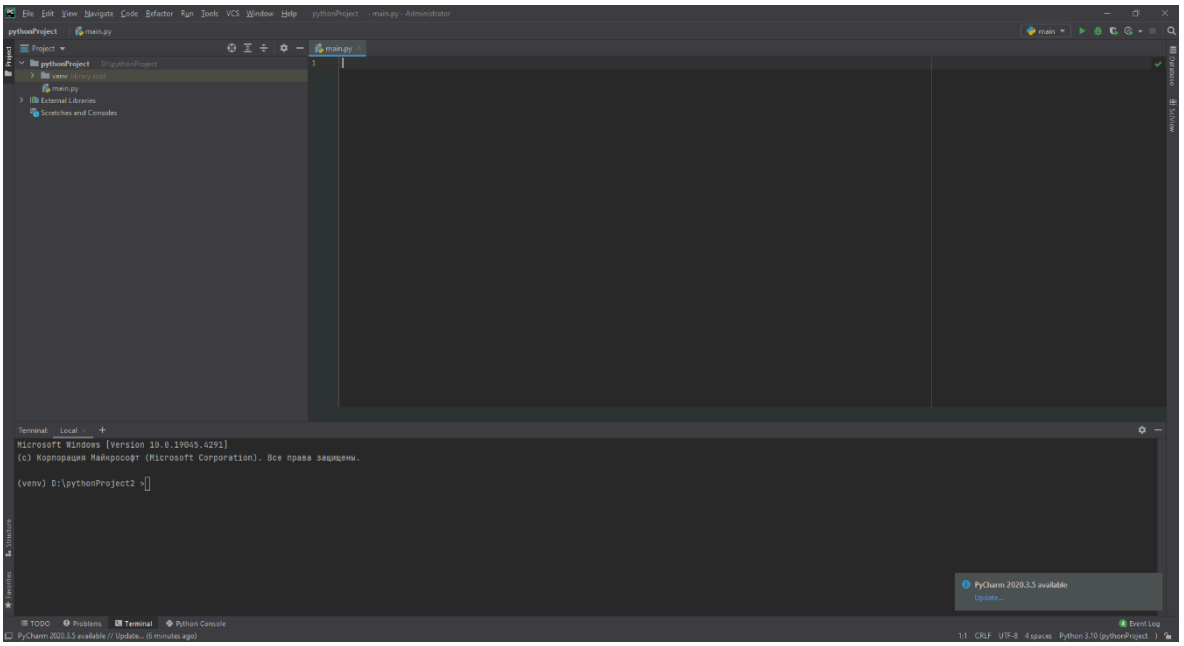


Рисунок 3.1 - Інтерфейс PyCharm

Google Colab має кілька переваг, коли йдеться про створення рекомендаційних систем. По-перше, він надає безкоштовний доступ до ресурсів GPU і TPU, що може значно прискорити процес навчання та експериментів, особливо для завдань з інтенсивними обчисленнями. Такий доступ до потужного обладнання може підвищити продуктивність моделей-рекомендантів і дозволити швидше проводити ітерації.

По-друге, Colab легко інтегрується з Python і популярними бібліотеками машинного навчання, такими як TensorFlow і PyTorch, що дозволяє використовувати наявні знання і кодову базу без додаткових налаштувань.

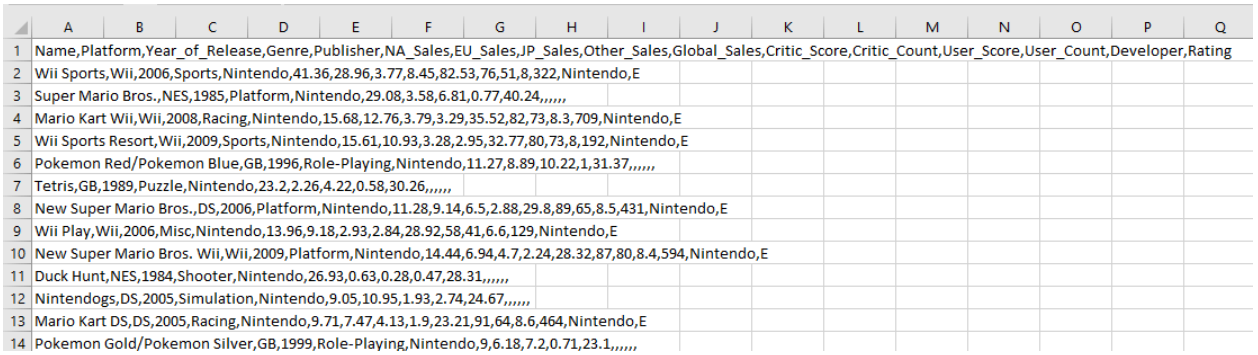
Знання екосистеми Python спрощує розробку та розгортання рекомендаційних систем. Крім того, хмарна інфраструктура Colab усуває потребу в локальних апаратних ресурсах, роблячи її доступною з будь-якого місця, де є підключення до Інтернету.

Більш детально інтерфейс Google Colab я описував в пункті «2.4 Опис середи розробки».

3.2 Розробка бази даних

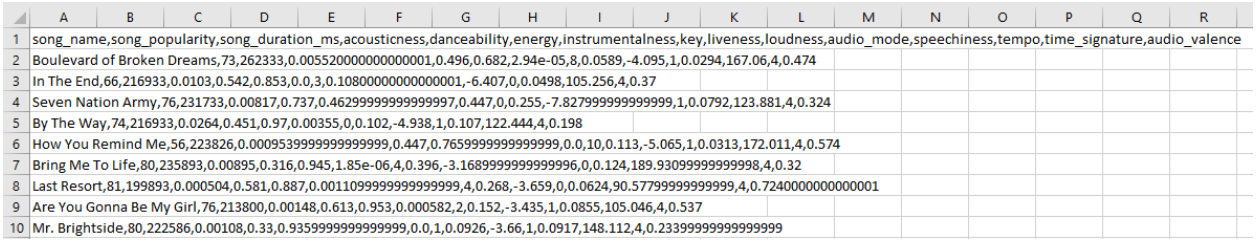
У роботі я використовував три CSV файли як бази даних для аналізу різноманітних категорій - музики, ігор та книг. Кожен з цих файлів містив відповідні дані, такі як інформація про виконавців та альбоми у випадку музики, назви гри та їх розробників для ігор, назви книг та їх авторів для літератури і т.д.

Я організував ці дані у вигляді таблиць з рядками та стовпцями, де кожен рядок представляв окремий запис про конкретного артиста, гру або книгу, а стовпці відображали різні атрибути цих записів, такі як назва, жанр, рік випуску тощо. Така структура дозволила зручно та ефективно здійснювати аналіз та обробку цих даних.



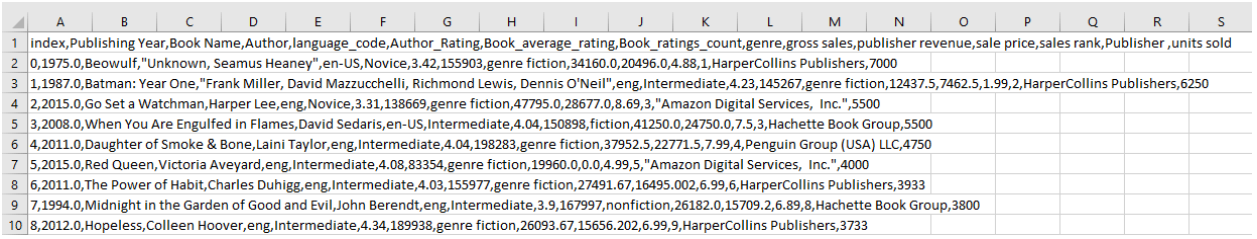
	A
1	Name,Platform,Year_of_Release,Genre,Publisher,NA_Sales,EU_Sales,JP_Sales,Other_Sales,Global_Sales,Critic_Score,Critic_Count,User_Score,User_Count,Developer,Rating
2	Wii Sports,Wii,2006,Sports,Nintendo,41.36,28.96,3.77,8.45,82.53,76,51,8,322,Nintendo,E
3	Super Mario Bros.,NES,1985,Platform,Nintendo,29.08,3.58,6.81,0.77,40.24,,,,,,
4	Mario Kart Wii,Wii,2008,Racing,Nintendo,15.68,12.76,3.79,3.29,35.52,82,73,8.3,709,Nintendo,E
5	Wii Sports Resort,Wii,2009,Sports,Nintendo,15.61,10.93,3.28,2.95,32.77,80,73,8,192,Nintendo,E
6	Pokemon Red/Pokemon Blue,GB,1996,Role-Playing,Nintendo,11.27,8.89,10.22,1,31.37,,,,,,
7	Tetris,GB,1989,Puzzle,Nintendo,23.2,2.26,4.22,0.58,30.26,,,,,,
8	New Super Mario Bros.,DS,2006,Platform,Nintendo,11.28,9.14,6.5,2.88,29.8,89,65,8.5,431,Nintendo,E
9	Wii Play,Wii,2006,Misc,Nintendo,13.96,9.18,2.93,2.84,28.92,58,41,6.6,129,Nintendo,E
10	New Super Mario Bros. Wii,Wii,2009,Platform,Nintendo,14.44,6.94,4.7,2.24,28.32,87,80,8.4,594,Nintendo,E
11	Duck Hunt,NES,1984,Shooter,Nintendo,26.93,0.63,0.28,0.47,28.31,,,,,,
12	Nintendogs,DS,2005,Simulation,Nintendo,9.05,10.95,1.93,2.74,24.67,,,,,,
13	Mario Kart DS,DS,2005,Racing,Nintendo,9.71,7.47,4.13,1.9,23.21,91,64,8.6,464,Nintendo,E
14	Pokemon Gold/Pokemon Silver,GB,1999,Role-Playing,Nintendo,9,6.18,7.2,0.71,23.1,,,,,,

Рисунок 3.2 - Датасет Games.csv



	A
1	song_name,song_popularity,song_duration_ms,acousticness,danceability,energy,instrumentalness,key,liveness,loudness,audio_mode,speechiness,tempo,time_signature,audio_valence
2	Boulevard of Broken Dreams,73,262333,0.005520000000000001,0.496,0.682,2.94e-05,8,0.0589,-4.095,1,0.0294,167.06,4,0.474
3	In The End,66,216933,0.0103,0.542,0.853,0.0,3,0.10800000000000001,-6.407,0,0.0498,105.256,4,0.37
4	Seven Nation Army,76,231733,0.00817,0.737,0.46299999999999997,0.447,0,0.255,-7.827999999999999,1,0.0792,123.881,4,0.324
5	By The Way,74,216933,0.0264,0.451,0.97,0.00355,0,0.102,-4.938,1,0.107,122.444,4,0.198
6	How You Remind Me,56,223826,0.0009539999999999999,0.447,0.7659999999999999,0.0,10,0.113,-5.065,1,0.0313,172.011,4,0.574
7	Bring Me To Life,80,235893,0.00895,0.316,0.945,1.85e-06,4,0.396,-3.1689999999999996,0,0.124,189.93099999999998,4,0.32
8	Last Resort,81,199893,0.000504,0.581,0.887,0.0011099999999999999,4,0.268,-3.659,0,0.0624,90.57799999999999,4,0.7240000000000001
9	Are You Gonna Be My Girl,76,213800,0.00148,0.613,0.953,0.000582,2,0.152,-3.435,1,0.0855,105.046,4,0.537
10	Mr. Brightside,80,222586,0.00108,0.33,0.9359999999999999,0.0,1,0.0926,-3.66,1,0.0917,148.112,4,0.23399999999999999

Рисунок 3.3 - Датасет Music.csv



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	index,Publishing Year,Book Name,Author,language_code,Author_Rating,Book_average_rating,Book_ratings_count,genre,gross sales,publisher revenue,sale price,sales rank,Publisher ,units sold																		
2	0,1975.0,Beowulf,"Unknown, Seamus Heaney",en-US,Novice,3.42,155903,genre fiction,34160.0,20496.0,4.88,1,HarperCollins Publishers,7000																		
3	1,1987.0,Batman: Year One,"Frank Miller, David Mazzucchelli, Richmond Lewis, Dennis O'Neil",eng,Intermediate,4.23,145267,genre fiction,12437.5,7462.5,1.99,2,HarperCollins Publishers,6250																		
4	2,2015.0,Go Set a Watchman,Harper Lee,eng,Novice,3.31,138669,genre fiction,47795.0,28677.0,8.69,3,"Amazon Digital Services, Inc.",5500																		
5	3,2008.0,When You Are Engulfed in Flames,David Sedaris,en-US,Intermediate,4.04,150898,fiction,41250.0,24750.0,7.5,3,Hachette Book Group,5500																		
6	4,2011.0,Daughter of Smoke & Bone,Laini Taylor,eng,Intermediate,4.04,198283,genre fiction,37952.5,22771.5,7.99,4,Penguin Group (USA) LLC,4750																		
7	5,2015.0,Red Queen,Victoria Aveyard,eng,Intermediate,4.08,83354,genre fiction,19960.0,0.0,4.99,5,"Amazon Digital Services, Inc.",4000																		
8	6,2011.0,The Power of Habit,Charles Duhigg,eng,Intermediate,4.03,155977,genre fiction,27491.67,16495.002,6.99,6,HarperCollins Publishers,3933																		
9	7,1994.0,Midnight in the Garden of Good and Evil,John Berendt,eng,Intermediate,3.9,167997,nonfiction,26182.0,15709.2,6.89,8,Hachette Book Group,3800																		
10	8,2012.0,Hopeless,Colleen Hoover,eng,Intermediate,4.34,189938,genre fiction,26093.67,15656.202,6.99,9,HarperCollins Publishers,3733																		

Рисунок 3.4 - Датасет Books.csv

Структура кожного датасету та опис кожного стовбця наступні:

Таблиця 3.1 - Books.csv

Стовбець	Опис
index	Унікальний ідентифікатор книги в наборі даних.
Publishing Year	Рік публікації книги.
Book Name	Назва книги.
Author	Автор книги.
language_code	Код мови книги.
Author_Rating	Рейтинг автора.
Book_average_rating	Середній рейтинг книги.
Book_ratings_count	Кількість рейтингів для книги.
genre	Жанр книги.
gross sales	Валовий дохід від продажу книги.
publisher revenue	Дохід від продажу книги для видавництва.
sale price	Ціна продажу книги.
sales rank	Рейтинг продажів книги.
Publisher	Видавництво книги.
units sold	Кількість проданих одиниць книги.

Таблиця 3.2 - Music.csv

Стовбець	Опис

song_name	Назва пісні.
song_popularity	Популярність пісні.
song_duration_ms	Тривалість пісні у мілісекундах.
acousticness	Міра акустичності пісні.
danceability	Міра танцювальності пісні.

Продовження таблиці 3.2

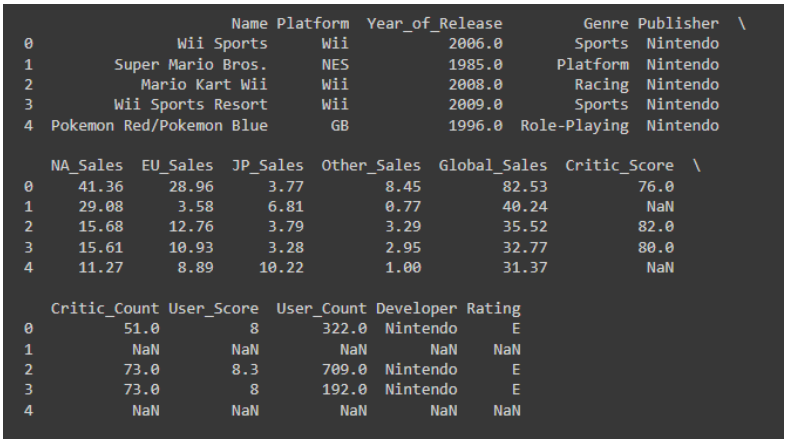
energy	Енергійність пісні.
instrumentalness	Міра інструментальності пісні.
key	Тональність пісні.
liveness	Живість під час запису.
loudness	Гучність пісні.
audio_mode	Аудіо-режим пісні.
speechiness	Співучість пісні.
tempo	Темп пісні.
time_signature	Музичний метр пісні.

Таблиця 3.3 - Games.csv

Стовбець	Опис
Name	Назва гри.
Platform	Платформа, на якій виходила гра.
Year_of_Release	Рік випуску гри.
Genre	Жанр гри.
Publisher	Видавець гри.
NA_Sales	Продажі в Північній Америці.
EU_Sales	Продажі в Європі.
JP_Sales	Продажі в Японії.
Other_Sales	Інші світові продажі.
Global_Sales	Загальні світові продажі.
Critic_Score	Рейтинг критиків.
Critic_Count	Кількість оцінок критиків.
User_Score	Рейтинг користувачів.
User_Count	Кількість оцінок користувачів.
Developer	Розробник гри.

Rating	Рейтинг гри.

Приклад виводу даних з цих датасетів має наступний вигляд:

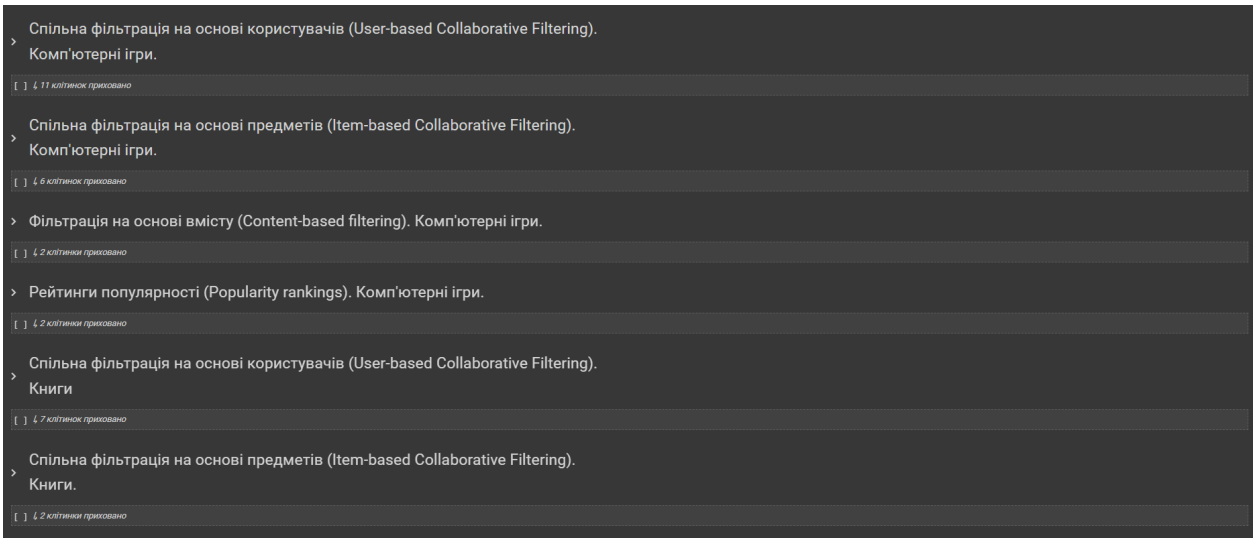


```
                    Name Platform  Year_of_Release         Genre Publisher  \
0             Wii Sports      Wii           2006.0        Sports  Nintendo
1      Super Mario Bros.      NES           1985.0      Platform  Nintendo
2         Mario Kart Wii      Wii           2008.0        Racing  Nintendo
3      Wii Sports Resort      Wii           2009.0        Sports  Nintendo
4  Pokemon Red/Pokemon Blue       GB           1996.0  Role-Playing  Nintendo

   NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales  Critic_Score  \
0     41.36     28.96      3.77         8.45         82.53          76.0
1     29.08      3.58      6.81         0.77         40.24           NaN
2     15.68     12.76      3.79         3.29         35.52          82.0
3     15.61     10.93      3.28         2.95         32.77          80.0
4     11.27      8.89     10.22         1.00         31.37           NaN

   Critic_Count User_Score  User_Count Developer Rating
0          51.0          8       322.0  Nintendo      E
1           NaN        NaN         NaN       NaN    NaN
2          73.0        8.3       709.0  Nintendo      E
3          73.0          8       192.0  Nintendo      E
4           NaN        NaN         NaN       NaN    NaN
```

Рисунок 3.5 - Приклад виводу даних з датасету Games.csv

3.3 Програмна реалізація основних функцій

Програмна реалізація розбита на 12 підпунктів. Для кожного типу даних реалізовано 3 основні методи рекомендаційних систем.



```
› Спільна фільтрація на основі користувачів (User-based Collaborative Filtering).
  Комп'ютерні ігри.
[ ] ↳ 11 клітинок приховано

› Спільна фільтрація на основі предметів (Item-based Collaborative Filtering).
  Комп'ютерні ігри.
[ ] ↳ 6 клітинок приховано

› Фільтрація на основі вмісту (Content-based filtering). Комп'ютерні ігри.
[ ] ↳ 2 клітинки приховано

› Рейтинги популярності (Popularity rankings). Комп'ютерні ігри.
[ ] ↳ 2 клітинки приховано

› Спільна фільтрація на основі користувачів (User-based Collaborative Filtering).
  Книги
[ ] ↳ 7 клітинок приховано

› Спільна фільтрація на основі предметів (Item-based Collaborative Filtering).
  Книги.
[ ] ↳ 2 клітинки приховано
```

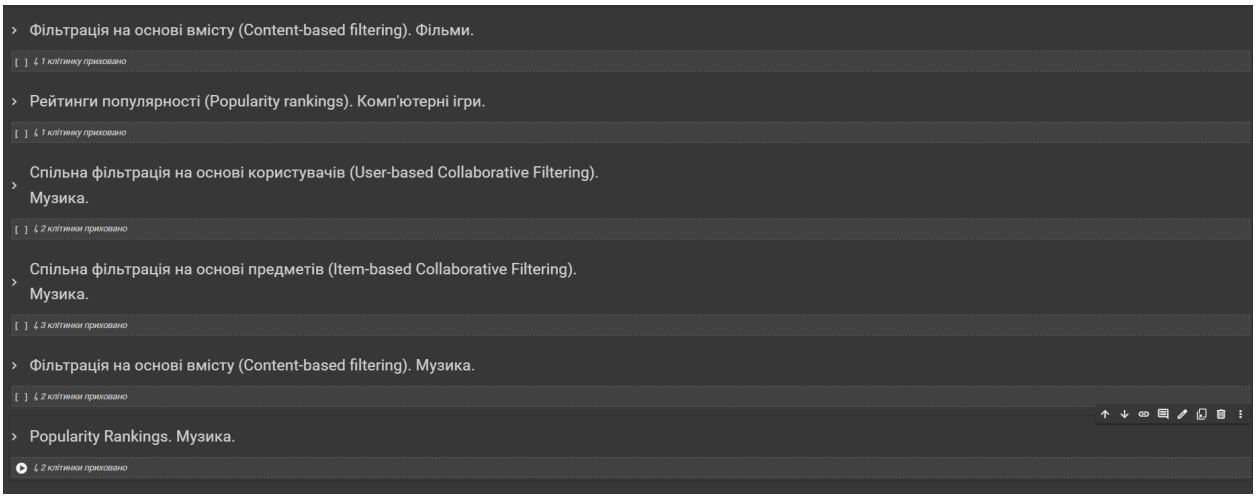


Рисунок 3.6 - Програмна реалізація розбита на 12 заголовків (підпунктів)

В кожному пункті виконується перевірка та обробка пропущених значень у наборі даних. Далі йдуть кроки нормалізації даних за допомогою MinMaxScaler, візуалізація та аналіз даних за допомогою гістограми та теплової карти кореляції.

```
# Перевірка пропущених значень
missing_values = df.isnull().sum()
print("Пропущені значення:\n", missing_values)

# Якщо є пропущені значення, ми можемо їх обробити, наприклад, заповнити середнім значенням
# В даному випадку, для спрощення, просто видалимо рядки з пропущеними значеннями
df = df.dropna()

# Перевірка типів даних
print("\nТипи даних:\n", df.dtypes)
```

```
Пропущені значення:
 Name                2
Platform             0
Year_of_Release    269
Genre                2
Publisher           54
NA_Sales             0
EU_Sales             0
JP_Sales             0
Other_Sales          0
Global_Sales         0
Critic_Score      8582
Critic_Count      8582
User_Score        6704
User_Count        9129
Developer         6623
Rating            6769
dtype: int64
```

Рисунок 3.7 - Виведення та обробка пропущених значень

Усі дані нормалізуються:

```
from sklearn.preprocessing import MinMaxScaler

# Вибираємо оцінки користувачів для нормалізації
ratings = df[['Critic_Score', 'User_Score']]

# Ініціалізуємо MinMaxScaler
scaler = MinMaxScaler()

# Нормалізуємо дані
normalized_ratings = scaler.fit_transform(ratings)

# Замінюємо оригінальні оцінки нормалізованими значеннями
df['Critic_Score'] = normalized_ratings[:, 0]
df['User_Score'] = normalized_ratings[:, 1]

# Виводимо перші декілька рядків для перевірки
print(df.head())
```

```
                 Name Platform  Year_of_Release     Genre Publisher  \
0          Wii Sports      Wii           2006.0    Sports  Nintendo
2      Mario Kart Wii      Wii           2008.0    Racing  Nintendo
3   Wii Sports Resort      Wii           2009.0    Sports  Nintendo
6  New Super Mario Bros.       DS           2006.0  Platform  Nintendo
7            Wii Play      Wii           2006.0      Misc  Nintendo

   NA_Sales  EU_Sales  JP_Sales  Other_Sales  Global_Sales  Critic_Score  \
0     41.36     28.96      3.77         8.45         82.53      0.741176
2     15.68     12.76      3.79         3.29         35.52      0.811765
3     15.61     10.93      3.28         2.95         32.77      0.788235
6     11.28      9.14      6.50         2.88         29.80      0.894118
7     13.96      9.18      2.93         2.84         28.92      0.529412
```

Рисунок 3.8 - Нормалізація даних за домогою MinMaxScaler

Для того щоб більш детально зрозуміти структуру даних які використовуються для реалізації рекомендаційної системи, дані потрібно візуалізувати. Реалізовую я це за допомогою бібліотеки matplotlib.

Для прикладу, проаналізуємо датасет комп’ютерних ігор.

Спочатку виведемо розподіл оцінок користувачів.

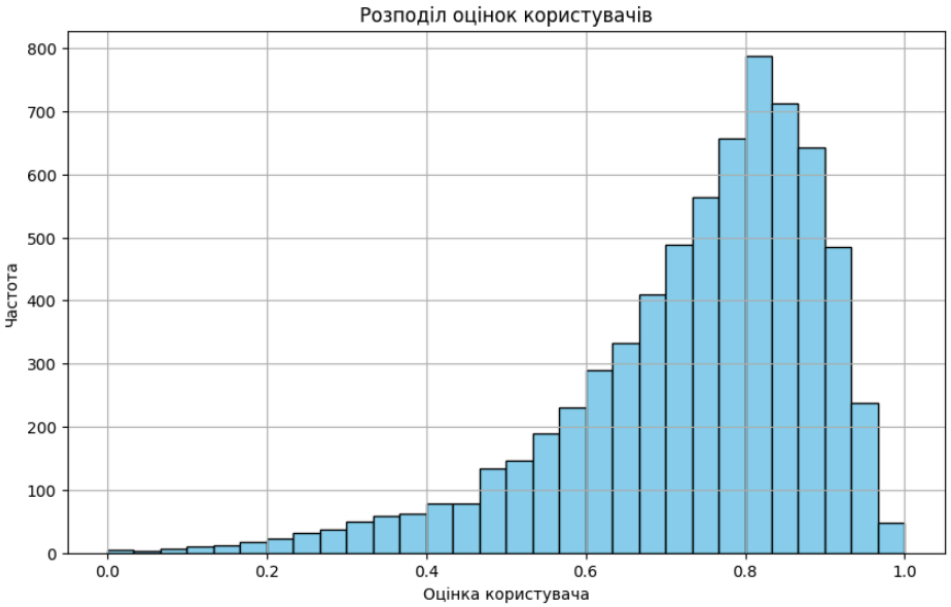


Рисунок 3.9 - Розподіл оцінок користувачів датасету Games.csv

Після чого можна вивести Heatmap[2] для того щоб оцінити кореляцію даних датасету.

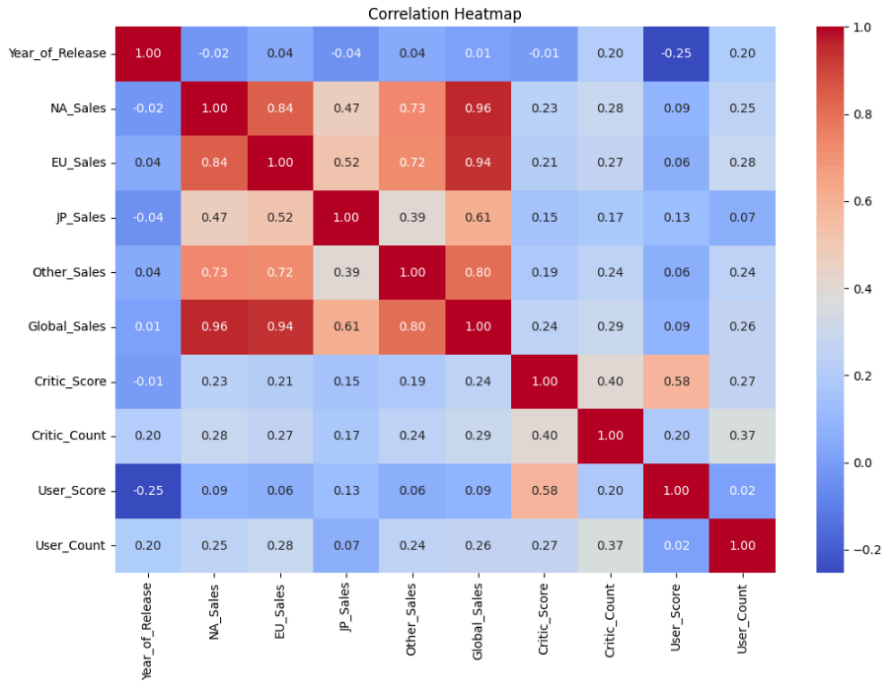


Рисунок 3.10 - Heatmap для розуміння кореляції даних в датасеті Games.csv

[2] Heatmap (або теплові карти) - це візуальне представлення даних, де значення зображуються кольором, що дозволяє легко візуалізувати складні дані та зрозуміти їх з першого погляду.

Після чого можна проаналізувати датасет книг. Для початку виведемо розподіл даних в датасеті.

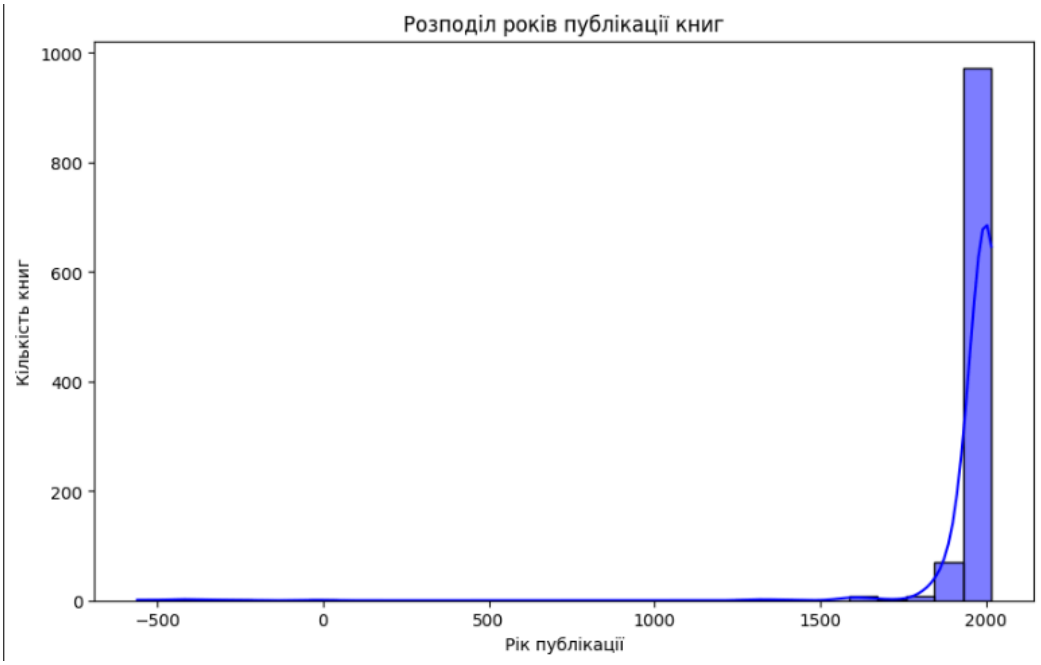


Рисунок 3.11 - Розподіл років публікації книг в датасеті Books.csv

Потім також виведемо розподіл книг за основними жанрами та середній рейтинг книг за роками публікації:

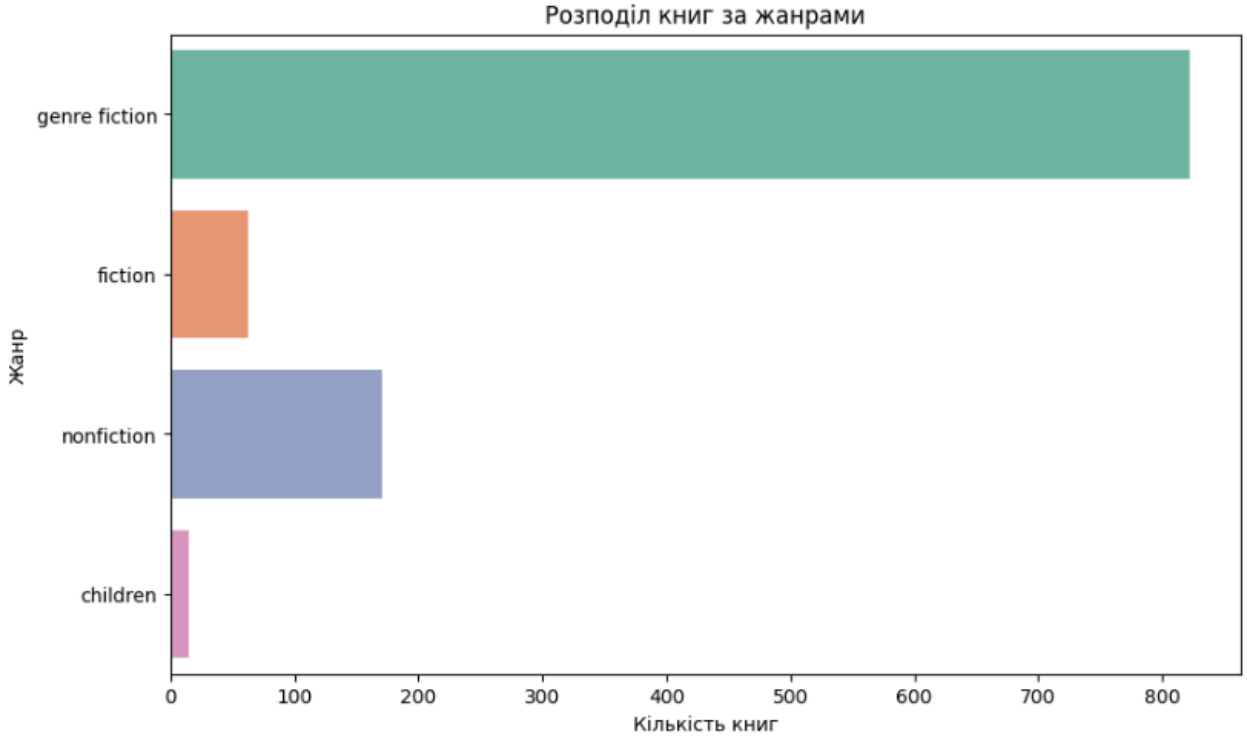


Рисунок 3.12 - Розподіл книг за жанрами в датасеті Books.csv

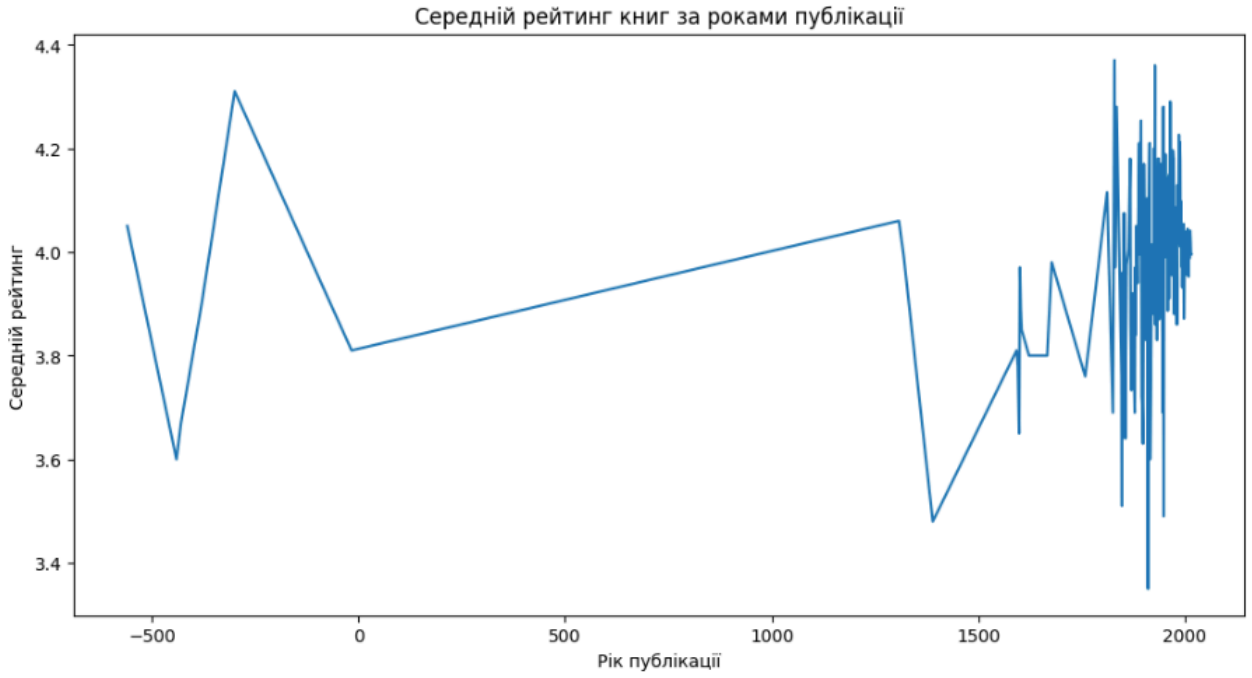


Рисунок 3.13 - Середній рейтинг книг за роками публікації в датасеті Books.csv

І на кінець, щоб мати краще уявлення про дані в датасеті можна вивести взаємозв’язок між середнім рейтингом та кількістю оцінок.

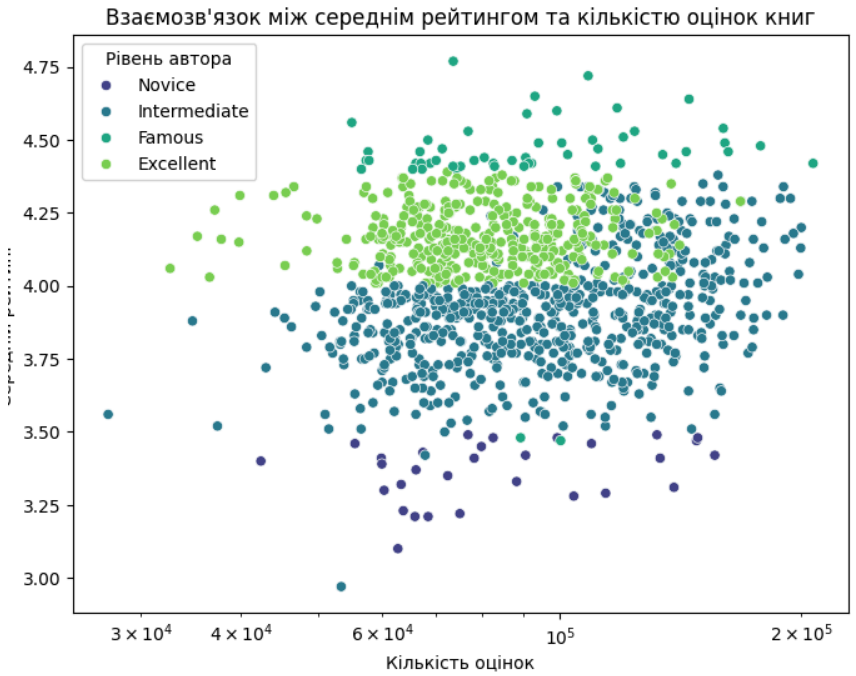


Рисунок 3.14 - Залежність середнього рейтингу від кількості оцінок в датасеті Books.csv

Тепер реалізуємо рекомендаційні системи (функції самих рекомендацій) для вже підготовлених та очищених даних.

Приклад реалізації РС спільної фільтрації на основі користувачів (User-based Collaborative Filtering) для комп’ютерних ігор:

```python
[ ] from sklearn.metrics.pairwise import cosine_similarity

    def user_based_recommendation(user_id, df, num_recommendations=5):
        # Вибираємо дані користувача
        user_data = df[df.index == user_id]

        # Вибираємо оцінки користувача для порівняння
        user_ratings = user_data[['Critic_Score', 'User_Score']]

        # Вибираємо оцінки всіх користувачів, окрім поточного
        other_users = df[df.index != user_id]
        other_users_ratings = other_users[['Critic_Score', 'User_Score']]

        # Обчислюємо схожість користувачів з використанням косинусної схожості
        similarities = cosine_similarity(user_ratings, other_users_ratings)

        # Отримуємо індекси користувачів, які мають найвищу схожість
        similar_users_indices = similarities.argsort()[0][::-1][:num_recommendations]

        # Вибираємо дані схожих користувачів
        similar_users_data = df.iloc[similar_users_indices]

        # Відфільтровуємо ігри, які вже переглянуті або оцінені поточним користувачем
        already_reviewed_games = user_data['Name'].tolist()
        similar_users_data = similar_users_data[~similar_users_data['Name'].isin(already_reviewed_games)]

        # Отримуємо рекомендації для користувача на основі ігор, які сподобалися схожим користувачам
        recommendations = similar_users_data[['Name', 'Critic_Score', 'User_Score']].head(num_recommendations)

        return recommendations

    # Приклад використання: рекомендації для користувача з ID=0
    user_id = 0
    recommendations = user_based_recommendation(user_id, df)
    print("Рекомендації для користувача з ID=", user_id)
    print(recommendations)
```

Рисунок 3.15 - Реалізація рекомендаційної системи для датасету Games.csv (User-base Collaborative Filtering)

На виході, користувач отримує список рекомендацій:

```
Рекомендації для користувача з ID= 0
                                Name  Critic_Score  User_Score
4086           Skylanders: SuperChargers      0.870588    0.560440
2318   Harry Potter: Quidditch World Cup      0.647059    0.670330
11151      The Suffering: Ties That Bind      0.741176    0.945055
2233                Wipeout: In The Zone      0.305882    0.560440
9033                     Class of Heroes      0.564706    0.725275
```

Рисунок 3.16 - Список рекомендованих комп’ютерних ігор для користувача на основі інших користувачів

Результати рекомендацій можна оцінити різними метриками, наприклад:

Таблиця 3.4 - Метрики для оцінки результатів рекомендацій

Метрика	Опис	Плюси	Мінуси
Повнота (Recall)	Співвідношення відфільтрованих рекомендацій до всіх наявних рекомендацій для користувача.	Допомагає визначити, наскільки добре система вміє охопити усі можливі рекомендації для користувача.	Не враховує кількість неправильних рекомендацій.
Точність (Precision)	Співвідношення правильних рекомендацій до всіх запропонованих рекомендацій.	Допомагає визначити, наскільки вірогідно, що рекомендація користувачеві буде відповідати його/її інтересам.	Не враховує кількість пропущених рекомендацій.
F-міра (F1-score)	Середнє гармонічне між точністю і повнотою.	Об'єднує інформацію про точність і повноту в одне значення.	Чутлива до зміни ваги між точністю і повнотою.
MAP (Mean Average Precision)	Середнє значення точності для кожного користувача на всій вибірці.	Враховує рангування рекомендацій, а не просто їх наявність чи відсутність.	Вимагає збереження порядку рекомендацій.

Для різних методів рекомендацій можна використовувати різні метрики оцінки. Але для більшої точності та можливості порівнювати набагато краще використовувати однакові метрики.

Наприклад для оцінки ефективності для User-based ком’ютерних ігор я використовував метод середньої схожості.

```python
def average_similarity(recommendations, df):
    total_similarity = 0
    num_users = len(recommendations)

    for index, row in recommendations.iterrows():
        user_id = index
        user_data = df[df.index == user_id]
        user_ratings = user_data[['Critic_Score', 'User_Score']]

        other_users = df[df.index != user_id]
        other_users_ratings = other_users[['Critic_Score', 'User_Score']]

        similarities = cosine_similarity(user_ratings, other_users_ratings)
        total_similarity += similarities.sum()

    average_similarity = total_similarity / num_users
    return average_similarity

# Приклад використання: оцінка середньої схожості
avg_similarity = average_similarity(recommendations, df)
print("Середня схожість між користувачами:", avg_similarity)
```

```
Середня схожість між користувачами: 6677.7648347201775
```

Рисунок 3.17 - Оцінка точності рекомендацій в датасеті Games.csv

Також використовувалася косинусна схожість (подібність) при реалізації Item-based рекомендаційних системах.

Косинусна подібність вимірює подібність між двома векторами внутрішнього простору. Він вимірюється косинусом кута між двома векторами і визначає, чи спрямовані два вектори приблизно в одному напрямку.

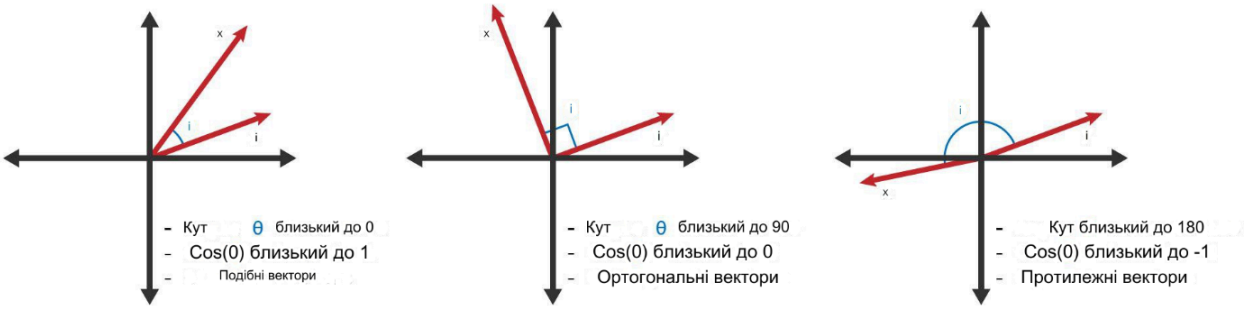


Рисунок 3.18 - Косинусна схожість

```
[ ]  import numpy as np

     def evaluate_recommendations(recommendations, game_name, similarity_matrix):
         # Знаходимо індекс гри у даних
         game_index = data[data['Name'] == game_name].index[0]

         # Отримуємо схожість гри з іншими
         game_similarity = similarity_matrix[game_index]

         # Вибираємо топ-N схожих ігор (не включаючи саму гру)
         recommended_games_indices = recommendations.index

         # Обчислюємо схожість між рекомендованими іграми та грою, для якої робимо рекомендації
         similarities = [game_similarity[index] for index in recommended_games_indices]

         # Середня косинусна схожість
         mean_similarity = np.mean(similarities)

         return mean_similarity

     # Оцінюємо ефективність рекомендацій для гри 'Super Mario Bros.'
     mean_similarity = evaluate_recommendations(recommendations, 'Super Mario Bros.', similarity_matrix)
     print("Середня косинусна схожість:", mean_similarity)

     Середня косинусна схожість: 0.9986405331003013
```

Рисунок 3.19 - Оцінка ефективності та точності рекомендацій на основі косинусної схожості для датасету Games.csv

Подібним способом було реалізовано нормалізацію даних, її аналіз та створення рекомендаційної системи для усіх типів даних та усіх реалізуємих методів РС в даній кваліфікаційній роботи.

3.4 Методика роботи користувача

Методика роботи користувача та взаємодії залежить від методу реалізації РС.

Наприклад при User-based системах необхідно створити користувача з списком йово вподобань, та на основі цього будувати рекомендації.

```python
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

# Завантаження даних
data = pd.read_csv("Music.csv")

# Розрахунок користувацької подібності
user_similarity = cosine_similarity(data.drop(columns=['song_name']), data.drop(columns=['song_name']))

# Вибір сусідів
def get_neighbors(user_id, k=5):
    user_similarity_series = pd.Series(user_similarity[user_id])
    user_similarity_series = user_similarity_series.drop(user_id)  # Видаляємо самого себе
    return user_similarity_series.nlargest(k).index

# Функція для рекомендації пісень
def recommend_songs(user_id, k=5):
    neighbors = get_neighbors(user_id)
    user_songs = data[data.index == user_id]
    neighbor_songs = data[data.index.isin(neighbors)]
    # Відбір пісень, які оцінили сусіди, але не оцінив користувач
    recommended_songs = neighbor_songs[~neighbor_songs.index.isin(user_songs.index)].sample(k)
    return recommended_songs

# Приклад використання
user_id = 0  # Припустимо, що це ідентифікатор користувача, якому ми робимо рекомендації
recommendations = recommend_songs(user_id)
print(recommendations)
```

```
                            song_name  song_popularity  song_duration_ms  \
15193               Focus - Netsky Remix               62            219428
434              Lola - Coca Cola Version               67            241040
5582                Focus - Netsky Remix               62            219428
11597  My Boy (feat. J. Cole) - Freestyle               72            269657
10689          Boulevard of Broken Dreams               71            261266
```

Рисунок 3.20 - На основі вподобань інших користувачів порівнюється схожість з користувачем якому ми рекомендуємо пісні

В Item-based підході у нас порівнюються самі об'єкти в датасеті. Тому рекомендації будуються на основі об’єкту. Наприклад якщо користувач хочу отримати ігри схожі на Super Mario Bros., він вводить назву гри та отримує список рекомендацій.

```python
from sklearn.metrics.pairwise import cosine_similarity
data = pd.read_csv("Games.csv")
# Функція для обчислення схожості між іграми на основі їх характеристик
def calculate_similarity(data):
    # Вибираємо характеристики для обчислення схожості (зараз беремо всі числові характеристики)
    features = ['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Count']

    # Заповнюємо пропущені значення нулями (або можна використовувати інші методи заповнення)
    data_filled = data[features].fillna(0)

    # Обчислюємо схожість між іграми за допомогою косинусної схожості
    similarity_matrix = cosine_similarity(data_filled)

    return similarity_matrix

# Функція для отримання рекомендацій для конкретної гри
def get_recommendations(game_name, data, similarity_matrix, n=5):
    # Шукаємо індекс гри у даних
    game_index = data[data['Name'] == game_name].index[0]

    # Отримуємо схожість гри з іншими
    game_similarity = similarity_matrix[game_index]

    # Сортуємо індекси за зменшенням схожості
    similar_games_indices = game_similarity.argsort()[::-1]

    # Вибираємо топ-N схожих ігор (не включаючи саму гру)
    top_similar_games_indices = similar_games_indices[1:n+1]

    # Отримуємо імена та інші дані про топ-N схожих ігор
    top_similar_games = data.iloc[top_similar_games_indices]

    return top_similar_games

# Реалізація рекомендаційної системи
similarity_matrix = calculate_similarity(data)

# Отримання рекомендацій для конкретної гри (наприклад, 'Super Mario Bros.')
recommendations = get_recommendations('Super Mario Bros.', data, similarity_matrix)
print(recommendations[['Name', 'Platform', 'Genre', 'Publisher']])
```

```
                                          Name Platform  \
5                                       Tetris       GB
755                 Mario & Luigi: Superstar Saga      GBA
299                                   Star Fox 64      N64
5930                       N3: Ninety-Nine Nights     X360
1390  Teenage Mutant Ninja Turtles III: The Manhatta...      NES
```

Рисунок 3.21 - Приклад роботи Item-based системи рекомендацій

3.5 Порівняння різних методів реалізації рекомендаційних систем для різних даних

1) User-Based Collaborative Filtering (UBCF):

1) музика. Цей метод досить ефективний для музичних даних, оскільки схожість між користувачами щодо їхніх музичних вподобань досить важлива для рекомендацій;

2) ігри. UBCF також добре працює і для ігрових даних, де враховується схожість у стилях гри, вподобаннях щодо жанрів та інші параметри;

3) книги. У цьому випадку UBCF є менш ефективний метод, оскільки він виходить з припущення, що подібність між користувачами є ключовою, а не подібність між об'єктами. Як показують дослідження саме порівняння предметів у книгах є ключовим фактором для хороших рекомендацій.

2) Item-Based Collaborative Filtering (IBCF):

1) музика. IBCF досить ефективно працює оскільки він зосереджується на схожості між музичними записами, їх стилем, темпом, жанром і т.д, що дозволяє досить точно реалізовувати рекомендації;

2) ігри. Як показують дослідження IBCF досить непогано працює с іграми але гірше ніш UBCF, адже саме схожість між вподобаннями користувачів дають кращі результати;

3) книги. IBCF може бути корисним, оскільки він враховує схожість між книгами, що може бути важливим для читачів з подібними смаками.

3) Content-Based Filtering:

1) музика. Цей метод може бути ефективним, оскільки він використовує характеристики самого контенту (наприклад, жанр,

виконавець, рік випуску), що може досить детально відображати смак користувача;

2) ігри. Content-Based Filtering також може працювати добре, зосереджуючись на особливостях ігрового процесу, жанрів, графіки тощо;

3) книги: Content-Based Filtering досить посередньо працює з рекомендаціями книг. Набагато краще використовувати User-based метод.

4) Popularity Rankings:

1) музика. PR може бути корисним для новачків або користувачів, які шукають популярні треки або виконавців. Але для людей з більш унікальними смаками в музиці не підходить;

2) ігри. Для ігор цей метод також може бути ефективним, оскільки популярні ігри часто мають широкий загальний інтерес;

3) книги. Для книг популярність може бути добрим показником, але не завжди відображає індивідуальні вподобання користувача. Зараз досить не часто можна зустріти книги які цікавлять більшість, у людей більш індивідуальні смаки в цьому.

Тому можна сказати що вибір методу реалізації рекомендаційної системи залежить від типу контенту, іноді досить значно.

Наприклад, для музики, де схожість між користувачами важлива, можна використовувати User-Based Collaborative Filtering, а для книг, де характеристики контенту є більш важливими - Content-Based Filtering.

У випадку ігор де важлива схожість між продуктами, може бути ефективним Item-Based Collaborative Filtering. Кожен метод має свої переваги та обмеження, і їх вибір важливо здійснювати з урахуванням конкретних властивостей даних та потреб аудиторії.

ВИСНОВКИ

На підставі проведеного порівняльного аналізу різних методів та підходів до визначення переваг користувачів у контексті рекомендаційних систем можна зробити висновок, що ефективність таких систем залежить від використаної методології та алгоритмів.

Враховуючи різноманітність критеріїв, на яких можуть ґрунтуватися рекомендації, важливо обирати підходи, які забезпечують персоналізовані та релевантні рекомендації для користувачів. Такий підхід допомагає підвищити залученість користувачів, зміцнити довіру та покращити загальний рівень задоволеності клієнтів.

На додаток до порівняльного аналізу різних методів та підходів до визначення переваг користувачів у контексті рекомендаційних систем, в роботі були реалізовані основні методи рекомендаційних систем на мові Python. Це дозволило детальніше дослідити та порівняти різні алгоритми, використовуючи реальні дані та реалістичні умови.

Реалізація методів у Python дозволила використовувати широкий спектр бібліотек для машинного навчання та обробки даних, що сприяло якісному аналізу та порівнянню результатів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Quadrana, M., Cremonesi, P., & Jannach, D. (2018). Sequence-Aware Recommender Systems. *ACM Computing Surveys (CSUR)*, 51, 1 - 36. https://doi.org/10.1145/3190616.

2. Deldjoo, Y., Schedl, M., Cremonesi, P., & Pasi, G. (2020). Recommender Systems Leveraging Multimedia Content. *ACM Computing Surveys (CSUR)*, 53, 1 - 38. https://doi.org/10.1145/3407190.

3. Hahsler, M. (2022). recommenderlab: An R Framework for Developing and Testing Recommendation Algorithms. *ArXiv*, abs/2205.12371. https://doi.org/10.48550/arXiv.2205.12371.

4. Tr, M., & V, V. (2021). Recommendation Systems: The Different Filtering Techniques, Challenges and Review Ways to Measure the Recommender System. *Social Science Research Network*. https://doi.org/10.2139/SSRN.3826124.

5. Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., & He, Q. (2020). A Survey on Knowledge Graph-Based Recommender Systems. *IEEE Transactions on Knowledge and Data Engineering*, 34, 3549-3568. https://doi.org/10.1360/ssi-2019-0274.

6. Almonte, L., Guerra, E., Cantador, I., & Lara, J. (2021). Recommender systems in model-driven engineering. *Software and Systems Modeling*, 21, 249-280. https://doi.org/10.1007/s10270-021-00905-x.

7. Ngaffo, A., Ayeb, W., & Choukair, Z. (2020). A Bayesian Inference Based Hybrid Recommender System. *IEEE Access*, 8, 101682-101701. https://doi.org/10.1109/ACCESS.2020.2998824.

8. Mauro, N., Hu, Z., & Ardissono, L. (2022). Service-aware Personalized Item Recommendation. *IEEE Access*, PP, 1-1. https://doi.org/10.1109/ACCESS.2022.3157442.

9. Shukla, N., Soni, N., Gupta, N., & Anand, N. (2022). Online Book Recommendation System using Custom Recommender. *2022 6th International Conference on Trends in Electronics and Informatics (ICOEI)*, 921-925. https://doi.org/10.1109/ICOEI53556.2022.9777131.

10. Zhang, Q., Lu, J., & Zhang, G. (2021). Recommender Systems in E-learning. *Journal of Smart Environments and Green Computing*. https://doi.org/10.20517/JSEGC.2020.06.

11. Oubalahcen, H., & Ouadghiri, M. (2023). Recommender Systems for Social Networks: A Short Review. *Proceedings of the 6th International Conference on Networking, Intelligent Systems & Security*. https://doi.org/10.1145/3607720.3607725.

Додаток А – Код програми

```
# -*- coding: utf-8 -*-

#Спільна фільтрація на основі користувачів (User-based Collaborative
Filtering). Комп'ютерні ігри.

Завантажимо та перевіримо дані.
"""

import pandas as pd

df = pd.read_csv('Games.csv')

print(df.head())

missing_values = df.isnull().sum()
print("Пропущені значення:\n", missing_values)

df = df.dropna()

print("\nТипи даних:\n", df.dtypes)

"""Нормалізуємо дані"""

from sklearn.preprocessing import MinMaxScaler

ratings = df[['Critic_Score', 'User_Score']]

scaler = MinMaxScaler()

normalized_ratings = scaler.fit_transform(ratings)

df['Critic_Score'] = normalized_ratings[:, 0]
df['User_Score'] = normalized_ratings[:, 1]

print(df.head())

"""Проаналізуємо та візуалізуємо дані"""

import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.hist(df['User_Score'], bins=30, color='skyblue', edgecolor='black')
plt.title('Розподіл оцінок користувачів')
plt.xlabel('Оцінка користувача')
plt.ylabel('Частота')
plt.grid(True)
plt.show()

import seaborn as sns

numeric_df = df.select_dtypes(include=['float64', 'int64'])

corr_matrix = numeric_df.corr()

plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()

"""Створимо рекомендаційну систему"""

from sklearn.metrics.pairwise import cosine_similarity
```

```
def user_based_recommendation(user_id, df, num_recommendations=5):
    user_data = df[df.index == user_id]

    user_ratings = user_data[['Critic_Score', 'User_Score']]

    other_users = df[df.index != user_id]
    other_users_ratings = other_users[['Critic_Score', 'User_Score']]

    similarities = cosine_similarity(user_ratings, other_users_ratings)

    similar_users_indices =
similarities.argsort()[0][::-1][:num_recommendations]
    similar_users_data = df.iloc[similar_users_indices]

    already_reviewed_games = user_data['Name'].tolist()
    similar_users_data =
similar_users_data[~similar_users_data['Name'].isin(already_reviewed_games)]

    recommendations = similar_users_data[['Name', 'Critic_Score',
'User_Score']].head(num_recommendations)

    return recommendations

user_id = 0
recommendations = user_based_recommendation(user_id, df)
print("Рекомендації для користувача з ID=", user_id)
print(recommendations)

def average_similarity(recommendations, df):
    total_similarity = 0
    num_users = len(recommendations)

    for index, row in recommendations.iterrows():
        user_id = index
        user_data = df[df.index == user_id]
        user_ratings = user_data[['Critic_Score', 'User_Score']]

        other_users = df[df.index != user_id]
        other_users_ratings = other_users[['Critic_Score', 'User_Score']]

        similarities = cosine_similarity(user_ratings, other_users_ratings)
        total_similarity += similarities.sum()

    average_similarity = total_similarity / num_users
    return average_similarity

avg_similarity = average_similarity(recommendations, df)
print("Середня схожість між користувачами:", avg_similarity)

"""#Спільна фільтрація на основі предметів (Item-based Collaborative
Filtering). Комп'ютерні ігри.

Завантажимо та нормалізуємо дані
"""

!pip install pandas scikit-learn

import pandas as pd

df = pd.read_csv("Games.csv")
```

```
print("Кількість пропущених значень у кожному стовбці:")
print(df.isnull().sum())

df.dropna(subset=['Critic_Score', 'User_Score'], inplace=True)

df['Year_of_Release'].fillna(df['Year_of_Release'].mean(), inplace=True)

selected_columns = ['Name', 'Platform', 'Genre', 'Critic_Score', 'User_Score']
df_selected = df[selected_columns]

print("\nПерші 5 записів після підготовки даних:")
print(df_selected.head())

"""Створимо рекомендаційну систему."""

from sklearn.metrics.pairwise import cosine_similarity
data = pd.read_csv("Games.csv")
def calculate_similarity(data):
    features = ['NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales',
'Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Count']

    data_filled = data[features].fillna(0)

    similarity_matrix = cosine_similarity(data_filled)

    return similarity_matrix

def get_recommendations(game_name, data, similarity_matrix, n=5):
    game_index = data[data['Name'] == game_name].index[0]

    game_similarity = similarity_matrix[game_index]

    similar_games_indices = game_similarity.argsort()[::-1]

    top_similar_games_indices = similar_games_indices[1:n+1]

    top_similar_games = data.iloc[top_similar_games_indices]

    return top_similar_games

similarity_matrix = calculate_similarity(data)

recommendations = get_recommendations('Super Mario Bros.', data,
similarity_matrix)
print(recommendations[['Name', 'Platform', 'Genre', 'Publisher']])

import numpy as np

def evaluate_recommendations(recommendations, game_name, similarity_matrix):
    game_index = data[data['Name'] == game_name].index[0]

    game_similarity = similarity_matrix[game_index]

    recommended_games_indices = recommendations.index

    similarities = [game_similarity[index] for index in
recommended_games_indices]

    mean_similarity = np.mean(similarities)

    return mean_similarity
```

```
mean_similarity = evaluate_recommendations(recommendations, 'Super Mario
Bros.', similarity_matrix)
print("Середня косинусна схожість:", mean_similarity)

"""#Фільтрація на основі вмісту (Content-based filtering). Комп'ютерні
ігри."""

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

features = ['Name', 'Genre', 'Publisher', 'Critic_Score', 'User_Score']

data[features] = data[features].fillna('')

def combine_features(row):
    return row['Name'] + ' ' + row['Genre'] + ' ' + row['Publisher'] + ' ' +
str(row['Critic_Score']) + ' ' + str(row['User_Score'])

data['combined_features'] = data.apply(combine_features, axis=1)

print(data.head())

tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(data['combined_features'])

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

print(cosine_sim.shape)

def recommend_games(game_title, cosine_sim=cosine_sim):
    idx = data[data['Name'] == game_title].index[0]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    sim_scores = sim_scores[1:11]

    game_indices = [i[0] for i in sim_scores]

    return data['Name'].iloc[game_indices]

recommendations = recommend_games('Super Mario Bros.')
print(recommendations)

"""# Рейтинги популярності (Popularity rankings). Комп'ютерні ігри."""

import pandas as pd

data = pd.read_csv("Games.csv")

sorted_data = data.sort_values(by='Global_Sales', ascending=False)

top_n = 10
top_games = sorted_data.head(top_n)

print("Топ-{} ігор за популярністю:".format(top_n))
print(top_games[['Name', 'Global_Sales']])


def calculate_mrr(actual, recommended):
```

```
    mrr = 0
    for actual_items, recommended_items in zip(actual, recommended):
        found = False
        for i, item in enumerate(recommended_items, 1):
            if item in actual_items:
                mrr += 1 / i
                found = True
                break
        if not found:
            mrr += 0
    mrr /= len(actual)
    return mrr

actual_games = [['Wii Sports', 'Super Mario Bros.'], ['Mario Kart Wii'], ['Wii
Sports Resort']]
recommended_games = [['Wii Sports', 'Super Mario Bros.', 'Mario Kart Wii'],
['Super Mario Bros.', 'Mario Kart Wii'], ['Wii Sports', 'Mario Kart Wii']]

mrr = calculate_mrr(actual_games, recommended_games)
print("Mean Reciprocal Rank (MRR):", mrr)

"""# Спільна фільтрація на основі користувачів (User-based Collaborative
Filtering). Книги"""

import pandas as pd

df = pd.read_csv('Books.csv')

print(df.head())

data = df[['index', 'Book Name', 'Book_average_rating']]

print(df.head())

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('Books.csv')

print(df.head())

plt.figure(figsize=(10, 6))
sns.histplot(df['Publishing Year'], bins=30, kde=True, color='blue')
plt.title('Розподіл років публікації книг')
plt.xlabel('Рік публікації')
plt.ylabel('Кількість книг')
plt.show()

plt.figure(figsize=(10, 6))
sns.countplot(y='genre', data=df, palette='Set2')
plt.title('Розподіл книг за жанрами')
plt.xlabel('Кількість книг')
plt.ylabel('Жанр')
plt.show()

plt.figure(figsize=(12, 6))
sns.lineplot(x='Publishing Year', y='Book_average_rating', data=df,
estimator='mean', ci=None)
plt.title('Середній рейтинг книг за роками публікації')
plt.xlabel('Рік публікації')
plt.ylabel('Середній рейтинг')
```

```
plt.show()

plt.figure(figsize=(8, 6))
sns.scatterplot(x='Book_ratings_count', y='Book_average_rating', data=df,
hue='Author_Rating', palette='viridis')
plt.title('Взаємозв\'язок між середнім рейтингом та кількістю оцінок книг')
plt.xlabel('Кількість оцінок')
plt.ylabel('Середній рейтинг')
plt.xscale('log')
plt.legend(title='Рівень автора')
plt.show()

import pandas as pd

df = pd.read_csv('Books.csv')

data = df[['index', 'Book Name', 'Book_average_rating']]

user_item_matrix = data.pivot_table(index='index', columns='Book Name',
values='Book_average_rating')

import numpy as np

def pearson_similarity(user1_ratings, user2_ratings):
    common_ratings = {}
    for item in user1_ratings:
        if item in user2_ratings:
            common_ratings[item] = 1

    num_common_ratings = len(common_ratings)

    if num_common_ratings == 0:
        return 0

    mean_user1 = np.mean(list(user1_ratings.values()))
    mean_user2 = np.mean(list(user2_ratings.values()))

    covariance = sum((user1_ratings[item] - mean_user1) * (user2_ratings[item]
- mean_user2)
                     for item in common_ratings)

    std_dev_user1 = np.sqrt(sum((user1_ratings[item] - mean_user1) ** 2 for
item in common_ratings))
    std_dev_user2 = np.sqrt(sum((user2_ratings[item] - mean_user2) ** 2 for
item in common_ratings))

    if std_dev_user1 * std_dev_user2 == 0:
        return 0
    else:
        pearson_corr = covariance / (std_dev_user1 * std_dev_user2)
        return pearson_corr

def get_similar_users(user_id, user_item_matrix):
    user_ratings = user_item_matrix.loc[user_id].dropna()

    similar_users = {}

    for user in user_item_matrix.index:
        if user == user_id:
            continue

        current_user_ratings = user_item_matrix.loc[user].dropna()
```

```
        similarity = pearson_similarity(user_ratings, current_user_ratings)

        similar_users[user] = similarity

    similar_users = sorted(similar_users.items(), key=lambda x: x[1],
reverse=True)

    return similar_users

similar_users = get_similar_users(0, user_item_matrix)

import pandas as pd

user0_data = {
    'Book Name': ['Beowulf', 'Batman: Year One', 'Go Set a Watchman'],
    'Rating': [5, 4, 3]
}

user0_df = pd.DataFrame(user0_data)

print("Тестові дані користувача 0:")
print(user0_df)

rated_books = user0_df['Book Name']

user1_rated_books =
user_item_matrix.loc[1][user_item_matrix.loc[1].notnull()].index

recommended_books = rated_books[~rated_books.isin(user1_rated_books)]

print("Рекомендації для користувача 1 на основі даних користувача 0:")
print(recommended_books.head(5))

def precision(actual, recommended):
    intersection = len(set(actual) & set(recommended))
    return intersection / len(recommended)

def coverage(recommended, all_items):
    return len(set(recommended)) / len(set(all_items))


book_ratings = df.groupby('Book Name')['Book_average_rating'].mean()


recommended_books = book_ratings.sort_values(ascending=False).index

recommended_books_user1 = recommended_books[:5]

print("Рекомендовані книги для користувача 1:")
print(recommended_books_user1)

"""# Спільна фільтрація на основі предметів (Item-based Collaborative
Filtering). Книги."""

import pandas as pd
import numpy as np
from scipy.spatial.distance import cosine

df = pd.read_csv('Books.csv')

print(df.head())
```

```
from sklearn.metrics.pairwise import cosine_similarity

ratings_matrix = df.pivot_table(index='Book Name', columns='index',
values='Book_average_rating')

ratings_matrix = ratings_matrix.fillna(0)

book_similarity = cosine_similarity(ratings_matrix.T)

def recommend_books(book_name, ratings_matrix, book_similarity):
    book_index = ratings_matrix.index.get_loc(book_name)

    similar_books = list(enumerate(book_similarity[book_index]))

    similar_books = sorted(similar_books, key=lambda x: x[1], reverse=True)

    recommended_books = []
    for book in similar_books[1:6]:
        recommended_books.append(ratings_matrix.index[book[0]])
    return recommended_books

book_name = "Beowulf"
recommendations = recommend_books(book_name, ratings_matrix, book_similarity)
print("Рекомендовані книги для користувача, який читав книгу
'{}':".format(book_name))
for i, book in enumerate(recommendations):
    print("{}. {}".format(i+1, book))

"""# Фільтрація на основі вмісту (Content-based filtering). Фільми."""

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

df = pd.read_csv('Books.csv')

features = ['Book Name', 'Author', 'genre', 'Publishing Year',
'Book_average_rating']

df[features] = df[features].fillna('')

def combine_features(row):
    return row['Book Name'] + ' ' + row['Author'] + ' ' + row['genre'] + ' ' +
str(row['Publishing Year']) + ' ' + str(row['Book_average_rating'])

df['combined_features'] = df.apply(combine_features, axis=1)

tfidf = TfidfVectorizer(stop_words='english')

tfidf_matrix = tfidf.fit_transform(df['combined_features'])

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

def get_recommendations(book_name, cosine_sim=cosine_sim):
    idx = df[df['Book Name'] == book_name].index[0]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    sim_scores = sim_scores[1:11]
```

```
    book_indices = [i[0] for i in sim_scores]

    return df['Book Name'].iloc[book_indices]

book_name = 'Batman: Year One'
print("Рекомендовані книги для:", book_name)
print(get_recommendations(book_name))

"""# Рейтинги популярності (Popularity rankings). Комп'ютерні ігри."""

import pandas as pd

df = pd.read_csv('Books.csv')

data = df[['Book Name', 'Book_average_rating', 'Book_ratings_count']]

data['popularity'] = data['Book_average_rating'] * data['Book_ratings_count']

sorted_data = data.sort_values(by='popularity', ascending=False)

top_10_popular_books = sorted_data.head(10)
print(top_10_popular_books)

"""# Спільна фільтрація на основі користувачів (User-based Collaborative
Filtering). Музика."""

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

data = pd.read_csv("Music.csv")

print(data)

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity

data = pd.read_csv("Music.csv")

user_similarity = cosine_similarity(data.drop(columns=['song_name']),
data.drop(columns=['song_name']))

def get_neighbors(user_id, k=5):
    user_similarity_series = pd.Series(user_similarity[user_id])
    user_similarity_series = user_similarity_series.drop(user_id)
    return user_similarity_series.nlargest(k).index

def recommend_songs(user_id, k=5):
    neighbors = get_neighbors(user_id)
    user_songs = data[data.index == user_id]
    neighbor_songs = data[data.index.isin(neighbors)]
    recommended_songs =
neighbor_songs[~neighbor_songs.index.isin(user_songs.index)].sample(k)
    return recommended_songs

user_id = 0
recommendations = recommend_songs(user_id)
print(recommendations)

"""# Спільна фільтрація на основі предметів (Item-based Collaborative
Filtering). Музика."""
```

```
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler

music_data = pd.read_csv('Music.csv')

features = ['song_popularity', 'song_duration_ms', 'acousticness',
'danceability', 'energy', 'instrumentalness',
            'key', 'liveness', 'loudness', 'audio_mode', 'speechiness',
'tempo', 'time_signature', 'audio_valence']

scaler = MinMaxScaler()
music_data_scaled = scaler.fit_transform(music_data[features])

similarity_matrix = cosine_similarity(music_data_scaled)

similarity_df = pd.DataFrame(similarity_matrix, index=music_data['song_name'],
columns=music_data['song_name'])

print(similarity_df.head())

def item_based_recommendation(song_name, similarity_df, k=5):
    song_similarity = similarity_df[song_name]

    similar_songs = song_similarity.sort_values(ascending=False)[1:k+1]

    recommended_songs = list(similar_songs.index)
    return recommended_songs

song_name = "In The End"
recommended_songs = item_based_recommendation(song_name, similarity_df, k=5)
print("Рекомендовані пісні для {}: {}".format(song_name, recommended_songs))

def precision(actual, recommended):
    """
    Функція для обчислення метрики Precision.

    Параметри:
        actual (list): Список пісень, які дійсно сподобалися користувачеві.
        recommended (list): Список рекомендованих пісень.

    Повертає:
        precision (float): Значення Precision.
    """
    intersection = set(actual) & set(recommended)
    precision = len(intersection) / len(recommended) if len(recommended) > 0
else 0
    return precision

def recall(actual, recommended):
    """
    Функція для обчислення метрики Recall.

    Параметри:
        actual (list): Список пісень, які дійсно сподобалися користувачеві.
        recommended (list): Список рекомендованих пісень.

    Повертає:
        recall (float): Значення Recall.
    """
    intersection = set(actual) & set(recommended)
    recall = len(intersection) / len(actual) if len(actual) > 0 else 0
```

```
    return recall

def mean_average_precision(actual_list, recommended_list):
    """
    Функція для обчислення середнього значення метрики Average Precision.

    Параметри:
        actual_list (list): Список списків пісень, які дійсно сподобалися
користувачам.
        recommended_list (list): Список списків рекомендованих пісень для
користувачів.

    Повертає:
        map (float): Середнє значення Average Precision.
    """
    total_precision = 0
    for actual, recommended in zip(actual_list, recommended_list):
        total_precision += precision(actual, recommended)
    map = total_precision / len(actual_list)
    return map

actual_songs = [['In The End', 'Boulevard of Broken Dreams'], ['Seven Nation
Army'], ['By The Way']]
recommended_songs = [['Boulevard of Broken Dreams', 'In The End'], ['By The
Way', 'How You Remind Me'], ['Seven Nation Army']]

map_score = mean_average_precision(actual_songs, recommended_songs)
print("Mean Average Precision:", map_score)

"""# Фільтрація на основі вмісту (Content-based filtering). Музика."""

import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler

data = pd.read_csv("Music.csv")

scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(data.iloc[:, 1:])

user_profile = scaled_data[[0, 1, 2], :]

similarity_matrix = cosine_similarity(user_profile, scaled_data)

similar_songs_indices = similarity_matrix.argsort()[0][::-1][1:]
recommended_songs = data.iloc[similar_songs_indices][:10]

print(recommended_songs[['song_name', 'song_popularity', 'song_duration_ms']])

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

data = pd.read_csv("Music.csv")

features = ['song_name', 'acousticness', 'danceability', 'energy',
'instrumentalness', 'key', 'liveness', 'loudness', 'audio_mode',
'speechiness', 'tempo', 'time_signature', 'audio_valence']

data['combined_features'] = data.apply(lambda x: ' '.join([str(x[feature]) for
feature in features]), axis=1)
```

```
tfidf_vectorizer = TfidfVectorizer()

tfidf_matrix = tfidf_vectorizer.fit_transform(data['combined_features'])

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

def get_recommendations(song_name, cosine_sim=cosine_sim):
    idx = data[data['song_name'] == song_name].index[0]

    sim_scores = list(enumerate(cosine_sim[idx]))

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    sim_scores = sim_scores[1:11]

    song_indices = [i[0] for i in sim_scores]

    return data['song_name'].iloc[song_indices]

recommendations = get_recommendations("In The End")
print(recommendations)

"""# Popularity Rankings. Музика."""

import pandas as pd

data = pd.read_csv('Music.csv')

sorted_data = data.sort_values(by='song_popularity', ascending=False)

def recommend_popular_songs(n=10):
    """
    Рекомендує найпопулярніші пісні користувачеві.

    Параметри:
    - n: Кількість пісень, які потрібно рекомендувати (за замовчуванням 10).
    """
    top_n_songs = sorted_data.head(n)

    return top_n_songs[['song_name', 'song_popularity']]

recommended_songs = recommend_popular_songs(n=10)
print(recommended_songs)

import pandas as pd
from sklearn.metrics import mean_squared_error
from math import sqrt

data = pd.read_csv('Music.csv')

sorted_data = data.sort_values(by='song_popularity', ascending=False)

def recommend_popular_songs(n=10):
    """
    Рекомендує найпопулярніші пісні користувачеві.

    Параметри:
    - n: Кількість пісень, які потрібно рекомендувати (за замовчуванням 10).
    """
    top_n_songs = sorted_data.head(n)

    return top_n_songs[['song_name', 'song_popularity']]
```

```
recommended_songs = recommend_popular_songs(n=10)
print(recommended_songs)

def evaluate_recommender(actual_data, recommended_data):
    """
    Оцінює ефективність рекомендаційної системи за допомогою середньої
квадратичної помилки (RMSE).

    Параметри:
    - actual_data: Дійсні дані (DataFrame).
    - recommended_data: Рекомендовані дані (DataFrame).
    """
    merged_data = pd.merge(actual_data, recommended_data, on='song_name',
how='inner')

    rmse = sqrt(mean_squared_error(merged_data['song_popularity_x'],
merged_data['song_popularity_y']))

    return rmse

actual_data = data.sample(frac=0.2)
rmse = evaluate_recommender(actual_data, recommended_songs)
print("RMSE:", rmse)
```