



МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Институт Информационных Технологий
Кафедра Вычислительной Техники

ПРАКТИЧЕСКАЯ РАБОТА №1

по дисциплине
«Разработка обеспечивающих подсистем систем поддержки
принятия решений»

Студент группы: ИКБО-04-22

Кликушин В.И.
(Ф. И.О. студента)

Преподаватель

Гуличева А.А.
(Ф.И.О. преподавателя)

Москва 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	3
1 ПОСТАНОВКА ЗАДАЧИ	5
2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	6
2.1 Постановка задачи.....	7
2.2 Анамнестические методы.....	7
2.3 Математические меры сходства	8
3 ДОКУМЕНТАЦИЯ К ДАННЫМ.....	11
3.1 Описание предметной области	11
3.2 Анализ данных.....	11
3.3 Предобработка данных	15
4 ПРАКТИЧЕСКАЯ ЧАСТЬ	16
4.1 Функциональные возможности	16
4.2 Рекомендательная система.....	20
ЗАКЛЮЧЕНИЕ	23
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ.....	24
ПРИЛОЖЕНИЯ.....	25

ВВЕДЕНИЕ

Современные информационные системы ежедневно обрабатывают огромное количество данных о пользователях и их взаимодействиях с цифровым контентом. В условиях растущего объёма данных проблема предоставления релевантной информации отдельному пользователю становится одной из ключевых. Пользователи интернет-магазинов, медиасервисов или онлайн-платформ регулярно сталкиваются с ситуацией информационной перегрузки: выбор нужного товара или контента вручную требует значительных временных и когнитивных ресурсов. Рекомендательные системы решают эту задачу, автоматически предлагая пользователям контент, который с наибольшей вероятностью будет им интересен, на основе анализа их собственных предпочтений и поведения других пользователей.

Актуальность данной темы обусловлена широким практическим применением рекомендательных систем в различных областях: от электронной коммерции и стриминговых сервисов до социальных сетей и образовательных платформ. Согласно исследованиям, внедрение эффективных рекомендательных алгоритмов позволяет существенно повысить вовлечённость пользователей и конверсию — до 35% от общего объёма продаж на платформах, таких как Amazon и Netflix, формируется за счёт рекомендаций. Помимо практической значимости, рекомендательные алгоритмы представляют собой интересный объект с точки зрения анализа данных, поскольку предполагают работу с разреженными матрицами, выбор мер сходства и реализацию различных стратегий прогнозирования предпочтений.

В данной работе основное внимание уделяется анамнестическим методам коллаборативной фильтрации, которые предполагают поиск соседей — схожих пользователей (user-based) или схожих объектов (item-based) — и использование их предпочтений для прогнозирования неизвестных оценок.

Для вычисления степени сходства между пользователями и объектами в литературе предлагаются различные меры: коэффициент корреляции Пирсона,

косинусное сходство (Отиаи), коэффициент Жаккара и L_p -нормы. В работе реализованы и сравниваются несколько таких мер, что позволяет эмпирически оценить их влияние на точность рекомендаций. Для оценки качества используются стандартные метрики ошибок предсказания — RMSE и MAE.

Таким образом, целью данной работы является реализация и сравнительный анализ анамнестических методов рекомендательных систем на основе пользовательских и предметных соседств, с использованием нескольких математических мер сходства. Для достижения этой цели в работе выполнены следующие этапы: анализ предметной области и структуры данных, реализация алгоритмов user-based и item-based коллаборативной фильтрации с различными метриками, а также оценка и визуализация результатов с помощью статистических показателей точности.

1 ПОСТАНОВКА ЗАДАЧИ

Цель работы: приобрести навыки реализации анамнестических коллаборативных методов рекомендательных систем, основанных на соседстве пользователей и объектов.

Задачи: создать программную реализацию анамнестического метода рекомендательной системы (РС), основанной либо на соседстве пользователей, либо на соседстве элементов (предметов), включающую актуальную предметную область для применения РС (вроде маркетплейса, медиа ресурсов, соц. сетей, экономической сферы и т. д.), математические меры сходств: расстояние Жаккара, норму Лебега пространства (L_p -норму), коэффициент Отиаи, коэффициент корреляции Пирсона и т.п., необходимо реализовать не менее 3-ёх любых методов (пояснить, почему были выбраны именно эти методы), сравнение вышеописанных методов и, исходя из результатов, выбор наиболее подходящего(-их) метода(-ов) для решения задачи.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Рекомендательная система подбирает и предлагает пользователю, релевантный контент, основываясь на своих знаниях о пользователе, контенте и взаимодействии пользователя и контента. Рекомендательная система стремится подобрать и предложить контент, который пользователь ещё не видел и который наиболее вероятно будет ему интересен, то есть это система прогнозирования предпочтений. Рекомендательные системы представляют собой программный комплекс, который определяет и распознает интересы и предпочтения пользователей, формируя рекомендации в соответствии с ними.

В контексте рекомендательных систем наиболее значимы следующие определения:

1. Прогноз – это предположение, насколько пользователю понравится контент.
2. Релевантность – расположение контента в соответствии с тем, что всего подходит пользователю в данный момент. Релевантность сочетает в себе контекст, демографические данные и (ожидаемые) оценки.
3. Рекомендации – лидеры по релевантности.
4. Персонализация - сочетание релевантности и наглядности.

Рекомендаторы используют данные о контенте, о пользователях и о взаимодействиях пользователей и контента. Такие рекомендации являются персональными, так как основаны на персональных предпочтениях и формируются специально для данного пользователя.

В базовых подходах для рекомендательных систем могут использоваться два вида данных: информация о взаимодействии пользователей с объектами интереса и информация, предоставленная самими пользователями, например, атрибуты, указанные в профиле или релевантные ключевые слова.

Первую группа методов чаще всего называют методами коллаборативной фильтрации, для методов второй группы обычно используется название рекомендаций на основе контента.

2.1 Постановка задачи

Имеются следующие данные:

- множество пользователей (*users*, $u \in U$);
- множество объектов (*items*, $i \in I$);
- множество событий (действия, которые пользователи совершают с объектами) (*events*, $(u, i, r_{ui}) \in D$);

Событие описывается так: пользователь u поставил оценку r_{ui} объекту i .

Требуется:

- предсказать оценку объекту, которого пользователь ещё не видел (Формула 2.1).

$$r_{ui} = \text{Predict}(u, i) \quad (2.1)$$

- вычислить персональные рекомендации для пользователя u (Формула 2.2).

$$u \rightarrow (i_1, \dots, i_k) = \text{Recommend}_K(u) \quad (2.2)$$

2.2 Анамнестические методы

Фильтрация в окрестности может быть реализована двумя методами: user-based (юзер-бейсд – сходства пользователей) и item-based (айтем-бейсд – сходства элементов), они основаны на построении матриц схожести.

В общем задача нахождения схожести может быть определена следующим образом: имеется два элемента i_1 и i_2 ; сходство между ними определяются

функцией $sim(i_1, i_2)$. Возвращаемое этой функцией значение пропорционально степени сходства между элементами. Тогда для идентичных элементов s $sim(i_1, i_2) = 1$, а для элементов, не имеющих ничего общего $sim(i_1, i_2) = 0$. Изменение сходства тесно связано с расчетом различия между элементами. Математически это можно выразить так: Сходство = 1 – Различие.

Целью обоих направлений user-based и item-based является выделение схожих объектов в группы на основе матрицы оценок. В первом случае определяется сходство пользователей: найти других пользователей, чьи прошлые оценки поведения похожи на те, что и у текущего пользователя, и использовать их оценки других элементов для прогнозирования предпочтения текущего пользователя. Второй подход, на основе сходства элементов, в этом случае вместо того, чтобы использовать подобие между поведением пользовательских оценок для прогнозирования предпочтения, используется сходство между оценками моделей элементов. Если два элемента, как правило, имеют одинаковые оценки пользователей, то они похожи, и пользователи должны иметь аналогичные предпочтения для подобных элементов.

Для определения сходства между пользователями или элементами используют различные подходы.

2.3 Математические меры сходства

1. Расстояние Жаккара.

Этот параметр называется коэффициентом сходства Жаккара, который показывает на сколько похожи два набора данных. Коэффициент Жаккара измеряет подобие между конечными множествами выборок, и определяется как размер пересечения, деленного на размере объединения множеств выборок (Формула 2.3).

$$sim(a, b) = \frac{|A \cap B|}{|A \cup B|} \quad (2.3)$$

2. L_1 -норма.

Расстояние L_1 также известно, как расстояние городских кварталов, манхэттенское расстояние, расстояние такси, метрика прямоугольного города — оно измеряет дистанцию не по кратчайшей прямой, а по блокам. Название «манхэттенское расстояние» связано с уличной планировкой Манхэттена. Манхэттенское расстояние вычисляется по Формуле 2.4.

$$\|x\|_1 = \sum_i |x_i| \quad (2.4)$$

3. L_2 -норма.

L_2 -норму иначе называется евклидовым расстоянием. Евклидова метрика (евклидово расстояние) — метрика в евклидовом пространстве — расстояние между двумя точками евклидова пространства, вычисляемое по теореме Пифагора. Для векторов $p = (p_1, p_2, \dots, p_n)$ и $q = (q_1, q_2, \dots, q_n)$ евклидово расстояние определяется по Формуле 2.5.

$$\|x\|_2(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} \quad (2.5)$$

Евклидова метрика — наиболее естественная функция расстояния, возникающая в геометрии, отражающая интуитивные свойства расстояния между точками.

4. Коэффициент Отиаи.

Коэффициент Отиаи (косинусный коэффициент, косинусное подобие) — бинарная мера сходства, предложенная японским биологом Акирой Отиаи. Косинусный коэффициент — мера подобия между двумя массивами данных, вычисляемая как косинус угла между векторами в многомерном пространстве. В самом деле, двух пользователей разумно считать похожими, если угол между их векторами предпочтений мал.

Пусть даны два вектора признаков, A и B , тогда косинусное сходство, $\cos(\theta)$, может быть представлено используя скалярное произведение и норму (Формула 2.6).

$$\text{sim}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \quad (2.6)$$

Косинусный коэффициент изменяется $-1 \leq \cos(\theta) \leq 1$. Если $\cos(\theta) = 1 \angle \theta = 0$, то вкусы пользователей похожи. Если $\cos(\theta) = -1 \angle \theta = 180$, то вкусы пользователей противоположны. Если $\cos(\theta) = 0 \angle \theta = 90$ – зависимость между предпочтениями пользователей не просматривается.

Одна из причин популярности косинусного сходства состоит в том, что оно эффективно в качестве оценочной меры, особенно для разреженных векторов, так как необходимо учитывать только ненулевые измерения.

5. Коэффициент корреляции Пирсона.

Похожесть объектов i и t определяется с помощью корреляции Пирсона по Формуле 2.7.

$$\text{sim}(i, t) = \frac{\sum_{u \in U} (r_{u,i} - \bar{r}_u)(r_{u,t} - \bar{r}_u)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{u \in U} (r_{u,t} - \bar{r}_u)^2}} \quad (2.7)$$

где U – множество пользователей, которые оценили объекты i и t ;

$r_{u,i}$ – оценка, поставленная пользователем u объекту i ;

$r_{u,t}$ – оценка, поставленная пользователем u объекту t ;

\bar{r}_u – средняя оценка пользователя u .

Коэффициент корреляции Пирсона изменяется в интервале от -1 до $+1$; безразмерен, т. е. не имеет единиц измерения; указывает, как близко расположены точки к прямой линии.

3 ДОКУМЕНТАЦИЯ К ДАННЫМ

3.1 Описание предметной области

Данные взяты из открытого сервиса MovieLens, предоставляемого исследовательской группой GroupLens Университета Миннесоты. Датасет ml-latest-small содержит информацию о рейтингах и тегах пользователей для фильмов.

Целью использования этих данных является построение рекомендательной системы фильмов, которая может предсказывать оценки для фильмов, не просмотренных пользователем, и рекомендовать новые фильмы на основе исторических оценок и тегов.

Датасет состоит из следующих сущностей:

- пользователи (User): анонимизированные идентификаторы пользователей, которые выставляли оценки фильмов и добавляли теги;
- фильмы (Movie): идентификаторы, название, жанры, год выпуска и ссылки на внешние источники (IMDb, TMDb);
- рейтинги (Rating): пользователь, фильм, оценка от 0.5 до 5 и временная метка;
- теги (Tag): пользователь, фильм, текст тега и временная метка.

3.2 Анализ данных

Датасет ml-latest-small представляет собой компактную выборку данных MovieLens, содержащую оценки пользователей и информацию о фильмах. Он включает чуть более ста тысяч рейтингов, чуть меньше десяти тысяч фильмов и около шести сотен пользователей, что делает его удобным для экспериментов и обучения рекомендательных систем без необходимости использования мощных вычислительных ресурсов.

Каждая запись о рейтинге состоит из идентификатора пользователя, идентификатора фильма, выставленной оценки и временной метки. Основная информация о данных в файле ratings.csv представлена на Рисунке 3.2.1.

```
><class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      100836 non-null  int64
1   movieId     100836 non-null  int64
2   rating      100836 non-null  float64
3   timestamp   100836 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

Рисунок 3.2.1 - Основная информация о данных в файле ratings.csv

Распределение выставленных оценок представлено на Рисунке 3.2.2.

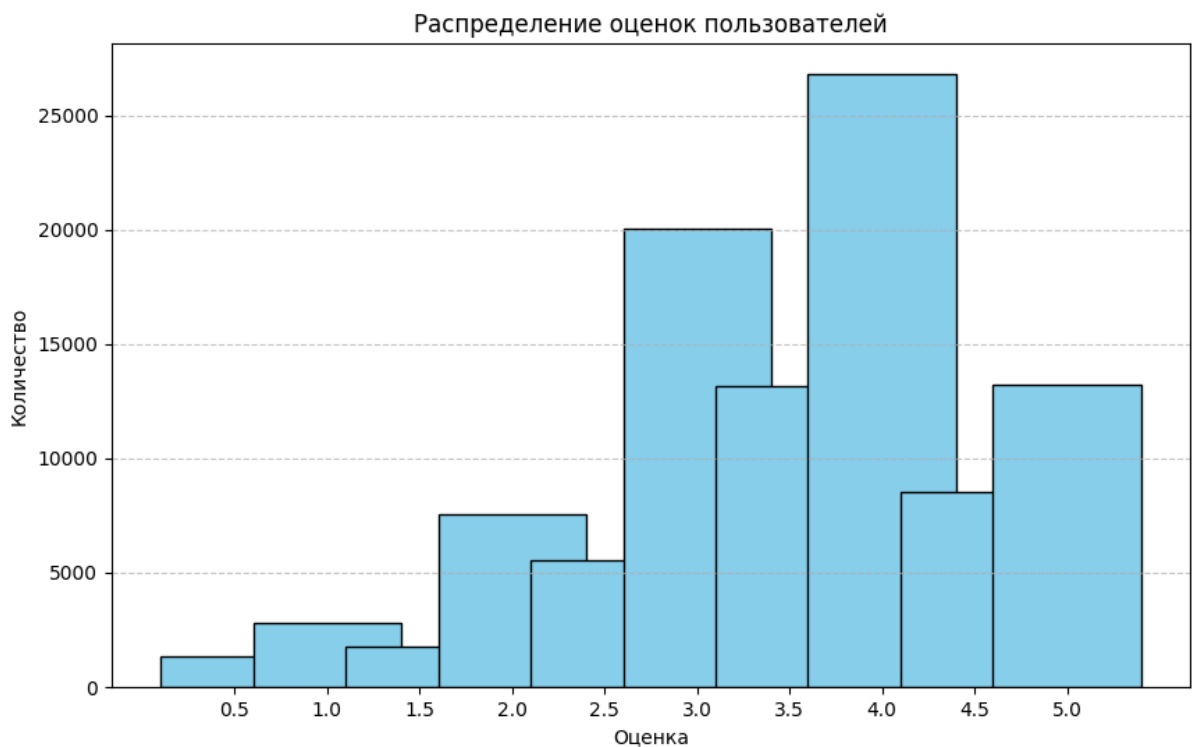
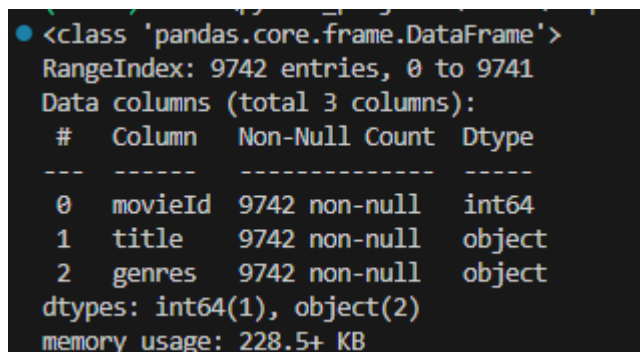


Рисунок 3.2.2 – Распределение оценок

Оценки распределены по шкале от 0.5 до 5 с шагом 0.5, что позволяет достаточно точно моделировать предпочтения пользователей, одновременно упрощая обработку данных. Анализ распределения оценок показывает, что наиболее популярные значения находятся в диапазоне от 3 до 4 баллов, что отражает склонность пользователей давать средние и положительные оценки.

Более экстремальные оценки, например 0.5 или 5, встречаются реже, но именно они могут быть особенно информативны при построении рекомендательных моделей, так как ясно отражают сильные предпочтения или отторжение.

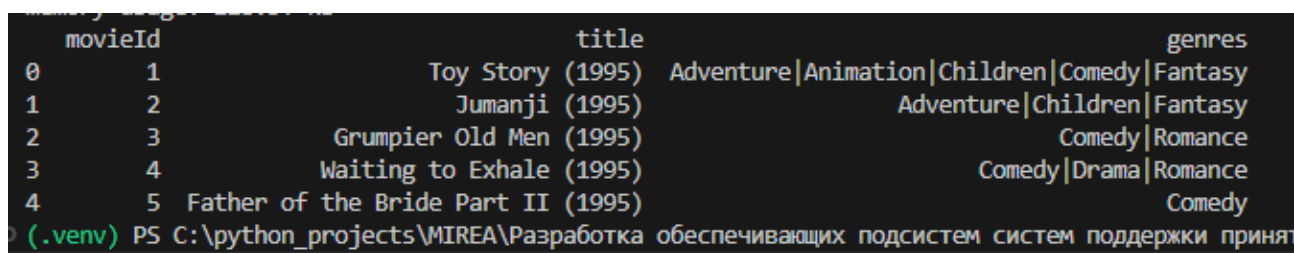
Фильмы в датасете сопровождаются информацией о жанрах. Основная информация о данных в файле movies.csv отображена на Рисунке 3.2.3.



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
#   Column      Non-Null Count  Dtype
---  -
0   movieId     9742 non-null   int64
1   title       9742 non-null   object
2   genres      9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
```

Рисунок 3.2.3 - Основная информация о данных в файле movies.csv

Датафрейм не содержит нулевых значений или пропусков. Первые 5 строк датафрейма представлены на Рисунке 3.2.4.



	movieId	title	genres
0	1	Toy Story (1995)	Adventure Animation Children Comedy Fantasy
1	2	Jumanji (1995)	Adventure Children Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama Romance
4	5	Father of the Bride Part II (1995)	Comedy

Рисунок 3.2.4 – Первые пять строк датафрейма movies

Каждый фильм может относиться к нескольким жанрам, которые объединены в одну строку через символ |. Анализ жанров показывает, что большинство фильмов имеют два-три жанра, что отражает реальную практику киноиндустрии, где произведения редко ограничиваются одним жанром. Также есть небольшое количество фильмов без указанных жанров, что требует дополнительной предобработки для корректного включения таких данных в рекомендательную систему.

В названиях фильмов часто содержится год выпуска, заключённый в скобки. Это позволяет извлечь дополнительную информацию о времени выхода фильма и анализировать данные по временным периодам.

Теги, оставленные пользователями, представляют собой текстовые описания, которые дают дополнительные сведения о предпочтениях. Информация о данных в файле tags.csv представлена на Рисунке 3.2.5.

```
><class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   userId      3683 non-null   int64
1   movieId     3683 non-null   int64
2   tag         3683 non-null   object
3   timestamp   3683 non-null   int64
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
```

Рисунок 3.2.5 - Основная информация о данных в файле tags.csv

Анализ тегов показывает, что пользователи используют их довольно разнообразно: одни ограничиваются жанровыми метками, другие оставляют более субъективные описания вроде «funny» или «thriller». Теги помогают расширить рекомендации, особенно в гибридных системах, где учитываются как оценки, так и семантические характеристики фильмов.

Временные метки, присутствующие как в рейтингах, так и в тегах, позволяют анализировать динамику активности пользователей. С их помощью можно определить, когда пользователи наиболее активно выставляли оценки, выявить сезонные колебания интереса к фильмам или построить модели, учитывающие эволюцию предпочтений во времени.

Общий анализ показывает, что датасет ml-latest-small хорошо сбалансирован для экспериментов с рекомендательными системами. Он содержит достаточно данных для выявления закономерностей в поведении пользователей и предпочтениях фильмов, но при этом не требует огромных вычислительных ресурсов, что позволяет быстро проводить предобработку и обучение моделей.

3.3 Предобработка данных

В рамках предобработки удален год выпуска фильма из названия и добавлен отдельный столбец `year` в соответствующий датафрейм.

Временные метки приведены к типу данных `datetime` для более удобной обработки. Жанры преобразованы в список, обработаны фильмы с отсутствующими жанрами.

4 ПРАКТИЧЕСКАЯ ЧАСТЬ

4.1 Функциональные возможности

В практической части реализована интерактивная консольная система для работы с данными MovieLens. Основная цель заключалась в создании среды, позволяющей пользователю просматривать фильмы, искать их по названию и жанрам, оценивать и получать рекомендации на основе собственных оценок и оценок других пользователей. Полный код реализации кинотеатра с рекомендательной системой представлен в Приложении А.

Главное меню кинотеатра представлено на Рисунке 4.1.1.

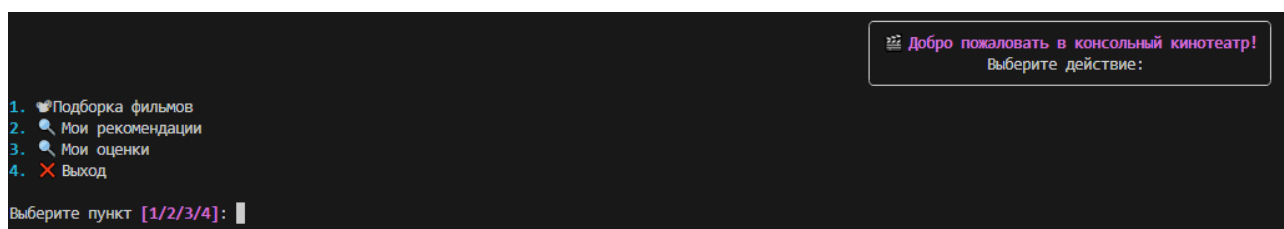


Рисунок 4.1.1 – Главное меню

Пользователю доступен просмотр каталога фильмов, ленты рекомендаций, ранее поставленных оценок, также реализована возможность выхода из приложения.

Каталог фильмов представлен на Рисунке 4.1.2.

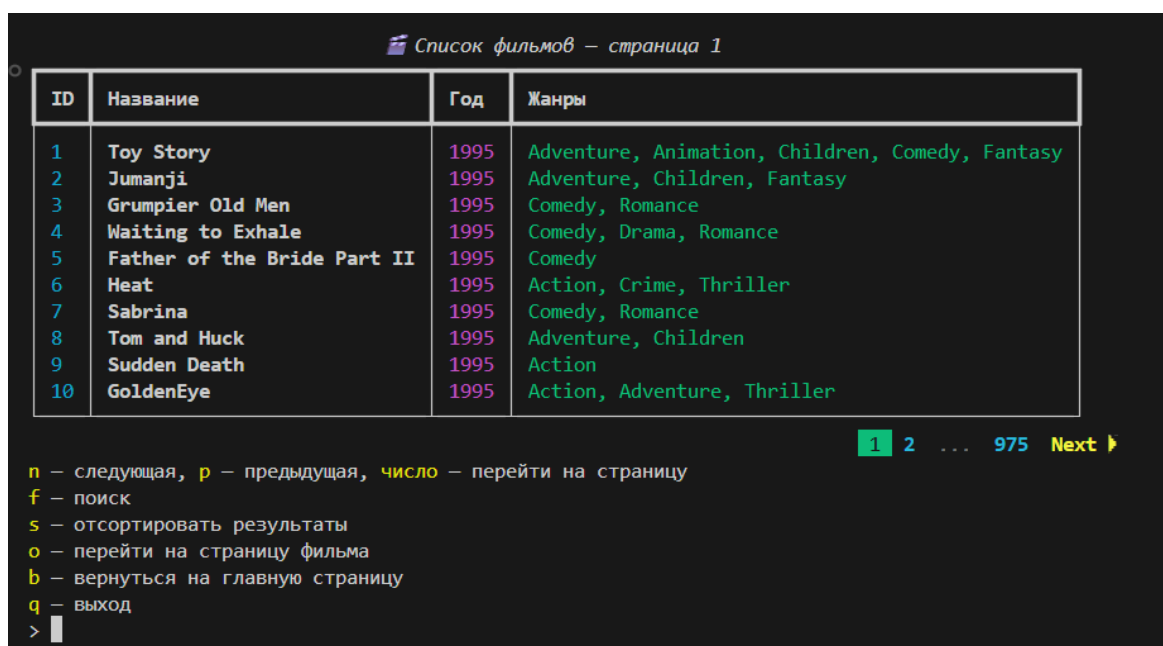


Рисунок 4.1.2 – Каталог фильмов

Сортировка каталога возможна по году выпуска фильм, по жанрам, по идентификатору. Пример сортировки каталога представлен на Рисунке 4.1.3.

Сортировка по Год (desc) – страница 1

ID	Название	Год	Жанры
183635	Maze Runner: The Death Cure	2018	Action, Mystery, Sci-Fi, Thriller
183959	Tom Segura: Disgraceful	2018	Comedy
185033	I Kill Giants	2018	Drama, Fantasy, Thriller
188797	Tag	2018	Comedy
185473	Blockers	2018	Comedy
185435	Game Over, Man!	2018	Action, Comedy
183611	Game Night	2018	Action, Comedy, Crime, Horror
185585	Pacific Rim: Uprising	2018	Action, Fantasy, Sci-Fi
184471	Tomb Raider	2018	Action, Adventure, Fantasy
183295	Insidious: The Last Key	2018	Horror, Mystery, Thriller

1 2 ... 975 Next ▶

n – следующая, p – предыдущая, число – перейти на страницу
 o – открыть страницу фильма, b – назад
 > █

Рисунок 4.1.3 – Сортировка каталога по году по убыванию

Добавлен механизм поиска по каталогу с фильтрацией по году выпуска, жанрам. Поиск осуществляется по названию путем проверки заданной подстроки. Интерфейс поиска представлен на Рисунке 4.1.4.

Введите название фильма (Enter – пропустить): alien

Выберите жанры

№	Жанр
1	Action
2	Adventure
3	Animation
4	Children's
5	Comedy
6	Crime
7	Documentary
8	Drama
9	Fantasy
10	Film-Noir
11	Horror
12	Musical
13	Mystery
14	Romance
15	Sci-Fi
16	Thriller
17	War
18	Western

Введите номера жанров через пробел (Enter – пропустить): █

Рисунок 4.1.4 – Интерфейс поиска

Результат поиска на запрос alien представлены на Рисунке 4.1.5.

Результаты поиска: 20

ID	Название	Год	Жанры
1200	Aliens	1986	Action, Adventure, Horror, Sci-Fi
1214	Alien	1979	Horror, Sci-Fi
1320	Alien³ (a.k.a. Alien 3)	1992	Action, Horror, Sci-Fi, Thriller
1690	Alien: Resurrection	1997	Action, Horror, Sci-Fi
3701	Alien Nation	1988	Crime, Drama, Sci-Fi, Thriller
4526	My Stepmother Is an Alien	1988	Comedy, Romance, Sci-Fi
5051	Italian for Beginners (Italiensk for begyndere)	2000	Comedy, Drama, Romance
5704	Without Warning (a.k.a. Alien Warning) (a.k.a. It Came Without Warning)	1980	Horror, Sci-Fi
6835	Alien Contamination	1980	Action, Horror, Sci-Fi
6899	Alien from L.A.	1988	Sci-Fi

1 2 Next ▶




Фильтры: Название: alien, Жанры: —, Год: —
 n — следующая, p — предыдущая, число — перейти на страницу
 s — отсортировать результаты
 o — открыть страницу фильма по ID
 r — сброс фильтров
 b — назад
 > |

Рисунок 4.1.5 – Результаты поиска на запрос alien

Для составлений рекомендаций пользователю необходимо поставить оценки фильмам. Для этого реализована возможность открытия «страницы» фильма, где доступен просмотр тэгов фильма и присутствует возможность оценить фильм.

Интерфейс страницы фильма представлен на Рисунке 4.1.6.

ID 1200

 Название: Aliens
  Год: 1986
  Жанры: ['Action', 'Adventure', 'Horror', 'Sci-Fi']

Теги

action, aliens, horror, sci-fi, space, space craft, SPACE TRAVEL, suspense, space

Доступные действия:

r — оценить фильм
 b — назад
 q — выйти из приложения
 > |

Рисунок 4.1.6 – Интерфейс страницы фильма

Пример выставления оценки показан на Рисунке 4.1.7.

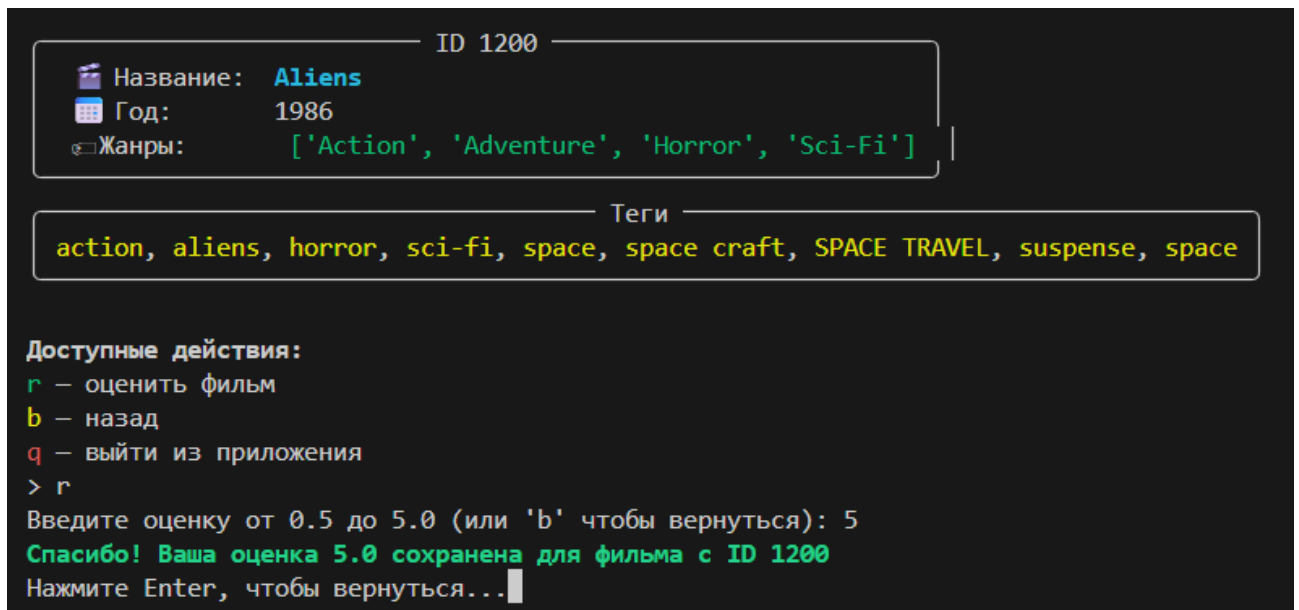


Рисунок 4.1.7 – Выставление оценки для фильма

Проставленные оценки можно посмотреть на отдельной странице в главном меню (Рисунок 4.1.8).

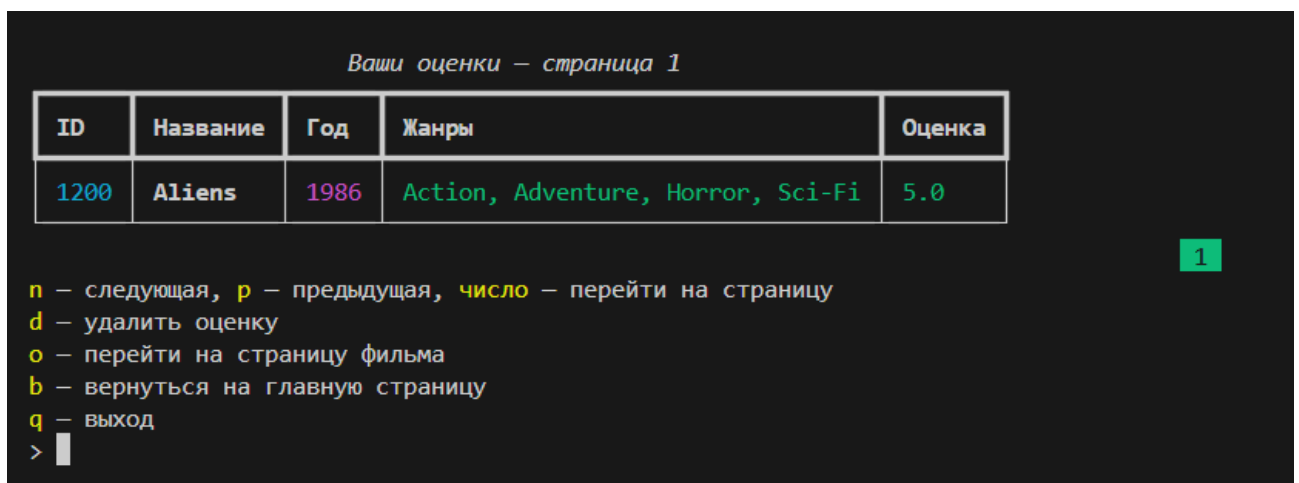


Рисунок 4.1.8 – Проставленные оценки

Добавлена возможность удалить оценку по идентификатору фильма (Рисунок 4.1.9).

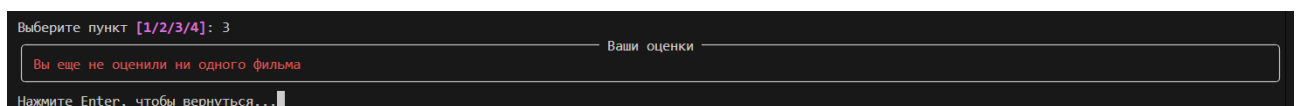


Рисунок 4.1.9 – Удаление оценки

Оценки на фильм с идентификатором 1200 успешно удалена, текущий пользователь не оценил ни одного фильма.

4.2 Рекомендательная система

Основные вычисления рекомендаций выполняются в методе `show_recommendations`. После того, как пользователь оценил некоторое число фильмов для него составляется лента рекомендаций.

Реализована возможность выбора стратегии рекомендаций (User-based, Item-Based), а также метрики близости (Коэффициент Пирсона, Евклидова норма, расстояние Жаккара, косинусная мера). Пользователь может выбирать метрику и стратегию при каждом запросе рекомендаций, что позволяет сравнивать качество работы различных подходов.

Алгоритм User-based стратегии основан на поиске пользователей, чьи профили наиболее похожи на профиль текущего пользователя. Для каждой пары пользователей вычисляется мера сходства по выбранной метрике. После этого для каждого фильма, который пользователь не оценивал, предсказывается рейтинг по формуле среднего с взвешенными отклонениями (Формула 4.1).

$$\widehat{r_{ui}} = \bar{r}_u + \frac{\sum_{v \in U_i} \text{sim}(u,v)(r_{vi} - \bar{r}_v)}{\sum_{v \in U_i} \text{sim}(u,v)} \quad (4.1)$$

где \bar{r}_u – средняя оценка, проставленная пользователем u ;

$\text{sim}(u, v)$ – мера схожести пользователей u и v .

Для Item-based стратегии прогноз строится на основе сходства между фильмами. Для каждого фильма, который пользователь ещё не оценивал, ищутся другие фильмы, которые он уже оценил. На основе выбранной меры близости рассчитываются веса, и итоговый рейтинг оценивается по формуле взвешенного отклонения от среднего по фильмам (Формула 4.2).

$$\widehat{r_{ui}} = \bar{r}_i + \frac{\sum_{j \in I_u} \text{sim}(i,j)(r_{uj} - \bar{r}_j)}{\sum_{j \in I_u} \text{sim}(i,j)} \quad (4.2)$$

где \bar{r}_i – средняя оценка, проставленная объекту i ;

$sim(u, v)$ – мера схожести пользователей u и v .

Создан пользователь с оценками, представленными на Рисунке 4.1.10.

Ваши оценки – страница 1

ID	Название	Год	Жанры	Оценка
1200	Aliens	1986	Action, Adventure, Horror, Sci-Fi	4.5
1214	Alien	1979	Horror, Sci-Fi	4.2
1320	Alien ³ (a.k.a. Alien 3)	1992	Action, Horror, Sci-Fi, Thriller	4.8
1690	Alien: Resurrection	1997	Action, Horror, Sci-Fi	4.6
169984	Alien: Covenant	2017	Action, Horror, Sci-Fi, Thriller	5.0
5219	Resident Evil	2002	Action, Horror, Sci-Fi, Thriller	4.0
5476	Halloween: Resurrection (Halloween 8)	2002	Horror, Thriller	3.0
6058	Final Destination 2	2003	Horror, Thriller	4.7
6379	Wrong Turn	2003	Horror, Thriller	4.0
6880	Texas Chainsaw Massacre, The	2003	Horror	3.9

1 2 Next ▶

n – следующая, p – предыдущая, число – перейти на страницу
d – удалить оценку
o – перейти на страницу фильма
b – вернуться на главную страницу
q – выход
> █

Рисунок 4.1.10 – Пользователь с заданными оценками

Преимущественно оценки поставлены фильмам ужасов.

Первая страница рекомендаций, основанных на подходе User-based изображена на Рисунке 4.1.11.

Рекомендации (user-based, pearson) – страница 1

ID	Название	Год	Жанры
7742	Baxter	1989	Drama, Horror
6967	Dead of Night	1945	Horror, Mystery
4517	Lady in White (a.k.a. The Mystery of the Lady in White)	1988	Horror, Mystery, Thriller
7114	Collector, The	1965	Drama, Horror, Thriller
5105	Don't Look Now	1973	Drama, Horror, Thriller
8632	Secret Society	2002	Comedy
5771	My Bloody Valentine	1981	Drama, Horror, Thriller
135216	The Star Wars Holiday Special	1978	Adventure, Children, Comedy, Sci-Fi
141994	Saving Christmas	2014	Children, Comedy
160872	Satanic	2016	Horror

1 2 ... 528 Next ▶

n – следующая, p – предыдущая, число – перейти на страницу
o – открыть страницу фильма, b – назад
> █

Рисунок 4.1.11 - Первая страница рекомендаций, основанных на подходе User-based

Рекомендованные фильмы преимущественно содержат жанр ужасы, как и ожидалось.

Для количественной оценки качества использованы метрики RMSE и MAE, которые рассчитываются на отложенной тестовой части данных (20 %

случайно выбранных известных оценок заменяются на NaN, после чего восстанавливаются алгоритмом). Расчёт метрик вынесен в отдельный метод `_compute_metrics_for_plot`, а визуализация производится функцией `plot_similarity_metrics`, которая строит столбчатые диаграммы для каждой метрики и меры сходства (Рисунок 4.1.12).

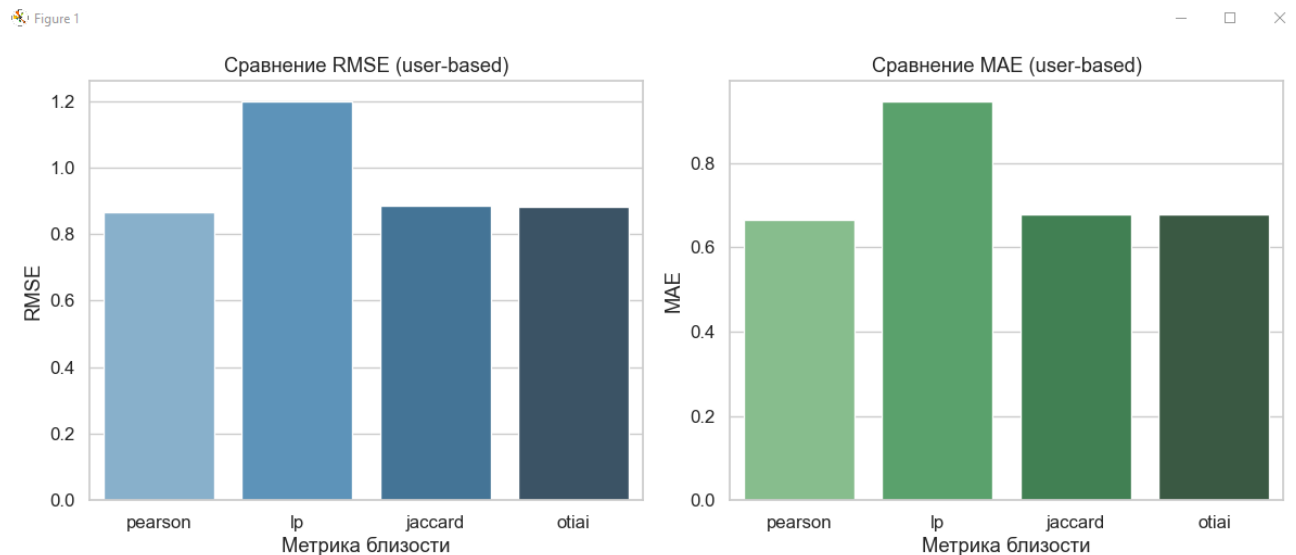


Рисунок 4.1.12 – Расчет метрик

Лучшей метрикой близости является коэффициент Пирсона, а худшая метрика – Евклидово расстояние.

ЗАКЛЮЧЕНИЕ

В ходе данной практической работы успешно достигнута поставленная цель: приобретены навыки реализации и проведен сравнительный анализ анамнестических методов коллаборативной фильтрации для построения рекомендательной системы. Все поставленные задачи были выполнены в полном объеме.

Разработана функциональная консольная система на языке Python, интегрирующая в себя интерактивный каталог фильмов на основе датасета MovieLens и две ключевые стратегии рекомендаций: User-Based (на основе схожести пользователей) и Item-Based (на основе схожести предметов). В рамках этих стратегий реализованы и эмпирически сравнены четыре математические меры сходства: коэффициент корреляции Пирсона, Евклидова норма, расстояние Жаккара и косинусная мера.

Проведенное исследование и расчет метрик ошибок (RMSE и MAE) позволили сделать обоснованный вывод о том, что наилучшее качество прогноза для данной предметной области и набора данных обеспечивает коэффициент корреляции Пирсона. Это объясняется его способностью учитывать не только абсолютные значения оценок, но и относительные отклонения от среднего пользовательского рейтинга, что делает его более устойчивым к индивидуальным особенностям выставления оценок. В то же время Евклидова метрика показала наихудшие результаты, что может быть связано с ее чувствительностью к величине и масштабу данных.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Сорокин, А. Б. Безусловная оптимизация. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин, О. В. Платонова, Л. М. Железняк — М. РТУ МИРЭА , 2020.
2. Сорокин, А. Б. Введение в генетические алгоритмы: теория, расчеты и приложения. [Электронный ресурс] : учебно-метод. пособие / А. Б. Сорокин — М. МИРЭА , 2018.
3. Рекомендательные системы: user-based и item-based [Электронный ресурс]: Habr. URL: <https://habr.com/ru/companies/surfingbird/articles/139518/> (Дата обращения: 14.10.2025).
4. Рекомендательные системы [Электронный ресурс]: URL: https://neerc.ifmo.ru/wiki/index.php?title=Рекомендательные_системы (Дата обращения: 12.10.2025).
5. MovieLens [Электронный ресурс]: URL: <https://grouplens.org/datasets/movielens/> (Дата обращения: 13.10.2025).
6. Item-Based Collaborative Filtering Recommendation Algorithms [Электронный ресурс]: URL: https://files.grouplens.org/papers/www10_sarwar.pdf (Дата обращения: 14.10.2025).

ПРИЛОЖЕНИЯ

Приложение А — Реализация кинотеатра с встроенной рекомендательной системой.

Приложение А

Реализация кинотеатра с встроенной рекомендательной системой

Листинг А – Реализация кинотеатра с встроенной рекомендательной системой

```
import os
import re
import pandas as pd
import numpy as np
from rich.console import Console
from rich.table import Table
from rich.prompt import Confirm, Prompt
from rich.panel import Panel
from rich.align import Align
from rich.text import Text
from enum import Enum
import seaborn as sns
import matplotlib.pyplot as plt

def extract_year(title):
    title = title.strip()
    match = re.search(r"\((\d{4})\)\\s*$", title)
    if match:
        return int(match.group(1))
    return np.nan

class Genres(Enum):
    Action = "Action"
    Adventure = "Adventure"
    Animation = "Animation"
    Children = "Children's"
    Comedy = "Comedy"
    Crime = "Crime"
    Documentary = "Documentary"
    Drama = "Drama"
    Fantasy = "Fantasy"
    FilmNoir = "Film-Noir"
    Horror = "Horror"
    Musical = "Musical"
    Mystery = "Mystery"
    Romance = "Romance"
    SciFi = "Sci-Fi"
    Thriller = "Thriller"
    War = "War"
    Western = "Western"

class MovieLensCinema:
    def __init__(self, path, per_page=10):
        self.path = path
        self.links = None
        self.movies = None
        self.ratings = None
        self.tags = None
        self.console = Console()
        self.rates = pd.DataFrame(columns=["movieId",
"rating"]).set_index("movieId")
        self.load_data()

        self.per_page = per_page

    def load_data(self):
```

```
links_path = os.path.join(self.path, "links.csv")
movies_path = os.path.join(self.path, "movies.csv")
ratings_path = os.path.join(self.path, "ratings.csv")
tags_path = os.path.join(self.path, "tags.csv")

if os.path.exists(links_path):
    self.links = pd.read_csv(
        links_path,
        encoding="utf-8",
        index_col="movieId",
        dtype={"imdbId": "int64", "tmdbId": "Int64"},
    )
else:
    raise FileNotFoundError("Не найдено файла links.csv")

if os.path.exists(movies_path):
    self.movies = pd.read_csv(
        movies_path, encoding="utf-8", index_col="movieId",
quotechar='"'
    )
    self.movies["title"] = self.movies["title"].str.strip()
    self.movies["genres"] = self.movies["genres"].apply(
        lambda x: [] if x == "(no genres listed)" else x.split("|")
    )
    self.movies["year"] = self.movies["title"].apply(extract_year)
    self.movies["year"] = self.movies["year"].astype("Int64")
    self.movies["title"] = self.movies["title"].apply(
        lambda t: re.sub(r"\s*\(\d{4}\)$", "", t)
    )
else:
    raise FileNotFoundError("Не найдено файла movies.csv")

if os.path.exists(ratings_path):
    self.ratings = pd.read_csv(
        ratings_path,
        encoding="utf-8",
        dtype={
            "userId": "int32",
            "movieId": "int32",
            "rating": "float32",
            "timestamp": "int64",
        },
    )
    self.ratings["datetime"] = pd.to_datetime(
        self.ratings["timestamp"], unit="s"
    )
    self.ratings = self.ratings.drop(columns=["timestamp"])
    self.ratings.set_index(["userId", "movieId"], inplace=True)
else:
    raise FileNotFoundError("Не найдено файла ratings.csv")

if os.path.exists(tags_path):
    self.tags = pd.read_csv(
        tags_path,
        dtype={
            "userId": "int32",
            "movieId": "int32",
            "tag": "string",
            "timestamp": "int64",
        },
    ),
```

```

        quotechar="'",
    )
    self.tags["datetime"] = pd.to_datetime(self.tags["timestamp"],
unit="s")
    self.tags.set_index(["userId", "movieId"], inplace=True)
else:
    raise FileNotFoundError("Не найдено файла tags.csv")

def get_rating_matrix(self):
    df = self.ratings.reset_index()

    rating_matrix = df.pivot_table(
        index="userId", columns="movieId", values="rating"
    )

    return rating_matrix

def show_movies(self, page=1):
    start = (page - 1) * self.per_page
    end = start + self.per_page
    subset = self.movies.iloc[start:end]

    table = Table(title=f"🎬 Список фильмов – страница {page}")

    table.add_column("ID", style="cyan", no_wrap=True)
    table.add_column("Название", style="bold")
    table.add_column("Год", justify="center", style="magenta")
    table.add_column("Жанры", style="green")

    for movie_id, row in subset.iterrows():
        genres_str = ", ".join(row["genres"]) if row["genres"] else "-"
        year_str = str(row["year"]) if not pd.isna(row["year"]) else "-"
        table.add_row(str(movie_id), row["title"], year_str, genres_str)

    self.console.print(table)

def run(self):
    page = 1
    total_pages = (len(self.movies) - 1) // self.per_page + 1

    while True:
        os.system("cls" if os.name == "nt" else "clear")
        self.show_movies(page=page)

        self.show_paginator(page, total_pages)

        self.console.print(
            "[yellow]n[/yellow] – следующая, [yellow]p[/yellow] –
предыдущая, "
            "[yellow]число[/yellow] – перейти на страницу\n"
            "[yellow]f[/yellow] – поиск\n"
            "[yellow]s[/yellow] – отсортировать результаты\n"
            "[yellow]o[/yellow] – перейти на страницу фильма\n"
            "[yellow]b[/yellow] – вернуться на главную страницу\n"
            "[yellow]q[/yellow] – выход"
        )

        choice = input("> ").strip().lower()
        if choice == "n" and page < total_pages:
            page += 1
        elif choice == "p" and page > 1:

```

Продолжение Листинга А

```
        page -= 1
    elif choice.isdigit():
        num = int(choice)
        if 1 <= num <= total_pages:
            page = num
    elif choice == "f":
        self.search_movies()
    elif choice == "s":
        self.sort_movies(self.movies)
    elif choice == "o":
        movie_id_str = Prompt.ask("Введите ID фильма (или 'b' для
отмены) ")
        if movie_id_str.lower() == "b":
            continue
        if movie_id_str.isdigit():
            self.show_movie_page(int(movie_id_str))
        else:
            self.console.print("[red]Некорректный ID[/red]")
            input("Нажмите Enter, чтобы вернуться...")
    elif choice == "b":
        break
    elif choice == "q":
        if Confirm.ask("Are you sure?"):
            exit(0)

def menu(self):
    while True:
        os.system("cls" if os.name == "nt" else "clear")

        self.console.print(
            Panel.fit(
                "[bold magenta]🎬 Добро пожаловать в консольный
кинотеатр![/bold magenta]\nВыберите действие:"
            ),
            justify="center",
        )

        self.console.print("[cyan]1.[/cyan] 🎬 Подборка фильмов")
        self.console.print("[cyan]2.[/cyan] 🔍 Мои рекомендации")
        self.console.print("[cyan]3.[/cyan] 🔍 Мои оценки")
        self.console.print("[cyan]4.[/cyan] ❌ Выход")

        choice = Prompt.ask("\nВыберите пункт", choices=["1", "2", "3",
"4"])

        if choice == "1":
            self.run()
        elif choice == "2":
            self.show_recommendations()
        elif choice == "3":
            self.show_my_ratings()
        elif choice == "4":
            if Confirm.ask("[bold red]Вы действительно хотите выйти?[/bold
red]"):
                break

def show_movie_page(self, movie_id: int):
    """Страница фильма с информацией, тегами и возможностью оценки"""
    if movie_id not in self.movies.index:
        self.console.print(f"[red]Фильм с ID {movie_id} не найден.[/red]")
```

Продолжение Листинга А

```
input("Нажмите Enter, чтобы вернуться...")
return

while True:
    os.system("cls" if os.name == "nt" else "clear")

    movie = self.movies.loc[movie_id]

    info_table = Table(show_header=False, box=None)
    info_table.add_row(
        "🎬 Название:", f"[bold cyan]{movie['title']}[/bold cyan]"
    )
    info_table.add_row(
        "📅 Год:", str(movie["year"]) if movie["year"] == movie["year"]
    else "-"
    )
    info_table.add_row("🎭 Жанры:", f"[green]{movie['genres']}[/green]")
    self.console.print(Panel(info_table, title=f"ID {movie_id}",
expand=False))

    if self.tags is not None:
        try:
            df = self.tags.xs(movie_id, level="movieId")
            movie_tags = df["tag"].tolist()
        except KeyError:
            movie_tags = []

        if movie_tags:
            tags_str = ", ".join(f"[yellow]{t}[/yellow]" for t in
movie_tags)
            self.console.print(Panel(tags_str, title="Теги",
expand=False))
        else:
            self.console.print(
                Panel(
                    "[italic grey]Для этого фильма нет тегов[/italic
grey]",
                    title="Теги",
                    expand=False,
                )
            )

    self.console.print(
        "\n[bold]Доступные действия:[/bold]\n"
        "[green]r[/green] – оценить фильм\n"
        "[yellow]b[/yellow] – назад\n"
        "[red]q[/red] – выйти из приложения"
    )

    choice = input("> ").strip().lower()
    if choice == "r":
        self.rate_movie(movie_id)
    elif choice == "b":
        break
    elif choice == "q":
        if Confirm.ask("Are you sure?"):
            exit(0)

def rate_movie(self, movie_id: int):
    """Простейшая система выставления оценки"""
    while True:
```

Продолжение Листинга А

```
        rating = Prompt.ask(
            "Введите оценку от 0.5 до 5.0 (или 'b' чтобы вернуться)"
        )
        if rating.lower() == "b":
            break
        try:
            rating = float(rating)
            if 0.5 <= rating <= 5.0:
                self.rates.loc[movie_id, "rating"] = rating
                self.console.print(
                    f"[bold green]Спасибо! Ваша оценка {rating} сохранена
для фильма с ID {movie_id}[/bold green]"
                )
                input("Нажмите Enter, чтобы вернуться...")
                break
            else:
                self.console.print(
                    "[red]Оценка должна быть в диапазоне 0.5-5.0[/red]"
                )
                input("Нажмите Enter, чтобы вернуться...")
                break
        except ValueError:
            self.console.print("[red]Введите корректное число или
'b'[/red]")
            input("Нажмите Enter, чтобы вернуться...")
            break

    def search_movies(self):
        """Поиск фильмов по названию с фильтрацией по году и жанрам"""
        os.system("cls" if os.name == "nt" else "clear")
        search_query = (
            Prompt.ask("Введите название фильма (Enter –
пропустить)").strip().lower()
        )
        selected_genres = self.choose_genres()
        year_op, year_val = self.choose_year_filter()

        def apply_filters():
            df = self.movies
            if search_query:
                df = df[df["title"].str.lower().str.contains(search_query,
na=False)]
            if selected_genres:
                df = df[
                    df["genres"].apply(
                        lambda g: all(gen in g for gen in selected_genres)
                    )
                ]
            if year_op is not None and year_val is not None:
                if year_op == "=":
                    df = df[df["year"] == year_val]
                elif year_op == ">":
                    df = df[df["year"] > year_val]
                elif year_op == "<":
                    df = df[df["year"] < year_val]
            return df

        filtered_df = apply_filters()
        if filtered_df.empty:
            self.console.print(
                Panel("[red]❌ Фильмы не найдены[/red]", title="Результат")
            )
```

```

    )
    input("Нажмите Enter, чтобы вернуться...")
    return

page = 1
per_page = self.per_page

while True:
    os.system("cls" if os.name == "nt" else "clear")

    total_pages = max(1, (len(filtered_df) - 1) // per_page + 1)
    page = min(page, total_pages)

    start = (page - 1) * per_page
    end = start + per_page
    page_data = filtered_df.iloc[start:end]

    table = Table(
        title=f"Результаты поиска: {len(filtered_df)}",
        show_header=True,
        header_style="bold magenta",
    )
    table.add_column("ID", style="cyan", width=6)
    table.add_column("Название", style="white")
    table.add_column("Год", style="yellow", width=8)
    table.add_column("Жанры", style="green")

    for idx, row in page_data.iterrows():
        year = str(row["year"]) if not pd.isna(row["year"]) else "-"
        genres = ", ".join(row["genres"]) if row["genres"] else "-"
        table.add_row(str(idx), row["title"], year, genres)

    self.console.print(table)

    self.show_paginator(page, total_pages)

    status = f"[bold]Фильтры:[/bold] Название: [cyan]{search_query or '—'}[/cyan], Жанры: [cyan]{'', '.join(selected_genres) if selected_genres else '—'}[/cyan], Год: [cyan]{year_op + str(year_val) if year_op and year_val else '—'}[/cyan]"
    self.console.print(status)

    self.console.print(
        "[yellow]n[/yellow] — следующая, [yellow]p[/yellow] — предыдущая, [yellow]число[/yellow] — перейти на страницу\n"
        "[yellow]s[/yellow] — отсортировать результаты\n"
        "[yellow]o[/yellow] — открыть страницу фильма по ID\n"
        "[yellow]r[/yellow] — сброс фильтров\n"
        "[yellow]b[/yellow] — назад"
    )

    choice = input("> ").strip().lower()
    if choice == "n" and page < total_pages:
        page += 1
    elif choice == "p" and page > 1:
        page -= 1
    elif choice == "s":
        self.sort_movies(filtered_df)
    elif choice.isdigit():
        num = int(choice)
        if 1 <= num <= total_pages:

```



```

        page = num
    elif choice == "o":
        movie_id_str = Prompt.ask("Введите ID фильма (или 'b' чтобы
вернуться)")
        if movie_id_str.lower() == "b":
            continue
        try:
            movie_id = int(movie_id_str)
            if movie_id in self.movies.index:
                self.show_movie_page(movie_id)
            else:
                self.console.print("[red]Фильм с таким ID не
найден[/red]")
                input("Нажмите Enter, чтобы продолжить...")
        except ValueError:
            self.console.print("[red]Введите корректный ID[/red]")
            input("Нажмите Enter, чтобы продолжить...")
    elif choice == "r":
        search_query = ""
        selected_genres = []
        year_op, year_val = None, None
        filtered_df = self.movies
        page = 1
    elif choice == "b":
        break

def choose_genres(self):
    """Выбор одного или нескольких жанров через консоль с таблицей"""
    genre_list = list(Genres)
    table = Table(
        title="Выберите жанры", show_header=True, header_style="bold
magenta"
    )
    table.add_column("№", justify="center", style="cyan", width=4)
    table.add_column("Жанр", justify="left", style="green")

    for i, genre in enumerate(genre_list, 1):
        table.add_row(str(i), genre.value)

    self.console.print(table)
    choice = Prompt.ask(
        "Введите номера жанров через пробел (Enter — пропустить)"
    ).strip()
    if not choice:
        return []

    selected_genres = []
    for num in choice.split():
        num = num.strip()
        if num.isdigit():
            idx = int(num) - 1
            if 0 <= idx < len(genre_list):
                selected_genres.append(genre_list[idx].value)
    return selected_genres

def choose_year_filter(self):
    """Выбор фильтра по году с оператором"""
    op = Prompt.ask(
        "Выберите оператор для фильтра по году",
        choices=[">", "<", "="],
        default="=",
    )

```

```

    )
    year_str = Prompt.ask("Введите год").strip()
    if not year_str.isdigit():
        self.console.print("[red]Некорректный год, фильтр не будет
применён[/red]")
        return None, None
    return op, int(year_str)

def show_paginator(self, page, total_pages):
    """Красивый центрированный пагинатор"""
    paginator_text = Text()
    last_was_ellipsis = False

    if page > 1:
        paginator_text.append("◀ Prev ", style="bold yellow")
    else:
        paginator_text.append(" ")

    for p in range(1, total_pages + 1):
        if p == 1 or p == total_pages or abs(p - page) <= 1:
            if p == page:
                paginator_text.append(f" {p} ", style="reverse green")
            else:
                paginator_text.append(f" {p} ", style="bold cyan")
            last_was_ellipsis = False
        else:
            if not last_was_ellipsis:
                paginator_text.append(" ... ", style="dim")
                last_was_ellipsis = True

    if page < total_pages:
        paginator_text.append(" Next ►", style="bold yellow")

    self.console.print(Align.center(paginator_text))

def show_my_ratings(self):
    if self.rates.empty:
        self.console.print(
            Panel(
                "[red]Вы еще не оценили ни одного фильма[/red]", title="Ваши
оценки"
            )
        )
    input("Нажмите Enter, чтобы вернуться...")
    return

page = 1
df = self.rates.merge(self.movies, on="movieId")
total_pages = (len(df) - 1) // self.per_page + 1

while len(df):
    os.system("cls" if os.name == "nt" else "clear")

    start = (page - 1) * self.per_page
    end = start + self.per_page
    subset = df.iloc[start:end]

    rates = Table(show_header=True, title=f"Ваши оценки — страница
{page}")
    rates.add_column("ID", style="cyan", no_wrap=True)
    rates.add_column("Название", style="bold")

```

Продолжение Листинга А

```
rates.add_column("Год", justify="center", style="magenta")
rates.add_column("Жанры", style="green")
rates.add_column("Оценка", style="green")

for movie_id, row in subset.iterrows():
    genres_str = ", ".join(row["genres"]) if row["genres"] else "-"
    year_str = str(row["year"]) if not pd.isna(row["year"]) else "-"
    rates.add_row(
        str(movie_id),
        row["title"],
        year_str,
        genres_str,
        str(row["rating"]),
    )

self.console.print(rates)

self.show_paginator(page, total_pages)

self.console.print(
    "[yellow]n[/yellow] – следующая, [yellow]p[/yellow] –
предыдущая, "
    "[yellow]число[/yellow] – перейти на страницу\n"
    "[yellow]d[/yellow] – удалить оценку\n"
    "[yellow]o[/yellow] – перейти на страницу фильма\n"
    "[yellow]b[/yellow] – вернуться на главную страницу\n"
    "[yellow]q[/yellow] – выход"
)
choice = input("> ").strip().lower()
if choice == "n" and page < total_pages:
    page += 1
elif choice == "p" and page > 1:
    page -= 1
elif choice.isdigit():
    num = int(choice)
    if 1 <= num <= total_pages:
        page = num
elif choice == "d":
    movie_id_str = Prompt.ask(
        "Введите ID фильма, у которого хотите удалить оценку (или
'b' для отмены)"
    )
    if movie_id_str.lower() == "b":
        continue
    if movie_id_str.isdigit():
        id = int(movie_id_str)
        if id in self.rates.index:
            self.rates = self.rates.drop(index=id)
            df = df.drop(index=id)
        else:
            self.console.print(
                "[red]Не найдено оценки для фильма с заданным
ID[/red]"
            )
            input("Нажмите Enter, чтобы вернуться...")
    else:
        self.console.print("[red]Некорректный ID[/red]")
        input("Нажмите Enter, чтобы вернуться...")
elif choice == "o":
    movie_id_str = Prompt.ask("Введите ID фильма (или 'b' для
отмены)")
```

```
        if movie_id_str.lower() == "b":
            continue
        if movie_id_str.isdigit():
            self.show_movie_page(int(movie_id_str))
        else:
            self.console.print("[red]Некорректный ID[/red]")
            input("Нажмите Enter, чтобы вернуться...")
    elif choice == "b":
        break
    elif choice == "q":
        if Confirm.ask("Are you sure?"):
            exit(0)

def sort_movies(self, frame):
    """Сортировка фильмов по различным полям"""
    sort_fields = [
        ("ID", "movieId"),
        ("Название", "title"),
        ("Год", "year"),
        ("Жанры", "genres"),
    ]

    table = Table(
        title="Выберите поле для сортировки",
        show_header=True,
        header_style="bold magenta",
    )
    table.add_column("№", justify="center", style="cyan", width=4)
    table.add_column("Поле", style="green")

    for i, (label, _) in enumerate(sort_fields, 1):
        table.add_row(str(i), label)

    self.console.print(table)

    choice = Prompt.ask("Введите номер поля (или Enter для отмены)").strip()

    if not choice.isdigit():
        return

    choice_num = int(choice)
    if not 1 <= choice_num <= len(sort_fields):
        self.console.print("[red]Некорректный выбор[/red]")
        input("Нажмите Enter, чтобы вернуться...")
        return

    field_label, field_name = sort_fields[choice_num - 1]

    direction = Prompt.ask(
        "Выберите направление сортировки", choices=["asc", "desc"],
default="asc"
    )

    ascending = direction == "asc"

    if field_name == "movieId":
        sorted_df = frame.sort_index(ascending=ascending)
    elif field_name == "genres":
        sorted_df = frame.copy()
        sorted_df["__sort_genre"] = sorted_df["genres"].apply(
            lambda g: g[0] if g else ""

```

```

        )
        sorted_df = sorted_df.sort_values("__sort_genre",
ascending=ascending).drop(
        columns="__sort_genre"
        )
    else:
        sorted_df = frame.sort_values(field_name, ascending=ascending)

    self.paginated_view(
        sorted_df, title=f"Сортировка по {field_label} ({direction})"
    )

def paginated_view(self, df, title="Список фильмов"):
    """Универсальный постраничный просмотр DataFrame фильмов"""
    page = 1
    total_pages = max(1, (len(df) - 1) // self.per_page + 1)

    while True:
        os.system("cls" if os.name == "nt" else "clear")
        start = (page - 1) * self.per_page
        end = start + self.per_page
        subset = df.iloc[start:end]

        table = Table(title=f"{title} - страница {page}", show_header=True)
        table.add_column("ID", style="cyan", no_wrap=True)
        table.add_column("Название", style="bold")
        table.add_column("Год", justify="center", style="magenta")
        table.add_column("Жанры", style="green")

        for movie_id, row in subset.iterrows():
            genres_str = ", ".join(row["genres"]) if row["genres"] else "-"
            year_str = str(row["year"]) if not pd.isna(row["year"]) else "-"
            table.add_row(str(movie_id), row["title"], year_str, genres_str)

        self.console.print(table)
        self.show_paginator(page, total_pages)
        self.console.print(
            f"[yellow]n[/yellow] - следующая, [yellow]p[/yellow] -
предыдущая, "
            f"[yellow]число[/yellow] - перейти на страницу\n"
            f"[yellow]o[/yellow] - открыть страницу фильма,
[yellow]b[/yellow] - назад"
        )

        choice = input("> ").strip().lower()
        if choice == "n" and page < total_pages:
            page += 1
        elif choice == "p" and page > 1:
            page -= 1
        elif choice.isdigit():
            num = int(choice)
            if 1 <= num <= total_pages:
                page = num
        elif choice == "o":
            movie_id_str = Prompt.ask("Введите ID фильма (или 'b' для
отмены)")

            if movie_id_str.lower() == "b":
                continue
            if movie_id_str.isdigit():
                self.show_movie_page(int(movie_id_str))
            elif choice == "b":

```

```
        break
        os.system("cls" if os.name == "nt" else "clear")

    def show_recommendations(self):
        """Вывод рекомендаций для текущего пользователя, включая оценки из
self.rates"""
        os.system("cls" if os.name == "nt" else "clear")

        R = self.get_rating_matrix()
        user_ids = R.index.tolist()
        movie_ids = R.columns.tolist()

        current_user_id = -1

        current_user_ratings = pd.Series(
            [np.nan] * len(movie_ids), index=movie_ids, dtype=float
        )

        for movie_id in self.rates.index:
            if movie_id in movie_ids:
                current_user_ratings[movie_id] = self.rates.loc[movie_id,
"rating"]

        current_user_df = pd.DataFrame([current_user_ratings],
index=[current_user_id])

        if current_user_df.notna().any().any():
            R = pd.concat([R, current_user_df])
            user_ids.append(current_user_id)
            current_user_idx = R.index.get_loc(current_user_id)
        else:
            self.console.print(
                Panel(
                    "[red]У вас нет оценок для рекомендаций. Сначала оцените
несколько фильмов.[/red]",
                    title="Рекомендации",
                )
            )
            input("Нажмите Enter, чтобы вернуться...")
            return

        strategy = Prompt.ask(
            "Выберите стратегию [user-based/item-based]",
            choices=["user-based", "item-based"],
            default="user-based",
        )

        metric = Prompt.ask(
            "Выберите меру сходства [jaccard/lp/otiai/pearson]",
            choices=["jaccard", "lp", "otiai", "pearson"],
            default="pearson",
        )

        R_values = R.values.astype(np.float64)
        n_users, n_items = R_values.shape
        user_means = np.nanmean(R_values, axis=1)
        item_means = np.nanmean(R_values, axis=0)
        preds = np.full(n_items, np.nan)

        if strategy == "user-based":
            sims = np.zeros(n_users)
```

```

        current_ratings = R_values[current_user_idx, :]
        for u in range(n_users):
            if u == current_user_idx:
                continue
            other_ratings = R_values[u, :]
            mask = ~np.isnan(current_ratings) & ~np.isnan(other_ratings)
            if np.sum(mask) == 0:
                sims[u] = 0
                continue
            if metric == "pearson":
                sims[u] = np.corrcoef(current_ratings[mask],
other_ratings[mask])[
                    0, 1
                ]
            elif metric == "lp":
                sims[u] = -np.linalg.norm(
                    current_ratings[mask] - other_ratings[mask], ord=2
                )
            elif metric == "jaccard":
                sims[u] = np.sum(
                    (current_ratings[mask] > 0) & (other_ratings[mask] > 0)
                ) / np.sum((current_ratings[mask] > 0) |
other_ratings[mask] > 0))
            elif metric == "otiai":
                sims[u] = np.sum(current_ratings[mask] *
other_ratings[mask]) / (
                    np.linalg.norm(current_ratings[mask])
                    * np.linalg.norm(other_ratings[mask])
                )
            else:
                sims[u] = 0

        for i in range(n_items):
            if not np.isnan(R_values[current_user_idx, i]):
                continue
            mask = ~np.isnan(R_values[:, i])
            if np.sum(mask) == 0:
                continue
            numerator = np.sum(sims[mask] * (R_values[mask, i] -
user_means[mask]))
            denominator = np.sum(np.abs(sims[mask])) + 1e-8
            preds[i] = user_means[current_user_idx] + numerator /
denominator

    else:
        for i in range(n_items):
            if not np.isnan(R_values[current_user_idx, i]):
                continue
            rated_mask = ~np.isnan(R_values[current_user_idx, :])
            sims_i = []
            ratings_i = []
            for j in np.where(rated_mask)[0]:
                mask = ~np.isnan(R_values[:, i]) & ~np.isnan(R_values[:, j])
                if np.sum(mask) == 0:
                    sim = 0
                else:
                    if metric == "pearson":
                        sim = np.corrcoef(R_values[mask, i], R_values[mask,
j])[
                            0, 1
                        ]

```

```

        elif metric == "lp":
            sim = -np.linalg.norm(
                R_values[mask, i] - R_values[mask, j], ord=2
            )
        elif metric == "jaccard":
            sim = np.sum(
                (R_values[mask, i] > 0) & (R_values[mask, j] >
0)
                ) / np.sum(
                (R_values[mask, i] > 0) | (R_values[mask, j] >
0)
                )
        elif metric == "otiai":
            sim = np.sum(R_values[mask, i] * R_values[mask, j])
/ (
            np.linalg.norm(R_values[mask, i])
            * np.linalg.norm(R_values[mask, j])
            )
        else:
            sim = 0
            sims_i.append(sim)
            ratings_i.append(R_values[current_user_idx, j] -
item_means[j])
            sims_i = np.array(sims_i)
            ratings_i = np.array(ratings_i)
            if np.sum(np.abs(sims_i)) > 0:
                preds[i] = item_means[i] + np.dot(sims_i, ratings_i) /
np.sum(
                    np.abs(sims_i)
                )

    recs_df = pd.DataFrame({"movieId": movie_ids, "pred_rating": preds})
    recs_df = recs_df.dropna().sort_values("pred_rating", ascending=False)

    self.paginated_view(
        df=self.movies.loc[recs_df["movieId"]],
        title=f"Рекомендации ({strategy}, {metric})",
    )

def plot_similarity_metrics(
    self, strategy: str = "user-based", test_ratio: float = 0.2
):
    metrics = ["pearson", "lp", "jaccard", "otiai"]
    results = []

    for metric in metrics:
        rmse, mae = self._compute_metrics_for_plot(metric, strategy,
test_ratio)
        results.append({"metric": metric, "RMSE": rmse, "MAE": mae})

    df = pd.DataFrame(results)

    sns.set_theme(style="whitegrid", font_scale=1.1)
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))

    sns.barplot(x="metric", y="RMSE", data=df, ax=axes[0],
palette="Blues_d")
    axes[0].set_title(f"Сравнение RMSE ({strategy})")
    axes[0].set_xlabel("Метрика близости")
    axes[0].set_ylabel("RMSE")

```



```

sns.barplot(x="metric", y="MAE", data=df, ax=axes[1],
palette="Greens_d")
axes[1].set_title(f"Сравнение MAE ({strategy})")
axes[1].set_xlabel("Метрика близости")
axes[1].set_ylabel("MAE")

plt.tight_layout()
plt.show()

def _compute_metrics_for_plot(self, metric: str, strategy: str, test_ratio:
float):
    R = self.get_rating_matrix().copy().values.astype(float)
    n_users, n_items = R.shape

    rng = np.random.default_rng(42)
    train = R.copy()
    test_mask = ~np.isnan(R) & (rng.random(R.shape) < test_ratio)
    test_true = np.full_like(R, np.nan)
    test_true[test_mask] = R[test_mask]
    train[test_mask] = np.nan

    user_means = np.nanmean(train, axis=1)
    item_means = np.nanmean(train, axis=0)
    preds = np.full_like(R, np.nan)

    for u in range(n_users):
        if strategy == "user-based":
            sims = np.zeros(n_users)
            current = train[u, :]
            for v in range(n_users):
                if v == u:
                    continue
                other = train[v, :]
                mask = ~np.isnan(current) & ~np.isnan(other)
                if np.sum(mask) == 0:
                    sims[v] = 0
                    continue
                if metric == "pearson":
                    sims[v] = np.corrcoef(current[mask], other[mask])[0, 1]
                elif metric == "lp":
                    sims[v] = -np.linalg.norm(current[mask] - other[mask])
                elif metric == "jaccard":
                    sims[v] = np.sum(
                        (current[mask] > 0) & (other[mask] > 0)
                    ) / np.sum((current[mask] > 0) | (other[mask] > 0))
                elif metric == "otiai":
                    sims[v] = np.sum(current[mask] * other[mask]) / (
                        np.linalg.norm(current[mask]) *
np.linalg.norm(other[mask])
                    )
            else:
                sims[v] = 0
        for i in range(n_items):
            if not np.isnan(train[u, i]):
                continue
            mask = ~np.isnan(train[:, i])
            if np.sum(mask) == 0:
                continue
            numerator = np.sum(sims[mask] * (train[mask, i] -
user_means[mask]))

```

```

        denominator = np.sum(np.abs(sims[mask])) + 1e-8
        preds[u, i] = user_means[u] + numerator / denominator

    else:
        for i in range(n_items):
            if not np.isnan(train[u, i]):
                continue
            rated_mask = ~np.isnan(train[u, :])
            sims_i = []
            ratings_i = []
            for j in np.where(rated_mask)[0]:
                mask = ~np.isnan(train[:, i]) & ~np.isnan(train[:, j])
                if np.sum(mask) == 0:
                    sim = 0
                else:
                    if metric == "pearson":
                        sim = np.corrcoef(train[mask, i], train[mask,
j]))[0, 1]

                    elif metric == "lp":
                        sim = -np.linalg.norm(train[mask, i] -
train[mask, j])

                    elif metric == "jaccard":
                        sim = np.sum(
                            (train[mask, i] > 0) & (train[mask, j] > 0)
                        ) / np.sum((train[mask, i] > 0) | (train[mask,
j] > 0))

                    elif metric == "otiai":
                        sim = np.sum(train[mask, i] * train[mask, j]) /
(
                            np.linalg.norm(train[mask, i])
                            * np.linalg.norm(train[mask, j])
                        )
                    else:
                        sim = 0
                sims_i.append(sim)
                ratings_i.append(train[u, j] - item_means[j])
            sims_i = np.array(sims_i)
            ratings_i = np.array(ratings_i)
            if np.sum(np.abs(sims_i)) > 0:
                preds[u, i] = item_means[i] + np.dot(
                    sims_i, ratings_i
                ) / np.sum(np.abs(sims_i))

        mask_eval = ~np.isnan(test_true) & ~np.isnan(preds)
        if np.sum(mask_eval) == 0:
            return np.nan, np.nan

        diff = preds[mask_eval] - test_true[mask_eval]
        rmse = np.sqrt(np.mean(diff**2))
        mae = np.mean(np.abs(diff))
        return rmse, mae

if __name__ == "__main__":
    cinema = MovieLensCinema(r"..\\ml-latest-small")
    cinema.menu()
    # cinema.plot_similarity_metrics()

```