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

**Институт информационных технологий (ИИТ)**

**Кафедра прикладной математики (ПМ)**

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ**

по дисциплине «Модели и методы предиктивной аналитики»

**Практическое занятие № 2**

|  |  |  |  |
| --- | --- | --- | --- |
| Студент группы ИНБО-01-17 | *ИМБО-02-22, Ким Кирилл Сергеевич* | (подпись) | |
| Преподаватель | *Крынецкий Борис Алексеевич, преподаватель* | (подпись) | |
| Отчет представлен | «\_\_\_»\_\_\_\_\_\_\_\_202\_\_г. | |  | |

Москва 2025 г.

СОДЕРЖАНИЕ

[ПРАКТИЧЕСКАЯ РАБОТА 2 3](#_Toc211350059)

[Введение 3](#_Toc211350060)

[Шаги выполнения 3](#_Toc211350061)

[Результат работы: 18](#_Toc211350062)

[Вывод: 18](#_Toc211350063)

[Список использованных источников и литературы: 19](#_Toc211350064)

ПРАКТИЧЕСКАЯ РАБОТА 2

Введение

**Цель**

Создать рекомендательную систему, которая будет рекомендовать элементы на основе предпочтений пользователей. Найти данные для рекомендаций, провести их обработку и построить модель для генерации рекомендаций.

Шаги выполнения

1. Сбор данных для рекомендаций.

Данные были взяты из Kaggle (<https://www.kaggle.com/datasets/syedjaferk/book-crossing-dataset?resource=download>)

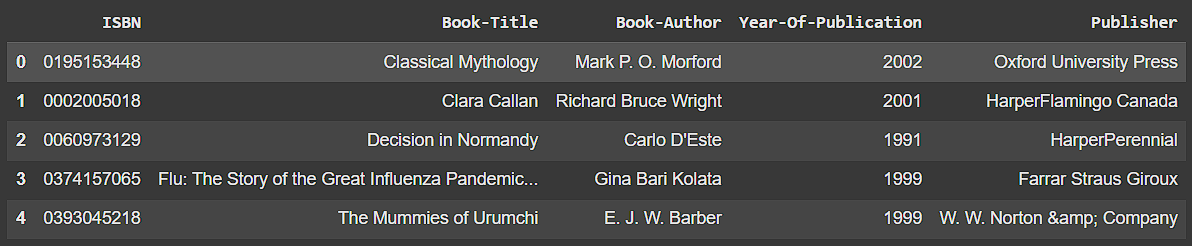
Листинг 1 – Сбор данных

|  |
| --- |
| import re  import pickle  import operator  import numpy as np  import pandas as pd  import seaborn as sns  import matplotlib.pyplot as plt  from collections import Counter  from scipy.sparse import csr\_matrix  from pandas.api.types import is\_numeric\_dtype  from sklearn.neighbors import NearestNeighbors  from sklearn.feature\_extraction import DictVectorizer  from sklearn.metrics.pairwise import cosine\_similarity  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error  from sklearn.feature\_extraction.text import TfidfVectorizer  books = pd.read\_csv("BX-Books.csv", sep=";", encoding="latin-1", quotechar='"', on\_bad\_lines="skip", low\_memory=False)  users = pd.read\_csv("BX-Users.csv", sep=";", encoding="latin-1", quotechar='"', on\_bad\_lines="skip", low\_memory=False)  ratings = pd.read\_csv("BX-Book-Ratings.csv", sep=";", encoding="latin-1", quotechar='"', on\_bad\_lines="skip", low\_memory=False)  print(f"Книги: {books.shape}")  print(f"Пользователи: {users.shape}")  print(f"Рейтинги: {ratings.shape}") |

1. Обработка данных

Листинг 2 – Обработка данных для таблицы книга

|  |
| --- |
| books.drop(['Image-URL-S', 'Image-URL-M', 'Image-URL-L'], axis=1, inplace=True)  books.head()  books.loc[books['Book-Author'].isnull(),:]  books.loc[books['Publisher'].isnull(),:]  books.at[117770 ,'Book-Author'] = 'Other'  books.at[118033 ,'Book-Author'] = 'Other'  books.at[187689 ,'Book-Author'] = 'Other'  books.at[128890 ,'Publisher'] = 'Other'  books.at[129037 ,'Publisher'] = 'Other'  books['Year-Of-Publication'].unique()  books.loc[books['Year-Of-Publication'] == 'DK Publishing Inc',:]  books.loc[books['Year-Of-Publication'] == 'Gallimard',:]  books.at[209538, 'Publisher'] = 'DK Publishing Inc'  books.at[209538, 'Year-Of-Publication'] = 2000  books.at[209538, 'Book-Title'] = 'DK Readers: Creating the X-Men, How It All Began (Level 4: Proficient Readers)'  books.at[209538, 'Book-Author'] = 'Michael Teitelbaum'  books.at[221678, 'Publisher'] = 'DK Publishing Inc'  books.at[221678, 'Year-Of-Publication'] = 2000  books.at[221678, 'Book-Title'] = 'DK Readers: Creating the X-Men, How Comic Books Come to Life (Level 4: Proficient Readers)'  books.at[221678, 'Book-Author'] = 'James Buckley'  books.at[220731, 'Publisher'] = 'Gallimard'  books.at[220731, 'Year-Of-Publication'] = 2003  books.at[220731, 'Book-Title'] = 'Peuple du ciel - Suivi de Les bergers'  books.at[220731, 'Book-Author'] = 'Jean-Marie Gustave Le Clézio'  count = Counter(books['Year-Of-Publication'])  [k for k, v in count.items() if v == max(count.values())]  books.loc[books['Year-Of-Publication'] > 2021, 'Year-Of-Publication'] = 2002  books.loc[books['Year-Of-Publication'] == 0, 'Year-Of-Publication'] = 2002  books.drop\_duplicates(keep='last', inplace=True)  books.reset\_index(drop = True, inplace = True) |

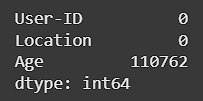


**Рисунок 1 – Данные о книгах**

1. Предварительная обработка о пользователях

Листинг 3 – Обработка данных о пользователях

|  |
| --- |
| print(users.isna().sum())  required = users[users['Age'] <= 80]  required = required[required['Age'] >= 10]  mean = round(required['Age'].mean())  mean  # возрастной диапазон от 10 до 80 лет  users.loc[users['Age'] > 80, 'Age'] = mean    # выбросы, возраст которых превышает 80 лет, заменяются средним значением  users.loc[users['Age'] < 10, 'Age'] = mean    # отклонения в возрасте менее 10 лет заменяются средним значением  users['Age'] = users['Age'].fillna(mean)      # заполнение нулевых значений средним значением  users['Age'] = users['Age'].astype(int)       # изменение типа данных на int  # В столбце «Местоположение» указаны 3 значения: город, штат и страна.  # Они разделены на 3 отдельных столбца: «Город», «Штат» и «Страна» соответственно.  # В случае отсутствия значения в качестве значения сущности указано «Другое». |

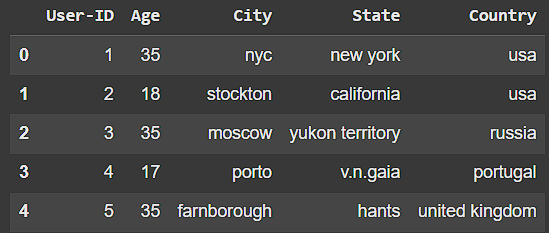


**Рисунок 2 – Данные о пользователях**

1. Обработка о пользователях

Листинг 4 – Обработка о пользователях

|  |
| --- |
| list\_ = users.Location.str.split(', ')  city = []  state = []  country = []  count\_no\_state = 0  count\_no\_country = 0  for i in range(0,len(list\_)):      if list\_[i][0] == ' ' or list\_[i][0] == '' or list\_[i][0]=='n/a' or list\_[i][0] == ',':  #removing invalid entries too          city.append('other')      else:          city.append(list\_[i][0].lower())      if(len(list\_[i])<2):          state.append('other')          country.append('other')          count\_no\_state += 1          count\_no\_country += 1      else:          if list\_[i][1] == ' ' or list\_[i][1] == '' or list\_[i][1]=='n/a' or list\_[i][1] == ',':   #removing invalid entries              state.append('other')              count\_no\_state += 1          else:              state.append(list\_[i][1].lower())          if(len(list\_[i])<3):              country.append('other')              count\_no\_country += 1          else:              if list\_[i][2] == ''or list\_[i][1] == ',' or list\_[i][2] == ' ' or list\_[i][2] == 'n/a':                  country.append('other')                  count\_no\_country += 1              else:                  country.append(list\_[i][2].lower())  users = users.drop('Location',axis=1)  temp = []  for ent in city:      c = ent.split('/')            # обработка случаев, когда данные о городе/ штате из списка городов уже указаны в качестве штата      temp.append(c[0])  df\_city = pd.DataFrame(temp,columns=['City'])  df\_state = pd.DataFrame(state,columns=['State'])  df\_country = pd.DataFrame(country,columns=['Country'])  users = pd.concat([users, df\_city], axis=1)  users = pd.concat([users, df\_state], axis=1)  users = pd.concat([users, df\_country], axis=1)  print(count\_no\_country)   # при выводе количество стран не имело никаких значений  print(count\_no\_state)     # вывод состояний, которые не имели никаких значений  # Удаление повторяющихся записей из таблицы  users.drop\_duplicates(keep='last', inplace=True)  users.reset\_index(drop=True, inplace=True)  users.head() |

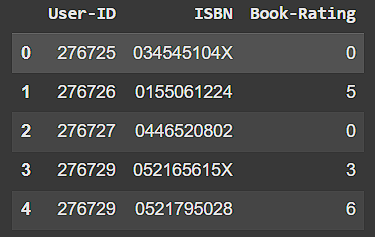


**Рисунок 2 – Данные о пользователях**

1. Предварительная обработка о рейтингах

Листинг 5 – Обработка данных о рейтингах

|  |
| --- |
| ratings.isnull().sum()  print(is\_numeric\_dtype(ratings['Book-Rating']))  print(is\_numeric\_dtype(ratings['User-ID']))  flag = 0  k =[]  reg = "[^A-Za-z0-9]"  for x in ratings['ISBN']:  z = re.search(reg,x)  if z:  flag = 1  if flag == 1:  print("False")  else:  print("True")  # удаляю лишние символы из ISBN  bookISBN = books['ISBN'].tolist()  reg = "[^A-Za-z0-9]"  for index, row\_Value in ratings.iterrows():      z = re.search(reg, row\_Value['ISBN'])      if z:          f = re.sub(reg,"",row\_Value['ISBN'])          if f in bookISBN:              ratings.at[index , 'ISBN'] = f  # Преобразовал все буквы в столбце ISBN в верхний регистр  ratings['ISBN'] = ratings['ISBN'].str.upper()  # Удалил повторяющиеся записи из таблицы  ratings.drop\_duplicates(keep='last', inplace=True)  ratings.reset\_index(drop=True, inplace=True)  ratings.head() |

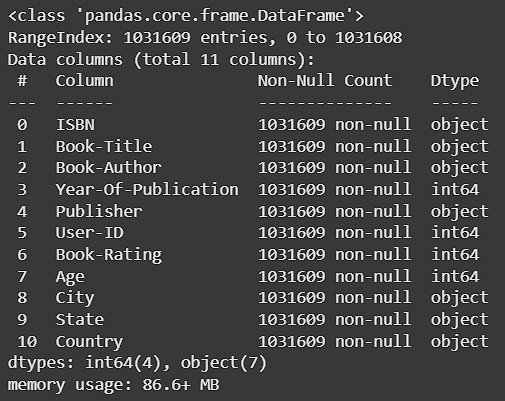


**Рисунок 3 – Данные о пользователях**

1. Объединяю три таблицы

Листинг 6 – Объединение таблиц

|  |
| --- |
| dataset = pd.merge(books, ratings, on='ISBN', how='inner')  dataset = pd.merge(dataset, users, on='User-ID', how='inner')  dataset.info() |



**Рисунок 4 – Информация о данных**

1. Разделение данные на явные и неявные

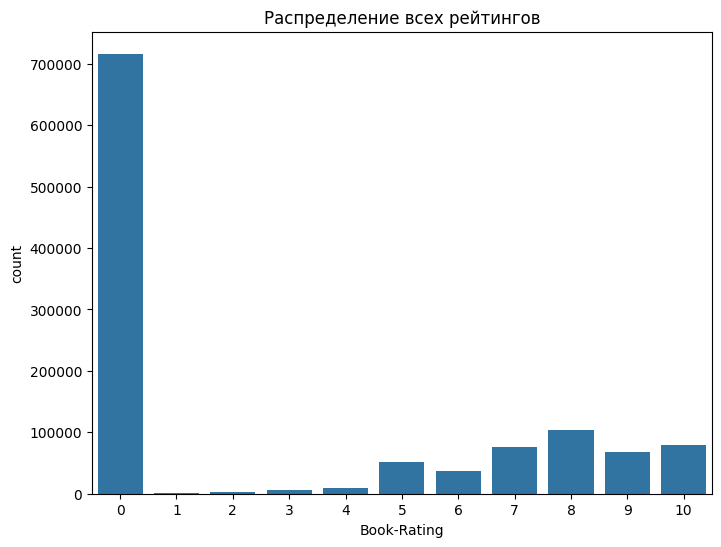
Листинг 7 – Объединение таблиц

|  |
| --- |
| dataset1 = dataset[dataset['Book-Rating'] != 0] # явные рейтинги (пользователь поставил оценку от 1 до 10)  dataset1 = dataset1.reset\_index(drop = True)  dataset1.shape  dataset2 = dataset[dataset['Book-Rating'] == 0] # неявные (оценка = 0, возможно, просто просмотр)  dataset2 = dataset2.reset\_index(drop = True)  dataset2.shape  dataset1.head() |

1. Визуализируем

Листинг 8 – Распределение всех рейтингов

|  |
| --- |
| # Распределение всех рейтингов  plt.figure(figsize=(8,6))  sns.countplot(x="Book-Rating", data=ratings)  plt.show() |

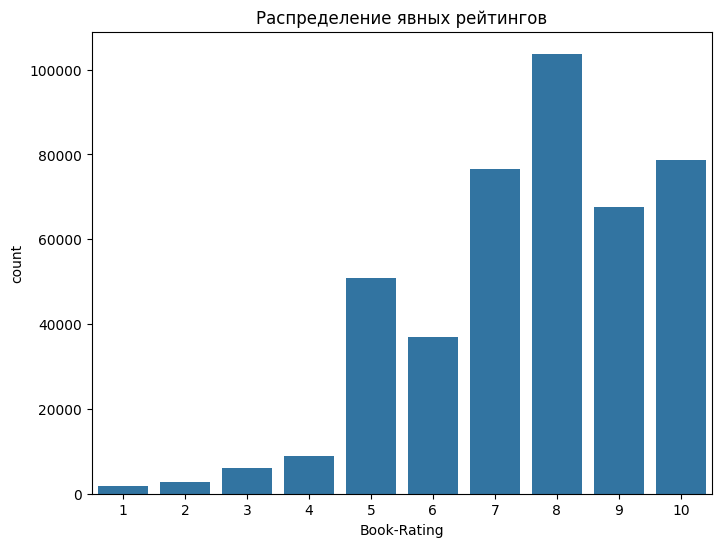


**Рисунок 5 – Информация о данных**

1. Визуализируем

Листинг 9 – Распределение явных рейтингов

|  |
| --- |
| # Распределение явных рейтингов  plt.figure(figsize=(8,6))  data = ratings[ratings['Book-Rating'] != 0]  sns.countplot(x="Book-Rating", data=data)  plt.title("Explicit Ratings")  plt.show() |

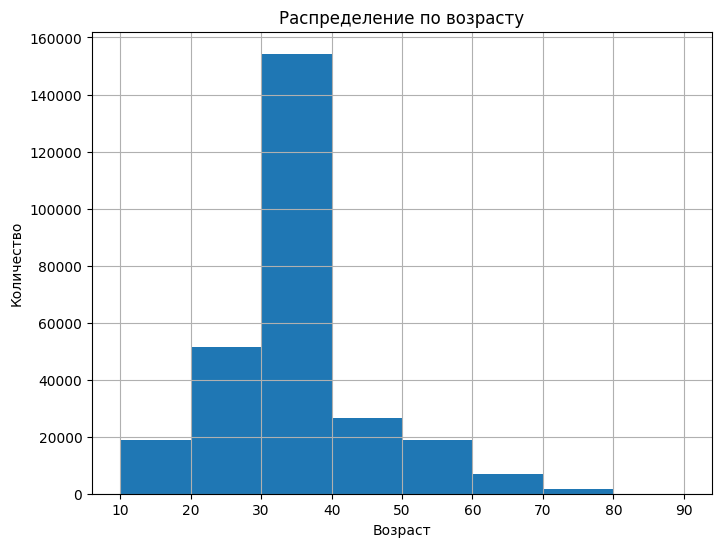


**Рисунок 6 – Информация о данных**

1. Визуализируем

Листинг 10 – Распределение явных рейтингов

|  |
| --- |
| # Распределение возрастов пользователей  plt.figure(figsize=(8,6))  users.Age.hist(bins=[10\*i for i in range(1, 10)])  plt.title('Распределение по возрасту')  plt.xlabel('Возраст')  plt.ylabel('Количество')  plt.show() |

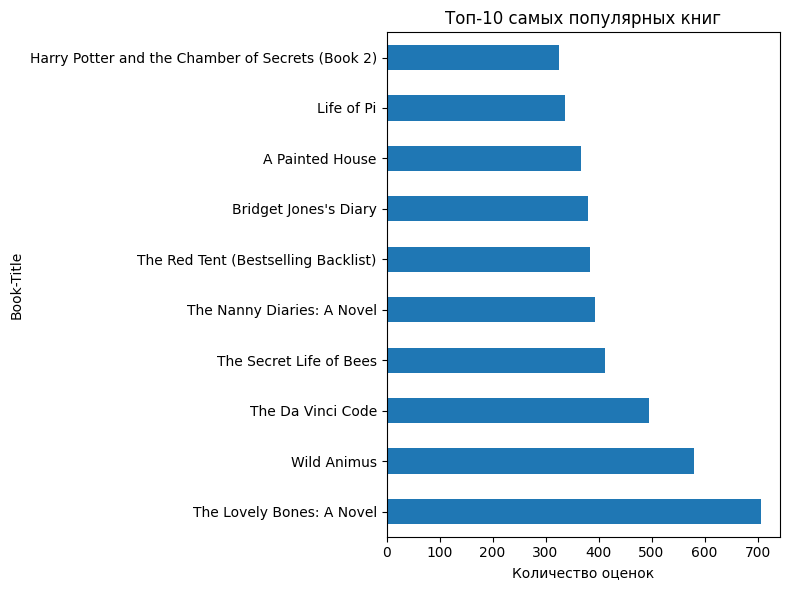


**Рисунок 7 – Информация о данных**

1. Визуализируем

Листинг 11 – Распределение явных рейтингов

|  |
| --- |
| # Топ-10 самых популярных книг  plt.figure(figsize=(8, 6))  top\_books = dataset1['Book-Title'].value\_counts().head(10)  top\_books.plot(kind='barh')  plt.title('Топ-10 самых популярных книг')  plt.xlabel('Количество оценок')  plt.tight\_layout()  plt.show() |



**Рисунок 8 – Информация о данных**

1. Рекомендательная система

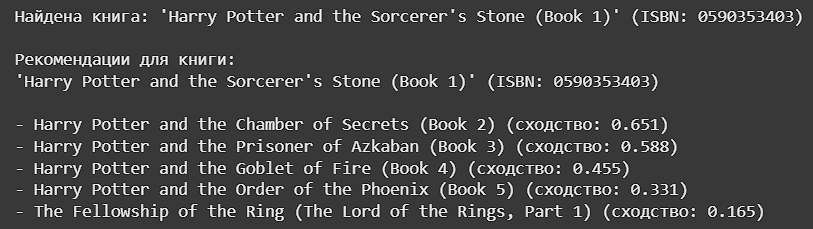
Листинг 12 – Коллаборативная фильтрация

|  |
| --- |
| # Берём только явные рейтинги (не 0)  explicit\_ratings = dataset1.copy()  book\_counts = explicit\_ratings['Book-Title'].value\_counts()  user\_counts = explicit\_ratings['User-ID'].value\_counts()  # книги, оценённые >=50 раз  # пользователей, поставивших >=20 оценок  filtered\_books = book\_counts[book\_counts >= 50].index  filtered\_users = user\_counts[user\_counts >= 20].index  cf\_data = explicit\_ratings[      (explicit\_ratings['Book-Title'].isin(filtered\_books)) &      (explicit\_ratings['User-ID'].isin(filtered\_users))  ].copy()  print(f"Размер данных после фильтрации: {cf\_data.shape}")  # Группируем по ISBN и Book-Title (на случай дубликатов ISBN с разными названиями), удаляю дубликаты оценок одного пользователя одной книги  cf\_data = cf\_data.drop\_duplicates(subset=['ISBN', 'User-ID'])  # Создаём сводную таблицу: строки — книги (ISBN), столбцы — пользователи, значения — рейтинги  book\_user\_matrix = cf\_data.pivot\_table(      index='ISBN',      columns='User-ID',      values='Book-Rating',      fill\_value=0  ) |

1. Рекомендательная система

Листинг 13 – Фильтрация

|  |
| --- |
| # Создаёт матрицу "книга × пользователь", где:  # строки — ISBN книг  # столбцы — ID пользователей  # значения — оценки (0, если не оценивал)  print(f"Матрица: {book\_user\_matrix.shape}")  # Преобразуем в разреженную матрицу  book\_sparse = csr\_matrix(book\_user\_matrix.values)  # Вычисляем косинусное сходство  book\_similarity = cosine\_similarity(book\_sparse)  book\_similarity\_df = pd.DataFrame(      book\_similarity,      index=book\_user\_matrix.index,      columns=book\_user\_matrix.index  )  def get\_book\_title(isbn):      # Возвращает название книги по ISBN из исходного датасета"""      titles = books[books['ISBN'] == isbn]['Book-Title'].values      if len(titles) > 0:          return titles[0]      else:          return f"Unknown Book (ISBN: {isbn})" |

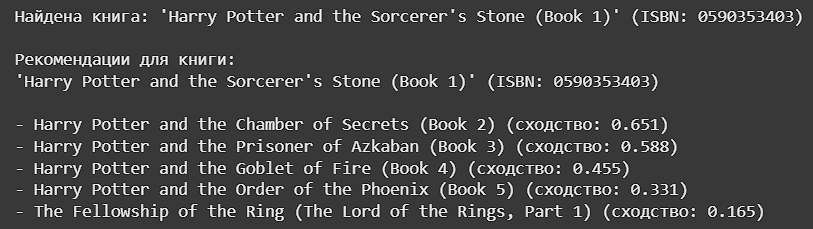


**Рисунок 9 – Информация о данных**

1. Рекомендательная система

Листинг 14 – Фильтрация

|  |
| --- |
| def recommend\_books\_by\_isbn(target\_isbn, n\_recommendations=5):      if target\_isbn not in book\_similarity\_df.index:          print(f"Книга с ISBN {target\_isbn} не найдена в данных для рекомендаций.")          return []      # Получаем сходства для целевой книги      similar\_books = book\_similarity\_df[target\_isbn].sort\_values(ascending=False)      # Исключаем саму книгу (сходство = 1.0)      similar\_books = similar\_books.iloc[1:]      # Берём топ-N      top\_books = similar\_books.head(n\_recommendations)      print(f"\nРекомендации для книги:\n'{get\_book\_title(target\_isbn)}' (ISBN: {target\_isbn})\n")      recommendations = []      for isbn, score in top\_books.items():          title = get\_book\_title(isbn)          print(f"- {title} (сходство: {score:.3f})")          recommendations.append((isbn, title, score))      return recommendations  # Поиск ISBN по названию книги (регистронезависимо, частичное совпадение)  def find\_isbn\_by\_title(title, books\_df, threshold=0.6):      """      Ищет ISBN по названию книги.      Возвращает первый найденный ISBN или None.      """      # Приводим к нижнему регистру для сравнения      title\_lower = title.lower()      matches = books\_df[books\_df['Book-Title'].str.lower().str.contains(title\_lower, na=False)]      if len(matches) == 0:          print(f"Книга с названием '{title}' не найдена.")          return None      # Берём первую найденную (можно улучшить: по автору, году и т.д.)      isbn = matches.iloc[0]['ISBN']      print(f"Найдена книга: '{matches.iloc[0]['Book-Title']}' (ISBN: {isbn})")      return isbn  target\_title = "Harry Potter and the Sorcerer's Stone"  target\_isbn = find\_isbn\_by\_title(target\_title, books)  if target\_isbn:      recommendations = recommend\_books\_by\_isbn(target\_isbn, n\_recommendations=5) |



**Рисунок 10 – Информация о данных**

1. Оценка и интерпретация результатов

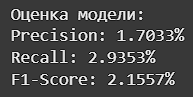
Листинг 15 – Оценка

|  |
| --- |
| def evaluate\_recommendations\_corrected():      train\_data, test\_data = train\_test\_split(cf\_data, test\_size=0.2, random\_state=42)      train\_matrix = train\_data.pivot\_table(          index='ISBN',          columns='User-ID',          values='Book-Rating',          fill\_value=0      )        train\_sparse = csr\_matrix(train\_matrix.values)      train\_similarity = cosine\_similarity(train\_sparse)        hits = 0      total\_recommendations = 0      actual\_positives = 0        for user\_id in test\_data['User-ID'].unique():          user\_test\_books = test\_data[test\_data['User-ID'] == user\_id]['ISBN'].tolist()            user\_train\_books = train\_data[train\_data['User-ID'] == user\_id]['ISBN'].tolist()          if len(user\_train\_books) < 1 or len(user\_test\_books) < 1:              continue            all\_recommendations = set()            for known\_book in user\_train\_books[:3]:              if known\_book in train\_matrix.index:                  similar\_books = pd.Series(                      train\_similarity[train\_matrix.index.get\_loc(known\_book)],                      index=train\_matrix.index                  ).sort\_values(ascending=False)                    recommendations = [                      isbn for isbn in similar\_books.index[1:6]                      if isbn not in user\_train\_books                  ]                  all\_recommendations.update(recommendations)          final\_recommendations = list(all\_recommendations)[:5]            if not final\_recommendations:              continue          hits\_user = len(set(final\_recommendations) & set(user\_test\_books))            hits += hits\_user          total\_recommendations += len(final\_recommendations)          actual\_positives += len(user\_test\_books)  corrected\_precision, corrected\_recall, corrected\_f1 = evaluate\_recommendations\_corrected() |

1. Оценка результатов

Листинг 16 – Оценка

|  |
| --- |
| # Вычисление метрик      precision = hits / total\_recommendations if total\_recommendations > 0 else 0      recall = hits / actual\_positives if actual\_positives > 0 else 0      f1 = 2 \* precision \* recall / (precision + recall) if (precision + recall) > 0 else 0        # print(f"Оценено пользователей: {len(test\_data['User-ID'].unique())}")      print(f"Precision: {precision\*100:.4f}%")      print(f"Recall: {recall\*100:.4f}%")      print(f"F1-Score: {f1\*100:.4f}%")      # print(f"Hit пользователей: {hits}")      # print(f"Всего рекомендаций: {total\_recommendations}")      # print(f"Всего тестовых книг: {actual\_positives}")        return precision, recall, f1  print("Оценка модели:")  corrected\_precision, corrected\_recall, corrected\_f1 = evaluate\_recommendations\_corrected() |



**Рисунок 11 – Оценка модели**

Результат работы:

Данную работу можете увидеть в блокноте Jupyter Notebook.

<https://drive.google.com/file/d/1GB8KUCrJpkU8wkUIw-ifZ81UUjskGLfb/view?usp=sharing>

Вывод:

В ходе выполнения практической работы была успешно разработана и реализована рекомендательная система для книг на основе метода коллаборативной фильтрации. Система использует данные о рейтингах пользователей для формирования персональных рекомендаций.

Список использованных источников и литературы:

1. Ростовцев В.С. Искусственные нейронные сети,   
   Издательство "Лань", 2019. — 216 с. — URL: https://e.lanbook.com/book/122180
2. Араки М. Манга: Машинное обучение,   
   Издательство "ДМК Пресс", 2020. — 214 с. — URL: <https://e.lanbook.com/book/179473>
3. Платонов, А. В. Машинное обучение : учебное пособие для вузов / А. В. Платонов. — Москва : Издательство Юрайт, 2022. — 85 с. — (Высшее образование). — ISBN 978-5-534-15561-7. — Текст : электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/508804