ABDELMALEK ESSAADI UNIVERSITY

Faculty of Sciences and Techniques – Al Hoceima

Module: Python for Data Science
Program: Data Engineering and Data Science (IDDL)
Semester 5

Project Report

Book Recommendation System

Prepared by:
Bouich Mohamed
Aissati Oussama

Supervised by:
Prof. Hayat Routaib

Academic Year: 2025–2026

Contents

Abstract

This report presents the design and implementation of a book recommendation system, developed as a collaborative college project. The system uses content-based filtering with TF-IDF (Term Frequency-Inverse Document Frequency) and cosine similarity to recommend books based on their textual features such as title, genre, and description.

The project follows a modular architecture with separate backend and frontend components. The backend handles data preprocessing, vectorization, and recommendation algorithms, while the frontend, built with Streamlit, provides an interactive interface supporting search, genre filtering, similarity-based recommendations, and random discovery. The application has been deployed online, allowing users to access all features without local setup.

Evaluation shows that the system provides relevant and coherent recommendations efficiently, supports exploration of different book categories, and maintains a smooth interactive experience. This project demonstrates practical application of machine learning techniques in real-world recommendation systems and highlights the benefits of collaborative software development.

1 Introduction

With the growth of digital libraries, users face challenges in selecting relevant books from vast collections. Recommendation systems help by filtering and suggesting items that match user interests. This project focuses on a content-based book recommendation system that relies on item features rather than user interactions.

2 Project Structure and Collaboration

2.1 Repository Organization

```
book-recommendation-system/
|-- data/
|   |-- books.csv
|-- backend/
|   |-- __init__.py
|   |-- data_processing.py
|   |-- recommendations.py
|-- frontend/
|   |-- __init__.py
|   |-- app.py
|-- tests/
|   |-- test_data_processing.py
|   |-- test_recommendations.py
|-- requirements.txt
|-- README.md
|-- report/
|   |-- report.tex
```

2.2 Team Collaboration

2.2.1 Backend Development (Aissati Oussama)

- Data preprocessing and cleaning
- TF-IDF vectorization and cosine similarity
- Recommendation algorithm implementation
- Unit testing for backend functions

2.2.2 Frontend Development (Bouich Mohamed)

- Streamlit user interface
- Interactive components for search, genre, and recommendations
- Backend integration
- User experience optimization

2.3 Version Control

Git was used with:

- Main branch for stable releases
- Feature branches for development
- Pull requests for review
- Conventional commit messages

3 Dataset Description

The project uses a dataset of 100 books for demonstration purposes. The dataset can easily be expanded depending on available RAM. Below is a sample of 5 books from the dataset to illustrate its structure.

Title	**Author**	**Genre**	**Description**	**Rating**	**Pages**	**Year**
To Kill a Mockingbird	Harper Lee	Fiction	A gripping tale of racial injustice and childhood innocence in the Depression-era South	4.8	324	1960
1984	George Orwell	Dystopian	A chilling dystopian novel about totalitarianism and surveillance in a future authoritarian state	4.7	328	1949
Pride and Prejudice	Jane Austen	Romance	A witty romantic novel exploring marriage and social class in Regency England	4.6	432	1813
The Great Gatsby	F. Scott Fitzgerald	Fiction	A portrait of the Jazz Age excess and the American Dream through mysterious millionaire Jay Gatsby	4.4	180	1925
Harry Potter and the Sorcerer's Stone	J.K. Rowling	Fantasy	A young wizard discovers his magical heritage and begins his adventures at Hogwarts School	4.7	309	1997

Table 3.1: Sample of Book Dataset

4 System Architecture

The system has three main layers:

- **Data Layer:** Stores and manages book information
- **Backend Layer:** Implements recommendation algorithms
- **Frontend Layer:** Interactive Streamlit interface

4.1 System Flow

1. User interacts with the frontend
2. Requests are sent to backend modules
3. Backend processes data and computes recommendations
4. Results are returned and displayed

5 Libraries Used

5.1 NumPy

NumPy provides efficient numerical operations and multi-dimensional arrays. Used internally in TF-IDF and cosine similarity computations.

5.2 Pandas

Pandas provides DataFrame structures, reading/writing CSV, and data cleaning functions. Used to manage the book dataset.

5.3 Scikit-learn

Used for TF-IDF vectorization and cosine similarity.

Machine Learning in the Project

- Converts textual data into numerical vectors (TF-IDF)
- Measures similarity using cosine similarity

TF-IDF Explanation

TF-IDF weights words based on frequency and uniqueness, highlighting important terms for content similarity.

5.4 Streamlit

Used to build a web interface with interactive elements: search, dropdowns, buttons, progress bars, and expandable sections.

6 Backend Implementation

The backend handles data processing and recommendation logic. It consists of two main modules: **data_processing.py** and **recommendations.py**.

6.1 Data Processing Module

```
import pandas as pd

def load_data(path="data/books.csv"):
    df = pd.read_csv(path)
    return df

def clean_data(df):
    df = df.drop_duplicates()
    for col in ["title", "author", "genre", "description"]:
        if col in df.columns:
            df[col] = df[col].fillna("")
    return df

def get_unique_genres(df):
    if "genre" not in df.columns:
        return []
    return sorted(df["genre"].dropna().unique())
```

Explanation: This module prepares the dataset for recommendation algorithms. It loads the CSV file, removes duplicate rows, fills missing text fields with empty strings, and provides a list of unique genres for filtering.

6.2 Recommendation Module

The recommendation module implements **multiple strategies**.

6.2.1 Genre-Based Recommendation

```
def recommend_by_genre(df, genre, limit=10):
    results = df[df["genre"].str.contains(genre, case=False, na=
        False)]
    return results.head(limit)
```

Explanation: Returns books from the specified genre. Useful for users who want recommendations within a particular category.

6.2.2 Random Recommendation

```
def recommend_random(df, limit=10):
    return df.sample(n=min(limit, len(df)))
```

Explanation: Provides random book suggestions to encourage exploration and discovery.

6.2.3 Search-Based Recommendation

```
def search_books(df, query, limit=10):
    mask = (df["title"].str.lower().str.contains(query.lower()) |
            df["author"].str.lower().str.contains(query.lower()))
    return df[mask].head(limit)
```

Explanation: Allows searching by book title or author. Case-insensitive string matching ensures flexible query handling.

6.2.4 TF-IDF Similarity Recommendation

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

def build_tfidf_matrix(df):
    vectorizer = TfidfVectorizer(stop_words="english")
    tfidf_matrix = vectorizer.fit_transform(df["description"])
    return tfidf_matrix, vectorizer

def recommend_similar(df, tfidf_matrix, book_title, limit=10):
    match = df[df["title"].str.lower() == book_title.lower()]
    if match.empty:
        return df.head(0)
    idx = match.index[0]
    sims = cosine_similarity(tfidf_matrix[idx], tfidf_matrix).
       flatten()
    similar_indices = sims.argsort()[::-1][1:limit+1]
    return df.iloc[similar_indices]
```

Explanation: Uses **TF-IDF (Term Frequency-Inverse Document Frequency)** vectorization on book descriptions and **cosine similarity** to find books with similar content. This is the core of content-based recommendations.

7 Frontend Implementation

The frontend is a **Streamlit web application** that interacts with the backend to display recommendations. The main file is **app.py**.

7.1 Page Layout and Navigation

```
import streamlit as st

st.set_page_config(
    page_title="Book Recommendation System",
    page_icon=":books:",
    layout="wide"
)

with st.sidebar:
    st.title("Navigation")
    recommendation_method = st.selectbox(
        "Choose Recommendation Method",
        ["Search Books", "By Genre", "Similar Books", "Random Picks
           "]
    )
```

Explanation: Sets up the page configuration and sidebar navigation. Users can select the recommendation type from four available options.

7.2 Search Interface

```
if recommendation_method == "Search Books":
    st.header("Search Books")
    search_query = st.text_input("Enter title or author name:")
    search_button = st.button("Search")

    if search_button and search_query:
        results = search_books(df, search_query)
        if not results.empty:
            for _, row in results.iterrows():
                st.subheader(row['title'])
                st.write(f"Author: {row['author']}")
                st.write(row['description'][:200] + "...")
                st.write(f"Rating: {row['rating']}/5 | Pages: {row
                    ['pages']}")
        else:
            st.warning("No books found. Try a different search term
                .")
```

Explanation: Provides a search box for users to find books by title or author. Displays results with a short description, rating, and page count.

7.3 Genre Interface

```
elif recommendation_method == "By Genre":
    st.header("Browse by Genre")
    selected_genre = st.selectbox("Select a genre:", genres)

    if selected_genre:
        genre_books = recommend_by_genre(df, selected_genre)
        for _, row in genre_books.iterrows():
            st.subheader(row['title'])
            st.write(f"Author: {row['author']}")
            st.write(f"Description: {row['description'][:150]}...")
            st.write(f"Rating: {row['rating']}/5")
            st.divider()
```

Explanation: Allows filtering books by genre. Shows a short description and key metadata for each recommended book.

7.4 Similar Books (TF-IDF)

```
elif recommendation_method == "Similar Books":
    st.header("Find Similar Books")
    selected_book = st.selectbox("Choose a book:", df['title'].
        tolist())

    if selected_book:
        book_info = df[df['title'] == selected_book].iloc[0]
        st.write(f"Author: {book_info['author']}")
        st.write(f"Genre: {book_info['genre']}")
        st.write(f"Rating: {book_info['rating']}/5 | Year: {
           book_info['year']}")

        similar_books = recommend_similar(df, tfidf_matrix,
           selected_book)
        for _, row in similar_books.iterrows():
            st.write(f"- {row['title']} by {row['author']} ({row['
               genre']})")
```

Explanation: Shows similar books using TF-IDF cosine similarity on descriptions. Users select a book and get content-based recommendations.

7.5 Random Picks Interface

```
else:  # Random Picks
    st.header("Discover Random Books")
    if st.button("Show Random Recommendations"):
        random_books = recommend_random(df, 5)
        for _, row in random_books.iterrows():
            st.write(f"{row['title']} by {row['author']} ({row['
               genre']})")
```

Explanation: Provides random book suggestions to encourage exploration. Useful for discovering books without specific criteria.

8 Implementation Details

8.1 System Integration

```python
# Integration in frontend (app.py)
import sys
sys.path.append('../backend')

from backend.data_processing import load_data, clean_data
from backend.recommendations import (
    recommend_by_genre,
    recommend_random,
    search_books,
    build_tfidf_matrix,
    recommend_similar
)

# Load and prepare data
df = load_data("data/books.csv")
df = clean_data(df)

# Build TF-IDF matrix once (cached for performance)
@st.cache_resource
def get_tfidf_matrix():
    return build_tfidf_matrix(df)

tfidf_matrix, vectorizer = get_tfidf_matrix()
```

Explanation: The frontend app imports the backend modules for data processing and recommendations. Data is loaded and cleaned once, and the TF-IDF matrix is cached to avoid recomputation, improving response time.

8.2 Data Flow and Request Handling

1. User selects a recommendation method in the Streamlit frontend.
2. Frontend calls the corresponding backend function, such as `recommend_similar`, `recommend_by_genre`, `recommend_random`, or `search_books`.
3. Backend processes the request using precomputed TF-IDF vectors or filters for genres, random picks, or search queries.
4. Backend returns a Pandas DataFrame with the recommended books.
5. Frontend displays the results dynamically with descriptions, ratings, and other metadata.

Explanation: This modular approach separates concerns: the frontend handles user interaction and display, while the backend focuses on data and recommendation logic. This improves maintainability and scalability.

8.3 Performance Optimization

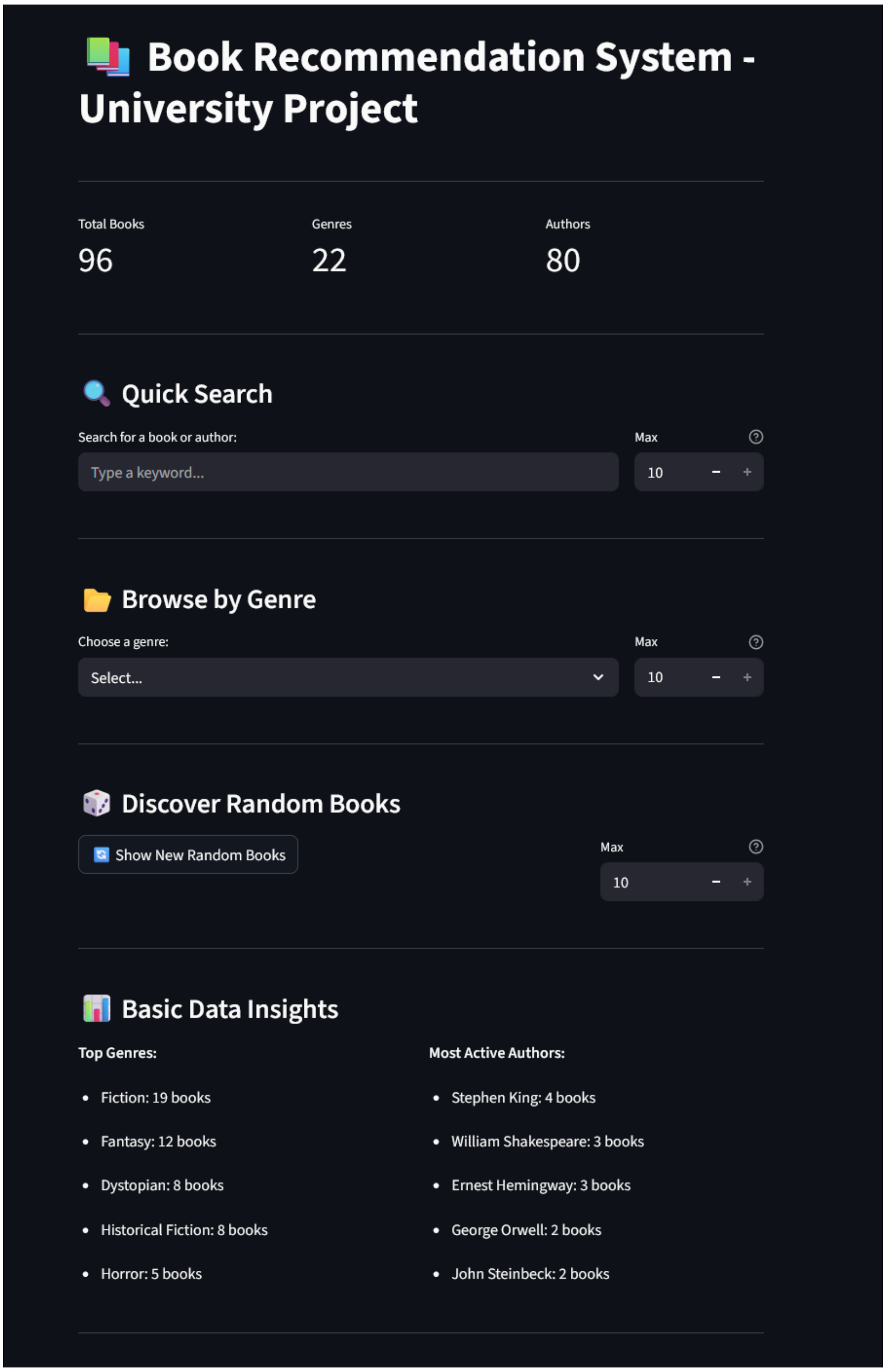
- **Caching:** `st.cache_resource` prevents redundant computations for TF-IDF vectors.
- **Efficient Searching:** Pandas string operations with case-insensitive matching optimize search performance.
- **Memory Management:** TF-IDF matrix is computed once and reused for all similarity recommendations.
- **Lazy Loading:** Data is loaded only when needed.
- **Frontend Responsiveness:** Pagination and collapsible sections prevent interface slowdown for large datasets.

9 Results and Evaluation

9.1 Deployment

The project has been deployed online using Streamlit and is accessible at: https://book-rec-system.streamlit.app/. Users can explore all recommendation methods directly through the web interface without local setup.

9.2 User Interface



Book Recommendation System - University Project

Total Books	Genres	Authors
96	22	80

Quick Search

Search for a book or author: Type a keyword... Max 10

Browse by Genre

Choose a genre: Select... Max 10

Discover Random Books

Show New Random Books Max 10

Basic Data Insights

Top Genres:

- Fiction: 19 books
- Fantasy: 12 books
- Dystopian: 8 books
- Historical Fiction: 8 books
- Horror: 5 books

Most Active Authors:

- Stephen King: 4 books
- William Shakespeare: 3 books
- Ernest Hemingway: 3 books
- George Orwell: 2 books
- John Steinbeck: 2 books

Figure 9.1: Streamlit Application Interface (deployed version)

9.3 System Performance and Recommendation Quality

The system was tested on the 100-book demo dataset. All recommendation methods respond quickly, providing a smooth and interactive user experience. Genre-based filtering and random recommendations are nearly instantaneous, while TF-IDF similarity calculations are slightly slower due to vector computations but remain responsive. Recommendations are generally relevant to the selected genre or book, and the system supports exploration and discovery of books effectively.

10 Challenges and Solutions

10.1 Technical

- Missing or inconsistent data $\rightarrow$ Data cleaning and normalization
- TF-IDF computation $\rightarrow$ Sparse matrix and caching
- UI responsiveness $\rightarrow$ Pagination and caching

10.2 Collaboration

- Integration issues $\rightarrow$ Defined clear API-like contracts
- Version conflicts $\rightarrow$ requirements.txt with fixed versions

11 Discussion

Content-based filtering works well for small datasets. Limitations: no personalization, small dataset, no user feedback. Future improvements: hybrid filtering, user profiles, scalability, advanced filtering by rating/year/pages.

12 Future Improvements

12.1 System Enhancements

- Hybrid content + collaborative filtering
- User accounts and preference history
- Sentiment analysis from reviews
- Advanced filtering (rating, year, pages)

12.2 Interface Enhancements

- Book covers from APIs
- User rating submissions
- Reading lists creation
- Mobile optimization

13 Conclusion

The project demonstrates a functional book recommendation system using TF-IDF and cosine similarity. Modular architecture allows future extensions. Collaboration split responsibilities effectively, producing a maintainable system. Practical ML application in real-world recommendations is illustrated.

References

- Jannach, D. et al., *Recommender Systems: An Introduction.*
- Manning, C., Raghavan, P., & Schutze, H., *Introduction to Information Retrieval* (TF-IDF explanation)
- Scikit-learn Documentation: https://scikit-learn.org/
- Streamlit Documentation: https://docs.streamlit.io/
- Pandas Documentation: https://pandas.pydata.org/
- NumPy Documentation: https://numpy.org/
- GitHub Repository: Book Recommendation System. https://github.com/0utc/book-recommendation-system