Dr. D. Y. PATIL VIDYAPEETH, PUNE
(Deemed to be University)

DR. D. Y. PATIL SCHOOL OF SCIENCE AND TECHNOLOGY

TATHAWADE, PUNE

A Mini- Project Report on

Book Recommendation System

SUBMITTED BY:

NAME OF STUDENT	ROLL NUMBER
1. Neha Mahajan	BTAI-26
2. Jasmin Kaur Randhawa	BTAI-16
3. Kumkum Kumari Singh	BTAI-25

GUIDED BY:

Dr. Mily Lal

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

ACADEMIC YEAR 2024-2025

Dr. D. Y. PATIL VIDYAPEETH, PUNE
(Deemed to be University)

DR. D. Y. PATIL SCHOOL OF SCIENCE AND TECHNOLOGY

TATHAWADE, PUNE

CERTIFICATE

This is to certify that the Mini- Project Report entitled

"Book Recommendation System "

is a bonafide work carried out by Ms **Jasmin Kaur Randhawa** under the supervision of **Dr. Mily Lal** and it is submitted towards the partial fulfillment of the requirement Foundations of Data Science.

Dr. Mily Lal
Project Guide

Dr Mnisha Bhende
Director I/C

ARTIFICIAL INTELLIGENCE & DATASCIENCE

ACADEMIC YEAR 2024-2025

Dr. D. Y. PATIL VIDYAPEETH, PUNE
(Deemed to be University)

DR. D. Y. PATIL SCHOOL OF SCIENCE AND TECHNOLOGY

TATHAWADE, PUNE

CERTIFICATE

This is to certify that the Mini- Project Report entitled

"Book Recommendation System "

is a bonafide work carried out by Ms **Neha Mahajan** under the supervision of **Dr. Mily Lal** and it is submitted towards the partial fulfillment of the requirement Foundations of Data Science.

Dr. Mily Lal
Project Guide

Dr. Manisha Bhende
Director I/C

ARTIFICIAL INTELLIGENCE & DATASCIENCE

ACADEMIC YEAR 2024-2025

Dr. D. Y. PATIL VIDYAPEETH, PUNE
(Deemed to be University)

DR. D. Y. PATIL SCHOOL OF SCIENCE AND TECHNOLOGY

TATHAWADE, PUNE

CERTIFICATE

This is to certify that the Mini- Project Report entitled

"Book Recommendation System "

is a bonafide work carried out by Ms **Kumkum Kumari Singh** under the supervision of **Dr. Mily Lal** and it is submitted towards the partial fulfillment of the requirement Foundations of Data Science.

Dr. Mily Lal
Project Guide

Dr. Manisha Bhende
Director I/C

ARTIFICIAL INTELLIGENCE & DATASCIENCE

ACADEMIC YEAR 2024-2025

ABSTRACT

The rapid growth of the online book market has resulted in an overwhelming amount of information, making it difficult for users to find books that match their preferences. This project aims to develop a personalized book recommendation system that takes into account user behavior, preferences, and book attributes to suggest relevant books. By utilizing a dataset containing user ratings, book details, and demographic data, the system will employ methods such as collaborative filtering, content-based filtering, and hybrid approaches to enhance recommendation accuracy. The objective is to create a system that improves user experience, minimizes decision fatigue, and assists users in discovering new books. Key stakeholders in this project include e-commerce platform owners, consumers seeking tailored recommendations, and developers working on recommendation technologies. The system will be evaluated based on its ability to deliver relevant, accurate, and engaging book suggestions.

Keywords:- book characteristics, collaborative filtering, content-based filtering, hybrid approaches, Personalized book recommendation system, User preferences.

INDEX

List of Figures

List of tables

Chapter 1

INTRODUCTION

In the present day, there is no lack of content. Different kinds of data can be generated and accessed in different ways. When there is a wide range of content available, people find it difficult to know what to access for their needs and interests [1] .Typically, people make their decision on either their personal observations or other people's experiences [2]. Recommendation systems are tools designed to offer personalized suggestions to users considering their preferences and behavior. They are frequently used on websites like Netflix and YouTube, where they provide movie or video recommendations based on users' past viewing preferences. Amazon makes product recommendations by looking at a user's past purchases and comparing them to those of other people who have similar tastes [3]. There are two types of recommender systems: content-based filtering (CB) and collaborative filtering (CF). With the help of patterns, viewpoints, and data sources, CF enables a system to make recommendations. CB evaluates elements according on the content that is available and the user profile. However, CB's nature is to identify similar objects that the user is familiar with and use that information to suggest other items that are similar but distinct in their own right [4]. Providing recommendations to users is the main objective of a recommender system. In addition to delivering relevant suggestions, recommender systems effectively handle the problem of information overload. Most recommender systems now in use merely aim to suggest the most suitable material based on the user's search and contextual information. For instance, the systems neglect to account for time or location. Nonetheless, contextual data and personalization elements are also incorporated into research by the contemporary recommendation systems currently in use [5].

1.1 PROBLEM STATEMENT

Users struggle to find relevant books that match their interests. Traditional search engines and filtering tools are insufficient for providing personalized recommendations. This project addresses the problem by designing a personalized book recommendation system that helps users discover books aligned with their preferences. The key stakeholders include e-commerce platform owners aiming to improve user engagement, consumers who want personalized book suggestions, and developers working to refine recommendation systems. By solving this issue, the project aims to enhance the user experience on e-commerce platforms and increase sales.

1.2 OBJECTIVE

The objectives of our project is

- To develop a recommender system that provides precise recommendations for books.
- To create a system that updates user profiles in real time to accommodate users' shifting interests.
- Applying different calculation techniques into practice to enhance the quality of recommendations.

- To use suitable evaluation methods to determine whether the system's recommendations are accurate
- To create a system that updates user profiles in real time to reflect users' shifting interests

1.3 SCOPE

Our project's scope entails a thorough examination and analysis of current recommender and profiler systems in order to determine their advantages and disadvantages. Our goal is to create an interactive user profiling algorithm for a book recommendation system by expanding on this research. Based on user choices and behavior, this system will be built to offer tailored recommendations. Verification and monitoring of the recommender system will be a crucial component of the project in order to guarantee its precision, applicability, and adaptability to changing user preferences. In order to maximize system performance and provide users with a comprehensive recommendation experience, the project also integrates a number of recommendation algorithms, including collaborative filtering, content-based filtering, and hybrid approaches.

1.4 SYSTEM ARCHITECTURE

The flowchart for the Book Recommendation System shows how the BaselineOnly algorithm is used to generate suggestions. When a user registers in using a web or app platform, the process starts at the user interface. The user's ID is sent to the backend by the system upon login. After retrieving the user's rating information, the recommendation engine uses the BaselineOnly algorithm, which takes into account user and item biases in addition to global averages to estimate ratings for books the user hasn't seen. After predictions are generated, the algorithm retrieves information about the most anticipated books from the books database. To give the user a personalized book suggestion experience, these suggestions are sorted, usually by anticipated rating, and the top-N results are shown back to them on the interface.

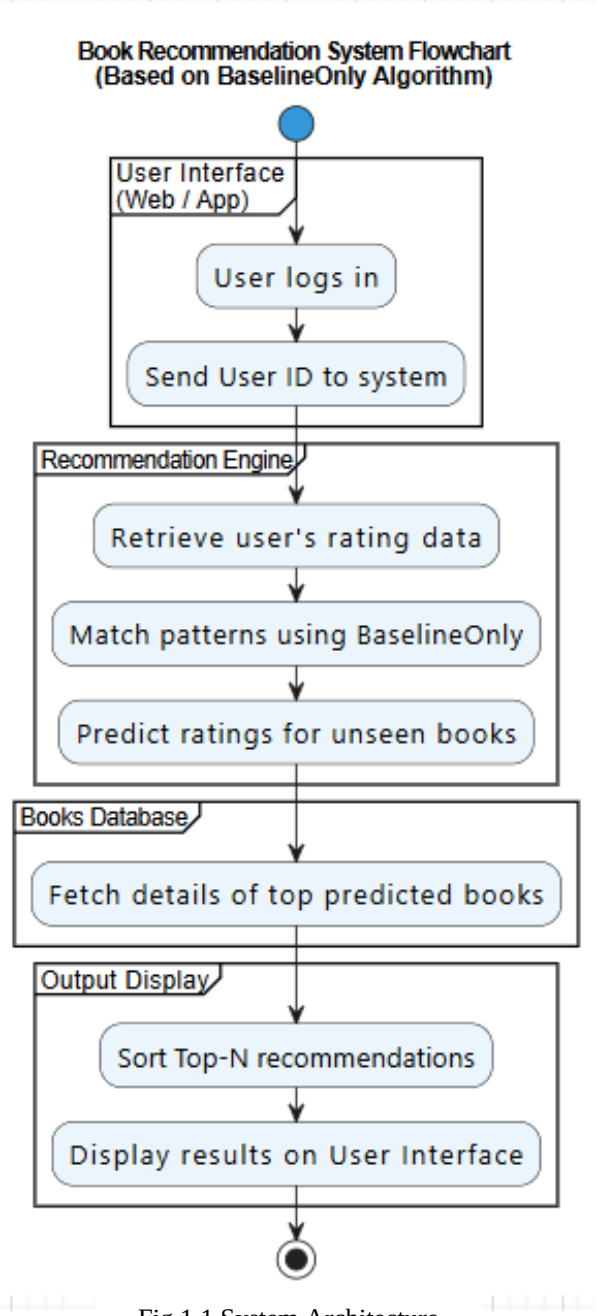

Fig 1.1 System Architecture

Chapter 2

DATA COLLECTION & PREPROCESSING

2.1 DATASET

The dataset used for this project is taken from Kaggle and consists of three key files: users.csv, ratings.csv, and books.csv. These datasets provide all the necessary information to build and evaluate a personalized book recommender system.

users.csv: Contains anonymized user IDs with demographic data like location and age, helping create user profiles for targeted recommendations.

ratings.csv: Includes user ratings (1-10) for books, used for collaborative filtering to suggest books based on similar user preferences.

books.csv: Provides book details (title, author, genre, etc.), used for content-based filtering to recommend books with similar characteristics to ones users have liked.

These datasets are highly relevant because they contain user behavior, book attributes, and interaction data, which are critical for building a robust and personalized recommendation system. By leveraging these datasets, the system can generate tailored suggestions that align with user preferences, ultimately enhancing the shopping experience and reducing the overload of book options.

Link : https://www.kaggle.com/datasets/arashnic/book-recommendation-dataset

2.2 DATA PREPROCESSING

Data preprocessing is the process of transforming raw data into a format that is easier to analyze. This process can include cleaning steps, such as handling missing values or smoothing noisy data.

I. PERFORM DATA CLEANING

- **Importing Libraries :**

```
# Install required library
!pip install scikit-surprise

# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# Data handling and visualization
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Statistical analysis and preprocessing
from scipy.stats import norm, stats
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import (
    roc_auc_score, confusion_matrix, accuracy_score, f1_score,
    roc_curve, classification_report, precision_score, recall_score)

# Cosine similarity and sparse matrices
from sklearn.metrics.pairwise import cosine_similarity
from scipy.sparse import csr_matrix
from sklearn.neighbors import NearestNeighbors

# Surprise library for recommendation systems
from surprise import Reader, Dataset, SVD, SVDpp, NMF, SlopeOne, CoClustering, accuracy
from surprise.model_selection import cross_validate, train_test_split
from surprise import accuracy

# Other utilities
from collections import defaultdict
import ast
import matplotlib
```

Fig 2.1 Importing Libraries

- **Load the data :** To load the user, book, and rating datasets, the pandas library is used. These databases offer the crucial details needed to produce book recommendations. When data is successfully loaded into dataframes, a success message is displayed.

Input :

```
# Importing necessary libraries
import pandas as pd
# Load datasets
users_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Users.csv"
books_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Books.csv"
ratings_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Ratings.csv"

# Load datasets
df_users = pd.read_csv(users_path)
df_books = pd.read_csv(books_path)
df_ratings = pd.read_csv(ratings_path)

# Print confirmation
print("Datasets Loaded Successfully!")
```

Fig 2.2 Code to Load Data

Output :

```
Datasets Loaded Successfully!
```

Fig 2.3 Code to Load Data

- **Checking for Missing Values :** Identify missing values and decide on appropriate strategies (removal, imputation, etc.).

Input :

```
print("Missing values in books data:")
# Check for missing values
print("Missing values in Users data:")
print(df_users.isnull().sum())

print("\nMissing values in Books data:")
print(df_books.isnull().sum())

print("\nMissing values in Ratings data:")
print(df_ratings.isnull().sum())
```

Fig 2.4 Code to check missing values

Output :

```
Missing values in books data:
Missing values in Users data:
User-ID          0
Location         0
Age         110762
dtype: int64

Missing values in Books data:
ISBN                 0
Book-Title           0
Book-Author          2
Year-Of-Publication  0
Publisher            2
Image-URL-S          0
Image-URL-M          0
Image-URL-L          3
dtype: int64

Missing values in Ratings data:
User-ID      0
ISBN         0
Book-Rating  0
dtype: int64
```

Fig 2.5 Output for missing values

- **Handling Missing Values :**

Input :

```
# Check how many are missing before
missing_before = df_users['Age'].isna().sum()

# Calculate median age (ignoring NaNs)
median_age = int(df_users['Age'].median(skipna=True))

# Fill NaN with median
df_users['Age'].fillna(median_age, inplace=True)

# Check how many are missing after
missing_after = df_users['Age'].isna().sum()

print(f"🧮 Filled missing 'Age' values with median: {median_age}")
print(f"✅ Missing 'Age' values after filling: {missing_after}")
```

Fig 2.6 Code to fill missing values

Output :

```
🧮 Filled missing 'Age' values with median: 32
✅ Missing 'Age' values after filling: 0
```

Fig 2.7 Output of filling missing values

- **Remove Duplicates :** Detect and eliminate duplicate entries.

Input (Users Dataset) :

```
df_users = pd.read_csv(users_path)

# Check for duplicates
print(f"Duplicates in users data: {df_users.duplicated().sum()}")
```

Fig 2.8 Code to check duplicates in users data

Output :

```
Duplicates in users data: 0
```

Fig 2.9 Output of checking duplicates in users data

Input (Books Data) :

```
# Remove duplicates
df_books = df_books.drop_duplicates()

# Check for duplicates
print(f"Duplicates in books data: {df_books.duplicated().sum()}")
```

Fig 2.10 Code to check duplicates in books data

Output :

```
Duplicates in books data: 0
```

Fig 2.11 Output of checking duplicates in books data

Input (Ratings data) :

```
df_ratings = pd.read_csv(ratings_path)

# Check for duplicates
print(f"Duplicates in ratings data: {df_ratings.duplicated().sum()}")
```

Fig 2.12 Code to check duplicates in Ratings data

Output :

```
Duplicates in ratings data: 0
```

Fig 2.13 Output of checking duplicates in ratings data

- **Correct Inconsistent Formats :** Ensure uniformity in date formats, categorical values, numerical precision, etc.

Input :

```
original_years = df_books['Year-Of-Publication'].copy()

# Convert non-numeric values to NaN
df_books['Year-Of-Publication'] = pd.to_numeric(df_books['Year-Of-Publication'], errors='coerce')

converted_count = df_books['Year-Of-Publication'].notna().sum()
total_count = len(df_books)
print(f"🔚 Converted to numeric: {converted_count} out of {total_count} entries")

# Replace weird years (like 0 or > 2025) with NaN
invalid_years = df_books.loc[(df_books['Year-Of-Publication'] < 1000) |
                             (df_books['Year-Of-Publication'] > 2025), 'Year-Of-Publication']
print(f"⚠️ Found {len(invalid_years)} invalid years (before 1000 or after 2025)")

df_books.loc[(df_books['Year-Of-Publication'] < 1000) |
             (df_books['Year-Of-Publication'] > 2025), 'Year-Of-Publication'] = pd.NA

# Fill with median year
median_year = int(df_books['Year-Of-Publication'].median(skipna=True))
df_books['Year-Of-Publication'].fillna(median_year, inplace=True)

# Count how many were filled
filled_count = df_books['Year-Of-Publication'].isna().sum()
print(f"🧹 Filled {filled_count} missing or invalid 'Year-Of-Publication' entries with median: {median_year}")

print("✅ 'Year-Of-Publication' column cleaned and standardized.")
```

Fig 2.14 Code to Convert Year of Publication to Numeric

Output :

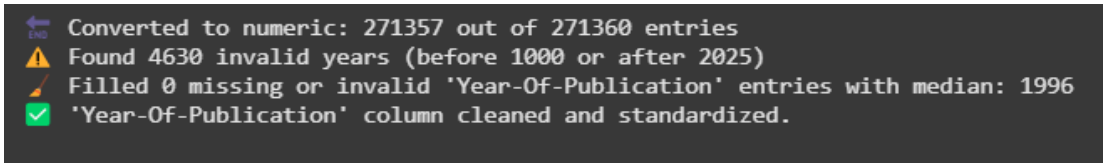

Fig 2.15 Output of Converting Year of Publication to Numeric

- **Detect Outliers :** Use statistical techniques (e.g., IQR, Z-score) to identify and address extreme values.

Input (Users Data):

```
Q1 = df_users['Age'].quantile(0.25)
Q3 = df_users['Age'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

age_outliers = df_users[(df_users['Age'] < lower_bound) | (df_users['Age'] > upper_bound)]

print("📌 Age Outliers:")
print(age_outliers[['User-ID', 'Age']])
print(f"Total Age Outliers: {len(age_outliers)}\n")
```

Fig 2.16 Code to detect outliers in users data

Output :

```
📌 Age Outliers:
        User-ID    Age
220         221   79.0
689         690   80.0
957         958   78.0
1147       1148   79.0
1288       1289  103.0
...         ...    ...
278301   278302  104.0
278317   278318   77.0
278348   278349   76.0
278412   278413   76.0
278471   278472   81.0

[1084 rows x 2 columns]
Total Age Outliers: 1084
```

Fig 2.17 Output of detecting outliers in users data

Input (Books Data):

```
# Year of Publication outliers
Q1 = df_books['Year-Of-Publication'].quantile(0.25)
Q3 = df_books['Year-Of-Publication'].quantile(0.75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

year_outliers = df_books[(df_books['Year-Of-Publication'] < lower_bound) | (df_books['Year-Of-Publication'] > upper_bound)]

print("📌 Year of Publication Outliers:")
print(year_outliers[['ISBN', 'Book-Title', 'Year-Of-Publication']])
print(f"Total Year Outliers: {len(year_outliers)}\n")
```

Fig 2.18 Code to detect outliers in books data

Output :

```
📌 Year of Publication Outliers:
             ISBN                                         Book-Title  \
75     0451625889                                         The Prince
196    3499110695                                Neun ErzÃ?Â¤hlungen.
241    0803251718                                        Crazy Horse
310    0520011171                          Sappho: A New Translation
388    0156528207                                  The Little Prince
...           ...                                                ...
271180 0822502763  Antonyms: Hot and Cold and Other Words That Ar...
271181 0877830592                                        Quetico Wolf
271250 073943828X  Murder at the Manor (Mystery Guild Lost Classi...
271293 0226751260  The Jack-Roller: A Delinquent Boy's Own Story ...
271301 0843101083                  Off-The-Wall (Mad Libs, No. 6)

       Year-Of-Publication
75                  1952.0
196                 1968.0
241                 1961.0
310                 1958.0
388                 1968.0
...                    ...
271180              1972.0
271181              1972.0
271250              1929.0
271293              1966.0
271301              1970.0

[4465 rows x 3 columns]
Total Year Outliers: 4465
```

Fig 2.19 Output of detecting outliers in books data

- **Handling Outliers :**

Input (Users Data):

```
# Drop the 'Age' column
df_users.drop(columns=['Age'], inplace=True)

print("🗑️ Dropped 'Age' column from users dataset.")
```

Fig 2.20 Code to handle outliers in users data

Output :

```
🗑️ Dropped 'Age' column from users dataset.
```

Fig 2.21 Output of handling outliers in books data

Input (Books Data):

```
# Keeping only rows where Year-Of-Publication is >= 1960
df_books = df_books[df_books['Year-Of-Publication'] >= 1960]

print("✅ Removed books published before 1960.")
print(f"📚 Remaining books: {df_books.shape[0]} rows")
df_books = df_books[df_books['Year-Of-Publication'] >= 1960]
```

Fig 2.22 Code to handle outliers in books data

Output :

```
✅ Removed books published before 1960.
📚 Remaining books: 270444 rows
```

Fig 2.23 Output of handling outliers in books data

Chapter 3

MODEL SELECTION & TRAINING

3.1 Feature Engineering :

Feature engineering involved creating user profiles based on their historical ratings and demographic information. For the books, features such as genre, author, and year of publication were used to calculate similarity with other books.

- **Book Rating Distribution (Histogram)**

The distribution of book ratings on a scale from 0 to 10 is displayed by the histogram. The majority of ratings fall between 0 and 10, with ratings between 5 and 10 having lesser frequencies. This suggests that a smaller sample of books have higher ratings, whereas the majority have either no rating at all or a poor rating.

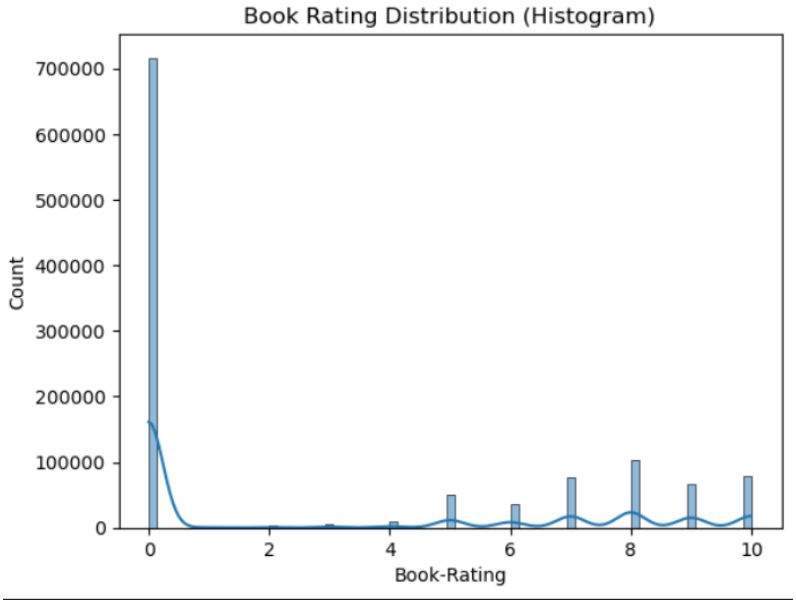

Fig 3.1 Histogram of book rating distribution

- **Book Rating**

The distribution of book ratings from 1 to 10 is shown in the chart. The majority of ratings fall between 7 and 10, with 8 being the most common. The fact that ratings under five are much less frequent indicates that readers are generally giving positive reviews.

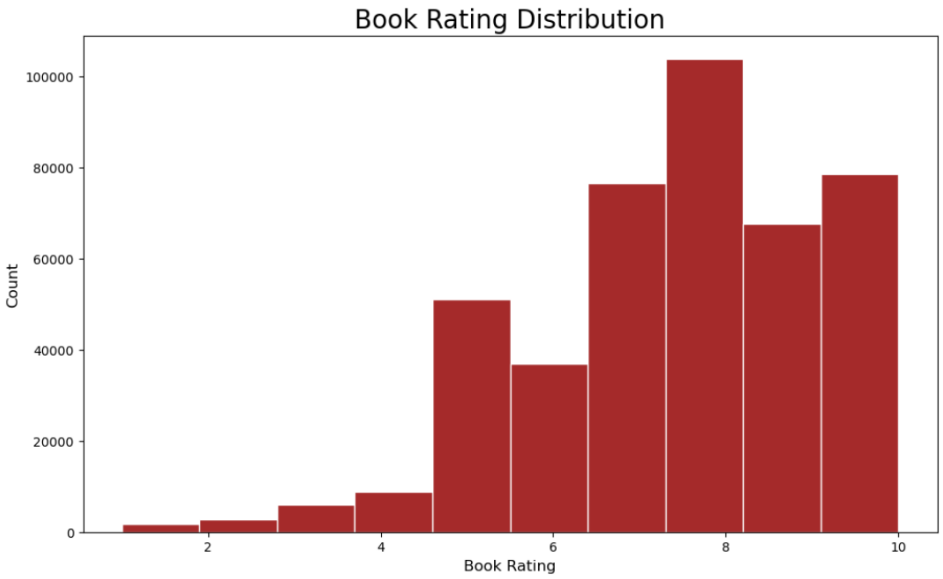

Fig 3.2 Barplot of book rating distribution

- **Number of Ratings per User VS Average Rating**

This scatter plot illustrates how the number of ratings given by a user relates to their average rating. The majority of users have submitted under 200 ratings. As users rate more books, their average ratings generally decline. This pattern implies that highly active users might rate books more strictly or consistently.

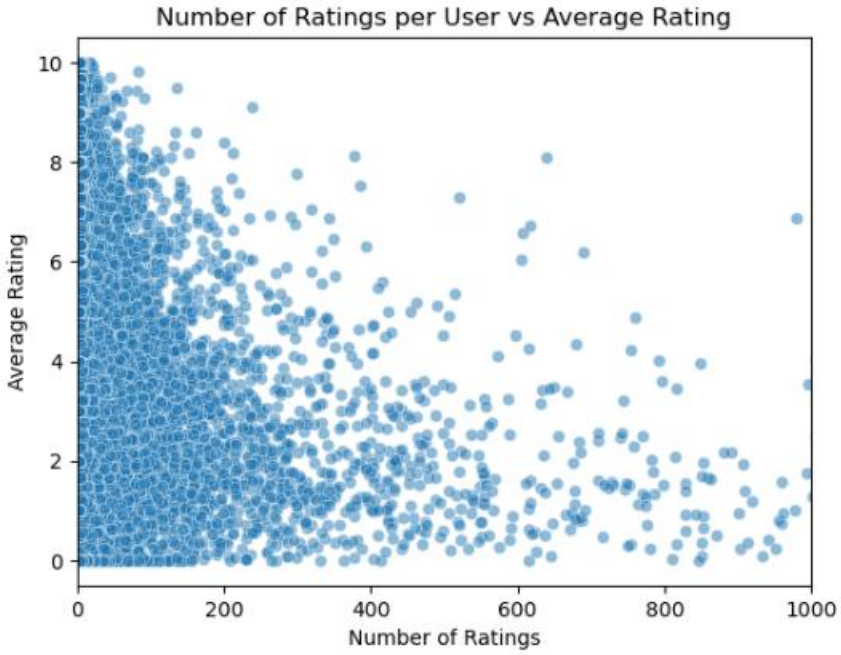

Fig 3.3 Number of Rating per user VS Average rating

- **Top 10 Most Read Books**

The chart shows the top 10 most read books based on the number of ratings. "Wild Animus" leads with the highest count, followed by "The Lovely Bones" and "The Da Vinci Code." Most books on the list received between 600 and 900 ratings. This highlights the most popular titles among readers.

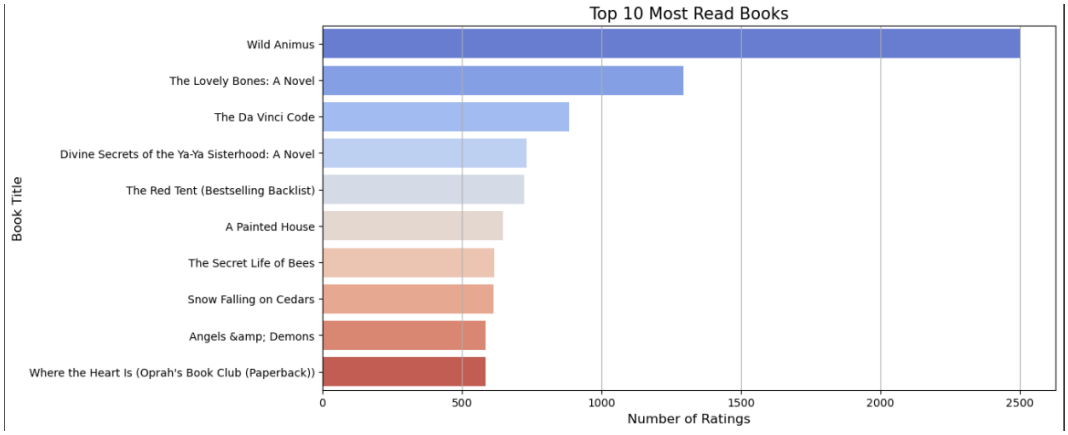

Fig 3.4 Top 10 most read books

- **Top 10 Authors**

The top 10 authors are shown in the bar chart according to the quantity of books they have authored. With more than 600 works, Agatha Christie is the most prolific author, followed by Stephen King and William Shakespeare. The substantial literary accomplishments of all listed authors are demonstrated by the fact that they have authored over 300 volumes.

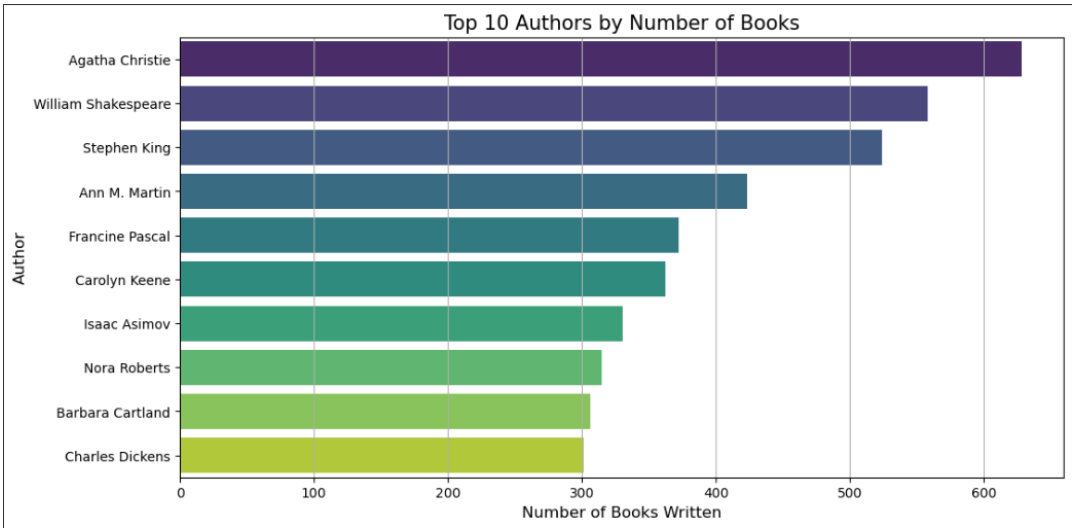

Fig 3.5 Top 10 Authors

- **Value Count of Year Of Publication**

The distribution of book publications over time is depicted in the bar chart. From the 1960s until the early 2000s, the number of books published increased steadily, reaching a peak between 2002 and 2004. Following that, there is a discernible decline, which could be the result of either missing data or fewer entries for recent years.

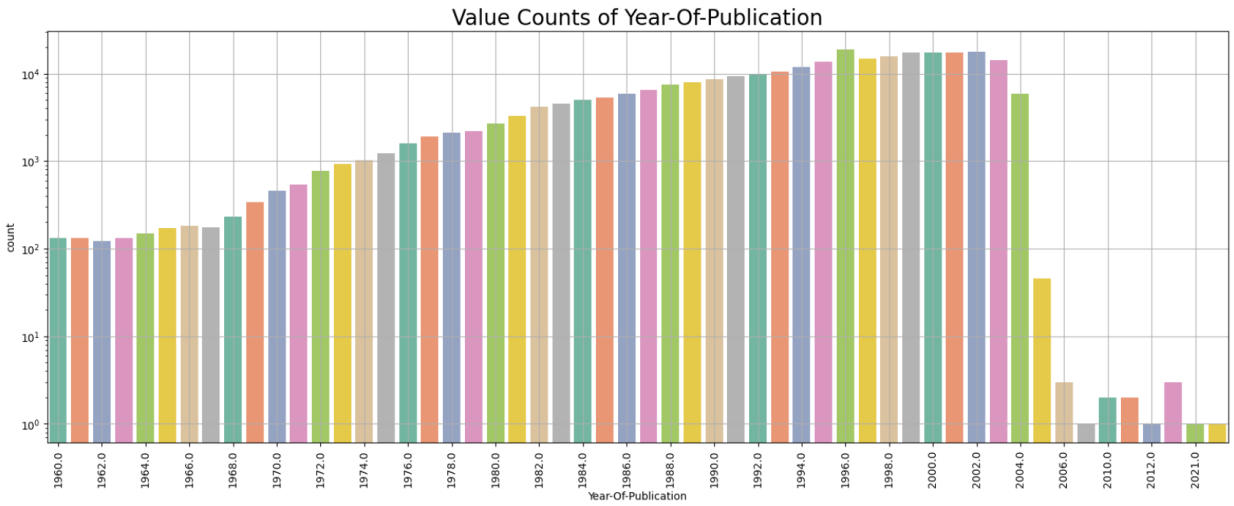

Fig 3.6 Value Count of Year Of Publication

- **Top 10 Publisher**

Based on the quantity of books released, the top 10 publishers are displayed in the bar chart. With a significant lead, Harlequin is followed by Pocket and Silhouette. The publication quantities of the other publishers, such as Scholastic and Penguin Books, are comparatively comparable. This suggests that a small number of significant people are producing the majority of the books.

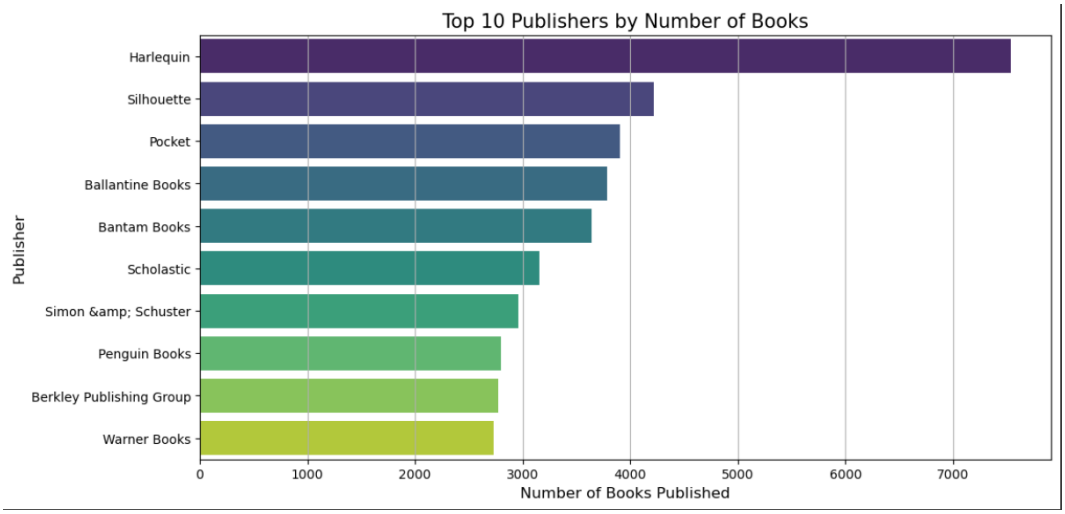

Fig 3.7 Top 10 Publishers

- **Top 10 Country**

The top 10 countries' distribution by percentage of book publications is shown in the pie chart. With 57% of all publications, the USA leads the field, with Canada, the UK, and Germany following with lower contributions.

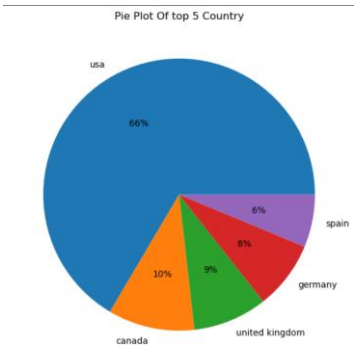

Fig 3.8 Top 10 Country

➢ Training and Testing Datasets

The dataset was divided into training and testing sets using an 80-20 split in order to assess these models' performance:

- **Training Set (80%):** used to train the algorithms that provide recommendations.
- **Testing Set (20%):** Used to assess the trained models' prediction accuracy.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Loaded datasets
ratings_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Ratings.csv"
users_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Users.csv"
books_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Books.csv"

ratings_df = pd.read_csv(ratings_path)
users_df = pd.read_csv(users_path)
books_df = pd.read_csv(books_path)
users_df.rename(columns={"UserID": "User-ID"}, inplace=True)
ratings_df.rename(columns={"UserID": "User-ID"}, inplace=True)

print("📌 Performing 80% Training / 20% Testing Split for All Datasets...\n")
# Ratings Dataset Split
target_col_ratings = 'Book-Rating'
X_ratings = ratings_df.drop(target_col_ratings, axis=1)
y_ratings = ratings_df[target_col_ratings]

X_train_r, X_test_r, y_train_r, y_test_r = train_test_split(
    X_ratings, y_ratings, test_size=0.2, random_state=42)
print("✅ Ratings dataset split complete.")
# Users Dataset Split
X_train_u, X_test_u = train_test_split(users_df, test_size=0.2, random_state=42)
print("✅ Users dataset split complete.")
# Books Dataset Split
X_train_b, X_test_b = train_test_split(books_df, test_size=0.2, random_state=42)
print("✅ Books dataset split complete.")
```

Fig 3.9 Code for train & test dataset

Output :

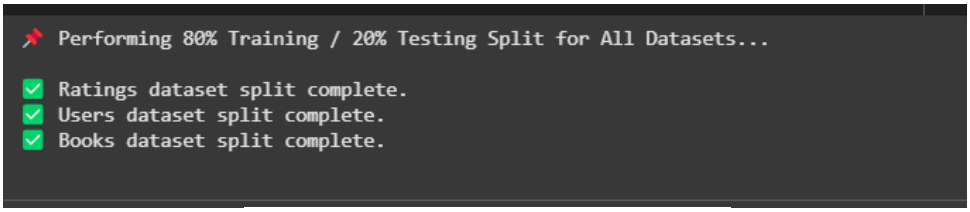

Fig 3.10 Output for train & test

➢ MODEL SELECTION

The process of identifying the most suitable model (algorithm) for the task based on performance indicators like accuracy, precision, recall, etc. is known as model selection. To develop an effective recommendation system, four collaborative filtering algorithms from the Surprise library were selected: BaselineOnly, SVD, NMF, and CoClustering. These models are widely used for rating prediction tasks in recommender systems

- **BaselineOnly** : It uses a straightforward baseline to forecast ratings, taking into consideration item bias, user prejudice, and the global average rating. It functions well as a benchmark model.

- **SVD :** A matrix factorization method called SVD (Singular Value Decomposition) lowers the dimensionality of the user-item interaction matrix in order to reveal latent variables that affect preferences.

- **NMF :** Matrix factorization is also carried out using NMF (Non-negative Matrix Factorization), but non-negative constraints on the factor matrices guarantee the interpretability of the latent features.

- **CoClustering:** This method groups people and things together at the same time and predicts ratings by looking for common patterns among them.

```python
from surprise import Dataset, Reader, BaselineOnly, NMF, SVD, CoClustering
from surprise.model_selection import train_test_split

ratings_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Ratings.csv
ratings_df = pd.read_csv(ratings_path)

# Prepare data for Surprise
reader = Reader(rating_scale=(ratings_df['Book-Rating'].min(), ratings_df['Book-Rating'].max()))
data = Dataset.load_from_df(ratings_df[['User-ID', 'ISBN', 'Book-Rating']], reader)
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)


# Function to train and print status
def train_model(model, model_name):
    print(f"\nTraining Model: {model_name}")
    model.fit(trainset)
    print(f"{model_name} model trained successfully.")

# Initialize and train models
train_model(BaselineOnly(), "BaselineOnly")
train_model(SVD(), "SVD")
train_model(NMF(), "NMF")
train_model(CoClustering(), "CoClustering")
```

Fig 3.11 Code for Model Training

Output :

```
Training Model: BaselineOnly
Estimating biases using als...
BaselineOnly model trained successfully.

Training Model: SVD
SVD model trained successfully.

Training Model: NMF
NMF model trained successfully.

Training Model: CoClustering
CoClustering model trained successfully.
```

Fig 3.12 Output for Model Training

Chapter 4

MODEL EVALUATION & VALIDATION

4.1 Performance Metrics

To evaluate the effectiveness of the recommendation algorithms implemented in this project, three commonly used error metrics were selected: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), and R-squared Score (R^2). These metrics provide insights into the accuracy and reliability of the predicted ratings.

- **Mean Absolute Error (MAE)**

MAE measures the average absolute difference between the predicted ratings and the actual ratings. It provides an intuitive understanding of how far off the predictions are from the real ratings, regardless of direction.

Formula :

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}|$$

Where,

$\hat{y}$ – *predicted value of* y

$\bar{y}$ – *mean value of* y

- **Root Mean Square Error (RMSE)**

RMSE is a quadratic scoring rule that also measures the average magnitude of the prediction error. However, it gives higher weight to larger errors, making it more sensitive to outliers compared to MAE.

Formula :

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2}$$

Lower RMSE values indicate that the model's predictions are closer to the actual ratings.

- **R-squared Score (R^2)**

The R-squared score, also known as the coefficient of determination, measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It indicates how well the predicted ratings approximate the actual ratings. An R^2 value closer to 1 indicates better predictive performance.

Formula :

$$R^2 = 1 - \frac{\sum(y_i - \hat{y})^2}{\sum(y_i - \bar{y})^2}$$

4.2 Model Evaluation

Each model was trained using the training set (80% of the data) and evaluated on the test set (20%) using MAE, RMSE, and R^2. These metrics help determine which model performs best in predicting user preferences. The evaluation was performed using the Surprise library's `accuracy` module.

Input :

```python
from surprise import accuracy
from sklearn.metrics import r2_score

# Function to evaluate model performance
def evaluate_model(model, model_name):
    print(f"\nEvaluating Model: {model_name}")

    # Predict on the testset
    predictions = model.test(testset)

    # Calculate Surprise metrics
    mae = accuracy.mae(predictions, verbose=False)
    rmse = accuracy.rmse(predictions, verbose=False)

    # Calculate R2 Score (manually from true and predicted values)
    true_ratings = [pred.r_ui for pred in predictions]
    predicted_ratings = [pred.est for pred in predictions]
    r2 = r2_score(true_ratings, predicted_ratings)

    # Print results
    print(f"{model_name} Evaluation:")
    print(f"MAE : {mae:.4f}")
    print(f"RMSE: {rmse:.4f}")
    print(f"R²   : {r2:.4f}")

# Evaluate all models
evaluate_model(BaselineOnly().fit(trainset), "BaselineOnly")
evaluate_model(SVD().fit(trainset), "SVD")
evaluate_model(NMF().fit(trainset), "NMF")
evaluate_model(CoClustering().fit(trainset), "CoClustering")
```

Fig 4.1 Code for model evaluation

Output :

```
Evaluating Model: BaselineOnly
BaselineOnly Evaluation:
MAE : 2.8046
RMSE: 3.4031
R²   : 0.2220

Evaluating Model: SVD
SVD Evaluation:
MAE : 2.8180
RMSE: 3.5009
R²   : 0.1766

Evaluating Model: NMF
NMF Evaluation:
MAE : 3.0102
RMSE: 3.9542
R²   : -0.0504

Evaluating Model: CoClustering
CoClustering Evaluation:
MAE : 2.9624
RMSE: 3.7958
R²   : 0.0320
```

Fig 4.2 Output for model evaluation

- **Model Performance**

Table 4.3 Model Performance Comparison Table

Model Name	MAE	RMSE	R2 Score
BaselineOnly	**2.8046**	**3.4031**	**0.2220**
SVD	**2.8184**	**3.5043**	**0.1750**
NMF	**3.0151**	**3.9572**	**0.0412**
CoClustering	**2.9653**	**3.7779**	**-0.0520**

➢ Accuracy Performance

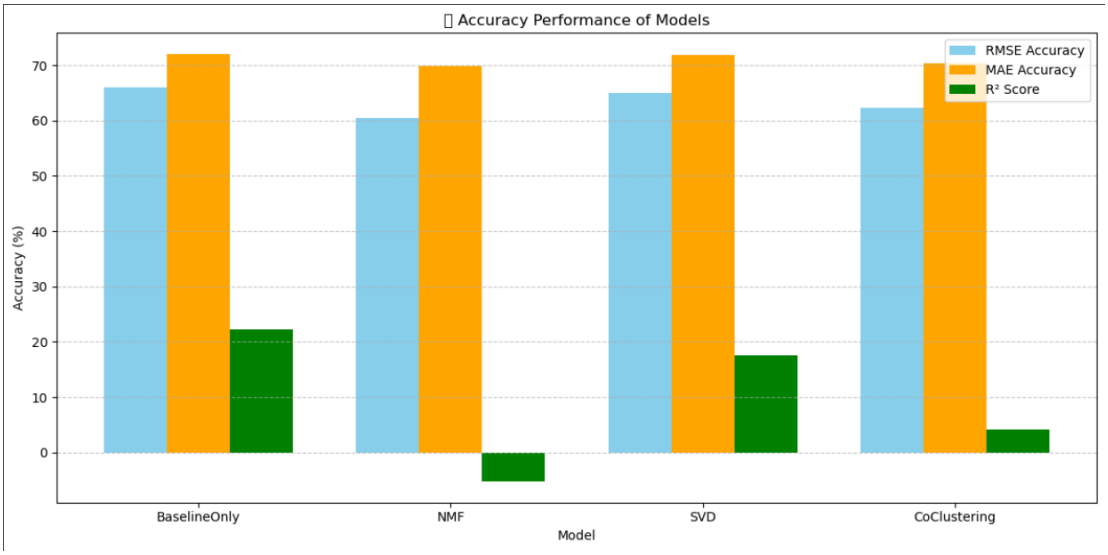

Fig 4.4 Accuracy of Model Performance

The bar chart compares the performance of four models—BaselineOnly, NMF, SVD, and CoClustering—based on RMSE Accuracy, MAE Accuracy, and R^2 Score.

- All models show good accuracy in RMSE and MAE, mostly above 60%.
- MAE accuracy is slightly higher than RMSE for most models.
- The main difference lies in the R^2 Score, which shows how well each model explains rating variance.

✅ Best Model Based on R^2 Score

- BaselineOnly performs best with the highest R^2 Score (~22%), meaning it explains rating behavior better than the others.
- SVD comes next with a decent R^2 (~17%).
- CoClustering has a low but positive R^2 (~4%).
- NMF performs poorly with a negative R^2 (~ -4%), indicating poor prediction quality.

Table 4.5 Accuracy of Model Performance Comparison Table

Model Name	MAE (%)	RMSE (%)	R2 Score (%)
BaselineOnly	**71.95**	**65.97**	**22.20**
SVD	**71.82**	**64.96**	**17.50**
NMF	**69.85**	**60.43**	**-5.20**
CoClustering	**70.35**	**62.22**	**4.12**

Recommendation system : (Taking Random User ID)

Input :

```python
# Prepare Surprise dataset
reader = Reader(rating_scale=(ratings_df['Book-Rating'].min(), ratings_df['Book-Rating'].max()))
data = Dataset.load_from_df(ratings_df[['User-ID', 'ISBN', 'Book-Rating']], reader)

# Train-test split
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

# Train model
algo = BaselineOnly()
algo.fit(trainset)

# -------------------------
# Recommend Books to a User
# -------------------------

def recommend_books(user_id, algo, ratings_df, books_df, top_n=5):
    # Get books the user has already rated
    rated_books = ratings_df[ratings_df['User-ID'] == user_id]['ISBN'].tolist()

    # Get all books the user hasn't rated
    all_books = books_df['ISBN'].unique()
    unrated_books = [isbn for isbn in all_books if isbn not in rated_books]

    # Predict ratings for unrated books
    predictions = []
    for isbn in unrated_books:
        try:
            pred = algo.predict(user_id, isbn)
            predictions.append((isbn, pred.est))
        except:
            continue  # Skip if prediction fails (e.g., unknown ISBN)

    # Sort predictions by estimated rating
    predictions.sort(key=lambda x: x[1], reverse=True)

    # Get top N recommendations
    top_recommendations = predictions[:top_n]

    # Create a readable DataFrame with book titles
    recommended_books = pd.DataFrame(top_recommendations, columns=['ISBN', 'Predicted Rating'])
    recommended_books = recommended_books.merge(books_df[['ISBN', 'Book-Title']], on='ISBN', how='left')

    return recommended_books[['Book-Title', 'Predicted Rating']]

# ----------------------------
# 🔍 Example Usage:
# ----------------------------

user_id_input = 276729  # Replace with any valid User-ID from your dataset
recommendations = recommend_books(user_id_input, algo, ratings_df, books_df, top_n=5)

print(f"📚 Top Book Recommendations for User {user_id_input}:\n")
print(recommendations)
```

Fig 4.6 Code for recommending book to user

Output :

```
Estimating biases using als...
📚 Top Book Recommendations for User 276729:

                                          Book-Title  Predicted Rating
0                                               Free          7.219257
1    Harry Potter and the Sorcerer's Stone (Book 1)          6.259453
2  Harry Potter and the Chamber of Secrets Postca...          6.232266
3  The Blue Day Book: A Lesson in Cheering Yourse...          6.162829
4  Harry Potter and the Prisoner of Azkaban (Book 3)          6.077901
```

Fig 4.7 Output for recommending book to user

➢ Content Based Filtering

Input :

```python
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.neighbors import NearestNeighbors

# Load books dataset
books_path = r"C:\Users\nehaa\Downloads\Sem-4 FDS Dataset Book Recommendation System\Books.csv"
books_df = pd.read_csv(books_path)

# Preprocessing
books_df['Book-Title'] = books_df['Book-Title'].fillna('').astype(str)
books_df['Book-Author'] = books_df['Book-Author'].fillna('').astype(str)
books_df['Publisher'] = books_df['Publisher'].fillna('').astype(str)

# Combine relevant text features
books_df['Combined'] = books_df['Book-Title'] + ' ' + books_df['Book-Author'] + ' ' + books_df['Publisher']

# Vectorize using TF-IDF
vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
tfidf_matrix = vectorizer.fit_transform(books_df['Combined'])

# Fit NearestNeighbors model
model_knn = NearestNeighbors(metric='cosine', algorithm='brute')
model_knn.fit(tfidf_matrix)

# Book title to index mapping
book_indices = pd.Series(books_df.index, index=books_df['Book-Title'].str.lower()).drop_duplicates()

# Recommendation function
def recommend_books(title, n_recommendations=5):
    title = title.strip().lower()
    if title not in book_indices:
        return f"❌ Book title '{title}' not found in dataset."

    idx = book_indices[title]
    book_vector = tfidf_matrix[idx]
    distances, indices = model_knn.kneighbors(book_vector, n_neighbors=n_recommendations + 1)

    print(f"\n📘 Because you liked: {books_df['Book-Title'].iloc[idx]}")
    print("📚 You may also like:")
    for i in range(1, len(indices[0])):
        similar_idx = indices[0][i]
        print(f"{i}. {books_df['Book-Title'].iloc[similar_idx]}")

# ===== Run It =====
user_input = input("🔍 Enter a book title you like: ")
recommend_books(user_input)
```

Fig 4.7 Code for Content based filtering

Output :

```
🔍 Enter a book title you like:  wild animals

📘 Because you liked: Wild Animals
📚 You may also like:
1. Snake: Animals in the Wild (Animals in the Wild)
2. DK Nature Encyclopedia
3. DK Revealed: Ancient Egypt (DK Revealed)
4. Kangaroo (Animals in the Wild)
5. Koala (Animals in the Wild)
```

Fig 4.7 Output for Content based filtering

Chapter 5

CONCLUSION & FUTURE SCOPE

Using collaborative filtering techniques, the Book Recommendation System created for this project aim to provide individualized book recommendations. The Surprise library was used to implement four important algorithms, BaselineOnly, SVD, NMF, and CoClustering, and their prediction accuracy was assessed.Evaluation metrics including R2 Score, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) were employed to evaluate the model's performance. The BaselineOnly model surpassed the others in terms of R2 Score, demonstrating its greater capacity to explain the variance in user ratings, even if models such as SVD and CoClustering demonstrated good accuracy in terms of MAE and RMSE. Overall, the system improves user experience by providing relevant book suggestions based on previous interactions, and it lays a solid foundation for developing more sophisticated, flexible recommendation systems in future applications like e-commerce, music, or movie platforms. This finding emphasizes that even basic models based on user as well as item biases can deliver highly accurate recommendations when trends in data are well captured. The project also highlights the practical use of collaborative filtering and the significance of choosing evaluation metrics that show model effectiveness in various ways.

Future Scope

There are multiple opportunities for the Book Recommendation System to be improved in the future, both in terms of enhancing the core algorithms and adding more features. Among the possible advancements in the future are:

- **Integration of Hybrid Recommendation Systems**: The system can offer recommendations that are even more accurate by fusing content-based and collaborative filtering strategies. In order to recommend books that fit user preferences and book attributes, a hybrid approach might examine both user ratings and the content elements of books (genre, author, keywords, etc.).

- **Adding User Demographics:** By adding user demographic data like age, location, and reading preferences, the system could be improved and make recommendations that are more relevant by customizing them based on users who are similar to each other within particular demographic groups.

- **Real-Time Recommendation Updates:** By integrating real-time data processing, the system may be able to constantly update recommendations according to users' most recent actions, including their most recent reviews or ratings, making sure that suggestions take into account their changing preferences.

Chapter 6

REFERENCES

1. Wayesa, F., Leranso, M., Asefa, G. *et al.* Pattern-based hybrid book recommendation system using semantic relationships. *Sci Rep* **13**, 3693 (2023). https://doi.org/10.1038/s41598-023-30987-0

2. Sathyabama Institute of Science and Technology. *B.E. CSE Batch No. 46 Report*.2022-2023, https://sist.sathyabama.ac.in/sist_naac/aqar_2022_2023/documents/1.3.4/b.e-cse-batchno-46.pdf.

3. Avi Rana and K. Deeba 2019 *J. Phys.: Conf. Ser.* **1362** 012130 https://iopscience.iop.org/article/10.1088/1742-6596/1362/1/012130/pdf

4. David Davis. *Book Recommendation System Project*. Scribd, n.d., https://www.scribd.com/document/630393612/Book-Recommendation-system-PROJECT-docx-pdf.

5. Javed, Umair, et al. "A Review of Content-Based and Context-Based Recommendation Systems". *International Journal of Emerging Technologies in Learning (iJET)*, vol. 16, no. 03, Feb. 2021, pp. pp. 274-306, doi:10.3991/ijet.v16i03.18851.

   (Javed, 2021)