INTRODUCTION TO DATA SCIENCE (CAP 5771)

PROJECT REPORT

Project Title: BOOK RECOMMENDATION SYSTEM USING CONTENT-BASED AND COLLABORATIVE FILTERING TECHINQUES

Name: Pranay Reddy Pullaiahgari

UF-ID: 6238-1134

Project Objective

The objective of the project is to develop a recommendation system that suggests books to the users based on the reading preferences and historical ratings. The goal of the project is to apply content-based filtering and collaborative filtering techniques to provide personalized book recommendations. The project aims to analyze the user behavior and books metadata to improve the recommendation system's accuracy.

This project will be implemented as a web application developed using Streamlit. This enables users to input their preferences, ratings, and then the system will generate recommendations based on user's profile of historical ratings and book preferences.

Tech Stack

The following are the tools, technologies and libraries required for this project:

Python – the core programming language for building this project.

Pandas & Numpy – Python libraries used for data manipulation and numerical computing.

Matplotlib & Seaborn – Python Library used for data visualization.

Surprise – a python scikit for building and analyzing recommender systems.

Scikit-Learn – used for feature engineering and model evaluation.

SqLite – Used for storing book and user data.

Streamlit – Python based web application development framework.

Project Timeline

Milestone 1: Data collection, Preprocessing and Exploratory Data Analysis. (Feb 5 – Feb 21)

Milestone 2: Feature Engineering, Feature Selection and Data Modeling (Feb 22 – Apr 03)

Milestone 3: Evaluation, Interpretation and Tool Development

Evaluation (Apr 04 – Apr 06)

Interpretation (Apr 06 – Apr 08)

Tool Development (Apr 09 – Apr 22)

MILESTONE 1: Data Collection, Preprocessing and Exploratory Data Analysis

(Feb 5, 2025 – Feb 21, 2025)

---

1. Data Collection

Three datasets taken from Kaggle are used to build this recommendation engine.

Books.csv:

It contains book metadata. Attributes are ISBN(Book-ID), Book-Title, Book-Author, Year-Of-Publication, Publisher, Image-URL-S, Image-URL-M, Image-URL-L. It has 271360 records in total.

```
Books.csv
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 271360 entries, 0 to 271359
Data columns (total 8 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   ISBN                 271360 non-null  object
 1   Book-Title           271360 non-null  object
 2   Book-Author          271358 non-null  object
 3   Year-Of-Publication  271360 non-null  object
 4   Publisher            271358 non-null  object
 5   Image-URL-S          271360 non-null  object
 6   Image-URL-M          271360 non-null  object
 7   Image-URL-L          271357 non-null  object
dtypes: object(8)
memory usage: 16.6+ MB
----------------------------------------------------
```

Fig 1.1 Books.csv

Ratings.csv:

It includes book ratings given by the users on the scale of 0 to 10. Attributes are User-ID, ISBN, Book-Rating. It has 1149780 records in total.

```
Ratings.csv
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1149780 entries, 0 to 1149779
Data columns (total 3 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   User-ID      1149780 non-null  int64
 1   ISBN         1149780 non-null  object
 2   Book-Rating  1149780 non-null  int64
dtypes: int64(2), object(1)
memory usage: 26.3+ MB
-----------------------------------------------
```

Fig 1.2 Ratings.csv

Users.csv:

It has the demographic information of the users, including User-ID, Location and Age. It has 278858 records.

```
Users.csv
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 278858 entries, 0 to 278857
Data columns (total 3 columns):
 #   Column    Non-Null Count   Dtype
---  ------    --------------   -----
 0   User-ID   278858 non-null  int64
 1   Location  278858 non-null  object
 2   Age       168096 non-null  float64
dtypes: float64(1), int64(1), object(1)
memory usage: 6.4+ MB
-----------------------------------------
```

Fig 1.3 Users.csv

2. Data Preprocessing

2.1 Data Preprocessing on Books.csv dataset

I dropped the image URL columns because they do not contribute to recommendation logic or machine learning models. These columns are considered irrelevant for analysis and were removed to simplify the dataset.

```
[14]: # dropping images columns from books.csv as they might not be useful
      df_books.drop(columns = ["Image-URL-S", "Image-URL-M"], inplace = True)
```

This code checks if the Year-Of-Publication column contains any non-numeric values. Identifying such anomalies helps prevent data type issues during visualization or modeling.

```
[16]: # checking if we have non-numeric entries for "Year of publication"
      non_numeric_years = df_books['Year-Of-Publication'].apply(lambda x: pd.to_numeric(x, errors='coerce')).isnull().sum()
```

I examined the dataset for missing values across all columns. This helps us decide whether to impute, drop, or treat missing data based on its impact.

```
[20]: # Checking for missing values in the dataset
      df_books.isnull().sum()
```

```
[20]: ISBN                 0
      Book-Title           0
      Book-Author          2
      Year-Of-Publication  0
      Publisher            2
      Image-URL-L          0
      dtype: int64
```

Missing entries in Book-Author and Publisher were filled with "Unknown" to maintain data completeness. This prevents errors in future steps like grouping or encoding without losing records.

```
[21]: # Handling missing values
      # Filling missing Book-Author and Publisher records with "Unknown"
      df_books["Book-Author"]=df_books["Book-Author"].fillna("Unknown")
      df_books["Publisher"]=df_books["Publisher"].fillna("Unknown")
```

I checked for duplicate rows to ensure data integrity. Fortunately, no duplicate entries were found in the dataset.

```
[23]: # checking for duplicates if any
      df_books.duplicated().sum()

[23]: 0
```

2.2 Data Preprocessing on Ratings.csv dataset

I checked for any missing values in the ratings dataset. Fortunately, no null entries were found, indicating the dataset is complete and ready for analysis.

```
[25]: # checking for any missing data
      df_ratings.isnull().sum()

[25]: User-ID        0
      ISBN           0
      Book-Rating    0
      dtype: int64
```

This line checks for duplicate rows in the ratings data. It ensures the dataset doesn't have repeated records that could skew the recommendation logic; in this case, none were found.

```
[26]: # checking for any duplicates
      df_ratings.duplicated().sum()

[26]: 0
```

I examined how many ratings each user has submitted. This is important for identifying active users versus inactive ones, which helps us later filter out users who haven't contributed enough data for collaborative filtering.

```
[27]: # number of ratings each user has given
      df_ratings["User-ID"].value_counts()
```

```
[27]: User-ID
      11676     13602
      198711     7550
      153662     6109
      98391      5891
      35859      5850
                ...
      116180        1
      116166        1
      116154        1
      116137        1
      276723        1
      Name: count, Length: 105283, dtype: int64
```

2.3 Data Preprocessing on Users.csv

I checked the Users.csv file for missing values and found that the "Age" column had a significant number of null entries. This indicated that further preprocessing was necessary to handle missing age data appropriately.

```
[29]: # checking for any missing values
      df_users.isnull().sum()
```

```
[29]: User-ID          0
      Location         0
      Age         110762
      dtype: int64
```

To deal with the missing age values, I calculated the median of the available ages and filled the null entries with it. I chose the median instead of the mean to reduce the impact of extreme outliers on the distribution.

```
[30]: # filling missing age values with median
      median_age=df_users["Age"].median()
      df_users["Age"]=df_users["Age"].fillna(median_age)
```

I decided to drop the "Location" column because it was not directly useful for building the recommendation system. Removing irrelevant features helps reduce dataset complexity and improves model efficiency.

```
[32]: # dropping "location", as it is not a significant attribute
      df_users.drop(columns = ["Location"], inplace = True)
```

To build a complete dataset, I first merged Ratings.csv with Users.csv on "User-ID" to include user demographic details alongside their ratings. Then, I merged this result with Books.csv using "ISBN" to attach book information. I used inner joins in both steps to ensure only valid and complete records were retained. The final dataset now includes user, book, and rating information, which is essential for training the recommendation system.

[37]:
```
# Merging Ratings.csv and Users.csv on "User-ID"
df_ratings_users = pd.merge(df_ratings, df_users, on="User-ID", how="inner")
```

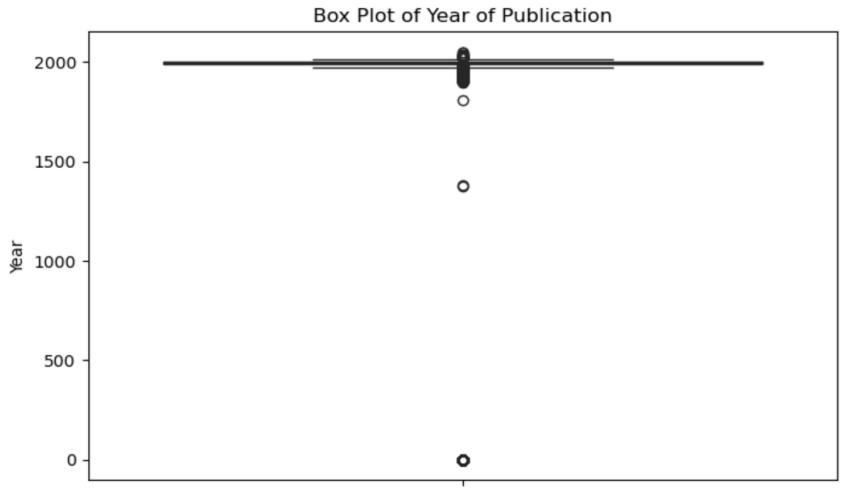
[39]:
```
# The above ratings_users dataset can be merged with Books.csv on "ISBN"
df = pd.merge(df_ratings_users, df_books, on="ISBN", how="inner")
```

3. Exploratory Data Analysis

EDA helps us in understanding the dataset structure, detect anomalies, and extracting valuable insights for predictive model development.

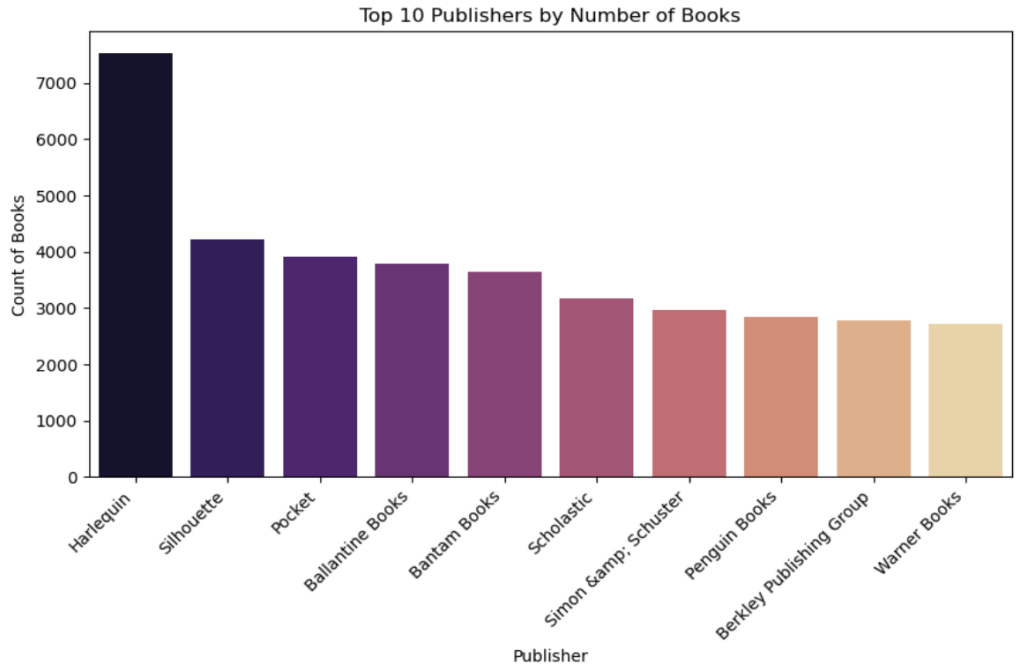
Books.csv Dataset

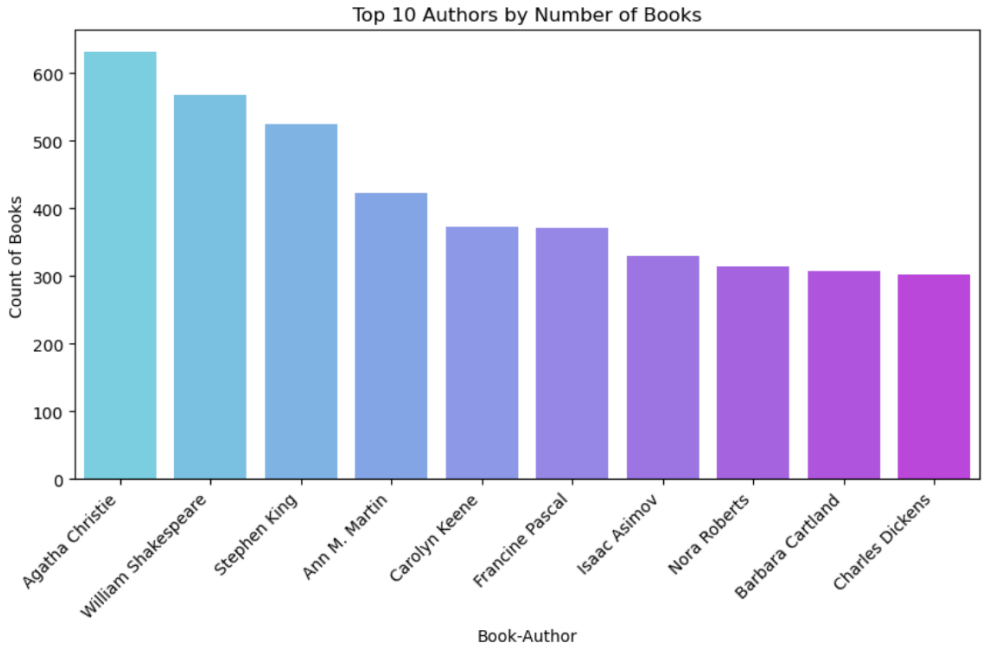
Boxplot for Year of Publication Distribution



We observe that there are some years which are unrealistic, example 0, and others <1800, which are not significant. These incorrect years can distort recommendations based on books age. So we can filter out the books that are published before 1800.

Bar plots for Top Publishers and Authors

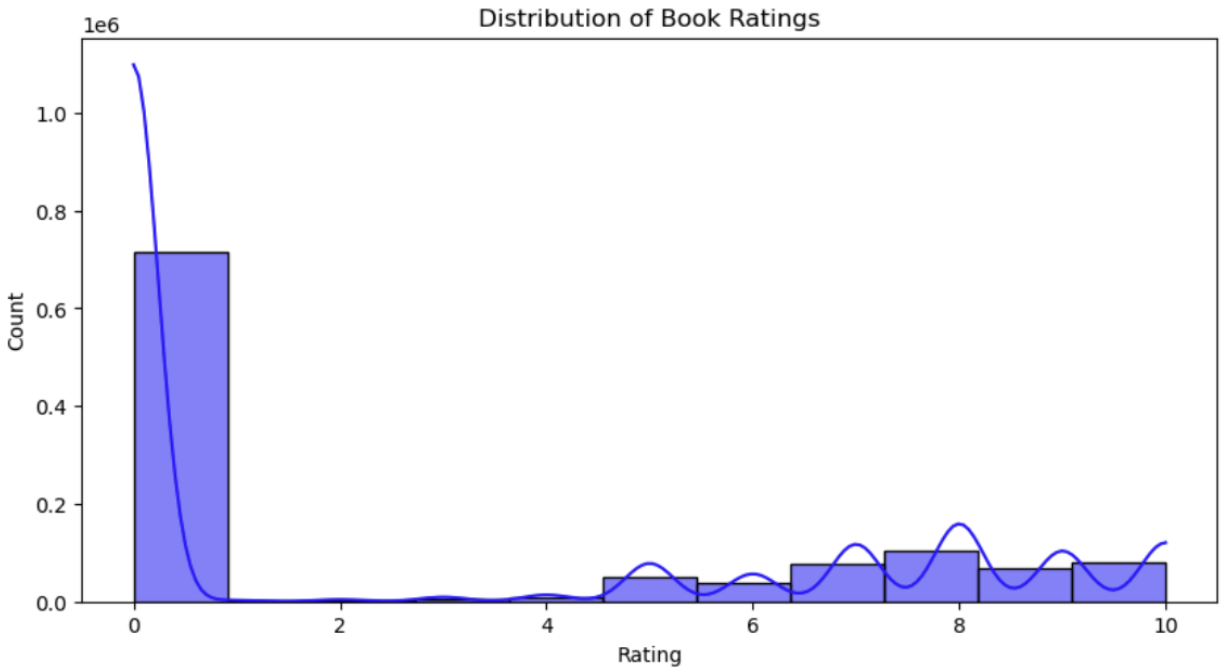




There are small number of publishers and authors responsible for a large portion of books. This visualization helps in identifying key contributors in the dataset, and understand the diversity of book sources available.

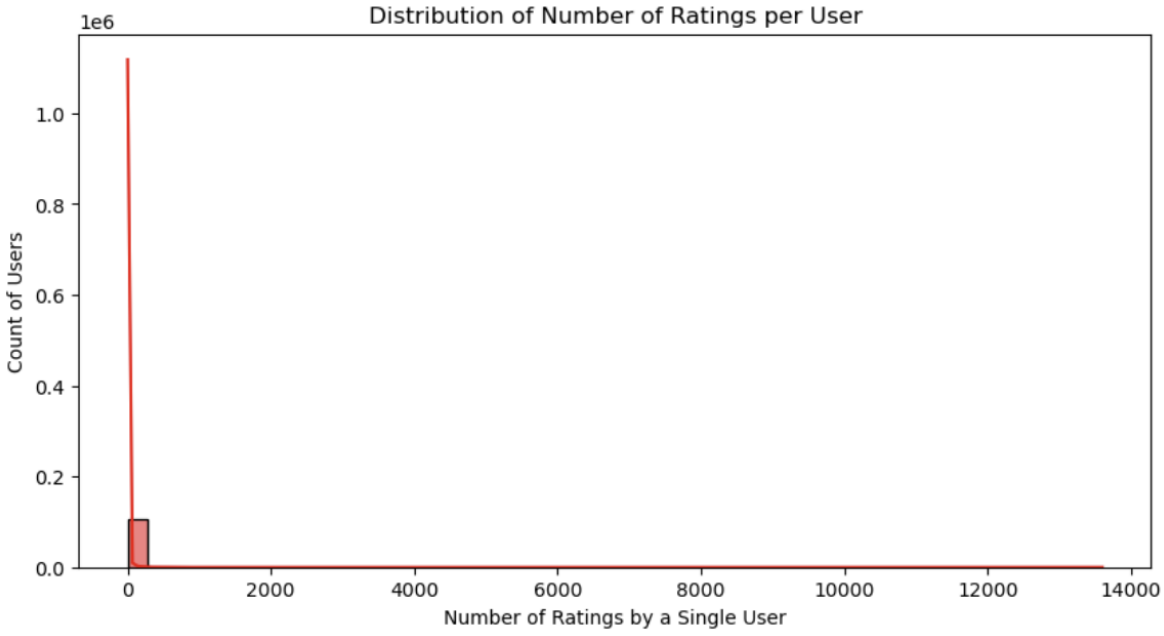
Ratings.csv Dataset

Histogram for ratings distribution



Ratings are highly skewed toward 0 or 10. This type of binary rating tendency might impact model performance. We can consider applying normalization or adjusting weighting to handle skewness.

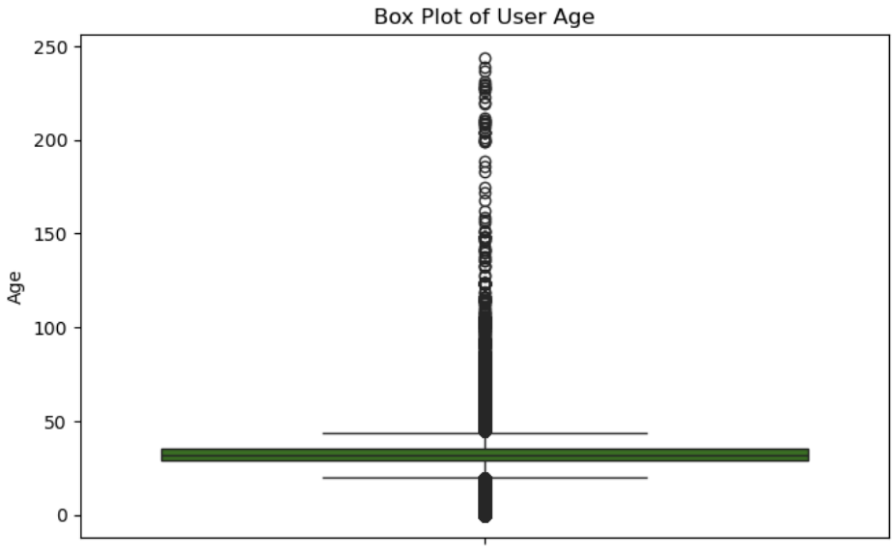
Histogram for Number of Ratings per user



Most of the users have rated very few books, whereas a few users have rated thousands. This sparse data makes collaborative filtering less effective for users with minimal interactions. We consider filtering out the inactive users who rated <5 books, so that model performance can be improved.

Users.csv Dataset

Box plot and Histogram for Age Distribution



Few users have unrealistic ages like 0 and 250. Such outliers can distort demographic based recommendations, and so we clip the age values to the range 5 to 100.

MILESTONE 2: Feature Engineering, Feature Selection and Data Modeling

(Feb 23, 2025 – Apr 03, 2025)

4. Data Cleaning and Filtering

Before proceeding with feature engineering, we perform data cleaning based on key insights discovered during exploratory data analysis. We observed anomalies such as unrealistic user ages, users with very few ratings, and books rated too few times to provide meaningful patterns. These cleaning steps help improve the quality of the dataset and ensure our recommendation models are trained on reliable data.

To improve the quality of the dataset, I started by clipping the user ages to a sensible range between 5 and 100. This helped eliminate unrealistic values like 0 or 250, which could introduce noise in the analysis. I then filtered out users who had rated fewer than 5 books, as these users don't provide enough data to model their preferences effectively in collaborative filtering. Similarly, I excluded books that had received fewer than 10 ratings since such books lack sufficient feedback to be recommended reliably. These steps reduced noise and focused the data on active users and popular books, making the dataset more meaningful for training recommendation models.

[71]:
```
# Clipping unrealistic ages to a sensible range (5 to 100). This helps us eliminate noisy outliers like 0 or 250 years old
cleaned_df['Age'] = cleaned_df['Age'].clip(lower=5, upper=100)
```

[72]:
```
# Filtering out users who have rated fewer than 5 books. These users provide too little data to be useful for recommendations
user_rating_counts = cleaned_df['User-ID'].value_counts()
active_users = user_rating_counts[user_rating_counts >= 5].index
cleaned_df = cleaned_df[cleaned_df['User-ID'].isin(active_users)]
```

[73]:
```
# Filtering out books that have been rated fewer than 10 times. These books don't have enough information for meaningful recommendations
book_rating_counts = cleaned_df['ISBN'].value_counts()
popular_books = book_rating_counts[book_rating_counts >= 10].index
cleaned_df = cleaned_df[cleaned_df['ISBN'].isin(popular_books)]
```

5. Feature Engineering

I performed feature engineering to enrich the dataset with additional information that could be useful for building more accurate and insightful recommendation models. I started by creating a copy of the cleaned dataset to avoid altering the original data. Then, I added four new features:

- **User-Rating-Count**: the number of ratings each user has given, which helps identify how active a user is.

- **User-Average-Rating**: the average score each user tends to give, which can be useful to normalize rating behavior.
- **Book-Rating-Count**: the number of times each book has been rated, helping assess a book's popularity.
- **Book-Average-Rating**: the overall perception of the book based on its average score.

These engineered features can be used for filtering, modeling, or providing weighted inputs in both collaborative and content-based filtering approaches to improve recommendation quality.

[77]:
```
# Creating a new dataframe to store engineered features from the cleaned data
feature_df = cleaned_df.copy()
```

[78]:
```
feature_df.shape
```

[78]:
```
(445391, 9)
```

[79]:
```
# Feature 1: Number of ratings given by each user
user_rating_count = feature_df.groupby('User-ID')['Book-Rating'].count().reset_index()
user_rating_count.columns = ['User-ID', 'User-Rating-Count']
feature_df = feature_df.merge(user_rating_count, on='User-ID', how='left')
```

[80]:
```
# Feature 2: Average rating given by each user
user_avg_rating = feature_df.groupby('User-ID')['Book-Rating'].mean().reset_index()
user_avg_rating.columns = ['User-ID', 'User-Average-Rating']
feature_df = feature_df.merge(user_avg_rating, on='User-ID', how='left')
```

[81]:
```
# Feature 3: Number of ratings received by each book
book_rating_count = feature_df.groupby('ISBN')['Book-Rating'].count().reset_index()
book_rating_count.columns = ['ISBN', 'Book-Rating-Count']
feature_df = feature_df.merge(book_rating_count, on='ISBN', how='left')
```

[82]:
```
# Feature 4: Average rating received by each book
book_avg_rating = feature_df.groupby('ISBN')['Book-Rating'].mean().reset_index()
book_avg_rating.columns = ['ISBN', 'Book-Average-Rating']
feature_df = feature_df.merge(book_avg_rating, on='ISBN', how='left')
```

[83]:
```
feature_df.shape
```

[83]:
```
(445391, 13)
```

6. Model Building

This book recommendation system works in three different ways, depending on how a user interacts with it.

6.1 Publisher-Based Classification from Title (Supervised ML Model). We classify the Publisher of a book based on its title using machine learning, then recommend the top-rated books from the predicted publisher.

I dropped rows with missing titles or publisher data because these fields are essential for training a supervised model that predicts the publisher from a book title. Without both inputs and labels, the model wouldn't function properly.

```
[89]: # Drop rows with missing publisher or title
      publisher_df = cleaned_df.dropna(subset=['Book-Title', 'Publisher'])
```

I used **TfidfVectorizer** to convert book titles into numerical features based on term frequency-inverse document frequency. This helps the model focus on important words that distinguish titles, while reducing noise from common English words. Also, I split the data into training and testing sets to evaluate the model performance. I used a standard 80-20 split, ensuring reproducibility with a fixed random seed.

```
[91]: # Vectorizing titles
      vectorizer = TfidfVectorizer(stop_words='english')
      X = vectorizer.fit_transform(publisher_df['Book-Title'])
      y = publisher_df['Publisher']
```

```
[92]: # Splitting into train and test
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

I trained three classifiers—Naive Bayes, Logistic Regression, and Decision Tree—to compare their effectiveness in predicting the publisher. Logistic Regression ultimately performed best and was used in the final recommendation logic.

```
[94]: # (i): Multinomial Naive Bayes
      model_nb = MultinomialNB()
      model_nb.fit(X_train, y_train)
      preds_nb = model_nb.predict(X_test)
      print("MultinomialNB Accuracy:", accuracy_score(y_test, preds_nb))
      print(classification_report(y_test, preds_nb))
```

```
MultinomialNB Accuracy: 0.6650388980567811 •••
```

```
[187]: # (ii): Logistic Regression
       model_lr = LogisticRegression(max_iter=1000)
       model_lr.fit(X_train, y_train)
       preds_lr = model_lr.predict(X_test)
       print("Logistic Regression Accuracy:", accuracy_score(y_test, preds_lr))
       print(classification_report(y_test, preds_lr))
```

```
Logistic Regression Accuracy: 0.8360107320468348
```

```
# (iii): Decision Tree
model_dt = DecisionTreeClassifier(random_state=42)
model_dt.fit(X_train, y_train)
preds_dt = model_dt.predict(X_test)
print("Decision Tree Accuracy:", accuracy_score(y_test, preds_dt))
print(classification_report(y_test, preds_dt))
```

Among all the three classifiers, Logistic Regression performed well with the accuracy of 83.6%.

6.2 Content-Based Filtering (Based on Book Features)

When a real-time user searches for a book, we recommend similar books using TF-IDF on features like title, author, and publisher.

[189]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Combine relevant textual features into a single string for every book
cleaned_df['Combined'] = (cleaned_df['Book-Title'].fillna('') + ' '
                          + cleaned_df['Book-Author'].fillna('') + ' '
                          + cleaned_df['Publisher'].fillna(''))
cleaned_df = cleaned_df.reset_index(drop=True)
```

[191]:
```
# creating the book_indices mapping
book_indices = pd.Series(cleaned_df.index, index=cleaned_df['Book-Title']).drop_duplicates()
# fitting the model
vectorizer_cb = TfidfVectorizer(stop_words='english')
tfidf_matrix = vectorizer_cb.fit_transform(cleaned_df['Combined'])
from sklearn.neighbors import NearestNeighbors
model_cb = NearestNeighbors(metric='cosine', algorithm='brute')
model_cb.fit(tfidf_matrix)
```

[191]:
```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

I implemented content-based filtering using the textual features of each book—such as title, author, publisher, and language. I combined these features into a single text field and vectorized them using TF-IDF to capture important terms while ignoring common ones. I then trained a Nearest Neighbors model using cosine similarity to identify books that are similar in content. When a user selects or rates a book, this model suggests other books with similar attributes, regardless of user behavior.

6.3 Collaborative Filtering (Based on User Ratings)

We recommend books to a user by identifying users with similar rating patterns using a K-Nearest Neighbors model. The system suggests books that those similar users have rated highly but the current user hasn't interacted with yet.

[175]:
```
# Creating a user-item matrix (rows: users, columns: books, values: ratings)
user_item_matrix = feature_df.pivot_table(index='User-ID', columns='Book-Title', values='Book-Rating').fillna(0)
```

[177]:
```
# Fitting the KNN model on user-item matrix using cosine similarity
knn_model = NearestNeighbors(metric='cosine', algorithm='brute')
knn_model.fit(user_item_matrix)
```

[177]:
```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine')
```

[179]:
```
# fro instance, consider a real-time user who rated 3 books
real_user_ratings = {
    'The Hobbit': 8,
    'Harry Potter and the Philosopher\'s Stone': 9,
    '1984': 7
}
```

[181]:
```
# Convert real-time user ratings to a DataFrame matching user-item matrix columns
real_user_df = pd.DataFrame([real_user_ratings])
real_user_df = real_user_df.reindex(columns=user_item_matrix.columns, fill_value=0)
```

[183]:
```
# Find the 5 nearest users based on rating similarity
distances, indices = knn_model.kneighbors(real_user_df, n_neighbors=5)
# Get User-IDs of the nearest users
similar_users = user_item_matrix.iloc[indices[0]].index.tolist()
# Retrieve their ratings
similar_ratings = user_item_matrix.loc[similar_users]
# Calculate average rating for each book rated by similar users
avg_ratings = similar_ratings.mean().sort_values(ascending=False)
# Remove books already rated by the real-time user
already_rated = set(real_user_ratings.keys())
final_recommendations = avg_ratings[~avg_ratings.index.isin(already_rated)].head(10)
```

[185]:
```
# Display final book recommendations
print("Recommended Books (Collaborative Filtering):")
print(final_recommendations)
```

```
Recommended Books (Collaborative Filtering):
Book-Title
 Murder of a Sleeping Beauty (Scumble River Mysteries (Paperback))              0.0
Stolen Lives : Twenty Years in a Desert Jail (Oprah's Book Club (Paperback))    0.0
Still Pumped From Using The Mouse                                              0.0
Still Waters                                                                   0.0
Still life with Woodpecker                                                     0.0
Still of the Night                                                             0.0
Stillwatch                                                                     0.0
Stitch 'N Bitch: The Knitter's Handbook                                        0.0
Stitches in Time                                                               0.0
Stolen                                                                         0.0
dtype: float64
```

[ ]:

I implemented a user-based collaborative filtering approach using the K-Nearest Neighbors (KNN) algorithm. First, I created a user-item matrix where each row represents a user and each column represents a book title, with the cell values indicating the rating a user gave to that book. I filled missing values with zero to maintain matrix integrity for similarity calculations. This matrix served as the foundation for identifying user-user relationships based on their rating behavior.

I then trained a KNN model using cosine similarity, which is effective for measuring how closely users' rating patterns align, regardless of the scale of ratings. To simulate a real-time scenario, I defined a mock user who rated three popular books. This user's input was converted into the same format as the user-item matrix to allow for similarity comparison.

Using the KNN model, I retrieved the top 5 users who had similar tastes to the mock user based on their ratings. I then aggregated these similar users' ratings and calculated the average rating for each book. Books that the mock user had already rated were excluded from the results to avoid redundancy. Finally, I displayed the top 10 highest-rated books (by similar users) that the real-time user hadn't rated yet. These recommendations are tailored to the user based on the preferences of like-minded readers, making this stage a core component of personalized suggestions in the system.