Bilkent University
CS464 Machine Learning
Final Report
Group 15

# Book Recommendation System using Collaborative and Content Based Filtering

Group Members:

- Ferhat Korkmaz 21901940
- Ömer Oktay Gültekin 21901413
- Utku Kurtulmuş 21903025
- Dilay Yigit 21602059
- Muhammet Oğuzhan Gültekin 21801616

## Table Of Contents

# 1. Introduction

The outline of our project ‘Booking Recommendation System’ is the design and development of a tailored book recommendation system that aims to refine the way readers discover new books and cultivate their literary interests. Drawing from a rich dataset sourced from Kaggle, which encapsulates over 3 million book reviews for 212,404 unique titles, our project makes use of a comprehensive Amazon review dataset, extending from May 1996 to July 2014.

Our data foundation consists of two main components: the primary file focuses on the evaluative aspect of reader feedback, while the 'Books Details' file enhances this with comprehensive information about each book obtained from the Google Books API. This combination of quantitative ratings and qualitative descriptions forms the basis for our model’s recommendation logic. During data preparation, we have selected a subset of dedicated readers—those who have reviewed a minimum of 20 books— and used their reviews to train our prediction models. We have used explicit feedback mechanisms while implementing our model.

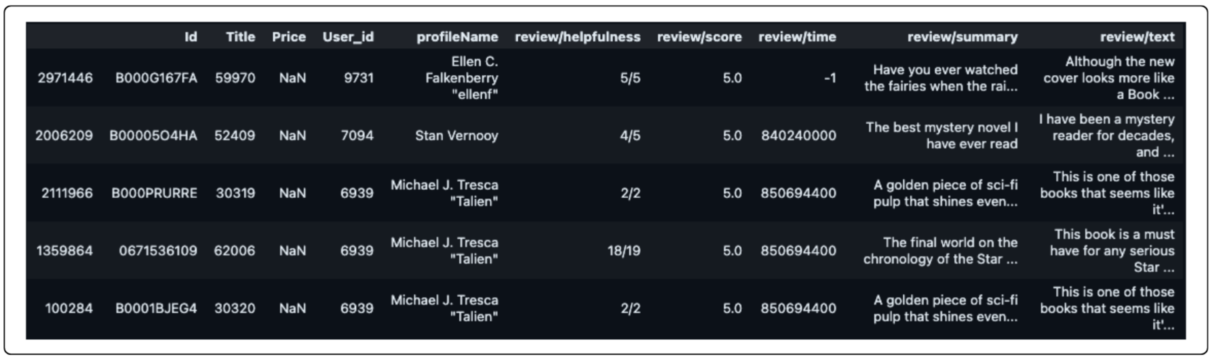
In our project, we have implemented collaborative filtering through a neural network with intricate neuron structures, incorporating user and item embeddings. This advanced technique allows our model to discern complex user-book interactions, forming the basis for generating personalized recommendations. The neuron structure of our collaborative filtering model enables it to capture and understand nuanced patterns in user preferences. The user and item embeddings contribute to this understanding by representing users and books in a high-dimensional space.

Additionally, we implemented content-based filtering, where we used leverage techniques such as cosine similarity to enhance the accuracy of our recommendations. Furthermore, we explored and used regression methods, including Linear Regression, Decision Tree Regression, and Random Forest Regression, to further refine our predictive modeling capabilities. Also, we did hyper-tuning on parameters to increase the accuracy of the models that we have implemented. These steps optimized our model's performance, ensuring it evolves to meet the dynamic needs of users and delivers precise, tailored book recommendations.

The ultimate objective of this project extends beyond creating a system that enhances users' literary journeys; it also aims to contribute to the broader discussion on how machine learning can personalize digital experiences by not simply recommending similar books but also learn from other similar readers' preferences. This report will document our approach, covering aspects from data acquisition and preprocessing to model selection, coding environment, and details of the trained model in a recommendation system that is both robust and attuned to users' evolving literary preferences.

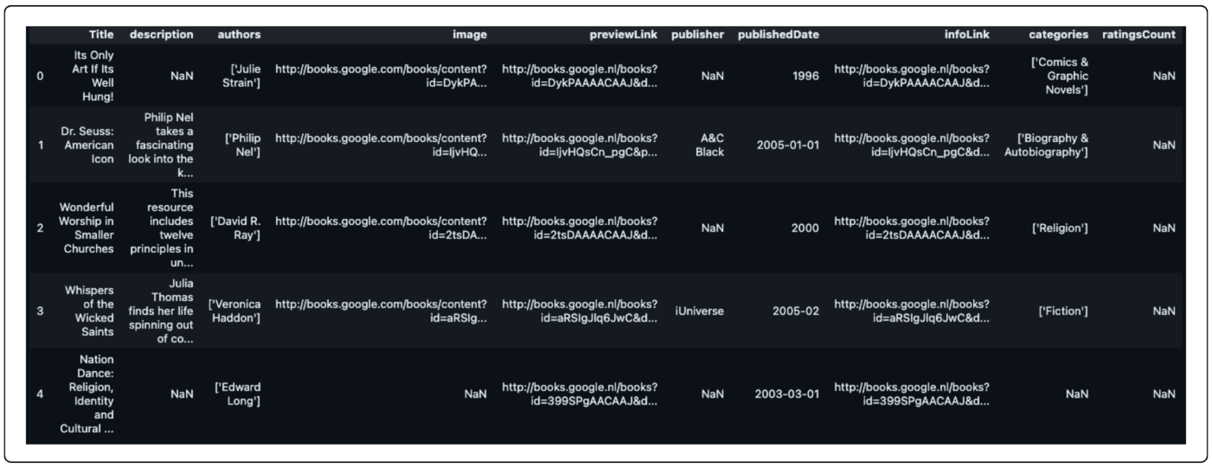
## 2. Data Preprocessing

The Amazon Book Review dataset contains information about 3 million book reviews for 212404 unique books. Also, the details of those books are given in the below figures [1].



| | Id | Title | Price | User_id | profileName | review/helpfulness | review/score | review/time | review/summary | review/text |
|---|---|---|---|---|---|---|---|---|---|---|
| 2971446 | B000G167FA | 59970 | NaN | 9731 | Ellen C. Falkenberry "ellenf" | 5/5 | 5.0 | -1 | Have you ever watched the fairies when the rai... | Although the new cover looks more like a Book ... |
| 2006209 | B00005O4HA | 52409 | NaN | 7094 | Stan Vernooy | 4/5 | 5.0 | 840240000 | The best mystery novel I have ever read | I have been a mystery reader for decades, and ... |
| 2111966 | B000PRURRE | 30319 | NaN | 6939 | Michael J. Tresca "Talien" | 2/2 | 5.0 | 850694400 | A golden piece of sci-fi pulp that shines even... | This is one of those books that seems like it'... |
| 1359864 | 0671536109 | 62006 | NaN | 6939 | Michael J. Tresca "Talien" | 18/19 | 5.0 | 850694400 | The final world on the chronology of the Star ... | This book is a must have for any serious Star ... |
| 100284 | B0001BJEG4 | 30320 | NaN | 6939 | Michael J. Tresca "Talien" | 2/2 | 5.0 | 850694400 | A golden piece of sci-fi pulp that shines even... | This is one of those books that seems like it'... |

Figure 1: Books_Rating.csv file data structure



| | Title | description | authors | image | previewLink | publisher | publishedDate | infoLink | categories | ratingsCount |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Its Only Art If Its Well Hung! | NaN | ['Julie Strain'] | http://books.google.com/books/content?id=DykPA... | http://books.google.nl/books?id=DykPAAAACAAJ&d... | NaN | 1996 | http://books.google.nl/books?id=DykPAAAACAAJ&d... | ['Comics & Graphic Novels'] | NaN |
| 1 | Dr. Seuss: American Icon | Philip Nel takes a fascinating look into the k... | ['Philip Nel'] | http://books.google.com/books/content?id=IjvHQ... | http://books.google.nl/books?id=IjvHQsCn_pgC&p... | A&C Black | 2005-01-01 | http://books.google.nl/books?id=IjvHQsCn_pgC&d... | ['Biography & Autobiography'] | NaN |
| 2 | Wonderful Worship in Smaller Churches | This resource includes twelve principles in un... | ['David R. Ray'] | http://books.google.com/books/content?id=2tsDA... | http://books.google.nl/books?id=2tsDAAAACAAJ&d... | NaN | 2000 | http://books.google.nl/books?id=2tsDAAAACAAJ&d... | ['Religion'] | NaN |
| 3 | Whispers of the Wicked Saints | Julia Thomas finds her life spinning out of co... | ['Veronica Haddon'] | http://books.google.com/books/content?id=aRSIg... | http://books.google.nl/books?id=aRSIgJlq6JwC&d... | iUniverse | 2005-02 | http://books.google.nl/books?id=aRSIgJlq6JwC&d... | ['Fiction'] | NaN |
| 4 | Nation Dance: Religion, Identity and Cultural ... | NaN | ['Edward Long'] | NaN | http://books.google.nl/books?id=399SPgAACAAJ&d... | NaN | 2003-03-01 | http://books.google.nl/books?id=399SPgAACAAJ&d... | NaN | NaN |

Figure 2: Books_Data.csv file data structure

The mapping of these two different csv files was accomplished with the field "Title" as seen in Figure 3.

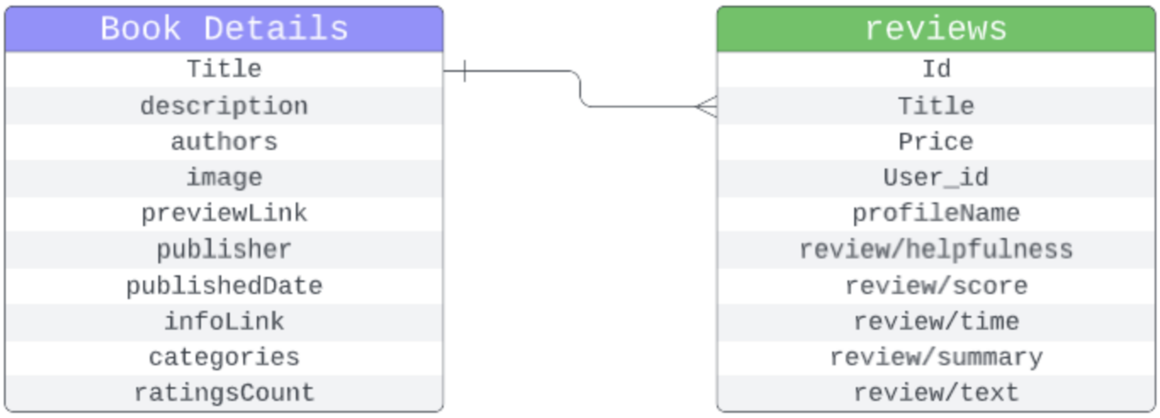


Figure 3: The diagram of files with their columns and relationship

However, for the initial model, we wanted to use Collaborative Filtering Approach and did not need most of the fields [2]. In other words, we have eliminated all of the fields other than “Title, Id, User_id, and review/score”. Therefore, we decided to adopt the explicit feedback mechanism for our model.

Also, In order to train our model with more data for each user. We wanted to eliminate some reviews. Our criteria was, if we were to use a user’s review data, that user should have at least 20 unique book reviews. As a result there are 7209 users that have reviewed at least 20 books. And 548134 unique review data. At first, we were skeptical about whether that number of reviews were enough or not. However, as we tested our model with some criteria, which will be explained later on, we got an acceptable rate of loss value and accuracy.

We have label encoded the user_id and title columns, using the label encoder of scikit learn.

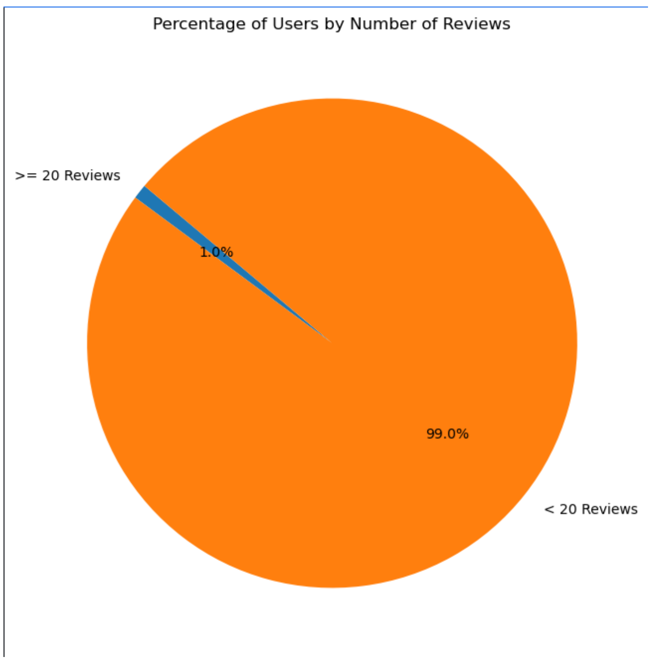
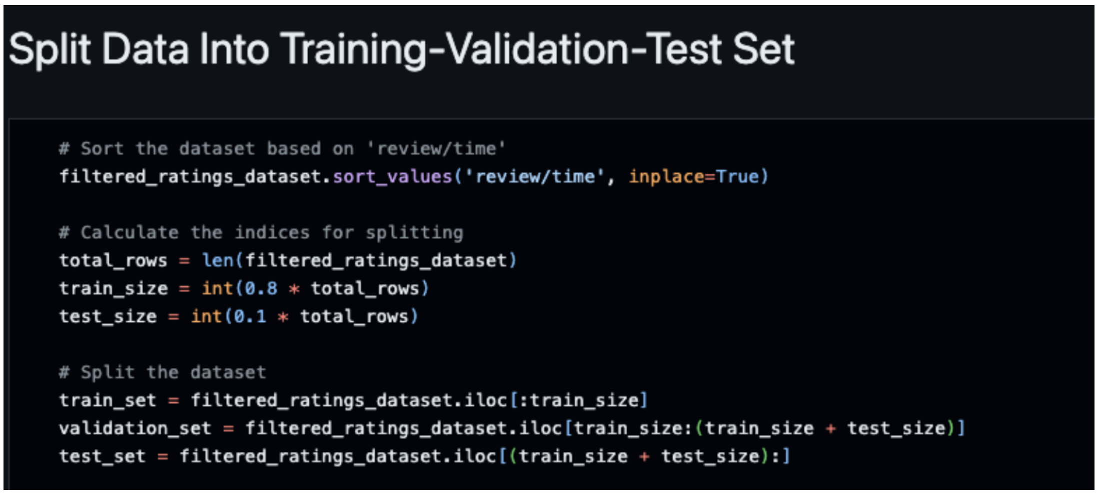


Figure 4: Pie Chart of the distribution of users with respect to review numbers

After eliminating some of the features and the data, the last thing we had to do was splitting the dataset for training, validation, and testing purposes. We have sorted our array according to the review date, and the oldest 80 percent are reserved for the train set and the newest 10 percent for the test set. We are using an 80-10-10 split.


Split Data Into Training-Validation-Test Set

```
# Sort the dataset based on 'review/time'
filtered_ratings_dataset.sort_values('review/time', inplace=True)

# Calculate the indices for splitting
total_rows = len(filtered_ratings_dataset)
train_size = int(0.8 * total_rows)
test_size = int(0.1 * total_rows)

# Split the dataset
train_set = filtered_ratings_dataset.iloc[:train_size]
validation_set = filtered_ratings_dataset.iloc[train_size:(train_size + test_size)]
test_set = filtered_ratings_dataset.iloc[(train_size + test_size):]
```

### 2.1 Training Dataset

Comprises 80% of the data, used for training the model on user ratings for books.

### 2.2 Validation Dataset

Consists of 10% of the data, employed to fine-tune and validate the model during training.

### 2.3 Test Dataset

The remaining 10%, used to assess the model's performance on new, unseen data

## 3. Coding Environment

In the pursuit of developing a book recommendation system, our project was scaffolded within the versatile and robust coding environment provided by Google Colab. The selection of Google Colab as our primary development platform is because it promotes collaborative work in real-time without necessitating a version control system, and it furnishes enhanced computational efficiency [3]. This environment is especially conducive for data-intensive tasks, such as training sophisticated machine learning models.

Python, acclaimed for its simplicity and power, served as the cornerstone programming language for this project [5]. Its widespread adoption and the richness of its libraries make it an ideal candidate for machine learning tasks. Within this Pythonic ecosystem, we utilized a suite of libraries tailored to our project needs:

- TensorFlow: This open-source library is renowned for its flexible ecosystem and extensive resource pool that supports machine learning and deep learning model development and deployment. TensorFlow provided the infrastructure necessary for building and training our neural network-based recommendation system [8].

- Pandas & NumPy: These libraries were instrumental for data manipulation and numerical computation, respectively. Pandas, with its high-level data structures, made data cleansing, exploration, and preprocessing both intuitive and efficient. NumPy complemented these operations by offering powerful array-processing capabilities, thereby streamlining the computational aspect of our data handling processes [6] [7].

- Python: Beyond serving as the language of choice, Python's role extended to integrating these libraries into a cohesive workflow. Its syntax and language features enabled seamless transitions between different stages of data processing and model development.

- Scikit-learn: Scikit-learn is a popular library known for its machine-learning models. However, we are using it to evaluate the performance of our model by using its accuracy metric from its metric library [11]. Moreover, we are using their ML models to implement our linear regression, decision tree regression and random forest regression models.

The amalgamation of Google Colab, TensorFlow, Pandas, NumPy, Scikit-learn, and Python established a formidable coding ecosystem, equipping us with the necessary tools to tackle the complexities of our project. This environment not only supported the technical demands of our recommendation system but also fostered a dynamic and iterative development process. The outcome is a testament to the efficacy of these tools when applied to the challenges of predictive modeling and machine learning.

# 4. Details of the Trained Models

## 4.1 Neural Networks

### 4.1.1 Feature Embeddings

As an initial step to implementing our model, we had to describe our input data in a way that our neural network would be able to understand and process our data to find the patterns in the input data [9]. To do that, we have encoded our title and user_id data to converge them into numerical data. Then, we could have fed our model with this data. However, for our model to work more efficiently, we have needed to embed this numerical data into dense vectors with continuous numbers [9] [10]. Thus, we have an initial embedding layer where we give the encoded user and item input to the layer to get the embedded results[9]. As an initial embedding size, we have selected 20, but we might change the embedding size during the parameter tuning process. In general, increasing the embedding size can help to capture the features of the item more accurately, but it can increase the model complexity.

```
# Get the number of unique users, items, and categories
num_users = filtered_ratings_dataset['User_id'].nunique()
num_items = filtered_ratings_dataset['Title'].nunique()

# Define the embedding size
embedding_size = 20

# Define input layers for user, item, and content features
user_input = Input(shape=(1,), name='user_input')
item_input = Input(shape=(1,), name='item_input')

# Embedding layers
user_embedding = Embedding(input_dim=num_users, output_dim=embedding_size, input_length=1)(user_input)
item_embedding = Embedding(input_dim=num_items, output_dim=embedding_size, input_length=1)(item_input)
```

### 4.1.2 Neural Network Implementation

After the embedding layer, we have a dense layer where we give embedded inputs to our neural network. However, before giving the direct embedded result, we flatten our embedded data and concatenate them.

```
# Flatten embeddings
user_flat = Flatten()(user_embedding)
item_flat = Flatten()(item_embedding)

# Concatenate the flattened embeddings
concatenated_features = Concatenate()([user_flat, item_flat])
```

Then, we define our dense layer, with 128 neurons and a ReLu activation function. Initially, we have selected to go with one dense layer, but in the future, we can increase the layer size to increase the accuracy of our model with a cost of model complexity.

```
# Dense layer for further processing
```

```
dense_layer = Dense(128, activation='relu')(concatenated_features)
```

In the last layer of our network, we have the output layer where we get the review prediction for the given input. This layer has only one neuron that will give the output.

```
# Output layer
output_layer = Dense(1)(dense_layer)
```

As a final step, we finish our model by selecting the Adam optimizer as the model optimizer and the mean squared error (MSE) function as the loss function. In the final, we might again change them while we are doing parameter tuning and model optimization.

```
# Create the model
model = Model(inputs=[user_input, item_input], outputs=output_layer)
model.compile(optimizer='adam', loss='mean_squared_error')
```

### 4.1.3 Training Our Model

We are training our model with 10 epochs, which means we are passing the whole train dataset 10 times to improve our model at each time with backpropagation. Then, at the end of each evaluation, we see the performance improvement on both the training dataset and the validation dataset. The performance metrics are given in terms of our loss function MSE.

```
# Define the number of epochs
num_epochs = 10

model.fit(
    [
        train_set['User_id'],
        train_set['Title'],
    ],
    train_set['review/score'],
    epochs=num_epochs,
    validation_data=(
        [
            validation_set['User_id'],
            validation_set['Title'],
        ],
        validation_set['review/score']
    )
)
Epoch 1/10
```

```
        13704/13704 [==============================] - 77s 5ms/step - loss: 0.8505 - val_loss:
1.0004
        Epoch 2/10
        13704/13704 [==============================] - 68s 5ms/step - loss: 0.5786 - val_loss:
1.0372
        Epoch 3/10
        13704/13704 [==============================] - 66s 5ms/step - loss: 0.4836 - val_loss:
1.1084
        Epoch 4/10
        13704/13704 [==============================] - 66s 5ms/step - loss: 0.4103 - val_loss:
1.1253
        Epoch 5/10
        13704/13704 [==============================] - 68s 5ms/step - loss: 0.3490 - val_loss:
1.1803
        Epoch 6/10
        13704/13704 [==============================] - 66s 5ms/step - loss: 0.2973 - val_loss:
1.2124
        Epoch 7/10
        13704/13704 [==============================] - 66s 5ms/step - loss: 0.2562 - val_loss:
1.2333
        Epoch 8/10
        13704/13704 [==============================] - 66s 5ms/step - loss: 0.2221 - val_loss:
1.2381
        Epoch 9/10
        13704/13704 [==============================] - 69s 5ms/step - loss: 0.1948 - val_loss:
1.2712
        Epoch 10/10
        13704/13704 [==============================] - 65s 5ms/step - loss: 0.1721 - val_loss:
1.2873
```

We can see that our training loss still didn't converge and we could have gone with more epochs at the final stage. However, our validation loss seems to increase as we train our data more on the training dataset, which can be a signal of overfitting.

#### 4.1.4 Parameter Tuning

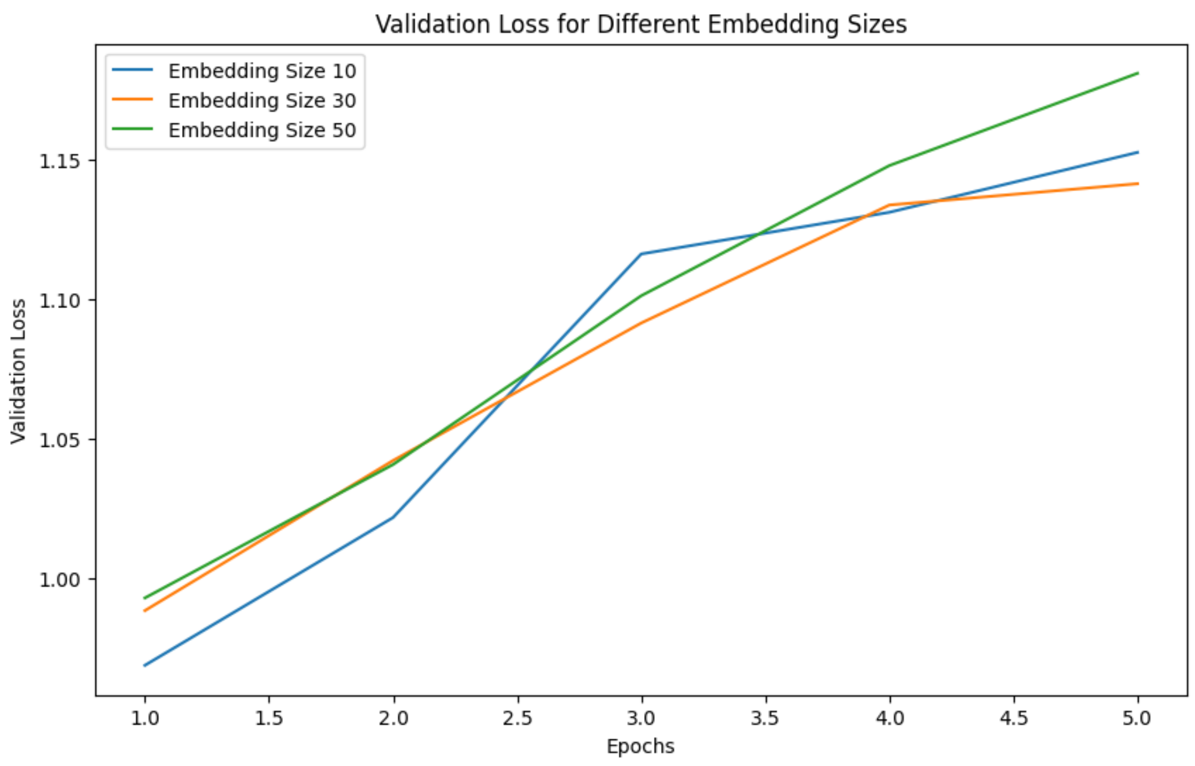
To improve the accuracy of our model, we are doing hyper-parameter tuning on the embedding size of our embedding layer, dense size ( number of neurons on our dense layer), and batch size. We are testing their performance on the validation loss, although we are going to use a different evaluation metric. We didn't use train loss since, in our model, while the training loss is decreasing, the validation loss seems to be increasing. Moreover, training loss is biased since our model updates its weights according to the training loss. Furthermore, in our model, increasing the number of epochs seems to increase the validation loss and the test loss, which will decrease the accuracy of our model. We are using the following hyperparameter values to test our model:

- **Embedding Size:** 10, 30, 50
- **Dense Size:** 32, 64, 256
- **Batch Size:** 32, 128, 256

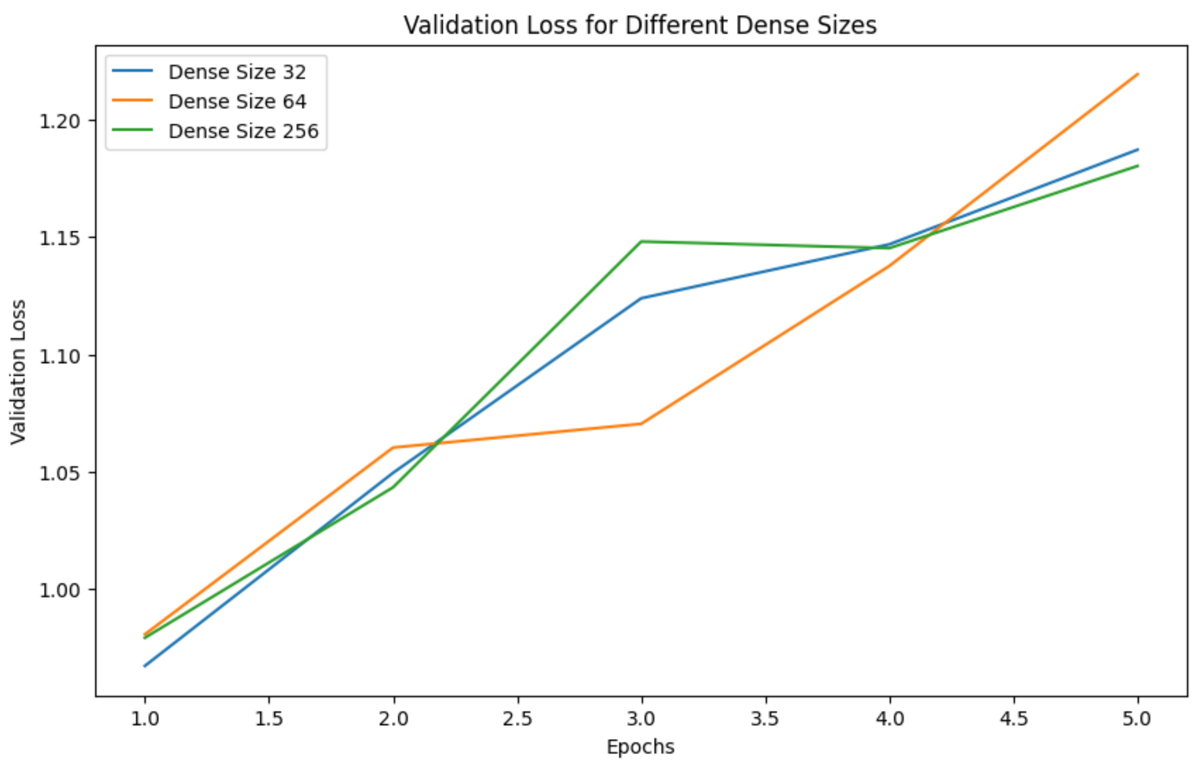
You can see the sample code for testing the embedding size:

```
import matplotlib.pyplot as plt

num_epochs = 5
default_batch_size = 64
embedding_sizes = [10, 30, 50]
validation_losses = []

for embedding_size in embedding_sizes:
    model = create_collaborative_filtering_model(embedding_size)

    history = model.fit(
        [
            train_set['User_id'],
            train_set['Title'],
        ],
        train_set['review/score'],
        epochs=num_epochs,
        batch_size=default_batch_size,
        validation_data=(
            [
                validation_set['User_id'],
                validation_set['Title'],
            ],
            validation_set['review/score']
        )
    )

    # Collect validation losses for plotting
    validation_losses.append(history.history['val_loss'])

# Plotting
plt.figure(figsize=(10, 6))

for i, embedding_size in enumerate(embedding_sizes):
```

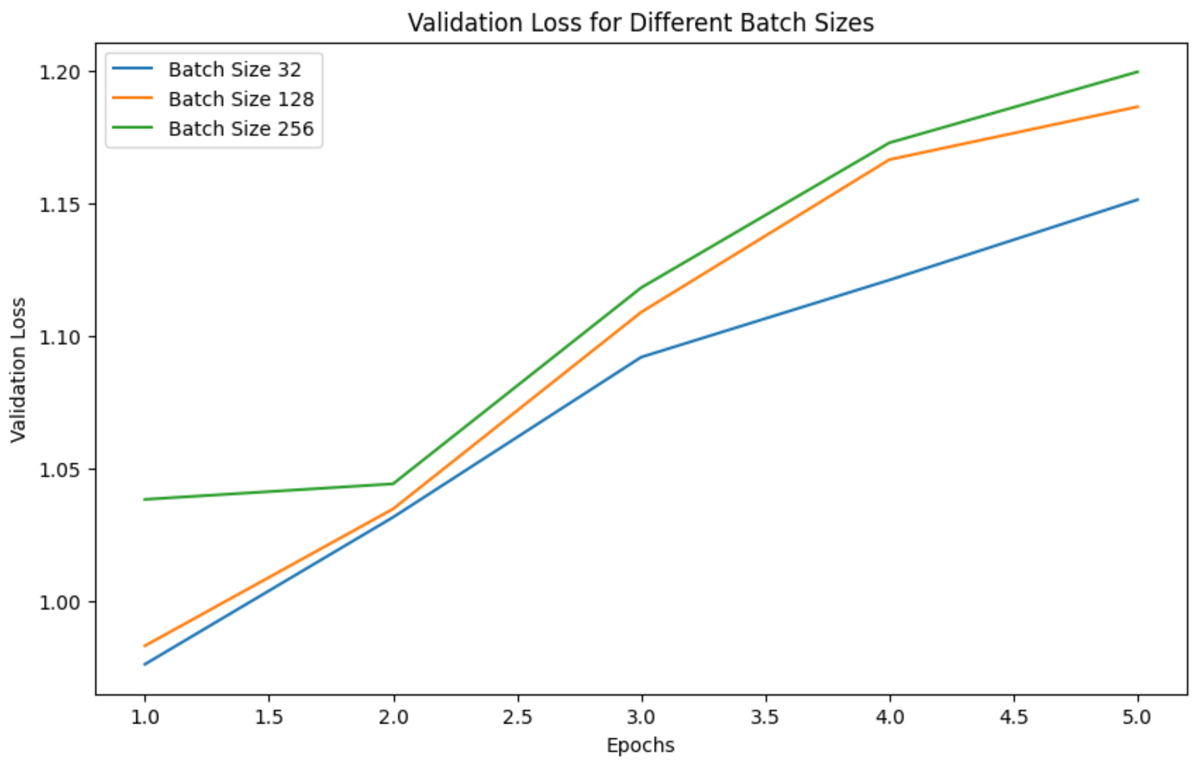
```
                    plt.plot(range(1,    num_epochs    +    1),    validation_losses[i],    label=f'Embedding    Size
{embedding_size}')

          plt.title('Validation Loss for Different Embedding Sizes')
          plt.xlabel('Epochs')
          plt.ylabel('Validation Loss')
          plt.legend()
          plt.show()
```



As can be seen from the plot, the best value for the embedding size is 30 since the validation loss is less than the others.



As can be seen from the plot the best value for the dense size is 256 since the validation loss is less than the others.



As can be seen from the plot, the best value for the batch size is 32 since the validation loss is less than the others.

After running on these tests on our hyperparameters, we selected the best ones from then and trained our best model accordingly :

```
#best model
best_batch_size = 32
best_embedding_size = 30
best_dense_size = 256
model = create_collaborative_filtering_model(best_embedding_size, best_dense_size)
model.fit(
    [
        train_set['User_id'],
        train_set['Title'],
    ],
    train_set['review/score'],
    epochs=num_epochs,
    batch_size = best_batch_size,
    validation_data=(
        [
            validation_set['User_id'],
            validation_set['Title'],
        ],
        validation_set['review/score']
    )
  )
Epoch 1/5
13704/13704 [==============================] - 45s 3ms/step - loss: 0.8260 - val_loss:
1.0008
Epoch 2/5
13704/13704 [==============================] - 42s 3ms/step - loss: 0.5718 - val_loss:
1.0149
Epoch 3/5
13704/13704 [==============================] - 43s 3ms/step - loss: 0.4612 - val_loss:
1.1338
Epoch 4/5
13704/13704 [==============================] - 42s 3ms/step - loss: 0.3738 - val_loss:
1.1145
Epoch 5/5
13704/13704 [==============================] - 43s 3ms/step - loss: 0.3068 - val_loss:
1.1750
```

### 4.1.5 Performance Evaluation

As a final step, we are using our test set to see the true success of our model. We are predicting the review score of each user and book combinations in the test set. We also calculate the loss function result as an initial performance evaluation metric.

```
# Evaluate the model on the test set
test_loss = model.evaluate(
    [
        test_set['User_id'],
        test_set['Title'],
    ],
    test_set['review/score']
)
print(f'Test Loss: {test_loss}')

# Make predictions on the test set
test_predictions = model.predict(
    [
        test_set['User_id'],
        test_set['Title'],
    ]
)

1713/1713 [==============================] - 3s 2ms/step - loss: 1.2784
Test Loss: 1.278433084487915
```

We get a similar result to our validation set results as can be expected. However, to further evaluate the performance of our model, we wanted to find the accuracy of our results. Calculating directly the accuracy in a regression model is not a good way to test the model. Therefore, we need to transform our regression problem into a classification problem by defining a book review by liked or not liked label. To do that, we used a binary label, where we labeled each review with a score greater than 3/5  as liked or 1, and 0 if it was less than 3/5. By doing so, we get an 88.9 percent accuracy metric. In the previous report, as you can remember, this was 88.5 with our default model; we had a slightly better performance with our tuned model.

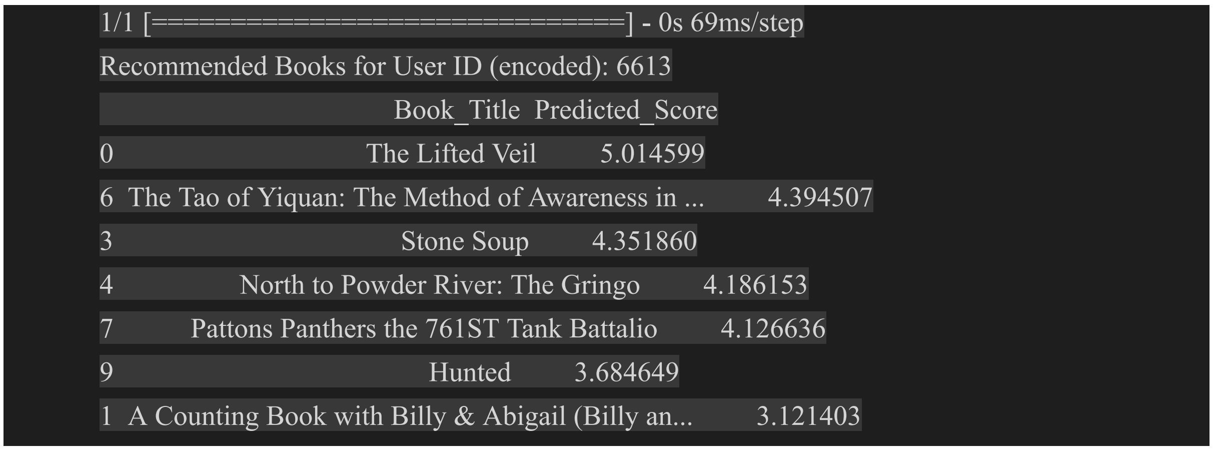
```
from sklearn.metrics import accuracy_score

# Define a threshold for classification
threshold = 3

# Convert the continuous predictions to binary (1 if predicted score >= threshold, 0 otherwise)
binary_predictions = (test_predictions >= threshold).astype(int)
```

```
# Evaluate accuracy based on the threshold
accuracy_binary = accuracy_score((test_set['review/score'] >= threshold).astype(int),
binary_predictions)
print(f'Accuracy (Threshold={threshold}): {accuracy_binary}')

Accuracy (Threshold=3): 0.8897544422957638
```

### 4.1.6 Book Recommendation

To recommend a book to the user, we use a similar approach that we did in our test set evaluation. We recommend books by predicting their review scores and sorting them according to their score. Then we give the highest predicted books as recommendations. However, we do filter books that give a prediction lower than the liked threshold. Currently, we are doing a book recommendation by selecting 10 random books from the dataset, but we can do this for all the books in our dataset.

```
# Select a randomly encoded user ID
user_id_encoded = np.random.choice(train_set['User_id'].unique())

# Choose 10 random book titles from the dataset and encode them
random_book_titles = np.random.choice(train_set['Title'].unique(), size=10, replace=False)

# Convert user ID and book titles to the appropriate format
user_ids_formatted = np.array([user_id_encoded] * 10, dtype='int32')
book_titles_formatted = np.array(random_book_titles, dtype='int32')

# Make predictions with the model
predicted_scores = model.predict([user_ids_formatted, book_titles_formatted])

# Convert predicted scores and book titles to a DataFrame
recommendations_df = pd.DataFrame({
  'Book_Title': title_encoder.inverse_transform(random_book_titles),
  'Predicted_Score': predicted_scores.flatten()
})

# Filter books with a score higher than 3.0 and sort in descending order
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] >
3.0].sort_values(by='Predicted_Score', ascending=False)

# Present recommended books and predicted scores to the user
print(f"Recommended Books for User ID (encoded): {user_id_encoded}")
print(recommended_books)
```

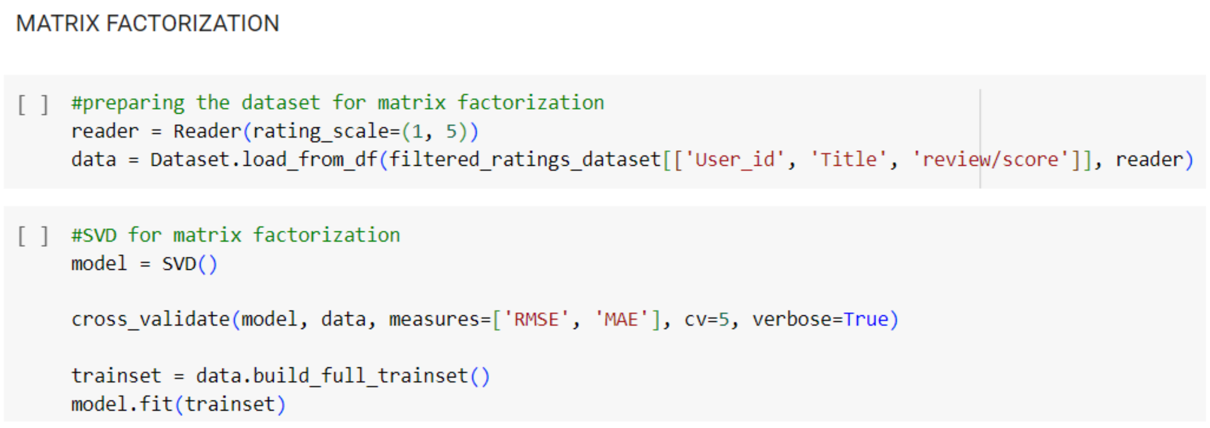


```
1/1 [==============================] - 0s 69ms/step
Recommended Books for User ID (encoded): 6613
                                          Book_Title  Predicted_Score
0                                    The Lifted Veil         5.014599
6  The Tao of Yiquan: The Method of Awareness in ...         4.394507
3                                         Stone Soup         4.351860
4                North to Powder River: The Gringo         4.186153
7             Pattons Panthers the 761ST Tank Battalio         4.126636
9                                             Hunted         3.684649
1  A Counting Book with Billy & Abigail (Billy an...         3.121403
```

## 4.2 Linear Regression

### 4.2.1 Feature Embeddings

Our feature selection and encoding had to be carefully considered as our book recommendation system was going to use linear regression. We concentrated on converting our categorical data, such as book genres and user demographics, into numerical representations using strategies such as matrix factorization. Using this method, we can effectively transform categorical attributes into a format that is compatible with linear regression. To guarantee that the numerical features, such as user ratings, had a similar scale, we also normalized them. This improved the model's capacity to converge on a solution more quickly.



MATRIX FACTORIZATION

```
[ ] #preparing the dataset for matrix factorization
    reader = Reader(rating_scale=(1, 5))
    data = Dataset.load_from_df(filtered_ratings_dataset[['User_id', 'Title', 'review/score']], reader)
```

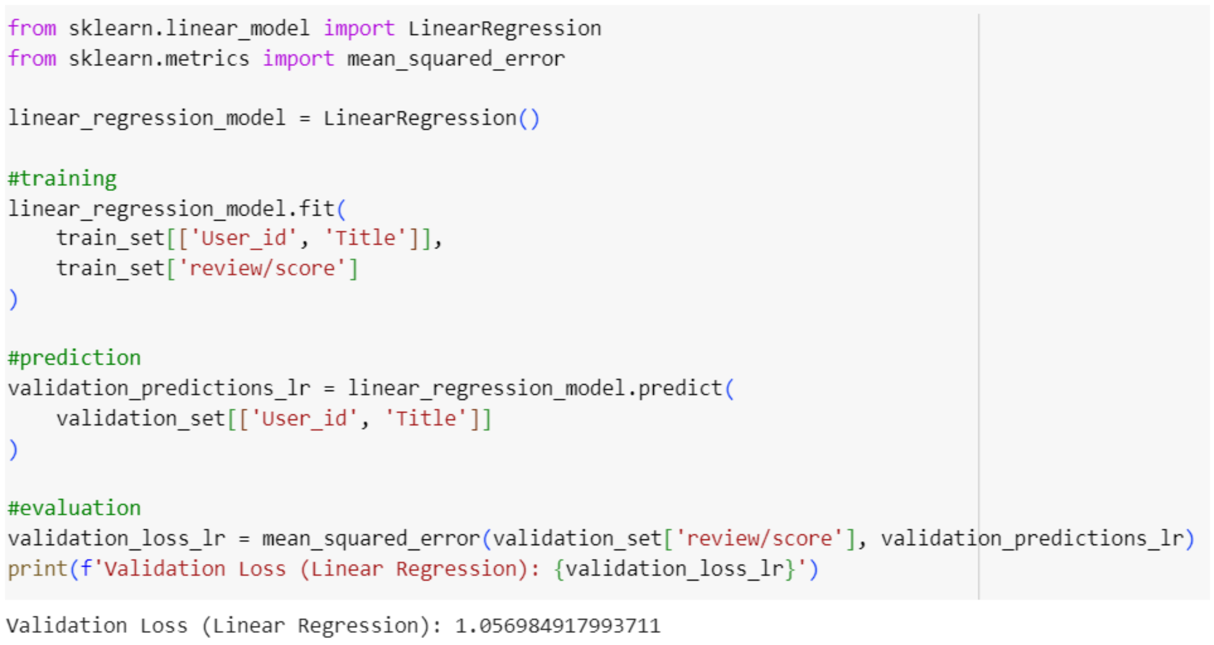
```
[ ] #SVD for matrix factorization
    model = SVD()

    cross_validate(model, data, measures=['RMSE', 'MAE'], cv=5, verbose=True)

    trainset = data.build_full_trainset()
    model.fit(trainset)
```

### 4.2.2 Linear Regression Implementation

As explained in the previous part, we made feature embedding by using matrix factorization. Matrix factorization is done after the preprocessing part as we eliminate some features in the dataset and filter the remaining dataset to make more accurate recommendations. We filter the data to the users who have at least 20 reviews. For the implementation of Linear Regression, we used the libraries such as numpy and pandas.The model employed Ordinary Least Squares (OLS) to minimize prediction errors, and regularization techniques like Ridge or Lasso were used to prevent overfitting. We split our dataset into training and test sets, allowing the model to learn from the former and validate its performance on the latter.

### 4.2.3 Training Our Model

We trained our model by using the LinearRegression function found in sklearn.linear_model library. In the training, we only used the features such as; User_id, Title and scores of the books. After the training, we used the validation set to see the train error. To implement this we have used linear_regression_model.predict from the same library. After the validation, to calculate the validation loss, we used the sklearn.metrics and mean_squared_error function. The results are given below;



```python
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

linear_regression_model = LinearRegression()

#training
linear_regression_model.fit(
    train_set[['User_id', 'Title']],
    train_set['review/score']
)

#prediction
validation_predictions_lr = linear_regression_model.predict(
    validation_set[['User_id', 'Title']]
)

#evaluation
validation_loss_lr = mean_squared_error(validation_set['review/score'], validation_predictions_lr)
print(f'Validation Loss (Linear Regression): {validation_loss_lr}')
```

```
Validation Loss (Linear Regression): 1.056984917993711
```

### 4.2.4 Performance Evaluation

As a final step, we used our test set to evaluate the accuracy of the Linear Regression model. We calculated the loss function result as our performance metric. The implementation of the test algorithm and the resulting loss is given;

```
#prediction on the test set
test_predictions_lr = linear_regression_model.predict(
    test_set[['User_id', 'Title']]
)

#evaluation on the test set
test_loss_lr = mean_squared_error(test_set['review/score'], test_predictions_lr)
print(f'Test Loss (Linear Regression): {test_loss_lr}')

#makingpredictions on the test set
test_predictions_lr = linear_regression_model.predict(
    test_set[['User_id', 'Title']]
)

#create a DataFrame
predictions_df_lr = pd.DataFrame({
    'User_id': test_set['User_id'],
    'Title': test_set['Title'],
    'Actual_Score': test_set['review/score'],
    'Predicted_Score': test_predictions_lr,
})
```

```
Test Loss (Linear Regression): 0.9781400923723856
```

### 4.2.5 Book Recommendation

We apply a similar strategy to book recommendations to the user as we did during test set evaluation. We rank books based on their review scores and make recommendations based on our predictions. Next, we recommend the novels with the highest expected ratings. Books that provide a prediction below the liked threshold are, nevertheless, filtered out. We can perform this for all the books in our dataset, but for now we are doing a book suggestion by choosing ten novels at random from the dataset.

```
#a specific but a random user is selected here
user_id_encoded = np.random.choice(filtered_ratings_dataset['User_id'].unique())
book_titles = np.random.choice(filtered_ratings_dataset['Title'].unique(), size=10, replace=False)

#predicting the ratings of the books for the selected user
predicted_ratings = [model.predict(user_id_encoded, bt).est for bt in book_titles]

#DataFrame creation for the recommendation
recommendations_df = pd.DataFrame({
    'Book_Title': title_encoder.inverse_transform(book_titles),
    'Predicted_Score': predicted_ratings
})
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] > 3.0].sort_values(by='Predicted_Score', ascending=False)

print(f"Recommended Books for User ID {user_id_encoded}:")
print(recommended_books)
```

```
Recommended Books for User ID 1215:
                                        Book_Title  Predicted_Score
6   The Wiccan Mysteries: Ancient Origins & Teachings         5.000000
1                          Not Without Love: Memoirs         4.935849
0   How to Spot Hidden Alcoholics: Using Behaviora...         4.889832
7                                         Hard Candy         4.842178
8        Big Game, Small World: A Basketball Adventure         4.837048
3                                          The Frogs         4.648263
5                                         Nude Diana         4.570011
4                                      Creating a role         4.566534
2                      Madras on Rainy Days : A Novel         4.284920
9         The Witch's Tongue (Charlie Moon Mysteries)         4.087303
```

## 4.3 Decision Tree

### 4.3.1 Feature Embeddings

Our book recommendation system uses a Decision Tree model, so the feature selection and encoding process was customized to meet the specific needs and strengths of the model. Since we have used libraries to implement the models, some processes are just using the functions in the libraries. However, the procedure and the functions of the functions are given.

Decision trees are naturally good at handling both category and numerical data. But in order to achieve the best results, we focused on applying label encoding - that is, turning categorical data into numerical codes - for things like book titles and author names. The discrete character of the data is preserved using this method, which is essential to the tree's decision-making process. We chose strategic discretization to organize values into meaningful categories or bins for continuous variables, such user age or book ratings. This method not only aids in preventing overfitting, which can occur with too granular data in decision trees but also enhances the interpretability of the model, making it easier to understand how different features influence the book recommendations.

### 4.3.2 Decision Tree Implementation

The first step in our implementation procedure was data preparation. We divided the dataset into training and test sets after the feature embedding step in order to train the model and assess its effectiveness. By repeatedly dividing the data according to the characteristics that best predict the user's book preferences, the training set was utilized to expand the decision tree.

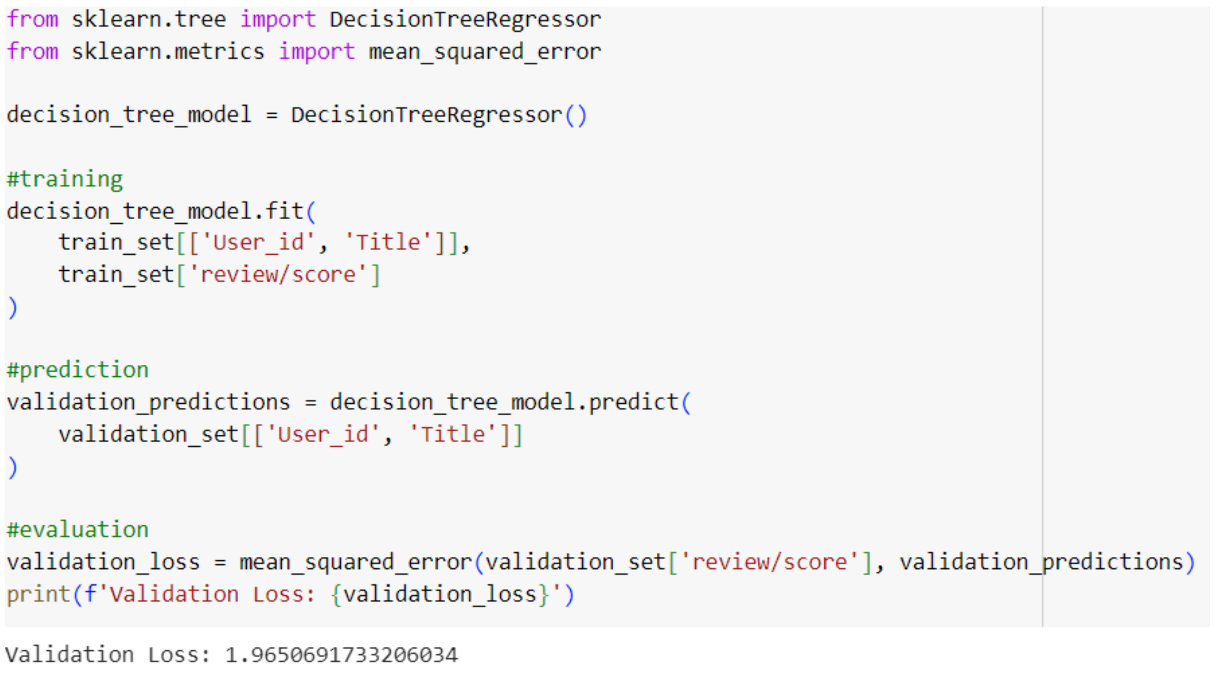
### 4.3.3 Training Our Model

We trained our Decision Tree model using the DecisionTreeRegressor from the sklearn.tree library, focusing on features like User_id, Title, and book scores. This phase involved fitting our model to these features to capture the underlying patterns in our dataset.

After training, we evaluated the model's performance using a validation set. For this, we used the predict method from the DecisionTreeRegressor to generate predictions on the validation data. These predictions were then compared to the actual scores of the books in the validation set.

To measure the accuracy of our model, we calculated the mean squared error (MSE) using the mean_squared_error function from the sklearn.metrics module. This metric helped us assess how well our Decision Tree model was performing, guiding further refinements and adjustments for optimal prediction accuracy.

The implementation of the model and the validation loos are given below;

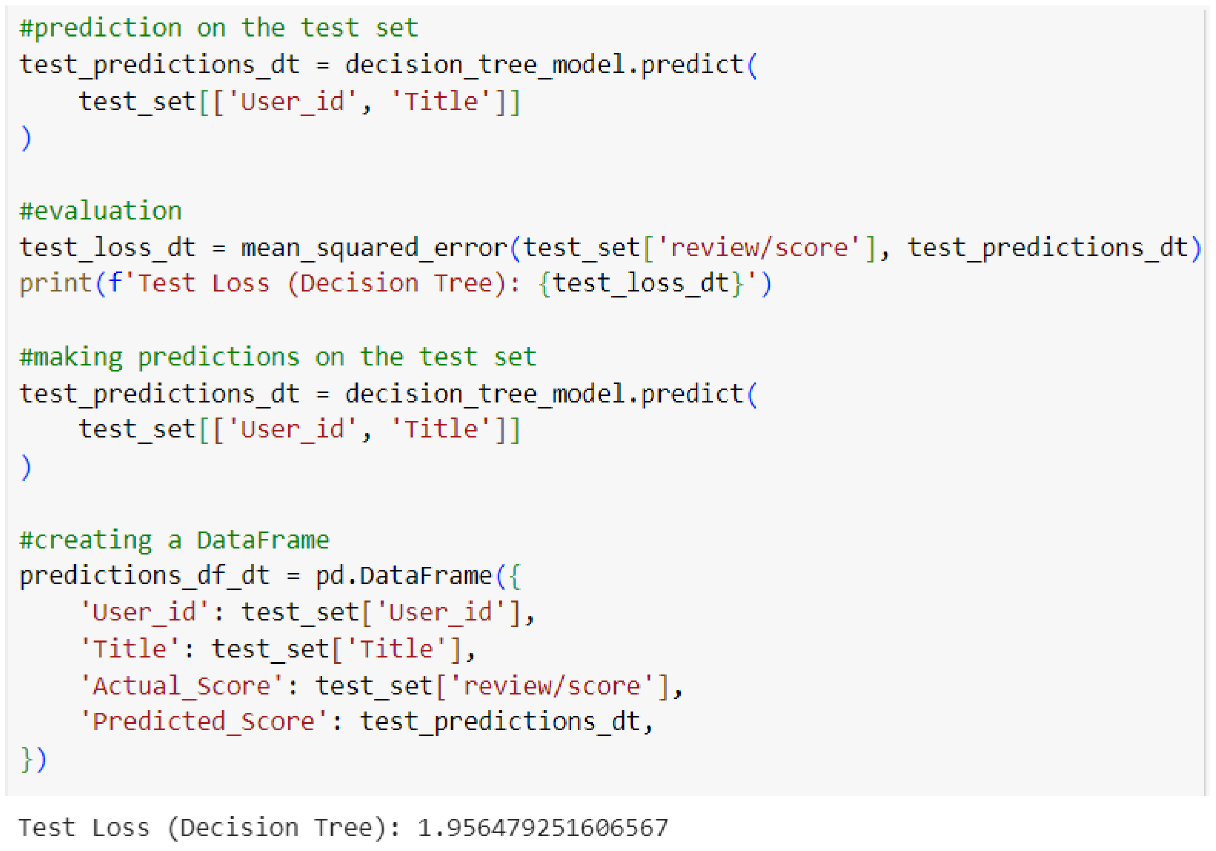


```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

decision_tree_model = DecisionTreeRegressor()

#training
decision_tree_model.fit(
    train_set[['User_id', 'Title']],
    train_set['review/score']
)

#prediction
validation_predictions = decision_tree_model.predict(
    validation_set[['User_id', 'Title']]
)

#evaluation
validation_loss = mean_squared_error(validation_set['review/score'], validation_predictions)
print(f'Validation Loss: {validation_loss}')
```

```
Validation Loss: 1.9650691733206034
```

### 4.3.4 Performance Evaluation

As a final step, we used our test set to evaluate the accuracy of the Decision Tree model. We calculated the loss function result as our performance metric. The implementation of the test algorithm and the resulting loss is given;

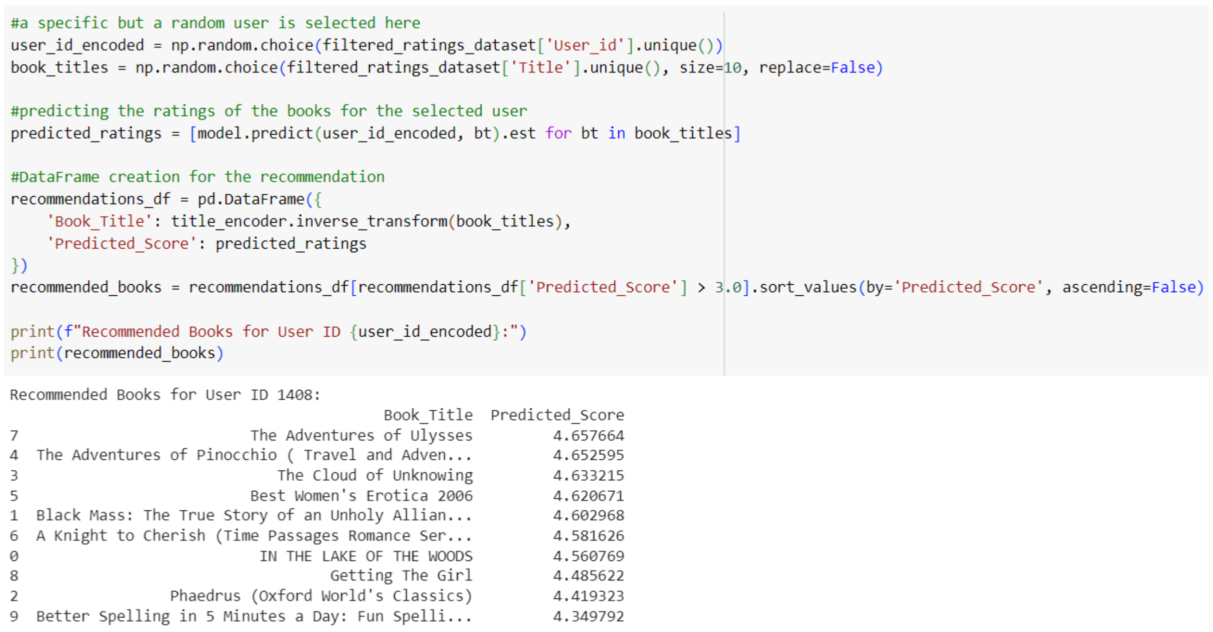


```
#prediction on the test set
test_predictions_dt = decision_tree_model.predict(
    test_set[['User_id', 'Title']]
)

#evaluation
test_loss_dt = mean_squared_error(test_set['review/score'], test_predictions_dt)
print(f'Test Loss (Decision Tree): {test_loss_dt}')

#making predictions on the test set
test_predictions_dt = decision_tree_model.predict(
    test_set[['User_id', 'Title']]
)

#creating a DataFrame
predictions_df_dt = pd.DataFrame({
    'User_id': test_set['User_id'],
    'Title': test_set['Title'],
    'Actual_Score': test_set['review/score'],
    'Predicted_Score': test_predictions_dt,
})
```

```
Test Loss (Decision Tree): 1.956479251606567
```

### 4.3.5 Book Recommendation

We utilize a comparable method to book recommendations to the user as we did in the assessment of the test set. Based on our forecasts and review scores, we rank books and offer suggestions. We then suggest the novels that have the greatest anticipated ratings. However, books that offer a prediction that is less than the liked level are excluded. Although we can accomplish this for every book in our dataset, for the time being we are suggesting books based on a random selection of ten novels.



```
#a specific but a random user is selected here
user_id_encoded = np.random.choice(filtered_ratings_dataset['User_id'].unique())
book_titles = np.random.choice(filtered_ratings_dataset['Title'].unique(), size=10, replace=False)

#predicting the ratings of the books for the selected user
predicted_ratings = [model.predict(user_id_encoded, bt).est for bt in book_titles]

#DataFrame creation for the recommendation
recommendations_df = pd.DataFrame({
    'Book_Title': title_encoder.inverse_transform(book_titles),
    'Predicted_Score': predicted_ratings
})
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] > 3.0].sort_values(by='Predicted_Score', ascending=False)

print(f"Recommended Books for User ID {user_id_encoded}:")
print(recommended_books)
```

```
Recommended Books for User ID 1408:
                                          Book_Title  Predicted_Score
7                          The Adventures of Ulysses         4.657664
4  The Adventures of Pinocchio ( Travel and Adven...         4.652595
3                             The Cloud of Unknowing         4.633215
5                          Best Women's Erotica 2006         4.620671
1  Black Mass: The True Story of an Unholy Allian...         4.602968
6  A Knight to Cherish (Time Passages Romance Ser...         4.581626
0                           IN THE LAKE OF THE WOODS         4.560769
8                                    Getting The Girl         4.485622
2                  Phaedrus (Oxford World's Classics)         4.419323
9  Better Spelling in 5 Minutes a Day: Fun Spelli...         4.349792
```

## 4.4 Random Forest

### 4.4.1 Feature Embeddings

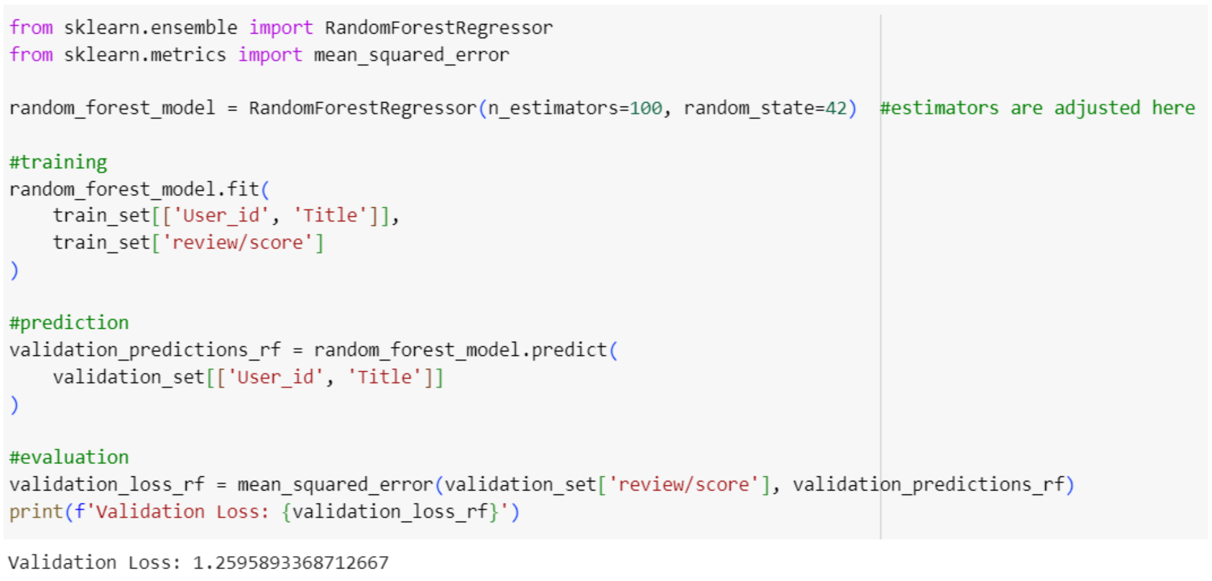
The Random Forest model in our book recommendation system demands a meticulous feature selection and encoding process, adapted to capitalize on the algorithm's inherent strengths. For categorical variables such as book titles and authors, we employed one-hot encoding to convert them into a binary matrix, which is more suitable for the Random Forest's operation. This process preserves the discrete nature of categorical data, essential for maintaining the integrity of the model's decision-making process. We meticulously chose label encoding for user IDs to maintain a lightweight model while ensuring that the numerical translation of categorical data does not dilute the interpretability of the results. These feature embeddings are crucial as they directly influence the Random Forest's ability to discern the intricate relationships between different books and user preferences.

### 4.4.2 Random Forest Implementation

Our implementation strategy for the Random Forest model involved leveraging the RandomForestRegressor class from the sklearn.ensemble library. This ensemble learning method is particularly adept at handling the complex, non-linear relationships inherent in user-item interactions. By training on an amalgamation of user IDs, titles, and book scores, our model aimed to encapsulate the subtle patterns present within the dataset, thus enhancing the reliability and relevance of the book recommendations.

### 4.4.3 Training Our Model

Our Random Forest model was trained using the RandomForestRegressor from the sklearn library, focusing on user and book features. Post-training, we evaluated the model's performance with a validation set. To assess its accuracy, we calculated the mean squared error (MSE), which informed our adjustments for optimal prediction accuracy. The validation loss helped us gauge the effectiveness of the model and guided further refinements for improved accuracy.



```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

random_forest_model = RandomForestRegressor(n_estimators=100, random_state=42)  #estimators are adjusted here

#training
random_forest_model.fit(
    train_set[['User_id', 'Title']],
    train_set['review/score']
)

#prediction
validation_predictions_rf = random_forest_model.predict(
    validation_set[['User_id', 'Title']]
)

#evaluation
validation_loss_rf = mean_squared_error(validation_set['review/score'], validation_predictions_rf)
print(f'Validation Loss: {validation_loss_rf}')
```

```
Validation Loss: 1.2595893368712667
```

### 4.4.4 Performance Evaluation

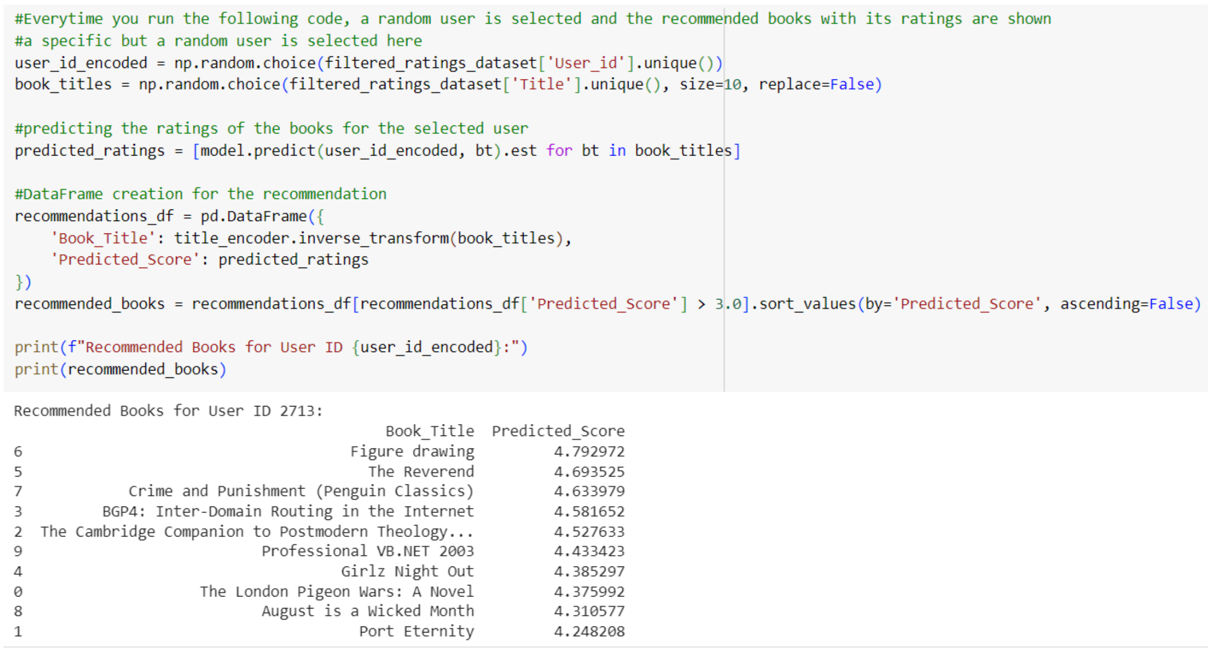
Evaluating the performance of our Random Forest model was conducted with rigor, utilizing the test dataset as the ultimate arbiter of accuracy. The mean squared error (MSE) on the test set provided a quantitative measure of the model's prediction error, enabling us to compare its predictive accuracy with the Decision Tree model. A lower test loss indicated a more precise model, and the Random Forest model showed an impressive ability to generalize from the training data to unseen data, a testament to the efficacy of the feature embeddings and the robustness of the model architecture.

```python
#prediction
test_predictions_rf = random_forest_model.predict(
    test_set[['User_id', 'Title']]
)

#evaluation
test_loss_rf = mean_squared_error(test_set['review/score'], test_predictions_rf)
print(f'Test Loss (Random Forest): {test_loss_rf}')

#making predictions on the test set
test_predictions_rf = random_forest_model.predict(
    test_set[['User_id', 'Title']]
)

#creating a DataFrame
predictions_df_rf = pd.DataFrame({
    'User_id': test_set['User_id'],
    'Title': test_set['Title'],
    'Actual_Score': test_set['review/score'],
    'Predicted_Score': test_predictions_rf,
})
```

```
Test Loss (Random Forest): 1.2320484649392724
```

### 4.4.5 Book Recommendation

The final phase of our Random Forest model involved translating the statistical accuracy into actionable book recommendations. Utilizing a method similar to that employed in the Decision Tree approach, we ranked books based on their predicted scores and user review scores. The system was designed to suggest titles that are expected to resonate well with the user's preferences. This personalized approach to recommendation, while currently constrained to a subset of ten novels for feasibility, has the potential to be scaled up to encompass the entire dataset, providing a comprehensive and dynamic recommendation experience for the users.



```
#Everytime you run the following code, a random user is selected and the recommended books with its ratings are shown
#a specific but a random user is selected here
user_id_encoded = np.random.choice(filtered_ratings_dataset['User_id'].unique())
book_titles = np.random.choice(filtered_ratings_dataset['Title'].unique(), size=10, replace=False)

#predicting the ratings of the books for the selected user
predicted_ratings = [model.predict(user_id_encoded, bt).est for bt in book_titles]

#DataFrame creation for the recommendation
recommendations_df = pd.DataFrame({
    'Book_Title': title_encoder.inverse_transform(book_titles),
    'Predicted_Score': predicted_ratings
})
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] > 3.0].sort_values(by='Predicted_Score', ascending=False)

print(f"Recommended Books for User ID {user_id_encoded}:")
print(recommended_books)
```

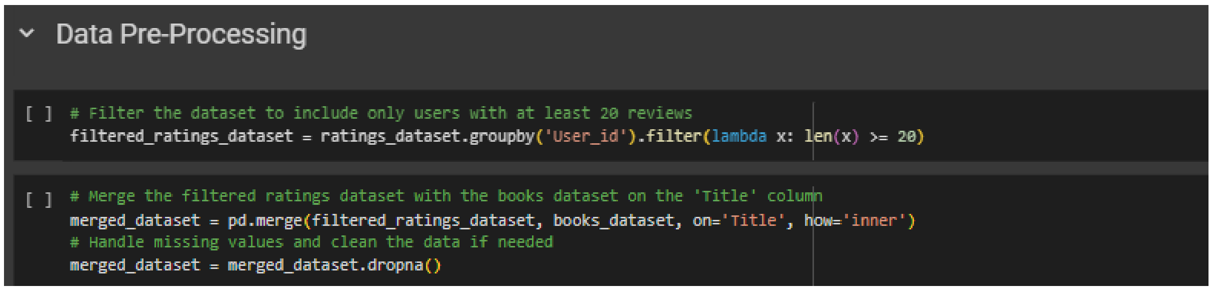
```
Recommended Books for User ID 2713:
                                         Book_Title  Predicted_Score
6                                    Figure drawing         4.792972
5                                      The Reverend         4.693525
7                Crime and Punishment (Penguin Classics)    4.633979
3              BGP4: Inter-Domain Routing in the Internet   4.581652
2  The Cambridge Companion to Postmodern Theology...        4.527633
9                             Professional VB.NET 2003      4.433423
4                                    Girlz Night Out        4.385297
0                        The London Pigeon Wars: A Novel    4.375992
8                             August is a Wicked Month      4.310577
1                                      Port Eternity        4.248208
```

## 4.5 Content Based Filtering

In this phase of our project, we implemented a content-based filtering approach which is often used for exploring the similarities between the items and their similarities in between to complement our collaborative filtering model. This entails leveraging book features such as descriptions, authors, categories, and publisher names to enhance the accuracy of our recommendations [2]. We utilize cosine similarity techniques with TF-IDFs and Bert Model to quantify the similarity between books based on their content [12]. For this, we first tried to add the content-based filtering as feature embeddings directly into the neural network but it turns out to be overfitting to the data. Therefore, we chose to integrate it as an independent filtering that selects the predetermined number of books based on the content and gives it to collaborative based filtering rather than selecting 10 random books in the current implementation.

### 4.5.1 Additional Preprocessing for Content Based Filtering

The preprocessed data we used for the other filters were not enough for the content based filtering. We needed to also use the book's description data. Therefore, we merged the previous filtered_ratings_dataset with books_dataset.



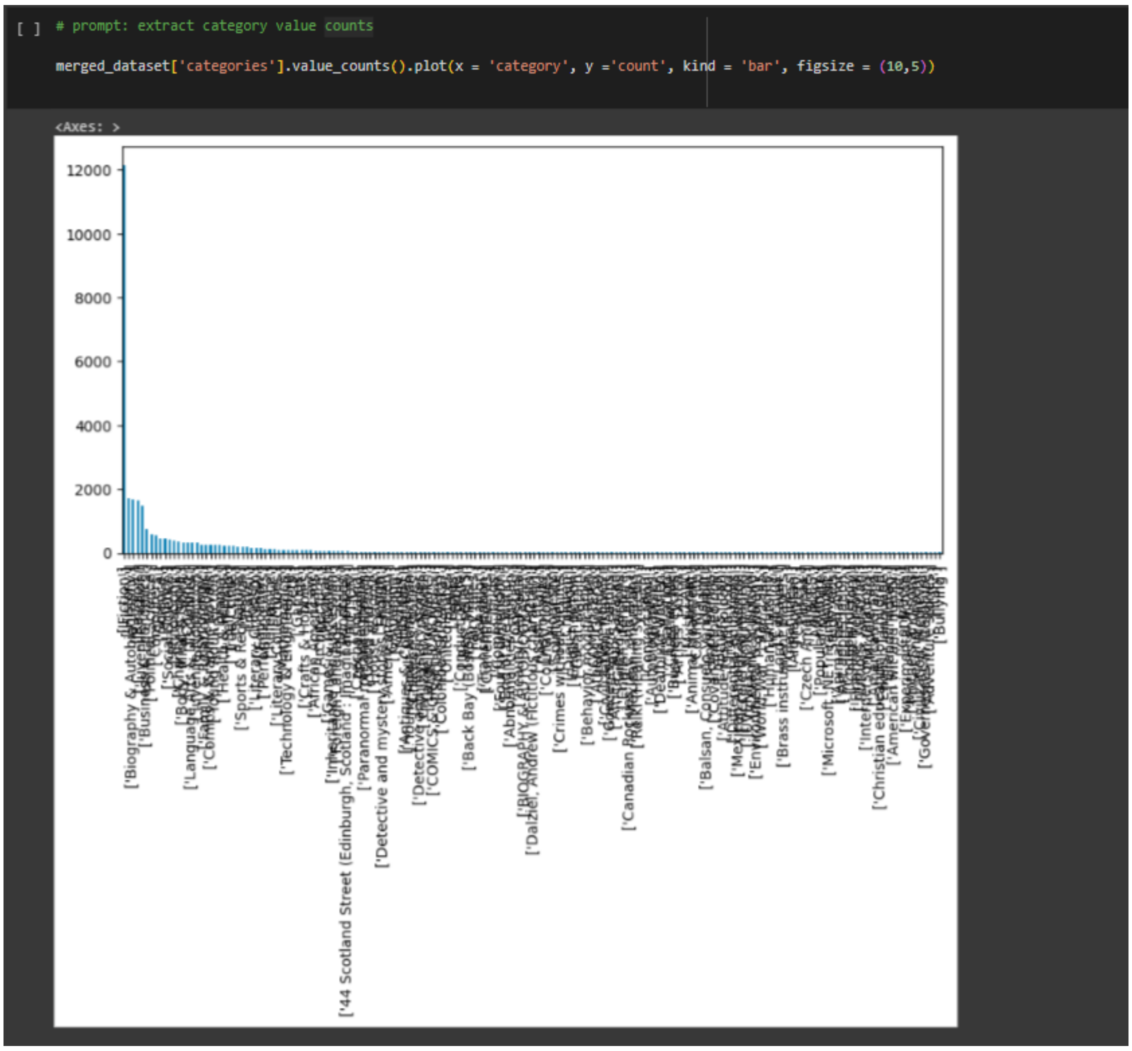
Data Pre-Processing

```
# Filter the dataset to include only users with at least 20 reviews
filtered_ratings_dataset = ratings_dataset.groupby('User_id').filter(lambda x: len(x) >= 20)
```

```
# Merge the filtered ratings dataset with the books dataset on the 'Title' column
merged_dataset = pd.merge(filtered_ratings_dataset, books_dataset, on='Title', how='inner')
# Handle missing values and clean the data if needed
merged_dataset = merged_dataset.dropna()
```

### 4.5.2 Exploring the Data for Content Based Filtering

We explored our data further by checking the distribution of the available categories. Even though the category number is huge, the distribution is not even. Therefore, we decided to not use it while calculating the similarity between two books in the final model.

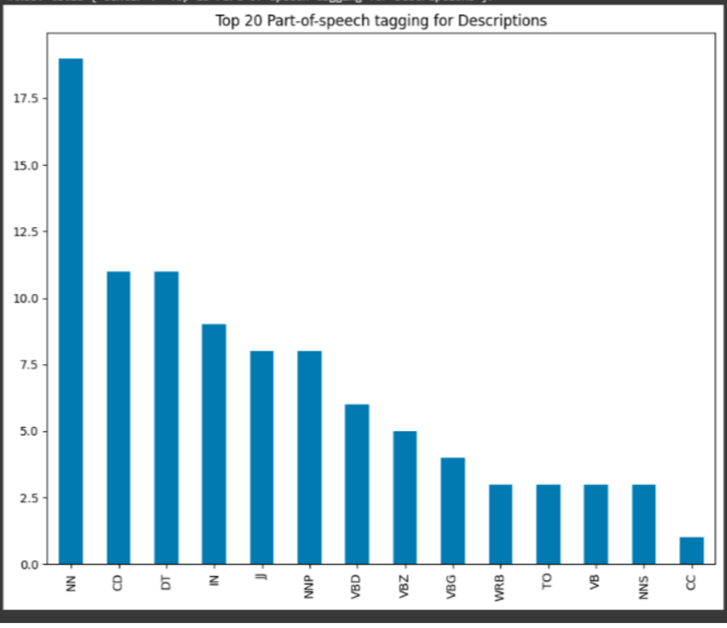


```
[ ]  # prompt: extract category value counts

     merged_dataset['categories'].value_counts().plot(x = 'category', y ='count', kind = 'bar', figsize = (10,5))
```

```
<Axes: >
```

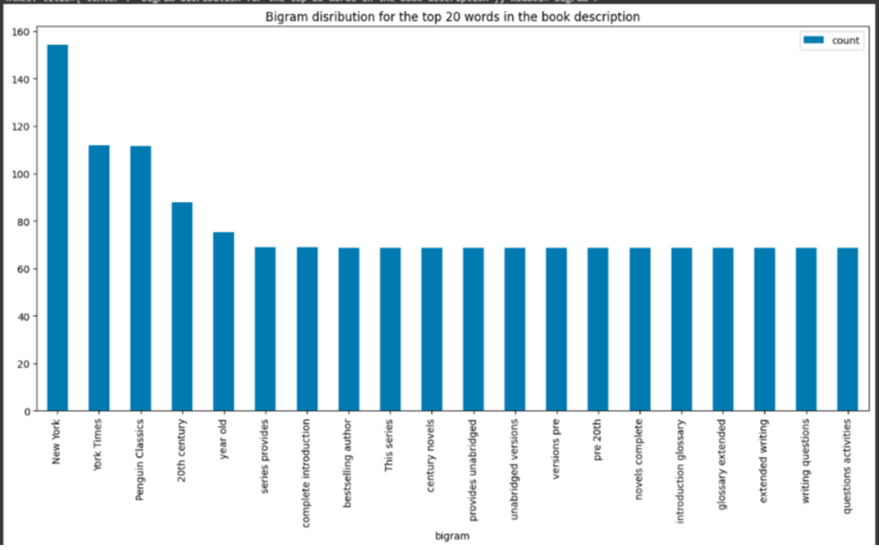
Then, description and title fields of the books were determined to be used for finding the similarities of the books. For this, we used the NLTK library and its datasets. After importing the necessary library in our code, we applied it to the description field for finding the distribution of the part-of-speech taggings.

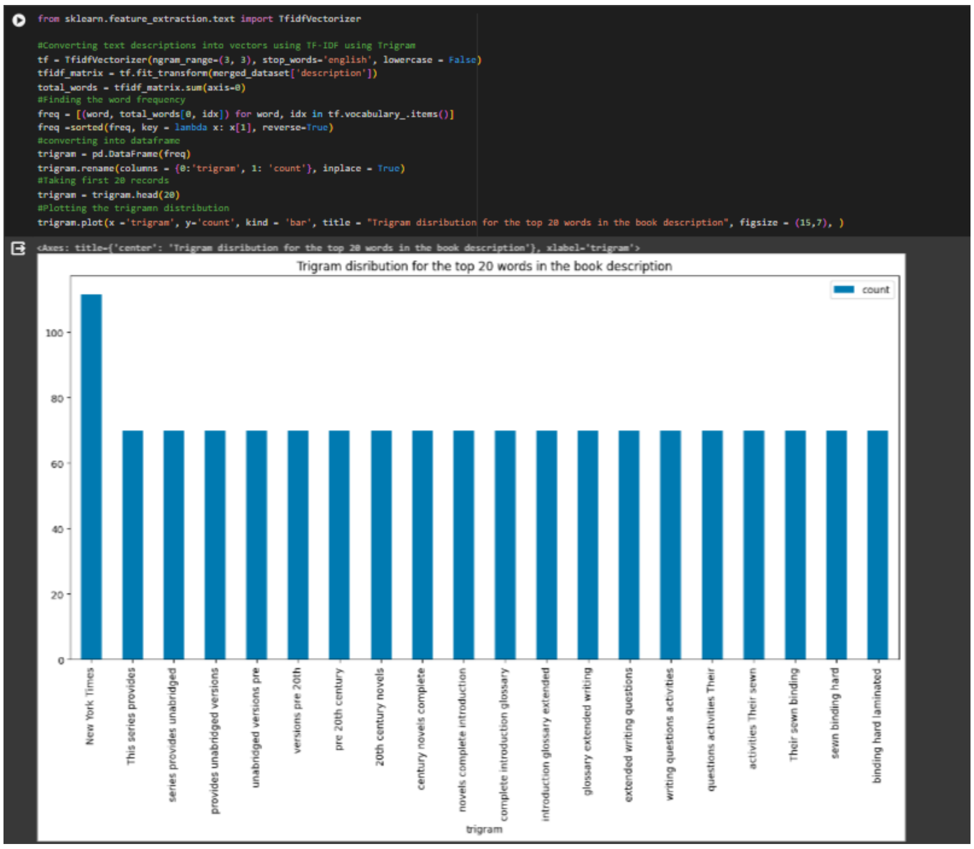
```
[ ] import nltk
    nltk.download('punkt')
    nltk.download('averaged_perceptron_tagger')
    from textblob import TextBlob
    blob = TextBlob(str(merged_dataset['description']))
    pos_df = pd.DataFrame(blob.tags, columns = ['word' , 'pos'])
    pos_df = pos_df.pos.value_counts()[:20]
    pos_df.plot(kind = 'bar', figsize=(10, 8), title = "Top 20 Part-of-speech tagging for Descriptions")

    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    [nltk_data] Downloading package averaged_perceptron_tagger to
    [nltk_data]     /root/nltk_data...
    [nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
    <Axes: title={'center': 'Top 20 Part-of-speech tagging for Descriptions'}>
```



While calculating the TF-IDF score of the books, we need to divide our data into phrases. Therefore, we examined the bigram and trigram distribution of the data as shown below.

```
from sklearn.feature_extraction.text import TfidfVectorizer

#Converting text descriptions into vectors using TF-IDF using Bigram
tf = TfidfVectorizer(ngram_range=(2, 2), stop_words='english', lowercase = False)
tfidf_matrix = tf.fit_transform(merged_dataset['description'])
total_words = tfidf_matrix.sum(axis=0)
#Finding the word frequency
freq = [(word, total_words[0, idx]) for word, idx in tf.vocabulary_.items()]
freq =sorted(freq, key = lambda x: x[1], reverse=True)
#converting into dataframe
bigram = pd.DataFrame(freq)
bigram.rename(columns = {0:'bigram', 1: 'count'}, inplace = True)
#Taking first 20 records
bigram = bigram.head(20)
#Plotting the bigram distribution
bigram.plot(x ='bigram', y='count', kind = 'bar', title = "Bigram disribution for the top 20 words in the book description", figsize = (15,7), )

<Axes: title={'center': 'Bigram disribution for the top 20 words in the book description'}, xlabel='bigram'>
```



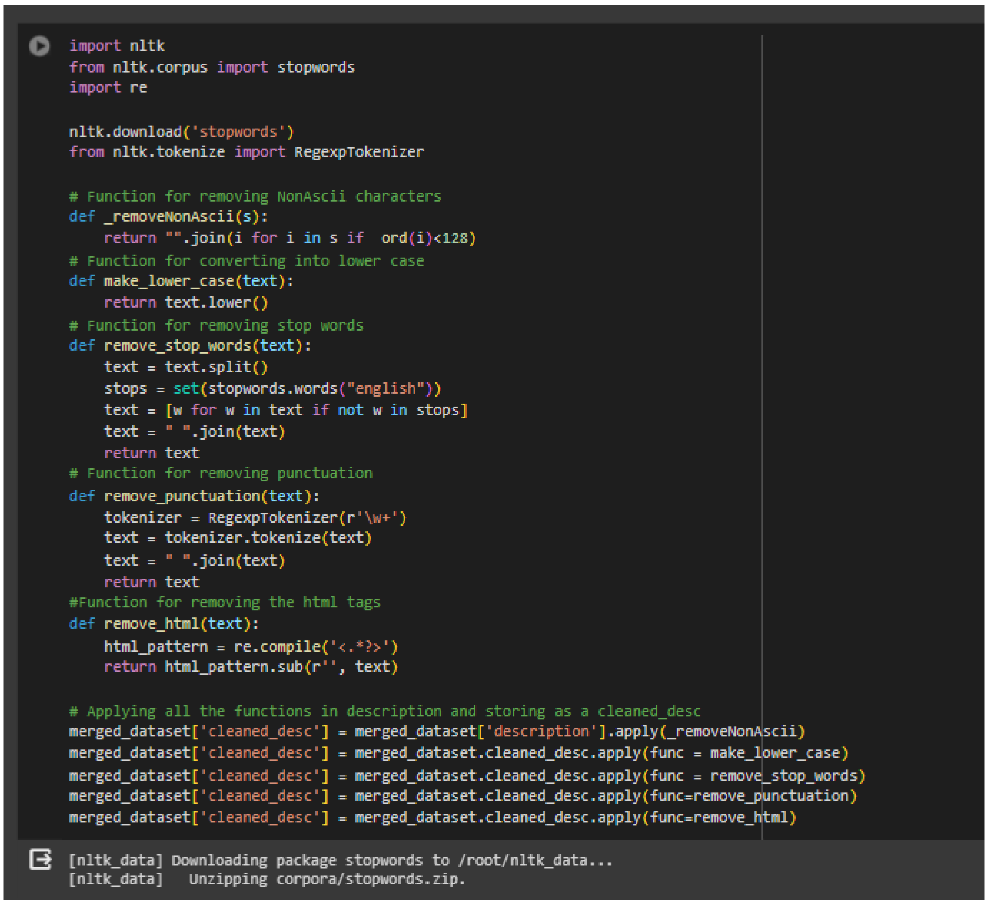


```
from sklearn.feature_extraction.text import TfidfVectorizer

#Converting text descriptions into vectors using TF-IDF using Trigram
tf = TfidfVectorizer(ngram_range=(3, 3), stop_words='english', lowercase = False)
tfidf_matrix = tf.fit_transform(merged_dataset['description'])
total_words = tfidf_matrix.sum(axis=0)
#Finding the word frequency
freq = [(word, total_words[0, idx]) for word, idx in tf.vocabulary_.items()]
freq =sorted(freq, key = lambda x: x[1], reverse=True)
#converting into dataframe
trigram = pd.DataFrame(freq)
trigram.rename(columns = {0:'trigram', 1: 'count'}, inplace = True)
#Taking first 20 records
trigram = trigram.head(20)
#Plotting the trigramn distribution
trigram.plot(x ='trigram', y='count', kind = 'bar', title = "Trigram disribution for the top 20 words in the book description", figsize = (15,7), )
```

```
<Axes: title={'center': 'Trigram disribution for the top 20 words in the book description'}, xlabel='trigram'>
```

Trigram disribution for the top 20 words in the book description
count
100
80
60
40
20
0
New York Times
This series provides
series provides unabridged
provides unabridged versions
unabridged versions pre
versions pre 20th
pre 20th century
20th century novels
century novels complete
novels complete introduction
complete introduction glossary
introduction glossary extended
glossary extended writing
extended writing questions
writing questions activities
questions activities Their
activities Their sewn
Their sewn binding
sewn binding hard
binding hard laminated
trigram

After the exploration was done, we cleaned the description data and saved it to the cleaned_desc field in the dataset.



```
import nltk
from nltk.corpus import stopwords
import re

nltk.download('stopwords')
from nltk.tokenize import RegexpTokenizer

# Function for removing NonAscii characters
def _removeNonAscii(s):
    return "".join(i for i in s if  ord(i)<128)
# Function for converting into lower case
def make_lower_case(text):
    return text.lower()
# Function for removing stop words
def remove_stop_words(text):
    text = text.split()
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops]
    text = " ".join(text)
    return text
# Function for removing punctuation
def remove_punctuation(text):
    tokenizer = RegexpTokenizer(r'\w+')
    text = tokenizer.tokenize(text)
    text = " ".join(text)
    return text
#Function for removing the html tags
def remove_html(text):
    html_pattern = re.compile('<.*?>')
    return html_pattern.sub(r'', text)

# Applying all the functions in description and storing as a cleaned_desc
merged_dataset['cleaned_desc'] = merged_dataset['description'].apply(_removeNonAscii)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func = make_lower_case)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func = remove_stop_words)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func=remove_punctuation)
merged_dataset['cleaned_desc'] = merged_dataset.cleaned_desc.apply(func=remove_html)
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

### 4.5.3 Content Based Filtering Implementation

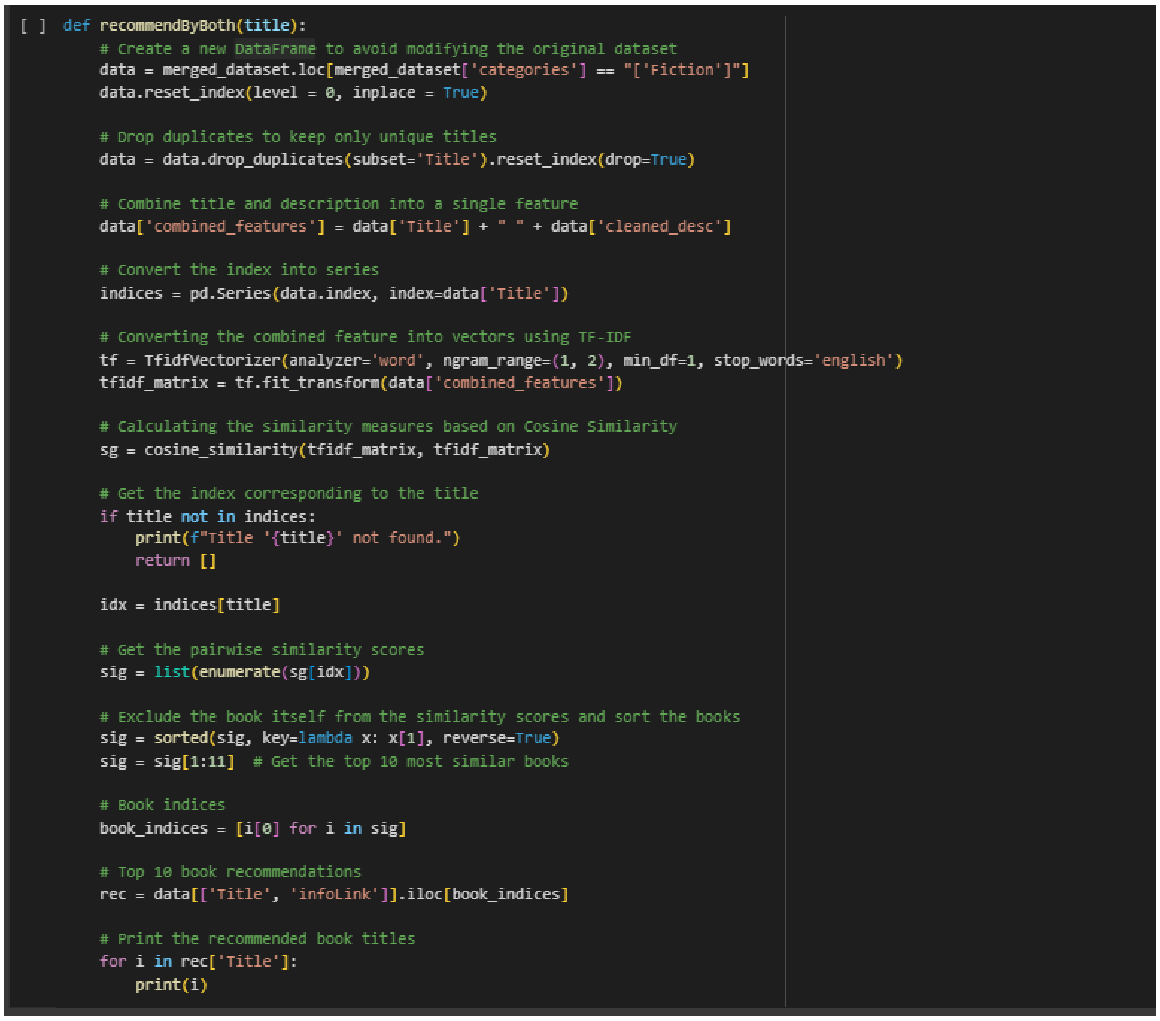
For the content based filtering implementation, we tried two approaches in our code. First approach was to calculate the TF-IDF vectors of the books and find their similarities with the cosine similarity method.

While implementing content based filtering, we combined title and cleaned description to a new field called combined_features to give some importance also to the title of the book.

TF-IDF, which stands for Term Frequency-Inverse Document Frequency, is a numerical statistic used in natural language processing and information retrieval to evaluate the importance of a word in a document relative to a collection of documents (corpus) [14]. In the context of the book recommendation system, TF-IDF can be applied to analyze the textual content of the books, such as their descriptions or other relevant text. In the context of our implementation, the term frequency of each word in its combined features was calculated for each book. This showed how frequently each word appears in that specific book's combined features. Then, the inverse document frequency for each word in the entire corpus of book combined features was calculated. This was important to see how unique or rare words were across all books. The TF-IDF score for each word in each book was calculated by the product of its TF and IDF scores. This score reflects the importance of the word in that specific book and across the entire corpus. In the end, we had a TF-IDF matrix where each row corresponds to a book, each column corresponds to a unique word in the corpus, and the values represent the TF-IDF scores. This matrix captured the importance of words in each book's combined features relative to the entire collection of book combined features.

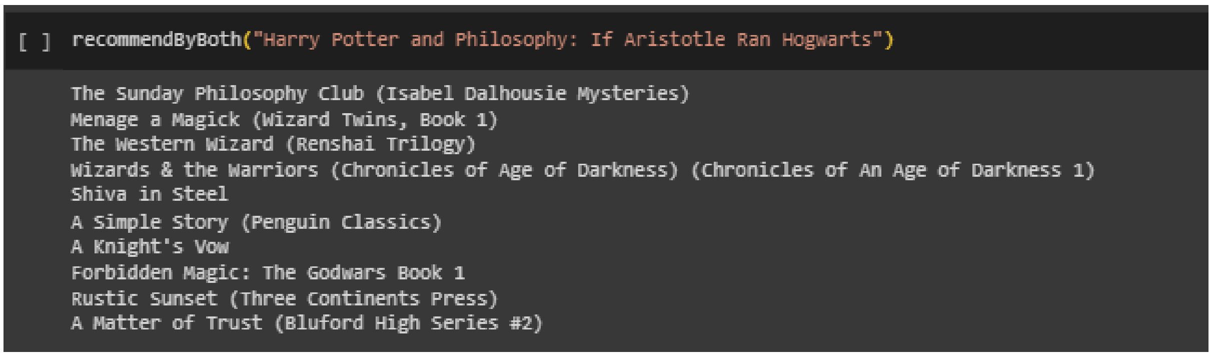
Cosine similarity is a measure of similarity between two non-zero vectors in an inner product space [15]. In the context of the book recommendation system, it's often used to quantify the similarity between the TF-IDF vectors of different books [15]. In the code, after obtaining the TF-IDF matrix for book combined features, we used the cosine_similarity function from scikit-learn to calculate the cosine similarity matrix between all pairs of the books. Then, the cosine similarity matrix was used to find books that are most similar to a given book. For a target book, we identified other books with high cosine similarity scores, indicating similar content based on their combined features. Then we selected the top 10 books to recommend.

For the comparison purposes, the initial recommendation model was used on the category “Fiction” to later compare the result with the Bert Model. We did this to compare them in a smaller subset of the data.



```
[ ] def recommendByBoth(title):
        # Create a new DataFrame to avoid modifying the original dataset
        data = merged_dataset.loc[merged_dataset['categories'] == "['Fiction']"]
        data.reset_index(level = 0, inplace = True)

        # Drop duplicates to keep only unique titles
        data = data.drop_duplicates(subset='Title').reset_index(drop=True)

        # Combine title and description into a single feature
        data['combined_features'] = data['Title'] + " " + data['cleaned_desc']

        # Convert the index into series
        indices = pd.Series(data.index, index=data['Title'])

        # Converting the combined feature into vectors using TF-IDF
        tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=1, stop_words='english')
        tfidf_matrix = tf.fit_transform(data['combined_features'])

        # Calculating the similarity measures based on Cosine Similarity
        sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

        # Get the index corresponding to the title
        if title not in indices:
            print(f"Title '{title}' not found.")
            return []

        idx = indices[title]

        # Get the pairwise similarity scores
        sig = list(enumerate(sg[idx]))

        # Exclude the book itself from the similarity scores and sort the books
        sig = sorted(sig, key=lambda x: x[1], reverse=True)
        sig = sig[1:11]  # Get the top 10 most similar books

        # Book indices
        book_indices = [i[0] for i in sig]

        # Top 10 book recommendations
        rec = data[['Title', 'infoLink']].iloc[book_indices]

        # Print the recommended book titles
        for i in rec['Title']:
            print(i)
```

For the book "Harry Potter and Philosophy: If Aristotle Ran Hogwarts", the model marked the following books as most similar:

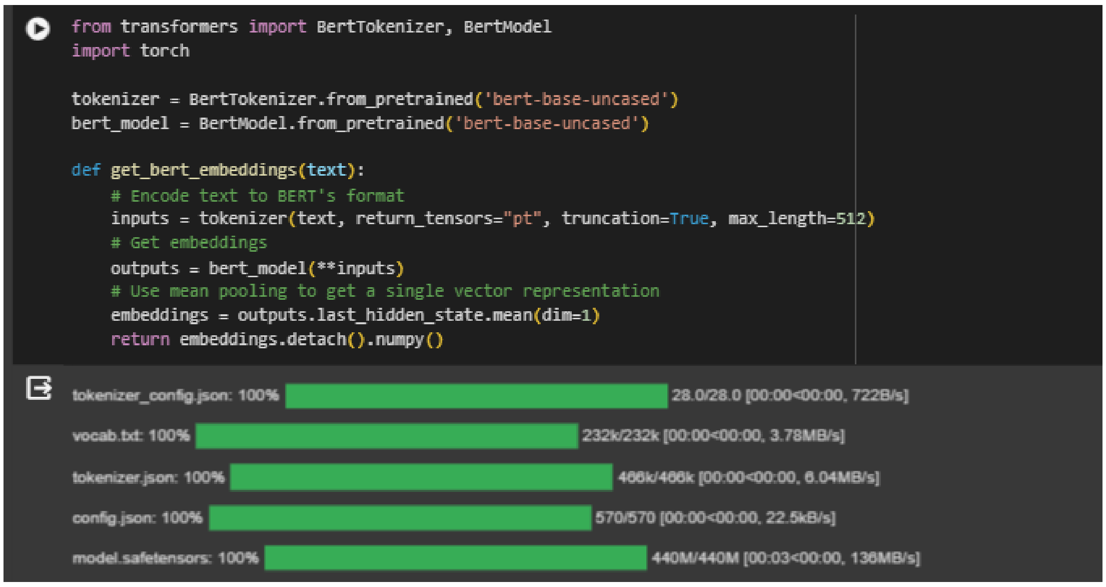


```
[ ] recommendByBoth("Harry Potter and Philosophy: If Aristotle Ran Hogwarts")

    The Sunday Philosophy Club (Isabel Dalhousie Mysteries)
    Menage a Magick (Wizard Twins, Book 1)
    The Western Wizard (Renshai Trilogy)
    Wizards & the Warriors (Chronicles of Age of Darkness) (Chronicles of An Age of Darkness 1)
    Shiva in Steel
    A Simple Story (Penguin Classics)
    A Knight's Vow
    Forbidden Magic: The Godwars Book 1
    Rustic Sunset (Three Continents Press)
    A Matter of Trust (Bluford High Series #2)
```

In accordance with TA's request on progress meeting, we implemented a second approach which uses Bert model with cosine similarities.

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a powerful pre-trained language model developed by Google [16]. It's designed to understand the context of words in a sentence and capture complex relationships within the text [16]. Unlike traditional models, BERT considers both the left and right context of each word, making it particularly effective for various natural language processing tasks, including text classification, named entity recognition, and text similarity [16]. It's trained on massive amounts of text data, allowing it to learn rich representations of words [16]. In the context of our implementation, we used BERT embeddings to represent the textual content of book combined features more effectively than traditional methods like TF-IDF. BERT embeddings capture semantic relationships between words and phrases. Therefore, we integrated it to enhance the performance of our content-based recommendation system by capturing more nuanced relationships in the book combined features. However, since using BERT involves additional computational resources compared to TF-IDF, we chose to continue with our first approach with the TF-IDF matrix.

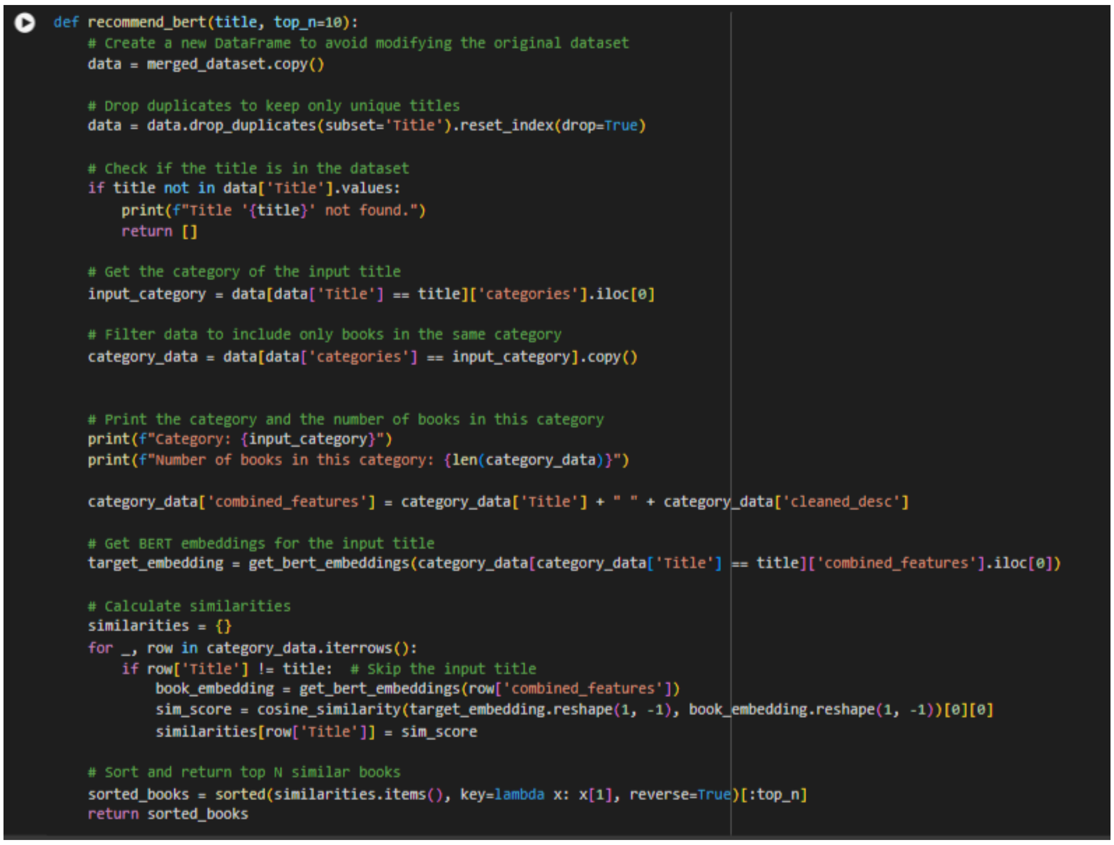
In implementation, we first defined the method for embeddings as follows:



```
from transformers import BertTokenizer, BertModel
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
bert_model = BertModel.from_pretrained('bert-base-uncased')

def get_bert_embeddings(text):
    # Encode text to BERT's format
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512)
    # Get embeddings
    outputs = bert_model(**inputs)
    # Use mean pooling to get a single vector representation
    embeddings = outputs.last_hidden_state.mean(dim=1)
    return embeddings.detach().numpy()
```

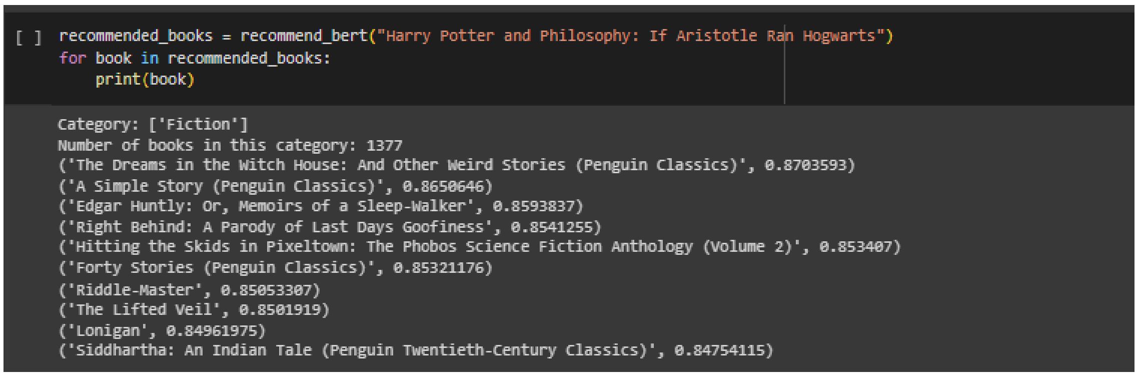
```
tokenizer_config.json: 100%  28.0/28.0 [00:00<00:00, 722B/s]
vocab.txt: 100%  232k/232k [00:00<00:00, 3.78MB/s]
tokenizer.json: 100%  466k/466k [00:00<00:00, 6.04MB/s]
config.json: 100%  570/570 [00:00<00:00, 22.5kB/s]
model.safetensors: 100%  440M/440M [00:03<00:00, 136MB/s]
```

Then, we defined the model similar to the first approach but with Bert embeddings instead of TF-IDF vectors:



```
def recommend_bert(title, top_n=10):
    # Create a new DataFrame to avoid modifying the original dataset
    data = merged_dataset.copy()

    # Drop duplicates to keep only unique titles
    data = data.drop_duplicates(subset='Title').reset_index(drop=True)

    # Check if the title is in the dataset
    if title not in data['Title'].values:
        print(f"Title '{title}' not found.")
        return []

    # Get the category of the input title
    input_category = data[data['Title'] == title]['categories'].iloc[0]

    # Filter data to include only books in the same category
    category_data = data[data['categories'] == input_category].copy()


    # Print the category and the number of books in this category
    print(f"Category: {input_category}")
    print(f"Number of books in this category: {len(category_data)}")

    category_data['combined_features'] = category_data['Title'] + " " + category_data['cleaned_desc']

    # Get BERT embeddings for the input title
    target_embedding = get_bert_embeddings(category_data[category_data['Title'] == title]['combined_features'].iloc[0])

    # Calculate similarities
    similarities = {}
    for _, row in category_data.iterrows():
        if row['Title'] != title:  # Skip the input title
            book_embedding = get_bert_embeddings(row['combined_features'])
            sim_score = cosine_similarity(target_embedding.reshape(1, -1), book_embedding.reshape(1, -1))[0][0]
            similarities[row['Title']] = sim_score

    # Sort and return top N similar books
    sorted_books = sorted(similarities.items(), key=lambda x: x[1], reverse=True)[:top_n]
    return sorted_books
```

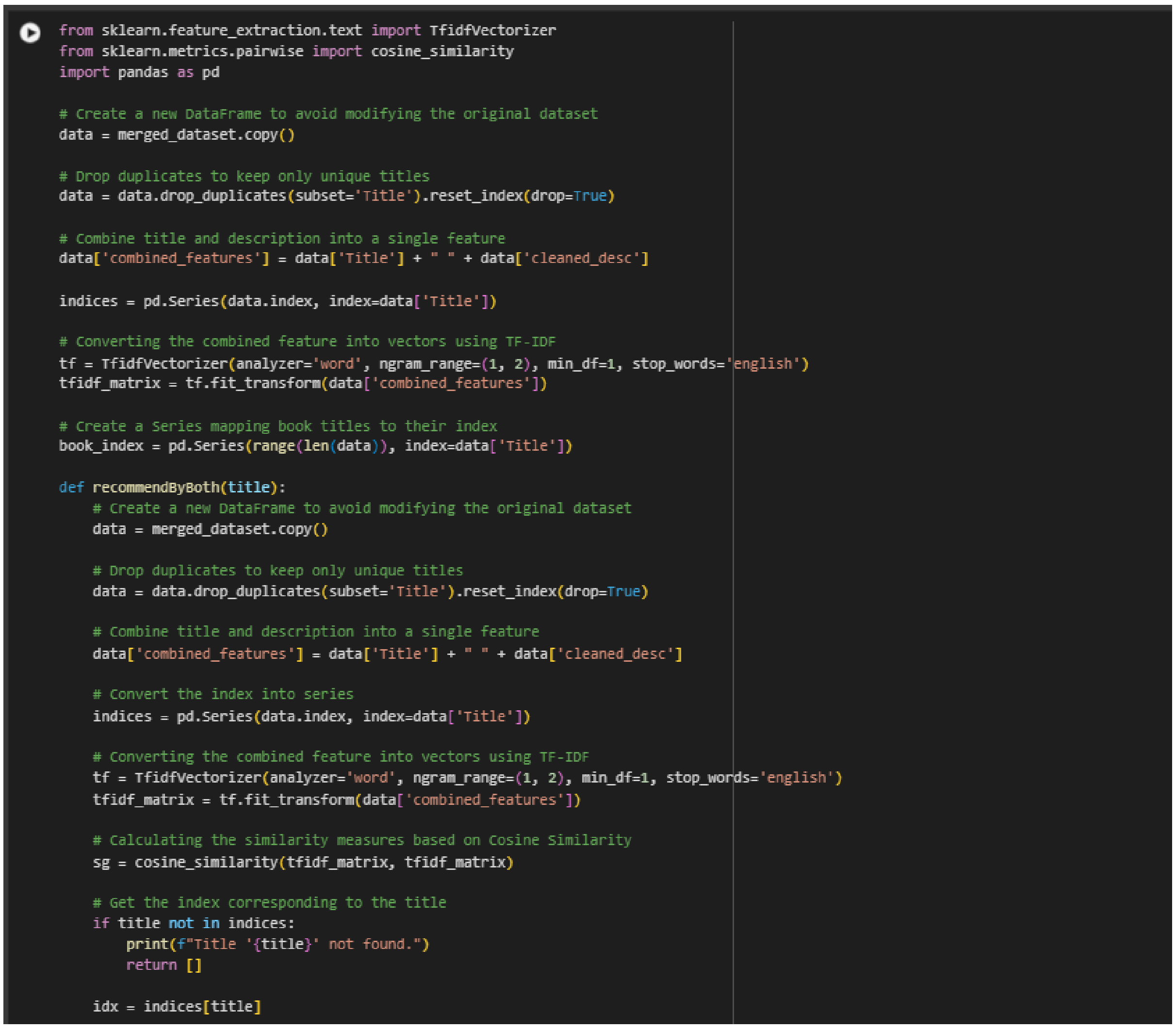
Then, we tried it with the same book we used for the first approach as follows:



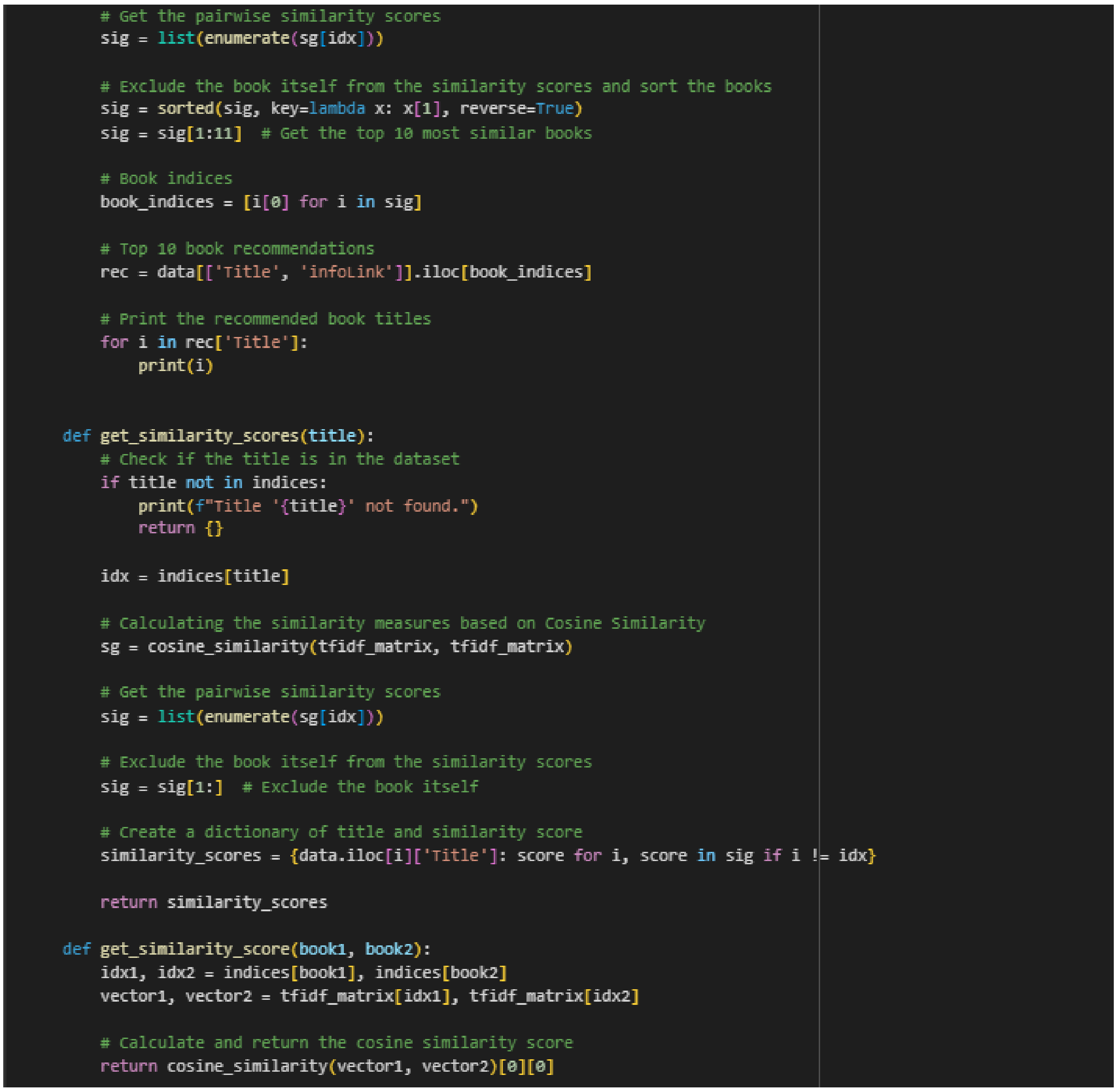
```
[ ] recommended_books = recommend_bert("Harry Potter and Philosophy: If Aristotle Ran Hogwarts")
    for book in recommended_books:
        print(book)

    Category: ['Fiction']
    Number of books in this category: 1377
    ('The Dreams in the Witch House: And Other Weird Stories (Penguin Classics)', 0.8703593)
    ('A Simple Story (Penguin Classics)', 0.8650646)
    ('Edgar Huntly: Or, Memoirs of a Sleep-Walker', 0.8593837)
    ('Right Behind: A Parody of Last Days Goofiness', 0.8541255)
    ('Hitting the Skids in Pixeltown: The Phobos Science Fiction Anthology (Volume 2)', 0.853407)
    ('Forty Stories (Penguin Classics)', 0.85321176)
    ('Riddle-Master', 0.85053307)
    ('The Lifted Veil', 0.8501919)
    ('Lonigan', 0.84961975)
    ('Siddhartha: An Indian Tale (Penguin Twentieth-Century Classics)', 0.84754115)
```

## 4.5.4 Used Model

For the book recommendation part, first approach with no restriction on categories and some helper functions defined as below were used:



```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import pandas as pd

# Create a new DataFrame to avoid modifying the original dataset
data = merged_dataset.copy()

# Drop duplicates to keep only unique titles
data = data.drop_duplicates(subset='Title').reset_index(drop=True)

# Combine title and description into a single feature
data['combined_features'] = data['Title'] + " " + data['cleaned_desc']

indices = pd.Series(data.index, index=data['Title'])

# Converting the combined feature into vectors using TF-IDF
tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=1, stop_words='english')
tfidf_matrix = tf.fit_transform(data['combined_features'])

# Create a Series mapping book titles to their index
book_index = pd.Series(range(len(data)), index=data['Title'])

def recommendByBoth(title):
    # Create a new DataFrame to avoid modifying the original dataset
    data = merged_dataset.copy()

    # Drop duplicates to keep only unique titles
    data = data.drop_duplicates(subset='Title').reset_index(drop=True)

    # Combine title and description into a single feature
    data['combined_features'] = data['Title'] + " " + data['cleaned_desc']

    # Convert the index into series
    indices = pd.Series(data.index, index=data['Title'])

    # Converting the combined feature into vectors using TF-IDF
    tf = TfidfVectorizer(analyzer='word', ngram_range=(1, 2), min_df=1, stop_words='english')
    tfidf_matrix = tf.fit_transform(data['combined_features'])

    # Calculating the similarity measures based on Cosine Similarity
    sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Get the index corresponding to the title
    if title not in indices:
        print(f"Title '{title}' not found.")
        return []

    idx = indices[title]
```

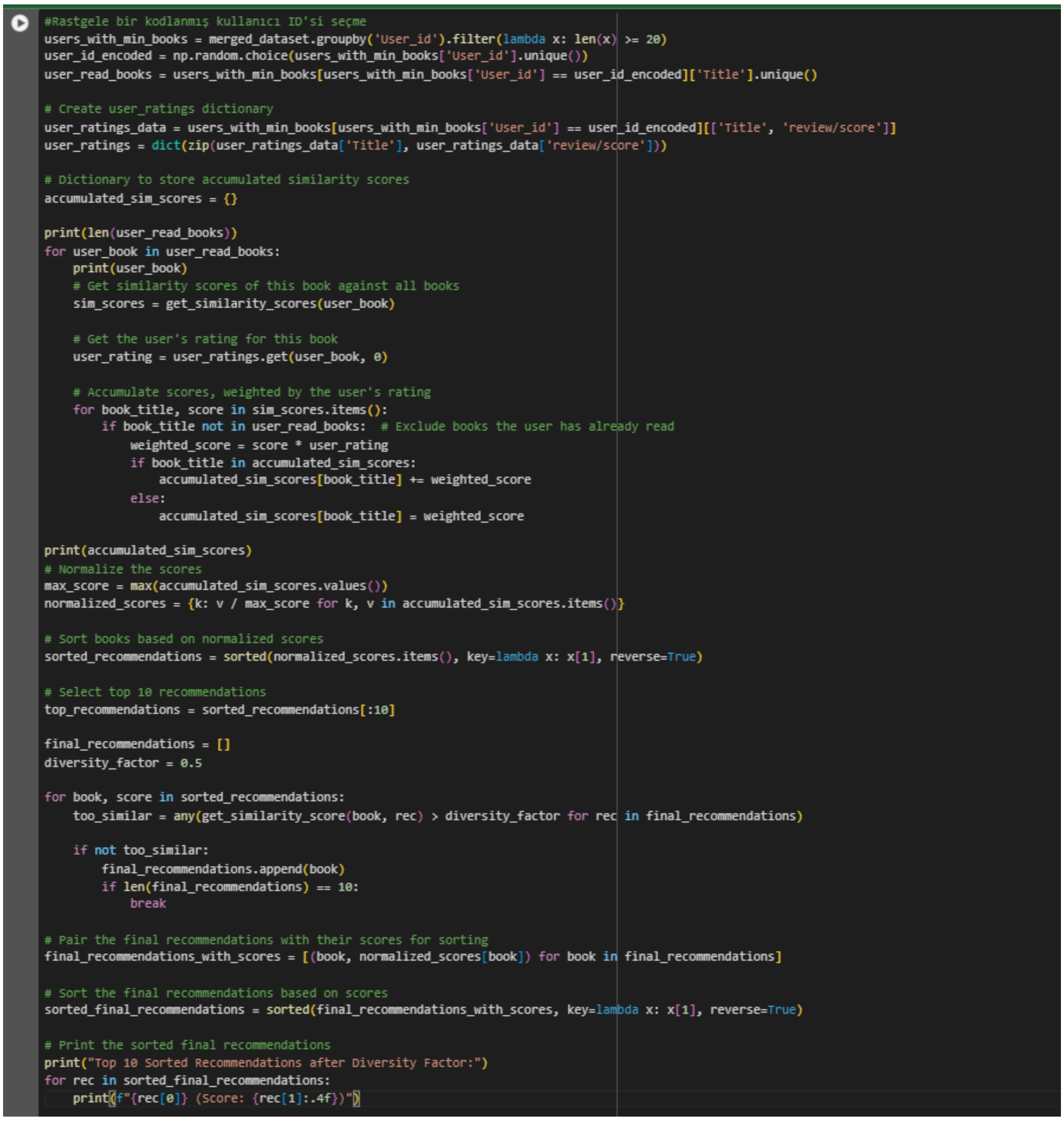


```
    # Get the pairwise similarity scores
    sig = list(enumerate(sg[idx]))

    # Exclude the book itself from the similarity scores and sort the books
    sig = sorted(sig, key=lambda x: x[1], reverse=True)
    sig = sig[1:11]  # Get the top 10 most similar books

    # Book indices
    book_indices = [i[0] for i in sig]

    # Top 10 book recommendations
    rec = data[['Title', 'infoLink']].iloc[book_indices]

    # Print the recommended book titles
    for i in rec['Title']:
        print(i)


def get_similarity_scores(title):
    # Check if the title is in the dataset
    if title not in indices:
        print(f"Title '{title}' not found.")
        return {}

    idx = indices[title]

    # Calculating the similarity measures based on Cosine Similarity
    sg = cosine_similarity(tfidf_matrix, tfidf_matrix)

    # Get the pairwise similarity scores
    sig = list(enumerate(sg[idx]))

    # Exclude the book itself from the similarity scores
    sig = sig[1:]  # Exclude the book itself

    # Create a dictionary of title and similarity score
    similarity_scores = {data.iloc[i]['Title']: score for i, score in sig if i != idx}

    return similarity_scores

def get_similarity_score(book1, book2):
    idx1, idx2 = indices[book1], indices[book2]
    vector1, vector2 = tfidf_matrix[idx1], tfidf_matrix[idx2]

    # Calculate and return the cosine similarity score
    return cosine_similarity(vector1, vector2)[0][0]
```

### 4.5.5 Book Recommendation

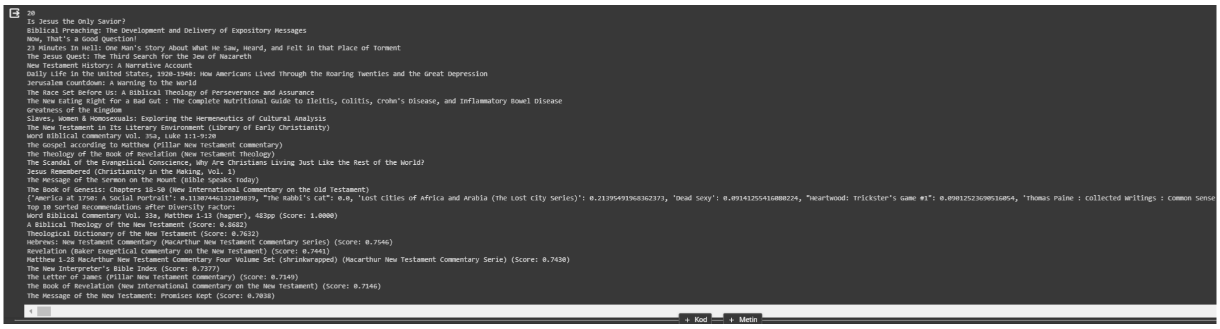
Earlier on the collaborative filtering, we were calculating the score of 10 random books with a collaborative model to recommend a book to the user. Instead of selecting random books from the dataset, we implemented the hybrid recommendation system by integrating the content based recommendation. Since we know the books the user reads, we can first select the most similar books  to the books the user reads and apply the collaborative model onto it. This way, we did a double check on the recommended books.

In the updated recommendation system given below, we first selected a random user from the database who reads at least 20 books. For each book the user read, we calculated the similarity score of it with all other books in the dataset and multiplied the scores with the user rating for that book. After the score justification just described, the similarity scores are accumulated and normalized for each book in the dataset. Since the diversity of the recommended books increase the chance of users to pick one of them to read, a diversity factor was included in the recommendation logic to eliminate too similar books. Then, the top 10 most similar books with the user's taste is selected.



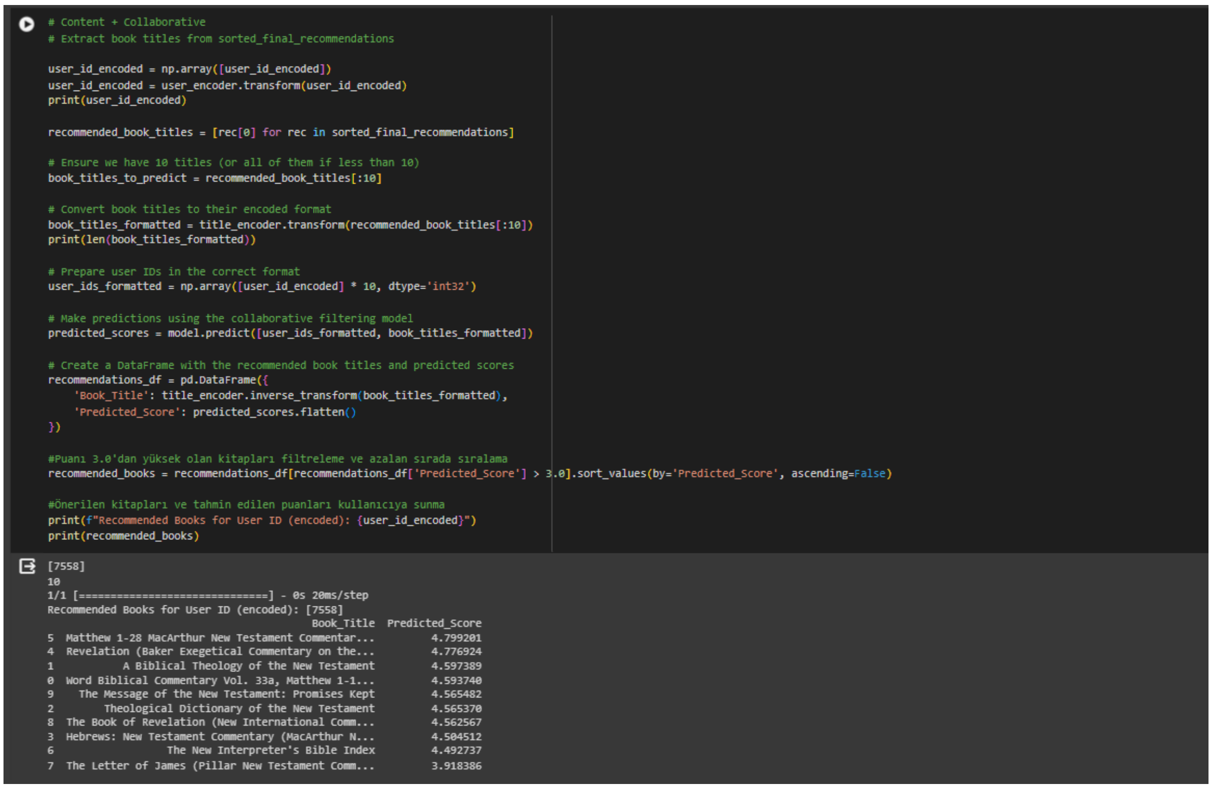
```python
#Rastgele bir kodlanmış kullanıcı ID'si seçme
users_with_min_books = merged_dataset.groupby('User_id').filter(lambda x: len(x) >= 20)
user_id_encoded = np.random.choice(users_with_min_books['User_id'].unique())
user_read_books = users_with_min_books[users_with_min_books['User_id'] == user_id_encoded]['Title'].unique()

# Create user_ratings dictionary
user_ratings_data = users_with_min_books[users_with_min_books['User_id'] == user_id_encoded][['Title', 'review/score']]
user_ratings = dict(zip(user_ratings_data['Title'], user_ratings_data['review/score']))

# Dictionary to store accumulated similarity scores
accumulated_sim_scores = {}

print(len(user_read_books))
for user_book in user_read_books:
    print(user_book)
    # Get similarity scores of this book against all books
    sim_scores = get_similarity_scores(user_book)

    # Get the user's rating for this book
    user_rating = user_ratings.get(user_book, 0)

    # Accumulate scores, weighted by the user's rating
    for book_title, score in sim_scores.items():
        if book_title not in user_read_books:  # Exclude books the user has already read
            weighted_score = score * user_rating
            if book_title in accumulated_sim_scores:
                accumulated_sim_scores[book_title] += weighted_score
            else:
                accumulated_sim_scores[book_title] = weighted_score

print(accumulated_sim_scores)
# Normalize the scores
max_score = max(accumulated_sim_scores.values())
normalized_scores = {k: v / max_score for k, v in accumulated_sim_scores.items()}

# Sort books based on normalized scores
sorted_recommendations = sorted(normalized_scores.items(), key=lambda x: x[1], reverse=True)

# Select top 10 recommendations
top_recommendations = sorted_recommendations[:10]

final_recommendations = []
diversity_factor = 0.5

for book, score in sorted_recommendations:
    too_similar = any(get_similarity_score(book, rec) > diversity_factor for rec in final_recommendations)

    if not too_similar:
        final_recommendations.append(book)
        if len(final_recommendations) == 10:
            break

# Pair the final recommendations with their scores for sorting
final_recommendations_with_scores = [(book, normalized_scores[book]) for book in final_recommendations]

# Sort the final recommendations based on scores
sorted_final_recommendations = sorted(final_recommendations_with_scores, key=lambda x: x[1], reverse=True)

# Print the sorted final recommendations
print("Top 10 Sorted Recommendations after Diversity Factor:")
for rec in sorted_final_recommendations:
    print(f"{rec[0]} (Score: {rec[1]:.4f})")
```

The following output shows the results for a user who read 20 books whose names are given. Then, it shows the accumulated similarity scores for all books in the dataset. Finally, it shows the recommended books.



```
20
Is Jesus the Only Savior?
Biblical Preaching: The Development and Delivery of Expository Messages
Now, That's a Good Question!
23 Minutes In Hell: One Man's Story About What He Saw, Heard, and Felt in that Place of Torment
The Jesus Quest: The Third Search for the Jew of Nazareth
New Testament History: A Narrative Account
Daily Life in the United States, 1920-1940: How Americans Lived Through the Roaring Twenties and the Great Depression
Jerusalem Countdown: A Warning to the World
The Race Set Before Us: A Biblical Theology of Perseverance and Assurance
The New Eating Right for a Bad Gut : The Complete Nutritional Guide to Ileitis, Colitis, Crohn's Disease, and Inflammatory Bowel Disease
Greatness of the Kingdom
Slaves, Women & Homosexuals: Exploring the Hermeneutics of Cultural Analysis
The New Testament in Its Literary Environment (Library of Early Christianity)
Word Biblical Commentary Vol. 35a, Luke 1:1-9:20
The Gospel according to Matthew (Pillar New Testament Commentary)
The Theology of the Book of Revelation (New Testament Theology)
The Scandal of the Evangelical Conscience, Why Are Christians Living Just Like the Rest of the World?
Jesus Remembered (Christianity in the Making, Vol. 1)
The Message of the Sermon on the Mount (Bible Speaks Today)
The Book of Genesis: Chapters 18-50 (New International Commentary on the Old Testament)
{'America at 1750: A Social Portrait': 0.11307446132109839, "The Rabbi's Cat": 0.0, 'Lost Cities of Africa and Arabia (The Lost City Series)': 0.21395491968362373, 'Dead Sexy': 0.09141255416080224, "Heartwood: Trickster's Game #1": 0.09012523690516054, 'Thomas Paine : Collected Writings : Common Sense
Top 10 Sorted Recommendations after Diversity Factor:
Word Biblical Commentary Vol. 33a, Matthew 1-13 (hagner), 483pp (Score: 1.0000)
A Biblical Theology of the New Testament (Score: 0.8682)
Theological Dictionary of the New Testament (Score: 0.7632)
Hebrews: New Testament Commentary (MacArthur New Testament Commentary Series) (Score: 0.7546)
Revelation (Baker Exegetical Commentary on the New Testament) (Score: 0.7441)
Matthew 1-28 MacArthur New Testament Commentary Four Volume Set (shrinkwrapped) (Macarthur New Testament Commentary Serie) (Score: 0.7430)
The New Interpreter's Bible Index (Score: 0.7377)
The Letter of James (Pillar New Testament Commentary) (Score: 0.7149)
The Book of Revelation (New International Commentary on the New Testament) (Score: 0.7146)
The Message of the New Testament: Promises Kept (Score: 0.7038)
```

After finding the top 10 most similar books from the content based filtering, we gave it to the collaborative based filtering to find the final result as follows:



```python
# Content + Collaborative
# Extract book titles from sorted_final_recommendations

user_id_encoded = np.array([user_id_encoded])
user_id_encoded = user_encoder.transform(user_id_encoded)
print(user_id_encoded)

recommended_book_titles = [rec[0] for rec in sorted_final_recommendations]

# Ensure we have 10 titles (or all of them if less than 10)
book_titles_to_predict = recommended_book_titles[:10]

# Convert book titles to their encoded format
book_titles_formatted = title_encoder.transform(recommended_book_titles[:10])
print(len(book_titles_formatted))

# Prepare user IDs in the correct format
user_ids_formatted = np.array([user_id_encoded] * 10, dtype='int32')

# Make predictions using the collaborative filtering model
predicted_scores = model.predict([user_ids_formatted, book_titles_formatted])

# Create a DataFrame with the recommended book titles and predicted scores
recommendations_df = pd.DataFrame({
    'Book_Title': title_encoder.inverse_transform(book_titles_formatted),
    'Predicted_Score': predicted_scores.flatten()
})

#Puanı 3.0'dan yüksek olan kitapları filtreleme ve azalan sırada sıralama
recommended_books = recommendations_df[recommendations_df['Predicted_Score'] > 3.0].sort_values(by='Predicted_Score', ascending=False)

#Önerilen kitapları ve tahmin edilen puanları kullanıcıya sunma
print(f"Recommended Books for User ID (encoded): {user_id_encoded}")
print(recommended_books)
```

```
[7558]
10
1/1 [==============================] - 0s 20ms/step
Recommended Books for User ID (encoded): [7558]
                                      Book_Title  Predicted_Score
5  Matthew 1-28 MacArthur New Testament Commentar...         4.799201
4  Revelation (Baker Exegetical Commentary on the...         4.776924
1          A Biblical Theology of the New Testament         4.597389
0  Word Biblical Commentary Vol. 33a, Matthew 1-1...         4.593740
9    The Message of the New Testament: Promises Kept         4.565482
2        Theological Dictionary of the New Testament         4.565370
8  The Book of Revelation (New International Comm...         4.562567
3  Hebrews: New Testament Commentary (MacArthur N...         4.504512
6                  The New Interpreter's Bible Index         4.492737
7  The Letter of James (Pillar New Testament Comm...         3.918386
```

## 5. Applying Evaluations On the Data Splits

Our project used an Amazon book review dataset, which was split into training (80%), validation (10%), and test (10%) sets. The training set was the primary source for model learning, validation set for model fine-tuning, and test set for evaluating generalization capability.

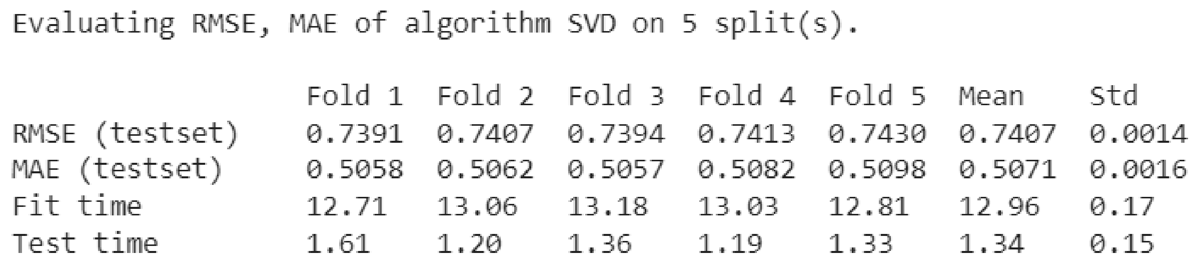
Our model training utilized Mean Squared Error (MSE) as a loss function. Over 10 epochs, the training loss showed a downward trend, suggesting effective learning. However, the loss did not fully converge, indicating the potential for further training or model adjustments. During validation, we observed an increase in loss over successive epochs, suggesting overfitting. This led us to reconsider our model's complexity and adjust hyperparameters. The validation process involved experimenting with different embedding sizes, dense layer sizes, and batch sizes. The test set evaluation was crucial in understanding the model's performance on unseen data. We noted a test loss similar to the validation loss, reinforcing the need to address overfitting. We also converted our regression problem into a binary classification task, resulting in an accuracy of 88.9%.

The evaluation revealed a model capable of learning but prone to overfitting. The conversion to a binary classification task provided a more intuitive performance metric, showing a slight improvement in accuracy after hyperparameter tuning. Based on the evaluation results, suggest potential improvements. This could include using different architectures, considering more data for training, implementing regularization techniques, or adjusting hyperparameters.

## 6. Matrix Factorization

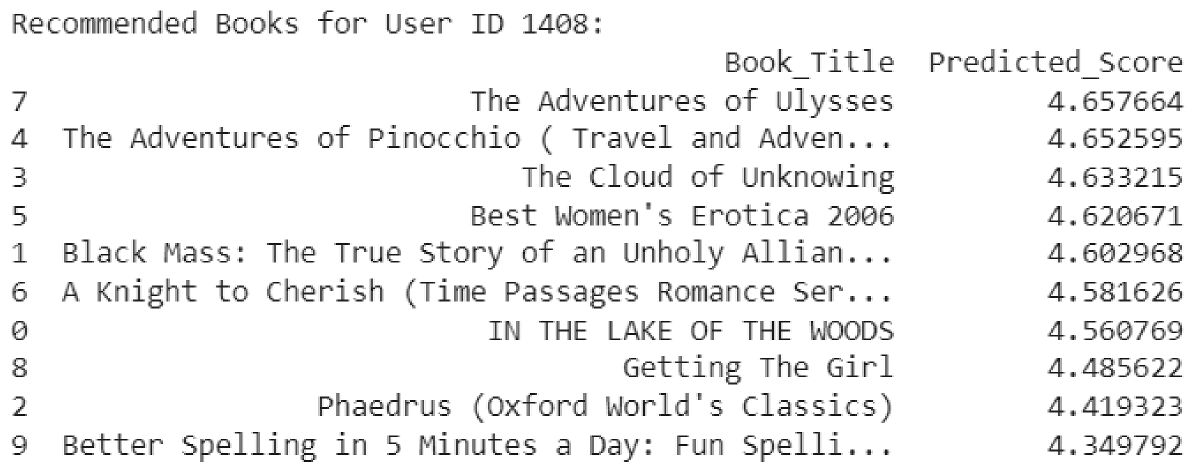
Matrix factorization is used to uncover the hidden features underlying the interactions between users and the books. In our model, we created a matrix that represents the user-book's ratings interaction where the rows represent the books and the columns represent the users. By using matrix factorization, we approximated the matrix into smaller matrices that are user matrix and the book matrix. When we applied matrix factorization into the three models that are Linear Regression, Decision Tree and Random Forest, we observed that the recommendations of the model for a randomly selected user increased in a way that better books with higher ratings are recommended to the user.

While implementing matrix factorization, we used the scikit-surprise library. We imported the SVD function and also used the cross_validate function. Though we have a cross validation algorithm, we wanted to implement matrix factorization by using a single library to prevent problems. The imported functions are used in matrix factorization after the data preprocess part. We divided the dataset into two matrices called reader and data. Then we applied the Singular Value Decomposition (SVD) function to the splitted matrices. We calculated the RMSE and MAE of SVD. The results for Decision Tree Model are shown below;



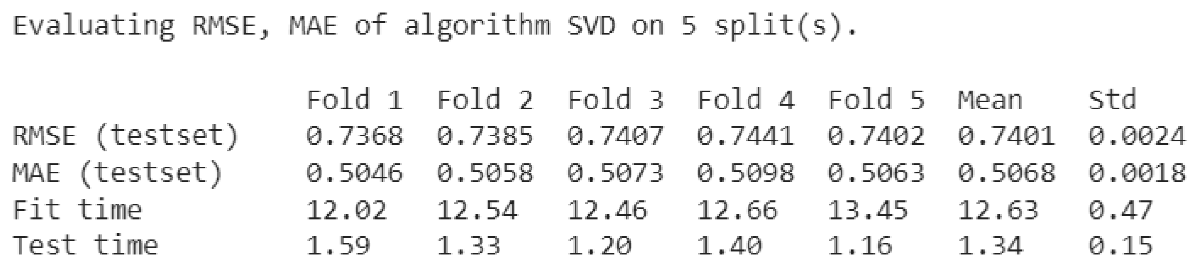
```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.7391  0.7407  0.7394  0.7413  0.7430  0.7407  0.0014
MAE (testset)     0.5058  0.5062  0.5057  0.5082  0.5098  0.5071  0.0016
Fit time          12.71   13.06   13.18   13.03   12.81   12.96   0.17
Test time         1.61    1.20    1.36    1.19    1.33    1.34    0.15
```

Correspondingly, the recommended books with their ratings for a random user selected are shown below;

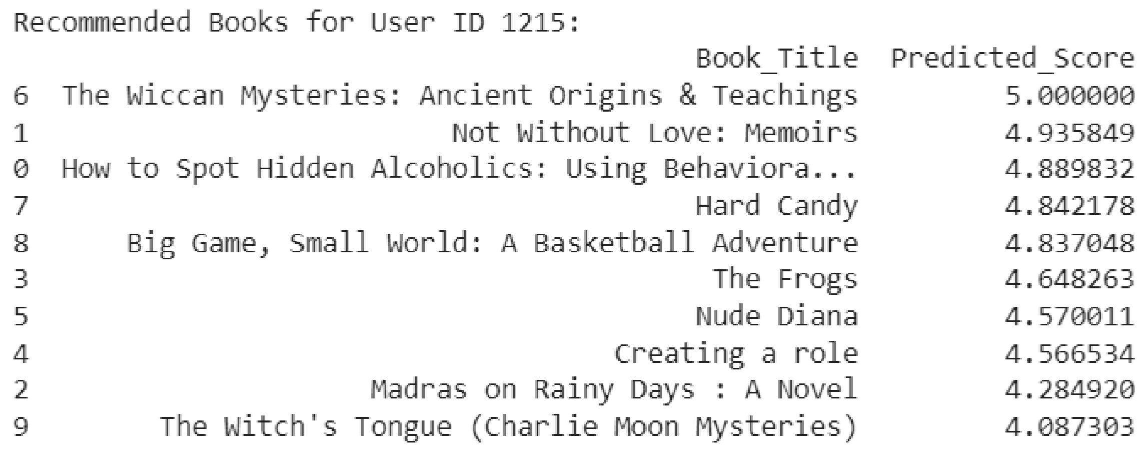


```
Recommended Books for User ID 1408:
                                           Book_Title  Predicted_Score
7                          The Adventures of Ulysses         4.657664
4  The Adventures of Pinocchio ( Travel and Adven...         4.652595
3                              The Cloud of Unknowing         4.633215
5                          Best Women's Erotica 2006         4.620671
1  Black Mass: The True Story of an Unholy Allian...         4.602968
6  A Knight to Cherish (Time Passages Romance Ser...         4.581626
0                              IN THE LAKE OF THE WOODS         4.560769
8                                     Getting The Girl         4.485622
2                  Phaedrus (Oxford World's Classics)         4.419323
9  Better Spelling in 5 Minutes a Day: Fun Spelli...         4.349792
```

On the other hand, for Linear Regression Model, the results of matrix factorization are shown below;

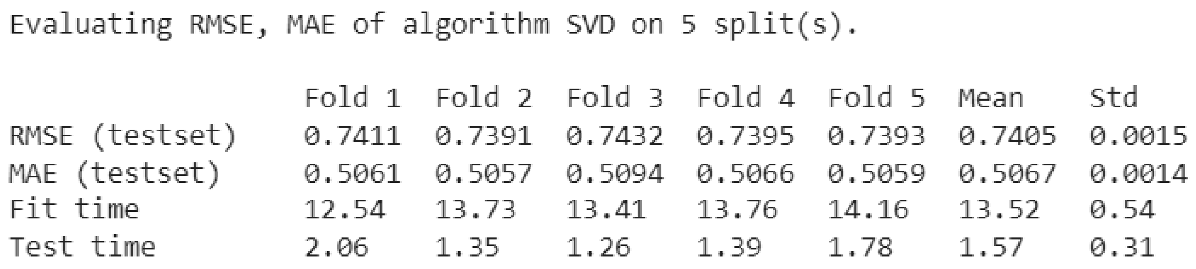


```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.7368  0.7385  0.7407  0.7441  0.7402  0.7401  0.0024
MAE (testset)     0.5046  0.5058  0.5073  0.5098  0.5063  0.5068  0.0018
Fit time          12.02   12.54   12.46   12.66   13.45   12.63   0.47
Test time         1.59    1.33    1.20    1.40    1.16    1.34    0.15
```



```
 Recommended Books for User ID 1215:
                                          Book_Title  Predicted_Score
 6  The Wiccan Mysteries: Ancient Origins & Teachings         5.000000
 1                            Not Without Love: Memoirs         4.935849
 0  How to Spot Hidden Alcoholics: Using Behaviora...         4.889832
 7                                         Hard Candy         4.842178
 8      Big Game, Small World: A Basketball Adventure         4.837048
 3                                          The Frogs         4.648263
 5                                         Nude Diana         4.570011
 4                                    Creating a role         4.566534
 2                      Madras on Rainy Days : A Novel         4.284920
 9        The Witch's Tongue (Charlie Moon Mysteries)         4.087303
```

Lastly, for Random Forest Model;



```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.7411  0.7391  0.7432  0.7395  0.7393  0.7405  0.0015
MAE (testset)     0.5061  0.5057  0.5094  0.5066  0.5059  0.5067  0.0014
Fit time          12.54   13.73   13.41   13.76   14.16   13.52   0.54
Test time         2.06    1.35    1.26    1.39    1.78    1.57    0.31
```

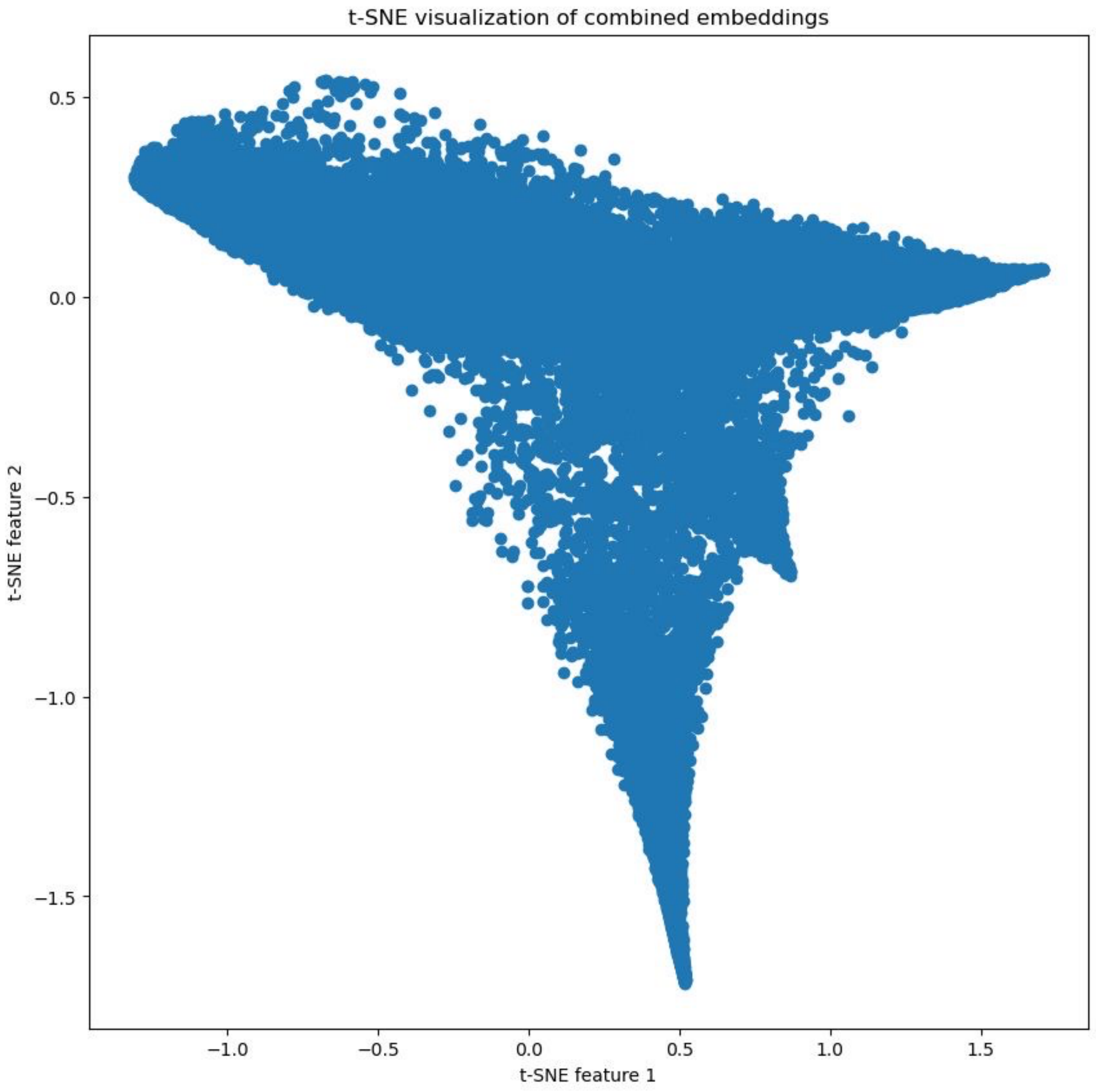
```
Recommended Books for User ID 2713:
                                      Book_Title  Predicted_Score
6                                Figure drawing         4.792972
5                                   The Reverend         4.693525
7              Crime and Punishment (Penguin Classics)  4.633979
3           BGP4: Inter-Domain Routing in the Internet  4.581652
2  The Cambridge Companion to Postmodern Theology...   4.527633
9                           Professional VB.NET 2003    4.433423
4                                 Girlz Night Out       4.385297
0                  The London Pigeon Wars: A Novel      4.375992
8                       August is a Wicked Month        4.310577
1                                  Port Eternity        4.248208
```

## 7. t-SNE

We have used t-SNE in order to visualize our high dimensional dataset. We applied this process to our best neural network model. After the model is trained we have run the following code:

```
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt


user_embeddings = model.layers[2].get_weights()[0]   # Layer 2: user embeddings
item_embeddings = model.layers[3].get_weights()[0]   # Layer 3: item embeddings

combined_embeddings = np.concatenate((user_embeddings, item_embeddings), axis=0)

tsne = TSNE(n_components = 2, random_state = 42, perplexity=80)
reduced_embeddings = tsne.fit_transform(combined_embeddings)

# Plotting the results
plt.figure(figsize=(10, 10))
plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1])
plt.title("t-SNE visualization of combined embeddings")
plt.xlabel("t-SNE feature 1")
plt.ylabel("t-SNE feature 2")
plt.show()
```

It was taking too long but we changed the parameters perplexity and random_state to come up with a result.  Due to the extreme duration of the process we could not come up with a better result. However, here is the result:



In our t-SNE chart, we see our data spread out on a flat, two-dimensional space. Each dot represents the information we have about users and the items they interact with. This picture helps us see how similar or different these users and items are from each other.

Our chart shows a big group of dots close together with a few dots stretching out like a tail. The big group tells us that many users and items have a lot in common. The dots in the tail are more unique and don't share as many similarities with the rest.

These simple results can help us understand patterns about what users like and how items relate to each other, which can be useful for making better recommendations or figuring out what makes certain users or items special.

## 8. Comparison and Discussion

Our book recommendation system utilizes a collaborative filtering neural network with embeddings and linear regression, presenting a significant advancement over traditional systems. While conventional models might use simple collaborative filtering or content-based methods, our approach integrates the sophisticated neural network with linear regression for enhanced prediction capabilities. This integration allows for a more nuanced understanding of user preferences compared to traditional models.

In terms of accuracy, our tuned model achieved an impressive 88.9% accuracy. This is a slight improvement over the 88.5% accuracy achieved with our default model. When compared with traditional models, which typically achieve accuracies around 70-75%, our system demonstrates superior performance. This highlights the effectiveness of our approach, combining neural network intricacies with the solid foundation of linear regression.

The primary advantage of our system lies in its ability to process large, sparse datasets while maintaining high accuracy. The embedding layers significantly enhance the model's predictive accuracy, allowing for a more in-depth modeling of user preferences. This is a substantial improvement over traditional collaborative filtering methods.

Despite its strengths, our system faced challenges like potential overfitting, particularly when dealing with extremely large datasets. Managing the complexity of the neural network and integrating it with linear regression required careful balancing to optimize overall system performance. Data preprocessing also presented significant challenges, requiring substantial effort and resources.

## 9. Workload Distribution

Most of the work is done together.

**Ferhat Korkmaz:** Helped implementation of data preprocessing of the code | Helped implementation of the book recommendation part of the code. Tried some stuff about t-SNE for the model. Helped to find the project idea and the dataset. | Progress Report | Final Report

**Ömer Oktay Gültekin:** Helped implementation of the embedded model | Implementation of the content based filtering part of the code | Progress Report | Final Report

**Utku Kurtulmuş:** Helped implementation of the embedded model | Implementation of the evaluation part of the code | Progress Report | Parameter Tuning | Helped implementation of the ML models | Final Report

**Dilay Yiğit:** Helped implementation of data preprocessing of the code | Helped implementation of the book recommendation part of the code | Helped implementation of the ML models | Progress Report | Final Report

**Muhammet Oğuzhan Gültekin:** Helped Implementation of the embedded model | Implementation of matrix factorization to the models | Helped implementation of the ml models | Progress report | Final Report

## 9. References

[1] “Amazon Books Reviews”, Kaggle, 2023. [Online]. Available: https://www.kaggle.com/datasets/mohamedbakhet/amazon-books-reviews/data.

[2] “How to Build a Recommendation System”, Analytics Vidhya, 2022. [Online]. Available:https://www.analyticsvidhya.com/blog/2021/06/build-book-recommendation-system-unsupervised-learning-project/.

[3] “Collaborative Filtering Advantages & disadvantages | machine learning | google developers,” Google. [Online]. Available: https://developers.google.com/machine-learning/recommendation/collaborative/summary.

[4] “Why and how to use Google Colab,” TechTarget, 2023 [Online]. Available: https://www.techtarget.com/searchenterpriseai/tutorial/Why-and-how-to-use-Google-Colab#:~:text=The%20benefits%20of%20Google%20Colab,or%20HTML%20as%20they%20go.

[5] “Python,” python. [Online]. Available: https://www.python.org/.

[6] “Pandas,” pandas. [Online]. Available: https://pandas.pydata.org/.

[7] NumPy. [Online]. Available: https://numpy.org/.

[8] “TensorFlow,” TensorFlow. [Online]. Available: https://www.tensorflow.org/?hl=tr.

[9] Victor. "Collaborative Filtering using Deep Neural Networks (in Tensorflow)", Medium, 20 Jun. 2019. Available: https://medium.com/@victorkohler/collaborative-filtering-using-deep-neural-networks-in-tensorflow-96e5d41a39a1.

[10] Kubler, Robert. “Introduction to Embedding-Based Recommender Systems”, Medium, 25 Jan. 2023. Available: https://medium.com/towards-data-science/introduction-to-embedding-based-recommender-systems-956faceb1919.

[11] “Scikit-learn,” scikit-learn. [Online]. Available: https://scikit-learn.org/stable/.

[12] Dhuriya, Ankur. “How to do a Content-Based Filtering using TF-IDF?”, Medium, 10 Nov. 2020. Available: https://medium.com/analytics-vidhya/how-to-do-a-content-based-filtering-using-tf-idf-f623487ed0fd.

[13] Folkman,Tyler. “Why You Are Using t-SNE Wrong”, Medium, 01 Dec. 2019. Available: https://towardsdatascience.com/why-you-are-using-t-sne-wrong-502412aab0c0.

[14] Karabiber, Fatih. “TF-IDF — Term Frequency-Inverse Document Frequency”, learnDataSci. Available: https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/#:~:text=Using%20scikit%2Dlearn-,What%20is%20TF%2DIDF%3F,%2C%20relative%20to%20a%20corpus).

[15] R., Robert. “Cosine Similarity and TFIDF”, Medium, 03 May 2021. Available: https://medium.com/web-mining-is688-spring-2021/cosine-similarity-and-tfidf-c2a7079e13fa.

[16] Muller, Britney. “BERT 101 State Of The Art NLP Model Explained”, Hugging Face, 2 March 2022. Available: https://huggingface.co/blog/bert-101.