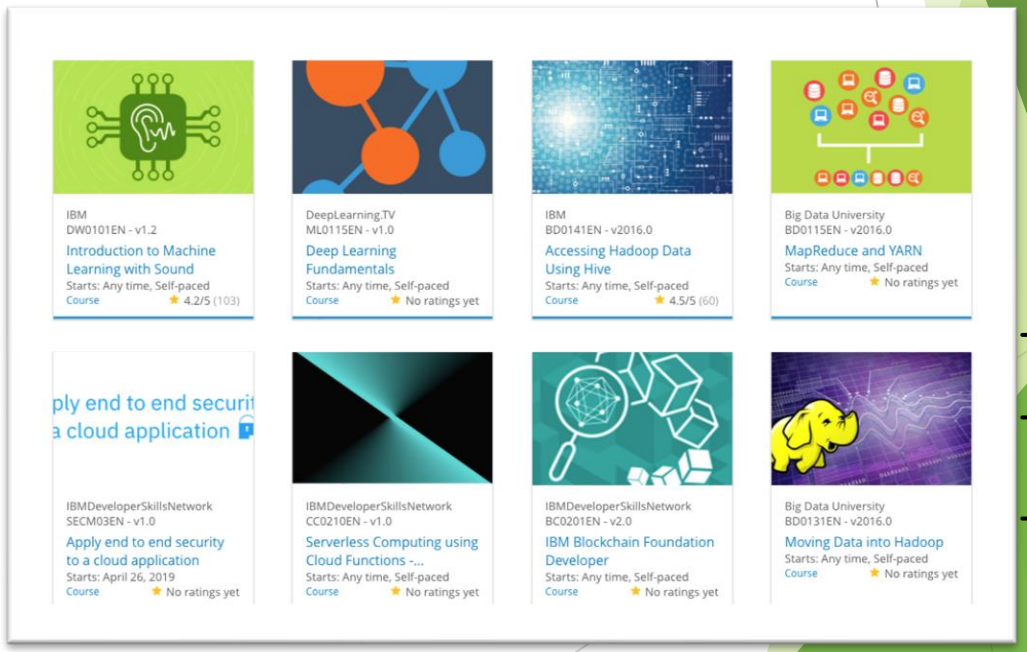
Build a Personalized Online Course Recommender System with Machine Learning

By : Sadhana Tukaram Jarag
Date :19th May 2023



OUTLINE

- Introduction and Background
- Exploratory Data Analysis
- Content-based Recommender System using Unsupervised Learning
- Collaborative-filtering based Recommender System using Supervised learning
- Conclusion
- Appendix

INTRODUCTION

- Project background and context
  - In this project, we have worked with online course enrollment data to explore the idea of a recommendation system. The dataset includes information about course titles, genres, and enrollments. The goal is to analyze the data, uncover insights, and develop a recommendation system to suggest relevant courses to users.
- Problem states and hypotheses
  - The problem statement revolves around understanding the dataset and making assumptions based on summary statistics and visualizations. We aim to investigate the relationships between course titles, genres, and enrollments to identify any patterns or trends. Additionally, we will calculate the cosine similarity between vectors to measure the similarity between different courses.
- Hypotheses:
  - Courses with similar genres are more likely to have higher enrollments.
  - Popular course titles tend to belong to specific genres.
  - Courses with higher enrollments have a higher likelihood of being recommended.

INTRODUCTION

- Assumptions:
  - The course titles and genres provided in the dataset accurately represent the content and category of the courses.
  - Enrollments reflect the popularity or demand for a particular course.
  - Similarity based on cosine similarity provides a reliable measure of course similarity.

- Reframing:
  - To reframe the problem, we will focus on the following key steps:
  - Exploratory data analysis: Analyze the dataset using summary statistics, visualizations, and graphical representations to gain insights into course titles, genres, and enrollments.
  - Cosine similarity calculation: Calculate the cosine similarity between two vectors (e.g., course descriptions, course genres) to determine the similarity between different courses.
  - Recommendation system: Develop a recommendation system based on course similarities to suggest relevant courses to users.
  - Evaluation: Evaluate the performance of the recommendation system using appropriate metrics and assess its effectiveness in providing valuable course recommendations.
- By following these steps, we can gain a deeper understanding of the dataset, explore relationships between variables, and build a recommendation system to enhance the user experience in online course enrollment.

EXPLORATORY DATA ANALYSYS

COURSE COUNT PER GERNE

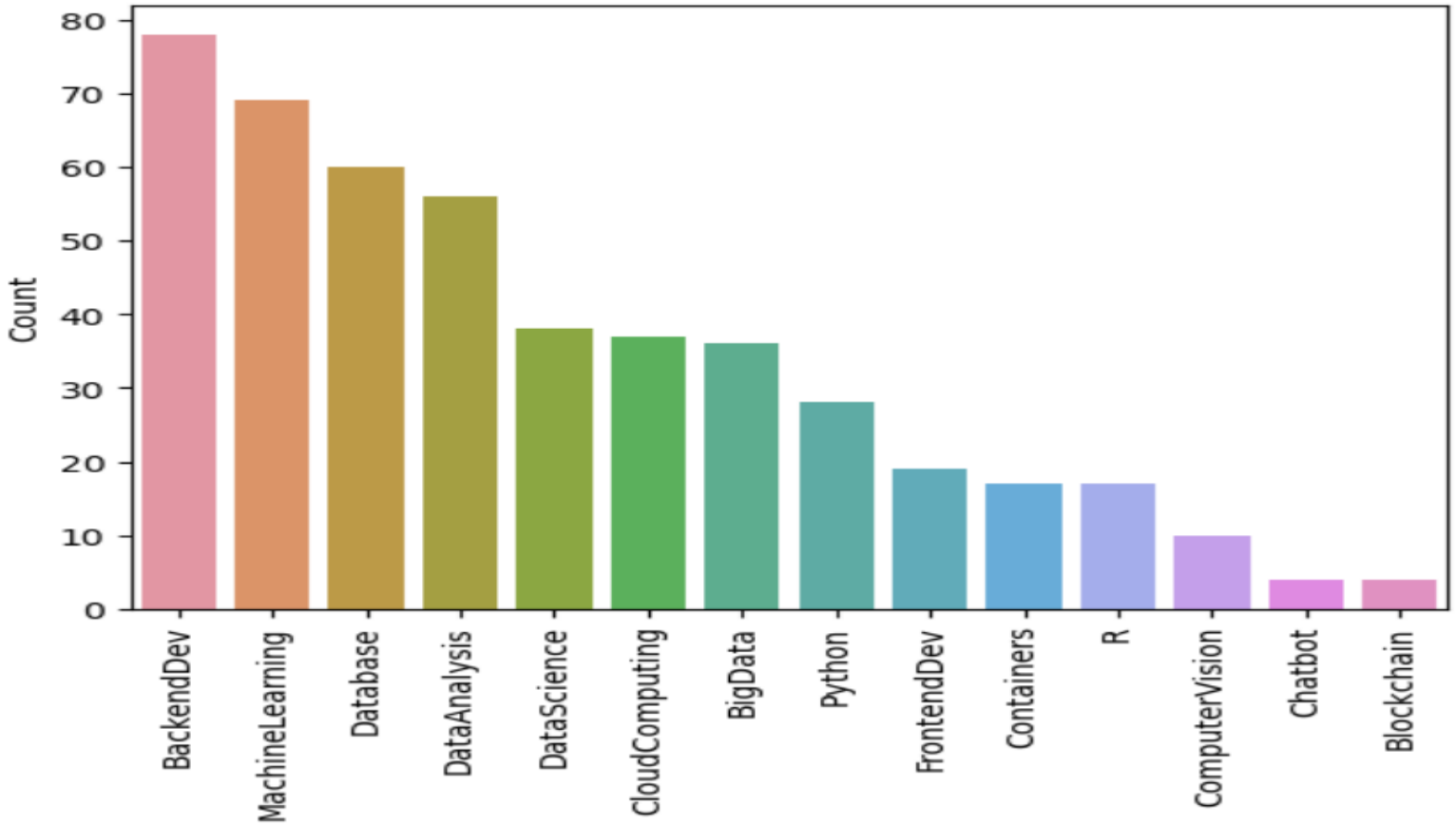
Objectives: To generate a bar graph showing the top 10 courses with the highest enrollment counts. This will provide insights into the popularity and demand for different courses.

Steps to Achieve the Objective:

- **Import Required Libraries:** Import the necessary libraries such as pandas, matplotlib, or any other libraries needed for data manipulation and visualization.
- **Load the Dataset:** Load the course enrollment dataset that contains information about course titles, enrollment counts, and other relevant data.
- **Clean the Data:** Perform data cleaning and preprocessing steps to handle missing values, remove duplicates, and ensure data quality.
- **Apply Queries:** Use appropriate queries or group the dataset based on the course titles and calculate the enrollment counts for each course. Sort the courses in descending order based on the enrollment counts.
- **Select Top 10 Courses:** Select the top 10 courses with the highest enrollment counts.
- **Visualize the Result:** Create a bar graph using a suitable visualization library (e.g., matplotlib) to display the top 10 courses and their respective enrollment counts. The x-axis of the bar graph should represent the course titles, while the y-axis represents the enrollment counts. Each bar represents a course, and its height corresponds to the enrollment count.

By following these steps, you can generate a bar graph that displays the top 10 courses with the highest enrollment counts, providing insights into the popularity of different courses among users.

COURSE COUNT PER GAME



The x-axis is the course genre, and the y-axis is the course count per genre.

COURSE ENROLLEMENT DISTRIBUTION

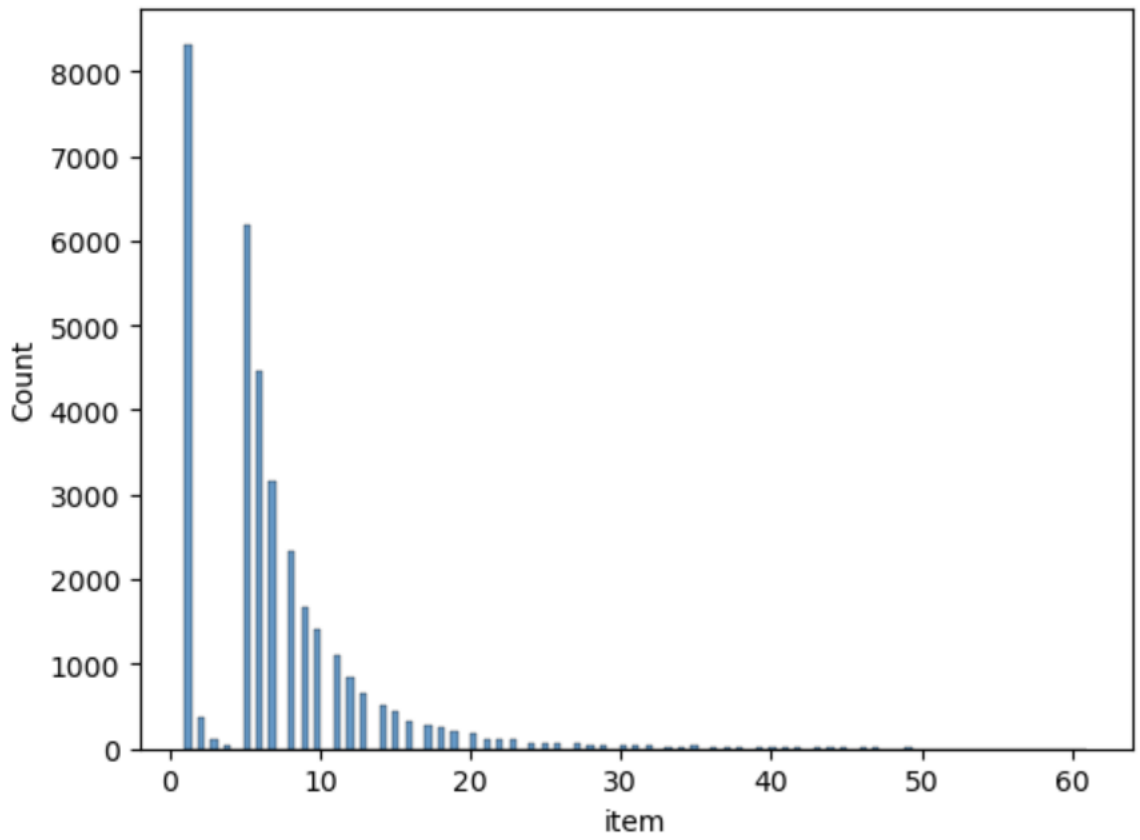
Objective: Enrollment Distributions

Steps to Achieve the Objective:

- **Load the Dataset:** Load the dataset containing enrollment data, which includes information about users and their course enrollments.
- **Apply GroupBy and Count:** Use Pandas' groupby() method on the "user" column to group the data by user and apply the count() method to calculate the enrollment count for each user.
- **Aggregate the Enrollment Counts:** After applying the groupby() and count() operations, you will have a new DataFrame that shows the enrollment count for each user.
- **Sort the Result:** Sort the DataFrame in descending order based on the enrollment count. This will allow you to obtain the enrollment distribution in a histogram-like format.
- **Report the Total Number of Users:** Retrieve the total number of unique users from the grouped and counted DataFrame. This will give you the total count of users after the aggregation.

By following these steps, you will be able to obtain the enrollment distributions and report the total number of users based on the enrollment count for each user

COURSE ENROLLEMENT DISTRIBUTION



Enrollment distributions that means how many users rated just 1 item or how many rated 10 items, etc

20 MOST POPULAR COURSES

- List the most popular 20 courses here means items with the most rating counts
- Follow below steps
  - Group the data by the course/item and count the number of ratings for each item.
  - Sort the courses in descending order based on the rating counts
  - Select the top 20 courses with the highest rating counts.
  - Print the list of the most popular 20 courses.

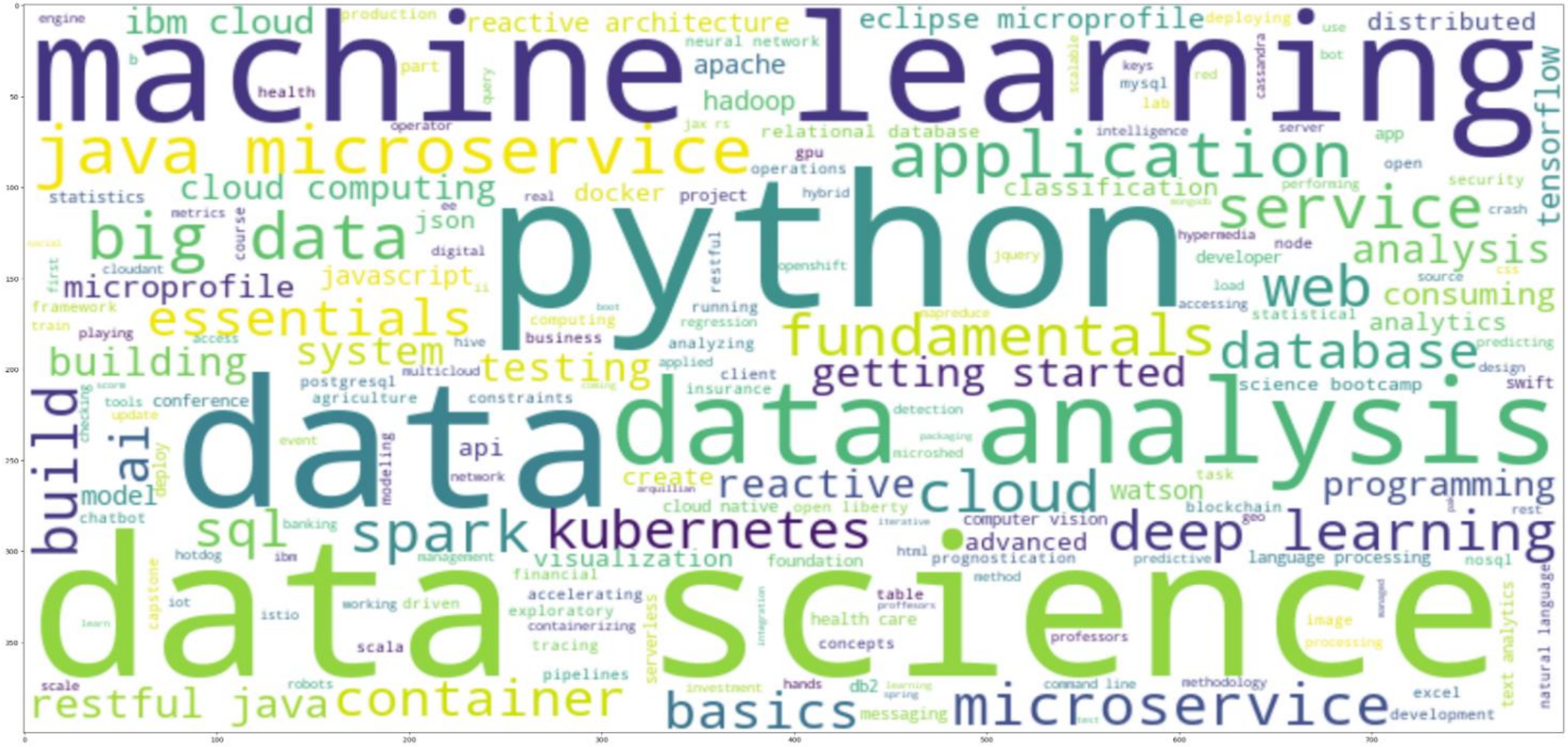
```
5]: # WRITE YOUR CODE HERE
pop_item = ratings_df.groupby(['item']).size().reset_index()
pop_item.columns=['course', 'Ratings']
sorted_item= pop_item.sort_values(by='Ratings', ascending=False).reset_index(drop=True)
top_20_sorted = sorted_item[:20]
print(top_20_sorted)
```

```
]: # WRITE YOUR CODE HERE
mergered_data = pd.merge(sorted_item[['course', 'Ratings']],course_df[['COURSE_ID', 'TITLE']],how='left',right_on='COURSE_ID',lef
mergered_data[['TITLE','Ratings']][:20]
```

OUTPUT

	TITLE	Ratings
0	python for data science	14936
1	introduction to data science	14477
2	big data 101	13291
3	hadoop 101	10599
4	data analysis with python	8303
5	data science methodology	7719
6	machine learning with python	7644
7	spark fundamentals i	7551
8	data science hands on with open source tools	7199
9	blockchain essentials	6719
10	data visualization with python	6709
11	deep learning 101	6323
12	build your own chatbot	5512
13	r for data science	5237
14	statistics 101	5015
15	introduction to cloud	4983
16	docker essentials a developer introduction	4480
17	sql and relational databases 101	3697
18	mapreduce and yarn	3670
19	data privacy fundamentals	3624

WORD CLOUD COURSE TITLE



WORD CLOUD COURSE TITLE

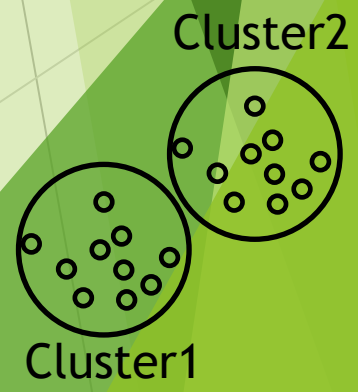
Objective: Word Cloud Visualization of Course Title Frequencies

Steps to Achieve the Objective:

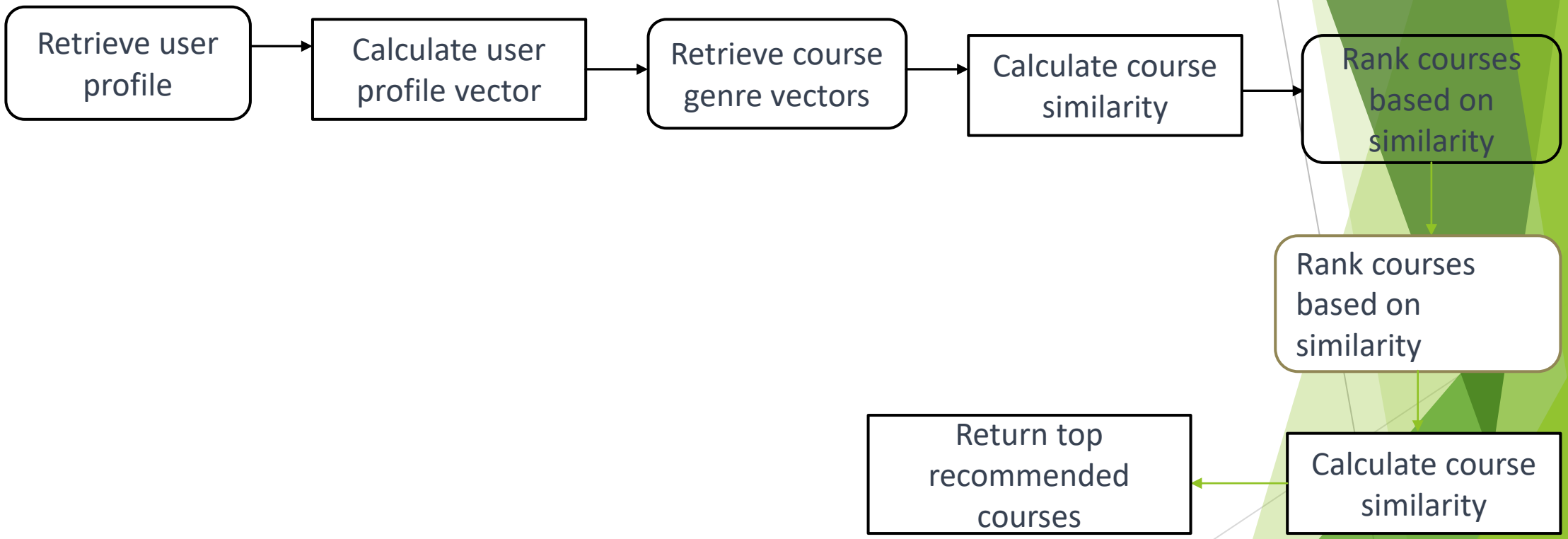
- **Load the Dataset:** Load the dataset containing course metadata, which includes information about course titles.
- **Clean the Data:** Preprocess the course titles by removing any special characters, punctuation marks, and unnecessary words (e.g., stop words) that do not provide meaningful insights. This step ensures that the word cloud focuses on relevant words.
- **Calculate Word Frequencies:** Calculate the frequency of each word appearing in the course titles. This can be done by iterating over the course titles and maintaining a count for each unique word.
- **Create the Word Cloud:** Use a word cloud library or tool (e.g., WordCloud in Python) to generate the word cloud visualization. Specify the frequency of each word as input, where larger frequencies result in larger word sizes in the cloud.
- **Visualize the Word Cloud:** Display the word cloud visualization, where words with higher frequencies are represented by larger font sizes. The word cloud provides an overview of the most commonly used words in the course titles, offering insights into popular topics or themes covered by the courses.

By following these steps, you will be able to generate a word cloud that visualizes the frequency of words appearing in course titles, allowing you to identify the most commonly used words and gain insights into the topics or themes covered by the courses.

CONTENT–BASED RECOMMENDER SYSTEM USING UNSUPERVISED LEARNING



FLOWCHART :USER PROFILE AND COURSE GERNE



STEPS

- **Retrieve user profile:** Obtain the user's profile, which may include information such as their preferred genres, previously watched courses, or ratings.
- **Calculate user profile vector:** Convert the user's profile information into a numerical vector representation. This vector captures the user's preferences and interests.
- **Retrieve course genre vectors:** Retrieve the genre vectors for all the available courses. These genre vectors represent the genres or topics associated with each course.
- **Calculate course similarity:** Measure the similarity between the user profile vector and each course genre vector. This can be done using similarity metrics such as cosine similarity or Euclidean distance.
- **Rank courses based on similarity:** Rank the courses based on their similarity to the user profile vector. The higher the similarity, the more relevant the course is to the user's preferences.
- **Filter out already watched courses:** Remove any courses that the user has already watched or interacted with to avoid recommending duplicates.
- **Return top recommended courses:** Select the top N courses with the highest similarity scores and present them as recommendations to the user.

RESULT

	USER	COURSE_ID	SCORE
0	37465	RP0105EN	[27.0]
1	37465	GPXX06RFEN	[12.0]
2	37465	CC0271EN	[15.0]
3	37465	BD0145EN	[24.0]
4	37465	DE0205EN	[15.0]
...	...	...	...
53406	2087663	excourse88	[15.0]
53407	2087663	excourse89	[15.0]
53408	2087663	excourse90	[15.0]
53409	2087663	excourse92	[15.0]
53410	2087663	excourse93	[15.0]

53411 rows × 3 columns

EVALUATION RESULT

On average, how many new/unseen courses have been recommended per user (in the test user dataset)

	userid	no_of_new_cr_recommended
0	37465	67
1	85625	131
2	108541	67
3	109915	2
4	149690	72
..	...	...
859	2056952	68
860	2061096	77
861	2074313	66
862	2074462	3
863	2087663	204

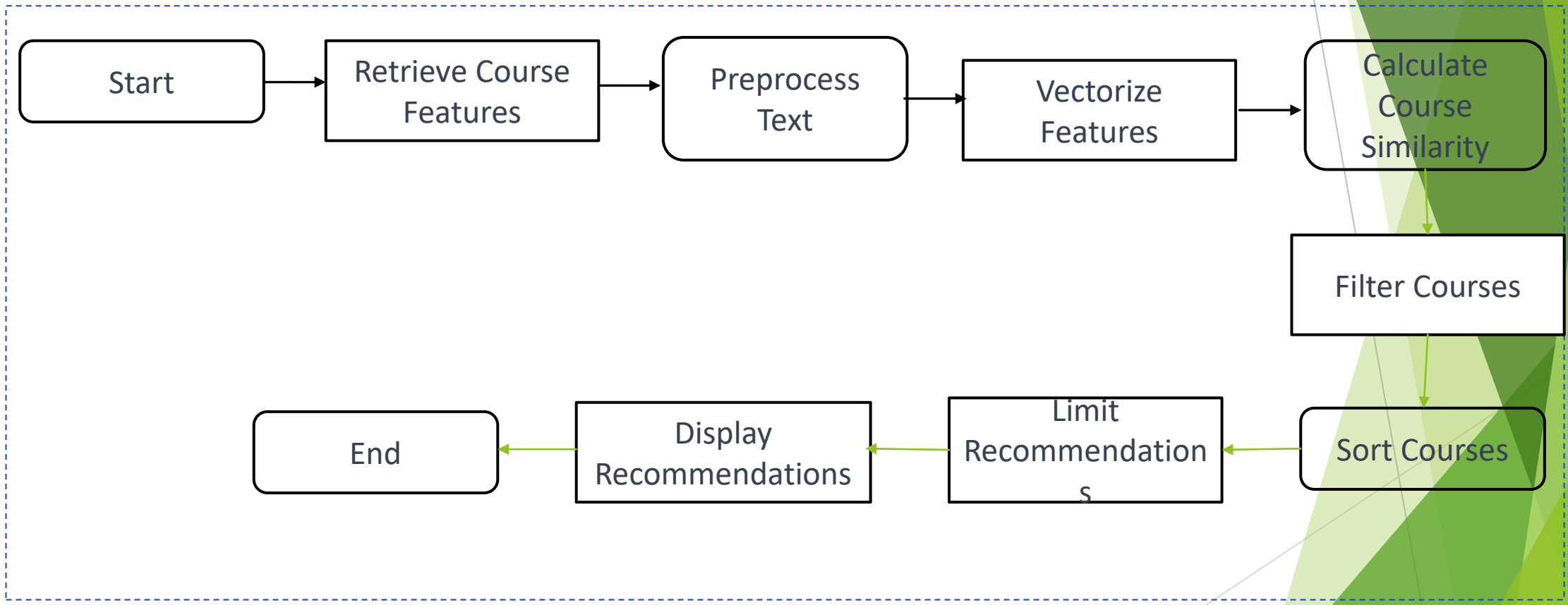
[864 rows x 2 columns]

What are the most frequently recommended courses? Return the top-10 commonly recommended courses across all users

```
print(df1)
```

	Course	Recc_score
0	TA0106EN	608
1	GPXX0IBEN	548
2	excourse21	547
3	excourse22	547
4	ML0122EN	544
5	GPXX0TY1EN	533
6	excourse04	533
7	excourse06	533
8	excourse31	524
9	excourse72	516

FLOWCHAT OF CONTENT - BASED RECOMMENDER STSTEM USING COURSE SIMILARITY



FIND COURSES WHICH ARE SIMILAR ENOUGH TO YOUR ENROLLED COURSES

- Retrieve the list of enrolled course IDs for a user from the course_df.
- Use the enrolled course IDs to obtain the corresponding course titles.
- Remove any duplicate course names from the enrolled courses.
- Obtain a list of all courses available.
- Identify the unselected courses by excluding the enrolled courses from the list of all courses.
- Call the function generate_recommendations_for_one_user for the user, providing the enrolled courses and unselected courses as inputs.
- Receive the recommended courses for the user.
- Repeat steps 1-7 for each user in the dataset.
- Analyze the recommended courses across all users to determine the most frequently recommended ones.

OUTPUT

```
res1 =  generate_recommendations_for_one_user(enrolled_course_ids, unselected_course_ids, id_idx_dict, sim_matrix)
res1
```

```
{'excourse23': 0.7397041774816828,
 'excourse36': 0.7397041774816828,
 'DS0110EN': 0.7329409123199365,
 'excourse32': 0.7220184676669504,
 'excourse63': 0.6945631057877047,
 'excourse38': 0.6816378987999318,
 'excourse33': 0.6645087922452757,
 'excourse65': 0.638641451227374,
 'excourse09': 0.6083303292403595}
```

STEPS : COURSE SIMILARITIES FOR ALL TEST USERS

Here are the steps to generate course recommendations based on course similarities for all test users:

- Load the test user data and identify the test users for whom you want to generate recommendations.
- Load the course similarity matrix that captures the similarity between courses.
- Load the course data that contains information about each course.
- Iterate over each test user.
- For each test user: a. Retrieve the enrolled courses of the user. b. Determine the unselected courses by excluding the enrolled courses. c. Calculate the similarity scores between the enrolled courses and unselected courses based on the course similarity matrix. d. Sort the unselected courses based on the similarity scores in descending order. e. Select the top recommended courses from the sorted list. f. Store the user, recommended courses, and their respective similarity scores.
- After iterating over all test users, you will have a list of recommended courses and their corresponding user and similarity scores.
- Analyze the recommended courses to determine the most frequently recommended ones across all users.
- Following these steps will help you generate course recommendations based on course similarities for all test users in your dataset.

RESULT :

```
res_dict = {}
users, courses, sim_scores = generate_recommendations_for_all()
res_dict['USER'] = users
res_dict['COURSE_ID'] = courses
res_dict['SCORE'] = sim_scores
res_df = pd.DataFrame(res_dict, columns=['USER', 'COURSE_ID', 'SCORE'])
res_df = res_df.dropna(subset=['COURSE_ID', 'SCORE'])
res_df = res_df[(res_df['COURSE_ID'].apply(len) > 0) | (res_df['SCORE'].apply(len) > 0)]
res_df
```

2	52091	[ML0120EN, ML0120ENv3, ML0120ENv2]	[0.9828731898973628, 0.9828731898973628, 0.982...
4	85625	[TMP0101EN, TA0105EN, BD0151EN]	[0.8894991799933215, 0.6598288790738579, 0.630...
5	108541	[excourse62, excourse22]	[0.6475015976638527, 0.6475015976638527]
6	109915	[WA0103EN]	[0.6311528416041716]
9	149690	[TMP0101EN, TA0105EN, BD0151EN]	[0.8894991799933215, 0.6598288790738579, 0.630...
...	...	...	...
988	2036996	[ML0115EN, ML0122EN]	[0.7071067811865475, 0.6815739441431503]
989	2046621	[excourse62, excourse22]	[0.6475015976638527, 0.6475015976638527]
992	2056704	[excourse62, excourse22]	[0.6475015976638527, 0.6475015976638527]
994	2057052	[excourse62, excourse22]	[0.6475015976638527, 0.6475015976638527]
996	2074313	[excourse62, excourse22]	[0.6475015976638527, 0.6475015976638527]

542 rows x 3 columns

RESULTS:

On average, how many new/unseen courses have been recommended per user (in the test user dataset)

how many new/unseen courses have been recommended to each user?

```
import numpy as np

# Assuming you have the DataFrame 'res_df' with 'USER' and 'COURSE_ID' columns

# Explode the nested lists in 'COURSE_ID' column
res_df = res_df.explode('COURSE_ID')

# Group by 'USER' and count the number of unique courses recommended for each user
user_course_counts = res_df.groupby('USER')['COURSE_ID'].nunique().reset_index()

# Calculate the average number of new/unseen courses recommended per user
average_new_courses = user_course_counts['COURSE_ID'].mean()

# Print the average number of new/unseen courses recommended per user
print("Average number of new/unseen courses recommended per user:", average_new_courses)
```

```
Average number of new/unseen courses recommended per user: 9.108856088560886
```

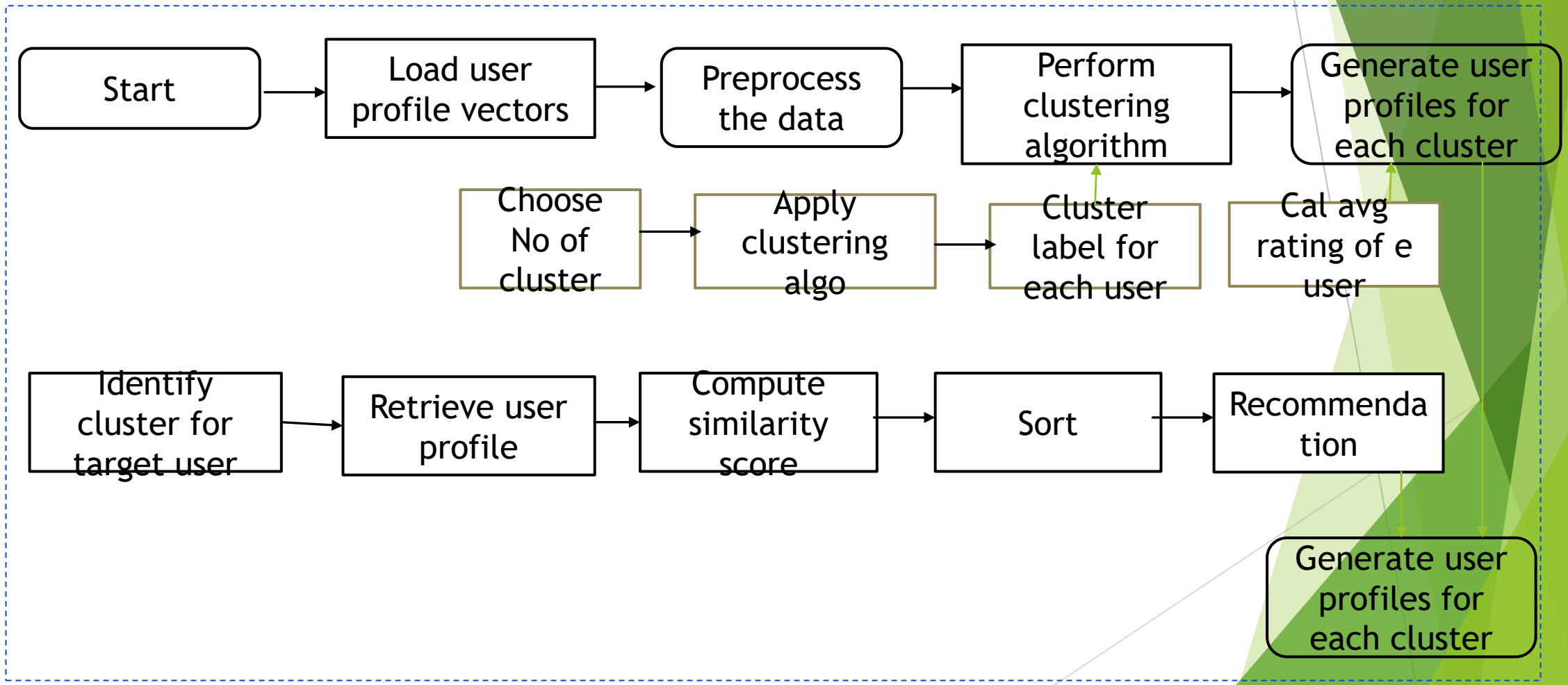
What are the most frequently recommended courses? Return the top-10 commonly recommended courses

What are the most frequently recommended courses? Return the top-10 commonly recommended courses across all users?

```
# Call the function and assign the returned DataFrame to a variable
recommendations_df = generate_recommendations_for_all()

# Count the occurrences of each course
course_counts = recommendations_df['COURSE_ID'].explode().value_counts()

# Get the top 10 most frequently recommended courses
top_10_courses = course_counts.head(10)

# Print the top 10 courses
print(top_10_courses)
```

```
excourse62    257
excourse22    257
WA0103EN      101
TA0105         41
DS0110EN       38
excourse47     24
excourse46     24
excourse65     23
excourse63     23
ML0151EN       17
Name: COURSE_ID, dtype: int64
```

FLOWCHART : CLUSTERING-BASED RECOMMENDER SYSTEM



STEPS

1. Load User Profile Vectors:

- Retrieve or generate user profile vectors.
- User profile vectors represent the preferences or characteristics of each user.
- Each user profile vector should contain relevant features or attributes that describe user preferences.
- Preprocess the Data:

2. Perform any necessary preprocessing steps on the user profile vectors.

- This may include handling missing values, normalizing or scaling the data, or encoding categorical variables.
- Preprocessing ensures that the user profile vectors are in a suitable format for clustering.

3. Perform Clustering Algorithm:

- Apply a clustering algorithm to group similar user profile vectors together.
- Common clustering algorithms for this task include k-means, hierarchical clustering, or density-based clustering.
- Determine the optimal number of clusters using techniques such as the elbow method or silhouette analysis.

4. Generate User Profiles for Each Cluster:

- Once the clustering algorithm has been applied, assign each user profile vector to its corresponding cluster.
- For each cluster, compute the average or centroid of the user profile vectors within that cluster.
- The resulting centroids represent the user profiles for each cluster, capturing the average preferences of users in that cluster.

5 Generate Recommendations for Users:

- Given a new user, assign their user profile vector to the closest cluster centroid based on distance or similarity measures.
- Within the assigned cluster, generate personalized recommendations for the user.
- Recommendations can be based on collaborative filtering techniques, content-based filtering, or hybrid approaches.

6. Evaluate and Refine the Recommender System:

- Evaluate the performance of the clustering-based recommender system using appropriate evaluation metrics.
- Compare the recommendations generated with user feedback or ratings to assess the effectiveness of the system.
- Iterate and refine the system based on evaluation results to improve recommendation quality

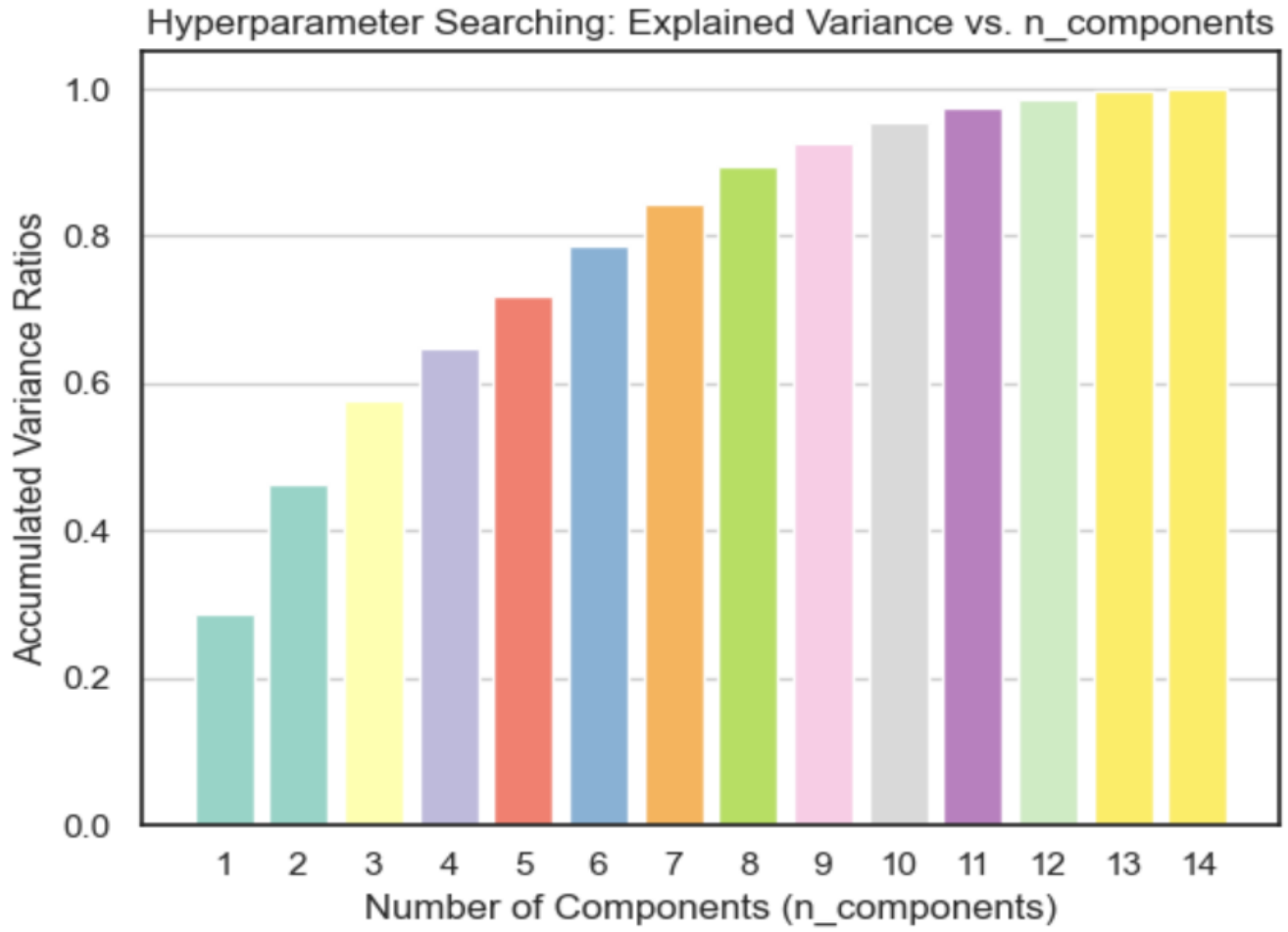
Please note that these steps provide a high-level overview of the process. The specific implementation and choice of algorithms may vary based on the dataset and requirements of the recommender system

RESULT

	user	cluster
0	2	4
1	4	4
2	5	4
3	7	8
4	8	8
...	...	...
33896	2102054	1
33897	2102356	8
33898	2102680	3
33899	2102983	8
33900	2103039	8

33901 rows × 2 columns

HYPERPARAMETER SEARCHING PROCESS



PCA TRANSFORMED DATAFRAME.

- Once you found the optimized n_component argument value, you can apply PCA on the user profile feature vectors and reduce the 14 features into n_component features.
- Your PCA transformed dataframe is below:

]:

	user	PC0	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8
0	2	17.772494	0.200681	1.730609	2.567359	-3.825814	2.707154	0.681042	2.312613	0.868272
1	4	7.145199	-2.847481	2.358636	-0.576654	0.398803	-0.134533	0.549769	0.469033	0.033405
2	5	11.363270	1.873619	-1.522077	1.076144	-1.711688	0.883212	1.677582	2.937669	2.097639
3	7	-1.834033	-0.277462	0.564905	0.053470	-0.064440	0.165757	0.030956	0.039519	0.210887
4	8	-1.049125	-0.684767	1.072765	0.006371	-0.005695	0.118686	0.118559	0.559292	0.186379
...	...	...	...	...	...	...	...	...	...	...
33896	2102054	0.633824	0.108815	-0.388871	-0.122665	-0.098364	0.358333	1.752049	1.486542	-0.523600
33897	2102356	-2.095339	0.135058	0.244727	-0.088185	0.025081	0.183641	0.046413	0.191709	0.260437
33898	2102680	0.625943	-0.547167	-1.692824	-0.630589	0.166632	0.676244	-0.055100	0.582091	1.703193
33899	2102983	-2.036832	-0.153534	0.162852	0.082651	-0.126419	0.255109	0.072496	0.113750	0.622900
33900	2103039	-2.036832	-0.153534	0.162852	0.082651	-0.126419	0.255109	0.072496	0.113750	0.622900

33901 rows × 10 columns

COURSE RECOMMENDATION: BASED ON POPULAR COURSES IN SAME CLUSTERS

	user	item	Cluster
0	1502801	RP0105EN	8
1	1502801	BD0131EN	8
2	1502801	BD0212EN	8
3	1502801	BD0115EN	8
4	1502801	BD0211EN	8
...	...	...	...
9397	630511	BD0121EN	0
9398	630511	SC0101EN	0
9399	630511	BD0111EN	0
9400	630511	BD0115EN	0
9401	630511	PY0101EN	0

9402 rows × 3 columns

EVALUATION RESULT

On average, how many new/unseen courses have been recommended per user (in the test user dataset)

On average, how many new/unseen courses have been recommended to each user?

57]:
```
total_recommendations = 0
num_users = len(recommendations)

for user_id, courses in recommendations.items():
    total_recommendations += len(courses)

average_recommendations = total_recommendations / num_users

print("Average number of new/unseen courses recommended per user:", average_recommendations)
```

```
Average number of new/unseen courses recommended per user: 50.0
```

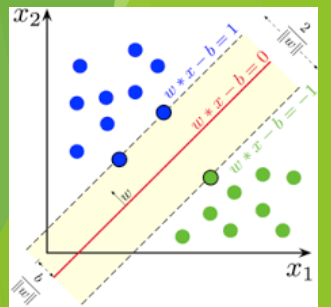
EVALUATION RESULT

What are the most frequently recommended courses? Return the top-10 commonly recommended courses

```
from collections import Counter

# Collect all recommended courses
all_courses = [course for courses in recommendations.values() for course in courses]

# Count the frequency of each course
course_counts = Counter(all_courses)

# Get the top-10 most common courses
top_10_courses = course_counts.most_common(10)

# Print the top-10 most common courses
print("Top-10 commonly recommended courses:")
for course, count in top_10_courses:
    print(course, ":", count)
```

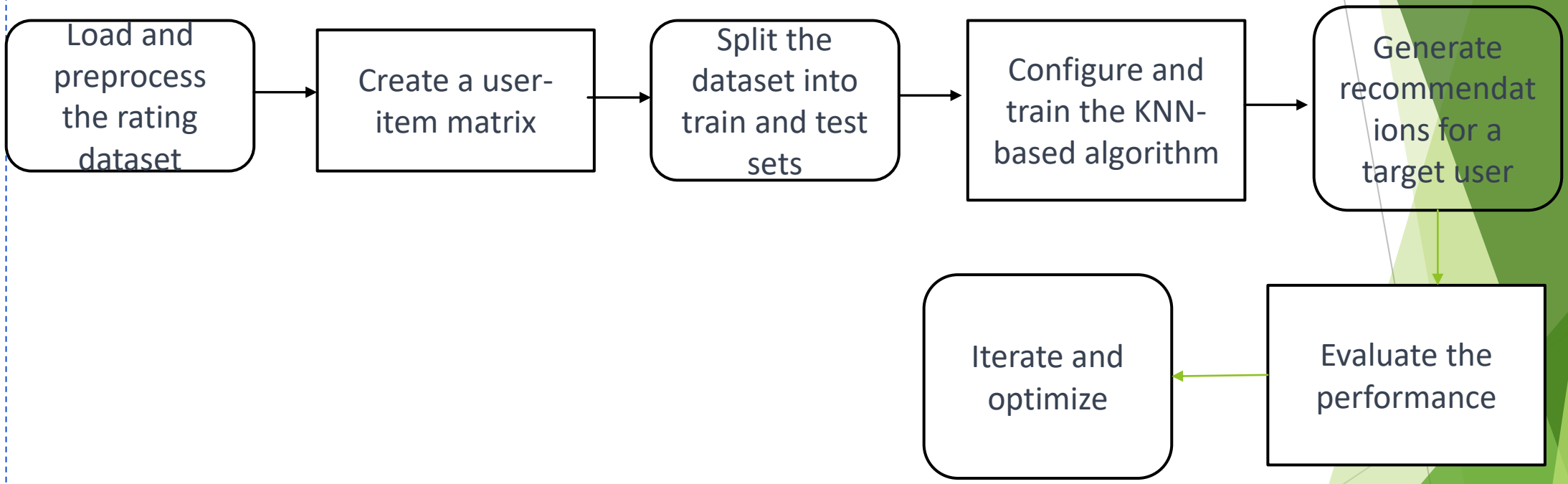
```
Top-10 commonly recommended courses:
DE0205EN : 1000
TA0105 : 1000
CC0101EN : 1000
ML0151EN : 1000
DV0101EN : 1000
BD0145EN : 1000
BD0212EN : 1000
BC0101EN : 1000
SC0105EN : 1000
DS0103EN : 1000
```

COLLABORATIVE - FILTERING RECOMMENDER SYSTEM USING SUPERVISED LEARNING



FLOWCHART OF KNN BASED RECOMMENDER SYSTEM

Here Explaining the flow chart of how to use KNN using surprise library.



STEPS:

- **Load and preprocess the rating dataset:** Load the rating dataset into the Surprise library's data structure (e.g., Dataset, Reader). Preprocess the dataset if necessary (e.g., handling missing values, normalizing ratings).
- **Create a user-item matrix:** Convert the dataset into a user-item matrix using the pivot operation. The user id will form the rows, the item id will form the columns, and the ratings will fill the matrix cells. This operation will create a sparse matrix representation of the user-item interactions.
- **Split the dataset into train and test sets:** Split the dataset into a training set and a test set to evaluate the performance of the recommender system.
- **Configure and train the KNN-based algorithm:** Instantiate a KNN-based algorithm in Surprise (e.g., KNNBasic, KNNWithMeans, KNNWithZScore). Configure the algorithm parameters such as the similarity measure, number of neighbors (K), etc. Fit the algorithm to the training set.
- **Generate recommendations for a target user:** Select a target user for whom you want to generate recommendations. Retrieve the target user's ratings from the dataset. Use the trained KNN algorithm to find the K nearest neighbors of the target user based on their ratings. Aggregate the ratings of the K neighbors to generate recommendations for the target user.
- **Evaluate the performance:** Use the test set to evaluate the performance of the KNN-based recommender system. Calculate relevant evaluation metrics such as accuracy, precision, recall, RMSE, etc.
- **Iterate and optimize:** Experiment with different algorithm configurations, such as different similarity measures, K values, and preprocessing techniques, to optimize the performance of the recommender system. Iterate and fine-tune the algorithm until satisfactory results are achieved.

KNN –BASIC AND WITH HYPERPARAMETER TUNNING

18]:
```
## WRITE YOUR CODE HERE:


# - Define a KNNBasic() model

knn = KNNBasic()

# Train the algorithm on the trainset, and predict ratings for the testset
knn.fit(trainset)
predictions = knn.test(testset)

# Then compute RMSE
accuracy.rmse(predictions)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.1964
```

18]:
```
0.19637678039310447
```

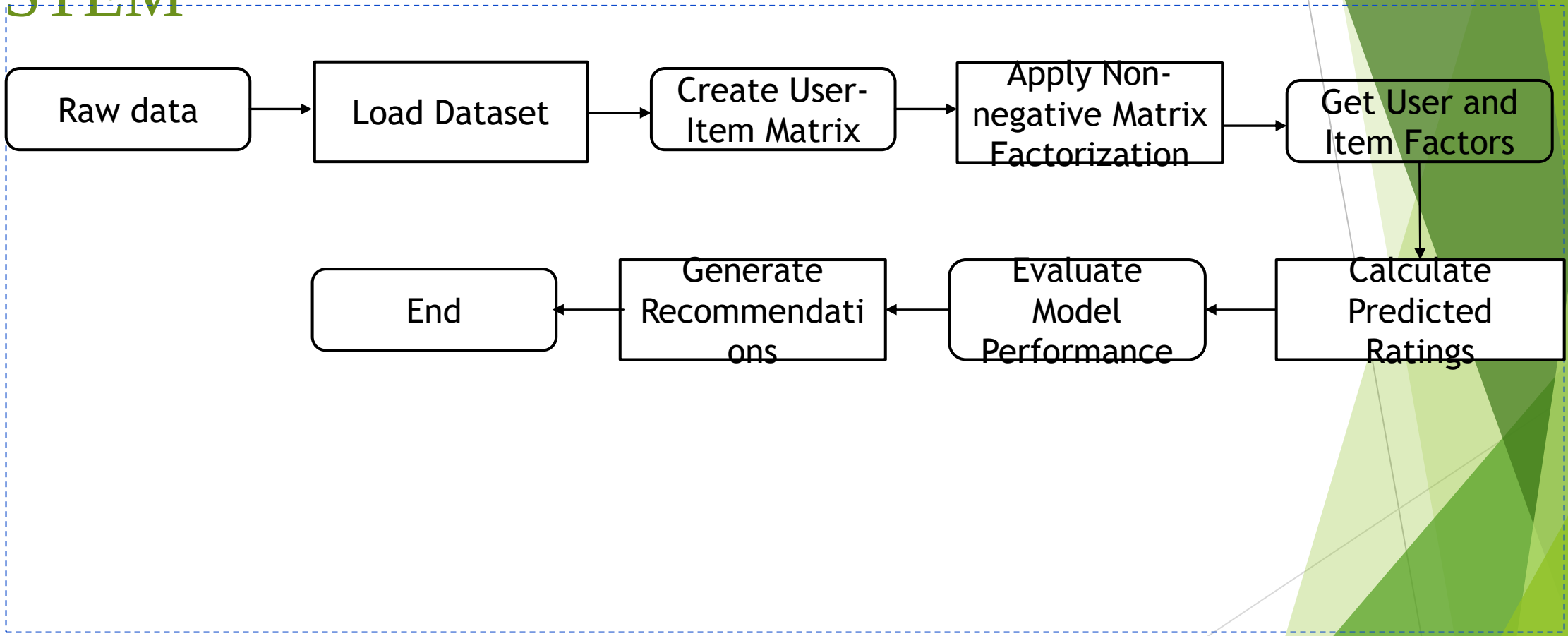
23]:
```
knn1 = KNNBasic(sim_option = {'name': 'cosine', 'user_based':False})

# Train the algorithm on the trainset, and predict ratings for the testset
knn1.fit(trainset)
predictions1 = knn.test(testset)

# Then compute RMSE
accuracy.rmse(predictions1)
```

```
Computing the msd similarity matrix...
Done computing similarity matrix.
RMSE: 0.1964
```

23]:
```
0.19637678039310447
```

- KNN-based collaborative filtering is probably the simplest but very effective.
- Since it is based on KNN, it inherits the main characteristics of KNN such as memory-intensive because you need to maintain a huge similarity matrix among users or items.

FLOWCHART : NMF BASED RECOMMENDER SYSTEM

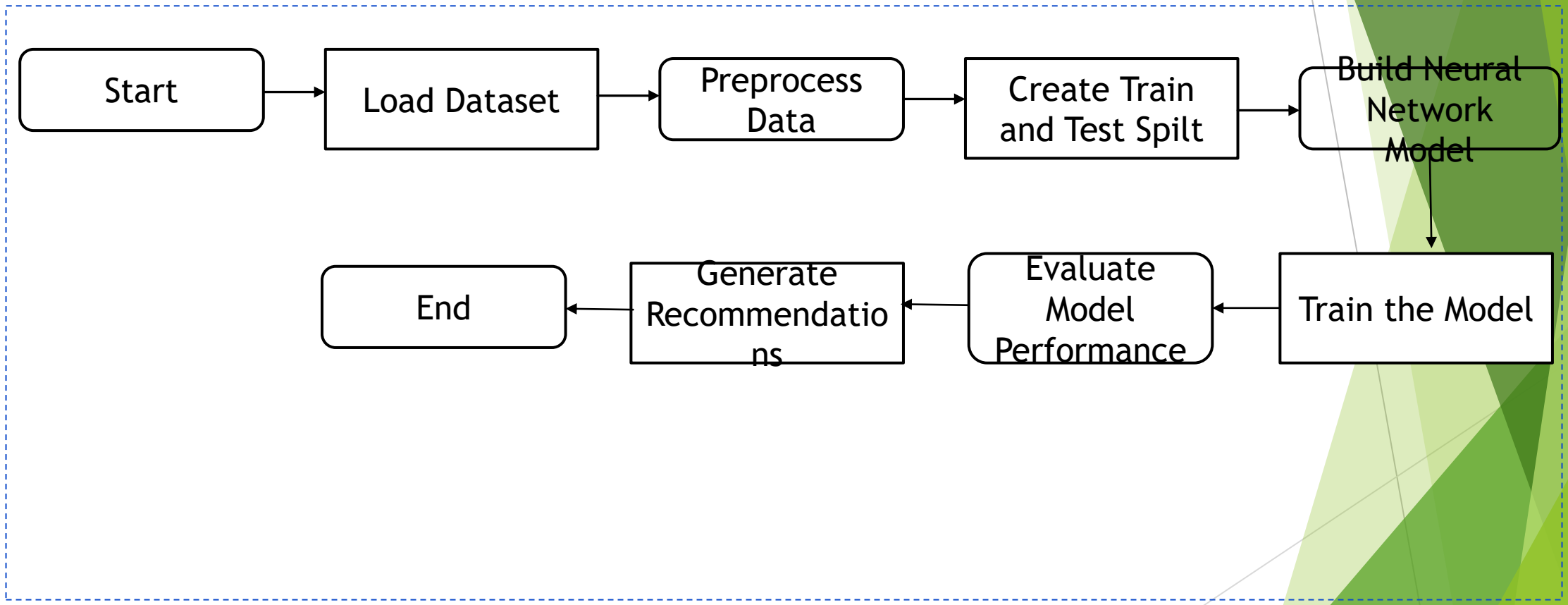


STEPS:

- **Load Dataset:** Load the rating dataset, which contains information about user-item interactions such as user ids, item ids, and ratings.
- **Create User-Item Matrix:** Convert the dataset into a user-item matrix representation, where rows represent users and columns represent items. The matrix will capture the ratings given by users to items.
- **Apply Non-negative Matrix Factorization:** Use the NMF algorithm to factorize the user-item matrix into two lower-rank matrices: the user factors matrix and the item factors matrix. NMF decomposes the original matrix into a product of these two matrices, capturing latent factors that represent user preferences and item characteristics.
- **Get User and Item Factors:** Retrieve the user factors matrix and the item factors matrix obtained from the NMF decomposition. These matrices contain the learned latent factors for each user and item.
- **Calculate Predicted Ratings:** Multiply the user factors matrix and the item factors matrix to reconstruct the original user-item matrix. The resulting matrix represents the predicted ratings for all users and items.
- **Evaluate Model Performance:** Measure the performance of the NMF-based recommender system using evaluation metrics such as RMSE (Root Mean Squared Error) or precision/recall. Compare the predicted ratings with the actual ratings in the test set to assess the accuracy of the recommendations.
- **Generate Recommendations:** Use the predicted ratings to generate recommendations for users. Sort the items based on the predicted ratings and recommend the top-rated items to each user.
- **End:** Finish the process.

It's important to note that this flowchart provides a high-level overview of the steps involved in building a Collaborative Filtering based Recommender System using Non-negative Matrix Factorization. The specific implementation details may vary depending on the programming language, libraries, and tools being used.

FLOWCHART : NEURAL NETWORK EMBEDDING BASED RECOMMENDER SYSTEM



STEPS

- **Load Dataset:** Load the dataset containing user-item interactions and ratings.
- **Preprocess Data:** Clean and preprocess the data as necessary. This may involve handling missing values, normalizing ratings, or encoding categorical variables.
- **Create Train and Test Sets:** Split the dataset into a training set and a test set. The training set will be used to train the neural network model, while the test set will be used to evaluate its performance.
- **Build Neural Network Model:** Design and build a neural network model for the recommender system. This typically involves defining the architecture of the neural network, including the number of layers, type of activation functions, and other hyperparameters.
- **Train the Model:** Train the neural network model using the training set. This involves feeding the training data through the model, adjusting the weights and biases using an optimization algorithm (e.g., stochastic gradient descent), and iteratively updating the model to minimize the loss function.
- **Evaluate Model Performance:** Evaluate the performance of the trained model using the test set. This may involve calculating evaluation metrics such as accuracy, precision, recall, or mean squared error to assess how well the model predicts user-item interactions.
- **Generate Recommendations:** Once the model is trained and evaluated, use it to generate recommendations for users. This can be done by feeding user profiles or queries into the model and obtaining predicted ratings or item rankings based on the learned embeddings.
- **End:** Finish the process.
- It's important to note that the specific architecture and design of the neural network model may vary depending on the specific goals and requirements of the recommender system. Additionally, other steps such as hyperparameter tuning, regularization techniques, or additional preprocessing steps may be incorporated based on the specific implementation.

OUTPUT:SURPRISE

TASK: Perform NMF-based collaborative filtering on the course-interaction matrix

TODO: Fit a NMF model using the trainset and evaluate the results using the testset The code will be very similar to the KNN-based collaborative filtering, you just need to use the `NMF()` model.

```
## WRITE YOUR CODE HERE:


# - Define a NMF model NMF(verbose=True, random_state=123)
model1=NMF()
model = NMF(verbose=True,random_state=123,init_low=0.5, init_high = 5.0, n_factors=32)
# - Train the NMF on the trainset, and predict ratings for the testset
model1.fit(trainset)
model.fit(trainset)
predictions1 = model1.test(testset)
predictions = model.test(testset)
# - Then compute RMSE
print("Accuracy of Basic model :",accuracy.rmse(predictions1))
print("Accuracy of hypertuned model :",accuracy.rmse(predictions))
# more NMF model hyperparamets can be found here:
```

```
Processing epoch 48
Processing epoch 49
RMSE: 0.2125
Accuracy of Basic model : 0.21249125426627416
RMSE: 0.1948
Accuracy of hypertuned model : 0.19480872123558915
```

OUTPUT : SKLEARN

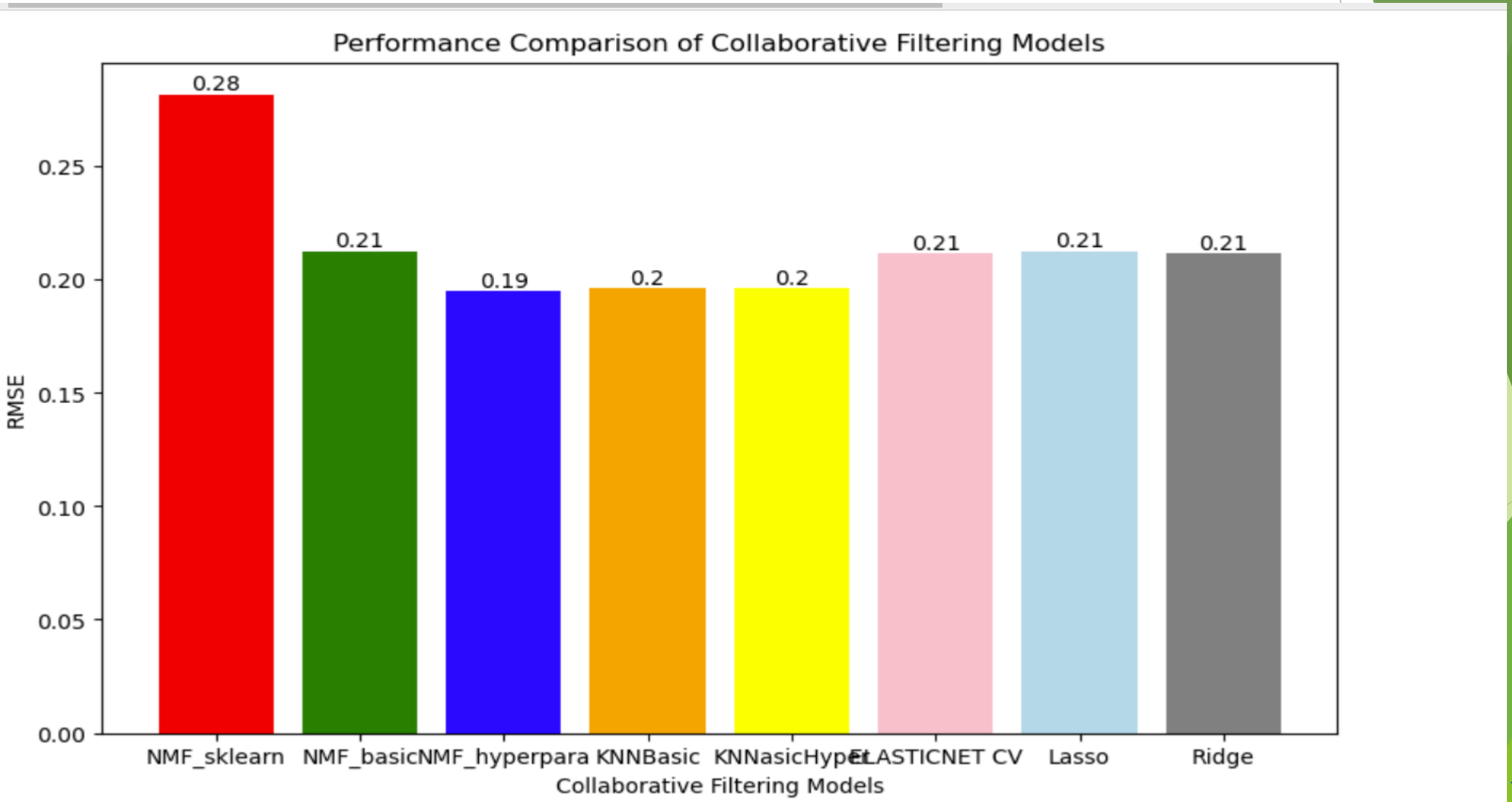
Implementation Option 2: Use `numpy`, `pandas`, and `sklearn`.

If you do not prefer the one-stop Suprise solution, you may implement the KNN model using `numpy`, `pandas`, and possibly `sklearn`:

```
]: ## WRITE YOUR CODE HERE:

from sklearn.decomposition import NMF
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Read the course rating dataset from the CSV file
course_ratings_df = pd.read_csv("course_ratings.csv")

# Create user-item matrix for the entire dataset
full_matrix = course_ratings_df.pivot(index='user', columns='item', values='rating')

# Replace NaN values with 0
full_matrix = full_matrix.fillna(0)

# Convert the full matrix to a numpy array
full_array = full_matrix.values

# Split the dataset into train and test sets
train_array, test_array = train_test_split(full_array, test_size=0.3)

# Define NMF model with desired number of components
model = NMF(n_components=32, random_state=123)

# Fit the model on the train array
```

```
  warnings.warn(

NMF RMSE: 0.2816597001453128
```

COMPARE THE PERFORMANCE OF COLLABORATIVE-FILTERING MODELS



CONCLUSION

Based on the problem statements and the analysis performed, we can draw the following conclusions:

- Enrollment Distributions:
  - The distribution of enrollments across users provides insights into the popularity and usage patterns of online courses.
  - By aggregating enrollment counts for each user, we can identify the total number of users and understand the distribution of enrollments among them.
- Word Cloud Visualization:
  - The word cloud visualization of course titles helps identify the most frequently occurring words, indicating popular topics or themes in the courses.
  - It provides a quick overview of the key areas of interest covered by the courses and helps understand the general focus of the course offerings.
- Clustering-Based Recommender System:
  - The clustering-based recommender system groups users based on their preferences or characteristics, as represented by user profile vectors.
  - By clustering users with similar preferences, the system can generate personalized recommendations for each user based on the preferences of users in the same cluster.
  - This approach allows for targeted recommendations and can help improve the relevance and effectiveness of the recommendation system.
- Overall, the analysis and techniques applied in this project provide valuable insights into user enrollments, course topics, and the implementation of a clustering-based recommender system. These insights can be used to optimize course offerings, improve user engagement, and enhance the recommendation system to deliver more personalized and relevant recommendations to users.

APPENDIX

- **Git Hub :** sadhanajarag/Machine-Learning-final-project (github.com)
- **Python code snippets :** sadhanajarag/Machine-Learning-final-project (github.com)
- **Notebook outputs :** sadhanajarag/Machine-Learning-final-project (github.com)

THANK YOU !!!!