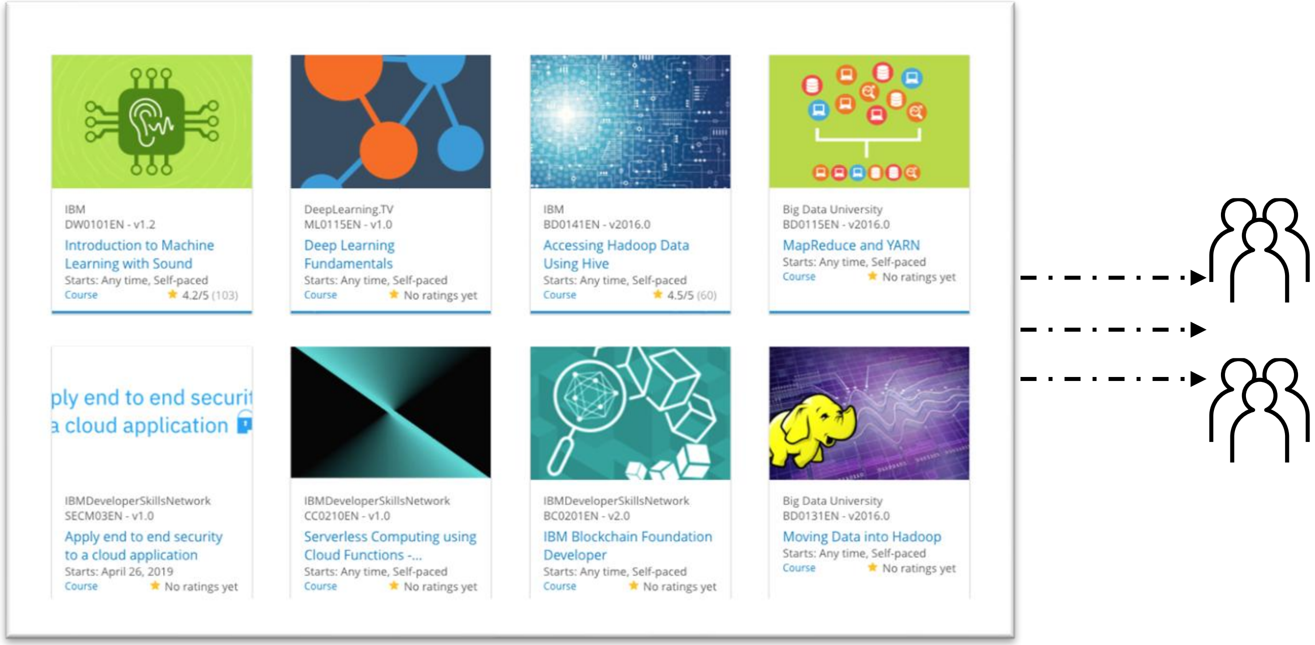
Build a Personalized Online Course Recommender System with Machine Learning

Name: Avik Das
Date: 23-07-2025



IBM
DW0101EN - v1.2
Introduction to Machine Learning with Sound
Starts: Any time, Self-paced
Course ★ 4.2/5 (103)

DeepLearning.TV
ML0115EN - v1.0
Deep Learning Fundamentals
Starts: Any time, Self-paced
Course ★ No ratings yet

IBM
BD0141EN - v2016.0
Accessing Hadoop Data Using Hive
Starts: Any time, Self-paced
Course ★ 4.5/5 (60)

Big Data University
BD0115EN - v2016.0
MapReduce and YARN
Starts: Any time, Self-paced
Course ★ No ratings yet

IBMDeveloperSkillsNetwork
SECM03EN - v1.0
Apply end to end security to a cloud application
Starts: April 26, 2019
Course ★ No ratings yet

IBMDeveloperSkillsNetwork
CC0210EN - v1.0
Serverless Computing using Cloud Functions -...
Starts: Any time, Self-paced
Course ★ No ratings yet

IBMDeveloperSkillsNetwork
BC0201EN - v2.0
IBM Blockchain Foundation Developer
Starts: Any time, Self-paced
Course ★ No ratings yet

Big Data University
BD0131EN - v2016.0
Moving Data into Hadoop
Starts: Any time, Self-paced
Course ★ No ratings yet

Outline

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

Introduction

Project background and context :

- This project focuses on Recommender Systems & Predictive Models.
- It is designed to apply and demonstrate proficiency in machine learning skills, leveraging Python-based libraries such as Pandas, sci-kit-learn, and Tensorflow/Keras.
- A central theme of the capstone is the creation of various recommender systems, with all subsequent labs building upon this concept.
- The project involves analyzing online course-related datasets, performing exploratory data analysis to find preliminary insights, and extracting features like a "bag of words" (BoW) from course titles and descriptions.
- The overarching goal of the project is to demonstrate the ability to successfully create and implement various recommender systems and predictive models by applying the machine learning algorithms and techniques learned.

Introduction

Problem statements & Hypotheses :

- How can we effectively recommend relevant courses to users?
    - This involves developing course recommendation systems using different methods, including those based on user profiles, course genres, and computed interest scores.
    - We will create recommendation systems by generating a course similarity matrix.
    - The project implements clustering-based recommender system algorithms using K-means clustering and Principal Component Analysis (PCA).
    - Collaborative filtering techniques will be applied, specifically KNN-based collaborative filtering and non-negative matrix factorization.
    - A key step is to calculate course similarity using cosine similarity based on extracted "bag of words" feature vectors.
- How can we accurately predict course ratings and learner engagement?
    - This includes training neural networks to predict course ratings and simultaneously extract users' and items' latent features.
    - We will construct regression models to calculate numerical rating scores that predict whether a student will audit or complete a course.
    - Classification models will also be developed to predict whether a learner will audit or complete a course, mapping an interaction feature vector to a rating mode.

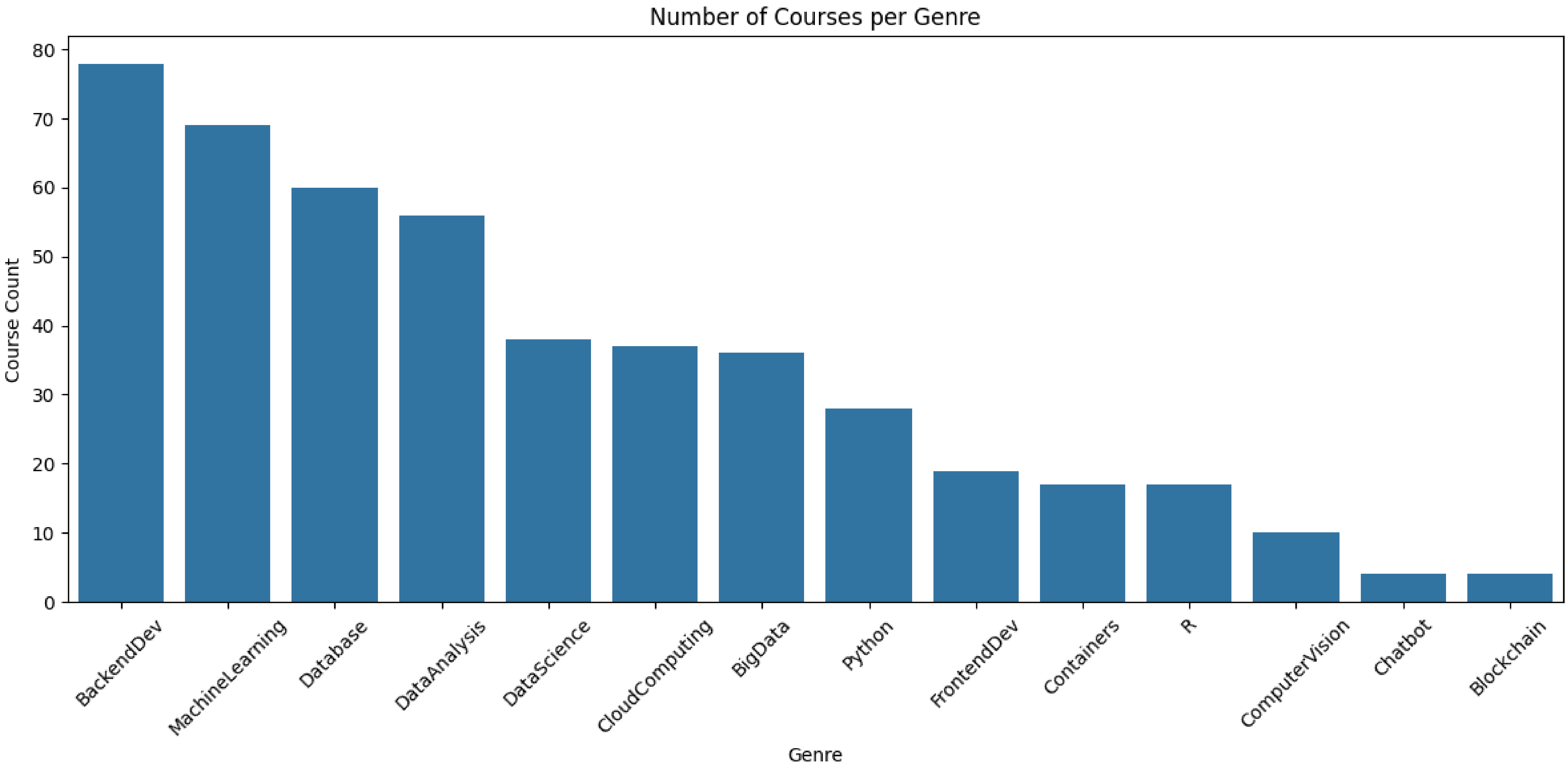
Introduction

Main Objectives :

- Compare and contrast different machine learning algorithms for recommender systems.
- Predict course ratings using neural networks, regression, and classification.
- Apply KNN, PCA, and non-negative matrix collaborative filtering for recommendations.
- Understand and implement exploratory data analysis, Bag of Words, and cosine similarity.
- Train neural networks for latent feature extraction and rating prediction.
- Construct and evaluate both regression and classification models for predictive tasks.

Exploratory Data Analysis

Course counts per genre

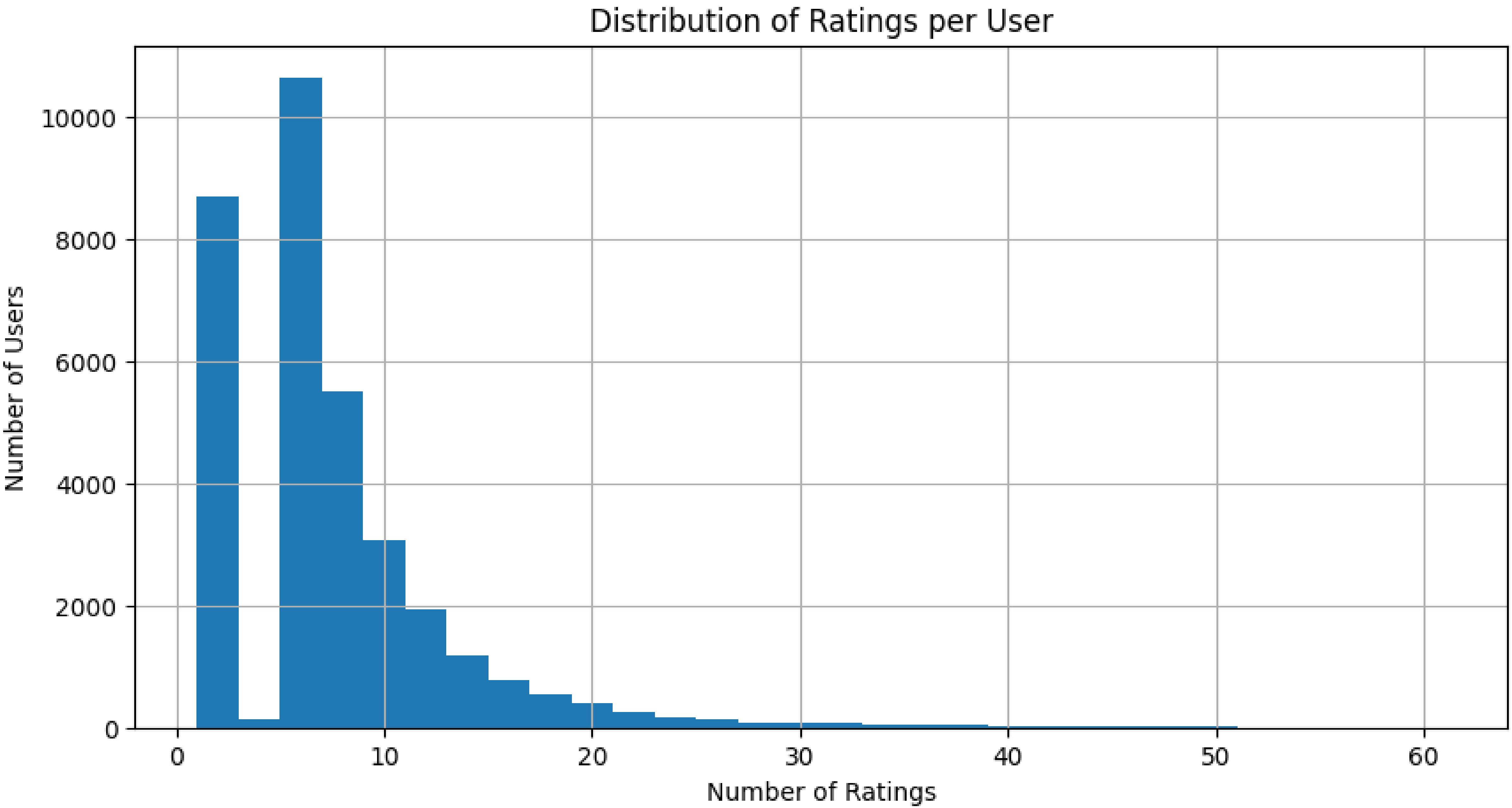


Course counts per genre

The barchart titled "**Number of Courses per Genre**" visually represents the **total count of courses associated with each specific genre**. This chart is generated after calculating the course count for each genre from the course_df dataset.

- It displays the **popularity of different course genres** within the dataset.
- The **x-axis** of the barchart represents the different **genres**, and the **y-axis** indicates the **Course Count** for each genre.
- The genres are typically **sorted in descending order by their course count**, allowing for quick identification of the most and least popular genres
- For example, based on the data used to create the barchart, **BackendDev** is the most popular genre with 78 courses, followed by **MachineLearning** with 69 courses, and **Database** with 60 courses. Genres like **Chatbot** and **Blockchain** have the lowest counts, each with 4 courses.
- The visualization helps provide a **solid understanding of course metadata and popular course genres** as part of the exploratory data analysis on online course enrollment data.

Course enrollment distribution



Course enrollment distribution

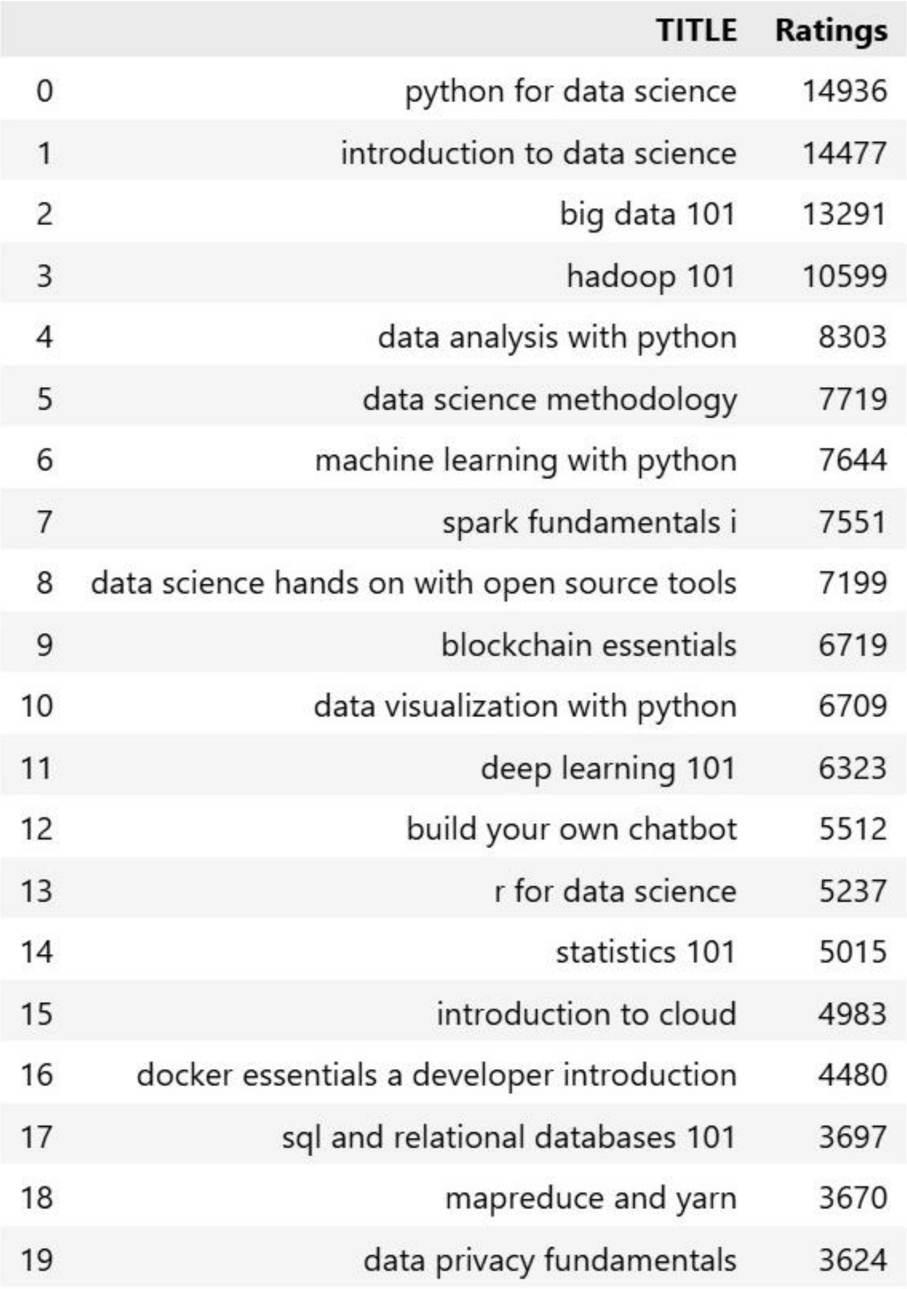
The histogram showing the enrollment distributions visually represents how many courses each user has rated.

- **Purpose**: The histogram allows us to understand the **distribution of ratings per user**. It answers questions like "how many users rated just 1 item or how many rated 10 items, etc.". This provides insight into user engagement and behavior within the online course platform.

- **Axes**:
    - The **x-axis** represents the **"Number of Ratings"** (i.e., the number of courses a user has rated).
    - The **y-axis** represents the **"Number of Users"** (i.e., how many users fall into each category of rating counts).

- **Insight from Data**: For example, based on the summary statistics of user rating counts, the average user rated approximately **6.88 courses**, with the minimum being **1 course** and the maximum being **61 courses**. The histogram would visually display how these 33,901 unique users are spread across these rating counts, showing, for instance, a large number of users who have rated only a few courses versus a smaller number of highly engaged users who have rated many courses.

20 most popular courses



	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

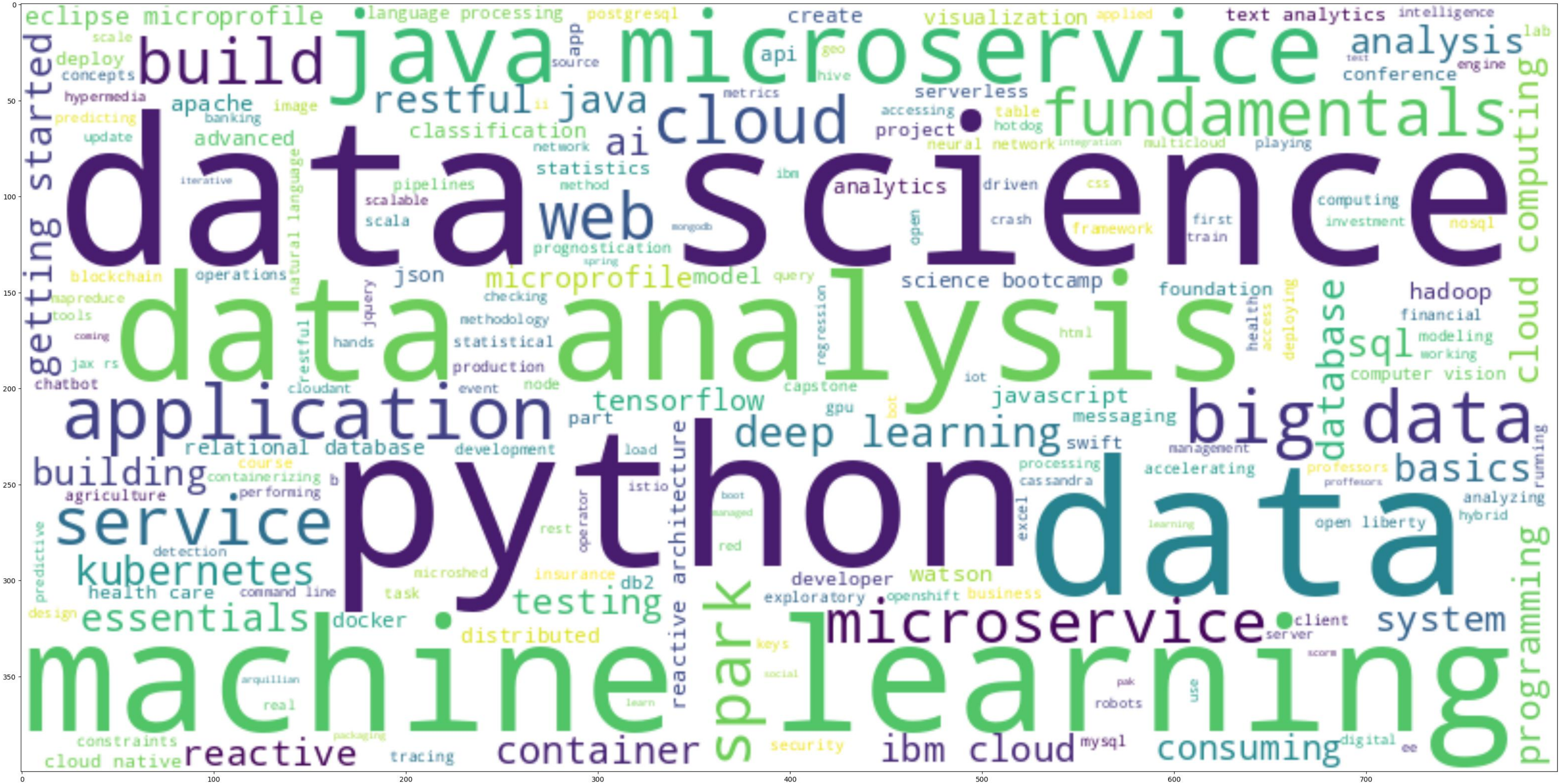
20 most popular courses

- **Purpose**: The list identifies the **top 20 courses** that have received the **most ratings/enrollments**. This helps in understanding which courses are highly engaged with and potentially recommended by learners.

- **Generation Process**:
  - The ratings_df dataset, which contains user IDs, course IDs (items), and ratings, is used.
  - Courses (item column) are **grouped together**, and the **total number of ratings (enrollments) for each course is counted**.
  - These course counts are then **sorted in descending order** to identify courses with the highest engagement.
  - Finally, the **top 20 courses** from this sorted list are selected.
  - To make the list more understandable, the course IDs are **merged with the course_df** to include their **actual titles**.

- **Key Insights**:
  - For example, courses like "**python for data science**" (14936 ratings), "**introduction to data science**" (14477 ratings), and "**big data 101**" (13291 ratings) are among the most popular.
  - This list reveals that these top 20 courses account for a **significant portion of the total enrollments**. Specifically, they represent **63.3% of all course enrollments** in the dataset.

Word cloud of course titles



Word cloud of course titles

The "Word cloud of course titles" is a visual representation designed to provide an **intuitive summary of the keywords present in all course titles** within the dataset. It is an effective tool for exploratory data analysis (EDA) to gain preliminary insights into the data.

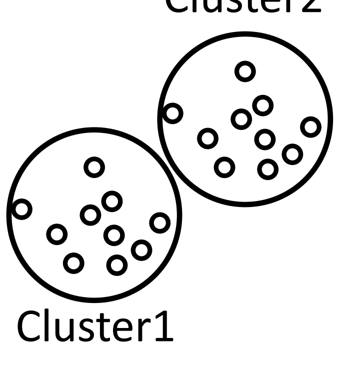
- **Purpose**: The primary goal is to **determine prevalent keywords** in the course titles. This helps in getting a general understanding of the types of courses available in the dataset.

- **Generation Process**:
  - First, all course titles from the course_df are joined into a single large string.
  - Common **"stop words" and less meaningful words** (e.g., "getting started," "using," "enabling," "template," "university," "end") are filtered out from this string to ensure that only significant terms are highlighted.
  - A WordCloud object is then created and generated from the processed text.
  - The plt.imshow() method is used to visualize the generated word cloud.

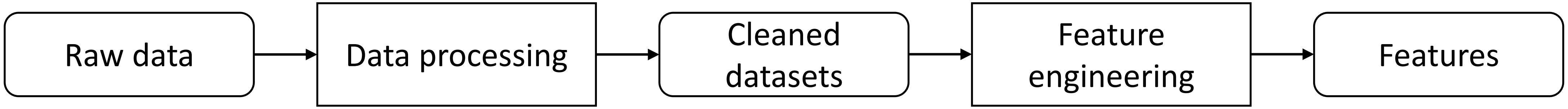
- **Visual Representation**: The word cloud displays words from the course titles, where the **size of each word corresponds to its frequency** or importance in the collection of titles. Larger words indicate higher frequency.

- **Key Insights**: From the word cloud, it becomes evident that many popular IT-related keywords are prominent, such as "**python**, **data science**, **machine learning**, **big data**, **ai**, **tensorflow**, **container**, and **cloud**". This indicates that the courses in the dataset primarily focus on **demanding IT skills**.

Content-based Recommender System using Unsupervised Learning



Flowchart of content-based recommender system using user profile and course genres



Raw data → Data processing → Cleaned datasets → Feature engineering → Features

The content-based recommender system works by recommending items to users based on their profiles, which reflect their preferences and tastes. The recommendation process relies on the similarity between items, measured by their content or features like genre.

1. **User Profile Generation**:
   - ◦ **User profiles** are mathematically represented as **user feature vectors** that quantify a user's learning interests.
   - ◦ These profiles are generated by **multiplying a user's rating vector with a course genre matrix**. This produces a "weighted genre vector" which shows a user's interest in each genre based on courses they have rated. For example, a user who completed a "Machine Learning with Python" course would show high interest in Python and Machine Learning genres.

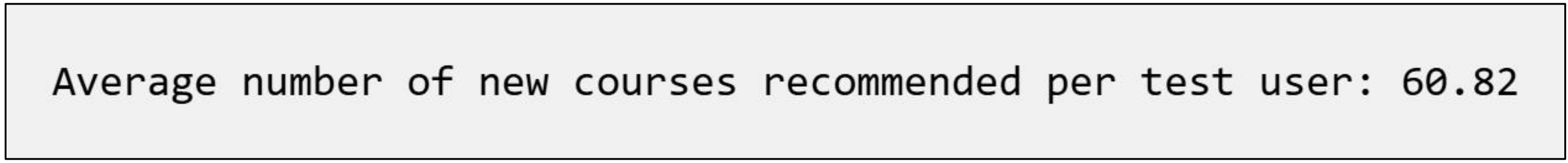
2. **Recommendation Score Calculation**:
   - ◦ Once **user profile vectors** and **course genre feature vectors** for new courses are constructed, a **simple dot product** is used to compute an **interest score** for each new course.
   - ◦ A **larger dot product value indicates higher similarity** between the user profile and the course genre vector, implying greater potential interest. This is because if a user is interested in certain genres and a course has those same genres, their vectors will share common dimensions, leading to a high dot product.

3. **Recommendation Filtering & Output**:
   - ◦ Courses are recommended based on these high interest scores.
   - ◦ Recommendations can be filtered by applying a **score threshold** (e.g., only scores above 10.0) or by **sorting scores and returning the top-ranked courses** for each user. This ensures that only relevant courses with high confidence are recommended, ideally keeping the number of recommendations per user to less than 20 to avoid overwhelming them.

Evaluation results of user profile-based recommender system

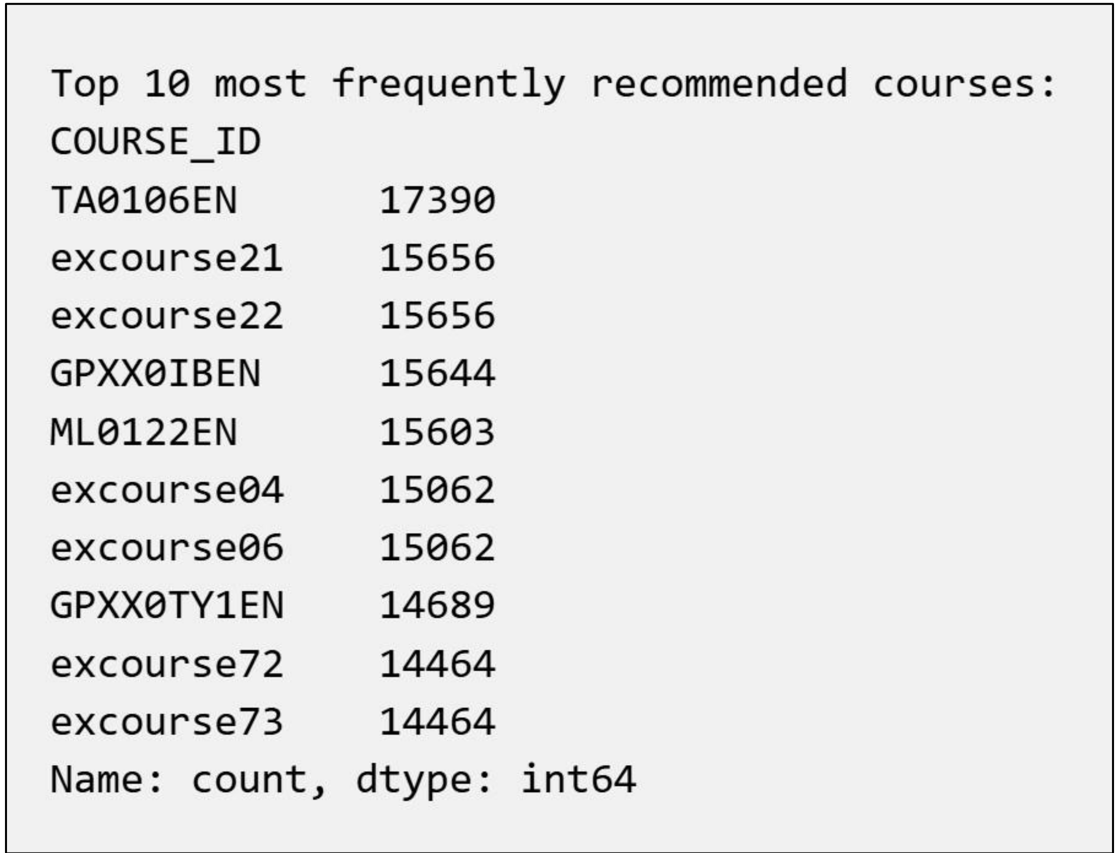
Hyper-Parameter Settings :- **score_threshold = 10.0**

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



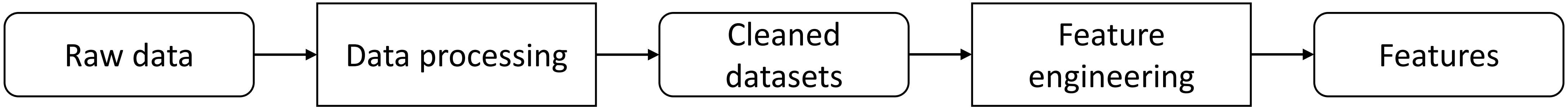
```
Average number of new courses recommended per test user: 60.82
```

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



```
Top 10 most frequently recommended courses:
COURSE_ID
TA0106EN       17390
excourse21     15656
excourse22     15656
GPXX0IBEN      15644
ML0122EN       15603
excourse04     15062
excourse06     15062
GPXX0TY1EN     14689
excourse72     14464
excourse73     14464
Name: count, dtype: int64
```

Flowchart of content-based recommender system using course similarity



The implementation of the course similarity-based recommender system primarily focuses on recommending new courses that are similar to a user's already enrolled courses. This process leverages a pre-computed course similarity matrix.
Here's an overview of the implementation steps:

- **Setup & Data Loading:** Essential libraries like numpy and pandas are installed and imported, followed by loading the pre-made course similarity matrix (sim_df), course content (course_df), and Bag of Words (bow_df) datasets.

- **Course ID-Index Mappings:** Dictionaries are created to map course IDs to numerical indices and vice versa (id_idx_dict, idx_id_dict) using the doc_index and doc_id columns from bow_df for efficient lookup in the similarity matrix.

- **Similarity Metric:** The sim_df is a symmetric matrix with values from 0 to 1, representing the similarity between any two courses, based on features like course genres and Bag of Words content.

- **Individual User Recommendation:** For a specific user, the system identifies their enrolled_course_ids and all unselected_course_ids. It then iterates through each unselected course, calculates its similarity to the user's enrolled courses using the sim_matrix and the ID-index mappings, and recommends it if the similarity exceeds a defined threshold (e.g., 0.5 or 0.6). The highest similarity score is retained for each recommended course.

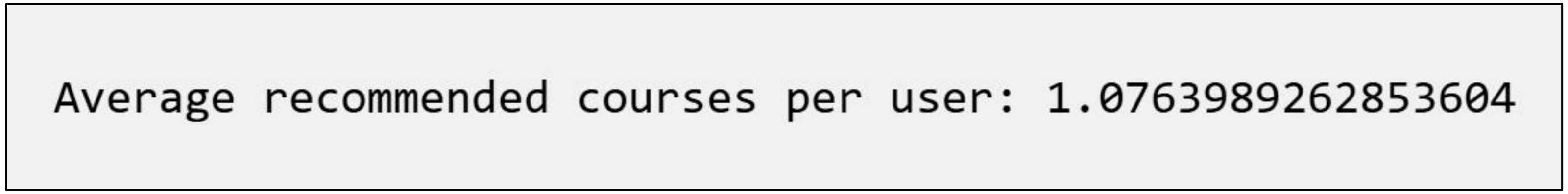
- **Batch Recommendation:** This individual recommendation logic is then applied across all test users to generate a comprehensive list of recommendations.

- **Analysis:** Finally, the aggregated recommendations are analyzed to determine metrics such as the average number of new courses recommended per user and the top-10 most frequently recommended courses across all users.

Evaluation results of course similarity based recommender system

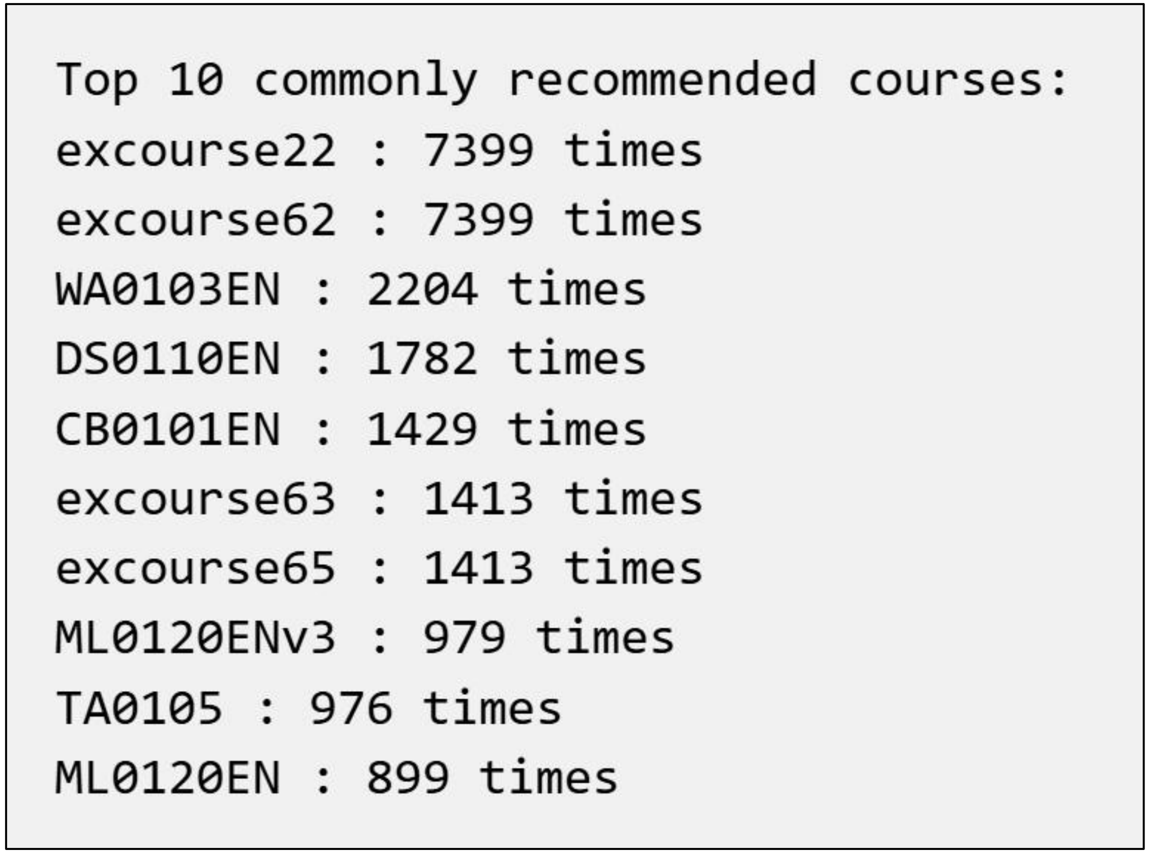
Hyper-Parameter Settings :- **similarity threshold = 0.6**

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



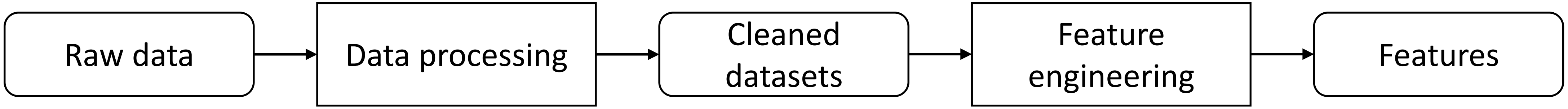
```
Average recommended courses per user: 1.0763989262853604
```

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



```
Top 10 commonly recommended courses:
excourse22 : 7399 times
excourse62 : 7399 times
WA0103EN : 2204 times
DS0110EN : 1782 times
CB0101EN : 1429 times
excourse63 : 1413 times
excourse65 : 1413 times
ML0120ENv3 : 979 times
TA0105 : 976 times
ML0120EN : 899 times
```

Flowchart of clustering-based recommender system



The user profile clustering-based recommender system's main goal is to **group users with similar learning interests** to provide course recommendations. The process involves:

- **Initial User Profile Data:** The system begins with user profile vectors, derived from course ratings and genres, detailing a user's interest across 14 course genres (e.g., 'Database', 'Python'). There are approximately 33,901 unique users.

- **Data Standardization:** User profile features are standardized using StandardScaler to have a mean of 0 and a standard deviation of 1, which is important for clustering and PCA.

- **User Clustering:** Users are grouped into "learning communities" using K-means clustering. Two methods are applied:

    ◦ **K-means on Original Features:** K-means is run directly on the standardized features. The optimal number of clusters (n_clusters) is determined using the Elbow Method to find the point of minimal sum of squared distances.

    ◦ **K-means on PCA-Transformed Features:** Principal Component Analysis (PCA) is first used to reduce the dimensionality of the features, as some are correlated (e.g., 'MachineLearning' and 'DataScience'). The optimal number of components (n_components) is found by selecting components that explain a high percentage (e.g., >= 90%) of the original data's variance. K-means is then applied to these reduced features.

- **User Cluster Assignment:** After clustering, each user is assigned a cluster label, linking them to a specific "learning community".

- **Recommendation Generation:** The system recommends courses based on the principle that a user will be interested in courses popular among their cluster members. For a test user, the algorithm identifies popular courses within their assigned cluster and then recommends unseen courses (those popular in the cluster but not yet completed by the user).

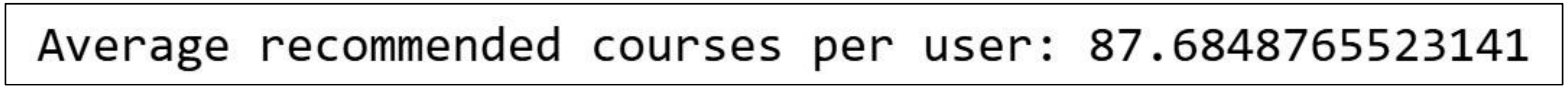
Evaluation results of clustering-based recommender system

Hyper-Parameter Settings :-

enrollment_threshold = 10

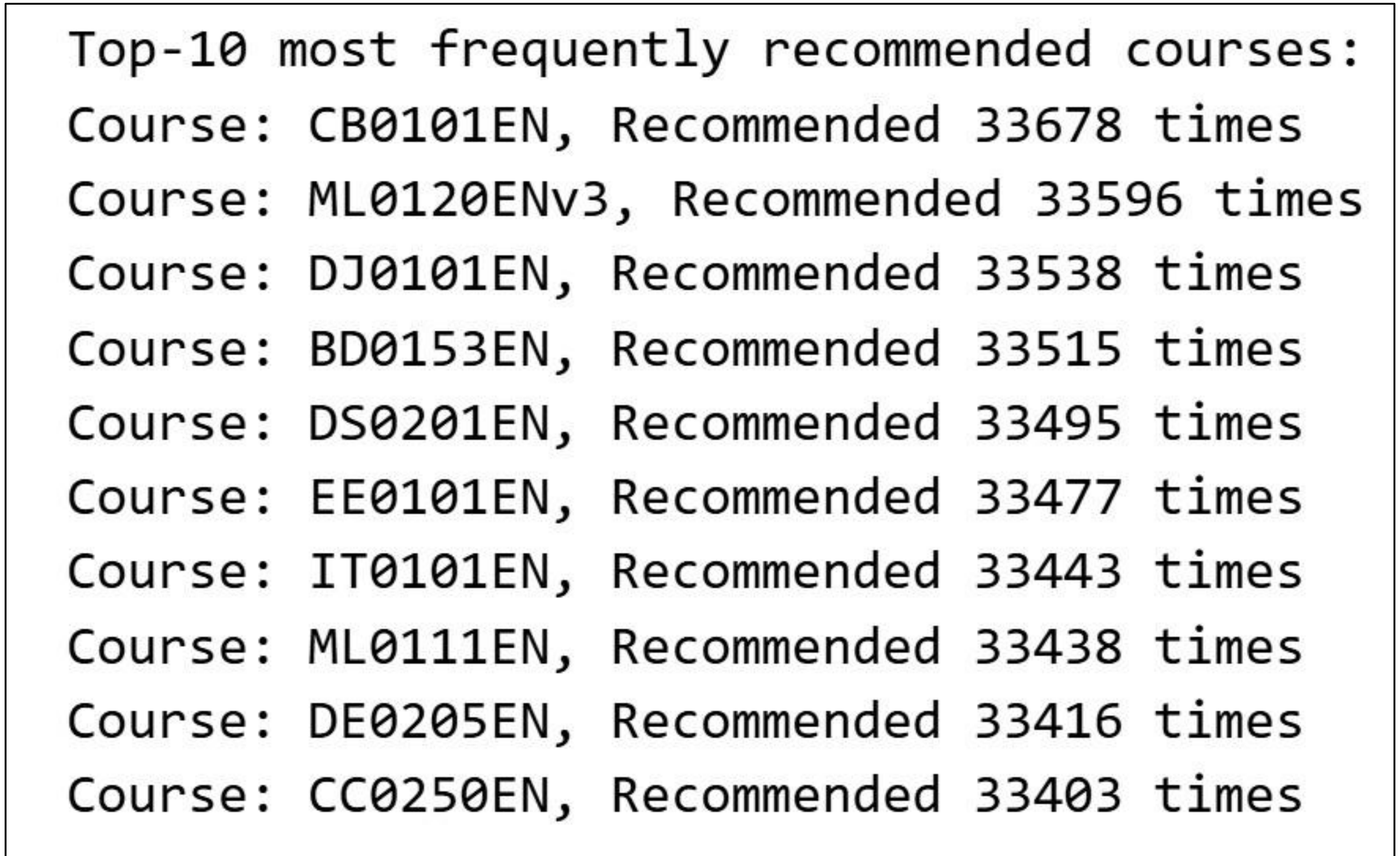
Number of Clusters (n_clusters) = 5

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



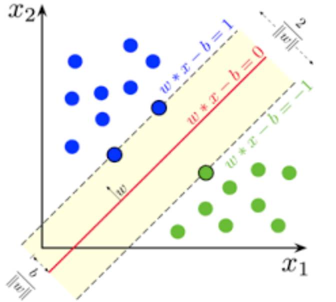
```
Average recommended courses per user: 87.6848765523141
```

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

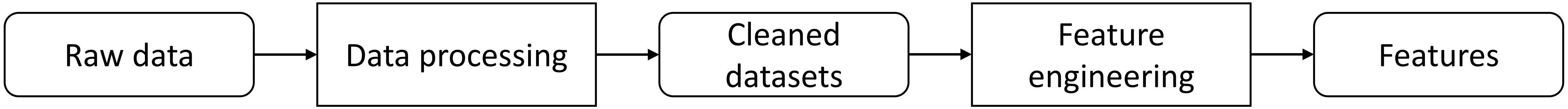


```
Top-10 most frequently recommended courses:
Course: CB0101EN, Recommended 33678 times
Course: ML0120ENv3, Recommended 33596 times
Course: DJ0101EN, Recommended 33538 times
Course: BD0153EN, Recommended 33515 times
Course: DS0201EN, Recommended 33495 times
Course: EE0101EN, Recommended 33477 times
Course: IT0101EN, Recommended 33443 times
Course: ML0111EN, Recommended 33438 times
Course: DE0205EN, Recommended 33416 times
Course: CC0250EN, Recommended 33403 times
```

Collaborative-filtering Recommender System using Supervised Learning



Flowchart of KNN based recommender system



Flowchart :-

Raw Course Enrollment Data → Data Preparation (User-Item Interaction Matrix) → Data Splitting (Train/Test) → KNN Model Definition → Model Training → Rating Prediction → Evaluation (RMSE)

The process begins with **Raw Course Enrollment Data**. This data typically includes user ID, item ID (course), and rating (enrollment mode).

1. Data Preparation (User-Item Interaction Matrix): The raw data is transformed into a sparse user-item interaction matrix. In this 2-D matrix, rows represent user IDs, columns represent item IDs, and the value at (i, j) indicates user i's rating for item j. This format allows for direct computations like cosine similarity.

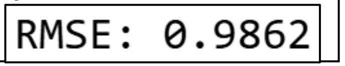
2. Data Splitting (Train/Test): The prepared dataset is divided into a training set and a test set. The training set builds the model, while the test set evaluates its performance.

3. KNN Model Definition: A KNN-based collaborative filtering model is defined. This involves choosing a similarity measurement (e.g., 'cosine') and specifying if it's user-based or item-based. For course recommendations, an item-based approach (user_based=False) is often used, finding similar courses.

4. Model Training: The defined KNN model is fitted on the training set, computing the similarity matrix between users or items based on their interaction history.

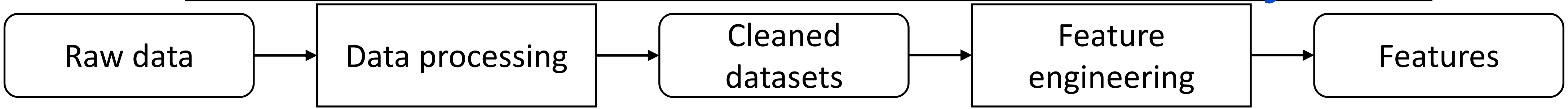
5. Rating Prediction: The trained model then predicts unknown ratings for user-item pairs in the test set. This is done by finding k nearest neighbors (similar users or items) and using their known ratings, weighted by similarity, to estimate the target rating.

6. Evaluation (RMSE): Finally, the system's performance is assessed using metrics like Root Mean Squared Error (RMSE). RMSE quantifies the difference between predicted and true ratings, with a lower RMSE indicating higher accuracy and confidence in recommendations.



```
RMSE: 0.9862
```

Flowchart of NMF based recommender system



Flowchart:-
Raw User-Item Interaction Data → Data Preprocessing: Convert to Sparse User-Item Matrix → Data Splitting: Train-Test Split → Model Definition: Instantiate NMF Model → Model Training: Fit NMF on Trainset (Matrix Decomposition) → Prediction: Estimate Ratings for Testset → Model Evaluation: Compute RMSE

• **Raw User-Item Interaction Data:** The process starts with the initial dataset containing user IDs, item IDs, and ratings, presented in a "dense or vertical form".

• **Data Preprocessing: Convert to Sparse User-Item Matrix**: This raw data is transformed into a sparse user-item interaction matrix using a pivot operation, which is the necessary input for NMF.

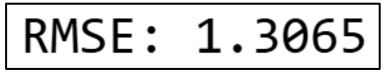
• **Data Splitting: Train-Test Split**: The prepared dataset is then split into trainset and testset to enable model training and subsequent evaluation on unseen data.

• **Model Definition: Instantiate NMF Model**: An NMF (Non-negative Matrix Factorization) model is defined by instantiating the NMF() class, which is a dimensionality reduction algorithm that can decompose large sparse matrices.

• **Model Training: Fit NMF on Trainset (Matrix Decomposition)**: The NMF model is trained on the trainset, decomposing the original sparse user-item matrix into two smaller, dense matrices: one for "transformed user features" (U) and another for "transformed item features" (I).

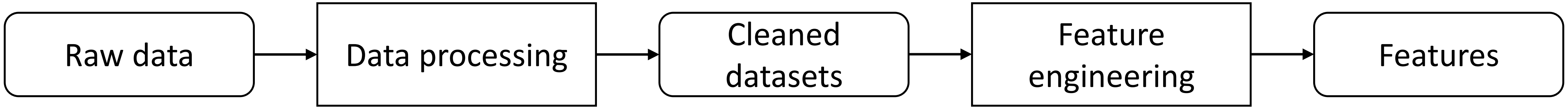
• **Prediction: Estimate Ratings for Testset**: Once U and I are learned, ratings for unknown items in the testset are estimated by performing the dot product of a user's latent feature vector (from U) and an item's latent feature vector (from I).

• **Model Evaluation: Compute RMSE**: Finally, the system's performance is evaluated by computing the Root Mean Squared Error (RMSE) between predicted and actual ratings from the testset.



```
RMSE: 1.3065
```

Flowchart of Neural Network Embedding based recommender system



Raw data → Data processing → Cleaned datasets → Feature engineering → Features

Flowchart :-

Raw Rating Data → Data Preprocessing & Splitting → Neural Network Model Definition (RecommenderNet) → Model Training & Evaluation → Latent Feature Extraction

• **Raw Data Input**: The process begins with raw rating data containing user IDs, item IDs, and their corresponding ratings.

• **Data Preprocessing**: This data is then preprocessed by converting the original user and item IDs into integer indices, which are suitable for neural network input. Ratings can also be scaled. The dataset is then split into training, validation, and test sets.

• **Neural Network Model**: A RecommenderNet model is defined, utilizing user and item embedding layers. These embedding layers transform one-hot encoded user and item vectors into smaller, dense "embedding feature vectors" in a latent space. The model predicts a rating by computing the dot product of these user and item embedding vectors, along with learned biases, and applies an activation function like ReLU.

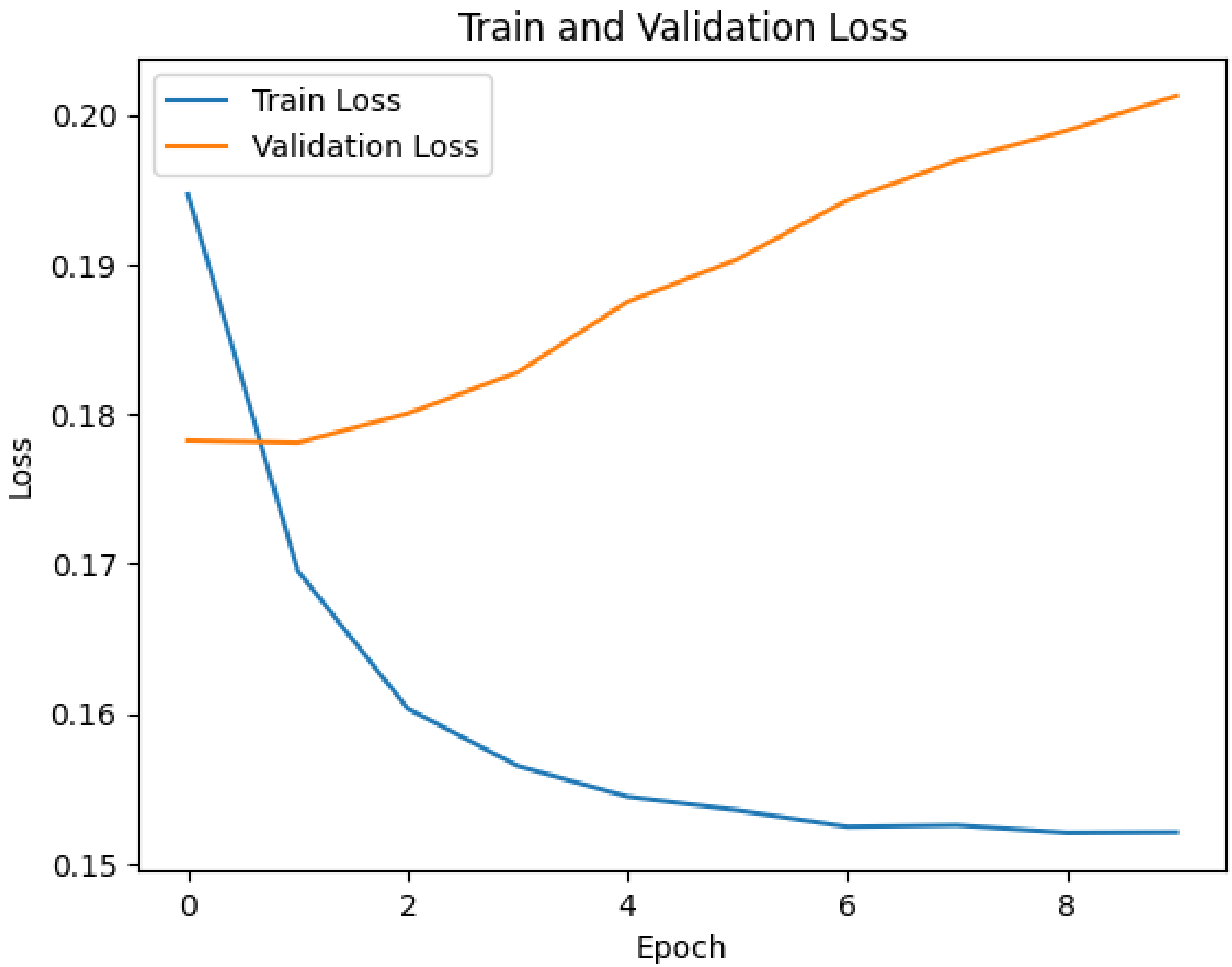
• **Model Training**: The model is compiled with a loss function (e.g., Mean Squared Error) and an optimizer (e.g., Adam), then trained using the prepared datasets. During training, the embedding layers simultaneously learn and store the latent features of users and items.

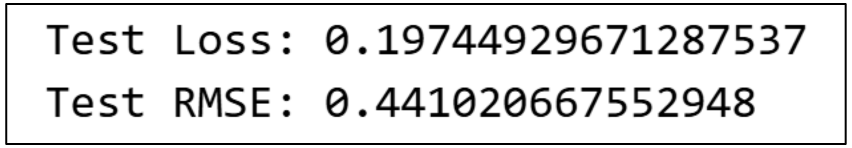
• **Evaluation & Embedding Extraction**: After training, the model's performance is evaluated using metrics like Root Mean Squared Error (RMSE) on the test set. The core advantage is that the user and item latent feature vectors can then be directly extracted from the weights of the trained embedding layers. These latent features can be further used for recommendations.

• **Customization**: The model can be customized by adding more hidden layers or tuning hyperparameters like embedding size for potential performance improvements.

Neural Network Embedding based recommender system

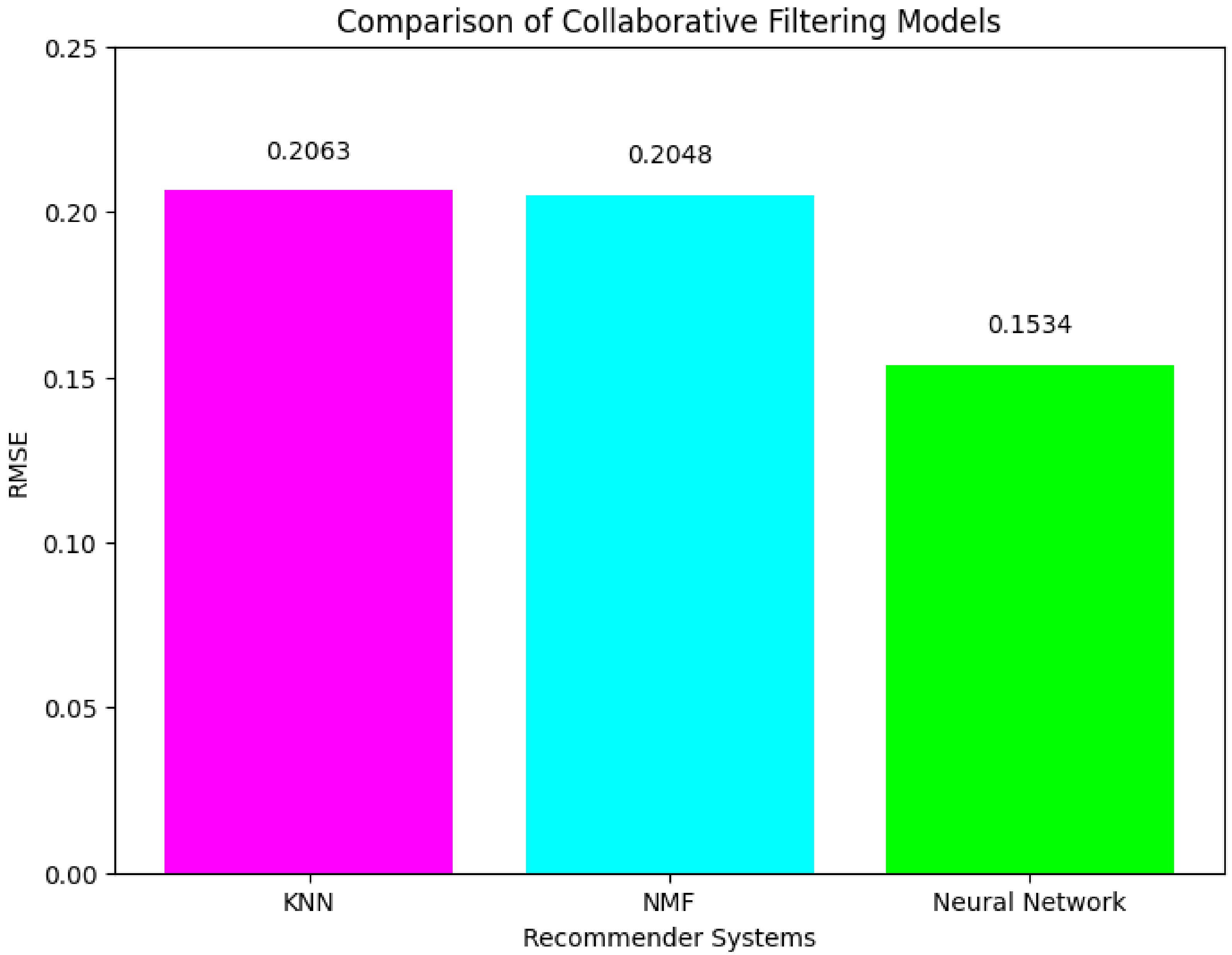
Plotting the **Train and Validation Loss** of the RecommenderNet() model :-





```
Test Loss: 0.19744929671287537
Test RMSE: 0.44102066755294 8
```

Compare the performance of collaborative-filtering models



Compare the performance of collaborative-filtering models

The bar chart in the previous slide, titled "Comparison of Collaborative Filtering Models", visualizes the **Root Mean Squared Error (RMSE)** for three collaborative filtering models: **KNN**, **NMF**, and **Neural Network**.

- **RMSE values**:
  - **KNN**: 0.2063
  - **NMF**: 0.2048
  - **Neural Network**: 0.1534
- **A lower RMSE indicates better model performance**, as it signifies less error in predictions.
- The **Neural Network model demonstrates the best performance** with the lowest RMSE of 0.1534. This is significantly lower than both KNN and NMF, suggesting its superior accuracy in predicting course ratings. Neural networks are effective because they are "very good at learning patterns from data" and can "extract latent features" within their hidden layers as weight matrices during training.

Conclusions

- A central achievement of the project was the development and implementation of diverse recommender systems, employing various methodologies. These included content-based methods which leveraged user profiles and course similarity through techniques like Bag of Words and cosine similarity, clustering-based algorithms utilizing K-means clustering and Principal Component Analysis (PCA) to group users with similar learning interests, and collaborative filtering techniques such as KNN-based collaborative filtering, Non-negative Matrix Factorization (NMF), and neural network embedding-based systems for predicting ratings.

- Through comprehensive Exploratory Data Analysis (EDA), the project provided significant insights into the online course dataset. Key findings included identifying the popularity of different course genres, with BackendDev being the most popular (78 courses), followed by MachineLearning (69 courses), and Database (60 courses). The EDA also revealed user engagement patterns through enrollment distributions, with the average user rating approximately 6.88 courses, pinpointed the most popular courses like "python for data science" (14936 ratings) and "introduction to data science" (14477 ratings), and highlighted prominent IT-related keywords such as "python, data science, machine learning, big data, ai, tensorflow" from course titles using a word cloud.

- The project effectively tackled the prediction of course ratings and learner engagement using neural networks, regression, and classification models. A key outcome of the comparison of collaborative filtering models (KNN, NMF, and Neural Network) was that the Neural Network model demonstrated the best performance with the lowest Root Mean Squared Error (RMSE) of 0.1534. This significantly lower RMSE, compared to KNN (0.2063) and NMF (0.2048), indicates the neural network's superior accuracy in predicting course ratings due to its effectiveness in learning complex patterns and extracting latent features within its hidden layers.

- Ultimately, the project successfully achieved its main objectives, which encompassed comparing and contrasting various machine learning algorithms for recommender systems, predicting course ratings using different model types, and applying a range of techniques including KNN, PCA, Non-negative Matrix collaborative filtering, exploratory data analysis, Bag of Words, and cosine similarity. This collectively demonstrates a robust ability to develop and implement effective machine learning solutions for online course recommendations and predictive tasks.