Department of Computer Science and Engineering (Data Science)

Subject: Recommender System Laboratory (DJS22DSL6012)

(A.Y. 2024-2025)

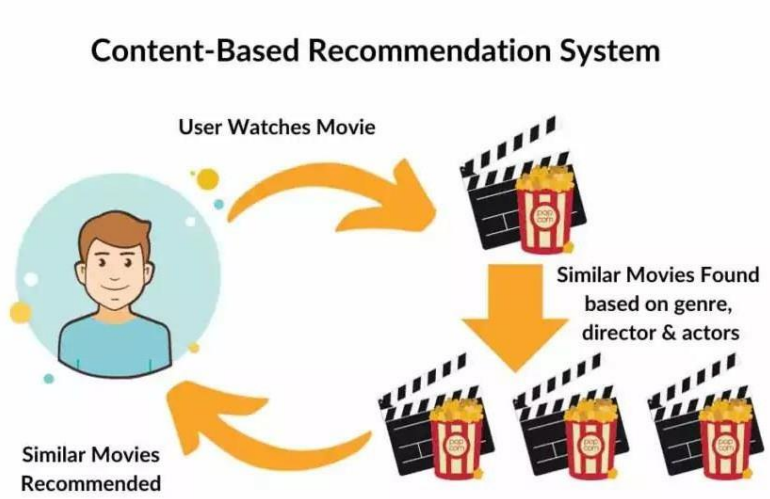
Experiment 2

Name : Soham Walam **Roll No : D 102**

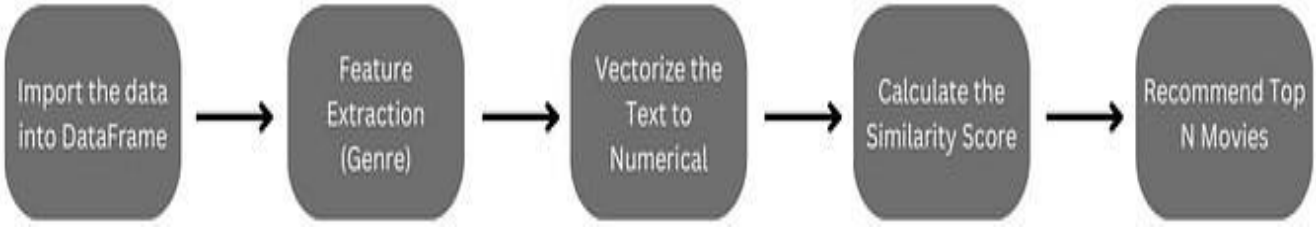
Aim: Implement Content based Recommender System on an appropriate dataset.

Theory:

Content-based recommender systems are a subset of recommender systems that tailor recommendations to users by analyzing items' intrinsic characteristics and attributes. These systems focus on understanding the content of items and mapping it to users' preferences. By examining features such as genre, keywords, metadata, and other descriptive elements, content- based recommender systems create profiles for both users and items.



This enables the system to make recommendations matching user preferences with items with similar content traits. Content-based systems operate independently, which makes them particularly useful in scenarios where user history is limited or unavailable. Through this personalized approach, content-based recommender systems play a vital role in enhancing user experiences across various domains, from suggesting movies and articles to guiding users in choosing products or destinations.



The concepts of Term Frequency (TF) and Inverse Document Frequency (IDF) play a crucial role in information retrieval systems and content-based filtering mechanisms, such as content

The **IMDb formula** to calculate the **weighted rating (WR)** of a movie is:

$$WR = \left(\frac{v}{v+m}\right) \cdot R + \left(\frac{m}{v+m}\right) \cdot C$$

Where:

- **WR** = Weighted Rating
- **R** = Average rating for the movie (mean) across all users
- **v** = Number of votes for the movie
- **m** = Minimum number of votes required to be listed in the chart (threshold)
- **C** = Mean vote across the whole dataset (e.g., average rating of all movies)

Example:

- **Movie A:**
  - `v = 20,000` votes, `R = 8.5` average rating
- **Movie B:**
  - `v = 300` votes, `R = 9.0` average rating
- **Global Average (`C`) = 6.8, Minimum Votes (`m`) = 15,000**

Weighted Rating for A:

$$\left(\frac{20000}{20000+15000} \times 8.5\right) + \left(\frac{15000}{20000+15000} \times 6.8\right) \approx 7.85$$

Weighted Rating for B:

$$\left(\frac{300}{300+15000} \times 9.0\right) + \left(\frac{15000}{300+15000} \times 6.8\right) \approx 6.82$$

Although Movie B has a higher raw rating (9.0), Movie A ranks higher due to its larger vote count, making it more statistically reliable.

based recommenders. These concepts help determine the relative importance of a document, article, news item, movie, etc.

All TF means is how often a given word occurs in a given document so within one web page one Wikipedia article, how common is a given word within that document , what is the ratio of that word occurrence rate throughout all the words in that document that's it. TF just measures how often a word occurs in a document. A word that occurs frequently is probably important to that document's meaning.

DF is how often a word occurs in an entire set of documents, i.e., all of Wikipedia or every web page. this tells us about common words that just appears everywhere no matter what the topic, like 'a', 'the', 'and', etc. Word with high TF and DF both might not be important measure relevancy of a word to a document.

The cold-start problem essentially consists of how a system handles new users or new items. Both pose a problem in collaborative filtering because it recommends items by grouping users according to inferred similarities of behavior and preference. New users do not have an evidenced similarity with others, however, and new items do not have enough user interaction (for example, ratings) for recommending them. While content-based filtering struggles with new users, it nevertheless adeptly handles incorporating new items. This is because it recommends items based on internal or metadata characteristics rather than past user interaction.

Content-based filtering enables greater degree of transparency by providing interpretable features that explain recommendations. For example, a movie recommendation system may explain why a certain movie is recommended, such as genre or actor overlap with previously watched movies. The user may therefore make a more informed decision on whether to watch the recommended movie.

One chief disadvantage of content-based filtering is feature limitation. Content-based recommendations are derived exclusively from the features used to describe items. A system's item features may not be able to capture what a user likes however. For instance, returning to the movie recommendation system example, assume a user watches and likes the 1944 movie Gaslight. A CBRS may recommend other movies directed by George Cukor or starring Ingrid Bergman, but those movies may not be similar to Gaslight. If the user rather relishes some specific plot device (for example, deceptive husband) or production element (for example, cinematographer) not represented in the item profile, the system will not present suitable recommendations. Accurate differentiation between a user's potential likes and dislikes cannot be

accomplished with insufficient data.

Because content-based filtering only recommends items based on a user's previously evidenced interests, its recommendations are often similar to items a user liked in the past. In other words, CBRSs lack a methodology for exploring the new and unpredicted. This is overspecialization. In contrast, because collaborative-based methods draw recommendations from a pool of users who have similar likes to one given user, they can often recommend items that a user may have not considered, appears with different features than a user's previously liked items but that retain some unrepresented element that appeals to a user type.

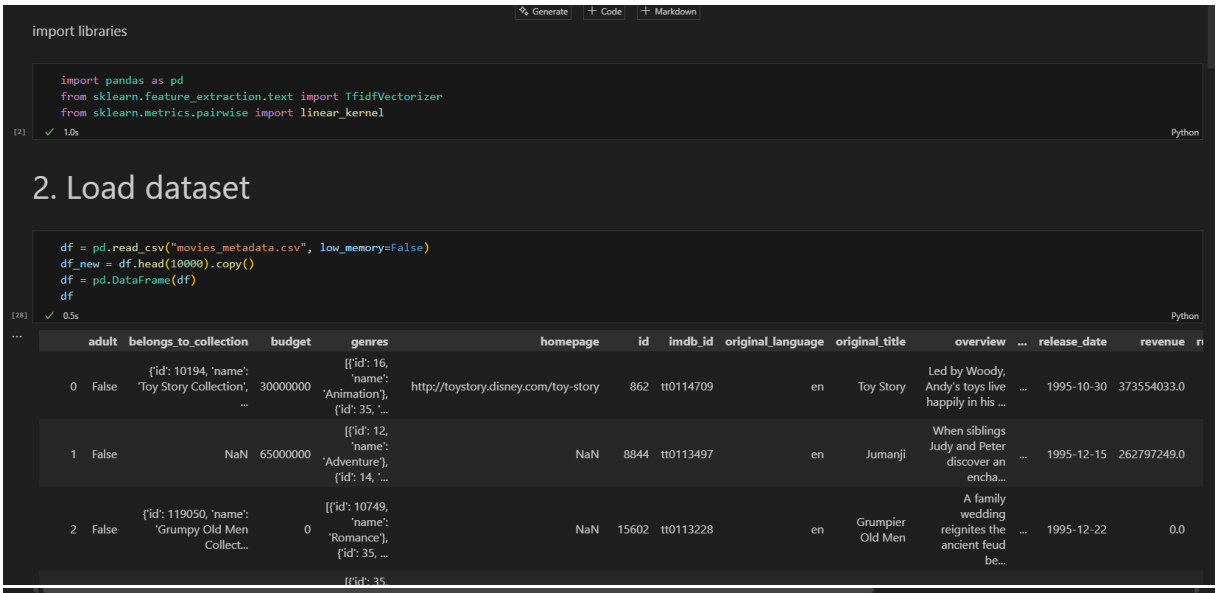
Department of Computer Science and Engineering (Data Science)

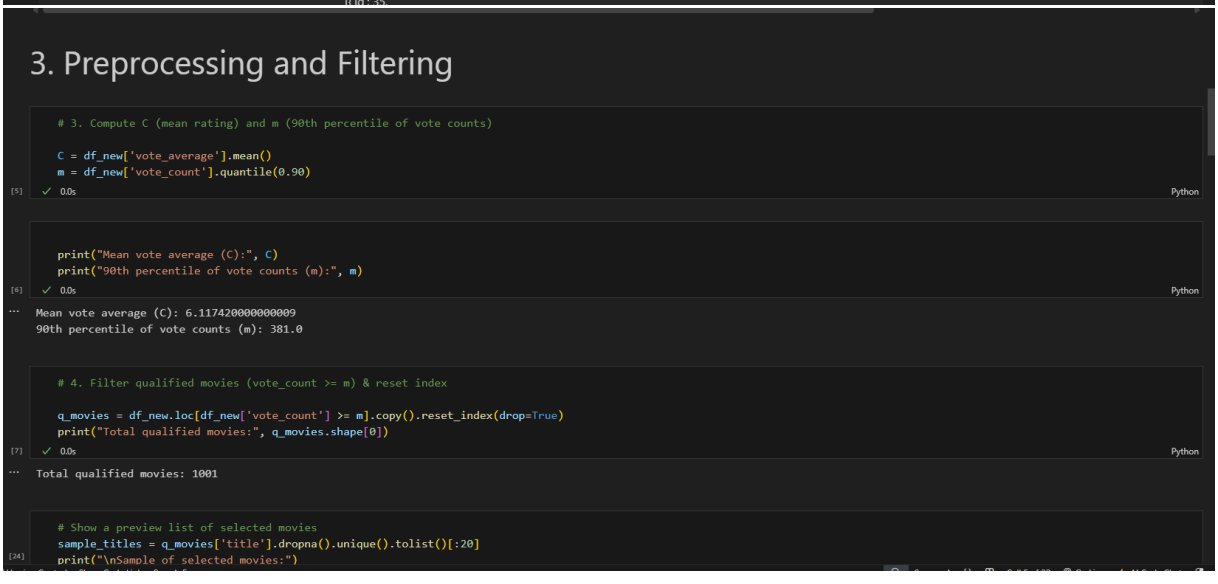
Dataset Link:
https://drive.google.com/drive/folders/1mFpbV1SmZT57bKvWUY-FzP3YSuM1pFJd?usp=drive_link

Lab Assignments to complete:

Perform the following tasks on the **movies_metadata.csv** dataset:

1. Calculate mean of vote average column.
2. Calculate the minimum number of votes required to be in the chart.
3. Compute the weighted rating (score) of each movie.
4. Apply TF-IDF method for fitting and transforming.
5. Compute the cosine similarity matrix.
6. Recommend appropriate movies based on similarity scores.





Department of Computer Science and Engineering (Data Science)

```python
# Show a preview list of selected movies
sample_titles = q_movies['title'].dropna().unique().tolist()[:20]
print("\nSample of selected movies:")
print(sample_titles)
```

```
Sample of selected movies:
['Toy Story', 'Jumanji', 'Heat', 'GoldenEye', 'Balto', 'Casino', 'Four Rooms', 'Ace Ventura: When Nature Calls', 'Assassins', 'Twelve Monkeys', 'Babe', 'Clueless', 'Mortal Kombat',
```

```python
# 5. Weighted rating function

def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    return (v/(v+m) * R) + (m/(m+v) * C)

# Add score column
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

```python
# ------------------------------
# 6. Add 'score' column using weighted rating
# ------------------------------
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```

```python
# 6. Add 'score' column using weighted rating
# ------------------------------
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)

# ------------------------------
# 7. Print Top 20 Movies by Score
# ------------------------------
top20 = q_movies.sort_values('score', ascending=False).head(20)
print("\n Top 20 Movies by Weighted Score:")
print(top20[['title', 'vote_count', 'vote_average', 'score']])
```

```
 Top 20 Movies by Weighted Score:
                                      title  vote_count  vote_average  \
54                     The Shawshank Redemption      8358.0           8.5
129                             The Godfather      6024.0           8.5
492                               Fight Club      9678.0           8.3
50                               Pulp Fiction      8670.0           8.3
87                           Schindler's List      4436.0           8.3
751                            Spirited Away      3968.0           8.3
61                              Forrest Gump      8147.0           8.2
400                          Life Is Beautiful      3643.0           8.3
197                      The Godfather: Part II      3418.0           8.3
178                     The Empire Strikes Back      5998.0           8.2
177              One Flew Over the Cuckoo's Nest      3001.0           8.3
49                     Leon: The Professional      4293.0           8.2
516                           The Green Mile      4166.0           8.2
867  The Lord of the Rings: The Return of the King      8226.0           8.1
195                                    Psycho      2405.0           8.3
45                                  Star Wars      6778.0           8.1
13                                    Se7en      5915.0           8.1
192                                GoodFellas      3211.0           8.2
401                        American History X      3120.0           8.2
99                     The Silence of the Lambs      4549.0           8.1

        score
...
13   7.980025
192  7.979103
```

```python
# Print overviews of first 5 qualified movies
print("\nOverviews of first 5 qualified movies:")
print(q_movies['overview'].head())
```

```
Overviews of first 5 qualified movies:
0    Led by Woody, Andy's toys live happily in his ...
1    When siblings Judy and Peter discover an encha...
2    Obsessive master thief, Neil McCauley leads a ...
3    James Bond must unmask the mysterious head of ...
4    An outcast half-wolf risks his life to prevent...
Name: overview, dtype: object
```

```python
# 6. Recommend Top Movies by Score

def recommend_by_score(n=10):
    return q_movies.sort_values('score', ascending=False).head(n)[['title', 'vote_count', 'vote_average', 'score']]

print("\n Top 10 Movies by Weighted Score:")
print(recommend_by_score(10))
```

```
 Top 10 Movies by Weighted Score:
                      title  vote_count  vote_average     score
54   The Shawshank Redemption      8358.0           8.5  8.396125
129             The Godfather      6024.0           8.5  8.358273
492               Fight Club      9678.0           8.3  8.217331
50               Pulp Fiction      8670.0           8.3  8.208125
87           Schindler's List      4436.0           8.3  8.127369
751            Spirited Away      3968.0           8.3  8.108792
61              Forrest Gump      8147.0           8.2  8.106958
400          Life Is Beautiful      3643.0           8.3  8.093349
197    The Godfather: Part II      3418.0           8.3  8.081110
178   The Empire Strikes Back      5998.0           8.2  8.075613
```

Department of Computer Science and Engineering (Data Science)

working with TF-IDF

```python
# 7. TF-IDF on overviews (only qualified movies)

q_movies['overview'] = q_movies['overview'].fillna('')
```

[13] ✓ 0.0s Python

```python
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(q_movies['overview'])

print("\nTF-IDF shape:", tfidf_matrix.shape)   # (1001, ~9000)
```

[14] ✓ 0.0s Python

```
TF-IDF shape: (1001, 9013)
```

```python
# 8. Compute Cosine Similarity

cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
print("Cosine similarity matrix shape:", cosine_sim.shape)   # (1001, 1001)
```

[15] ✓ 0.0s Python

```
Cosine similarity matrix shape: (1001, 1001)
```

```python
# 9. Reverse lookup map: movie title -> index

q_movies['title_clean'] = q_movies['title'].str.strip().str.lower()
indices = pd.Series(q_movies.index, index=q_movies['title_clean']).drop_duplicates()
```

[16] ✓ 0.0s Python

```python
# 10. Content-based Recommender Function
```

[17]

```python
# 10. Content-based Recommender Function

def get_recommendations(title, cosine_sim=cosine_sim, top_n=10):
    title_clean = title.strip().lower()

    if title_clean not in indices:
        return f"❌ '{title}' not found in qualified movies."

    idx = indices[title_clean]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)[1:top_n+1]
    movie_indices = [i[0] for i in sim_scores]

    results = q_movies.iloc[movie_indices][['title', 'score']].reset_index(drop=True)
    return results
```

[17] ✓ 0.0s Python

```python
# 11. Test Both Modes

print("\n🎬 Top 10 Movies by Weighted Score:")
print(recommend_by_score(10))

movie_to_test = "Toy Story"
print(f"\n🎬 Content-based Recommendations for '{movie_to_test}':")
print(get_recommendations(movie_to_test, top_n=10))
```

[18] ✓ 0.0s Python

```
🎬 Top 10 Movies by Weighted Score:
                        title  vote_count  vote_average     score
54   The Shawshank Redemption      8358.0           8.5  8.396125
129             The Godfather      6024.0           8.5  8.358273
492               Fight Club      9678.0           8.3  8.217331
50               Pulp Fiction      8670.0           8.3  8.208125
87           Schindler's List      4436.0           8.3  8.127369
```

```
# 11. Test Both Modes

print("\n🎬 Top 10 Movies by Weighted Score:")
print(recommend_by_score(10))

movie_to_test = "Toy Story"
print(f"\n🎬 Content-based Recommendations for '{movie_to_test}':")
print(get_recommendations(movie_to_test, top_n=10))
```

[18] ✓ 0.0s Python

```
🎬 Top 10 Movies by Weighted Score:
                        title  vote_count  vote_average     score
54   The Shawshank Redemption      8358.0           8.5  8.396125
129             The Godfather      6024.0           8.5  8.358273
492               Fight Club      9678.0           8.3  8.217331
50               Pulp Fiction      8670.0           8.3  8.208125
87           Schindler's List      4436.0           8.3  8.127369
751             Spirited Away      3968.0           8.3  8.108792
61               Forrest Gump      8147.0           8.2  8.106958
400         Life Is Beautiful      3643.0           8.3  8.093349
197    The Godfather: Part II      3418.0           8.3  8.081110
178    The Empire Strikes Back      5998.0           8.2  8.075613

🎬 Content-based Recommendations for 'Toy Story':
                                  title     score
0                           Toy Story 2  7.195096
1                        Man on the Moon  6.538590
2                The Shawshank Redemption  8.396125
3                      Maid in Manhattan  5.825557
4  Harry Potter and the Philosopher's Stone  7.430405
5                         Monsters, Inc.  7.419344
6                      The Thin Red Line  6.848669
7                           Pretty Woman  6.846315
8                             Four Rooms  6.341562
9                          Small Soldiers  6.165157
```

Theory Questions:

1. What is the purpose of using TF-IDF in a content-based recommender system?

TF-IDF (Term Frequency–Inverse Document Frequency) is used to transform movie descriptions into numerical representations that reflect how significant each word is within that description. Simple word counts would give too much weight to common words like *"the"* or *"man"* that appear in nearly all movies, making comparisons meaningless. TF-IDF corrects this by lowering the importance of such frequent words while giving higher importance to rare, descriptive terms such as *"prison," "redemption,"* or *"galaxy."* This allows the recommender system to highlight the unique themes of each movie and make more accurate content-based matches.

2. Why is cosine similarity used for comparing movie descriptions?

Cosine similarity is preferred because it measures the similarity between two text vectors by considering the angle between them, not their absolute size. This is crucial since movie overviews vary in length — some are just one line while others may be a full paragraph. Using cosine similarity ensures that two movies with similar themes and vocabulary are judged close, regardless of how many total words were used. This makes it especially effective with TF-IDF vectors, as it captures how closely two movies align in terms of descriptive content rather than sheer word frequency.

3. What does filtering movies by a vote count threshold (like 90th percentile) help achieve?

Applying a vote count threshold, such as considering only movies above the 90th percentile of votes, ensures that the recommendations are both meaningful and reliable. Without this, a little-known film with only a handful of votes could appear as a top-rated movie due to statistical flukes. By filtering for movies with a sufficient number of votes, the system balances quality and popularity, surfacing titles that not only have high average ratings but also a strong level of consensus among viewers. This is why widely appreciated classics like The Shawshank Redemption naturally rise to the top.

Conclusion: Hence, we have successfully implemented Content based Recommender System.