Milestone 2 Report

Project Title: Movie Recommendation Engine

Course: Introduction to Data Science

Student: Ashfaq Ahmed Mohammed (UFID: 86835927)

GitHub: GitHub Repository

Section 1: Objective, Tools, and Datasets Used

1.1 Project Objective

This project aims to develop a Movie Recommendation Engine that suggests movies to users based on their preferences. The recommendation system will leverage user ratings, movie metadata, and other relevant attributes to generate personalized recommendations. As part of Milestone 1, I have performed Exploratory Data Analysis on my data to gain insights and gather relevant information to solve the problem. Additionally, I have handled null values and dealt with outliers. Now, the Milestone 2 specifically focuses on the feature encoding, Feature Engineering, and Model Training aspects of the project

1.2 Tools and Libraries Used

Data Handling and Processing

• pandas – for handling datasets (loading, cleaning, and manipulating data).
• numpy – for numerical operations and array processing.

Data Visualization

• matplotlib – for plotting graphs and charts.
• seaborn – for statistical data visualization.

Machine Learning and Data Preprocessing

• scikit-learn – for building and evaluating recommendation models.
• MinMaxScaler (from sklearn.preprocessing
• Sklearn.cluster DBSCAN, KMeans
• Sklearn.Mixture - GaussianMixture
• Sklearn.feature – TfidfVectorizer
• Sklearn.decomposition – TruncatedSVD
• Sklearn – Cosine_similarity
• Sklearn – PCA
• Fasttext

File Handling and Utilities

• os – for interacting with the file system and handling dataset directories.

1.3 Original Datasets Used

The project utilizes multiple datasets for building the recommendation engine:

Dataset Name	File Name	Description
The Movies Dataset (The Movies Dataset)	movies_metadata.csv ratings.csv links.csv credits.csv	Contains detailed information about movies such as title, genres, budget, revenue, runtime, and popularity. user ratings for movies. Links movies across different platforms such as IMDb and TMDb for cross-referencing Consists of Cast and Crew Information of the movies.

Streaming Platforms (Streaming Platforms)	MoviesOnStreamingPlatforms.csv	Contains information on whether a movie is available on Netflix, Hulu, Prime Video, or Disney+.
Top 1000 Highest grossing Movies (Highest Grossers per Year)	Top_1000_Highest_Grossing_Movies_Of_All_Time.csv	Contains information on the highest-grossing movies of all time.

1.4 Dataset Dimensions

CSV File	Rows (Approx.)	Columns	Description
movies_metadata.csv	45000	24	Contains metadata about movies including title, release date, genres, etc.
credits.csv	45000	3	Includes cast and crew details for each movie in JSON format.
ratings.csv	Millions	4	Contains user ratings for movies with user IDs, ratings, and timestamps.
links.csv	45000	3	Maps movie IDs to IMDb and TMDb identifiers.
highest_grossing_movies.csv	1000	6	Contains data on the top 1000 highest-grossing movies of all time, including title, year, and worldwide gross.
movies_on_streaming_platforms.csv	16000	12	Contains information about movies available on Netflix, Prime Video, Hulu, and Disney+, including title, year, and platform availability.

1.4.1 Cleaned and Merged Dataset

After performing EDA on the Original datasets, I handled the missing values handled the outliers of the dataset. After this, I merged the **movies_metadata.csv, credits.csv, ratings.csv, movies_on_stereaming_platforms.csv, and links.csv** files to form a unified data frame that I used in Milestone 2 to engineer new features and train models.

The Google Drive link to the final merged dataset can be found from this link: Dataset_Milestone2

Note: I merged the datasets as part of MileStone1. The relevant code can be found in the Milestone 1 Notebook; However, I'm also attaching the snippets here as well.

```
ratings_summary = ratings_df.groupby("movieId")["rating"].agg(avg_rating="mean", num_reviews="count").
reset_index()
movies_df["imdb_id"] = movies_df["imdb_id"].astype(str).str.replace("tt", "", regex=False)
movies_df["imdb_id"] = pd.to_numeric(movies_df["imdb_id"], errors="coerce")
ratings_summary
```

Figure 1 Code – Calculate the average vote for each movie from the ratings.csv

```
movies = links.merge(movies_df, left_on="imdbId", right_on="imdb_id", how="left")[["movieId", "title"]]
movies_final = ratings_summary.merge(movies, on="movieId", how="left")
movies_final.head()
```

Figure 2 Code - Merging average votes for each movie to movie_metadata.csv

```
credits_df = pd.read_csv("/kaggle/input/the-movies-dataset/credits.csv", low_memory=False)
```

Figure 3 Code - Loading the credits.csv that has data of the actors and Director of each movie.

```
final_df_unmnodified = credits_df.merge(movies_final, on="id", how="left")
```

Figure 4 Code - Adding the Actor and Director data to the movies_metadata.csv data frame

```
ott_df.rename(columns={'Title': 'title'}, inplace=True)

final_df = final_df.merge(
    ott_df[['title', 'Netflix', 'Hulu', 'Prime Video', 'Disney+']],
    on='title',
    how='left'
)
```

Figure 5 Code - Merging the OTT data frame to the original to get the final data frame

1.5 Project Plan

S.No	Task	Deadline
1	Data Collection and Preprocessing	23rd February 2025
2	EDA	23rd February 2025
3	Feature Engineering and Feature Selection	4th April 2025
4	Data Modelling	4th April 2025
5	Evaluation and Testing	15th April 2025
6	User Interface design and Implementation	23th April 2025

Section 2: EDA Insights

2.1 Key EDA Insights

- **Genre Dominance:** The analysis shows that genres like Drama, Comedy, and Thriller are highly prevalent, which might introduce a bias in recommendations if not addressed through balancing techniques.

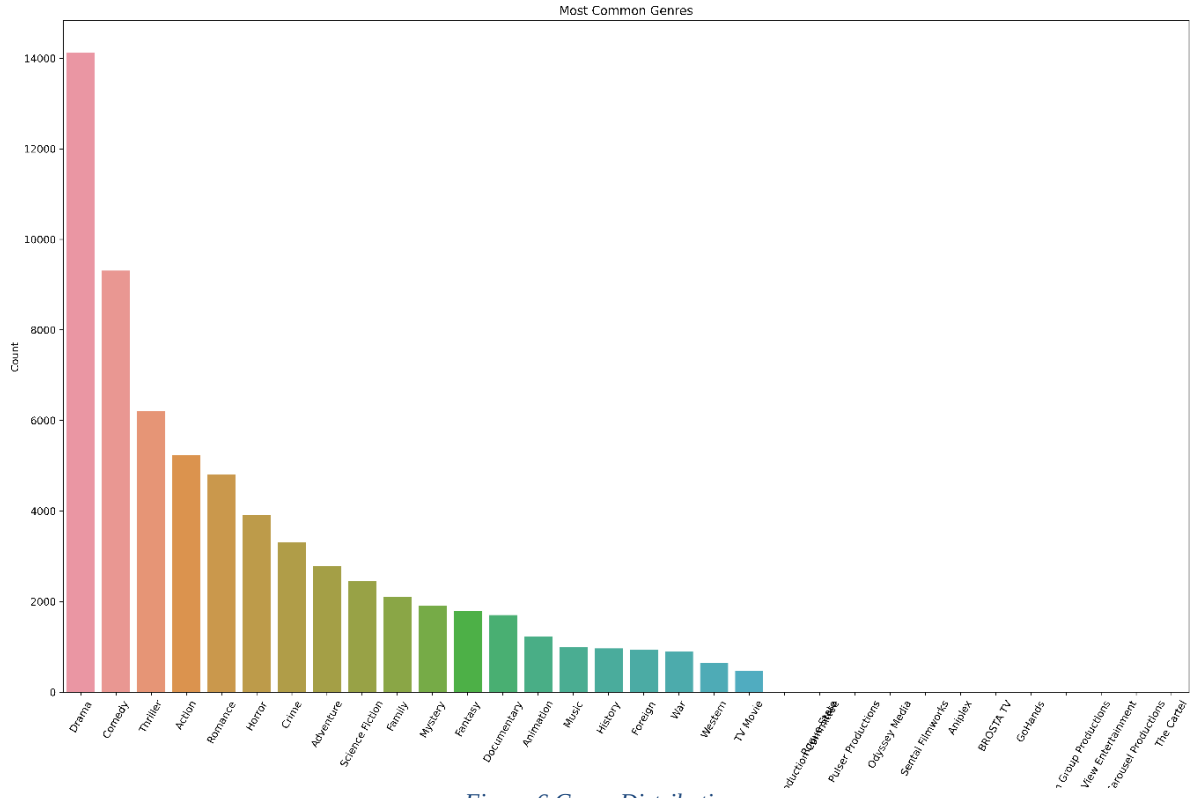

Figure 6 Genre Distribution

- **Budget and Revenue Trends:** There's a strong positive correlation between budget and revenue, indicating that higher-budget movies generally generate more revenue. However, exceptions exist, justifying the use of scaling and log transformations to manage outliers.

- **Movie Age:** The release pattern reveals a significant increase in movie production since the 1990s, peaking around 2018–2019. This suggests that incorporating a movie's age as a feature could help balance between contemporary trends and classic films.

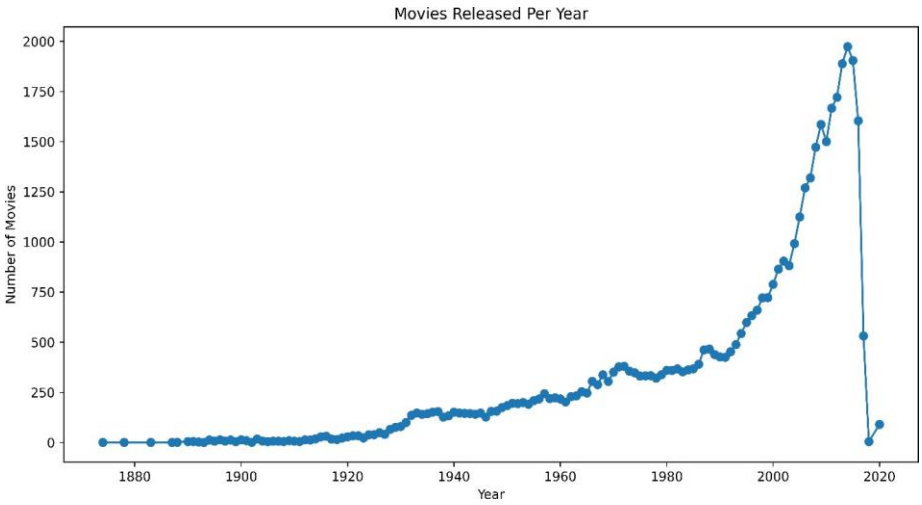

Figure 7 Movies Release per Year

- **Budget vs. Ratings Relationship:** The scatter plots indicate that a higher budget does not necessarily equate to higher movie ratings, emphasizing that audience reception is influenced by factors beyond financial investment.

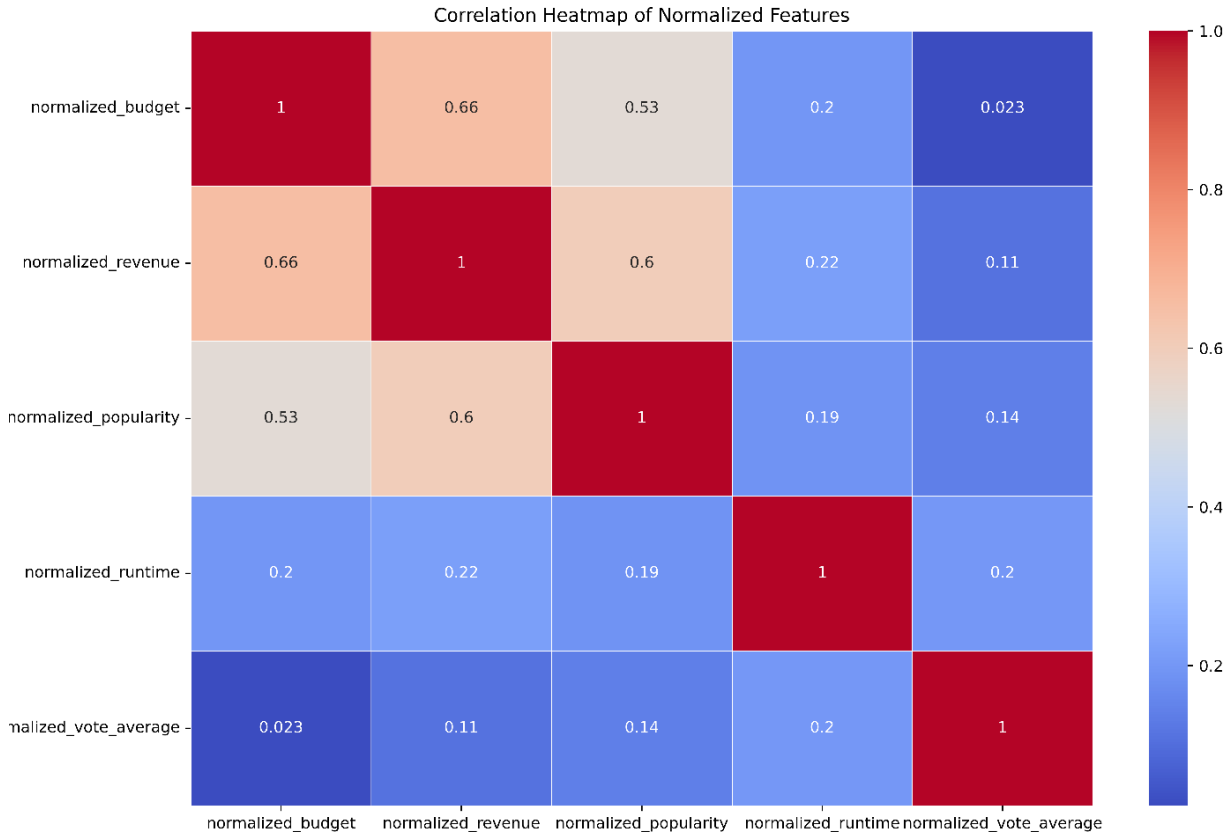

Figure 8 Correlation Matrix of Numerical features

- **Runtime Distribution:** Most movies have runtimes between 80–120 minutes, with a few outliers. This distribution suggests that categorizing films by runtime (short, standard, long) might be beneficial for feature engineering.

- **Streaming Availability (OTT Score):** Analysis of streaming platforms data reveals that movies available on popular services like Netflix and Prime Video are overrepresented. This supports the idea of creating an OTT score to boost recommendations for films that are more accessible.

Section 3: Feature Engineering and Categorical and Text Data Handling

3.1 Feature Engineering

Budget Popularity

This feature is designed to capture the combined impact of a movie's financial backing (budget) and its public appeal (popularity). The intuition is that movies with both high budgets and strong popularity signals are likely to perform well, and this multiplicative effect can help the model prioritize films with significant market and audience influence.

```
df['budget_popularity'] = df['budget'] * df['popularity']
```

Figure 9 Feature Engineering - Budget Popularity

Movie Age

The age of a movie is critical for distinguishing between recent releases and classics. Including the movie age allows the recommendation engine to account for trends over time, balancing contemporary films with older, potentially genre-defining movies and addressing the temporal bias observed in the dataset.

```
current_year = pd.Timestamp.now().year
df['movie_age'] = current_year - df['year']
df['movie_age'] = df['movie_age'].astype(float)
```

Figure 10 Feature Engineering - Movie age

OTT Score

The OTT score is engineered to summarize a movie's availability across popular streaming platforms. Movies available on multiple platforms (e.g., Netflix, Hulu, Prime Video, Disney+) are more accessible and, therefore, more likely to be recommended to users. This feature quantifies streaming accessibility, giving a positive bias to films that can be easily watched.

```
df['ott_score'] = df[['Netflix', 'Hulu', 'Prime Video', 'Disney+']].sum(axis=1)
```

Figure 11 Feature Engineering - OTT Score

Each of these engineered features addresses specific insights revealed during the EDA and supports the overall goal of refining the recommendation engine. The code snippets above, directly taken from the notebook, show how these features were implemented to improve the model's understanding of financial dynamics, temporal trends, and streaming accessibility.

3.2 Categorical and Text Data Handling

In the preprocessing phase, categorical variables representing streaming platforms (e.g., Netflix, Hulu, Prime Video, and Amazon) were encoded as binary indicators (One Hot Encoding) to ensure that each platform's availability could be effectively integrated into the recommendation model. I handled text encoding in two ways. In one approach, I used the TFIDF measure, and in the other, I employed the FastText word embeddings.

Text-based attributes:

The table below lists all text-based attributes, these attributes were extracted from the JSON attribute in the movies_metadata.csv file

Attribute	Description
director	The director of the movie
lead_actor	The Lead actor in the movie
original_language	Original language the movie was shot in
genre_list	List of Genres into which the movie is classified.
Production_list	List of production houses that made the film
Overview	An overview of the movie plot

TF-IDF measure

TF-IDF was chosen to encode the text-based features, actor, director, genre_list, production_list, original_language, and overview because it effectively captures the importance of terms within each movie relative to the entire dataset. This helps highlight distinguishing words (e.g., unique actor or genre names) while reducing the weight of common or less informative terms.
Unlike simple count-based methods, TF-IDF down-weights frequently occurring generic words and up-weights meaningful, rare terms. This makes it especially useful for short fields like director or genre and also for longer descriptions like the movie overview.
We configured the TF-IDF vectorizer with:

- max_features=5000 to limit dimensionality,
- stop_words='english' to remove common words,
- and sublinear_tf=True for better term scaling.

The resulting vectors provide a sparse but informative representation of movie content and metadata useful for short fields like actor, director, and genre because even small variations (e.g., "Brad Pitt" vs. "Leonardo DiCaprio") get encoded meaningfully. This makes it well-suited for clustering and similarity-based recommendations.

```python
def tokenize_features(row):
    tokens = []
    overview = row.get('overview', '')
    if isinstance(overview, str):
        tokens.extend(overview.split())

    for feature in ['director', 'lead_actor', 'original_language', 'genre_list',
    'production_list']:
        val = row.get(feature, '')
        if isinstance(val, str):
            try:
                parsed = ast.literal_eval(val)
                if isinstance(parsed, list):
                    temp_list = []
                    for item in parsed:
                        temp_list.append(str(item))
                    tokens.extend(temp_list)
                else:
                    tokens.append(str(parsed))
            except:
                tokens.append(val)
        else:
            tokens.append(str(val))
    return tokens
```

Figure 12 Code - Function used to tokenize the text-based Features

```python
df['tokens'] = df.apply(tokenize_features, axis=1)
```

Figure 13 Code - New column to store movie attribute tokens

```python
df['document'] = df['tokens'].apply(lambda tokens: " ".join(tokens))
```

Figure 14 - Creating a "document" with the generated tokens for TFIDF

```python
tfidf_vectorizer = TfidfVectorizer(stop_words='english', max_features=5000)
tfidf_matrix = tfidf_vectorizer.fit_transform(df['document'])
```

Figure 15 Code - TF-IDF Vectorizer

Further down in the report, when I discuss the models used for this recommendation engine, I will elaborate on how I used the tf-idf vectorized text attributes in the model along with the numerical attributes.

SVD Dimensionality Reduction

Since the TF-IDF matrix is high-dimensional (5000 features), **Truncated Singular Value Decomposition (SVD)** is applied to reduce its dimensionality.

- The reduced matrix (with a specified number of components, 100 in this case) captures the most significant latent semantic features.
- This step helps remove noise and improves the efficiency and quality of subsequent clustering.

```python
n_components = 100
svd = TruncatedSVD(n_components=n_components, random_state=42)
svd_matrix = svd.fit_transform(tfidf_matrix)
```

Figure 16 Code - SVD Dimensionality reduction

To visualize the SVD components, I further reduced the dimensionality of the SVD matrix to 3 components using principal component analysis (PCA) and plotted a 3D plot of the points.

```
pca_3d = PCA(n_components=3, random_state=21)
X_3d = pca_3d.fit_transform(svd_matrix)
labels = None
```

Figure 17 Code - PCA for visualizing the vectorized features

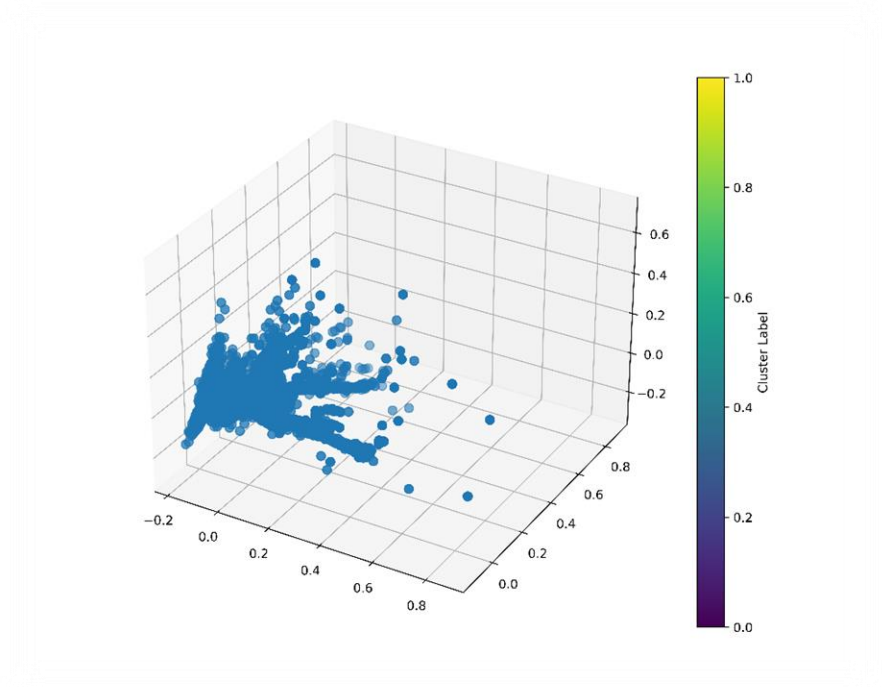

Figure 18 3D visualization of vectorized features

FastText

FastText and TF-IDF both convert text data into numerical data that can be used in computations, however, they differ significantly in both approach and output. TF-IDF is a statistical approach that depends on the frequency of word occurrence in a sentence or a document, generating sparse vectors. FastText generates dense, continuous embeddings by training on the context of words and even subword units, which allows it to capture semantic relationships and handle out-of-vocabulary words effectively.

A FastText skipgram model was trained on the same tokens that were used for the tfidf vectorizer. The dmin parameter was set to 100, while the number of epochs was 5.

```
with tempfile.NamedTemporaryFile(mode="w+", delete=False) as temp_file:
    for tokens in df['tokens']:
        temp_file.write(" ".join(tokens) + "\n")
    training_file = temp_file.name

ft_model = fasttext.train_unsupervised(input=training_file, model='skipgram', dim=100, epoch=5)
```

Figure 19Code - FastText Training

In the case of vectorization through FastText I did not employ dimensionality reduction because the resultant output vector has a dimension of 100, which suffices for my requirements.

To visualize the combined feature vector of 100 dimensions, I performed PCA to reduce the dimensionality to 2 vectors and plotted a scatter plot.

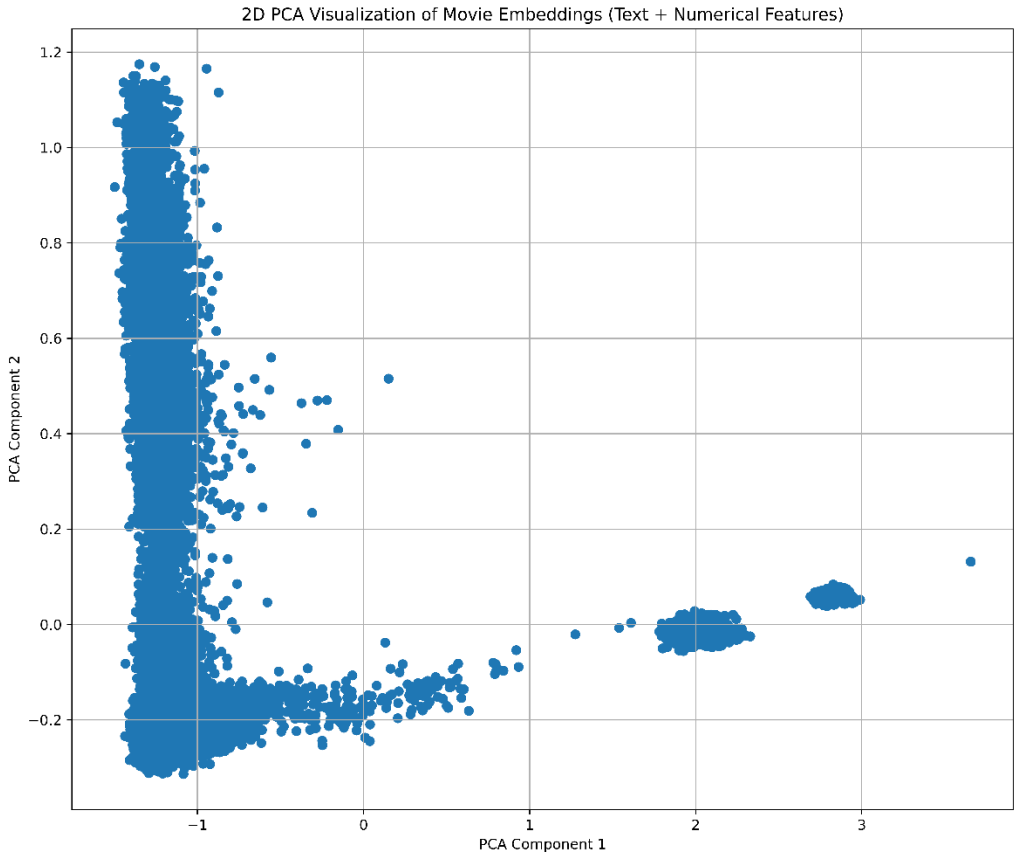

Figure 20 2D PCA visualization of Movie Embeddings

Section 4: Feature Importance and Selection

4.1 Features used for the Model

After performing the EDA on the data, I have decided to use the following attributes for developing my recommendation engine.

Director:

The EDA highlighted that directors tend to develop a consistent style or genre. Movies by the same director often share thematic and stylistic similarities, meaning that if a user enjoys one of a director's films, they are likely to enjoy others.

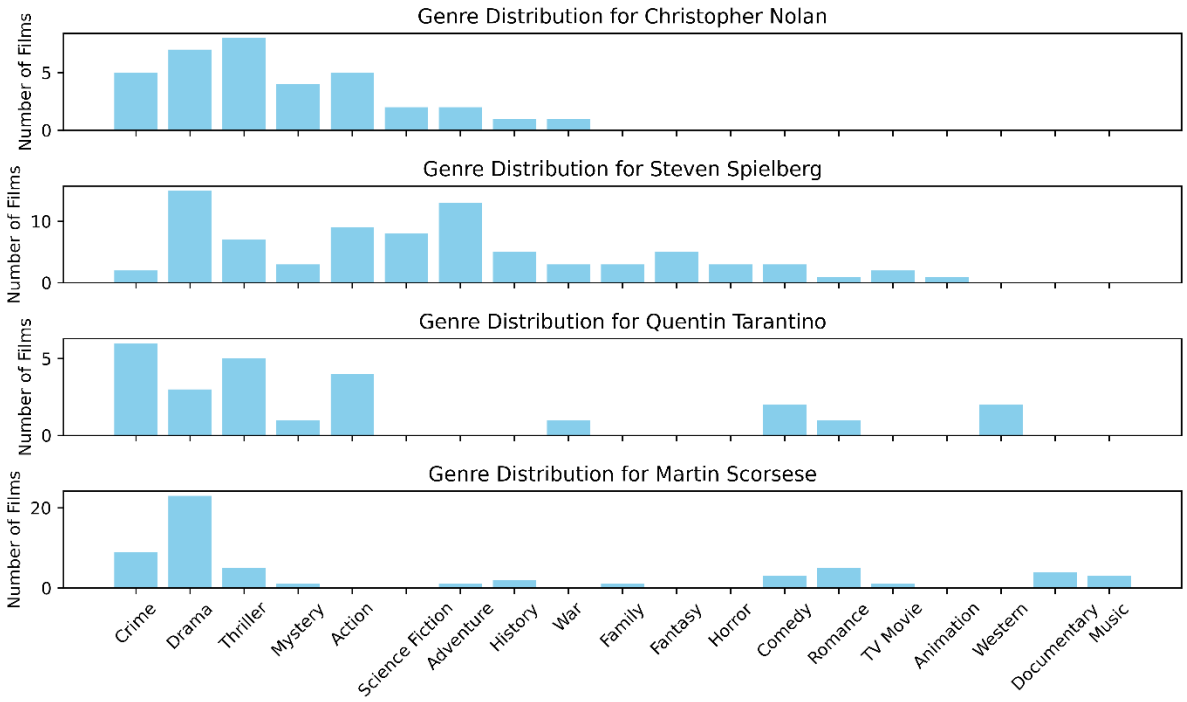

Figure 21 Director X Movie Genre Mapping

The above figure shows a mapping of directors and the number of movies in a particular genre that the director has made. As we can see, this clearly shows directors specializing in certain types of genres. Take Martin Scorsese for instance, his movies predominantly fall in the Drama – Crime category which is also evident in his filmography (Goodfellas, the Irishman etc), So, a person who liked a movie made my Scorsese is likely to like more movies made by him because that genre is the director's speciality.

Actor:

The analysis showed that lead actors play a crucial role in defining a movie's appeal. Popular actors attract loyal audiences, so including actor information helps capture viewer preferences based on star power. The rhetoric used for why "director" of a movie should be used as an attribute can also be used here.

Genre:

The genre distribution analysis revealed that genres such as Drama, Comedy, and Thriller dominate the dataset. This characteristic is vital because genre directly influences the tone and content of movies, making it a key factor in tailoring recommendations.

Production House:

The report indicated that major production houses (like Warner Bros, Paramount, etc.) are prominent and influence movie quality and style. Their consistent production values and signature storytelling techniques make them important for understanding movie similarities.

Language:

The EDA noted that most movies are produced in a few dominant languages, which affects user accessibility and cultural context. Incorporating language ensures that recommendations align with user language preferences.

Budget:

An analysis of budget trends and their correlation with revenue showed that higher-budget films generally have better production quality and market performance. This financial indicator is essential for understanding the scale and potential success of a movie.

Popularity:

Popularity was shown to be a strong signal of audience engagement. High popularity scores indicate movies that are well-received by the public, making it a useful predictor for recommending widely appreciated films.

Revenue:

Revenue, being strongly correlated with budget, serves as a financial performance metric. It helps in identifying movies that have successfully captured audience interest and commercial success. Movies earn more because these are the movies that a general audience likes, so it is more likely that such movies are liked by our users as well. Hence, the revenue of a movie becomes an important attribute. It is also evident from our EDA that movies that have a high gross revenue often share similarities in themes and storytelling style; they are often full of flamboyant action and visual effects. So, a person who has liked a high grossing movie is more likely to watch another high grossing movie.

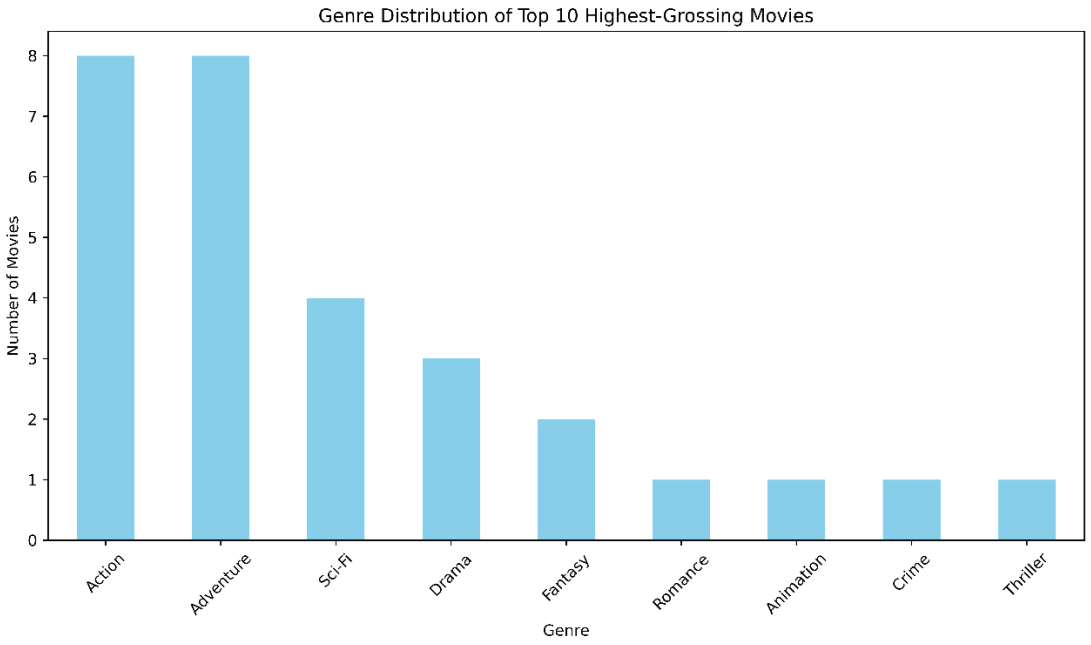

Figure 22 genre distribution of top grossing movies

Runtime:

Runtime parameters can also influence the recommendations as there could be used who lose interest in a movie if the movie is too long or a movie fails to engage them as it is too short. So, while recommending movies, I'm considering the runtime to be a contributing attribute.

Movie Age:

The EDA on release patterns revealed a significant increase in movies in recent years, with classic films forming a distinct subset. Movie age helps differentiate between contemporary trends and established, genre-defining movies.

OTT (Streaming Availability):

Analysis of streaming platform data indicated that movies available on major OTT services (like Netflix, Hulu, Prime Video, and Disney+) are more accessible to users. This feature was selected to ensure that recommendations favor films that can be readily watched, enhancing the overall user experience.

Vote Average:

Vote Average was considered a critical attribute for the recommendation engine due to its direct reflection of audience sentiment and movie quality. While budget and revenue provide insights into the financial aspects of a movie, the vote average captures how well a movie resonates with its viewers.

The EDA showed that the vote average does not have a strong correlation with budget or revenue, indicating that high-budget films do not always translate to better ratings. This suggests that audience ratings are influenced by factors beyond just financial investment, such as storytelling, acting, and direction, which may be more relevant for users looking for movies they will enjoy. Therefore, including the vote average ensures that the recommendation engine factors in genuine audience reactions, making the suggestions more aligned with user preferences.

Overview:

The overview is a text-based attribute because it provides a detailed description of the movie's plot, themes, and characters. This unstructured data contains essential contextual information that helps in understanding the movie's content, which is important for content-based recommendations. The overview allows the recommendation engine to identify similarities between movies with similar themes, even if their genres are different. It also complements other attributes like director or genre by adding an extra layer of context.

4.3 Feature combining

In the above mentioned features some features are numerical is nature and there are some which are text based. I vectorized the text-based feature using the TF-IDF vectorizer and the FastText embedding model, as discussed previously. After vectorizing the text-based feature, I combined these vectors with the numerical features. While combining, I multiplied each of these numerical features with a bias term; the multiplication bias factor was chosen arbitrarily based on my intuition and the EDA. The below code snippet shows how bias was added and the data was combined.

```
num_features = [
    'budget_popularity',
    'revenue',
    'runtime',
    'ott_score',
    'vote_average',
    'movie_age'
]
# Bias Dictionary - i.e. Multiplication factor for each feature
# This dictionary is used to adjust the importance of each feature in the final feature set.
bias_dict = {
    'budget_popularity': 1.0,
    'revenue': 1.0,
    'runtime': 1.2,
    'ott_score': 2.0,
    'movie_age': 1.5,
    'vote_average': 1.0
}
# Combing the features
numerical_data = df[num_features].fillna(0)
scaler = MinMaxScaler()
features = scaler.fit_transform(numerical_data)
numerical_features = np.zeros(features.shape)
for i, feat in enumerate(num_features):
    numerical_features[:, i] = features[:, i] * bias_dict[feat]

combined_features = np.hstack([svd_matrix, numerical_features])
```

Figure 23 Code - Combining the text-based features with the numerical features

The resulting combined feature matrix has both the text-based vectorized features and the bias-added numerical features for each movie.

4.2 Features that were discarded

In the context of unsupervised learning, feature selection presents unique challenges due to the absence of labelled data. Unlike supervised learning, where statistical tests like chi-square can assess the relationship between features and a target variable, unsupervised models lack such direct associations. Consequently, traditional statistical feature selection methods were not applicable.

For this project, feature selection was guided by a combination of exploratory data analysis (EDA), domain knowledge, and a practical understanding of user behaviour and recommendation relevance.

The below features were not considered when modelling the recommendation engine.

Production Country:
While this feature could offer cultural insight, modern viewing trends show that users regularly consume content from a wide range of countries. As a result, the country of production was deemed unlikely to significantly influence user preferences across global audiences.

Release Date:
This was considered redundant, as the **movie_age** feature effectively captures the same temporal information in a more directly interpretable way. Retaining both would introduce unnecessary overlap.

Tagline:
Taglines are often short, promotional phrases that do not contribute substantial descriptive or contextual value to a recommendation system. Their brevity and marketing-driven nature limit their usefulness in measuring similarity between movies.

Title:
Titles are typically abstract and inconsistent in conveying content themes or genres. Since they do not offer structured or semantically rich information, they were excluded in favor of more informative metadata like genre, director, and overview.

Adult Rating:
This binary flag is better suited for content filtering rather than preference modeling. It does not reflect user engagement or thematic similarity and showed minimal variation across the dataset.

Section 5: Data Modelling and Evaluation

5.1 Data Splitting - Test Train Data

Since the movie recommendation engine is an unsupervised learning problem, there are no predefined labels or target variables for training. As a result, traditional train-test split strategies used in supervised learning are not applicable here. Instead, the focus is on identifying patterns or similarities between movies based on their features, such as genre, budget, popularity, and overview.

To evaluate model performance without labelled data, we rely on internal validation metrics such as the Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index, which measure the cohesion and separation of clusters. These metrics act as a substitute for a formal train-test split and help assess the quality and usefulness of the clustering-based recommendation outputs.

5.2 Data Modelling

5.2.1 K-Means Clustering

The first model I chose for this task is the K-Means clustering model. The input for this model is the combined features matrix that was generated in the feature combining step (Discussed in Section 4.3 Feature combining of the report). I used the elbow method to get an understanding of the optimal number of clusters for the Data.

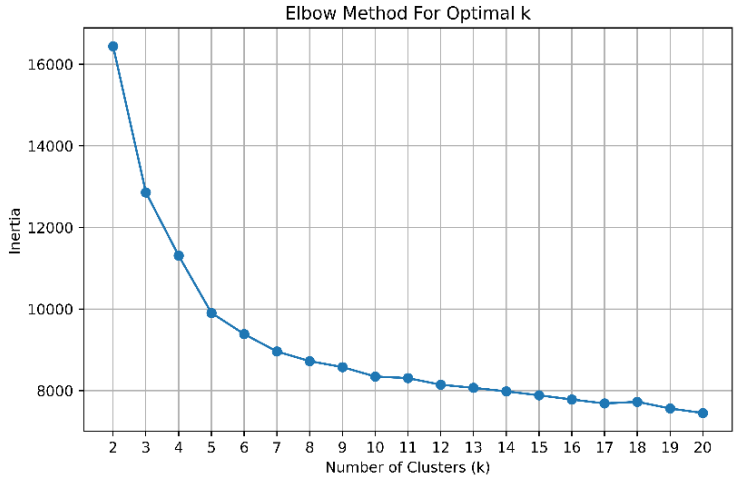

Figure 24 Elbow Method

By analyzing the elbow curve, we can see that the elbow of the curve is forming at approximately 7-8 clusters, however, I chose the number of clusters to be 10 as a personal choice.

```
num_clusters = 10
kmeans = KMeans(n_clusters=num_clusters, random_state=18)
df['cluster'] = kmeans.fit_predict(combined_features)
```

Figure 25 Code - K-means Clustering

In an attempt to visualize the clustering process, I plotted two components (out of 100) of the SVD matrix and mapped them to the cluster label they were assigned to.

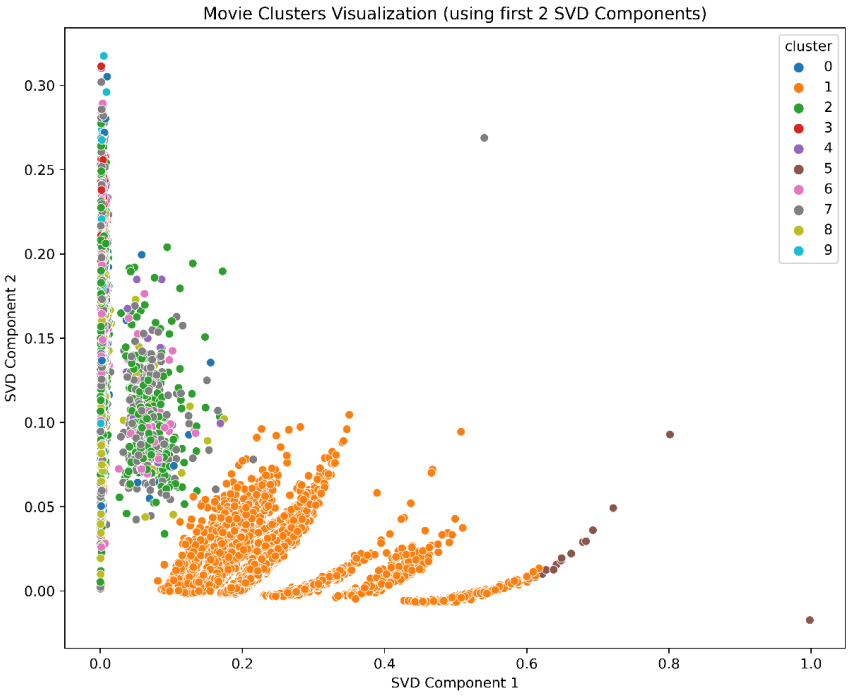

Figure 26 Cluster visualization across two components of the SVD (Total 100)

To get a list of recommendations, I took a list of all the movies in the same cluster as the user selected movie (index movie) and then using sklearn's cosine similarity function, computed the cosine similarity of each point (Movie) in the cluster to the index movie and returned a list of movies sorted by the cosine similarity.

Evaluation Metrics

In the evaluation of the K-means clustering model, three different clustering quality metrics were used: Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index.

The **Silhouette Score of 0.041** indicates that the clusters are poorly defined, as values close to 1 suggest well-separated clusters and values near 0 or negative indicate overlapping or poorly separated clusters.
The Davies-Bouldin Index value of 1.888 suggests that the clusters have a moderate level of overlap, as lower values indicate better clustering.
The **Calinski-Harabasz Index of 12472.554**, on the other hand, suggests that the model has a relatively high degree of separation between clusters, as higher values indicate better-defined clusters.

5.2.2 DBSCAN

K-Means clustering only groups the points into well-defined centric clusters, However, the actual data may not fall into a perfect circle, hence, the second model I chose is the DBSCAN, i.e. density scan that can form an arbitrary-shaped cluster that could better approximate the vector space the movies are mapped into.

```
dbscan_clusterer = DBSCAN(eps=0.2, min_samples=16, metric='cosine')
df['dbscan_cluster'] = dbscan_clusterer.fit_predict(combined_features)
```

Figure 27 Code - DBSCAN

For getting a list of recommendations the same approach as that of k-means was followed. First, the cluster was identified, and then, based on cosine similarity to the index movie, a list was returned.

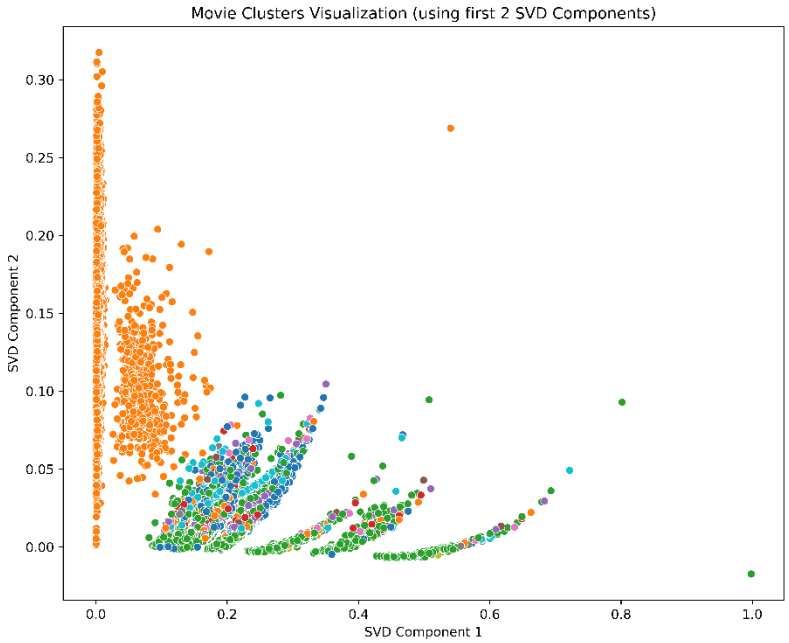

Figure 28 DB Scan Cluster visualization across two components of the SVD

Evaluation Metrics

The evaluation of the DBSCAN clustering model shows the following results:

DBSCAN Silhouette Score: 0.672
The silhouette score of 0.672 indicates that the clusters formed by DBSCAN are relatively well-defined, with reasonable cohesion within clusters and adequate separation between them.

DBSCAN Davies-Bouldin Index: 1.638
The Davies-Bouldin index of 1.638 suggests that the clusters have moderate overlap. A lower value would indicate better separation, but this score suggests that while the clustering is decent, there may still be some overlap or less clear distinctions between clusters.

DBSCAN Calinski-Harabasz Index: 858.332
The Calinski-Harabasz index of 858.332 indicates that the DBSCAN model has relatively well-separated clusters. A higher value suggests good cluster separation and compactness, meaning the model is effective in grouping similar items together while keeping distinct groups apart.

5.2.3 GMM (Gaussian Mixture Model)

The Gaussian mixture model offers a more flexible way of clustering data points, it does not classify a data point to a single

cluster, instead it offers a probability of a data point belonging to a multiple cluster. This offers more nuanced recommendations. For instance, a movie like “The Dark Knight” can be a part of both the Action and Thriller clusters; this offers a more flexible representation of movies.

```
from sklearn.mixture import GaussianMixture
num_components = 10
gmm = GaussianMixture(n_components=num_components, random_state=42)
gmm_labels = gmm.fit_predict(combined_features)
df['gmm_cluster'] = gmm_labels
```

Figure 29 Code - Gaussian Mixture Model

To get a list of recommendations, the same approach as that of k-means was followed. First, the cluster was identified, and then, based on cosine similarity to the index movie, a list was returned.

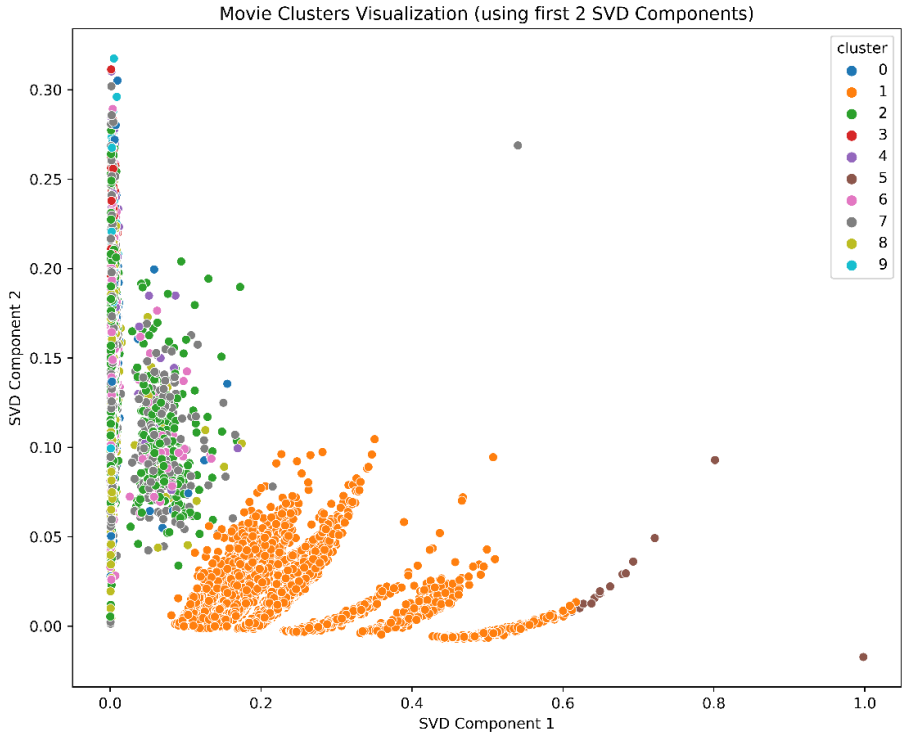

Figure 30 Gaussian Cluster visualization across two components of the SVD (Total 100)

Recommendation Function:

The below code snippet is an implementation of a function that selects movies in the same cluster and computes the cosine distance to gather recommendations. All the three clustering models (K-Means, DBSCAN, and GMM) use a similar implementation of this function.

```python
def get_recommendations_gmm(movie_title, top_n=10):

    movie_title_lower = movie_title.lower()
    if movie_title_lower not in title_to_index:
        raise ValueError(f"Movie '{movie_title}' not found in the dataset.")

    idx = title_to_index[movie_title_lower]
    target_cluster = df.loc[idx, 'gmm_cluster']
    # Retrieve movies from the same cluster.
    cluster_movies = df[df['gmm_cluster'] == target_cluster]

    # Compute cosine similarity between the target movie and movies in the same cluster.
    target_vec = combined_features[idx].reshape(1, -1)
    cluster_indices = cluster_movies.index
    cos_sim = cosine_similarity(target_vec, combined_features[cluster_indices]).flatten()

    cluster_movies = cluster_movies.copy()
    cluster_movies['similarity'] = cos_sim

    cluster_movies = cluster_movies.drop(idx)

    recommendations = cluster_movies.sort_values(by='similarity', ascending=False)['title'].head(top_n).tolist()
    return recommendations
```

Figure 31 Code - Recommendations Function

Evaluation Metrics

GMM Silhouette Score: -0.066

The negative Silhouette Score of -0.066 indicates that the clusters formed by the model are not well-separated, and some of the points may be misclassified or poorly assigned to clusters. However, it's important to remember that GMM is a probabilistic model, meaning that each data point has a probability of belonging to multiple clusters. A negative silhouette score may reflect these overlapping memberships, suggesting that the model is not as confident about the cluster assignments, which is characteristic of GMM's flexibility.

GMM Davies-Bouldin Index: 7.078

The Davies-Bouldin Index of 7.078 is higher than ideal, indicating that the clusters have some level of overlap. This index measures the ratio of intra-cluster scatter to inter-cluster separation, so a higher value suggests that the clusters are not as distinct. Given that GMM produces soft clusters, some level of overlap is expected, and this score might be reflecting that inherent characteristic of GMM.

GMM Calinski-Harabasz Index: 4751.553

The Calinski-Harabasz Index of 4751.553 suggests that the clusters, while not perfectly defined, are relatively well-separated in terms of variance between clusters. This metric values cluster separation and compactness, and in the case of GMM, it can be higher despite some overlap, as the model effectively separates data points based on their probabilities of belonging to each cluster.

5.2.4 FastText-Based Embeddings:

FastText is an extension of Word2Vec that improves word embeddings by breaking words into subword units (n-grams). This enables it to better capture the structure of words and handle rare or unseen terms more effectively. Using the Skipgram training algorithm, FastText predicts context words from a target word and generates dense vector representations that encode semantic relationships.

The tokenized textual features—such as director, lead actor, genre, production house, and movie overview—are processed through FastText to generate 100-dimensional dense embeddings. These are then combined with numerical features (e.g., budget, popularity), which are scaled using MinMaxScaler. This fusion of textual and numerical data creates a rich, compact representation for each movie.

The final recommendation is generated using cosine similarity: for a given input movie, the cosine distance between its vector and others is computed, and the closest matches are returned as recommendations.

```
with tempfile.NamedTemporaryFile(mode="w+", delete=False) as temp_file:
    for tokens in df['tokens']:
        temp_file.write(" ".join(tokens) + "\n")
    training_file = temp_file.name

ft_model = fasttext.train_unsupervised(input=training_file, model='skipgram', dim=100, epoch=10)
```

Figure 32 Code - FastText Training

Unlike clustering models (K-Means, DBSCAN, GMM), this approach does not segment movies into discrete groups. Therefore, internal clustering metrics such as Silhouette Score or Davies-Bouldin Index are not applicable here. Instead, qualitative evaluation was performed using ground truth tests.

5.3 Ground truth test case:

The above-mentioned metrics evaluate the quality of the cluster, to understand if true quality of the recommendations, I gave a movie as an input to the model and evaluated the output though common knowledge, and to maintain uniformity in the test scenario, I gave the same movie as input to all the models being discussed.

Input Movie: Skyfall

K-Means Clustering	Casino Royale, Mission: Impossible - Ghost Protocol, Django Unchained, Captain America: The Winter Soldier, Eagle Eye, Total Recall, The Avengers, Quantum of Solace, Star Wars: The Force Awakens, Avengers: Age of Ultron.
DBScan	Casino Royale, Mission: Impossible - Ghost Protocol, Django Unchained, Captain America: The Winter Soldier, Eagle Eye, Total Recall, The Avengers, Quantum of Solace, Star Wars: The Force Awakens, Avengers: Age of Ultron.
GMM	Captain America: The Winter Soldier, Thor: The Dark World, The Dark Knight, The Chronicles of Narnia: The Lion, The Witch and the Wardrobe, Hugo, Captain America: Civil War, Batman Begins, Guardians of the Galaxy, G.I. Joe: The Rise of Cobra, Eight Below.
FastText Embeddings	Captain America: The Winter Soldier, Iron Man 2, Pirates of the Caribbean: Dead Men Tell No Tales, The Dark Knight, Total Recall, Mission: Impossible - Ghost Protocol, The Expendables 3, Star Trek Into Darkness, Avengers: Age of Ultron, Casino Royale.

Input Movie: Interstellar

K-Means Clustering	Inception, Watchmen, Pacific Rim, The Dark Knight Rises, The Matrix, The Matrix Reloaded, Valerian and the City of a Thousand Planets, The Revenant, Troy, The Hunger Games: Catching Fire.
DBScan	Inception, Watchmen, Pacific Rim, The Dark Knight Rises, The Matrix, The Matrix Reloaded, Valerian and the City of a Thousand Planets, The Revenant, Troy, The Hunger Games: Catching Fire.
GMM	Inception, The Dark Knight Rises, The Prestige, Into the Wild, Superman Returns, Mission: Impossible - Rogue Nation, The Pursuit of Happyness, Iron Man, Maze Runner: The Scorch Trials, Saving Private Ryan.
FastText Embeddings	Fantastic Beasts and Where to Find Them, The Hobbit: The Battle of the Five Armies, Valerian and the City of a Thousand Planets, Tomorrowland, Transformers: Age of Extinction, Aliens, Pacific Rim, Prometheus, Battleship, Transformers: The Last Knight.

Input Movie: Toy Story

K-Means Clustering	Toy Story 2, Chicken Run, Shrek, Toy Story 3, The Princess Bride, Ratatouille, Kung Fu Panda, Sister Act, Wreck-It Ralph, Finding Nemo.
DBScan	Toy Story 2, Chicken Run, Shrek, Toy Story 3, The Princess Bride, Ratatouille, Kung Fu

	Panda, Sister Act, Wreck-It Ralph, Finding Nemo.
GMM	Toy Story 2, Ratatouille, Aladdin, The Polar Express, Penguins of Madagascar, The Addams Family, Philadelphia, Austin Powers: The Spy Who Shagged Me, The Italian Job, Quantum of Solace.
FastText Embeddings	Toy Story 2, Toy Story 3, Burn After Reading, Finding Nemo, Bolt, Cars 3, The Emperor's New Groove, Cloudy with a Chance of Meatballs 2, The Incredibles, The Twilight Saga: Breaking Dawn - Part 1.

Summary of Observations:

Skyfall (Action, Spy Thriller):

- **K-Means & DBSCAN:** Accurately recommended other spy-action thrillers like *Casino Royale*, *Mission: Impossible*, and *Quantum of Solace*. These models focused tightly on the action-thriller genre.
- **GMM:** While some recommendations (e.g., *The Dark Knight*) were thematically consistent, others (e.g., *Chronicles of Narnia*, *Hugo*) strayed into unrelated fantasy genres, showing genre leakage.
- **FastText:** Maintained relevance with recommendations like *Iron Man 2*, *Total Recall*, and *Avengers: Age of Ultron*, showing strong semantic understanding while introducing some stylistic variety.

Interstellar (Sci-Fi, Drama):

- **K-Means & DBSCAN:** Consistently returned sci-fi and dystopian titles (*Inception, The Matrix, Pacific Rim*), ideal for users with focused preferences.
- **GMM:** Added some emotionally complex films (*The Prestige, Saving Private Ryan*), demonstrating genre blending but with acceptable thematic overlap.
- **FastText:** Delivered a futuristic and effects-heavy set (*Valerian, Prometheus, Transformers*), reflecting strong textual associations, though a few selections leaned toward blockbuster action rather than deep sci-fi drama.

Toy Story (Animated, Family):

- **K-Means & DBSCAN:** Returned highly relevant family animations (*Toy Story 2, Shrek, Kung Fu Panda*), suggesting genre-focused precision.
- **GMM:** Mixed animation (*Aladdin*) with less related entries like *Philadelphia* and *Quantum of Solace*, weakening genre consistency.
- **FastText:** Captured animation themes well (*Finding Nemo, The Incredibles*), but introduced outliers like *Burn After Reading* and *Twilight*, likely due to looser semantic associations in plot descriptions.

5.3.1 Ground Truth Analysis:

For all test cases, K-Means and DBSCAN produced highly consistent recommendations. Their suggestions mostly stayed within the same genre cluster, making them reliable for users with well-defined preferences. However, they lacked diversity, often recommending very similar films.

The GMM model introduced more variety, often mixing genres. While this sometimes led to insightful cross-genre picks (e.g., *The Prestige* for *Interstellar*), it occasionally returned less relevant results (*Quantum of Solace* for *Toy Story*), likely due to its probabilistic nature and soft clustering.

The FastText-based model delivered the most nuanced recommendations. It captured deeper semantic relationships between movies, especially for complex narratives like *Skyfall* or *Interstellar*. However, it also introduced occasional outliers (*Burn After Reading*, *Twilight* for *Toy Story*), reflecting the challenge of relying solely on text-based embeddings. Despite this, FastText's flexibility makes it particularly suitable for users seeking variety or hidden gems.

Section 6: Conclusion

In this report, we successfully developed a movie recommendation engine that leverages a combination of clustering algorithms (K-Means, DBSCAN, GMM) and FastText-based embeddings to generate personalized movie suggestions. The engine uses key movie attributes such as director, genre, and vote average, along with new features like movie age, budget-popularity interaction, and OTT score, to enhance recommendation accuracy. We evaluated the model using metrics like the Silhouette Score, Davies-Bouldin Index, and Calinski-Harabasz Index to assess cluster quality. After evaluation, GMM and FastText proved to offer better results in providing more meaningful, relevant, and accurate movie recommendations. The system effectively combines text and numerical data, making it a valuable tool for personalized movie suggestions.