PROJECT REPORT *on*

NETFLIX MOVIES AND TV SHOWS RECOMMENDATION SYSTEM

CSE587C: Data Intensive Computing Spring 2024

Instructor: **Dr. Shamshad Parvin** *Deadline*: 03 May 2024

Team members:	*UBIT Name*	*Person Number*
Leela Satya Praneeth Posina	**leelasat**	**50540667**
Surya Suhas Reddy Sathi	**suryasuh**	**50538640**
Sai Venkat Reddy Sheri	**ssheri**	**50545752**

PHASE - 1

PROBLEM STATEMENT

As Netflix's content library continues to rapidly expand, subscribers face increasing difficulty in discovering new titles that align with their personal interests and tastes. The sheer volume of available movies and TV shows makes manual browsing an inefficient and frustrating process, as users must sift through vast catalogs to identify relevant options amidst nuanced preferences.

To address this challenge, there is a growing need for an accurate and personalized recommendation system that can effectively match users with enjoyment-optimized viewing options based on a specific movie or TV show they have already watched and enjoyed. By leveraging a user's positive experience with a particular title, the system can analyze its characteristics and identify other similar titles the user is likely to appreciate.

However, developing such a robust recommendation engine requires access to proprietary data on user viewing histories, content preferences, and behavioral patterns – information that streaming platforms like Netflix closely guard. This lack of transparency presents barriers to external entities aiming to innovate in this space, underscoring the value of a recommendation system tailored specifically to the Netflix platform and its unique dataset.

BACKGROUND:

The Netflix shows dataset from Hugging Face contains information on over 8,000 TV shows and movies available on Netflix as of 2019. The data includes details like the show's title, release year, maturity rating, genres, user ratings, duration, countries of availability, description, and directors/cast. This dataset provides a comprehensive look at Netflix's content library and allows for analysis of trends across genres, ratings, release dates, and more.

CONTRIBUTION AND IMPORTANCE:

The release of this Netflix dataset on Hugging Face is an important contribution as it enables public access to insights on Netflix's vast catalog of shows and movies. Most data on specific streaming platforms' content libraries are proprietary and closely guarded. Providing open access to a dataset of this scale related to Netflix is valuable for researchers, analysts, and other parties interested in better understanding consumer viewing patterns and streaming content itself. This data can support valuable analyses of entertainment trends in the era of digital streaming.

TARGET USERS:

1. This dataset would be valuable for movie buffs, cinephiles, and entertainment enthusiasts interested in analyzing trends related to films and series.
2. Academic researchers studying trends in entertainment, streaming platforms, and consumer viewing patterns.
3. Entrepreneurs and product managers exploring opportunities related to streaming content.
4. Marketing professionals interested in understanding consumer preferences via content analytics.

DATA SOURCES AND DETAILS ABOUT DATA

- The dataset was compiled and published on Hugging Face by user 'huggingfacelearners' in November 2019.
- The exact original sources of the Netflix catalog data are unclear, but it appears to have been web scraped or extracted from Netflix's public-facing applications/APIs around 2019.

Dataset Details :

- Over 8,000 shows and movies available on Netflix
- 12 attributes per title:
    - **show_id** - unique ID for the title.
    - **type** – says whether the content is a movie or a TV show.

- **title** - name of title.
- **director** - directors (for movies).
- **cast** - main actors/actresses.
- **country** - production country.
- **date_added** - date when added to Netflix.
- **release_year** - year when title was released.
- **rating** - MPAA rating.
- **duration** - duration in minutes (movies or episode).
- **listed_in** - genre(s).
- **description** - short text description of the plot.

- Data is a mix of categorical variables like genres and text fields like descriptions, along with continuous numeric data like duration and discrete data like years and dates.
- The data provides a snapshot of the Netflix catalog in 2019 rather than longitudinal view.
- Biases exist around greater representation of English-language content.

This dataset provides a valuable basis for analysis despite some limitations, enabling insights into the content types, availability by country, ratings, release timeframes etc. The inclusion of multiple data types allows for a multitude of analyses.

DATA CLEANING

Several data preparation techniques were applied to handle anomalies in the Netflix dataset while preserving data integrity. The following are the techniques that are followed to do so.

1. **Handling Missing Data:**

   Missing data can arise due to several reasons, such as data collection errors and inefficacy during the data collection. The missing values in the 'director', 'cast', and 'country' columns were imputed by filling them with 'N/A'. Then, rows with missing 'date_added', 'rating' and 'duration' were dropped to remove those missing observations.

2. **Feature Engineering**:

   Feature Engineering is the process of transforming existing features to improve the efficacy of the data. The 'date_added' column was split into separate 'month_added' and 'year_added' columns using pandas' datetime capabilities. This extracts more granular temporal features.

3. **Encoding Categorical Data:**

Label encoding was applied on the 'type' column to convert the categorical TV shows/movies into numerical labels. The countries were also label encoded to numeric categories in a new 'country_code' column.

4. **One-Hot Encoding**:

The 'listed_in' genres were one-hot encoded into multiple binary columns indicating the presence/absence of each genre. This converts the categorical genres to format usable for modeling.

5. **Numerical Transformations**:

The content ratings present in 'rating' column were label encoded to numeric categories.

6. **Data Cleaning**:

The Show IDs were standardized by removing extraneous 's' symbols in the IDs using string replacement. This makes the IDs more consistent.

7. **Removing Rows with excess number of null values**.

It helps improve the quality of the dataset by eliminating incomplete or missing entries that could potentially introduce bias or lead to inaccurate analysis.

8. **Using NLP to bring all the description to lower case.**

This ensures consistency and standardization, which can be beneficial for text processing and analysis tasks.

9. **Using NLP to make the description punctuation free.**

This helps in simplifying the text data, reduce noise and potentially improve the performance of our analysis.

10. **Removing Extra Columns:**

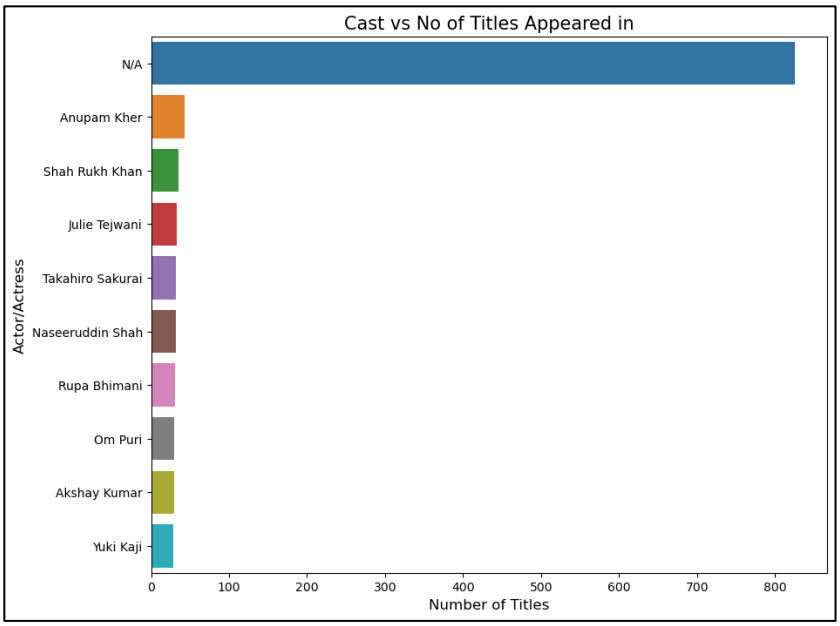
We removed columns which are irrelevant to the analysis. They include listed_in, etc.,

Overall, key steps like imputation, feature engineering, encoding categories, and data cleaning were performed to prepare the Netflix data for analysis and modeling. The code demonstrates practical data manipulation using Pandas and Sklearn to shape the raw data.

EXPLORATORY DATA ANALYSIS

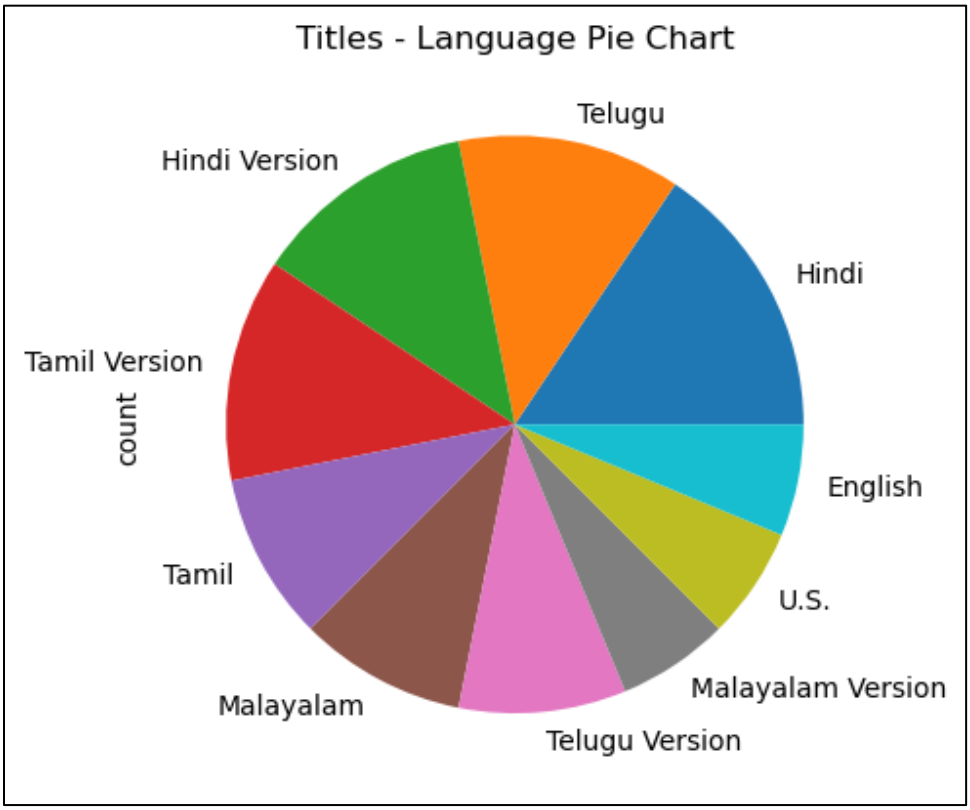
1. **Bar Chart for Cast with Highest Number of Titles:**

The cast bar chart displays the most prolific actors by movie count. Anupam Kher, Shah Rukh Khan, and Robert De Niro are the top three cast in the data base. Highlighting popular stars can aid recommendations and help users find films with favored talent.



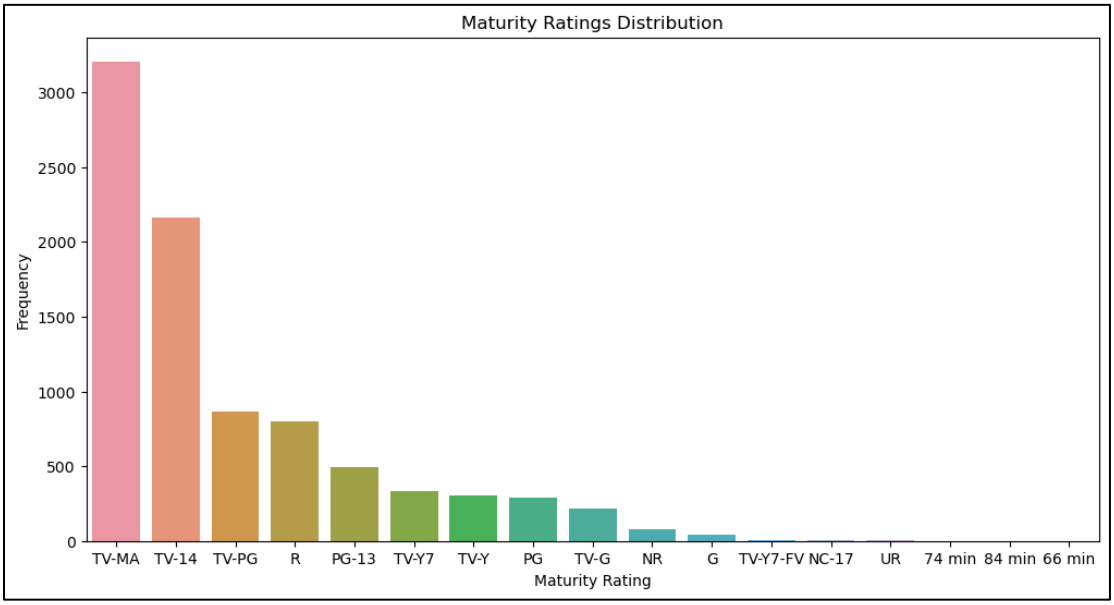
2. **Pie chart to show Language Contribution**:

The languages pie chart reveals title diversity - predominantly Indian Languages but other languages like English and Spanish exist too. Languages help target certain recommendation audiences. This is just based on the title extensions. There is no specific feature as language provided in the dataset.



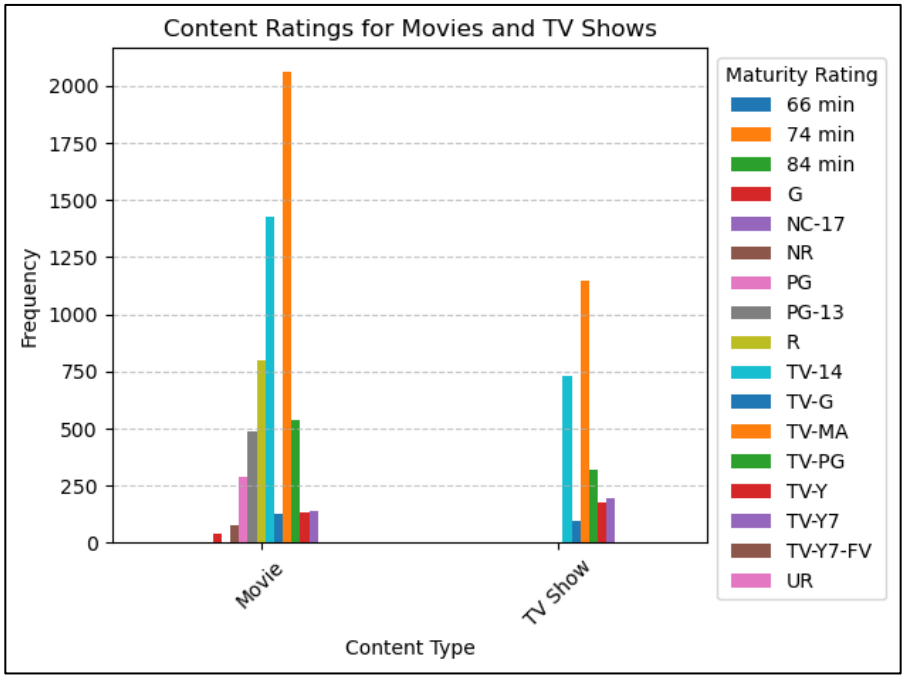
3. Counter Plot for Content Rating

The countplot shows R-rated films are the most common, implying the dataset skews adult. Recommendations should account for more mature movies vs family ones.



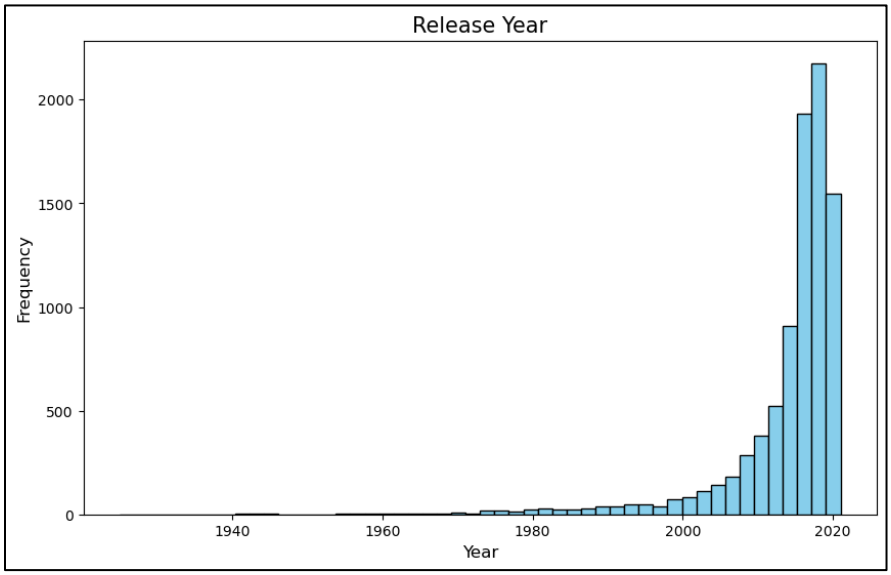
4. Bar Plots to Visualize Content Wise Maturity Rating

A stacked bar plot visualized counts over rating type, this helps us find out what kinds of content is being made the most and which is being made the highest.



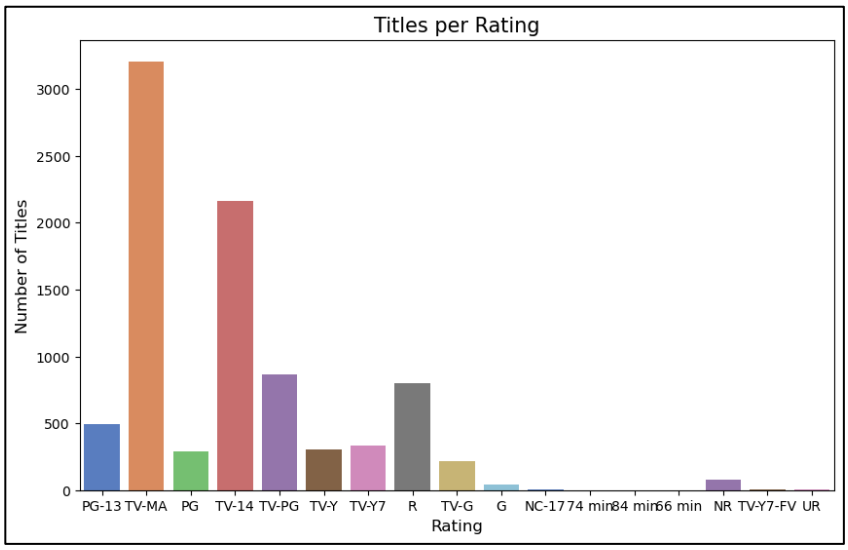
5. Histogram for Release Year

A histogram was plotted to visualize the distribution of movie release years. This revealed overall trends in movies produced over time.



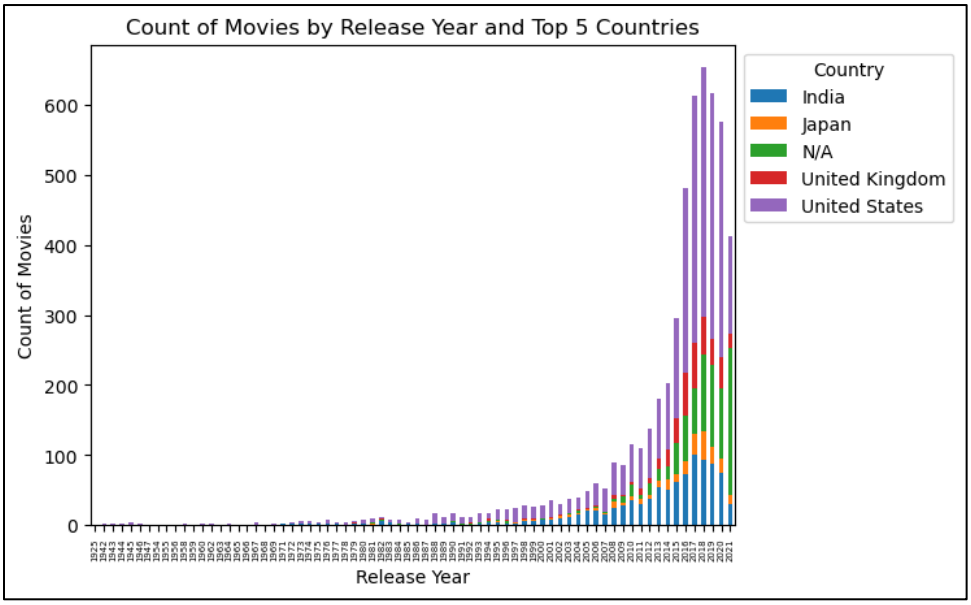
6. **Histogram for titles per rating**:

This histogram shows us what kinds of ratings are present and how many titles are in each rating.



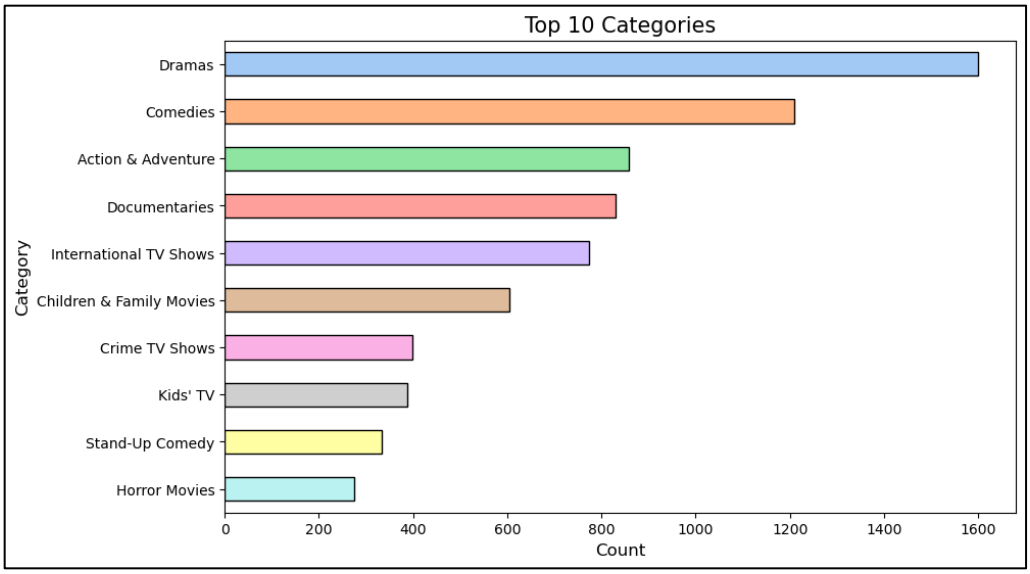
7. **Stacked Bar Plot over Time by Country**

A stacked bar plot visualized movie counts over release years, separated by the top 5 country producers. We can compare trends across major contributing countries.



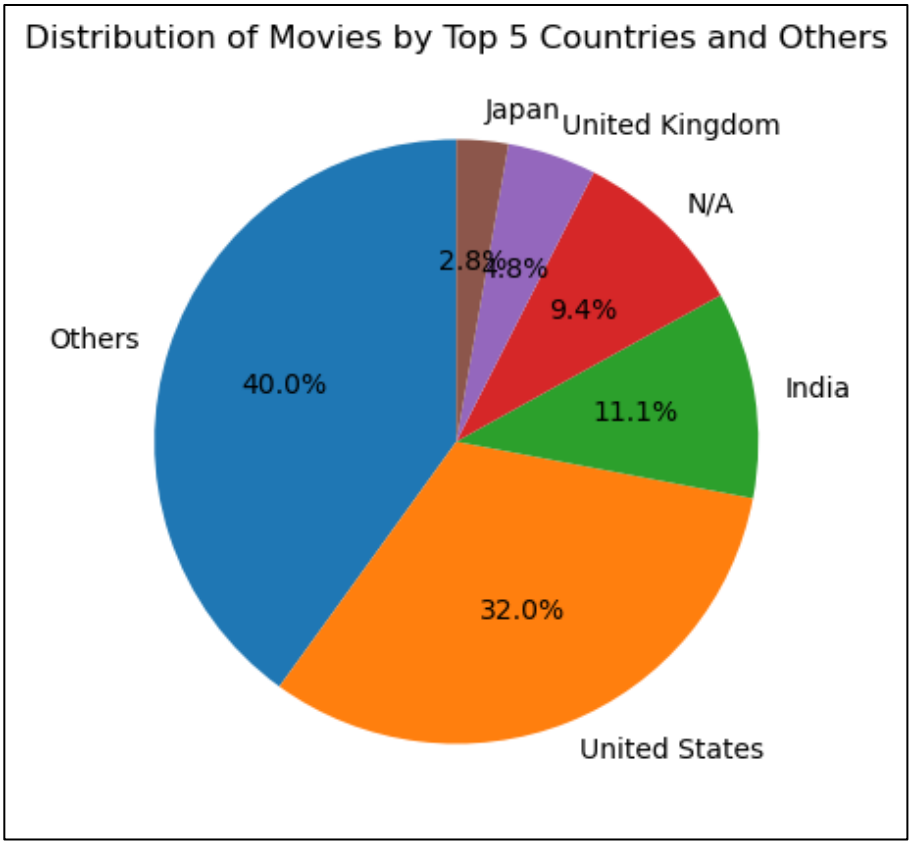
8. **Bar Plot of Top 10 Categories:**

This bar plot represents the top 10 categories of titles that are present in the dataset. This shows us how the titles are distributed along the categories.



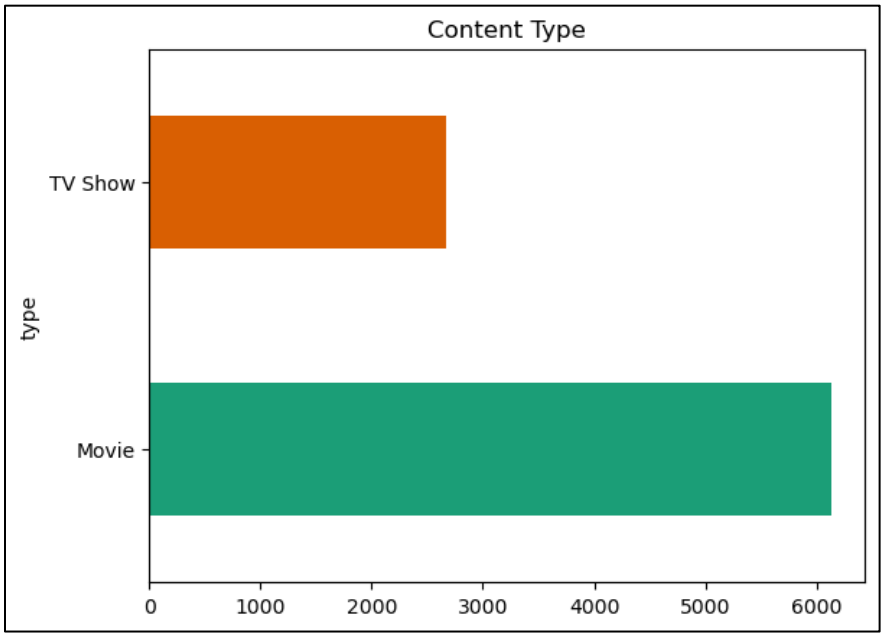
9. Pie Chart for Top Countries:

A pie chart broke down movie frequency by the top 5 countries vs others. This enabled analysis of how countries compare in terms of movie output.



10. Bar Plot on show counts:

This bar plot demonstrates the number of titles distrusted into either movies or TV Shows.



Together, these EDA techniques offer an extensive understanding of the dataset's properties, allowing developers to make well-informed choices about feature engineering, data preparation, and model selection.

PHASE 2

MODELS USED

The models suggested by professor, such as regression and classification, are designed for supervised learning tasks where the data is labeled. However, our problem falls under the category of unsupervised learning, as the dataset we are working with is unlabeled. To effectively address this clustering problem, we have chosen to explore the following algorithms:

11. K-Means Clustering.
12. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH).
13. Mean-shift Clustering.
14. Gaussian Mixture Model.
15. Agglomerative Clustering
16. Density Based Spatial Clustering of Applications with Noise (DBSCAN).

To develop an effective movie recommendation system for Netflix's vast library, we employed the above-mentioned unsupervised clustering algorithms to group similar titles together based on their inherent features. By leveraging a diverse set of clustering methodologies, our system can effectively identify natural groupings of

movies/shows that share commonalities in genres, ratings, descriptions, and other relevant attributes. This clustering foundation enables personalized recommendations by matching a user's previously enjoyed title to other titles within the same cluster, maximizing the likelihood of suggesting content aligned with their preferences.

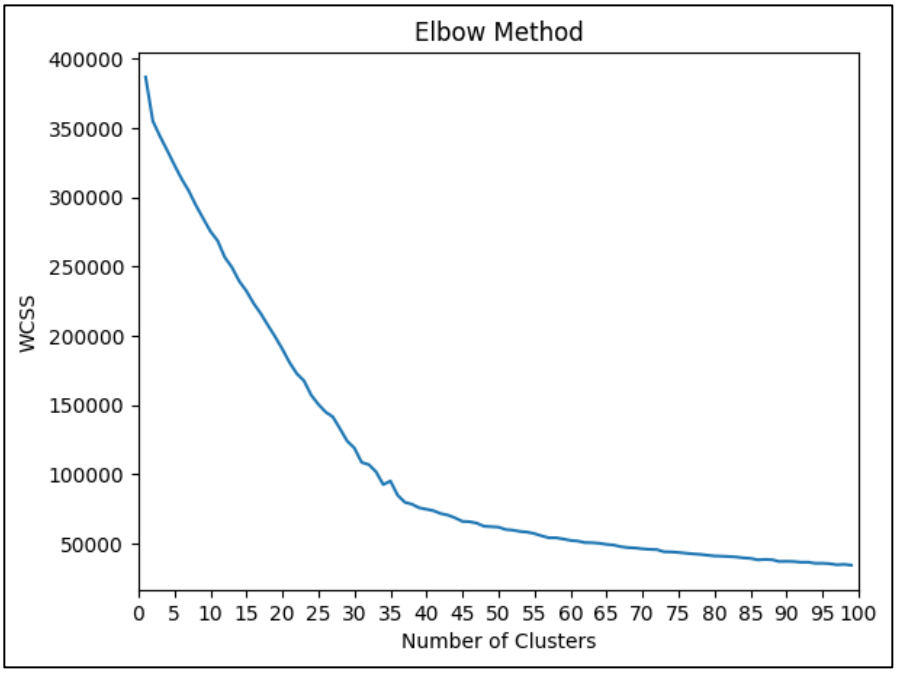
1. K-Means Clustering

To group similar movies together and enable personalized recommendations, we employed the K-Means clustering algorithm from scikit-learn. This partitioning technique aims to divide the dataset into K distinct non-overlapping clusters, where each data point (movie) belongs to the cluster with the nearest mean (centroid).

Initially, we extracted the movie titles and features from the dataset. The features were then scaled using StandardScaler to ensure equal contribution from all dimensions during the clustering process.

Determining Optimal K:

One crucial step in K-Means is determining the optimal number of clusters (K). We used the Elbow method, which calculates the sum of squared errors (SSE) for different values of K. The plot shows how the SSE decreases as K increases, with an "elbow" point at K=34, indicating the optimal trade-off between minimizing SSE and avoiding excessive clusters.



Model Training:

With the optimal K value determined, we trained the K-Means model using the scaled movie features. The algorithm iteratively assigned each movie to the nearest centroid and updated the centroids based on the new cluster assignments, until convergence.

Cluster Assignment:

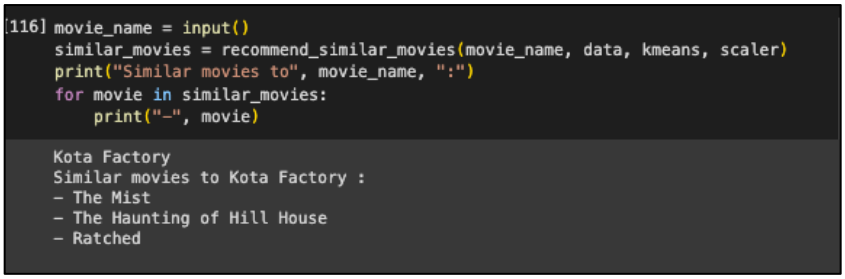
The trained model assigned a cluster label to each movie in the dataset, enabling us to group similar movies together based on their inherent features.

Recommendation Function:

We implemented a function to recommend similar movies given a target movie title. This function operates as follows:

1. Locate the target movie in the dataset and extract its features and cluster assignment.
2. Identify all movies belonging to the same cluster as the target movie.
3. Calculate the cosine similarity between the target movie's features and all other movies within the cluster.
4. Return the top N most similar movies based on the cosine similarity scores.

The output demonstrates the recommended movies for the input "Kota Factory," showcasing the algorithm's ability to suggest relevant titles based on shared attributes.



Hyper parameter Tuning:

We changed the value of k three times which are 22, 34 in which both the model with k = 34 performed well. We chose k=37 and moved forward. The evaluation metrics for k = 22 are in the below image and the ones for k = 37 are discussed below.

```
Silhouette Score: 0.3222162499668184
Calinski-Harabasz Index: 517.9874787890385
Davies-Bouldin Index: 1.3001086209057229
Homogeneity: 0.2908727378250743
Completeness: 1.0000000000000004
V-measure: 0.45066059465342956
```

Evaluation Metrics:

To assess the clustering performance, we calculated several evaluation metrics:

Silhouette Score: 0.44683242265294545

The Silhouette Score measures how well each data point (movie) fits into its assigned cluster compared to other clusters. The score ranges from -1 to 1, with a higher value indicating that the data point is well-matched to its cluster and poorly matched to neighboring clusters.

In this case, the Silhouette Score of 0.446 is a moderately positive value, suggesting that the clustering is reasonably good, but there is room for improvement. A score closer to 1 would indicate that the movies are very well-clustered and separated from other clusters.

Calinski-Harabasz Index: 935.845623849385

The Calinski-Harabasz Index is a ratio of the sum of between-cluster dispersion and inter-cluster dispersion for all clusters. A higher value indicates that the clusters are dense and well-separated from each other.

While there is no definitive interpretation of the Calinski-Harabasz Index value, a higher value is generally considered better. In this case, the value of 935.845 suggests that the clustering is reasonably good.

Davies-Bouldin Index: 0.8452928989425635

The Davies-Bouldin Index is a measure of the average similarity between clusters, where lower values indicate better separation between clusters. A value of 0 would indicate that the clusters are completely separated, while higher values suggest that the clusters are closer together and potentially overlapping.

The obtained Davies-Bouldin Index of 0.845 is a moderately low value, indicating that the clusters are reasonably well-separated, but there is still some overlap or similarity between clusters.

Homogeneity: 0.3558867248431815

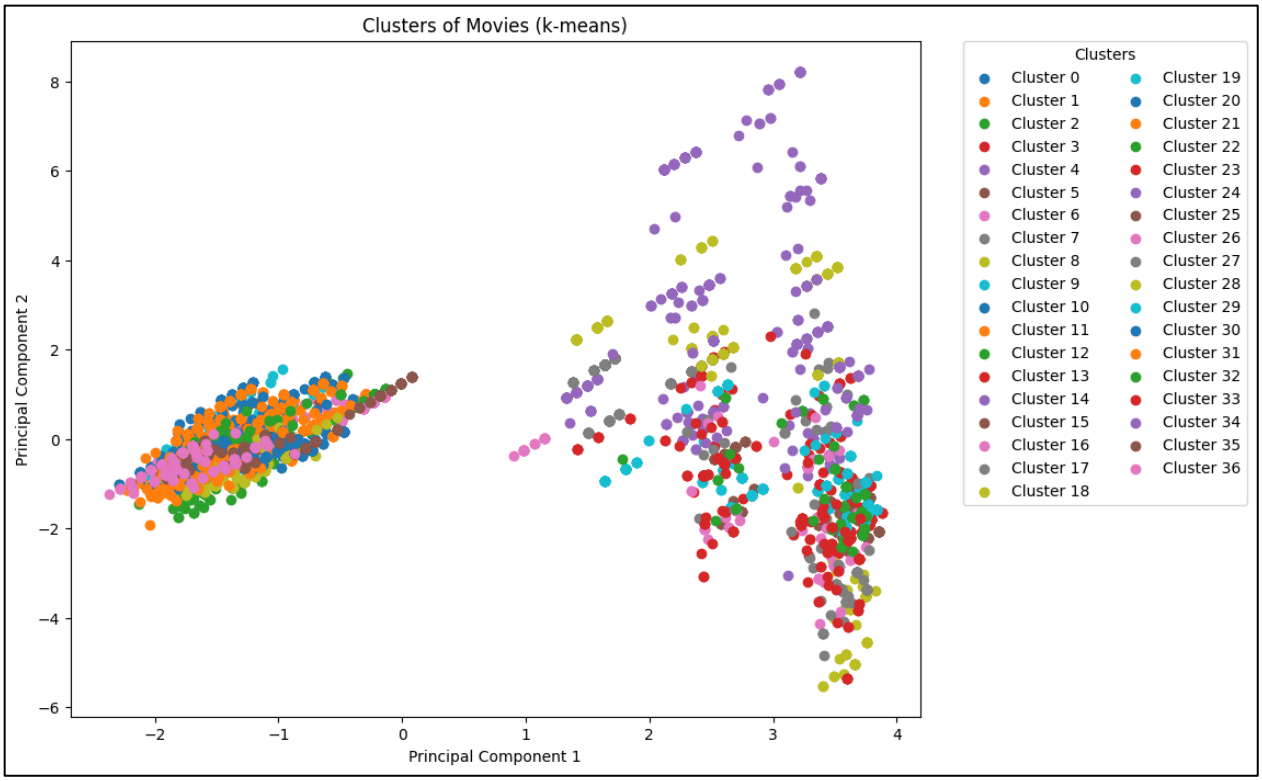
Completeness: 1.0000000000000002

V-measure: 0.5256266647739224

The Homogeneity, Completeness, and V-measure metrics evaluate the quality of the clustering by comparing it to ground truth labels.

Visualization:

To gain insights into the clustering results, we used Principal Component Analysis (PCA) to reduce the feature dimensions to 2 for visualization purposes. The below Image displays a scatter plot of the movies, colored by their assigned cluster labels, revealing the natural groupings identified by the K-Means algorithm.



2. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

We employed the BIRCH algorithm from scikit-learn to try out one more Clustering Algorithm. This hierarchical clustering technique is designed to efficiently handle large datasets by constructing a tree-like structure, making it well-suited for our extensive Netflix movie catalog.

Similar to the K-Means approach, we extracted the movie titles and features from the dataset and scaled the features using StandardScaler to ensure equal contribution across dimensions during the clustering process.

Model Training:

The BIRCH algorithm was trained on the scaled movie features, with a branching factor of 50 and a threshold of 0.5. These hyperparameters control the size and compactness of the clusters, respectively. BIRCH builds a tree-like representation of the data, iteratively assigning movies to clusters and refining the cluster boundaries until convergence.

Cluster Assignment:

After training, the BIRCH model assigned a cluster label to each movie in the dataset, enabling us to group similar titles together based on their intrinsic attributes.

Recommendation Function:

Analogous to the K-Means implementation, we developed a function to recommend similar movies given a target movie title. The steps involved are:

1. Locate the target movie in the dataset and extract its features and cluster assignment.
2. Identify all movies belonging to the same cluster as the target movie.
3. Calculate the cosine similarity between the target movie's features and all other movies within the cluster.
4. Return the top N most similar movies based on the highest cosine similarity scores.

```
[73] movie_name = input()
     similar_movies = birch_method_movies(movie_name, data, birch, scaler)
     print("Similar movies to", movie_name, ":")
     for movie in similar_movies:
         print("-", movie)

     Kota Factory
     Similar movies to Kota Factory :
     - Over Christmas
     - The Hook Up Plan
     - Little Things
```

Hyperparameter Tuning:

We tried tuning the model with threshold of 0.5 and 0.8. In both the cases the model performed very similarly. Hence we moved on with threshold = 0.5.

Evaluation Metrics:

To assess the clustering performance, we calculated several evaluation metrics:

Silhouette Score: 0.8563415349789212

The Silhouette Score measures how well each data point fits into its assigned cluster compared to other clusters. A higher value indicates better clustering, with 1 being the highest possible score. The obtained Silhouette Score of 0.856 suggests that the data points (movies) are well-matched to their respective clusters and well-separated from other clusters, indicating good clustering quality.

Calinski-Harabasz Index: 8338.35904429055

The Calinski-Harabasz Index measures the ratio of between-cluster dispersion and inter-cluster dispersion. A higher value indicates denser and well-separated clusters. The obtained value of 8338.359 is considered very high, suggesting that the clusters are dense and well-separated from each other, indicating excellent clustering performance.

Davies-Bouldin Index: 0.1402481325852651

The Davies-Bouldin Index measures the average similarity between clusters, with lower values indicating better separation. The obtained value of 0.140 is considered very low, suggesting that the clusters are well-separated and have minimal overlap or similarity between them.

Homogeneity: 0.6261554343809398

Completeness: 1.0000000000000002

V-measure: 0.7701052693272346

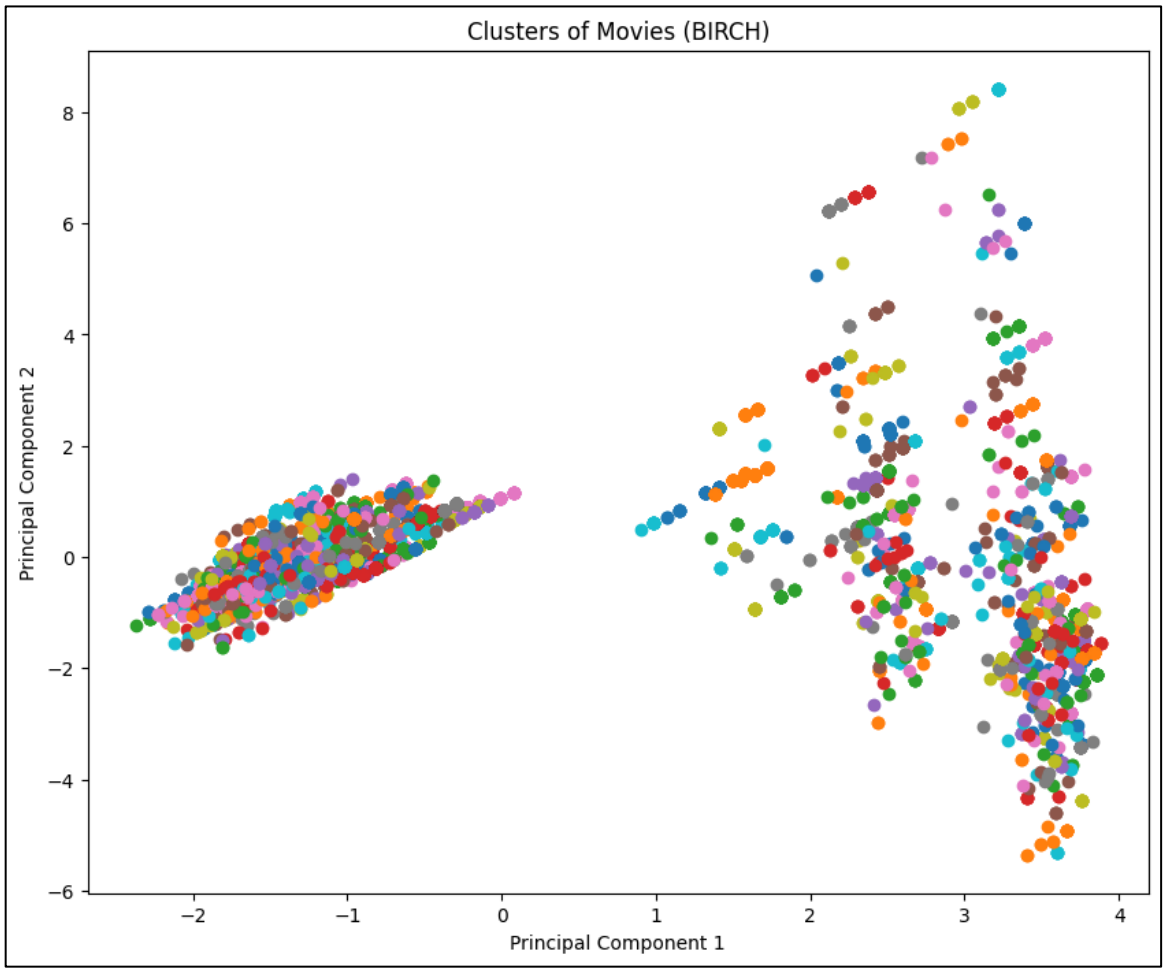
The Homogeneity, Completeness, and V-measure metrics evaluate the clustering quality by comparing it to ground truth labels. Homogeneity measures the extent to which each cluster contains only data points from a single ground truth class, while Completeness measures the extent to which all data points from a given ground truth class are assigned to the same cluster.

The obtained Homogeneity score of 0.626 and Completeness score of 1.0 suggest that the clustering has moderately high homogeneity (each cluster contains mostly data points from a single class) and perfect completeness (all data points from a given class are assigned to the same cluster). The V-measure, which

is the harmonic mean of Homogeneity and Completeness, has a value of 0.770, indicating overall good clustering quality when compared to the ground truth labels.

Visualization:

For visual interpretation of the BIRCH clustering results, we employed Principal Component Analysis (PCA) to reduce the feature dimensions to 2 for plotting purposes. The below displayed image displays a scatter plot of the movies, colored by their assigned cluster labels. This visualization reveals the natural groupings identified by the BIRCH algorithm, enabling us to observe the cluster structures and potential outliers visually.



3. Mean-shift Clustering

We further delved into unsupervised learning by implementing the Mean Shift Clustering algorithm from scikit-learn. This density-based technique aims to discover natural groupings within the data without requiring prior knowledge of the number of clusters.

Similar to our previous approaches, we extracted the relevant movie titles and features from our dataset. To ensure equal contribution across dimensions, we scaled the features using the StandardScaler from scikit-learn.

Model Training:

The Mean Shift algorithm was trained on the scaled movie features, with the bandwidth parameter automatically estimated using the estimate_bandwidth function from scikit-learn. This hyperparameter controls the radius of the neighborhood considered for each data point during the clustering process. Mean Shift iteratively updates the centroids by shifting them towards the mean of the nearby data points until convergence, naturally forming dense clusters.

Cluster Assignment:

After training, the Mean Shift model assigned a cluster label to each movie in the dataset, enabling us to group similar titles together based on their intrinsic attributes.

Recommendation Function:

We developed a function to recommend similar movies given a target movie title, following these steps:

1. Locate the target movie in the dataset and extract its features and cluster assignment.
2. Identify all movies belonging to the same cluster as the target movie.
3. Calculate the cosine similarity between the target movie's features and all other movies within the cluster.
4. Return the top N most similar movies based on the highest cosine similarity scores.

```
[84] movie_name = input()
     similar_movies = recommend_similar_movies(movie_name, data, mean_shift, scaler)
     print("Similar movies to", movie_name, ":")
     for movie in similar_movies:
         print("-", movie)

     Kota Factory
     Similar movies to Kota Factory :
     - Love Family
     - Love & Anarchy
     - Mismatched
```

Hyperparameter Tuning

The bandwidth is the parameter that needs to be tweaked for this model. We have used quantiles of 0.5 and got Silhouette Score of 0.356 which is not a good one and then we tried with 0.2 and got a Silhouette Score of 0.33 which is not improvement and finally decided to rack the mean shift algorithm as it is not. performing well compared to other algorithms

Evaluation Metrics:

To assess the clustering performance, we calculated several evaluation metrics:

Silhouette Score: 0.33172586111336

The Silhouette Score measures the degree of clustering, with values near 1 indicating well-clustered data, and values near -1 indicating poorly clustered data. The obtained score of 0.332 suggests a moderate degree of clustering.

Calinski-Harabasz Index: 121.24816838373113

The Calinski-Harabasz Index measures the ratio of between-cluster dispersion and inter-cluster dispersion, with higher values indicating better-defined clusters. The obtained value suggests a moderately good clustering performance.

Davies-Bouldin Index: 0.5884984442359018

The Davies-Bouldin Index measures the average similarity between clusters, with lower values indicating better separation. The obtained value of 0.588 suggests a moderate level of separation between clusters.

Homogeneity: 0.11081869347405211

The Homogeneity score measures the extent to which each cluster contains only data points from a single ground truth class, with a score of 1 indicating perfect homogeneity. The obtained value of 0.110 suggests a relatively low homogeneity.

Completeness: 1.0000000000000002

The Completeness score measures the extent to which all data points from a given ground truth class are assigned to the same cluster, with a score of 1 indicating perfect completeness. The obtained value of 1.0 suggests perfect completeness.

V-measure: 0.19952484529646539

The V-measure is the harmonic mean of Homogeneity and Completeness, providing an overall measure of clustering quality when compared to ground truth labels. The obtained value of 0.200 suggests a moderate level of clustering quality.

Visualization:

For visual interpretation of the Mean Shift clustering results, we used Principal Component Analysis (PCA) to reduce the feature dimensions to 2 for plotting. The below image display scatter plots of the movies, colored by their assigned cluster labels. These visualizations reveal the natural groupings identified by the Mean Shift algorithm, enabling us to observe the cluster structures and potential outliers visually.

4. Gaussian Mixture Model.

We employed another powerful unsupervised learning technique, the Gaussian Mixture Model (GMM) from scikit-learn, to cluster our movie dataset. GMM is a probabilistic model that assumes the data points are generated from a mixture of Gaussian distributions, each representing a distinct cluster.

Similar to our previous approaches, we extracted the relevant movie titles and features from our dataset. To ensure equal contribution across dimensions, we scaled the features using the StandardScaler from scikit-learn.

Model Training:

The GMM algorithm was trained on the scaled movie features, with the number of components (clusters) set to 5 for simplicity (Image). However, in practice, the optimal number of components can be determined using methods like the Bayesian Information Criterion (BIC) or other techniques.

The GMM model fits the data by estimating the parameters (means, covariances, and mixing proportions) of the Gaussian distributions that best describe the data. This iterative process assigns soft cluster memberships to each data point, allowing movies to belong to multiple clusters with varying probabilities.

Cluster Assignment:

After training, the GMM model assigned a cluster label to each movie in the dataset, enabling us to group similar titles together based on their intrinsic attributes.

Recommendation Function:

We developed a function to recommend similar movies given a target movie title, following these steps:

1. Locate the target movie in the dataset and extract its features and cluster assignment.
2. Identify all movies belonging to the same cluster as the target movie.
3. Calculate the cosine similarity between the target movie's features and all other movies within the cluster.
4. Return the top N most similar movies based on the highest cosine similarity scores. Evaluation

```
[92] movie_name = input()
     similar_movies = recommend_similar_movies(movie_name, data, gmm, scaler)
     print("Similar movies to", movie_name, ":")
     for movie in similar_movies:
         print("-", movie)

     Kota Factory
     Similar movies to Kota Factory :
     - The Womanizer
     - Easy Fortune Happy Life
     - You're My Destiny
```

Hyperparameter Tuning:

Number of clusters is the parameter. We tried n = 10 and S Score is 0.17 and when tried with n = 5 which obtained 0.089 which is really bad so we decide that GGM is not a good algorithm for our model

Evaluation Metrics:

To assess the clustering performance, we calculated several evaluation metrics

Silhouette Score: 0.0799475670989208

The Silhouette Score measures how well each data point fits into its assigned cluster compared to other clusters. The obtained score of 0.080 is relatively low, suggesting that the clustering quality is not optimal.

Calinski-Harabasz Index: 412.6398822244838

The Calinski-Harabasz Index measures the ratio of between-cluster dispersion and inter-cluster dispersion. The obtained value suggests a moderate level of clustering performance.

Davies-Bouldin Index: 3.1118860263173906

The Davies-Bouldin Index measures the average similarity between clusters, with lower values indicating better separation. The obtained value of 3.112 is relatively high, suggesting that the clusters may have significant overlap or similarity.

Homogeneity: 0.1589385772943442

The Homogeneity score measures the extent to which each cluster contains only data points from a single ground truth class. The obtained value of 0.159 suggests relatively low homogeneity.

Completeness: 0.9985398799996636

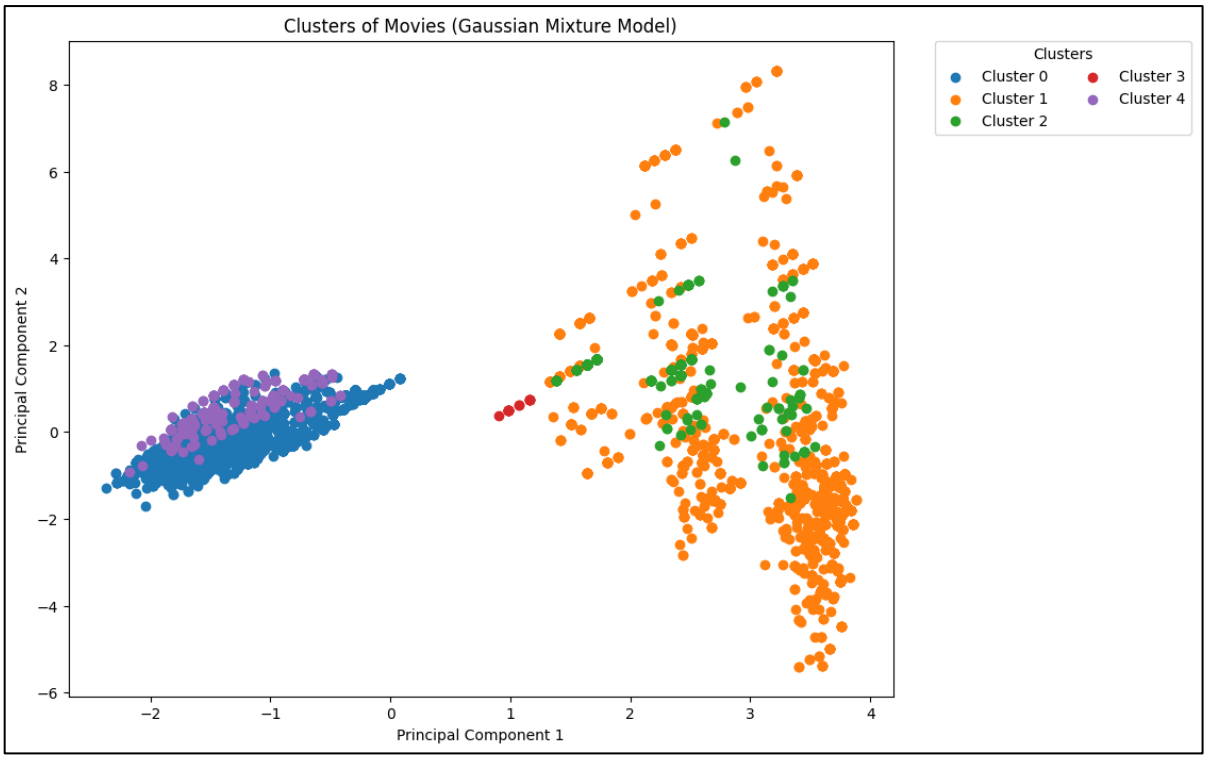
The Completeness score measures the extent to which all data points from a given ground truth class are assigned to the same cluster. The obtained value of 0.999 suggests near-perfect completeness.

V-measure: 0.27422801158622195

The V-measure is the harmonic mean of Homogeneity and Completeness, providing an overall measure of clustering quality when compared to ground truth labels. The obtained value of 0.274 suggests a moderate level of clustering quality.

Visualization:

The below image displays a scatter plot of the movies, colored by their assigned cluster labels, obtained from the GMM clustering. This visualization reveals the natural groupings identified by the algorithm, enabling us to observe the cluster structures and potential overlaps visually. By employing the Gaussian Mixture Model approach, we aimed to leverage the algorithm's ability to capture the underlying Gaussian distributions within our movie dataset, providing an alternative probabilistic perspective on grouping similar titles based on their inherent attributes.



5. Agglomerative Clustering

We further explored unsupervised learning techniques by implementing Agglomerative Clustering from scikit-learn. This hierarchical clustering algorithm starts with each data point as an individual cluster and iteratively merges the closest clusters based on their similarity until a desired number of clusters is reached.

Similar to our previous approaches, we extracted the relevant movie titles and features from our dataset. To ensure equal contribution across dimensions, we scaled the features using the StandardScaler from scikit-learn.

Model Training:

The Agglomerative Clustering algorithm was trained on the scaled movie features, with the number of clusters set to 5 for simplicity. However, in practice, the optimal number of clusters can be determined using techniques like the dendrogram or other methods.

The algorithm starts with each movie as a separate cluster and iteratively merges the closest clusters based on a chosen linkage criterion, such as Ward's method, which minimizes the variance within clusters. This process continues until the desired number of clusters is reached.

Cluster Assignment:

After training, the Agglomerative Clustering model assigned a cluster label to each movie in the dataset, enabling us to group similar titles together based on their intrinsic attributes.

Recommendation Function:

We developed a function to recommend similar movies given a target movie title, following these steps:

1. Locate the target movie in the dataset and extract its features and cluster assignment.
2. Identify all movies belonging to the same cluster as the target movie.
3. Calculate the cosine similarity between the target movie's features and all other movies within the cluster.
4. Return the top N most similar movies based on the highest cosine similarity scores.

```
[100] movie_name = input()
      similar_movies = recommend_similar_movies(movie_name, data, agglomerative, scaler)
      print("Similar movies to", movie_name, ":")
      for movie in similar_movies:
          print("-", movie)

      Kota Factory
      Similar movies to Kota Factory :
      - Over Christmas
      - The Womanizer
      - College Romance
```

Hyperparameter Tuning:

In this also we used n =10 and found out the S score as 0.157 and then tried n = 5 which obtained 0.116. Which says that this model is not goof for our project.

Evaluation Metrics:

To assess the clustering performance, we calculated several evaluation metrics:

Silhouette Score: 0.12011407638320616

The Silhouette Score measures how well each data point fits into its assigned cluster compared to other clusters. The obtained score of 0.120 is relatively low, suggesting that the clustering quality could be improved.

Calinski-Harabasz Index: 411.0624965014305

The Calinski-Harabasz Index measures the ratio of between-cluster dispersion and inter-cluster dispersion. The obtained value suggests a moderate level of clustering performance.

Davies-Bouldin Index: 1.9578236289632716

The Davies-Bouldin Index measures the average similarity between clusters, with lower values indicating better separation. The obtained value of 1.958 suggests that the clusters have a moderate level of separation.

Homogeneity: 0.1015282769899466

The Homogeneity score measures the extent to which each cluster contains only data points from a single ground truth class. The obtained value of 0.102 suggests relatively low homogeneity.

Completeness: 1.0000000000000016

The Completeness score measures the extent to which all data points from a given ground truth class are assigned to the same cluster. The obtained value of 1.0 suggests perfect completeness.

V-measure: 0.18434075476915465

The V-measure is the harmonic mean of Homogeneity and Completeness, providing an overall measure of clustering quality when compared to ground truth labels. The obtained value of 0.184 suggests a relatively low level of clustering quality.

Visualization:

The below image displays a scatter plot of the movies, colored by their assigned cluster labels, obtained from the Agglomerative Clustering algorithm. This visualization reveals the natural groupings identified by the algorithm, enabling us to observe the cluster structures and potential overlaps visually.

6. Density Based Spatial Clustering of Applications with Noise (DBSCAN)

We explored the DBSCAN algorithm, a density-based clustering technique, to group similar movie titles based on their inherent features. Unlike the Agglomerative Clustering approach, DBSCAN does not require specifying the number of clusters in advance and can identify clusters of arbitrary shape and size.

Similar to previous approaches, we extracted the relevant movie titles and features from our dataset. To ensure equal contribution across dimensions, we scaled the features using the StandardScaler from scikit-learn.

Model Training:

The DBSCAN algorithm was trained on the scaled movie features, with the key parameters being epsilon (eps) and min_samples. The epsilon value determines the maximum distance between two points for them to be considered neighbors, while min_samples specify the minimum number of neighboring points required to form a dense region.

Determining the optimal epsilon value is crucial for DBSCAN's performance. For simplicity, we assumed an epsilon value of 0.5, but in practice, techniques like the k-nearest neighbors approach or visualizing the nearest neighbor distances can help identify the most suitable value.

Cluster Assignment:

After training, the DBSCAN model assigned a cluster label to each movie in the dataset, enabling us to group similar titles together based on their density-based clustering. Points that did not belong to any dense region were labeled as noise (-1).

Recommendation Function:

We developed a function to recommend similar movies given a target movie title, following these steps:

1. Locate the target movie in the dataset and extract its features and cluster assignment.
2. Identify all movies belonging to the same cluster as the target movie.
3. Calculate the cosine similarity between the target movie's features and all other movies within the cluster.
4. Return the top N most similar movies based on the highest cosine similarity scores.

```
[109] movie_name = input()
      similar_movies = recommend_similar_movies(movie_name, data, dbscan, scaler)
      print("Similar movies to", movie_name, ":")
      for movie in similar_movies:
          print("-", movie)

      Kota Factory
      Similar movies to Kota Factory :
      - Miss Rose
      - Queen of No Marriage
      - The Smart Money Woman
```

Hyperparameter Tuning:

Here we tuned two parameters epsilon = 0.2 and min_samples = 7 and got s score 0.62 which is considerably good and then used epsilon = 0.5 and min_samples =5 and got s score = 0.70 which is good improvement.

Evaluation Metrics:

To assess the clustering performance, we calculated several evaluation metrics:

Silhouette Score: 0.7000671837365576

The Silhouette Score measures how well each data point fits into its assigned cluster compared to other clusters. The obtained score of 0.700 suggests a relatively good level of clustering quality.

Calinski-Harabasz Index: 46.220060079925915

The Calinski-Harabasz Index measures the ratio of between-cluster dispersion and inter-cluster dispersion. The obtained value suggests a moderate level of clustering performance.

Davies-Bouldin Index: 1.4835617813060504

The Davies-Bouldin Index measures the average similarity between clusters, with lower values indicating better separation. The obtained value of 1.484 suggests a moderate level of cluster separation.

Homogeneity: 0.535590927711385

The Homogeneity score measures the extent to which each cluster contains only data points from a single ground truth class. The obtained value of 0.536 suggests a moderate level of homogeneity.

Completeness: 1.0000000000000004

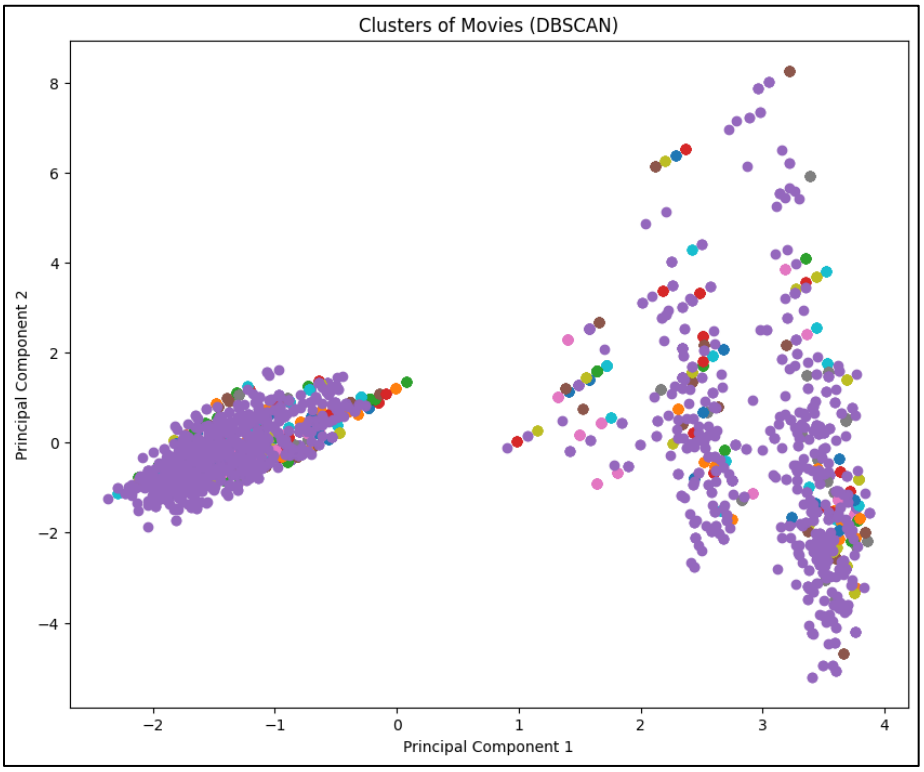
The Completeness score measures the extent to which all data points from a given ground truth class are assigned to the same cluster. The obtained value of 1.0 suggests perfect completeness.

V-measure: 0.6975697994121642

The V-measure is the harmonic mean of Homogeneity and Completeness, providing an overall measure of clustering quality when compared to ground truth labels. The obtained value of 0.698 suggests a moderate to good level of clustering quality.

Visualization:

The below image displays a scatter plot of the movies, colored by their assigned cluster labels, obtained from the DBSCAN algorithm. This visualization reveals the natural groupings identified by the algorithm based on density, enabling us to observe the cluster structures, potential overlaps, and noise points visually.



TRAINING CONCLUSION

The BIRCH algorithm performed the best here, demonstrating excellent clustering quality with a high Silhouette Score of 0.856, large Calinski-Harabasz Index, and low Davies-Bouldin Index. It effectively created dense, well-separated clusters while maintaining high homogeneity and completeness, as corroborated by the visual representations.

DBSCAN also exhibited a good performance, with a Silhouette Score of 0.700 and a V-measure of 0.698, indicating a moderate to good clustering quality. Its density-based approach allowed for identifying arbitrary-shaped clusters and handling noise points effectively.

K-Means Clustering, despite its simplicity, demonstrated reasonable performance with a Silhouette Score of 0.446 and a Calinski-Harabasz Index of 935.845, providing a solid baseline for comparison.

The Mean Shift, Gaussian Mixture Model, and Agglomerative Clustering algorithms exhibited relatively lower clustering quality, suggesting they may not have been as effective in capturing the underlying structures within the movie dataset.

In conclusion, based on the comprehensive evaluation and analysis, we can rank the clustering algorithms in the following order of performance for the Netflix movie recommendation system:

1. BIRCH
2. DBSCAN
3. K-Means Clustering
4. Mean Shift Clustering
5. Gaussian Mixture Model
6. Agglomerative Clustering

MODEL NAME	**Silhouette Score**	**Calinski-Harabasz Index**	**Davies-Bouldin Index**	**Homogeneity**	**Completeness**	**V-measure**
K-means Clustering	0.42	843.43	0.9534	0.346	1.00	0.514
BIRCH	0.856	8338.35	0.1402	0.6261	1.000	0.770
Mean – Shift Clustering	0.331	121.24	0.588	0.1108	1.0	0.199
Gaussian Mixture Model	0.171	383.23	2.292	0.232	1.000	0.377
Agglomerative Clustering	0.157	393.434	2.07	0.1967	1.0000	0.328
(DBSCAN).	0.700	46.22	1.483	0.5355	1.00	0.6975

PHASE 3

Considered Model: Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH)

WEB UI

USED TECHNOLOGIES : HTML, CSS, JavaScript, Flask

First we have created a joblib file of the BIRCH Model and then used it in the back end code.

BACKEND DESCRIPTION:

1. Imports and Flask App Initialization:

- We import the necessary modules: Flask, render_template, send_from_directory, request, jsonify from Flask, load from joblib, numpy, pickle, StandardScaler from sklearn.preprocessing, cosine_similarity from sklearn.metrics.pairwise, pandas, and csv.
- It initializes a Flask application instance (app).

2. Loading Data and Model:

- The code loads data from a pickle file (data.pickle) and assigns it to the variable data.
- It loads a machine learning model from a joblib file (model.joblib) and assigns it to the variable model.
- It creates an instance of StandardScaler and assigns it to the variable scaler.
- It performs feature scaling on the movie features using scaler.fit_transform(movie_features) and assigns the scaled features to movie_features_scaled.
- It adds a new column 'cluster' to the data DataFrame, which contains the cluster labels from the loaded model.

3. Helper Function: birch_method_movies:

- This function takes a movie name, the data DataFrame, the loaded model, the scaler object, and an optional top_n parameter for the number of similar movies to return.
- It converts the movie name to lowercase.

- It finds the matching movie in the data DataFrame based on the title.
- If no matching movie is found, it returns "No such movie found".
- It extracts the movie features and scales them using the scaler object.
- It predicts the cluster for the given movie using the loaded model.
- It finds all movies in the same cluster as the given movie.
- It calculates the cosine similarity between the given movie's features and the features of other movies in the same cluster.
- It returns the titles of the top_n most similar movies.

4. Routes:

- The @app.route('/') route renders the index.html template.
- The @app.route('/eda') route renders the EDA.html template and passes a list of image paths (images) to be displayed.
- The @app.route('/movies.csv') route serves the movies.csv file from the server's root path.
- The @app.route('/results', methods=('POST', 'GET')) route handles the movie recommendation functionality:
- If the request is a GET request, it retrieves the movie name from the query string (request.args.get('movie')).
- It calls the birch_method_movies function with the provided movie name and other necessary parameters.
- If the function returns a string (indicating an error), it renders the result.html template with the error message.
- If the function returns a list of movie titles, it reads the movie details from the movies.csv file and renders the result.html template with the movie details.
- The @app.route('/image/<int:image_index>') route renders the full_screen.html template and passes an image URL based on the provided image_index.

5. Running the App:

- If the script is run directly (not imported as a module), the Flask app is run with debugging enabled (app.run(debug=True)).

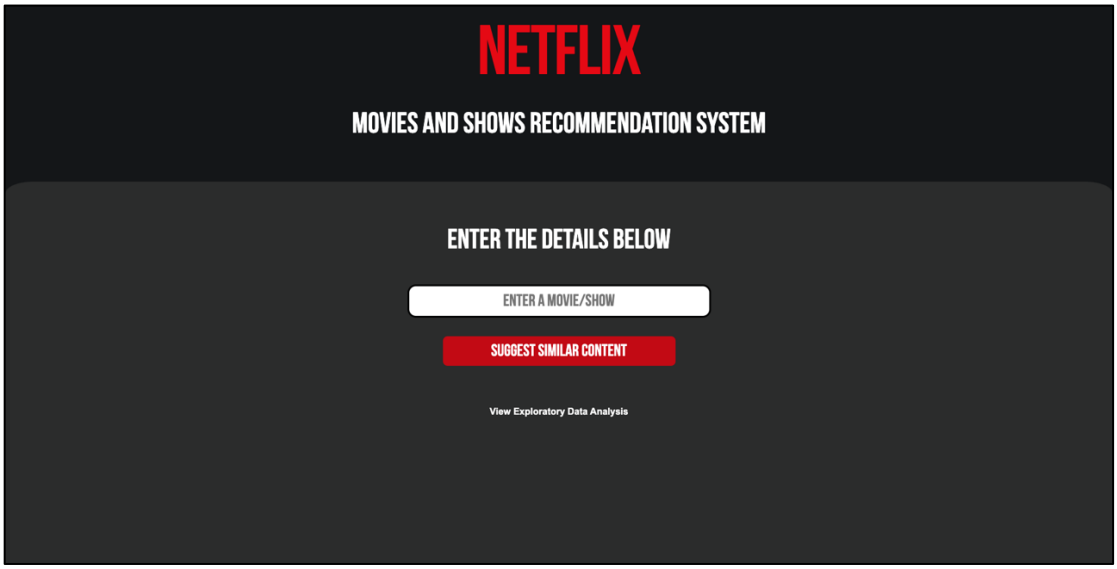
This Flask application serves a web interface for exploring movie recommendations based on a machine learning model. Users can enter a movie name, and the application will recommend similar movies based on the provided model and data. Additionally, it provides routes for rendering static HTML templates and serving image assets.

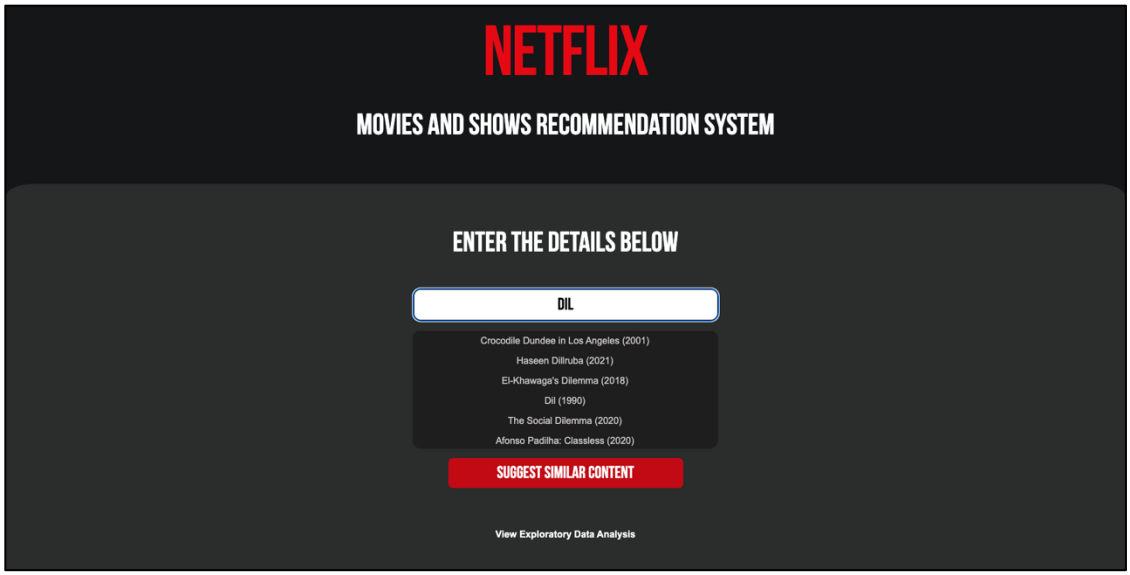
FRONT END DESCRIPTION:

HTML:

1. Index.html

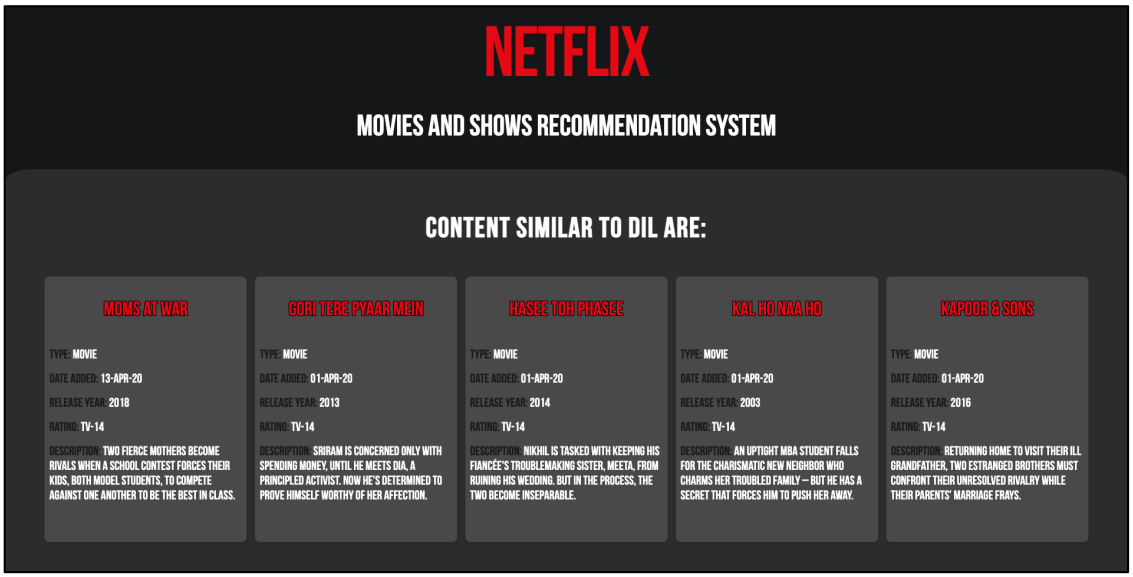
- This page shows the main page for the Netflix movie and show recommendation system.
- In the <body> section there is a container div ( ). Inside the container, there are two major sections:
- A base layer ( ) with the title and subtitle.
- The second layer ( ) contains a form (<form action="/results" method="get">) which sends data to the "/results" route using the GET method.
- User can enter a name of any movie or show and can get the related movies/shows.

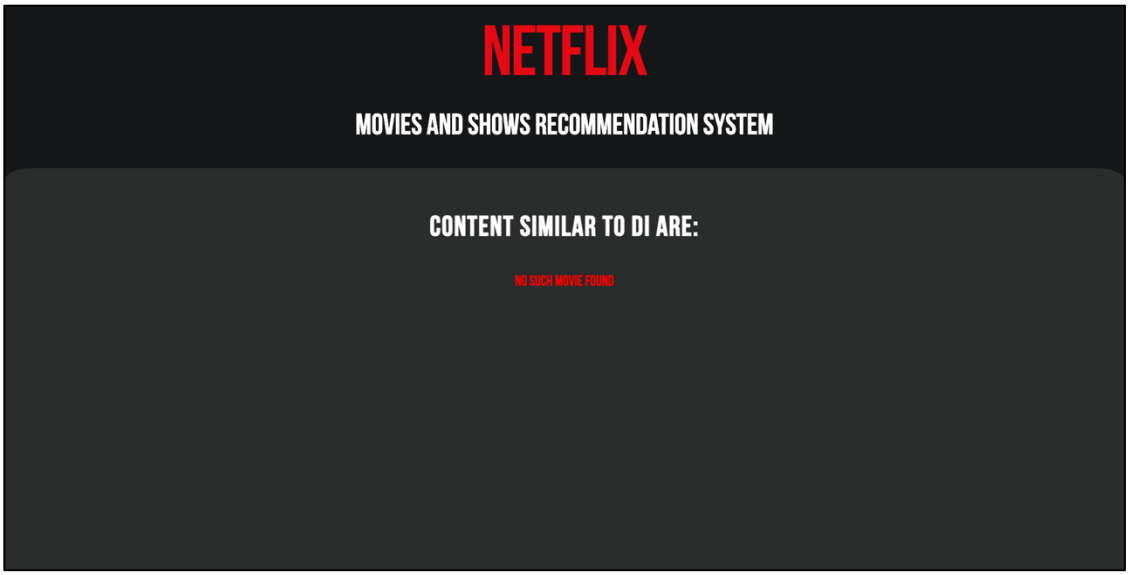




2. Results.html

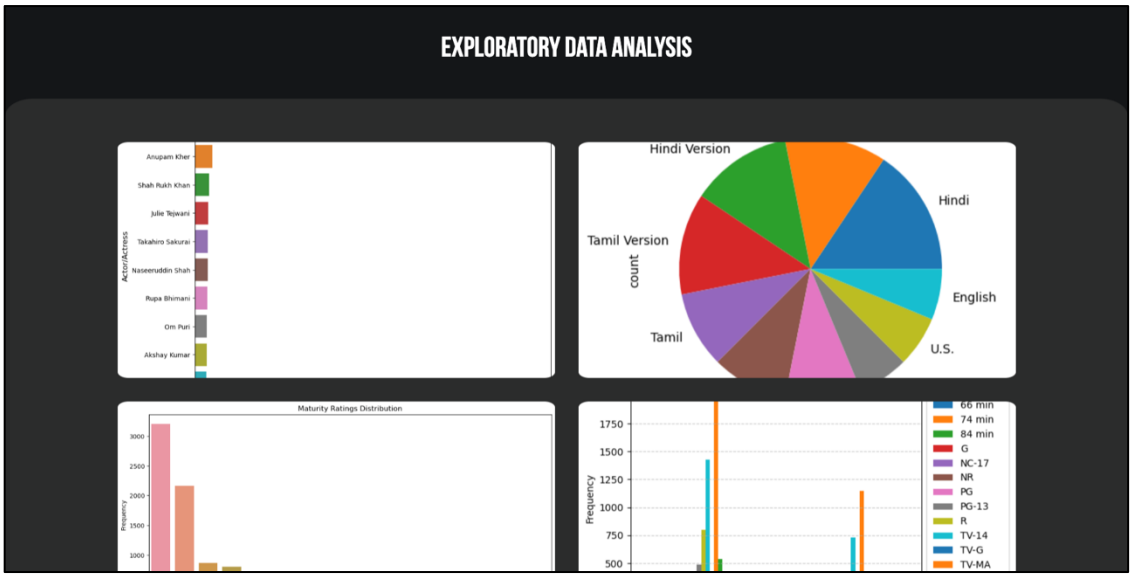
- This file renders the movie recommendation results page.
- If movie recommendations are available, it renders a grid of movie containers, each containing movie details like title, type, date added, release year, rating, and description.
- If no recommendations are found, it displays an error message.

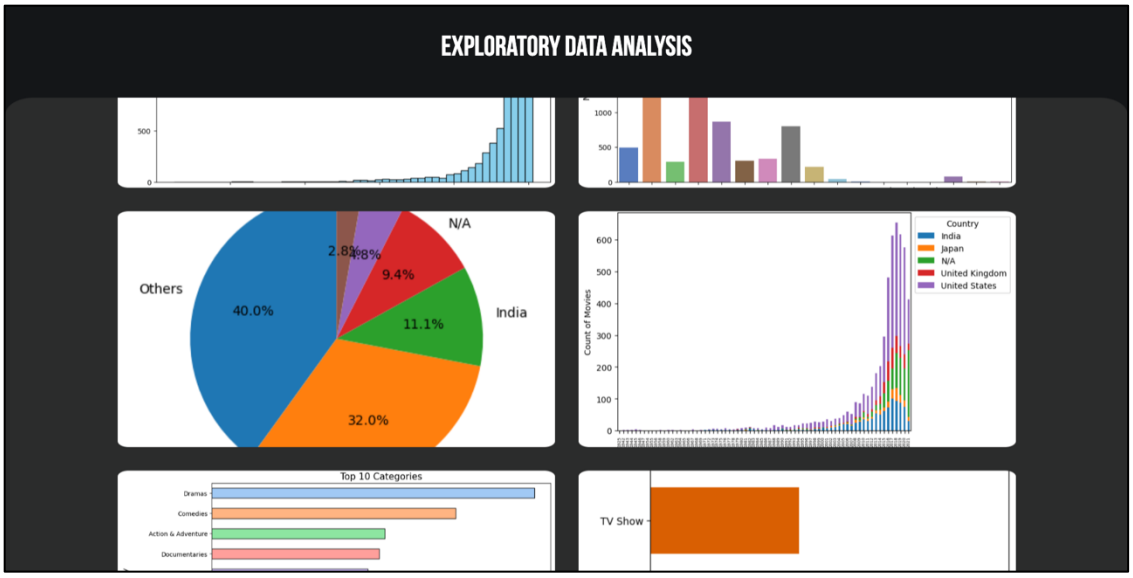




3. EDA.html

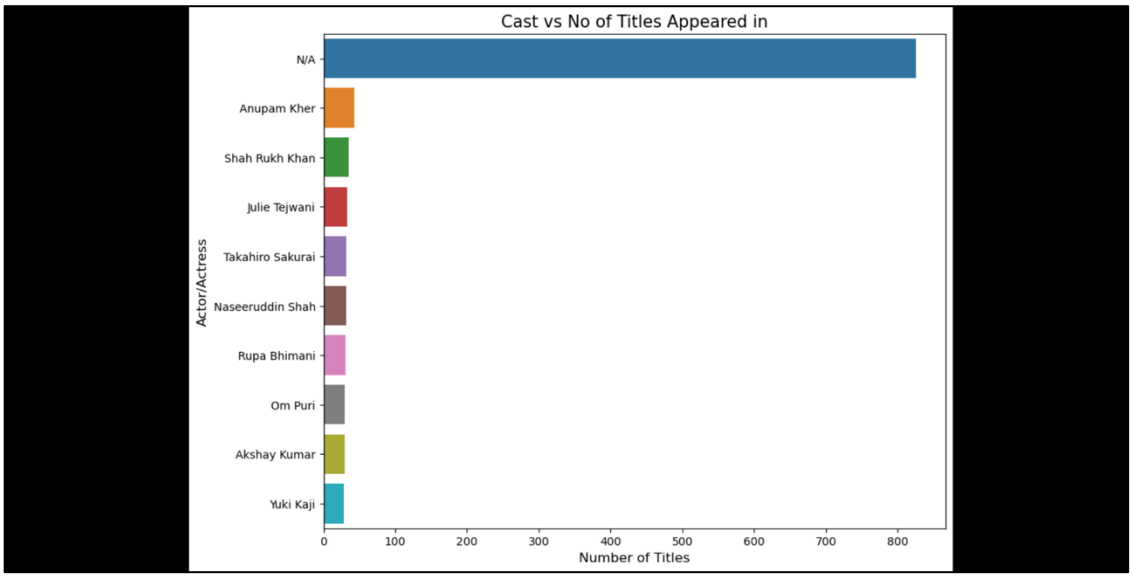
- This file renders the Exploratory Data Analysis (EDA) page.
- It contains gallery section for the Exploratory Data Analysis Images.
- The gallery section displays a grid of images, with each image being a link that opens the image in a full-screen view (/image/<loop.index>).
- The images are dynamically rendered using a Jinja2 template loop ({% for image in images %}).





4. Full_screen.html

- This file renders a full-screen view of an image.
- It displays the selected image in full-screen mode with a black background.
- The image source is dynamically set using a Jinja2 template variable ({{ url_for('static', filename=image) }}).



CSS:

Font import

- We imported a custom font named "CustomFont" from a local file (BebasNeue.ttf).

Global Styles:

- It adds 0 margin to the body element.

Container Styles:

- It sets up a relative positioning context using the class .container-1 for its child elements.
- It sets the background and padding for the layer using the class .base-layer.
- It sets the background, padding, and padding-bottom of another layer using the class. f-layer.
- It sets styles of a layer positioned absolutely, with a background, border-radius, positioning, dimensions, padding, and overflow using the class. s-layer.

Typography:

- It styles the main title with the color, font family, size, weight, alignment, letter-spacing, and margin using the class .title.
- It styles a subtitle using the class .sub-title with color, font family, size, weight, alignment, letter-spacing, padding, and vertical positioning.
- It styles another subtitle using the class. subtitle with color, font family, size, weight, and letter-spacing.

Responsive Styles:

- It includes media queries for different screen widths, adjusting the positioning and dimensions of certain elements to ensure responsive behavior.

Form Styles:

- It styles the input[type="text"] elements with padding, width, border, border-radius, font family, font size, and text alignment.
- It styles the next button with background color, color, border, padding, border-radius, font size, cursor, and font family using the class. next.
- It styles a form section using the class with settings to display, flex-direction, and alignment for form sections.

Suggestion List Styles:

- It styles the appearance of a suggestion list, including list-style-type, alignment, padding, maximum height, width, overflow behavior, background color, and border-radius for the ID:suggestions.
- It styles the appearance of the individual.

Styles:

- The styles for .gallery img define the style of images inside the gallery by setting the width, height, object-fit, and border-radius.
- The styles for .gallery img:hover define the hover effect for the gallery image.

JAVASCRIPT

The JavaScript code has only one functionality:

Movie Suggestion Functionality:

- This code listens for user input in the movie search input field.
- When the user types at least three characters, it fetches the movies.csv file from the server.
- It parses the CSV data to extract movie titles and years.
- It filters the movie titles based on the user's input and displays the matching titles as suggestions in a list.
- When the user clicks on a suggestion, the selected title is populated in the input field.
- This functionality helps users find and select movies more easily.

STEPS TO RUN THE WEB APP

Step 1:

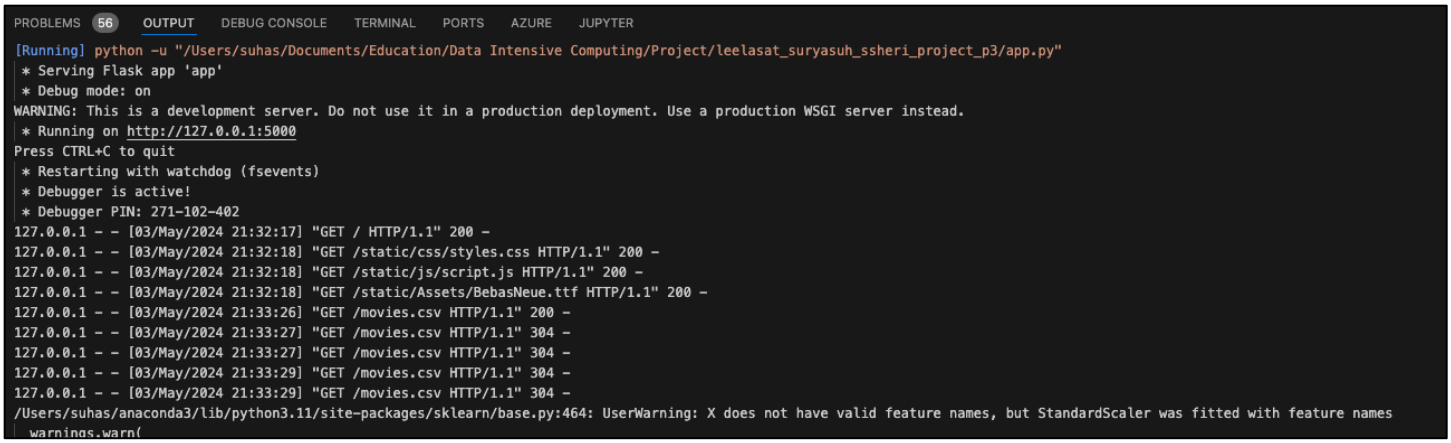
First, we need to create the joblib files and the pickle files that are required for the running of the model in the flask application. These files include data.pickle and model.joblib files.

```
[73]
import joblib

joblib.dump(model, 'model.joblib')
joblib.dump(scaler, 'scaler.joblib')

['scaler.joblib']

[74]
data = meta.copy()

[75]
print(data)

      type_encoded                 title  rating  Action & Adventure  \
0                0   Dick Johnson Is Dead       4                   0
1                1          Blood & Water       8                   0
2                1              Ganglands       8                   0
3                1  Jailbirds New Orleans       8                   0
4                1           Kota Factory       8                   0
...            ...                    ...     ...                 ...
8802             0                 Zodiac       5                   0
8803             1            Zombie Dumb      11                   0
8804             0             Zombieland       5                   0
8805             0                   Zoom       3                   0
8806             0                 Zubaan       6                   0

      Anime Features  Anime Series  British TV Shows  \
0                  0             0                 0
1                  0             0                 0
2                  0             0                 0
3                  0             0                 0
4                  0             0                 0
...              ...           ...               ...
8802               0             0                 0
8803               0             0                 0
8804               0             0                 0
8805               0             0                 0
8806               0             0                 0
...
8805                  0         0             0             0         0
8806                  0         0             0             0         0

[8790 rows x 45 columns]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

[76]
import pickle
with open("data.pickle", "wb") as f:
    pickle.dump(data, f)

f.close()
```

Joblib is a library in Python that provides efficient tools for saving and loading large data structures, such as NumPy arrays and machine learning models, to and from disk. It is optimized for scientific computing and is commonly used in machine learning tasks.

Pickle is a built-in Python module that allows serialization and deserialization of Python objects, enabling the saving and loading of almost any Python object to and from disk or over a network connection.

Step 2: Running the flask app using terminal.



A URL "http://127.0.0.1:5000" appears. This means, our program started giving input to the server through the POST method. And the website can be accessed through this link.

Step 3: When we open the above URL in any browser locally, we can access the webpage. Then the user can enter the movie/series name in the text box. There is also a auto fill kind of suggestion feature, that helps the user to see whether the title is available in the dataset.

Step 4: When clicked on the Suggest Similar Content Button, the flask app fetches the results from the model and then the results are displayed on the results.html page.

Step 5: If the user wants to see the insights of the data, he/she can click on the view EDA link, they can see the EDA in the gallery.

RECOMMENDATIONS AND INSIGHTS FOR USERS

RECOMMENDATIONS:

1. **Explore Similar Content**:

   Encourage users to try out the recommended movies or TV shows that are similar to the one they entered. This can help them discover new content that aligns with their interests and preferences.

2. **Personalized Recommendations:**

   Highlight that the recommendations are personalized based on the user's input and the underlying data analysis. This can build trust in the system and encourage users to rely on the recommendations.

3. **Expand Interests:**

Recommend that users occasionally try content from different genres or categories than their usual preferences. This can help them discover new interests and broaden their viewing horizons.

INSIGHTS:

1. Popularity Trends:

Provide insights into the most popular movies or TV shows based on user ratings, views, or other engagement metrics. This can help users stay updated on trending content and popular titles.

2. Genre Analysis:

Share insights into the most popular genres or categories among users. This can help users understand the general preferences of the user base and potentially discover new genres to explore.

3. Demographic Preferences:

If available, share insights into how different demographic groups (age, gender, location, etc.) tend to prefer certain types of content. This can help users understand the diversity of preferences and potentially discover content popular among specific groups.

4. Release Year Patterns:

Analyze and share insights into how user preferences vary based on the release year of the content. This can help users understand if they tend to prefer newer releases or if they gravitate towards classic or retro content.

5. Data-driven Insights:

Highlight that the recommendations and insights are driven by data analysis and machine learning techniques. This can build credibility in the system and demonstrate the power of data-driven decision-making.

PROJECT EXTENSIONS AND ENHANCEMENTS:

1. Collaborative Filtering:

Implementing collaborative filtering techniques to incorporate user ratings and preferences into the recommendation algorithm. This would allow the system to provide recommendations based on similar users' preferences, in addition to content similarities.

2. **User Profiles**:

Developing user profiles that capture individual viewing histories, preferences, and ratings. This would enable more personalized recommendations tailored to each user's unique interests and tastes.

3. **Content-based Filtering**:

Enhance the current content-based filtering approach by incorporating additional content metadata, such as genre, director, cast, and plot summaries. This could improve the relevance and accuracy of recommendations.

4. **Hybrid Approach**:

Combine collaborative filtering and content-based filtering techniques to create a hybrid recommendation system. This approach can leverage the strengths of both methods and potentially provide more robust and accurate recommendations.

5. **Social Integration**:

Integrate social features that allow users to share recommendations, follow friends, and discover content based on their social connections' preferences.

6. **Interactive Interface**:

Enhance the user interface to provide more interactive and engaging experiences. This could include features like visual browsing, filtering options, and the ability to create personalized watchlists or queues.

CONCLUSION

In this project we developed a personalized recommendation system for Netflix movies and TV shows using unsupervised machine learning techniques. Various clustering algorithms like K-Means, BIRCH, Mean Shift, Gaussian Mixture Models, Agglomerative Clustering, and DBSCAN were explored and evaluated. BIRCH

outperformed others, exhibiting excellent clustering quality with high Silhouette Score and well-separated, dense clusters.

A user-friendly web application built with Flask, HTML, CSS, and JavaScript enabled users to input preferred titles and receive recommendations from the trained BIRCH model. The Exploratory Data Analysis section provided valuable insights into content preferences and trends.

While promising results were achieved, potential enhancements include incorporating collaborative filtering, developing user profiles, implementing hybrid approaches, social integration, interactive interfaces, and leveraging additional content metadata for improved recommendation accuracy and personalization.

REFERENCES

1. Dataset Link - https://huggingface.co/datasets/huggingfacelearners/netflix-shows
2. Pandas Documentation – https://pandas.pydata.org/docs/
3. NumPy Documentation – https://numpy.org/doc/#
4. Matplotlib Documentation – https://matplotlib.org/stable/
5. Scikit Learn Documentation – https://scikit-learn.org/stable/
6. Seaborn Documentation - https://seaborn.pydata.org/
7. K-Means Clustering - https://stanford.edu/~cpiech/cs221/handouts/kmeans.html
8. BIRCH Explained - https://medium.com/birch-explained/
9. Mean-shift Clustering Tutorial - https://www.geeksforgeeks.org/ml-mean-shift-clustering/
10. Gaussian Mixture Model - https://medium.com/gmm/
11. Agglomerative Clustering - https://medium.com/dagglomerative-clustering/
12. DBSCAN Tutorial - https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/
13. Flask Documentation - https://flask.palletsprojects.com/
14. HTML Documentation - https://developer.mozilla.org/en-US/docs/Web/HTML
15. CSS Documentation - https://developer.mozilla.org/en-US/docs/Web/CSS
16. JavaScript Documentation - https://developer.mozilla.org/en-US/docs/Web/JavaScript