1.)Data Collection & Exploration:

In this step, we loaded the Netflix dataset using the Pandas library and performed a basic inspection to understand its structure. We checked the shape of the dataset to see how many records and features it contains, and previewed the first few rows to get an idea of the data's content and layout. This serves as the foundation for all further data exploration and preprocessing.

2.)Data Preprocessing:

In this step, we handled missing values in key columns to clean the dataset and prepare it for analysis. For textual fields like 'director' and 'cast', we filled missing entries with the placeholder 'Unknown'. For categorical fields such as 'country' and 'rating', we used the most frequent value (mode) to fill in gaps. The 'date_added' column was forward-filled to propagate previous dates into missing rows. Finally, we verified that missing values were addressed by printing the updated null value counts for each column.

we extracted and separated numerical values and units from the 'duration' column, which originally contained strings like "90 min" or "1 Season". Using a custom function, we split the duration into two parts: a numerical duration (duration_int) and its type (duration_type, such as 'min' or 'Season'). If extraction failed, we assigned NaN and 'Unknown'. Missing numerical durations were then filled using the median value to maintain consistency. This transformation allows us to work with duration in a more structured, analyzable form for clustering.

we created a new feature called content_age by subtracting each show's release year from the current year (2025). This gives us the age of each movie or TV show, which can help reveal how recent or old the content is — a useful attribute for clustering and understanding content trends over time.

we converted categorical columns—'type', 'rating', and 'duration_type'—into numerical format using one-hot encoding. This process creates new binary columns for each category, allowing clustering algorithms to interpret categorical data. We used drop_first=True to avoid multicollinearity by dropping the first category from each set. The resulting DataFrame, df_encoded, is now ready for clustering with properly formatted numeric features.

we engineered a new feature called genre_count by counting the number of genres associated with each title in the 'listed_in' column. This was done by splitting the genre string at each comma and measuring the length of the resulting list. This feature provides

insight into the diversity or complexity of a show's genre classification, which can be useful for clustering similar content.

we applied TF-IDF vectorization to the 'listed_in' column, which contains genre information. The TF-IDF (Term Frequency–Inverse Document Frequency) technique transforms the textual data into a numerical format that reflects the importance of each genre term relative to others. After vectorization, we converted the result into a DataFrame called genre_tfidf_df, where each genre term becomes a feature with a corresponding weight. This enriched feature set can help clustering models better distinguish between content based on genre relevance.

we standardized the numerical features—'release_year', 'duration_int', 'content_age', and 'genre_count'—using StandardScaler, which scales the data to have a mean of 0 and standard deviation of 1. This ensures that all features contribute equally to the clustering process, preventing features with larger scales from dominating the distance calculations. The result was stored in a new DataFrame called scaled_df, which is now ready for use in clustering algorithms.

we assembled the final dataset (final_df) for clustering by combining three key components: the standardized numerical features, the one-hot encoded categorical features, and the TF-IDF genre features. Irrelevant or redundant columns like identifiers and raw text fields were dropped to ensure only meaningful features remained. This consolidated feature set now contains a fully numeric and well-preprocessed representation of the Netflix content, ready to be fed into clustering algorithms.

3. Feature Engineering:

In this step, we dynamically calculated the content_age of each show or movie by subtracting its release year from the current calendar year using Python's datetime module. This ensures the feature remains accurate no matter when the code is run. We then printed a few sample rows to verify the correctness of the new feature, showing the title, release year, and calculated age of the content.

Here, we added a new feature called genre_count by counting how many genres are listed for each title in the 'listed_in' column. This was done by splitting the genre string on commas and calculating the number of resulting items. We then displayed a few sample rows to confirm the feature was created correctly. This feature helps capture the genre diversity of each movie or show, which can be useful in clustering similar content.

In this step, we applied one-hot encoding to the categorical columns 'type', 'rating', and 'duration_type', converting each category into binary columns while dropping the first category of each to avoid multicollinearity. This transformation allows machine learning models to interpret categorical data numerically. Finally, we printed the newly added column names to verify the successful creation of one-hot encoded features.

4. Clustering Model Selection:

In this step, we selected all numerical columns from the encoded dataset and applied standardization using StandardScaler to ensure that each feature has a mean of 0 and a standard deviation of 1. This step is crucial for clustering algorithms like K-Means and DBSCAN, which are sensitive to feature scale. We then confirmed the shape of the standardized feature matrix to ensure it was ready for clustering.

In this step, we used the K-Means clustering algorithm to determine the optimal number of clusters (k) by evaluating performance over a range of values from 2 to 10. We calculated two key metrics: **Inertia**, which measures how tightly data points are grouped within a cluster (used in the Elbow Method), and **Silhouette Score**, which evaluates how well-separated the clusters are. These scores were plotted to visually assess the best k value for clustering Netflix content. This step is essential for choosing a meaningful number of clusters before applying K-Means.

In this step, we performed hierarchical clustering on a sample of 200 records from the standardized dataset to visualize the clustering structure using a dendrogram. We used the Ward linkage method, which minimizes the variance within clusters. The dendrogram helps us understand how data points group together at different distances and assists in estimating an appropriate number of clusters by identifying natural break points in the hierarchy. This visualization is particularly useful for exploring the data's internal structure before finalizing a clustering approach.

In this step, we applied DBSCAN, a density-based clustering algorithm, to identify natural groupings in the data without requiring a predefined number of clusters. Before applying DBSCAN, we reduced the dimensionality of the dataset to two principal components using PCA for easier visualization and noise reduction. DBSCAN was then run with eps=0.7 and min_samples=4, which identified both clusters and noise points. Finally, we visualized the results in a scatter plot, where each point was colored by its cluster label, making it easy to interpret how well the algorithm grouped the content.

5. Model Training & Optimization:

In this step, we applied K-Means clustering with 4 clusters to the standardized dataset and visualized the clustering results using PCA. First, we reduced the high-dimensional feature set to two principal components with PCA to enable 2D visualization. Then, K-Means was run to assign each data point to a cluster, and the results were plotted in a scatter plot with different colors representing different clusters. This helped us visually assess how well K-Means separated the content based on the features.

In this step, we performed hierarchical clustering using the Agglomerative Clustering algorithm with 4 clusters and Ward linkage. We then visualized the clustering result using the previously computed PCA components. Each data point was colored based on its assigned cluster, allowing us to compare the cluster structure formed by hierarchical clustering with those from K-Means and DBSCAN. This visualization provides insight into how the hierarchical approach groups similar Netflix content based on their feature profiles.

In this step, we re-applied the DBSCAN clustering algorithm using PCA-reduced features for better visualization. DBSCAN, being a density-based method, automatically detected clusters and labeled outliers (noise) without needing a predefined number of clusters. The resulting clusters were visualized in a scatter plot where each point's color represented its assigned cluster or noise (-1). This visualization helped us assess how DBSCAN captured the structure of the data and highlighted its ability to detect clusters of varying shape and size, as well as isolate noise effectively.

6. Visualization & Interpretation:

In this step, we analyzed the characteristics of each K-Means cluster by visualizing the average values of numerical features within them. After confirming that the lengths of the encoded dataset and cluster labels matched, we added the cluster labels to the numeric features and computed the mean feature values for each cluster group. These averages were then displayed using a heatmap, which visually highlights the differences and similarities across clusters. This helped in interpreting what defines each cluster — such as higher content age, specific ratings, or type distribution — and drawing insights from the clustering results.

In this step, we explored the genre distribution within each K-Means cluster to better understand the type of content grouped together. We began by assigning the K-Means cluster labels back to the original dataset. Then, we extracted individual genres by splitting the 'listed_in' column and exploding it so that each genre had its own row. We grouped the data by cluster and genre to count their occurrences, and finally, for each cluster, we plotted

the top 5 most common genres using bar charts. This helped us interpret what kind of content dominates each cluster and revealed genre-based similarities among grouped titles.

In this final visualization step, we analyzed how content ratings (like TV-MA, PG, R) are distributed across different K-Means clusters. We created a bar plot using countplot, where each bar represents the count of titles within a specific rating category, and each color corresponds to a different cluster. This helped us understand which content ratings are more prevalent in certain clusters, providing further insights into the audience type or maturity level of the content grouped together.

7. Evaluation & Refinement:

In this step, we visualized a comparison of Silhouette Scores across different clustering models using a bar plot. By plotting the scores from models like K-Means, Hierarchical Clustering, and DBSCAN, we were able to quickly identify which algorithm provided the most well-separated and cohesive clusters. This visual summary helped in selecting the best-performing model for categorizing Netflix content based on clustering quality.

In this step, we plotted a bar chart to compare the **Davies-Bouldin Index** scores of different clustering models. Since a lower Davies-Bouldin Index indicates better-defined and more compact clusters, this visualization helped us evaluate which model achieved the best separation and tightness of clusters. It complemented the Silhouette Score comparison, offering another perspective to support the selection of the most effective clustering approach for the Netflix dataset.