Exploring Netflix Content Patterns: An Analytical Approach Using Metadata

In [1]:
```
import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
# Load the dataset
netflix_data = pd.read_csv('netflix_titles.csv')

# Display the first few rows to inspect the data
netflix_data.head()
```

Out[1]:

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
0	s1	Movie	Dick Johnson Is Dead	Kirsten Johnson	NaN	United States	September 25, 2021	2020	PG-13	90 min	Documentaries	As her father nears the end of his life, filmm...
1	s2	TV Show	Blood & Water	NaN	Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...	South Africa	September 24, 2021	2021	TV-MA	2 Seasons	International TV Shows, TV Dramas, TV Mysteries	After crossing paths at a party, a Cape Town t...
2	s3	TV Show	Ganglands	Julien Leclercq	Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...	NaN	September 24, 2021	2021	TV-MA	1 Season	Crime TV Shows, International TV Shows, TV Act...	To protect his family from a powerful drug lor...
3	s4	TV Show	Jailbirds New Orleans	NaN	NaN	NaN	September 24, 2021	2021	TV-MA	1 Season	Docuseries, Reality TV	Feuds, flirtations and toilet talk go down amo...
4	s5	TV Show	Kota Factory	NaN	Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...	India	September 24, 2021	2021	TV-MA	2 Seasons	International TV Shows, Romantic TV Shows, TV ...	In a city of coaching centers known to train I...

In [2]:
```
# Directly modify the DataFrame without using inplace=True
netflix_data['country'] = netflix_data['country'].fillna('Unknown')
netflix_data['director'] = netflix_data['director'].fillna('Unknown')
netflix_data['cast'] = netflix_data['cast'].fillna('Unknown')

# If dropping rows with missing 'date_added'
netflix_data = netflix_data.dropna(subset=['date_added'])
```

In [3]:
```
# Dropping rows where 'rating' or 'duration' is missing
netflix_data = netflix_data.dropna(subset=['rating', 'duration'])

# Verify the removal of missing values
missing_data_after_drop = netflix_data.isna().sum()
```

In [4]:
```
netflix_data.isnull().sum()
```

Out[4]:
```
show_id         0
type            0
title           0
director        0
cast            0
country         0
date_added      0
release_year    0
rating          0
duration        0
listed_in       0
description     0
dtype: int64
```

In [5]:
```
netflix_data['date_added'].dtype
```

Out[5]:
```
dtype('O')
```

In [6]:
```
#Convert its Data Type from object to Date
netflix_data['date_added']=netflix_data['date_added'].apply(pd.to_datetime)
```

```
In [6]:
netflix_data['date_added'].dtype
```

```
Out[6]:
dtype('<M8[ns]')
```

```
In [7]:
#splilting Year from date_added col
netflix_data['loading_year']=netflix_data['date_added'].dt.year
netflix_data['loading_year']
```

```
Out[7]:
0       2021
1       2021
2       2021
3       2021
4       2021
        ...
8802    2019
8803    2019
8804    2019
8805    2020
8806    2019
Name: loading_year, Length: 8790, dtype: int32
```

```
In [8]:
#splilting month from date_added col and covert val to Month name
netflix_data['loading_Month']=netflix_data['date_added'].dt.month_name()
netflix_data['loading_Month']
```

```
Out[8]:
0       September
1       September
2       September
3       September
4       September
          ...
8802     November
8803         July
8804     November
8805      January
8806        March
Name: loading_Month, Length: 8790, dtype: object
```

```
In [9]:
# now we can drop date_added column
netflix_data.drop("date_added",axis=1,inplace=True)
```

Data After Cleaning

```
In [10]:
netflix_data
```

Out[10]:

	show_id	type	title	director	cast	country	release_year	rating	duration	listed_in	description	loading_year	lo
0	s1	Movie	Dick Johnson Is Dead	Kirsten Johnson	Unknown	United States	2020	PG-13	90 min	Documentaries	As her father nears the end of his life, filmm...	2021	
1	s2	TV Show	Blood & Water	Unknown	Ama Qamata, Khosi Ngema, Gail Mabalane, Thaban...	South Africa	2021	TV-MA	2 Seasons	International TV Shows, TV Dramas, TV Mysteries	After crossing paths at a party, a Cape Town t...	2021	
2	s3	TV Show	Ganglands	Julien Leclercq	Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabi...	Unknown	2021	TV-MA	1 Season	Crime TV Shows, International TV Shows, TV Act...	To protect his family from a powerful drug lor...	2021	
3	s4	TV Show	Jailbirds New Orleans	Unknown	Unknown	Unknown	2021	TV-MA	1 Season	Docuseries, Reality TV	Feuds, flirtations and toilet talk go down amo...	2021	
4	s5	TV Show	Kota Factory	Unknown	Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...	India	2021	TV-MA	2 Seasons	International TV Shows, Romantic TV Shows, TV ...	In a city of coaching centers known to train I...	2021	
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8802	s8803	Movie	Zodiac	David Fincher	Mark Ruffalo, Jake Gyllenhaal, Robert Downey J...	United States	2007	R	158 min	Cult Movies, Dramas, Thrillers	A political cartoonist, a crime reporter and a...	2019	
8803	s8804	TV Show	Zombie Dumb	Unknown	Unknown	Unknown	2018	TV-Y7	2 Seasons	Kids' TV, Korean TV Shows, TV Comedies	While living alone in a spooky town, a young g...	2019	
8804	s8805	Movie	Zombieland	Ruben Fleischer	Jesse Eisenberg, Woody Harrelson, Emma Stone, ...	United States	2009	R	88 min	Comedies, Horror Movies	Looking to survive in a world taken over by zo...	2019	
8805	s8806	Movie	Zoom	Peter Hewitt	Tim Allen, Courteney Cox, Chevy Chase, Kate Ma...	United States	2006	PG	88 min	Children & Family Movies, Comedies	Dragged from civilian life, a former superhero...	2020	
8806	s8807	Movie	Zubaan	Mozez Singh	Vicky Kaushal, Sarah-Jane Dias, Raaghav Chanan...	India	2015	TV-14	111 min	Dramas, International Movies, Music & Musicals	A scrappy but poor boy worms his way into a ty...	2019	

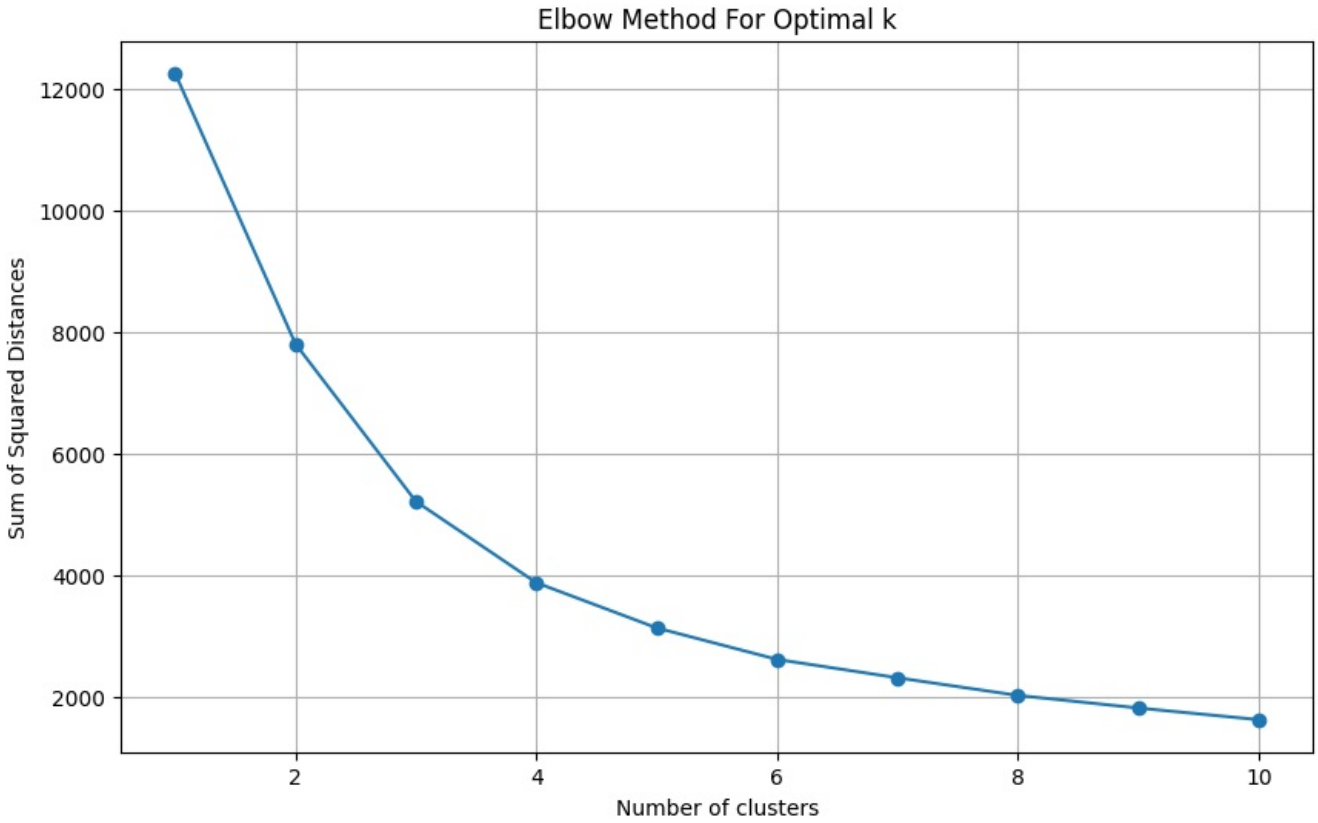
8790 rows × 13 columns

Clustering

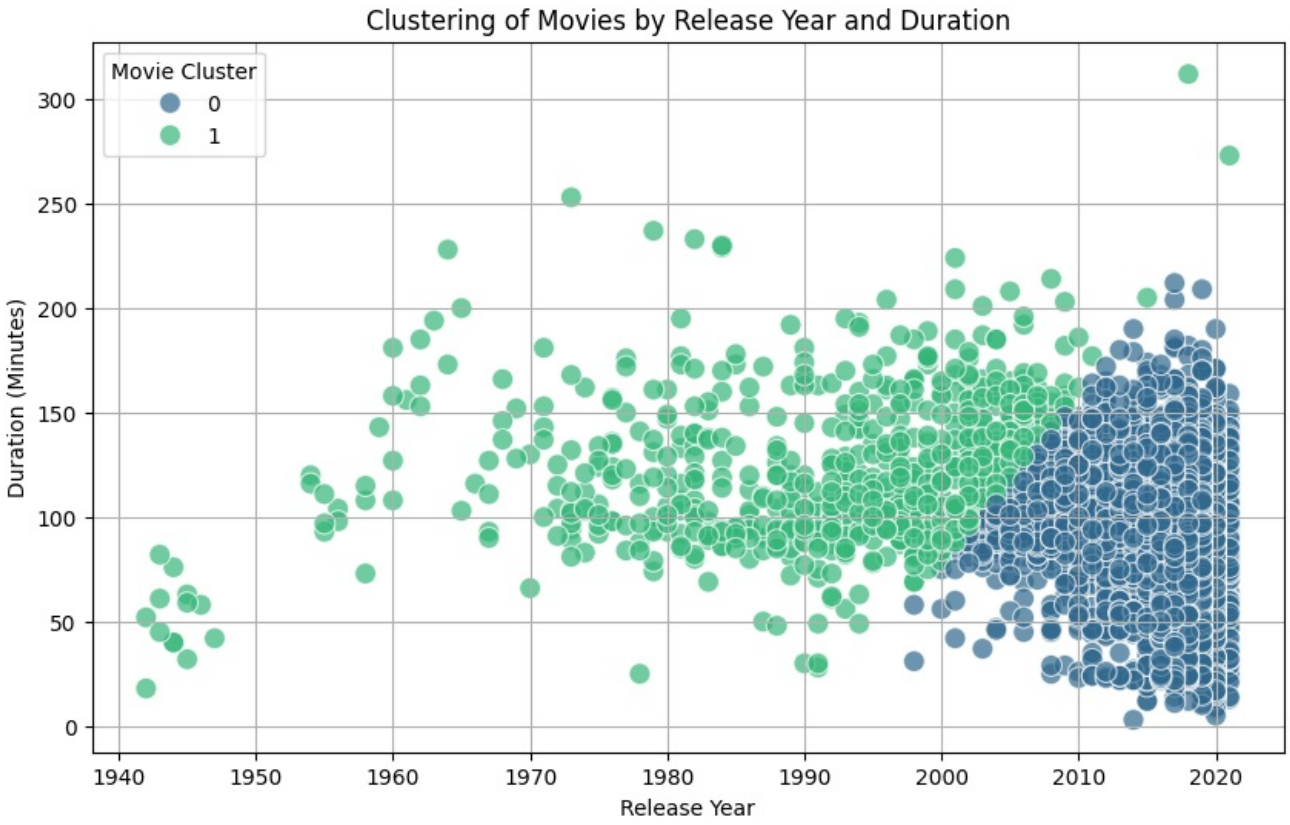
In [11]:
```
# Load and prepare the movie data
movies_data = netflix_data[netflix_data['type'] == 'Movie'].copy()
movies_data['Minutes'] = pd.to_numeric(movies_data['duration'].str.extract('(\d+)')[0], errors='coerce')
#load and prepare the TV shows data
tv_shows_data = netflix_data[netflix_data['type'] == 'TV Show'].copy()
tv_shows_data['Seasons'] = pd.to_numeric(tv_shows_data['duration'].str.extract('(\d+)')[0], errors='coerce')
```

In [12]:
```
# Scale the features
scaler = StandardScaler()
scaled_movie_features = scaler.fit_transform(movies_data[['Minutes', 'release_year']])

# Calculate SSD for a range of cluster counts
ssd = []
cluster_range = range(1, 11)  # Adjust this range as needed
```

```
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)  # n_init is set to 10 as previously discussed
    kmeans.fit(scaled_movie_features)
    ssd.append(kmeans.inertia_)

# Plotting the SSD values to find the "elbow"
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, ssd, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('Sum of Squared Distances')
plt.grid(True)
plt.show()
```

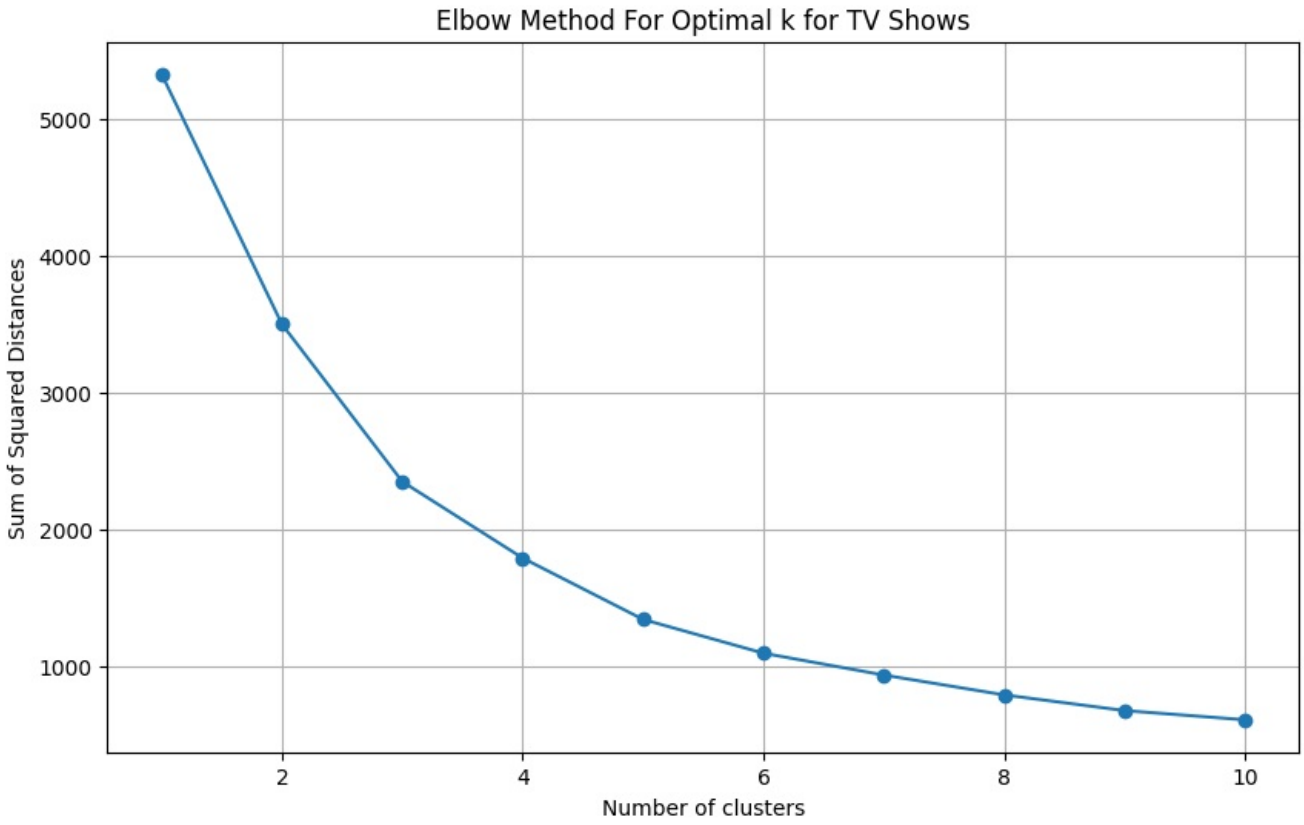


In [13]:
```
# Scale the features
scaler = StandardScaler()
scaled_movie_features = scaler.fit_transform(movies_data[['Minutes', 'release_year']])

# Apply K-Means Clustering with explicitly set n_init
movie_kmeans = KMeans(n_clusters=2, random_state=42, n_init=10)  # Explicitly setting n_init to 10
movies_data['cluster'] = movie_kmeans.fit_predict(scaled_movie_features)

# Plotting the results
plt.figure(figsize=(10, 6))
sns.scatterplot(data=movies_data, x='release_year', y='Minutes', hue='cluster', palette='viridis', s=100, alpha
plt.title('Clustering of Movies by Release Year and Duration')
plt.xlabel('Release Year')
plt.ylabel('Duration (Minutes)')
plt.legend(title='Movie Cluster')
plt.grid(True)
plt.show()
```

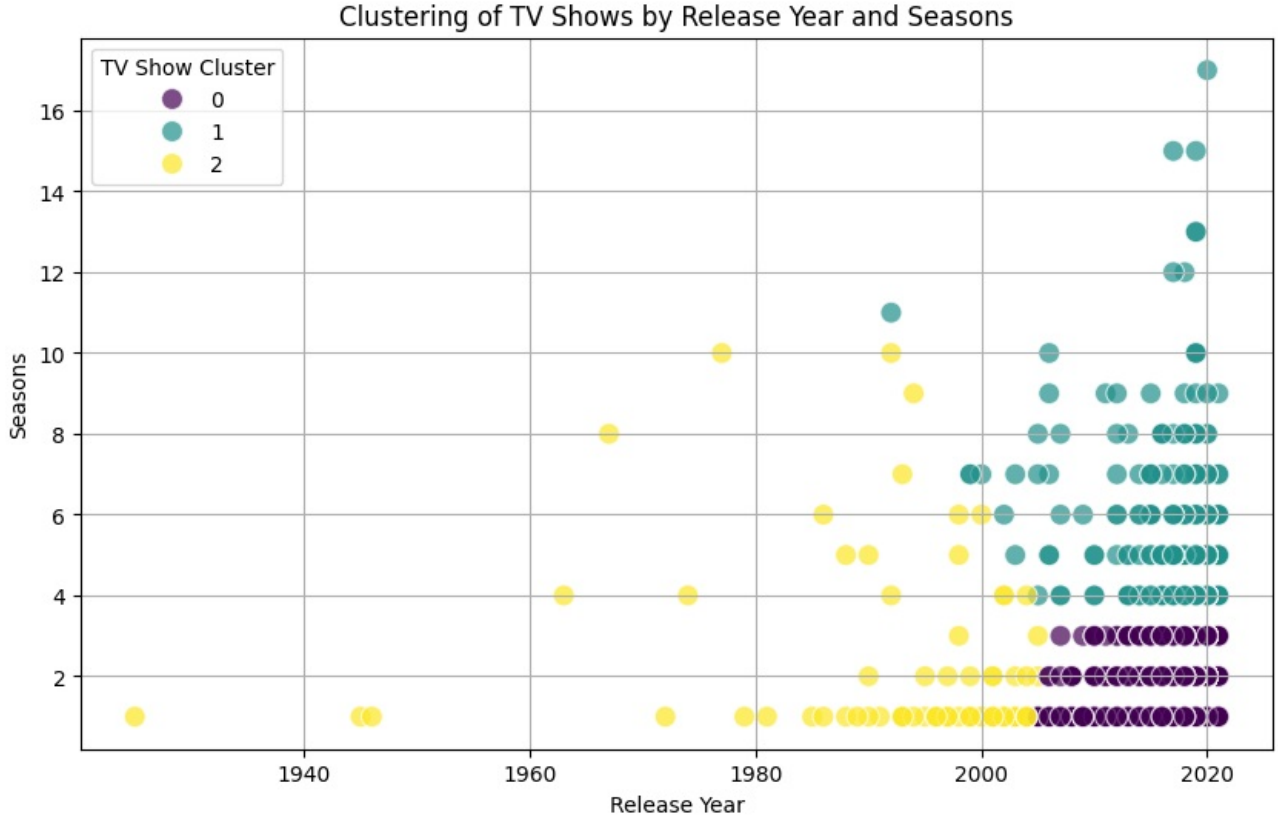


In [14]:

```
# Scale the features
scaler = StandardScaler()
scaled_tv_show_features = scaler.fit_transform(tv_shows_data[['Seasons', 'release_year']])

# Calculate SSD for a range of cluster counts
ssd = []
cluster_range = range(1, 11)  # Adjust this range based on your specific needs
for k in cluster_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)  # Explicitly setting n_init to 10
    kmeans.fit(scaled_tv_show_features)
    ssd.append(kmeans.inertia_)

# Plotting the SSD values to find the "elbow"
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, ssd, marker='o')
plt.title('Elbow Method For Optimal k for TV Shows')
plt.xlabel('Number of clusters')
plt.ylabel('Sum of Squared Distances')
plt.grid(True)
plt.show()
```

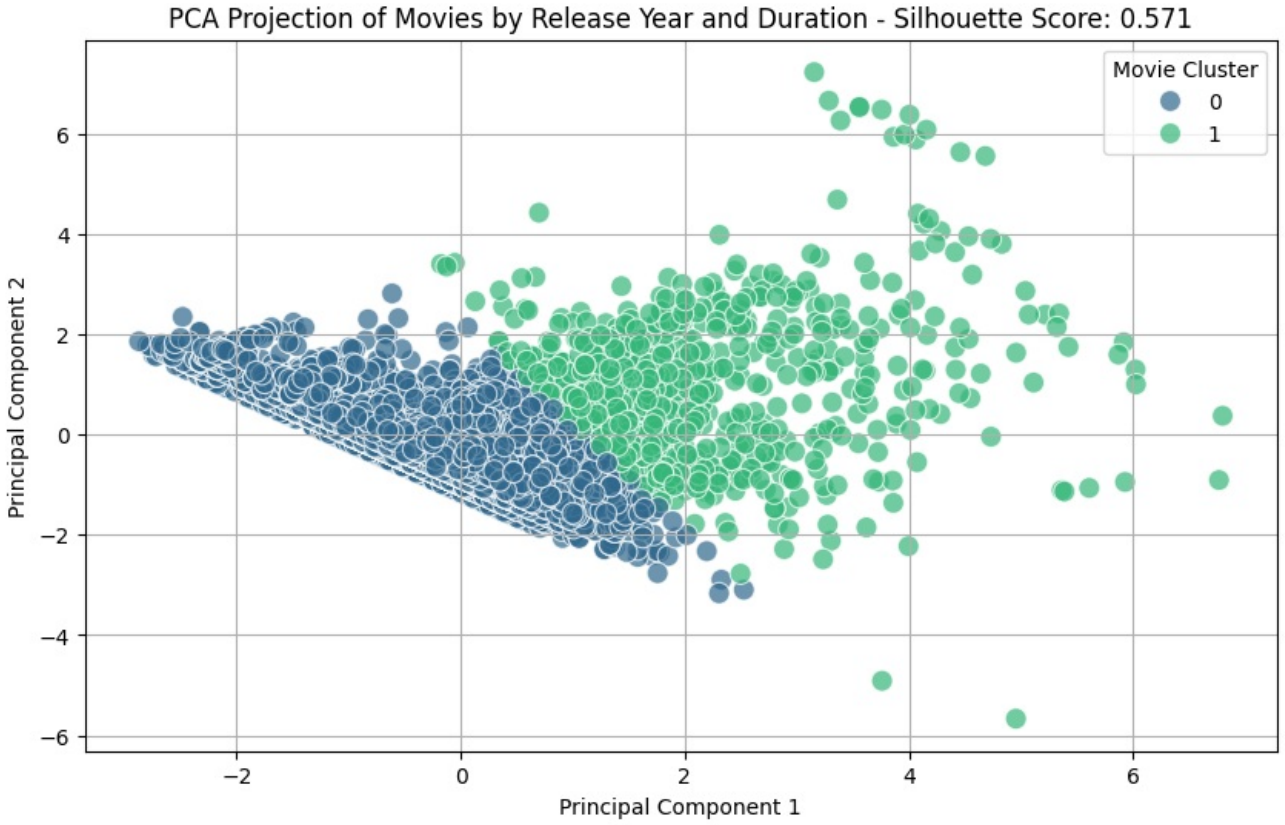


In [15]:
```
# Scale the features
scaler = StandardScaler()
scaled_tv_show_features = scaler.fit_transform(tv_shows_data[['Seasons', 'release_year']])

# Apply K-Means Clustering
tv_show_kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)  # Explicitly setting n_init to 10
tv_shows_data['cluster'] = tv_show_kmeans.fit_predict(scaled_tv_show_features)

# Plotting the results
plt.figure(figsize=(10, 6))
sns.scatterplot(data=tv_shows_data, x='release_year', y='Seasons', hue='cluster', palette='viridis', s=100, alp
plt.title('Clustering of TV Shows by Release Year and Seasons')
plt.xlabel('Release Year')
plt.ylabel('Seasons')
plt.legend(title='TV Show Cluster')
plt.grid(True)
plt.show()
```



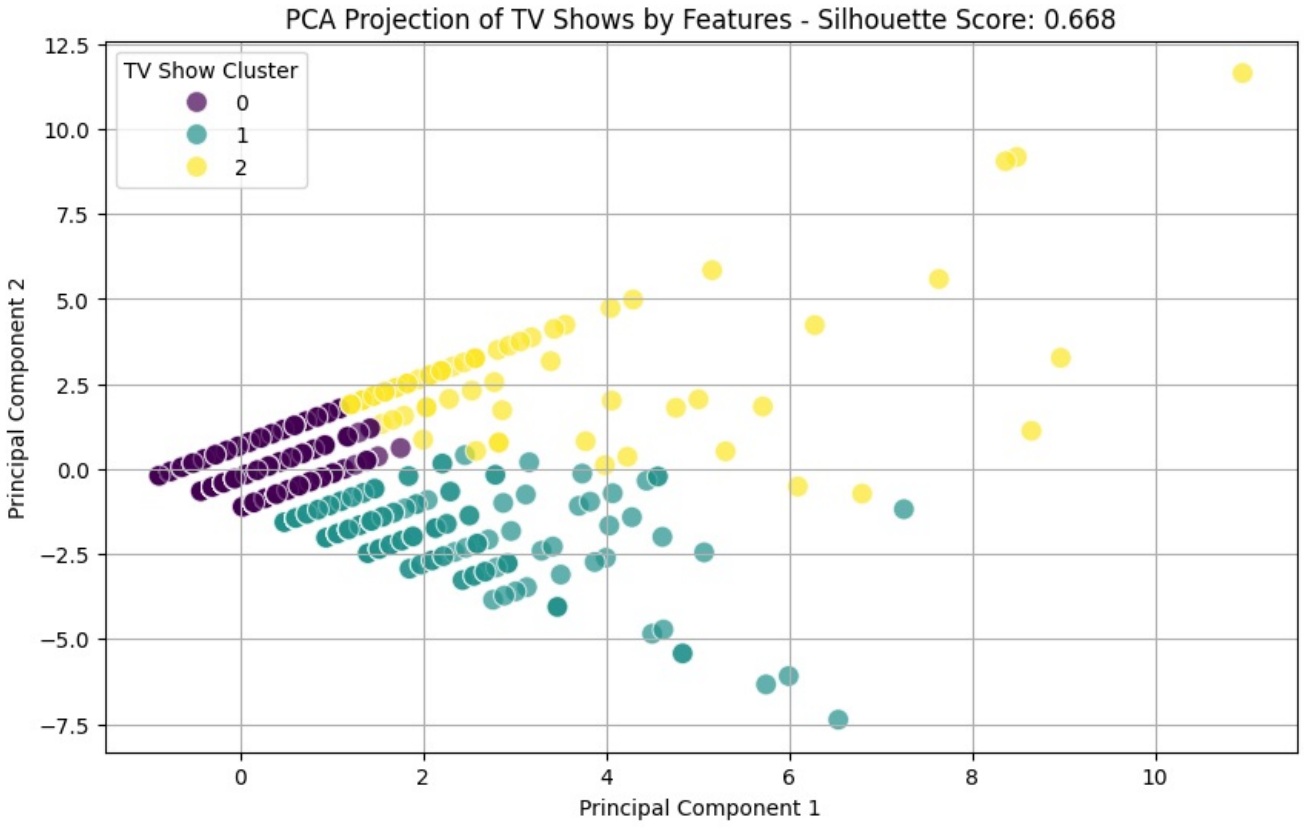
PCA Dimension Reduction

In [16]:
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming you have already loaded and prepared the data 'movies_data' with 'Minutes' and 'release_year'

# Scale the features
scaler = StandardScaler()
scaled_movie_features = scaler.fit_transform(movies_data[['Minutes', 'release_year']])

# Apply PCA to reduce dimensions to 2
pca = PCA(n_components=2)
principal_components = pca.fit_transform(scaled_movie_features)

# KMeans clustering on the PCA results
# Setting n_init explicitly to suppress the warning and assuming the number of clusters is predetermined
movie_kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)
movie_kmeans.fit(principal_components)

# Create a DataFrame with the PCA results
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
pca_df['cluster'] = movie_kmeans.labels_

# Calculate Silhouette Score
silhouette_avg = silhouette_score(principal_components, movie_kmeans.labels_)

# Plot the PCA results
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=pca_df, palette='viridis', s=100, alpha=0.7)
plt.title(f'PCA Projection of Movies by Release Year and Duration - Silhouette Score: {silhouette_avg:.3f}')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='Movie Cluster')
plt.grid(True)
plt.show()
```



In [17]:
```
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'scaled_tv_show_features' is already scaled features of the TV shows data

# Apply PCA to reduce dimensions to 2
```

```
pca_tv = PCA(n_components=2)
principal_components_tv = pca_tv.fit_transform(scaled_tv_show_features)

# KMeans clustering on the PCA results
# Replace 'n_clusters' with the actual number of clusters you're using, and set n_init to avoid future warnings
tv_show_kmeans = KMeans(n_clusters=3, n_init=10, random_state=42)
tv_show_kmeans.fit(principal_components_tv)

# Create a DataFrame with the PCA results
pca_tv_df = pd.DataFrame(data=principal_components_tv, columns=['PC1', 'PC2'])
pca_tv_df['cluster'] = tv_show_kmeans.labels_

# Calculate Silhouette Score
silhouette_avg_tv = silhouette_score(principal_components_tv, tv_show_kmeans.labels_)


# Plot the PCA results
plt.figure(figsize=(10, 6))
sns.scatterplot(x='PC1', y='PC2', hue='cluster', data=pca_tv_df, palette='viridis', s=100, alpha=0.7)
plt.title(f'PCA Projection of TV Shows by Features - Silhouette Score: {silhouette_avg_tv:.3f}')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend(title='TV Show Cluster')
plt.grid(True)
plt.show()
```

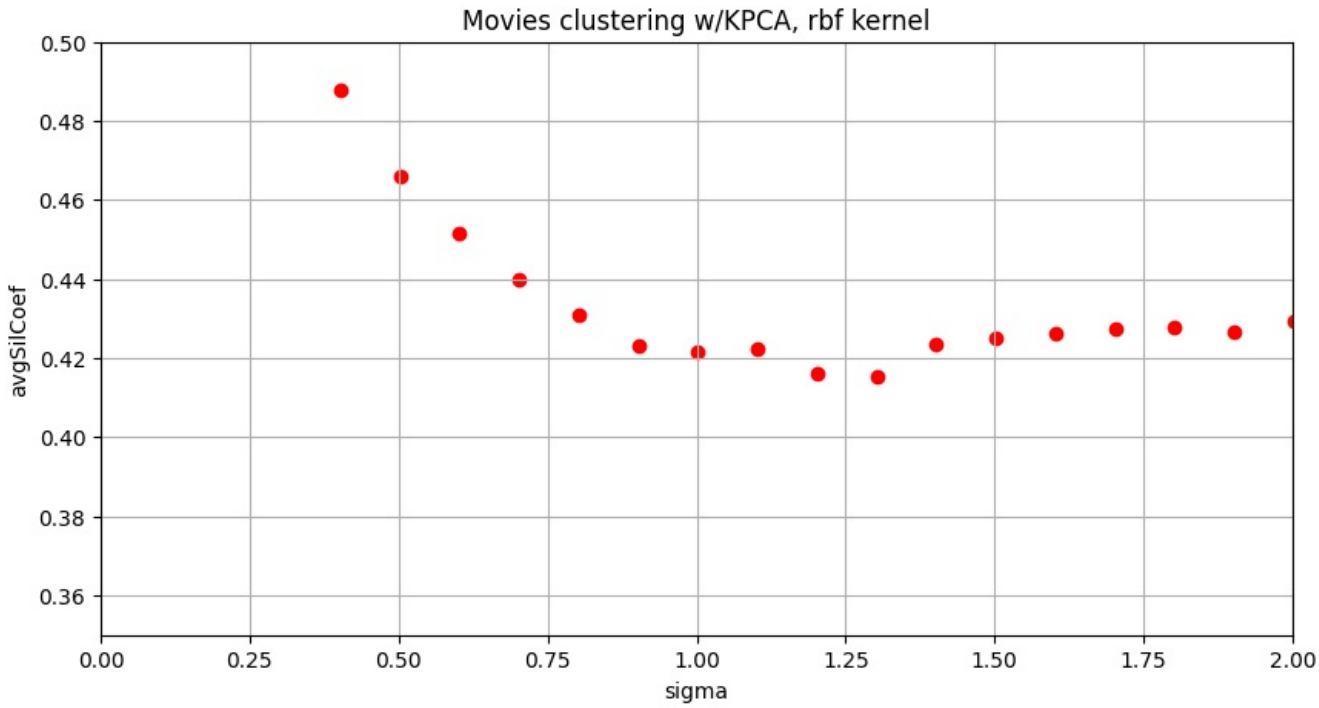


KPCA

In [18]:

```
import numpy as np
import pandas as pd
from sklearn.decomposition import KernelPCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Assuming 'movies_data' with 'Minutes' and 'release_year' is loaded and scaled
scaler = StandardScaler()
scaled_movie_features = scaler.fit_transform(movies_data[['Minutes', 'release_year']])

# Sigma values range from 0.002 to 2, in increments of 0.1
sigmas = np.arange(0.002, 2.1, 0.1)
silhouette_scores = []

for sigma in sigmas:
    gamma = 1 / (2 * sigma ** 2)  # Convert sigma to gamma for the RBF kernel
    kpca = KernelPCA(n_components=2, kernel='rbf', gamma=gamma)
    kernel_principal_components = kpca.fit_transform(scaled_movie_features)

    # KMeans clustering with explicit n_init setting to suppress the warning
    kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)
```

```
    kmeans.fit(kernel_principal_components)

    # Compute Silhouette Score
    score = silhouette_score(kernel_principal_components, kmeans.labels_)
    silhouette_scores.append(score)

# Plotting the silhouette coefficient versus sigma
plt.figure(figsize=(10, 5))
plt.scatter(sigmas, silhouette_scores, color='red')
plt.title("Movies clustering w/KPCA, rbf kernel")
plt.xlabel("sigma")
plt.ylabel("avgSilCoef")
plt.xlim(0, 2)
plt.ylim(0.35, 0.5)
plt.grid(True)
plt.show()
```



While KPCA can be a powerful tool for capturing complex patterns in data by introducing non-linearity, it is not universally better than PCA. For this particular dataset, PCA provides a more effective clustering based on the silhouette score. The linear relationships in the data might be more significant than the non-linear ones that KPCA tries to model with the RBF kernel. Therefore, PCA remains the preferred method for this dataset based on the clustering evaluation metric provided.

Now for TV Shows

Apply PCA and Record the Silhouette Score

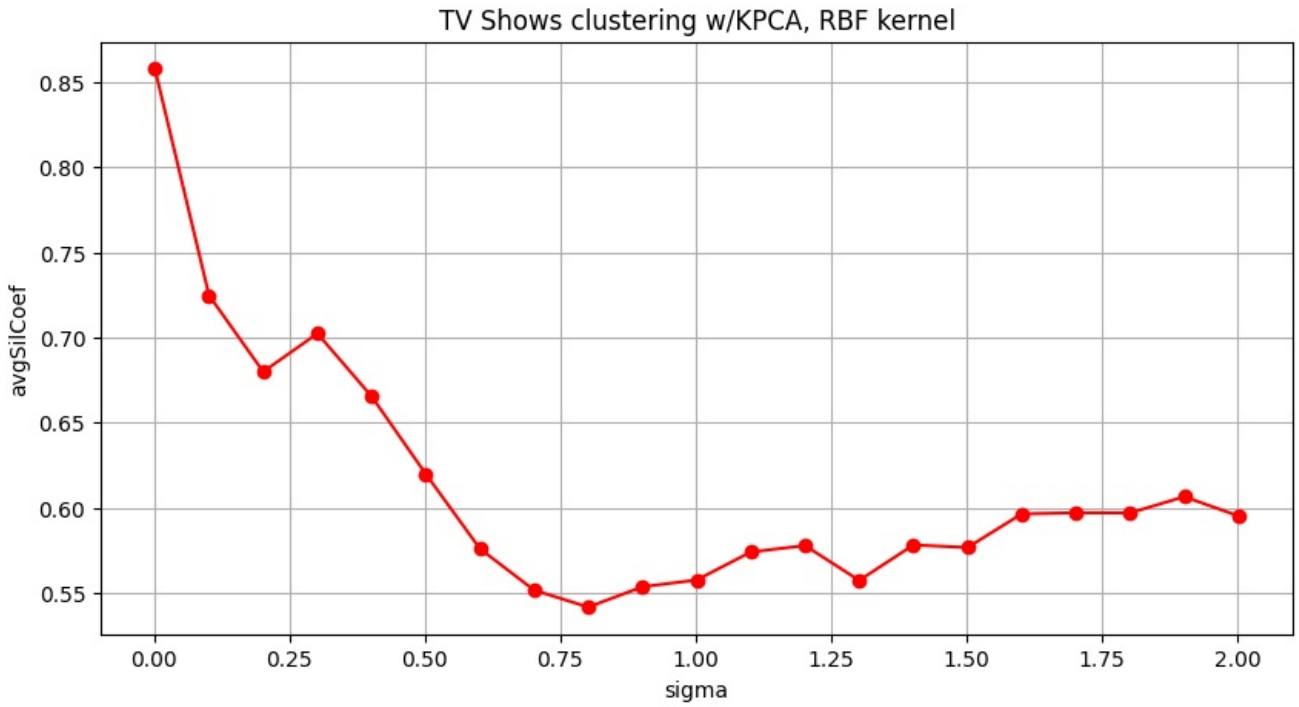
This has been done as shown in your code snippet, with a resulting silhouette score which provides a benchmark for comparison. Apply KPCA for Different Sigma Values

Using the RBF kernel, vary sigma from 0.002 to 2 in increments of 0.1, and perform KMeans clustering on the KPCA-transformed data to calculate silhouette scores for each configuration. Plot Silhouette Scores for KPCA

Compare these scores with the baseline PCA score to determine if KPCA offers any improvement.

In [19]:
```
import numpy as np
import pandas as pd
from sklearn.decomposition import KernelPCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Assuming 'scaled_tv_show_features' is already scaled features of the TV shows data

# Sigma values range from 0.002 to 2, in increments of 0.1
sigmas = np.arange(0.002, 2.1, 0.1)
silhouette_scores_kpca = []

for sigma in sigmas:
    gamma = 1 / (2 * sigma ** 2)  # Convert sigma to gamma for the RBF kernel
    kpca_tv = KernelPCA(n_components=2, kernel='rbf', gamma=gamma)
    kernel_principal_components_tv = kpca_tv.fit_transform(scaled_tv_show_features)

    # KMeans clustering on the KPCA results
```

```
    tv_show_kmeans_kpca = KMeans(n_clusters=3, n_init=10, random_state=42)
    tv_show_kmeans_kpca.fit(kernel_principal_components_tv)

    # Compute Silhouette Score
    score_kpca = silhouette_score(kernel_principal_components_tv, tv_show_kmeans_kpca.labels_)
    silhouette_scores_kpca.append(score_kpca)

# Plotting the silhouette coefficient versus sigma for KPCA
plt.figure(figsize=(10, 5))
plt.plot(sigmas, silhouette_scores_kpca, 'o-', color='red')
plt.title("TV Shows clustering w/KPCA, RBF kernel")
plt.xlabel("sigma")
plt.ylabel("avgSilCoef")
plt.grid(True)
plt.show()

# Print PCA Silhouette Score for comparison
print(f"PCA Silhouette Score: {silhouette_avg_tv:.3f}")
```




```
PCA Silhouette Score: 0.668
```

The optimal sigma range for better clustering compared to PCA is very small, approximately from 0.00 to 0.05. In this range, KPCA achieves a higher silhouette score than PCA, indicating better cluster definition.

In [20]:
```
!pip install mlxtend==0.23.3
```

```
Collecting mlxtend==0.23.3
  Downloading mlxtend-0.23.3-py3-none-any.whl (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 57.5 MB/s eta 0:00:00
Requirement already satisfied: scipy>=1.2.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from mlxt
end==0.23.3) (1.9.3)
Requirement already satisfied: numpy>=1.16.2 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from mlx
tend==0.23.3) (1.23.4)
Collecting scikit-learn>=1.3.1
  Downloading scikit_learn-1.5.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (13.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.4/13.4 MB 105.2 MB/s eta 0:00:00
Requirement already satisfied: pandas>=0.24.2 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from ml
xtend==0.23.3) (2.1.4)
Requirement already satisfied: matplotlib>=3.0.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
mlxtend==0.23.3) (3.6.0)
Requirement already satisfied: joblib>=0.13.2 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from ml
xtend==0.23.3) (1.2.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
matplotlib>=3.0.0->mlxtend==0.23.3) (1.4.4)
Requirement already satisfied: python-dateutil>=2.7 in /shared-libs/python3.9/py-core/lib/python3.9/site-packag
es (from matplotlib>=3.0.0->mlxtend==0.23.3) (2.8.2)
Requirement already satisfied: pillow>=6.2.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from mat
plotlib>=3.0.0->mlxtend==0.23.3) (9.2.0)
Requirement already satisfied: cycler>=0.10 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from matp
lotlib>=3.0.0->mlxtend==0.23.3) (0.11.0)
Requirement already satisfied: contourpy>=1.0.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
matplotlib>=3.0.0->mlxtend==0.23.3) (1.0.5)
Requirement already satisfied: packaging>=20.0 in /shared-libs/python3.9/py-core/lib/python3.9/site-packages (f
rom matplotlib>=3.0.0->mlxtend==0.23.3) (21.3)
Requirement already satisfied: pyparsing>=2.2.1 in /shared-libs/python3.9/py-core/lib/python3.9/site-packages (
from matplotlib>=3.0.0->mlxtend==0.23.3) (3.0.9)
Requirement already satisfied: fonttools>=4.22.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from
matplotlib>=3.0.0->mlxtend==0.23.3) (4.37.4)
Requirement already satisfied: pytz>=2020.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from pand
as>=0.24.2->mlxtend==0.23.3) (2022.5)
Requirement already satisfied: tzdata>=2022.1 in /shared-libs/python3.9/py/lib/python3.9/site-packages (from pa
ndas>=0.24.2->mlxtend==0.23.3) (2022.5)
Requirement already satisfied: threadpoolctl>=3.1.0 in /shared-libs/python3.9/py/lib/python3.9/site-packages (f
rom scikit-learn>=1.3.1->mlxtend==0.23.3) (3.1.0)
Requirement already satisfied: six>=1.5 in /shared-libs/python3.9/py-core/lib/python3.9/site-packages (from pyt
hon-dateutil>=2.7->matplotlib>=3.0.0->mlxtend==0.23.3) (1.16.0)
Installing collected packages: scikit-learn, mlxtend
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.1.2
    Not uninstalling scikit-learn at /shared-libs/python3.9/py/lib/python3.9/site-packages, outside environment
/root/venv
    Can't uninstall 'scikit-learn'. No files were found to uninstall.
Successfully installed mlxtend-0.23.3 scikit-learn-1.5.2

[notice] A new release of pip is available: 23.0.1 -> 24.3.1
[notice] To update, run: pip install --upgrade pip
```

Netflix Recommender System

Content Based Recommender System

In [21]:

```python
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Combine relevant features into a single string
def combine_features(row):
    return f"{row['type']} {row['director']} {' '.join(row['cast'].split(', '))} {row['country']} {row['rating'

# Update the combine_features function to handle missing or non-string values
def combine_features(row):
    return f"{str(row['type'])} {str(row['director'])} {' '.join(str(row['cast']).split(', ') if isinstance(row

# Apply the function to create the 'combined_features' column
netflix_data['combined_features'] = netflix_data.apply(combine_features, axis=1)

# Check if the 'combined_features' column was created correctly
print(netflix_data[['title', 'combined_features']].head())

# Create a combined features column
netflix_data['combined_features'] = netflix_data.apply(combine_features, axis=1)

# Convert the combined features into TF-IDF vectors
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(netflix_data['combined_features'])

# Compute cosine similarity matrix
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

print("Shape of Cosine Similarity Matrix:", cosine_sim.shape)
```

```
                  title                                    combined_features
0   Dick Johnson Is Dead  Movie Kirsten Johnson Unknown United States PG-13
1          Blood & Water  TV Show Unknown Ama Qamata Khosi Ngema Gail Ma...
2              Ganglands  TV Show Julien Leclercq Sami Bouajila Tracy Go...
3  Jailbirds New Orleans              TV Show Unknown Unknown Unknown TV-MA
4           Kota Factory  TV Show Unknown Mayur More Jitendra Kumar Ranj...
Shape of Cosine Similarity Matrix: (8790, 8790)
```

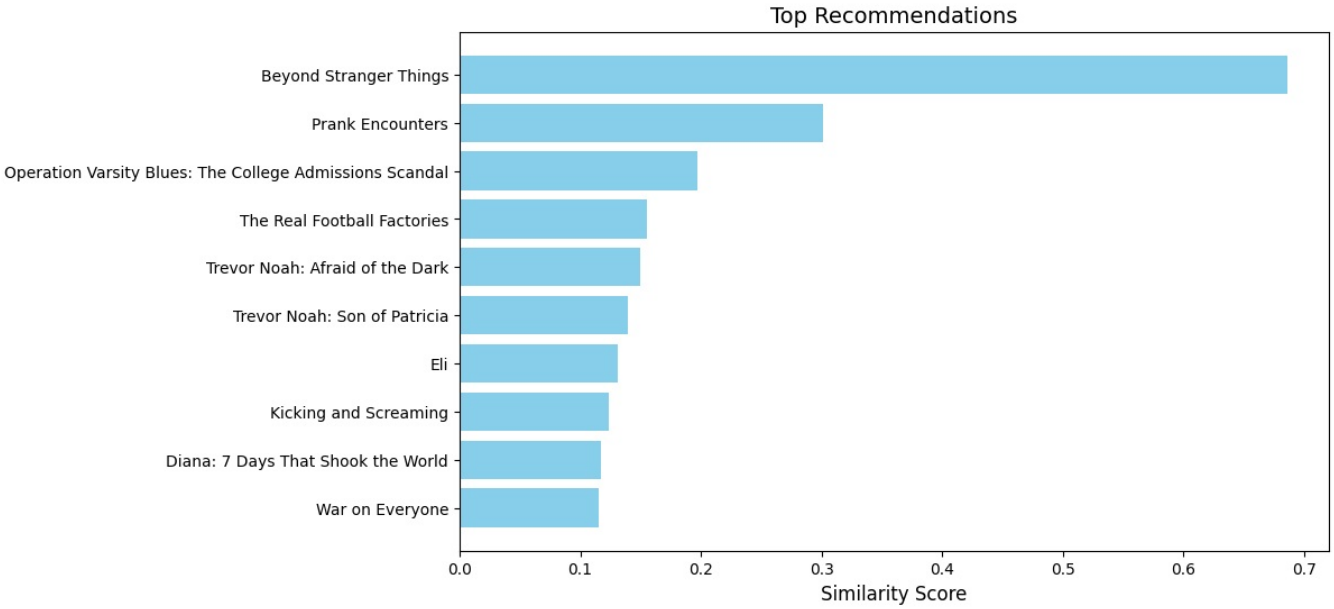
Recommender System Function

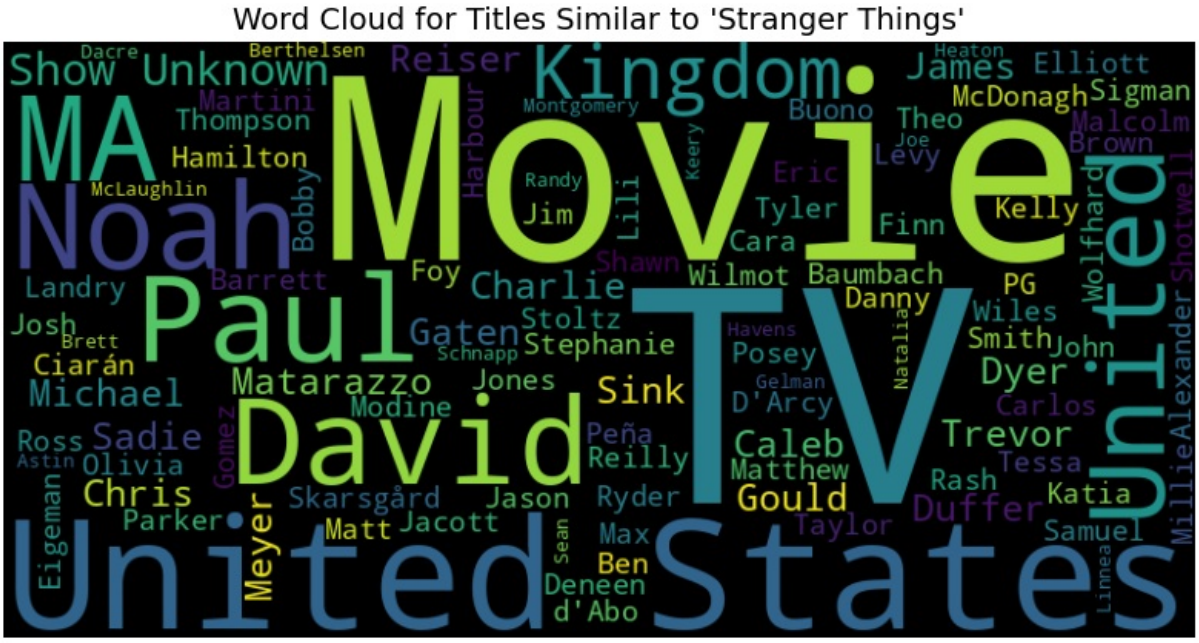
In [22]:

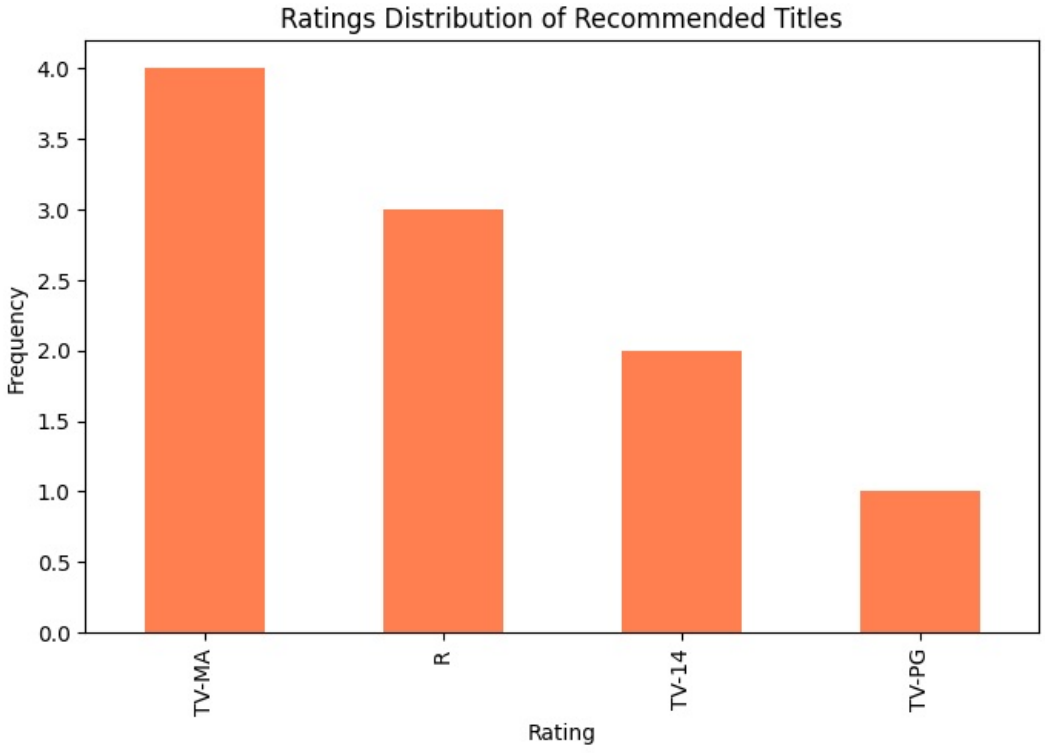
```python
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

# Combine features into a single column
def combine_features(row):
    return f"{str(row['type'])} {str(row['director'])} {' '.join(str(row['cast']).split(', ') if isinstance(row

netflix_data['combined_features'] = netflix_data.apply(combine_features, axis=1)

# TF-IDF vectorization
tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(netflix_data['combined_features'])

# Compute cosine similarity
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)

# Recommender function
def recommend(title, cosine_sim, data):
    try:
        idx = data[data['title'].str.lower() == title.lower()].index[0]
        sim_scores = list(enumerate(cosine_sim[idx]))
        sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
        sim_indices = [i[0] for i in sim_scores[1:11]]
        recommendations = data.iloc[sim_indices][['title', 'type', 'rating', 'release_year', 'duration', 'count
        recommendations['similarity_score'] = [sim_scores[i][1] for i in range(1, 11)]
        return recommendations
    except IndexError:
        return f"'{title}' not found in the dataset. Please check your input."

# Visualize Recommendations
def visualize_recommendations(recommendations):
    plt.figure(figsize=(10, 6))
    plt.barh(recommendations['title'], recommendations['similarity_score'], color='skyblue')
    plt.xlabel('Similarity Score', fontsize=12)
    plt.title('Top Recommendations', fontsize=14)
    plt.gca().invert_yaxis()  # Invert Y-axis for ranking
    plt.show()

# Generate Word Cloud
def generate_wordcloud(title, data):
    try:
        idx = data[data['title'].str.lower() == title.lower()].index[0]
        similar_indices = cosine_similarity(tfidf_matrix[idx], tfidf_matrix).argsort()[0, -11:-1]
        similar_data = data.iloc[similar_indices]
        text = ' '.join(similar_data['combined_features'])
        wordcloud = WordCloud(width=800, height=400, background_color='black').generate(text)

        plt.figure(figsize=(10, 6))
        plt.imshow(wordcloud, interpolation='bilinear')
        plt.axis('off')
        plt.title(f"Word Cloud for Titles Similar to '{title}'", fontsize=14)
        plt.show()
    except IndexError:
        print(f"'{title}' not found in the dataset. Please check your input.")

# Ratings Distribution
def ratings_distribution(recommendations):
    plt.figure(figsize=(8, 5))
    recommendations['rating'].value_counts().plot(kind='bar', color='coral')
    plt.title("Ratings Distribution of Recommended Titles")
    plt.xlabel("Rating")
    plt.ylabel("Frequency")
    plt.show()

# Interactive User Input
while True:
    user_input = input("Enter a movie or TV show title (or 'exit' to quit): ").strip()
    if user_input.lower() == 'exit':
        print("Exiting the recommender system. Goodbye!")
        break

    recommendations = recommend(user_input, cosine_sim, netflix_data)
    if isinstance(recommendations, str):  # Handle missing titles
        print(recommendations)
```

```
        continue

    # Display recommendations
    print("\nRecommended Titles:")
    print(recommendations)

    # Visualize recommendations
    visualize_recommendations(recommendations)

    # Generate a word cloud for metadata of similar titles
    generate_wordcloud(user_input, netflix_data)

    # Show ratings distribution of recommendations
    ratings_distribution(recommendations)

    print("\n")
```

```
Recommended Titles:
                                                title     type rating  \
5200                              Beyond Stranger Things  TV Show  TV-14
1127                                   Prank Encounters  TV Show  TV-MA
1195  Operation Varsity Blues: The College Admission...    Movie      R
8476                        The Real Football Factories  TV Show  TV-MA
5594                     Trevor Noah: Afraid of the Dark    Movie  TV-14
4377                        Trevor Noah: Son of Patricia    Movie  TV-MA
3398                                                Eli    Movie  TV-MA
7193                                Kicking and Screaming    Movie      R
6606                  Diana: 7 Days That Shook the World    Movie  TV-PG
8697                                   War on Everyone    Movie      R

      release_year   duration         country  similarity_score
5200          2017   1 Season   United States          0.686167
1127          2021  2 Seasons   United States          0.300895
1195          2021    100 min   United States          0.197121
8476          2006   1 Season  United Kingdom          0.155618
5594          2017     67 min   United States          0.149262
4377          2018     64 min   United States          0.139583
3398          2019     98 min   United States          0.131298
7193          1995     97 min   United States          0.123951
6606          2017     93 min  United Kingdom          0.117401
8697          2016     98 min  United Kingdom          0.115766
```





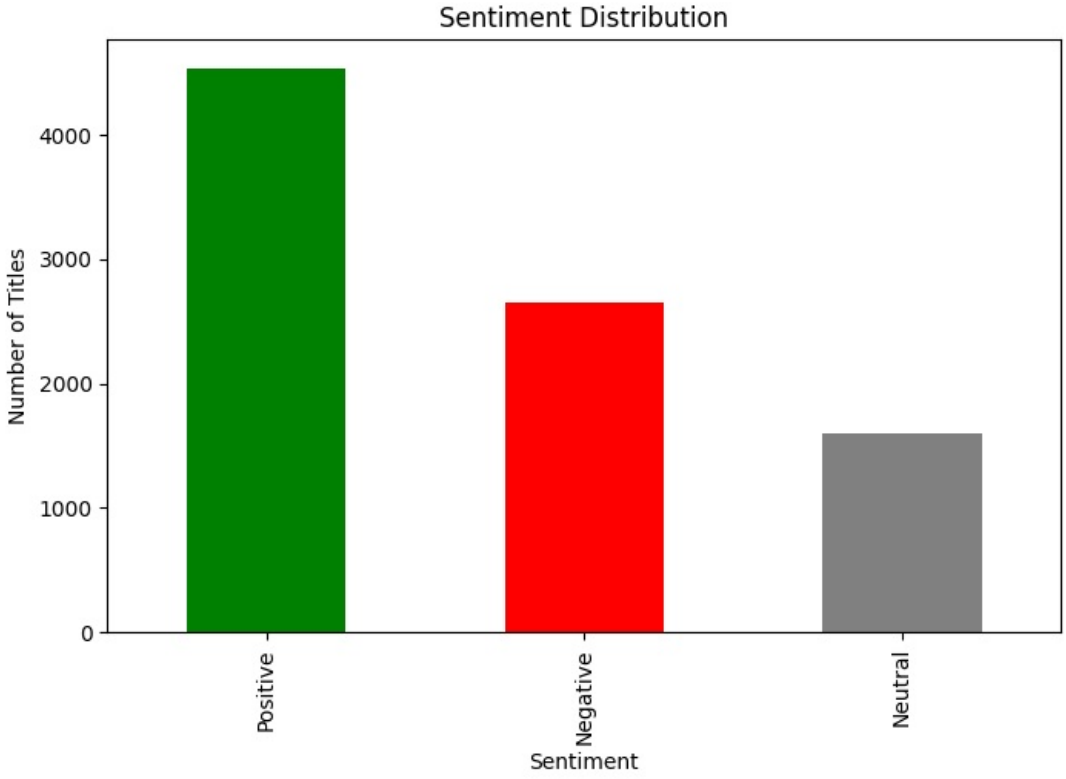



```
Exiting the recommender system. Goodbye!
```

Sentiment Analysis

In [23]:
```
from textblob import TextBlob
```

```
# Function to calculate sentiment polarity
def get_sentiment(text):
    return TextBlob(text).sentiment.polarity

# Apply sentiment analysis to the 'description' column
netflix_data['sentiment'] = netflix_data['description'].dropna().apply(get_sentiment)

# Categorize sentiment
netflix_data['sentiment_category'] = netflix_data['sentiment'].apply(
    lambda x: 'Positive' if x > 0 else ('Negative' if x < 0 else 'Neutral')
)

# Plot sentiment distribution
sentiment_counts = netflix_data['sentiment_category'].value_counts()
sentiment_counts.plot(kind='bar', color=['green', 'red', 'gray'], figsize=(8, 5), title="Sentiment Distribution
plt.xlabel("Sentiment")
plt.ylabel("Number of Titles")
plt.show()
```

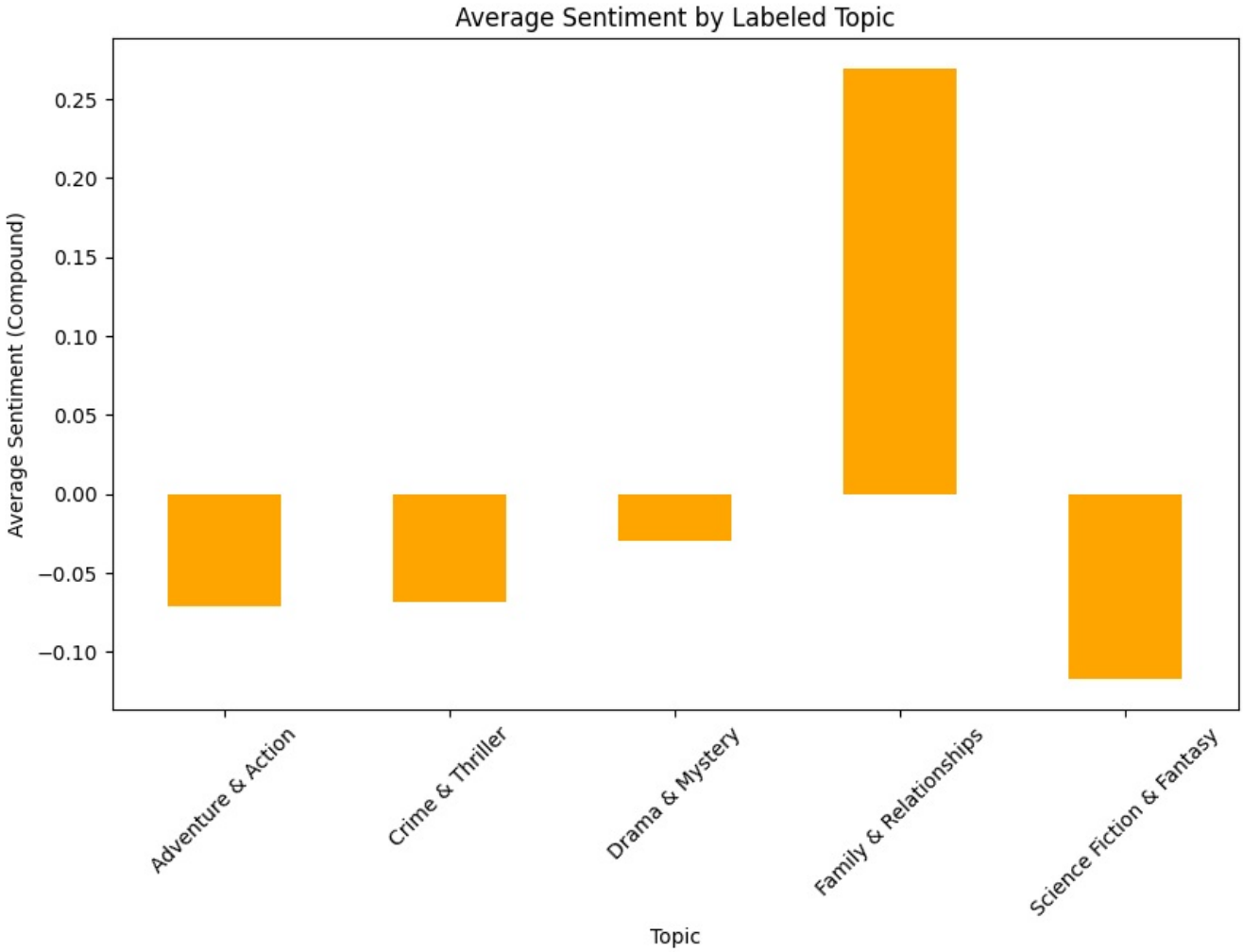


In [30]:

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import LatentDirichletAllocation

# Vectorize the 'description' column for LDA
vectorizer = CountVectorizer(stop_words='english', max_features=1000)
text_matrix = vectorizer.fit_transform(netflix_data['description'].dropna())

# Fit the LDA model
lda = LatentDirichletAllocation(n_components=5, random_state=42)  # 5 topics
lda.fit(text_matrix)

# Extract and label topics
words = vectorizer.get_feature_names_out()
topics = {}
n_top_words = 10  # Number of top words to display per topic
print("Topics and Top Words:")
for topic_idx, topic in enumerate(lda.components_):
    top_words = [words[i] for i in topic.argsort()[-n_top_words:][::-1]]
    topics[topic_idx] = top_words
    print(f"Topic {topic_idx}: {', '.join(top_words)}")

# Manually label topics based on extracted keywords
topics_labels = {
    0: "Family & Relationships",
    1: "Crime & Thriller",
    2: "Science Fiction & Fantasy",
    3: "Adventure & Action",
    4: "Drama & Mystery"
}

# Assign topics to the dataset
netflix_data['topic'] = lda.transform(text_matrix).argmax(axis=1)
netflix_data['topic_label'] = netflix_data['topic'].map(topics_labels)
```

```
# Visualize sentiment by labeled topic
topic_sentiment = netflix_data.groupby('topic_label')['compound_sentiment'].mean()

plt.figure(figsize=(10, 6))
topic_sentiment.plot(kind='bar', color='orange')
plt.title("Average Sentiment by Labeled Topic")
plt.xlabel("Topic")
plt.ylabel("Average Sentiment (Compound)")
plt.xticks(rotation=45)
plt.show()

# Display dataset with labeled topics and sentiment
print(netflix_data[['title', 'description', 'topic_label', 'compound_sentiment']].head())
```

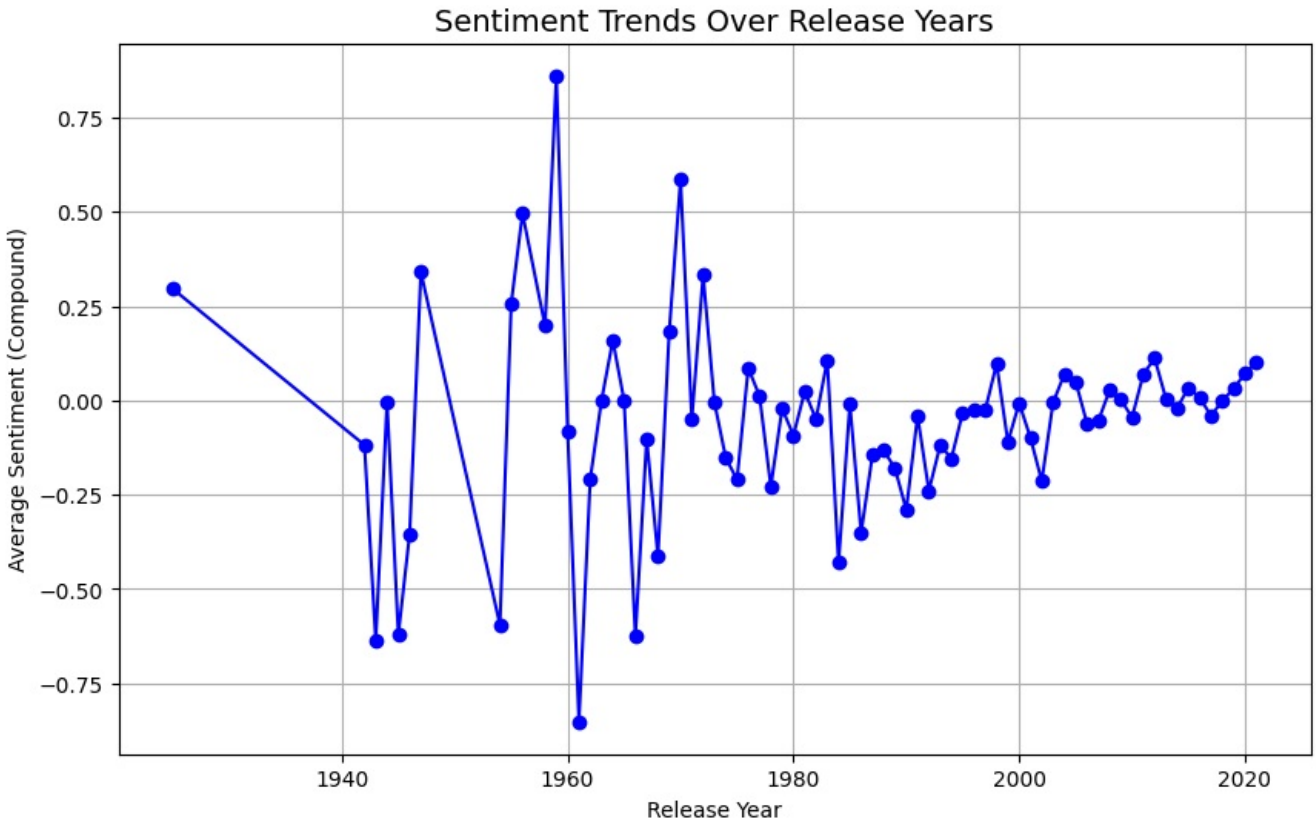
```
Topics and Top Words:
Topic 0: life, documentary, series, friends, world, comedy, new, stand, stories, special
Topic 1: school, family, high, story, father, true, young, based, world, war
Topic 2: woman, young, man, love, new, town, life, father, falls, son
Topic 3: years, life, family, lives, couple, finds, marriage, turns, home, new
Topic 4: new, save, world, old, friends, fight, year, evil, team, help
```




```
                title                                        description  \
0   Dick Johnson Is Dead  As her father nears the end of his life, filmm...
1          Blood & Water  After crossing paths at a party, a Cape Town t...
2              Ganglands  To protect his family from a powerful drug lor...
3  Jailbirds New Orleans  Feuds, flirtations and toilet talk go down amo...
4           Kota Factory  In a city of coaching centers known to train I...

             topic_label  compound_sentiment
0        Crime & Thriller             -0.2960
1      Adventure & Action             -0.1531
2        Crime & Thriller             -0.7783
3  Family & Relationships              0.2263
4  Family & Relationships              0.7430
```

In [29]:

```
# Group sentiment by release year
sentiment_trend = netflix_data.groupby('release_year')['compound_sentiment'].mean()

# Plot sentiment trends over time
plt.figure(figsize=(10, 6))
plt.plot(sentiment_trend.index, sentiment_trend.values, marker='o', color='blue')
plt.title("Sentiment Trends Over Release Years", fontsize=14)
plt.xlabel("Release Year")
plt.ylabel("Average Sentiment (Compound)")
plt.grid(True)
plt.show()
```



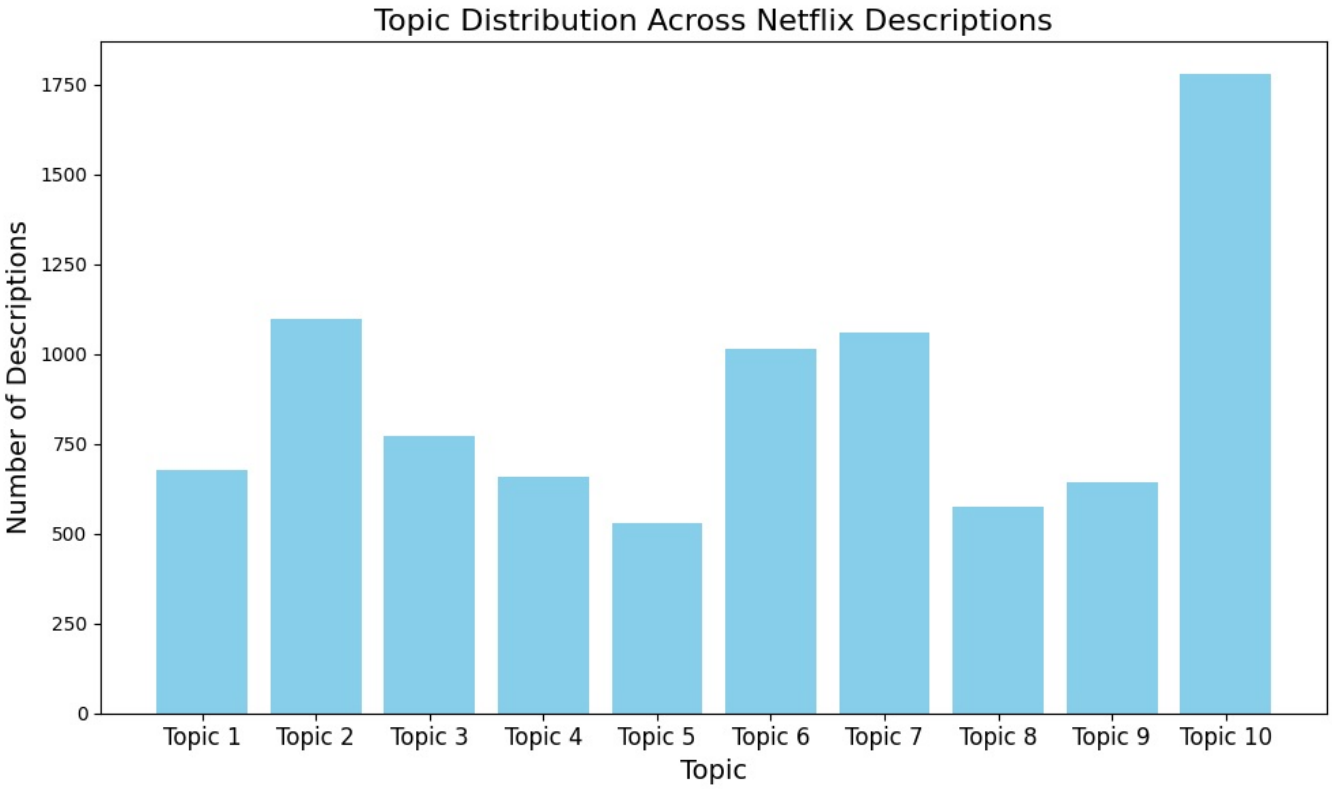
Topic Modeling

In [4]:

```python
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from gensim.models.ldamodel import LdaModel
from gensim.corpora.dictionary import Dictionary
from gensim.matutils import Sparse2Corpus
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import nltk
import string

# Preprocess the Netflix dataset
file_path = 'netflix_titles.csv'
df = pd.read_csv(file_path)
descriptions = df['description'].dropna()

def preprocess_text(text):
    stop_words = set(stopwords.words('english'))
    words = text.lower().split()
    return [word.strip(string.punctuation) for word in words if word.isalpha() and word not in stop_words]
processed_descriptions = descriptions.apply(preprocess_text)


# Convert to a document-term matrix in Sparse format
vectorizer = CountVectorizer(analyzer=lambda x: x)
doc_term_matrix = vectorizer.fit_transform(processed_descriptions)

# Convert to gensim corpus and dictionary
corpus = Sparse2Corpus(doc_term_matrix, documents_columns=False)
id2word = Dictionary([list(vectorizer.vocabulary_.keys())])

# Train LDA model
optimal_topic_count = 10
lda_model = LdaModel(
    corpus,
    num_topics=optimal_topic_count,
    id2word=id2word,
    passes=10,
    iterations=1000,
    random_state=42
)

# Print the topics and their top words
print("\nIdentified Topics and Their Top Words:")
for i, topic in lda_model.show_topics(num_topics=optimal_topic_count, num_words=10, formatted=False):
    print(f"Topic {i + 1}: {[word for word, _ in topic]}")

# Assign topics to descriptions
topic_assignments = []
```

```
for doc in corpus:
    topic_probs = lda_model.get_document_topics(doc)
    dominant_topic = max(topic_probs, key=lambda x: x[1])[0]
    topic_assignments.append(dominant_topic)
df['Assigned_Topic'] = topic_assignments

# Visualize the topic distribution
topic_counts = df['Assigned_Topic'].value_counts().sort_index()
plt.figure(figsize=(10, 6))
plt.bar(topic_counts.index, topic_counts.values, color='skyblue')
plt.title("Topic Distribution Across Netflix Descriptions", fontsize=16)
plt.xlabel("Topic", fontsize=14)
plt.ylabel("Number of Descriptions", fontsize=14)
plt.xticks(ticks=range(optimal_topic_count), labels=[f"Topic {i + 1}" for i in range(optimal_topic_count)], fon
plt.tight_layout()
plt.show()
```

```
Identified Topics and Their Top Words:
Topic 1: ['life', 'soccer', 'brother', 'actor', 'film', 'new', 'family', 'lives', 'india', 'documentary']
Topic 2: ['new', 'police', 'young', 'crime', 'power', 'secret', 'gang', 'two', 'team', 'war']
Topic 3: ['life', 'documentary', 'takes', 'young', 'take', 'world', 'explores', 'true', 'successful', 'man']
Topic 4: ['series', 'documentary', 'stories', 'follows', 'tv', 'host', 'show', 'examines', 'four', 'world']
Topic 5: ['returns', 'new', 'life', 'world', 'team', 'help', 'big', 'school', 'king', 'without']
Topic 6: ['new', 'friends', 'three', 'life', 'one', 'music', 'best', 'high', 'lives', 'find']
Topic 7: ['two', 'young', 'find', 'one', 'save', 'family', 'world', 'friends', 'help', 'take']
Topic 8: ['comedian', 'social', 'life', 'media', 'comic', 'turning', 'documentary', 'comedy', 'personal', 'spec
ial']
Topic 9: ['young', 'man', 'woman', 'must', 'love', 'falls', 'travels', 'life', 'musical', 'fight']
Topic 10: ['young', 'man', 'new', 'woman', 'two', 'family', 'father', 'girl', 'old', 'love']
```



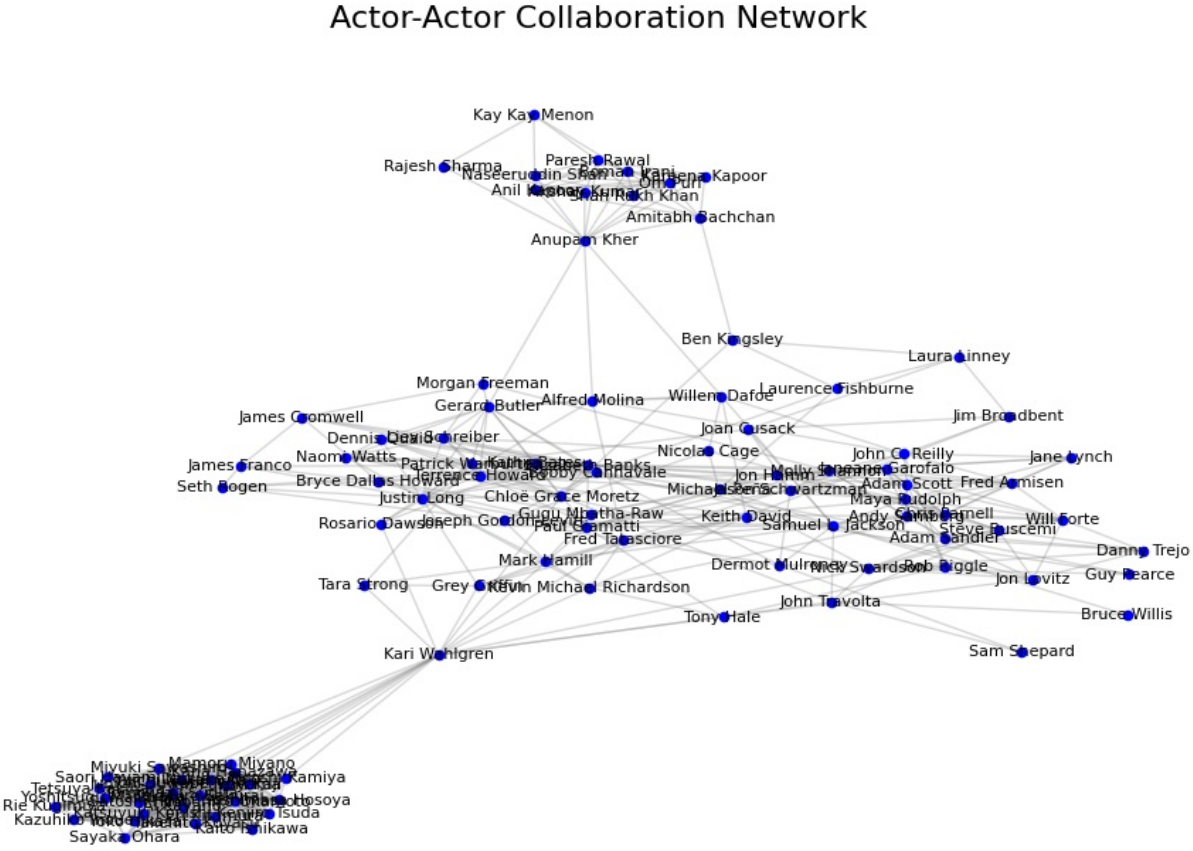
Link Analysis

In [8]:

```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt
from itertools import combinations
from collections import Counter
import datetime

# Load the Netflix dataset
file_path = 'netflix_titles.csv'
df = pd.read_csv(file_path)

# Data Cleaning
df = df.dropna(subset=['cast', 'listed_in', 'release_year'])
df['release_year'] = pd.to_numeric(df['release_year'], errors='coerce')

# Actor-Actor Network
actor_graph = nx.Graph()

for cast in df['cast']:
    actors = [actor.strip() for actor in cast.split(',')]
    for actor1, actor2 in combinations(actors, 2):
        if actor_graph.has_edge(actor1, actor2):
```

```
            actor_graph[actor1][actor2]['weight'] += 1
        else:
            actor_graph.add_edge(actor1, actor2, weight=1)

#Top Actor Pairs
top_actor_pairs = sorted(actor_graph.edges(data=True), key=lambda x: x[2]['weight'], reverse=True)[:10]
print("Top Actor Pairs by Collaboration:")
for pair in top_actor_pairs:
    print(f"{pair[0]} and {pair[1]} collaborated {pair[2]['weight']} times")

top_actors = sorted(actor_graph.degree, key=lambda x: x[1], reverse=True)[:100]
actor_subgraph = actor_graph.subgraph([actor for actor, _ in top_actors])

# Visualize Actor-Actor Network with actor names
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(actor_subgraph, k=0.1)
nx.draw_networkx_nodes(actor_subgraph, pos, node_size=20, node_color="blue")
nx.draw_networkx_edges(actor_subgraph, pos, alpha=0.3, edge_color="gray")
nx.draw_networkx_labels(actor_subgraph, pos, font_size=8, font_color="black")
plt.title("Actor-Actor Collaboration Network", fontsize=16)
plt.axis("off")
plt.show()


# Genre Co-Occurrence
genre_graph = nx.Graph()
for genres in df['listed_in']:
    genre_list = [genre.strip() for genre in genres.split(',')]
    for genre1, genre2 in combinations(genre_list, 2):
        if genre_graph.has_edge(genre1, genre2):
            genre_graph[genre1][genre2]['weight'] += 1
        else:
            genre_graph.add_edge(genre1, genre2, weight=1)

# Top Genre Pairs
top_genre_pairs = sorted(genre_graph.edges(data=True), key=lambda x: x[2]['weight'], reverse=True)[:10]
print("\nTop Genre Pairs by Co-Occurrence:")
for pair in top_genre_pairs:
    print(f"{pair[0]} and {pair[1]} co-occurred {pair[2]['weight']} times")

# Visualize Genre Co-Occurrence Network
plt.figure(figsize=(12, 8))
pos = nx.spring_layout(genre_graph, k=0.3)
nx.draw_networkx_nodes(genre_graph, pos, node_size=300, node_color="green")
nx.draw_networkx_edges(genre_graph, pos, alpha=0.5, edge_color="gray")
nx.draw_networkx_labels(genre_graph, pos, font_size=10, font_color="black")
plt.title("Genre Co-Occurrence Network", fontsize=16)
plt.axis("off")
plt.show()
```

```
Top Actor Pairs by Collaboration:
Julie Tejwani and Rupa Bhimani collaborated 31 times
Julie Tejwani and Rajesh Kava collaborated 24 times
Rupa Bhimani and Rajesh Kava collaborated 22 times
Julie Tejwani and Jigna Bhardwaj collaborated 21 times
Rupa Bhimani and Jigna Bhardwaj collaborated 20 times
Jigna Bhardwaj and Rajesh Kava collaborated 20 times
Vatsal Dubey and Julie Tejwani collaborated 18 times
Vatsal Dubey and Rupa Bhimani collaborated 18 times
Vatsal Dubey and Jigna Bhardwaj collaborated 18 times
Vatsal Dubey and Rajesh Kava collaborated 17 times
```



Top Genre Pairs by Co-Occurrence:
Dramas and International Movies co-occurred 1476 times
International Movies and Comedies co-occurred 798 times
Dramas and Independent Movies co-occurred 587 times
International TV Shows and TV Dramas co-occurred 508 times
Dramas and Comedies co-occurred 502 times
International Movies and Action & Adventure co-occurred 393 times
International Movies and Romantic Movies co-occurred 366 times
International TV Shows and Romantic TV Shows co-occurred 309 times
Dramas and Romantic Movies co-occurred 303 times
Independent Movies and International Movies co-occurred 290 times