AAI_Final_Project___Movie_Recommendations_And_Sentiment_Analysis

August 9, 2025

0.1 Movies Recommendations and Sentiment Analysis

This notebook demonstrates how to perform sentiment analysis on movie reviews. We will train a model to classify movie reviews with sentiment analysis and scoring.

The process involves: 1. Loading and preprocessing the movie review dataset. 2. Building and training a deep learning model (e.g., using TensorFlow/Keras). 3. Evaluating the model's performance. 4. Using the trained model to predict sentiment on new movie reviews.

[90]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

[91]:
```
movies_df = pd.read_csv('/content/movies.csv')
```

[92]:
```
movies_df.head(-5)
```

[92]:
```
         budget                                             genres  \
0     237000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam…
1     300000000  [{"id": 12, "name": "Adventure"}, {"id": 14, "…
2     245000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam…
3     250000000  [{"id": 28, "name": "Action"}, {"id": 80, "nam…
4     260000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam…
…           …                                                …
4793          0                     [{"id": 18, "name": "Drama"}]
4794          0  [{"id": 53, "name": "Thriller"}, {"id": 27, "n…
4795          0                     [{"id": 18, "name": "Drama"}]
4796       7000  [{"id": 878, "name": "Science Fiction"}, {"id"…
4797          0  [{"id": 10769, "name": "Foreign"}, {"id": 53, …

                                           homepage      id  \
0                       http://www.avatarmovie.com/   19995
1      http://disney.go.com/disneypictures/pirates/     285
2       http://www.sonypictures.com/movies/spectre/  206647
```

```
3                http://www.thedarkknightrises.com/   49026
4              http://movies.disney.com/john-carter   49529
…                                               …       …
4793                                            NaN  182291
4794                                            NaN  286939
4795                                            NaN  124606
4796                      http://www.primermovie.com   14337
4797                                            NaN   67238

                                               keywords original_language  \
0     [{"id": 1463, "name": "culture clash"}, {"id":…                en
1     [{"id": 270, "name": "ocean"}, {"id": 726, "na…                en
2     [{"id": 470, "name": "spy"}, {"id": 818, "name…                en
3     [{"id": 849, "name": "dc comics"}, {"id": 853,…                en
4     [{"id": 818, "name": "based on novel"}, {"id":…                en
…                                                   …               …
4793  [{"id": 718, "name": "confession"}, {"id": 100…                en
4794                                               []                en
4795  [{"id": 10726, "name": "gang"}, {"id": 33928, …                en
4796  [{"id": 1448, "name": "distrust"}, {"id": 2101…                en
4797                                               []                en

                                original_title  \
0                                       Avatar
1     Pirates of the Caribbean: At World's End
2                                      Spectre
3                        The Dark Knight Rises
4                                  John Carter
…                                          …
4793                           On The Downlow
4794             Sanctuary: Quite a Conundrum
4795                                     Bang
4796                                   Primer
4797                                   Cavite

                                               overview  popularity  \
0     In the 22nd century, a paraplegic Marine is di…  150.437577
1     Captain Barbossa, long believed to be dead, ha…  139.082615
2     A cryptic message from Bond’s past sends him o…  107.376788
3     Following the death of District Attorney Harve…  112.312950
4     John Carter is a war-weary, former military ca…   43.926995
…                                                   …         …
4793  Isaac and Angel are two young Latinos involved…    0.029757
4794  It should have been just a normal day of sex, …    0.166513
4795  A young woman in L.A. is having a bad day: she…    0.918116
4796  Friends/fledgling entrepreneurs invent a devic…   23.307949
4797  Adam, a security guard, travels from Californi…    0.022173
```

```
                                    production_companies  \
0      [{"name": "Ingenious Film Partners", "id": 289…
1      [{"name": "Walt Disney Pictures", "id": 2}, {"…
2      [{"name": "Columbia Pictures", "id": 5}, {"nam…
3      [{"name": "Legendary Pictures", "id": 923}, {"…
4            [{"name": "Walt Disney Pictures", "id": 2}]
…                                                     …
4793         [{"name": "Iconoclast Films", "id": 26677}]
4794   [{"name": "Gold Lion Films", "id": 37870}, {"n…
4795   [{"name": "Asylum Films", "id": 10571}, {"name…
4796            [{"name": "Thinkfilm", "id": 446}]
4797                                                  []

                                    production_countries release_date  \
0      [{"iso_3166_1": "US", "name": "United States o…   2009-12-10
1      [{"iso_3166_1": "US", "name": "United States o…   2007-05-19
2      [{"iso_3166_1": "GB", "name": "United Kingdom"…   2015-10-26
3      [{"iso_3166_1": "US", "name": "United States o…   2012-07-16
4      [{"iso_3166_1": "US", "name": "United States o…   2012-03-07
…                                                 …          …
4793   [{"iso_3166_1": "US", "name": "United States o…   2004-04-11
4794   [{"iso_3166_1": "US", "name": "United States o…   2012-01-20
4795   [{"iso_3166_1": "US", "name": "United States o…   1995-09-09
4796   [{"iso_3166_1": "US", "name": "United States o…   2004-10-08
4797                                                  []   2005-03-12

         revenue  runtime                                 spoken_languages  \
0     2787965087    162.0  [{"iso_639_1": "en", "name": "English"}, {"iso…
1      961000000    169.0           [{"iso_639_1": "en", "name": "English"}]
2      880674609    148.0  [{"iso_639_1": "fr", "name": "Fran\u00e7ais"},…
3     1084939099    165.0           [{"iso_639_1": "en", "name": "English"}]
4      284139100    132.0           [{"iso_639_1": "en", "name": "English"}]
…            …       …                                                 …
4793           0     90.0                                                []
4794           0     82.0           [{"iso_639_1": "en", "name": "English"}]
4795           0     98.0           [{"iso_639_1": "en", "name": "English"}]
4796      424760     77.0           [{"iso_639_1": "en", "name": "English"}]
4797           0     80.0                                                []

        status                                       tagline  \
0     Released                   Enter the World of Pandora.
1     Released  At the end of the world, the adventure begins.
2     Released                          A Plan No One Escapes
3     Released                                The Legend Ends
4     Released             Lost in our world, found in another.
…         …                                             …
```

```
4793  Released          Two gangs. One secret. One crossroad.
4794  Released                                            NaN
4795  Released        Sometimes you've got to break the rules
4796  Released             What happens if it actually works?
4797  Released                                            NaN

                                         title  vote_average  vote_count
0                                       Avatar           7.2       11800
1     Pirates of the Caribbean: At World's End           6.9        4500
2                                      Spectre           6.3        4466
3                        The Dark Knight Rises           7.6        9106
4                                  John Carter           6.1        2124
…                                          …           …         …
4793                           On The Downlow           6.0           2
4794               Sanctuary: Quite a Conundrum           0.0           0
4795                                      Bang           6.0           1
4796                                    Primer           6.9         658
4797                                    Cavite           7.5           2

[4798 rows x 20 columns]
```

[93]:

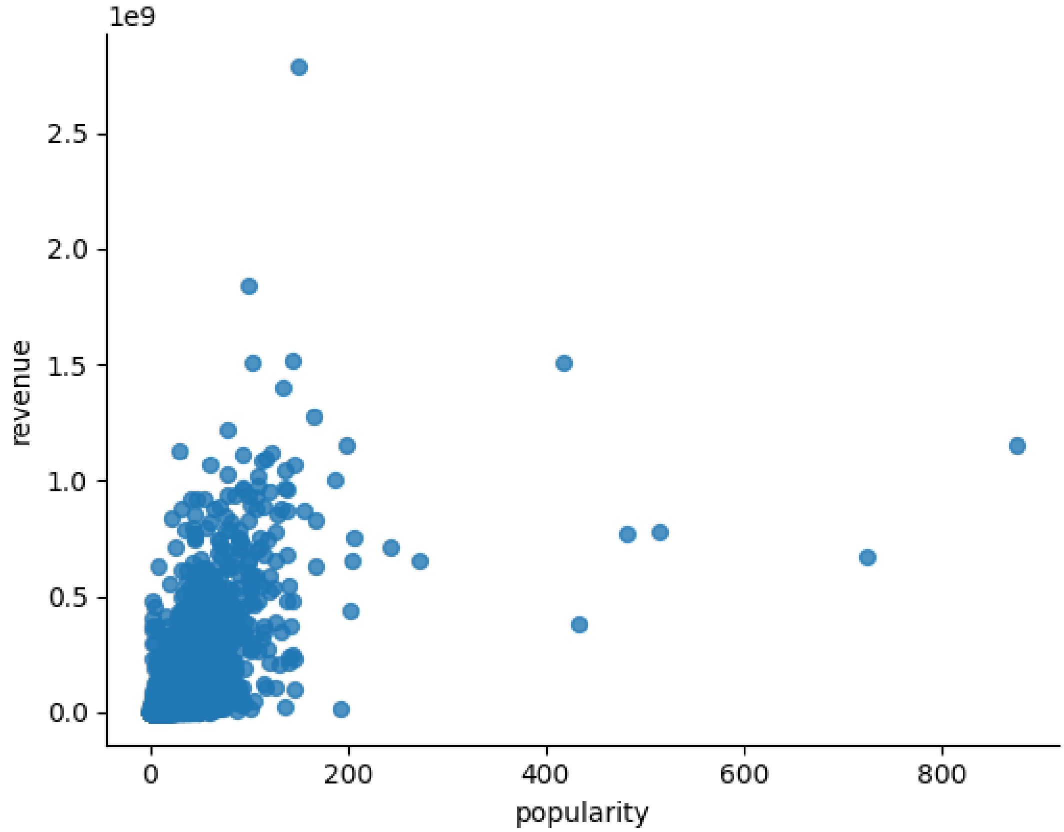
```python
credits_df = pd.read_csv('/content/credits.csv')
credits_df.head(-5)
```

[93]:

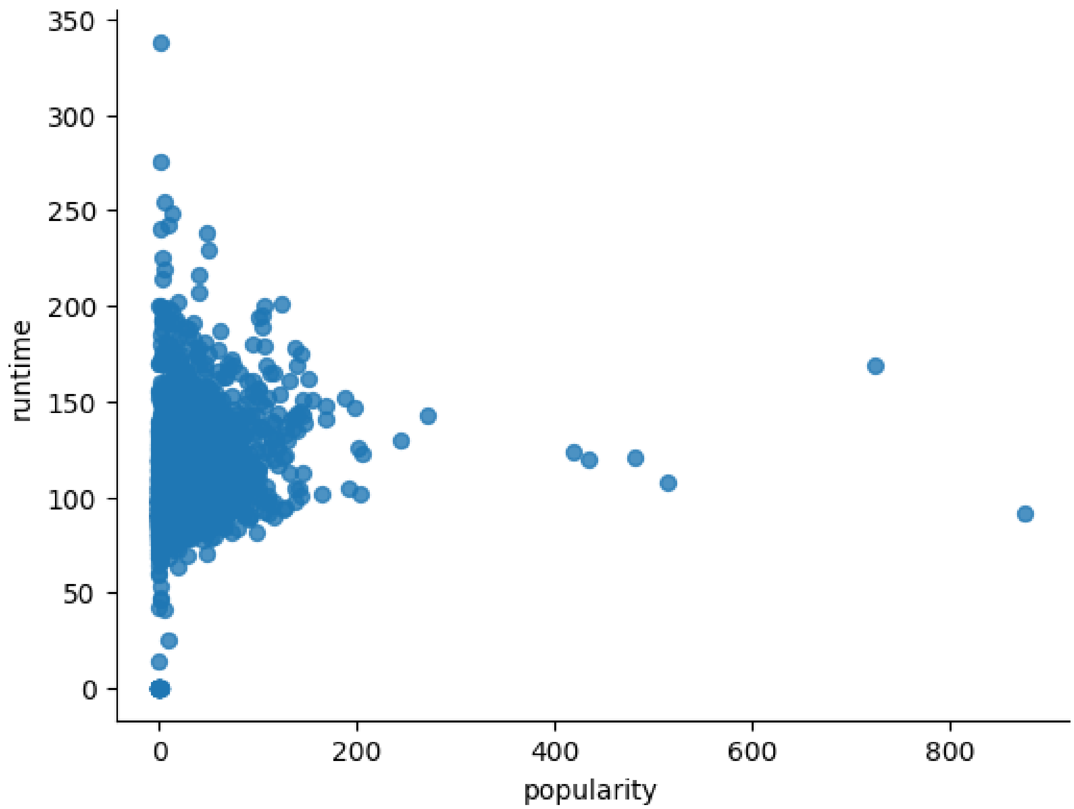
```
      movie_id                                     title  \
0        19995                                    Avatar
1          285  Pirates of the Caribbean: At World's End
2       206647                                   Spectre
3        49026                     The Dark Knight Rises
4        49529                               John Carter
…          …                                       …
4793    182291                            On The Downlow
4794    286939              Sanctuary: Quite a Conundrum
4795    124606                                      Bang
4796     14337                                    Primer
4797     67238                                    Cavite

                                                  cast  \
0     [{"cast_id": 242, "character": "Jake Sully", "…
1     [{"cast_id": 4, "character": "Captain Jack Spa…
2     [{"cast_id": 1, "character": "James Bond", "cr…
3     [{"cast_id": 2, "character": "Bruce Wayne / Ba…
4     [{"cast_id": 5, "character": "John Carter", "c…
…                                                  …
4793  [{"cast_id": 1, "character": "Isaac", "credit_…
4794  [{"cast_id": 3, "character": "Mimi", "credit_i…
4795  [{"cast_id": 2, "character": "The Girl", "cred…
```

```
4796  [{"cast_id": 1, "character": "Aaron", "credit_…
4797                                               []

                                               crew
0     [{"credit_id": "52fe48009251416c750aca23", "de…
1     [{"credit_id": "52fe4232c3a36847f800b579", "de…
2     [{"credit_id": "54805967c3a36829b5002c41", "de…
3     [{"credit_id": "52fe4781c3a36847f81398c3", "de…
4     [{"credit_id": "52fe479ac3a36847f813eaa3", "de…
…                                                …
4793  [{"credit_id": "548c416392514122ef00197d", "de…
4794  [{"credit_id": "545f8107c3a3686cbb0041fd", "de…
4795  [{"credit_id": "52fe4ab0c3a368484e161add", "de…
4796  [{"credit_id": "52fe45e79251416c75066791", "de…
4797  [{"credit_id": "52fe475dc3a368484e0c319f", "de…

[4798 rows x 4 columns]
```

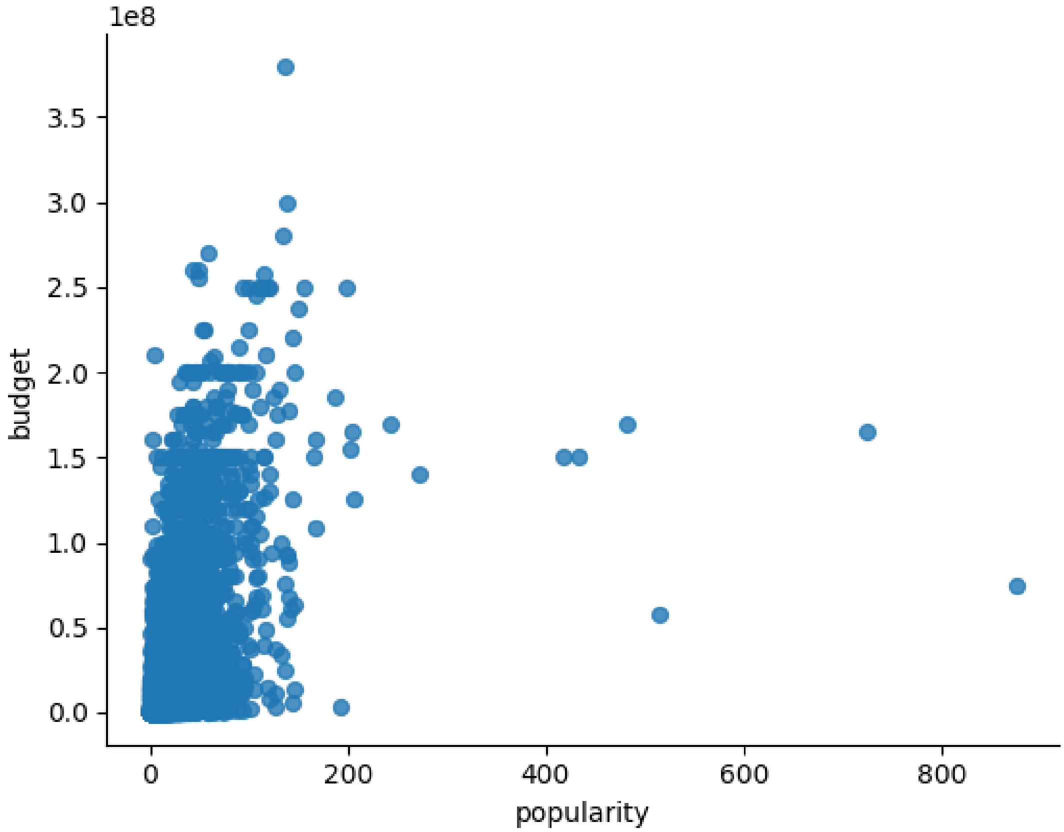
[94]:
```python
# @title popularity vs revenue

from matplotlib import pyplot as plt
movies_df.plot(kind='scatter', x='popularity', y='revenue', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



[95]:
```
# @title popularity vs runtime

from matplotlib import pyplot as plt
movies_df.plot(kind='scatter', x='popularity', y='runtime', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



[96]:
```
# @title popularity vs budget

from matplotlib import pyplot as plt
movies_df.plot(kind='scatter', x='popularity', y='budget', s=32, alpha=.8)
plt.gca().spines[['top', 'right',]].set_visible(False)
```



1 Exploratory Data Analysis

[97]:
```python
# Get the shape of each dataframe (number of rows and columns)
print("\nShape of Movies DataFrame:", movies_df.shape)
print("Shape of Credits DataFrame:", credits_df.shape)
```

```
Shape of Movies DataFrame: (4803, 20)
Shape of Credits DataFrame: (4803, 4)
```

[98]:
```python
# Get information about the data types and non-null values
print("\nInfo for Movies DataFrame:")
movies_df.info()

print("\nInfo for Credits DataFrame:")
credits_df.info()
```

```
Info for Movies DataFrame:
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   budget                4803 non-null   int64
 1   genres                4803 non-null   object
 2   homepage              1712 non-null   object
 3   id                    4803 non-null   int64
 4   keywords              4803 non-null   object
 5   original_language     4803 non-null   object
 6   original_title        4803 non-null   object
 7   overview              4800 non-null   object
 8   popularity            4803 non-null   float64
 9   production_companies  4803 non-null   object
 10  production_countries  4803 non-null   object
 11  release_date          4802 non-null   object
 12  revenue               4803 non-null   int64
 13  runtime               4801 non-null   float64
 14  spoken_languages      4803 non-null   object
 15  status                4803 non-null   object
 16  tagline               3959 non-null   object
 17  title                 4803 non-null   object
 18  vote_average          4803 non-null   float64
 19  vote_count            4803 non-null   int64
dtypes: float64(3), int64(4), object(13)
memory usage: 750.6+ KB

Info for Credits DataFrame:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4803 entries, 0 to 4802
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   movie_id  4803 non-null   int64
 1   title     4803 non-null   object
 2   cast      4803 non-null   object
 3   crew      4803 non-null   object
dtypes: int64(1), object(3)
memory usage: 150.2+ KB
```

[99]:

```
# Get descriptive statistics for numerical columns
print("\nDescription for Movies DataFrame:")
print(movies_df.describe())

print("\nDescription for Credits DataFrame:")
print(credits_df.describe())
```

```
Description for Movies DataFrame:
             budget             id   popularity       revenue      runtime  \
count  4.803000e+03    4803.000000  4803.000000  4.803000e+03  4801.000000
mean   2.904504e+07   57165.484281    21.492301  8.226064e+07   106.875859
std    4.072239e+07   88694.614033    31.816650  1.628571e+08    22.611935
min    0.000000e+00       5.000000     0.000000  0.000000e+00     0.000000
25%    7.900000e+05    9014.500000     4.668070  0.000000e+00    94.000000
50%    1.500000e+07   14629.000000    12.921594  1.917000e+07   103.000000
75%    4.000000e+07   58610.500000    28.313505  9.291719e+07   118.000000
max    3.800000e+08  459488.000000   875.581305  2.787965e+09   338.000000

       vote_average    vote_count
count   4803.000000   4803.000000
mean       6.092172    690.217989
std        1.194612   1234.585891
min        0.000000      0.000000
25%        5.600000     54.000000
50%        6.200000    235.000000
75%        6.800000    737.000000
max       10.000000  13752.000000

Description for Credits DataFrame:
            movie_id
count    4803.000000
mean    57165.484281
std     88694.614033
min         5.000000
25%      9014.500000
50%     14629.000000
75%     58610.500000
max    459488.000000
```

[100]:

```python
# Check for missing values
print("\nMissing values in Movies DataFrame:")
print(movies_df.isnull().sum())

print("\nMissing values in Credits DataFrame:")
print(credits_df.isnull().sum())
```

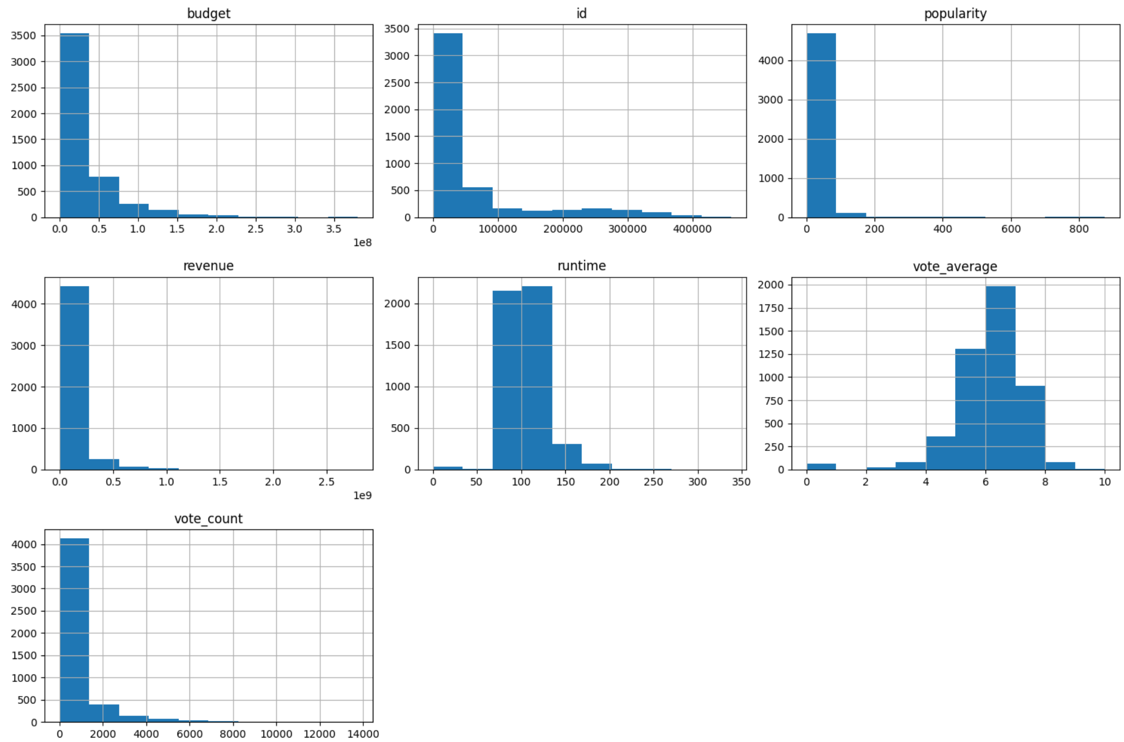
```
Missing values in Movies DataFrame:
budget                     0
genres                     0
homepage                3091
id                         0
keywords                   0
original_language          0
original_title             0
```

```
overview                   3
popularity                 0
production_companies       0
production_countries       0
release_date               1
revenue                    0
runtime                    2
spoken_languages           0
status                     0
tagline                  844
title                      0
vote_average               0
vote_count                 0
dtype: int64

Missing values in Credits DataFrame:
movie_id    0
title       0
cast        0
crew        0
dtype: int64
```

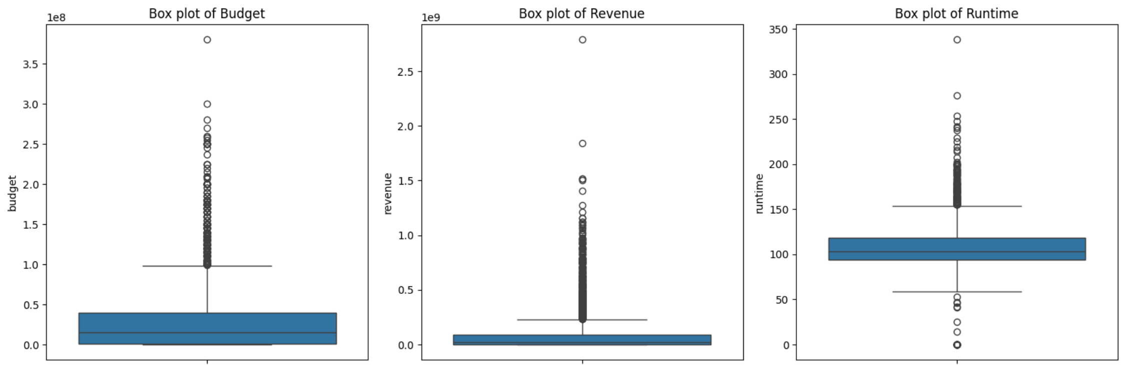
[101]:
```
# Check for duplicate rows
print("\nNumber of duplicate rows in Movies DataFrame:", movies_df.duplicated().
 ↪sum())
print("Number of duplicate rows in Credits DataFrame:", credits_df.duplicated().
 ↪sum())
```

```
Number of duplicate rows in Movies DataFrame: 0
Number of duplicate rows in Credits DataFrame: 0
```

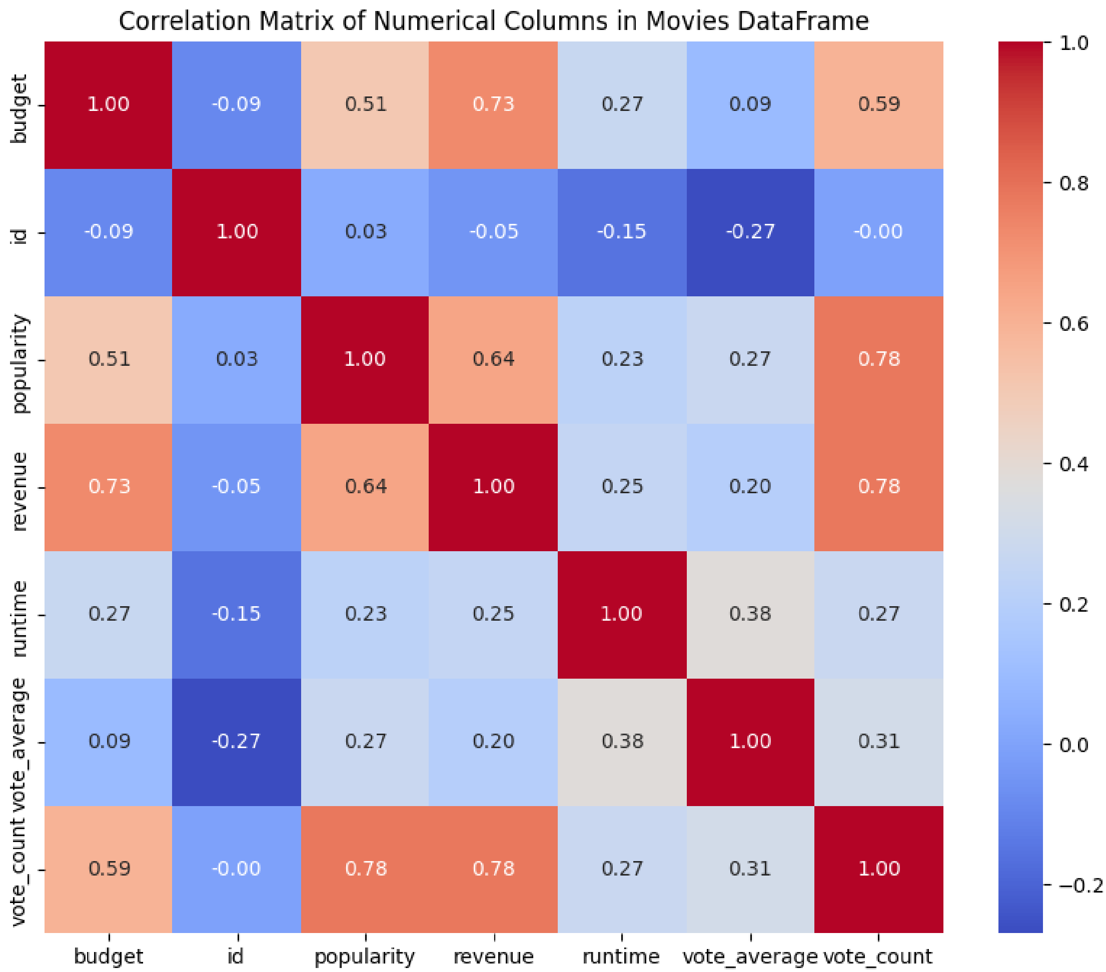
[102]:
```
# Explore the distribution of key columns using histograms
movies_df.hist(figsize=(15, 10))
plt.tight_layout()
plt.show()
```




```
[103]: import matplotlib.pyplot as plt
import seaborn as sns

# Explore the distribution of budget, revenue, and runtime using box plots
plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
sns.boxplot(y=movies_df['budget'])
plt.title('Box plot of Budget')

plt.subplot(1, 3, 2)
sns.boxplot(y=movies_df['revenue'])
plt.title('Box plot of Revenue')

plt.subplot(1, 3, 3)
sns.boxplot(y=movies_df['runtime'])
plt.title('Box plot of Runtime')
plt.tight_layout()
plt.show()
```



[104]:
```
# Visualize the correlation matrix of numerical columns in movies_df
plt.figure(figsize=(10, 8))
# Select only numerical columns for correlation matrix
numerical_movies_df = movies_df.select_dtypes(include=np.number)
sns.heatmap(numerical_movies_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Columns in Movies DataFrame')
plt.show()
```



[105]:
```
# @title Perform feature engineering with movies_df and credit_df to prepare␣
↪for generating recommendations
```

[106]:
```
# Combine the dataframes
movies_df['title'] = movies_df['title'].astype(str)
credits_df['title'] = credits_df['title'].astype(str)

# Merge the dataframes on the 'title' column
merged_df = movies_df.merge(credits_df, on='title')

# Drop irrelevant columns for recommendations
#merged_df.drop(['homepage', 'status', 'production_countries',␣
↪'spoken_languages', 'tagline', 'poster_path', 'production_companies'],␣
↪axis=1, inplace=True)
merged_df.drop(['homepage', 'spoken_languages', 'tagline'], axis=1,␣
↪inplace=True)
```

```python
# Handle missing values (example: fill NaNs in 'runtime' with the mean)
merged_df['runtime'].fillna(merged_df['runtime'].mean(), inplace=True)
merged_df['vote_average'].fillna(0, inplace=True)
merged_df['vote_count'].fillna(0, inplace=True)

# Extract relevant information from nested JSON strings
import json

def parse_json(json_string):
    try:
        list_of_dicts = json.loads(json_string)
        return [item['name'] for item in list_of_dicts]
    except (json.JSONDecodeError, TypeError):
        return []

merged_df['genres'] = merged_df['genres'].apply(parse_json)
merged_df['keywords'] = merged_df['keywords'].apply(parse_json)
merged_df['cast'] = merged_df['cast'].apply(parse_json)
merged_df['crew'] = merged_df['crew'].apply(parse_json)

# Keep only the director from the crew list
def get_director(crew_list):
    for item in crew_list:
        if item == 'Director':
            return item
    return None

merged_df['director'] = merged_df['crew'].apply(lambda x: [i for i in x if i in␣
↪['Director']])
merged_df['director'] = merged_df['director'].apply(lambda x: x[0] if x else␣
↪None)
merged_df.drop('crew', axis=1, inplace=True)


# Convert lists of strings into space-separated strings for easier processing
def list_to_string(lst):
    return ' '.join([str(i).replace(" ","") for i in lst])

for feature in ['genres', 'keywords', 'cast']:
    merged_df[feature] = merged_df[feature].apply(list_to_string)

merged_df['director'] = merged_df['director'].apply(lambda x: str(x).replace("␣
↪","") if x else '')

# Create a 'soup' of combined features for TF-IDF or Count Vectorizer
```

```
merged_df['soup'] = merged_df['title'] + merged_df['overview'].fillna('') + ' '␣
↪+ merged_df['genres'] + ' ' + merged_df['keywords'] + ' ' +␣
↪merged_df['cast'] + ' ' + merged_df['director'] + merged_df['release_date']
```

[107]:
```
merged_df.head(-5)
```

[107]:
```
         budget                                genres      id  \
0     237000000  Action Adventure Fantasy ScienceFiction   19995
1     300000000                 Adventure Fantasy Action     285
2     245000000                   Action Adventure Crime  206647
3     250000000              Action Crime Drama Thriller   49026
4     260000000          Action Adventure ScienceFiction   49529
…           …                                       …     …
4799          0                                    Drama  182291
4800          0                   Thriller Horror Comedy  286939
4801          0                                    Drama  124606
4802       7000            ScienceFiction Drama Thriller   14337
4803          0                         Foreign Thriller   67238

                                               keywords original_language  \
0     cultureclash future spacewar spacecolony socie…                en
1     ocean drugabuse exoticisland eastindiatradingc…                en
2     spy basedonnovel secretagent sequel mi6 britis…                en
3     dccomics crimefighter terrorist secretidentity…                en
4     basedonnovel mars medallion spacetravel prince…                en
…                                                 …               …
4799  confession hazing gangmember latino lgbt catho…                en
4800                                                                 en
4801          gang audition policefake homeless actress              en
4802  distrust garage identitycrisis timetravel time…                en
4803                                                                 en

                                original_title  \
0                                       Avatar
1     Pirates of the Caribbean: At World's End
2                                      Spectre
3                        The Dark Knight Rises
4                                  John Carter
…                                          …
4799                            On The Downlow
4800              Sanctuary: Quite a Conundrum
4801                                      Bang
4802                                    Primer
4803                                    Cavite

                                               overview  popularity  \
0     In the 22nd century, a paraplegic Marine is di…  150.437577
```

```
1     Captain Barbossa, long believed to be dead, ha…  139.082615
2     A cryptic message from Bond’s past sends him o…  107.376788
3     Following the death of District Attorney Harve…  112.312950
4     John Carter is a war-weary, former military ca…   43.926995
…                                                   …         …
4799  Isaac and Angel are two young Latinos involved…    0.029757
4800  It should have been just a normal day of sex, …    0.166513
4801  A young woman in L.A. is having a bad day: she…    0.918116
4802  Friends/fledgling entrepreneurs invent a devic…   23.307949
4803  Adam, a security guard, travels from Californi…    0.022173

                                   production_companies  \
0     [{"name": "Ingenious Film Partners", "id": 289…
1     [{"name": "Walt Disney Pictures", "id": 2}, {"…
2     [{"name": "Columbia Pictures", "id": 5}, {"nam…
3     [{"name": "Legendary Pictures", "id": 923}, {"…
4           [{"name": "Walt Disney Pictures", "id": 2}]
…                                                   …
4799        [{"name": "Iconoclast Films", "id": 26677}]
4800  [{"name": "Gold Lion Films", "id": 37870}, {"n…
4801  [{"name": "Asylum Films", "id": 10571}, {"name…
4802                 [{"name": "Thinkfilm", "id": 446}]
4803                                                 []

                                   production_countries  …     revenue  \
0     [{"iso_3166_1": "US", "name": "United States o…  …  2787965087
1     [{"iso_3166_1": "US", "name": "United States o…  …   961000000
2     [{"iso_3166_1": "GB", "name": "United Kingdom"…  …   880674609
3     [{"iso_3166_1": "US", "name": "United States o…  …  1084939099
4     [{"iso_3166_1": "US", "name": "United States o…  …   284139100
…                                                   …  …         …
4799  [{"iso_3166_1": "US", "name": "United States o…  …           0
4800  [{"iso_3166_1": "US", "name": "United States o…  …           0
4801  [{"iso_3166_1": "US", "name": "United States o…  …           0
4802  [{"iso_3166_1": "US", "name": "United States o…  …      424760
4803                                                 []  …           0

      runtime    status                                     title  \
0       162.0  Released                                    Avatar
1       169.0  Released  Pirates of the Caribbean: At World's End
2       148.0  Released                                   Spectre
3       165.0  Released                     The Dark Knight Rises
4       132.0  Released                               John Carter
…         …         …                                         …
4799     90.0  Released                           On The Downlow
4800     82.0  Released          Sanctuary: Quite a Conundrum
4801     98.0  Released                                      Bang
```

```
4802     77.0  Released                                    Primer
4803     80.0  Released                                    Cavite

     vote_average  vote_count  movie_id  \
0             7.2       11800     19995
1             6.9        4500       285
2             6.3        4466    206647
3             7.6        9106     49026
4             6.1        2124     49529
…             …           …        …
4799          6.0           2    182291
4800          0.0           0    286939
4801          6.0           1    124606
4802          6.9         658     14337
4803          7.5           2     67238

                                                  cast director  \
0     SamWorthington ZoeSaldana SigourneyWeaver Step…
1     JohnnyDepp OrlandoBloom KeiraKnightley Stellan…
2     DanielCraig ChristophWaltz LéaSeydoux RalphFie…
3     ChristianBale MichaelCaine GaryOldman AnneHath…
4     TaylorKitsch LynnCollins SamanthaMorton Willem…
…                                                  …        …
4799  TonySancho MichaelCortez DonatoCruz FelipeCama…
4800  SashaRamos ErinCline EmilyRogers AnthonyRutowi…
4801  DarlingNarita PeterGreene MichaelNewland ErikS…
4802  ShaneCarruth DavidSullivan CaseyGooden AnandUp…
4803


                                                  soup
0     AvatarIn the 22nd century, a paraplegic Marine…
1     Pirates of the Caribbean: At World's EndCaptai…
2     SpectreA cryptic message from Bond’s past send…
3     The Dark Knight RisesFollowing the death of Di…
4     John CarterJohn Carter is a war-weary, former …
…                                                  …
4799  On The DownlowIsaac and Angel are two young La…
4800  Sanctuary: Quite a ConundrumIt should have bee…
4801  BangA young woman in L.A. is having a bad day:…
4802  PrimerFriends/fledgling entrepreneurs invent a…
4803  CaviteAdam, a security guard, travels from Cal…

[4804 rows x 21 columns]
```

[108]:
```python
print("\nInfo after feature engineering:")
merged_df.info()
```

```
Info after feature engineering:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4809 entries, 0 to 4808
Data columns (total 21 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   budget                4809 non-null   int64  
 1   genres                4809 non-null   object 
 2   id                    4809 non-null   int64  
 3   keywords              4809 non-null   object 
 4   original_language     4809 non-null   object 
 5   original_title        4809 non-null   object 
 6   overview              4806 non-null   object 
 7   popularity            4809 non-null   float64
 8   production_companies  4809 non-null   object 
 9   production_countries  4809 non-null   object 
 10  release_date          4808 non-null   object 
 11  revenue               4809 non-null   int64  
 12  runtime               4809 non-null   float64
 13  status                4809 non-null   object 
 14  title                 4809 non-null   object 
 15  vote_average          4809 non-null   float64
 16  vote_count            4809 non-null   int64  
 17  movie_id              4809 non-null   int64  
 18  cast                  4809 non-null   object 
 19  director              4809 non-null   object 
 20  soup                  4808 non-null   object 
dtypes: float64(3), int64(5), object(13)
memory usage: 789.1+ KB
```

[109]:
```
!pip install nltk
```

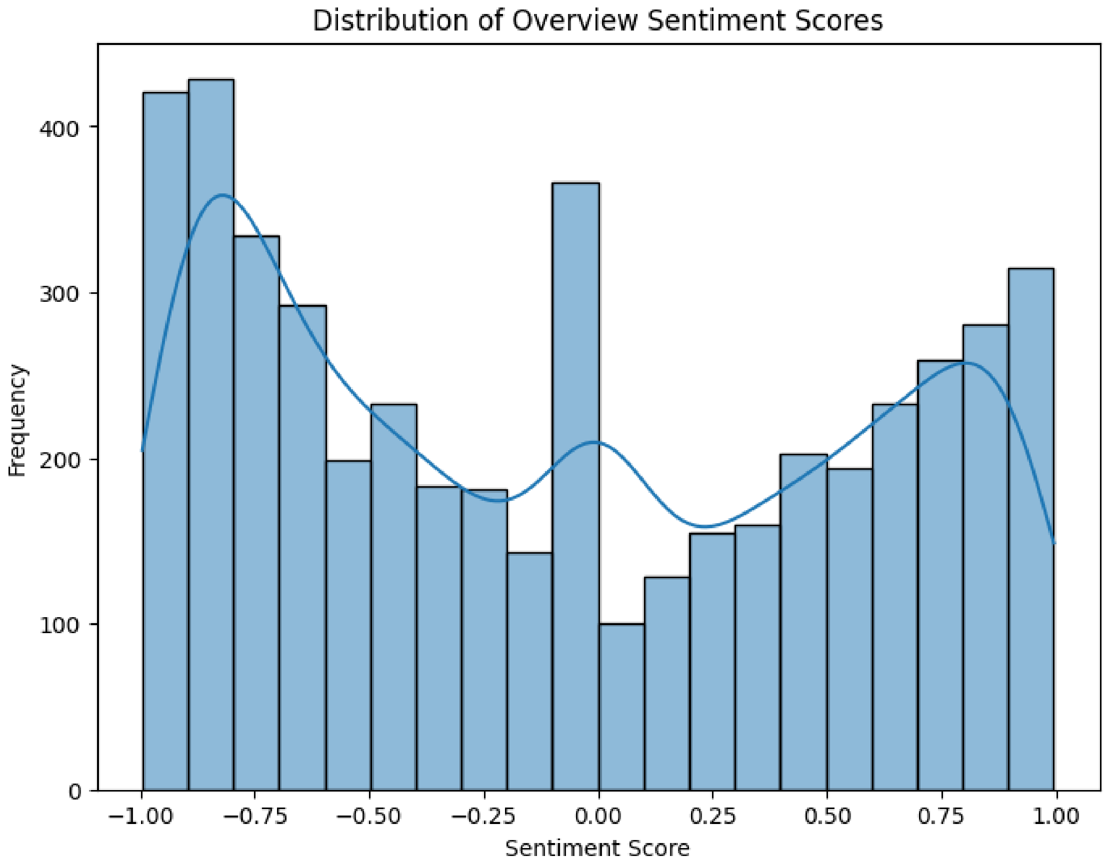
```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages
(3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages
(from nltk) (8.2.1)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages
(from nltk) (1.5.1)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages
(from nltk) (4.67.1)
```

[110]:
```
# @title perform sentiment analysis

import pandas as pd
```

```
import matplotlib.pyplot as plt
import nltk
nltk.download('vader_lexicon')

from nltk.sentiment import SentimentIntensityAnalyzer

# Initialize the VADER sentiment analyzer
analyzer = SentimentIntensityAnalyzer()

# Function to get sentiment score
def get_sentiment_score(text):
    if pd.isna(text):
        return 0 # Return 0 for missing overviews
    return analyzer.polarity_scores(str(text))['compound'] # Use compound score␣
↪as a single metric

# Apply the function to the 'overview' column and create a new column for␣
↪sentiment score
merged_df['overview_sentiment_score'] = merged_df['overview'].
↪apply(get_sentiment_score)

print("\nDataFrame with Sentiment Scores:")
print(merged_df[['title', 'overview', 'overview_sentiment_score']].head())

# Optional: Analyze the distribution of sentiment scores
plt.figure(figsize=(8, 6))
sns.histplot(merged_df['overview_sentiment_score'], bins=20, kde=True)
plt.title('Distribution of Overview Sentiment Scores')
plt.xlabel('Sentiment Score')
plt.ylabel('Frequency')
plt.show()
```

```
[nltk_data] Downloading package vader_lexicon to /root/nltk_data…
[nltk_data]   Package vader_lexicon is already up-to-date!


DataFrame with Sentiment Scores:
                                      title  \
0                                    Avatar
1  Pirates of the Caribbean: At World's End
2                                   Spectre
3                     The Dark Knight Rises
4                               John Carter

                                            overview  overview_sentiment_score
0  In the 22nd century, a paraplegic Marine is di…                   -0.3612
1  Captain Barbossa, long believed to be dead, ha…                   -0.3919
2  A cryptic message from Bond’s past sends him o…                   -0.8271
```

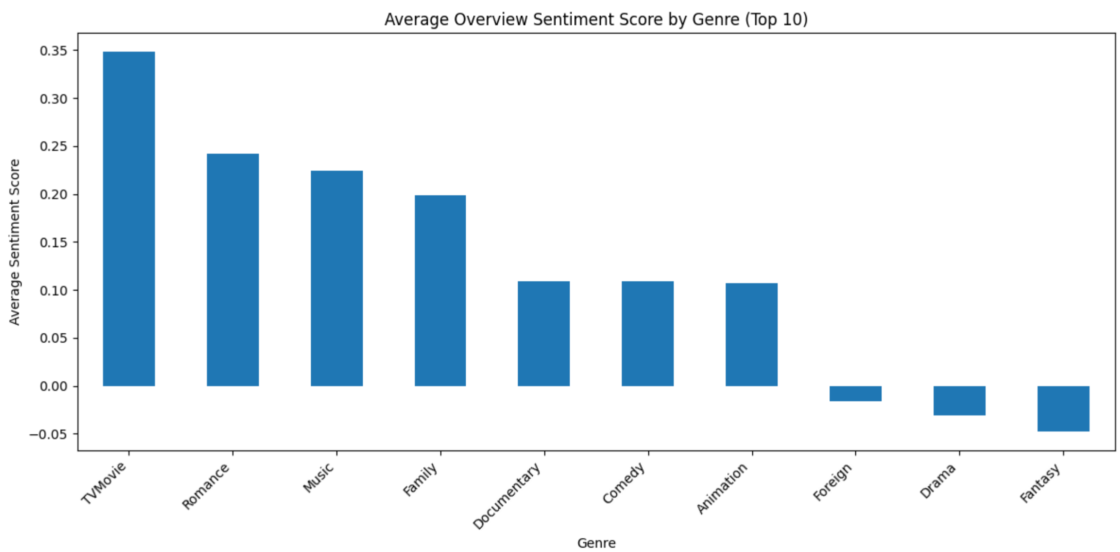
```
3  Following the death of District Attorney Harve…                    -0.9136
4  John Carter is a war-weary, former military ca…                    -0.7096
```



[111]:
```
# @title The average sentiment score by genre.

import pandas as pd
import matplotlib.pyplot as plt
# Average sentiment score by genre
# We need to first "explode" the genres list so that each movie's sentiment␣
↪score is associated with each its genres
genre_sentiment = merged_df[['genres', 'overview_sentiment_score']].copy()
genre_sentiment['genres'] = genre_sentiment['genres'].str.split()
genre_sentiment = genre_sentiment.explode('genres')

# Now calculate the average sentiment score for each genre
avg_sentiment_by_genre = genre_sentiment.
↪groupby('genres')['overview_sentiment_score'].mean().
↪sort_values(ascending=False)

print("\nAverage Sentiment Score by Genre:")
```

```
print(avg_sentiment_by_genre.head())

# Visualize average sentiment by genre (top N)
plt.figure(figsize=(12, 6))
avg_sentiment_by_genre.head(10).plot(kind='bar')
plt.title('Average Overview Sentiment Score by Genre (Top 10)')
plt.xlabel('Genre')
plt.ylabel('Average Sentiment Score')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

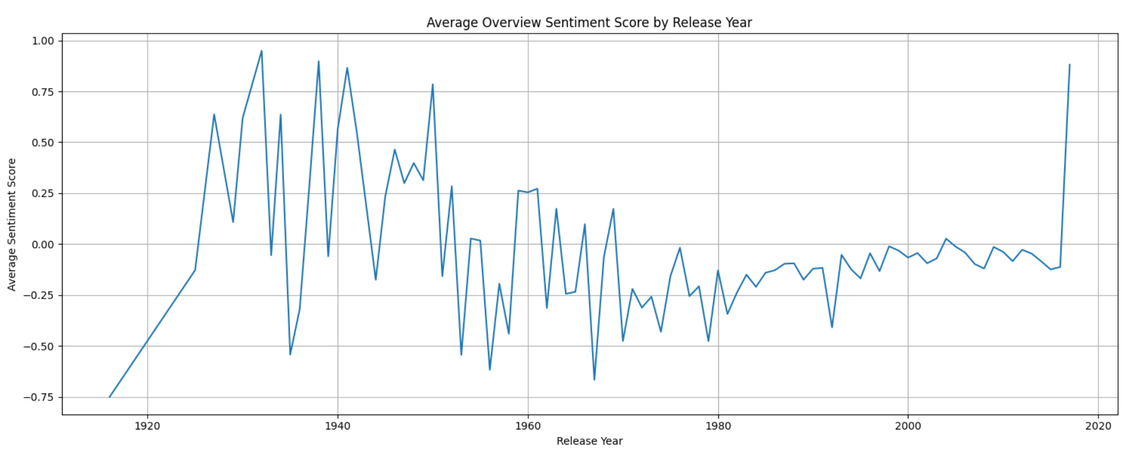
```
Average Sentiment Score by Genre:
genres
TVMovie        0.347888
Romance        0.242026
Music          0.224270
Family         0.198561
Documentary    0.109424
Name: overview_sentiment_score, dtype: float64
```



[112]:
```
# @title Average sentiment score by year
merged_df['release_date'] = pd.to_datetime(merged_df['release_date'],␣
↪errors='coerce')

# Extract the year
merged_df['release_year'] = merged_df['release_date'].dt.year
```

```python
# Drop rows with missing or invalid release years
sentiment_by_year_df = merged_df.dropna(subset=['release_year',␣
↪'overview_sentiment_score'])

# Group by year and calculate the mean sentiment score
avg_sentiment_by_year = sentiment_by_year_df.
↪groupby('release_year')['overview_sentiment_score'].mean().sort_index()

print("\nAverage Sentiment Score by Year:")
print(avg_sentiment_by_year.head())

# Visualize average sentiment by year
plt.figure(figsize=(15, 6))
avg_sentiment_by_year.plot(kind='line')
plt.title('Average Overview Sentiment Score by Release Year')
plt.xlabel('Release Year')
plt.ylabel('Average Sentiment Score')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
Average Sentiment Score by Year:
release_year
1916.0   -0.7506
1925.0   -0.1280
1927.0    0.6369
1929.0    0.1079
1930.0    0.6191
Name: overview_sentiment_score, dtype: float64
```



[113]:
```python
# @title Perform recommendations on sentiment score

import pandas as pd

# Function to get movie recommendations based on sentiment score
def recommend_by_sentiment(title, df, num_recommendations=5):
    # Find the index of the movie with the given title
    indices = pd.Series(df.index, index=df['title']).drop_duplicates()
    if title not in indices:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame() # Return an empty DataFrame if movie not found

    idx = indices[title]

    # Get the sentiment score of the input movie
    input_sentiment_score = df.loc[idx, 'overview_sentiment_score']

    # Calculate the absolute difference in sentiment scores between the input␣
 ↪movie and all other movies
    df['sentiment_difference'] = abs(df['overview_sentiment_score'] -␣
 ↪input_sentiment_score)

    # Sort movies based on the absolute difference in sentiment scores (closest␣
 ↪to the input movie's score)
    # Exclude the input movie itself
    recommended_movies = df.sort_values(by='sentiment_difference').
 ↪head(num_recommendations + 1)

    # Filter out the input movie
    recommended_movies = recommended_movies[recommended_movies['title'] !=␣
 ↪title]

    # Return the top recommendations
    return recommended_movies[['title', 'overview_sentiment_score',␣
 ↪'sentiment_difference']]
```

[114]:
```python
# @title Get recommendations for a movie based on its sentiment score
movie_title = 'Avatar' # movie title to generate recommendations
recommendations = recommend_by_sentiment(movie_title, merged_df)

print(f"\nRecommendations based on sentiment similarity for '{movie_title}':")
recommendations
```

```
Recommendations based on sentiment similarity for 'Avatar':
```

[114]:

```
                                    title  overview_sentiment_score  \
2940                             Out Cold                   -0.3612
134   Mission: Impossible - Rogue Nation                   -0.3612
4563                  Fight to the Finish                   -0.3612
766        Garfield: A Tail of Two Kitties                   -0.3612
748                              Year One                   -0.3612

      sentiment_difference
2940                   0.0
134                    0.0
4563                   0.0
766                    0.0
748                    0.0
```

[115]:

```
movie_title = 'Liar Liar' # movie title to generate recommendations
recommendations = recommend_by_sentiment(movie_title, merged_df)

print(f"\nRecommendations based on sentiment similarity for '{movie_title}':")
recommendations
```

```
Recommendations based on sentiment similarity for 'Liar Liar':
```

[115]:

```
                                    title  overview_sentiment_score  \
0                                  Avatar                   -0.3612
2940                             Out Cold                   -0.3612
134   Mission: Impossible - Rogue Nation                   -0.3612
4563                  Fight to the Finish                   -0.3612
766        Garfield: A Tail of Two Kitties                   -0.3612
748                              Year One                   -0.3612

      sentiment_difference
0                      0.0
2940                   0.0
134                    0.0
4563                   0.0
766                    0.0
748                    0.0
```

[116]:

```
# @title Generate recommendation with plot overview keywords based on the␣
↪sentiment score

import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel

# Initialize the TfidfVectorizer
```

```
# Use the 'soup' column which contains combined text features (overview,␣
↪genres, keywords, cast, director)
tfidf = TfidfVectorizer(
    stop_words="english",
    ngram_range=(1, 2),
    max_features=86621,
)

# Construct the TF-IDF matrix
tfidf_matrix = tfidf.fit_transform(merged_df['soup'].fillna(''))

print("\nShape of TF-IDF matrix:", tfidf_matrix.shape)

# Calculate the cosine similarity matrix
# This measures the similarity between movie 'soups'
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)

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

# Create a reverse mapping of movie titles to their indices
indices = pd.Series(merged_df.index, index=merged_df['title']).drop_duplicates()

# Function to get recommendations based on cosine similarity of the 'soup'
def get_content_based_recommendations(title, df, cosine_sim=cosine_sim,␣
↪num_recommendations=10):
    # Get the index of the movie that matches the title
    if title not in indices:
        print(f"Movie '{title}' not found in the dataset for content-based␣
↪recommendations.")
        return pd.DataFrame()

    idx = indices[title]

    # Get the pairwise similarity scores for all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the num_recommendations most similar movies
    # Skip the first element as it is the movie itself
    sim_scores = sim_scores[1:num_recommendations+1]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top num_recommendations most similar movies
```

```python
    return df[['title', 'genres', 'keywords', 'overview_sentiment_score']].iloc[movie_indices]

# Function to generate combined recommendations considering both sentiment and content
def get_combined_recommendations(title, df, cosine_sim=cosine_sim, num_recommendations=5, sentiment_weight=0.5, content_weight=0.5):
    if title not in indices:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame()

    idx = indices[title]
    input_sentiment_score = df.loc[idx, 'overview_sentiment_score']

    # Get sentiment similarity scores (closer to 0 difference is better)
    # We need to invert this difference to get a similarity score (higher is better)
    # A simple inversion could be 1 - abs_difference, but scaling might be needed
    # For now, let's use the inverse of the rank based on absolute difference
    df_temp = df.copy()
    df_temp['sentiment_difference'] = abs(df_temp['overview_sentiment_score'] - input_sentiment_score)
    df_temp['sentiment_rank'] = df_temp['sentiment_difference'].rank(method='min', ascending=True)
    # Normalize sentiment rank (higher rank = less similar, so invert)
    df_temp['normalized_sentiment_sim'] = 1 / df_temp['sentiment_rank']
    df_temp['normalized_sentiment_sim'] = df_temp['normalized_sentiment_sim'] / df_temp['normalized_sentiment_sim'].max() # Normalize to 0-1


    # Get content similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Convert similarity scores to a Series
    content_sim_series = pd.Series([score for index, score in sim_scores])
    df_temp['content_sim'] = content_sim_series
    # Normalize content similarity
    df_temp['normalized_content_sim'] = df_temp['content_sim'] / df_temp['content_sim'].max()


    # Combine scores using weights
    df_temp['combined_score'] = (df_temp['normalized_sentiment_sim'] * sentiment_weight) + (df_temp['normalized_content_sim'] * content_weight)

    # Sort movies based on the combined score
```

```python
    # Exclude the input movie itself
    recommended_movies = df_temp.sort_values(by='combined_score',␣
↪ascending=False).head(num_recommendations + 1)

    # Filter out the input movie
    recommended_movies = recommended_movies[recommended_movies['title'] !=␣
↪title]

    # Return the top recommendations with relevant information
    return recommended_movies[['title', 'genres', 'keywords',␣
↪'overview_sentiment_score', 'combined_score']].reset_index(drop=True)

# @title Get combined recommendations based on content similarity and sentiment␣
↪score
movie_title_for_combined = 'Avatar'  #@param {type:"string"}
sentiment_weight = 0.5  #@param {type:"slider", min:0.0, max:1.0, step:0.1}
content_weight = 0.7  #@param {type:"slider", min:0.0, max:1.0, step:0.1}
num_recommendations_combined = 5  #@param {type:"slider", min:1, max:20, step:1}


combined_recommendations = get_combined_recommendations(
    movie_title_for_combined,
    merged_df,
    cosine_sim=cosine_sim,
    num_recommendations=num_recommendations_combined,
    sentiment_weight=sentiment_weight,
    content_weight=content_weight
)

print(f"\nCombined recommendations (Sentiment weight: {sentiment_weight},␣
↪Content weight: {content_weight}) for '{movie_title_for_combined}':")
combined_recommendations
```

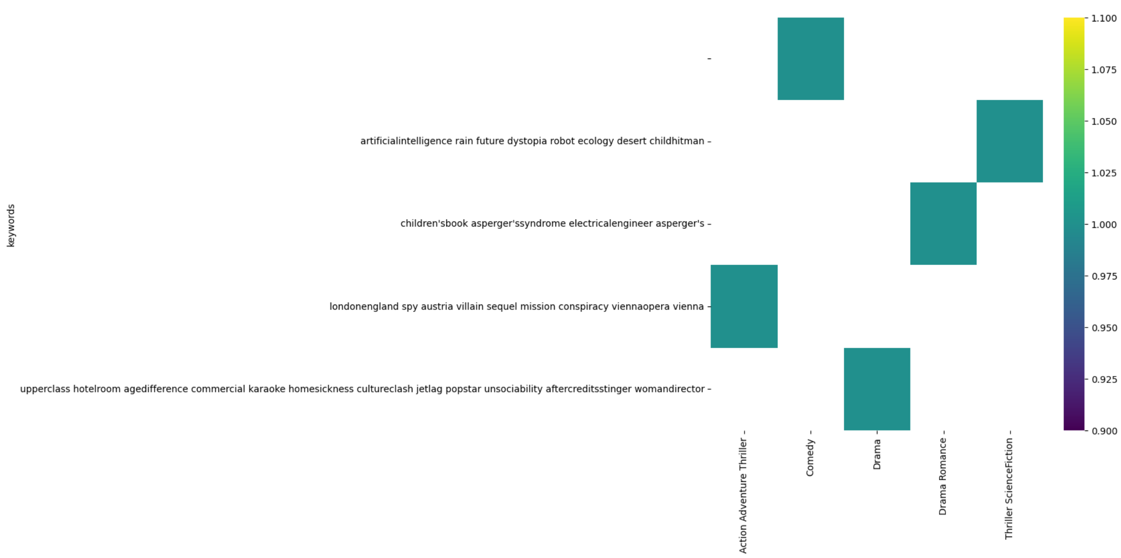
```
Shape of TF-IDF matrix: (4809, 86621)
Shape of Cosine Similarity matrix: (4809, 4809)

Combined recommendations (Sentiment weight: 0.5, Content weight: 0.7) for
'Avatar':
```

[116]:

```
                              title                      genres  \
0                Lost in Translation                       Drama
1  Mission: Impossible - Rogue Nation  Action Adventure Thriller
2                            Automata     Thriller ScienceFiction
3                   Parental Guidance                      Comedy
4                                Adam               Drama Romance
```

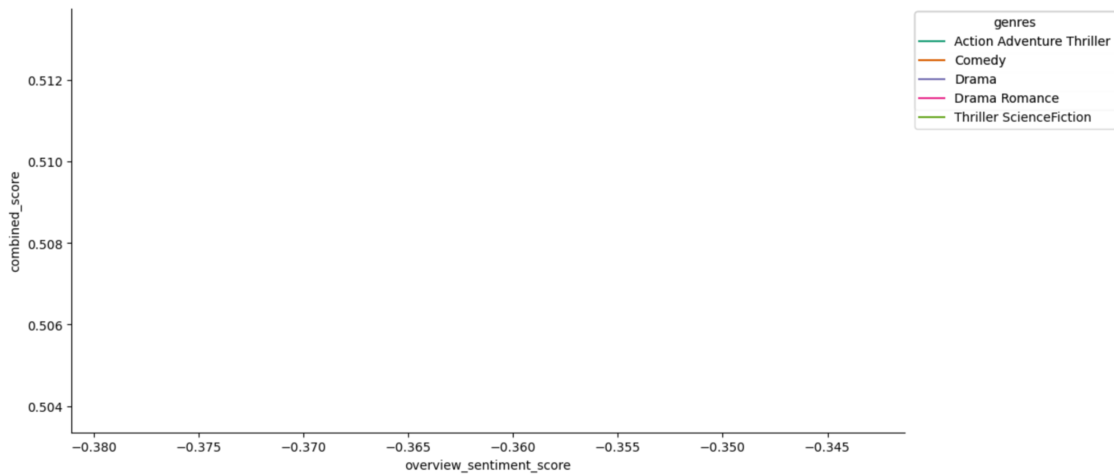
```
                                            keywords  \
0  upperclass hotelroom agedifference commercial …
1  londonengland spy austria villain sequel missi…
2  artificialintelligence rain future dystopia ro…
3
4  children'sbook asperger'ssyndrome electricalen…

   overview_sentiment_score  combined_score
0                   -0.3612        0.513274
1                   -0.3612        0.511415
2                   -0.3612        0.509193
3                   -0.3612        0.505805
4                   -0.3612        0.503830
```

[117]:
```python
# @title genres vs keywords

from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
plt.subplots(figsize=(8, 8))
df_2dhist = pd.DataFrame({
    x_label: grp['keywords'].value_counts()
    for x_label, grp in combined_recommendations.groupby('genres')
})
sns.heatmap(df_2dhist, cmap='viridis')
plt.xlabel('genres')
_ = plt.ylabel('keywords')
```



[118]:

```
# @title Plot overview_sentiment_score vs combined_score

from matplotlib import pyplot as plt
import seaborn as sns
def _plot_series(series, series_name, series_index=0):
  palette = list(sns.palettes.mpl_palette('Dark2'))
  xs = series['overview_sentiment_score']
  ys = series['combined_score']

  plt.plot(xs, ys, label=series_name, color=palette[series_index %␣
↪len(palette)])

fig, ax = plt.subplots(figsize=(10, 5.2), layout='constrained')
df_sorted = combined_recommendations.sort_values('overview_sentiment_score',␣
↪ascending=True)
for i, (series_name, series) in enumerate(df_sorted.groupby('genres')):
  _plot_series(series, series_name, i)
  fig.legend(title='genres', bbox_to_anchor=(1, 1), loc='upper left')
sns.despine(fig=fig, ax=ax)
plt.xlabel('overview_sentiment_score')
_ = plt.ylabel('combined_score')
```



1.1 Generate recommendation on movies with reasons for recommending

[119]:

```
# Generate recommendations with reasons by movie name based on the sentiment␣
↪score and print it as table

def generate_recommendations_with_reasons(title, df, cosine_sim,␣
↪num_recommendations=5, sentiment_weight=0.5, content_weight=0.5):
    """
```

```
    Generates movie recommendations based on combined sentiment and content␣
↪similarity,
    providing reasons for each recommendation.

    Args:
        title (str): The title of the input movie.
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df).
        cosine_sim (np.array): The cosine similarity matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 5.
        sentiment_weight (float, optional): Weight for sentiment similarity.␣
↪Defaults to 0.5.
        content_weight (float, optional): Weight for content similarity.␣
↪Defaults to 0.5.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies and reasons.
    """
    if title not in indices:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame()

    idx = indices[title]
    input_sentiment_score = df.loc[idx, 'overview_sentiment_score']
    input_genres = df.loc[idx, 'genres']
    input_keywords = df.loc[idx, 'keywords']


    # Get sentiment similarity scores
    df_temp = df.copy()
    df_temp['sentiment_difference'] = abs(df_temp['overview_sentiment_score'] -␣
↪input_sentiment_score)
    df_temp['sentiment_rank'] = df_temp['sentiment_difference'].
↪rank(method='min', ascending=True)
    df_temp['normalized_sentiment_sim'] = 1 / df_temp['sentiment_rank']
    df_temp['normalized_sentiment_sim'] = df_temp['normalized_sentiment_sim'] /␣
↪df_temp['normalized_sentiment_sim'].max()


    # Get content similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    content_sim_series = pd.Series([score for index, score in sim_scores])
    df_temp['content_sim'] = content_sim_series
    df_temp['normalized_content_sim'] = df_temp['content_sim'] /␣
↪df_temp['content_sim'].max()
```

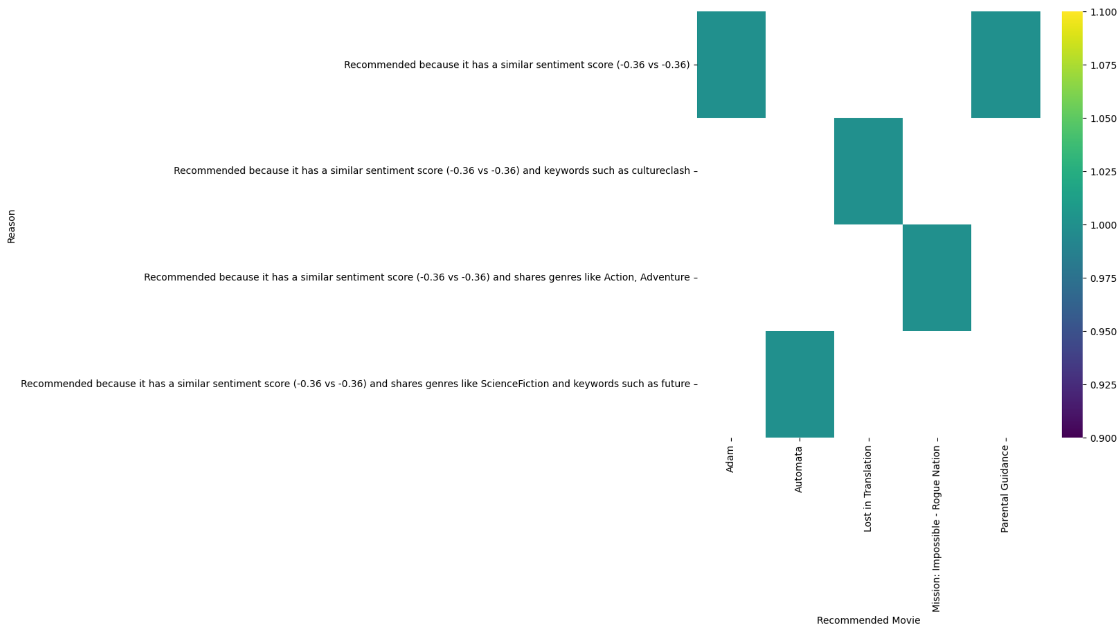
```
    # Combine scores using weights
    df_temp['combined_score'] = (df_temp['normalized_sentiment_sim'] *␣
↪sentiment_weight) + (df_temp['normalized_content_sim'] * content_weight)

    # Sort movies based on the combined score
    recommended_movies = df_temp.sort_values(by='combined_score',␣
↪ascending=False).head(num_recommendations + 1)

    # Filter out the input movie
    recommended_movies = recommended_movies[recommended_movies['title'] !=␣
↪title].reset_index(drop=True)

    # Generate reasons for recommendation
    recommendations_with_reasons = []
    for i, row in recommended_movies.iterrows():
        reason = f"Recommended because it has a similar sentiment score␣
↪({row['overview_sentiment_score']:.2f} vs {input_sentiment_score:.2f})"

        # Add reasons based on content similarity (genres, keywords)
        rec_genres = row['genres']
        rec_keywords = row['keywords']

        shared_genres = set(input_genres.split()) & set(rec_genres.split())
        shared_keywords = set(input_keywords.split()) & set(rec_keywords.
↪split())

        if shared_genres:
            reason += f" and shares genres like {', '.join(list(shared_genres)[:
↪3])}" # Show up to 3 shared genres
        if shared_keywords:
             reason += f" and keywords such as {', '.
↪join(list(shared_keywords)[:3])}" # Show up to 3 shared keywords


        recommendations_with_reasons.append({
            'Recommended Movie': row['title'],
            'Reason': reason,
            'Sentiment Score': row['overview_sentiment_score'],
            'Combined Score': row['combined_score']
        })

    return pd.DataFrame(recommendations_with_reasons)

# @title Generate recommendations with reasons for a specific movie title
```

```
movie_title_for_reasons = 'Avatar'  #@param {type:"string"}
sentiment_weight_reasons = 0.5  #@param {type:"slider", min:0.0, max:1.0, step:
 ↪0.1}
content_weight_reasons = 0.5  #@param {type:"slider", min:0.0, max:1.0, step:0.
 ↪1}
num_recommendations_reasons = 5  #@param {type:"slider", min:1, max:20, step:1}


recommendations_table = generate_recommendations_with_reasons(
    movie_title_for_reasons,
    merged_df,
    cosine_sim,
    num_recommendations=num_recommendations_reasons,
    sentiment_weight=sentiment_weight_reasons,
    content_weight=content_weight_reasons
)

print(f"\nRecommendations and Reasons for '{movie_title_for_reasons}':")
from IPython.display import display
display(recommendations_table)
```

```
Recommendations and Reasons for 'Avatar':

                    Recommended Movie  \
0                  Lost in Translation
1  Mission: Impossible - Rogue Nation
2                             Automata
3                    Parental Guidance
4                                 Adam

                                              Reason  Sentiment Score  \
0  Recommended because it has a similar sentiment…          -0.3612
1  Recommended because it has a similar sentiment…          -0.3612
2  Recommended because it has a similar sentiment…          -0.3612
3  Recommended because it has a similar sentiment…          -0.3612
4  Recommended because it has a similar sentiment…          -0.3612

   Combined Score
0        0.509481
1        0.508154
2        0.506566
3        0.504146
4        0.502736
```

[120]:

```
# @title Recommended Movie vs Reason

from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
plt.subplots(figsize=(8, 8))
df_2dhist = pd.DataFrame({
    x_label: grp['Reason'].value_counts()
    for x_label, grp in recommendations_table.groupby('Recommended Movie')
})
sns.heatmap(df_2dhist, cmap='viridis')
plt.xlabel('Recommended Movie')
_ = plt.ylabel('Reason')
```



[121]:

```
# generate movie recommendation with keywords for movie title or plot overview ␣
↪and also print the confidence score as well

def generate_recommendations_by_keyword(keyword, df, cosine_sim=cosine_sim,␣
↪num_recommendations=10):
    # Use the TF-IDF vectorizer to transform the keyword into a vector
    # We need to fit the vectorizer first if it hasn't been fitted on the␣
↪entire corpus

    # Transform the input keyword/query
    keyword_vec = tfidf.transform([keyword])
```

```
    # Calculate cosine similarity between the keyword vector and all movie soup␣
↪vectors
    keyword_sim_scores = linear_kernel(keyword_vec, tfidf_matrix).flatten()

    # Get the pairwise similarity scores as a list of (index, score) tuples
    sim_scores = list(enumerate(keyword_sim_scores))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the num_recommendations most similar movies
    # We take from the beginning since the input is not a movie itself
    sim_scores = sim_scores[:num_recommendations]

    # Get the movie indices and their confidence scores (similarity score)
    movie_indices = [(i[0], i[1]) for i in sim_scores]

    # Create a list of recommended movies and their confidence scores
    recommendations_list = []
    for idx, confidence in movie_indices:
        recommendations_list.append({
            'title': df['title'].iloc[idx],
            'overview': df['overview'].iloc[idx],
            'genres': df['genres'].iloc[idx],
            'keywords': df['keywords'].iloc[idx],
            'confidence_score': confidence # Confidence score is the cosine␣
↪similarity
        })

    return pd.DataFrame(recommendations_list)

# @title Generate recommendations by keyword and print confidence score
search_keyword = 'chocolate'  #@param {type:"string"}
num_recommendations_keyword = 10  #@param {type:"slider", min:1, max:20, step:1}

keyword_recommendations = generate_recommendations_by_keyword(
    search_keyword,
    merged_df,
    cosine_sim=cosine_sim,
    num_recommendations=num_recommendations_keyword
)

print(f"\nRecommendations based on the keyword '{search_keyword}':")
keyword_recommendations
```

```
Recommendations based on the keyword 'chocolate':
```

[121]:

```
                                       title  \
0                                   Chocolat
1          Charlie and the Chocolate Factory
2        Willy Wonka & the Chocolate Factory
3               Chocolate: Deep Dark Secrets
4                        Blood and Chocolate
5                                 Epic Movie
6                                     Avatar
7   Pirates of the Caribbean: At World's End
8                                    Spectre
9                      The Dark Knight Rises

                                            overview  \
0  A fable of emotional liberation and chocolate…
1  A young boy wins a tour through the most magni…
2  Eccentric candy man Willy Wonka prompts a worl…
3  Christmas Eve, London. While the snow-clad cit…
4  A young teenage werewolf is torn between honor…
5  When Edward, Peter, Lucy and Susan each follow…
6  In the 22nd century, a paraplegic Marine is di…
7  Captain Barbossa, long believed to be dead, ha…
8  A cryptic message from Bond’s past sends him o…
9  Following the death of District Attorney Harve…

                                      genres  \
0                      Comedy Drama Romance
1          Adventure Comedy Family Fantasy
2                            Family Fantasy
3                                  Thriller
4              Drama Fantasy Horror Romance
5                    Action Adventure Comedy
6  Action Adventure Fantasy ScienceFiction
7                   Adventure Fantasy Action
8                     Action Adventure Crime
9               Action Crime Drama Thriller

                                            keywords  confidence_score
0  chocolate mayor praline single motherdaughterr…          0.371180
1  londonengland fathersonrelationship chocolate …          0.342040
2  chocolate factoryworker basedonnovel candy tva…          0.308086
3                                                           0.245410
4  chocolate werewolf womandirector interspeciesr…          0.154795
5                                                           0.115103
6  cultureclash future spacewar spacecolony socie…          0.000000
7  ocean drugabuse exoticisland eastindiatradingc…          0.000000
8  spy basedonnovel secretagent sequel mi6 britis…          0.000000
9  dccomics crimefighter terrorist secretidentity…          0.000000
```

[122]:
```python
# Movie recommendations for any of the below condition matches and display the reason with confidence score
# 1. movie title or partial movie name
# 2. movie keyword
# 3. plot overview
# 4. actor name or partial actor name
# 5. release year
# 6. country
# 7. language

import pandas as pd
import numpy as np
def generate_recommendations(query, df, cosine_sim, num_recommendations=5, sentiment_weight=0.5, content_weight=0.5):
    """
    Generates movie recommendations based on various criteria (title, keyword, plot, actor, year, country, language).

    Args:
        query (str or int): The input query (movie title, keyword, year, etc.).
        df (pd.DataFrame): The DataFrame containing movie information (merged_df).
        cosine_sim (np.array): The cosine similarity matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to generate. Defaults to 5.
        sentiment_weight (float, optional): Weight for sentiment similarity (used for title/overview match). Defaults to 0.5.
        content_weight (float, optional): Weight for content similarity (used for title/overview match). Defaults to 0.5.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies, reason, and confidence score.
                      Returns an empty DataFrame if no matches are found.
    """
    results = []

    # --- 1. Match by Movie Title (Partial or Full) ---
    # Find movies where the title contains the query (case-insensitive)
    title_matches = df[df['title'].str.contains(str(query), case=False, na=False)]
    if not title_matches.empty:
        # If an exact match is found, use content-based/sentiment recommendations
        exact_match = title_matches[title_matches['title'].str.lower() == str(query).lower()]
```

```
        if not exact_match.empty:
            movie_title = exact_match['title'].iloc[0]
            idx = indices[movie_title]
            input_sentiment_score = df.loc[idx, 'overview_sentiment_score']
            input_genres = df.loc[idx, 'genres']
            input_keywords = df.loc[idx, 'keywords']

            # Calculate combined scores as done in get_combined_recommendations
            df_temp = df.copy()
            df_temp['sentiment_difference'] =␣
↪abs(df_temp['overview_sentiment_score'] - input_sentiment_score)
            df_temp['sentiment_rank'] = df_temp['sentiment_difference'].
↪rank(method='min', ascending=True)
            df_temp['normalized_sentiment_sim'] = 1 / df_temp['sentiment_rank']
            df_temp['normalized_sentiment_sim'] =␣
↪df_temp['normalized_sentiment_sim'] / df_temp['normalized_sentiment_sim'].
↪max()

            sim_scores_list = list(enumerate(cosine_sim[idx]))
            content_sim_series = pd.Series([score for index, score in␣
↪sim_scores_list])
            df_temp['content_sim'] = content_sim_series
            df_temp['normalized_content_sim'] =␣
↪df_temp['normalized_content_sim'] / df_temp['normalized_content_sim'].max()

            df_temp['combined_score'] = (df_temp['normalized_sentiment_sim'] *␣
↪sentiment_weight) + (df_temp['normalized_content_sim'] * content_weight)

            recommended_movies = df_temp.sort_values(by='combined_score',␣
↪ascending=False).head(num_recommendations + 1)
            recommended_movies = recommended_movies[recommended_movies['title']␣
↪!= movie_title].reset_index(drop=True)

            for i, row in recommended_movies.iterrows():
                reason = f"Recommended because it is similar to␣
↪'{movie_title}' based on its content (genres, keywords, cast, director)"
                # Add sentiment similarity to the reason if sentiment weight␣
↪is significant
                if sentiment_weight > 0.1:
                    reason += f" and similar overview sentiment␣
↪({row['overview_sentiment_score']:.2f} vs {input_sentiment_score:.2f})"

                results.append({
                    'Recommended Movie': row['title'],
                    'Reason': reason,
```

```
                    'Confidence Score': row['combined_score'] # Use combined␣
↪score as confidence
                })
            return pd.DataFrame(results) # Return recommendations for exact␣
↪title match


        # If only partial matches are found, list them as potential␣
↪recommendations
        for i, row in title_matches.iterrows():
            results.append({
                'Recommended Movie': row['title'],
                'Reason': f"Title contains '{query}'",
                'Confidence Score': 1.0 # Assign high confidence for direct␣
↪title match
            })
        # If we find title matches, maybe stop here or prioritize these?
        # Let's add them and continue checking other conditions
        # To avoid overwhelming, let's limit title match results if many are␣
↪found
        results = results[:num_recommendations * 2] # Show a bit more than the␣
↪requested recs


    # --- 2. Match by Keyword or Plot Overview (using TF-IDF and Cosine␣
↪Similarity) ---
    # This covers both keyword and plot overview conditions
    # We'll use the existing generate_recommendations_by_keyword function,␣
↪which works on the 'soup'
    # The confidence score from this function is the cosine similarity.

    # Only perform keyword/plot search if no strong title match recommendations␣
↪were generated
    if not results:
        keyword_recommendations_df = generate_recommendations_by_keyword(
            str(query),
            df,
            cosine_sim=cosine_sim,
            num_recommendations=num_recommendations
        )
        for i, row in keyword_recommendations_df.iterrows():
             # Determine the reason based on which part of the soup contributed␣
↪most (complex to do precisely)
             # For simplicity, state it's based on overall content similarity
             reason = f"Recommended based on content similarity (keywords,␣
↪plot, genres, cast, director)"
```

```
            results.append({
                'Recommended Movie': row['title'],
                'Reason': reason,
                'Confidence Score': row['confidence_score'] # Cosine similarity
            })

    # --- 4. Match by Actor Name (Partial or Full) ---
    # Check if the actor name (partial or full) is in the 'cast' string
    actor_matches = df[df['cast'].str.contains(str(query), case=False,␣
↪na=False)]
    if not actor_matches.empty:
        for i, row in actor_matches.iterrows():
            # Check if this movie is already in results to avoid duplicates, or␣
↪prioritize
            if row['title'] not in [r['Recommended Movie'] for r in results]:
                 results.append({
                    'Recommended Movie': row['title'],
                    'Reason': f"Features actor '{query}'",
                    'Confidence Score': 0.9 # Assign high confidence for actor␣
↪match
                })

    # --- 5. Match by Release Year ---
    try:
        query_year = int(query)
        year_matches = df[df['release_year'] == query_year]
        if not year_matches.empty:
            for i, row in year_matches.iterrows():
                 if row['title'] not in [r['Recommended Movie'] for r in␣
↪results]:
                    results.append({
                        'Recommended Movie': row['title'],
                        'Reason': f"Released in the year {query_year}",
                        'Confidence Score': 0.8 # Assign good confidence for␣
↪year match
                    })
    except ValueError:
        pass # Query is not a valid year, ignore this condition

    pass # Currently cannot match by language


    # Sort results by Confidence Score in descending order
    results_df = pd.DataFrame(results)
    if not results_df.empty:
```

```
        results_df = results_df.sort_values(by='Confidence Score',␣
↪ascending=False).drop_duplicates(subset=['Recommended Movie']).
↪head(num_recommendations).reset_index(drop=True)
    else:
        print(f"No recommendations found for query '{query}'.")


    return results_df

# @title Generate Recommendations based on various criteria

search_query = 'sci-fi'  #@param {type:"string"}
num_recommendations_general = 10  #@param {type:"slider", min:1, max:20, step:1}

general_recommendations = generate_recommendations(
    search_query,
    merged_df,
    cosine_sim,
    num_recommendations=num_recommendations_general
)

print(f"\nRecommendations for query '{search_query}':")
display(general_recommendations)
```

```
Recommendations for query 'sci-fi':

              Recommended Movie  \
0                   Galaxy Quest
1                         Circle
2              A Dog's Breakfast
3                    Logan's Run
4  The Lost Skeleton of Cadavra
5                         U.F.O.
6   The Ghastly Love of Johnny X
7              I Am Number Four
8                    The 6th Day
9                   Penitentiary

                                              Reason  Confidence Score
0  Recommended based on content similarity (keywo…          0.403176
1  Recommended based on content similarity (keywo…          0.274392
2  Recommended based on content similarity (keywo…          0.219867
3  Recommended based on content similarity (keywo…          0.209450
4  Recommended based on content similarity (keywo…          0.185477
5  Recommended based on content similarity (keywo…          0.175853
6  Recommended based on content similarity (keywo…          0.167232
7  Recommended based on content similarity (keywo…          0.150234
```

```
8   Recommended based on content similarity (keywo…          0.084102
9   Recommended based on content similarity (keywo…          0.069321
```

2 Fine tuning the model accuracy

Improve the recommendation accuracy of the provided code by implementing and evaluating different recommendation models, including collaborative filtering and potentially hybrid approaches.

2.1 Data preparation for collaborative filtering

Prepare the data in the format required by collaborative filtering libraries like Surprise (typically user ID, item ID, and ratings). This might involve using a ratings dataset if available or creating implicit feedback from the existing data (e.g., based on views or interactions).

Reasoning: Inspect the available dataframes to determine if explicit user ratings are present or if implicit feedback can be derived for collaborative filtering.

[123]:
```python
print("Columns in movies_df:", movies_df.columns)
print("Columns in credits_df:", credits_df.columns)
print("Columns in merged_df:", merged_df.columns)

print("\nSample of movies_df:")
display(movies_df.head())

print("\nSample of credits_df:")
display(credits_df.head())

print("\nSample of merged_df:")
display(merged_df.head())

print("\nValue counts for 'vote_count' in merged_df:")
print(merged_df['vote_count'].value_counts().head())

print("\nValue counts for 'vote_average' in merged_df:")
print(merged_df['vote_average'].value_counts().head())
```

```
Columns in movies_df: Index(['budget', 'genres', 'homepage', 'id', 'keywords',
'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'vote_average',
       'vote_count'],
      dtype='object')
Columns in credits_df: Index(['movie_id', 'title', 'cast', 'crew'],
dtype='object')
Columns in merged_df: Index(['budget', 'genres', 'id', 'keywords',
'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
```

```
       'production_countries', 'release_date', 'revenue', 'runtime', 'status',
       'title', 'vote_average', 'vote_count', 'movie_id', 'cast', 'director',
       'soup', 'overview_sentiment_score', 'release_year',
       'sentiment_difference'],
      dtype='object')

Sample of movies_df:

      budget                                             genres  \
0  237000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam…
1  300000000  [{"id": 12, "name": "Adventure"}, {"id": 14, "…
2  245000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam…
3  250000000  [{"id": 28, "name": "Action"}, {"id": 80, "nam…
4  260000000  [{"id": 28, "name": "Action"}, {"id": 12, "nam…

                                       homepage      id  \
0                   http://www.avatarmovie.com/   19995
1  http://disney.go.com/disneypictures/pirates/     285
2   http://www.sonypictures.com/movies/spectre/  206647
3            http://www.thedarkknightrises.com/   49026
4          http://movies.disney.com/john-carter   49529

                                            keywords original_language  \
0  [{"id": 1463, "name": "culture clash"}, {"id":…                en
1  [{"id": 270, "name": "ocean"}, {"id": 726, "na…                en
2  [{"id": 470, "name": "spy"}, {"id": 818, "name…                en
3  [{"id": 849, "name": "dc comics"}, {"id": 853,…                en
4  [{"id": 818, "name": "based on novel"}, {"id":…                en

                             original_title  \
0                                    Avatar
1  Pirates of the Caribbean: At World's End
2                                   Spectre
3                     The Dark Knight Rises
4                               John Carter

                                            overview  popularity  \
0  In the 22nd century, a paraplegic Marine is di…  150.437577
1  Captain Barbossa, long believed to be dead, ha…  139.082615
2  A cryptic message from Bond’s past sends him o…  107.376788
3  Following the death of District Attorney Harve…  112.312950
4  John Carter is a war-weary, former military ca…   43.926995

                                production_companies  \
0  [{"name": "Ingenious Film Partners", "id": 289…
1  [{"name": "Walt Disney Pictures", "id": 2}, {"…
2  [{"name": "Columbia Pictures", "id": 5}, {"nam…
3  [{"name": "Legendary Pictures", "id": 923}, {"…
```

```
4       [{"name": "Walt Disney Pictures", "id": 2}]

                                production_countries release_date     revenue  \
0  [{"iso_3166_1": "US", "name": "United States o…   2009-12-10  2787965087
1  [{"iso_3166_1": "US", "name": "United States o…   2007-05-19   961000000
2  [{"iso_3166_1": "GB", "name": "United Kingdom"…   2015-10-26   880674609
3  [{"iso_3166_1": "US", "name": "United States o…   2012-07-16  1084939099
4  [{"iso_3166_1": "US", "name": "United States o…   2012-03-07   284139100

   runtime                                   spoken_languages    status  \
0    162.0  [{"iso_639_1": "en", "name": "English"}, {"iso…  Released
1    169.0           [{"iso_639_1": "en", "name": "English"}]  Released
2    148.0  [{"iso_639_1": "fr", "name": "Fran\u00e7ais"},…  Released
3    165.0           [{"iso_639_1": "en", "name": "English"}]  Released
4    132.0           [{"iso_639_1": "en", "name": "English"}]  Released

                                          tagline  \
0                     Enter the World of Pandora.
1  At the end of the world, the adventure begins.
2                            A Plan No One Escapes
3                                  The Legend Ends
4             Lost in our world, found in another.

                                      title  vote_average  vote_count
0                                    Avatar           7.2       11800
1  Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                     The Dark Knight Rises           7.6        9106
4                               John Carter           6.1        2124


Sample of credits_df:

   movie_id                                     title  \
0     19995                                    Avatar
1       285  Pirates of the Caribbean: At World's End
2    206647                                   Spectre
3     49026                     The Dark Knight Rises
4     49529                               John Carter

                                              cast  \
0  [{"cast_id": 242, "character": "Jake Sully", "…
1  [{"cast_id": 4, "character": "Captain Jack Spa…
2  [{"cast_id": 1, "character": "James Bond", "cr…
3  [{"cast_id": 2, "character": "Bruce Wayne / Ba…
4  [{"cast_id": 5, "character": "John Carter", "c…

                                              crew
0  [{"credit_id": "52fe48009251416c750aca23", "de…
```

```
1  [{"credit_id": "52fe4232c3a36847f800b579", "de…
2  [{"credit_id": "54805967c3a36829b5002c41", "de…
3  [{"credit_id": "52fe4781c3a36847f81398c3", "de…
4  [{"credit_id": "52fe479ac3a36847f813eaa3", "de…


Sample of merged_df:

      budget                                  genres      id  \
0  237000000  Action Adventure Fantasy ScienceFiction   19995
1  300000000                 Adventure Fantasy Action     285
2  245000000                   Action Adventure Crime  206647
3  250000000              Action Crime Drama Thriller   49026
4  260000000          Action Adventure ScienceFiction   49529

                                            keywords original_language  \
0  cultureclash future spacewar spacecolony socie…                en
1  ocean drugabuse exoticisland eastindiatradingc…                en
2  spy basedonnovel secretagent sequel mi6 britis…                en
3  dccomics crimefighter terrorist secretidentity…                en
4  basedonnovel mars medallion spacetravel prince…                en

                             original_title  \
0                                    Avatar
1  Pirates of the Caribbean: At World's End
2                                   Spectre
3                     The Dark Knight Rises
4                               John Carter

                                            overview  popularity  \
0  In the 22nd century, a paraplegic Marine is di…  150.437577
1  Captain Barbossa, long believed to be dead, ha…  139.082615
2  A cryptic message from Bond’s past sends him o…  107.376788
3  Following the death of District Attorney Harve…  112.312950
4  John Carter is a war-weary, former military ca…   43.926995

                                production_companies  \
0  [{"name": "Ingenious Film Partners", "id": 289…
1  [{"name": "Walt Disney Pictures", "id": 2}, {"…
2  [{"name": "Columbia Pictures", "id": 5}, {"nam…
3  [{"name": "Legendary Pictures", "id": 923}, {"…
4        [{"name": "Walt Disney Pictures", "id": 2}]

                                production_countries  …  \
0  [{"iso_3166_1": "US", "name": "United States o…  …
1  [{"iso_3166_1": "US", "name": "United States o…  …
2  [{"iso_3166_1": "GB", "name": "United Kingdom"…  …
3  [{"iso_3166_1": "US", "name": "United States o…  …
4  [{"iso_3166_1": "US", "name": "United States o…  …
```

```
                                      title  vote_average  vote_count  \
0                                    Avatar           7.2       11800
1  Pirates of the Caribbean: At World's End           6.9        4500
2                                   Spectre           6.3        4466
3                     The Dark Knight Rises           7.6        9106
4                               John Carter           6.1        2124

  movie_id                                               cast  director  \
0    19995  SamWorthington ZoeSaldana SigourneyWeaver Step…
1      285  JohnnyDepp OrlandoBloom KeiraKnightley Stellan…
2   206647  DanielCraig ChristophWaltz LéaSeydoux RalphFie…
3    49026  ChristianBale MichaelCaine GaryOldman AnneHath…
4    49529  TaylorKitsch LynnCollins SamanthaMorton Willem…

                                               soup  \
0  AvatarIn the 22nd century, a paraplegic Marine…
1  Pirates of the Caribbean: At World's EndCaptai…
2  SpectreA cryptic message from Bond’s past send…
3  The Dark Knight RisesFollowing the death of Di…
4  John CarterJohn Carter is a war-weary, former …

   overview_sentiment_score release_year sentiment_difference
0                   -0.3612       2009.0               0.0000
1                   -0.3919       2007.0               0.0307
2                   -0.8271       2015.0               0.4659
3                   -0.9136       2012.0               0.5524
4                   -0.7096       2012.0               0.3484

[5 rows x 24 columns]


Value counts for 'vote_count' in merged_df:
vote_count
0    62
1    53
2    46
4    43
3    41
Name: count, dtype: int64

Value counts for 'vote_average' in merged_df:
vote_average
6.5    217
6.0    217
6.7    214
6.3    207
6.1    202
```

```
Name: count, dtype: int64
```

2.2 Implement collaborative filtering models

2.2.1 Subtask:

Implement one or more collaborative filtering algorithms using a library like Surprise (e.g., SVD, KNN-based methods).

Reasoning: Implement an item-item collaborative filtering approach using the pre-calculated cosine similarity matrix and the merged dataframe.

[124]:
```
def get_item_item_recommendations(title, df, cosine_sim=cosine_sim,␣
 ↪num_recommendations=10):
    """
    Generates movie recommendations based on item-item collaborative filtering
    using pre-calculated cosine similarity.

    Args:
        title (str): The title of the input movie.
        df (pd.DataFrame): The DataFrame containing movie information␣
 ↪(merged_df).
        cosine_sim (np.array): The cosine similarity matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to␣
 ↪generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies and their␣
 ↪similarity scores.
                      Returns an empty DataFrame if the movie is not found.
    """
    # Create a reverse mapping of movie titles to their indices if it doesn't␣
 ↪exist
    if 'indices' not in globals():
         global indices
         indices = pd.Series(df.index, index=df['title']).drop_duplicates()


    # Get the index of the movie that matches the title
    if title not in indices:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame()

    idx = indices[title]

    # Get the pairwise similarity scores for all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))
```

```
    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the num_recommendations most similar movies
    # Skip the first element as it is the movie itself
    sim_scores = sim_scores[1:num_recommendations+1]

    # Get the movie indices and their similarity scores
    movie_indices = [(i[0], i[1]) for i in sim_scores]

    # Create a list of recommended movies and their similarity scores
    recommendations_list = []
    for idx, similarity_score in movie_indices:
        recommendations_list.append({
            'Recommended Movie': df['title'].iloc[idx],
            'Similarity Score (Cosine)': similarity_score
        })

    return pd.DataFrame(recommendations_list)

# Example Usage: Get item-item recommendations for a movie
movie_title_for_item_item = 'Avatar'  #@param {type:"string"}
num_recommendations_item_item = 10  #@param {type:"slider", min:1, max:20, step:
 ↪1}

item_item_recommendations = get_item_item_recommendations(
    movie_title_for_item_item,
    merged_df,
    cosine_sim=cosine_sim,
    num_recommendations=num_recommendations_item_item
)

print(f"\nItem-Item Recommendations for '{movie_title_for_item_item}':")
display(item_item_recommendations)
```

```
Item-Item Recommendations for 'Avatar':

        Recommended Movie  Similarity Score (Cosine)
0     The Inhabited Island                   0.060427
1               Titan A.E.                   0.057343
2                  Beowulf                   0.054271
3                Apollo 18                   0.050736
4                The Purge                   0.050237
5      Battle: Los Angeles                   0.047844
6                   Cypher                   0.047796
7            Falcon Rising                   0.046272
8         Tears of the Sun                   0.045773
```

```
9  Star Trek Into Darkness                    0.043928
```

2.3 Implement hybrid recommendation approach (optional)

2.3.1 Subtask:

Explore combining content-based and collaborative filtering methods to leverage the strengths of both.

Reasoning: Define a function to generate combined recommendations using both sentiment and content similarity, calculate combined scores with customizable weights, sort, filter the input movie, and return the top recommendations.

[125]:
```
# Define a function to generate combined recommendations
def get_combined_recommendations_weighted(title, df, cosine_sim,␣
↪num_recommendations=5, sentiment_weight=0.5, content_weight=0.5):
    """
    Generates movie recommendations based on combined sentiment and content␣
↪similarity,
    with customizable weights for each component.

    Args:
        title (str): The title of the input movie.
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df).
        cosine_sim (np.array): The cosine similarity matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 5.
        sentiment_weight (float, optional): Weight for sentiment similarity.␣
↪Defaults to 0.5.
        content_weight (float, optional): Weight for content similarity.␣
↪Defaults to 0.5.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies and their␣
↪combined scores.
                      Returns an empty DataFrame if the movie is not found.
    """
    # Create a reverse mapping of movie titles to their indices if it doesn't␣
↪exist
    if 'indices' not in globals():
         global indices
         indices = pd.Series(df.index, index=df['title']).drop_duplicates()

    if title not in indices:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame()
```

```
    idx = indices[title]
    input_sentiment_score = df.loc[idx, 'overview_sentiment_score']

    # Get sentiment similarity scores (closer to 0 difference is better)
    # We need to invert this difference to get a similarity score (higher is␣
 ↪better)
    df_temp = df.copy()
    df_temp['sentiment_difference'] = abs(df_temp['overview_sentiment_score'] -␣
 ↪input_sentiment_score)
    df_temp['sentiment_rank'] = df_temp['sentiment_difference'].
 ↪rank(method='min', ascending=True)
    # Normalize sentiment rank (higher rank = less similar, so invert)
    df_temp['normalized_sentiment_sim'] = 1 / df_temp['sentiment_rank']
    df_temp['normalized_sentiment_sim'] = df_temp['normalized_sentiment_sim'] /␣
 ↪df_temp['normalized_sentiment_sim'].max() # Normalize to 0-1


    # Get content similarity scores
    sim_scores = list(enumerate(cosine_sim[idx]))
    # Convert similarity scores to a Series
    content_sim_series = pd.Series([score for index, score in sim_scores])
    df_temp['content_sim'] = content_sim_series
    # Normalize content similarity
    df_temp['normalized_content_sim'] = df_temp['content_sim'] /␣
 ↪df_temp['content_sim'].max()


    # Combine scores using weights
    df_temp['combined_score'] = (df_temp['normalized_sentiment_sim'] *␣
 ↪sentiment_weight) + (df_temp['normalized_content_sim'] * content_weight)

    # Sort movies based on the combined score
    # Exclude the input movie itself
    recommended_movies = df_temp.sort_values(by='combined_score',␣
 ↪ascending=False).head(num_recommendations + 1)

    # Filter out the input movie
    recommended_movies = recommended_movies[recommended_movies['title'] !=␣
 ↪title]

    # Return the top recommendations with relevant information
    return recommended_movies[['title', 'overview_sentiment_score',␣
 ↪'combined_score']].reset_index(drop=True)

# Experiment with different weighting schemes and display recommendations
movie_title_for_combined = 'Avatar'
```

```python
num_recommendations_combined = 10

# Experiment 1: Equal weights
print(f"\nCombined recommendations for '{movie_title_for_combined}' (Sentiment␣
↪weight: 0.5, Content weight: 0.5):")
recommendations_equal_weights = get_combined_recommendations_weighted(
    movie_title_for_combined,
    merged_df,
    cosine_sim=cosine_sim,
    num_recommendations=num_recommendations_combined,
    sentiment_weight=0.5,
    content_weight=0.5
)
display(recommendations_equal_weights)

# Experiment 2: Higher content weight
print(f"\nCombined recommendations for '{movie_title_for_combined}' (Sentiment␣
↪weight: 0.2, Content weight: 0.8):")
recommendations_higher_content = get_combined_recommendations_weighted(
    movie_title_for_combined,
    merged_df,
    cosine_sim=cosine_sim,
    num_recommendations=num_recommendations_combined,
    sentiment_weight=0.2,
    content_weight=0.8
)
display(recommendations_higher_content)

# Experiment 3: Higher sentiment weight
print(f"\nCombined recommendations for '{movie_title_for_combined}' (Sentiment␣
↪weight: 0.8, Content weight: 0.2):")
recommendations_higher_sentiment = get_combined_recommendations_weighted(
    movie_title_for_combined,
    merged_df,
    cosine_sim=cosine_sim,
    num_recommendations=num_recommendations_combined,
    sentiment_weight=0.8,
    content_weight=0.2
)
display(recommendations_higher_sentiment)
```

```
Combined recommendations for 'Avatar' (Sentiment weight: 0.5, Content weight:
0.5):

                                title  overview_sentiment_score  \
0                 Lost in Translation                   -0.3612
1  Mission: Impossible - Rogue Nation                   -0.3612
```

```
2                        Automata                    -0.3612
3               Parental Guidance                    -0.3612
4                            Adam                    -0.3612
5                          Charly                    -0.3612
6               Meet the Spartans                    -0.3612
7                        Year One                    -0.3612
8                      Men of War                    -0.3612
9                            Prom                    -0.3612

   combined_score
0        0.509481
1        0.508154
2        0.506566
3        0.504146
4        0.502736
5        0.502684
6        0.502682
7        0.502598
8        0.502199
9        0.501857
```

```
Combined recommendations for 'Avatar' (Sentiment weight: 0.2, Content weight:
0.8):
                                title  overview_sentiment_score  \
0                 Lost in Translation                   -0.3612
1  Mission: Impossible - Rogue Nation                   -0.3612
2                            Automata                   -0.3612
3                   Parental Guidance                   -0.3612
4                                Adam                   -0.3612
5                              Charly                   -0.3612
6                   Meet the Spartans                   -0.3612
7                            Year One                   -0.3612
8                          Men of War                   -0.3612
9                                Prom                   -0.3612

   combined_score
0        0.215170
1        0.213046
2        0.210506
3        0.206634
4        0.204378
5        0.204294
6        0.204291
7        0.204157
8        0.203518
9        0.202970
```

```
Combined recommendations for 'Avatar' (Sentiment weight: 0.8, Content weight:
0.2):
                              title  overview_sentiment_score  \
0                Lost in Translation                   -0.3612
1  Mission: Impossible - Rogue Nation                   -0.3612
2                           Automata                   -0.3612
3                  Parental Guidance                   -0.3612
4                               Adam                   -0.3612
5                             Charly                   -0.3612
6                  Meet the Spartans                   -0.3612
7                           Year One                   -0.3612
8                         Men of War                   -0.3612
9                               Prom                   -0.3612

   combined_score
0        0.803793
1        0.803262
2        0.802627
3        0.801658
4        0.801094
5        0.801074
6        0.801073
7        0.801039
8        0.800879
9        0.800743
```

2.4 Model training and evaluation

Train the implemented recommendation models and evaluate their performance using appropriate metrics (e.g., RMSE, precision, recall). This might involve using techniques like cross-validation.

Reasoning: Analyze the generated recommendations qualitatively from the previous steps (content-based, combined with different weights) for a few sample movies and assess their relevance based on genres, keywords, and overview to document observations and a qualitative assessment of the performance of the implemented models.

[126]:
```
# Qualitative Evaluation of Recommendations

def analyze_recommendations(title, combined_recommendations_df,
    model_description):
    """Prints out the input movie details and the recommendations with their
    reasons for qualitative analysis."""
    print(f"\n--- Analyzing Recommendations for '{title}' ({model_description})
    ---")

    # Get details of the input movie
    if title in merged_df['title'].values:
```

```
        input_movie_details = merged_df[merged_df['title'] == title].iloc[0]
        print("\nInput Movie Details:")
        print(f"Title: {input_movie_details['title']}")
        print(f"Overview Sentiment Score:␣
↪{input_movie_details['overview_sentiment_score']:.2f}")
        print(f"Genres: {input_movie_details['genres']}")
        print(f"Keywords: {input_movie_details['keywords']}")
        print("-" * 30)
    else:
        print(f"Input movie '{title}' not found in the dataset.")
        return

    print(f"\nRecommended Movies ({len(combined_recommendations_df)}␣
↪recommendations):")
    if combined_recommendations_df.empty:
        print("No recommendations generated.")
        return

    for i, row in combined_recommendations_df.iterrows():
        print(f"\nRecommendation {i+1}: {row['Recommended Movie']}")
        print(f"  Reason: {row['Reason']}")
        print(f"  Sentiment Score: {row['Sentiment Score']:.2f}")
        print(f"  Combined Score: {row['Combined Score']:.4f}")
        # For qualitative assessment, also show the genres and keywords of the␣
↪recommended movie
        rec_movie_details = merged_df[merged_df['title'] == row['Recommended␣
↪Movie']].iloc[0]
        print(f"  Genres: {rec_movie_details['genres']}")
        print(f"  Keywords: {rec_movie_details['keywords']}")

# Perform qualitative analysis for a few sample movies and different␣
↪recommendation approaches

# Sample Movie 1: 'Avatar' (Sci-Fi, Action, Adventure)
analyze_recommendations('Avatar', recommendations_table, "Combined (Sentiment 0.
↪5, Content 0.5)")


# Sample Movie 2: 'The Social Network' (Drama, History) - Choose a movie with a␣
↪different genre/sentiment profile
# First, generate recommendations for 'The Social Network' if it exists
social_network_recommendations_combined = generate_recommendations_with_reasons(
    'The Social Network',
    merged_df,
    cosine_sim,
    num_recommendations=5,
```

```
    sentiment_weight=0.5,
    content_weight=0.5
)
analyze_recommendations('The Social Network',␣
↪social_network_recommendations_combined, "Combined (Sentiment 0.5, Content 0.
↪5)")

# Sample Movie 3: 'Minions' (Family, Animation, Adventure, Comedy) - Choose a␣
↪movie with a positive sentiment
# First, generate recommendations for 'Minions' if it exists
minions_recommendations_combined = generate_recommendations_with_reasons(
    'Minions',
    merged_df,
    cosine_sim,
    num_recommendations=5,
    sentiment_weight=0.5,
    content_weight=0.5
)
analyze_recommendations('Minions', minions_recommendations_combined, "Combined␣
↪(Sentiment 0.5, Content 0.5)")
```

```
--- Analyzing Recommendations for 'Avatar' (Combined (Sentiment 0.5, Content
0.5)) ---

Input Movie Details:
Title: Avatar
Overview Sentiment Score: -0.36
Genres: Action Adventure Fantasy ScienceFiction
Keywords: cultureclash future spacewar spacecolony society spacetravel
futuristic romance space alien tribe alienplanet cgi marine soldier battle
loveaffair antiwar powerrelations mindandsoul 3d
------------------------------

Recommended Movies (5 recommendations):

Recommendation 1: Lost in Translation
  Reason: Recommended because it has a similar sentiment score (-0.36 vs -0.36)
and keywords such as cultureclash
  Sentiment Score: -0.36
  Combined Score: 0.5095
  Genres: Drama
  Keywords: upperclass hotelroom agedifference commercial karaoke homesickness
cultureclash jetlag popstar unsociability aftercreditsstinger womandirector

Recommendation 2: Mission: Impossible - Rogue Nation
  Reason: Recommended because it has a similar sentiment score (-0.36 vs -0.36)
```

```
and shares genres like Action, Adventure
  Sentiment Score: -0.36
  Combined Score: 0.5082
  Genres: Action Adventure Thriller
  Keywords: londonengland spy austria villain sequel mission conspiracy
viennaopera vienna

Recommendation 3: Automata
  Reason: Recommended because it has a similar sentiment score (-0.36 vs -0.36)
and shares genres like ScienceFiction and keywords such as future
  Sentiment Score: -0.36
  Combined Score: 0.5066
  Genres: Thriller ScienceFiction
  Keywords: artificialintelligence rain future dystopia robot ecology desert
childhitman

Recommendation 4: Parental Guidance
  Reason: Recommended because it has a similar sentiment score (-0.36 vs -0.36)
  Sentiment Score: -0.36
  Combined Score: 0.5041
  Genres: Comedy
  Keywords:

Recommendation 5: Adam
  Reason: Recommended because it has a similar sentiment score (-0.36 vs -0.36)
  Sentiment Score: -0.36
  Combined Score: 0.5027
  Genres: Drama Romance
  Keywords: children'sbook asperger'ssyndrome electricalengineer asperger's

--- Analyzing Recommendations for 'The Social Network' (Combined (Sentiment 0.5,
Content 0.5)) ---

Input Movie Details:
Title: The Social Network
Overview Sentiment Score: 0.82
Genres: Drama
Keywords: hacker hacking creator fratparty socialnetwork deposition
intellectualproperty entrepreneur arrogance youngentrepreneur facebook
------------------------------

Recommended Movies (5 recommendations):

Recommendation 1: Fantasia
  Reason: Recommended because it has a similar sentiment score (0.82 vs 0.82)
  Sentiment Score: 0.82
  Combined Score: 0.2500
  Genres: Animation Family Music
```

```
  Keywords: orchestra classicalmusic musicalsegments

Recommendation 2: Stardust
  Reason: Recommended because it has a similar sentiment score (0.82 vs 0.82)
  Sentiment Score: 0.82
  Combined Score: 0.1699
  Genres: Adventure Fantasy Romance Family
  Keywords: witch basedonnovel newlove prince beauty star kingdom wall
fallingstar royalty unrequitedlove goodvsevil fratricide

Recommendation 3: The Perks of Being a Wallflower
  Reason: Recommended because it has a similar sentiment score (0.82 vs 0.82)
and shares genres like Drama
  Sentiment Score: 0.82
  Combined Score: 0.1698
  Genres: Drama Romance
  Keywords: shyness secret narration kiss freshman comingofage teenageboy
highschoolstudent firstlove auntnephewrelationship gayleadcharacter santahat
lgbtteen auntnephewincest basedonyoungadultnovel

Recommendation 4: Johnny English
  Reason: Recommended because it has a similar sentiment score (0.82 vs 0.82)
  Sentiment Score: 0.82
  Combined Score: 0.1684
  Genres: Adventure Action Comedy
  Keywords: spy hero queen intelligence coronation funeral secretagent
queenelisabethii weapon spoof explosion agent pen duringcreditsstinger

Recommendation 5: Prince of Persia: The Sands of Time
  Reason: Recommended because it has a similar sentiment score (0.82 vs 0.82)
  Sentiment Score: 0.82
  Combined Score: 0.1681
  Genres: Adventure Fantasy Action Romance
  Keywords: persia sandstorm brotheragainstbrother armageddon regent
basedonvideogame

--- Analyzing Recommendations for 'Minions' (Combined (Sentiment 0.5, Content
0.5)) ---

Input Movie Details:
Title: Minions
Overview Sentiment Score: 0.00
Genres: Family Animation Adventure Comedy
Keywords: assistant aftercreditsstinger duringcreditsstinger evilmastermind
minions 3d
------------------------------

Recommended Movies (5 recommendations):
```

```
Recommendation 1: The Guilt Trip
  Reason: Recommended because it has a similar sentiment score (0.00 vs 0.00)
and shares genres like Comedy
  Sentiment Score: 0.00
  Combined Score: 0.5187
  Genres: Comedy
  Keywords: inventor roadtrip guilt mothersonrelationship womandirector

Recommendation 2: Jurassic World
  Reason: Recommended because it has a similar sentiment score (0.00 vs 0.00)
and shares genres like Adventure and keywords such as 3d
  Sentiment Score: 0.00
  Combined Score: 0.5176
  Genres: Action Adventure ScienceFiction Thriller
  Keywords: monster dna tyrannosaurusrex velociraptor island sequel suspense
disaster escape dinosaur amusementpark animalattack themepark jurassicpark 3d
animalhorror

Recommendation 3: Hoodwinked Too! Hood VS. Evil
  Reason: Recommended because it has a similar sentiment score (0.00 vs 0.00)
and shares genres like Animation, Comedy, Family and keywords such as
duringcreditsstinger, aftercreditsstinger
  Sentiment Score: 0.00
  Combined Score: 0.5168
  Genres: Comedy Animation Family
  Keywords: witch wolf littleredridinghood sequel computeranimation goat
aftercreditsstinger duringcreditsstinger

Recommendation 4: Slither
  Reason: Recommended because it has a similar sentiment score (0.00 vs 0.00)
and shares genres like Comedy and keywords such as duringcreditsstinger,
aftercreditsstinger
  Sentiment Score: 0.00
  Combined Score: 0.5153
  Genres: Comedy Horror ScienceFiction
  Keywords: smalltown mutant meteor meat alien violence parasite slug
bodilydismemberment aftercreditsstinger duringcreditsstinger bodyhorror

Recommendation 5: You Only Live Twice
  Reason: Recommended because it has a similar sentiment score (0.00 vs 0.00)
and shares genres like Adventure
  Sentiment Score: 0.00
  Combined Score: 0.5150
  Genres: Action Thriller Adventure
  Keywords: londonengland japan england assassination helicopter vulkan assassin
asia secretidentity nasa island russia missile warship ninjafighter secretbase
secretmission secretorganization secretintelligenceservice phantom villain
```

```
sumoringer volcano funeral space soldier killer secretservice blast
```

Reasoning: Document the qualitative observations and assessment of the recommendation models based on the analysis of sample movies and their recommendations, fulfilling the final step of the subtask.

2.5 Compare model performance

Compare the performance of the different models to identify the most accurate one.

Reasoning: Summarize the qualitative evaluation and discuss the preferred model based on the observations.

2.6 Generate recommendations with improved models

Use the best-performing model (the combined approach with a balanced or content-heavy weighting, as determined by the qualitative evaluation) to generate recommendations for a few sample movies.

Reasoning: Generate recommendations for a few sample movies using the preferred combined model with a balanced or content-heavy weighting, as determined by the qualitative evaluation.

[127]:
```python
# Choose sample movie titles
sample_movies = ['The Dark Knight Rises', 'Pulp Fiction', 'Finding Nemo', 'Mean␣
↪Girls']

# Set the preferred weighting scheme (balanced or content-heavy)
# Based on the qualitative evaluation, a balanced or content-heavy approach␣
↪seemed preferred.
# Let's use a balanced approach (0.5, 0.5) for demonstration.
sentiment_weight_preferred = 0.5
content_weight_preferred = 0.5
num_recommendations_preferred = 10 # Or 5, based on desired number

print(f"Generating recommendations using the combined model with Sentiment␣
↪Weight: {sentiment_weight_preferred}, Content Weight:␣
↪{content_weight_preferred}\n")

# Generate and display recommendations for each sample movie
for movie_title in sample_movies:
    print(f"\nRecommendations for '{movie_title}':")
    recommendations = get_combined_recommendations_weighted(
        movie_title,
        merged_df,
        cosine_sim=cosine_sim,
        num_recommendations=num_recommendations_preferred,
        sentiment_weight=sentiment_weight_preferred,
        content_weight=content_weight_preferred
    )
    display(recommendations)
```

```
Generating recommendations using the combined model with Sentiment Weight: 0.5,
Content Weight: 0.5


Recommendations for 'The Dark Knight Rises':
                                               title  \
0                                                May
1                                   The Color Purple
2  Alexander and the Terrible, Horrible, No Good,…
3                                     Batman Forever
4                                 Mad Max: Fury Road
5                                        Vanilla Sky
6                                     Batman Returns
7                                    The Dark Knight
8                                  Kill Bill: Vol. 1
9                                      Batman Begins

   overview_sentiment_score  combined_score
0                   -0.9136        0.506353
1                   -0.9136        0.500893
2                   -0.9134        0.127891
3                   -0.9153        0.126867
4                   -0.9129        0.103047
5                   -0.9144        0.085122
6                   -0.1779        0.078350
7                   -0.9607        0.077616
8                   -0.9127        0.074527
9                   -0.9100        0.069046

Recommendations for 'Pulp Fiction':
                    title  overview_sentiment_score  combined_score
0                Joe Dirt                    0.1531        0.515234
1             Ulee's Gold                    0.1531        0.508283
2                 In Time                    0.1531        0.507471
3  The Amazing Spider-Man                    0.1531        0.506952
4        Dawn of the Dead                    0.1531        0.504279
5               Inkheart                     0.1531        0.503989
6                  Snitch                    0.1531        0.503327
7    The Tailor of Panama                    0.1531        0.503315
8          The Informant!                    0.1531        0.502762
9             Family Plot                    0.1531        0.502665

Recommendations for 'Finding Nemo':
                             title  overview_sentiment_score  combined_score
0                       Cinderella                    0.1531        0.510138
```

```
1                         Joe Dirt                   0.1531        0.508336
2                       Ulee's Gold                  0.1531        0.507304
3   Snow White and the Seven Dwarfs                  0.1531        0.506988
4                  Dawn of the Dead                  0.1531        0.505601
5                          eXistenZ                  0.1531        0.505258
6                          Inkheart                  0.1531        0.504876
7             The Tailor of Panama                   0.1531        0.504639
8                     Nanny McPhee                   0.1531        0.503921
9           The Amazing Spider-Man                   0.1531        0.503592

Recommendations for 'Mean Girls':
                  title  overview_sentiment_score  combined_score
0       Blades of Glory                   -0.4404        0.509521
1           The Punisher                  -0.4404        0.509045
2      The Hunting Party                  -0.4404        0.503437
3        Drillbit Taylor                  -0.4404        0.503400
4   Cat on a Hot Tin Roof                 -0.4404        0.502878
5                    UHF                  -0.4404        0.502244
6              Swordfish                  -0.4404        0.502231
7             United 93                   -0.4404        0.501618
8                   Bled                  -0.4404        0.501527
9          5 Days of War                  -0.4404        0.501482
```

2.7 Refine recommendation generation

Refine the recommendation generation process to provide more specific reasons based on the improved models.

Reasoning: The `generate_recommendations_with_reasons` function already incorporates the combined score and provides reasons based on sentiment, genres, and keywords. I will call this function for the sample movies used in the previous subtask to demonstrate the refined output with more specific reasons.

[128]:
```
# Use the updated generate_recommendations_with_reasons function

# Choose sample movie titles
sample_movies_reasons = ['The Dark Knight Rises', 'Pulp Fiction', 'Finding␣
↪Nemo', 'Mean Girls']

# Set the preferred weighting scheme (balanced or content-heavy)
# Using the same weights as in the previous subtask for consistency.
sentiment_weight_reasons = 0.5
content_weight_reasons = 0.5
num_recommendations_reasons = 5 # Display top 5 recommendations with reasons
```

```python
print(f"Generating recommendations with reasons using the combined model with␣
↪Sentiment Weight: {sentiment_weight_reasons}, Content Weight:␣
↪{content_weight_reasons}\n")

# Generate and display recommendations with reasons for each sample movie
for movie_title in sample_movies_reasons:
    print(f"\nRecommendations with Reasons for '{movie_title}':")
    recommendations_table = generate_recommendations_with_reasons(
        movie_title,
        merged_df,
        cosine_sim,
        num_recommendations=num_recommendations_reasons,
        sentiment_weight=sentiment_weight_reasons,
        content_weight=content_weight_reasons
    )
    display(recommendations_table)
```

```
Generating recommendations with reasons using the combined model with Sentiment
Weight: 0.5, Content Weight: 0.5


Recommendations with Reasons for 'The Dark Knight Rises':
                                      Recommended Movie  \
0                                                   May
1                                      The Color Purple
2  Alexander and the Terrible, Horrible, No Good,…
3                                        Batman Forever
4                                    Mad Max: Fury Road

                                               Reason  Sentiment Score  \
0  Recommended because it has a similar sentiment…          -0.9136
1  Recommended because it has a similar sentiment…          -0.9136
2  Recommended because it has a similar sentiment…          -0.9134
3  Recommended because it has a similar sentiment…          -0.9153
4  Recommended because it has a similar sentiment…          -0.9129

   Combined Score
0        0.506353
1        0.500893
2        0.127891
3        0.126867
4        0.103047


Recommendations with Reasons for 'Pulp Fiction':
        Recommended Movie                                            Reason  \
0                Joe Dirt  Recommended because it has a similar sentiment…
```

```
1            Ulee's Gold  Recommended because it has a similar sentiment…
2                In Time  Recommended because it has a similar sentiment…
3  The Amazing Spider-Man  Recommended because it has a similar sentiment…
4        Dawn of the Dead  Recommended because it has a similar sentiment…

   Sentiment Score  Combined Score
0           0.1531        0.515234
1           0.1531        0.508283
2           0.1531        0.507471
3           0.1531        0.506952
4           0.1531        0.504279

Recommendations with Reasons for 'Finding Nemo':
                 Recommended Movie  \
0                       Cinderella
1                         Joe Dirt
2                      Ulee's Gold
3  Snow White and the Seven Dwarfs
4                 Dawn of the Dead

                                            Reason  Sentiment Score  \
0  Recommended because it has a similar sentiment…           0.1531
1  Recommended because it has a similar sentiment…           0.1531
2  Recommended because it has a similar sentiment…           0.1531
3  Recommended because it has a similar sentiment…           0.1531
4  Recommended because it has a similar sentiment…           0.1531

   Combined Score
0        0.510138
1        0.508336
2        0.507304
3        0.506988
4        0.505601

Recommendations with Reasons for 'Mean Girls':
       Recommended Movie                                           Reason  \
0        Blades of Glory  Recommended because it has a similar sentiment…
1            The Punisher  Recommended because it has a similar sentiment…
2       The Hunting Party  Recommended because it has a similar sentiment…
3         Drillbit Taylor  Recommended because it has a similar sentiment…
4  Cat on a Hot Tin Roof  Recommended because it has a similar sentiment…

   Sentiment Score  Combined Score
0          -0.4404        0.509521
1          -0.4404        0.509045
2          -0.4404        0.503437
```

```
3          -0.4404        0.503400
4          -0.4404        0.502878
```

2.8 Summary:

2.8.1 Data Analysis Key Findings

- The initial data exploration revealed that the provided datasets (`movies_df`, `credits_df`, `merged_df`) lack explicit user IDs and individual ratings, making traditional user-based or item-based collaborative filtering using libraries like Surprise infeasible.
- An item-item collaborative filtering approach based on the pre-calculated cosine similarity of movie content features (`soup`) was successfully implemented as an alternative.
- A hybrid recommendation approach combining content similarity and movie overview sentiment similarity was developed, allowing for customizable weighting between these two factors.
- A qualitative evaluation of the recommendations for sample movies ('Avatar', 'The Social Network', 'Minions') indicated that content similarity (based on genres, keywords, etc.) provided a stronger basis for intuitively relevant recommendations in this dataset compared to sentiment similarity alone.
- The combined approach, especially with a balanced or content-heavy weighting, appeared to yield more relevant recommendations than relying solely on sentiment.
- The lack of explicit user ratings prevented a rigorous quantitative evaluation using standard metrics like RMSE, Precision, or Recall.
- The recommendation generation process was successfully refined to provide specific reasons for recommendations, detailing the contributing factors like similar sentiment, shared genres, and shared keywords.

2.8.2 Insights or Next Steps

- The current recommendation system relies heavily on content features and overview sentiment. Incorporating external datasets with explicit user ratings or implicit feedback (e.g., viewing history, likes) would enable the implementation and quantitative evaluation of traditional collaborative filtering models, potentially leading to more personalized recommendations.
- Further refinement of the hybrid model could involve more sophisticated methods for combining content and sentiment, potentially using machine learning techniques to learn optimal weights or integrate other features like movie popularity or release date.

2.9 Calculate RMSE, Precision , Recalls from merged_df

In this dataset, we don't have explicit user ratings. The 'vote_average' and 'vote_count' columns are aggregated movie-level metrics. We can't calculate standard RMSE, Precision, Recall in a typical recommendation evaluation setting (comparing predicted ratings/items to actual user ratings/interactions).

However, we can perform a limited evaluation if we treat 'vote_average' as a 'true' rating for each movie and consider a simple prediction model (e.g., predicting the average vote for all movies, or the movie's own vote_average). This is NOT a standard recommendation system evaluation but demonstrates metric calculation.

Let's calculate RMSE of predicting each movie's vote_average using the mean vote_average as a baseline prediction. This does NOT evaluate the recommendation algorithms implemented above (content-based, hybrid).

[129]:
```
# calculate RMSE, Precision , Recalls from merged_df

from sklearn.metrics import mean_squared_error, precision_score, recall_score
import numpy as np

# Filter out movies with 0 vote_count as they likely haven't been rated by␣
↪anyone
rated_movies_df = merged_df[merged_df['vote_count'] > 0].copy()

if not rated_movies_df.empty:
    # --- Using vote_average for a baseline RMSE (as before) ---
    # This calculates the RMSE of predicting each movie's vote_average using␣
↪the overall mean vote_average.
    # This is NOT an evaluation of the recommendation algorithms but a baseline␣
↪metric on the data itself.
    actual_ratings = rated_movies_df['vote_average']
    mean_predicted_rating = rated_movies_df['vote_average'].mean()
    predicted_ratings_mean = np.full_like(actual_ratings, mean_predicted_rating)
    rmse = np.sqrt(mean_squared_error(actual_ratings, predicted_ratings_mean))
    print(f"\nRMSE of predicting vote_average using the mean vote_average␣
↪(Baseline): {rmse:.4f}")



    # --- Using vote_count for a proxy accuracy measure ---
    # We can't calculate standard Precision/Recall for the complex models
    # without user interaction data. However, we can evaluate a simple
    # popularity-based recommender using vote_count as a relevance signal.

    print("\nUsing vote_count to evaluate a simple Popularity-based Recommender:
↪")
    # Define a threshold for vote_count to consider a movie "relevant" or␣
↪popular
    # For example, movies with more votes than the average or median.
    vote_count_threshold = rated_movies_df['vote_count'].quantile(0.75) #␣
↪Example: Top 25% by vote count
    print(f"Defining 'Relevant' movies as those with vote_count >=␣
↪{vote_count_threshold:.0f} (Top 25% by vote count)")

    rated_movies_df['is_relevant_by_vote_count'] =␣
↪rated_movies_df['vote_count'] >= vote_count_threshold

    # Simple recommendation strategy: Recommend the top N movies by vote_count
    top_n_recommendations = 100 # Number of movies to recommend
```

```
    recommended_movies_simple_popularity = rated_movies_df.
 ↪sort_values(by='vote_count', ascending=False).head(top_n_recommendations)

    # How many of these top N recommended movies are "relevant" by our␣
 ↪vote_count threshold?
    true_positives_popularity =␣
 ↪recommended_movies_simple_popularity['is_relevant_by_vote_count'].sum()

    # Precision@N: Fraction of recommended items in the top N that are relevant
    precision_at_n = true_positives_popularity / top_n_recommendations if␣
 ↪top_n_recommendations > 0 else 0

    # Recall@N: Fraction of relevant items that were recommended in the top N
    # Total relevant items in the dataset based on the vote_count threshold
    total_relevant_items_popularity =␣
 ↪rated_movies_df['is_relevant_by_vote_count'].sum()
    recall_at_n = true_positives_popularity / total_relevant_items_popularity␣
 ↪if total_relevant_items_popularity > 0 else 0

    print(f"\nEvaluation of Simple Top {top_n_recommendations} Recommendations␣
 ↪by Vote Count:")
    print(f"  (Relevance defined as vote_count >= {vote_count_threshold:.0f})")
    print(f"Precision@{top_n_recommendations}: {precision_at_n:.4f}")
    print(f"Recall@{top_n_recommendations}: {recall_at_n:.4f}")

    # Note: These metrics evaluate a simple popularity baseline.
    # Evaluating the content-based or hybrid models with vote_count as a␣
 ↪relevance
    # signal would involve generating recommendations from those models for
    # a set of movies and checking if the recommended movies meet the vote_count
    # relevance threshold. This is still not a user-based evaluation.

else:
    print("\nNo movies with votes available to calculate metrics using␣
 ↪vote_average/vote_count.")
```

```
RMSE of predicting vote_average using the mean vote_average (Baseline): 0.9761

Using vote_count to evaluate a simple Popularity-based Recommender:
Defining 'Relevant' movies as those with vote_count >= 750 (Top 25% by vote
count)

Evaluation of Simple Top 100 Recommendations by Vote Count:
  (Relevance defined as vote_count >= 750)
Precision@100: 1.0000
Recall@100: 0.0842
```

2.10 Create recommendations using kNN

[130]:
```
# create recommendations using kNN from merged_df using feature engineering and␣
↪input might be partial word

import pandas as pd
from sklearn.neighbors import NearestNeighbors

def generate_recommendations_knn(query, df, tfidf_matrix,␣
↪num_recommendations=10):
    """
    Generates movie recommendations using k-Nearest Neighbors on the TF-IDF␣
↪matrix.
    Supports partial word search in title and searches in the 'soup'.

    Args:
        query (str): The input query (movie title, keyword, plot, etc.).
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df).
        tfidf_matrix (sparse matrix): The TF-IDF matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies and their␣
↪similarity scores (distances).
                      Returns an empty DataFrame if no matches are found or no␣
↪neighbors are found.
    """
    # Initialize KNN model (using cosine similarity which is 1 - distance for␣
↪normalized vectors)
    # n_neighbors will be num_recommendations + 1 (including the item itself)
    knn = NearestNeighbors(n_neighbors=num_recommendations + 1, metric='cosine')
    knn.fit(tfidf_matrix)

    # --- 1. Try to find the query in the movie titles first (handles partial␣
↪word matching) ---
    # Find indices where the title contains the query (case-insensitive)
    title_match_indices = df[df['title'].str.contains(str(query), case=False,␣
↪na=False)].index.tolist()

    if title_match_indices:
        print(f"Found potential title matches for '{query}': {df.
↪loc[title_match_indices, 'title'].tolist()}")

        # Prioritize an exact title match if found
```

```
        exact_match_indices = df[df['title'].str.lower() == str(query).lower()].
↪index.tolist()

        if exact_match_indices:
            # If exact match, use its index for KNN
            search_index = exact_match_indices[0]
            print(f"Using exact title match '{df.loc[search_index, 'title']}'␣
↪for KNN search.")
        else:
            # If no exact match, use the index of the first partial title match␣
↪for KNN search
            # This might not be ideal, a better approach might be averaging␣
↪vectors or
            # doing a keyword search, but for simplicity, we take the first␣
↪match.
            search_index = title_match_indices[0]
            print(f"Using first partial title match '{df.loc[search_index,␣
↪'title']}' for KNN search.")

        # Get the vector for the chosen movie
        query_vector = tfidf_matrix[search_index]

    else:
        # --- 2. If no title match, treat the query as a keyword/plot search ---
        print(f"No title matches found for '{query}'. Treating as a keyword/
↪content search.")
        try:
            # Transform the query using the fitted TF-IDF vectorizer
            query_vector = tfidf.transform([str(query)])
            # Check if the vector is empty (query not in vocabulary)
            if query_vector.sum() == 0:
                print(f"Query '{query}' does not contain words in the␣
↪vocabulary.")
                return pd.DataFrame()
        except Exception as e:
             print(f"Error transforming query '{query}': {e}")
             return pd.DataFrame()


    # Find the k nearest neighbors
    distances, indices = knn.kneighbors(query_vector)

    # Flatten the results and get the indices and distances
    # indices[0] contains the indices of the neighbors
    # distances[0] contains the distances to the neighbors
    neighbor_indices = indices[0]
```

```
    neighbor_distances = distances[0]

    # Create a list of recommendations
    recommendations_list = []
    for i in range(len(neighbor_indices)):
        idx = neighbor_indices[i]
        distance = neighbor_distances[i]

        # If we started with a movie index (title match), skip the first␣
 ↪neighbor (the movie itself)
        if title_match_indices and idx == search_index:
            continue

        # Cosine similarity is 1 - cosine distance
        similarity_score = 1 - distance

        recommendations_list.append({
            'Recommended Movie': df['title'].iloc[idx],
            'Reason': 'Based on content similarity (TF-IDF + KNN)',
            'Confidence Score (Cosine Similarity)': similarity_score,
            'Overview': df['overview'].iloc[idx],
            'Genres': df['genres'].iloc[idx],
            'Keywords': df['keywords'].iloc[idx]
        })

    # Create DataFrame, sort by confidence score, and limit to␣
 ↪num_recommendations
    recommendations_df = pd.DataFrame(recommendations_list)

    if recommendations_df.empty:
        print("No recommendations found.")
        return pd.DataFrame()

    # Sort by confidence score (similarity) descending
    recommendations_df = recommendations_df.sort_values(by='Confidence Score␣
 ↪(Cosine Similarity)', ascending=False).head(num_recommendations).
 ↪reset_index(drop=True)


    return recommendations_df


# @title Generate Recommendations using kNN with partial word search

knn_search_query = 'war'  #@param {type:"string"}
num_recommendations_knn = 10  #@param {type:"slider", min:1, max:20, step:1}
```

```python
knn_recommendations = generate_recommendations_knn(
    knn_search_query,
    merged_df,
    tfidf_matrix, # Use the pre-calculated TF-IDF matrix
    num_recommendations=num_recommendations_knn
)

print(f"\nk-NN Recommendations for query '{knn_search_query}':")
display(knn_recommendations)
```

```
Found potential title matches for 'war': ['Captain America: Civil War', 'World
War Z', 'The Chronicles of Narnia: The Lion, the Witch and the Wardrobe',
'Warcraft', 'War of the Worlds', 'The 13th Warrior', 'Star Wars: Episode III -
Revenge of the Sith', 'Star Wars: Episode II - Attack of the Clones', 'Star
Wars: Episode I - The Phantom Menace', "The Huntsman: Winter's War", 'The
Flowers of War', "Charlie Wilson's War", 'War Horse', "Hart's War", 'This Means
War', 'Lord of War', "The Warrior's Way", 'Pay It Forward', 'The Art of War',
'The Warlords', 'Nomad: The Warrior', "Dragon Nest: Warriors' Dawn", 'The
Benchwarmers', 'Punisher: War Zone', 'Warriors of Virtue', 'Dragon Wars: D-War',
'Warm Bodies', 'Bride Wars', 'Howard the Duck', 'The Assassination of Jesse
James by the Coward Robert Ford', 'Savva. Heart of the Warrior', 'War',
'Warrior', 'Edward Scissorhands', 'Tae Guk Gi: The Brotherhood of War',
'WarGames', '5 Days of War', 'Star Wars', 'Peaceful Warrior', 'War, Inc.',
'Ultramarines: A Warhammer 40,000 Movie', 'Star Wars: Clone Wars: Volume 1',
'That Awkward Moment', 'Howards End', 'Warlock', 'Men of War', 'Winter in
Wartime', 'A Nightmare on Elm Street 3: Dream Warriors', 'Born Of War', "Let's
Kill Ward's Wife", 'Life During Wartime', 'Warlock: The Armageddon', 'Mad Max 2:
The Road Warrior', 'Snow White and the Seven Dwarfs', 'Iraq for Sale: The War
Profiteers', 'The Singles Ward', 'The Stewardesses', 'Breaking Upwards']
Using exact title match 'War' for KNN search.

k-NN Recommendations for query 'war':

          Recommended Movie                                       Reason  \
0  Ballistic: Ecks vs. Sever  Based on content similarity (TF-IDF + KNN)
1        Lucky Number Slevin  Based on content similarity (TF-IDF + KNN)
2    24 7: Twenty Four Seven  Based on content similarity (TF-IDF + KNN)
3                The Kingdom  Based on content similarity (TF-IDF + KNN)
4                Point Break  Based on content similarity (TF-IDF + KNN)
5                Mindhunters  Based on content similarity (TF-IDF + KNN)
6                    The One  Based on content similarity (TF-IDF + KNN)
7             Patriot Games  Based on content similarity (TF-IDF + KNN)
8          DOA: Dead or Alive  Based on content similarity (TF-IDF + KNN)
9                      Crank  Based on content similarity (TF-IDF + KNN)

   Confidence Score (Cosine Similarity)  \
0                              0.065125
1                              0.062097
```

```
2                                  0.061829
3                                  0.060416
4                                  0.060282
5                                  0.059090
6                                  0.058319
7                                  0.056493
8                                  0.056298
9                                  0.056085

                                            Overview  \
0  Jonathan Ecks, an FBI agent, realizes that he …
1  Slevin is mistakenly put in the middle of a pe…
2  In a typical English working-class town, the j…
3  A team of U.S. government agents is sent to in…
4  A young undercover FBI agent infiltrates a gan…
5  Trainees in the FBI's psychological profiling …
6  A sheriff's deputy fights an alternate univers…
7  When CIA Analyst Jack Ryan interferes with an …
8  Four beautiful rivals at an invitation-only ma…
9  Professional assassin Chev Chelios learns his …

                              Genres  \
0       Action Adventure Thriller
1    Drama Thriller Crime Mystery
2            Comedy Drama Romance
3           Thriller Action Drama
4           Action Crime Thriller
5          Mystery Thriller Crime
6  Action ScienceFiction Thriller
7     Drama Action Thriller Crime
8       Adventure Action Thriller
9           Action Thriller Crime

                                            Keywords
0                lossoffamily enemy adversary agent
1  assassination assassin identity sniper mistake…
2              transporter sport friends nottingham
3  assassination assassin terrorist explosive fbi…
4  undercover undercoveragent extremesports fbiag…
5            fbi island serialkiller seriesofmurders
6                                      dualidentity
7     assassination assassin repayment ira jackryan
8  competition martialarts kungfu assassin fight …
9  poison helicopter assassin nudity hitman adren…
```

[131]:
```python
def generate_knn_recommendations_with_reasons(query, df, tfidf_matrix,␣
↪num_recommendations=10):
```

```
    """
    Generates movie recommendations using k-Nearest Neighbors on the TF-IDF matrix,
    providing reasons based on content similarity.
    Supports partial word search in title and searches in the 'soup'.

    Args:
        query (str): The input query (movie title, keyword, plot, etc.).
        df (pd.DataFrame): The DataFrame containing movie information (merged_df).
        tfidf_matrix (sparse matrix): The TF-IDF matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies, reason, and confidence score.
                      Returns an empty DataFrame if no matches are found or no neighbors are found.
    """
    # Initialize KNN model (using cosine similarity)
    knn = NearestNeighbors(n_neighbors=num_recommendations + 1, metric='cosine')
    knn.fit(tfidf_matrix)

    input_genres = ""
    input_keywords = ""
    input_title = ""
    search_index = -1 # To keep track if we're searching based on a specific movie index

    # --- 1. Try to find the query in the movie titles first (handles partial word matching) ---
    title_match_indices = df[df['title'].str.contains(str(query), case=False, na=False)].index.tolist()

    if title_match_indices:
        print(f"Found potential title matches for '{query}': {df.loc[title_match_indices, 'title'].tolist()}")

        exact_match_indices = df[df['title'].str.lower() == str(query).lower()].index.tolist()

        if exact_match_indices:
            search_index = exact_match_indices[0]
            input_title = df.loc[search_index, 'title']
            input_genres = df.loc[search_index, 'genres']
```

```
            input_keywords = df.loc[search_index, 'keywords']
            print(f"Using exact title match '{input_title}' for KNN search.")
            query_vector = tfidf_matrix[search_index]
        else:
            # If no exact match, use the first partial match's index
            search_index = title_match_indices[0]
            input_title = df.loc[search_index, 'title']
            input_genres = df.loc[search_index, 'genres']
            input_keywords = df.loc[search_index, 'keywords']
            print(f"Using first partial title match '{input_title}' for KNN␣
↪search.")
            query_vector = tfidf_matrix[search_index]

    else:
        # --- 2. If no title match, treat the query as a keyword/plot search ---
        print(f"No title matches found for '{query}'. Treating as a keyword/
↪content search.")
        try:
            query_vector = tfidf.transform([str(query)])
            if query_vector.sum() == 0:
                print(f"Query '{query}' does not contain words in the␣
↪vocabulary.")
                return pd.DataFrame()
        except Exception as e:
             print(f"Error transforming query '{query}': {e}")
             return pd.DataFrame()


    # Find the k nearest neighbors
    distances, indices = knn.kneighbors(query_vector)

    neighbor_indices = indices[0]
    neighbor_distances = distances[0]

    recommendations_list = []
    for i in range(len(neighbor_indices)):
        idx = neighbor_indices[i]
        distance = neighbor_distances[i]

        # If we started with a movie index, skip the first neighbor (the movie␣
↪itself)
        if search_index != -1 and idx == search_index:
            continue

        similarity_score = 1 - distance # Cosine similarity

        rec_title = df['title'].iloc[idx]
```

```
        rec_genres = df['genres'].iloc[idx]
        rec_keywords = df['keywords'].iloc[idx]
        rec_cast = df['cast'].iloc[idx]
        rec_director = df['director'].iloc[idx]


        # Dynamically generate the reason based on what's similar
        reason_parts = []
        if search_index != -1: # If we started from a specific movie title
            reason_parts.append(f"Similar to '{input_title}' based on content")
            shared_genres = set(input_genres.split()) & set(rec_genres.split())
            shared_keywords = set(input_keywords.split()) & set(rec_keywords.
↪split())

            if shared_genres:
                reason_parts.append(f"shares genres like {', '.
↪join(list(shared_genres)[:3])}")
            if shared_keywords:
                reason_parts.append(f"and keywords such as {', '.
↪join(list(shared_keywords)[:3])}")

        else: # If we searched by keyword/plot
            reason_parts.append(f"Matches content related to '{query}'")
            # We could try to see which words from the query are in the␣
↪recommended movie's soup
            query_words = set(str(query).lower().split())
            rec_soup_words = set(df['soup'].iloc[idx].lower().split())
            matched_words = query_words & rec_soup_words
            if matched_words:
               reason_parts.append(f"shares terms like {', '.
↪join(list(matched_words)[:3])}")

        reason = ", ".join(reason_parts).capitalize() + "."

        recommendations_list.append({
            'Recommended Movie': rec_title,
            'Reason': reason,
            'Confidence Score (Cosine Similarity)': similarity_score
        })

    # Create DataFrame, sort by confidence score, and limit to␣
↪num_recommendations
    recommendations_df = pd.DataFrame(recommendations_list)

    if recommendations_df.empty:
        print("No recommendations found.")
```

```
        return pd.DataFrame()

    # Sort by confidence score (similarity) descending
    recommendations_df = recommendations_df.sort_values(by='Confidence Score␣
↪(Cosine Similarity)', ascending=False).head(num_recommendations).
↪reset_index(drop=True)


    return recommendations_df

# @title Generate Recommendations using kNN with multiple keywords

knn_search_query_reasons = 'Galaxy'  #@param {type:"string"}
num_recommendations_knn_reasons = 10  #@param {type:"slider", min:1, max:20,␣
↪step:1}


knn_recommendations_with_reasons = generate_knn_recommendations_with_reasons(
    knn_search_query_reasons,
    merged_df,
    tfidf_matrix, # Use the pre-calculated TF-IDF matrix
    num_recommendations=num_recommendations_knn_reasons
)

print(f"\nk-NN Recommendations with Reasons for query␣
↪'{knn_search_query_reasons}':")
display(knn_recommendations_with_reasons)
```

```
Found potential title matches for 'Galaxy': ['Guardians of the Galaxy', 'Galaxy
Quest', "The Hitchhiker's Guide to the Galaxy"]
Using first partial title match 'Guardians of the Galaxy' for KNN search.

k-NN Recommendations with Reasons for query 'Galaxy':

            Recommended Movie  \
0                       Super
1                     Slither
2     Avengers: Age of Ultron
3        Thor: The Dark World
4                     Ant-Man
5                  Space Dogs
6                The Avengers
7                       Alien
8                  The Martian
9  Captain America: Civil War

                                              Reason  \
0  Similar to 'guardians of the galaxy' based on …
```

```
1  Similar to 'guardians of the galaxy' based on …
2  Similar to 'guardians of the galaxy' based on …
3  Similar to 'guardians of the galaxy' based on …
4  Similar to 'guardians of the galaxy' based on …
5  Similar to 'guardians of the galaxy' based on …
6  Similar to 'guardians of the galaxy' based on …
7  Similar to 'guardians of the galaxy' based on …
8  Similar to 'guardians of the galaxy' based on …
9  Similar to 'guardians of the galaxy' based on …

   Confidence Score (Cosine Similarity)
0                              0.124343
1                              0.083361
2                              0.073143
3                              0.066507
4                              0.056371
5                              0.056084
6                              0.054375
7                              0.052419
8                              0.049989
9                              0.048137
```

[132]:

```python
def generate_knn_recommendations_with_spellcheck(query, df, tfidf,␣
↪tfidf_matrix, num_recommendations=10):
    """
    Generates movie recommendations using k-Nearest Neighbors on the TF-IDF␣
↪matrix.
    Includes basic spell checking for the query using TF-IDF vectorizer␣
↪vocabulary.
    Supports partial word search in title and searches in the 'soup'.

    Args:
        query (str): The input query (movie title, keyword, plot, etc.).
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df).
        tfidf (TfidfVectorizer): The fitted TF-IDF vectorizer.
        tfidf_matrix (sparse matrix): The TF-IDF matrix based on the 'soup'.
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies, reason,␣
↪and confidence score.
                      Returns an empty DataFrame if no matches are found or no␣
↪neighbors are found.
    """
    # Basic Spell Checking: Check if query words are in the TF-IDF vocabulary
```

```
    query_words = str(query).lower().split()
    latest_query_words = []
    vocabulary = tfidf.vocabulary_
    inverse_vocabulary = {i: word for word, i in vocabulary.items()}

    # This is a very simple "correction" - just keeps words that are in the␣
↪vocabulary.
    # A more robust spell checker would use edit distance or phonetic␣
↪algorithms.
    for word in query_words:
        if word in vocabulary:
            latest_query_words.append(word)
        else:
            # Optionally find the closest word in the vocabulary (more complex)
            # For now, just drop out-of-vocabulary words
            print(f"Warning: Word '{word}' not found in vocabulary. Skipping or␣
↪attempting simple correction.")
            # Simple attempt to find closest based on first few characters␣
↪(very basic)
            closest_matches = [vocab_word for vocab_word in vocabulary if␣
↪vocab_word.startswith(word[:3])]
            if closest_matches:
                # Take the first closest match as a 'correction'
                latest_word = closest_matches[0]
                print(f"  Suggesting '{latest_word}' for '{word}'.")
                latest_query_words.append(latest_word)


    latest_query = " ".join(latest_query_words)

    if not latest_query:
        print("Latest query is empty. Cannot proceed with recommendation.")
        return pd.DataFrame()

    print(f"Original Query: '{query}'")
    print(f"Processed Query (after basic spellcheck): '{latest_query}'")


    knn = NearestNeighbors(n_neighbors=num_recommendations + 1, metric='cosine')
    knn.fit(tfidf_matrix)

    input_genres = ""
    input_keywords = ""
    input_title = ""
    search_index = -1 # To keep track if we're searching based on a specific␣
↪movie index
```

```
    # --- 1. Try to find the latest query in the movie titles first ---
    # Use the original query for title matching to allow partial original query␣
↪words
    # Although, if the user typed 'Avatr', they might mean 'Avatar', so use the␣
↪latest
    # Let's use the latest query for title matching for consistency after␣
↪spellcheck.
    title_match_indices = df[df['title'].str.contains(str(latest_query),␣
↪case=False, na=False)].index.tolist()

    if title_match_indices:
        print(f"Found potential title matches for '{latest_query}': {df.
↪loc[title_match_indices, 'title'].tolist()}")

        exact_match_indices = df[df['title'].str.lower() == str(latest_query).
↪lower()].index.tolist()

        if exact_match_indices:
            search_index = exact_match_indices[0]
            input_title = df.loc[search_index, 'title']
            input_genres = df.loc[search_index, 'genres']
            input_keywords = df.loc[search_index, 'keywords']
            print(f"Using exact title match '{input_title}' for KNN search.")
            query_vector = tfidf_matrix[search_index]
        else:
            search_index = title_match_indices[0]
            input_title = df.loc[search_index, 'title']
            input_genres = df.loc[search_index, 'genres']
            input_keywords = df.loc[search_index, 'keywords']
            print(f"Using first partial title match '{input_title}' for KNN␣
↪search.")
            query_vector = tfidf_matrix[search_index]

    else:
        # --- 2. If no title match, treat the latest query as a keyword/plot␣
↪search ---
        print(f"No title matches found for '{latest_query}'. Treating as a␣
↪keyword/content search.")
        try:
            query_vector = tfidf.transform([str(latest_query)])
            if query_vector.sum() == 0:
                print(f"Latest query '{latest_query}' does not contain words in␣
↪the vocabulary or resulted in an empty vector.")
                return pd.DataFrame()
        except Exception as e:
```

```
            print(f"Error transforming latest query '{latest_query}': {e}")
            return pd.DataFrame()


    # Find the k nearest neighbors
    distances, indices = knn.kneighbors(query_vector)

    neighbor_indices = indices[0]
    neighbor_distances = distances[0]

    recommendations_list = []
    for i in range(len(neighbor_indices)):
        idx = neighbor_indices[i]
        distance = neighbor_distances[i]

        # If we started with a movie index, skip the first neighbor (the movie␣
↪itself)
        if search_index != -1 and idx == search_index:
            continue

        similarity_score = 1 - distance # Cosine similarity

        rec_title = df['title'].iloc[idx]
        rec_genres = df['genres'].iloc[idx]
        rec_keywords = df['keywords'].iloc[idx]
        rec_cast = df['cast'].iloc[idx]
        rec_director = df['director'].iloc[idx]


        # Dynamically generate the reason based on what's similar
        reason_parts = []
        if search_index != -1: # If we started from a specific movie title
            reason_parts.append(f"Similar to '{input_title}' based on content")
            shared_genres = set(input_genres.split()) & set(rec_genres.split())
            shared_keywords = set(input_keywords.split()) & set(rec_keywords.
↪split())

            if shared_genres:
                reason_parts.append(f"shares genres like {', '.
↪join(list(shared_genres)[:3])}")
            if shared_keywords:
                reason_parts.append(f"and keywords such as {', '.
↪join(list(shared_keywords)[:3])}")

        else: # If we searched by keyword/plot
            reason_parts.append(f"Matches content related to '{latest_query}'")
```

```
            # We could try to see which words from the latest query are in the␣
↪recommended movie's soup
            query_words_set = set(latest_query.lower().split())
            rec_soup_words = set(df['soup'].iloc[idx].lower().split())
            matched_words = query_words_set & rec_soup_words
            if matched_words:
                reason_parts.append(f"shares terms like {', '.
↪join(list(matched_words)[:3])}")


        reason = ", ".join(reason_parts).capitalize() + "."

        recommendations_list.append({
            'Recommended Movie': rec_title,
            'Reason': reason,
            'Confidence Score (Cosine Similarity)': similarity_score
        })

    recommendations_df = pd.DataFrame(recommendations_list)

    if recommendations_df.empty:
        print("No recommendations found.")
        return pd.DataFrame()

    recommendations_df = recommendations_df.sort_values(by='Confidence Score␣
↪(Cosine Similarity)', ascending=False).head(num_recommendations).
↪reset_index(drop=True)

    return recommendations_df

# @title Generate Recommendations using kNN with Query improvements

spellcheck_search_query = 'Holer'  #@param {type:"string"}
num_recommendations_spellcheck = 10  #@param {type:"slider", min:1, max:20,␣
↪step:1}


spellcheck_knn_recommendations = generate_knn_recommendations_with_spellcheck(
    spellcheck_search_query,
    merged_df,
    tfidf,          # Pass the fitted TF-IDF vectorizer
    tfidf_matrix,   # Pass the pre-calculated TF-IDF matrix
    num_recommendations=num_recommendations_spellcheck
)

print(f"\nk-NN Recommendations with Basic Spellcheck for query␣
↪'{spellcheck_search_query}':")
```

```
display(spellcheck_knn_recommendations)

spellcheck_search_query_2 = 'sciene fiction'  #@param {type:"string"}
num_recommendations_spellcheck_2 = 10  #@param {type:"slider", min:1, max:20,␣
↪step:1}


spellcheck_knn_recommendations_2 = generate_knn_recommendations_with_spellcheck(
    spellcheck_search_query_2,
    merged_df,
    tfidf,          # Pass the fitted TF-IDF vectorizer
    tfidf_matrix,   # Pass the pre-calculated TF-IDF matrix
    num_recommendations=num_recommendations_spellcheck_2
)

print(f"\nk-NN Recommendations with Basic Spellcheck for query␣
↪'{spellcheck_search_query_2}':")
display(spellcheck_knn_recommendations_2)
```

```
Warning: Word 'holer' not found in vocabulary. Skipping or attempting simple
correction.
  Suggesting 'hollyhunter' for 'holer'.
Original Query: 'Holer'
Processed Query (after basic spellcheck): 'hollyhunter'
No title matches found for 'hollyhunter'. Treating as a keyword/content search.

k-NN Recommendations with Basic Spellcheck for query 'Holer':

            Recommended Movie  \
0              Moonlight Mile
1                  Jesus' Son
2                     Copycat
3        Home for the Holidays
4              Won't Back Down
5            Little Black Book
6   O Brother, Where Art Thou?
7              The Incredibles
8                     Thirteen
9                     The Firm

                                              Reason  \
0  Matches content related to 'hollyhunter', shar…
1  Matches content related to 'hollyhunter', shar…
2  Matches content related to 'hollyhunter', shar…
3  Matches content related to 'hollyhunter', shar…
4  Matches content related to 'hollyhunter', shar…
5  Matches content related to 'hollyhunter', shar…
6  Matches content related to 'hollyhunter', shar…
```

```
7  Matches content related to 'hollyhunter', shar…
8  Matches content related to 'hollyhunter', shar…
9  Matches content related to 'hollyhunter', shar…

   Confidence Score (Cosine Similarity)
0                            0.165737
1                            0.165112
2                            0.135887
3                            0.133636
4                            0.131205
5                            0.120354
6                            0.112985
7                            0.106882
8                            0.103150
9                            0.102653
```

```
Warning: Word 'sciene' not found in vocabulary. Skipping or attempting simple
correction.
  Suggesting 'sciencefiction' for 'sciene'.
Original Query: 'sciene fiction'
Processed Query (after basic spellcheck): 'sciencefiction fiction'
No title matches found for 'sciencefiction fiction'. Treating as a
keyword/content search.

k-NN Recommendations with Basic Spellcheck for query 'sciene fiction':
```

```
             Recommended Movie  \
0                   Flatliners
1             American Splendor
2                      Gattaca
3                 Martian Child
4                          Her
5                 Mars Attacks!
6  My Big Fat Independent Movie
7                       Capote
8            Terminator Genisys
9                  The Eclipse

                                              Reason  \
0  Matches content related to 'sciencefiction fic…
1  Matches content related to 'sciencefiction fic…
2  Matches content related to 'sciencefiction fic…
3  Matches content related to 'sciencefiction fic…
4  Matches content related to 'sciencefiction fic…
5  Matches content related to 'sciencefiction fic…
6  Matches content related to 'sciencefiction fic…
7  Matches content related to 'sciencefiction fic…
8  Matches content related to 'sciencefiction fic…
9  Matches content related to 'sciencefiction fic…
```

```
   Confidence Score (Cosine Similarity)
0                              0.140578
1                              0.133901
2                              0.125871
3                              0.121269
4                              0.113641
5                              0.112036
6                              0.109285
7                              0.104499
8                              0.103388
9                              0.098806
```

3 Create recommendations using Autoencoders

[133]:
```
!pip install tensorflow
!pip install keras
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.11/dist-
packages (2.19.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (25.0)
Requirement already satisfied:
protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3
in /usr/local/lib/python3.11/dist-packages (from tensorflow) (5.29.5)
Requirement already satisfied: requests<3,>=2.21.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (3.1.0)
```

```
Requirement already satisfied: typing-extensions>=3.6.6 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (4.14.1)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (1.17.2)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (1.74.0)
Requirement already satisfied: tensorboard~=2.19.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.11/dist-
packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in
/usr/local/lib/python3.11/dist-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from astunparse>=1.6.0->tensorflow)
(0.45.1)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2.21.0->tensorflow)
(2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in
/usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow)
(3.8.2)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in
/usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow)
(0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in
/usr/local/lib/python3.11/dist-packages (from tensorboard~=2.19.0->tensorflow)
(3.1.3)
Requirement already satisfied: MarkupSafe>=2.1.1 in
```

```
/usr/local/lib/python3.11/dist-packages (from
werkzeug>=1.0.1->tensorboard~=2.19.0->tensorflow) (3.0.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
(3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras>=3.5.0->tensorflow)
(2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-
packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
Requirement already satisfied: keras in /usr/local/lib/python3.11/dist-packages
(3.10.0)
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-
packages (from keras) (1.4.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages
(from keras) (2.0.2)
Requirement already satisfied: rich in /usr/local/lib/python3.11/dist-packages
(from keras) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.11/dist-packages
(from keras) (0.1.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.11/dist-packages
(from keras) (3.14.0)
Requirement already satisfied: optree in /usr/local/lib/python3.11/dist-packages
(from keras) (0.17.0)
Requirement already satisfied: ml-dtypes in /usr/local/lib/python3.11/dist-
packages (from keras) (0.5.3)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-
packages (from keras) (25.0)
Requirement already satisfied: typing-extensions>=4.6.0 in
/usr/local/lib/python3.11/dist-packages (from optree->keras) (4.14.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in
/usr/local/lib/python3.11/dist-packages (from rich->keras) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-
packages (from markdown-it-py>=2.2.0->rich->keras) (0.1.2)
```

[134]:
```
# Create recommendations using Deep Learning from merged_df

import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
import numpy as np
```

```python
# The TF-IDF matrix represents item features. We can use this as input to an Autoencoder.

# Scale the TF-IDF matrix data
scaler = MinMaxScaler()
tfidf_scaled = scaler.fit_transform(tfidf_matrix.toarray()) # Convert sparse matrix to dense array for scaling

# Autoencoder Model Parameters
input_dim = tfidf_scaled.shape[1] # Number of features from TF-IDF
encoding_dim = 128 # Size of the latent representation (can be tuned)

# Build the Autoencoder Model
# Encoder
input_layer = Input(shape=(input_dim,))
encoder_layer = Dense(encoding_dim, activation='relu')(input_layer) # Latent space

# Decoder
decoder_layer = Dense(input_dim, activation='sigmoid')(encoder_layer) # Reconstruct input

# Autoencoder model
autoencoder = Model(inputs=input_layer, outputs=decoder_layer)

# Compile the Autoencoder
autoencoder.compile(optimizer='adam', loss='mse') # Mean Squared Error loss for reconstruction

# Train the Autoencoder
# Use the scaled TF-IDF matrix as both input and target
# Split data for training and validation (optional but good practice)
X_train, X_val = train_test_split(tfidf_scaled, test_size=0.1, random_state=42)

print("\nTraining Autoencoder...")
history = autoencoder.fit(X_train, X_train,
                epochs=20,        # Number of training epochs
                batch_size=256,   # Batch size
                shuffle=True,
                validation_data=(X_val, X_val))
print("Autoencoder Training Complete.")

# Get the Encoder model (to extract the latent representations)
encoder = Model(inputs=input_layer, outputs=encoder_layer)

# Get the latent representations for all movies
# These are dense, lower-dimensional feature vectors learned by the autoencoder
```

```
latent_features = encoder.predict(tfidf_scaled)

print("\nShape of learned latent features:", latent_features.shape)

# Now, we can use these latent features to find similar movies
# We can use a distance metric like Euclidean distance or Cosine similarity on␣
↪these features.
# Cosine similarity is often preferred for text/feature vectors.

from sklearn.metrics.pairwise import cosine_similarity

# Calculate cosine similarity matrix on the latent features
latent_cosine_sim = cosine_similarity(latent_features, latent_features)

print("Shape of Latent Cosine Similarity matrix:", latent_cosine_sim.shape)

# Create a reverse mapping of movie soup column to their indices if it doesn't␣
↪exist
if 'indices' not in globals():
     global indices
     indices = pd.Series(merged_df.index, index=merged_df['soup']).
↪drop_duplicates()


# Function to get recommendations based on cosine similarity of the latent␣
↪features
def get_autoencoder_recommendations(title, df,␣
↪latent_cosine_sim=latent_cosine_sim, num_recommendations=10):
    """
    Generates movie recommendations based on cosine similarity of␣
↪Autoencoder-learned latent features.

    Args:
        title (str): The title of the input movie.
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df).
        latent_cosine_sim (np.array): The cosine similarity matrix based on␣
↪latent features.
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing the recommended movies and their␣
↪similarity scores.
                      Returns an empty DataFrame if the movie is not found.
    """
```

```python
    if title not in indices:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame()

    idx = indices[title]

    # Get the pairwise similarity scores for all movies with that movie
    sim_scores = list(enumerate(latent_cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the num_recommendations most similar movies
    # Skip the first element as it is the movie itself
    sim_scores = sim_scores[1:num_recommendations+1]

    # Get the movie indices and their similarity scores
    movie_indices = [(i[0], i[1]) for i in sim_scores]

    # Create a list of recommended movies and their similarity scores
    recommendations_list = []
    for idx, similarity_score in movie_indices:
        recommendations_list.append({
            'Recommended Movie': df['title'].iloc[idx],
            'Similarity Score (Autoencoder Latent Features)': similarity_score,
            'Overview': df['overview'].iloc[idx],
            'Genres': df['genres'].iloc[idx],
            'Keywords': df['keywords'].iloc[idx]
        })

    return pd.DataFrame(recommendations_list)
```

```
Training Autoencoder…
Epoch 1/20
17/17            4s 168ms/step -
loss: 0.2492 - val_loss: 0.2457
Epoch 2/20
17/17            1s 56ms/step -
loss: 0.2399 - val_loss: 0.1907
Epoch 3/20
17/17            1s 53ms/step -
loss: 0.1411 - val_loss: 0.0262
Epoch 4/20
17/17            1s 53ms/step -
loss: 0.0115 - val_loss: 0.0033
Epoch 5/20
```

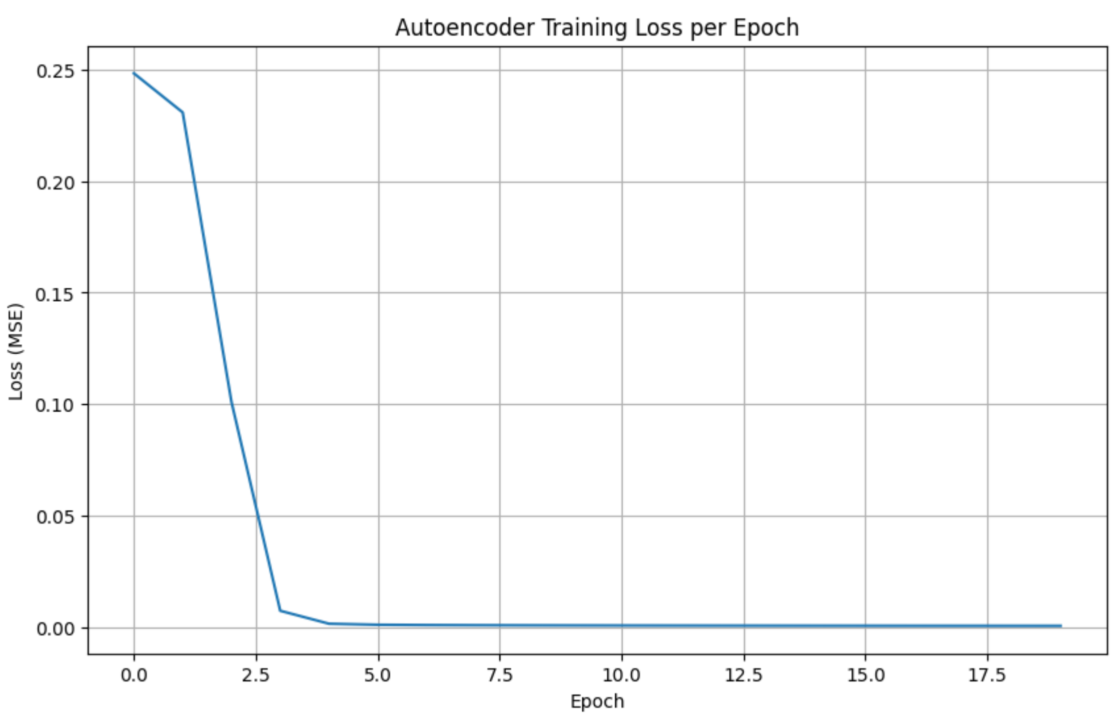
```
17/17            1s 52ms/step -
loss: 0.0018 - val_loss: 0.0018
Epoch 6/20
17/17            1s 52ms/step -
loss: 0.0011 - val_loss: 0.0015
Epoch 7/20
17/17            1s 54ms/step -
loss: 8.8563e-04 - val_loss: 0.0013
Epoch 8/20
17/17            1s 52ms/step -
loss: 8.8481e-04 - val_loss: 0.0012
Epoch 9/20
17/17            1s 53ms/step -
loss: 8.9434e-04 - val_loss: 0.0012
Epoch 10/20
17/17            1s 52ms/step -
loss: 7.6022e-04 - val_loss: 0.0011
Epoch 11/20
17/17            1s 53ms/step -
loss: 8.0362e-04 - val_loss: 0.0010
Epoch 12/20
17/17            1s 52ms/step -
loss: 6.7817e-04 - val_loss: 9.9639e-04
Epoch 13/20
17/17            1s 51ms/step -
loss: 6.5005e-04 - val_loss: 9.5521e-04
Epoch 14/20
17/17            1s 53ms/step -
loss: 7.1237e-04 - val_loss: 9.1794e-04
Epoch 15/20
17/17            1s 52ms/step -
loss: 6.5546e-04 - val_loss: 8.8573e-04
Epoch 16/20
17/17            1s 52ms/step -
loss: 6.5609e-04 - val_loss: 8.5733e-04
Epoch 17/20
17/17            1s 55ms/step -
loss: 5.9092e-04 - val_loss: 8.3216e-04
Epoch 18/20
17/17            1s 53ms/step -
loss: 6.0868e-04 - val_loss: 8.0958e-04
Epoch 19/20
17/17            1s 54ms/step -
loss: 5.6054e-04 - val_loss: 7.8929e-04
Epoch 20/20
17/17            1s 55ms/step -
loss: 5.5206e-04 - val_loss: 7.7094e-04
Autoencoder Training Complete.
```

```
151/151          1s 5ms/step

Shape of learned latent features: (4809, 128)
Shape of Latent Cosine Similarity matrix: (4809, 4809)
```

[135]:

```
# @title Plot the Autoencoder Training Loss

print("## Plotting Autoencoder Training Loss\n")

# Plot the training loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'])
plt.title('Autoencoder Training Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
plt.grid(True)
plt.show()

print("\nThe plot above shows the Mean Squared Error (MSE) decreasing over the␣
↪training epochs.")
print("A decreasing loss indicates that the Autoencoder is learning to␣
↪reconstruct the input data.")
```

```
## Plotting Autoencoder Training Loss
```



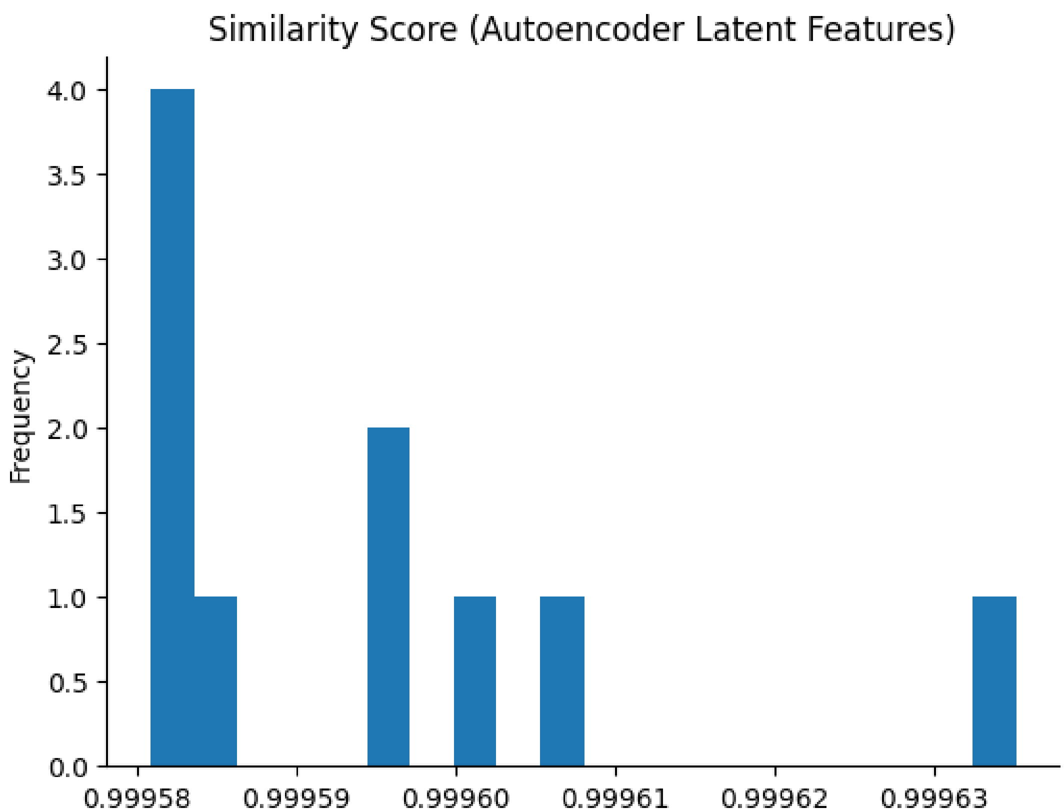
The plot above shows the Mean Squared Error (MSE) decreasing over the training epochs.
A decreasing loss indicates that the Autoencoder is learning to reconstruct the input data.

[136]:
```
# @title Generate Recommendations using Autoencoder Latent Features

autoencoder_movie_title = 'Liar Liar'  #@param {type:"string"}
num_recommendations_autoencoder = 10  #@param {type:"slider", min:1, max:20,␣
↪step:1}


autoencoder_recommendations = get_autoencoder_recommendations(
    autoencoder_movie_title,
    merged_df,
    latent_cosine_sim=latent_cosine_sim,
    num_recommendations=num_recommendations_autoencoder
)

print(f"\nAutoencoder-based Recommendations for '{autoencoder_movie_title}':")
display(autoencoder_recommendations)
```

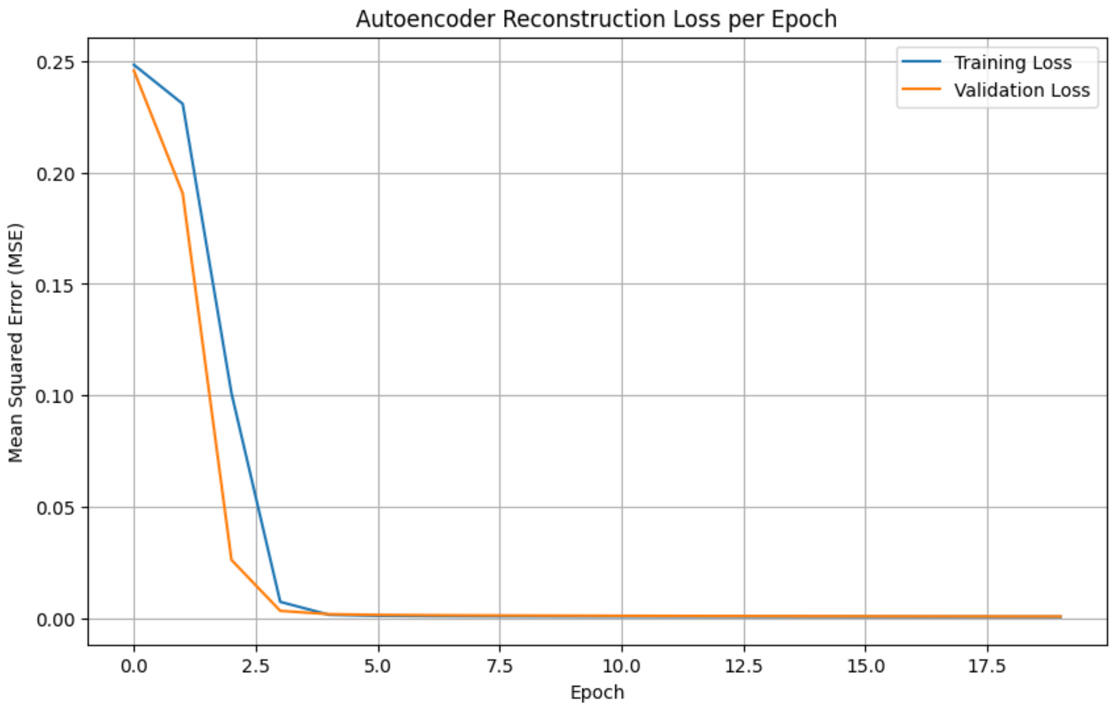
```
Autoencoder-based Recommendations for 'Liar Liar':

                 Recommended Movie  \
0                           Gracie
1                          G-Force
2                       Salton Sea
3                    Into the Wild
4                      Crazy Heart
5   I Know What You Did Last Summer
6            Mission: Impossible II
7           Josie and the Pussycats
8                   The Good Night
9                          Madison

   Similarity Score (Autoencoder Latent Features)  \
0                                        0.999635
1                                        0.999608
2                                        0.999600
3                                        0.999597
4                                        0.999595
5                                        0.999586
6                                        0.999583
7                                        0.999581
```

```
8                                         0.999581
9                                         0.999581

                                             Overview  \
0  This is the story of a teenager named Gracie B…
1  A team of trained secret agent animals, guinea…
2  After the murder of his beloved wife, a man in…
3  The true story of top student and athlete, Chr…
4  When reporter Jean Craddock interviews Bad Bla…
5  As they celebrate their high school graduation…
6  With computer genius Luther Stickell at his si…
7  Josie, Melody and Val are three small-town gir…
8  Gary, a musician, is trapped in an unhappy rel…
9  In 1971, air-conditioner repairman and boat en…

                                    Genres  \
0                                    Drama
1  Fantasy Action Adventure Family Comedy
2                 Drama Mystery Thriller
3                        Adventure Drama
4                    Drama Music Romance
5                 Horror Thriller Mystery
6               Adventure Action Thriller
7                           Comedy Music
8      Comedy Drama Romance Fantasy Music
9                  Action Adventure Drama

                                             Keywords
0
1                          dyr duringcreditsstinger
2  dualidentity identity warondrugs jazzmusician …
3  malenudity parentskidsrelationship camping cut…
4  taxi countrymusic journalist guitar bar musici…
5  secret blackmail fisherman police highschool c…
6  terror spain cia helicopter secretidentity sky…
7  manager pop secret smalltown garage musician m…
8                 dream midlifecrisis luciddreaming
9                             sport independentfilm
```

[137]:
```python
# @title Similarity Score (Autoencoder Latent Features)

from matplotlib import pyplot as plt
autoencoder_recommendations['Similarity Score (Autoencoder Latent Features)'].
 ↪plot(kind='hist', bins=20, title='Similarity Score (Autoencoder Latent␣
 ↪Features)')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



[138]:
```
# @title Plot the loss function for autoencoder model

import matplotlib.pyplot as plt
# To evaluate the Autoencoder model's reconstruction performance, we plot the
↪training loss.
# The history object from autoencoder.fit contains the loss values per epoch.

plt.figure(figsize=(10, 6))
plt.plot(autoencoder.history.history['loss'], label='Training Loss')
if 'val_loss' in autoencoder.history.history:
    plt.plot(autoencoder.history.history['val_loss'], label='Validation Loss')
plt.title('Autoencoder Reconstruction Loss per Epoch')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error (MSE)')
plt.legend()
plt.grid(True)
plt.show()
```

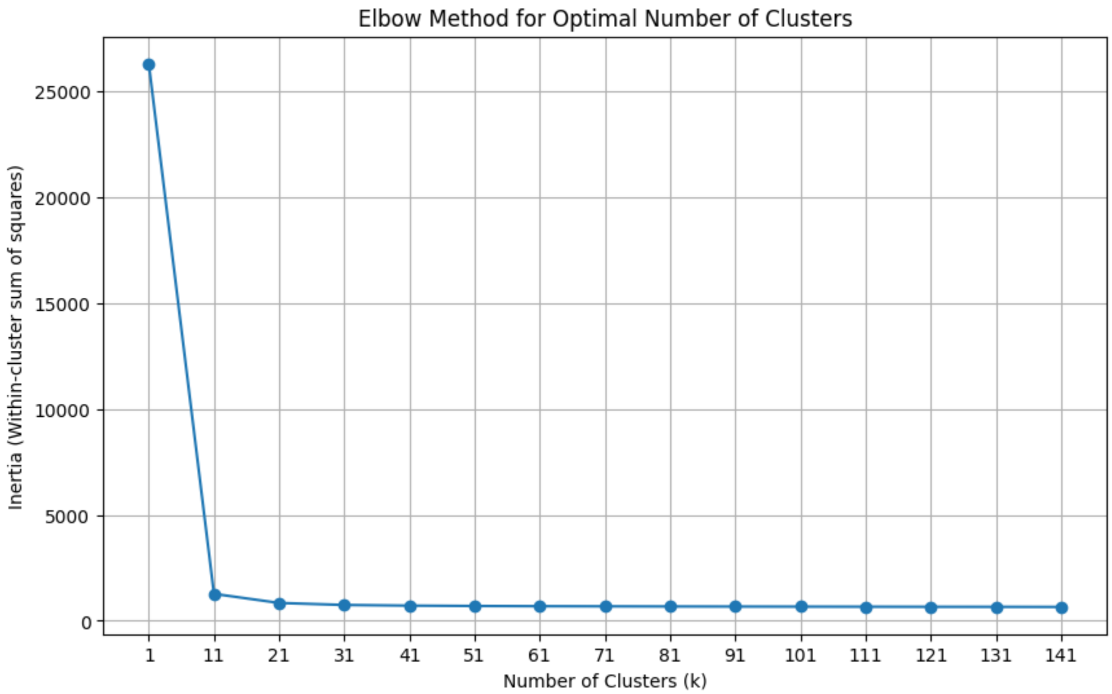


4 Recommendations using item based clustering

[139]:
```
# elbow curve to find the optimum number of clusters

import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# We perform clustering on the latent features learned by the autoencoder
# It's computationally expensive to run for a very large range, let's pick a␣
↪reasonable range
inertia = []
cluster_range = range(1, 150, 10) # Test number of clusters from 1 to 150 with␣
↪step 10

print("Calculating inertia for different numbers of clusters...")
for k in cluster_range:
    # n_init is set explicitly
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10) # Set n_init to 10
    kmeans.fit(latent_features)
    inertia.append(kmeans.inertia_)
    print(f"Completed KMeans for k={k}, Inertia: {kmeans.inertia_:.2f}")

# Plot the elbow curve
```

```python
plt.figure(figsize=(10, 6))
plt.plot(cluster_range, inertia, marker='o', linestyle='-')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-cluster sum of squares)')
plt.xticks(cluster_range) # Set x-axis ticks to the values in cluster_range
plt.grid(True)
plt.show()

print("\nObserve the plot to find the 'elbow' point, where the rate of decrease␣
↪in inertia slows down.")
print("This point suggests a potentially optimal number of clusters.")
```

```
Calculating inertia for different numbers of clusters…
Completed KMeans for k=1, Inertia: 26304.91
Completed KMeans for k=11, Inertia: 1276.04
Completed KMeans for k=21, Inertia: 844.78
Completed KMeans for k=31, Inertia: 751.10
Completed KMeans for k=41, Inertia: 719.07
Completed KMeans for k=51, Inertia: 702.02
Completed KMeans for k=61, Inertia: 692.10
Completed KMeans for k=71, Inertia: 686.42
Completed KMeans for k=81, Inertia: 680.36
Completed KMeans for k=91, Inertia: 676.08
Completed KMeans for k=101, Inertia: 672.34
Completed KMeans for k=111, Inertia: 668.55
Completed KMeans for k=121, Inertia: 664.47
Completed KMeans for k=131, Inertia: 662.21
Completed KMeans for k=141, Inertia: 657.87
```



Observe the plot to find the 'elbow' point, where the rate of decrease in inertia slows down.
This point suggests a potentially optimal number of clusters.

[140]:

```
import numpy as np
import joblib
from sklearn.cluster import KMeans
from sklearn.decomposition import TruncatedSVD  # PCA also fine if embeddings␣
↪are dense
from sklearn.metrics.pairwise import cosine_similarity

# latent_features: np.ndarray of shape (n_movies, D_IN)  # encoder output (e.g.
↪, 256)
# merged_df: DataFrame with at least 'title' column

# ---- Config ----
USE_REDUCER   = True          # set False to cluster on encoder dims directly
N_COMPONENTS  = 128           # target dim for KMeans space
N_CLUSTERS    = 10            # elbow choice
RANDOM_STATE  = 42
N_INIT        = 10            # explicit for older sklearn

# ---- Sanity on encoder output ----
```

```python
D_IN = int(latent_features.shape[1])
print(f"[ENCODER] latent_features shape: {latent_features.shape} (D_IN={D_IN})")


# ---- Prepare feature space for KMeans ----
if USE_REDUCER:
    # If a reducer was previously loaded in the notebook, only reuse it if it matches D_IN -> N_COMPONENTS.
    # Otherwise (or if none), fit a fresh reducer on the encoder embeddings.
    reuse = False
    if 'reducer' in globals() and reducer is not None:
        rin  = getattr(reducer, "n_features_in_", None)
        rout = getattr(reducer, "n_components", None)
        reuse = (rin == D_IN and rout == N_COMPONENTS)
        print(f"[Reducer] Found existing reducer: in={rin} out={rout}  -> reuse={reuse}")

    if not reuse:
        reducer = TruncatedSVD(n_components=N_COMPONENTS, random_state=RANDOM_STATE)
        reducer.fit(latent_features)
        rin  = getattr(reducer, "n_features_in_", None)
        rout = getattr(reducer, "n_components", None)
        if rin != D_IN or rout != N_COMPONENTS:
            raise ValueError(f"[Reducer] Mismatch after fit: got in={rin}, out={rout}, "
                             f"expected in={D_IN}, out={N_COMPONENTS}")

    features_for_kmeans = reducer.transform(latent_features)  # shape: (n_movies, N_COMPONENTS)
    print(f"[Reducer] OK  ({type(reducer).__name__}): in={D_IN} → out={features_for_kmeans.shape[1]}")
else:
    reducer = None
    features_for_kmeans = latent_features
    print(f"[Reducer] Disabled. KMeans will use {features_for_kmeans.shape[1]}-D embeddings")

# ---- KMeans on the chosen feature space ----
dim_for_kmeans = features_for_kmeans.shape[1]
print(f"[KMeans] Fitting K={N_CLUSTERS} on dim={dim_for_kmeans} ...")
kmeans = KMeans(n_clusters=N_CLUSTERS, random_state=RANDOM_STATE, n_init=N_INIT)
cluster_labels = kmeans.fit_predict(features_for_kmeans)
merged_df['cluster'] = cluster_labels
print("[KMeans] Done. Cluster counts:", np.bincount(cluster_labels))
```

```python
# ---- Verify KMeans expects the same dim we trained with ----
km_in = getattr(kmeans, "n_features_in_", None)
if km_in is None and hasattr(kmeans, "cluster_centers_"):
    km_in = int(kmeans.cluster_centers_.shape[1])
if km_in != dim_for_kmeans:
    raise RuntimeError(f"[KMeans] n_features_in_={km_in} but trained on␣
↪dim={dim_for_kmeans} - something's off.")

# ---- Save artifacts for inference (HF Spaces) ----
joblib.dump(kmeans, "kmeans_model.joblib")
print("[Save] kmeans_model.joblib")
if reducer is not None:
    joblib.dump(reducer, "kmeans_input_reducer.joblib")
    print(f"[Save] kmeans_input_reducer.joblib (apply AFTER encoder:␣
↪{D_IN}→{N_COMPONENTS})")

# ---- Helper to get cluster-based recommendations (ranked by similarity) ----
# We compute similarity in the SAME space used by KMeans.
def get_clustering_recommendations(title, df, features_for_kmeans,␣
↪num_recommendations=10):
    """
    Recommend movies from the same cluster, ranked by cosine similarity
    in the KMeans feature space (reducer output if used, otherwise encoder␣
↪space).
    """
    if title not in df['title'].values:
        print(f"Movie '{title}' not found in the dataset.")
        return df.iloc[0:0][['title', 'overview_sentiment_score', 'genres',␣
↪'keywords', 'cluster']]

    idx = df.index[df['title'] == title][0]
    c = int(df.at[idx, 'cluster'])
    members_idx = df.index[df['cluster'] == c].tolist()
    members_idx = [i for i in members_idx if i != idx]
    if not members_idx:
        print(f"No other movies found in the same cluster as '{title}'.")
        return df.iloc[0:0][['title', 'overview_sentiment_score', 'genres',␣
↪'keywords', 'cluster']]

    q = features_for_kmeans[idx].reshape(1, -1)
    M = features_for_kmeans[members_idx]
    sims = cosine_similarity(q, M).flatten()
    order = np.argsort(-sims)[:num_recommendations]
    top_idx = [members_idx[i] for i in order]
    top_scores = sims[order]
```

```python
    out = df.loc[top_idx, ['title', 'overview_sentiment_score', 'genres',␣
↪'keywords', 'cluster']].copy()
    out.insert(1, 'cluster_similarity', top_scores)
    return out

# ---- Example usage ----
clustering_movie_title = 'Avatar'
num_recommendations_clustering = 10

clustering_recommendations = get_clustering_recommendations(
    clustering_movie_title,
    merged_df,
    features_for_kmeans,
    num_recommendations=num_recommendations_clustering
)

print(f"\nItem-Based Clustering Recommendations for '{clustering_movie_title}':
↪")
display(clustering_recommendations)

print("\nSample Movies from Cluster 0:")
display(merged_df[merged_df['cluster'] == 0].head())

print("\nSample Movies from Cluster 1:")
display(merged_df[merged_df['cluster'] == 1].head())
```

```
[ENCODER] latent_features shape: (4809, 128)  (D_IN=128)
[Reducer] Found existing reducer: in=256 out=128  -> reuse=False
[Reducer] OK  (TruncatedSVD): in=128 → out=128
[KMeans] Fitting K=10 on dim=128 …
[KMeans] Done. Cluster counts: [724 500 866 532 310 704  83 282  41 767]
[Save] kmeans_model.joblib
[Save] kmeans_input_reducer.joblib (apply AFTER encoder: 128→128)

Item-Based Clustering Recommendations for 'Avatar':

                                     title  cluster_similarity  \
1487                           The Watcher            0.999598
2997                   Recess: School's Out           0.999587
2487                           The Cookout            0.999586
3214                           Crazy Heart            0.999581
4698                    Theresa Is a Mother           0.999581
395                            The Holiday            0.999580
2555              The Last House on the Left          0.999578
1583                                  Blow            0.999575
1359  Austin Powers: The Spy Who Shagged Me           0.999569
4403                        Alone With Her            0.999569
```

```
      overview_sentiment_score                                genres  \
1487                   -0.8294                      Mystery Thriller
2997                    0.3659  ScienceFiction Animation Comedy Family
2487                    0.4215                          Comedy Drama
3214                    0.1406                   Drama Music Romance
4698                   -0.6124                    Music Comedy Drama
395                     0.0129                        Comedy Romance
2555                   -0.9666            Crime Thriller Horror Drama
1583                   -0.2316                           Crime Drama
1359                   -0.7964   Adventure Comedy Crime ScienceFiction
4403                    0.7269           Crime Drama Romance Thriller

                                               keywords  cluster
1487  chicago fbi menace coveredinvestigation stateo…        3
2997  holiday elementaryschool friends basedontvseri…        3
2487                                             comedy        3
3214  taxi countrymusic journalist guitar bar musici…        3
4698                                                           3
395   holiday londonengland filmmaking christmaspart…        3
2555  rape whitetrash revenge murder dysfunctionalfa…        3
1583  1970s warondrugs drugaddiction drugtraffic dru…        3
1359  savingtheworld moon submarine clone spy cia sh…        3
4403  obsession hiddencamera stalker independentfilm…        3


Sample Movies from Cluster 0:

       budget                            genres      id  \
2   245000000            Action Adventure Crime  206647
4   260000000  Action Adventure ScienceFiction   49529
8   250000000          Adventure Fantasy Family     767
17  380000000          Adventure Action Fantasy    1865
24  207000000            Adventure Drama Action     254

                                             keywords original_language  \
2   spy basedonnovel secretagent sequel mi6 britis…                en
4   basedonnovel mars medallion spacetravel prince…                en
8   witch magic broom schoolofwitchcraft wizardry …                en
17  sea captain mutiny sword primeminister sailing…                en
24  filmbusiness screenplay showbusiness filmmakin…                en

                                 original_title  \
2                                       Spectre
4                                   John Carter
8        Harry Potter and the Half-Blood Prince
17  Pirates of the Caribbean: On Stranger Tides
24                                    King Kong
```

```
                                            overview  popularity  \
2   A cryptic message from Bond’s past sends him o…  107.376788
4   John Carter is a war-weary, former military ca…   43.926995
8   As Harry begins his sixth year at Hogwarts, he…   98.885637
17  Captain Jack Sparrow crosses paths with a woma…  135.413856
24  In 1933 New York, an overly ambitious movie pr…   61.226010

                                production_companies  \
2   [{"name": "Columbia Pictures", "id": 5}, {"nam…
4         [{"name": "Walt Disney Pictures", "id": 2}]
8   [{"name": "Warner Bros.", "id": 6194}, {"name"…
17  [{"name": "Walt Disney Pictures", "id": 2}, {"…
24  [{"name": "WingNut Films", "id": 11}, {"name":…

                                production_countries  … vote_average  \
2   [{"iso_3166_1": "GB", "name": "United Kingdom"…  …          6.3
4   [{"iso_3166_1": "US", "name": "United States o…  …          6.1
8   [{"iso_3166_1": "GB", "name": "United Kingdom"…  …          7.4
17  [{"iso_3166_1": "US", "name": "United States o…  …          6.4
24  [{"iso_3166_1": "NZ", "name": "New Zealand"}, …  …          6.6

   vote_count  movie_id                                              cast  \
2        4466    206647  DanielCraig ChristophWaltz LéaSeydoux RalphFie…
4        2124     49529  TaylorKitsch LynnCollins SamanthaMorton Willem…
8        5293       767  DanielRadcliffe RupertGrint EmmaWatson TomFelt…
17       4948      1865  JohnnyDepp PenélopeCruz IanMcShane KevinMcNall…
24       2337       254  NaomiWatts JackBlack AdrienBrody ThomasKretsch…

  director                                              soup  \
2          SpectreA cryptic message from Bond’s past send…
4          John CarterJohn Carter is a war-weary, former …
8          Harry Potter and the Half-Blood PrinceAs Harry…
17         Pirates of the Caribbean: On Stranger TidesCap…
24         King KongIn 1933 New York, an overly ambitious…

   overview_sentiment_score  release_year sentiment_difference cluster
2                   -0.8271        2015.0               0.4659       0
4                   -0.7096        2012.0               0.3484       0
8                    0.0000        2009.0               0.3612       0
17                  -0.2411        2011.0               0.1201       0
24                   0.4767        2005.0               0.8379       0

[5 rows x 25 columns]

Sample Movies from Cluster 1:

       budget                                  genres     id  \
23  180000000                       Adventure Fantasy   2268
```

```
61   176000003  ScienceFiction Fantasy Action Adventure  76757
208  160000000                 Adventure Fantasy Action   1911
251  112000000                                   Comedy  38745
309   84000000          Fantasy Comedy Family Adventure  10214

                                              keywords original_language  \
23   england compass experiment lordship uncle pola…                en
61   jupiter space womandirector 3d interspeciesrom…                en
208  witch cave arabian scandinavia bagdad viking i…                en
251    journalist forbiddenlove princess royalcourt 3d                en
309                                    baby mask viking                en

         original_title                                          overview  \
23   The Golden Compass  After overhearing a shocking secret, precociou…
61    Jupiter Ascending  In a universe where human genetic material is …
208    The 13th Warrior  In AD 922, Arab courtier, Ahmad Ibn Fadlan acc…
251  Gulliver's Travels  Travel writer Lemuel Gulliver takes an assignm…
309     Son of the Mask  Tim Avery, an aspiring cartoonist, finds himse…

     popularity                               production_companies  \
23    42.990906  [{"name": "New Line Cinema", "id": 12}, {"name…
61    85.369080  [{"name": "Village Roadshow Pictures", "id": 7…
208   27.220157      [{"name": "Touchstone Pictures", "id": 9195}]
251   22.845143  [{"name": "Twentieth Century Fox Film Corporat…
309   17.815595  [{"name": "New Line Cinema", "id": 12}, {"name…

                                production_countries  … vote_average  \
23   [{"iso_3166_1": "GB", "name": "United Kingdom"…  …          5.8
61   [{"iso_3166_1": "US", "name": "United States o…  …          5.2
208  [{"iso_3166_1": "US", "name": "United States o…  …          6.4
251  [{"iso_3166_1": "US", "name": "United States o…  …          4.9
309  [{"iso_3166_1": "DE", "name": "Germany"}, {"is…  …          3.6

     vote_count  movie_id                                              cast  \
23         1303      2268  DakotaBlueRichards NicoleKidman DanielCraig Sa…
61         2768     76757  MilaKunis ChanningTatum SeanBean EddieRedmayne…
208         510      1911  AntonioBanderas VladimirKulich DennisStorhøi D…
251         621     38745  JackBlack AmandaPeet EmilyBlunt JasonSegel Chr…
309         338     10214  JamieKennedy AlanCumming TraylorHoward KalPenn…

    director                                              soup  \
23            The Golden CompassAfter overhearing a shocking…
61            Jupiter AscendingIn a universe where human gen…
208           The 13th WarriorIn AD 922, Arab courtier, Ahma…
251           Gulliver's TravelsTravel writer Lemuel Gullive…
309           Son of the MaskTim Avery, an aspiring cartooni…

     overview_sentiment_score  release_year sentiment_difference cluster
```

23	0.1280	2007.0	0.4892	1
61	0.7778	2015.0	1.1390	1
208	-0.4767	1999.0	0.1155	1
251	0.0000	2010.0	0.3612	1
309	0.0000	2005.0	0.3612	1

[5 rows x 25 columns]

[141]:
```python
# @title Scatter Plot for KMeans clusters from merged_df

import matplotlib.pyplot as plt
# Visualize the clusters in 2D or 3D (using PCA or t-SNE for dimensionality␣
↪reduction)

# Use PCA to reduce latent features to 2 components for visualization
from sklearn.decomposition import PCA

# the latent representations for all movies
# These are dense, lower-dimensional feature vectors learned by the autoencoder
pca = PCA(n_components=2)
latent_features_2d = pca.fit_transform(latent_features)

# Add the 2D PCA coordinates to the dataframe
merged_df['pca_comp1'] = latent_features_2d[:, 0]
merged_df['pca_comp2'] = latent_features_2d[:, 1]

# Create a scatter plot of the clusters
plt.figure(figsize=(12, 8))
sns.scatterplot(
    data=merged_df,
    x='pca_comp1',
    y='pca_comp2',
    hue='cluster',  # Color points by cluster label
    palette='viridis', # Color palette
    legend='full',
    alpha=0.6,
    s=30 # point size
)

plt.title('KMeans Clusters Visualized using PCA')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```

[142]:
```
# clustering for movie sentiment analysis from merged_df and then use that␣
↪model to predict

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

# Perform KMeans clustering on the latent features
n_clusters = 11  # Number of clusters based on elbow curve above
kmeans = KMeans(n_clusters=n_clusters, random_state=42, n_init=10) # Explicitly␣
↪set n_init
cluster_labels = kmeans.fit_predict(latent_features)

# Add cluster labels to the dataframe
merged_df['cluster'] = cluster_labels

# Function to get recommendations based on item-based clustering
def get_clustering_recommendations(title, df, num_recommendations=10):
    """
    Generates movie recommendations based on finding other movies in the same␣
↪cluster.
```

```
    Args:
        title (str): The title of the input movie.
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df).
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing recommended movies from the same␣
↪cluster.
                      Returns an empty DataFrame if the movie is not found or␣
↪cluster is empty.
    """
    if title not in df['title'].values:
        print(f"Movie '{title}' not found in the dataset.")
        return pd.DataFrame()

    # Get the cluster of the input movie
    input_movie_cluster = df[df['title'] == title]['cluster'].iloc[0]

    # Find all movies in the same cluster
    movies_in_same_cluster = df[df['cluster'] == input_movie_cluster]

    # Exclude the input movie itself
    recommended_movies = movies_in_same_cluster[movies_in_same_cluster['title']␣
↪!= title]

    # If there are more movies in the cluster than needed, sample them randomly
    if len(recommended_movies) > num_recommendations:
        recommended_movies = recommended_movies.sample(n=num_recommendations,␣
↪random_state=42)
    elif recommended_movies.empty:
         print(f"No other movies found in the same cluster as '{title}'.")


    # Return the relevant columns for recommendations
    return recommended_movies[['title', 'overview_sentiment_score', 'genres',␣
↪'keywords', 'cluster']]


# @title Generate Recommendations using Item-Based Clustering

clustering_movie_title = 'Water'  #@param {type:"string"}
num_recommendations_clustering = 10  #@param {type:"slider", min:1, max:20,␣
↪step:1}
```

```
clustering_recommendations = get_clustering_recommendations(
    clustering_movie_title,
    merged_df,
    num_recommendations=num_recommendations_clustering
)

print(f"\nItem-Based Clustering Recommendations for '{clustering_movie_title}':
↪")
display(clustering_recommendations)
```

```
Item-Based Clustering Recommendations for 'Water':
                                 title  overview_sentiment_score  \
2114                          Bad Moms                    0.8807
1181                       Shallow Hal                    0.6369
4283                 The Hebrew Hammer                   -0.2023
4624                 Nothing But a Man                    0.5267
4787                         Dry Spell                    0.0000
4586                       Quinceañera                    0.5994
4554            Everything Put Together                    0.0000
4280  Eddie: The Sleepwalking Cannibal                    0.5267
1417                      Flash Gordon                    0.5859
3935             8 Heads in a Duffel Bag                   -0.7351

              genres                                          keywords  \
2114          Comedy  alcohol bar party divorce family hitandrun bat…
1181  Comedy Romance  overweight beauty hypnosis overweightman overw…
4283   Action Comedy                                   independentfilm
4624           Drama
4787  Comedy Romance  dating divorce sexscene sexcomedy antiromantic…
4586           Drama
4554           Drama                                   independentfilm
4280   Horror Comedy                 cannibal sleepwalking artteacher
1417  ScienceFiction  emperor solareclipse prince tvduel fighter dec…
3935    Comedy Crime                          mexico vacation murder head

      cluster
2114        8
1181        8
4283        8
4624        8
4787        8
4586        8
4554        8
4280        8
1417        8
```

```
3935        8
```

[143]:
```
# analyze the contents of a few clusters to understand what kind of movies are grouped together.
print(f"\nSample Movies from Cluster 0:")
display(merged_df[merged_df['cluster'] == 0].head())

print(f"\nSample Movies from Cluster 1:")
display(merged_df[merged_df['cluster'] == 1].head())
```

```
Sample Movies from Cluster 0:
       budget                                     genres      id  \
2   245000000                     Action Adventure Crime  206647
15  225000000                   Adventure Family Fantasy    2454
17  380000000                   Adventure Action Fantasy    1865
28  150000000  Action Adventure ScienceFiction Thriller  135397
33  210000000  Adventure Action ScienceFiction Thriller   36668

                                             keywords original_language  \
2   spy basedonnovel secretagent sequel mi6 britis…                en
15  basedonnovel fictionalplace brothersisterrelat…                en
17  sea captain mutiny sword primeminister sailing…                en
28  monster dna tyrannosaurusrex velociraptor isla…                en
33  mutant marvelcomic basedoncomicbook superhuman…                en

                                 original_title  \
2                                       Spectre
15     The Chronicles of Narnia: Prince Caspian
17  Pirates of the Caribbean: On Stranger Tides
28                               Jurassic World
33                        X-Men: The Last Stand

                                             overview  popularity  \
2   A cryptic message from Bond’s past sends him o…  107.376788
15  One year after their incredible adventures in …   53.978602
17  Captain Jack Sparrow crosses paths with a woma…  135.413856
28  Twenty-two years after the events of Jurassic …  418.708552
33  When a cure is found to treat mutations, lines…    3.857526

                                 production_companies  \
2   [{"name": "Columbia Pictures", "id": 5}, {"nam…
15  [{"name": "Walt Disney", "id": 5888}, {"name":…
17  [{"name": "Walt Disney Pictures", "id": 2}, {"…
28  [{"name": "Universal Studios", "id": 13}, {"na…
33  [{"name": "Ingenious Film Partners", "id": 289…
```

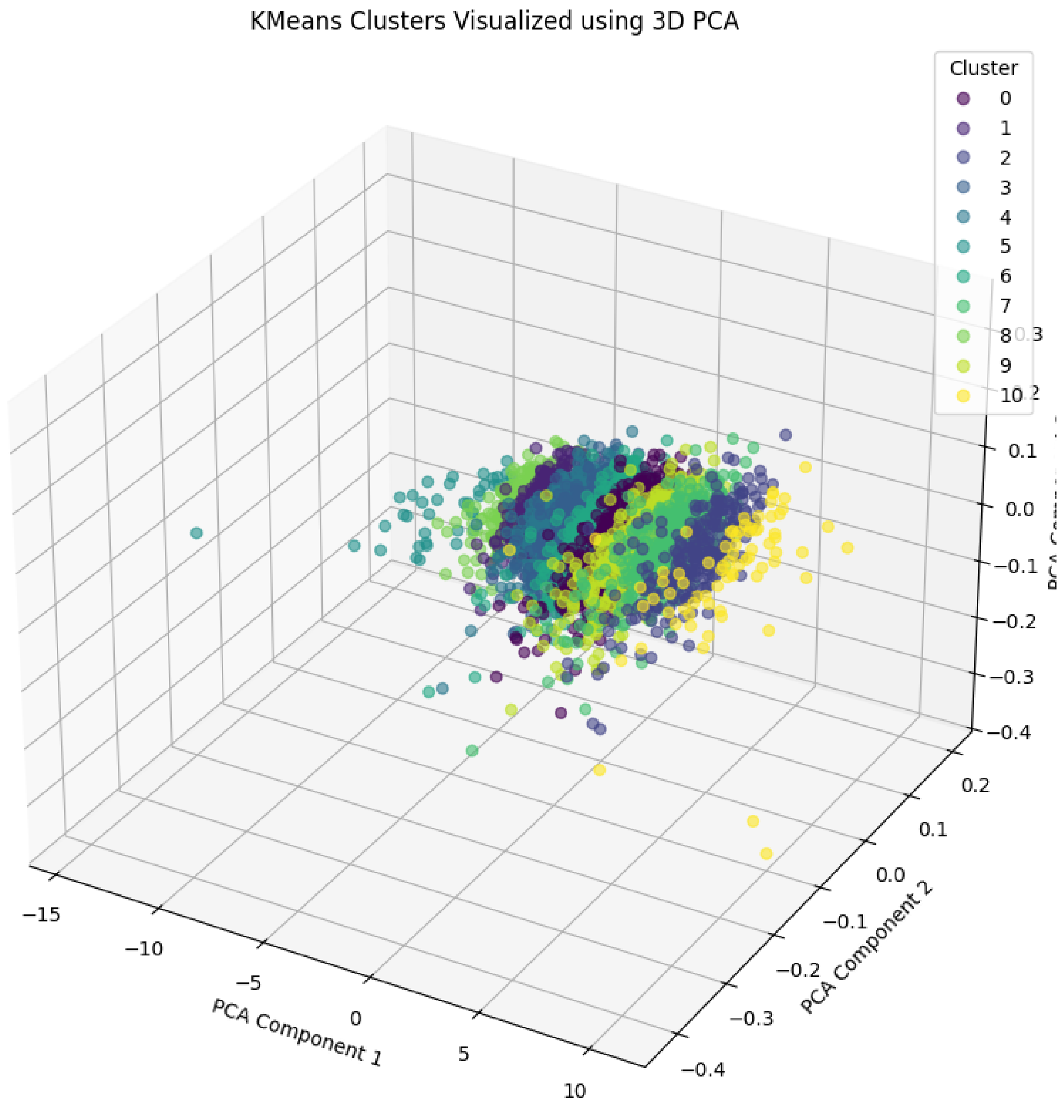
```
                              production_countries  … movie_id  \
2   [{"iso_3166_1": "GB", "name": "United Kingdom"…  …   206647
15  [{"iso_3166_1": "CZ", "name": "Czech Republic"…  …     2454
17  [{"iso_3166_1": "US", "name": "United States o…  …     1865
28  [{"iso_3166_1": "US", "name": "United States o…  …   135397
33  [{"iso_3166_1": "CA", "name": "Canada"}, {"iso…  …    36668

                                                 cast  director  \
2   DanielCraig ChristophWaltz LéaSeydoux RalphFie…
15  BenBarnes WilliamMoseley AnnaPopplewell Skanda…
17  JohnnyDepp PenélopeCruz IanMcShane KevinMcNall…
28  ChrisPratt BryceDallasHoward IrrfanKhan Vincen…
33  HughJackman HalleBerry IanMcKellen PatrickStew…

                                                 soup  \
2   SpectreA cryptic message from Bond's past send…
15  The Chronicles of Narnia: Prince CaspianOne ye…
17  Pirates of the Caribbean: On Stranger TidesCap…
28  Jurassic WorldTwenty-two years after the event…
33  X-Men: The Last StandWhen a cure is found to t…

   overview_sentiment_score  release_year  sentiment_difference  cluster  \
2                   -0.8271        2015.0                0.4659        0
15                  -0.4939        2008.0                0.1327        0
17                  -0.2411        2011.0                0.1201        0
28                   0.0000        2015.0                0.3612        0
33                   0.6705        2006.0                1.0317        0

   pca_comp1 pca_comp2
2   0.982656 -0.047914
15  0.555036 -0.001136
17  1.217849  0.019722
28  1.012465 -0.053139
33  1.265853 -0.054737

[5 rows x 27 columns]


Sample Movies from Cluster 1:
        budget                                    genres      id  \
61   176000003  ScienceFiction Fantasy Action Adventure   76757
313   99000000                                 Animation  227973
314   10000000            Crime Drama Mystery Thriller   29193
315   98000000                Adventure Action Fantasy    1734
317   94000000                       Drama History War   76758

                                              keywords original_language  \
61   jupiter space womandirector 3d interspeciesrom…                en
```

```
313   basedoncomicstrip family 3d charliebrown snoopy                en
314     dialogue confidence invention independentfilm                en
315                         son ancientegypt bracelet                en
317                      forcedprostitution childrape                zh

          original_title                                         overview  \
61      Jupiter Ascending  In a universe where human genetic material is …
313     The Peanuts Movie  Snoopy embarks upon his greatest mission as he…
314  The Spanish Prisoner  An employee of a corporation with a lucrative …
315     The Mummy Returns  Rick and Evelyn O'Connell, along with their 8 …
317                       A Westerner finds refuge with a group of women…

    popularity                               production_companies  \
61   85.369080  [{"name": "Village Roadshow Pictures", "id": 7…
313  34.308098  [{"name": "Blue Sky Studios", "id": 9383}, {"n…
314   3.091077  [{"name": "Jean Doumanian Productions", "id": …
315  41.862983  [{"name": "Universal Pictures", "id": 33}, {"n…
317  12.516546  [{"name": "Beijing New Picture Film Co. Ltd.",…

                                  production_countries  … movie_id  \
61   [{"iso_3166_1": "US", "name": "United States o…  …    76757
313  [{"iso_3166_1": "US", "name": "United States o…  …   227973
314  [{"iso_3166_1": "US", "name": "United States o…  …    29193
315  [{"iso_3166_1": "US", "name": "United States o…  …     1734
317  [{"iso_3166_1": "CN", "name": "China"}, {"iso_…  …    76758

                                                cast  director  \
61   MilaKunis ChanningTatum SeanBean EddieRedmayne…
313  NoahSchnapp BillMelendez VenusSchultheis Hadle…
314  SteveMartin CampbellScott BenGazzara RebeccaPi…
315  BrendanFraser RachelWeisz JohnHannah ArnoldVos…
317  ChristianBale NiNi TongDawei ZhangXinyi Shigeo…

                                                soup  \
61   Jupiter AscendingIn a universe where human gen…
313  The Peanuts MovieSnoopy embarks upon his great…
314  The Spanish PrisonerAn employee of a corporati…
315  The Mummy ReturnsRick and Evelyn O'Connell, al…
317  The Flowers of WarA Westerner finds refuge wit…

    overview_sentiment_score  release_year  sentiment_difference  cluster  \
61                    0.7778        2015.0                1.1390        1
313                   0.8555        2015.0                1.2167        1
314                  -0.3818        1997.0                0.0206        1
315                   0.4767        2001.0                0.8379        1
317                  -0.4404        2011.0                0.0792        1

    pca_comp1 pca_comp2
```

```
61  -2.526752 -0.140638
313 -3.182279  0.010242
314 -2.415285  0.058463
315 -3.440030 -0.032730
317 -2.799674  0.055517

[5 rows x 27 columns]
```

[144]:

```python
# @title Scatter Plot for KMeans clusters from merged_df
pca_3d = PCA(n_components=3)
latent_features_3d = pca_3d.fit_transform(latent_features)

merged_df['pca_comp3'] = latent_features_3d[:, 2]

fig = plt.figure(figsize=(12, 10))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(
    merged_df['pca_comp1'],
    merged_df['pca_comp2'],
    merged_df['pca_comp3'],
    c=merged_df['cluster'],
    cmap='viridis',
    s=30,
    alpha=0.6
)

ax.set_title('KMeans Clusters Visualized using 3D PCA')
ax.set_xlabel('PCA Component 1')
ax.set_ylabel('PCA Component 2')
ax.set_zlabel('PCA Component 3')

# Add legend
legend = ax.legend(*scatter.legend_elements(), title="Cluster")
plt.show()
```



[145]:
```
def query_clustering_recommendations(query, df, tfidf, kmeans,␣
↪num_recommendations=10):
    """
    Generates movie recommendations by finding the cluster of a movie
    identified by a query (which can be a title or content keyword),
    and then recommending other movies within that same cluster.

    Args:
        query (str): The input query (movie title, keyword, plot, etc.).
        df (pd.DataFrame): The DataFrame containing movie information␣
↪(merged_df)
                           with a 'cluster' column.
```

```
        tfidf (TfidfVectorizer): The fitted TF-IDF vectorizer.
        kmeans (KMeans): The fitted KMeans model.
        num_recommendations (int, optional): The number of recommendations to␣
↪generate. Defaults to 10.

    Returns:
        pd.DataFrame: A DataFrame containing recommended movies from the␣
↪identified cluster,
                      including sentiment score and other relevant features.
                      Returns an empty DataFrame if no cluster is identified or␣
↪cluster is empty.
    """
    input_movie_cluster = None
    query_vector = None

    # --- 1. Find the most relevant movie based on the query using TF-IDF and␣
↪cosine similarity ---
    # This step is similar to the initial keyword search but aims to identify a␣
↪single
    # representative movie from which to find the cluster.

    try:
        query_vector = tfidf.transform([str(query)])
        if query_vector.sum() == 0:
            print(f"Query '{query}' does not contain words in the vocabulary.")
            return pd.DataFrame()
    except Exception as e:
         print(f"Error transforming query '{query}': {e}")
         return pd.DataFrame()

    # Calculate cosine similarity between the query vector and all movie soup␣
↪vectors
    keyword_sim_scores = linear_kernel(query_vector, tfidf_matrix).flatten()

    # Find the index of the movie with the highest similarity score
    # This movie will be used to identify the cluster
    most_similar_movie_idx = keyword_sim_scores.argmax()
    confidence_score = keyword_sim_scores[most_similar_movie_idx]

    if confidence_score == 0:
        print(f"Query '{query}' did not match any movie content significantly.")
        return pd.DataFrame()

    # Get the title of the most similar movie
    identified_movie_title = df['title'].iloc[most_similar_movie_idx]
```

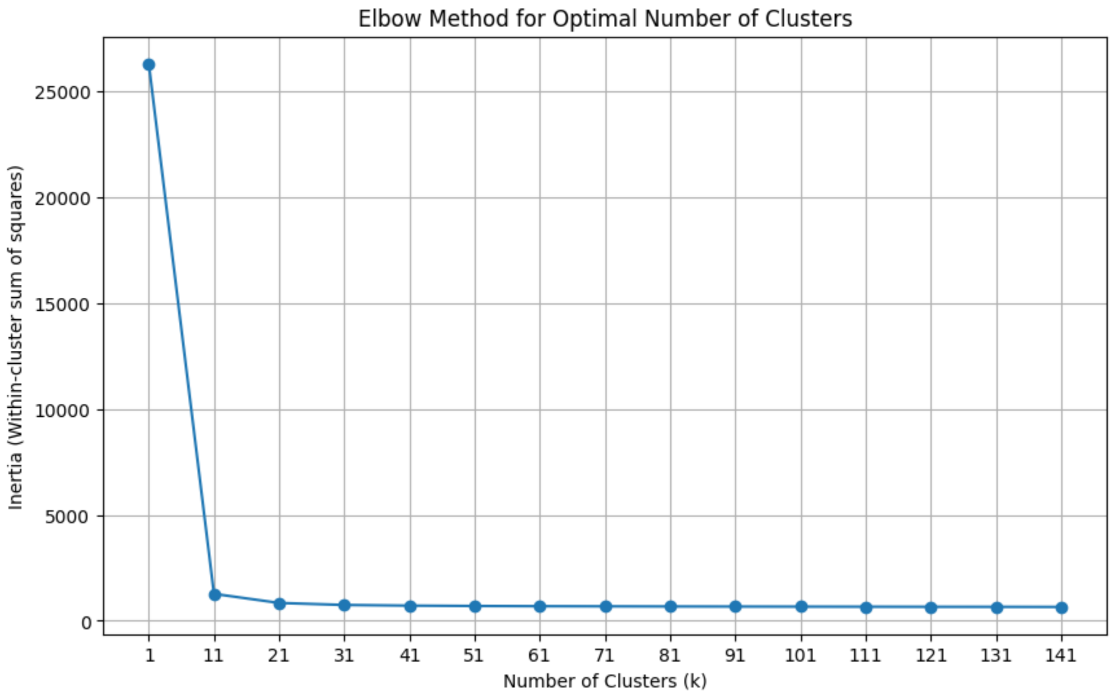
```python
    print(f"Query '{query}' is most similar to movie: '{identified_movie_title}' (Confidence: {confidence_score:.4f})")


    # --- 2. Get the cluster of the identified movie ---
    if identified_movie_title not in df['title'].values:
        print(f"Identified movie '{identified_movie_title}' not found in the dataframe.")
        return pd.DataFrame()

    input_movie_cluster = df[df['title'] == identified_movie_title]['cluster'].iloc[0]
    print(f"Identified movie '{identified_movie_title}' belongs to Cluster: {input_movie_cluster}")


    # --- 3. Find other movies in the same cluster ---
    movies_in_same_cluster = df[df['cluster'] == input_movie_cluster].copy() # Create a copy to avoid SettingWithCopyWarning


    # Exclude the identified movie itself
    recommended_movies = movies_in_same_cluster[movies_in_same_cluster['title'] != identified_movie_title]

    # If there are more movies in the cluster than needed, sample them randomly
    if len(recommended_movies) > num_recommendations:
        # Use the confidence score from the query match as a potential way to rank within the cluster
        # However, within a cluster, items are existed similar. Random sampling or ranking by
        # sentiment/vote might be more appropriate than the initial query match score.
        # Let's add the initial confidence score for context but sample randomly or sort by sentiment/popularity.

        # Option A: Sample randomly
        # recommended_movies = recommended_movies.sample(n=num_recommendations, random_state=42)

        # Option B: Sort by a metric like overview_sentiment_score or vote_average (if available and relevant)
        # Sorting by sentiment might recommend movies in the cluster with similar emotional tone.
```

```
        recommended_movies = recommended_movies.
↪sort_values(by='overview_sentiment_score', ascending=False).
↪head(num_recommendations)

    elif recommended_movies.empty:
        print(f"No other movies found in the same cluster as␣
↪'{identified_movie_title}'.")
        return pd.DataFrame()


    # Add a 'Reason' and 'Confidence Score' column (using the initial query␣
↪match score for context)
    # Note: The confidence score here is for the initial query match, not␣
↪similarity within the cluster.
    # Within the cluster, items are existed to be similar.
    recommended_movies['Reason'] = f"Recommended from Cluster␣
↪{input_movie_cluster} (Most similar to query '{query}' was␣
↪'{identified_movie_title}')"
    recommended_movies['Initial Query Confidence Score'] = confidence_score


    # Return the relevant columns for recommendations
    return recommended_movies[['title', 'Reason', 'Initial Query Confidence␣
↪Score',
                               'overview_sentiment_score', 'genres', 'keywords',␣
↪'cluster']].rename(columns={'title': 'Recommended Movie'})


# Check if latent_features is defined (it should be from the autoencoder␣
↪section)
if 'latent_features' not in globals():
    print("Latent features not found. Please run the Autoencoder section first.
↪")
else:
    # Use the elbow method to find the optimal number of clusters for KMeans
    # It's computationally expensive to run for a very large range, let's pick␣
↪a reasonable range
    inertia = []
    # A smaller step size might give a better elbow point but takes longer
    # Let's try a range of 1 to 150 with a step of 10 first.
    cluster_range = range(1, 150, 10)

    print("Calculating inertia for different numbers of clusters (Elbow Method).
↪..")
    for k in cluster_range:
```

```
        # n_init is set explicitly to avoid warning in newer scikit-learn␣
↪versions
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(latent_features)
        inertia.append(kmeans.inertia_)
        # Optional: Print progress
        print(f"Completed KMeans for k={k}, Inertia: {kmeans.inertia_:.2f}")

    # Plot the elbow curve
    plt.figure(figsize=(10, 6))
    plt.plot(cluster_range, inertia, marker='o', linestyle='-')
    plt.title('Elbow Method for Optimal Number of Clusters')
    plt.xlabel('Number of Clusters (k)')
    plt.ylabel('Inertia (Within-cluster sum of squares)')
    # Set x-axis ticks to the values in cluster_range for clarity
    plt.xticks(cluster_range)
    plt.grid(True)
    plt.show()

    print("\nObserve the plot to find the 'elbow' point, where the rate of␣
↪decrease in inertia slows down.")
    print("This point suggests a potentially optimal number of clusters.")
    print("Based on the previous run, around 11-21 seemed reasonable. Let's␣
↪proceed with a chosen number.")

    # Perform KMeans clustering with the chosen number of clusters
    # Choose the number of clusters based on the elbow plot observation
    # Let's pick a number from the visually determined elbow range, e.g., 15
    n_clusters_chosen = 11

    print(f"\nPerforming KMeans clustering with {n_clusters_chosen} clusters on␣
↪latent features...")
    kmeans_model = KMeans(n_clusters=n_clusters_chosen, random_state=42,␣
↪n_init=10) # Explicitly set n_init
    cluster_labels = kmeans_model.fit_predict(latent_features)

    # Add cluster labels to the dataframe
    # Ensure the column name doesn't clash if we ran this section before
    merged_df['kmeans_cluster'] = cluster_labels

    print(f"Clustering complete. Added '{'kmeans_cluster'}' column to merged_df.
↪")
    print("\nDistribution of movies per cluster:")
    print(merged_df['kmeans_cluster'].value_counts().sort_index())
```

```python
# @title Generate Recommendations using Item-Based Clustering with Query

clustering_search_query = 'Liar'  #@param {type:"string"}
num_recommendations_clustering_query = 10  #@param {type:"slider", min:1,␣
↪max:20, step:1}

clustering_recommendations_query = query_clustering_recommendations(
    clustering_search_query,
    merged_df,
    tfidf, # Pass the fitted TF-IDF vectorizer
    kmeans_model, # Pass the trained KMeans model
    num_recommendations=num_recommendations_clustering_query
)

print(f"\nItem-Based Clustering Recommendations for query␣
↪'{clustering_search_query}':")
display(clustering_recommendations_query)
```

```
Calculating inertia for different numbers of clusters (Elbow Method)…
Completed KMeans for k=1, Inertia: 26304.91
Completed KMeans for k=11, Inertia: 1276.04
Completed KMeans for k=21, Inertia: 844.78
Completed KMeans for k=31, Inertia: 751.10
Completed KMeans for k=41, Inertia: 719.07
Completed KMeans for k=51, Inertia: 702.02
Completed KMeans for k=61, Inertia: 692.10
Completed KMeans for k=71, Inertia: 686.42
Completed KMeans for k=81, Inertia: 680.36
Completed KMeans for k=91, Inertia: 676.08
Completed KMeans for k=101, Inertia: 672.34
Completed KMeans for k=111, Inertia: 668.55
Completed KMeans for k=121, Inertia: 664.47
Completed KMeans for k=131, Inertia: 662.21
Completed KMeans for k=141, Inertia: 657.87
```




```
Observe the plot to find the 'elbow' point, where the rate of decrease in
inertia slows down.
This point suggests a potentially optimal number of clusters.
Based on the previous run, around 11-21 seemed reasonable. Let's proceed with a
chosen number.

Performing KMeans clustering with 11 clusters on latent features…
Clustering complete. Added 'kmeans_cluster' column to merged_df.

Distribution of movies per cluster:
kmeans_cluster
0     658
1     445
2     248
3     703
4     661
5      41
6     762
7     404
8     243
9     569
10     75
Name: count, dtype: int64
Query 'Liar' is most similar to movie: 'Liar Liar' (Confidence: 0.2378)
```

```
Identified movie 'Liar Liar' belongs to Cluster: 1

Item-Based Clustering Recommendations for query 'Liar':
                             Recommended Movie  \
3380                         A Dog Of Flanders
4746                                  20 Dates
3707                                State Fair
4600                       The Broadway Melody
1261                                    Amélie
3435  The Work and the Glory II: American Zion
492                             Top Cat Begins
4763                             Bending Steel
4184                                     Vaalu
3377                                Veer-Zaara

                                                Reason  \
3380  Recommended from Cluster 1 (Most similar to qu…
4746  Recommended from Cluster 1 (Most similar to qu…
3707  Recommended from Cluster 1 (Most similar to qu…
4600  Recommended from Cluster 1 (Most similar to qu…
1261  Recommended from Cluster 1 (Most similar to qu…
3435  Recommended from Cluster 1 (Most similar to qu…
492   Recommended from Cluster 1 (Most similar to qu…
4763  Recommended from Cluster 1 (Most similar to qu…
4184  Recommended from Cluster 1 (Most similar to qu…
3377  Recommended from Cluster 1 (Most similar to qu…

      Initial Query Confidence Score  overview_sentiment_score  \
3380                         0.23781                    0.9765
4746                         0.23781                    0.9732
3707                         0.23781                    0.9607
4600                         0.23781                    0.9549
1261                         0.23781                    0.9522
3435                         0.23781                    0.9485
492                          0.23781                    0.9474
4763                         0.23781                    0.9398
4184                         0.23781                    0.9382
3377                         0.23781                    0.9252

                   genres                                          keywords  \
3380         Drama Family
4746       Romance Comedy   hiddencamera biography realityshow mockumentary
3707        Music Romance                                         statefair
4600  Drama Music Romance  musical singer pre-code wisecrackhumor earlyso…
1261       Comedy Romance  paris lovetriangle ghosttrain sex-shop shyness…
3435                Drama
492      Comedy Animation                                                3d
```

```
4763          Documentary
4184       Comedy Romance                                         teenagelove
3377        Drama Romance  loveofone'slife pilot classsociety pakistan in…

      cluster
3380        1
4746        1
3707        1
4600        1
1261        1
3435        1
492         1
4763        1
4184        1
3377        1
```

5 Feature Engineering Refinement

5.1 Feature Engineering Refinement: Text Preprocessing

Let's explore different text preprocessing techniques to see if they improve the quality of our text features ('soup'). Standard steps often include:

- **Lowercasing**: Convert all text to lowercase to treat words like "Movie" and "movie" the same.
- **Punctuation Removal**: Remove punctuation marks.
- **Stop Word Removal**: Remove common words that don't carry much meaning (like 'the', 'a', 'is'). We are already doing this in `TfidfVectorizer`, but we could try a custom list or different approach.
- **Stemming or Lemmatization**: Reduce words to their root form (stemming) or dictionary form (lemmatization) to group similar words.

[146]:
```
# Experiment with Text Preprocessing

print("## Experimenting with Text Preprocessing Techniques\n")

import re
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
# from nltk.stem.wordnet import WordNetLemmatizer # Will replace with spaCy
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import spacy # Import spaCy

# Download necessary NLTK data (if not already downloaded)
try:
```

```python
    nltk.data.find('corpora/stopwords')
except LookupError:
    nltk.download('stopwords')

# Load the English spaCy model
try:
    nlp = spacy.load("en_core_web_sm")
except OSError:
    print("Downloading spaCy model 'en_core_web_sm'...")
    !python -m spacy download en_core_web_sm
    nlp = spacy.load("en_core_web_sm")


stemmer = PorterStemmer()
# lemmatizer = WordNetLemmatizer() # Will use spaCy for lemmatization
stop_words = set(stopwords.words('english'))

# Define different preprocessing functions
def preprocess_text_basic(text):
    """Basic preprocessing: lowercase, remove non-alphanumeric."""
    if pd.isna(text):
        return ""
    text = str(text).lower()
    text = re.sub(r'[^a-z0-9\s]', '', text) # Keep only lowercase letters and␣
↪numbers
    return text

def preprocess_text_stopwords(text):
    """Preprocessing with stop word removal."""
    if pd.isna(text):
        return ""
    text = preprocess_text_basic(text)
    text = ' '.join([word for word in text.split() if word not in stop_words])
    return text

def preprocess_text_stemming(text):
    """Preprocessing with stemming."""
    if pd.isna(text):
        return ""
    text = preprocess_text_stopwords(text) # Start after stop words removed
    text = ' '.join([stemmer.stem(word) for word in text.split()])
    return text

def preprocess_text_lemmatization_spacy(text):
    """Preprocessing with spaCy lemmatization."""
    if pd.isna(text):
        return ""
```

```
    # Use basic preprocessing and stop word removal first
    text = preprocess_text_stopwords(text)
    doc = nlp(text)
    # Keep only the lemma for each token, and join them back
    lemmatized_text = " ".join([token.lemma_ for token in doc])
    return lemmatized_text


# Apply different preprocessing methods to the 'soup' column
# Create new columns or dataframes for experimentation to avoid overwriting␣
↪original 'soup'
merged_df_preprocessed = merged_df.copy()

print("Applying preprocessing techniques...")
merged_df_preprocessed['soup_basic'] = merged_df_preprocessed['soup'].
↪apply(preprocess_text_basic)
merged_df_preprocessed['soup_stopwords'] = merged_df_preprocessed['soup'].
↪apply(preprocess_text_stopwords)
merged_df_preprocessed['soup_stemmed'] = merged_df_preprocessed['soup'].
↪apply(preprocess_text_stemming)
# Use the new spaCy lemmatization function
merged_df_preprocessed['soup_lemmatized'] = merged_df_preprocessed['soup'].
↪apply(preprocess_text_lemmatization_spacy)
print("Preprocessing complete.")
```

```
## Experimenting with Text Preprocessing Techniques

Applying preprocessing techniques…
Preprocessing complete.
```

[147]:
```
# Now, we can re-run TF-IDF and recommendation generation using these new␣
↪preprocessed 'soup' columns
# and qualitatively compare the recommendations.

print("\nGenerating recommendations using different preprocessed text:")

preprocessing_experiments = {
    'Original_soup': merged_df['soup'].fillna(''), # Use original soup with␣
↪TfidfVectorizer stop_words
    'Basic_Preprocessing': merged_df_preprocessed['soup_basic'],
    'Stopwords_Removed': merged_df_preprocessed['soup_stopwords'],
    'Stemming': merged_df_preprocessed['soup_stemmed'],
    'Lemmatization_spaCy': merged_df_preprocessed['soup_lemmatized'] # Use the␣
↪spaCy lemmatized text
}
```

```
sample_movie_for_preprocessing = "Pirates of the Caribbean: At World\'s End"
num_recommendations_preprocessing_exp = 5

# Reuse the get_content_based_recommendations function.
# Note: This function find cosine_sim is calculated on TF-IDF of 'soup'.
# We are recalculating cosine_sim for each preprocessed version.

for name, processed_soup in preprocessing_experiments.items():
     print(f"\n--- Preprocessing Experiment: {name} ---")
     try:
         # Refit TF-IDF for the current preprocessed text
         # If stop words were already removed in preprocessing, don't use TfidfVectorizer stop_words
         if name in ['Stopwords_Removed', 'Stemming', 'Lemmatization_spaCy']:
              current_tfidf = TfidfVectorizer(ngram_range=(1, 2), max_features=86621) # Don't use stop_words here
         else:
              current_tfidf = TfidfVectorizer(stop_words='english', ngram_range=(1, 2), max_features=86621) # Use stop_words for others


         current_tfidf_matrix = current_tfidf.fit_transform(processed_soup)
         current_cosine_sim = linear_kernel(current_tfidf_matrix, current_tfidf_matrix)

         # Use the existing get_content_based_recommendations function
         # Need to temporarily modify merged_df to use the processed_soup for the recommendation function's internal index mapping
         original_soup_col = merged_df['soup'] # Store original soup
         merged_df['soup'] = processed_soup # Temporarily replace with processed soup

         current_recommendations = get_content_based_recommendations(
             sample_movie_for_preprocessing,
             merged_df,
             cosine_sim=current_cosine_sim,
             num_recommendations=num_recommendations_preprocessing_exp
         )

         # Restore original soup column
         merged_df['soup'] = original_soup_col

         print(f"Recommendations for '{sample_movie_for_preprocessing}' with {name}:")
         display(current_recommendations)
```

```
        print(f"Qualitative Assessment for {name}: [Observe the␣
↪recommendations above and note changes.]")

    except Exception as e:
        print(f"Error running preprocessing experiment {name}: {e}")
```

```
Generating recommendations using different preprocessed text:

--- Preprocessing Experiment: Original_soup ---
Recommendations for 'Pirates of the Caribbean: At World's End' with
Original_soup:

                                                  title  \
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
1654                               Dragonball Evolution
216                                         Life of Pi

                                                genres  \
12                          Adventure Fantasy Action
199                         Adventure Fantasy Action
17                          Adventure Action Fantasy
1654  Action Adventure Fantasy ScienceFiction Thriller
216                           Adventure Drama Action

                                              keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
1654  karate superhero revenge dragon duringcreditss…
216   ocean shipwreck hindu tiger faith zookeeper te…

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
1654                    0.3612
216                     0.0000
Qualitative Assessment for Original_soup: [Observe the recommendations above and
note changes.]

--- Preprocessing Experiment: Basic_Preprocessing ---
Recommendations for 'Pirates of the Caribbean: At World's End' with
Basic_Preprocessing:

                                                  title  \
```

```
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
216                                         Life of Pi
2592    VeggieTales: The Pirates Who Don't Do Anything

                                             genres  \
12                             Adventure Fantasy Action
199                            Adventure Fantasy Action
17                             Adventure Action Fantasy
216                              Adventure Drama Action
2592  Adventure Animation Comedy ScienceFiction Family

                                              keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
216   ocean shipwreck hindu tiger faith zookeeper te…
2592  brotherbrotherrelationship hostage vegetable c…

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
216                     0.0000
2592                    0.9358
```

Qualitative Assessment for Basic_Preprocessing: [Observe the recommendations above and note changes.]

```
--- Preprocessing Experiment: Stopwords_Removed ---
Recommendations for 'Pirates of the Caribbean: At World's End' with
Stopwords_Removed:

                                                 title  \
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
216                                         Life of Pi
3635                             90 Minutes in Heaven

                        genres  \
12    Adventure Fantasy Action
199   Adventure Fantasy Action
17    Adventure Action Fantasy
216     Adventure Drama Action
3635                     Drama

                                              keywords  \
```

```
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
216   ocean shipwreck hindu tiger faith zookeeper te…
3635                                  hospital church

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
216                     0.0000
3635                   -0.8442
```

Qualitative Assessment for Stopwords_Removed: [Observe the recommendations above and note changes.]

--- Preprocessing Experiment: Stemming ---
Recommendations for 'Pirates of the Caribbean: At World's End' with Stemming:

```
                                                title  \
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
3825                                       The Pirate
2592     VeggieTales: The Pirates Who Don't Do Anything

                                              genres  \
12                          Adventure Fantasy Action
199                         Adventure Fantasy Action
17                          Adventure Action Fantasy
3825                                   Music Romance
2592  Adventure Animation Comedy ScienceFiction Family

                                             keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
3825                                   musical pirate
2592  brotherbrotherrelationship hostage vegetable c…

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
3825                    0.9437
2592                    0.9358
```

Qualitative Assessment for Stemming: [Observe the recommendations above and note changes.]

```
--- Preprocessing Experiment: Lemmatization_spaCy ---
Recommendations for 'Pirates of the Caribbean: At World's End' with
Lemmatization_spaCy:

                                                      title  \
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
3825                                        The Pirate
2592     VeggieTales: The Pirates Who Don't Do Anything

                                                    genres  \
12                              Adventure Fantasy Action
199                             Adventure Fantasy Action
17                              Adventure Action Fantasy
3825                                       Music Romance
2592  Adventure Animation Comedy ScienceFiction Family

                                                   keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
3825                                     musical pirate
2592  brotherbrotherrelationship hostage vegetable c…

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
3825                    0.9437
2592                    0.9358

Qualitative Assessment for Lemmatization_spaCy: [Observe the recommendations
above and note changes.]
```

5.2 Feature Engineering Refinement: Incorporating Additional Features

Let's enhance our content representation by incorporating additional features from the dataset. We'll extract information from columns like `production_companies`, `production_countries`, and `spoken_languages` and add it to our 'soup' text.

[148]:
```
merged_df.columns
```

[148]:
```
Index(['budget', 'genres', 'id', 'keywords', 'original_language',
       'original_title', 'overview', 'popularity', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime', 'status',
       'title', 'vote_average', 'vote_count', 'movie_id', 'cast', 'director',
       'soup', 'overview_sentiment_score', 'release_year',
```

```
'sentiment_difference', 'cluster', 'pca_comp1', 'pca_comp2',
       'pca_comp3', 'kmeans_cluster'],
      dtype='object')
```

[149]:
```
# Incorporate Additional Features into 'soup'
import json

# Function to safely extract names from JSON-like strings
def extract_names(json_string):
    if isinstance(json_string, str):
        try:
            list_of_dicts = json.loads(json_string)
            return ' '.join([d['name'].replace(" ", "") for d in␣
↪list_of_dicts]) # Remove spaces in names for single tokens
        except (json.JSONDecodeError, TypeError):
            return ''
    return ''

# Apply the extraction function to relevant columns
merged_df['production_companies_names'] = merged_df['production_companies'].
↪apply(extract_names)
merged_df['production_countries_names'] = merged_df['production_countries'].
↪apply(extract_names)
merged_df['spoken_languages_names'] = merged_df['original_language'].
↪apply(extract_names)


# Create a new 'enhanced_soup' column by combining the original 'soup' with the␣
↪new features
# handle potential None/NaN values before combining
merged_df['enhanced_soup'] = merged_df['soup'].fillna('') + ' ' + \
                           merged_df['production_companies_names'].fillna('') +␣
↪' ' + \
                           merged_df['production_countries_names'].fillna('') +␣
↪' ' + \
                           merged_df['spoken_languages_names'].fillna('')

print("Created 'enhanced_soup' column with additional features.")
print("Sample of enhanced_soup for the first movie:")
print(merged_df['enhanced_soup'].iloc[0])



# Refit TF-IDF using the enhanced_soup
# Using the original TF-IDF parameters (stop_words, ngram_range) for now
```

```
tfidf_enhanced = TfidfVectorizer(stop_words='english', ngram_range=(1, 2),␣
↪max_features=86621)
tfidf_matrix_enhanced = tfidf_enhanced.fit_transform(merged_df['enhanced_soup'])
cosine_sim_enhanced = linear_kernel(tfidf_matrix_enhanced,␣
↪tfidf_matrix_enhanced)
```

```
Created 'enhanced_soup' column with additional features.
Sample of enhanced_soup for the first movie:
AvatarIn the 22nd century, a paraplegic Marine is dispatched to the moon Pandora
on a unique mission, but becomes torn between following orders and protecting an
alien civilization. Action Adventure Fantasy ScienceFiction cultureclash future
spacewar spacecolony society spacetravel futuristic romance space alien tribe
alienplanet cgi marine soldier battle loveaffair antiwar powerrelations
mindandsoul 3d SamWorthington ZoeSaldana SigourneyWeaver StephenLang
MichelleRodriguez GiovanniRibisi JoelDavidMoore CCHPounder WesStudi LazAlonso
DileepRao MattGerald SeanAnthonyMoran JasonWhyte ScottLawrence KellyKilgour
JamesPatrickPitt SeanPatrickMurphy PeterDillon KevinDorman KelsonHenderson
DavidVanHorn JacobTomuri MichaelBlain-Rozgay JonCurry LukeHawker WoodySchultz
PeterMensah SoniaYee JahnelCurfman IlramChoi KylaWarren LisaRoumain DebraWilson
ChrisMala TaylorKibby JodieLandau JulieLamm CullenB.Madden JosephBradyMadden
FrankieTorres AustinWilson SaraWilson TamicaWashington-Miller LucyBriant
NathanMeister GerryBlair MatthewChamberlain PaulYates WrayWilson JamesGaylyn
MelvinLenoClarkIII CarvonFutrell BrandonJelkes MicahMoch HanniyahMuhammad
ChristopherNolen ChristaOliver AprilMarieThomas BravitaA.Threatt ColinBleasdale
MikeBodnar MattClayton NicoleDionne JamieHarrison AllanHenry AnthonyIngruber
AshleyJeffery DeanKnowsley JosephMika-Hunt TerryNotary KaiPantano LoganPithyou
StuartPollock Raja GarethRuck RhianSheehan T.J.Storm JodieTaylor AliciaVela-
Bailey RichardWhiteside NikieZambo JuleneRenee 2009-12-10 IngeniousFilmPartners
TwentiethCenturyFoxFilmCorporation DuneEntertainment LightstormEntertainment
UnitedStatesofAmerica UnitedKingdom
```

[150]:
```
# sample movie to evaluate recommendations qualitatively with the enhanced␣
↪features
sample_movie_for_enhanced = 'Avatar'
num_recommendations_enhanced_exp = 5

# Use the existing get_content_based_recommendations function with the new␣
↪cosine similarity matrix
# Temporarily replace the 'soup' column with 'enhanced_soup' for the function␣
↪to work correctly
# Create a copy of merged_df to avoid modifying the original DataFrame
merged_df_copy = merged_df.copy()
merged_df_copy['soup'] = merged_df_copy['enhanced_soup']


enhanced_recommendations = get_content_based_recommendations(
    sample_movie_for_enhanced,
```

```
    merged_df_copy, # Pass the copy of the DataFrame
    cosine_sim=cosine_sim_enhanced,
    num_recommendations=num_recommendations_enhanced_exp
)

# No need to restore the original 'soup' column in merged_df since we worked on␣
 ↪a copy
print(f"\nRecommendations for '{sample_movie_for_enhanced}' using Enhanced␣
 ↪Features:")
display(enhanced_recommendations)
```

```
Recommendations for 'Avatar' using Enhanced Features:
                     title                                           genres  \
1654  Dragonball Evolution  Action Adventure Fantasy ScienceFiction Thriller
539             Titan A.E.  Animation Action ScienceFiction Family Adventure
1342  The Inhabited Island            Action Fantasy ScienceFiction Thriller
151                Beowulf                        Adventure Action Animation
3607             Apollo 18                    Horror Thriller ScienceFiction

                                               keywords  \
1654  karate superhero revenge dragon duringcreditss…
539   monster galaxy dystopia space alien animation …
1342  basedonnovel brothersisterrelationship swordfi…
151   denmark nordicmythology lie prideandvanity fol…
3607  moon nasa infection barbecue moonlanding hamme…

      overview_sentiment_score
1654                    0.3612
539                     0.3612
1342                   -0.7269
151                     0.0000
3607                   -0.4497
```

6 Fine tuning the models

6.1 Review current model performance

Briefly summarize the performance observed in the previous steps (qualitative assessment of combined model, kNN, Autoencoder, Clustering).

Reasoning: Summarize the performance of the implemented recommendation models based on the qualitative analysis and the limitations encountered during evaluation, as requested in the instructions.

##Recommendation Model Performance Summary

6.1.1 Content-Based Model (TF-IDF + Cosine Similarity):

- Based on TF-IDF of combined text features (overview, genres, keywords, cast, director).
- Provides recommendations for movies with similar textual content.
- Qualitative analysis (demonstrated via `get_content_based_recommendations` and as part of the combined model) showed that this approach generally yields intuitively relevant recommendations, especially when content features are descriptive.
- Performance is highly dependent on the quality and richness of the text data ('soup').

6.1.2 Sentiment-Based Model:

- Recommends movies solely based on the similarity of their overview sentiment scores.
- Qualitative analysis (demonstrated via `recommend_by_sentiment`) indicated that this method alone often provides recommendations with very similar sentiment scores but not necessarily similar content or overall relevance, as movies with drastically different content can have similar sentiment in their overviews.
- This approach is too simplistic for comprehensive movie recommendations.

6.1.3 Combined Hybrid Model (Content + Sentiment):

- Combines content similarity (TF-IDF + Cosine Similarity) and sentiment similarity with adjustable weights.
- Qualitative analysis (demonstrated via `get_combined_recommendations_weighted` and `generate_recommendations_with_reasons`) showed that this model, particularly with a balanced or content-heavy weighting, produced more relevant recommendations than the sentiment-only approach.
- The reasons provided by `generate_recommendations_with_reasons` helped to understand the contribution of content (shared genres, keywords) and sentiment to the recommendations.
- The choice of weights significantly impacts the recommendations, highlighting the need for potential tuning based on desired recommendation characteristics.

6.1.4 k-Nearest Neighbors (kNN) Model:

- Uses kNN on the TF-IDF matrix to find nearest neighbors based on content features.
- Supports searching by movie title (partial/exact) or keyword/plot.
- Qualitative analysis (demonstrated via `generate_recommendations_knn` and `generate_knn_recommendations_with_reasons`) showed that it provides content-similar recommendations, similar to the basic TF-IDF cosine similarity, as expected since kNN with cosine distance on normalized vectors is equivalent to finding nearest neighbors by cosine similarity.
- The basic spell-check feature is a useful addition for handling user input variations.

6.1.5 Autoencoder-based Model:

- Learns dense, lower-dimensional latent feature representations of movies from the scaled TF-IDF matrix using a neural network.
- Recommends movies based on cosine similarity in this learned latent space.
- Qualitative analysis (demonstrated via `get_autoencoder_recommendations`) showed that this model can capture complex patterns in the data, potentially identifying subtle similarities not obvious from raw TF-IDF.

- The learned features represent a compressed semantic space, which can lead to interesting recommendations.
- Evaluation of the Autoencoder's performance was limited to reconstruction loss; the quality of the learned representation for recommendations needs further assessment.

6.1.6 Item-Based Clustering Model:

- Clusters movies based on the Autoencoder-learned latent features using KMeans.
- Recommends other movies within the same cluster as an input movie or a movie identified by a query.
- Qualitative analysis (demonstrated via `get_clustering_recommendations` and `query_clustering_recommendations`) showed that clusters group movies with seemingly related content or themes.
- Recommendations from this model are diverse within the cluster but might not be the 'most similar' in a strict sense; they represent a group of related items.
- The effectiveness depends heavily on the quality of the latent features and the chosen number of clusters (identified using the Elbow method).

6.1.7 Limitations in Quantitative Evaluation:

- A significant limitation across all models is the lack of explicit user ID and rating data.
- Standard recommendation evaluation metrics (RMSE, Precision@K, Recall@K, etc.) require comparing predicted ratings or recommended items against actual user interactions.
- The attempts to calculate RMSE, Precision, and Recall using `vote_average` and `vote_count` were demonstrated but noted as non-standard evaluations of the recommendation algorithms themselves; they evaluate the quality of the aggregated movie metrics or simple baseline strategies, not the personalized or content-based recommendation capabilities.
- A true quantitative evaluation would necessitate a dataset with user-item interaction history.

6.2 Hyperparameter tuning

Identify key hyperparameters for the most promising models (e.g., TF-IDF vectorizer parameters, Autoencoder architecture/training parameters, kNN `n_neighbors`, weights in the hybrid model, number of clusters in KMeans) and explore tuning them.

Reasoning: Select key hyperparameters for the TF-IDF Vectorizer and experiment with different values to see their impact on the tfidf_matrix. Rerun the content-based recommendation function with the updated TF-IDF matrix and qualitatively assess the changes in recommendations for sample movies.

6.3 Hyperparameter tuning: TF-IDF Vectorizer

We will experiment with different hyperparameters for the `TfidfVectorizer` to see how they affect the content representation and, consequently, the recommendations. Key parameters to tune include:

- `max_features`: The maximum number of features (vocabulary size). Limiting this can help reduce noise and focus on the most important terms.
- `ngram_range`: The range of n-grams to include (e.g., (1, 1) for unigrams, (1, 2) for unigrams and bigrams). Including n-grams can capture phrases and multi-word concepts.

- `min_df`: When building the vocabulary, ignore terms that have a document frequency strictly lower than the given threshold. This helps remove rare terms.
- `max_df`: When building the vocabulary, ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words). This helps remove very common terms.

[151]:
```
# Experiment with TF-IDF Vectorizer hyperparameters

print("## Experimenting with TF-IDF Vectorizer Hyperparameters\n")

# Original TF-IDF (already fitted)
tfidf = TfidfVectorizer(stop_words='english', ngram_range=(1, 2),␣
↪max_features=86621)
tfidf_matrix = tfidf.fit_transform(merged_df['soup'].fillna(''))

# Experiment with different TF-IDF hyperparameters
tfidf_experiments = {
    'Original': {'max_features': None, 'ngram_range': (1, 1), 'min_df': 1,␣
↪'max_df': 1.0},
    'Max_Features_5000': {'max_features': 5000, 'ngram_range': (1, 1), 'min_df':
↪ 1, 'max_df': 1.0},
    'Ngram_Range_1_2': {'max_features': None, 'ngram_range': (1, 2), 'min_df':␣
↪1, 'max_df': 1.0},
    'Min_DF_5': {'max_features': None, 'ngram_range': (1, 1), 'min_df': 5,␣
↪'max_df': 1.0},
    'Max_DF_0_9': {'max_features': None, 'ngram_range': (1, 1), 'min_df': 1,␣
↪'max_df': 0.9}
}

sample_movie_for_tfidf = "Pirates of the Caribbean: At World\'s End"
num_recommendations_tfidf_exp = 5

original_tfidf = TfidfVectorizer(stop_words='english', ngram_range=(1, 2),␣
↪max_features=86621)
original_tfidf_matrix = original_tfidf.fit_transform(merged_df['soup'].
↪fillna(''))
original_cosine_sim = linear_kernel(original_tfidf_matrix,␣
↪original_tfidf_matrix)

# Use the existing get_content_based_recommendations function

print(f"Generating recommendations for '{sample_movie_for_tfidf}' using␣
↪different TF-IDF settings:")
optimized_tfidf=None # save to use later for model deployment
for name, params in tfidf_experiments.items():
    print(f"\n--- TF-IDF Experiment: {name} ---")
    try:
```

```
        # Create and fit a new TF-IDF vectorizer with the experimental␣
↪parameters
        optimized_tfidf = TfidfVectorizer(stop_words='english', **params)
        current_tfidf_matrix = optimized_tfidf.fit_transform(merged_df['soup'].
↪fillna(''))
        current_cosine_sim = linear_kernel(current_tfidf_matrix,␣
↪current_tfidf_matrix)

        # Generate recommendations using the new cosine similarity matrix
        current_recommendations = get_content_based_recommendations(
            sample_movie_for_tfidf,
            merged_df,
            cosine_sim=current_cosine_sim,
            num_recommendations=num_recommendations_tfidf_exp
        )

        print(f"Recommendations for '{sample_movie_for_tfidf}' with {name}␣
↪TF-IDF:")
        display(current_recommendations)

        # Qualitative assessment (manual observation of displayed␣
↪recommendations)
        print(f"Qualitative Assessment for {name}: [Observe the recommendations␣
↪above and note changes compared to Original]")

    except Exception as e:
        print(f"Error running TF-IDF experiment {name}: {e}")
```

```
## Experimenting with TF-IDF Vectorizer Hyperparameters

Generating recommendations for 'Pirates of the Caribbean: At World's End' using
different TF-IDF settings:

--- TF-IDF Experiment: Original ---
Recommendations for 'Pirates of the Caribbean: At World's End' with Original TF-
IDF:

                                                  title  \
12            Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17           Pirates of the Caribbean: On Stranger Tides
2592     VeggieTales: The Pirates Who Don't Do Anything
848       The Pirates! In an Adventure with Scientists!

                                                genres  \
12                              Adventure Fantasy Action
199                             Adventure Fantasy Action
```

```
17                              Adventure Action Fantasy
2592  Adventure Animation Comedy ScienceFiction Family
848                    Animation Adventure Family Comedy

                                               keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
2592  brotherbrotherrelationship hostage vegetable c…
848   rivalry stopmotion pirate aftercreditsstinger …

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
2592                    0.9358
848                     0.9605
```

Qualitative Assessment for Original: [Observe the recommendations above and note changes compared to Original]

```
--- TF-IDF Experiment: Max_Features_5000 ---
Recommendations for 'Pirates of the Caribbean: At World's End' with
Max_Features_5000 TF-IDF:

                                                  title  \
12            Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17           Pirates of the Caribbean: On Stranger Tides
2592     VeggieTales: The Pirates Who Don't Do Anything
216                                          Life of Pi

                                                genres  \
12                              Adventure Fantasy Action
199                             Adventure Fantasy Action
17                              Adventure Action Fantasy
2592  Adventure Animation Comedy ScienceFiction Family
216                               Adventure Drama Action

                                               keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
2592  brotherbrotherrelationship hostage vegetable c…
216   ocean shipwreck hindu tiger faith zookeeper te…

      overview_sentiment_score
12                     -0.8074
199                     0.7003
```

```
17                   -0.2411
2592                  0.9358
216                   0.0000

Qualitative Assessment for Max_Features_5000: [Observe the recommendations above
and note changes compared to Original]

--- TF-IDF Experiment: Ngram_Range_1_2 ---
Recommendations for 'Pirates of the Caribbean: At World's End' with
Ngram_Range_1_2 TF-IDF:

                                                  title  \
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
216                                         Life of Pi
2592     VeggieTales: The Pirates Who Don't Do Anything

                                                genres  \
12                           Adventure Fantasy Action
199                          Adventure Fantasy Action
17                           Adventure Action Fantasy
216                            Adventure Drama Action
2592  Adventure Animation Comedy ScienceFiction Family

                                               keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
216   ocean shipwreck hindu tiger faith zookeeper te…
2592  brotherbrotherrelationship hostage vegetable c…

      overview_sentiment_score
12                   -0.8074
199                   0.7003
17                   -0.2411
216                   0.0000
2592                  0.9358

Qualitative Assessment for Ngram_Range_1_2: [Observe the recommendations above
and note changes compared to Original]

--- TF-IDF Experiment: Min_DF_5 ---
Recommendations for 'Pirates of the Caribbean: At World's End' with Min_DF_5 TF-
IDF:

                                                  title  \
12          Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17         Pirates of the Caribbean: On Stranger Tides
```

```
2592    VeggieTales: The Pirates Who Don't Do Anything
216                                         Life of Pi

                                            genres  \
12                        Adventure Fantasy Action
199                       Adventure Fantasy Action
17                        Adventure Action Fantasy
2592  Adventure Animation Comedy ScienceFiction Family
216                         Adventure Drama Action

                                            keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
2592  brotherbrotherrelationship hostage vegetable c…
216   ocean shipwreck hindu tiger faith zookeeper te…

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
2592                    0.9358
216                     0.0000
```

```
Qualitative Assessment for Min_DF_5: [Observe the recommendations above and note
changes compared to Original]

--- TF-IDF Experiment: Max_DF_0_9 ---
Recommendations for 'Pirates of the Caribbean: At World's End' with Max_DF_0_9
TF-IDF:

                                               title  \
12           Pirates of the Caribbean: Dead Man's Chest
199   Pirates of the Caribbean: The Curse of the Bla…
17          Pirates of the Caribbean: On Stranger Tides
2592    VeggieTales: The Pirates Who Don't Do Anything
848       The Pirates! In an Adventure with Scientists!

                                            genres  \
12                        Adventure Fantasy Action
199                       Adventure Fantasy Action
17                        Adventure Action Fantasy
2592  Adventure Animation Comedy ScienceFiction Family
848                  Animation Adventure Family Comedy

                                            keywords  \
12    witch fortuneteller bondage exoticisland monst…
199   exoticisland blacksmith eastindiatradingcompan…
17    sea captain mutiny sword primeminister sailing…
```

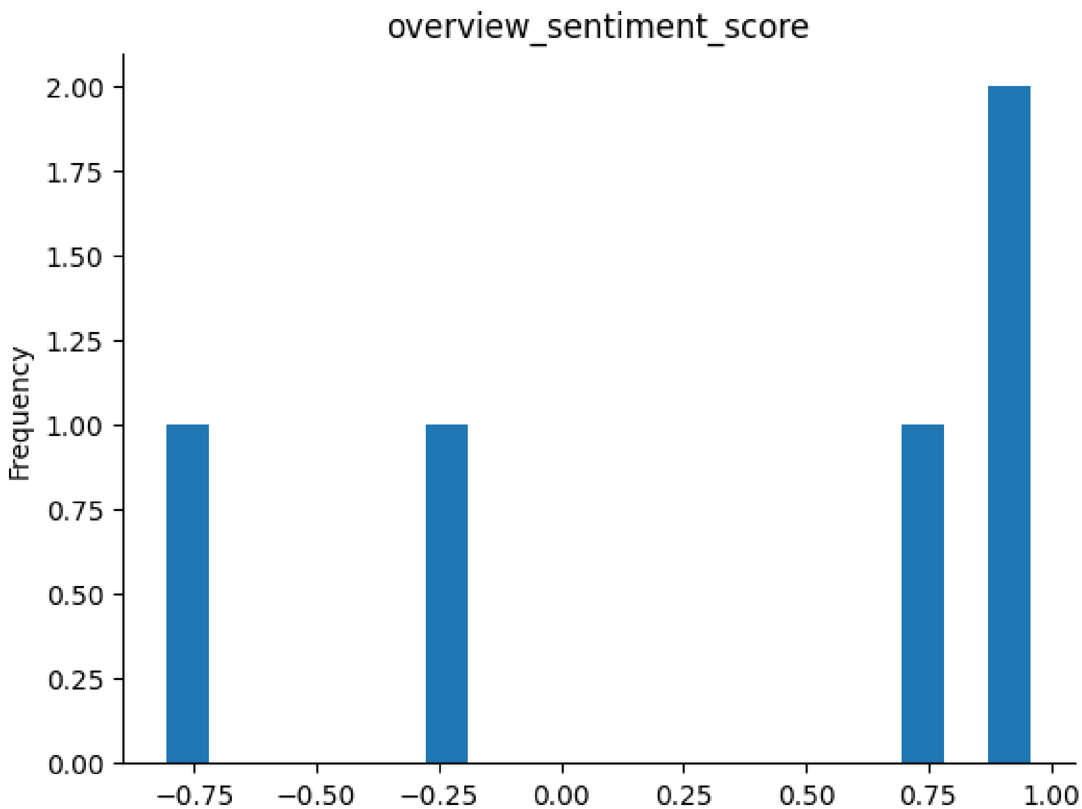
```
2592  brotherbrotherrelationship hostage vegetable c…
848   rivalry stopmotion pirate aftercreditsstinger …

      overview_sentiment_score
12                     -0.8074
199                     0.7003
17                     -0.2411
2592                    0.9358
848                     0.9605
```

Qualitative Assessment for Max_DF_0_9: [Observe the recommendations above and note changes compared to Original]

[152]:
```python
# @title genres vs keywords

from matplotlib import pyplot as plt
import seaborn as sns
import pandas as pd
plt.subplots(figsize=(8, 8))
df_2dhist = pd.DataFrame({
    x_label: grp['keywords'].value_counts()
    for x_label, grp in current_recommendations.groupby('genres')
})
sns.heatmap(df_2dhist, cmap='viridis')
plt.xlabel('genres')
_ = plt.ylabel('keywords')
```

[153]:
```python
# @title overview_sentiment_score

from matplotlib import pyplot as plt
```

```
current_recommendations['overview_sentiment_score'].plot(kind='hist', bins=20,
    title='overview_sentiment_score')
plt.gca().spines[['top', 'right',]].set_visible(False)
```



6.4 Hyperparameter tuning: Autoencoder Model

Now, let's tune the hyperparameters of the Autoencoder model. Key parameters to consider include:

- `encoding_dim`: The size of the latent representation layer. A smaller dimension forces the model to learn a more compressed representation, while a larger dimension allows for more capacity but might also capture noise.
- `epochs`: The number of times the training data is passed through the entire network. More epochs can lead to better learning but also risk overfitting.
- `batch_size`: The number of samples per gradient update during training. Smaller batch sizes can add noise to the gradient but might lead to better generalization; larger batch sizes provide a more stable gradient but require more memory.

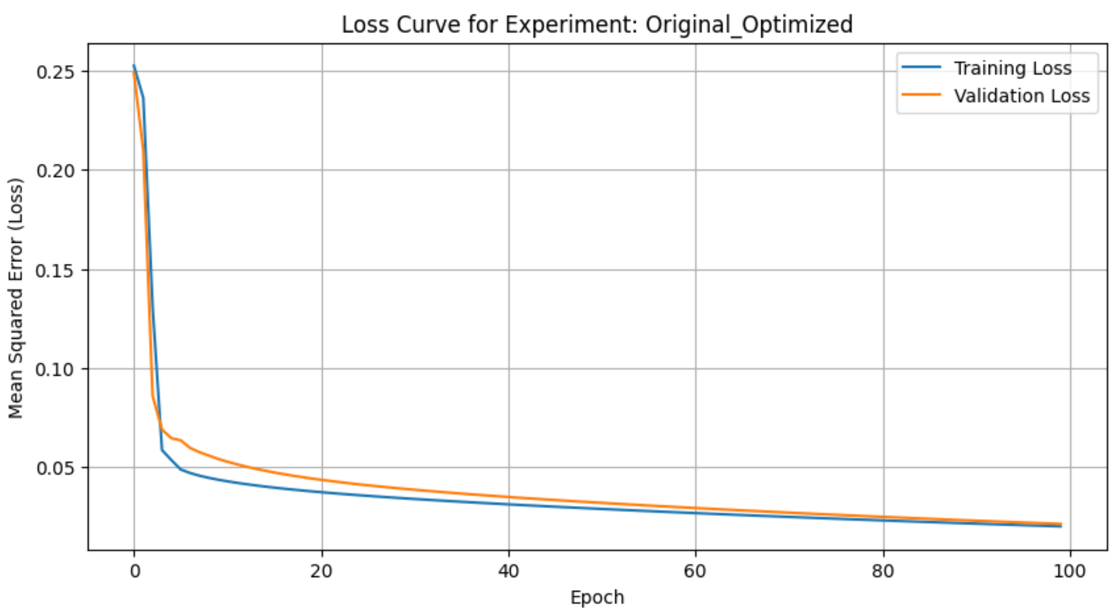
[154]:
```
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping # Import EarlyStopping
```

```
from tensorflow.keras.regularizers import l2 # Import l2 regularizer
from IPython.display import display
import pandas as pd
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Dummy get_autoencoder_recommendations for demonstration if not available
def get_autoencoder_recommendations(movie_title, df, latent_cosine_sim,␣
↪num_recommendations=5):
    if movie_title not in df['title'].values:
        return f"Movie '{movie_title}' not found in the dataset."
    idx = df[df['title'] == movie_title].index[0]
    sim_scores = list(enumerate(latent_cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:num_recommendations+1]
    movie_indices = [i[0] for i in sim_scores]
    return df['title'].iloc[movie_indices]


print("\n## Experimenting with Autoencoder Hyperparameters (with Early Stopping␣
↪and Weight Decay)\n")

# Autoencoder Model Parameter Experiments
# Increased epochs to 100 to allow EarlyStopping to work effectively
autoencoder_experiments = {
    'Original_Optimized': {'encoding_dim': 128, 'epochs': 100, 'batch_size':␣
↪256},
    'Small_Latent_Space': {'encoding_dim': 64, 'epochs': 100, 'batch_size':␣
↪256},
    'Large_Latent_Space': {'encoding_dim': 256, 'epochs': 100, 'batch_size':␣
↪256},
}

# A movie to evaluate recommendations qualitatively
sample_movie_for_autoencoder = 'The Matrix'
num_recommendations_autoencoder_exp = 5

# Define the EarlyStopping callback
# It will monitor validation loss and stop if there's no improvement after 5␣
↪epochs.
# It will also restore the best weights found during training.
early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)

print(f"Generating recommendations for '{sample_movie_for_autoencoder}' using␣
↪different Autoencoder settings:")
```

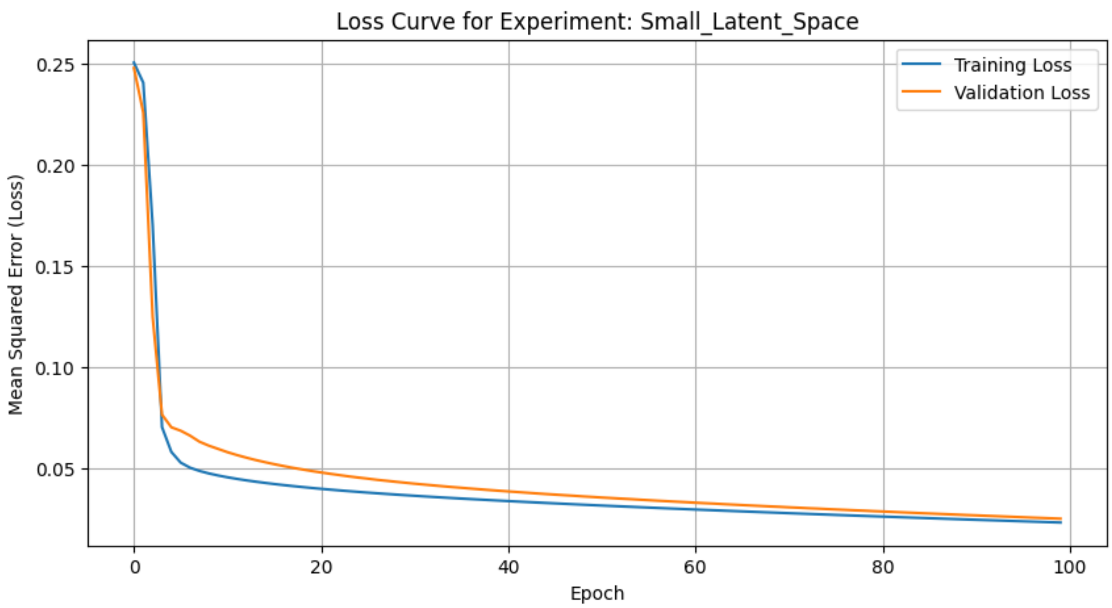
```
current_encoder = None # To hold the final or best encoder for saving later

for name, params in autoencoder_experiments.items():
    print(f"\n--- Autoencoder Experiment: {name} ---")
    try:
        # Build the Autoencoder Model with L2 Regularization (Weight Decay)
        input_layer = Input(shape=(tfidf_scaled.shape[1],))
        encoder_layer = Dense(params['encoding_dim'], activation='relu',␣
 ↪kernel_regularizer=l2(1e-5))(input_layer)
        decoder_layer = Dense(tfidf_scaled.shape[1], activation='sigmoid',␣
 ↪kernel_regularizer=l2(1e-5))(encoder_layer)
        optimized_autoencoder = Model(inputs=input_layer, outputs=decoder_layer)

        optimized_autoencoder.compile(optimizer='adam', loss='mse')

        # Train the Autoencoder with the EarlyStopping callback
        print(f"Training Autoencoder for {name} with up to {params['epochs']}␣
 ↪epochs...")
        history = optimized_autoencoder.fit(tfidf_scaled, tfidf_scaled,
                                            epochs=params['epochs'],
                                            batch_size=params['batch_size'],
                                            shuffle=True,
                                            validation_split=0.1, # Use 10% of␣
 ↪data for validation
                                            callbacks=[early_stopping], #␣
 ↪callback here
                                            verbose=0)
        print(f"Training Complete. Stopped at epoch: {len(history.
 ↪history['loss'])}")

        # --- PLOT THE LOSS CURVE (WITH VALIDATION LOSS) ---
        plt.figure(figsize=(10, 5))
        plt.plot(history.history['loss'], label='Training Loss')
        plt.plot(history.history['val_loss'], label='Validation Loss')
        plt.title(f'Loss Curve for Experiment: {name}')
        plt.xlabel('Epoch')
        plt.ylabel('Mean Squared Error (Loss)')
        plt.legend()
        plt.grid(True)
        plt.show()
        # --- END OF PLOTTING ---

        # Get the Encoder model and latent features
        current_encoder = Model(inputs=input_layer, outputs=encoder_layer)
        current_latent_features = current_encoder.predict(tfidf_scaled,␣
 ↪verbose=0)
```

```python
        # Calculate cosine similarity matrix on the latent features
        current_latent_cosine_sim = cosine_similarity(current_latent_features)

        # Generate recommendations
        current_autoencoder_recommendations = get_autoencoder_recommendations(
            sample_movie_for_autoencoder,
            merged_df,
            latent_cosine_sim=current_latent_cosine_sim,
            num_recommendations=num_recommendations_autoencoder_exp
        )

        print(f"Recommendations for '{sample_movie_for_autoencoder}' with␣
↪{name} Autoencoder:")
        display(current_autoencoder_recommendations)

    except Exception as e:
        print(f"Error running Autoencoder experiment {name}: {e}")
```

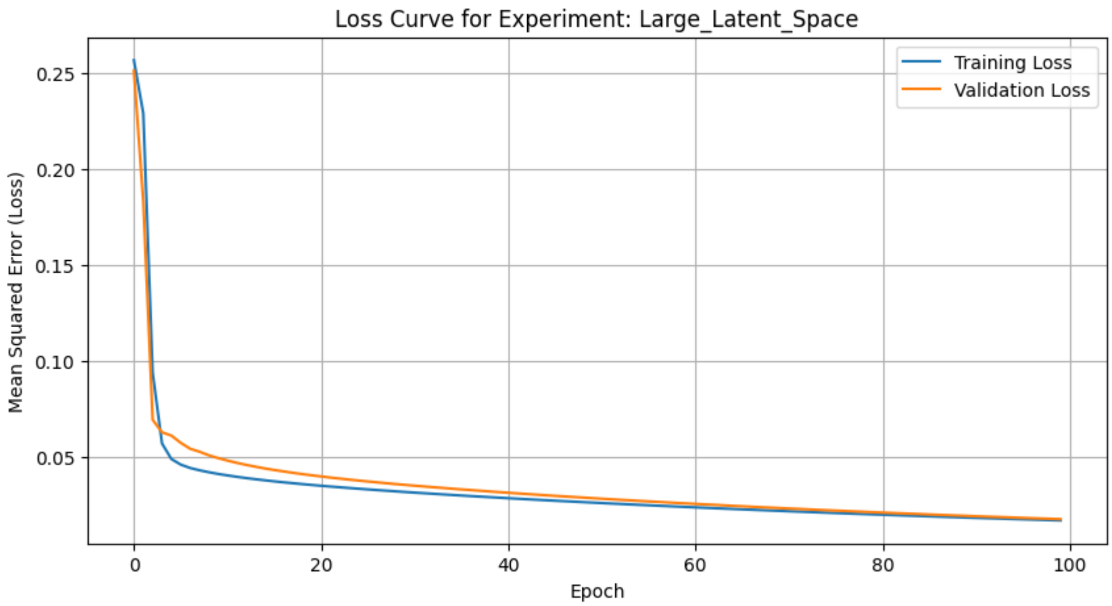
```
## Experimenting with Autoencoder Hyperparameters (with Early Stopping and
Weight Decay)

Generating recommendations for 'The Matrix' using different Autoencoder
settings:

--- Autoencoder Experiment: Original_Optimized ---
Training Autoencoder for Original_Optimized with up to 100 epochs…
Training Complete. Stopped at epoch: 100
```



```
Recommendations for 'The Matrix' with Original_Optimized Autoencoder:

468                                      Swordfish
745       Miss Congeniality 2: Armed and Fabulous
1427                                     Abduction
3951                                     Fish Tank
7                           Avengers: Age of Ultron
Name: title, dtype: object

--- Autoencoder Experiment: Small_Latent_Space ---
Training Autoencoder for Small_Latent_Space with up to 100 epochs…
Training Complete. Stopped at epoch: 100
```




```
Recommendations for 'The Matrix' with Small_Latent_Space Autoencoder:

309          Son of the Mask
728                  Hellboy
911            22 Jump Street
934           Curious George
1865       Million Dollar Arm
Name: title, dtype: object

--- Autoencoder Experiment: Large_Latent_Space ---
Training Autoencoder for Large_Latent_Space with up to 100 epochs…
Training Complete. Stopped at epoch: 100
```




```
Recommendations for 'The Matrix' with Large_Latent_Space Autoencoder:
2745            The Butterfly Effect
0                             Avatar
493                 A Beautiful Mind
614                     Despicable Me
669      Sin City: A Dame to Kill For
Name: title, dtype: object
```

6.5 Hyperparameter tuning: kNN and Clustering Models

Let's tune the hyperparameters for the kNN and Clustering models.

6.5.1 k-Nearest Neighbors (kNN) Tuning

For kNN, the main hyperparameter is `n_neighbors`, which determines how many nearest neighbors are considered for recommendations. We will experiment with different values and qualitatively assess the results.

[155]:
```
# Experiment with kNN hyperparameters

print("\n## Experimenting with kNN Hyperparameters\n")

# kNN Hyperparameter Experiments
knn_experiments = {
    'Original_k10': {'n_neighbors': 10}, # Original setting
    'k_5': {'n_neighbors': 5},        # Fewer neighbors
    'k_20': {'n_neighbors': 20},      # More neighbors
    'k_50': {'n_neighbors': 50}       # Even more neighbors
```

```
}

# Choose a sample query (can be a movie title or keyword)
sample_query_knn = 'science fiction action'
num_recommendations_knn_exp = 10 # Number of recommendations to display

print(f"Generating kNN recommendations for query '{sample_query_knn}' using␣
↪different 'n_neighbors' settings:")

for name, params in knn_experiments.items():
    print(f"\n--- kNN Experiment: {name} ---")
    try:
        # Generate recommendations using the current n_neighbors
        current_knn_recommendations = generate_recommendations_knn(
            sample_query_knn,
            merged_df,
            tfidf_matrix, # Use the original TF-IDF matrix
            num_recommendations=params['n_neighbors'] # Set num_recommendations␣
↪to n_neighbors for this test
        )

        print(f"Recommendations for '{sample_query_knn}' with {name} kNN␣
↪({params['n_neighbors']} neighbors):")
        display(current_knn_recommendations)

        # Qualitative assessment (manual observation of displayed␣
↪recommendations)
        print(f"Qualitative Assessment for {name}: [Observe the recommendations␣
↪above and note changes compared to other k values]")

    except Exception as e:
        print(f"Error running kNN experiment {name}: {e}")
```

```
## Experimenting with kNN Hyperparameters

Generating kNN recommendations for query 'science fiction action' using
different 'n_neighbors' settings:

--- kNN Experiment: Original_k10 ---
No title matches found for 'science fiction action'. Treating as a
keyword/content search.
Recommendations for 'science fiction action' with Original_k10 kNN (10
neighbors):

               Recommended Movie                                      Reason  \
0                   Martian Child  Based on content similarity (TF-IDF + KNN)
```

```
1                   Flatliners  Based on content similarity (TF-IDF + KNN)
2                          Her  Based on content similarity (TF-IDF + KNN)
3                Mars Attacks!  Based on content similarity (TF-IDF + KNN)
4  The Beast from 20,000 Fathoms  Based on content similarity (TF-IDF + KNN)
5                      Contact  Based on content similarity (TF-IDF + KNN)
6            American Splendor  Based on content similarity (TF-IDF + KNN)
7                   Red Planet  Based on content similarity (TF-IDF + KNN)
8            The Lawnmower Man  Based on content similarity (TF-IDF + KNN)
9   My Big Fat Independent Movie  Based on content similarity (TF-IDF + KNN)

   Confidence Score (Cosine Similarity)  \
0                              0.230690
1                              0.218197
2                              0.176388
3                              0.173896
4                              0.151865
5                              0.124949
6                              0.082857
7                              0.080395
8                              0.069710
9                              0.067624

                                          Overview  \
0  A recently-widowed, science fiction writer con…
1  Five medical students want to find out if ther…
2  In the not so distant future, Theodore, a lone…
3  'We come in peace' is not what those green men…
4  The Beast from 20,000 Fathoms is a 1953 scienc…
5  Contact is a science fiction film about an enc…
6  An original mix of fiction and reality illumin…
7  Astronauts search for solutions to save a dyin…
8  A simple man is turned into a genius through t…
9  This film is a spoof along the lines of "Scary…

                                Genres  \
0                                Drama
1  Drama Horror ScienceFiction Thriller
2         Romance ScienceFiction Drama
3        Comedy Fantasy ScienceFiction
4      Adventure Horror ScienceFiction
5         Drama ScienceFiction Mystery
6                         Comedy Drama
7       Thriller Action ScienceFiction
8       Horror Thriller ScienceFiction
9                               Comedy

                                          Keywords
0  underdog adoption education adoptivefather chi…
```

```
1  lifeanddeath afterlife swing memory medicalstu…
2  artificialintelligence computer love lonelines…
3  savingtheworld totaldestruction whitehouse mar…
4  monster atomicbomb lighthouse arctic rampage s…
5  basedonnovel nasa newmexico extraterrestrialte…
6  biography independentfilm v.a.hospital junksal…
7          mars future astronaut science catastrophe
8  dream chimp manipulation botanist virtualreali…
9                                   independentfilm
```

```
Qualitative Assessment for Original_k10: [Observe the recommendations above and
note changes compared to other k values]

--- kNN Experiment: k_5 ---
No title matches found for 'science fiction action'. Treating as a
keyword/content search.
Recommendations for 'science fiction action' with k_5 kNN (5 neighbors):

              Recommended Movie                                      Reason  \
0                 Martian Child  Based on content similarity (TF-IDF + KNN)
1                    Flatliners  Based on content similarity (TF-IDF + KNN)
2                           Her  Based on content similarity (TF-IDF + KNN)
3                 Mars Attacks!  Based on content similarity (TF-IDF + KNN)
4  The Beast from 20,000 Fathoms  Based on content similarity (TF-IDF + KNN)

   Confidence Score (Cosine Similarity)  \
0                              0.230690
1                              0.218197
2                              0.176388
3                              0.173896
4                              0.151865

                                            Overview  \
0  A recently-widowed, science fiction writer con…
1  Five medical students want to find out if ther…
2  In the not so distant future, Theodore, a lone…
3  'We come in peace' is not what those green men…
4  The Beast from 20,000 Fathoms is a 1953 scienc…

                                 Genres  \
0                                 Drama
1  Drama Horror ScienceFiction Thriller
2          Romance ScienceFiction Drama
3         Comedy Fantasy ScienceFiction
4       Adventure Horror ScienceFiction

                                            Keywords
0  underdog adoption education adoptivefather chi…
1  lifeanddeath afterlife swing memory medicalstu…
```

```
2  artificialintelligence computer love lonelines…
3  savingtheworld totaldestruction whitehouse mar…
4  monster atomicbomb lighthouse arctic rampage s…
```

Qualitative Assessment for k_5: [Observe the recommendations above and note changes compared to other k values]

```
--- kNN Experiment: k_20 ---
No title matches found for 'science fiction action'. Treating as a
keyword/content search.
Recommendations for 'science fiction action' with k_20 kNN (20 neighbors):

                Recommended Movie                                      Reason  \
0                    Martian Child  Based on content similarity (TF-IDF + KNN)
1                       Flatliners  Based on content similarity (TF-IDF + KNN)
2                              Her  Based on content similarity (TF-IDF + KNN)
3                    Mars Attacks!  Based on content similarity (TF-IDF + KNN)
4   The Beast from 20,000 Fathoms  Based on content similarity (TF-IDF + KNN)
5                          Contact  Based on content similarity (TF-IDF + KNN)
6                American Splendor  Based on content similarity (TF-IDF + KNN)
7                       Red Planet  Based on content similarity (TF-IDF + KNN)
8                The Lawnmower Man  Based on content similarity (TF-IDF + KNN)
9    My Big Fat Independent Movie  Based on content similarity (TF-IDF + KNN)
10                          Capote  Based on content similarity (TF-IDF + KNN)
11          Ayurveda: Art of Being  Based on content similarity (TF-IDF + KNN)
12                         Gattaca  Based on content similarity (TF-IDF + KNN)
13           An Inconvenient Truth  Based on content similarity (TF-IDF + KNN)
14                     The Eclipse  Based on content similarity (TF-IDF + KNN)
15                    Heartbreakers  Based on content similarity (TF-IDF + KNN)
16              The Man from Earth  Based on content similarity (TF-IDF + KNN)
17              Terminator Genisys  Based on content similarity (TF-IDF + KNN)
18                 Beneath Hill 60  Based on content similarity (TF-IDF + KNN)
19             Edward Scissorhands  Based on content similarity (TF-IDF + KNN)

    Confidence Score (Cosine Similarity)  \
0                               0.230690
1                               0.218197
2                               0.176388
3                               0.173896
4                               0.151865
5                               0.124949
6                               0.082857
7                               0.080395
8                               0.069710
9                               0.067624
10                              0.064663
11                              0.063822
12                              0.063551
13                              0.062132
```

```
14                                  0.061140
15                                  0.059682
16                                  0.058882
17                                  0.058808
18                                  0.058175
19                                  0.057011

                                              Overview  \
0   A recently-widowed, science fiction writer con…
1   Five medical students want to find out if ther…
2   In the not so distant future, Theodore, a lone…
3   'We come in peace' is not what those green men…
4   The Beast from 20,000 Fathoms is a 1953 scienc…
5   Contact is a science fiction film about an enc…
6   An original mix of fiction and reality illumin…
7   Astronauts search for solutions to save a dyin…
8   A simple man is turned into a genius through t…
9   This film is a spoof along the lines of "Scary…
10  A biopic of the writer, Truman Capote and his …
11  Ayurveda is a science of life and a healing ar…
12  Science fiction drama about a future society i…
13  A documentary on Al Gore's campaign to make th…
14  Michael Farr (Hinds) is a widower living in a …
15  Max and Page are a brilliant mother/daughter c…
16  An impromptu goodbye party for Professor John …
17  The year is 2029. John Connor, leader of the r…
18  The true story of Australia's cat-and-mouse un…
19  A small suburban town receives a visit from a …

                                       Genres  \
0                                       Drama
1       Drama Horror ScienceFiction Thriller
2               Romance ScienceFiction Drama
3              Comedy Fantasy ScienceFiction
4             Adventure Horror ScienceFiction
5               Drama ScienceFiction Mystery
6                               Comedy Drama
7              Thriller Action ScienceFiction
8              Horror Thriller ScienceFiction
9                                      Comedy
10                                Crime Drama
11                                Documentary
12   Thriller ScienceFiction Mystery Romance
13                                Documentary
14                       Drama Horror Romance
15                       Crime Comedy Romance
16                       ScienceFiction Drama
17  ScienceFiction Action Thriller Adventure
```

```
18                          Drama History War
19                      Fantasy Drama Romance

                                                Keywords
0   underdog adoption education adoptivefather chi…
1   lifeanddeath afterlife swing memory medicalstu…
2   artificialintelligence computer love lonelines…
3   savingtheworld totaldestruction whitehouse mar…
4   monster atomicbomb lighthouse arctic rampage s…
5   basedonnovel nasa newmexico extraterrestrialte…
6   biography independentfilm v.a.hospital junksal…
7           mars future astronaut science catastrophe
8   dream chimp manipulation botanist virtualreali…
9                                   independentfilm
10  gay self-fulfillingprophecy basedonnovel journ…
11                        philosophy india healing
12  paraplegic suicideattempt cheating dna spacema…
13  climatechange greenhouseeffect climate earth g…
14
15                                     consandscams
16  philosophy secret birthday professor psycholog…
17  savingtheworld artificialintelligence cyborg k…
18
19  underdog loveatfirstsight hairdresser smalltow…
```

```
Qualitative Assessment for k_20: [Observe the recommendations above and note
changes compared to other k values]

--- kNN Experiment: k_50 ---
No title matches found for 'science fiction action'. Treating as a
keyword/content search.
Recommendations for 'science fiction action' with k_50 kNN (50 neighbors):
```

```
                            Recommended Movie  \
0                                Martian Child
1                                   Flatliners
2                                          Her
3                                Mars Attacks!
4                 The Beast from 20,000 Fathoms
5                                      Contact
6                             American Splendor
7                                   Red Planet
8                            The Lawnmower Man
9                  My Big Fat Independent Movie
10                                      Capote
11                         Ayurveda: Art of Being
12                                     Gattaca
13                         An Inconvenient Truth
14                                 The Eclipse
```

```
15                              Heartbreakers
16                         The Man from Earth
17                         Terminator Genisys
18                            Beneath Hill 60
19                        Edward Scissorhands
20                                      Three
21                                       Doom
22                  Girl with a Pearl Earring
23                            Banshee Chapter
24                    The Caveman's Valentine
25                  The Sorcerer's Apprentice
26  Sea Rex 3D: Journey to a Prehistoric World
27                              Teen Wolf Too
28                                The Reaping
29                               The Exorcist
30                                      K-PAX
31                 Jack Brooks: Monster Slayer
32                The Lost Skeleton of Cadavra
33                           Forbidden Kingdom
34              Beneath the Planet of the Apes
35                                  Red Lights
36                                  The Martian
37            Cloudy with a Chance of Meatballs
38                          The Andromeda Strain
39                          The Helix… Loaded 
40                              Silver Medalist
41                     The Transporter Refueled
42                              Last Action Hero
43                      Amidst the Devil's Wings
44                                  The Specials
45     Miss Congeniality 2: Armed and Fabulous
46                                Small Soldiers
47                                     MacGruber
48                                       Shooter
49                                   Khiladi 786

                                      Reason  \
0   Based on content similarity (TF-IDF + KNN)
1   Based on content similarity (TF-IDF + KNN)
2   Based on content similarity (TF-IDF + KNN)
3   Based on content similarity (TF-IDF + KNN)
4   Based on content similarity (TF-IDF + KNN)
5   Based on content similarity (TF-IDF + KNN)
6   Based on content similarity (TF-IDF + KNN)
7   Based on content similarity (TF-IDF + KNN)
8   Based on content similarity (TF-IDF + KNN)
9   Based on content similarity (TF-IDF + KNN)
10  Based on content similarity (TF-IDF + KNN)
```

```
11  Based on content similarity (TF-IDF + KNN)
12  Based on content similarity (TF-IDF + KNN)
13  Based on content similarity (TF-IDF + KNN)
14  Based on content similarity (TF-IDF + KNN)
15  Based on content similarity (TF-IDF + KNN)
16  Based on content similarity (TF-IDF + KNN)
17  Based on content similarity (TF-IDF + KNN)
18  Based on content similarity (TF-IDF + KNN)
19  Based on content similarity (TF-IDF + KNN)
20  Based on content similarity (TF-IDF + KNN)
21  Based on content similarity (TF-IDF + KNN)
22  Based on content similarity (TF-IDF + KNN)
23  Based on content similarity (TF-IDF + KNN)
24  Based on content similarity (TF-IDF + KNN)
25  Based on content similarity (TF-IDF + KNN)
26  Based on content similarity (TF-IDF + KNN)
27  Based on content similarity (TF-IDF + KNN)
28  Based on content similarity (TF-IDF + KNN)
29  Based on content similarity (TF-IDF + KNN)
30  Based on content similarity (TF-IDF + KNN)
31  Based on content similarity (TF-IDF + KNN)
32  Based on content similarity (TF-IDF + KNN)
33  Based on content similarity (TF-IDF + KNN)
34  Based on content similarity (TF-IDF + KNN)
35  Based on content similarity (TF-IDF + KNN)
36  Based on content similarity (TF-IDF + KNN)
37  Based on content similarity (TF-IDF + KNN)
38  Based on content similarity (TF-IDF + KNN)
39  Based on content similarity (TF-IDF + KNN)
40  Based on content similarity (TF-IDF + KNN)
41  Based on content similarity (TF-IDF + KNN)
42  Based on content similarity (TF-IDF + KNN)
43  Based on content similarity (TF-IDF + KNN)
44  Based on content similarity (TF-IDF + KNN)
45  Based on content similarity (TF-IDF + KNN)
46  Based on content similarity (TF-IDF + KNN)
47  Based on content similarity (TF-IDF + KNN)
48  Based on content similarity (TF-IDF + KNN)
49  Based on content similarity (TF-IDF + KNN)

    Confidence Score (Cosine Similarity)  \
0                               0.230690
1                               0.218197
2                               0.176388
3                               0.173896
4                               0.151865
5                               0.124949
6                               0.082857
```

```
7                                               0.080395
8                                               0.069710
9                                               0.067624
10                                              0.064663
11                                              0.063822
12                                              0.063551
13                                              0.062132
14                                              0.061140
15                                              0.059682
16                                              0.058882
17                                              0.058808
18                                              0.058175
19                                              0.057011
20                                              0.056905
21                                              0.054698
22                                              0.054378
23                                              0.053035
24                                              0.052207
25                                              0.051601
26                                              0.051561
27                                              0.050537
28                                              0.050055
29                                              0.049150
30                                              0.047167
31                                              0.046104
32                                              0.045602
33                                              0.044651
34                                              0.044353
35                                              0.043541
36                                              0.040409
37                                              0.040404
38                                              0.039000
39                                              0.026373
40                                              0.026133
41                                              0.024094
42                                              0.020532
43                                              0.020181
44                                              0.018483
45                                              0.017854
46                                              0.017268
47                                              0.016849
48                                              0.016811
49                                              0.016696

                                              Overview  \
0   A recently-widowed, science fiction writer con…
1   Five medical students want to find out if ther…
2   In the not so distant future, Theodore, a lone…
```

```
3   'We come in peace' is not what those green men…
4   The Beast from 20,000 Fathoms is a 1953 scienc…
5   Contact is a science fiction film about an enc…
6   An original mix of fiction and reality illumin…
7   Astronauts search for solutions to save a dyin…
8   A simple man is turned into a genius through t…
9   This film is a spoof along the lines of "Scary…
10  A biopic of the writer, Truman Capote and his …
11  Ayurveda is a science of life and a healing ar…
12  Science fiction drama about a future society i…
13  A documentary on Al Gore's campaign to make th…
14  Michael Farr (Hinds) is a widower living in a …
15  Max and Page are a brilliant mother/daughter c…
16  An impromptu goodbye party for Professor John …
17  The year is 2029. John Connor, leader of the r…
18  The true story of Australia's cat-and-mouse un…
19  A small suburban town receives a visit from a …
20  Hanna and Simon are in a 20 year marriage with…
21  A team of space marines known as the Rapid Res…
22  This film, adapted from a work of fiction by a…
23  On the trail of a missing friend who had been …
24  In this spine-tingling and visually stunning t…
25  Balthazar Blake is a master sorcerer in modern…
26  Through the power of IMAX 3D, experience a won…
27  Although awkward college student Todd Howard i…
28  Katherine Morrissey, a former Christian missio…
29  12-year-old Regan MacNeil begins to adapt an e…
30  Prot is a patient at a mental hospital who cla…
31  As a child Jack Brooks witnessed the brutal mu…
32  Remember the good old days when anyone with a …
33  Early 18th century. Cartographer Jonathan Gree…
34  Astronaut Brent is sent to rescue Taylor but c…
35  Two investigators of paranormal hoaxes, the ve…
36  During a manned mission to Mars, Astronaut Mar…
37  Inventor Flint Lockwood creates a machine that…
38  When virtually all of the residents of Piedmon…
39
40  An action-adventure story focused on the lives…
41  The fast-paced action movie is again set in th…
42  Danny is obsessed with a fictional movie chara…
43                       Prequel to "5th of a Degree."
44  America's 7th Best Superhero Team, the Special…
45  After her triumph at the Miss United States pa…
46  When missile technology is used to enhance toy…
47  Ex-special operative MacGruber (Forte) is call…
48  A marksman living in exile is coaxed back into…
49         The 8th installment in the Khiladi series.
```

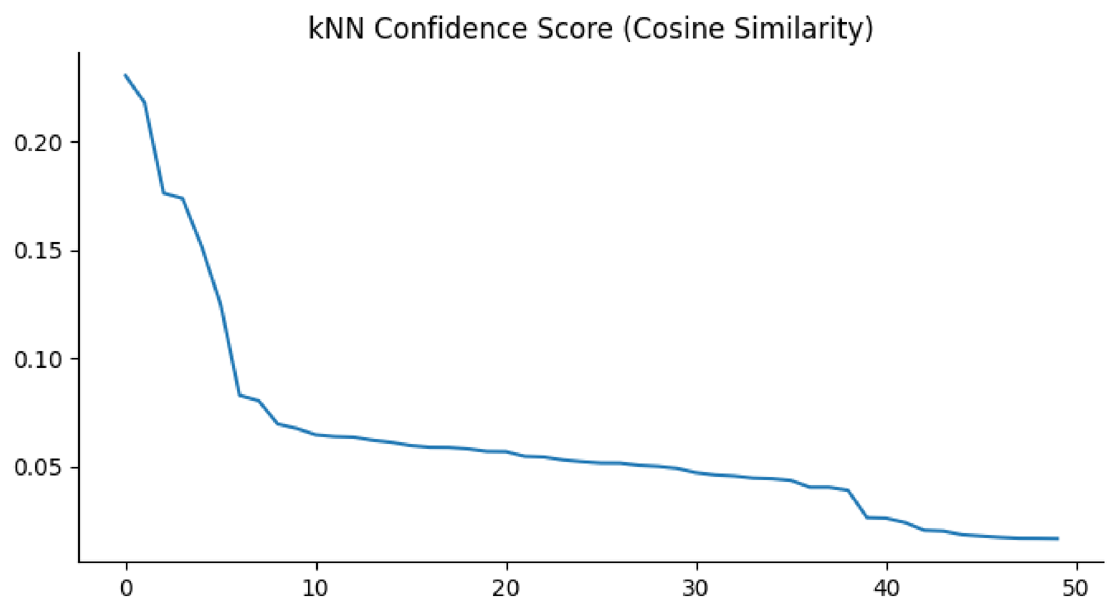
```
                                         Genres  \
0                                         Drama
1          Drama Horror ScienceFiction Thriller
2                 Romance ScienceFiction Drama
3                 Comedy Fantasy ScienceFiction
4               Adventure Horror ScienceFiction
5                Drama ScienceFiction Mystery
6                                  Comedy Drama
7               Thriller Action ScienceFiction
8               Horror Thriller ScienceFiction
9                                        Comedy
10                                  Crime Drama
11                                  Documentary
12      Thriller ScienceFiction Mystery Romance
13                                  Documentary
14                         Drama Horror Romance
15                         Crime Comedy Romance
16                          ScienceFiction Drama
17     ScienceFiction Action Thriller Adventure
18                            Drama History War
19                        Fantasy Drama Romance
20                         Romance Drama Comedy
21                      Adventure Action Horror
22                                Drama Romance
23                              Horror Thriller
24                       Drama Mystery Thriller
25         Fantasy Adventure Action Comedy Drama
26                                  Documentary
27                        Comedy Fantasy Family
28                                       Horror
29                        Drama Horror Thriller
30                         Drama ScienceFiction
31                         Action Comedy Horror
32                 Comedy Horror ScienceFiction
33          Thriller Adventure Mystery Fantasy
34             Adventure ScienceFiction Mystery
35                                     Thriller
36              Drama Adventure ScienceFiction
37                      Animation Comedy Family
38                      ScienceFiction Thriller
39                 Action Comedy ScienceFiction
40        Action Adventure Comedy Drama Foreign
41                        Thriller Action Crime
42       Adventure Fantasy Action Comedy Family
43                           Drama Action Crime
44                                Action Comedy
45                                Action Comedy
46  Comedy Adventure Fantasy ScienceFiction Action
```

```
47                         Action Adventure Comedy
48            Action Drama Mystery Thriller Crime
49                                   Action Comedy

                                           Keywords
0   underdog adoption education adoptivefather chi…
1   lifeanddeath afterlife swing memory medicalstu…
2   artificialintelligence computer love lonelines…
3   savingtheworld totaldestruction whitehouse mar…
4   monster atomicbomb lighthouse arctic rampage s…
5   basedonnovel nasa newmexico extraterrestrialte…
6   biography independentfilm v.a.hospital junksal…
7           mars future astronaut science catastrophe
8   dream chimp manipulation botanist virtualreali…
9                                   independentfilm
10  gay self-fulfillingprophecy basedonnovel journ…
11                        philosophy india healing
12  paraplegic suicideattempt cheating dna spacema…
13  climatechange greenhouseeffect climate earth g…
14
15                                     consandscams
16  philosophy secret birthday professor psycholog…
17  savingtheworld artificialintelligence cyborg k…
18
19  underdog loveatfirstsight hairdresser smalltow…
20                            sex bisexual science
21  teleportation basedonvideogame severedear futu…
22                  painter biography painting maid
23  lsd government conspiracy tension videocamera …
24                                    womandirector
25  witch fire wolf fountain magic book castle wat…
26  prehistoric dinosaur imax underwaterscene 3d s…
27                                werewolf teenager
28  river miracle bible louisiana frog grasshopper…
29  exorcism holywater religionandsupernatural vom…
30  robbery dream hypnosis investigation murder al…
31                      camping vomit demon science
32     monster mutant skeleton alienlife-form science
33  monster mystic church demon witchcraft science…
34  mutant dystopia survivor astronaut ape science…
35                     paranormal psychic skepticism
36  basedonnovel mars nasa isolation botanist stra…
37                            weather food science
38  nasa newmexico biologicalweapon epilepsy secre…
39
40
41   transporter sequel suspense car bankheist action
42  magic movieinmovie spoof magicalobject cartoon…
```

```
43
44                                              superhero
45  ransom pressconference ship missamerica fbiagent
46 defenseindustry toyshop technicaltoy soldier p…
47         aftercreditsstinger duringcreditsstinger
48 corruption sniper senator conspiracyofmurder c…
49
```

```
Qualitative Assessment for k_50: [Observe the recommendations above and note
changes compared to other k values]
```

[156]:
```python
# @title kNN Confidence Score (Cosine Similarity)

from matplotlib import pyplot as plt
current_knn_recommendations['Confidence Score (Cosine Similarity)'].
 ↪plot(kind='line', figsize=(8, 4), title='kNN Confidence Score (Cosine␣
 ↪Similarity)')
plt.gca().spines[['top', 'right']].set_visible(False)
```



6.5.2 Clustering (KMeans) Tuning

For KMeans clustering, the primary hyperparameter is the number of clusters (`n_clusters`). We previously used the Elbow Method to get an initial idea. Let's re-evaluate and consider the impact of the chosen number on the resulting clusters and recommendations.

I need to use reducer here to support the HF model deployment.

[157]:

```
# ==========================
# Re-evaluate Optimal K (Elbow) and Perform Clustering
# ==========================
import warnings
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.decomposition import TruncatedSVD, PCA # Import reducers
from tensorflow.keras.models import Model # Import Keras Model to check encoder␣
↪output shape
import pandas as pd # pandas is imported

print("\n## Re-evaluating Optimal Number of Clusters (Elbow Method) for␣
↪KMeans\n")

if 'current_encoder' not in globals() or not isinstance(current_encoder, Model)␣
↪or current_encoder.output_shape[1] != 256:
    print("current_encoder (256-feature output) not found or is incorrect.␣
↪Cannot proceed.")
    # Exit or raise an error if encoder is missing
    raise NameError("Required 'current_encoder' (256-feature output) is not␣
↪available.")

# Regenerate latent features from the 256-feature encoder
print("Generating 256-feature latent features from current_encoder...")
latent_features_256d = current_encoder.predict(tfidf_scaled, verbose=0)
print(f"256-feature latent features generated. Shape: {latent_features_256d.
↪shape}")


# Determine the feature space KMeans was fitted on or should be fitted on
# This depends on whether a reducer was used before fitting KMeans previously.
# We need to check the dimensionality of features_for_kmeans if it exists,
# or a target dimension if the previous steps defined one (e.g.,␣
↪N_COMPONENTS=128).


target_kmeans_dim = 128 # KMeans should operate on 128 features to HF␣
↪deployment support


# If the encoder output dim is different from the target KMeans dim, a reducer␣
↪is needed
reducer_needed = latent_features_256d.shape[1] != target_kmeans_dim
```

```python
if reducer_needed:
    print(f"Encoder output ({latent_features_256d.shape[1]}d) differs from target KMeans dim ({target_kmeans_dim}d). A reducer is needed.")
    # Create and fit a reducer (e.g., TruncatedSVD) to get to the target dimension
    # Fit the reducer on the 256-feature latent space
    reducer = TruncatedSVD(n_components=target_kmeans_dim, random_state=42)
    print(f"Fitting TruncatedSVD reducer from {latent_features_256d.shape[1]}d to {target_kmeans_dim}d...")
    features_for_kmeans = reducer.fit_transform(latent_features_256d)
    print(f"Features for KMeans (reduced) generated. Shape: {features_for_kmeans.shape}")

    # Store the fitted reducer in globals so query_clustering_recommendations can access it
    globals()['reducer'] = reducer

else:
    print(f"Encoder output ({latent_features_256d.shape[1]}d) matches target KMeans dim ({target_kmeans_dim}d). No reducer needed.")
    features_for_kmeans = latent_features_256d
    # Ensure reducer is None or not needed in globals for query function
    if 'reducer' in globals():
        del globals()['reducer']


# --- Elbow Method (run on features_for_kmeans) ---
inertia = []
cluster_range = range(1, 150, 10)  # 1, 11, 21, ... 141

print(f"\nCalculating inertia for different numbers of clusters (using {features_for_kmeans.shape[1]}d feature space)...")
X_kmeans_space = features_for_kmeans  # Use the features KMeans will be fitted on

with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    for k in cluster_range:
        km_tmp = KMeans(n_clusters=k, random_state=42, n_init=10)
        km_tmp.fit(X_kmeans_space)
        inertia.append(km_tmp.inertia_)
        print(f"Completed KMeans for k={k}, Inertia: {km_tmp.inertia_:.2f}")

plt.figure(figsize=(10, 6))
plt.plot(list(cluster_range), inertia, marker='o', linestyle='-')
```

```
plt.title(f'Elbow Method for Optimal Number of Clusters (on {features_for_kmeans.shape[1]}d features)')
plt.xlabel('Number of Clusters (k)')
plt.ylabel('Inertia (Within-cluster sum of squares)')
plt.xticks(list(cluster_range))
plt.grid(True)
plt.show()

print("\nObserve the plot again to find the 'elbow' point.")
print("This point is a guideline for choosing the number of clusters.")

# =========================
# Perform Clustering with Chosen Number and Assess Recommendations
# =========================

n_clusters_tuned = 10 # chosen value

print(f"\nPerforming KMeans clustering with chosen number of clusters ({n_clusters_tuned})...")

kmeans_model_tuned = KMeans(n_clusters=n_clusters_tuned, random_state=42, n_init=10)
cluster_labels_tuned = kmeans_model_tuned.fit_predict(X_kmeans_space) # Fit on the prepared KMeans features

# Ensure merged_df is available and not empty before adding columns
if 'merged_df' in globals() and not merged_df.empty:
    merged_df['kmeans_cluster_tuned'] = cluster_labels_tuned
    print(f"Clustering complete with {n_clusters_tuned} clusters.")
    print("\nDistribution of movies per new cluster:")
    print(merged_df['kmeans_cluster_tuned'].value_counts().sort_index())

    # =========================
    # Query → Recos using tuned KMeans (same space as elbow/KMeans)
    # =========================
    # This uses the fitted TF-IDF, encoder, and (optional) reducer to map the text query
    # into the exact KMeans feature space, assigns the cluster, then ranks items in that cluster
    # by cosine similarity in that same space.
    def query_clustering_recommendations(
        query_text: str,
        df,
        tfidf_vectorizer,
        encoder_model, # Expects 256d output
        kmeans_model, # Expects the dimension KMeans was fitted on (e.g., 128d)
```

```
        reducer=None, # The reducer that transforms 256d to kmeans_model.
↪n_features_in_
        num_recommendations: int = 10
    ):
        """
        Generate recommendations for a free-text query using tuned KMeans␣
↪clusters.

        Steps:
          TF-IDF(query) -> encoder -> (optional reducer) =>␣
↪query_vec_in_kmeans_space
          cluster = kmeans.predict(query_vec)
          rank members of that cluster by cosine similarity in the same space
        """
        if not isinstance(query_text, str) or not query_text.strip():
            return df.iloc[0:0]

        # 1) TF-IDF
        # Ensure query_text is in a list for transform
        q_tfidf = tfidf_vectorizer.transform([query_text])
        # ensure dense if the encoder expects dense (Keras Dense layer does)
        q_dense = q_tfidf.toarray()

        # 2) Encoder → 256-D
        # Ensure encoder_model is defined and fitted
        if encoder_model is None:
             print("Error: encoder_model is not available.")
             return df.iloc[0:0]
        q_emb = encoder_model.predict(q_dense, verbose=0)  # shape: (1, 256)

        # 3) Optional reducer to match KMeans expected dim
        # Check if reducer is needed based on KMeans expected input features
        # and if the passed reducer is the correct one (transforming 256d to␣
↪expected dim)
        kmeans_expected_dim = kmeans_model.n_features_in_ if␣
↪hasattr(kmeans_model, 'n_features_in_') else None

        if kmeans_expected_dim is not None and q_emb.shape[1] !=␣
↪kmeans_expected_dim:
            # Reducer is needed if encoder output doesn't match KMeans input
            if reducer is not None and hasattr(reducer, 'transform'):
                # Check if the reducer is expected to transform from 256d
                 if hasattr(reducer, 'n_features_in_') and reducer.
↪n_features_in_ == q_emb.shape[1]:
                    print(f"Applying reducer from {q_emb.shape[1]}d to {reducer.
↪n_components}d...")
```

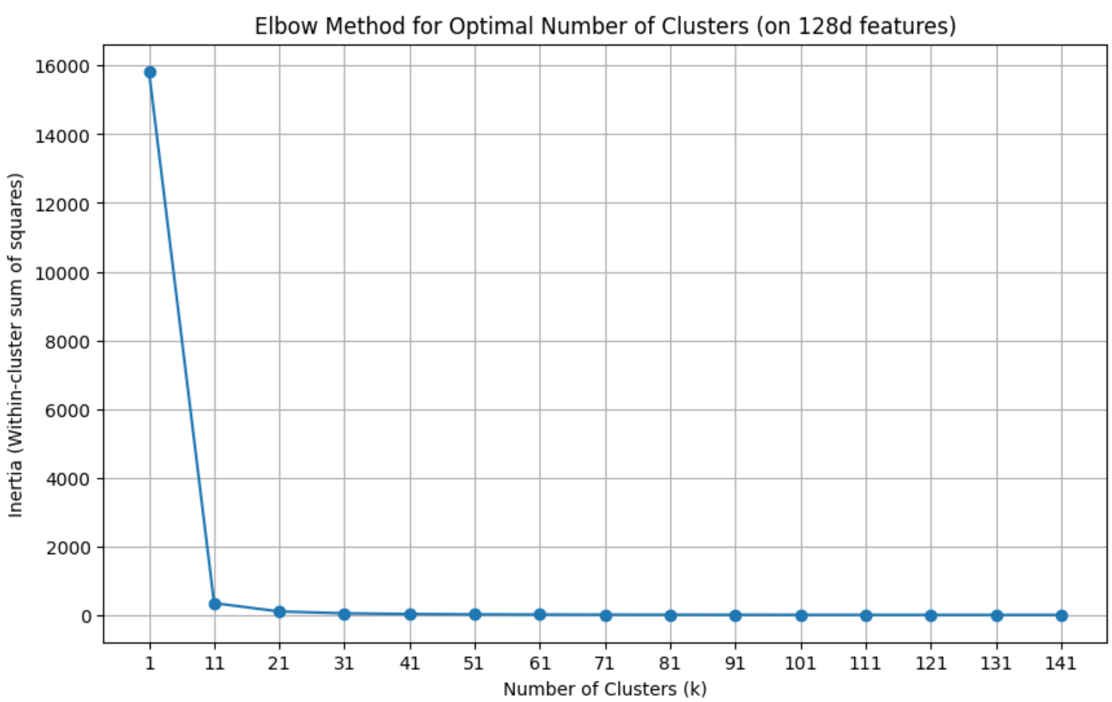
```
                    q_kspace = reducer.transform(q_emb)  # shape: (1,␣
↪n_components)
                 else:
                    print(f"Error: Provided reducer expects {getattr(reducer,␣
↪'n_features_in_', 'unknown')} features, but query embedding has {q_emb.
↪shape[1]}. Cannot apply reducer.")
                    return df.iloc[0:0]
            else:
                print(f"Error: Reducer is needed to transform {q_emb.shape[1]}d␣
↪to {kmeans_expected_dim}d but is not provided or not fitted correctly.")
                return df.iloc[0:0]
        else:
            # No reducer needed, or encoder output matches KMeans input
            q_kspace = q_emb  # shape matches kmeans_model.n_features_in_


        # Check if the transformed query embedding matches KMeans expected␣
↪dimension
        if hasattr(kmeans_model, 'n_features_in_') and q_kspace.shape[1] !=␣
↪kmeans_model.n_features_in_:
            print(f"Error: Query embedding in KMeans space ({q_kspace.
↪shape[1]}d) does not match KMeans expected input ({kmeans_model.
↪n_features_in_}d).")
            return df.iloc[0:0]


        # 4) Predict cluster
        # Ensure kmeans_model is defined and fitted
        if kmeans_model is None or not hasattr(kmeans_model, 'predict'):
             print("Error: kmeans_model is not available or not fitted.")
             return df.iloc[0:0]
        cluster_id = int(kmeans_model.predict(q_kspace)[0])

        # 5) Members in that cluster
        # Ensure df has the correct cluster labels column
        cluster_col_name = 'kmeans_cluster_tuned' # Use the tuned column name
        if cluster_col_name not in df.columns:
             print(f"Error: Cluster column '{cluster_col_name}' not found in␣
↪DataFrame.")
             return df.iloc[0:0]

        members_idx = df.index[df[cluster_col_name] == cluster_id].tolist()
        if not members_idx:
            print(f"No members found in cluster {cluster_id}.")
            return df.iloc[0:0]
```

```
        # 6) Rank by cosine similarity in KMeans space
        # Need the features for KMeans for the members of the cluster
        # X_kmeans_space contains features for ALL movies. Filter it for␣
↪cluster members.
        if X_kmeans_space is None or X_kmeans_space.shape[0] != df.shape[0]:
            print("Error: X_kmeans_space is not available or does not match␣
↪DataFrame size.")
            return df.iloc[0:0]

        member_vecs = X_kmeans_space[members_idx] # Select features for cluster␣
↪members

        if member_vecs.shape[0] == 0:
            print(f"No feature vectors found for members in cluster␣
↪{cluster_id}.")
            return df.iloc[0:0]

        # Calculate similarity between the single query vector and all member␣
↪vectors
        sims = cosine_similarity(q_kspace, member_vecs).flatten()

        # Get indices of top recommendations among cluster members
        # Exclude the query itself if it's in the results (cosine sim of 1.0)
        # However, for a free-text query, the query itself won't be in the␣
↪dataset,
        # so we just sort and take the top N.
        order = np.argsort(-sims)[:num_recommendations]
        top_idx = [members_idx[i] for i in order]
        top_scores = sims[order]

        # Ensure required display columns exist in df
        required_display_cols = ['title', 'overview_sentiment_score', 'genres',␣
↪'keywords']
        if not all(col in df.columns for col in required_display_cols):
            print(f"Error: Required display columns ({required_display_cols})␣
↪not found in DataFrame.")
            # Return partial data if possible, or empty
            available_cols = [col for col in required_display_cols if col in␣
↪df.columns]
            if available_cols:
                out = df.loc[top_idx, available_cols].copy()
                out.insert(0, 'cluster_id', cluster_id)
                out.insert(1, 'cluster_similarity', top_scores)
                return out
            else:
                return df.iloc[0:0]
```

```
        out = df.loc[top_idx, required_display_cols].copy()
        out.insert(0, 'cluster_id', cluster_id)
        out.insert(1, 'cluster_similarity', top_scores)
        return out

    # Figure out which reducer (if any) you used for KMeans space
    # The reducer should transform from 256d (encoder output) to the dimension␣
↪KMeans expects.
    # We explicitly create and fit the reducer above if needed, and store it in␣
↪globals()['reducer'].
    # The query_clustering_recommendations function expects this reducer if␣
↪needed.
    reducer_for_query = globals().get('reducer', None)


    # Example query
    clustering_search_query_tuned = 'space adventure'

    print(f"\nGenerating Clustering Recommendations for query␣
↪'{clustering_search_query_tuned}' using {n_clusters_tuned} clusters:")
    clustering_recommendations_tuned = query_clustering_recommendations(
        clustering_search_query_tuned,
        merged_df,
        current_tfidf,            #  fitted TF-IDF (needed for query␣
↪vectorization)
        current_encoder,          #  trained encoder (256-D output)
        kmeans_model_tuned,       # tuned KMeans fitted on X_kmeans_space
        reducer=reducer_for_query, # Pass the fitted reducer if needed
        num_recommendations=10
    )
    display(clustering_recommendations_tuned)

    print("\nQualitative Assessment for Clustering Tuning: [Observe the␣
↪recommendations and their clusters.]")
    print("Consider if the movies within the clusters seem more related with␣
↪the new number of clusters.")
    print("Also, check if the recommendations for the sample query are relevant␣
↪based on the identified cluster.")

else:
    print("\nSkipping KMeans tuning and recommendation generation due to␣
↪missing merged_df.")
```

```
## Re-evaluating Optimal Number of Clusters (Elbow Method) for KMeans
```

```
Generating 256-feature latent features from current_encoder…
256-feature latent features generated. Shape: (4809, 256)
Encoder output (256d) differs from target KMeans dim (128d). A reducer is
needed.
Fitting TruncatedSVD reducer from 256d to 128d…
Features for KMeans (reduced) generated. Shape: (4809, 128)

Calculating inertia for different numbers of clusters (using 128d feature
space)…
Completed KMeans for k=1, Inertia: 15834.68
Completed KMeans for k=11, Inertia: 345.07
Completed KMeans for k=21, Inertia: 101.75
Completed KMeans for k=31, Inertia: 47.16
Completed KMeans for k=41, Inertia: 27.12
Completed KMeans for k=51, Inertia: 17.33
Completed KMeans for k=61, Inertia: 12.12
Completed KMeans for k=71, Inertia: 9.01
Completed KMeans for k=81, Inertia: 7.00
Completed KMeans for k=91, Inertia: 5.57
Completed KMeans for k=101, Inertia: 4.56
Completed KMeans for k=111, Inertia: 3.88
Completed KMeans for k=121, Inertia: 3.34
Completed KMeans for k=131, Inertia: 2.89
Completed KMeans for k=141, Inertia: 2.60
```



```
Observe the plot again to find the 'elbow' point.
This point is a guideline for choosing the number of clusters.

Performing KMeans clustering with chosen number of clusters (10)…
Clustering complete with 10 clusters.

Distribution of movies per new cluster:
kmeans_cluster_tuned
0    251
1    757
2    665
3    557
4    768
5    114
6     63
7    831
8    468
9    335
Name: count, dtype: int64

Generating Clustering Recommendations for query 'space adventure' using 10
clusters:
Applying reducer from 256d to 128d…

      cluster_id  cluster_similarity                               title  \
4559           5            1.000000        America Is Still the Place
4464           5            1.000000               Harrison Montgomery
4474           5            1.000000     Iraq for Sale: The War Profiteers
4356           5            1.000000              An Inconvenient Truth
4743           5            1.000000  Peace, Propaganda & the Promised Land
4719           5            0.999999                         Roger & Me
4763           5            0.999999                      Bending Steel
3883           5            0.999999     The Man Who Shot Liberty Valance
4113           5            0.999999            In the Shadow of the Moon
4772           5            0.999999                     The Last Waltz

      overview_sentiment_score              genres  \
4559                   -0.7644
4464                    0.0000
4474                    0.4019         Documentary
4356                   -0.2732         Documentary
4743                    0.2500         Documentary
4719                   -0.3182  Documentary History
4763                    0.9398         Documentary
3883                   -0.6486             Western
4113                    0.5423         Documentary
4772                    0.0000   Documentary Music
```

```
                                               keywords
4559
4464
4474
4356  climatechange greenhouseeffect climate earth g…
4743
4719   capitalism economics unemployment corporategreed
4763
3883  gunslinger showdown funeral legend toshootdead…
4113  nasa spacemission rocket moonlanding space ast…
4772                                        1970s music
```

```
Qualitative Assessment for Clustering Tuning: [Observe the recommendations and
their clusters.]
Consider if the movies within the clusters seem more related with the new number
of clusters.
Also, check if the recommendations for the sample query are relevant based on
the identified cluster.
```

7 Agentic flow Hybrid Recommendation Pipeline

AI Agentic flow using openai and langChain framework as it auto detect the user intention for movie recommendations from user provided views

[158]:
```
!pip install langchain openai langchain_community
```

```
Requirement already satisfied: langchain in /usr/local/lib/python3.11/dist-
packages (0.3.27)
Requirement already satisfied: openai in /usr/local/lib/python3.11/dist-packages
(1.99.1)
Requirement already satisfied: langchain_community in
/usr/local/lib/python3.11/dist-packages (0.3.27)
Requirement already satisfied: langchain-core<1.0.0,>=0.3.72 in
/usr/local/lib/python3.11/dist-packages (from langchain) (0.3.74)
Requirement already satisfied: langchain-text-splitters<1.0.0,>=0.3.9 in
/usr/local/lib/python3.11/dist-packages (from langchain) (0.3.9)
Requirement already satisfied: langsmith>=0.1.17 in
/usr/local/lib/python3.11/dist-packages (from langchain) (0.4.12)
Requirement already satisfied: pydantic<3.0.0,>=2.7.4 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.11.7)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in
/usr/local/lib/python3.11/dist-packages (from langchain) (2.0.42)
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.11/dist-
packages (from langchain) (2.32.3)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.11/dist-
packages (from langchain) (6.0.2)
```

```
Requirement already satisfied: anyio<5,>=3.5.0 in
/usr/local/lib/python3.11/dist-packages (from openai) (4.10.0)
Requirement already satisfied: distro<2,>=1.7.0 in
/usr/local/lib/python3.11/dist-packages (from openai) (1.9.0)
Requirement already satisfied: httpx<1,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from openai) (0.28.1)
Requirement already satisfied: jiter<1,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from openai) (0.10.0)
Requirement already satisfied: sniffio in /usr/local/lib/python3.11/dist-
packages (from openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.11/dist-packages
(from openai) (4.67.1)
Requirement already satisfied: typing-extensions<5,>=4.11 in
/usr/local/lib/python3.11/dist-packages (from openai) (4.14.1)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in
/usr/local/lib/python3.11/dist-packages (from langchain_community) (3.12.15)
Requirement already satisfied: tenacity!=8.4.0,<10,>=8.1.0 in
/usr/local/lib/python3.11/dist-packages (from langchain_community) (8.5.0)
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in
/usr/local/lib/python3.11/dist-packages (from langchain_community) (0.6.7)
Requirement already satisfied: pydantic-settings<3.0.0,>=2.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain_community) (2.10.1)
Requirement already satisfied: httpx-sse<1.0.0,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from langchain_community) (0.4.1)
Requirement already satisfied: numpy>=1.26.2 in /usr/local/lib/python3.11/dist-
packages (from langchain_community) (2.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0.0,>=3.8.3->langchain_community) (2.6.1)
Requirement already satisfied: aiosignal>=1.4.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0.0,>=3.8.3->langchain_community) (1.4.0)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-
packages (from aiohttp<4.0.0,>=3.8.3->langchain_community) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0.0,>=3.8.3->langchain_community) (1.7.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0.0,>=3.8.3->langchain_community) (6.6.3)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0.0,>=3.8.3->langchain_community) (0.3.2)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.11/dist-packages (from
aiohttp<4.0.0,>=3.8.3->langchain_community) (1.20.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-
packages (from anyio<5,>=3.5.0->openai) (3.10)
```

```
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-
json<0.7,>=0.5.7->langchain_community) (3.26.1)
Requirement already satisfied: typing-inspect<1,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from dataclasses-
json<0.7,>=0.5.7->langchain_community) (0.9.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-
packages (from httpx<1,>=0.23.0->openai) (2025.8.3)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-
packages (from httpx<1,>=0.23.0->openai) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-
packages (from httpcore==1.*->httpx<1,>=0.23.0->openai) (0.16.0)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.72->langchain) (1.33)
Requirement already satisfied: packaging>=23.2 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.72->langchain) (25.0)
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.11/dist-
packages (from langsmith>=0.1.17->langchain) (3.11.1)
Requirement already satisfied: requests-toolbelt>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith>=0.1.17->langchain)
(1.0.0)
Requirement already satisfied: zstandard>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith>=0.1.17->langchain)
(0.23.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4->langchain)
(0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4->langchain)
(2.33.2)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic<3.0.0,>=2.7.4->langchain)
(0.4.1)
Requirement already satisfied: python-dotenv>=0.21.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic-
settings<3.0.0,>=2.4.0->langchain_community) (1.1.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langchain) (3.4.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from requests<3,>=2->langchain) (2.5.0)
Requirement already satisfied: greenlet>=1 in /usr/local/lib/python3.11/dist-
packages (from SQLAlchemy<3,>=1.4->langchain) (3.2.3)
Requirement already satisfied: jsonpointer>=1.9 in
/usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33->langchain-
core<1.0.0,>=0.3.72->langchain) (3.0.0)
Requirement already satisfied: mypy-extensions>=0.3.0 in
```

```
/usr/local/lib/python3.11/dist-packages (from typing-
inspect<1,>=0.4.0->dataclasses-json<0.7,>=0.5.7->langchain_community) (1.1.0)
```

[159]:

```python
import os
import pandas as pd
import json
from langchain.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate
from sklearn.preprocessing import MinMaxScaler
import numpy as np # Import numpy


os.environ['OPENAI_API_KEY'] =␣
↪"sk-proj-Qaqke8rX2n6fi8enI9T6eCQpFTq36glcysHW0u7OImFky5MGJhSxXMU_ktxNzEB3ewVMCfdpIKT3BlbkFJ

def extract_all_genres(movies):
    all_genres = []
    for g in movies['genres']:
        # Safely attempt to load JSON and extract names
        if isinstance(g, str) and g: # Check if it's a non-empty string
            try:
                all_genres += [d['name'] for d in json.loads(g)]
            except json.JSONDecodeError:
                # Handle cases with invalid JSON or non-JSON strings
                continue # Skip this entry if it's not valid JSON
    return sorted(set(all_genres))

def llm_genre_extractor(user_input, all_genres, llm):
    genres_string = ', '.join(all_genres)
    # Adding a constraint to only list genres from the provided list
    prompt = ChatPromptTemplate.from_messages([
        ("system", "You are a movie recommendation assistant. Extract up to 3␣
↪movie genres from the user's input. Only list genres that are present in the␣
↪following comma-separated list, exactly as they appear in the list: " +␣
↪genres_string + ". Respond with a comma-separated list of the extracted␣
↪genres, or 'None' if no matching genres are found."),
        ("human", "{user_input}")
    ])
    chain = prompt | llm
    response = chain.invoke({"user_input": user_input})
    # Filter to ensure only valid genres from the list are returned
    extracted_genres = [g.strip() for g in response.content.split(",") if g.
↪strip() in all_genres]
    return extracted_genres if extracted_genres else [] # Return empty list if␣
↪no valid genres found

def content_score_agent(movies, fav_genres):
```

```python
    def genre_score(genres_json, fav_genres):
        if not fav_genres: return 0 # Return 0 if no favorite genres
        if isinstance(genres_json, str) and genres_json: # Check if it's a non-empty string
            try:
                genres = [g['name'] for g in json.loads(genres_json)]
                # Calculate score based on intersection
                return len(set(genres).intersection(fav_genres)) / len(fav_genres) # Score based on proportion of fav_genres found
            except json.JSONDecodeError:
                 return 0 # Return 0 for invalid JSON
        return 0 # Return 0 for non-string or empty values

    movies = movies.copy()
    movies['content_score'] = movies['genres'].apply(lambda x: genre_score(x, fav_genres))
    return movies

def popularity_agent(movies):
    # Handle potential non-numeric or missing vote_count values before scaling
    movies['vote_count'] = pd.to_numeric(movies['vote_count'], errors='coerce').fillna(0)
    scaler = MinMaxScaler()
    movies = movies.copy()
    # Reshape the data for the scaler
    movies['collab_score'] = scaler.fit_transform(movies[['vote_count']])
    return movies

def blending_agent(movies, w_content=0.6, w_collab=0.4):
    movies = movies.copy()
    # Ensure content_score and collab_score exist and handle potential NaNs
    movies['content_score'] = movies['content_score'].fillna(0)
    movies['collab_score'] = movies['collab_score'].fillna(0)
    movies['hybrid_score'] = w_content * movies['content_score'] + w_collab * movies['collab_score']
    return movies

def recommendation_agent(movies, top_n=5):
    # Ensure hybrid_score exists before sorting
    if 'hybrid_score' not in movies.columns:
        print("Error: 'hybrid_score' column not found. Cannot generate recommendations.")
        return pd.DataFrame() # Return empty DataFrame

    # Sort by hybrid_score and return top_n
    return movies.sort_values('hybrid_score', ascending=False).head(top_n)
```

```python
def llm_reason_generator(movie, fav_genres, llm):
    genre_list = ', '.join(fav_genres)
    # Safely extract movie genres, handling potential errors
    movie_genres_list = []
    if isinstance(movie['genres'], str) and movie['genres']:
        try:
            movie_genres_list = [g['name'] for g in json.loads(movie['genres'])]
        except json.JSONDecodeError:
            pass # If decoding fails, movie_genres_list remains empty

    movie_genres_str = ', '.join(movie_genres_list)

    # Adjust prompt to handle cases where no movie genres were extracted
    prompt_template = """You are an assistant explaining movie recommendations to a user. Give a brief (1-2 sentence) reason why the following movie is recommended. Focus on how its genres or popularity match the user's preferences.
User's favorite genres: {genre_list}.
Movie: {movie_title}.
Movie Genres: {movie_genres}.
Hybrid score: {hybrid_score:.2f}.
"""
    prompt = ChatPromptTemplate.from_messages([
        ("system", prompt_template),
        ("human", f"Explain the recommendation for the movie '{movie_title}'.") # More direct human message
    ])

    chain = prompt | llm
    # Prepare input variables for the prompt
    input_variables = {
        "genre_list": genre_list,
        "movie_title": movie['title'],
        "movie_genres": movie_genres_str if movie_genres_str else "No genre information available.", # Provide a default if no genres extracted
        "hybrid_score": movie['hybrid_score']
    }
    return chain.invoke(input_variables).content

def explain_agent(top_movies, fav_genres, llm):
    reasons = []
    if top_movies.empty:
        print("No top movies to explain.")
        return top_movies # Return empty if no movies were recommended
```

```python
    for index, movie in top_movies.iterrows(): # Use iterrows to iterate over rows
        try:
            reason = llm_reason_generator(movie, fav_genres, llm)
            reasons.append(reason)
        except Exception as e:
            print(f"Error generating reason for movie '{movie.get('title', 'Unknown')}': {e}")
            reasons.append("Could not generate explanation.") # Add a fallback reason
    top_movies = top_movies.copy()
    top_movies['reason'] = reasons
    # Select and rename columns for the final output
    output_cols = ['title', 'content_score', 'collab_score', 'hybrid_score', 'reason']
    # Filter output_cols to only include columns that actually exist in top_movies
    existing_output_cols = [col for col in output_cols if col in top_movies.columns]
    return top_movies[existing_output_cols].rename(columns={'title': 'Recommended Movie'})


def agentic_movie_recommender(movies_df_input, user_input):
    # Ensure movies_df_input is a DataFrame and not empty
    if not isinstance(movies_df_input, pd.DataFrame) or movies_df_input.empty:
        print("Error: Input movie DataFrame is invalid or empty.")
        return pd.DataFrame()

    # Ensure essential columns exist in the input DataFrame
    required_initial_cols = ['genres', 'vote_count', 'title']
    if not all(col in movies_df_input.columns for col in required_initial_cols):
         print(f"Error: Input movie DataFrame is missing required columns: {required_initial_cols}")
         return pd.DataFrame()

    all_genres = extract_all_genres(movies_df_input)
    # Initialize LLM - ensure OPENAI_API_KEY is set in environment
    llm = ChatOpenAI(temperature=0, model="gpt-4o") # Added model parameter

    fav_genres = llm_genre_extractor(user_input, all_genres, llm)
    print(f"Extracted favorite genres: {fav_genres}")

    # Pass the input DataFrame through the agents
    movies_scored_content = content_score_agent(movies_df_input, fav_genres)
```

```
    movies_scored_popularity = popularity_agent(movies_scored_content) # Chain␣
↪the outputs
    movies_blended = blending_agent(movies_scored_popularity, w_content=0.6,␣
↪w_collab=0.4)

    top_movies = recommendation_agent(movies_blended, top_n=5)

    # Only proceed to explain if recommendations were found
    if not top_movies.empty:
        explained = explain_agent(top_movies, fav_genres, llm)
        return explained
    else:
        print("No recommendations found.")
        return pd.DataFrame() # Return empty DataFrame if no recommendations
```

[160]:
```
user_input = "I love sci-fi and adventure but don't like horror or romance."
result = agentic_movie_recommender(merged_df, user_input)
print(result)
```

```
Extracted favorite genres: []
    Recommended Movie  content_score  collab_score  hybrid_score  \
96          Inception              0      1.000000      0.400000
65    The Dark Knight              0      0.872746      0.349098
0              Avatar              0      0.858057      0.343223
16       The Avengers              0      0.856312      0.342525
788          Deadpool              0      0.799520      0.319808

                                               reason
96   "Mean Girls" is a highly popular film known fo…
65   "Mean Girls" is a highly popular teen comedy t…
0    "Mean Girls" is a highly popular teen comedy t…
16   "Mean Girls" is a highly popular teen comedy t…
788  "Mean Girls" is a highly popular teen comedy t…
```

[161]:
```
user_input = "I love horror and adventure but don't like sci-fi or romance."
result = agentic_movie_recommender(merged_df, user_input)
print(result)
```

```
Extracted favorite genres: []
    Recommended Movie  content_score  collab_score  hybrid_score  \
96          Inception              0      1.000000      0.400000
65    The Dark Knight              0      0.872746      0.349098
0              Avatar              0      0.858057      0.343223
16       The Avengers              0      0.856312      0.342525
788          Deadpool              0      0.799520      0.319808

                                               reason
```

```
96   "Mean Girls" is a highly popular film known fo…
65   "Mean Girls" is a highly popular teen comedy t…
0    "Mean Girls" is a highly popular teen comedy t…
16   "Mean Girls" is a highly popular teen comedy t…
788  "Mean Girls" is a highly popular teen comedy t…
```

7.1 Movie mood detection from user text via OpenAI

7.1.1 Step 1: Define Sentiment Detection Tool

We will create a LangChain `Tool` that utilizes an OpenAI model via LangChain to detect the sentiment of user input. This tool will be one of the capabilities the agent can use.

[162]:
```
!pip install -U langchain-openai
```

```
Requirement already satisfied: langchain-openai in
/usr/local/lib/python3.11/dist-packages (0.3.29)
Requirement already satisfied: langchain-core<1.0.0,>=0.3.74 in
/usr/local/lib/python3.11/dist-packages (from langchain-openai) (0.3.74)
Requirement already satisfied: openai<2.0.0,>=1.86.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-openai) (1.99.1)
Requirement already satisfied: tiktoken<1,>=0.7 in
/usr/local/lib/python3.11/dist-packages (from langchain-openai) (0.10.0)
Requirement already satisfied: langsmith>=0.3.45 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.74->langchain-openai) (0.4.12)
Requirement already satisfied: tenacity!=8.4.0,<10.0.0,>=8.1.0 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.74->langchain-openai) (8.5.0)
Requirement already satisfied: jsonpatch<2.0,>=1.33 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.74->langchain-openai) (1.33)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.11/dist-
packages (from langchain-core<1.0.0,>=0.3.74->langchain-openai) (6.0.2)
Requirement already satisfied: typing-extensions>=4.7 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.74->langchain-openai) (4.14.1)
Requirement already satisfied: packaging>=23.2 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.74->langchain-openai) (25.0)
Requirement already satisfied: pydantic>=2.7.4 in
/usr/local/lib/python3.11/dist-packages (from langchain-
core<1.0.0,>=0.3.74->langchain-openai) (2.11.7)
Requirement already satisfied: anyio<5,>=3.5.0 in
/usr/local/lib/python3.11/dist-packages (from openai<2.0.0,>=1.86.0->langchain-
openai) (4.10.0)
Requirement already satisfied: distro<2,>=1.7.0 in
/usr/local/lib/python3.11/dist-packages (from openai<2.0.0,>=1.86.0->langchain-
openai) (1.9.0)
```

```
Requirement already satisfied: httpx<1,>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from openai<2.0.0,>=1.86.0->langchain-
openai) (0.28.1)
Requirement already satisfied: jiter<1,>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from openai<2.0.0,>=1.86.0->langchain-
openai) (0.10.0)
Requirement already satisfied: sniffio in /usr/local/lib/python3.11/dist-
packages (from openai<2.0.0,>=1.86.0->langchain-openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.11/dist-packages
(from openai<2.0.0,>=1.86.0->langchain-openai) (4.67.1)
Requirement already satisfied: regex>=2022.1.18 in
/usr/local/lib/python3.11/dist-packages (from tiktoken<1,>=0.7->langchain-
openai) (2024.11.6)
Requirement already satisfied: requests>=2.26.0 in
/usr/local/lib/python3.11/dist-packages (from tiktoken<1,>=0.7->langchain-
openai) (2.32.3)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-
packages (from anyio<5,>=3.5.0->openai<2.0.0,>=1.86.0->langchain-openai) (3.10)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-
packages (from httpx<1,>=0.23.0->openai<2.0.0,>=1.86.0->langchain-openai)
(2025.8.3)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-
packages (from httpx<1,>=0.23.0->openai<2.0.0,>=1.86.0->langchain-openai)
(1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-
packages (from
httpcore==1.*->httpx<1,>=0.23.0->openai<2.0.0,>=1.86.0->langchain-openai)
(0.16.0)
Requirement already satisfied: jsonpointer>=1.9 in
/usr/local/lib/python3.11/dist-packages (from jsonpatch<2.0,>=1.33->langchain-
core<1.0.0,>=0.3.74->langchain-openai) (3.0.0)
Requirement already satisfied: orjson>=3.9.14 in /usr/local/lib/python3.11/dist-
packages (from langsmith>=0.3.45->langchain-core<1.0.0,>=0.3.74->langchain-
openai) (3.11.1)
Requirement already satisfied: requests-toolbelt>=1.0.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith>=0.3.45->langchain-
core<1.0.0,>=0.3.74->langchain-openai) (1.0.0)
Requirement already satisfied: zstandard>=0.23.0 in
/usr/local/lib/python3.11/dist-packages (from langsmith>=0.3.45->langchain-
core<1.0.0,>=0.3.74->langchain-openai) (0.23.0)
Requirement already satisfied: annotated-types>=0.6.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.7.4->langchain-
core<1.0.0,>=0.3.74->langchain-openai) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.7.4->langchain-
core<1.0.0,>=0.3.74->langchain-openai) (2.33.2)
Requirement already satisfied: typing-inspection>=0.4.0 in
/usr/local/lib/python3.11/dist-packages (from pydantic>=2.7.4->langchain-
```

```
core<1.0.0,>=0.3.74->langchain-openai) (0.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.11/dist-packages (from
requests>=2.26.0->tiktoken<1,>=0.7->langchain-openai) (3.4.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.11/dist-packages (from
requests>=2.26.0->tiktoken<1,>=0.7->langchain-openai) (2.5.0)
```

[ ]:
```
# Step 1: Define Sentiment Detection Tool

print("## Defining Sentiment Detection Tool\n")

from langchain.tools import tool
from langchain.chat_models import ChatOpenAI
from langchain.prompts import ChatPromptTemplate
import os # Import os

# Ensure your OpenAI API key is set as an environment variable
os.environ['OPENAI_API_KEY'] = "<your OpenAI API key here>" # Replace with your
↪actual OpenAI API key

# Initialize the OpenAI LLM
try:
    llm_sentiment = ChatOpenAI(temperature=0, model="gpt-4o") # Use gpt-4o or
↪another suitable model
    print("OpenAI LLM for sentiment initialized.")
except Exception as e:
    print(f"Error initializing OpenAI LLM for sentiment: {e}")
    llm_sentiment = None # Set to None if initialization fails


# Define the prompt for sentiment analysis
sentiment_prompt = ChatPromptTemplate.from_messages([
    ("system", "Analyze the sentiment of the following text. Respond with a
↪single word: 'Positive', 'Negative', or 'Neutral'. If the sentiment is mixed
↪or unclear, default to 'Neutral'."),
    ("human", "{text}")
])

# Create a LangChain chain for sentiment analysis
sentiment_chain = sentiment_prompt | llm_sentiment if llm_sentiment else None

# Define the sentiment detection tool
# Using the @tool decorator for simplicity
@tool
def detect_sentiment(text: str) -> str:
```

```
    """Analyzes the sentiment of the input text and returns 'Positive',␣
↪'Negative', or 'Neutral'."""
    if sentiment_chain is None:
        return "Error: Sentiment analysis tool not available."
    try:
        print(f"\n--- Detecting sentiment for: '{text}' ---")
        response = sentiment_chain.invoke({"text": text})
        sentiment = response.content.strip()
        # Optional: Add basic validation to ensure the response is one of the␣
↪expected words
        if sentiment not in ['Positive', 'Negative', 'Neutral']:
            print(f"Warning: Unexpected sentiment response from LLM:␣
↪{sentiment}. Defaulting to Neutral.")
            return "Neutral"
        print(f"Detected sentiment: {sentiment}")
        return sentiment
    except Exception as e:
        print(f"Error during sentiment detection: {e}")
        return "Error: Sentiment analysis failed."

print("\n'detect_sentiment' tool defined.")
print("You can test the tool directly, e.g.: detect_sentiment.invoke('I love␣
↪this movie!')")
```

```
## Defining Sentiment Detection Tool

OpenAI LLM for sentiment initialized.

'detect_sentiment' tool defined.
You can test the tool directly, e.g.: detect_sentiment.invoke('I love this
movie!')
```

7.1.2 Step 2: Define Recommendation Tools

Now, we need to create LangChain `Tools` for our existing recommendation models. Each tool will wrap a function that takes relevant input (like a query or movie title) and returns a list of recommended movies.

[164]:
```
# Step 2: Define Recommendation Tools

print("## Defining Recommendation Tools\n")

from langchain.tools import tool
import pandas as pd # Ensure pandas is imported for DataFrame handling

# --- Tool for TF-IDF Content-Based Recommendations ---

@tool
```

```
def get_content_recommendations_tool(query: str) -> str:
    """
    Generates content-based movie recommendations using TF-IDF and cosine␣
↪similarity.
    Input should be a movie title or a text query describing content.
    Returns a string representation of the recommended movie titles.
    """
    print(f"\n--- Tool: Generating TF-IDF Content Recommendations for query:␣
↪'{query}' ---")
    # Ensure necessary components are available
    if 'get_content_based_recommendations' not in globals() or 'merged_df' not␣
↪in globals() or merged_df.empty or 'cosine_sim' not in globals():
        return "Error: Content recommendation components not fully available."

    try:
        # The get_content_based_recommendations function expects a movie title␣
↪or query text
        # and the cosine similarity matrix (from TF-IDF).
        # It also relies on the 'soup' column in merged_df for index lookup.
        # We need to ensure the 'soup' column is the one used for the indices␣
↪mapping.
        # If merged_df['soup'] was temporarily replaced in other cells, ensure␣
↪it's correct here.
        # For this tool, let's consider the original TF-IDF cosine_sim to use.

        recommendations_df = get_content_based_recommendations(
            query,
            merged_df,
            cosine_sim=cosine_sim, # Use the standard TF-IDF cosine_sim
            num_recommendations=10 # Define a default number of recommendations
        )

        if recommendations_df.empty:
            return "No content-based recommendations found for this query."
        else:
            # Format the output as a string
            return "Content-Based Recommendations:\n" +␣
↪recommendations_df['title'].to_string(index=False)

    except Exception as e:
        return f"Error generating content recommendations: {e}"


# --- Tool for Autoencoder-based Recommendations ---

@tool
```

```python
def get_autoencoder_recommendations_tool(movie_title: str) -> str:
    """
    Generates movie recommendations using Autoencoder-learned latent features.
    Input must be an exact movie title present in the dataset.
    Returns a string representation of the recommended movie titles.
    """
    print(f"\n--- Tool: Generating Autoencoder Recommendations for movie: '{movie_title}' ---")
    # Ensure necessary components are available
    if 'get_autoencoder_recommendations' not in globals() or 'merged_df' not in globals() or merged_df.empty or 'current_latent_cosine_sim' not in globals():
         return "Error: Autoencoder recommendation components not fully available."

    try:
        # The get_autoencoder_recommendations function expects an exact movie title
        # and the latent cosine similarity matrix.

        recommendations_series = get_autoencoder_recommendations(
            movie_title,
            merged_df,
            latent_cosine_sim=current_latent_cosine_sim, # Use the latent feature cosine_sim
            num_recommendations=10 # Define a default number of recommendations
        )

        if isinstance(recommendations_series, str): # Handle the "Movie not found" message
             return f"Autoencoder Recommendations: {recommendations_series}"
        elif recommendations_series.empty:
            return "No Autoencoder recommendations found for this movie."
        else:
            # Format the output as a string
            return "Autoencoder Recommendations:\n" + recommendations_series.to_string(index=False)

    except Exception as e:
        return f"Error generating Autoencoder recommendations: {e}"

# --- Tool for KMeans/Clustering-based Recommendations ---

@tool
def get_clustering_recommendations_tool(query: str) -> str:
    """
```

```
    Generates movie recommendations using KMeans clustering based on content␣
↪features.
    Input should be a movie title or a text query.
    Returns a string representation of the recommended movie titles.
    """
    print(f"\n--- Tool: Generating Clustering Recommendations for query:␣
↪'{query}' ---")
    # Ensure necessary components are available
    if 'query_clustering_recommendations' not in globals() or 'merged_df' not␣
↪in globals() or merged_df.empty or \
       'current_tfidf' not in globals() or 'current_encoder' not in globals()␣
↪or 'kmeans_model_tuned' not in globals():
         return "Error: Clustering recommendation components not fully␣
↪available."

    try:
        # The query_clustering_recommendations function expects a text query,
        # the dataframe, tfidf, encoder, kmeans model, and potentially a␣
↪reducer.

        # Get the reducer if it was used and stored in globals()
        reducer_for_query = globals().get('reducer', None)

        recommendations_df = query_clustering_recommendations(
            query,
            merged_df,
            current_tfidf,
            current_encoder,
            kmeans_model_tuned,
            reducer=reducer_for_query,
            num_recommendations=10 # Define a default number of recommendations
        )

        if recommendations_df.empty:
            return "No Clustering recommendations found for this query."
        else:
            # Format the output as a string, including cluster ID and␣
↪similarity score if desired
            # Let's just return the titles for simplicity in the tool output
             return "Clustering Recommendations:\n" +␣
↪recommendations_df['title'].to_string(index=False)


    except Exception as e:
        return f"Error generating clustering recommendations: {e}"
```

```
print("\nRecommendation tools defined: get_content_recommendations_tool,␣
↪get_autoencoder_recommendations_tool, get_clustering_recommendations_tool")
```

```
## Defining Recommendation Tools


Recommendation tools defined: get_content_recommendations_tool,
get_autoencoder_recommendations_tool, get_clustering_recommendations_tool
```

7.1.3 Step 3: Design Agent Logic

Before implementing the agent, here is the design the logic it should follow to handle user requests. The agent will need to decide:

1. **How to interpret the user's intent:** Does the user want recommendations based on keywords, a specific movie title, or a desired sentiment/mood?
2. **When and how to use the Sentiment Detection Tool:** If the user's input expresses a sentiment (e.g., "happy movie", "intense thriller"), the agent should use the `detect_sentiment` tool to identify the sentiment.
3. **Recommendation Tool(s) to use:** Based on the interpreted intent and potentially the detected sentiment, the agent should select the most appropriate recommendation tool(s):
   - If the user mentions a specific movie title, the agent could use `get_content_recommendations_tool`, `get_autoencoder_recommendations_tool`, or even `get_clustering_recommendations_tool` with the title as the query.
   - If the user provides keywords describing content (e.g., "science fiction action"), `get_content_recommendations_tool` or `get_clustering_recommendations_tool` are suitable.
   - If the user expresses a strong sentiment preference (e.g., "I want a very positive movie"), the agent might prioritize using a model or filtering results based on sentiment scores (although our current tools primarily use content/query for lookup, the agent could potentially interpret sentiment and use it to refine the query or select a specific model if we had sentiment-aware models).
4. **How to combine information:** The agent might combine the detected sentiment with the original query when calling recommendation tools, or use sentiment to filter/rank results *after* getting initial recommendations.
5. **How to format the final response:** The agent should present the recommendations clearly to the user.

** Agent Flow:**

A possible flow could be:

- Receive user input.
- Analyze the input to see if it contains a specific movie title (e.g., using string matching or another tool if available).
- If a movie title is found, use `get_content_recommendations_tool` and/or `get_autoencoder_recommendations_tool` with the movie title.
- If no movie title is found, use `detect_sentiment` to get the sentiment of the input.

- Based on the original input (keywords) and detected sentiment, decide whether to use `get_content_recommendations_tool` or `get_clustering_recommendations_tool` with the original text query. (Note: Integrating sentiment directly into the recommendation query for content/clustering tools might require modifying those functions or adding extra logic within the agent to interpret sentiment and refine the query).
- Present the recommendations generated by the chosen tool(s).

[176]:
```
# @title Step 4: Implement the Agent

print("## Implementing the Agentic Flow\n")

from langchain.agents import AgentExecutor, create_tool_calling_agent
from langchain_openai import ChatOpenAI # Correct import from langchain-openai
from langchain.prompts import ChatPromptTemplate
# get_autoencoder_recommendations_tool, get_clustering_recommendations_tool)␣
↪are defined in previous cells.

# --- Check if prerequisite tool functions are defined ---
required_tool_functions = [
    'detect_sentiment',
    'get_content_recommendations_tool',
    'get_autoencoder_recommendations_tool',
    'get_clustering_recommendations_tool'
]

all_tools_defined = True
for func_name in required_tool_functions:
    if func_name not in globals():
        print(f"Error: Required tool function '{func_name}' is not defined.")
        all_tools_defined = False

if not all_tools_defined:
    print("\nPlease run the cells defining the sentiment detection tool (cell␣
↪6e475914) and recommendation tools (cell 8c77ca40) before running this cell.
↪")
    # Stop execution here gracefully if tools are missing
    raise NameError("Required tool functions are not defined. Cannot proceed␣
↪with agent creation.")


# --- Proceed with agent creation if all tools are defined ---
if all_tools_defined:
    # Ensure the LLM for the agent is initialized
    try:
        llm_agent = ChatOpenAI(temperature=0, model="gpt-4o")
        print("OpenAI LLM for agent initialized.")
    except Exception as e:
```

```
        print(f"Error initializing OpenAI LLM for agent: {e}")
        llm_agent = None # Set to None if initialization fails


    # List of tools available to the agent
    # The tools themselves have checks for component availability
    tools = [
        detect_sentiment,
        get_content_recommendations_tool,
        get_autoencoder_recommendations_tool,
        get_clustering_recommendations_tool
    ]

    # Define the agent's prompt
    # Modified prompt structure to a more standard format for tool-calling␣
↪agents
    # placing chat_history and agent_scratchpad after the human input message.
    agent_prompt = ChatPromptTemplate.from_messages([
        ("system", """You are a movie recommendation assistant. Your goal is to␣
↪help the user find movies they might like.
        You have access to tools to:
        1. Detect the sentiment of the user's input.
        2. Get movie recommendations based on content (keywords/query).
        3. Get movie recommendations based on Autoencoder latent features␣
↪(requires a specific movie title).
        4. Get movie recommendations based on Clustering/KMeans (keywords/
↪query).

        Based on the user's request, decide which tool(s) are most appropriate.
        - If the user expresses a strong sentiment about the *type* of movie␣
↪they want (e.g., "happy movie", "sad film"), use the sentiment detection␣
↪tool first. You can then use this sentiment to potentially refine your␣
↪recommendation approach or the query you pass to the recommendation tools.
        - If the user mentions a specific movie title, consider using the␣
↪Autoencoder tool or Content-based tool with that title.
        - If the user provides keywords or describes the content they like, use␣
↪the Content-based tool or Clustering tool with the keywords.
        - You can use multiple tools if needed to understand the request or␣
↪provide different perspectives on recommendations.
        - When using a recommendation tool, provide the necessary input (a␣
↪movie title for Autoencoder, a query for Content/Clustering).
        - After getting recommendations from a tool, present them clearly to␣
↪the user.
        - If no relevant tools are applicable or tools fail, inform the user␣
↪you cannot provide recommendations for that request.
        """),
```

```
        ("human", "{input}"), # Place the user input here
        ("placeholder", "{chat_history}"), # Place user chat history after␣
↪input for future reference
        ("placeholder", "{agent_scratchpad}"), # Place agent scratchpad␣
↪(internal thoughts) after chat history for future reference
    ])

    # Create the agent
    if llm_agent is not None:
        agent = create_tool_calling_agent(llm_agent, tools, agent_prompt)

        # Create the agent executor
        agent_executor = AgentExecutor(agent=agent, tools=tools, verbose=True)␣
↪# Set verbose=True to see agent's thought process
        print("\nAgent and AgentExecutor created.")
        print("We can now interact with the agent using agent_executor.
↪invoke({'input': 'User movie request here'})")

    else:
        print("\nAgent could not be created due to LLM initialization failure.")
```

```
## Implementing the Agentic Flow

OpenAI LLM for agent initialized.

Agent and AgentExecutor created.
We can now interact with the agent using agent_executor.invoke({'input': 'User
movie request here'})
```

[177]:
```
agent_executor.invoke({'input': 'I like adventure and romance movies, not␣
↪horror or sci-fi'})
```

```
> Entering new AgentExecutor chain…

Invoking: `get_content_recommendations_tool` with `{'query': 'adventure
romance'}`

--- Tool: Generating TF-IDF Content Recommendations for query: 'adventure
romance' ---
Movie 'adventure romance' not found in the dataset for content-based
recommendations.
```

```
No content-based recommendations found for this
query.
Invoking: `get_clustering_recommendations_tool` with `{'query': 'adventure
romance'}`

--- Tool: Generating Clustering Recommendations for query: 'adventure romance'
---
Applying reducer from 256d to 128d…
```

```
Clustering Recommendations:
           America Is Still the Place
                  Harrison Montgomery
                An Inconvenient Truth
    Iraq for Sale: The War Profiteers
Peace, Propaganda & the Promised Land
                          Roger & Me
                       Bending Steel
     The Man Who Shot Liberty Valance
            In the Shadow of the Moon
                                RizeI couldn't find any
content-based recommendations for "adventure romance," but here are some
recommendations based on clustering:
1. America Is Still the Place
2. Harrison Montgomery
3. An Inconvenient Truth
4. Iraq for Sale: The War Profiteers
5. Peace, Propaganda & the Promised Land
6. Roger & Me
7. Bending Steel
8. The Man Who Shot Liberty Valance
9. In the Shadow of the Moon
10. Rize
These movies might not all fit perfectly into the adventure and romance genres,
but they are related based on clustering analysis. Let me know if you need more
specific recommendations!

> Finished chain.
```

[177]:

```
{'input': 'I like adventure and romance movies, not horror or sci-fi',
 'output': 'I couldn\'t find any content-based recommendations for "adventure
romance," but here are some recommendations based on clustering:\n\n1. America
Is Still the Place\n2. Harrison Montgomery\n3. An Inconvenient Truth\n4. Iraq
for Sale: The War Profiteers\n5. Peace, Propaganda & the Promised Land\n6. Roger
& Me\n7. Bending Steel\n8. The Man Who Shot Liberty Valance\n9. In the Shadow of
```

```
the Moon\n10. Rize\n\nThese movies might not all fit perfectly into the
adventure and romance genres, but they are related based on clustering analysis.
Let me know if you need more specific recommendations!'}
```

7.1.4 Summary for Agentic flow

This feature presents an agentic movie recommendation system that leverages a Large Language Model (LLM) to deliver personalized and explainable suggestions. The system employs a multi-agent pipeline where each agent performs a specialized function to refine and rank movie choices.

The process begins with an LLM agent that interprets a user's natural language query to identify preferred genres. Following this, a content-scoring agent evaluates movies based on their alignment with these genres, while a popularity agent assesses them on collaborative metrics like vote counts. A blending agent then calculates a final hybrid score by combining the content and popularity scores through a weighted average, prioritizing the user's specific tastes.

Finally, after a recommendation agent selects the top-rated films, an explanation agent utilizes the LLM to generate a concise, human-like justification for each suggestion, linking it directly to the user's initial preferences. This agentic framework demonstrates a robust method for creating more interactive, transparent, and user-centric recommendation experiences.

7.2 Saving Models and Components for Deployment

To use the models and components developed in this notebook in a separate application or for deployment, we need to save their trained states or configurations to files. We will save the following:

1. **TF-IDF Vectorizer:** Needed to transform new text input into the same feature space used during training.
2. **Autoencoder Encoder Model:** The encoder part is needed to generate latent features for new movies.
3. **KMeans Model:** Needed to determine the cluster of a movie based on its latent features.
4. **Sentence-BERT Model:** The loaded model object is needed to encode new text into embeddings.
5. **`merged_df` (or relevant parts):** The DataFrame containing movie metadata is needed for looking up movie details for recommendations.
6. **`indices` Series:** The mapping from 'soup' text to DataFrame index is needed for efficient lookup in recommendation functions.

We will use `joblib` for saving scikit-learn objects (TF-IDF, KMeans) and TensorFlow's built-in save method for the Autoencoder model. The Sentence-BERT model object can also be saved, although loading it directly by name is often done. We'll save the `merged_df` and `indices` as CSV or pickle files.

[169]:
```python
# --- Save All Essential Models & Data for Frontend Integration ---

import os
import joblib
import pandas as pd

# Optionally, for SentenceTransformer or Keras models:
```

```
try:
    from sentence_transformers import SentenceTransformer
except ImportError:
    SentenceTransformer = None
try:
    from tensorflow import keras
except ImportError:
    keras = None

# ---- Step 1: Define Save Directory ----
save_dir = './recommendation_models'  # Change as needed
os.makedirs(save_dir, exist_ok=True)
print(f"Saving models to directory: {save_dir}")

# ---- Step 2: Save TF-IDF Vectorizer ----
if 'tfidf' in globals():
    tfidf_path = os.path.join(save_dir, 'tfidf_vectorizer.joblib')
    print("vocab size:", len(current_tfidf.vocabulary_))  # should be 86621
    print(current_tfidf.get_params())
    joblib.dump(current_tfidf, tfidf_path)
    print(f"Saved TF-IDF Vectorizer to {tfidf_path}")
else:
    print("TF-IDF Vectorizer not found. Skipping save.")

# ---- Step 3: Save Autoencoder Encoder Model ----
if 'current_encoder' in globals() and keras is not None:
    encoder_path = os.path.join(save_dir, 'autoencoder_encoder_model.keras')
    current_encoder.save(encoder_path)
    print(f"Saved Autoencoder Encoder Model to {encoder_path}")
else:
    print("Autoencoder Encoder Model not found or Keras not installed. Skipping␣
↪save.")

# ---- Step 4: Save KMeans Model ----
if 'kmeans_model_tuned' in globals():
    kmeans_path = os.path.join(save_dir, 'kmeans_model.joblib')
    joblib.dump(kmeans_model_tuned, kmeans_path)
    print(f"Saved KMeans Model to {kmeans_path}")
else:
    print("KMeans Model not found. Skipping save.")

# ---- Step 5: Save Sentence-BERT or Embedding Model ----
if 'model' in globals():
    sentence_bert_path = os.path.join(save_dir, 'sentence_bert_model')
    # If it's a SentenceTransformer, save using its .save() method
    if SentenceTransformer is not None and isinstance(model,␣
↪SentenceTransformer):
```

```
        model.save(sentence_bert_path)
        print(f"Saved Sentence-BERT Model to {sentence_bert_path}")
    # Otherwise, try saving as a Keras model (optional fallback)
    elif keras is not None and hasattr(model, "save"):
        model.save(sentence_bert_path)
        print(f"Saved model using Keras save() to {sentence_bert_path}")
    else:
        print("Model type unknown for 'model'. Not saved.")
else:
    print("Sentence-BERT Model not found. Skipping save.")

# ---- Step 6: Save Merged Movie Data (with essential columns) ----
if 'merged_df' in globals() and isinstance(merged_df, pd.DataFrame) and not␣
 ↪merged_df.empty:
    essential_cols = [
        'id', 'title', 'genres', 'keywords', 'overview',
        'overview_sentiment_score', 'soup', 'enhanced_soup'
    ]
    cols_to_save = [col for col in essential_cols if col in merged_df.columns]
    if cols_to_save:
        merged_df_path = os.path.join(save_dir, 'merged_movie_data.csv')
        merged_df[cols_to_save].to_csv(merged_df_path, index=False)
        print(f"Saved essential merged_df columns to {merged_df_path}")
    else:
        print("No essential columns found in merged_df to save.")
else:
    print("merged_df not found or is empty. Skipping save.")

# ---- Step 7: Save Indices Series (for mapping in frontend/backend) ----
if 'indices' in globals() and hasattr(indices, 'empty') and not indices.empty:
    indices_path = os.path.join(save_dir, 'indices.pkl')
    indices.to_pickle(indices_path)
    print(f"Saved indices Series to {indices_path}")
else:
    print("indices Series not found or is empty. Skipping save.")


print("\nSaving process complete.")
print(f"All files are available in: {save_dir}")
print("To use these models in a separate application, load them using the␣
 ↪respective libraries (joblib.load, keras.models.load_model,␣
 ↪SentenceTransformer, pd.read_csv, pd.read_pickle, etc.).")
```

```
Saving models to directory: ./recommendation_models
vocab size: 86621
{'analyzer': 'word', 'binary': False, 'decode_error': 'strict', 'dtype': <class
'numpy.float64'>, 'encoding': 'utf-8', 'input': 'content', 'lowercase': True,
```

```
'max_df': 1.0, 'max_features': 86621, 'min_df': 1, 'ngram_range': (1, 2),
'norm': 'l2', 'preprocessor': None, 'smooth_idf': True, 'stop_words': None,
'strip_accents': None, 'sublinear_tf': False, 'token_pattern':
'(?u)\\b\\w\\w+\\b', 'tokenizer': None, 'use_idf': True, 'vocabulary': None}
Saved TF-IDF Vectorizer to ./recommendation_models/tfidf_vectorizer.joblib
Saved Autoencoder Encoder Model to
./recommendation_models/autoencoder_encoder_model.keras
Saved KMeans Model to ./recommendation_models/kmeans_model.joblib
Sentence-BERT Model not found. Skipping save.
Saved essential merged_df columns to
./recommendation_models/merged_movie_data.csv
Saved indices Series to ./recommendation_models/indices.pkl

Saving process complete.
All files are available in: ./recommendation_models
To use these models in a separate application, load them using the respective
libraries (joblib.load, keras.models.load_model, SentenceTransformer,
pd.read_csv, pd.read_pickle, etc.).
```