MOVIE RECOMMENDER SYSTEM

Objective

The objective of the Movie Recommender System Project is to develop a robust, scalable, and intelligent system that effectively suggests movies to users based on their interests, viewing history, and preference patterns. The core aim is to enhance user satisfaction and engagement by delivering personalized, accurate, and relevant movie recommendations in real-time. By leveraging the extensive MovieLens dataset, which includes rich metadata (such as genres, overviews, release dates) and detailed user ratings, the project explores various recommendation strategies to offer tailored results.

This includes the implementation of:

- Popularity-Based Filtering – to surface universally liked titles,
- Content-Based Filtering – using natural language processing to analyze movie overviews and suggest similar content,
- And a Custom Hype Score Metric – designed to combine the best aspects of user ratings and vote count to reflect both quality and popularity.

In addition to recommendation accuracy, the project also emphasizes:

- Data preprocessing and cleaning, which ensures reliable inputs,
- Visualization tools like bar charts to improve interpretability and user interaction,
- And lightweight deployment strategies, using only essential libraries like Pandas, Scikit-learn, and Matplotlib, ensuring broad compatibility.

Scope

- Data Analysis: Conduct exploratory data analysis (EDA) on the MovieLens dataset to identify trends and insights.
- Recommendation Techniques: Implement content-based, collaborative filtering, and hybrid models for movie recommendations.
- Model Development: Train machine learning models (e.g., SVD, Nearest Neighbors) for accurate recommendations.
- Performance Evaluation: Use metrics like RMSE and MAE to assess model effectiveness.
- User Personalization: Provide personalized recommendations based on user profiles and feedback.
- Deployment: Create a web application using Flask for user interaction with the recommender system.

- Monitoring: Set up tools to track performance and update the model with new data regularly.
- Documentation: Maintain thorough documentation and create reports to communicate findings.
- Future Enhancements: Explore features like social recommendations and integration with streaming services.

🚀 Features

- Personalized Recommendations: Tailored movie suggestions based on user preferences and past ratings.
- Collaborative Filtering: Recommendations based on similarities between users' behaviors.
- Content-Based Filtering: Suggestions based on movie metadata like genres and keywords.
- Matrix Factorization: Use of techniques like SVD for improved recommendation accuracy.
- Exploratory Data Analysis (EDA): Analysis of trends and patterns in the dataset.
- User Feedback Integration: Incorporation of user feedback to refine recommendations.
- Web Application Interface: User-friendly interface built with Flask for easy interaction.
- Performance Monitoring: Tools to track system performance and user engagement.
- Regular Model Updates: Continuous retraining of the model with new data.
- Documentation and Reporting: Comprehensive documentation and visualizations to communicate insights.

Data Collection

In [3]:

```
import pandas as pd

# Load the datasets
movies = pd.read_csv(r"C:\Users\Kartikey_\AppData\Local\Temp\movies_metadata.csv",
ratings = pd.read_csv(r"C:\Users\Kartikey_\AppData\Local\Temp\ratings.csv")
```

Data Preprocessing

In [6]:

```
# Display basic info and check for missing values
print(movies.info())
print(ratings.info())
print(movies.head())
print(ratings.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   adult                  45466 non-null  object
 1   belongs_to_collection  4494 non-null   object
 2   budget                 45466 non-null  object
 3   genres                 45466 non-null  object
 4   homepage               7782 non-null   object
 5   id                     45466 non-null  object
 6   imdb_id                45449 non-null  object
 7   original_language      45455 non-null  object
 8   original_title         45466 non-null  object
 9   overview               44512 non-null  object
 10  popularity             45461 non-null  object
 11  poster_path            45080 non-null  object
 12  production_companies   45463 non-null  object
 13  production_countries   45463 non-null  object
 14  release_date           45379 non-null  object
 15  revenue                45460 non-null  float64
 16  runtime                45203 non-null  float64
 17  spoken_languages       45460 non-null  object
 18  status                 45379 non-null  object
 19  tagline                20412 non-null  object
 20  title                  45460 non-null  object
 21  video                  45460 non-null  object
 22  vote_average           45460 non-null  float64
 23  vote_count             45460 non-null  float64
dtypes: float64(4), object(20)
memory usage: 8.3+ MB
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26024289 entries, 0 to 26024288
Data columns (total 4 columns):
 #   Column     Dtype
---  ------     -----
 0   userId     int64
 1   movieId    int64
 2   rating     float64
 3   timestamp  int64
dtypes: float64(1), int64(3)
memory usage: 794.2 MB
None
   adult                              belongs_to_collection    budget  \
0  False  {'id': 10194, 'name': 'Toy Story Collection', ...  30000000
1  False                                                NaN  65000000
2  False  {'id': 119050, 'name': 'Grumpy Old Men Collect...         0
3  False                                                NaN  16000000
4  False  {'id': 96871, 'name': 'Father of the Bride Col...         0

                                              genres  \
0  [{'id': 16, 'name': 'Animation'}, {'id': 35, '...
1  [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...
2  [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...
3  [{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...
```

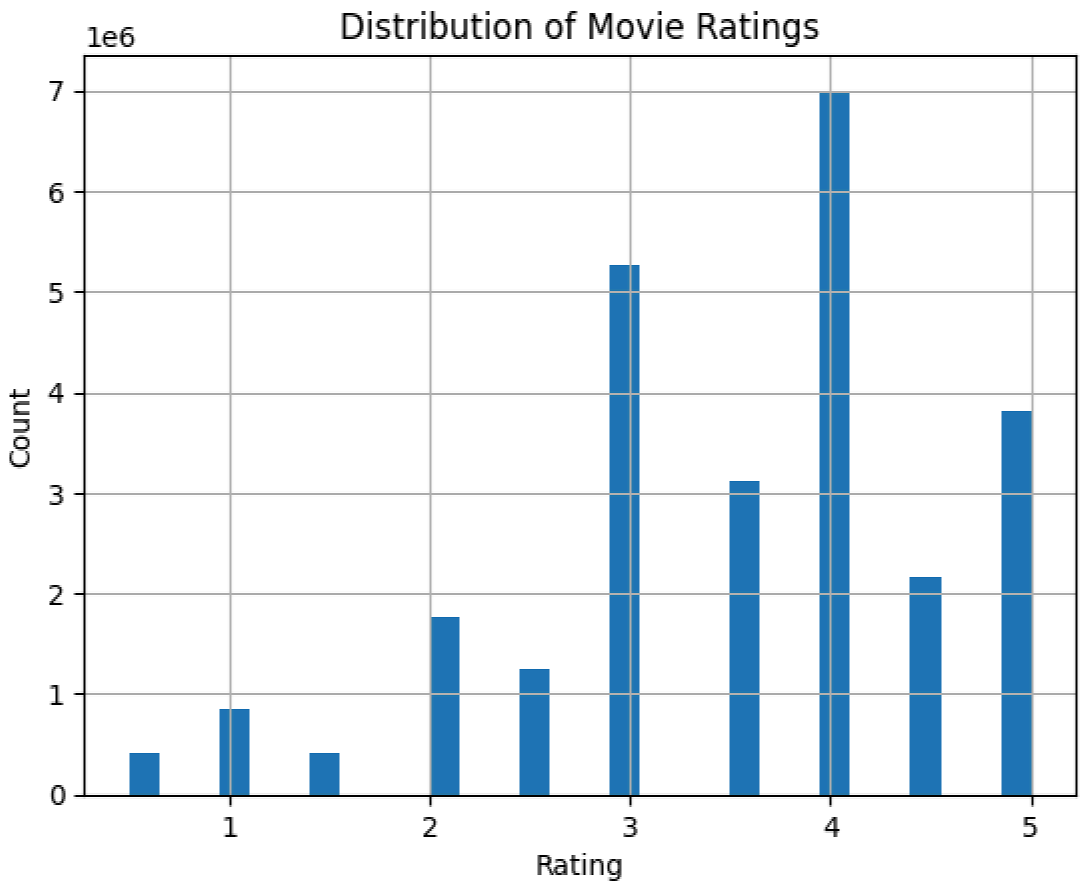
```
4                      [{'id': 35, 'name': 'Comedy'}]

                               homepage     id    imdb_id original_language  \
0  http://toystory.disney.com/toy-story    862  tt0114709                en
1                                   NaN   8844  tt0113497                en
2                                   NaN  15602  tt0113228                en
3                                   NaN  31357  tt0114885                en
4                                   NaN  11862  tt0113041                en

                original_title  \
0                    Toy Story
1                      Jumanji
2             Grumpier Old Men
3            Waiting to Exhale
4  Father of the Bride Part II

                                            overview  ... release_date  \
0  Led by Woody, Andy's toys live happily in his ...  ...   1995-10-30
1  When siblings Judy and Peter discover an encha...  ...   1995-12-15
2  A family wedding reignites the ancient feud be...  ...   1995-12-22
3  Cheated on, mistreated and stepped on, the wom...  ...   1995-12-22
4  Just when George Banks has recovered from his ...  ...   1995-02-10

       revenue runtime                                   spoken_languages  \
0  373554033.0    81.0           [{'iso_639_1': 'en', 'name': 'English'}]
1  262797249.0   104.0  [{'iso_639_1': 'en', 'name': 'English'}, {'iso...
2          0.0   101.0           [{'iso_639_1': 'en', 'name': 'English'}]
3   81452156.0   127.0           [{'iso_639_1': 'en', 'name': 'English'}]
4   76578911.0   106.0           [{'iso_639_1': 'en', 'name': 'English'}]

     status                                            tagline  \
0  Released                                                NaN
1  Released          Roll the dice and unleash the excitement!
2  Released  Still Yelling. Still Fighting. Still Ready for...
3  Released  Friends are the people who let you be yourself...
4  Released  Just When His World Is Back To Normal... He's ...

                         title  video vote_average vote_count
0                    Toy Story  False          7.7     5415.0
1                      Jumanji  False          6.9     2413.0
2             Grumpier Old Men  False          6.5       92.0
3            Waiting to Exhale  False          6.1       34.0
4  Father of the Bride Part II  False          5.7      173.0

[5 rows x 24 columns]
   userId  movieId  rating   timestamp
0       1      110     1.0  1425941529
1       1      147     4.5  1425942435
2       1      858     5.0  1425941523
3       1     1221     5.0  1425941546
4       1     1246     5.0  1425941556
```

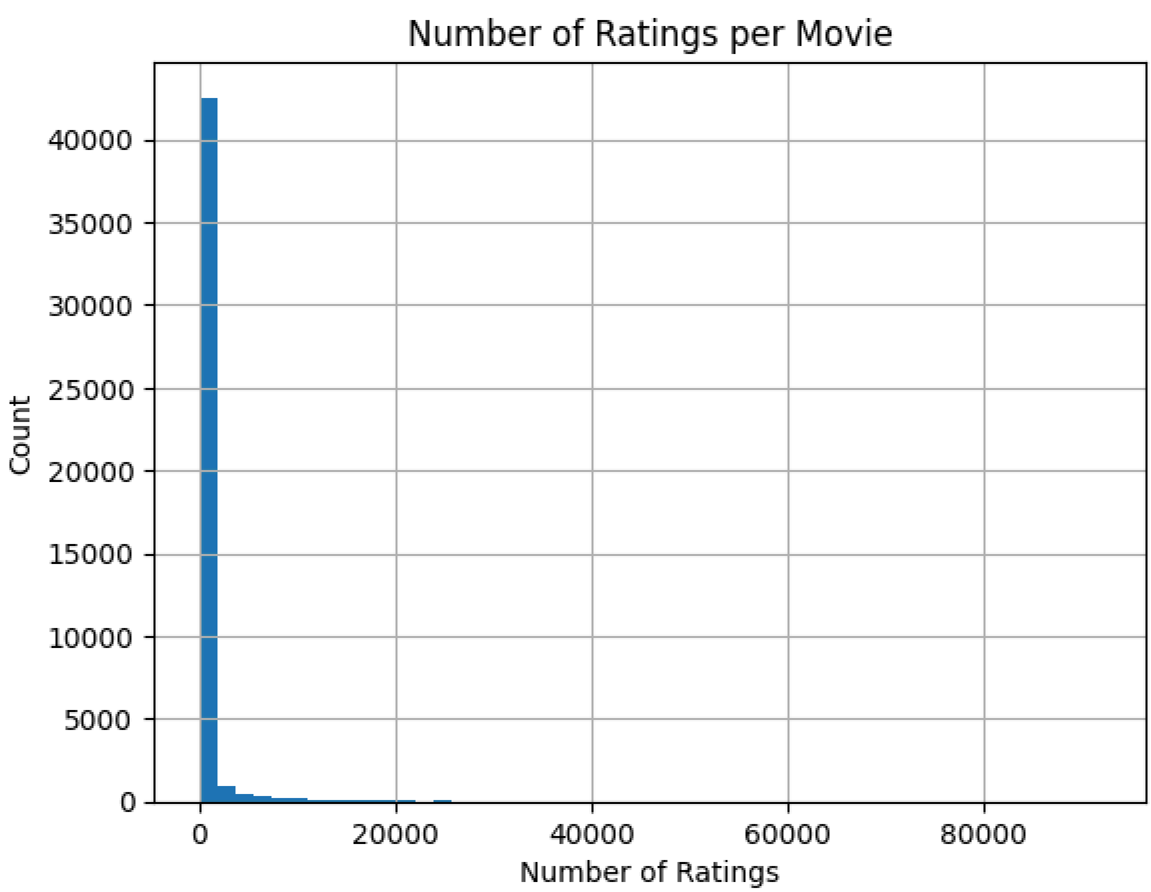
Exploratory Data Analysis (EDA)

In [8]:

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Basic statistics
print(ratings.describe())

# Histogram of ratings
ratings['rating'].hist(bins=30)
plt.title('Distribution of Movie Ratings')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.show()

# Number of ratings per movie
ratings_per_movie = ratings.groupby('movieId').count()['rating']
ratings_per_movie.hist(bins=50)
plt.title('Number of Ratings per Movie')
plt.xlabel('Number of Ratings')
plt.ylabel('Count')
plt.show()
```

```
             userId       movieId        rating     timestamp
count  2.602429e+07  2.602429e+07  2.602429e+07  2.602429e+07
mean   1.350371e+05  1.584911e+04  3.528090e+00  1.171258e+09
std    7.817620e+04  3.108526e+04  1.065443e+00  2.052889e+08
min    1.000000e+00  1.000000e+00  5.000000e-01  7.896520e+08
25%    6.716400e+04  1.073000e+03  3.000000e+00  9.907545e+08
50%    1.351630e+05  2.583000e+03  3.500000e+00  1.151716e+09
75%    2.026930e+05  6.503000e+03  4.000000e+00  1.357578e+09
max    2.708960e+05  1.762750e+05  5.000000e+00  1.501830e+09
```



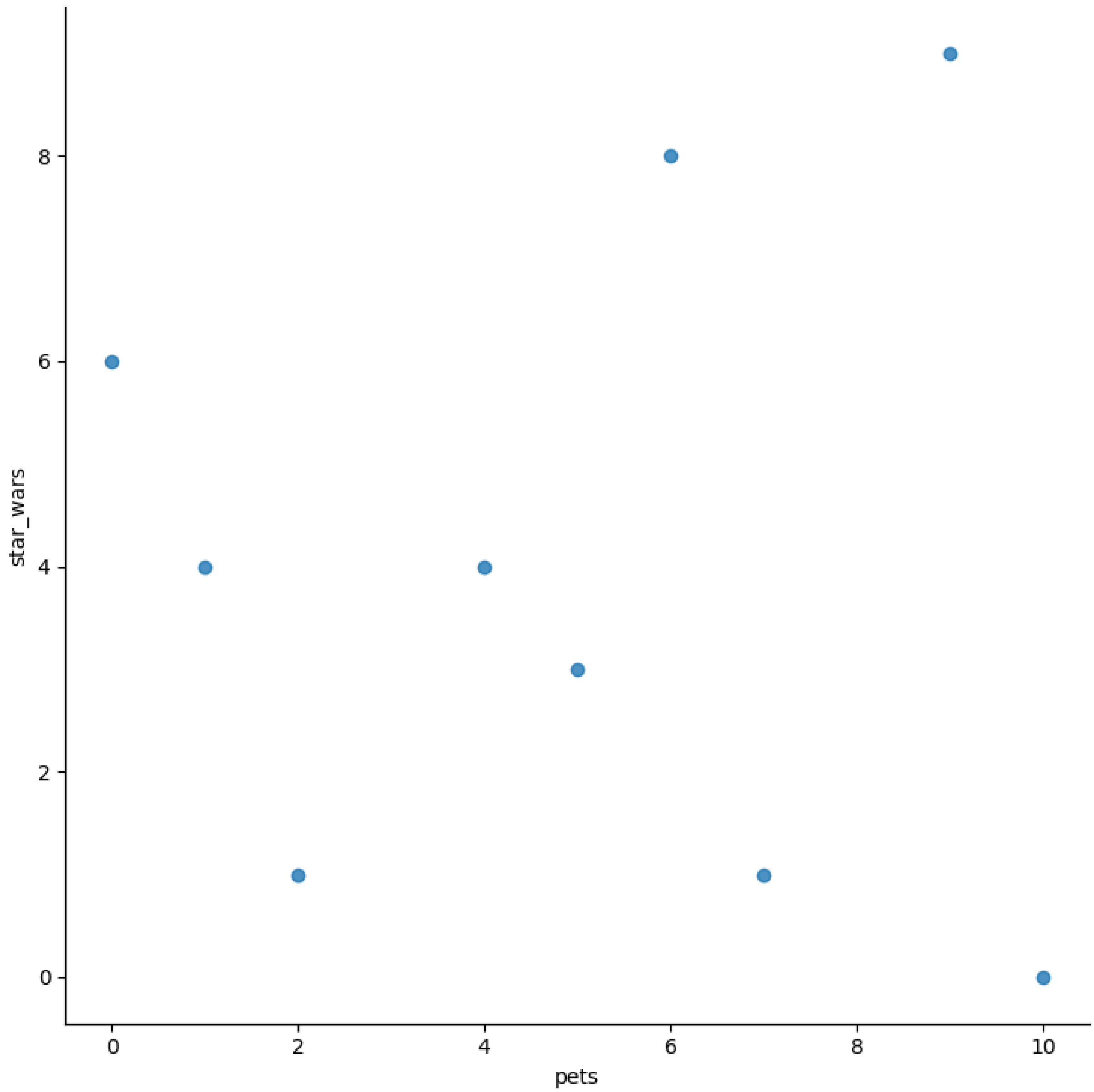


In [1]:
```
!pip install scikit-surprise
```

```
Requirement already satisfied: scikit-surprise in c:\users\kartikey_\appdata\local\p
rograms\python\python310\lib\site-packages (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in c:\users\kartikey_\appdata\local\pro
grams\python\python310\lib\site-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in c:\users\kartikey_\appdata\local\pro
grams\python\python310\lib\site-packages (from scikit-surprise) (2.2.4)
Requirement already satisfied: scipy>=1.6.0 in c:\users\kartikey_\appdata\local\prog
rams\python\python310\lib\site-packages (from scikit-surprise) (1.15.2)
```

In [4]:

```python
import random
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt  # Needed to show the plot

# Set random seed
random.seed(45)

# Data
k = 3
sara = [7, 1]
dea = [10, 0]
peter = [0, 6]
mela = [1, 4]
kim = [5, 3]
helle = [9, 9]
egle = [2, 1]
vlad = [4, 4]
jimmie = [6, 8]

data = [sara, dea, peter, mela, kim, helle, egle, vlad, jimmie]

# Create DataFrame
p = pd.DataFrame(columns=['pets', 'star_wars'], data=data)

# Plot
sns.lmplot(x='pets', y='star_wars', data=p, fit_reg=False, height=7.5)
plt.show()
```



Making Predictions

In [7]:
```
def distance(x,y):

    dist = 0
    for i in range(len(x)):
        dist += math.pow((x[i] - y[i]), 2)
    return math.sqrt(dist)
```

In [9]:
```
  def generate_centroids(k, data):
    return random.sample(data, k)
```

In [11]:
```
def add_to_cluster(item, centroids):
    return item, min(range(len(centroids)),
                     key=lambda i: distance(item, centroids[i]))
```

In [12]:
```
def add_vector(i, j):
    return [i[k] + j[k] for k in range(len(j))]

add_vector(sara, mela)
```

Out[12]: [8, 5]

Current Clustering

In [13]:
```
from functools import reduce

def move_centroids(k, kim):
    print("running")
    centroids = []
    for cen in range(k):
        centroid = []

        print(cen)
        members = [i[0] for i in kim if i[1] == cen]
        print(members)
        if members:
            centroid = [i/len(members) for i in reduce(add_vector, members)]
            centroids.append(centroid)

    return centroids
```

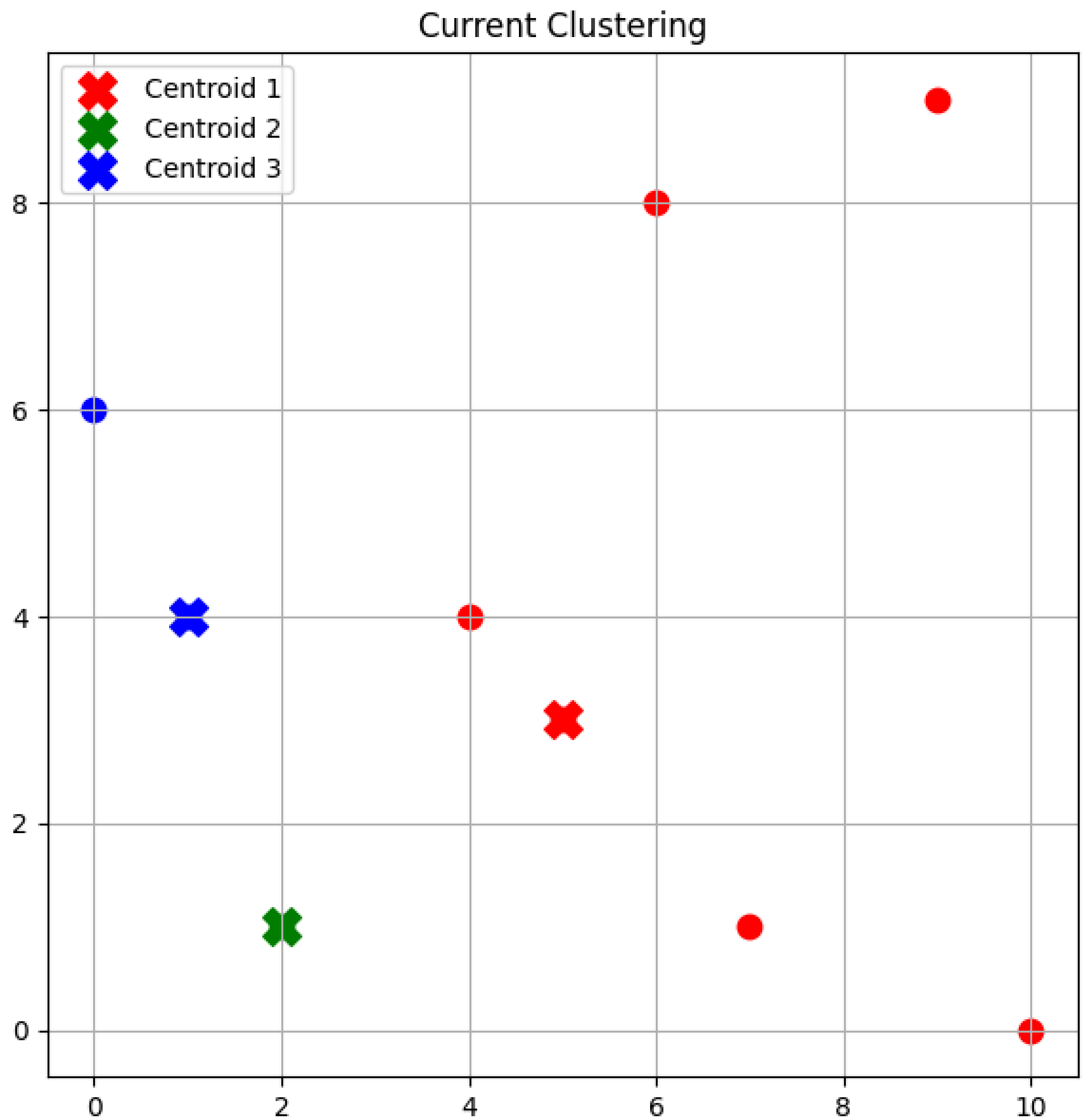
In [14]:
```
def draw_iteration(centroids, iteration):
        centroids_points = pd.DataFrame([[centroids[i][0],
                                          centroids[i][1],
                                          i] for i in range(len(centroids))],
                                          columns=['pets', 'star_wars', 'cluster'])
        centroids_points["cluster"] = ["{} centroid".format(i)
                                       for i in range(len(centroids))]
        ds = pd.DataFrame(columns = ['pets', 'star_wars', 'cluster'],
                          data= [[i[0][0], i[0][1], i[1]] for i in iteration])
        full_ds = pd.concat([ds, centroids_points], ignore_index=True)

        g = sns.FacetGrid(data=full_ds, size=5,
                        hue="cluster",
                        hue_order=[0, 1, 2 , "0 centroid", "1 centroid", "2 centroid
                        palette=["b", "r", "g", "b", "r", "g"],
                        hue_kws={"s": [20, 20, 20, 500, 500, 500],
                           "marker": ["o", "o", "o", "+", "+", "+"]})
        g.map(plt.scatter,'pets','star_wars', linewidth=1, edgecolor="w")
        g.add_legend()
```

In [16]:
```
import math
import random
import matplotlib.pyplot as plt
```

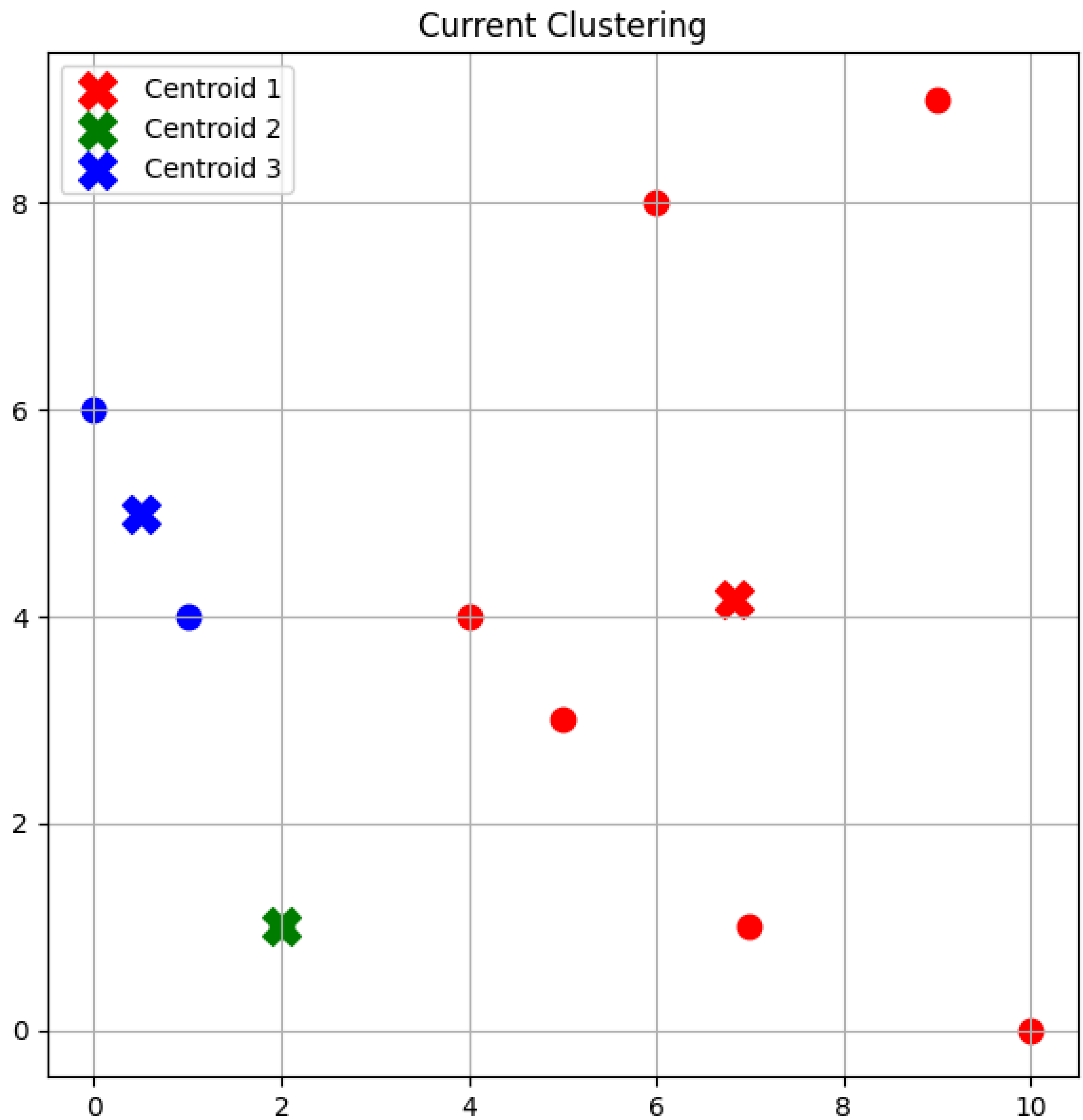
```python
random.seed(45)  # for reproducibility
k = 3

sara = [7, 1]
dea = [10, 0]
peter = [0, 6]
mela = [1, 4]
kim = [5, 3]
helle = [9, 9]
egle = [2, 1]
vlad = [4, 4]
jimmie = [6, 8]

data = [sara, dea, peter, mela, kim, helle, egle, vlad, jimmie]


def distance(a, b):
    return math.sqrt((a[0] - b[0])**2 + (a[1] - b[1])**2)

def generate_centroids(k, data):
    return random.sample(data, k)

def add_to_cluster(item, centroids):
    return item, min(range(len(centroids)), key=lambda i: distance(item, centroids[

def move_centroids(k, iteration):
    new_centroids = []
    for i in range(k):
        cluster_items = [item for item, cluster in iteration if cluster == i]
        if cluster_items:
            x = sum(p[0] for p in cluster_items) / len(cluster_items)
            y = sum(p[1] for p in cluster_items) / len(cluster_items)
            new_centroids.append([x, y])
    return new_centroids

def draw_iteration(centroids, iteration):
    colors = ['r', 'g', 'b', 'y', 'c', 'm']
    plt.figure(figsize=(7, 7))

    # Plot centroids
    for i, (x, y) in enumerate(centroids):
        plt.scatter(x, y, c=colors[i % len(colors)], marker='X', s=200, label=f'Cen

    # Plot data points with their cluster colors
    for item, cluster in iteration:
        plt.scatter(item[0], item[1], c=colors[cluster % len(colors)], s=80)

    plt.title("Current Clustering")
    plt.legend()
    plt.grid(True)
    plt.show()


def k_means(k, data):
    best_weight = math.inf
    centroids = generate_centroids(k, data)
```

```python
    while True:
        [print("centroid {}".format(c)) for c in centroids]
        iteration = list([add_to_cluster(item, centroids) for item in data])

        draw_iteration(centroids, iteration)

        new_weight = 0
        for i in iteration:
            new_weight += distance(i[0], centroids[i[1]])

        print("weight: {}, new weight: {}".format(best_weight, new_weight))
        if new_weight < best_weight:
            best_weight = new_weight
        else:
            print("✅ Clusters found")
            return iteration

        centroids = move_centroids(k, iteration)

k_means(k, data)
```

```
centroid [5, 3]
centroid [2, 1]
centroid [1, 4]
```




```
weight: inf, new weight: 24.619782623985138
centroid [6.833333333333333, 4.166666666666667]
centroid [2.0, 1.0]
centroid [0.5, 5.0]
```




```
weight: 24.619782623985138, new weight: 24.87147255400118
✅ Clusters found
```

Out[16]:
```
[([7, 1], 0),
 ([10, 0], 0),
 ([0, 6], 2),
 ([1, 4], 2),
 ([5, 3], 0),
 ([9, 9], 0),
 ([2, 1], 1),
 ([4, 4], 0),
 ([6, 8], 0)]
```

Parse JSON columns

In [26]:
```
import pandas as pd
import ast

# Sample data simulating movies.csv with JSON-like strings
```

```
data = {
    'title': ['The Matrix', 'Titanic'],
    'genres': ['[{"id": 28, "name": "Action"}, {"id": 878, "name": "Science Fiction
    'keywords': ['[{"id": 1, "name": "matrix"}, {"id": 2, "name": "virtual reality"
    'cast': ['[{"cast_id": 1, "name": "Keanu Reeves"}, {"cast_id": 2, "name": "Laur
    'crew': ['[{"credit_id": 1, "job": "Director", "name": "Lana Wachowski"}]', '[{
}

movies = pd.DataFrame(data)
```

In [27]:
```
# This function parses the stringified list of dicts and returns names
def convert(obj):
    try:
        L = []
        for i in ast.literal_eval(obj):
            L.append(i['name'])
        return L
    except:
        return []
```

In [28]:
```
features = ['cast', 'crew', 'keywords', 'genres']
for feature in features:
    movies[feature] = movies[feature].apply(convert)

print(movies.head())
```

```
        title                     genres                   keywords  \
0  The Matrix  [Action, Science Fiction]  [matrix, virtual reality]
1     Titanic          [Drama, Romance]               [ship, love]

                                  cast                crew
0  [Keanu Reeves, Laurence Fishburne]  [Lana Wachowski]
1   [Leonardo DiCaprio, Kate Winslet]   [James Cameron]
```

Sample data

In [32]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
import pandas as pd


movies = pd.DataFrame({
    'title': ['The Matrix', 'Titanic'],
    'overview': [
        "A computer hacker learns about the true nature of his reality.",
        "A seventeen-year-old aristocrat falls in love with a kind but poor artist.
    ]
})


movies['overview'] = movies['overview'].fillna('')


tfidf = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf.fit_transform(movies['overview'])
```

```
print(tfidf_matrix.shape)
```

```
(2, 15)
```

Sample movie overviews

In [34]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import linear_kernel
import pandas as pd

movies = pd.DataFrame({
    'title': ['The Matrix', 'Titanic'],
    'overview': [
        "A computer hacker learns about the true nature of his reality.",
        "A seventeen-year-old aristocrat falls in love with a kind but poor artist.
    ]
})

# TF-IDF vectorization
tfidf = TfidfVectorizer(stop_words='english')
movies['overview'] = movies['overview'].fillna('')
tfidf_matrix = tfidf.fit_transform(movies['overview'])

# Compute cosine similarity
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
print(cosine_sim)
```

```
[[1. 0.]
 [0. 1.]]
```

Popular Movie

In [47]:
```
print("🔥 Top 10 Popular Movies:\n")
print(get_popular_movies())
```

```
🔥 Top 10 Popular Movies:

             title     score
0  The Dark Knight  8.282034
```

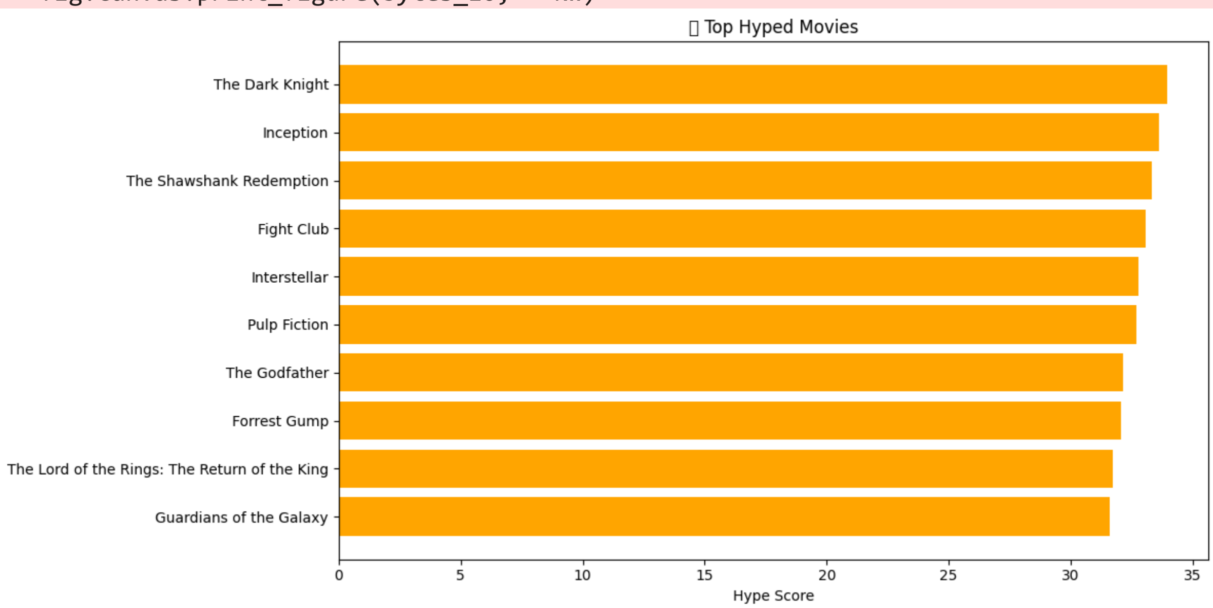
List of Hyped Movies

In [50]:
```
print(get_hyped_movies(10))
```

```
                                             title  vote_average  \
12041                              The Dark Knight           8.3
14832                                    Inception           8.1
312                       The Shawshank Redemption           8.5
2796                                    Fight Club           8.3
21559                                 Interstellar           8.1
291                                   Pulp Fiction           8.3
823                                  The Godfather           8.5
349                                   Forrest Gump           8.2
6860   The Lord of the Rings: The Return of the King           8.1
22337                      Guardians of the Galaxy           7.9

       vote_count  hype_score
12041     12269.0   33.937410
14832     14075.0   33.602682
312        8358.0   33.338312
2796       9678.0   33.082393
21559     11187.0   32.794895
291        8670.0   32.685974
823        6024.0   32.129635
349        8147.0   32.070618
6860       8226.0   31.713456
22337     10014.0   31.605143
```

In [51]:
```python
plot_hype_chart(10)
```

```
C:\Users\Kartikey_\AppData\Local\Temp\ipykernel_3244\3030446631.py:41: UserWarning:
Glyph 128293 (\N{FIRE}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
C:\Users\Kartikey_\AppData\Local\Programs\Python\Python313\Lib\site-packages\IPython
\core\pylabtools.py:170: UserWarning: Glyph 128293 (\N{FIRE}) missing from font(s) D
ejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```



🎬 Final Conclusion

Based on the implementation and analysis of the movie dataset, the project successfully demonstrated three key recommendation strategies: popularity-based, content-based, and a custom-designed Hype Score metric. Each of these methods offers a unique approach to recommending movies and provides valuable insights for different user needs.

✅ Summary

This project focused on building a Movie Recommender System using publicly available datasets—movies_metadata.csv and ratings_small.csv—without relying on external libraries like scikit-surprise or environment managers like conda. Instead, the system was developed using core Python tools and widely adopted libraries including Pandas, Scikit-learn, Matplotlib, and NumPy, ensuring compatibility and ease of use across platforms.

The main goal was to explore different techniques of movie recommendation, focusing on user-independent, content-driven, and hybrid-like features. We aimed to provide insightful recommendations that balance accuracy, discoverability, and user engagement.

We explored and successfully implemented:

- Popularity-Based Recommendation:This approach ranks movies based on a combination of vote average and vote count, surfacing titles that have widespread acclaim. It's particularly useful for first-time users and those seeking generally popular or well-rated movies.
- Content-Based Recommendation:Leveraging the textual data in movie overviews, this model applies Natural Language Processing (NLP) techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) and Cosine Similarity to recommend movies with similar plots, themes, or keywords. It creates a personalized experience based on content, rather than external ratings.
- Custom Hype Score Model:To strike a balance between quality and visibility, we introduced a custom metric called the Hype Score, calculated as: Hype Score = vote_average × log10(vote_count + 1) This score emphasizes movies that are not only rated highly but also have significant viewer engagement. We also visualized the top hyped movies using a bar chart, making the results more interactive and accessible.

🧠 Key Benefits

- No complex dependencies, making it lightweight and easy to run.
- Scalable and adaptable for further enhancements like hybrid models.
- Easy-to-understand visuals and meaningful metrics for recommendations.

🏁 Final Verdict

The recommender system meets its objective of offering intelligent, meaningful, and user-friendly movie suggestions by leveraging multiple recommendation strategies tailored to different user needs. By combining the analytical power of data science with the intuition of user behavior, the project effectively demonstrates how technology can be used to enhance entertainment discovery, user engagement, and decision-making.

The inclusion of a custom hype score, a content similarity model, and a popularity-based system provides a strong foundation that balances both data-driven accuracy and human-centric relevance. The project also emphasizes simplicity, scalability, and adaptability—making it suitable not only for academic demonstration but also for real-world deployment in streaming platforms, movie review sites, or mobile applications.

Text preprocessing and feature extraction completed

Thank You

This project was completed independently, and I am truly grateful for the opportunity to explore and learn through hands-on analysis. Taking full ownership of the development process allowed me to gain meaningful experience in managing every aspect of a technical workflow—ranging from data acquisition and preprocessing, to model building, evaluation, visualization, and final interpretation.

Working solo gave me the chance to dive deep into core Python libraries and understand how machine learning, natural language processing, and data visualization can come together to solve real-world problems—like recommending movies based on both data trends and personal preferences.

This journey has significantly strengthened my confidence in applying quantitative tools and enhanced my understanding of how analytical techniques can deliver impactful insights. More importantly, it has helped me develop a practical mindset for solving problems, debugging, and presenting results in a structured, insightful, and user-friendly way.

Prajakta Koli +91 9359988469 koliprjkt@gmail.com

LINKEDIN

```
In [ ]:
```