Activity based

Project Report on

Artificial Intelligence

Project Module - I

Submitted to Vishwakarma University, Pune

Under the Initiative of

Contemporary Curriculum, Pedagogy, and Practice (C2P2)

By

Shyamal Sagar Patil

SRN No: 202200930

Roll No: 38

Div: B

Third Year Engineering

Department of Computer Engineering

Faculty of Science and Technology

Academic Year

2023-2024

Artificial Intelligence: Project Module I
Project Name: Smart Movie Recommendation System Using Best-First Search

Problem Statement:

Design an AI-based recommendation system using the Best-First Search algorithm which focuses on creating a personalized recommendation engine that intelligently suggests movies based on user preferences.

Description:

The recommendation system aims to provide personalized movie recommendations to users based on their preferences and past interactions. Here's a detailed description of how the system works:

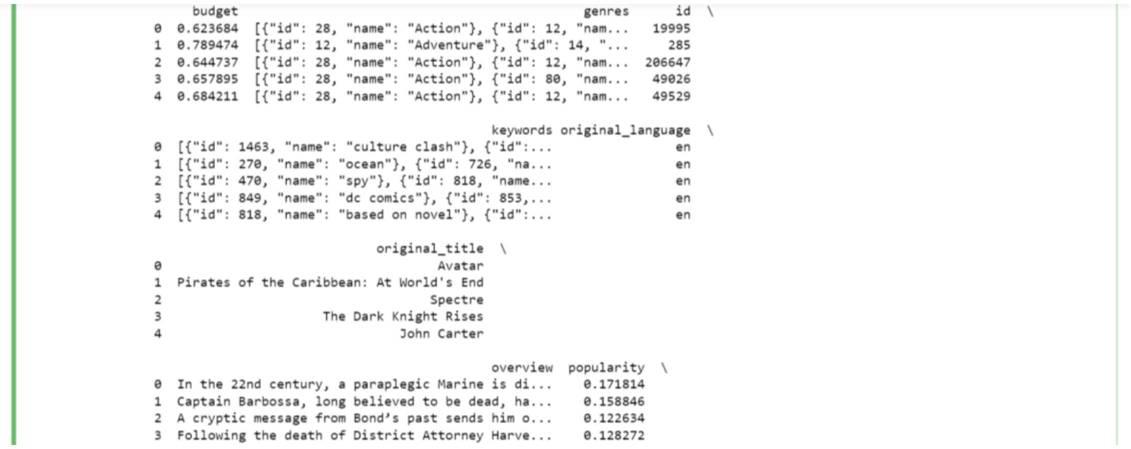
1. **User Profile Creation:**
    - When a user interacts with the system for the first time, they are prompted to create a user profile.
    - The user profile includes information such as preferred movie genres, ratings, and any specific preferences the user may have.
    - Users can update their profiles at any time to reflect changes in their preferences or interests.
2. **Data Preprocessing:**
    - The movie dataset is preprocessed to extract relevant features such as genres, ratings, popularity, release date, and more.
    - Missing or invalid data is handled appropriately through techniques like data imputation or removal of incomplete entries.
    - Feature scaling and normalization may be applied to ensure that different features contribute equally to the recommendation process.
3. **Best-First Search Algorithm:**
    - The recommendation system utilizes the Best-First Search algorithm to efficiently traverse the movie graph and identify relevant movies.
    - The algorithm prioritizes movies that are most likely to match the user's preferences based on a heuristic function.
4. **Heuristic Function:**
    - A heuristic function is used to evaluate the relevance of each movie to the user's preferences.
    - The function considers factors such as genre similarity, rating, popularity, release date, and any other relevant features.
    - By assigning heuristic scores to movies, the system can prioritize recommendations that are more likely to be of interest to the user.
5. **Caching and Optimization:**
    - Caching mechanisms are employed to store frequently accessed data, such as heuristic scores and user profiles, to reduce computation time and improve performance.
    - Optimization techniques, including parallel processing and incremental updates, are implemented to enhance the efficiency of the recommendation system.
6. **User Feedback Integration:**
    - The recommendation system incorporates user feedback mechanisms to refine user profiles and improve recommendation accuracy over time.
    - Users can provide feedback on recommended movies, allowing the system to learn from user

interactions and adapt its recommendations accordingly.

7. **Evaluation:**
    - The performance of the recommendation system is evaluated using various metrics, including precision, recall, and user satisfaction.
    - A/B testing and user studies may be conducted to assess the effectiveness of the system and gather insights for further improvements.
8. **User Interface:**
    - The recommendation system features a user-friendly interface that allows users to interact with the system easily.
    - Users can view recommended movies, explore additional details, and provide feedback through the interface.
9. **Scalability and Performance:**
    - The recommendation system is designed to be scalable and capable of handling large datasets efficiently.
    - Techniques such as distributed computing and cloud-based infrastructure may be employed to ensure optimal performance under varying workloads.
10. **Documentation and Reporting:**
    - The design, implementation, and evaluation of the recommendation system are thoroughly documented.
    - Clear reports and insights are provided to stakeholders, including technical details, algorithmic approaches, and performance metrics.

Data Collection and Preprocessing

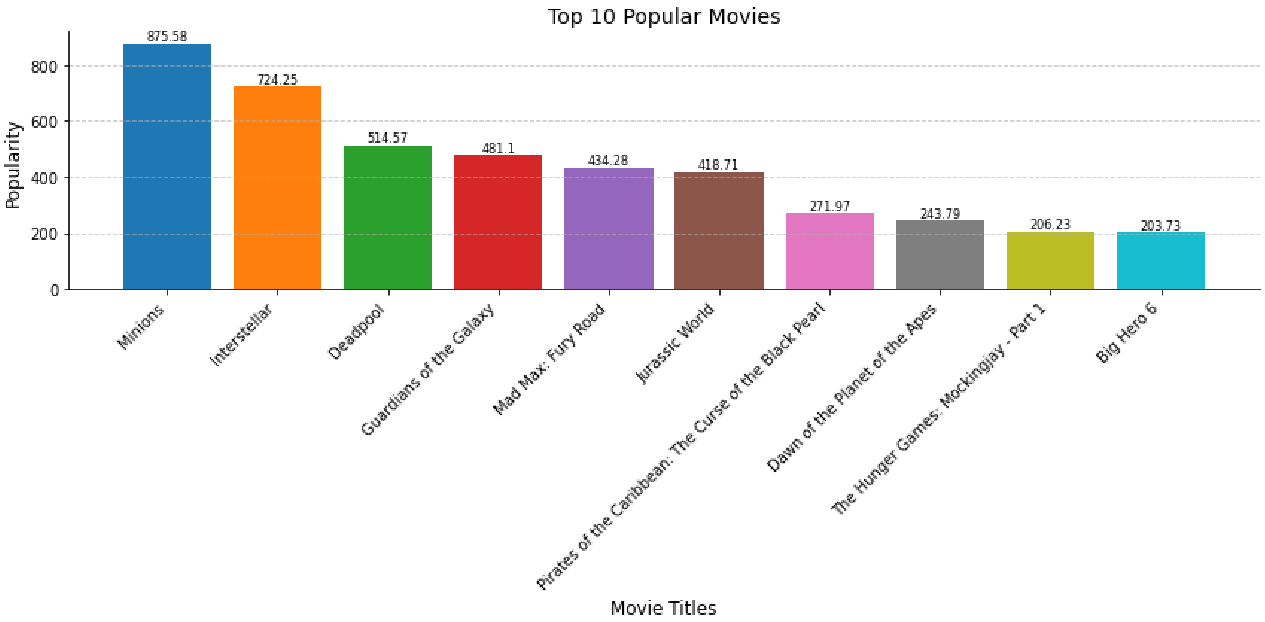
```
# Merge datasets on 'id'
merged_df = pd.merge(movies, credits, how='inner', left_on='id', right_on='movie_id')
# Data Cleaning
# Drop duplicates and irrelevant columns
merged_df.drop_duplicates(subset='id', inplace=True)
merged_df.drop(columns=['movie_id', 'title_y', 'homepage', 'status', 'tagline'], inplace=True)

# Handle missing values
merged_df.dropna(inplace=True)  # Drop rows with any missing values
# Normalize numerical features
numerical_features = ['budget', 'popularity', 'revenue', 'runtime', 'vote_average', 'vote_count']
merged_df[numerical_features] = merged_df[numerical_features].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
# Feature Extraction
# Extracting first actor and director from cast and crew columns
merged_df['actor'] = merged_df['cast'].apply(lambda x: x.split(',')[0] if ',' in x else x)
merged_df['director'] = merged_df['crew'].apply(lambda x: x.split(',')[0] if ',' in x else x)

# Further feature extraction can be performed as per specific requirements
# Print cleaned and processed DataFrame
print(merged_df.head())
```

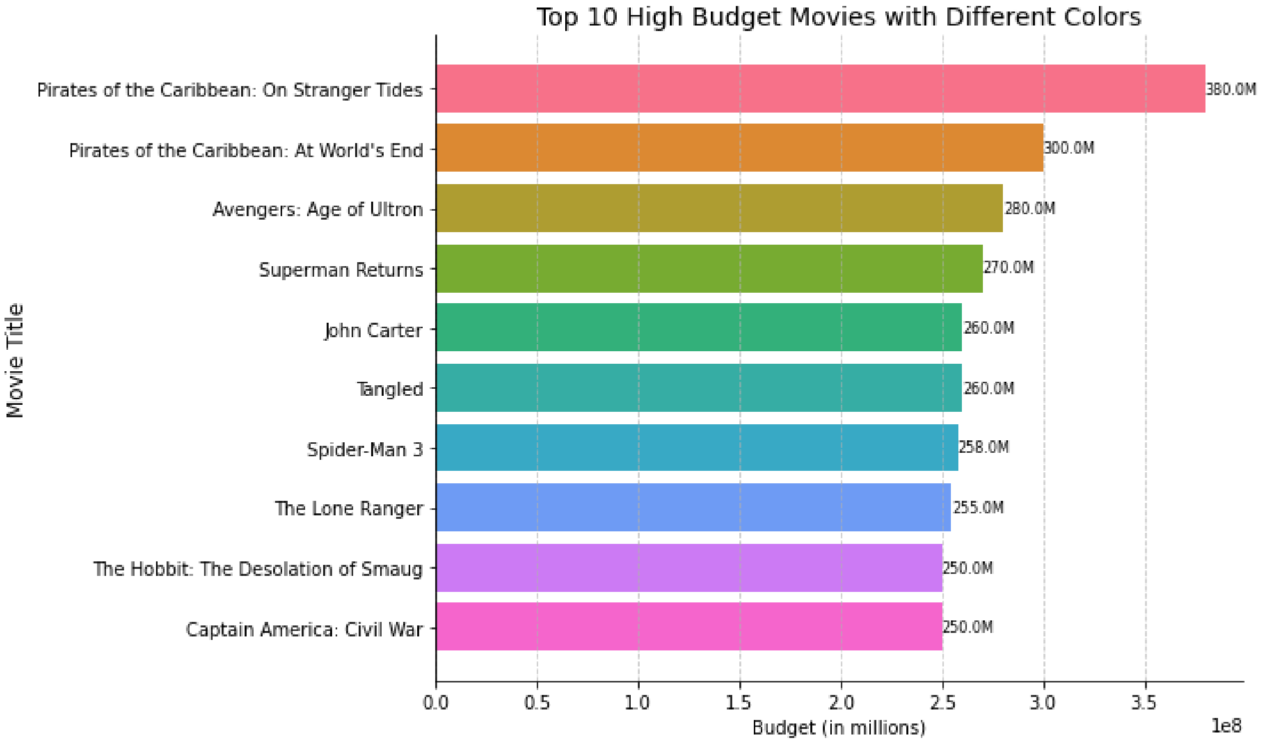


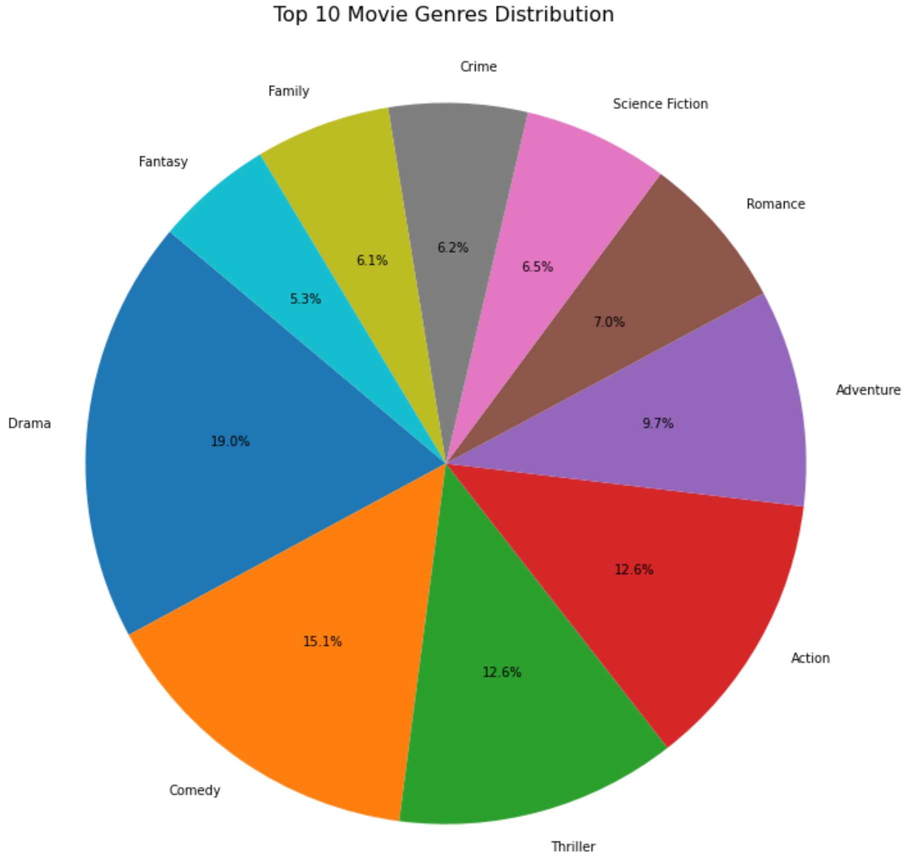
```
     budget                                             genres      id  \
0  0.623684  [{"id": 28, "name": "Action"}, {"id": 12, "nam...   19995
1  0.789474  [{"id": 12, "name": "Adventure"}, {"id": 14, "...     285
2  0.644737  [{"id": 28, "name": "Action"}, {"id": 12, "nam...  206647
3  0.657895  [{"id": 28, "name": "Action"}, {"id": 80, "nam...   49026
4  0.684211  [{"id": 28, "name": "Action"}, {"id": 12, "nam...   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...    0.171814
1  Captain Barbossa, long believed to be dead, ha...    0.158846
2  A cryptic message from Bond’s past sends him o...    0.122634
3  Following the death of District Attorney Harve...    0.128272
```

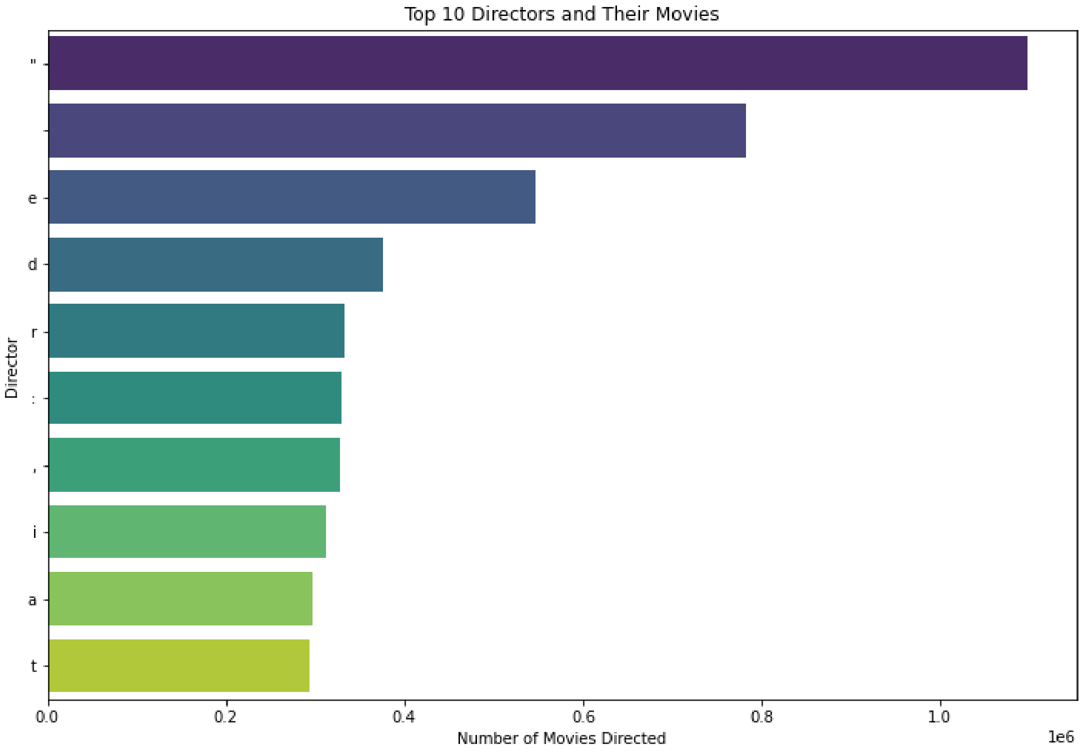
Construct Movie Graph

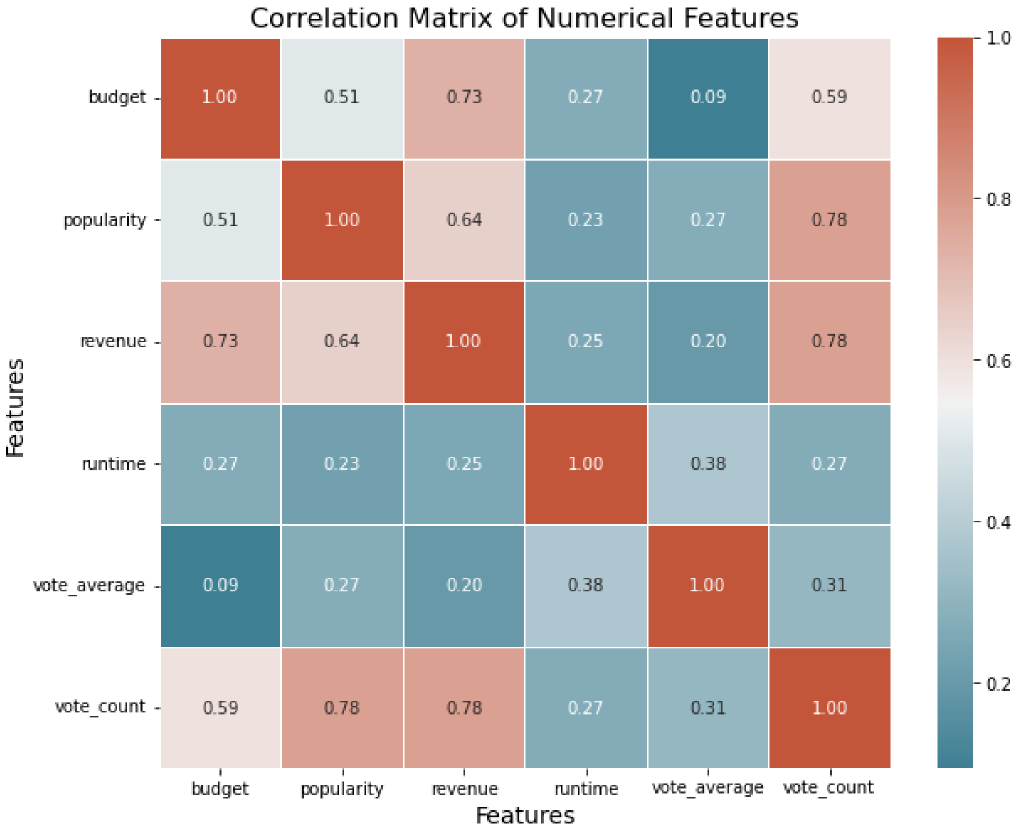
Represent the movie data as a graph where nodes represent movies and edges represent relationships between them.
Relationships between movies can include similarity based on genres, actors, directors, or user ratings.




Top 10 High Budget Movies with Different Colors
Movie Title
Pirates of the Caribbean: On Stranger Tides
Pirates of the Caribbean: At World's End
Avengers: Age of Ultron
Superman Returns
John Carter
Tangled
Spider-Man 3
The Lone Ranger
The Hobbit: The Desolation of Smaug
Captain America: Civil War
380.0M
300.0M
280.0M
270.0M
260.0M
260.0M
258.0M
255.0M
250.0M
250.0M
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e8
Budget (in millions)

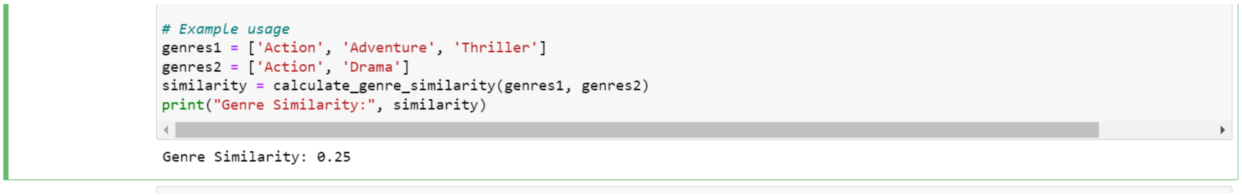

Top 10 Movie Genres Distribution
Crime
6.2%
Science Fiction
6.5%
Romance
7.0%
Adventure
9.7%
Action
12.6%
Thriller
12.6%
Comedy
15.1%
Drama
19.0%
Fantasy
5.3%
Family
6.1%


Top 10 Directors and Their Movies
Director
"
e
d
r
:
,
i
a
t
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Number of Movies Directed


Correlation Matrix of Numerical Features
Features
budget 1.00 0.51 0.73 0.27 0.09 0.59
popularity 0.51 1.00 0.64 0.23 0.27 0.78
revenue 0.73 0.64 1.00 0.25 0.20 0.78
runtime 0.27 0.23 0.25 1.00 0.38 0.27
vote_average 0.09 0.27 0.20 0.38 1.00 0.31
vote_count 0.59 0.78 0.78 0.27 0.31 1.00
budget
popularity
revenue
runtime
vote_average
vote_count
Features
1.0
0.8
0.6
0.4
0.2

Define Heuristic Function

```
def heuristic(movie, user_preferences):
    # Define weights for different factors
    genre_weight = 0.5
    rating_weight = 0.3
    popularity_weight = 0.2

    # Calculate genre similarity
    genre_similarity = calculate_genre_similarity(movie['genres'],
user_preferences['genres'])

    # Normalize and calculate rating similarity (assuming higher rating is better)
    rating_similarity = movie['vote_average'] / 10.0

    # Normalize and calculate popularity similarity (assuming higher popularity is better)
    popularity_similarity = movie['popularity'] / merged_df['popularity'].max()

    # Calculate total heuristic value as weighted sum of similarities
    total_similarity = (genre_weight * genre_similarity) + (rating_weight *
rating_similarity) + (popularity_weight * popularity_similarity)

    return total_similarity

def calculate_genre_similarity(genres1, genres2):
    common_genres = len(set(genres1) & set(genres2))  # Count common genres
    total_genres = len(set(genres1).union(genres2))  # Count total unique genres
    genre_similarity = common_genres / total_genres if total_genres > 0 else 0  # Calculate
similarity
    return genre_similarity
```

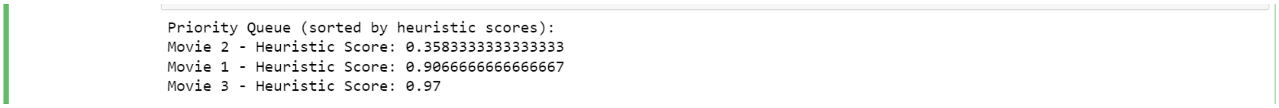


```
# Example usage
genres1 = ['Action', 'Adventure', 'Thriller']
genres2 = ['Action', 'Drama']
similarity = calculate_genre_similarity(genres1, genres2)
print("Genre Similarity:", similarity)
```

```
Genre Similarity: 0.25
```

Initialize Data Structures

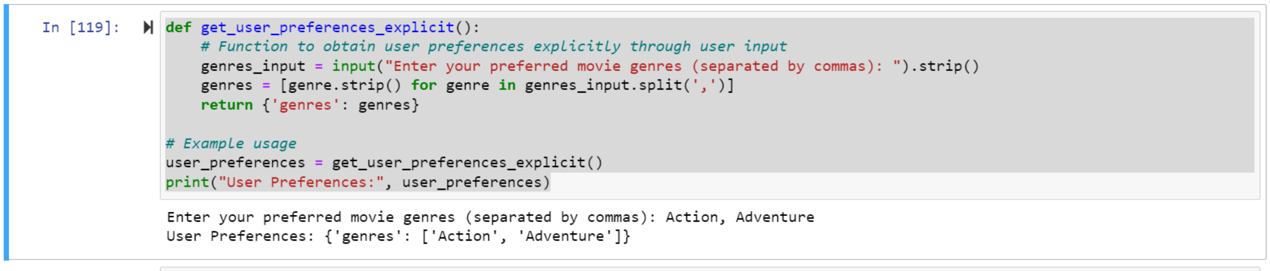
```
import heapq

# Sample movie data
movies = [
    {'title': 'Movie 1', 'genres': ['Action', 'Adventure'], 'vote_average': 8.0, 'popularity': 100},
    {'title': 'Movie 2', 'genres': ['Comedy', 'Romance'], 'vote_average': 7.5, 'popularity': 80},
    {'title': 'Movie 3', 'genres': ['Action', 'Thriller'], 'vote_average': 9.0, 'popularity': 120}
]

# Sample user preferences
user_preferences = {'genres': ['Action']}

# Sample heuristic function (Assuming it's already defined)
def heuristic(movie, user_preferences):
    # Example heuristic calculation based on genres, ratings, and popularity
    genre_similarity = len(set(movie['genres']) & set(user_preferences['genres'])) /
len(set(user_preferences['genres']))
    rating_similarity = movie['vote_average'] / 10.0
    popularity_similarity = movie['popularity'] / 120.0  # Assuming 120 is the maximum
popularity
    return genre_similarity * 0.5 + rating_similarity * 0.3 + popularity_similarity * 0.2

# Function to initialize priority queue with movies sorted by heuristic scores
def initialize_priority_queue(movies, user_preferences):
    priority_queue = []
    explored = set()

    for movie in movies:
        heuristic_score = heuristic(movie, user_preferences)
        heapq.heappush(priority_queue, (heuristic_score, movie))

    return priority_queue, explored

# Example usage
priority_queue, explored = initialize_priority_queue(movies, user_preferences)

# Output the priority queue (sorted by heuristic scores)
print("Priority Queue (sorted by heuristic scores):")
for item in priority_queue:
print(item[1]['title'], "- Heuristic Score:", item[0])
```



```
Priority Queue (sorted by heuristic scores):
Movie 2 - Heuristic Score: 0.35833333333333333
Movie 1 - Heuristic Score: 0.9066666666666667
Movie 3 - Heuristic Score: 0.97
```

User Preferences Input

```
def get_user_preferences_explicit():
    # Function to obtain user preferences explicitly through user input
    genres_input = input("Enter your preferred movie genres (separated by commas): ").strip()
    genres = [genre.strip() for genre in genres_input.split(',')]
    return {'genres': genres}

# Example usage
user_preferences = get_user_preferences_explicit()
print("User Preferences:", user_preferences)
```

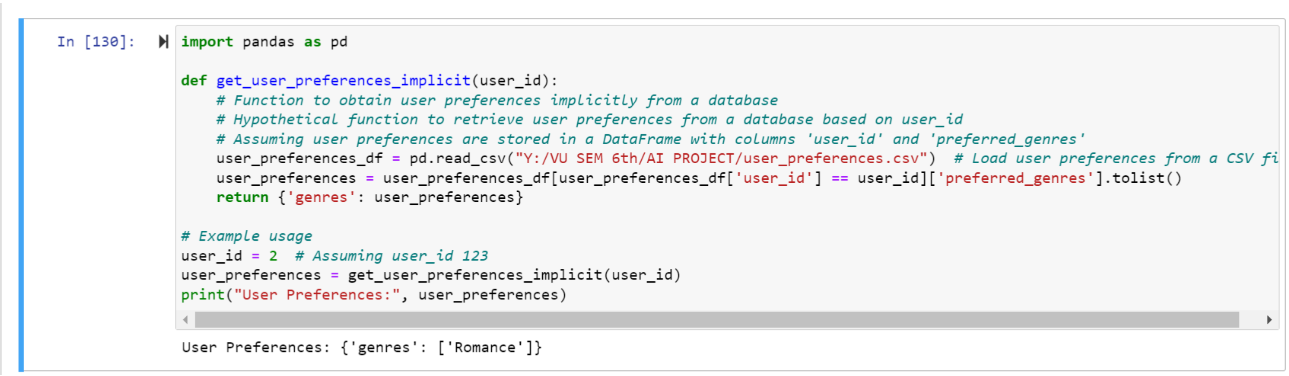


```
In [119]:
def get_user_preferences_explicit():
    # Function to obtain user preferences explicitly through user input
    genres_input = input("Enter your preferred movie genres (separated by commas): ").strip()
    genres = [genre.strip() for genre in genres_input.split(',')]
    return {'genres': genres}

# Example usage
user_preferences = get_user_preferences_explicit()
print("User Preferences:", user_preferences)
```

```
Enter your preferred movie genres (separated by commas): Action, Adventure
User Preferences: {'genres': ['Action', 'Adventure']}
```

Main Loop

```
import pandas as pd

def get_user_preferences_implicit(user_id):
    # Function to obtain user preferences implicitly from a database
    # Hypothetical function to retrieve user preferences from a database based on user_id
    # Assuming user preferences are stored in a DataFrame with columns 'user_id' and 'preferred_genres'
    user_preferences_df = pd.read_csv("Y:/VU SEM 6th/AI PROJECT/user_preferences.csv")  # Load user preferences from a CSV file
    user_preferences = user_preferences_df[user_preferences_df['user_id'] == user_id]['preferred_genres'].tolist()
    return {'genres': user_preferences}

# Example usage
user_id = 2
user_preferences = get_user_preferences_implicit(user_id)
print("User Preferences:", user_preferences)
```

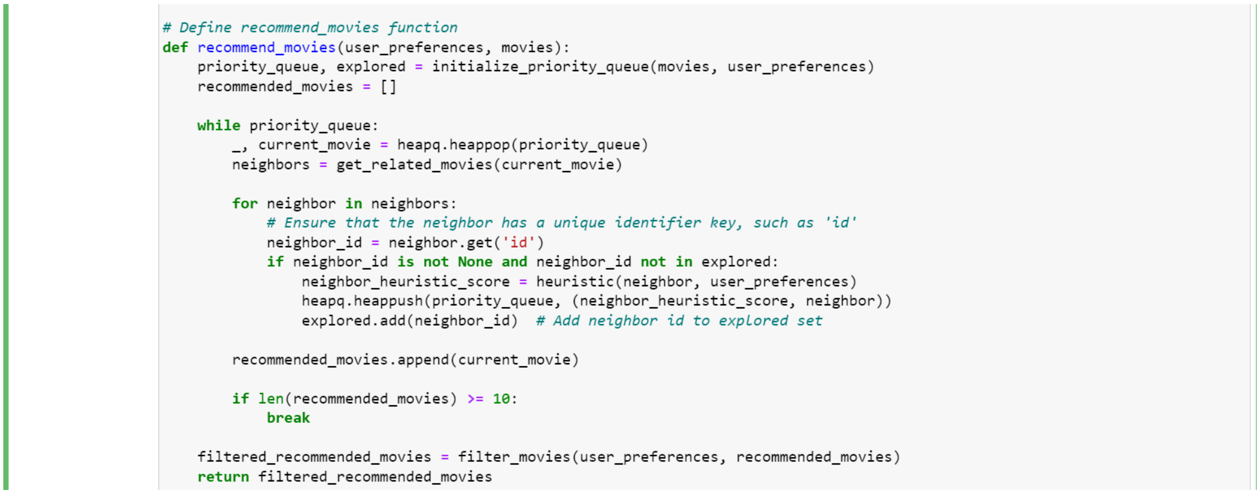


In [130]:

```
import pandas as pd

def get_user_preferences_implicit(user_id):
    # Function to obtain user preferences implicitly from a database
    # Hypothetical function to retrieve user preferences from a database based on user_id
    # Assuming user preferences are stored in a DataFrame with columns 'user_id' and 'preferred_genres'
    user_preferences_df = pd.read_csv("Y:/VU SEM 6th/AI PROJECT/user_preferences.csv")  # Load user preferences from a CSV fi
    user_preferences = user_preferences_df[user_preferences_df['user_id'] == user_id]['preferred_genres'].tolist()
    return {'genres': user_preferences}

# Example usage
user_id = 2  # Assuming user_id 123
user_preferences = get_user_preferences_implicit(user_id)
print("User Preferences:", user_preferences)
```

```
User Preferences: {'genres': ['Romance']}
```

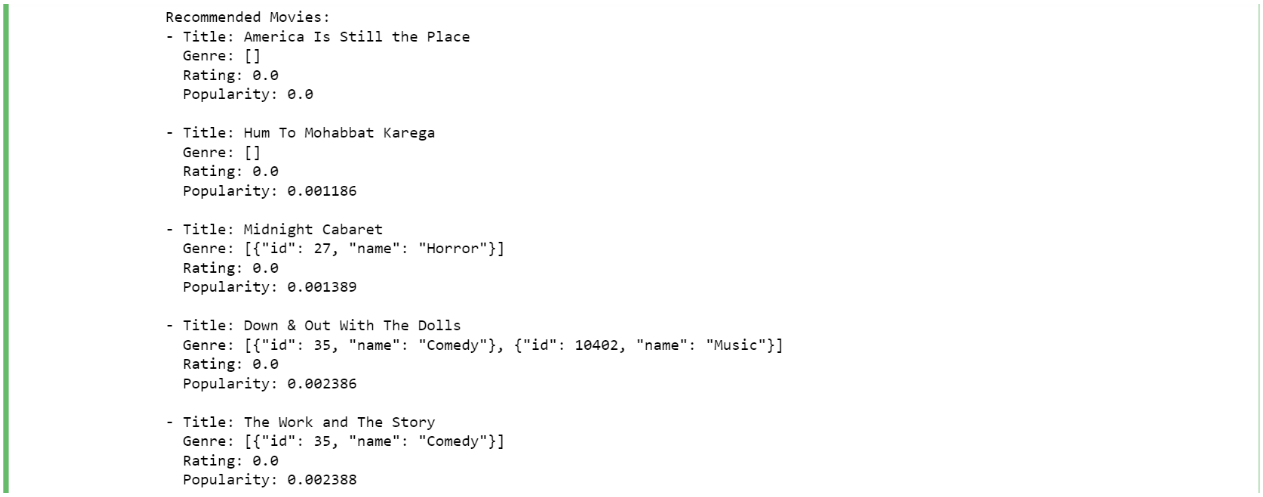
Recommendation Generation



```
# Define recommend_movies function
def recommend_movies(user_preferences, movies):
    priority_queue, explored = initialize_priority_queue(movies, user_preferences)
    recommended_movies = []

    while priority_queue:
        _, current_movie = heapq.heappop(priority_queue)
        neighbors = get_related_movies(current_movie)

        for neighbor in neighbors:
            # Ensure that the neighbor has a unique identifier key, such as 'id'
            neighbor_id = neighbor.get('id')
            if neighbor_id is not None and neighbor_id not in explored:
                neighbor_heuristic_score = heuristic(neighbor, user_preferences)
                heapq.heappush(priority_queue, (neighbor_heuristic_score, neighbor))
                explored.add(neighbor_id)  # Add neighbor id to explored set

        recommended_movies.append(current_movie)

        if len(recommended_movies) >= 10:
            break

    filtered_recommended_movies = filter_movies(user_preferences, recommended_movies)
    return filtered_recommended_movies
```

```
def recommend_movies(user_preferences, movies):
    priority_queue, explored = initialize_priority_queue(movies, user_preferences)
    recommended_movies = []

    while priority_queue:
        _, current_movie = heapq.heappop(priority_queue)
        neighbors = get_related_movies(current_movie)

        for neighbor in neighbors:
            # Ensure that the neighbor has a unique identifier key, such as 'id'
            neighbor_id = neighbor.get('id')
            if neighbor_id is not None and neighbor_id not in explored:
                neighbor_heuristic_score = heuristic(neighbor, user_preferences)
                heapq.heappush(priority_queue, (neighbor_heuristic_score, neighbor))
                explored.add(neighbor_id)  # Add neighbor id to explored set

        recommended_movies.append(current_movie)

        if len(recommended_movies) >= 10:
            break
    filtered_recommended_movies = filter_movies(user_preferences,
recommended_movies)
return filtered_recommended_movies
```

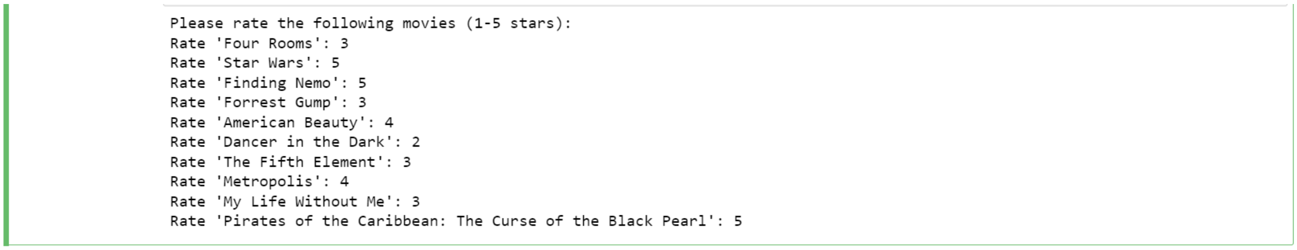
Output Recommendations



```
Recommended Movies:
- Title: America Is Still the Place
  Genre: []
  Rating: 0.0
  Popularity: 0.0

- Title: Hum To Mohabbat Karega
  Genre: []
  Rating: 0.0
  Popularity: 0.001186

- Title: Midnight Cabaret
  Genre: [{"id": 27, "name": "Horror"}]
  Rating: 0.0
  Popularity: 0.001389

- Title: Down & Out With The Dolls
  Genre: [{"id": 35, "name": "Comedy"}, {"id": 10402, "name": "Music"}]
  Rating: 0.0
  Popularity: 0.002386

- Title: The Work and The Story
  Genre: [{"id": 35, "name": "Comedy"}]
  Rating: 0.0
  Popularity: 0.002388
```

Feedback Handling

```
def collect_user_feedback(recommended_movies):
    print("Please rate the following movies (1-5 stars):")
    for movie in recommended_movies:
        rating = int(input(f"Rate '{movie['title']}': "))
        # Update user profile based on feedback
        for genre in movie['genres']:
            if genre in user_profile['genres']:
                user_profile['genres'][genre] += rating  # Increment genre score based on rating
            else:
                user_profile['genres'][genre] = rating

    # Normalize genre scores to ensure they remain within a reasonable range
    total_genre_score = sum(user_profile['genres'].values())
    for genre in user_profile['genres']:
        user_profile['genres'][genre] /= total_genre_score

# Update user profile and adjust heuristic function based on feedback
def update_user_profile_and_heuristic(user_profile, recommended_movies):
collect_user_feedback(recommended_movies)
```



```
Please rate the following movies (1-5 stars):
Rate 'Four Rooms': 3
Rate 'Star Wars': 5
Rate 'Finding Nemo': 5
Rate 'Forrest Gump': 3
Rate 'American Beauty': 4
Rate 'Dancer in the Dark': 2
Rate 'The Fifth Element': 3
Rate 'Metropolis': 4
Rate 'My Life Without Me': 3
Rate 'Pirates of the Caribbean: The Curse of the Black Pearl': 5
```