Project Report
Movie Recommender Systems
(A Content-Based Approach)

jayraj Raulji
Unified mentor

15/11/24 to 15/12/24

Table of Contents

1. Introduction:-

Recommendation systems are tools that help users find the content they like. For example, streaming platforms like Netflix use them to suggest movies or TV shows based on a user's preferences. These systems save time and improve the user experience by reducing the effort needed to search for relevant content.

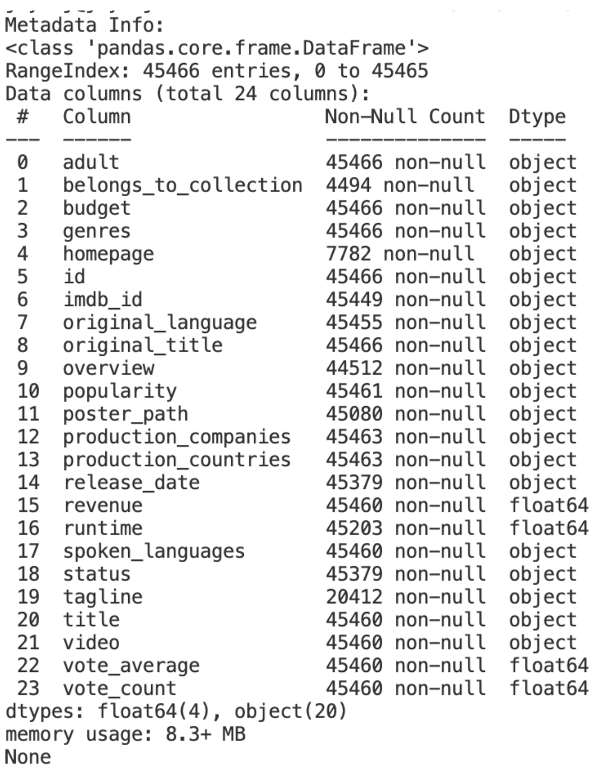
2. Data Preprocessing:-

Step:-1 Import Required Libraries

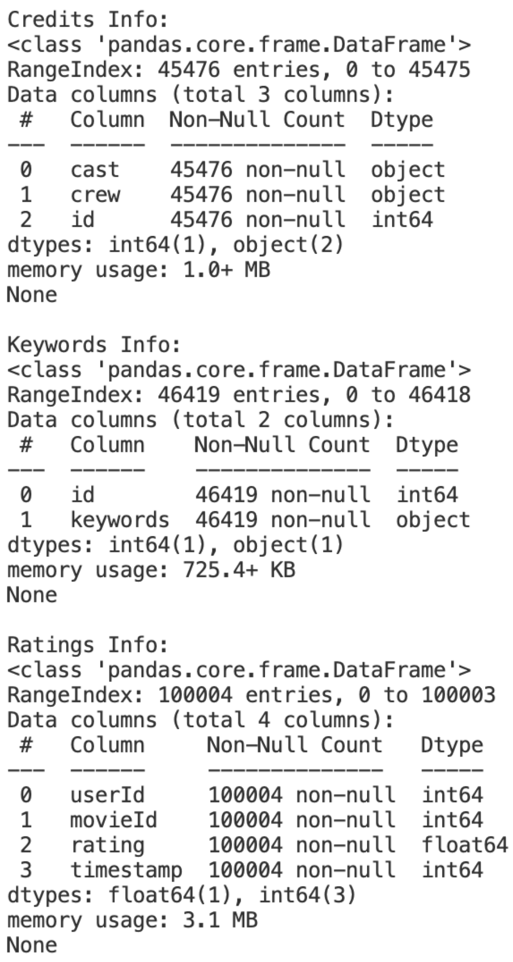
```
# Import Libraries
import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

Step-2 :-Data Loading

```
# Load Data
ratings = pd.read_csv('/Users/jayraj/Desktop/study/gitdemo/
code-demo/Movie_recumentdation/ratings_small.csv')
credits = pd.read_csv('/Users/jayraj/Desktop/study/gitdemo/
credits.csv')
metadata = pd.read_csv('/Users/jayraj/Desktop/study/gitdemo/
code-demo/Movie_recumentdation/movies_metadata.csv',
low_memory=False)
links = pd.read_csv('/Users/jayraj/Desktop/study/gitdemo/
code-demo/Movie_recumentdation/links.csv')
keywords = pd.read_csv('/Users/jayraj/Desktop/study/gitdemo/
code-demo/Movie_recumentdation/keywords.csv')


# Display dataset summaries
print("Metadata Info:")
print(metadata.info())
print("\nCredits Info:")
print(credits.info())
print("\nKeywords Info:")
print(keywords.info())
print("\nRatings Info:")
print(ratings.info())
```



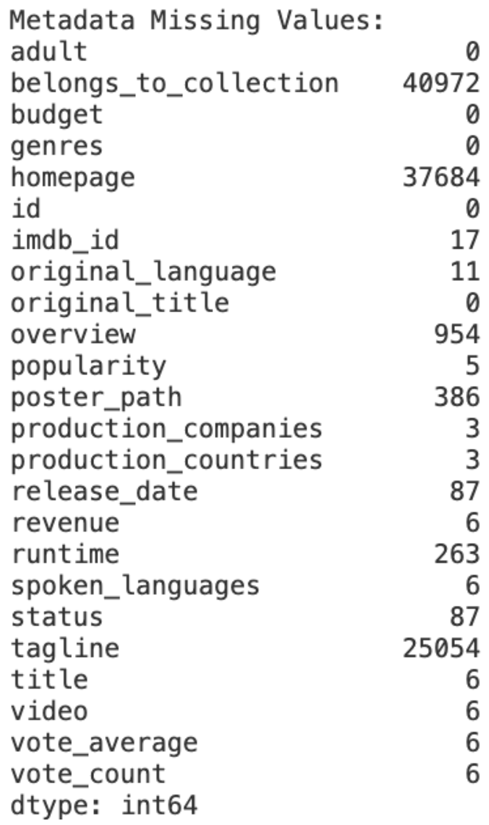
```
Metadata Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45466 entries, 0 to 45465
Data columns (total 24 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   adult                  45466 non-null  object 
 1   belongs_to_collection  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
```



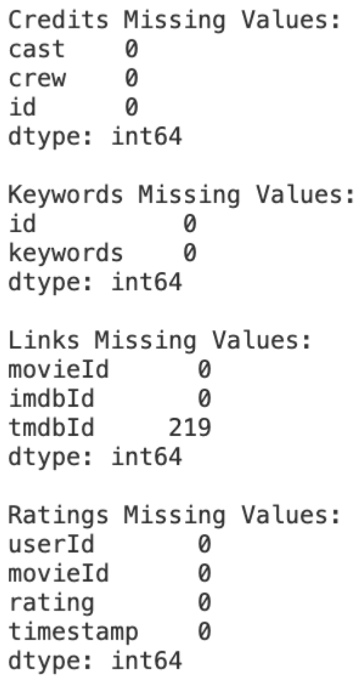
```
Credits Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 45476 entries, 0 to 45475
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   cast    45476 non-null  object
 1   crew    45476 non-null  object
 2   id      45476 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 1.0+ MB
None

Keywords Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 46419 entries, 0 to 46418
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype 
---  ------    --------------  ----- 
 0   id        46419 non-null  int64 
 1   keywords  46419 non-null  object
dtypes: int64(1), object(1)
memory usage: 725.4+ KB
None

Ratings Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100004 entries, 0 to 100003
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   userId     100004 non-null  int64  
 1   movieId    100004 non-null  int64  
 2   rating     100004 non-null  float64
 3   timestamp  100004 non-null  int64  
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
None
```

Step:-3 Data Preprocessing and Cleaning

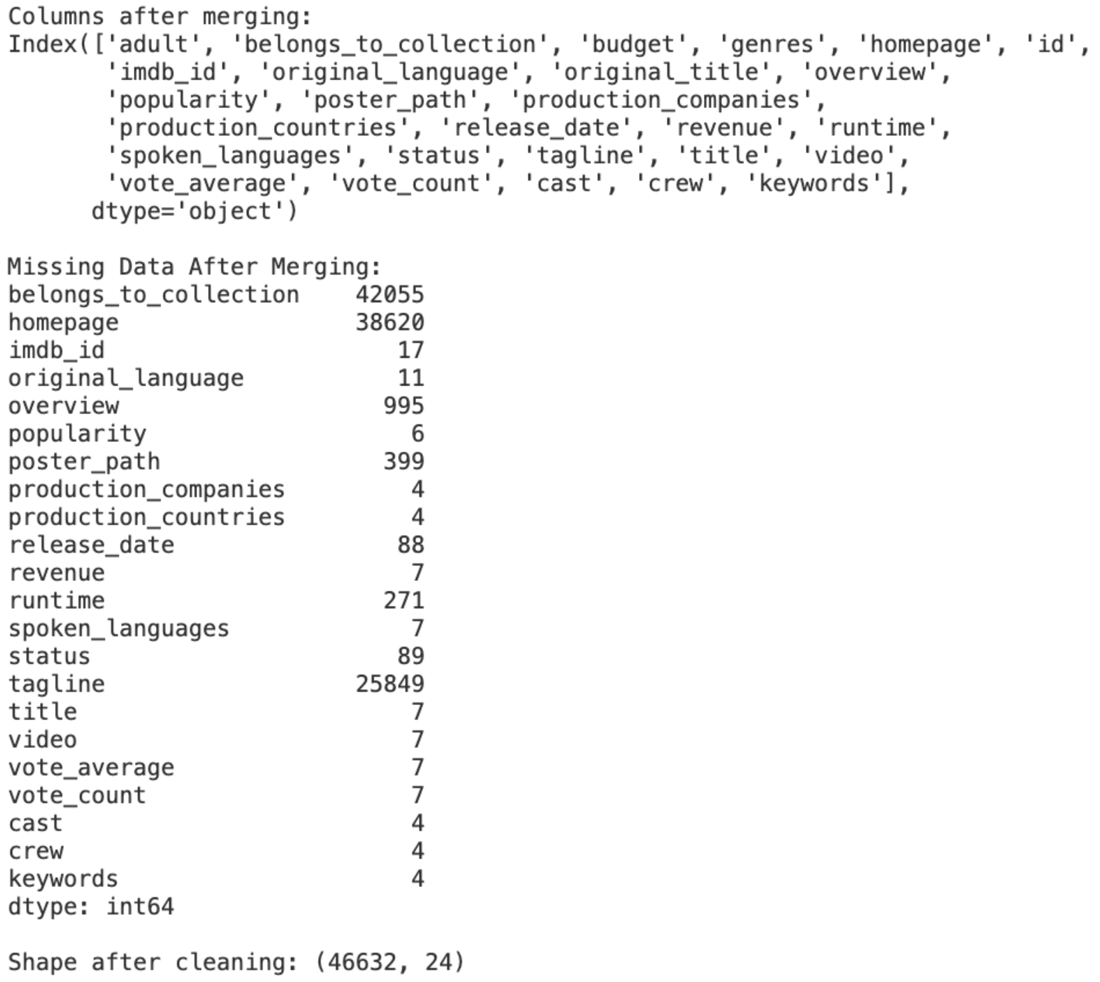
```
# Check initial missing values
datasets = {'Metadata': metadata, 'Credits': credits,
'Keywords': keywords, 'Links': links, 'Ratings': ratings}
for name, df in datasets.items():
    print(f"\n{name} Missing Values:")
    print(df.isnull().sum())

# Merge Datasets
# Ensure 'id' is of string type for consistent merging
metadata['id'] = metadata['id'].astype(str)
credits['id'] = credits['id'].astype(str)
keywords['id'] = keywords['id'].astype(str)

# Merge credits and keywords into metadata
metadata = metadata.merge(credits, on='id', how='left')
metadata = metadata.merge(keywords, on='id', how='left')

print("\nColumns after merging:")
print(metadata.columns)

# Handle Missing Values
missing_data = metadata.isnull().sum()
print("\nMissing Data After Merging:")
print(missing_data[missing_data > 0])

# Fill missing values
metadata.fillna({
    'revenue': 0,
    'runtime': metadata['runtime'].mean(),
    'budget': 0,
    'popularity': 0
}, inplace=True)

# Check shape after cleaning
print("\nShape after cleaning:", metadata.shape)
```



```
Metadata Missing Values:
adult                       0
belongs_to_collection   40972
budget                      0
genres                      0
homepage                37684
id                          0
imdb_id                    17
original_language          11
original_title              0
overview                  954
popularity                  5
poster_path               386
production_companies        3
production_countries        3
release_date               87
revenue                     6
runtime                   263
spoken_languages            6
status                     87
tagline                 25054
title                       6
video                       6
vote_average                6
vote_count                  6
dtype: int64
```

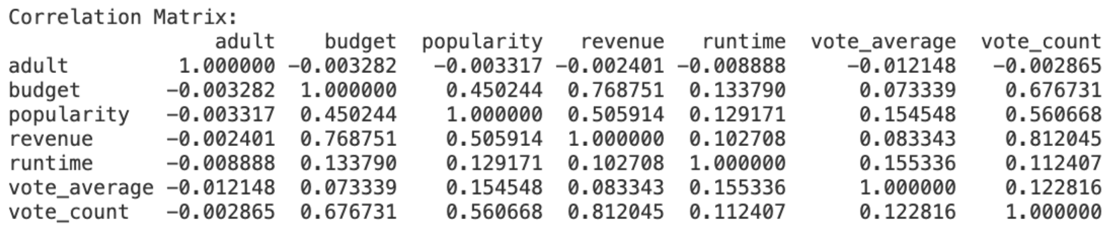


```
Credits Missing Values:
cast    0
crew    0
id      0
dtype: int64

Keywords Missing Values:
id          0
keywords    0
dtype: int64

Links Missing Values:
movieId      0
imdbId       0
tmdbId     219
dtype: int64

Ratings Missing Values:
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64
```



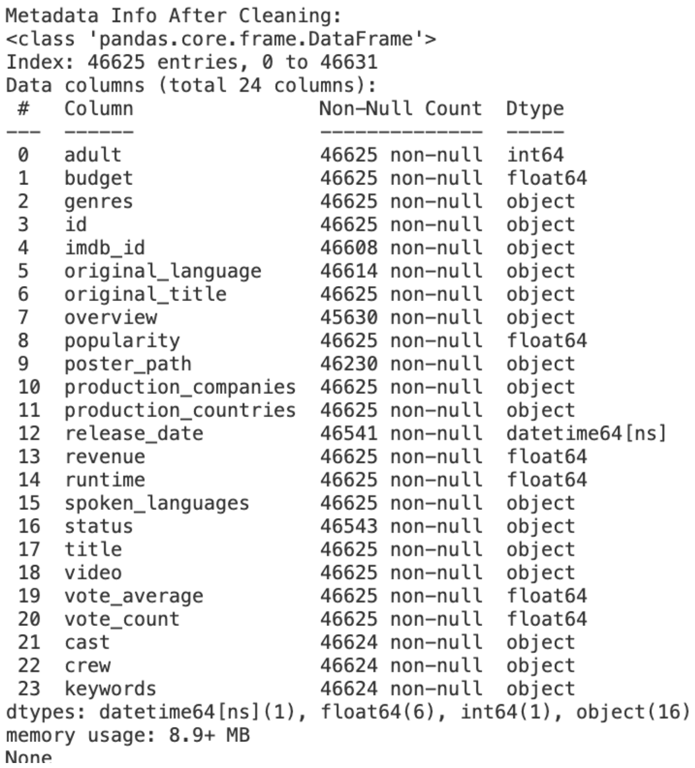
```
Columns after merging:
Index(['adult', 'belongs_to_collection', 'budget', 'genres', 'homepage', 'id',
       'imdb_id', 'original_language', 'original_title', 'overview',
       'popularity', 'poster_path', 'production_companies',
       'production_countries', 'release_date', 'revenue', 'runtime',
       'spoken_languages', 'status', 'tagline', 'title', 'video',
       'vote_average', 'vote_count', 'cast', 'crew', 'keywords'],
      dtype='object')

Missing Data After Merging:
belongs_to_collection    42055
homepage                 38620
imdb_id                     17
original_language           11
overview                   995
popularity                   6
poster_path                399
production_companies         4
production_countries         4
release_date                88
revenue                      7
runtime                    271
spoken_languages             7
status                      89
tagline                  25849
title                        7
video                        7
vote_average                 7
vote_count                   7
cast                         4
crew                         4
keywords                     4
dtype: int64

Shape after cleaning: (46632, 24)
```

```
# Handle Data Types
# Convert boolean-like columns
if 'adult' in metadata.columns:
    metadata['adult'] = metadata['adult'].map({'True': 1,
'False': 0}).fillna(0).astype(int)

# Convert numeric columns
numeric_columns = ['budget', 'revenue', 'popularity']
for col in numeric_columns:
    if col in metadata.columns:
        metadata[col] = pd.to_numeric(metadata[col],
errors='coerce').fillna(0)

# Convert release_date to datetime
if 'release_date' in metadata.columns:
    metadata['release_date'] =
pd.to_datetime(metadata['release_date'], errors='coerce')

# Drop rows with critical missing values (like title or id)
metadata.dropna(subset=['id', 'title'], inplace=True)

# Check correlations among numeric columns
numeric_metadata =
metadata.select_dtypes(include=['float64', 'int64'])
correlation_matrix = numeric_metadata.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)
```

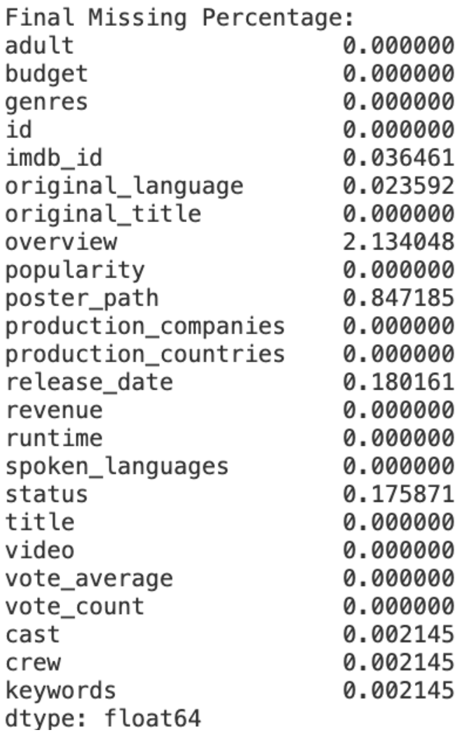


```
Correlation Matrix:
                 adult    budget  popularity   revenue   runtime  vote_average  vote_count
adult         1.000000 -0.003282   -0.003317 -0.002401 -0.008888     -0.012148   -0.002865
budget       -0.003282  1.000000    0.450244  0.768751  0.133790      0.073339    0.676731
popularity   -0.003317  0.450244    1.000000  0.505914  0.129171      0.154548    0.560668
revenue      -0.002401  0.768751    0.505914  1.000000  0.102708      0.083343    0.812045
runtime      -0.008888  0.133790    0.129171  0.102708  1.000000      0.155336    0.112407
vote_average -0.012148  0.073339    0.154548  0.083343  0.155336      1.000000    0.122816
vote_count   -0.002865  0.676731    0.560668  0.812045  0.112407      0.122816    1.000000
```

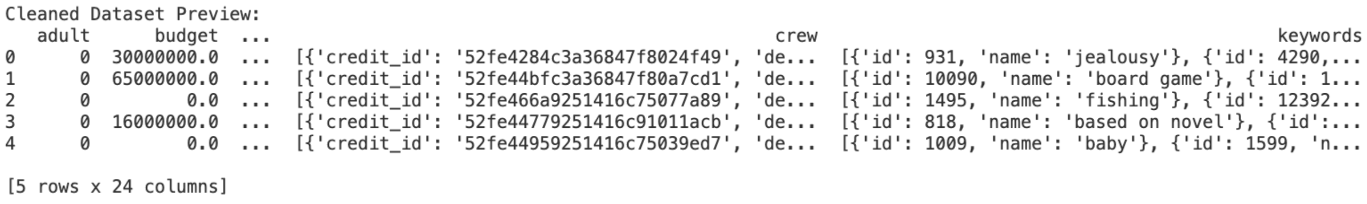
```python
# Final Overview
print("\nMetadata Info After Cleaning:")
print(metadata.info())

# Missing percentage for the final dataset
missing_percentage_final = metadata.isnull().sum() /
len(metadata) * 100
print("\nFinal Missing Percentage:")
print(missing_percentage_final)

# Display a preview of the cleaned dataset
print("\nCleaned Dataset Preview:")
print(metadata.head())
```



```
Metadata Info After Cleaning:
<class 'pandas.core.frame.DataFrame'>
Index: 46625 entries, 0 to 46631
Data columns (total 24 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   adult                 46625 non-null  int64
 1   budget                46625 non-null  float64
 2   genres                46625 non-null  object
 3   id                    46625 non-null  object
 4   imdb_id               46608 non-null  object
 5   original_language     46614 non-null  object
 6   original_title        46625 non-null  object
 7   overview              45630 non-null  object
 8   popularity            46625 non-null  float64
 9   poster_path           46230 non-null  object
 10  production_companies  46625 non-null  object
 11  production_countries  46625 non-null  object
 12  release_date          46541 non-null  datetime64[ns]
 13  revenue               46625 non-null  float64
 14  runtime               46625 non-null  float64
 15  spoken_languages      46625 non-null  object
 16  status                46543 non-null  object
 17  title                 46625 non-null  object
 18  video                 46625 non-null  object
 19  vote_average          46625 non-null  float64
 20  vote_count            46625 non-null  float64
 21  cast                  46624 non-null  object
 22  crew                  46624 non-null  object
 23  keywords              46624 non-null  object
dtypes: datetime64[ns](1), float64(6), int64(1), object(16)
memory usage: 8.9+ MB
None
```



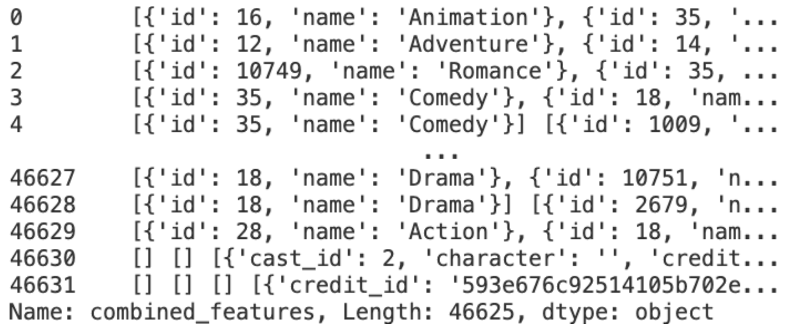
```
Final Missing Percentage:
adult                   0.000000
budget                  0.000000
genres                  0.000000
id                      0.000000
imdb_id                 0.036461
original_language       0.023592
original_title          0.000000
overview                2.134048
popularity              0.000000
poster_path             0.847185
production_companies    0.000000
production_countries    0.000000
release_date            0.180161
revenue                 0.000000
runtime                 0.000000
spoken_languages        0.000000
status                  0.175871
title                   0.000000
video                   0.000000
vote_average            0.000000
vote_count              0.000000
cast                    0.002145
crew                    0.002145
keywords                0.002145
dtype: float64
```



```
Cleaned Dataset Preview:
   adult      budget  ...                                               crew                                           keywords
0      0  30000000.0  ...  [{'credit_id': '52fe4284c3a36847f8024f49', 'de...  [{'id': 931, 'name': 'jealousy'}, {'id': 4290,...
1      0  65000000.0  ...  [{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...  [{'id': 10090, 'name': 'board game'}, {'id': 1...
2      0         0.0  ...  [{'credit_id': '52fe466a9251416c75077a89', 'de...  [{'id': 1495, 'name': 'fishing'}, {'id': 12392...
3      0  16000000.0  ...  [{'credit_id': '52fe44779251416c91011acb', 'de...  [{'id': 818, 'name': 'based on novel'}, {'id':...
4      0         0.0  ...  [{'credit_id': '52fe44959251416c75039ed7', 'de...  [{'id': 1009, 'name': 'baby'}, {'id': 1599, 'n...

[5 rows x 24 columns]
```

Step:4-Feature Engineering

```
# Combine Features
metadata['combined_features'] = (
    metadata['genres'].fillna('') + ' ' +
    metadata['keywords'].fillna('') + ' ' +
    metadata['cast'].fillna('') + ' ' +
    metadata['crew'].fillna('')
)
print(metadata[‘combined_features'])
```

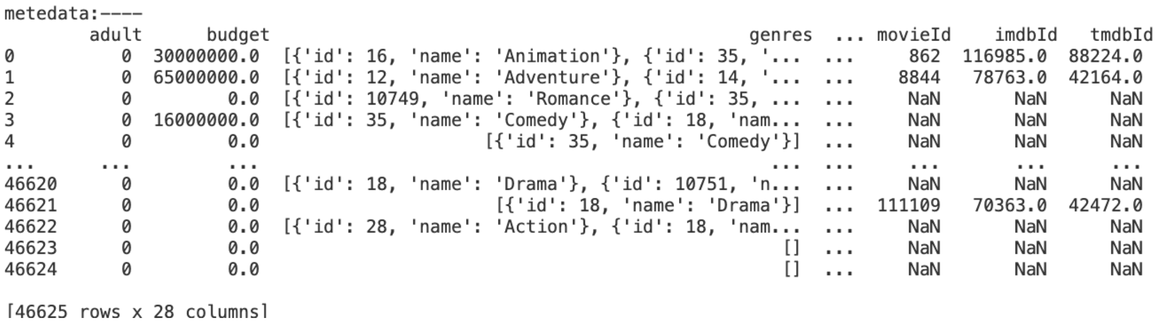


```
0        [{'id': 16, 'name': 'Animation'}, {'id': 35, '...
1        [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...
2        [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...
3        [{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...
4        [{'id': 35, 'name': 'Comedy'}] [{'id': 1009, '...
                                ...
46627    [{'id': 18, 'name': 'Drama'}, {'id': 10751, 'n...
46628    [{'id': 18, 'name': 'Drama'}] [{'id': 2679, 'n...
46629    [{'id': 28, 'name': 'Action'}, {'id': 18, 'nam...
46630    [] [] [{'cast_id': 2, 'character': '', 'credit...
46631    [] [] [] [{'credit_id': '593e676c92514105b702e...
Name: combined_features, Length: 46625, dtype: object
```

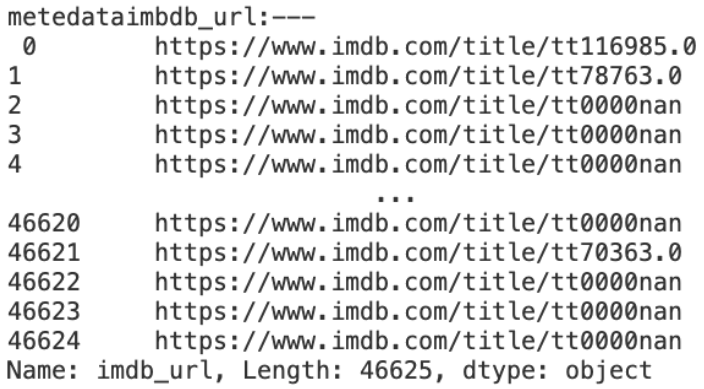
```
# Limit TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english',
max_features=3000)
tfidf_matrix =
vectorizer.fit_transform(metadata['combined_features'])

# Merge with Links to create IMDb URL
# Convert 'movieId' to string to avoid type mismatch during
merge
links['movieId'] = links['movieId'].astype(str)

# Now merge with Links to create IMDb URL
metadata = metadata.merge(links, left_on='id',
right_on='movieId', how='left')
print(‘metedata:----\n',metadata)
```

Step:-5 IMDb URL Construction

```
# Create IMDb URL
metadata['imdb_url'] = 'https://www.imdb.com/title/tt' +
metadata['imdbId'].astype(str).str.zfill(7)
# Fill missing IMDb URLs
metadata['imdb_url'] =
metadata['imdb_url'].fillna('Unavailable')
print(‘metedataimbdb_url:---\n',metadata['imdb_url'])
```



```
metedata:----
       adult      budget                                             genres  ... movieId    imdbId   tmdbId
0          0  30000000.0  [{'id': 16, 'name': 'Animation'}, {'id': 35, '...  ...     862  116985.0  88224.0
1          0  65000000.0  [{'id': 12, 'name': 'Adventure'}, {'id': 14, '...  ...    8844   78763.0  42164.0
2          0         0.0  [{'id': 10749, 'name': 'Romance'}, {'id': 35, ...  ...     NaN       NaN      NaN
3          0  16000000.0  [{'id': 35, 'name': 'Comedy'}, {'id': 18, 'nam...  ...     NaN       NaN      NaN
4          0         0.0                     [{'id': 35, 'name': 'Comedy'}]  ...     NaN       NaN      NaN
...      ...         ...                                                ...  ...     ...       ...      ...
46620      0         0.0  [{'id': 18, 'name': 'Drama'}, {'id': 10751, 'n...  ...     NaN       NaN      NaN
46621      0         0.0                      [{'id': 18, 'name': 'Drama'}]  ...  111109   70363.0  42472.0
46622      0         0.0  [{'id': 28, 'name': 'Action'}, {'id': 18, 'nam...  ...     NaN       NaN      NaN
46623      0         0.0                                                 []  ...     NaN       NaN      NaN
46624      0         0.0                                                 []  ...     NaN       NaN      NaN

[46625 rows x 28 columns]
```



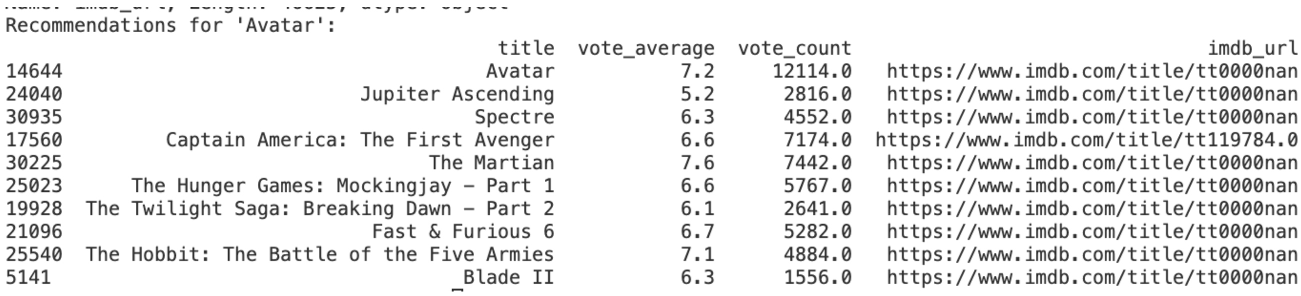
```
metedataimbdb_url:---
 0         https://www.imdb.com/title/tt116985.0
1          https://www.imdb.com/title/tt78763.0
2          https://www.imdb.com/title/tt0000nan
3          https://www.imdb.com/title/tt0000nan
4          https://www.imdb.com/title/tt0000nan
                            ...
46620      https://www.imdb.com/title/tt0000nan
46621      https://www.imdb.com/title/tt70363.0
46622      https://www.imdb.com/title/tt0000nan
46623      https://www.imdb.com/title/tt0000nan
46624      https://www.imdb.com/title/tt0000nan
Name: imdb_url, Length: 46625, dtype: object
```

Step:-6 Recommendation System

```
#Recommendation_movie function
def recommend_movies(title, metadata, tfidf_matrix, 
top_n=10):
    indices = pd.Series(metadata.index, 
index=metadata['title']).drop_duplicates()
    if title not in indices:
        return f"'{title}' not found in the dataset."

    idx = indices[title]
    similarity_scores = cosine_similarity(tfidf_matrix[idx], 
tfidf_matrix).flatten()
    similar_indices = np.argpartition(similarity_scores, 
-top_n)[-top_n:]
    similar_indices = 
similar_indices[np.argsort(similarity_scores[similar_indices
])[::-1]]

    return metadata.iloc[similar_indices][['title', 
'vote_average', 'vote_count', 'imdb_url']]
```

Step:-7 Testing the System

```
# Test recommendation system
movie_title = "Avatar"
recommendations = recommend_movies(movie_title, metadata,
tfidf_matrix, top_n=10)
print(f"Recommendations for '{movie_title}':")
print(recommendations)
```



```
Recommendations for 'Avatar':
                                         title  vote_average  vote_count                                imdb_url
14644                                   Avatar           7.2     12114.0  https://www.imdb.com/title/tt0000nan
24040                        Jupiter Ascending           5.2      2816.0  https://www.imdb.com/title/tt0000nan
30935                                  Spectre           6.3      4552.0  https://www.imdb.com/title/tt0000nan
17560          Captain America: The First Avenger           6.6      7174.0 https://www.imdb.com/title/tt119784.0
30225                               The Martian           7.6      7442.0  https://www.imdb.com/title/tt0000nan
25023       The Hunger Games: Mockingjay - Part 1           6.6      5767.0  https://www.imdb.com/title/tt0000nan
19928   The Twilight Saga: Breaking Dawn - Part 2           6.1      2641.0  https://www.imdb.com/title/tt0000nan
21096                           Fast & Furious 6           6.7      5282.0  https://www.imdb.com/title/tt0000nan
25540  The Hobbit: The Battle of the Five Armies           7.1      4884.0  https://www.imdb.com/title/tt0000nan
5141                                  _Blade II           6.3      1556.0  https://www.imdb.com/title/tt0000nan
```

Step:-8 Conclusion

This project successfully created a Movie Recommendation System using content-based filtering. It recommends movies similar to a given title by analyzing details like genres, cast, crew, and keywords.

The project involved cleaning and combining data, extracting important features, and using cosine similarity to find similar movies. It works well for finding recommendations based on movie metadata.

However, the system does not use user ratings or preferences, which could make it more personalized. In the future, it can be improved by adding more advanced methods like collaborative filtering.

Overall, this project is a good starting point for building movie recommendation systems and shows how data can be used to solve practical problems.

Github - link:-

https://github.com/Jayraj2201/code-demo/tree/cd45b1c720b8d4a610771fabf3e743b2e721131f/Movie_recumentdation