Introduction to the IMDB Top 250 Shows Dataset

The IMDB Top 250 Shows dataset is a comprehensive collection of information about the top-rated television shows as ranked by the Internet Movie Database (IMDB). This dataset is widely used for analysis and development of recommendation systems, providing insights into the attributes that make these shows popular among viewers.

This dataset is having the data of the top 250 Shows as per their IMDB rating listed on the official website of IMDB

Features

- rank - Show Rank as per IMDB rating
- show_id - Show ID
- title - Name of the Show
- year - Year of Show release
- link - URL for the Show
- imdb_votes - Number of people who voted for the IMDB rating
- imdb_rating - Rating of the Show
- certificate - Show Certification
- duration - Duration of the Show
- genre - Genre of the Show
- cast_id - ID of the cast member who have worked on the Show
- cast_name - Name of the cast member who have worked on the Show
- director_id - ID of the director who have directed the Show
- director_name - Name of the director who have directed the Show
- writer_id - ID of the writer who have wrote script for the Show
- writer_name - Name of the writer who have wrote script for the Show
- storyline - Storyline of the Show
- user_id - ID of the user who wrote review for the Show
- user_name - Name of the user who wrote review for the Show
- review_id - ID of the user review
- review_title - Short review
- review_content - Long review

Source

https://www.kaggle.com/datasets/karkavelrajaj/imdb-top-250-shows/data

Importing Necessary Libraries

```
import pandas as pd #Pandas is a powerful library for data manipulation and analysis.
```

```
df = pd.read_csv('shows.csv') #Loading the dataset.
```

We will now read the data from a CSV file into a Pandas DataFrame Let us have a look at how our dataset looks like using df.head()

```
df.head() #Displays the first 5 rows of the dataset.
```

	rank	show_id	title	year	link	imbd_votes	imbd_rati
0	1	tt5491994	Planet Earth II	2016	https://www.imdb.com/title/tt5491994	145,597	(
1	2	tt0903747	Breaking Bad	2008	https://www.imdb.com/title/tt0903747	1,881,190	(
2	3	tt0795176	Planet Earth	2006	https://www.imdb.com/title/tt0795176	210,164	(
3	4	tt0185906	Band of Brothers	2001	https://www.imdb.com/title/tt0185906	469,081	(
4	5	tt7366338	Chernobyl	2019	https://www.imdb.com/title/tt7366338	751,884	(

5 rows × 22 columns

Exploring the Data:

Understanding the dataset by exploring its structure and contents.

```
df.columns # Displays the names of the columns
```

```
Index(['rank', 'show_id', 'title', 'year', 'link', 'imbd_votes', 'imbd_rating',
       'certificate', 'duration', 'genre', 'cast_id', 'cast_name',
       'director_id', 'director_name', 'writer_id', 'writer_name', 'storyline',
       'user_id', 'user_name', 'review_id', 'review_title', 'review_content'],
      dtype='object')
```

```
df.shape # Displays the total count of the Rows and Columns respectively.
```

```
(250, 22)
```

```
df.info() #Displays the total count of values present in the particular column along with th
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 22 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   rank            250 non-null    int64  
 1   show_id         250 non-null    object 
 2   title           250 non-null    object 
 3   year            250 non-null    int64  
 4   link            250 non-null    object 
 5   imbd_votes      250 non-null    object 
 6   imbd_rating     250 non-null    float64
 7   certificate     246 non-null    object 
 8   duration        249 non-null    object 
 9   genre           250 non-null    object 
 10  cast_id         250 non-null    object 
 11  cast_name       250 non-null    object 
 12  director_id     250 non-null    object 
 13  director_name   250 non-null    object 
 14  writer_id       250 non-null    object 
 15  writer_name     250 non-null    object 
 16  storyline       250 non-null    object 
 17  user_id         250 non-null    object 
 18  user_name       250 non-null    object 
 19  review_id       250 non-null    object 
 20  review_title    250 non-null    object 
 21  review_content  250 non-null    object 
dtypes: float64(1), int64(2), object(19)
memory usage: 43.1+ KB
```

Data Cleaning:

Checking for missing values, duplicates, or any inconsistencies and clean the data accordingly.

```
df.isnull().sum()
```

```
rank              0
show_id           0
title             0
year              0
link              0
imbd_votes        0
imbd_rating       0
certificate       4
duration          1
genre             0
```

```
cast_id          0
cast_name        0
director_id      0
director_name    0
writer_id        0
writer_name      0
storyline        0
user_id          0
user_name        0
review_id        0
review_title     0
review_content   0
dtype: int64
```

As we can check there is only 4 null value in the certificate column and duration has 1 null value. As the count of the null value is much less, we can drop the null value as it will not affect the the out come as what we want to predict.

```
df = df.dropna() #Dropping the null values in the dataset.
```

```
df.isnull().sum() #Displays the total count of the null values in the particular columns.
```

```
rank             0
show_id          0
title            0
year             0
link             0
imbd_votes       0
imbd_rating      0
certificate      0
duration         0
genre            0
cast_id          0
cast_name        0
director_id      0
director_name    0
writer_id        0
writer_name      0
storyline        0
user_id          0
user_name        0
review_id        0
review_title     0
review_content   0
dtype: int64
```

Now there is no null value in the dataset.

```
df.drop_duplicates(inplace=True) #Dropping the duplicate values in the dataset.
```

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, OneHotEncoder
```

Content-Based Filtering:

Normalizing the ratings using StandardScaler ensures that the ratings are on a comparable scale, leading to improved performance and stability of the recommendation system

```
# Normalize ratings
scaler = StandardScaler()
df['normalized_rating'] = scaler.fit_transform(df[['imbd_rating']])
```

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

The code snippet tfidf = TfidfVectorizer(stop_words='english'); tfidf_matrix = tfidf.fit_transform(df['genre']) is part of the process to convert text data into numerical features that can be used in machine learning models.

TF-IDF Vectorization

TF-IDF stands for Term Frequency-Inverse Document Frequency. It is a statistical measure used to evaluate the importance of a word in a document relative to a collection of documents (or corpus). The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

```
# Feature extraction
tfidf = TfidfVectorizer(stop_words='english') #This creates an instance of TfidfVectorizer t
tfidf_matrix = tfidf.fit_transform(df['genre'])
```

fit_transform(df['genre']) does two things:

Fit: It learns the vocabulary from the genre column, determining the term frequency and document frequency for each term in the genres.

Transform: It then converts each genre string into a TF-IDF vector. Each row in the resulting tfidf_matrix corresponds to a show, and each column corresponds to a term from the genre data, with the cell values representing the TF-IDF

```
# Calculate similarity
cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
```

The cosine_similarity function takes two matrices as input and computes the cosine similarity between the rows of these matrices.

By passing tfidf_matrix twice, you compute the similarity between every pair of shows in the dataset.

```
# Function to get recommendations
def get_recommendations(title, cosine_sim=cosine_sim): #title: The title of the show for whi
    idx = df[df['title'] == title].index[0] #This line finds the index of the show with the
    sim_scores = list(enumerate(cosine_sim[idx])) #This line retrieves the cosine similarity
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True) #This line sorts the l
    sim_scores = sim_scores[1:11] #This line selects the top 10 most similar shows, excludir
    show_indices = [i[0] for i in sim_scores] #This line extracts the indices of the top 10
    return df['title'].iloc[show_indices] #This line returns the titles of the shows corresp
```

```
recommendations = get_recommendations('Planet Earth') #As we input the name of the show, we
recommendations
```

```
2                   Planet Earth
6                  Blue Planet II
8     Cosmos: A Spacetime Odyssey
10                        Cosmos
11                    Our Planet
17                          Life
25                The Blue Planet
29                  Human Planet
30                 Frozen Planet
50                        Africa
Name: title, dtype: object
```

```
recommendations = get_recommendations('Chernobyl') #As we input the name of the movie, we get
recommendations
```