In [ ]:
```
"""
Created on Thu Apr  4 21:27:20 2024

@author: enwezorifeanyi
"""
```

In [ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import polars as pl
import datetime as dt
```

In [ ]:
```
url = 'https://github.com/rashida048/Datasets/raw/master/movie_dataset.csv'
df = pd.read_csv(url)
```

In [ ]:
```
df.head()
pd.set_option('display.max_columns',None)
```

In [ ]:
```
# Number of Rows
print("Number of rows",df.shape[0])
```

In [ ]:
```
# Number of columns
print("number of colums",df.shape[1])
```

In [ ]:
```
df.info()
# convert time to universal format
df['release_date']= pd.to_datetime(Movies['release_date'])
df.columns
```

In [ ]:
```
#drop uncessary columns
df.drop(['homepage','tagline'], axis=1, inplace=True)
```

In [ ]:
```
#handling missing values & nan
df.isna().sum()
```

In [ ]:
```
#drop nan's because there are a lot however a fraction of the entire data set
df.dropna(inplace=True)
```

In [ ]:
```
#check for duplicates
df[df.duplicated()]
```

In [ ]:
```
#handling missing values
df.isna.sum
```

In [ ]:
```
df['cast'] = df['cast'].apply(lambda x: x.split('|'))
df['genres'] = df['genres'].apply(lambda x: x.split('|'))
df['production_companies'] = df['production_companies'].apply(lambda x: x.split('|'))
```

In [ ]:
```
df['genres'].head()
```

In [ ]:
```
#handling unrealisitic values
zero_budget = df.loc[df['budget'] == 0]
```

In [ ]:
```
#We have a lot of rows with zero budget, we may count them as outliers
yrs_counts_zero_budget = zero_budget['release_date'].value_counts()
```

In [ ]:
```
# I Choosed to make it as percentages because counts of movies is increasing over time
# Percentages Are More Accurate
budget_zero_percent = (yrs_counts_zero_budget / df['release_date'].value_counts()) *100
```

In [ ]:
```
''''''DEMOGRAPHIC FILTERING''''''
        # To apply this we need-
        # we need a metric to score or rate movie
        # Calculate the score for every movie
        # Sort the scores and recommend the best rated movie to the users.

        # problem - if a movie have 9.5 rating with 3 votes and
        #            another movie with 7.9 rating with 100 votes.
        #            which one is better? ans: 2nd one.
    #imdb's weight rating(wr) = (v/(v+m) * R) + (m/(m+v) * C)
    # here-
    # v- is the number of votes for the movie; (vote_count from the dataset)
    # m- is the minimum votes required to be listed in the chart;
```

```
    # R- is the average rating of the movie; (vote_average)
    # C- is the mean vote across the whole report
# a movie have to get 90% vote to make a place in the list
# quantile is a pandas dataset here .9 means 90%
```

In [ ]:
```
m= df['vote_count'].quantile(0.75)
# now we can find out which movies got 90% vote. to do that
# we can use .loc | .loc indexer is used to filter rows
# lets try to predict if a movie is popular or not
```

.shape have give 2 value 1st one tell us how many rows in other word how many movies vote got over >=90% vote 2nd one in column which include header

In [ ]:
```
r_movies = df.copy()
r_movies = r_movies.loc[df['vote_count']>= m]
r_movies.shape
```

In [ ]:
```
c = df['vote_average'].mean()
c
```

In [ ]:
```
def imdb_wr(x, m=m, c=c):
    v = x['vote_count']
    r = x['vote_average']
    return (v/(v+m)*r) + (m/(m+v)*c)
```

In [ ]:
```
r_movies['score'] = r_movies.apply(imdb_wr, axis=1)
```

In [ ]:
```
#Let's sort the dataset
r_movies = r_movies.sort_values('score', ascending=False)
#display top 10 movies
r_movies[['title','vote_count','vote_average','score','director']].head(10)
```

In [ ]:
```
''''MOST POPULAR MOVIE BY SORTING DATASET BY POPULARITY COLUMN''''
pop = df.sort_values('popularity',ascending=False)
plt.figure(figsize=(12,4))
plt.xlabel('Popularity')
plt.title('Popular Movies')
plt.barh(pop['title'].head(5), pop['popularity'].head(5), align ='center', color='indianred')
plt.gca().invert_yaxis()
```

In [ ]:
```
'''RECOMMENDER SYSTEM BASED ON CONTENT FILTERING'''
#In the recommender i will build, the content of the movie (overview, cast, crew, keyword, tagline, et
#will be used to find the similarity between movies and then recommendations made
```

In [ ]:
```
df['overview'].head()
#Tokenization: Each movie overview is split into individual words or tokens
#Term Frequency (TF) calculation: dividing the number of occurrences of a term / otal number of terms
#Inverse Document Frequency (IDF) calculation: It is calculated by taking the logarithm of the ratio b
#TF-IDF vector generation: multiplying the TF of each term by its IDF value
#The higher the TF-IDF value for a term in a movie's vector, the more significant that term is in desc
```

In [ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer # https://scikit-learn.org/stable/modules/
#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')
# Replace NaN with an empty string in the 'overview' column of the DataFrame
df['overview'] = df['overview'].fillna('')
#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df['overview']) # fit_transform-  vectorizer that fits the vectori
```

In [ ]:
```
#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

since we have used the TF-IDF vectorizer, calculating the dot product will directly give us the cosine similarity score.

In [ ]:
```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel #https://scikit-learn.org/stable/modules/generated/
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

In [ ]:
```
# removing duplicate -- optional incase if there is some duplicate
indices = pd.Series(df.index, index=df['title']).drop_duplicates()
indices
```

In [ ]:
```
df['title']
```

In [ ]:
```python
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]
    # Get the pairwsie similarity scores of 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 10 most similar movies
    sim_scores = sim_scores[1:11]
    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]
    # Return the top 10 most similar movies
    return df['title'].iloc[movie_indices]
```

In [ ]:
```python
print(get_recommendations('The Dark Knight Rises'))
```

In [ ]:
```python
get_recommendations('Iron Man 3')
```