```
import numpy as np
import pandas as pd

creditsdata = pd.read_csv('/content/drive/MyDrive/machinelearningrecommendation/tmdb_5000_
moviesdata = pd.read_csv('/content/drive/MyDrive/machinelearningrecommendation/tmdb_5000_m
```

joining the two data sets with similar key as ID

```
creditsdata.columns = ['id','tittle','cast','crew']
moviesdata = moviesdata.merge(creditsdata,on='id')
```

now lets have a look at moviesdata that we merged

```
moviesdata.head()
```

	budget	genres	
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http:/.

algorithm for finding the score of every movie

```
VoteMean = moviesdata['vote_average'].mean()
```

this vode mean indicates the mean rating of all the movies that exists in the database.

"name":

```
MinVotes = moviesdata['vote_count'].quantile(0.9)
MinVotes
```

```
1838.4000000000015
```

now we want the min number of votes that a movie should have to cross the criteria of consideration for this we first take out the vote count of all the movies group them and quantile them into different groups. after that I want to fetch out the 90% cutoff of the movie votes

{"id": 12,

```
qualified_movies = moviesdata.copy().loc[moviesdata['vote_count']>=MinVotes]
qualified_movies.shape
```

```
(481, 23)
```

Now we find out the number of movies that pass the MinVotes criteria, after this we have to calculate the rating metric for each movie using the imdb formula.

```
def rating(totaldata, VoteMean=VoteMean, MinVotes=MinVotes):
  VoteCount = totaldata['vote_count']
  VoteAverage = totaldata['vote_average']
  return (VoteCount/(VoteCount+VoteMean) * VoteAverage ) + (VoteMean/(VoteMean+VoteCount)
```

this is the IMDB rating calculation formula to find the single entity unit of comparison

```
qualified_movies['score'] = qualified_movies.apply(rating, axis=1)
```

```
qualified_movies
```

	budget	genres	
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http:/
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.cor
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictu
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.

```
qualified_movies = qualified_movies.sort_values('score', ascending=False)

qualified_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

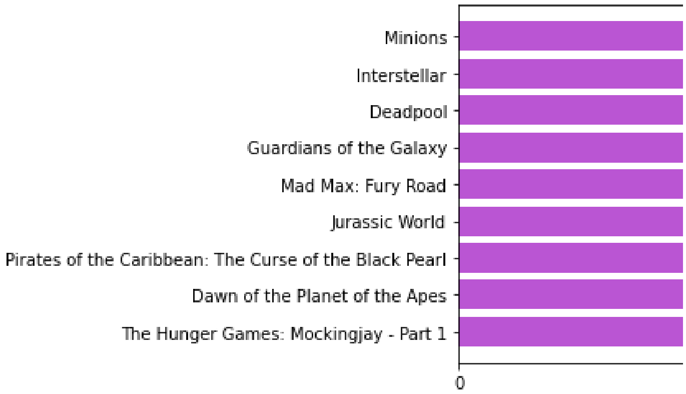
	title	vote_count	vote_av
1405	The Pianist	1864	
2247	Princess Mononoke	1983	
1987	Howl's Moving Castle	1991	
3940	Oldboy	1945	
1819	The Help	1910	
4602	12 Angry Men	2078	
1525	Apocalypse Now	2055	
2585	The Hurt Locker	1840	
2862	About Time	2067	
583	Big Fish	1994	

"Western"}]

the function to show the trending section of the recommendation system "name":

```
pop = moviesdata.sort_values('popularity', ascending = False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))
plt.barh(pop['title'].head(9),pop['popularity'].head(9),align='center',color='mediumorchid
plt.gca().invert_yaxis()
plt.xlabel("REPUTATION PRECEDS ME")
plt.title("I AM POPULAR")
```

```
Text(0.5, 1.0, 'I AM POPULAR')
```



Figsize is a method from the pyplot class which allows you to change the dimensions of the graph The Matplotlib barh () function is used to plot horizontal bar graphs in Python

Now we become more proffesional and make a customer mood, interest and time sensitive recommendation system. parameters like cast, crew, genre, tagline are used to group the data into similar groups.

```
brief_desc_movie = moviesdata['overview'].head(10)
```

now we convert this text to the word view vector inorder to use this for applying various comparisons computing term frequency inverse Document frequency to find the frequency of the occurent of a word in a document which is maintained by a 2D matrix columns go for the number of words atleast occured once and rows represents the movies

```
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words = 'english' )
#this is to remove all the stop words from the description like the and a and other englis

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

vectorizer_matrix = vectorizer.fit_transform(moviesdata['overview'])
```

now since the matrix is sorted with words and their occurence in the movie we need to group the movies that have same words in the description in them as other movies for this we use the cosine comparison function to find the dot product between two component matrix as they if similar will give dot product as+ve and 1 means a linear relationship

```
from sklearn.metrics.pairwise import linear_kernel

similarity_frame = linear_kernel(vectorizer_matrix, vectorizer_matrix)
```

now we need to reverse map the movies to get the similar movies, this my recommendation systems one of the most significant output for this I first try to map the title with the help of index

```
indices = pd.Series(moviesdata.index, moviesdata['title']).drop_duplicates()
```

now what we will do is with the help of title we will locate the similarity score and then group the 'data with same similarity frame and give the following results

```
def title_recommendation(title, similarity_frame=similarity_frame):
  idx = indices[title]
  sim_scores = list(enumerate(similarity_frame[idx]))
  sim_score = sorted(sim_scores, key=lambda x: x[1], reverse=True)
```

```
  sim_score = sim_score[1:12]

  movie_indices = [i[0] for i in sim_score]
  #tuple creation with similarity score

  return moviesdata['title'].iloc[movie_indices]
```

```
title_recommendation('The Dark Knight Rises')
```

```
65                                The Dark Knig
299                                Batman Forev
428                                Batman Retur
1359                                       Batm
3854    Batman: The Dark Knight Returns, Part
119                                 Batman Begi
2507                                    Slow Bu
9            Batman v Superman: Dawn of Justi
1181                                          J
210                                Batman & Rob
879                           Law Abiding Citiz
Name: title, dtype: object
```

since the movie can be recommended on the basis of similarities in the overview the perspective of watching a movie on basis of genre, actors in the movie, director and relavant plots

```
from ast import literal_eval

features = ['cast', 'crew', 'keywords', 'genres']

for feature in features:
  moviesdata[feature] = moviesdata[feature].apply(literal_eval)
  # to parse the string feature into their python object oncluding cast, crew, keywords an
  #extraction of top actors from the cast and director from crew and genres and plot from
```

```
def get_director(x):
  for i in x:
    if i['job']=='Director':
      return i['name']
  return np.nan
```

```
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        #Check if more than 3 elements exist. If yes, return only first three. If no, retu
        if len(names) > 3:
            names = names[:3]
        return names
```

```
    #Return empty list in case of missing/malformed data
    return []
```

we find the top three instances from the list to find top 3 actors or in general names

for finding the names of the directors

```
moviesdata['director'] = moviesdata['crew'].apply(get_director)

features = ['cast', 'keywords', 'genres']

for feature in features:
  moviesdata[feature] = moviesdata[feature].apply(get_list)

# since we made 3 more parameters that is cast that contains 3 top actors, director, keywo
```

```
def clean_data(x):
  if isinstance(x ,list):
    return [str.lower(i.replace(" ","")) for i in x]
  else:
    if isinstance(x, str):
      return str.lower(x.replace(" ",""))
    else:
      return ''
```

Darshan Hirani
12:09 AM Today
Resolve

This is the movie recommendation part of the project code .
note:
pls avoid typing, retyping or overwriting as the core code is exposed with editing access

for the features that is cast, keyword and genre its a list so we have to iterate through every word and remove spaces and lowercase the text and for dictionary that is not a list we just lowercase and remove the space

```
features = ['cast', 'keywords', 'director', 'genres']

for feature in features:
  moviesdata[feature] = moviesdata[feature].apply(clean_data)
```

now since all the data processing and cleaning is over we make a mix vector that we feed for the comparison

```
def vector_soup(x):
  return ' '.join(x['keywords'])+ ' '+' '.join(x['cast'])+' '+x['director']+ ' ' +' '.join
moviesdata['datasoup'] = moviesdata.apply(vector_soup, axis=1)
```

now we do the same things as we did for the keywords comparison and recommendation matrix we use CountVectorizer and not tfid because even if actors come in lots of movies he is not given intuitive preference

```
from sklearn.feature_extraction.text import CountVectorizer

count = CountVectorizer(stop_words='english')
countmat = count.fit_transform(moviesdata['datasoup'])
```

```
from sklearn.metrics.pairwise import cosine_similarity

sim_feature_improved = cosine_similarity(countmat, countmat)
```

```
moviesdata = moviesdata.reset_index()
# we use reset because we have already set the index based on prevous cosine comparison
indices = pd.Series(moviesdata.index, index=moviesdata['title'])
```

```
title_recommendation('The Dark Knight Rises', sim_feature_improved)
```

```
65               The Dark Knight
119                Batman Begins
4638    Amidst the Devil's Wings
1196                The Prestige
3073           Romeo Is Bleeding
3326              Black November
1503                      Takers
1986                      Faster
303                     Catwoman
747                Gangster Squad
1253                Kiss of Death
Name: title, dtype: object
```

0s completed at 11:59 PM