Movie Recommendation System

UE20CS312

Data Analytics-Project Report

G Chirag PES1UG20CS140

Jampani Gnana Karthik PES1UG20CS174

Movie Recommendation System

<u>Objective:</u> To recommend movies to the users using content based and Collaborative filtering Methods.

<u>Dataset:</u> TMDB Movies dataset

It consists of Movies and Credits csv tables

URL: https://www.kaggle.com/datasets/tmdb/tmdb-movie-metadata

Methodology:-

- Movies and credits datasets are used.
- Importing required python modules
- Preprocessing the dataset
- Visualizing the data
- Recommendation based on rating and genre
- Recommendation based on Content Based Filtering
- Predicting Ratings of the movies using Collaborative Filtering.

Columns:-

Dataset 1: tmdb_5000_credits

- Movie_id
- Title
- Cast
- Crew

Dataset 2: tmdb_5000_movies

- Budget
- Genres
- Homepage
- Id
- Keywords
- Original_title
- Overview
- Popularity
- Production_companies
- Production_countries
- Release_date
- Revenue
- Runtime
- Spoken_language
- Status
- Tagline
- Title
- Vote_average
- Vote_count

Scope of the Project

The objective of this project is to provide accurate movie recommendations to users. The goal of the project is to improve the quality of movie recommendation system, such as accuracy, quality and scalability of system than the pure approaches. This is done using the content based filtering and collaborative filtering methods. There is a huge scope of exploration in this field for improving scalability, accuracy and quality of movie recommendation systems. Movie Recommendation system is very powerful and important system. But, due to the problems associated with pure collaborative approach, movie recommendation systems also suffers with poor recommendation quality and scalability issues.

Agile Methodology:

1.**Collecting the data sets**: Collecting all the required data set from Kaggle web site.in this project we require *tmdb_5000_credits.csv* and *tmdb_5000_movies.csv*

2.**Data Analysis**: make sure that that the collected data sets are correct and analysing the data in the csv files. i.e. checking whether all the column Felds are present in the data sets.

3.**Algorithms:** in our project we have only two algorithms one is cosine similarity and other is KNN are used to build the machine learning recommendation model.

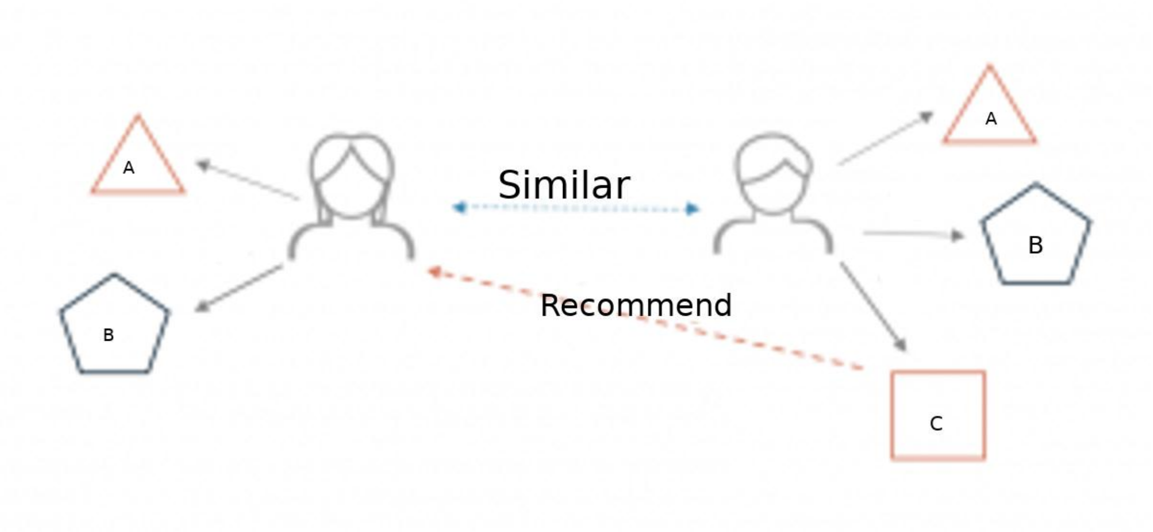
4.**Training and Testing the model**: once the implementation of algorithm is completed . we have to train the model to get the result. We have tested it several times the model is recommend different set of movies to different users.

5.**Improvements in the project**: In the later stage we can implement different algorithms and methods for better recommendation.

What is a recommender system?

A recommender system is a simple algorithm whose aim is to provide the most relevant information to a user by discovering patterns in a dataset. The algorithm rates the items and shows the user the items that they would rate highly. An example of recommendation in action is when you visit Amazon and you notice that some items are being recommended to you or when Netflix recommends certain movies to you. They are also used by Music streaming applications such as Spotify and Youtube to recommend music that you might like.

Below is a very simple illustration of how recommender systems work in the context of an e-commerce site.



Two users buy the same items A and B from an e-commerce store. When this happens the similarity index of these two users is computed. Depending on the score the system can recommend item C to the other user because it detects that those two users are similar in terms of the items they purchase.

Different types of recommendation engines

The most common types of recommendation systems are content-based and collaborative filtering recommender systems. In collaborative filtering, the behavior of a group of users is used to make recommendations to other users. The recommendation is based on the preference of other users. A simple example would be recommending a movie to a user based on the fact that their friend liked the movie. There are two types of collaborative models Memory-based methods and Model-based methods. The advantage of memory-based techniques is that they are simple to implement and the resulting recommendations are often easy to explain. They are divided into three:

Content-based systems

These filtering methods are based on the description of an item and a profile of the user's preferred choices. In a content-based recommendation system, keywords are used to describe the items, besides, a user profile is built to state the type of item this user likes. In other words, the algorithms try to recommend products that are similar to the ones that a user has liked in the past.**Content-based systems are based on the idea that if you liked a certain item you are most likely to like something that is similar to it.**

Collaborative-based systems

In Collaborative Filtering, we tend to find similar users and recommend what similar users like. In this type of recommendation system, we don't use the features of the item to recommend it, rather we classify the users into the clusters of similar types, and recommend each user according to the preference of its cluster.

Hybrid Recommendation Systems

Hybrid recommendations combines both content based and collaborative filterin algorithms. Hybrid approaches can be implemented in several ways, by making content-based and collaborative-based predictions separately and then combining them, by adding content-based capabilities to a collaborative-based approach (and vice versa), or by unifying the approaches into one model.

IMPLEMENTATION

Simple Recommendation System:

Recommending the top 15 movies based on popularity and ratings.

- The Simple Recommender offers **generalized recommendations** to every user **based on movie popularity and (sometimes) genre**.
- The **basic idea** behind this recommender is that **movies that are more popular and more critically acclaimed will have a higher probability of being liked by the average audience.**
- This model **does not give personalized recommendations** based on the user.

$$Weighted\ Rating(WR) = (\frac{v}{v+m}.R) + (\frac{m}{v+m}.C)$$

```
where,
    v is the number of votes for the movie
    m is the minimum votes required to be listed in the chart
    R is the average rating of the movie
    C is the mean vote across the whole report
```

- The next step, we need to determine an appropriate value for `m`, the minimum votes required to be listed in the chart.
- We will use 95th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 95% of the movies in the list.

```
In [28]: m = vote_counts.quantile(0.95)
         m
Out[28]: 3040.5999999999995
```

We have categorized the movies as **qualified** if it has the vote count greater than 3040, which is the 95th percentile.

```
In [30]: qualified = movies2[(movies2['vote_count'] >= m) &
                        (movies2['vote_count'].notnull()) &
                        (movies2['vote_average'].notnull())][['title',
                                                              'year',
                                                              'vote_count',
                                                              'vote_average',
                                                              'popularity',
                                                              'genres']]

         qualified['vote_count'] = qualified['vote_count'].astype('int')
         qualified['vote_average'] = qualified['vote_average'].astype('int')
         qualified.shape
Out[30]: (241, 6)
```

Here we have defined a function which will accept the genre as a parameter and will be displaying the top 15 movies based on popularity.

```
In [36]: def build_chart(genre, percentile=0.85):
    df = genre_movies[genre_movies['genre'] == genre]
    vote_counts = df[df['vote_count'].notnull()]['vote_count'].astype('int')
    vote_averages = df[df['vote_average'].notnull()]['vote_average'].astype('int')
    C = vote_averages.mean()
    m = vote_counts.quantile(percentile)

    qualified = df[(df['vote_count'] >= m) & (df['vote_count'].notnull()) &
                   (df['vote_average'].notnull())][['title', 'year', 'vote_count', 'vote_average', 'popularity']]
    qualified['vote_count'] = qualified['vote_count'].astype('int')
    qualified['vote_average'] = qualified['vote_average'].astype('int')

    qualified['wr'] = qualified.apply(lambda x:
                        (x['vote_count']/(x['vote_count']+m) * x['vote_average']) + (m/(m+x['vote_count']) * C),
                        axis=1)
    qualified = qualified.sort_values('wr', ascending=False).head(250)

    return qualified
```

```
In [37]: build_chart('Action').head(15)
```

Out[37]:

	title	year	vote_count	vote_average	popularity	wr
96	Inception	2010	13752	8	167.583710	7.643635
65	The Dark Knight	2008	12002	8	187.322927	7.600152
262	The Lord of the Rings: The Fellowship of the Ring	2001	8705	8	138.049577	7.480795
329	The Lord of the Rings: The Return of the King	2003	8064	8	123.630332	7.448806
330	The Lord of the Rings: The Two Towers	2002	7487	8	106.914973	7.416443
2917	Star Wars	1977	6624	8	126.393695	7.360261
1996	The Empire Strikes Back	1980	5879	8	78.517830	7.302273
1856	Scarface	1983	2948	8	70.105981	6.915541
0	Avatar	2009	11800	7	150.437577	6.759928
16	The Avengers	2012	11776	7	144.448633	6.759520
788	Deadpool	2016	10995	7	514.569956	6.745435
94	Guardians of the Galaxy	2014	9742	7	481.098624	6.719035
127	Mad Max: Fury Road	2015	9427	7	434.278564	6.711514
3	The Dark Knight Rises	2012	9106	7	112.312950	6.703423
634	The Matrix	1999	8907	7	104.309993	6.698176

```
In [103]: build_chart('Romance').head(15)
```

Out[103]:

	title	year	vote_count	vote_average	popularity	wr
809	Forrest Gump	1994	7927	8	138.133331	7.765517
25	Titanic	1997	7562	7	100.025899	6.864168
81	Maleficent	2014	4496	7	110.620647	6.787311
2003	Her	2013	4097	7	53.682367	6.770405
49	The Great Gatsby	2013	3769	7	61.196071	6.754353
2844	The Fault in Our Stars	2014	3759	7	74.358971	6.753828
2158	Eternal Sunshine of the Spotless Mind	2004	3652	7	56.481487	6.748070
2114	Edward Scissorhands	1990	3601	7	47.513630	6.745230
1701	Aladdin	1992	3416	7	92.982009	6.734367
2553	The Theory of Everything	2014	3311	7	61.182331	6.727779
1263	Amélie	2001	3310	7	73.720244	6.727714
100	The Curious Case of Benjamin Button	2008	3292	7	60.269279	6.726551
1372	The Devil Wears Prada	2006	3088	7	83.893257	6.712642
1565	The Notebook	2004	3067	7	55.109138	6.711130
493	A Beautiful Mind	2001	3009	7	59.248437	6.706868

Conclusion:

- **The Simple Recommender provides every user with generalised suggestions based on the popularity and (sometimes) genre of movies.**
- **The fundamental tenet of this recommender is that more well-known and highly acclaimed films are more likely to be enjoyed by the general public.**

This model does not provide user-specific recommendations.

This type of recommendation can be applied to Netflix, Prime, and other OTT services to provide users with the most well-liked material.

RECOMMENDATION MODELS

1. **Content Based Recommendation System-**Using cosine similarity

- Step 1: Importing the necessary modules

Movie Recommendation System

In [1]:
```
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import plotly.express as px
import plotly.graph_objects as go
import ast
from collections import Counter
import nltk
```

- Step 2: Reading the dataset.

In [2]:
```
credits_df = pd.read_csv('tmdb_5000_credits.csv')
credits_df.shape
credits_df.head()
```

Out[2]:

	movie_id	title	cast	crew
0	19995	Avatar	[{"cast_id": 242, "character": "Jake Sully", "...	[{"credit_id": "52fe48009251416c750aca23", "de...
1	285	Pirates of the Caribbean: At World's End	[{"cast_id": 4, "character": "Captain Jack Spa...	[{"credit_id": "52fe4232c3a36847f800b579", "de...
2	206647	Spectre	[{"cast_id": 1, "character": "James Bond", "cr...	[{"credit_id": "54805967c3a36829b5002c41", "de...
3	49026	The Dark Knight Rises	[{"cast_id": 2, "character": "Bruce Wayne / Ba...	[{"credit_id": "52fe4781c3a36847f81398c3", "de...
4	49529	John Carter	[{"cast_id": 5, "character": "John Carter", "c...	[{"credit_id": "52fe479ac3a36847f813eaa3", "de...

In [3]:
```
movies_df = pd.read_csv('tmdb_5000_movies.csv')
movies_df.shape
movies_df.head()
```

Out[3]:

- Step 3: Merging movies and credits dataset.

Merging movies and credits datasets

In [4]:
```python
movies_df=pd.merge( left = movies_df, right = credits_df, on='title')
movies_df.head()
```

Out[4]:

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_comp
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	[{"name": "Ing Film Partners
1	300000000	[{"id": 12, "name": "Adventure"}, {"id": 14, "...	http://disney.go.com/disneypictures/pirates/	285	[{"id": 270, "name": "ocean"}, {"id": 726, "na...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	[{"name": "Walt D Pictures", "id": :
2	245000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.sonypictures.com/movies/spectre/	206647	[{"id": 470, "name": "spy"}, {"id": 818, "name...	en	Spectre	A cryptic message from Bond's past sends him o...	107.376788	[{"name": "Col Pictures", " {"
3	250000000	[{"id": 28, "name": "Action"}, {"id": 80, "nam...	http://www.thedarkknightrises.com/	49026	[{"id": 849, "name": "dc comics"}, {"id": 853,...	en	The Dark Knight Rises	Following the death of District Attorney Harve...	112.312950	[{"name": "Lege Pictures", "id": 92:
		[{"id": 28, "name":			[{"id": 818, "name":			John Carter is a war-		[{"name": "Walt D

- Step 4: Data Cleaning

In [9]:
```python
movies_df.isnull().sum()
```

Out[9]:
```
movie_id    0
title       0
overview    3
genres      0
keywords    0
cast        0
crew        0
dtype: int64
```

In [10]:
```python
movies_df.dropna(inplace=True)
movies_df.isnull().sum()
```

Out[10]:
```
movie_id    0
title       0
overview    0
genres      0
keywords    0
cast        0
crew        0
dtype: int64
```

In [11]:
```python
movies_df.duplicated().sum()
```

Out[11]: 0

- Step 5: Data Preprocessing

```
In [5]: movies_df.shape

Out[5]: (4809, 23)
```

```
In [6]: movies_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 4809 entries, 0 to 4808
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   budget                4809 non-null   int64
 1   genres                4809 non-null   object
 2   homepage              1713 non-null   object
 3   id                    4809 non-null   int64
 4   keywords              4809 non-null   object
 5   original_language     4809 non-null   object
 6   original_title        4809 non-null   object
 7   overview              4806 non-null   object
 8   popularity            4809 non-null   float64
 9   production_companies  4809 non-null   object
 10  production_countries  4809 non-null   object
 11  release_date          4808 non-null   object
 12  revenue               4809 non-null   int64
 13  runtime               4807 non-null   float64
 14  spoken_languages      4809 non-null   object
 15  status                4809 non-null   object
 16  tagline               3965 non-null   object
 17  title                 4809 non-null   object
 18  vote_average          4809 non-null   float64
 19  vote_count            4809 non-null   int64
 20  movie_id              4809 non-null   int64
 21  cast                  4809 non-null   object
 22  crew                  4809 non-null   object
dtypes: float64(3), int64(5), object(15)
memory usage: 901.7+ KB
```

```
In [8]: #droppind unnecessary columns
movies_df = movies_df[['movie_id','title','overview','genres','keywords','cast','crew']]
movies_df.head()
```

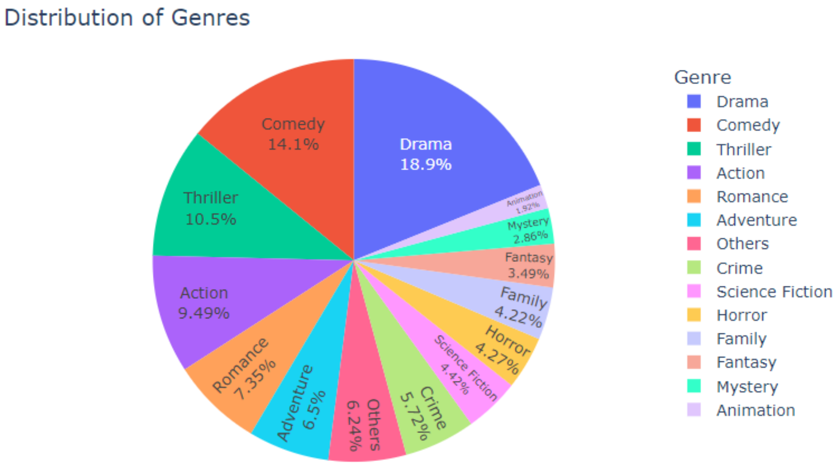
```
In [14]: # Duartion of the data
movies1['release_date'] = pd.to_datetime(movies1['release_date'])
print(movies1['release_date'].max()-movies1['release_date'].min())

36677 days 00:00:00
```

```
In [15]: # Tidying up genre, production_companies and production_countries column
def func(obj):
    List = []
    for i in ast.literal_eval(obj):
        List.append(i['name'])
    return List
```

```
In [16]: movies1['genres'] = movies1['genres'].apply(func)
movies1['production_companies'] = movies1['production_companies'].apply(func)
movies1['production_countries'] = movies1['production_countries'].apply(func)
```

- Step 6: EDA

In [17]:
```python
genres = Counter()
for i in range(movies1.shape[0]):
    for j in movies1.genres[i]:
        genres[j]+=1
Genres = pd.DataFrame.from_dict(genres, orient='index').reset_index()
Genres = Genres.rename(columns = {'index': 'Genres' ,0: 'Frequency'})

Genres.loc[Genres['Frequency'] < 200, 'Genres'] = 'Others'
fig = px.pie(Genres, values='Frequency', names='Genres',width=800,height=500)
fig.update_layout(
    title="Distribution of Genres",
    legend_title="Genre",
    font=dict(
        size=14
    )
)
fig.layout.template = 'plotly'
fig.update_traces(textposition='inside', textinfo='percent+label')
fig.show()
```

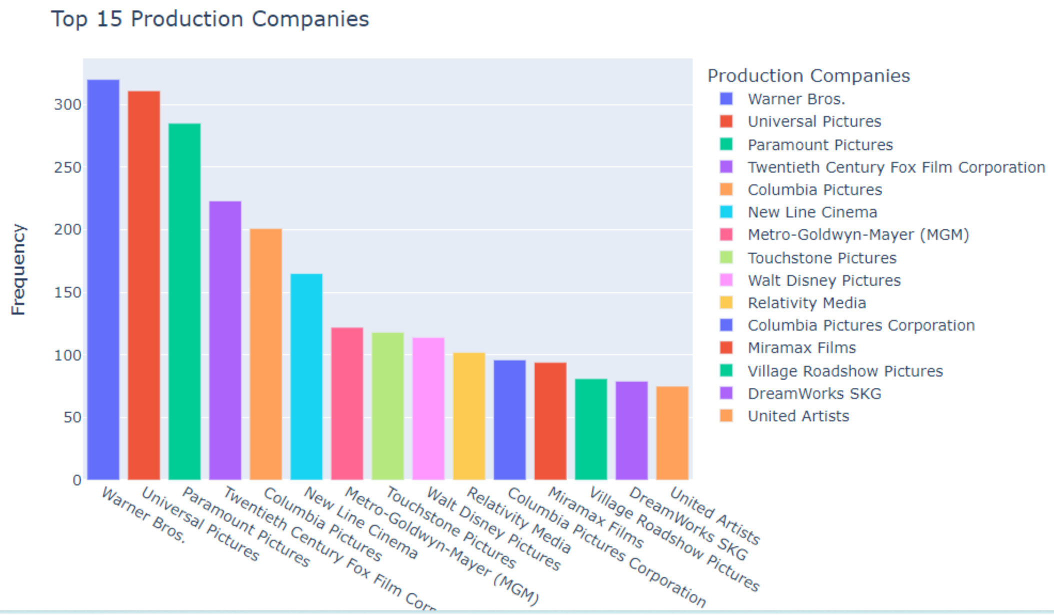


In [18]:
```python
# Top production Comapanies
prod = Counter()
for i in range(movies1.shape[0]):
    for j in movies1.production_companies[i]:
        prod[j]+=1
movie_prod = pd.DataFrame.from_dict(prod, orient='index').reset_index()
movie_prod = movie_prod.rename(columns = {'index': 'Production Company' ,0: 'Frequency'})
movie_prod=movie_prod.sort_values(by = ['Frequency'],ascending=False).reset_index().head(15)
movie_prod.drop(columns='index',axis=0,inplace=True)
movie_prod.style.background_gradient(cmap='RdBu_r')
```

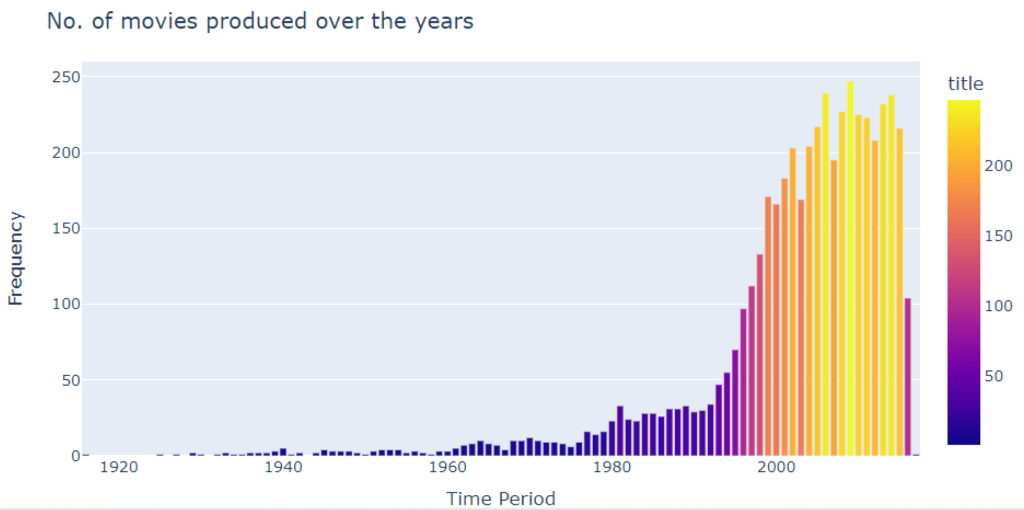
Out[18]:

	Production Company	Frequency
0	Warner Bros.	320
1	Universal Pictures	311
2	Paramount Pictures	285
3	Twentieth Century Fox Film Corporation	223
4	Columbia Pictures	201
5	New Line Cinema	165
6	Metro-Goldwyn-Mayer (MGM)	122
7	Touchstone Pictures	118
8	Walt Disney Pictures	114
9	Relativity Media	102
10	Columbia Pictures Corporation	96
11	Miramax Films	94
12	Village Roadshow Pictures	81
13	DreamWorks SKG	79
14	United Artists	75

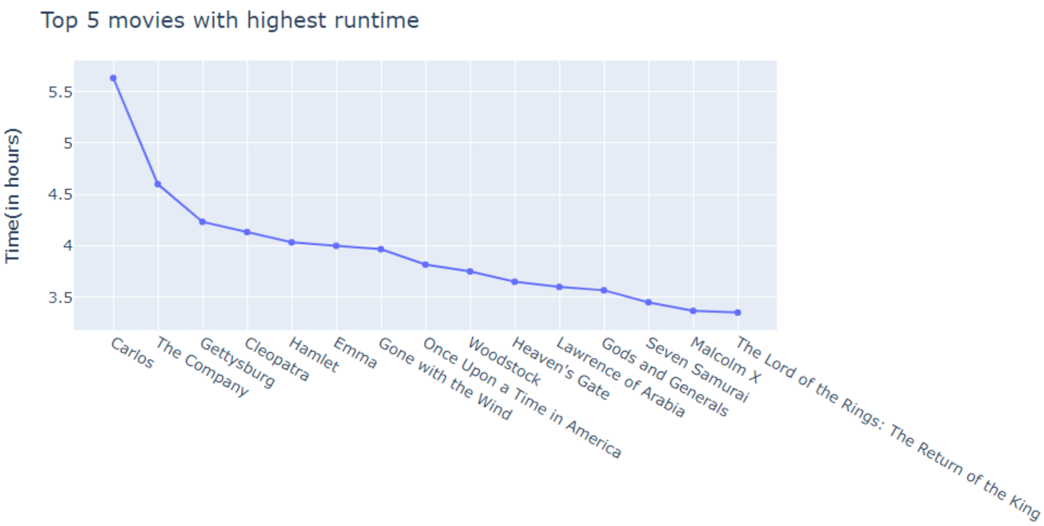
In [19]:
```
fig = px.bar( movie_prod, x='Production Company',y='Frequency', color ='Production Company',width=1000, height=650)
fig.update_layout(
    title="Top 15 Production Companies",
    xaxis_title="Production Companies",
    yaxis_title="Frequency",
    legend_title="Production Companies",
    font=dict(
        size=14
    )
)
fig.layout.template = 'plotly'
fig.show()
```



In [21]:
```
pd.options.mode.chained_assignment = None
release_date=movies1[['title','release_date']]
release_date['release_date']=pd.to_datetime(release_date['release_date'])
release_date['Year']=release_date['release_date'].dt.year
release= release_date.groupby('Year')[['title']].count()

fig = px.bar(release, x=release.index,y='title', color ='title',width=950, height=500)
fig.update_layout(
    title="No. of movies produced over the years",
    xaxis_title="Time Period",
    yaxis_title="Frequency",
    legend_title="Frequency",
    font=dict(
        size=14
    )
)
fig.layout.template = 'plotly'
fig.show()
```



```
In [25]: runtime=movies1[['title','runtime']].sort_values(by='runtime',ascending=False).reset_index().head(15)
runtime.drop(columns='index',axis=0,inplace=True)
runtime['runtime']=runtime['runtime']/60

fig=px.line(runtime,y='runtime',x='title')
fig.update_layout(
    title="Top 5 movies with highest runtime",
    yaxis_title="Time(in hours)",
    xaxis_title="Movies",
    font=dict(
        size=14
    )
)
fig.update_traces(mode='markers+lines')
fig.layout.template = 'plotly'
fig.show()
```



- Step 7: A function named *convert* is created which is used to convert the columns from JSON format to string.

```
In [27]: def convert(obj):
    l=[]
    for i in ast.literal_eval(obj):
        l.append(i['name'])
    return l
```

```
In [28]: movies_df['genres'] = movies_df['genres'].apply(convert)
movies_df['keywords'] = movies_df['keywords'].apply(convert)
movies_df['cast'] = movies_df['cast'].apply(convert)
```

```
In [29]: def fetch_director(obj):
    l=[]
    for i in ast.literal_eval(obj):
        if i['job'] == 'Director':
            l.append(i['name'])
    return l
movies_df['crew'] = movies_df['crew'].apply(fetch_director)
```

```
In [30]: movies_df.head()
```

Out[30]:

	movie_id	title	overview	genres	keywords	cast	crew
0	19995	Avatar	In the 22nd century, a paraplegic Marine is di...	[Action, Adventure, Fantasy, Science Fiction]	[culture clash, future, space war, space colon...	[Sam Worthington, Zoe Saldana, Sigourney Weave...	[James Cameron]
1	285	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	[Adventure, Fantasy, Action]	[ocean, drug abuse, exotic island, east india ...	[Johnny Depp, Orlando Bloom, Keira Knightley, ...	[Gore Verbinski]
2	206647	Spectre	A cryptic message from Bond's past sends him o...	[Action, Adventure, Crime]	[spy, based on novel, secret agent, sequel, mi...	[Daniel Craig, Christoph Waltz, Léa Seydoux, R...	[Sam Mendes]
3	49026	The Dark Knight Rises	Following the death of District Attorney Harve...	[Action, Crime, Drama, Thriller]	[dc comics, crime fighter, terrorist, secret i...	[Christian Bale, Michael Caine, Gary Oldman, A...	[Christopher Nolan]
4	49529	John Carter	John Carter is a war-weary, former military ca...	[Action, Adventure, Science Fiction]	[based on novel, mars, medallion, space travel...	[Taylor Kitsch, Lynn Collins, Samantha Morton,...	[Andrew Stanton]

- Step 8: A new column named **tags** is created which contains each and every word from the other columns.

```
In [35]: movies_df['tags'] = movies_df['overview']+movies_df['genres']+movies_df['keywords']+movies_df['cast']+movies_df['crew']

In [36]: movies_df['tags'][0]

Out[36]: ['In',
          'the',
          '22nd',
          'century,',
          'a',
          'paraplegic',
          'Marine',
          'is',
          'dispatched',
          'to',
          'the',
          'moon',
          'Pandora',
          'on',
          'a',
          'unique',
          'mission,',
          'but',
          'becomes',

In [37]: new_df = movies_df[['movie_id','title','tags']]
         new_df

Out[37]:
```

	movie_id	title	tags
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...
2	206647	Spectre	[A, cryptic, message, from, Bond's, past, send...
3	49026	The Dark Knight Rises	[Following, the, death, of, District, Attorney...

- Step 9: Feature Extraction is done by importing the **CountVectorizer** tool from scikit learn library
  This tool basically converts a given text into a Vector based on the frequency of each word that occurs.

```
In [42]: from sklearn.feature_extraction.text import CountVectorizer
         cv = CountVectorizer(max_features = 5000,stop_words='english')

In [43]: cv.fit_transform(new_df['tags']).toarray().shape

Out[43]: (4806, 5000)

In [44]: vectors = cv.fit_transform(new_df['tags']).toarray()
         vectors[0]

Out[44]: array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

All the words in the column *'tags'* are converted into vectors.
The fit_transform function transforms each token to a specific position in the output function.

Then obtained vectors are converted into array using the toarray() function.

- Step 10: **nltk(natural language tool kit)** library is imported for text processing. From the nltk library **PorterStemmer** imported which is used for stemming.

Stemming is **the process of reducing a word to its word stem that affixes to suffixes and prefixes or to the roots of words known as a lemma.**

```
In [46]: import nltk #natural language tool kit
         from nltk.stem.porter import PorterStemmer
         ps = PorterStemmer()
```

- Step 11: Now we need to apply stemming to each and every word in the tag column. For this we have created a function named ***stem***.
  This function is applied to each and every word in the stem.

```
In [46]: import nltk #natural language tool kit
         from nltk.stem.porter import PorterStemmer
         ps = PorterStemmer()

In [47]: def stem(text):
             y=[]
             for i in text.split():
                 y.append(ps.stem(i))
             return " ".join(y)

In [48]: new_df['tags'] = new_df['tags'].apply(stem)
```

- Step 12: Cosine similarity tool is imported from sklearn.

Cosine Similarity: measures the similarity between two vectors. It calculates the dot product of the vectors and depending on the values it determines whether the two vectors are pointing roughly in the same direction.

```
In [49]: from sklearn.metrics.pairwise import cosine_similarity

In [50]: cosine_similarity(vectors)

Out[50]: array([[1.        , 0.07142857, 0.05143445, ..., 0.02326211, 0.02571722,
                 0.        ],
                [0.07142857, 1.        , 0.07715167, ..., 0.02326211, 0.        ,
                 0.        ],
                [0.05143445, 0.07715167, 1.        , ..., 0.02512595, 0.        ,
                 0.        ],
                ...,
                [0.02326211, 0.02326211, 0.02512595, ..., 1.        , 0.07537784,
                 0.05025189],
                [0.02571722, 0.        , 0.        , ..., 0.07537784, 1.        ,
                 0.05555556],
                [0.        , 0.        , 0.        , ..., 0.05025189, 0.05555556,
                 1.        ]])

In [51]: cosine_similarity(vectors).shape

Out[51]: (4806, 4806)
```

Step 13: The cosine similarity function is applied to all the vectors. This function measures the similarity between all the vectors pairwise.

```
In [52]: similarity = cosine_similarity(vectors)

In [53]: similarity[0].shape

Out[53]: (4806,)
```

Now the recommend function is created which applies the cosine similarity function to all the vectors (i.e the vectors which was created from the tags column).

The similarity values are converted into a list and sorted in descending order. From the sorted list we will be selecting only the top 6 values.

```
In [54]: sorted(list(enumerate(similarity[0])),reverse=True,key=lambda x:x[1])[1:6]

Out[54]: [(1920, 0.23473823893078552),
          (1216, 0.23294541397390256),
          (582, 0.23097828906119441),
          (539, 0.2252817784447915),
          (507, 0.21912524504463887)]

In [55]: def recommend(movie):
             movie_index = new_df[new_df['title'] == movie].index[0]
             distances = similarity[movie_index]
             movies_list = sorted(list(enumerate(distances)), reverse=True, key=lambda x:x[1])[1:7]
             for i in movies_list:
                 print(new_df.iloc[i[0]].title)

In [56]: recommend('Titan A.E.')

         Spirit: Stallion of the Cimarron
         Home
         U.F.O.
         Escape from Planet Earth
         E.T. the Extra-Terrestrial
         The Host
```

CONCLUSION:

The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.The model can capture the specific interests of a user, and can recommend niche items that very few other users are interested in.

2. **Collaborative Filtering-**Using KNN (K-Nearest Neighbour)

Item-Item Collaborative Filtering

Here, we explore the relationship between the pair of items (the user who bought Y, also bought Z). We find the missing rating with the help of the ratings given to the other items by the user.

- Step 1: **Data conversion from JSON to String**

We can see that genres, keywords, production_companies, production_countries, spoken_languages are in the JSON format. Similarly in the other CSV file, cast and crew are in the JSON format. Now let's convert these columns into a format that can be easily read and interpreted. We will convert them into strings and later convert them into lists for easier interpretation.

In [63]:
```
# changing the genres column from json to string
movies['genres'] = movies['genres'].apply(json.loads)
for index,i in zip(movies.index,movies['genres']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name'])) # the key 'name' contains the name of the genre
    movies.loc[index,'genres'] = str(list1)

# changing the keywords column from json to string
movies['keywords'] = movies['keywords'].apply(json.loads)
for index,i in zip(movies.index,movies['keywords']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    movies.loc[index,'keywords'] = str(list1)

# changing the production_companies column from json to string
movies['production_companies'] = movies['production_companies'].apply(json.loads)
for index,i in zip(movies.index,movies['production_companies']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    movies.loc[index,'production_companies'] = str(list1)

# changing the cast column from json to string
credits['cast'] = credits['cast'].apply(json.loads)
for index,i in zip(credits.index,credits['cast']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    credits.loc[index,'cast'] = str(list1)
```

```
# changing the cast column from json to string
credits['cast'] = credits['cast'].apply(json.loads)
for index,i in zip(credits.index,credits['cast']):
    list1 = []
    for j in range(len(i)):
        list1.append((i[j]['name']))
    credits.loc[index,'cast'] = str(list1)

# changing the crew column from json to string
credits['crew'] = credits['crew'].apply(json.loads)
def director(x):
    for i in x:
        if i['job'] == 'Director':
            return i['name']
credits['crew'] = credits['crew'].apply(director)
credits.rename(columns={'crew':'director'},inplace=True)
```

In [64]:
```
movies.head()
```

Out[64]:

	budget	genres	homepage	id	keywords	original_language	original_title	overview	popularity	production_comp
0	237000000	['Action', 'Adventure', 'Fantasy', 'Science Fi...	http://www.avatarmovie.com/	19995	['culture clash', 'future', 'space war', 'spac...	en	Avatar	In the 22nd century, a paraplegic Marine is di...	150.437577	['Ingeniou Partners', 'Twe Cer
1	300000000	['Adventure', 'Fantasy', 'Action']	http://disney.go.com/disneypictures/pirates/	285	['ocean', 'drug abuse', 'exotic island', 'east...	en	Pirates of the Caribbean: At World's End	Captain Barbossa, long believed to be dead, ha...	139.082615	['Walt Disney Pic 'Jerry Bruckheim

- Step 2: **Merging the two csv files**

In [66]:
```
movies = movies.merge(credits,left_on='id',right_on='movie_id',how='left')
movies = movies[['id','original_title','genres','cast','vote_average','director','keywords']]
```

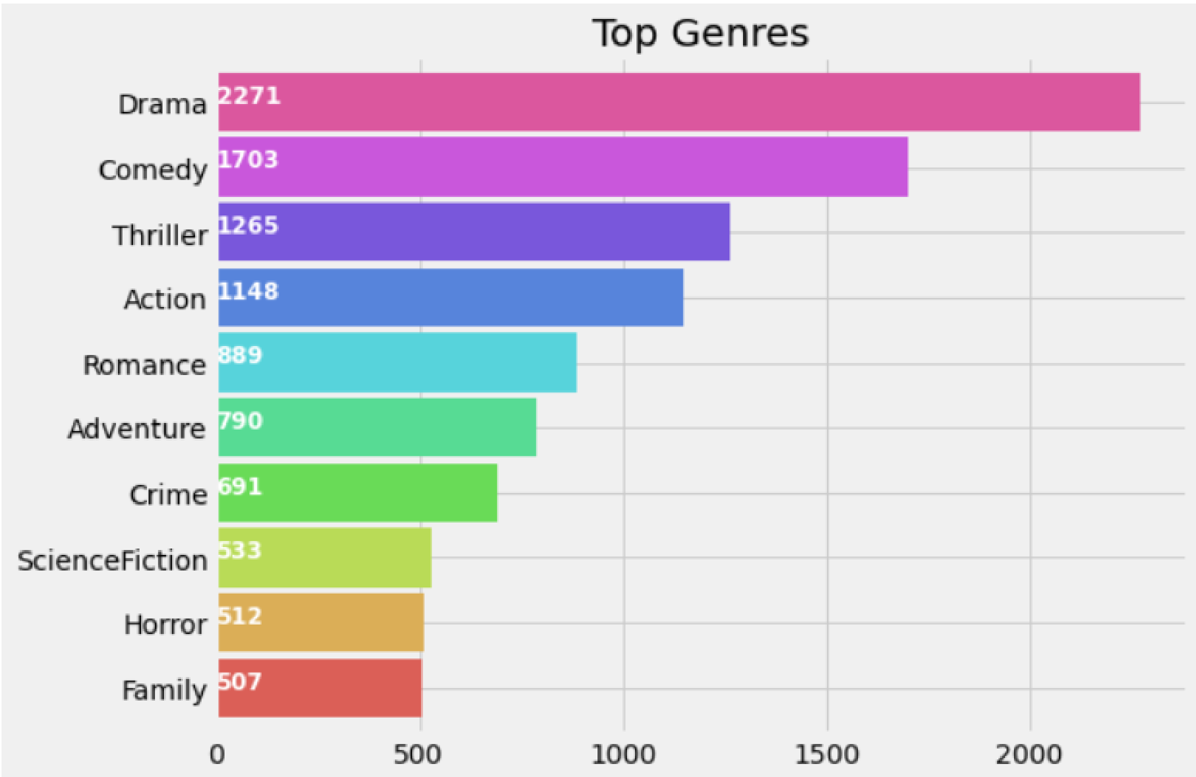
- Step 3:**Working with Genres column**

We will clean the genre column to find the genre_list and plot them.

Genre_list consists of all the unique genres.

```
In [70]: movies['genres'] = movies['genres'].str.strip('[]').str.replace(' ','').str.replace("'",'')
         movies['genres'] = movies['genres'].str.split(',')
```

```
In [71]: plt.subplots(figsize=(12,10))
         list1 = []
         for i in movies['genres']:
             list1.extend(i)
         ax = pd.Series(list1).value_counts()[:10].sort_values(ascending=True).plot.barh(width=0.9,color=sns.color_palette('hls',10))
         for i, v in enumerate(pd.Series(list1).value_counts()[:10].sort_values(ascending=True).values):
             ax.text(.8, i, v,fontsize=12,color='white',weight='bold')
         plt.title('Top Genres')
         plt.show()
```



A new list is created which has all the unique genre names.

```
In [73]: genreList = []
for index, row in movies.iterrows():
    genres = row["genres"]

    for genre in genres:
        if genre not in genreList:
            genreList.append(genre)
genreList[:10] #now we have a list with unique genres

Out[73]: ['Action',
 'Adventure',
 'Fantasy',
 'ScienceFiction',
 'Crime',
 'Drama',
 'Thriller',
 'Animation',
 'Family',
 'Western']
```

'genreList' will now hold all the genres. But how do we come to know about the genres each movie falls into. Now some movies will be 'Action', some will be 'Action, Adventure', etc. We need to classify the movies according to their genres. We have created a new column in the dataframe that will hold the binary values whether a genre is present or not in it. First, let's create a method that will return back a list of binary values for the genres of each movie. The 'genreList' will be useful now to compare against the values.

The same procedure will be applied to the cast,director and the keywords column.

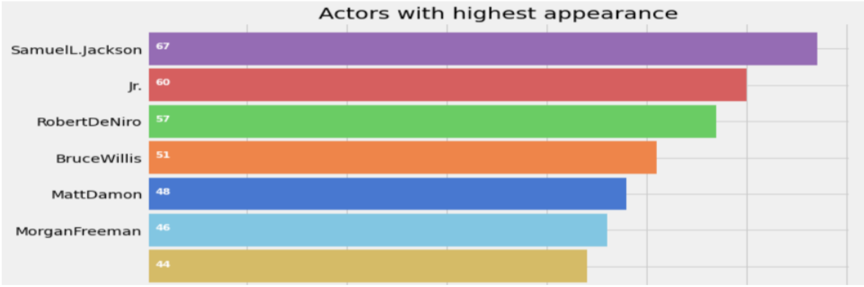
```
In [74]: def binary(genre_list):
    binaryList = []

    for genre in genreList:
        if genre in genre_list:
            binaryList.append(1)
        else:
            binaryList.append(0)

    return binaryList

In [75]: movies['genres_bin'] = movies['genres'].apply(lambda x: binary(x))
movies['genres_bin'].head()

Out[75]: 0    [1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
1    [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
2    [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
3    [1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, ...
4    [1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
Name: genres_bin, dtype: object
```

- Step 4: **Working with Cast column**

Let's plot a graph of Actors with Highest Appearances

```
In [78]: movies['cast'] = movies['cast'].str.strip('[]').str.replace(' ','').str.replace("'",'').str.replace('"','')
         movies['cast'] = movies['cast'].str.split(',')
```

```
In [79]: plt.subplots(figsize=(10,12))
         list1=[]
         for i in movies['cast']:
             list1.extend(i)
         ax=pd.Series(list1).value_counts()[:15].sort_values(ascending=True).plot.barh(width=0.9,color=sns.color_palette('muted',40))
         for i, v in enumerate(pd.Series(list1).value_counts()[:15].sort_values(ascending=True).values):
             ax.text(.8, i, v,fontsize=10,color='white',weight='bold')
         plt.title('Actors with highest appearance')
         plt.show()
```

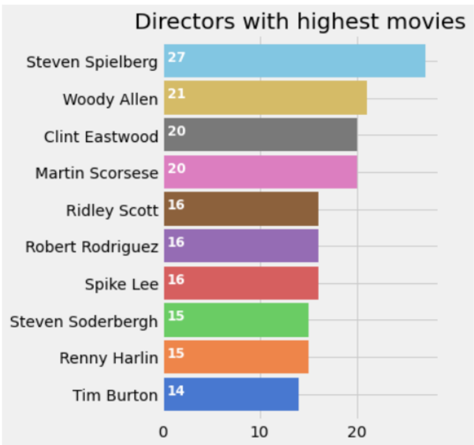


I have selected the main 4 actors from each movie.

- Step 5: **Working with director's column**

Let's plot Directors with maximum movies

```
In [101]: plt.subplots(figsize=(4,6))
          ax = movies[movies['director']!=''].director.value_counts()[:10].sort_values(ascending=True).plot.barh(width=0.9,color=sns.color_
          for i, v in enumerate(movies[movies['director']!=''].director.value_counts()[:10].sort_values(ascending=True).values):
              ax.text(.5, i, v,fontsize=12,color='white',weight='bold')
          plt.title('Directors with highest movies')
          plt.show()
```



- Step 6: **Working with the keywords column**

```
In [89]: movies['keywords'] = movies['keywords'].str.strip('[]').str.replace(' ','').str.replace("'",'').str.replace('"','')
         movies['keywords'] = movies['keywords'].str.split(',')
         for i,j in zip(movies['keywords'],movies.index):
             list2 = []
             list2 = i
             movies.loc[j,'keywords'] = str(list2)
         movies['keywords'] = movies['keywords'].str.strip('[]').str.replace(' ','').str.replace("'",'')
         movies['keywords'] = movies['keywords'].str.split(',')
         for i,j in zip(movies['keywords'],movies.index):
             list2 = []
             list2 = i
             list2.sort()
             movies.loc[j,'keywords'] = str(list2)
         movies['keywords'] = movies['keywords'].str.strip('[]').str.replace(' ','').str.replace("'",'')
         movies['keywords'] = movies['keywords'].str.split(',')
```

```
In [90]: words_list = []
         for index, row in movies.iterrows():
             genres = row["keywords"]

             for genre in genres:
                 if genre not in words_list:
                     words_list.append(genre)
```

- Step 7: **Similarity Between movies**

We will be using **Cosine Similarity** for finding the similarity between 2 movies.

I have defined a function Similarity, which will check the similarity between the movies.

```
In [93]: from scipy import spatial

         def Similarity(movieId1, movieId2):
             a = movies.iloc[movieId1]
             b = movies.iloc[movieId2]

             genresA = a['genres_bin']
             genresB = b['genres_bin']

             genreDistance = spatial.distance.cosine(genresA, genresB)

             scoreA = a['cast_bin']
             scoreB = b['cast_bin']
             scoreDistance = spatial.distance.cosine(scoreA, scoreB)

             directA = a['director_bin']
             directB = b['director_bin']
             directDistance = spatial.distance.cosine(directA, directB)

             wordsA = a['words_bin']
             wordsB = b['words_bin']
             wordsDistance = spatial.distance.cosine(directA, directB)
             return genreDistance + directDistance + scoreDistance + wordsDistance
```

- Step 8: **Score Predictor**

The main function working under the hood will be the ***Similarity()*** function, which will calculate the similarity between movies, and will find 10 most similar movies. These 10 movies will help in predicting the score for our desired movie. We will take the average of the scores of similar movies and find the score for the desired movie.

Here, We have arbitrarily chosen the value K=10.

A small value of K means that noise will have a higher influence on the result and a large value make it computationally expensive.

```
In [95]: import operator

def predict_score(name):
    #name = input('Enter a movie title: ')
    new_movie = movies[movies['original_title'].str.contains(name)].iloc[0].to_frame().T
    print('Selected Movie: ',new_movie.original_title.values[0])
    def getNeighbors(baseMovie, K):
        distances = []

        for index, movie in movies.iterrows():
            if movie['new_id'] != baseMovie['new_id'].values[0]:
                dist = Similarity(baseMovie['new_id'].values[0], movie['new_id'])
                distances.append((movie['new_id'], dist))

        distances.sort(key=operator.itemgetter(1))
        neighbors = []

        for x in range(K):
            neighbors.append(distances[x])
        return neighbors

    K = 10
    avgRating = 0
    neighbors = getNeighbors(new_movie, K)

    print('\nRecommended Movies: \n')
    for neighbor in neighbors:
        avgRating = avgRating+movies.iloc[neighbor[0]][2]
        print( movies.iloc[neighbor[0]][0]+" | Genres: "+str(movies.iloc[neighbor[0]][1]).strip('[]').replace(' ','')+" | Rating:

    print('\n')
    avgRating = avgRating/K
    print('The predicted rating for %s is: %f' %(new_movie['original_title'].values[0],avgRating))
    print('The actual rating for %s is %f' %(new_movie['original_title'].values[0],new_movie['vote_average']))
```

Now when the ***predict_score*** function is called with a movie name as a parameter it will display 10 other similar movies and their predicted ratings.

Predicted ratings will be nothing but the average of the ratings of the 10 movies which are recommended.

```
In [96]: predict_score('Godfather')

Selected Movie:  The Godfather: Part III

Recommended Movies:

The Rainmaker | Genres: 'Crime','Drama','Thriller' | Rating: 6.7
The Godfather: Part II | Genres: 'Crime','Drama' | Rating: 8.3
The Godfather | Genres: 'Crime','Drama' | Rating: 8.4
The Outsiders | Genres: 'Crime','Drama' | Rating: 6.9
The Conversation | Genres: 'Crime','Drama','Mystery' | Rating: 7.5
The Cotton Club | Genres: 'Crime','Drama','Music','Romance' | Rating: 6.6
Apocalypse Now | Genres: 'Drama','War' | Rating: 8.0
Twixt | Genres: 'Horror','Thriller' | Rating: 5.0
New York Stories | Genres: 'Comedy','Drama','Romance' | Rating: 6.2
Peggy Sue Got Married | Genres: 'Comedy','Drama','Fantasy','Romance' | Rating: 5.9


The predicted rating for The Godfather: Part III is: 6.950000
The actual rating for The Godfather: Part III is 7.100000
```

CONCLUSION:

The KNN algorithm is implemented in the collaborative based model along with the principle of cosine similarity as it gives more accuracy than the other distance metrics with additional low complexity advantage. KNN makes use of multiple attributes to filter the similar results and increase accuracy. There is a scope for improvement in recommendation