

Movie Recommendation systems

Manikya Spandana

Dt:18-05-2024

Abstract:

The recommendation engine filters information using specific algorithms and recommends high quality Content to customers. It starts capturing more consumer behavior and based on that, recommends Products that consumers can purchase.

Why do we need recommendation systems?

- Previously users tend to purchase a product based on the opinion or reviews by their friends, neighbors or people they trust.
- But in the today's digital world most of the products are being purchased in the online stores where user can review the product and purchase based on the ratings given by the other customers.
- Hence there was a necessity for the companies to use the data getting collected to provide recommendations for the products which are more likely to be purchased by the customer, hence making the user experience a better and an interactive one.
- Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors.

What is a recommendation engine?

A recommendation engine provides suggestions to the customers by filtering the data based on the users past behavior and purchase history. This can be done with the help of various algorithms applied on the user data with the help of machine learning.

1. Problem Statement

In the recommendation system the problem is trying to forecast the option the users will have on the dissimilar substance and be able to recommends the finest items to each user. Another some problems in recommendation system are data sparsity, scalability and gray sheep

2. Market/Customer Need Assessment

The primary goal of movie recommendation systems is to filter and predict only those movies that a corresponding user is most likely to want to watch. The ML algorithms for these recommendation systems use the data about this user from the system's database.

3.Target Specification and characterization

One of the applications of recommender systems is suggesting movies to watch to customers based on their preferences data. Movie recommender systems work by assessing the characteristic features of the users to make endorsements to the customers on what is best suited for them.

The main objective of the recommender system is to use approaches suggest demographic filtering, content-based filtering, collaborative filtering to find the set of movies with every user likes for specific set of users.

4.External Search (information sources)

The dataset can be found on the Kaggle.

Link: <https://www.kaggle.com/rounakbanik/movie-recommender-systems/data>

Steps involved in this Project: -

- 1.Data Preprocessing
- 2.Exploratory data analysis
- 3.Data Visualization
- 4.Model Building
- 5.Popularity based
- 6.content based
- 7.Collabarative filtering
- 8.Interpretition of results

DATASET INFORMATION

- Total number of movies present in the dataset :45,466
- Total Attributes:24
- Nan- values are present in the dataset

This dataset consists of the following files:

movies_metadata.csv: The main Movies Metadata file. Contains information on 45,000 movies featured in the Full Movie Lens dataset. Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies.

keywords.csv: Contains the movie plot keywords for our Movie Lens movies. Available in the form of a stringified JSON Object.

credits.csv: Consists of Cast and Crew Information for all our movies. Available in the form of a stringified JSON Object.

links.csv: The file that contains the TMDb and IMDb IDs of all the movies featured in the Full Movie Lens dataset.

links_small.csv: Contains the TMDb and IMDb IDs of a small subset of 9,000 movies of the Full Dataset.

ratings_small.csv: The subset of 100,000 ratings from 700 users on 9,000 movies.

As it contains 6 different datasets, it had been merged by common column and finally it dumped in a single dataset for exploratory data analysis and visualization.

DATA PREPROCESSING:

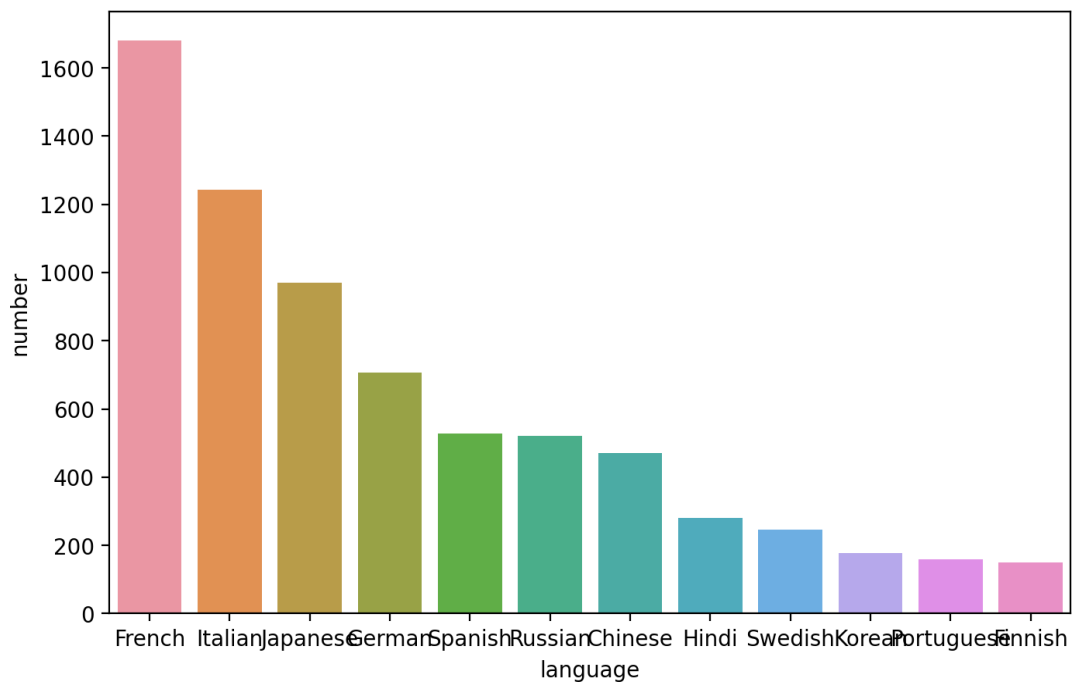
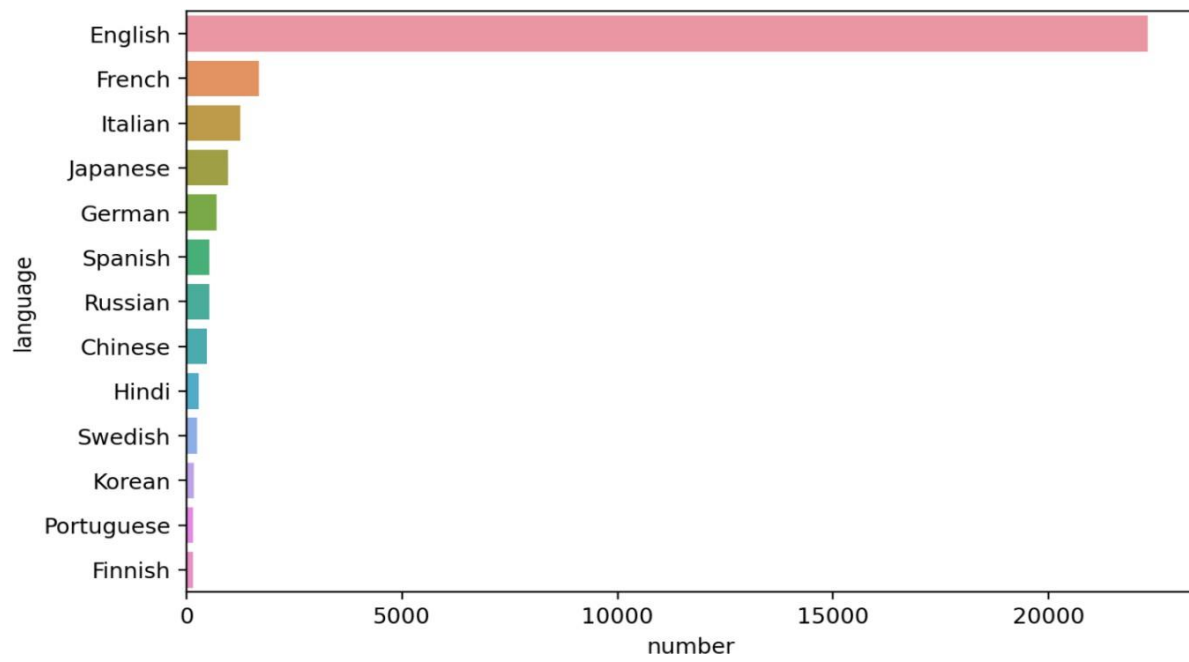
- The features like genres cast crew production companies production countries and keywords are present in the datasets as a list of dictionaries. They were flattened using literal. Val function from ast module, making a list of names. They were joined into a string using ".join function.
- All the datasets available like movies_metadata, credits, keywords and ratings were merged based on common features to make a combined dataset for easy comparison of features and visualization purpose.
- Some of the data is contained in object format where it was supposed to be int64 or datetime format. All the features without its native datatype were typecasted.
- The features and rows which contained many missing values and features which are not important for the recommendation purpose were dropped using dropna function.
- Language feature was contained in a code format which was mapped to complete language name.
- From the release year, year was extracted for visualization purpose.
- The average ratings were calculated by combining the values using movieid and sorting into descending order to get an idea which movies have received good ratings.

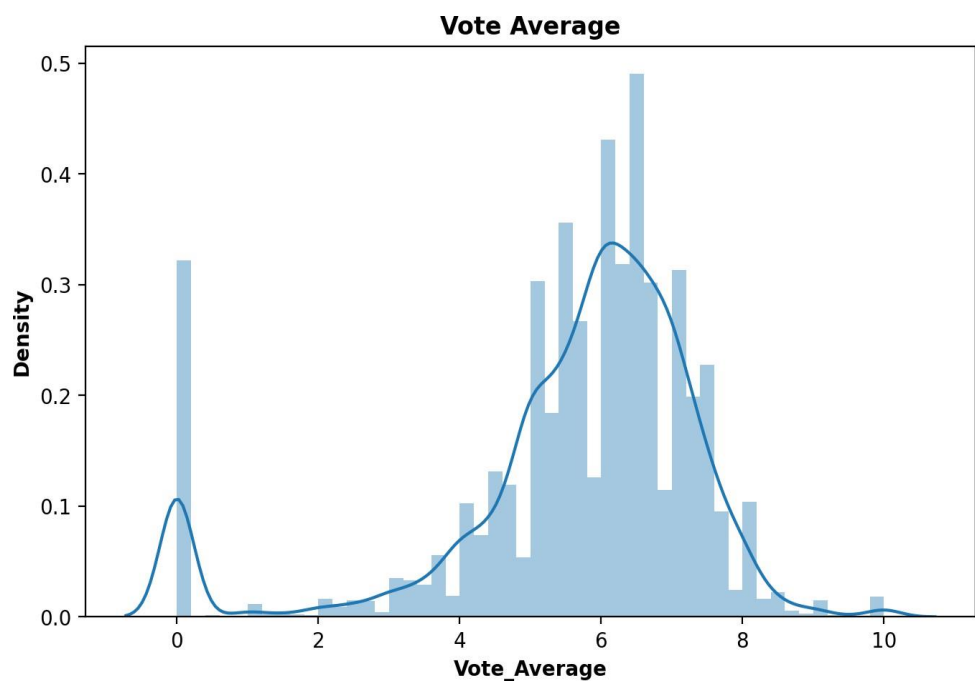
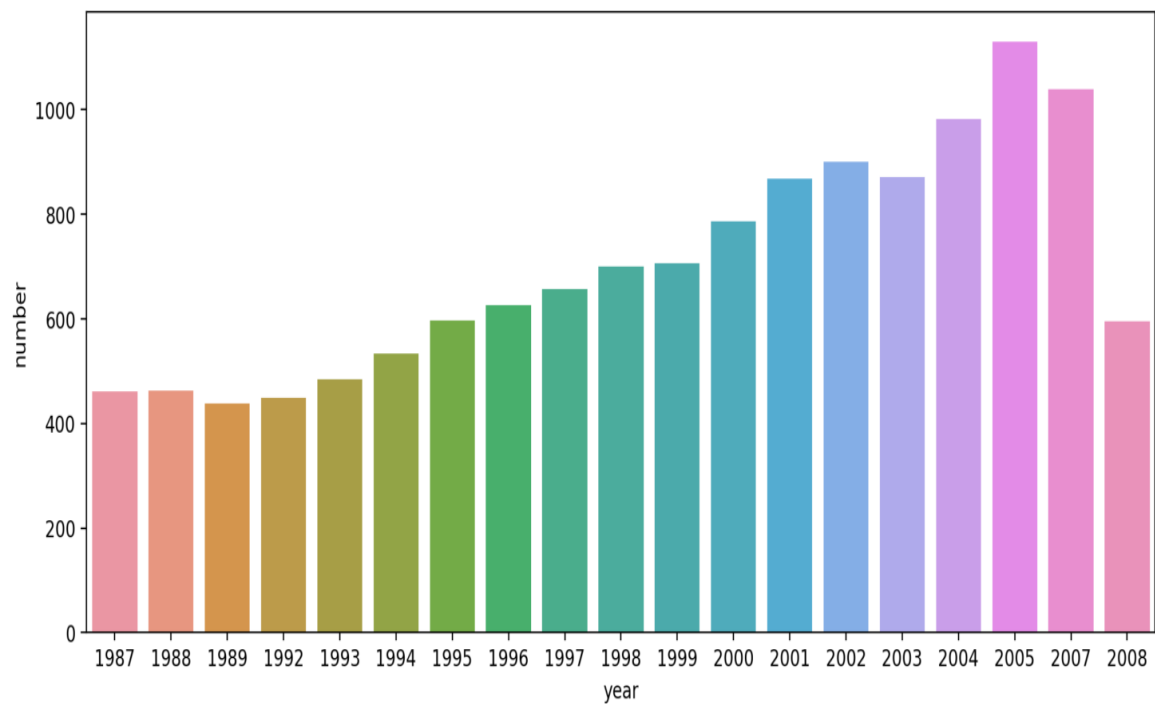
EDA: (Exploratory Data Analysis)

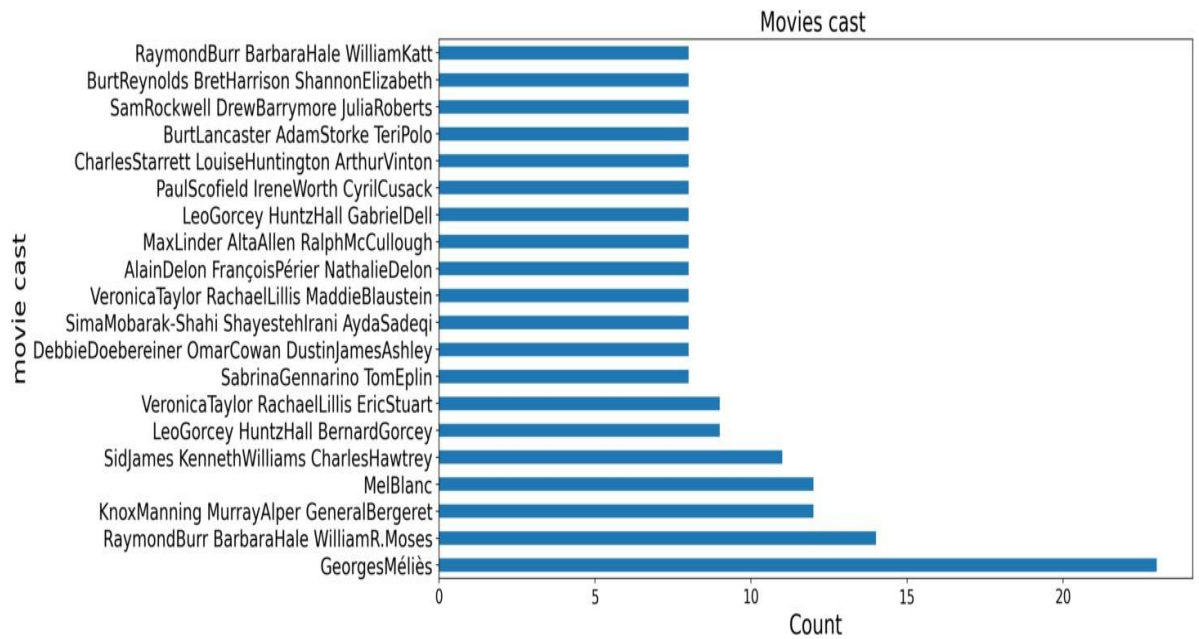
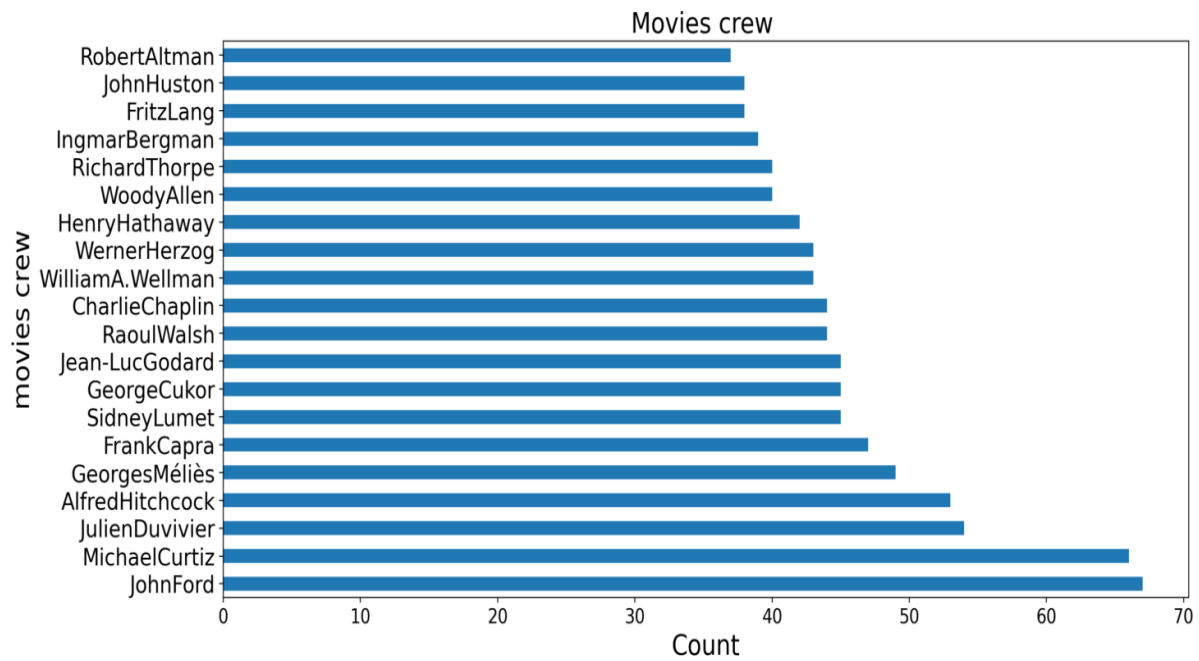
- All the features were explored and visualized using visualization techniques.
- Seaborn, matplotlib libraries were used for this purpose.
- Univariate, bivariate and multivariate analysis was done to understand the data easily and to draw meaningful insights.

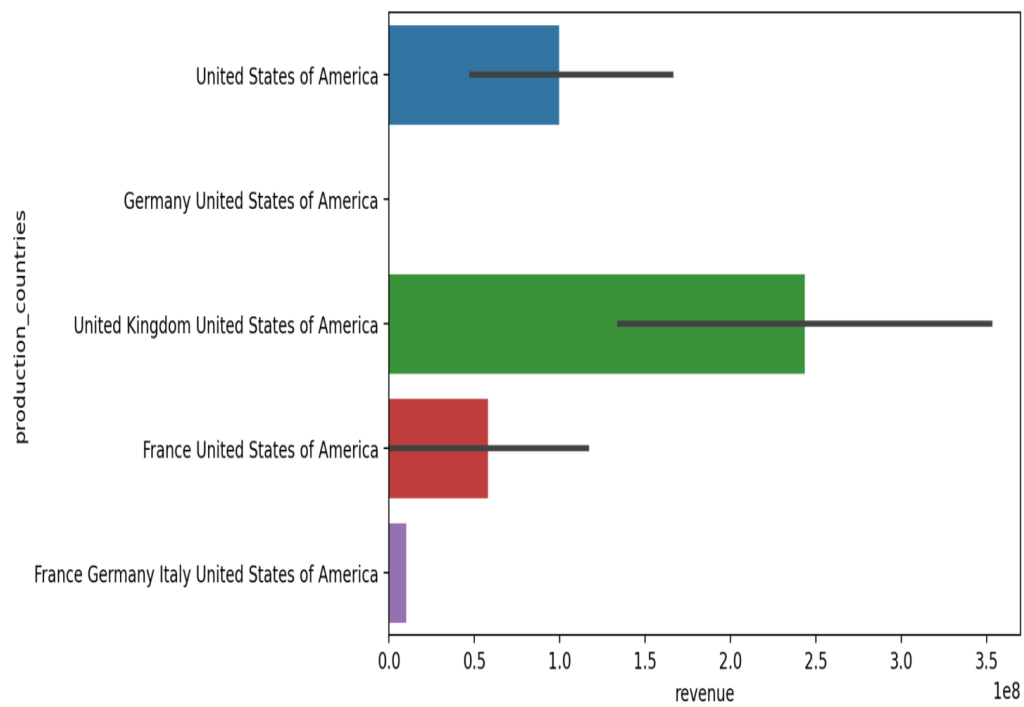
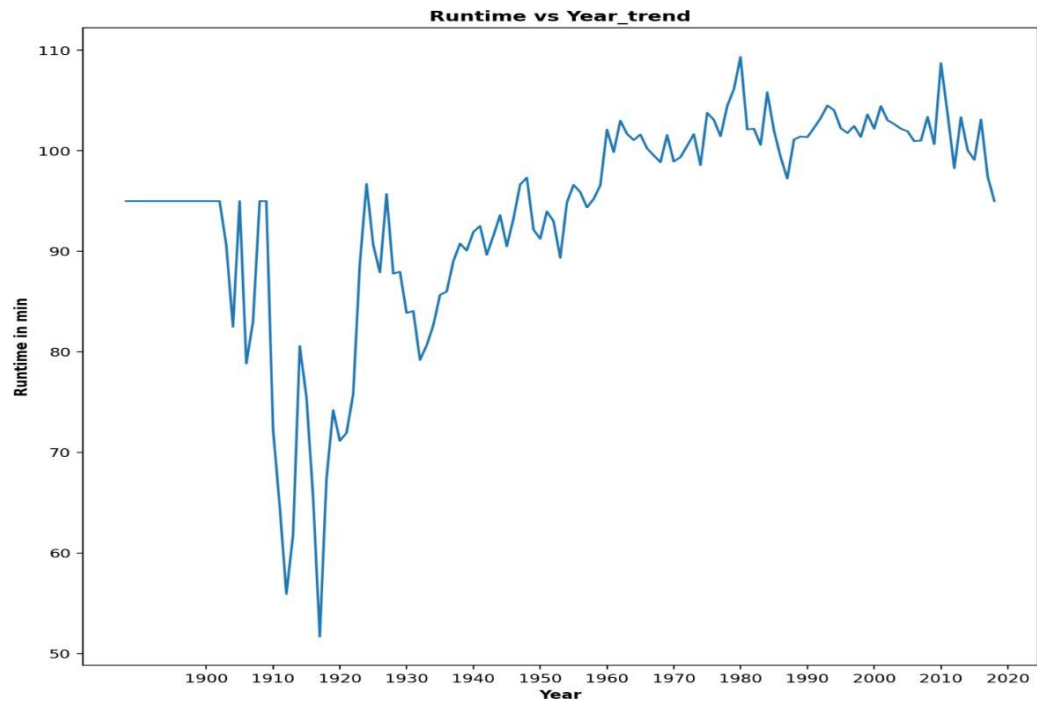
Some of plots of EDA are as below

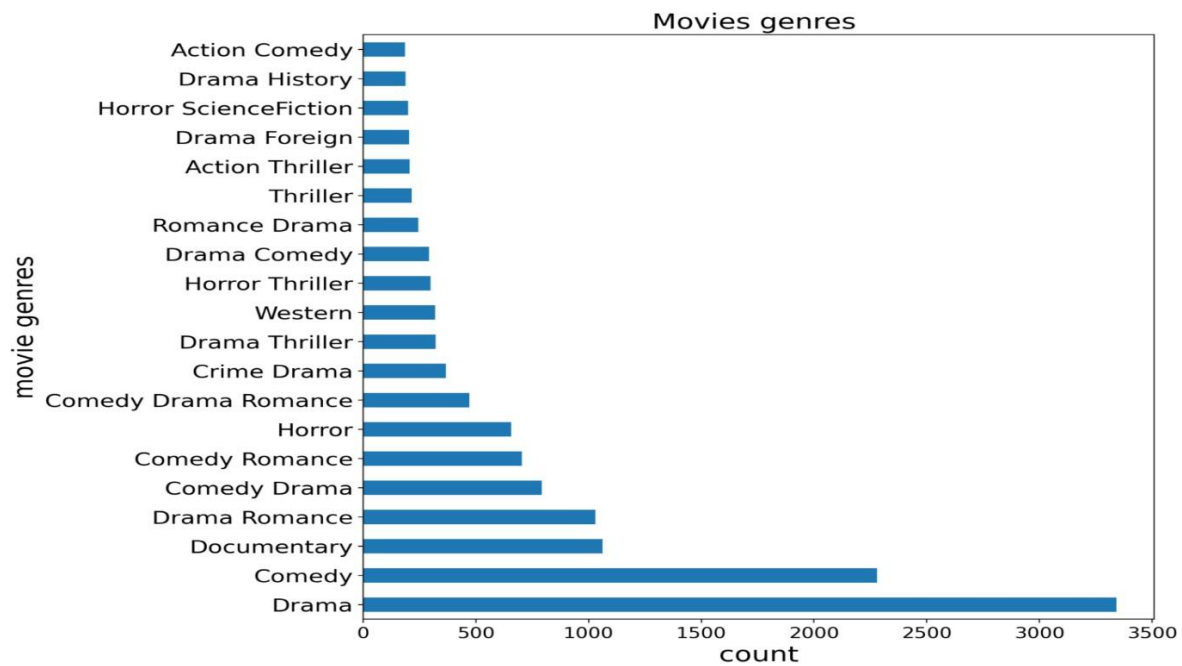
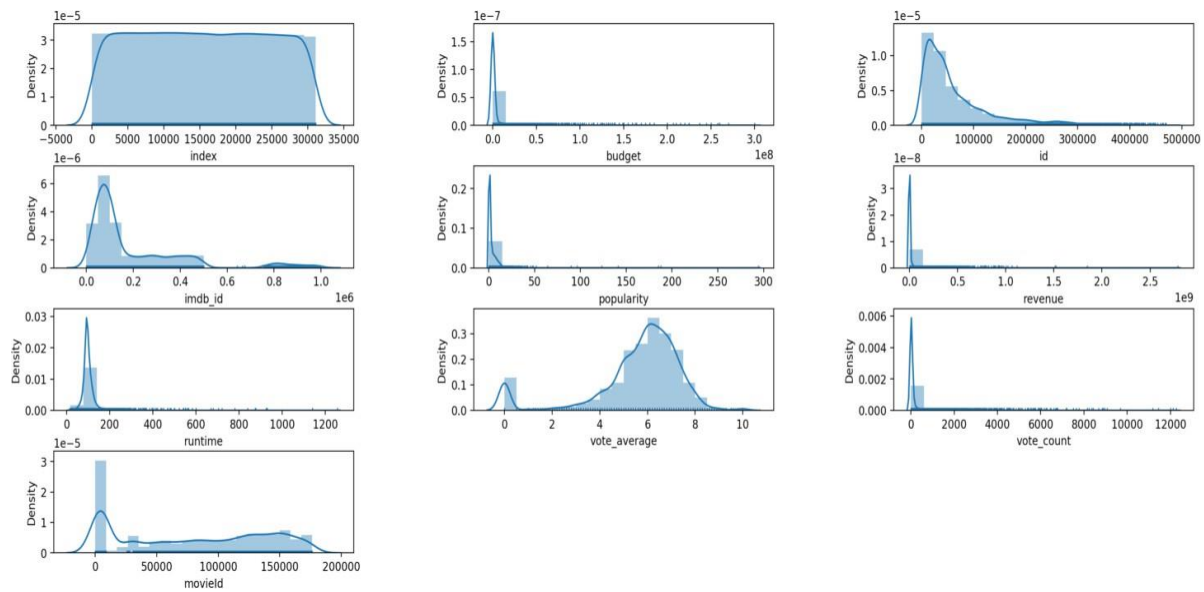


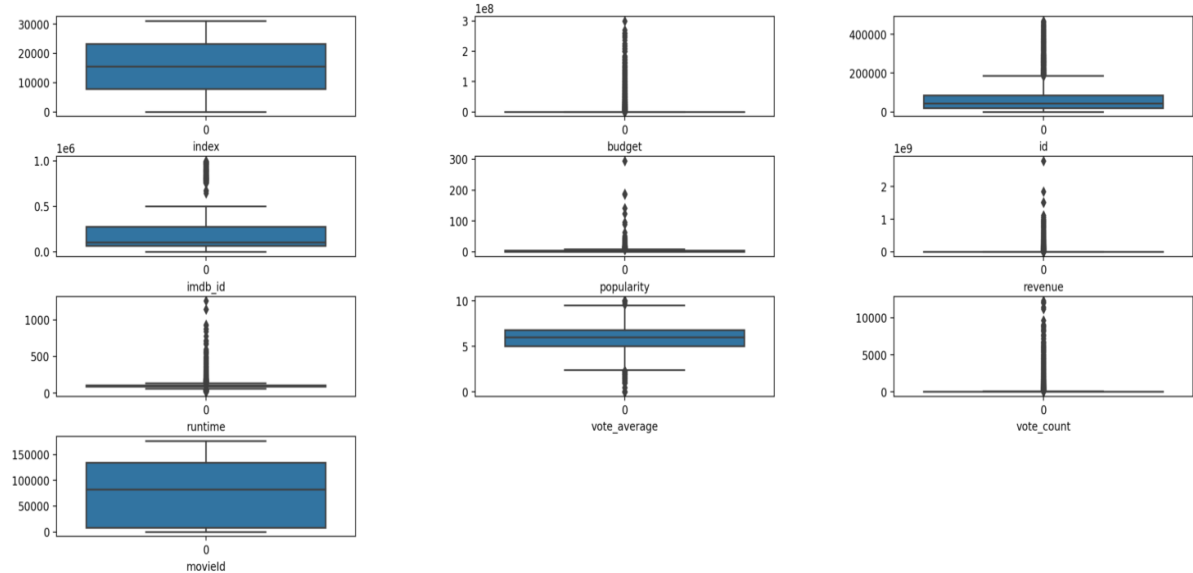












DIFFERENT TYPES OF RECOMMENDATION SYSTEM

a) Popularity based recommendation system-

- This recommendation system works by suggesting the user the movies which are trending right now.
- When the user has completely no idea of what to watch popularity-based recommendations come to rescue.
- Top websites maintain database of movies and their popularity based on user ratings and reviews.
- This eliminated the need for user preferences and other factors like genres, cast etc.

$$W = \frac{Rv + Cm}{v + m}$$

where:

W = Weighted Rating

R = average for the movie as a number from 0 to 10 (mean) = (Rating)

v = number of votes for the movie = (votes)

m = minimum votes required to be listed in the Top 250 (currently 3000)

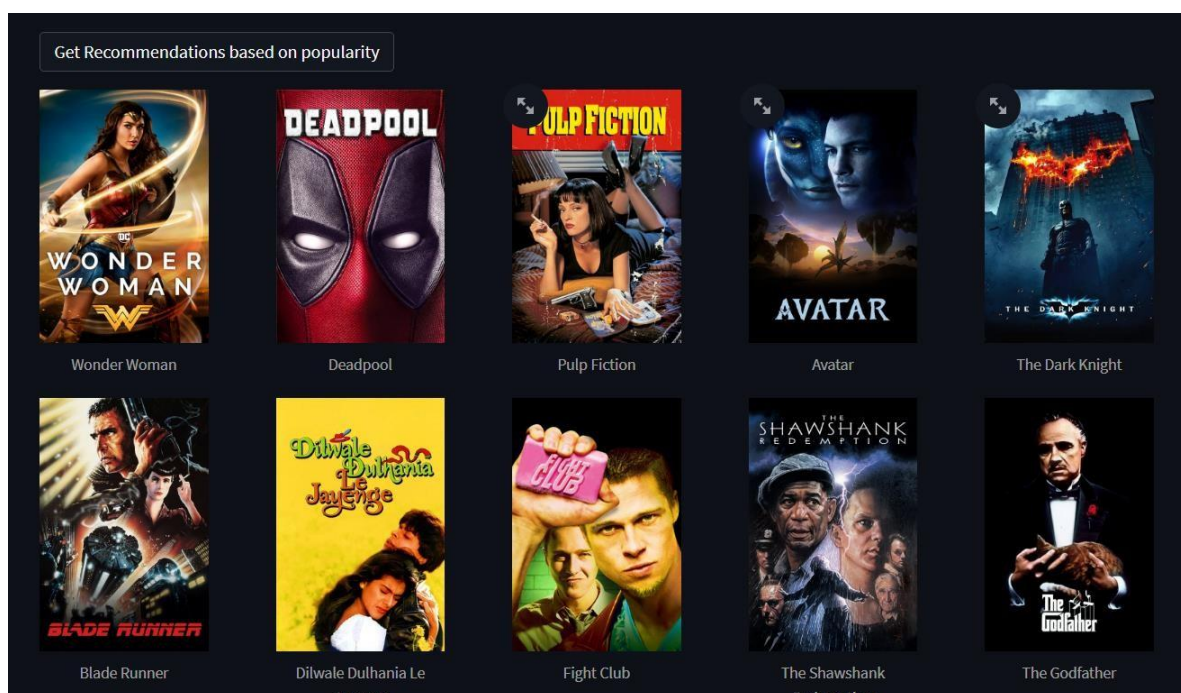
C = the mean vote across the whole report (currently 6.9)

In this type of recommendation system vote average, vote count and popularity features are used for recommendations. The features have been scaled and equal weightage was given for both weighted average and popularity as movies which were popular may not be voted good and movies which had a good vote average may not be popular. Based on score obtained by using the above formula, movies were sorted in descending order and top ten movies were recommended to the user.

Here is the code snippet for popularity-based recommendations:

```
if x == 'GET RECOMMENDATIONS':
    select = st.selectbox(label='Select the type of recommendation', options=['Popularity based recommendations'],
    if select == 'Popularity based recommendations':
        v = df['vote_count']
        R = df['vote_average']
        C = df['vote_average'].mean()
        m = df['vote_count'].quantile(0.70)
        df['weighted_average'] = ((R * v) + (C * m)) / (v + m)
        scaler = MinMaxScaler()
        movies_scaled = scaler.fit_transform(df[['weighted_average', 'popularity']])
        movies_tf = pd.DataFrame(movies_scaled, columns=['weighted_average', 'popularity'])
        df[['weight_average_tf', 'popularity_tf']] = movies_tf
        df['score'] = df['weight_average_tf']*0.5 + df['popularity_tf']*0.5
        df_pop = df.sort_values(['score'], ascending=False)
        submit = st.button('Get Recommendations based on popularity')
```

Here are the top ten recommendations:



CONTENT BASED RECOMMENDER SYSTEM

- Movies are suggested to the user based on similarity to the previous movie watched.
- This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations.
- The general idea behind these recommender systems is that if a person likes a particular item, he or she will also like an item that is similar to it.
- A good example could be YouTube, where based on your history, it suggests you new videos that you could potentially watch.

$$w_{i,j} = tf_{i,j} \cdot \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
 df_i = number of documents
 N = total number of documents

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \sqrt{\sum_{i=1}^n B_i^2}},$$

Over here id, genres, title features are used to tune the content based recommender system .

Lemmatization was applied on genres feature to remove duplicates and extract the root word in a meaningful way.

TfIdf stands for term frequency-inverse document frequency. It is a 2D data matrix where each item denotes the occurrence of a particular word in a particular document as compared to other documents. Cosine similarity is a measure of angle between two non-zero vectors of an inner product space that measures the cosine of angle between them.

Advantage

One significant advantage of this recommender engine technique is that it does not need any additional data about other users since the recommendations are specific to this user. Also, this model can capture the particular interests of a user and suggest niche objects that very few other users are interested in.

Here is a code snippet of Content based recommendation system






```
lemma = WordNetLemmatizer()
def lemmatize_text(text):
    return [lemma.lemmatize(text)]
df_c['genres'] = df_c.genres.apply(lemmatize_text)
df_c['genres'] = df_c['genres'].apply(lambda x: ' '.join(x))
tf = TfidfVectorizer(min_df=3, max_features=None, ngram_range=(1, 6), stop_words='english', analyzer='word')
tf_idf = tf.fit_transform(df_c['genres'])
sigmoid = sigmoid_kernel(tf_idf, tf_idf)
indices = pd.Series(df_c['genres'].index, index=df_c['title'])

def get_rec(title, sigmoid=sigmoid):
    idx = indices[title]
    sig_scores = list(enumerate(sigmoid[idx]))
    sig_scores1 = sorted(sig_scores, key=lambda x: x[1], reverse=True)
    sig_scores2 = sig_scores1[1:11]
    movie_indices = [i[0] for i in sig_scores2]
    return df_c['title'].iloc[movie_indices]
```




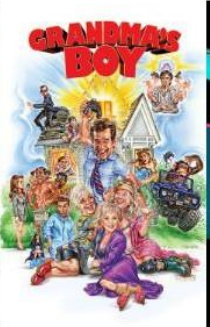


Here are the top ten recommended movies for Jumanji

Jumanji ▼

Get recommendations based on content



Dungeons & Dragons Geri's Game eXistenZ Stay Alive The Last Starfighter



Peter Pan Grandma's Boy Manhunter Panic Room Love Me If You Dare

ITEM BASED COLLABORATIVE FILTERING:

The recommender system tries to find out items based on previous user preferences of the user and then recommend similar items to the user. These items might be of interest to the user.

Advantage

One significant advantage of collaborative filtering is that it doesn't need to analyze or understand the object (products, films, books) to recommend complex items precisely. There is no dependence on analyzable machine content, which means it chooses recommendations based on what it knows about the user.

Here is the code snippet for collaborative filtering:-
Item based collaborative filtering:-

```
elif select == 'Item based Collaborative filtering':
    movies_id = ratings['movieId'].unique()
    df_l = df[df['id'].isin(movies_id)]
    ratings1 = ratings[ratings['movieId'].isin(df['id'])]
    ids = st.selectbox('Please select a movie id', options=df_l.id.to_list())

    def create_matrix(df):
        p = len(df['movieId'].unique())
        q = len(df['userId'].unique())

        map_user = dict(zip(np.unique(df["userId"]), list(range(q))))
        map_movie = dict(zip(np.unique(df["movieId"]), list(range(p))))

        map_user_i = dict(zip(list(range(q)), np.unique(df["userId"])))
        map_mov_i = dict(zip(list(range(p)), np.unique(df["movieId"])))

        user_index = [map_user[i] for i in df['userId']]
```

```
def find_similar_movies(movie_id, matrix, k, metric='cosine', show_distance=False):

    neighbour_ids = []

    movie_ind = map_movie[movie_id]
    movie_vec = matrix[movie_ind]
    k += 1
    kNN = NearestNeighbors(n_neighbors=k, algorithm="brute", metric=metric)
    kNN.fit(matrix)
    movie_vec = movie_vec.reshape(1, -1)
    neighbour = kNN.kneighbors(movie_vec, return_distance=show_distance)
    for i in range(0, k):
        n = neighbour.item(i)
        neighbour_ids.append(map_mov_i[n])
    neighbour_ids.pop(0)
    return neighbour_ids
```

User based collaborative filtering:

```
elif select=='User based Collaborative filtering':
    movies_id = ratings['movieId'].unique()
    df_l = df[df['id'].isin(movies_id)]
    ratings1 = ratings[ratings['movieId'].isin(df_l['id'])]
    ids = st.selectbox('Please select a user id', options=ratings1['userId'].unique())
    rating_matrix = ratings1.pivot_table(index='userId', columns='movieId', values='rating')
    rating_matrix = rating_matrix.fillna(0)

    def sim(user_id, r_matrix, k=10):
        user = r_matrix[r_matrix.index == user_id]
        other_users = r_matrix[r_matrix.index != user_id]
        sim = cosine_similarity(user, other_users)[0].tolist()
        idx = other_users.index.tolist()
        idx_sim = dict(zip(idx, sim))
        idx_sim_sorted = sorted(idx_sim.items(), key=lambda x: x[1])
        idx_sim_sorted.reverse()
        top_user_similarities = idx_sim_sorted[:10]

    def recommend_movie(user_index, similar_user_indices, r_matrix, items=10):
        similar_users = r_matrix[r_matrix.index.isin(similar_user_indices)]
        similar_users = similar_users.mean(axis=0)
        similar_df = pd.DataFrame(similar_users, columns=['mean'])
        user_df = r_matrix[r_matrix.index == user_index]
        user_df_transposed = user_df.transpose()
        user_df_transposed.columns = ['rating']
        user_df_transposed = user_df_transposed[user_df_transposed['rating'] == 0]
        movies_unseen = user_df_transposed.index.tolist()
        similar_users_filtered = similar_df[similar_df.index.isin(movies_unseen)]
        similar_users_ordered = similar_df.sort_values(by=['mean'], ascending=False)

        top_movies = similar_users_ordered.head(items)
        top_movie_indices = top_movies.index.tolist()
        movie_title = df_l[df_l['id'].isin(top_movie_indices)][['title']]
        movie_id = df_l[df_l['id'].isin(top_movie_indices)][['id']]

        return list(zip(movie_title, movie_id))
```


Here are the top ten recommendations for movied 949:

949


Get collaborative filtered recommendations

Since you watched Heat


Following are the top ten recommendations for you




Halloween



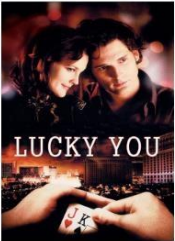
Good Neighbor Sam




Nell



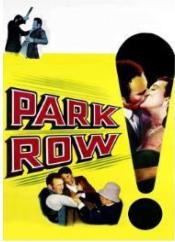
Adaptation.




Lucky You




eXistenZ



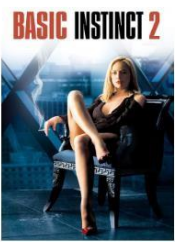
Park Row



Galaxy Quest



Laura



BASIC INSTINCT 2


Here are the top ten recommendations based for userId:1:

Please select a user id

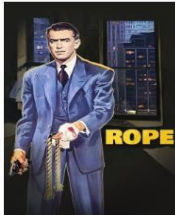
1

Get recommendations based on movies watched

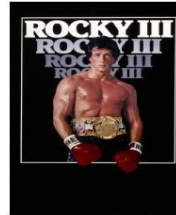
Following are the top ten recommendations for you



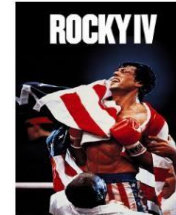
The 39 Steps




Rope



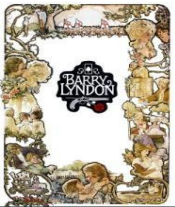
Rocky III




Rocky IV



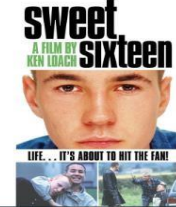
American Pie




The 400 Blows




The Man With the Golden Arm



sweet sixteen



WILL PENNY



PERSEPOLIS

Hybrid Technique:

The hybrid recommendation system is a special type of recommendation system that used data of both collaborative data and content-based data simultaneously which helps to suggest a similar or close item to the users. Combining the two above approaches helps to resolve the bugs from individual recommender systems. In this, the system suggests similar items which are already used by the user or suggests the items which are likely to be used by another user with some similarities. The model is trained in such a way that it has both the functionality of content-based and collaborative filtering techniques.

Below is the code snippet of hybrid technique:

```
def hybrid(userId, title):
    idx = indices[title]
    tmdbId = id_map.loc[title]['id']
    movie_id = id_map.loc[title]['movieId']

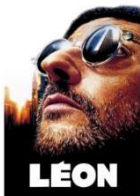
    sim_scores = list(enumerate(sigmoid(int(idx))))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:26]
    movie_indices = [i[0] for i in sim_scores]

    movies = df_c.iloc[movie_indices][['title', 'id']]
    movies['est'] = movies['id'].apply(lambda x: svd.predict(userId, indices_map.loc[x]['movieId']).est)
    movies = movies.sort_values('est', ascending=False)
    return movies.head(10)

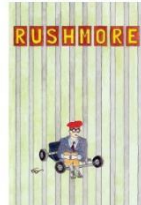
submit = st.button('Get Recommendations')
if submit:
    h = hybrid(userid,movie)
    st.write(f"Following are the top ten recommendations for you based on Hybrid technique")
```

Below are the recommendations for userid:1 for Jumanji movie based on hybrid approach::

Following are the top ten recommendations for you based on Hybrid technique



Leon: The Professional



Rushmore



The Unforgiven



Collateral



White Palace



The Trip



Mambo Italiano



Piranha 3D



Cape Fear



Go Fish

Libraries Used:-

- NumPy
- Matplotlib
- Seaborn
- Sklearn
- Wordcloud
- Pandas
- Plotly

- Requests
- Json
- Nltk
- SciPy
- PIL
- Surprise

Technologies Used:

- Python
- NLP concepts like Tf-Idf
- Machine learning

CHALLENGES FACED:

a)Typecasting of features.

Some columns contained mixed datatypes with categorical and datetime where the actual datatype is int64.It was handled by deleting the rows with datetime and was typecasted to integer.

b)While running visualizations on streamlit some of the plots were not supported. They were handled by changing the plots which had support

c)While doing content based filtering, due to large number of rows in the features column error was thrown which indicated lack of memory. Hence we have used overview feature after doing all necessary preprocessing in order to allocate memory to run within the limits available.

D)While running the applications for some input values key value error and other types of errors have been raised. Initially it was a challenge to figure it out but later understood that some of the movie titles were same maybe some movies released with similar movie names in different years. All the duplicated values are dropped by which error was resolved.

FUTURE SCOPE:

- Deep learning techniques can be used to improve the accuracy of predictions of our model.
- Database can be integrated to store the user inputs and preferences to provide better recommendations. Also new movies need to be updated regularly so that user can also get recommendations for that.
- This techniques can be integrated with similar apps like YouTube, Spotify, etc. to generate recommendations.

GITHUB Link: -

[https://github.com/9505669725/Movie_recommendation_system.g](https://github.com/9505669725/Movie_recommendation_system.git)

it