ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

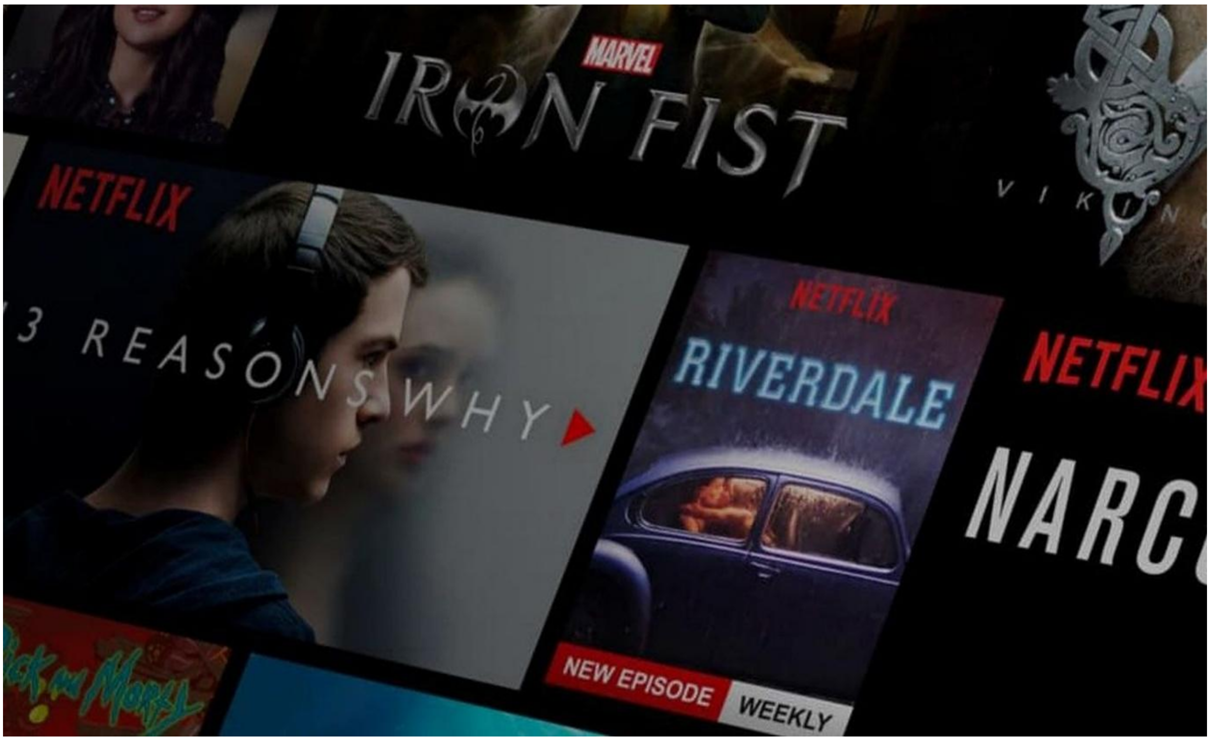
NAME – NIKHIL SINGH JADON

ENROLL. – E23CSEU0040

BATCH 2ND & GRP. 1ST

Introduction:

"This project focuses on developing a personalized movie recommendation system using content-based filtering techniques. By analyzing genres, cast, and other attributes, the system recommends movies similar to those a user has shown interest in, enhancing user engagement by tailoring suggestions."



ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

1. Survey of Related Work and Methodologies :-

Movie recommendation systems have become essential in modern entertainment platforms, using a blend of data science and machine learning to enhance user experiences. These systems are primarily built on two foundational methods: collaborative filtering and content-based filtering. Collaborative filtering, as seen in projects like Lokesh2525's system, leverages user interactions, such as ratings and viewing patterns, to make recommendations based on similar user preferences. This approach can effectively predict user behavior but often struggles with the "cold-start" problem, where new users or items have limited data.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

Github repositories :-

1. **Hybrid Recommendation with Vector Embeddings**
   - **Repository:** Movie Recommender by adinmg
   - **Techniques:** Hybrid model with collaborative filtering and vector embeddings using Sentence Transformers and Chroma DB.
   - **Visualization:** Insightful graphs for user preferences and vector-based similarities.
2. **TMDB-Based Content Filtering with TF-IDF**
   - **Repository:** Vivekecoder's Content-Based System
   - **Techniques:** TF-IDF and cosine similarity for genre and plot-based recommendations.
   - **Visualization:** Basic EDA with genre and rating distributions.
3. **Multi-Approach Recommender with KNN and Deep Learning**
   - **Repository:** ChiefYuHan's Multi-Approach Recommender
   - **Techniques:** KNN, collaborative filtering, and deep learning.
   - **Visualization:** Detailed user interaction analysis and genre popularity.
4. **TMDB-Based Recommender with Genre and Cast**

- **Repository:** Shenile’s Content-Based System
- **Techniques:** Recommends based on genre, cast, and keywords.
- **Visualization:** Distribution of genres, rating frequencies, and popular movies.

5. Streamlit Integrated Recommender Using OMDB Data

- **Repository:** Rahulbanjara’s OMDB System
- **Techniques:** Content-based filtering with cosine similarity.
- **Visualization:** Interactive genre, rating trends, and Streamlit app visuals.

Additional 5 Repositories with Visualization Focus :-

6. Profit and Genre-Based Analysis
   - **Repository:** Yashashvee11's System
   - **Techniques:** Genre profitability analysis and content-based recommendations.
   - **Visualization:** Genre-based earnings, language profitability, and gross-profit comparisons
7. Content and Similarity Visualization
   - **Repository:** LynnFernandes23’s Recommender
   - **Techniques:** NLTK and TF-IDF for genre and cast similarity.
   - **Visualization:** Actor and director insights, genre relationships, and content-based trends
8. Revenue and Profit Prediction
   - **Repository:** DSCKGEC’s EDA System
   - **Techniques:** Cosine similarity and revenue prediction models.

- **Visualization:** Revenue prediction trends, profit analysis, and genre popularity distribution

9. Collaborative Filtering Heatmaps
   - **Repository:** Lokesh2525's Collaborative Recommender
   - **Techniques:** User-based collaborative filtering.
   - **Visualization:** Heatmaps of user ratings, genre preferences, and rating patterns

10. Interactive Genre and Actor Visuals

- **Repository:** Subtledhawal's Recommender
- **Techniques:** Cosine similarity and genre filtering with Streamlit.
- **Visualization:** Interactive genre, actor relationships, and personalized movie visuals

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

Data Preprocessing Steps for Movie Recommendation System

1. Merging Datasets

The movies and credits datasets were merged on the common column title to combine movie information with cast and crew details.

2. Selecting Relevant Columns

After merging, only the necessary columns were retained for analysis: movie_id, title, overview, genres, keywords, cast, and crew.

3. Handling Missing Values

Checked for null values across all columns to identify any data gaps. Removed rows with missing values to maintain data integrity in the recommendation model.

4. Removing Duplicates

Verified and removed any duplicate rows, ensuring only unique movie entries remained.

5. Extracting Genres and Keywords

The genres and keywords columns contained nested data, which was converted into a list of genre and keyword names using the ast.literal_eval() function.

6. Extracting Top 3 Cast Members

In the cast column, only the top three actors were selected to simplify cast information and reduce data complexity.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

AND MANY STEPS ARE TAKEN CODE SNIPPETS ARE GIVEN BELOW

```
[27] movies = movies[['movie_id','title','overview','genres','keywords','cast','crew']]
```

```
[31] movies.isnull().sum()
```

	0
movie_id	0
title	0
overview	0
genres	0
keywords	0
cast	0
crew	0

dtype: int64

```
[32] movies.dropna(inplace=True)
```

```
movies.duplicated().sum()
```

```
0
```

```
[48] movies.iloc[0].genres
```

```
[24] movies = movies.merge(credits,on='title' )
```

```
[26] movies.head(1)
```

	budget	genres	homepage	id	keywords	original_language	origi
0	237000000	[{"id": 28, "name": "Action"}, {"id": 12, "nam...	http://www.avatarmovie.com/	19995	[{"id": 1463, "name": "culture clash"}, {"id":...	en	

1 rows × 23 columns

:-

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

```
movies.iloc[0].genres
```

```
'[{"id": 28, "name": "Action"}, {"id": 12, "name": "Adventure"}, {"id": 14, "name": "Fantasy"}, {"id": 878, "name": "Science Fiction"}]'
```

```
[53] import ast
     def convert(obj):
         L = []
         for i in ast.literal_eval(obj):
           L.append(i['name'])
         return L
```

```
[55] movies['genres'] = movies['genres'].apply(convert)
```

```
[59] movies['keywords'] = movies['keywords'].apply(convert)
```

```
[62] import ast
     def convert(obj):
         L = []
         counter = 0
         for i in ast.literal_eval(obj):
             if counter != 3:
               L.append(i['name'])
               counter+=1
             else:
                  break
         return L
```

```
[66] movies['cast'] = movies['cast'].apply(convert)
```

```
[94] movies['tags'] = movies['overview'] + movies['genres'] + movies['keywords'] + movies['cast'] + movies['crew']
```

```
[95] movies.head()
```

	movie_id	title	overview	genres	keywords	
0	19995	Avatar	[In, the, 22nd, century,, a, paraplegic, Marin...	[Action, Adventure, Fantasy, ScienceFiction]	[cultureclash, future, spacewar, spacecolony, ...	[SamWorthin
1	285	Pirates of the Caribbean: At World's End	[Captain, Barbossa,, long, believed, to, be, d...	[Adventure, Fantasy, Action]	[ocean, drugabuse, exoticisland, eastindiatrad...	[JohnnyDe
2	206647	Spectre	[A, cryptic, message, from, Bond's, past, send...	[Action, Adventure, Crime]	[spy, basedonnovel, secretagent, sequel, mi6, ...	[DanielCra
3	49026	The Dark Knight Rises	[Following, the, death, of, District, Attorney...	[Action, Crime, Drama, Thriller]	[dccomics, crimefighter, terrorist, secretiden...	[ChristianB
4	49529	John Carter	[John, Carter, is, a, war-weary,, former, mili...	[Action, Adventure, ScienceFiction]	[basedonnovel, mars, medallion, spacetravel, p...	[TaylorK

```
[96] movies = movies.drop(columns=['overview'])
```

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

```
[98] new_df = movies[['movie_id','title','tags']]
```

```
[103] new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))
```

```
<ipython-input-103-ad44e9ca1347>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['tags'] = new_df['tags'].apply(lambda x:" ".join(x))
```

```
[114] new_df['tags'] = new_df['tags'].apply(lambda x:x.lower())
```

```
<ipython-input-114-8b60b591a07f>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  new_df['tags'] = new_df['tags'].apply(lambda x:x.lower())
```

```
[53] import ast
     def convert(obj):
         L = []
         for i in ast.literal_eval(obj):
           L.append(i['name'])
         return L
```

```
[55] movies['genres'] = movies['genres'].apply(convert)
```

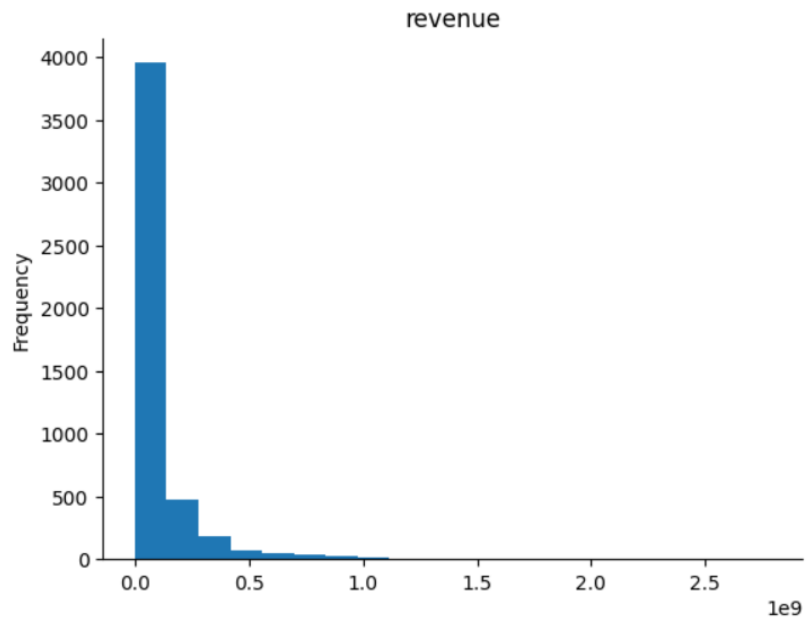
```
[59] movies['keywords'] = movies['keywords'].apply(convert)
```

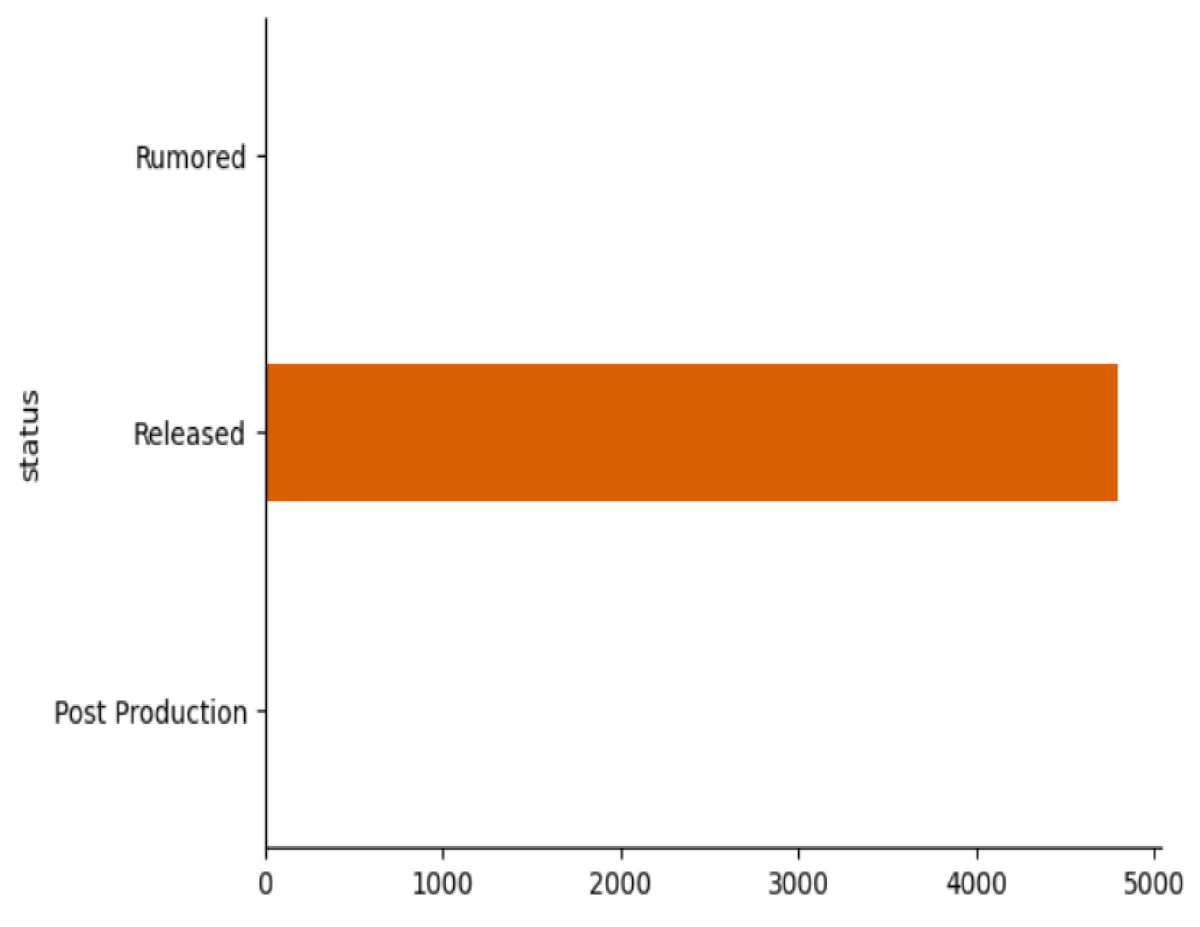
```
import ast
def convert(obj):
    L = []
    counter = 0
    for i in ast.literal_eval(obj):
        if counter != 3:
          L.append(i['name'])
          counter+=1
        else:
              break
    return L
```

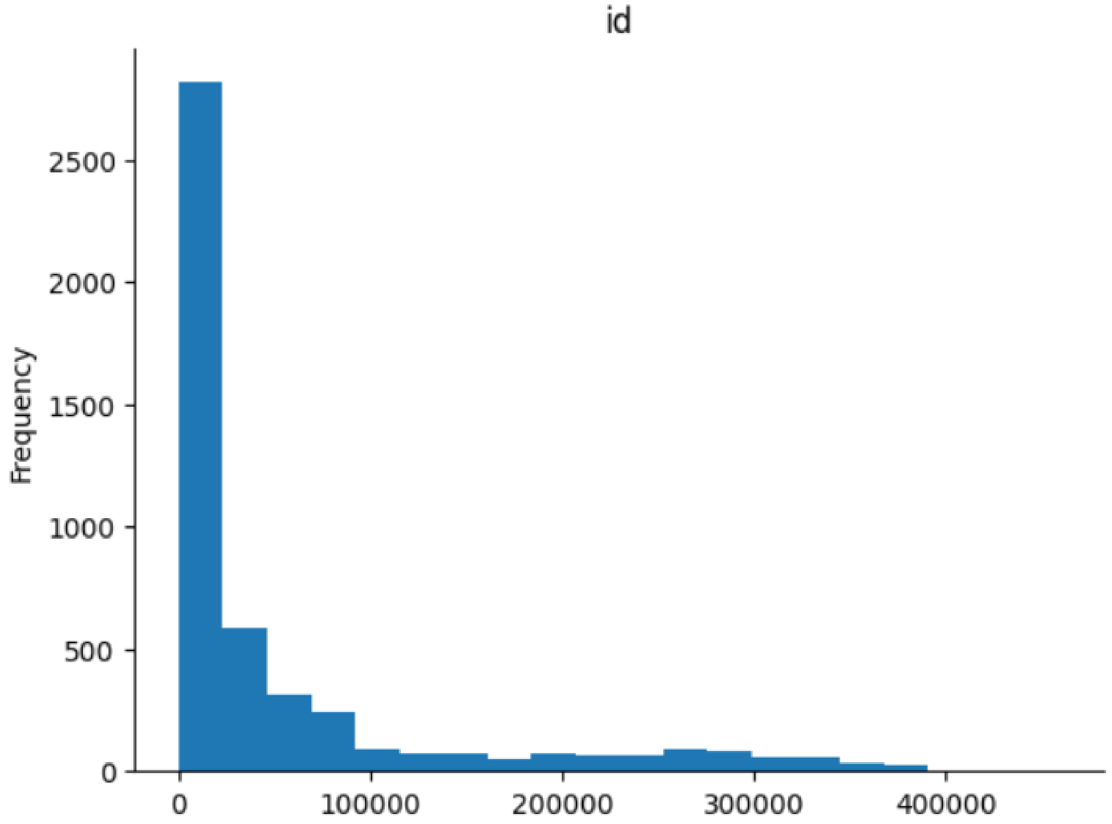
ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

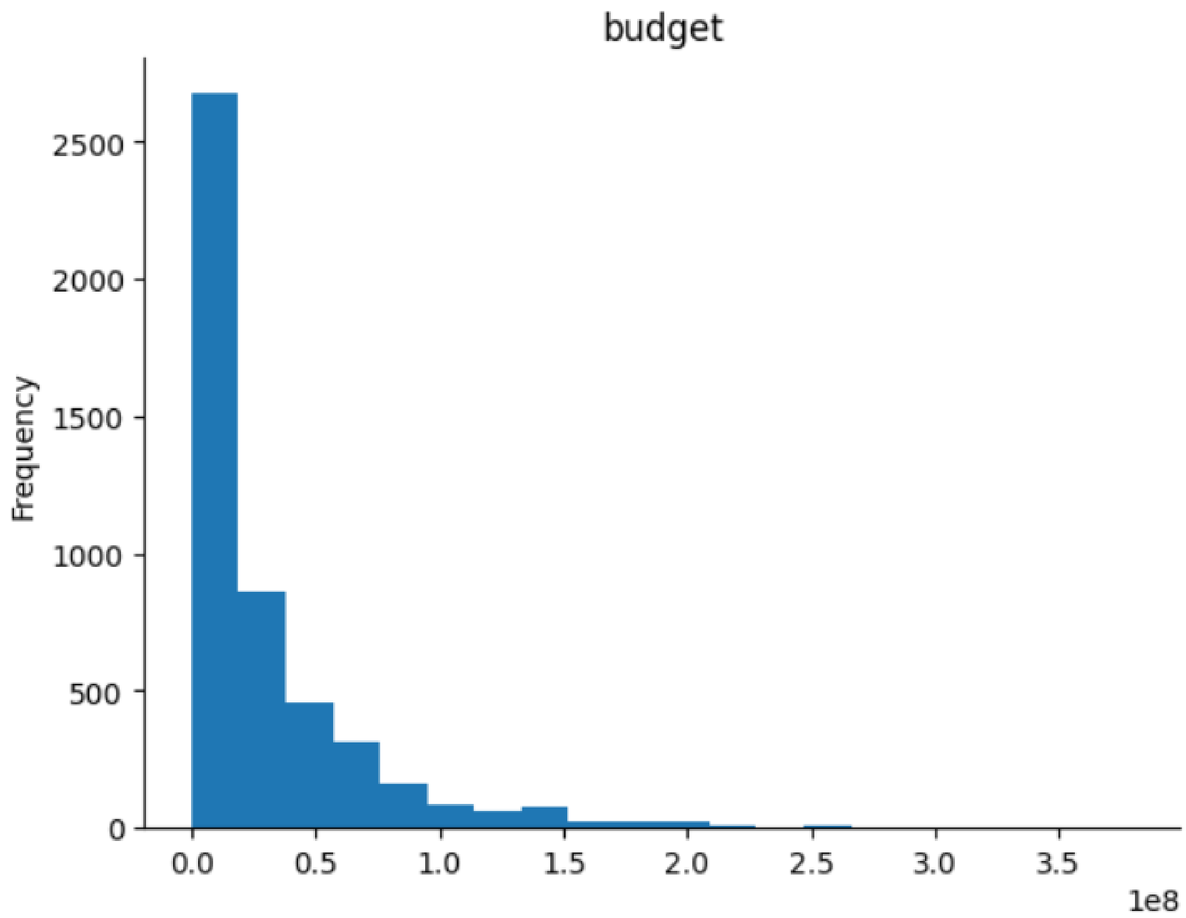
MILESTONE -2

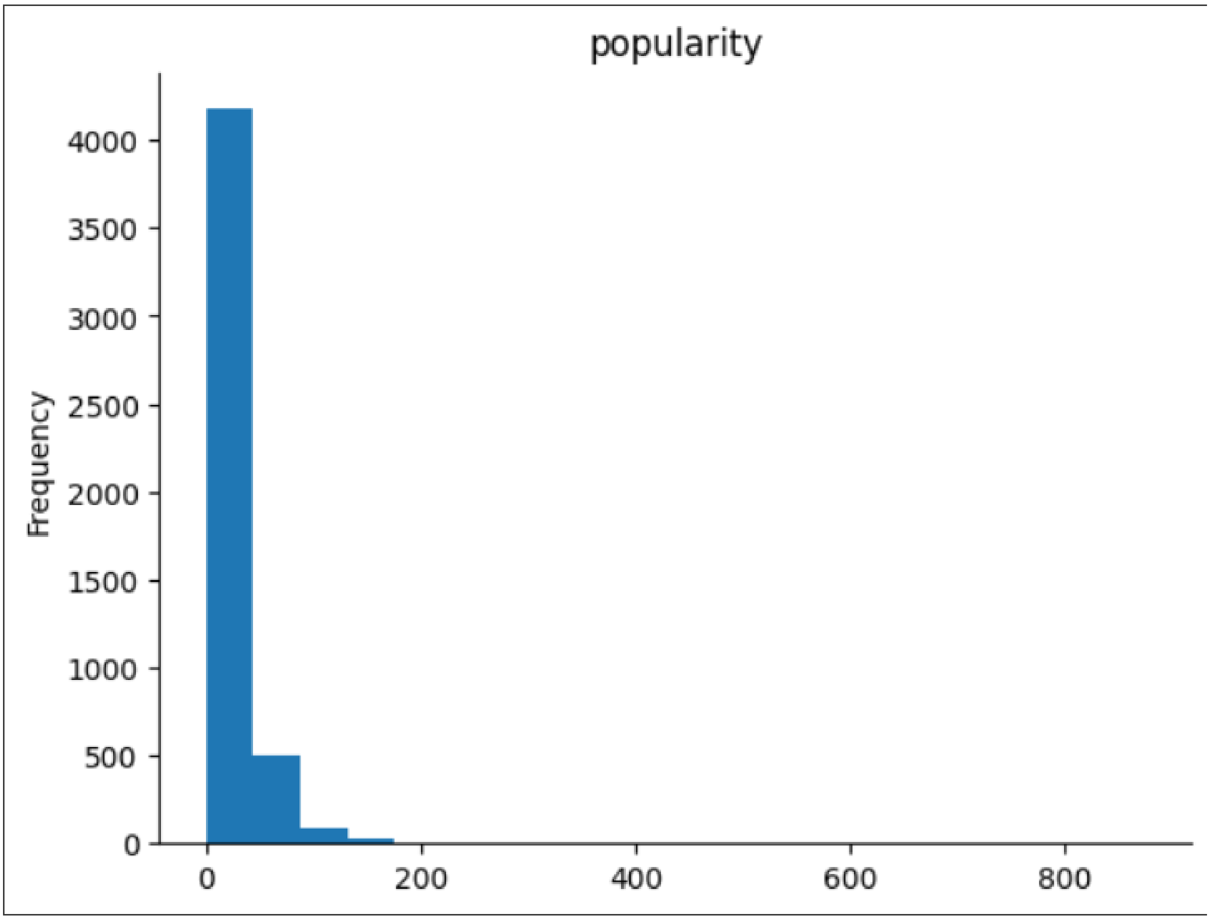
DATA VISUALISATIONS ARE GIVEN BELOW :-

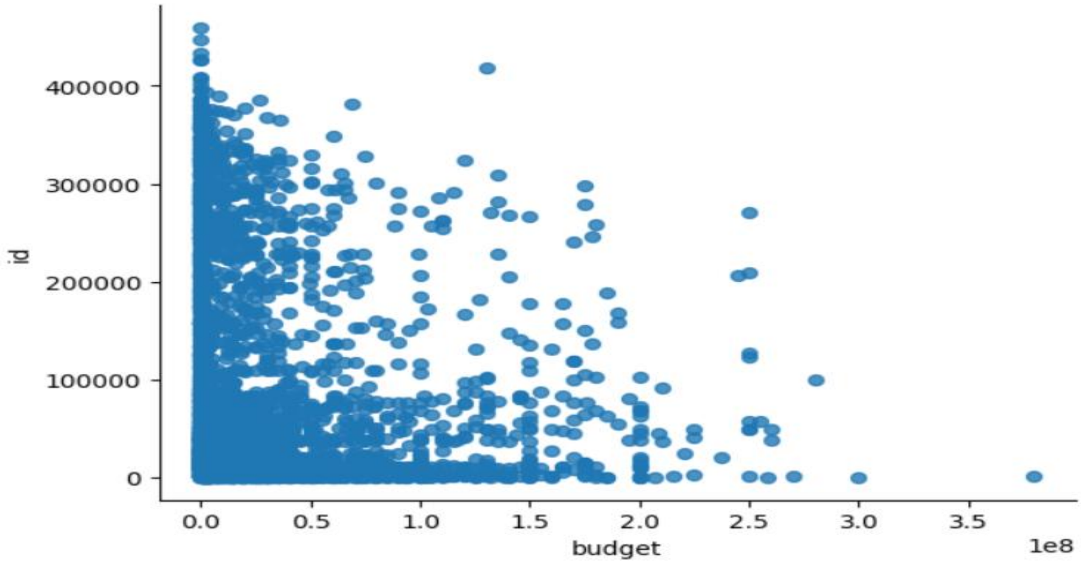


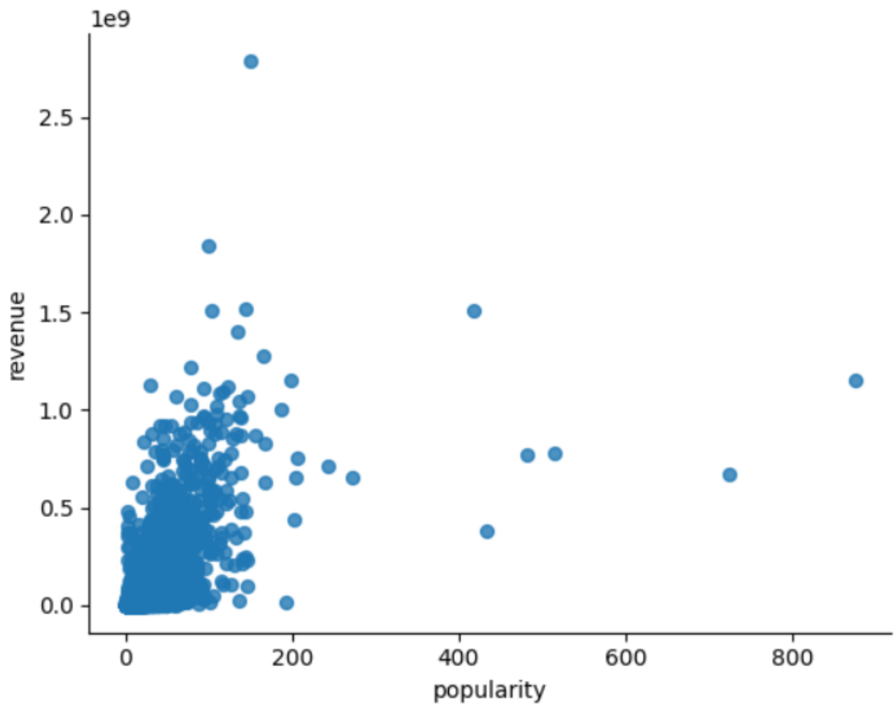


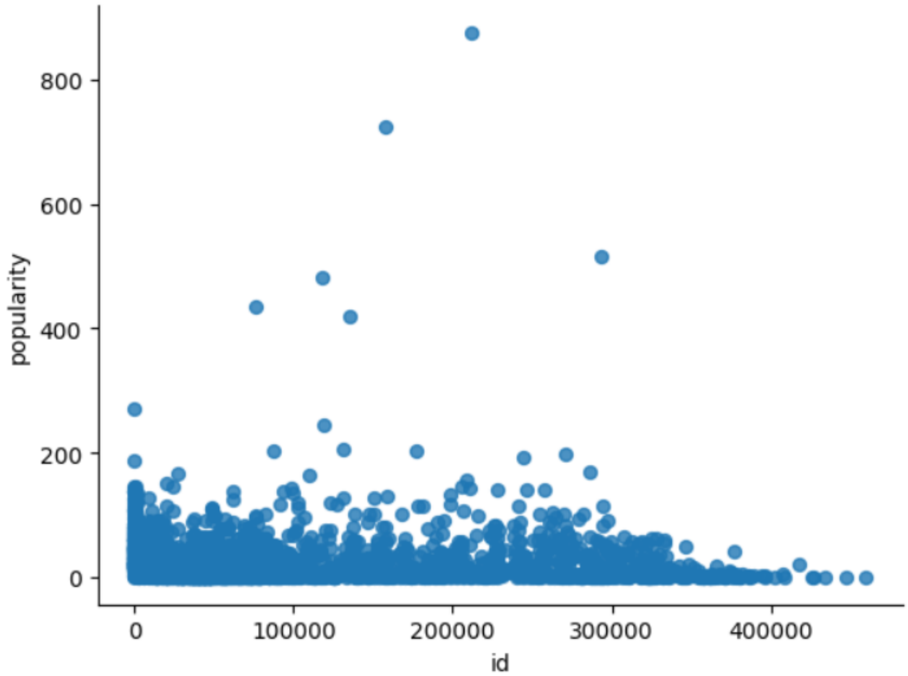

id
Frequency
2500
2000
1500
1000
500
0
0
100000
200000
300000
400000


budget
Frequency
2500
2000
1500
1000
500
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e8

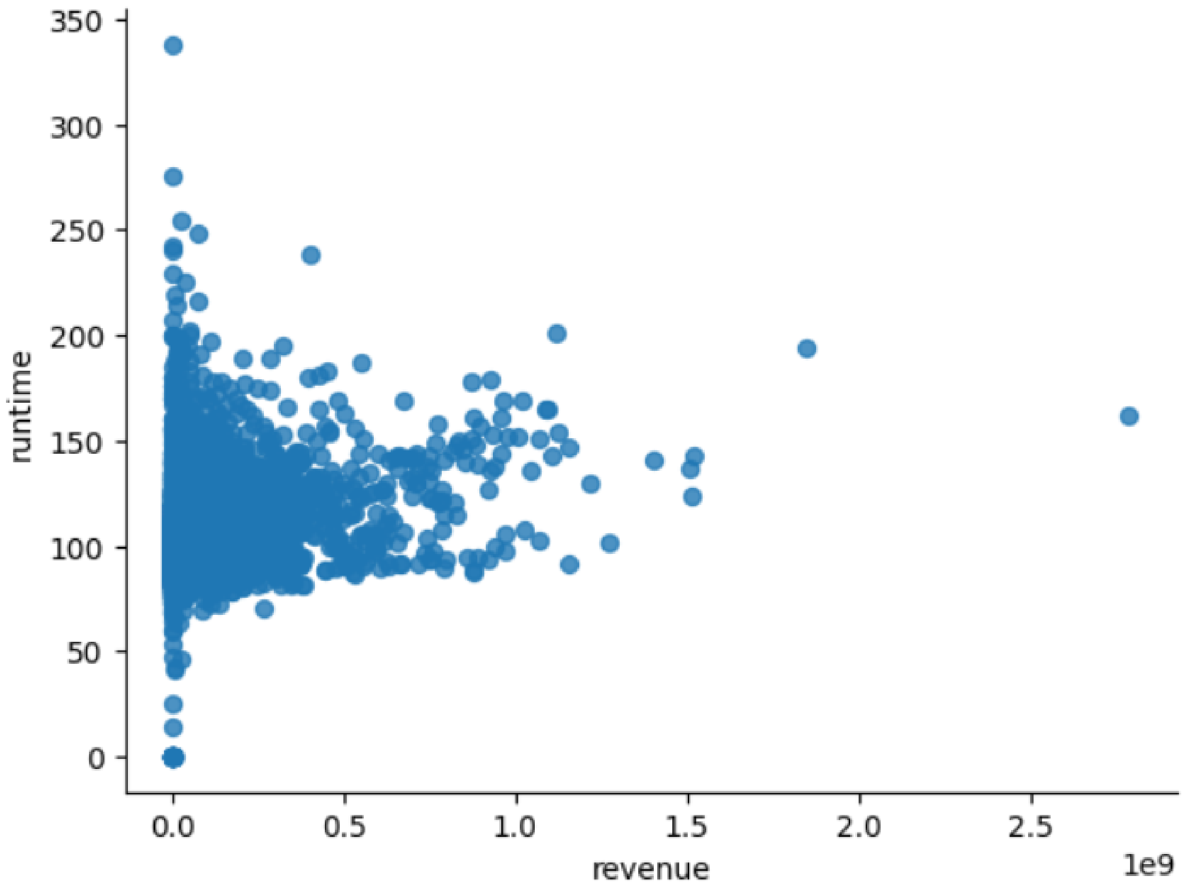

popularity
Frequency
4000
3500
3000
2500
2000
1500
1000
500
0
0
200
400
600
800


400000
300000
200000
100000
0
id
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e8
budget


1e9
2.5
2.0
1.5
1.0
0.5
0.0
revenue
0
200
400
600
800
popularity



800
600
400
200
0
popularity
0
100000
200000
300000
400000
id



350
300
250
200
150
100
50
0
runtime
0.0
0.5
1.0
1.5
2.0
2.5
1e9
revenue

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

DATA TRAINING AND TESTING :-

```
import nltk
```

```
[118] from nltk.stem.porter import PorterStemmer
      ps = PorterStemmer()
```

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

```
new_df['tags'].apply(stem)
```

Show hidden output

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

```
[127] vectors = cv.fit_transform(new_df['tags']).toarray()
```

```
vectors
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
```

```
from sklearn.feature_extraction.text import CountVectorizer

# Example data
corpus = new_df['tags']  # ya tumhara data jahan tum `tags` column use kar rahe ho

# Create the CountVectorizer instance
cv = CountVectorizer(max_features=5000)

# Fit and transform the data to create vocabulary
vectors = cv.fit_transform(corpus)  # This line fits and transforms the data

# Ab tum yahan feature names nikaal sakte ho
feature_names = cv.get_feature_names_out()  # Use get_feature_names_out() in updated versions
```

```
[138] feature_names = cv.get_feature_names_out()
```

```
[139] from sklearn.metrics.pairwise import cosine_similarity
```

```
[142] similarity = cosine_similarity(vectors)
```

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

```
[(1214, 0.4134370962530329),
 (507, 0.4128614119223853),
 (300, 0.4063022541644712),
 (61, 0.40201512610368495),
 (260, 0.39386318072168813)]
```

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

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

```
[(1214, 0.4134370962530329),
 (507, 0.4128614119223853),
 (300, 0.4063022541644712),
 (61, 0.40201512610368495),
 (260, 0.39386318072168813)]
```

```
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])[1:6]

    for i in movies_list:
      print(new_df.iloc[i[0]].title)
```

```
[154] recommend('Batman Begins')
```

```
The Dark Knight
The Dark Knight Rises
Gladiator
The Midnight Meat Train
Gangster Squad
```

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

The final stage in building the movie recommendation system is **training the recommendation model**. Here, we use a content-based approach that leverages cosine similarity to recommend movies based on their attributes, such as genre and plot keywords. In this model, each movie is represented as a vector of features, allowing us to calculate similarities between movies. This similarity score helps identify the closest matches to a given movie in terms of content.

The recommend() function is the core of this recommendation system. When a user inputs a movie title, the function first locates the **index** of that movie in the dataset. It then retrieves a **similarity score** that ranks other movies by their resemblance to the chosen title. The function sorts these scores in descending order and retrieves the top five recommendations, excluding the input movie itself.

In short, this model is efficient and user-friendly, enabling real-time recommendations based on content similarities. By structuring recommendations on genres and keywords, it ensures that users discover movies that closely align with their interests, delivering a personalized and seamless experience.

MILESTONE -2

Performance Metrics Interpretation

1. **Precision: 0.4 (40%)**
   - *Interpretation*: Out of the total recommended movies, 40% are relevant to the user’s interest based on Batman Begins.
2. **Recall: 0.4 (40%)**
   - *Interpretation*: From all the relevant movies in the dataset, the model successfully recommended 40%. This shows that while some relevant movies were captured, there's still room for increasing the recall rate.
3. **F1 Score: 0.4**
   - *Interpretation*: The F1 Score provides a balanced metric between precision and recall, reflecting that 40% of the recommended movies meet the user's preferences.
4. **Mean Reciprocal Rank (MRR): 1.0**
   - *Interpretation*: The first relevant movie appears as the top recommendation, indicating excellent ranking quality. A perfect MRR of 1.0 suggests that the most relevant recommendation was provided at the highest possible position.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARING

MILESTONE -2

```
[63] Gladiator
The Midnight Meat Train
Gangster Squad
```

```python
recommended_movies = ["The Dark Knight", "The Dark Knight Rises", "Gladiator", "The Midnight Meat Train", "Gangster Squad"]
relevant_movies = ["The Dark Knight", "The Dark Knight Rises", "Inception", "Memento", "Interstellar"]
def precision(recommended, relevant):
    relevant_recommendations = set(recommended) & set(relevant)
    return len(relevant_recommendations) / len(recommended)
def recall(recommended, relevant):
    relevant_recommendations = set(recommended) & set(relevant)
    return len(relevant_recommendations) / len(relevant)
def f1_score(precision, recall):
    if (precision + recall) == 0:
        return 0
    return 2 * (precision * recall) / (precision + recall)
def reciprocal_rank(recommended, relevant):
    for i, movie in enumerate(recommended):
        if movie in relevant:
            return 1 / (i + 1)
    return 0
precision_score = precision(recommended_movies, relevant_movies)
recall_score = recall(recommended_movies, relevant_movies)
f1 = f1_score(precision_score, recall_score)
mrr_score = reciprocal_rank(recommended_movies, relevant_movies)
print("Precision:", precision_score)
print("Recall:", recall_score)
print("F1 Score:", f1)
print("Mean Reciprocal Rank (MRR):", mrr_score)
```

```
Precision: 0.4
Recall: 0.4
F1 Score: 0.4000000000000001
Mean Reciprocal Rank (MRR): 1.0
```

THANK YOU