BUSINESS CASE STUDY

ZEE RECOMMENDER SYSTEM

From the company's perspective:

● Zee Recommender Systems represents an ambitious venture by Zee to enhance user experience through personalized movie recommendations.

● The focus is on leveraging user ratings and similarities among users to create a

robust, personalized movie recommender system.

● Utilizing a comprehensive dataset of movie ratings, user demographics, and

movie details, Zee aims to develop a system that can accurately predict user

preferences and suggest movies accordingly.

● The insights gained from this system are expected to drive user engagement,

increase satisfaction, and foster a more intuitive user experience.

Assuming you're a data scientist at Zee, your responsibility involves creating a

personalized movie recommender system. Your primary goals are:

● To analyze user ratings, demographic data, and movie characteristics to

understand viewing preferences.

● To apply collaborative filtering, Pearson Correlation, Cosine Similarity, and Matrix

Factorization techniques to build an effective recommender system.

● To evaluate the system's performance and refine it for accuracy and user

relevance.

Dataset Explanation: Zee Recommender Systems Data

The dataset is first made to proper format in excel and all the columns are delimited.

Post formatting, we have three datasets

https://docs.google.com/spreadsheets/d/1_ldbbUrQZP06Yw7nrV7fkfTt0MqVPmTK/edit?usp=drive_link&ouid=110268629074414432662&rtpof=true&sd=true

https://docs.google.com/spreadsheets/d/1jK0o0_ODkOqKQEdleef-KchBDHfU9MwZ/edit?usp=drive_link&ouid=110268629074414432662&rtpof=true&sd=true

https://docs.google.com/spreadsheets/d/1AMprD3SpvIy7E9NcBWl8tShk4D4_mm0H/edit?usp=drive_link&ouid=110268629074414432662&rtpof=true&sd=true

The dataset for this project is composed of three primary files, each contributing

essential information for building the recommender system:

1. Ratings File (ratings.dat):

● Format: UserID::MovieID::Rating::Timestamp

● Contains user ratings for movies on a 5-star scale.

● Includes a timestamp representing when the rating was given.

● Each user has rated at least 20 movies.

2. Users File (users.dat):

● Format: UserID::Gender::Age::Occupation::Zip-code

● Provides demographic information about the users, including gender, age group, occupation, and zip code.

● Demographic data is voluntarily provided by users and varies in accuracy and completeness.

3. Movies File (movies.dat):

● Format: MovieID::Title::Genres

● Lists movie titles alongside their respective genres.

● Genres are categorized into multiple types like Action, Comedy, Drama, etc., and are pipe-separated.

Key Points to Note:

1. UserID and MovieID serve as unique identifiers for users and movies, respectively.

2. Ratings reflect user preferences and are crucial for understanding individual and collective tastes.

3. User Demographics (gender, age, occupation) can provide insights into user preferences and behavior patterns.

4. Movie Details (title, genres) are essential for categorizing movies and understanding their appeal to different user segments.

The datasets were loaded into Pandas DataFrames

```
df_user = pd.read_excel('zee-users.xlsx')
df_user.head()
```

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	2460
4	5	M	25	20	55455

```
df_rating = pd.read_excel('zee-ratings.xlsx')
df_rating.head()
```

	UserID	MovieID	Rating	Timestamp
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

```
df_movies = pd.read_excel('zee-movies.xlsx')
df_movies.head()
```

	Movie ID	Title	Genres
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

DATA PREPROCESSING

The structure and info of the data sets is checked.

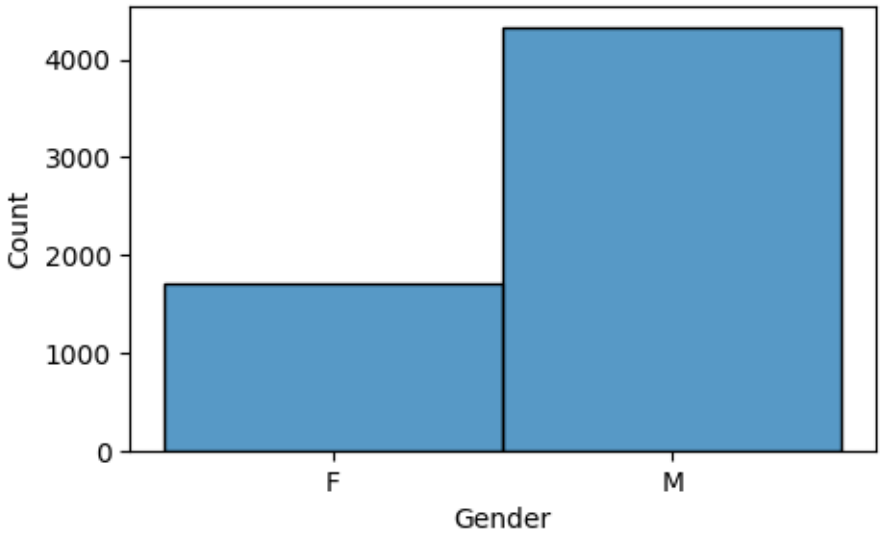
```
df_user.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   UserID      6040 non-null   int64 
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   int64 
 3   Occupation  6040 non-null   int64 
 4   Zip-code    6040 non-null   object
dtypes: int64(3), object(2)
memory usage: 236.1+ KB
```

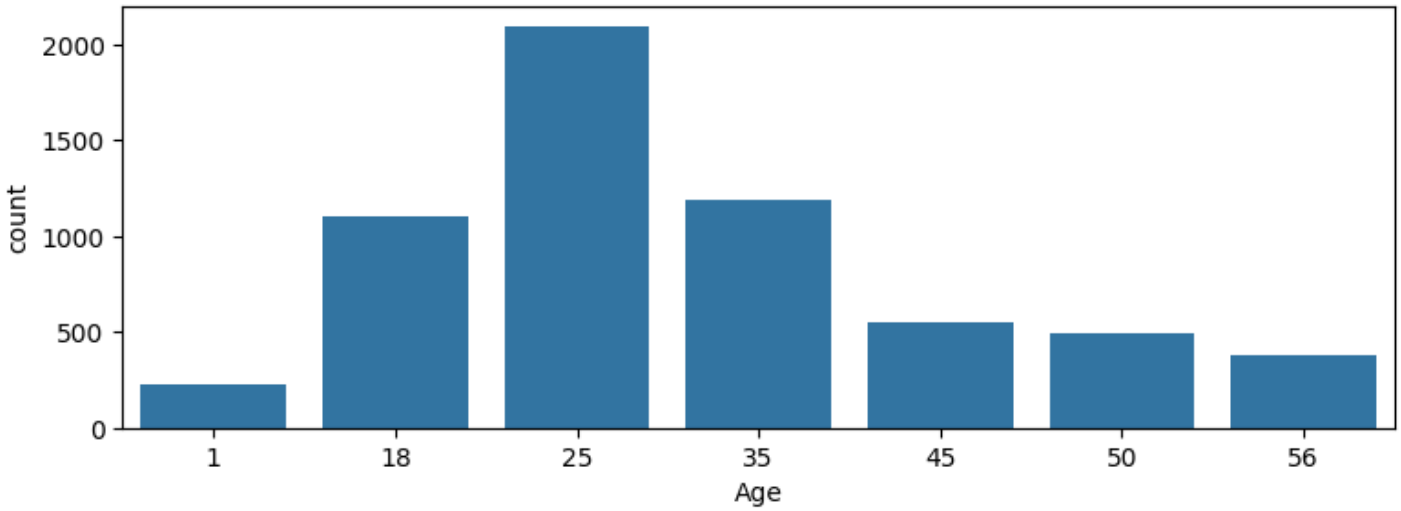
EXPLORATORY DATA ANALYSIS

The missing values and duplicated values are checked in the dataframes and found that there are no missing values neither there are any duplicated values.

UNIVARIATE ANALYSIS



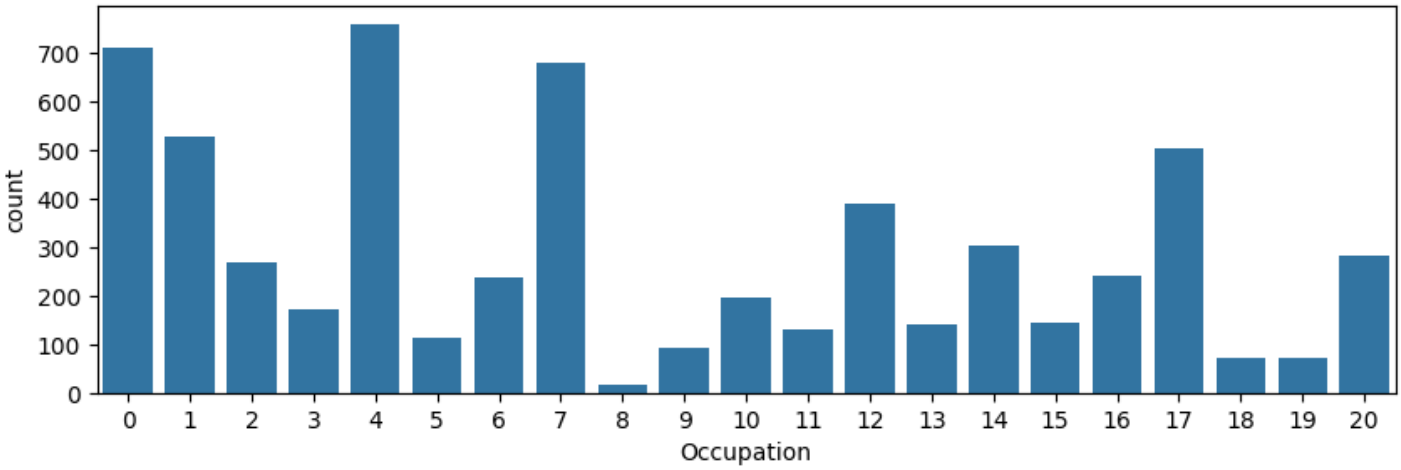
Inference : The histogram is plotted between male female and it is observed that males have rated more movies than females



- 1: "Under 18"
- 18: "18-24"
- 25: "25-34"
- 35: "35-44"
- 45: "45-49"
- 50: "50-55"
- 56: "56+"

Inference: The age is group into above mentioned bins and it can be inferred that age group 25-34 significantly rate more movies.

And spectators under 18 have rated approx. 100 movies.



Inference: hence occupations are given inform of code..

- 0: "other" or not specified
- 1: "academic/educator"
- 2: "artist"
- 3: "clerical/admin"
- 4: "college/grad student"
- 5: "customer service"
- 6: "doctor/health care"
- 7: "executive/managerial"
- 8: "farmer"
- 9: "homemaker"
- 10: "K-12 student"
- 11: "lawyer"
- 12: "programmer"
- 13: "retired"
- 14: "sales/marketing"
- 15: "scientist"
- 16: "self-employed"
- 17: "technician/engineer"
- 18: "tradesman/craftsman"
- 19: "unemployed"
- 20: "writer"

Mostly college going students or corporate workers have rated more movies.

Whereas people with occupation of farmer, craftsmen, lawyers, retired people have rated less than 100 movies.

```
# get only the movies which have count more than 100

x = df_rating['MovieID'].value_counts()[df_rating['MovieID'].value_counts() > 100].index
df_rating = df_rating[df_rating.MovieID.isin(x)]
```

```
# to get only the users who have rated atleast 100 movies
df_rating['UserID'].value_counts()[df_rating['UserID'].value_counts() > 200]
```

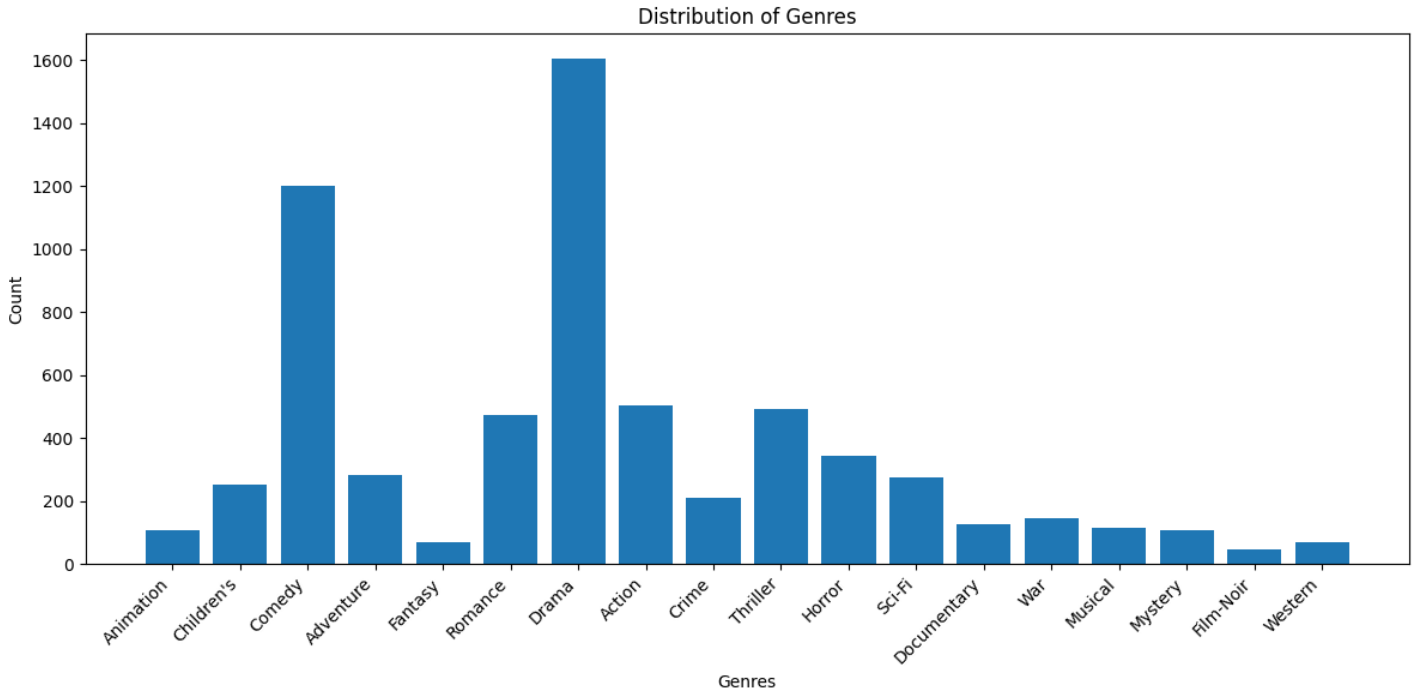
We have taken only the movies which are rated more than 100 times and the users who have rated at least 200 movies. So, as to reduce the variation in data.

```
[14]
df_movies['Genres'] = df_movies['Genres'].str.replace('|', ' ')
```

```
[ ] df_movies.head(20)
```

	Movie ID	Title	Genres
0	1	Toy Story (1995)	Animation Children's Comedy
1	2	Jumanji (1995)	Adventure Children's Fantasy
2	3	Grumpier Old Men (1995)	Comedy Romance
3	4	Waiting to Exhale (1995)	Comedy Drama
4	5	Father of the Bride Part II (1995)	Comedy
5	6	Heat (1995)	Action Crime Thriller
6	7	Sabrina (1995)	Comedy Romance
7	8	Tom and Huck (1995)	Adventure Children's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action Adventure Thriller
10	11	American President, The (1995)	Comedy Drama Romance

The pipe operator between the different genres in removes for better vizualisation.



Inference: Clearly people have rated more comedy and drama movies. However, the least focused genres are fantasy, animation, war etc.

A new dataframe is obtained by merging all the three datasets .

```
# drop one movie id column

df_merged = df_merged.drop('Movie ID', axis=1)
df_merged.head()
```

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	Zip-code	Title	Genres
0	10	2622	5	978228212	F	35	1	95370	Midsummer Night's Dream, A (1999)	Comedy Fantasy
1	18	2622	5	978152574	F	18	3	95825	Midsummer Night's Dream, A (1999)	Comedy Fantasy
2	19	2622	5	978300144	M	1	10	48073	Midsummer Night's Dream, A (1999)	Comedy Fantasy
3	26	2622	3	978275603	M	25	7	23112	Midsummer Night's Dream, A (1999)	Comedy Fantasy
4	45	2622	3	977991794	F	45	16	94110	Midsummer Night's Dream, A (1999)	Comedy Fantasy

BIVARIATE ANALYSIS

To check which age group, prefer which genre

```
# prompt: most count of genre  for each age group

# Find the most frequent genre for each age group
most_frequent_genres = genre_by_age_group_pivot.idxmax(axis=1)

# Display the result
print(most_frequent_genres)
```

```
Age
1     Comedy
18    Comedy
25    Comedy
35    Comedy
45     Drama
50     Drama
56     Drama
dtype: object
```

Inference: The relation ship between age group and genre is obtained and it is observed that people till age 44 prefer comedy movies and 45-56+ prefer drama movies.

FEATURE ENGINEERING

additional column created to show average rating given by user

MODEL BUILDING

HANDLING SPARSE DATA

```
[87] # measure of sparcity
     (df_final > 0).sum().sum() / (df_final.shape[0] * df_final.shape[1])  #the data is too sparse
```

```
0.19899928592600577
```

Inference: It is clearly observed that only 19.8%vis filled nearly 80% of the data is missing.

Hence the data is too sparse. which should be treated by matrix factorization to fill the missing values in the original data

MATRIX FACTORISATION

```python
# create a matrix factorization on df_final

# Assuming df_final is user-item rating matrix
from sklearn.decomposition import NMF

# Initialize NMF model with desired number of latent factors
n_latent_factors = 50
model = NMF(n_components=n_latent_factors, init='random', random_state=0)

# Fit the model to the rating matrix
W = model.fit_transform(df_final)  # User feature matrix
H = model.components_  # Item feature matrix

# Reconstruct the predicted rating matrix
predicted_ratings = np.dot(W, H)
```

The model is created keeping the latent factors = 50 and the performance is checked and found RMSE to be 0.96 which is a good value to consider and we can proceed further analysis by predicted_ratings

```python
# check the predicted_ratings performance with the df_final_mov

from sklearn.metrics import mean_squared_error

# Calculate RMSE
rmse = np.sqrt(mean_squared_error(df_final, predicted_ratings))
print("RMSE:", rmse)
```

```
RMSE: 0.96423151290579
```

Predicted_rating_df is converted to pandas data frame below

```
predicted_ratings_df.head()
```

UserID	2	5	8	9	10	11	13	15	17	18	...	6021	6023	6025	6030
Title															
'burbs, The (1989)	0.015750	0.034841	0.043786	0.014027	1.219927	0.035383	0.001969	0.017686	0.026686	0.205767	...	0.000399	0.019374	0.276087	0.027151
...And Justice for All (1979)	0.223965	0.007786	0.023061	0.021107	0.125214	0.032396	0.000000	0.212592	0.040208	0.001120	...	0.000000	0.052934	0.000912	0.000000
10 Things I Hate About You (1999)	0.037754	0.958652	0.342485	0.368480	0.754388	0.585427	0.000000	1.243775	0.361917	0.701054	...	0.208693	0.138908	1.584014	0.093910
101 Dalmatians (1961)	0.082345	0.043218	0.011486	0.018185	2.230126	0.007119	0.000000	0.028148	0.049431	4.783660	...	0.941504	0.005753	0.084301	0.012123
101 Dalmatians (1996)	0.056415	0.026886	0.049036	0.121055	1.495583	0.120784	0.000000	0.171924	0.129438	0.693799	...	0.137597	0.013936	0.418808	0.000915

COLLABORATIVE FILTERING WITH PEARSON CORRELATION

A movie similarity matrix is created for the similar movies using Pearson correlation

```
movie_similarity = predicted_ratings_df.T.corr(method='pearson')

movie_similarity.head()  # Display the user similarity matrix
```

Title	'burbs, The (1989)	...And Justice for All (1979)	10 Things I Hate About You (1999)	101 Dalmatians (1961)	101 Dalmatians (1996)	12 Angry Men (1957)	13th Warrior, The (1999)	2 Days in the Valley (1996)	20 Dates (1998)	20,000 Leagues Under the Sea (1954)	...	Yellow Submarine (1968)	Yojimbo (1961)	You've Got Mail (1998)	F
Title															
'burbs, The (1989)	1.000000	0.094368	0.355304	0.296814	0.429788	0.065270	0.255106	0.197713	0.150577	0.160545	...	0.206372	0.004585	0.330648	
...And Justice for All (1979)	0.094368	1.000000	0.001105	0.199338	0.139763	0.439226	0.050900	0.348664	0.128174	0.316817	...	0.426842	0.260385	0.094901	
10 Things I Hate About You (1999)	0.355304	0.001105	1.000000	0.174142	0.314275	-0.074473	0.496137	0.292806	0.615914	-0.075547	...	0.010899	-0.109108	0.531931	
101 Dalmatians (1961)	0.296814	0.199338	0.174142	1.000000	0.605194	0.253784	0.192638	0.006343	-0.016732	0.455062	...	0.620990	0.147313	0.216109	

Created a function for Top 5 movie suggestions based on the similarity

```
# Function to create recommendation
def recommend_movies(movie_title, movie_similarity, top_n=5):
  if movie_title not in movie_similarity.index:
    print(f"Movie '{movie_title}' not found in the dataset.")
    return []
  # Get similarity scores for the given movie
  similar_movies = movie_similarity[movie_title]
  # Sort movies by similarity in descending order (excluding the movie itself)
  recommended_movies = similar_movies.sort_values(ascending=False)[1:top_n+1]
  return recommended_movies.index.tolist()
# Example usage:
movie_to_recommend = "Toy Story (1995)"  # Replace with the movie you want recommendations for
recommendations = recommend_movies(movie_to_recommend, movie_similarity)

print(f"Top 5 movies similar to '{movie_to_recommend}':")
for movie in recommendations:
  print(movie)
```

```
Top 5 movies similar to 'Toy Story (1995)':
Aladdin (1992)
Bug's Life, A (1998)
Toy Story 2 (1999)
Lion King, The (1994)
Antz (1998)
```

Creating a user similarity matrix

```python
# Calculate Pearson correlation for all pairs of users
user_similarity = predicted_ratings_df.corr(method='pearson')
# Display the user similarity matrix
print(user_similarity.head())  # Display the user similarity matrix
```

```
UserID      10        15        18        19        22        23        26    \
UserID
10      1.000000  0.274100  0.651109  0.513542  0.567250  0.466885  0.503456
15      0.274100  1.000000  0.299287  0.481927  0.539032  0.379156  0.399192
18      0.651109  0.299287  1.000000  0.646456  0.479532  0.381286  0.274746
19      0.513542  0.481927  0.646456  1.000000  0.535797  0.524531  0.223991
22      0.567250  0.539032  0.479532  0.535797  1.000000  0.596438  0.467539

UserID      33        36        42    ...      6001      6002      6003  \
UserID                                ...
10      0.457929  0.527977  0.580172  ...  0.268071  0.379620  0.434441
15      0.306060  0.657116  0.264906  ...  0.516662  0.136888  0.353588
18      0.371943  0.487070  0.456393  ...  0.263428  0.177062  0.279533
19      0.536767  0.685667  0.620207  ...  0.371002  0.290296  0.262431
22      0.370845  0.687514  0.605127  ...  0.436466  0.189136  0.286881

UserID      6007      6010      6016      6025      6035      6036      6040
UserID
10      0.320869  0.581067  0.545857  0.258821  0.537260  0.450726  0.305576
15      0.370009  0.394154  0.233490  0.330404  0.520524  0.293221  0.133792
18      0.317106  0.424127  0.389815  0.159575  0.327001  0.386617  0.199829
```

Created a function for Top 5 users suggestions based on the similarity

```python
# function to create recommendation for similar users
def recommend_users(user_id, user_similarity, top_n=5):
  user_id = int(user_id)
  if user_id not in user_similarity.index:
    print(f"User '{user_id}' not found in the dataset.")
    return []
  # Get similarity scores for the given user
  similar_users = user_similarity[user_id]
  # Sort users by similarity in descending order (excluding the user itself)
  recommended_users = similar_users.sort_values(ascending=False)[1:top_n+1]
  return recommended_users.index.tolist()
# Example usage:
user_to_recommend = 10  # Replace with the user ID you want recommendations for
recommendations = recommend_users(user_to_recommend, user_similarity)

print(f"Top 5 users similar to user '{user_to_recommend}':")
for user in recommendations:
  print(user)
```

```
Top 5 users similar to user '10':
1120
5222
2454
4958
4704
```

TOP 5 MOVIES SUGGESTIONS FOR A PARTICULAR USER BY PEARSON CORRELATION

```python
# top 5 movies suggestion to particular user
def recommend_movies_to_user(user_id, predicted_ratings_df, top_n=5):

  user_id = int(user_id)
  if user_id not in predicted_ratings_df.columns:
    print(f"User '{user_id}' not found in the dataset.")
    return []
  # Get predicted ratings for the given user
  user_ratings = predicted_ratings_df[user_id]
  # Sort movies by predicted rating in descending order
  recommended_movies = user_ratings.sort_values(ascending=False)[:top_n]
  return recommended_movies.index.tolist()
# Example usage:
user_id_to_recommend = 10  # Replace with the user ID you want recommendations for
recommendations = recommend_movies_to_user(user_id_to_recommend, predicted_ratings_df)
print(f"Top 5 movie recommendations for user '{user_id_to_recommend}':")
for movie in recommendations:
  print(movie)
```

```
Top 5 movie recommendations for user '10':
Wizard of Oz, The (1939)
Star Wars: Episode IV - A New Hope (1977)
E.T. the Extra-Terrestrial (1982)
Back to the Future (1985)
Star Wars: Episode V - The Empire Strikes Back (1980)
```

COLLABORATIVE FILTERING WITH COSINE SIMILARITY

```python
[40] # cosine similarity

    from sklearn.metrics.pairwise import cosine_similarity
    similarity_matrix = cosine_similarity(predicted_ratings_df)
    similarity_matrix.shape
```

```
(2006, 2006)
```

```python
def recommend(movie_name):
  if movie_name in predicted_ratings_df.index:
    index = np.where(predicted_ratings_df.index == movie_name)[0][0]
    distances = similarity_matrix[index]
    enumerated_distances = list(enumerate(distances))
    sorted_distances = sorted(enumerated_distances, key=lambda x:x[1], reverse=True)[1:6]
    for i in sorted_distances:
      print(predicted_ratings_df.index[i[0]]) # Print the movie title instead of the index
  else:
    print(f"Movie '{movie_name}' not found in the dataset.")
```

```
[40] # cosine similarity

     from sklearn.metrics.pairwise import cosine_similarity
     similarity_matrix = cosine_similarity(predicted_ratings_df)
     similarity_matrix.shape
```

```
(2006, 2006)
```

A similarity matrix is obtained and a function is created to recommend top 5 movies based on the cosine similarity between user and rating.

The final recommender using cosine similarity

The model is evaluated using RMSE post matrix factorization and hyper parameter- latent features are tuned to get the least RMSE.

A function is created to predicted top 5 movies based on collaborative filtering using cosine similarity between user and movies rating

```
def recommend(movie_name):
  if movie_name in predicted_ratings_df.index:
    index = np.where(predicted_ratings_df.index == movie_name)[0][0]
    distances = similarity_matrix[index]
    enumerated_distances = list(enumerate(distances))
    sorted_distances = sorted(enumerated_distances, key=lambda x:x[1], reverse=True)[1:6]
    for i in sorted_distances:
      print(predicted_ratings_df.index[i[0]]) # Print the movie title instead of the index
  else:
    print(f"Movie '{movie_name}' not found in the dataset.")
```

```
def recommend(movie_name):
  if movie_name in predicted_ratings_df.index:
    index = np.where(predicted_ratings_df.index == movie_name)[0][0]
    distances = similarity_matrix[index]
    enumerated_distances = list(enumerate(distances))
    sorted_distances = sorted(enumerated_distances, key=lambda x:x[1], reverse=True)[1:6]
    for i in sorted_distances:
      print(predicted_ratings_df.index[i[0]]) # Print the movie title instead of the index
  else:
    print(f"Movie '{movie_name}' not found in the dataset.")
```

```
recommend('101 Dalmatians (1961)')
```

```
Bambi (1942)
Pinocchio (1940)
Winnie the Pooh and the Blustery Day (1968)
Aristocats, The (1970)
Rescuers, The (1977)
```

BUSINESS INSIGHTS

A movie recommendation system can provide valuable business insights that can be leveraged to improve customer experience, increase engagement, and drive revenue. Here are some key insights that can be derived:

1. User Preferences and Behaviour Patterns

- **Personalization**: Understanding individual user preferences allows businesses to tailor content recommendations, improving user satisfaction and retention.
- **Viewing Habits**: Analysis of the types of movies that users prefer (e.g., genres, directors, actors) helps in predicting future trends and adjusting the content library accordingly.

2. Content Popularity and Trends

- **Popular Movies**: Identifying which movies are frequently recommended or watched can help in understanding current trends and popular content. This can guide decisions on which movies to promote or acquire.
- **Emerging Trends**: By tracking changes in movie preferences over time, businesses can identify emerging trends and shifts in viewer interests, allowing them to stay ahead of the curve.

3. Customer Segmentation

- **Segmenting Users**: The system can help segment users based on their preferences and behaviours, such as casual viewers, genre enthusiasts, or niche audiences. This segmentation can be used for targeted marketing campaigns.
- **Personalized Marketing**: Segmentation allows businesses to create personalized marketing messages and offers, such as recommending new releases to frequent viewers of a particular genre.

4. Content Gaps and Opportunities

- **Content Acquisition**: Identifying content that is frequently searched for but not available can inform decisions on acquiring new movies or producing similar content.
- **Investment Opportunities**: Insights into underrepresented genres or themes that have a growing audience can guide investment in new content production.

5. Churn Prediction and User Retention

- **Identifying At-Risk Users**: Users who suddenly stop watching or whose engagement drops can be flagged for potential churn. Businesses can then take proactive measures, such as offering discounts or personalized recommendations, to retain these users.
- **User Engagement**: High engagement with the recommendation system can indicate strong user satisfaction, while low engagement may suggest a need for improving the recommendation algorithm or content diversity.

6. Revenue Optimization

- **Monetization Strategies**: Understanding what types of content drive the most engagement can help in creating effective monetization strategies, such as premium content offerings, pay-per-view, or targeted advertising.

- **Upselling and Cross-Selling**: Recommendations can be used to upsell related products or services, such as movie merchandise, or cross-sell other entertainment services like music or books.

7. Content Lifecycle Management

- **Content Longevity**: Insights into how long a movie remains popular can inform decisions on when to refresh the content library or promote older movies that might still have appeal.
- **Seasonal Trends**: Understanding how user preferences change with seasons, holidays, or special events can help in scheduling content releases and promotions.

8. User Feedback Loop

- **Improving Content Quality**: Analysing user feedback on recommended content can provide insights into what makes certain recommendations successful or unsuccessful, allowing for continuous improvement of the recommendation system.
- **Predictive Analytics**: Over time, the data collected from the recommendation system can be used for predictive analytics, helping to forecast future viewing trends and content demand.

9. Competitive Analysis

- **Benchmarking**: By comparing the performance of your recommendation system with competitors, you can identify strengths and weaknesses, helping to improve your service offerings.
- **Content Differentiation**: Insights from user behaviour and preferences can help differentiate your content offerings from competitors, providing a unique value proposition.

10. Operational Efficiency

- **Inventory Management**: Understanding which movies are frequently recommended but less often watched can help in optimizing inventory management and reducing licensing costs for underperforming content.
- **Resource Allocation**: Insights into content demand can guide the allocation of resources for content curation, marketing, and user experience improvements.

Link to Colab Notebook

https://colab.research.google.com/drive/1iNFkbpjgXVAMI0S4dP2Mjlr4jTkZ8e0z?usp=sharing

Link to Presentation

https://www.canva.com/design/DAGN1dq70u8/weR1jWupZ6WlGZihu2UgXw/edit?utm_content=DAGN1dq70u8&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton