MOVIE RECCOMENDER SYSTEM

BUSINESS UNDERSTANDING

Problem Statement

HTG,a movie streaming company asked there users to rate them on google play store. The feedback received was that the movies recommended to the users didn't match their interests thus most customers were dissatisfied.

They have approached us, a data analytics company to help them solve their problem. We will therefore, build a movie recommender system that will aid in suggesting top 5 movies to the streaming site users based on their ratings and the genres they prefer.

OBJECTIVES

main objective

To build a movie recommender system,to suggest top movies to the streaming users based on the movie ratings and genres prefered

specific objectives

To find out the average rating of movies To determine the number of movies per genre To determine the most popular movies

Metric for sucess

We will use RMSE as our metric for success,the model having the lowest RMSE score being our best model.

Data understanding

The data used has been sourced from MovieLens dataset from the GroupLens research lab at the University of Minnesota.

It contains 100836 ratings and 3683 tag applications across 9742 movies. These data were created by 610 users.

The dataset is distributed among four csv files:

links.csv

movies.csv

ratings.csv

tags.csv

```
#importing necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
import random
import pickle

# installing surprise and importing some of its needed modules
! pip install surprise
from surprise import Dataset, Reader, SVD
from surprise.prediction_algorithms import KNNBasic, KNNWithMeans, KNNWithZScore, KNNBaseline, knns, SVDpp
from surprise.model_selection import cross_validate
from surprise.model_selection import train_test_split
from surprise.model_selection import GridSearchCV
```

```
Requirement already satisfied: surprise in /usr/local/lib/python3.11/dist-packages (0.1)
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.11/dist-packages (from surprise) (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise->surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise->surprise) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise->surprise) (1.13.1)
```

```
#checking the links csv
links_df=pd.read_csv('links.csv')
links_df.head()
```

| | movieId | imdbId | tmdbId |
|---|---|---|---|
| 0 | 1 | 114709 | 862.0 |
| 1 | 2 | 113497 | 8844.0 |
| 2 | 3 | 113228 | 15602.0 |
| 3 | 4 | 114885 | 31357.0 |
| 4 | 5 | 113041 | 11862.0 |

Next steps: Generate code with links_df View recommended plots New interactive sheet

```
#checking the movies csv
movies_df=pd.read_csv('movies.csv')
movies_df.head()
```

| | movieId | title | genres |
|---|---|---|---|
| 0 | 1 | Toy Story (1995) | Adventure\|Animation\|Children\|Comedy\|Fantasy |
| 1 | 2 | Jumanji (1995) | Adventure\|Children\|Fantasy |
| 2 | 3 | Grumpier Old Men (1995) | Comedy\|Romance |
| 3 | 4 | Waiting to Exhale (1995) | Comedy\|Drama\|Romance |
| 4 | 5 | Father of the Bride Part II (1995) | Comedy |

Next steps: Generate code with movies_df View recommended plots New interactive sheet

```
#checking the ratings csv
ratings_df=pd.read_csv('ratings.csv')
ratings_df.head()
```

| | userId | movieId | rating | timestamp |
|---|---|---|---|---|
| 0 | 1 | 1 | 4.0 | 964982703 |
| 1 | 1 | 3 | 4.0 | 964981247 |
| 2 | 1 | 6 | 4.0 | 964982224 |
| 3 | 1 | 47 | 5.0 | 964983815 |
| 4 | 1 | 50 | 5.0 | 964982931 |

```
#checking the tags csv
tags_df=pd.read_csv('tags.csv')
tags_df.head()
```

| | userId | movieId | tag | timestamp |
|---|---|---|---|---|
| 0 | 2 | 60756 | funny | 1445714994 |
| 1 | 2 | 60756 | Highly quotable | 1445714996 |
| 2 | 2 | 60756 | will ferrell | 1445714992 |
| 3 | 2 | 89774 | Boxing story | 1445715207 |
| 4 | 2 | 89774 | MMA | 1445715200 |

Next steps: Generate code with tags_df View recommended plots New interactive sheet

```
#checking for null values in links csv
print(links_df.isna().sum())
print(links_df.info())
```

```
movieId    0
imdbId     0
tmdbId     8
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   imdbId   9742 non-null   int64
 2   tmdbId   9734 non-null   float64
dtypes: float64(1), int64(2)
```

```
memory usage: 228.5 KB
None
```

```
#checking for null values in movies csv
print(movies_df.isna().sum())
print(movies_df.info())
```

```
movieId    0
title      0
genres     0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9742 entries, 0 to 9741
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   movieId  9742 non-null   int64
 1   title    9742 non-null   object
 2   genres   9742 non-null   object
dtypes: int64(1), object(2)
memory usage: 228.5+ KB
None
```

```
#checking for null values in ratngs csv
print(ratings_df.isna().sum())
print(ratings_df.info())
```

```
userId       0
movieId      0
rating       0
timestamp    0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100836 entries, 0 to 100835
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   userId     100836 non-null  int64
 1   movieId    100836 non-null  int64
 2   rating     100836 non-null  float64
 3   timestamp  100836 non-null  int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
None
```

```
#checking for null values in tags csv
print(tags_df.isna().sum())
print(tags_df.info())
```

```
userId       0
movieId      0
tag          0
timestamp    0
dtype: int64
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3683 entries, 0 to 3682
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   userId     3683 non-null   int64
 1   movieId    3683 non-null   int64
 2   tag        3683 non-null   object
 3   timestamp  3683 non-null   int64
dtypes: int64(3), object(1)
memory usage: 115.2+ KB
None
```

```
# printing the number of records in every dataframe
dataframes = [links_df, movies_df, ratings_df, tags_df]
dataframe_names = ['links_df', 'movies_df', 'ratings_df', 'tags_df']

for i in range(len(dataframes)):
    print(f" {dataframe_names[i]} has {dataframes[i].shape[0]} records.")
```

```
 links_df has 9742 records.
 movies_df has 9742 records.
 ratings_df has 100836 records.
 tags_df has 3683 records.
```

## Data Cleaning

```
#Making the letters in genres column lowercase and storing them in a list.
movies_df_cleaned = movies_df.copy()
```

```
movies_df_cleaned = movies_df.copy()
movies_df_cleaned.genres = movies_df.genres.map(lambda x: x.replace('|', ',').lower().split(','))
movies_df_cleaned
```

| | movieId | title | genres |
|---|---|---|---|
| 0 | 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] |
| 1 | 2 | Jumanji (1995) | [adventure, children, fantasy] |
| 2 | 3 | Grumpier Old Men (1995) | [comedy, romance] |
| 3 | 4 | Waiting to Exhale (1995) | [comedy, drama, romance] |
| 4 | 5 | Father of the Bride Part II (1995) | [comedy] |
| ... | ... | ... | ... |
| 9737 | 193581 | Black Butler: Book of the Atlantic (2017) | [action, animation, comedy, fantasy] |
| 9738 | 193583 | No Game No Life: Zero (2017) | [animation, comedy, fantasy] |
| 9739 | 193585 | Flint (2017) | [drama] |
| 9740 | 193587 | Bungo Stray Dogs: Dead Apple (2018) | [action, animation] |
| 9741 | 193609 | Andrew Dice Clay: Dice Rules (1991) | [comedy] |

9742 rows × 3 columns

Next steps: Generate code with movies_df_cleaned | View recommended plots | New interactive sheet

```
#Dropping timestamp column
ratings_df_cleaned = ratings_df.drop('timestamp', axis=1)
tags_df_cleaned = tags_df.drop('timestamp', axis=1)

display(ratings_df_cleaned.head(2))
tags_df_cleaned.head(2)
```

| | userId | movieId | rating |
|---|---|---|---|
| 0 | 1 | 1 | 4.0 |
| 1 | 1 | 3 | 4.0 |

| | userId | movieId | tag |
|---|---|---|---|
| 0 | 2 | 60756 | funny |
| 1 | 2 | 60756 | Highly quotable |

Next steps: Generate code with tags_df_cleaned | View recommended plots | New interactive sheet | Generate code with tags_df_cleaned | View

**EDA- Explanatory Data Analysis

```
#movies made per year

# extracting years from the title column
movies_df_cleaned['year'] = movies_df_cleaned['title'].str.extract('.*\((.*)\).*',expand = False)
movies_df_cleaned.year.unique()
```

```
array(['1995', '1994', '1996', '1976', '1992', '1967', '1993', '1964',
       '1977', '1965', '1982', '1990', '1991', '1989', '1937', '1940',
       '1969', '1981', '1973', '1970', '1955', '1959', '1968', '1988',
       '1997', '1972', '1943', '1952', '1951', '1957', '1961', '1958',
       '1954', '1934', '1944', '1960', '1963', '1942', '1941', '1953',
       '1939', '1950', '1946', '1945', '1938', '1947', '1935', '1936',
       '1956', '1949', '1932', '1975', '1974', '1971', '1979', '1987',
       '1986', '1980', '1978', '1985', '1966', '1962', '1983', '1984',
       '1948', '1933', '1931', '1922', '1998', '1929', '1930', '1927',
       '1928', '1999', '2000', '1926', '1919', '1921', '1925', '1923',
       '2001', '2002', '2003', '1920', '1915', '1924', '2004', '1916',
       '1917', '2005', '2006', '1902', nan, '1903', '2007', '2008',
       '2009', '2010', '2011', '2012', '2013', '2014', '2015', '2016',
       '2017', '2018', '1908', '2006-2007'], dtype=object)
```

```
# checking record(s) that have 2006--2007 in the year column
movies_df_cleaned[movies_df_cleaned['year'] == "2006-2007"]
```

| | movieId | title | genres | year |
|---|---|---|---|---|
| 9518 | 171749 | Death Note: Desu nôto (2006–2007) | [(no genres listed)] | 2006-2007 |

```
# changing it to 2007
movies_df_cleaned.year = movies_df_cleaned.year.replace('2006-2007', '2007')
movies_df_cleaned[movies_df_cleaned['year'] == "2006-2007"]
```

| movieId | title | genres | year |
|---|---|---|---|

```
# checking record(s) that contain null values in the year column
display(movies_df_cleaned[pd.isna(movies_df_cleaned.year)])
len(movies_df_cleaned[pd.isna(movies_df_cleaned.year)])
```

| | movieId | title | genres | year |
|---|---|---|---|---|
| **6059** | 40697 | Babylon 5 | [sci-fi] | NaN |
| **9031** | 140956 | Ready Player One | [action, sci-fi, thriller] | NaN |
| **9091** | 143410 | Hyena Road | [(no genres listed)] | NaN |
| **9138** | 147250 | The Adventures of Sherlock Holmes and Doctor W... | [(no genres listed)] | NaN |
| **9179** | 149334 | Nocturnal Animals | [drama, thriller] | NaN |
| **9259** | 156605 | Paterson | [(no genres listed)] | NaN |
| **9367** | 162414 | Moonlight | [drama] | NaN |
| **9448** | 167570 | The OA | [(no genres listed)] | NaN |
| **9514** | 171495 | Cosmos | [(no genres listed)] | NaN |
| **9515** | 171631 | Maria Bamford: Old Baby | [(no genres listed)] | NaN |
| **9525** | 171891 | Generation Iron 2 | [(no genres listed)] | NaN |
| **9611** | 176601 | Black Mirror | [(no genres listed)] | NaN |

```
#Getting the first year

print(movies_df_cleaned['year'].dropna().astype(int).min())

#Getting the latest year
print(movies_df_cleaned['year'].dropna().astype(int).max())
```

```
1902
2018
```

```
movies_df_cleaned['year']=movies_df_cleaned['year'].dropna().astype(int)
```

```
movies_df_cleaned
```

| | movieId | title | genres | year |
|---|---|---|---|---|
| **0** | 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 1995.0 |
| **1** | 2 | Jumanji (1995) | [adventure, children, fantasy] | 1995.0 |
| **2** | 3 | Grumpier Old Men (1995) | [comedy, romance] | 1995.0 |
| **3** | 4 | Waiting to Exhale (1995) | [comedy, drama, romance] | 1995.0 |
| **4** | 5 | Father of the Bride Part II (1995) | [comedy] | 1995.0 |
| **...** | ... | ... | ... | ... |
| **9737** | 193581 | Black Butler: Book of the Atlantic (2017) | [action, animation, comedy, fantasy] | 2017.0 |
| **9738** | 193583 | No Game No Life: Zero (2017) | [animation, comedy, fantasy] | 2017.0 |
| **9739** | 193585 | Flint (2017) | [drama] | 2017.0 |
| **9740** | 193587 | Bungo Stray Dogs: Dead Apple (2018) | [action, animation] | 2018.0 |
| **9741** | 193609 | Andrew Dice Clay: Dice Rules (1991) | [comedy] | 1991.0 |

9742 rows × 4 columns

Next steps: Generate code with movies_df_cleaned | View recommended plots | New interactive sheet

```
#Binning  the years in groups of 10
movies_df_cleaned['year_bins'] = pd.cut(x=movies_df_cleaned['year'], bins=list(range(1900,2018,10)))
```

```
movies_df_cleaned
```

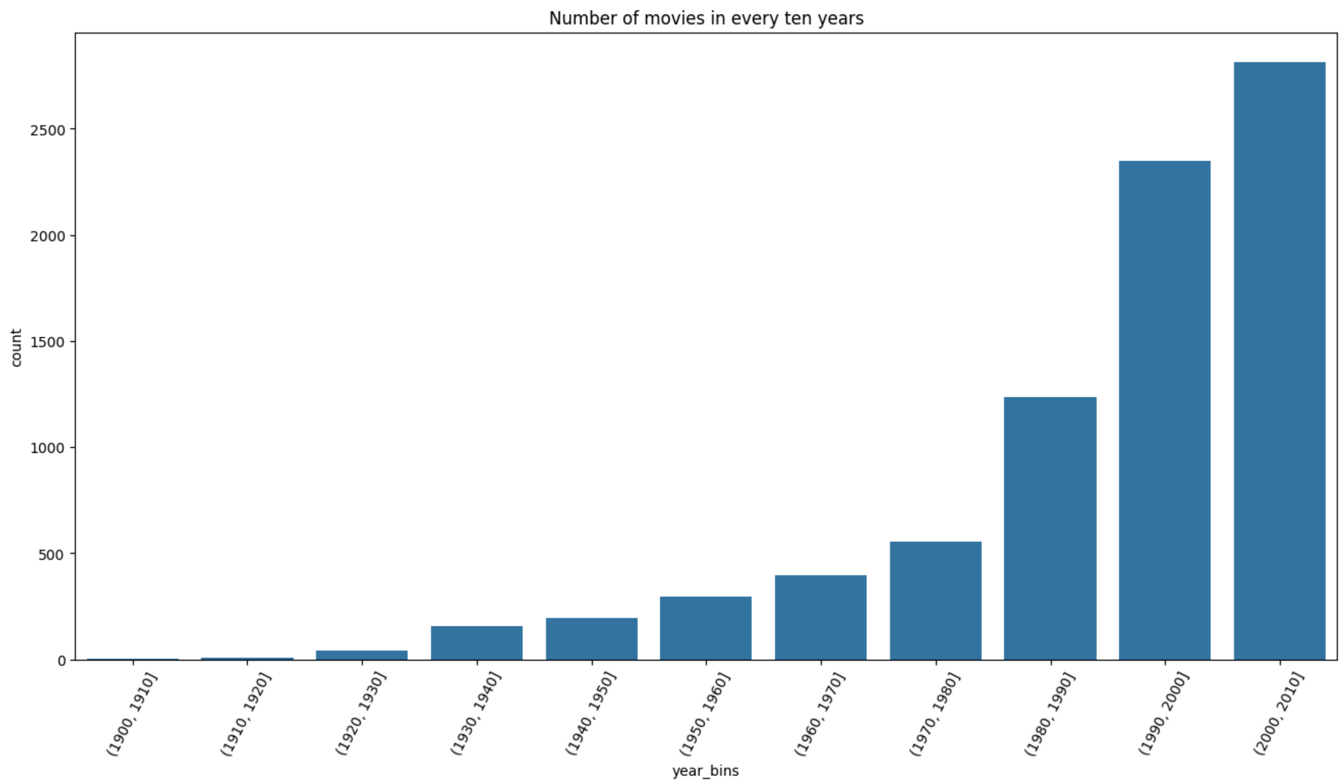
| | movieId | title | genres | year | year_bins |
|---|---|---|---|---|---|
| **0** | 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 1995.0 | (1990.0, 2000.0] |
| **1** | 2 | Jumanji (1995) | [adventure, children, fantasy] | 1995.0 | (1990.0, 2000.0] |
| **2** | 3 | Grumpier Old Men (1995) | [comedy, romance] | 1995.0 | (1990.0, 2000.0] |
| **3** | 4 | Waiting to Exhale (1995) | [comedy, drama, romance] | 1995.0 | (1990.0, 2000.0] |
| **4** | 5 | Father of the Bride Part II (1995) | [comedy] | 1995.0 | (1990.0, 2000.0] |
| **...** | ... | ... | ... | ... | ... |
| **9737** | 193581 | Black Butler: Book of the Atlantic (2017) | [action, animation, comedy, fantasy] | 2017.0 | NaN |
| **9738** | 193583 | No Game No Life: Zero (2017) | [animation, comedy, fantasy] | 2017.0 | NaN |
| **9739** | 193585 | Flint (2017) | [drama] | 2017.0 | NaN |
| **9740** | 193587 | Bungo Stray Dogs: Dead Apple (2018) | [action, animation] | 2018.0 | NaN |
| **9741** | 193609 | Andrew Dice Clay: Dice Rules (1991) | [comedy] | 1991.0 | (1990.0, 2000.0] |

9742 rows × 5 columns

Next steps: Generate code with movies_df_cleaned | View recommended plots | New interactive sheet

This dataset contains movie released between year 1902 and 2018.

```
#Plotting the number of movies released every 10 years between 1902 and 2018.
plt.figure(figsize=(16,8))
sns.countplot(x=movies_df_cleaned.year_bins.dropna())
plt.xticks(rotation=63)
plt.title('Number of movies in every ten years')
plt.show()
```




Movies rating counts

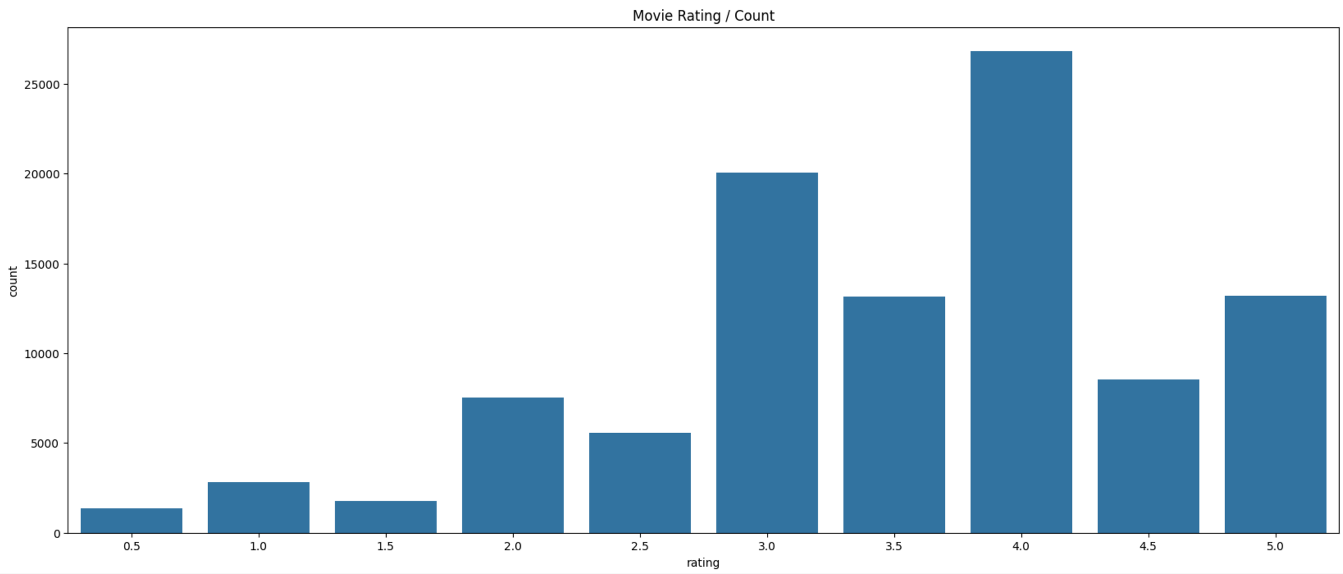
```
#Calculating the mean of ratings.
ratings_df_cleaned['rating'].mean()
```

```
3.501556983616962
```

```
# investigating the range of ratings in the ratings df
ratings_df_cleaned['rating'].value_counts().to_frame()
```

| rating | count |
|---|---|
| 4.0 | 26818 |
| 3.0 | 20047 |
| 5.0 | 13211 |
| 3.5 | 13136 |
| 4.5 | 8551 |
| 2.0 | 7551 |
| 2.5 | 5550 |
| 1.0 | 2811 |
| 1.5 | 1791 |
| 0.5 | 1370 |

```
# Plotting a count plot of the ratings
plt.figure(figsize=(20,8))
sns.countplot(x=ratings_df_cleaned['rating'])
plt.title('Movie Rating / Count');
plt.show()
```




Genre average ratings

Getting all the genres present in the dataset

```
# creating a list to store the movie genres
genres = []

for record in range(len(movies_df_cleaned)):
    genres_list = movies_df_cleaned.loc[record, 'genres']

    for genre in genres_list:
        if genre not in genres:
            genres.append(genre)


genres_list = genres


#  creating a df to store the average ratings of every genre
genre_ratings_records = pd.DataFrame(index=genres, columns=['ratings'], data=np.zeros(len(genres)))
genre_ratings_records['no_records'] = np.zeros(len(genres))

for genre in genres:
    # iterating through every record to
    ratings = []
    no_records = 0
    for record in range(movies_df_cleaned.shape[0]):
```

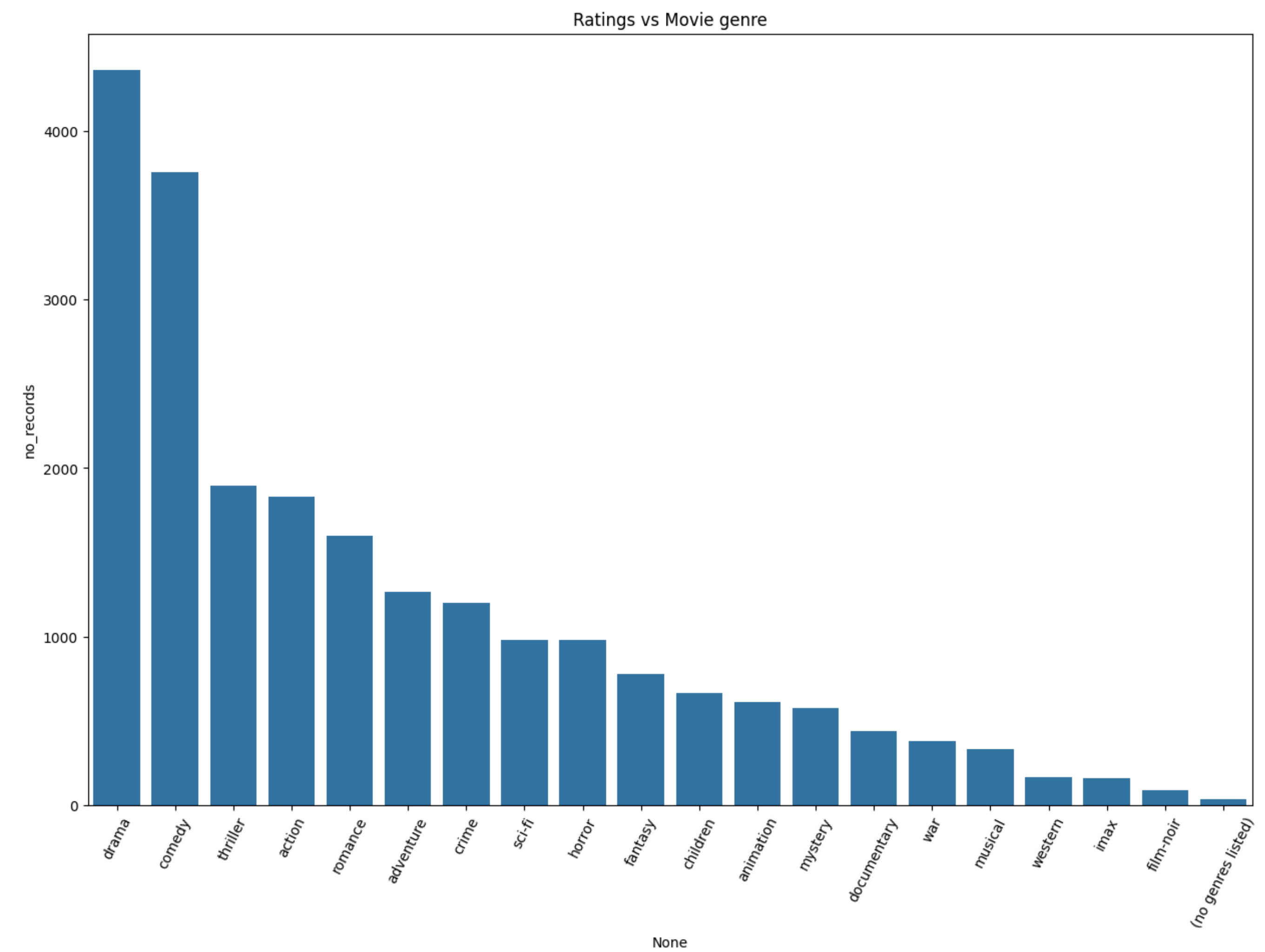
```
            genres = movies_df_cleaned.loc[record, 'genres']

            if genre in genres:
                movie_id = movies_df_cleaned.loc[record, 'movieId']

                # fetching the ratings from the ratings df
                ratings.append(ratings_df[ratings_df.movieId == movie_id]['rating'].mean())
                no_records+=1


        genre_ratings_records.loc[genre, 'ratings'] = np.mean(ratings)
        genre_ratings_records.loc[genre, 'no_records'] = no_records

genre_ratings_sorted = genre_ratings_records
genre_ratings_sorted.head()
```

| | ratings | no_records |
|---|---|---|
| **adventure** | NaN | 1263.0 |
| **animation** | NaN | 611.0 |
| **children** | 3.10769 | 664.0 |
| **comedy** | NaN | 3756.0 |
| **fantasy** | NaN | 779.0 |

Next steps: Generate code with genre_ratings_sorted | View recommended plots | New interactive sheet

Total movies per genre

```
# printing the popularity of the genres in ascending order.
plt.figure(figsize=(15,10))
sns.barplot(x=genre_ratings_sorted.sort_values('no_records', ascending=False).index, y=genre_ratings_sorted.sort_values('no_records', a
plt.xticks(rotation=63)
plt.title('Ratings vs Movie genre')
plt.show()
```



The most rated genre is drama. It is seen to be the most popular as mystery is the least popular with a lower rating count.

```
# returning the movies_df_cleaned
movies_df_cleaned.drop(['year','year_bins'], axis=1, inplace=True)
```

```
#Viewing the shape of  the movies_df dataset.
movies_df_cleaned.shape
```

```
(9742, 3)
```

```
#Checcking for the number of null values in movies_df
movies_df_cleaned.isna().sum()
```

| | 0 |
|---|---|
| movieId | 0 |
| title | 0 |
| genres | 0 |

```
# movies with no ratings receive 0 as their average rating
mean_rating = ratings_df_cleaned.groupby('movieId').rating.mean().rename('mean rating')
num_rating = ratings_df_cleaned.groupby('movieId').userId.count().rename('num rating')
```

```
#Viewing the first 5 rows of cleaned ratings dataset.
ratings_df_cleaned.head()
```

| | userId | movieId | rating |
|---|---|---|---|
| 0 | 1 | 1 | 4.0 |
| 1 | 1 | 3 | 4.0 |
| 2 | 1 | 6 | 4.0 |
| 3 | 1 | 47 | 5.0 |
| 4 | 1 | 50 | 5.0 |

```
#Converting the mean rating to dataframe
test_mean= pd.DataFrame(mean_rating, index= movies_df_cleaned.movieId)
```

```
#Converting the num rating to a dataframe
test_num = pd.DataFrame(num_rating, index= movies_df_cleaned.movieId)
```

```
#Merging mean rating and num rating
test_final = pd.merge(test_mean, test_num, on= "movieId")
test_final.head()
```

| movieId | mean rating | num rating |
|---|---|---|
| 1 | 3.920930 | 215.0 |
| 2 | 3.431818 | 110.0 |
| 3 | 3.259615 | 52.0 |
| 4 | 2.357143 | 7.0 |
| 5 | 3.071429 | 49.0 |

Next steps: Generate code with test_final | View recommended plots | New interactive sheet

```
#Merging the cleaned df with the ratings
df_final= pd.merge(movies_df_cleaned, test_final, on= "movieId")
df_final.head()
```

| | movieId | title | genres | mean rating | num rating |
|---|---|---|---|---|---|
| 0 | 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 3.920930 | 215.0 |
| 1 | 2 | Jumanji (1995) | [adventure, children, fantasy] | 3.431818 | 110.0 |
| 2 | 3 | Grumpier Old Men (1995) | [comedy, romance] | 3.259615 | 52.0 |
| 3 | 4 | Waiting to Exhale (1995) | [comedy, drama, romance] | 2.357143 | 7.0 |
| 4 | 5 | Father of the Bride Part II (1995) | [comedy] | 3.071429 | 49.0 |

Next steps: Generate code with df_final View recommended plots New interactive sheet

```
#Checking for null values
df_final.isna().sum()
```

| | 0 |
|---|---|
| movieId | 0 |
| title | 0 |
| genres | 0 |
| mean rating | 18 |
| num rating | 18 |

**dtype:** int64

```
#Checking for the null values in out dataset
df_final[df_final["num rating"].isna()]
```

| | movieId | title | genres | mean rating | num rating |
|---|---|---|---|---|---|
| 816 | 1076 | Innocents, The (1961) | [drama, horror, thriller] | NaN | NaN |
| 2211 | 2939 | Niagara (1953) | [drama, thriller] | NaN | NaN |
| 2499 | 3338 | For All Mankind (1989) | [documentary] | NaN | NaN |
| 2587 | 3456 | Color of Paradise, The (Rang-e khoda) (1999) | [drama] | NaN | NaN |
| 3118 | 4194 | I Know Where I'm Going! (1945) | [drama, romance, war] | NaN | NaN |
| 4037 | 5721 | Chosen, The (1981) | [drama] | NaN | NaN |
| 4506 | 6668 | Road Home, The (Wo de fu qin mu qin) (1999) | [drama, romance] | NaN | NaN |
| 4598 | 6849 | Scrooge (1970) | [drama, fantasy, musical] | NaN | NaN |
| 4704 | 7020 | Proof (1991) | [comedy, drama, romance] | NaN | NaN |
| 5020 | 7792 | Parallax View, The (1974) | [thriller] | NaN | NaN |
| 5293 | 8765 | This Gun for Hire (1942) | [crime, film-noir, thriller] | NaN | NaN |
| 5421 | 25855 | Roaring Twenties, The (1939) | [crime, drama, thriller] | NaN | NaN |
| 5452 | 26085 | Mutiny on the Bounty (1962) | [adventure, drama, romance] | NaN | NaN |
| 5749 | 30892 | In the Realms of the Unreal (2004) | [animation, documentary] | NaN | NaN |
| 5824 | 32160 | Twentieth Century (1934) | [comedy] | NaN | NaN |
| 5837 | 32371 | Call Northside 777 (1948) | [crime, drama, film-noir] | NaN | NaN |
| 5957 | 34482 | Browning Version, The (1951) | [drama] | NaN | NaN |
| 7565 | 85565 | Chalet Girl (2011) | [comedy, romance] | NaN | NaN |

There are null values in our final dataset. Therefore, we will go ahead to drop them.

```
# dropping the null values
df_final.dropna(inplace= True)
```

```
# setting movie_id to be the index

df_final.set_index('movieId', inplace=True)
```

```
#Checking for the shape of our final dataset.
df_final.shape
```

```
(9724, 4)
```

One hot encoding the genres column

```
#Checking the first 5 rows of our final dataset.
df_final.head()
```

| movieId | title | genres | mean rating | num rating |
|---|---|---|---|---|
| 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 3.920930 | 215.0 |
| 2 | Jumanji (1995) | [adventure, children, fantasy] | 3.431818 | 110.0 |
| 3 | Grumpier Old Men (1995) | [comedy, romance] | 3.259615 | 52.0 |
| 4 | Waiting to Exhale (1995) | [comedy, drama, romance] | 2.357143 | 7.0 |
| 5 | Father of the Bride Part II (1995) | [comedy] | 3.071429 | 49.0 |

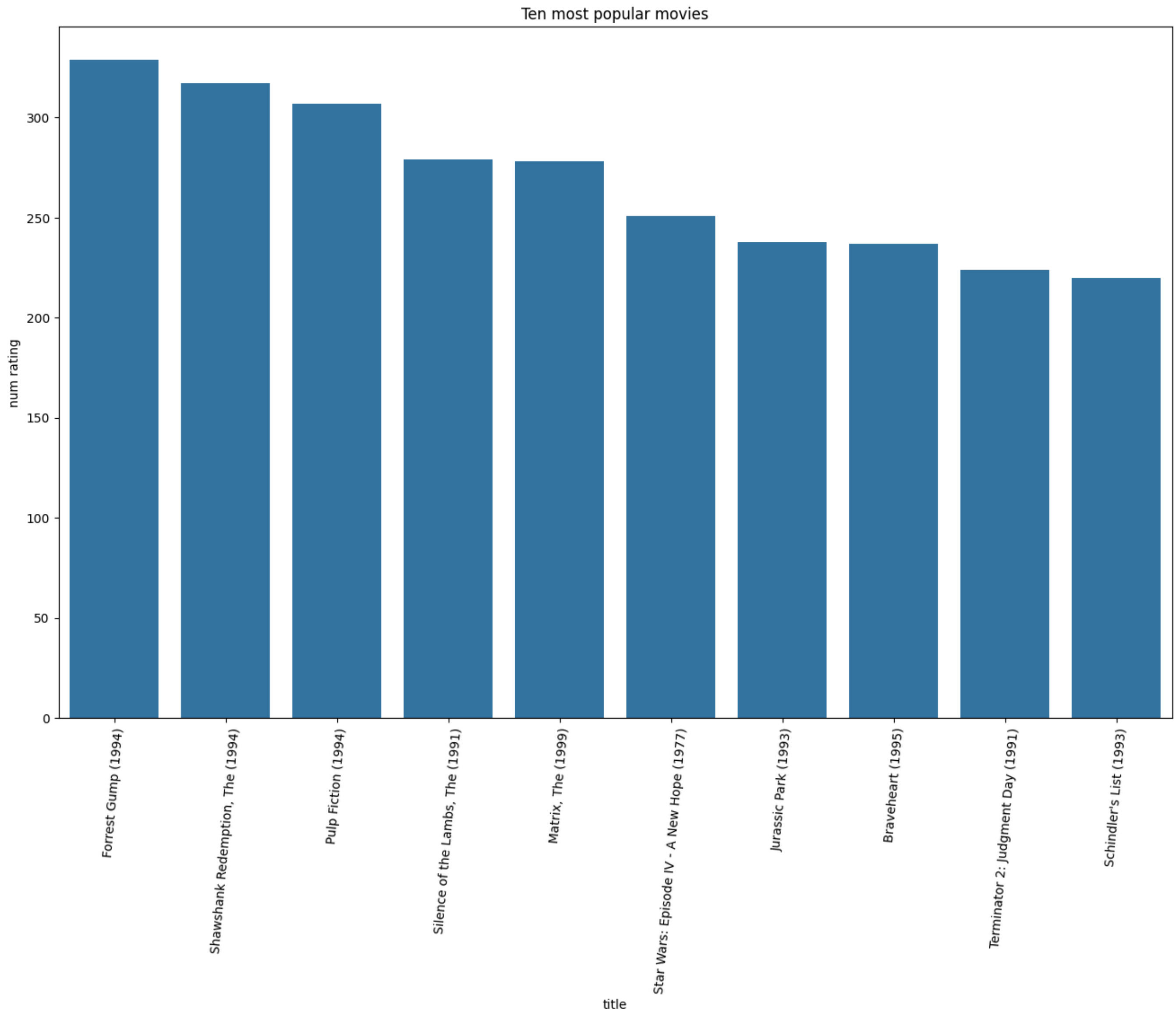
Next steps: Generate code with `df_final` View recommended plots New interactive sheet

```
#Ten most popular movies
popular = df_final.sort_values(by=['num rating'], ascending=False).head(10)
popular
```

| movieId | title | genres | mean rating | num rating |
|---|---|---|---|---|
| 356 | Forrest Gump (1994) | [comedy, drama, romance, war] | 4.164134 | 329.0 |
| 318 | Shawshank Redemption, The (1994) | [crime, drama] | 4.429022 | 317.0 |
| 296 | Pulp Fiction (1994) | [comedy, crime, drama, thriller] | 4.197068 | 307.0 |
| 593 | Silence of the Lambs, The (1991) | [crime, horror, thriller] | 4.161290 | 279.0 |
| 2571 | Matrix, The (1999) | [action, sci-fi, thriller] | 4.192446 | 278.0 |
| 260 | Star Wars: Episode IV - A New Hope (1977) | [action, adventure, sci-fi] | 4.231076 | 251.0 |
| 480 | Jurassic Park (1993) | [action, adventure, sci-fi, thriller] | 3.750000 | 238.0 |
| 110 | Braveheart (1995) | [action, drama, war] | 4.031646 | 237.0 |
| 589 | Terminator 2: Judgment Day (1991) | [action, sci-fi] | 3.970982 | 224.0 |
| 527 | Schindler's List (1993) | [drama, war] | 4.225000 | 220.0 |

Next steps: Generate code with `popular` View recommended plots New interactive sheet

```
# Plotting ten most popular movies
plt.figure(figsize=(16,10))
sns.barplot(x=popular.title, y=popular['num rating'])
plt.xticks(rotation=85)
plt.title("Ten most popular movies")
plt.show()
#plt.savefig("Ten most popular movies")
```




The most popular movie from the above plot is Forrest Gump(1994) with 329 ratings.

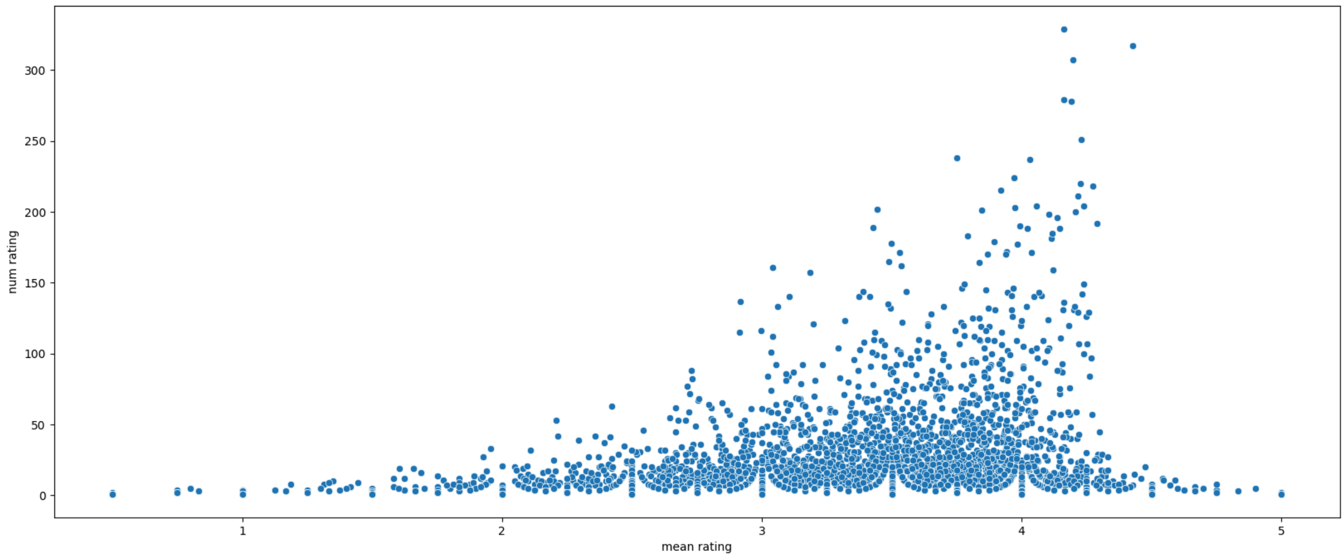
```
# selecting columns to be used for one hot encoding
genres_columns = genres_list[0:-1]

# creating genres columns and filling them with zeros
for genre in genres_columns:
    df_final[genre] = np.zeros(len(df_final))

# filling the genre columns with a 1 if the record exist and a 0 if the record does not exist
for movieId in df_final.index:
    rec_genres = df_final.loc[movieId, 'genres']

    for genre in rec_genres:
        df_final.loc[movieId, genre] = 1


# dropping the '(no genres listed)' column
df_final.drop('(no genres listed)', axis=1, inplace=True)
df_final.head()
```

| movieId | title | genres | mean rating | num rating | adventure | animation | children | comedy | fantasy | romance | ... | thriller | horror | myste |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 3.920930 | 215.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | |
| **2** | Jumanji (1995) | [adventure, children, fantasy] | 3.431818 | 110.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | |
| **3** | Grumpier Old Men (1995) | [comedy, romance] | 3.259615 | 52.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | |
| **4** | Waiting to Exhale (1995) | [comedy, drama, romance] | 2.357143 | 7.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | |
| **5** | Father of the Bride Part II (1995) | [comedy] | 3.071429 | 49.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |

5 rows × 23 columns

Investigating correlation between movies with higher number number of ratings and movies with average ratings

```
# mean rating and  total number of rating scatterplot
plt.figure(figsize=(20,8))
sns.scatterplot(data=df_final, x='mean rating', y ='num rating');
```




It can be deduced that movies that are good, also have a high number of ratings. Hence a correlation between the two.

Naive Recommendation Engine

Naive Recommendation Engine will make use of the overall Ratings and genres in order to make movie recommendtions. This would be helpful especially when resolving Cold-start problem. Cold-start problem occurs when the system encounters new visitors to a website, with no browsing history or known preferences. creating a personalized experience for them becomes a challenge because the data normally used for generating recommendations is missing.

Solution

For the first model we will recommend the top 10 most popular movies. i.e Movies with the most number of ratings that are highly rated.

```
#selecting specific columns from the final data frame.
#assign the columns to a new variable called user ratigs
user_ratings = df_final[['title', 'mean rating', 'num rating']]
user_ratings
```

| movieId | title | mean rating | num rating |
|---|---|---|---|
| 1 | Toy Story (1995) | 3.920930 | 215.0 |
| 2 | Jumanji (1995) | 3.431818 | 110.0 |
| 3 | Grumpier Old Men (1995) | 3.259615 | 52.0 |
| 4 | Waiting to Exhale (1995) | 2.357143 | 7.0 |
| 5 | Father of the Bride Part II (1995) | 3.071429 | 49.0 |
| ... | ... | ... | ... |
| 193581 | Black Butler: Book of the Atlantic (2017) | 4.000000 | 1.0 |
| 193583 | No Game No Life: Zero (2017) | 3.500000 | 1.0 |
| 193585 | Flint (2017) | 3.500000 | 1.0 |
| 193587 | Bungo Stray Dogs: Dead Apple (2018) | 3.500000 | 1.0 |
| 193609 | Andrew Dice Clay: Dice Rules (1991) | 4.000000 | 1.0 |

9724 rows × 3 columns

Next steps: Generate code with user_ratings | View recommended plots | New interactive sheet

```
#sort the new dataFrame by number of ratings from highest to lowest
#Top 10 movies by its number of ratings
user_ratings.sort_values(by=['num rating'], ascending=False).head(10)
```

| movieId | title | mean rating | num rating |
|---|---|---|---|
| 356 | Forrest Gump (1994) | 4.164134 | 329.0 |
| 318 | Shawshank Redemption, The (1994) | 4.429022 | 317.0 |
| 296 | Pulp Fiction (1994) | 4.197068 | 307.0 |
| 593 | Silence of the Lambs, The (1991) | 4.161290 | 279.0 |
| 2571 | Matrix, The (1999) | 4.192446 | 278.0 |
| 260 | Star Wars: Episode IV - A New Hope (1977) | 4.231076 | 251.0 |
| 480 | Jurassic Park (1993) | 3.750000 | 238.0 |
| 110 | Braveheart (1995) | 4.031646 | 237.0 |
| 589 | Terminator 2: Judgment Day (1991) | 3.970982 | 224.0 |
| 527 | Schindler's List (1993) | 4.225000 | 220.0 |

The above dataFrame shows the movies with the highest number of Rating. Lets also analyse the movies with the highest rating.

```
# top 10 movies by its mean ratings
user_ratings.sort_values(by=['mean rating'], ascending=False).head(10)
```

| movieId | title | mean rating | num rating |
|---|---|---|---|
| 88448 | Paper Birds (Pájaros de papel) (2010) | 5.0 | 1.0 |
| 100556 | Act of Killing, The (2012) | 5.0 | 1.0 |
| 143031 | Jump In! (2007) | 5.0 | 1.0 |
| 143511 | Human (2015) | 5.0 | 1.0 |
| 143559 | L.A. Slasher (2015) | 5.0 | 1.0 |
| 6201 | Lady Jane (1986) | 5.0 | 1.0 |
| 102217 | Bill Hicks: Revelations (1993) | 5.0 | 1.0 |
| 102084 | Justice League: Doom (2012) | 5.0 | 1.0 |
| 6192 | Open Hearts (Elsker dig for evigt) (2002) | 5.0 | 1.0 |
| 145994 | Formula of Love (1984) | 5.0 | 1.0 |

The above dataFrame does not provide very useful information the recomender since some movies might have only been rated by one user thus its average rating being 5. We have to also consider movies that have a higher average rating that have been rated by many users.

In the cell below we will analyse top 10 highly rated movies that have been rated by 200 users200 num of ratings

```
#creating minimum number of ratings
#find the most popular movies in the data frame.
threshold = 200
user_ratings[user_ratings['num rating']>threshold ].sort_values(by=['mean rating'], ascending=False).head(10)
```

| movieId | title | mean rating | num rating |
|---|---|---|---|
| **318** | Shawshank Redemption, The (1994) | 4.429022 | 317.0 |
| **2959** | Fight Club (1999) | 4.272936 | 218.0 |
| **50** | Usual Suspects, The (1995) | 4.237745 | 204.0 |
| **260** | Star Wars: Episode IV - A New Hope (1977) | 4.231076 | 251.0 |
| **527** | Schindler's List (1993) | 4.225000 | 220.0 |
| **1196** | Star Wars: Episode V - The Empire Strikes Back... | 4.215640 | 211.0 |
| **296** | Pulp Fiction (1994) | 4.197068 | 307.0 |
| **2571** | Matrix, The (1999) | 4.192446 | 278.0 |
| **356** | Forrest Gump (1994) | 4.164134 | 329.0 |
| **593** | Silence of the Lambs, The (1991) | 4.161290 | 279.0 |

The dataframe above finaly gives the most popular movies which are also loved by most users.

To further refine our recommendations we will use Genre information from the final EDA dataFrame. A user might want to be recommended a specific genre e.g Animation Hence the need to refine the recommendation to the users preference

```
#using the final data frame from EDA
df_final.head()
```

| movieId | title | genres | mean rating | num rating | adventure | animation | children | comedy | fantasy | romance | ... | thriller | horror | myste |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 3.920930 | 215.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | |
| **2** | Jumanji (1995) | [adventure, children, fantasy] | 3.431818 | 110.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | |
| **3** | Grumpier Old Men (1995) | [comedy, romance] | 3.259615 | 52.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | |
| **4** | Waiting to Exhale (1995) | [comedy, drama, romance] | 2.357143 | 7.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | |
| **5** | Father of the Bride Part II (1995) | [comedy] | 3.071429 | 49.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | |

5 rows × 23 columns

```
# selecting the top movies given a certain genre and number of rating threshold
genre = 'animation'
rating_thresh = 200
df_final[(df_final[genre] == 1) & (df_final['num rating']>rating_thresh)].sort_values(by=['mean rating'], ascending=False).head(10)
```

| | title | genres | mean rating | num rating | adventure | animation | children | comedy | fantasy | romance | ... | thriller | horror | mystery |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| movieId | | | | | | | | | | | | | | |
| 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 3.92093 | 215.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 |

1 rows × 23 columns

Naive Recommendation for a new user

Building the basic Recommendation Engine.

For cases where there is no information for users, the recommender will recommend movies rated more than the set threshold with the highest rating. For cases where the user wants recommendation for specific genre, the recommender will recommend only movies from that specified genre that have the highest rating

```
# function to return all top movies
def good_movies(threshold=10):
    return df_final[df_final['num rating']>rating_thresh ].sort_values(by=['mean rating'], ascending=False).iloc[:,0:4].head(threshold)



# function to recommend to a user certain movies given a certain genre
def recommender(genre, threshold=10):
    # getting the last record
    rating_thresh = list(df_final[ df_final[genre] == 1 ]['num rating'].to_frame().sort_values('num rating', ascending=False)['num ratin

    result = df_final[(df_final[genre] == 1) & (df_final['num rating']>=rating_thresh)].sort_values(by=['mean rating'], ascending=False)

    # print('\n\nThese are the recommendations for the users with the following filters')
    # print('Minimum number of ratings:',threshold)
    # print("User's choice of genre:",genre)
    # display(result)
    return result


def movie_recommender(genre=None, threshold=10):
    if genre:
        return recommender(genre,threshold)
    else:
        return good_movies(threshold)
```

```
movie_recommender()
```

| | title | genres | mean rating | num rating |
|---|---|---|---|---|
| movieId | | | | |
| 318 | Shawshank Redemption, The (1994) | [crime, drama] | 4.429022 | 317.0 |
| 2959 | Fight Club (1999) | [action, crime, drama, thriller] | 4.272936 | 218.0 |
| 50 | Usual Suspects, The (1995) | [crime, mystery, thriller] | 4.237745 | 204.0 |
| 260 | Star Wars: Episode IV - A New Hope (1977) | [action, adventure, sci-fi] | 4.231076 | 251.0 |
| 527 | Schindler's List (1993) | [drama, war] | 4.225000 | 220.0 |
| 1196 | Star Wars: Episode V - The Empire Strikes Back... | [action, adventure, sci-fi] | 4.215640 | 211.0 |
| 296 | Pulp Fiction (1994) | [comedy, crime, drama, thriller] | 4.197068 | 307.0 |
| 2571 | Matrix, The (1999) | [action, sci-fi, thriller] | 4.192446 | 278.0 |
| 356 | Forrest Gump (1994) | [comedy, drama, romance, war] | 4.164134 | 329.0 |
| 593 | Silence of the Lambs, The (1991) | [crime, horror, thriller] | 4.161290 | 279.0 |

```
movie_recommender('sci-fi',5)
```

| movieId | title | genres | mean rating | num rating |
|---|---|---|---|---|
| 260 | Star Wars: Episode IV - A New Hope (1977) | [action, adventure, sci-fi] | 4.231076 | 251.0 |
| 1196 | Star Wars: Episode V - The Empire Strikes Back... | [action, adventure, sci-fi] | 4.215640 | 211.0 |
| 2571 | Matrix, The (1999) | [action, sci-fi, thriller] | 4.192446 | 278.0 |
| 1210 | Star Wars: Episode VI - Return of the Jedi (1983) | [action, adventure, sci-fi] | 4.137755 | 196.0 |
| 1270 | Back to the Future (1985) | [adventure, comedy, sci-fi] | 4.038012 | 171.0 |

Content-Based Recommender

A content based recommender works with data that the user provides, either explicitly rating or implicitly clicking on a link. Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

```
ratings_df_cleaned.head()
```

| | userId | movieId | rating |
|---|---|---|---|
| 0 | 1 | 1 | 4.0 |
| 1 | 1 | 3 | 4.0 |
| 2 | 1 | 6 | 4.0 |
| 3 | 1 | 47 | 5.0 |
| 4 | 1 | 50 | 5.0 |

```
movies_df_cleaned.head()
```

| | movieId | title | genres |
|---|---|---|---|
| 0 | 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] |
| 1 | 2 | Jumanji (1995) | [adventure, children, fantasy] |
| 2 | 3 | Grumpier Old Men (1995) | [comedy, romance] |
| 3 | 4 | Waiting to Exhale (1995) | [comedy, drama, romance] |
| 4 | 5 | Father of the Bride Part II (1995) | [comedy] |

Next steps: Generate code with movies_df_cleaned | View recommended plots | New interactive sheet

```
# merging ratings and movies data
df = pd.merge(ratings_df_cleaned, movies_df_cleaned, on = 'movieId')
df.head()
```

| | userId | movieId | rating | title | genres |
|---|---|---|---|---|---|
| 0 | 1 | 1 | 4.0 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] |
| 1 | 1 | 3 | 4.0 | Grumpier Old Men (1995) | [comedy, romance] |
| 2 | 1 | 6 | 4.0 | Heat (1995) | [action, crime, thriller] |
| 3 | 1 | 47 | 5.0 | Seven (a.k.a. Se7en) (1995) | [mystery, thriller] |
| 4 | 1 | 50 | 5.0 | Usual Suspects, The (1995) | [crime, mystery, thriller] |

```
ratings = pd.DataFrame(df.groupby('title')['rating'].mean())
ratings['num rating'] =pd.DataFrame(df.groupby('title')['rating'].count())
ratings.head()
```

| title | rating | num rating |
|---|---|---|
| '71 (2014) | 4.0 | 1 |
| 'Hellboy': The Seeds of Creation (2004) | 4.0 | 1 |
| 'Round Midnight (1986) | 3.5 | 2 |
| 'Salem's Lot (2004) | 5.0 | 1 |
| 'Til There Was You (1997) | 4.0 | 2 |

Next steps: Generate code with ratings View recommended plots New interactive sheet

For this second Recommender Engine we will use correlation between the ratings assigned to different movies,in order to find the similarity between the movies. We created a matrix that has the userID on one axis and movie title on another axis.Each cell will then consist of the ratings the user gave to the movie.

```
# creating a pivot table
movie_matrix = df.pivot_table(index='userId', columns= 'title',values='rating')
movie_matrix.head()
```

| title<br>userId | '71 (2014) | 'Hellboy': The Seeds of Creation (2004) | 'Round Midnight (1986) | 'Salem's Lot (2004) | 'Til There Was You (1997) | 'Tis the Season for Love (2015) | 'burbs, The (1989) | 'night Mother (1986) | (500) Days of Summer (2009) | *batteries not included (1987) | ... | Zulu (2013) | [REC] (2007) | [REC]² (2009) | [R Gén (2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |
| 3 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |
| 4 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |
| 5 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | |

5 rows × 9719 columns

```
# top 5 movies with the most number of ratings
ratings.sort_values('num rating',ascending = False).head()
```

| title | rating | num rating |
|---|---|---|
| Forrest Gump (1994) | 4.164134 | 329 |
| Shawshank Redemption, The (1994) | 4.429022 | 317 |
| Pulp Fiction (1994) | 4.197068 | 307 |
| Silence of the Lambs, The (1991) | 4.161290 | 279 |
| Matrix, The (1999) | 4.192446 | 278 |

We will choose the Matrix, The (1999) to use as an example. Imagine a user who has watched the Matrix, The (1999) movie and wants to be recommeneded a new movie to watch that is similar to the Matrix, The (1999)

First we find the ratings for the Matrix, The (1999) movie from the matrix created before

```
#finds the user ratings for the matrix movie from the movie matrix
movie_name = "Matrix, The (1999)"
matrix_user_ratings = movie_matrix[movie_name]
matrix_user_ratings .head()
```

| userId | Matrix, The (1999) |
|---|---|
| 1 | 5.0 |
| 2 | NaN |
| 3 | NaN |
| 4 | 1.0 |
| 5 | NaN |

We then used the corrwith method in pandas to find movies that are similar to the matrix movie based on the user ratigs

```
# using correlation to find movies similar to the matrix
similar_to_matrix = movie_matrix.corrwith(matrix_user_ratings)
corr_matrix =  pd.DataFrame(similar_to_matrix, columns=['Correlation'])
#drop null values from the dataframe
corr_matrix.dropna(inplace=True)
corr_matrix.head()
```

```
/usr/local/lib/python3.11/dist-packages/numpy/lib/function_base.py:2889: RuntimeWarning: Degrees of freedom <= 0 for slice
  c = cov(x, y, rowvar, dtype=dtype)
/usr/local/lib/python3.11/dist-packages/numpy/lib/function_base.py:2748: RuntimeWarning: divide by zero encountered in divide
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.11/dist-packages/numpy/lib/function_base.py:2748: RuntimeWarning: invalid value encountered in multiply
  c *= np.true_divide(1, fact)
/usr/local/lib/python3.11/dist-packages/numpy/lib/function_base.py:2897: RuntimeWarning: invalid value encountered in divide
  c /= stddev[:, None]
/usr/local/lib/python3.11/dist-packages/numpy/lib/function_base.py:2898: RuntimeWarning: invalid value encountered in divide
  c /= stddev[None, :]
```

| title | Correlation |
|---|---|
| 'burbs, The (1989) | -0.160843 |
| (500) Days of Summer (2009) | 0.302316 |
| *batteries not included (1987) | 0.392232 |
| ...And Justice for All (1979) | 0.654654 |
| 10 Cent Pistol (2015) | -1.000000 |

Next steps: Generate code with corr_matrix | View recommended plots | New interactive sheet

```
#sort the values based on correlation with the matrix movie
corr_matrix.sort_values('Correlation',ascending = False).head(10)
```

| title | Correlation |
|---|---|
| Haywire (2011) | 1.0 |
| Highway 61 (1991) | 1.0 |
| World on a Wire (Welt am Draht) (1973) | 1.0 |
| War Zone, The (1999) | 1.0 |
| Hitcher, The (1986) | 1.0 |
| Gross Anatomy (a.k.a. A Cut Above) (1989) | 1.0 |
| Paper Towns (2015) | 1.0 |
| Juwanna Mann (2002) | 1.0 |
| Topsy-Turvy (1999) | 1.0 |
| All the King's Men (2006) | 1.0 |

The above dataFrame may be biased since some movies might have only been rated by only one user. Therefore we need to also consider number of ratings

```
#joining the number of rating colum to the above data frame
corr_with_matrix = corr_matrix.join(ratings['num rating'])
corr_with_matrix.head()
```

| title | Correlation | num rating |
|---|---|---|
| 'burbs, The (1989) | -0.160843 | 17 |
| (500) Days of Summer (2009) | 0.302316 | 42 |
| *batteries not included (1987) | 0.392232 | 7 |
| ...And Justice for All (1979) | 0.654654 | 3 |
| 10 Cent Pistol (2015) | 1.000000 | 2 |

Next steps: Generate code with corr_with_matrix View recommended plots New interactive sheet

We can the get the most similar movies to the Matrix, The (1999) and recommend to the user.

```
#choosing a threshold of number of ratings to be considered
#sort the data frame based on correltion and number of ratings
threshhold = 100
corr_with_matrix[corr_with_matrix['num rating']>threshhold].sort_values('Correlation',ascending = False).head()
```

| title | Correlation | num rating |
|---|---|---|
| Matrix, The (1999) | 1.000000 | 278 |
| Die Hard (1988) | 0.544466 | 145 |
| Inception (2010) | 0.514767 | 143 |
| Braveheart (1995) | 0.496045 | 237 |
| Aliens (1986) | 0.470865 | 126 |

Implementing a Recommendation Engine using surprise library

```
# reading values as a surprise dataset
reader = Reader(rating_scale=(0,5))
data = Dataset.load_from_df(ratings_df_cleaned, reader)

# genreating a trainset
dataset = data.build_full_trainset()
print('Number of users:', dataset.n_users, '\n')
print('Number of items:', dataset.n_items)
```

```
Number of users: 610

Number of items: 9724
```

We have fewer users compared to the number of items. We will take that into account, and prefer to use a user based reccomendation.

Training Various models

```
import pandas as pd
from surprise import SVD, KNNBaseline, KNNBasic, KNNWithMeans, KNNWithZScore
from surprise.model_selection import cross_validate

benchmark = []
# Iterate over all algorithms
for algorithm in [SVD(), KNNBaseline(), KNNBasic(), KNNWithMeans(), KNNWithZScore()]:
    # Perform cross validation
    results = cross_validate(algorithm, data, measures=['RMSE', 'MAE'], cv=3, verbose=False)

    # Get results & append algorithm name
    tmp = pd.DataFrame.from_dict(results).mean(axis=0)
    algorithm_name = str(algorithm).split(' ')[0].split('.')[-1]
    tmp = pd.concat([tmp, pd.Series([algorithm_name], index=['Algorithm'])])
    benchmark.append(tmp)

# Create a DataFrame from the benchmark list and sort by test_rmse
benchmark_df = pd.DataFrame(benchmark).set_index('Algorithm').sort_values('test_rmse')
print(benchmark_df)
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
Computing the msd similarity matrix...
Done computing similarity matrix.
               test_rmse  test_mae  fit_time  test_time
Algorithm
SVD             0.880200  0.676906  1.707945   0.264520
KNNBaseline     0.884781  0.676075  0.357997   2.205466
KNNWithZScore   0.903358  0.684660  0.185782   2.159288
KNNWithMeans    0.906545  0.692642  0.106231   2.040224
KNNBasic        0.960092  0.736215  0.082879   2.085297
```

While SVDpp had the better performance in terms of error rate, it is very time consuming to train. A grid search on SVDpp will last for a long time. Therefore we will chose to optimize the SVD model for number of epochs, learning rate and regularization and the KNNBaseline model as well. This are so far our most promising models with lower error rates.

```
# We perform gridsearch using svd
param_grid = {"n_factors": [10, 20, 30, 40, 50, 60], "reg_all": [.02, .05, .1]}
grid_search = GridSearchCV(SVD, param_grid= param_grid)
grid_search.fit(data)
```

```
# printing out optimal parameters for the gridsearch
print("Optimal gridsearch params: ", grid_search.best_params)
print("")
print("Best scores: ", grid_search.best_score)
```

```
Optimal gridsearch params:  {'rmse': {'n_factors': 30, 'reg_all': 0.05}, 'mae': {'n_factors': 50, 'reg_all': 0.05}}

Best scores:  {'rmse': 0.8693234603651032, 'mae': 0.6684247775741651}
```

```
# using obtained optimals
param_grid= {"n_factors": [30], "reg_all":[.05], "n_epochs":[5, 10, 20, 30], "lr_all":[.0025, .005, .001, .01]}
grid_search = GridSearchCV(SVD, param_grid= param_grid)
grid_search.fit(data)
```

```
# printing out optimal parameters for the gridsearch
print("Optimal gridsearch params: ", grid_search.best_params)
print("")
print("Best scores: ", grid_search.best_score)
```

```
Optimal gridsearch params:  {'rmse': {'n_factors': 30, 'reg_all': 0.05, 'n_epochs': 20, 'lr_all': 0.01}, 'mae': {'n_factors': 30, 'r

Best scores:  {'rmse': 0.8605650823920522, 'mae': 0.659767781257759}
```

From above our best SVD model has the following parameters:

n_factors= 30 reg_all= .05 n_epochs= 20 lr_all= .01

Below we instantiate the model with this parameters

```
svd_final = SVD(n_factors= 30, reg_all= .05, n_epochs= 20, lr_all= .01)
svd_final.fit(dataset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7d7b28ec1a10>
```

We tune the KNNBaseline model below

```
param_grid = {"k": list(range(5, 100, 5))}
grid_search_knn_base = GridSearchCV(KNNBaseline, param_grid= param_grid)
grid_search_knn_base.fit(data)
```

```
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
Computing the msd similarity matrix...
Done computing similarity matrix.
Estimating biases using als...
```

```
# printing out optimal parameters for the gridsearch
print("Optimal gridsearch params: ", grid_search_knn_base.best_params)
print("")
print("Best scores: ", grid_search_knn_base.best_score)
```

```
Optimal gridsearch params:  {'rmse': {'k': 30}, 'mae': {'k': 30}}

Best scores:  {'rmse': 0.8745377630129649, 'mae': 0.6686380497455451}
```

Below we make a loop to iterate over the different parameters as we alternate between user based and item based recommendation.

```
method = ["cosine", "pearson", "pearson_baseline", "msd"]
user_base = [False, True]

for m in method:
  for u in user_base:
    knn_base = cross_validate(KNNBaseline(k= 30, sim_options= {"name":m, "user_based":u}, verbose= False), data, cv= 5)
    rmse_mean = np.mean(knn_base["test_rmse"])
    print(f"The mean rmse of the KNNBaseline model with {m} and user based {u} is {rmse_mean}")
```

```
The mean rmse of the KNNBaseline model with cosine and user based False is 0.8960639629178981
The mean rmse of the KNNBaseline model with cosine and user based True is 0.8773954251194903
```

```
The mean rmse of the KNNBaseline model with pearson and user based False is 0.8838639689955679
The mean rmse of the KNNBaseline model with pearson and user based True is 0.8780110799109695
The mean rmse of the KNNBaseline model with pearson_baseline and user based False is 0.8523396742612281
The mean rmse of the KNNBaseline model with pearson_baseline and user based True is 0.8779325612909356
The mean rmse of the KNNBaseline model with msd and user based False is 0.8701759075624584
The mean rmse of the KNNBaseline model with msd and user based True is 0.8743229080241871
```

The best performing model from above is the item based and uses the pearson_baseline method, we will instantiate the method below then fit it to the dataset. But, since we already have decided not to use an item based recommender, we will proceed with the SVD model.

```
knn_baseline_final = KNNBaseline(k= 30, sim_options= {"name": "pearson_baseline", "user_based": False})
knn_baseline_final.fit(dataset)
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNBaseline at 0x7d7b357dfe10>
```

Picking user 610, we will attempt to predict the rating the user will provide for Toy story 3 given how they highly rated Toy story.

```
# df_final[(df_final['userId'] == 610) & (df_final['movieId'] == 1)][['userId', 'movieId', 'rating', 'title']]
ratings_df_cleaned[(ratings_df_cleaned["userId"] == 1) & (ratings_df_cleaned["movieId"] == 1)]
```

| | userId | movieId | rating |
|---|---|---|---|
| 0 | 1 | 1 | 4.0 |

```
knn_baseline_final.predict(610, 78499)
```

```
Prediction(uid=610, iid=78499, r_ui=None, est=4.6114622486933605, details={'actual_k': 30, 'was_impossible': False})
```

We predict a rating of about 4.5 which is pretty high in the rating scale. We try the same using the SVD model:

```
svd_final.predict(610, 78499)
```

```
Prediction(uid=610, iid=78499, r_ui=None, est=4.300877158521295, details={'was_impossible': False})
```

A prediction of a rating of about 4.5, this indicates that both models could be performing the same, despite the different rmse scores.

```
ratings_df_cleaned.head()
```

| | userId | movieId | rating |
|---|---|---|---|
| 0 | 1 | 1 | 4.0 |
| 1 | 1 | 3 | 4.0 |
| 2 | 1 | 6 | 4.0 |
| 3 | 1 | 47 | 5.0 |
| 4 | 1 | 50 | 5.0 |

```
movies_df_cleaned.head()
```

| | movieId | title | genres |
|---|---|---|---|
| 0 | 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] |
| 1 | 2 | Jumanji (1995) | [adventure, children, fantasy] |
| 2 | 3 | Grumpier Old Men (1995) | [comedy, romance] |
| 3 | 4 | Waiting to Exhale (1995) | [comedy, drama, romance] |
| 4 | 5 | Father of the Bride Part II (1995) | [comedy] |

Next steps: Generate code with movies_df_cleaned | View recommended plots | New interactive sheet

```
df_final.head(3)
```

| movieId | title | genres | mean rating | num rating | adventure | animation | children | comedy | fantasy | romance | ... | thriller | horror | myste |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | Toy Story (1995) | [adventure, animation, children, comedy, fantasy] | 3.920930 | 215.0 | 1.0 | 1.0 | 1.0 | 1.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | |
| 2 | Jumanji (1995) | [adventure, children, fantasy] | 3.431818 | 110.0 | 1.0 | 0.0 | 1.0 | 0.0 | 1.0 | 0.0 | ... | 0.0 | 0.0 | |
| 3 | Grumpier Old Men (1995) | [comedy, romance] | 3.259615 | 52.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 1.0 | ... | 0.0 | 0.0 | |

3 rows × 23 columns

```
# pickle the final model
filename = 'svd_final.sav'
pickle.dump(svd_final, open(filename, 'wb'))


# reloading the model
load_svd = pickle.load(open(filename, 'rb'))
```

Final Engine

```
def hybrid_recommendation_engine(user_id='new', preferred_genre=None, threshold=50):

    if user_id=='new':
        if preferred_genre:
            # result = df_final[(df_final['num rating']>minimum_num_ratings)].sort_values(by=['mean rating'], ascending=False).head(10)
            result = movie_recommender(threshold= threshold)
        else:
            # result = df_final[(df_final[preferred_genre] == 1) & (movies_df['num rating']>minimum_num_ratings)].sort_values(by=['mean
            result = movie_recommender(preferred_genre, threshold= threshold)

    else:
        new_df = df_final.copy()

        rating_thresh = list(df_final[ df_final[genre] == 1 ]['num rating'].to_frame().sort_values('num rating', ascending=False)['num r
        # filtering out by genre
        if preferred_genre !='all':
            new_df = new_df[new_df[preferred_genre]==1]

        # filtering out by number of ratings
        new_df = new_df[new_df['num rating']>=rating_thresh]

        # filtering out all movies already rated by user
        movies_already_watched = set(ratings_df_cleaned[ratings_df_cleaned['userId']==user_id].movieId.values)
        new_df= new_df[~new_df.index.isin(movies_already_watched)]

        # finding expected ratings for all remaining movies in the dataset
        all_movie_ids = set(new_df.index)
        all_movie_ratings = []

        for i in all_movie_ids:
            expected_rating = load_svd.predict(uid=user_id, iid=i).est
            all_movie_ratings.append((i,round(expected_rating,1)))

        # extracting top five movies by expected rating
        expected_df = pd.DataFrame(all_movie_ratings, columns=['movieId','Expected Rating'])
        result = pd.concat([expected_df, df_final[['title','num rating']]], axis= 1)
        result = result.sort_values(['num rating'],ascending= False)
        result.dropna(inplace= True)
        result = result.head()


    print('\n\nThese are the recommendations for the users with the following filters')
    print('User id:',user_id)
    print('Minimum number of ratings:',threshold)
    print("User's choice of genre:", preferred_genre)
    display(result)


hybrid_recommendation_engine()
```

```
These are the recommendations for the users with the following filters
User id: new
Minimum number of ratings: 50
User's choice of genre: None
```

| movieId | title | genres | mean rating | num rating |
|---|---|---|---|---|
| 318 | Shawshank Redemption, The (1994) | [crime, drama] | 4.429022 | 317.0 |
| 2959 | Fight Club (1999) | [action, crime, drama, thriller] | 4.272936 | 218.0 |
| 50 | Usual Suspects, The (1995) | [crime, mystery, thriller] | 4.237745 | 204.0 |
| 260 | Star Wars: Episode IV - A New Hope (1977) | [action, adventure, sci-fi] | 4.231076 | 251.0 |
| 527 | Schindler's List (1993) | [drama, war] | 4.225000 | 220.0 |
| 1196 | Star Wars: Episode V - The Empire Strikes Back... | [action, adventure, sci-fi] | 4.215640 | 211.0 |