Exercise 9.2 Recommender System

Using the small MovieLens data set, create a recommender system that allows users to input a movie they like (in the data set) and recommends ten other movies for them to watch. In your write-up, clearly explain the recommender system process and all steps performed.

Setting up data

Importing required packages Uploading data - Ratings and Movies list data into dataframes

In [1]:
```
## Importing the required pacakges
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
## Load the ratings data into a dataframe
ratings_df = pd.read_csv("ml-latest-small/ratings.csv")
ratings_df
```

Out[2]:

	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
...	...	...	...	...
100831	610	166534	4.0	1493848402
100832	610	168248	5.0	1493850091
100833	610	168250	5.0	1494273047
100834	610	168252	5.0	1493846352
100835	610	170875	3.0	1493846415

100836 rows × 4 columns

In [3]:
```
movies_df = pd.read_csv("ml-latest-small/movies.csv")
movies_df.head()
```

Out[3]:

	movieId	title	genres

	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

Data Analysis

In [4]:
```
## statistics from the ratings_df dataset

n_ratings = len(ratings_df)
n_movies = len(ratings_df['movieId'].unique())
n_users = len(ratings_df['userId'].unique())

print(f"Number of ratings: {n_ratings}")
print(f"Number of unique movieId's: {n_movies}")
print(f"Number of unique users: {n_users}")
print(f"Average ratings per user: {round(n_ratings/n_users, 2)}")
print(f"Average ratings per movie: {round(n_ratings/n_movies, 2)}")
```

```
Number of ratings: 100836
Number of unique movieId's: 9724
Number of unique users: 610
Average ratings per user: 165.3
Average ratings per movie: 10.37
```

There are 100836 ratings in the dataset given by 610 unique users on 9724 movies. On an average a user has given a rating of 165.3 and each movie has received an average rating of 10.37.

Deriving the user engagement in terms of number of ratings provided by each user.

In [5]:
```
user_freq = ratings_df[['userId', 'movieId']].groupby('userId').count().reset_index()
user_freq.columns = ['userId', 'n_ratings']
user_freq
```

Out[5]:

	userId	n_ratings
0	1	232
1	2	29
2	3	39
3	4	216
4	5	44
...	...	...
605	606	1115
606	607	187

	userId	n_ratings
607	608	831
608	609	37
609	610	1302

610 rows × 2 columns

Finding the lowest and highest rated movies and the number of users providing those ratings.

In [6]:
```
# Find Lowest and Highest rated movies:
mean_rating = ratings_df.groupby('movieId')[['rating']].mean()
# Lowest rated movies
lowest_rated = mean_rating['rating'].idxmin()
movies_df.loc[movies_df['movieId'] == lowest_rated]
```

Out[6]:

	movieId	title	genres
2689	3604	Gypsy (1962)	Musical

In [7]:
```
# show number of people who rated movies rated movie lowest
ratings_df[ratings_df['movieId']==lowest_rated]
```

Out[7]:

	userId	movieId	rating	timestamp
13633	89	3604	0.5	1520408880

One user provided the lowest rating to the movie 'Gypsy'

In [8]:
```
# Highest rated movies
highest_rated = mean_rating['rating'].idxmax()
movies_df.loc[movies_df['movieId'] == highest_rated]
```

Out[8]:

	movieId	title	genres
48	53	Lamerica (1994)	Adventure|Drama

In [9]:
```
# show number of people who rated movies rated movie highest
ratings_df[ratings_df['movieId']==highest_rated]
```

Out[9]:

	userId	movieId	rating	timestamp
13368	85	53	5.0	889468268
96115	603	53	5.0	963180003

Two users provided the highest rating to the movie 'Lamerica'

In [10]:
```
## the above movies has very low dataset. We will use bayesian average
movie_stats = ratings_df.groupby('movieId')[['rating']].agg(['count', 'mean'])
movie_stats.columns = movie_stats.columns.droplevel()
```

In [11]:
```
movie_stats
```

Out[11]:

movieId	count	mean
1	215	3.920930
2	110	3.431818
3	52	3.259615
4	7	2.357143
5	49	3.071429
...	...	...
193581	1	4.000000
193583	1	3.500000
193585	1	3.500000
193587	1	3.500000
193609	1	4.000000

9724 rows × 2 columns

By using the bayesian average, we have derived the statistics for each movie, how many users provided rating and the average rating.

Creating the User and Movie matrix using scipy csr_matrix package

This will give a matrix for all movie titles against each user. The value will be the rating the user has provided for each movie.

In [12]:
```
# Now, we create user-item matrix using scipy csr matrix
from scipy.sparse import csr_matrix
```

In [13]:
```
## Function to create user item matrix
def create_matrix(df):

    N = len(df['userId'].unique())
    M = len(df['movieId'].unique())

    # Map Ids to indices
    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))
```

```
    # Map indices to IDs
    user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

    X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

    return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper
```

In [14]:
```
## Calling the create matrix function and assign to the variables
X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_matrix(ratings
```

In [15]:
```
X
```

Out[15]:
```
<9724x610 sparse matrix of type '<class 'numpy.float64'>'
        with 100836 stored elements in Compressed Sparse Row format>
```

CSR Matrix has been created with the list of user id and movie ids present in the dataset. Upon creating the matrix, the following values are returned from the function.

X: A Matrix of movie ids and user ids and the ratings against each movie as values user_mapper: a dict of unique User Ids movie_mapper: a dict of Unique Movie Ids user_inv_mapper: Mapping indices to each user id movie_inv_mapper: Mapping indices to each movie id

In [26]:
```
## Creating dictionary with movie id as key and title as value
movie_titles = dict(zip(movies_df['movieId'], movies_df['title']))
```

In [24]:
```
movie_mapper.keys()
```

Out[24]:
```
dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 34, 36, 38, 39, 40, 41, 42, 43, 44, 45, 46, 4
7, 48, 49, 50, 52, 53, 54, 55, 57, 58, 60, 61, 62, 63, 64, 65, 66, 68, 69, 70, 71, 72, 7
3, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 85, 86, 87, 88, 89, 92, 93, 94, 95, 96, 97, 9
9, 100, 101, 102, 103, 104, 105, 106, 107, 108, 110, 111, 112, 113, 116, 117, 118, 119,
121, 122, 123, 125, 126, 128, 129, 132, 135, 137, 140, 141, 144, 145, 146, 147, 148, 14
9, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166,
168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 183, 184, 185, 18
6, 187, 188, 189, 190, 191, 193, 194, 195, 196, 198, 199, 201, 202, 203, 204, 205, 206,
207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 222, 223, 224, 22
5, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243,
246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 26
3, 265, 266, 267, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282,
283, 284, 285, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 298, 299, 300, 301, 30
2, 303, 304, 305, 306, 307, 308, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320,
321, 322, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 33
9, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356,
357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 37
4, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 393,
405, 406, 407, 408, 409, 410, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 42
3, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 440, 441,
442, 444, 445, 446, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 46
1, 464, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481,
482, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 499, 500, 50
```

```
1, 151557, 151559, 151653, 151687, 151695, 151739, 151745, 151759, 151763, 151769, 15177
7, 151781, 152037, 152063, 152065, 152071, 152077, 152079, 152081, 152083, 152085, 15209
1, 152105, 152173, 152270, 152284, 152372, 152591, 152658, 152711, 152970, 153070, 15323
6, 153386, 153408, 154065, 154358, 154975, 155064, 155168, 155288, 155358, 155509, 15558
9, 155659, 155743, 155774, 155812, 155820, 155892, 156025, 156371, 156387, 156553, 15660
5, 156607, 156609, 156675, 156706, 156726, 156781, 156783, 157108, 157110, 157122, 15713
0, 157172, 157200, 157270, 157296, 157312, 157340, 157369, 157407, 157432, 157699, 15777
5, 157865, 158022, 158027, 158035, 158238, 158254, 158388, 158398, 158402, 158528, 15872
1, 158783, 158813, 158830, 158842, 158872, 158874, 158882, 158956, 158966, 158972, 15906
1, 159069, 159077, 159093, 159161, 159193, 159403, 159415, 159441, 159510, 159690, 15971
7, 159755, 159779, 159811, 159817, 159849, 159858, 159976, 160080, 160271, 160289, 16034
1, 160400, 160422, 160438, 160440, 160527, 160563, 160565, 160567, 160569, 160571, 16057
3, 160644, 160646, 160684, 160718, 160730, 160836, 160848, 160872, 160954, 160978, 16098
0, 161008, 161024, 161032, 161044, 161127, 161131, 161290, 161354, 161580, 161582, 16159
4, 161634, 161830, 161918, 161922, 161966, 162082, 162344, 162350, 162414, 162478, 16257
8, 162590, 162598, 162600, 162602, 162606, 162828, 162968, 162982, 163056, 163072, 16311
2, 163134, 163386, 163527, 163639, 163645, 163653, 163809, 163925, 163937, 163981, 16398
5, 164179, 164200, 164226, 164280, 164367, 164375, 164540, 164647, 164655, 164707, 16475
3, 164881, 164909, 164917, 165075, 165101, 165103, 165139, 165343, 165347, 165483, 16548
9, 165529, 165549, 165551, 165635, 165639, 165645, 165671, 165843, 165947, 165959, 16596
9, 166015, 166024, 166183, 166203, 166291, 166461, 166492, 166526, 166528, 166534, 16655
8, 166568, 166635, 166643, 166705, 166946, 167018, 167036, 167064, 167296, 167370, 16738
0, 167538, 167570, 167634, 167706, 167732, 167746, 167772, 167790, 167854, 168026, 16809
0, 168144, 168174, 168218, 168248, 168250, 168252, 168254, 168266, 168326, 168350, 16835
8, 168366, 168418, 168456, 168492, 168608, 168612, 168632, 168712, 168846, 169034, 16918
0, 169670, 169904, 169912, 169958, 169982, 169984, 169992, 170289, 170297, 170355, 17035
7, 170399, 170401, 170411, 170551, 170597, 170697, 170705, 170777, 170813, 170817, 17082
7, 170837, 170875, 170897, 170907, 170937, 170939, 170945, 170957, 170993, 171011, 17102
3, 171251, 171495, 171631, 171695, 171701, 171749, 171751, 171759, 171763, 171765, 17181
1, 171867, 171891, 171917, 172013, 172215, 172229, 172233, 172253, 172321, 172461, 17249
7, 172547, 172577, 172583, 172585, 172587, 172589, 172591, 172637, 172705, 172793, 17282
5, 172875, 172881, 172887, 172909, 173145, 173197, 173205, 173209, 173235, 173253, 17325
5, 173291, 173307, 173317, 173351, 173355, 173535, 173619, 173751, 173873, 173925, 17394
1, 173963, 174045, 174053, 174055, 174141, 174403, 174479, 174551, 174681, 174727, 17473
7, 174815, 174909, 175197, 175199, 175293, 175303, 175387, 175397, 175401, 175431, 17543
5, 175475, 175485, 175569, 175577, 175585, 175661, 175693, 175705, 175707, 175743, 17578
1, 176051, 176101, 176329, 176371, 176389, 176413, 176415, 176419, 176423, 176579, 17660
1, 176621, 176751, 176805, 176935, 177185, 177285, 177593, 177615, 177763, 177765, 17793
9, 178061, 178111, 178129, 178323, 178613, 178615, 178827, 179053, 179073, 179119, 17913
3, 179135, 179211, 179401, 179427, 179491, 179511, 179709, 179749, 179813, 179815, 17981
7, 179819, 179953, 180031, 180045, 180095, 180231, 180263, 180265, 180297, 180497, 18077
7, 180985, 180987, 181065, 181139, 181315, 181413, 181659, 181719, 182293, 182297, 18229
9, 182639, 182715, 182727, 182731, 182749, 182793, 182823, 183011, 183197, 183199, 18322
7, 183295, 183301, 183317, 183611, 183635, 183897, 183911, 183959, 184015, 184053, 18424
5, 184253, 184257, 184349, 184471, 184641, 184721, 184791, 184931, 184987, 184997, 18502
9, 185031, 185033, 185135, 185435, 185473, 185585, 186587, 187031, 187541, 187593, 18759
5, 187717, 188189, 188301, 188675, 188751, 188797, 188833, 189043, 189111, 189333, 18938
1, 189547, 189713, 190183, 190207, 190209, 190213, 190215, 190219, 190221, 191005, 19356
5, 193567, 193571, 193573, 193579, 193581, 193583, 193585, 193587, 193609])
```

There are some missing movie ids in the dataset. If the user inputs a value that is not available in the movie ids list, then the program will throw an error. Therefore, we have to handle those exceptions/scenarios appropriately.

Building a Recommender system

using KNN, we find a list of similar movies to the movie the user inputs

In [18]:

```
## Importing the library to calculate the similar movies using KNN
from sklearn.neighbors import NearestNeighbors
```

In [19]:
```
def find_similar_movies(movie_id, X, k, metric='cosine', show_distance=False):

    neighbour_ids = []

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

The below method will return the movie Id based on the word/phrase/movie name the user will input

In [20]:
```
def getMovieId(name):

    # from the original Movies_Data set and use str.contains on the recommendation
    movie_id=0
    uname=name.upper()
    movies = movies_df[(movies_df['title'].str.upper()).str.contains(uname)]

    if len(movies) == 0:
        return movie_id
    else:
        movie_id = movies.iloc[0]['movieId']
        return movie_id
```

This method will bring the first movie id based on the user's input of the movie name. Here I am using the contains method to search for the movie name that contain the word or phrase the user has typed.

Next, based on the user input, the below code will return the recommended list of movies using the knn.

In [22]:
```
## Get the input movie id from the user
print("\nPlease enter a movie name: ")
while True:

    movie_name = str(input())
    movie_id = getMovieId(movie_name)

    if int(movie_id) in movie_mapper.keys():
        movie_title = movie_titles[movie_id]
        print("The movie {} is present in the movie list".format(movie_title))
        similar_ids = find_similar_movies(movie_id, X, k=10)

        print(f"\n\033[1mSince you watched the movie \'{movie_title}\', below are some
```

```
        for i in similar_ids:
            print(movie_titles[i])
        print("\nDo you want to check for other movies (Y/N):")
        user_yn = input()
        if user_yn.upper() == 'Y':
            print("Please enter another movie name: ")
            continue
        else:
            print('OK, See you later, Alligator!')
            break
    else:
        print("The movie is not present in the movie list")
        print("\nDo you want to check for other movies (Y/N):")
        user_yn = input()
        if user_yn.upper() == 'Y':
            print("Please enter another movie name: ")
            continue
        else:
            print('OK, See you later, Alligator!')
            break
```

```
Please enter a movie name:
jumanji
The movie Jumanji (1995) is present in the movie list
```

Since you watched the movie 'Jumanji (1995)', below are some other recommendations

```
Lion King, The (1994)
Mrs. Doubtfire (1993)
Mask, The (1994)
Jurassic Park (1993)
Home Alone (1990)
Nightmare Before Christmas, The (1993)
Aladdin (1992)
Beauty and the Beast (1991)
Ace Ventura: When Nature Calls (1995)
Santa Clause, The (1994)

Do you want to check for other movies (Y/N):
N
OK, See you later, Alligator!
```

The above recommender will ask for a movie title and will search for all the movies that contains that word but will consider the first title that comes up and then displays a list of 10 movies that are recommended based on that title. The user can search for more movies if he/she wishes to.

Below are the references and links to the dataset and code -

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872

Recommendation system in Python - https://www.geeksforgeeks.org/recommendation-system-in-python/

How To Build Your First Recommender System Using Python & MovieLens Dataset - https://analyticsindiamag.com/how-to-build-your-first-recommender-system-using-python-

movielens-dataset/