Movie recommendation POC Project

In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
```

In [2]:
```
movies = pd.read_csv('movies.csv')
ratings = pd.read_csv('ratings.csv')
```

In [3]:
```
movies.head()
```

Out[3]:

	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

In [4]:
```
ratings.head()
```

Out[4]:

	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

Pivot dataset

In [5]:
```
final_data = ratings.pivot(index="movieId", columns="userId", values="rating")
```

In [6]:
```
final_data
```

Out[6]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
193581	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193583	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193585	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193587	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
193609	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

9724 rows × 610 columns

In [7]:
```
final_data.head()
```

Out[7]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	NaN	NaN	NaN	4.0	NaN	4.5	NaN	NaN	NaN	...	4.0	NaN	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	NaN	NaN	NaN	NaN	NaN	4.0	NaN	4.0	NaN	NaN	...	NaN	4.0	NaN	5.0	3.5	NaN	NaN	2.0	NaN	NaN
3	4.0	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2.0	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	3.0	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 610 columns

EDA

Handling missing values

In [8]:
```
final_data.fillna(0, inplace = True)
```

In [9]:
```
final_data.head()
```

Out[9]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 610 columns

Removing noice from dataset

In [10]:
```
no_user_voted = ratings.groupby('movieId')['rating'].agg('count')
```

In [11]:
```
no_movies_voted = ratings.groupby('userId')['rating'].agg('count')
```

In [12]:
```
no_user_voted
```

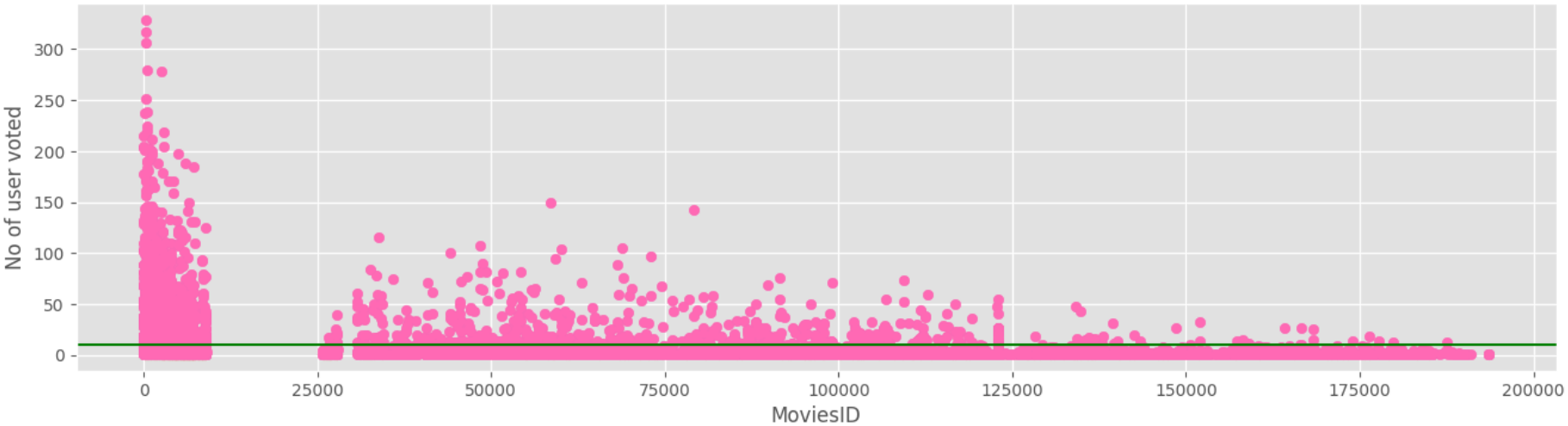
Out[12]:
```
movieId
1         215
2         110
3          52
4           7
5          49
         ...
193581       1
193583       1
193585       1
193587       1
193609       1
Name: rating, Length: 9724, dtype: int64
```

In [13]:
```
no_movies_voted
```

Out[13]:
```
userId
1       232
2        29
3        39
4       216
5        44
       ...
606    1115
607     187
608     831
609      37
610    1302
Name: rating, Length: 610, dtype: int64
```

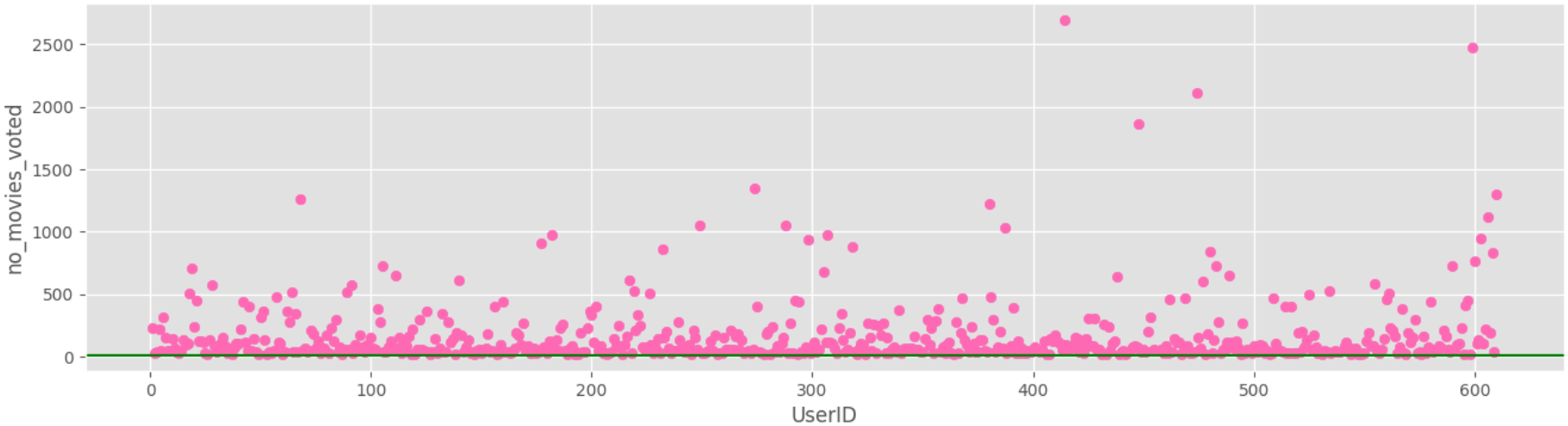
Visuals for ratings from users

In [14]:
```
plt.style.use('ggplot')
fig,axes = plt.subplots(1,1, figsize=(16,4))
plt.scatter(no_user_voted.index, no_user_voted, color='hotpink')
plt.axhline(y=10, color ='green')
plt.xlabel('MoviesID')
plt.ylabel('No of user voted')
plt.show()
```



Visuals for ratings for movies

In [15]:
```
plt.style.use('ggplot')
fig,axes = plt.subplots(1,1, figsize=(16,4))
plt.scatter(no_movies_voted.index, no_movies_voted, color='hotpink')
plt.axhline(y=10, color ='green')
plt.xlabel('UserID')
plt.ylabel('no_movies_voted')
plt.show()
```



In [16]:
```
final_data = final_data.loc[no_user_voted[no_user_voted > 10].index, :]
```

In [17]:
```
final_data
```

Out[17]:

userId	1	2	3	4	5	6	7	8	9	10	...	601	602	603	604	605	606	607	608	609	610
movieId																					
1	4.0	0.0	0.0	0.0	4.0	0.0	4.5	0.0	0.0	0.0	...	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	3.0	5.0
2	0.0	0.0	0.0	0.0	0.0	4.0	0.0	4.0	0.0	0.0	...	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0	0.0
3	4.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	...	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
176371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
187593	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2121 rows × 610 columns

In [18]:
```
final_data = final_data.loc[: , no_movies_voted[no_movies_voted > 50].index]
```

In [19]:
```
final_data.shape
```

Out[19]:
```
(2121, 378)
```

In [20]:
```
final_data
```

Out[20]:

userId	1	4	6	7	10	11	15	16	17	18	...	600	601	602	603	604	605	606	607	608	610
movieId																					
1	4.0	0.0	0.0	4.5	0.0	0.0	2.5	0.0	4.5	3.5	...	2.5	4.0	0.0	4.0	3.0	4.0	2.5	4.0	2.5	5.0
2	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	3.0	...	4.0	0.0	4.0	0.0	5.0	3.5	0.0	0.0	2.0	0.0
3	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	0.0
5	0.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.5	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0
6	4.0	0.0	4.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	...	0.0	0.0	3.0	4.0	3.0	0.0	0.0	0.0	0.0	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
174055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
176371	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
177765	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
179819	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
187593	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

2121 rows × 378 columns

In [21]:
```
2121*378
```

Out[21]:
```
801738
```

Using sparse matrices to store data that contains a large number of zero-valued elements

Convert into sparse matrics

In [22]:
```
from scipy.sparse import csr_matrix
```

In [23]:
```
csr_data = csr_matrix(final_data.values)
final_data.reset_index(inplace = True)
```

In [24]:
```
csr_data
```

Out[24]:
```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 72893 stored elements and shape (2121, 378)>
```

In [25]:
```
print(csr_data)
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 72893 stored elements and shape (2121, 378)>
  Coords        Values
  (0, 0)        4.0
  (0, 3)        4.5
  (0, 6)        2.5
  (0, 8)        4.5
  (0, 9)        3.5
  (0, 10)       4.0
  (0, 12)       3.5
  (0, 16)       3.0
  (0, 19)       3.0
  (0, 20)       3.0
  (0, 25)       5.0
  (0, 28)       5.0
  (0, 29)       4.0
  (0, 31)       3.0
  (0, 34)       5.0
  (0, 38)       5.0
  (0, 39)       4.0
  (0, 40)       4.0
  (0, 41)       2.5
  (0, 43)       4.5
  (0, 46)       0.5
  (0, 47)       4.0
  (0, 50)       2.5
  (0, 53)       4.0
  (0, 55)       3.0
  :     :
  (2118, 205)   4.0
  (2118, 345)   1.5
  (2118, 357)   4.0
  (2118, 369)   4.5
  (2119, 37)    3.5
  (2119, 62)    3.0
  (2119, 98)    0.5
  (2119, 127)   4.5
  (2119, 156)   4.5
  (2119, 236)   0.5
  (2119, 256)   4.5
  (2119, 317)   2.0
```

```
(2119, 345)   2.0
(2119, 357)   5.0
(2119, 365)   3.5
(2120, 37)    4.0
(2120, 62)    5.0
(2120, 146)   2.5
(2120, 155)   4.5
(2120, 156)   5.0
(2120, 186)   5.0
(2120, 205)   4.0
(2120, 236)   3.0
(2120, 317)   3.5
(2120, 357)   4.0
```

In [26]:
```
movies
```

Out[26]:

	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

Model building

In [27]:
```
from sklearn.neighbors import NearestNeighbors

knn = NearestNeighbors(metric = 'cosine',
                       algorithm = 'brute',
                       n_neighbors =20,
                       n_jobs = -1)

knn.fit(csr_data)
```

Out[27]:
```
NearestNeighbors
NearestNeighbors(algorithm='brute', metric='cosine', n_jobs=-1, n_neighbors=20)
```

In [28]:
```
### Operation
```

In [29]:
```
def movie_recommendation(movie_name):
    movie_list = movies[movies['title'].str.contains(movie_name)]
    # print(movie_list)
    if len(movie_list):
        movie_idx = movie_list.iloc[0]['movieId']
        movie_idx = final_data[final_data['movieId'] == movie_idx].index[0]
        distance, indices = knn.kneighbors(csr_data[movie_idx], n_neighbors = 6)
        rec_movie_indices = sorted(list(zip(indices.squeeze().tolist(), distance.squeeze().tolist())), key=lambda x: x[1])[:0:
        recommended_movies = []
        for val in rec_movie_indices:
            movie_idx = final_data.iloc[val[0]]['movieId']
            idx = movies[movies['movieId'] == movie_idx].index
            recommended_movies.append({'Title':movies.iloc[idx]['title'].values[0], 'Distance': val[1]})
        df = pd.DataFrame(recommended_movies, index = range(1,6))
        return df
    else:
        return 'Movie not found...'
```

In [30]:
```
movie_recommendation('Iron Man')
```

Out[30]:

	Title	Distance
1	Avatar (2009)	0.310893
2	Iron Man 2 (2010)	0.307492
3	WALL·E (2008)	0.298138
4	Dark Knight, The (2008)	0.285835
5	Avengers, The (2012)	0.285319

In [31]:
```
movie_recommendation('Avatar')
```

Out[31]:

	Title	Distance
1	Kung Fu Panda (2008)	0.358604
2	Iron Man (2008)	0.310893
3	District 9 (2009)	0.309947
4	WALL·E (2008)	0.306969
5	Up (2009)	0.289607

In [32]:
```
import gradio as gr

def recommendation_system(movie_name):
    df = movie_recommendation(movie_name)
    if isinstance(df, pd.DataFrame):
        return df.to_string(index = False)
    else:
        return df

app = gr.Interface(
    fn = recommendation_system,
    inputs = "text",
    outputs = "text",
    title = "Movie Recommendation System",
```

```
    description = "Enter your movie",
    )

app.launch(share=True)
```

```
* Running on local URL:  http://127.0.0.1:7861

Could not create share link. Please check your internet connection or our status page: https://status.gradio.app.
```

Movie Recommendation System

Enter your movie

movie_name

output

Clear Submit **Flag**

Use via API · Built with Gradio

Out[32]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: