MovieLens Case Study

Background of Problem Statement :
The GroupLens Research Project is a research group in the Department of Computer Science and Engineering at the University of Minnesota. Members of the GroupLens Research Project are involved in many research projects related to the fields of information filtering, collaborative filtering, and recommender systems. The project is led by professors John Riedl and Joseph Konstan. The project began to explore automated collaborative filtering in 1992 but is most well known for its worldwide trial of an automated collaborative filtering system for Usenet news in 1996. Since then the project has expanded its scope to research overall information by filtering solutions, integrating into content-based methods, as well as, improving current collaborative filtering technology.

Problem Objective :
Here, we ask you to perform the analysis using the Exploratory Data Analysis technique. You need to find features affecting the ratings of any particular movie and build a model to predict the movie ratings.

Domain: Entertainment

Analysis Tasks to be performed:

- Import the three datasets
- Create a new dataset [Master_Data] with the following columns MovieID Title UserID Age Gender Occupation Rating. (Hint: (i) Merge two tables at a time. (ii) Merge the tables using two primary keys MovieID & UserId)
- Explore the datasets using visual representations (graphs or tables), also include your comments on the following:
    1. User Age Distribution
    2. User rating of the movie “Toy Story”
    3. Top 25 movies by viewership rating
    4. Find the ratings for all the movies reviewed by for a particular user of user id = 2696

- Feature Engineering:

  Use column genres:
    1. Find out all the unique genres (Hint: split the data in column genre making a list and then process the data to find out only the unique categories of genres)
    2. Create a separate column for each genre category with a one-hot encoding (1 and 0) whether or not the movie belongs to that genre.
    3. Determine the features affecting the ratings of any particular movie.
    4. Develop an appropriate model to predict the movie ratings

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
%matplotlib inline
```

Import Data Sets

In [2]:
```
movie_data = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Python/Assessment/MovieLens Data Analysis/Data Set/movies.dat",sep='::', names =['MovieID','Title','Genres'])
movie_data.head()
```

```
C:\Users\VAIO\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex separators (se
parators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Out[2]:

	MovieID	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

In [3]:
```
ratings_data = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Python/Assessment/MovieLens Data Analysis/Data Set/ratings.dat",sep='::', names =['UserID','MovieID','Rating','Tim
estamp'])
ratings_data.head()
```

```
C:\Users\VAIO\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex separators (se
parators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Out[3]:

	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

In [4]:
```
users_data = pd.read_csv("C:/Users/VAIO/Downloads/SimpliLearn/Python/Assessment/MovieLens Data Analysis/Data Set/users.dat",sep='::', names =['UserID','Gender','Age','Occupation'
,'Zip-code'])
users_data.head()
```

```
C:\Users\VAIO\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex separators (se
parators > 1 char and different from '\s+' are interpreted as regex); you can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Out[4]:

	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	02460
4	5	M	25	20	55455

Merge the data sets to Master_Data

In [5]:
```
merged_data = pd.merge(movie_data,ratings_data)
merged_data
```

Out[5]:

	MovieID	Title	Genres	UserID	Rating	Timestamp
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474
...	...	...	...	...	...	...
1000204	3952	Contender, The (2000)	Drama|Thriller	5812	4	992072099
1000205	3952	Contender, The (2000)	Drama|Thriller	5831	3	986223125
1000206	3952	Contender, The (2000)	Drama|Thriller	5837	4	1011902656
1000207	3952	Contender, The (2000)	Drama|Thriller	5927	1	979852537
1000208	3952	Contender, The (2000)	Drama|Thriller	5998	4	1001781044

1000209 rows × 6 columns

In [6]:
```
Meta_Data = pd.merge(merged_data,users_data)
Meta_Data.head()
```

Out[6]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10	48067
1	48	Pocahontas (1995)	Animation|Children's|Musical|Romance	1	5	978824351	F	1	10	48067
2	150	Apollo 13 (1995)	Drama	1	5	978301777	F	1	10	48067
3	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi	1	4	978300760	F	1	10	48067
4	527	Schindler's List (1993)	Drama|War	1	5	978824195	F	1	10	48067

In [7]:
```
Master_Data = Meta_Data.drop(['Genres','Timestamp','Zip-code'], axis=1)
Master_Data.head()
```

Out[7]:

	MovieID	Title	UserID	Rating	Gender	Age	Occupation
0	1	Toy Story (1995)	1	5	F	1	10
1	48	Pocahontas (1995)	1	5	F	1	10
2	150	Apollo 13 (1995)	1	5	F	1	10
3	260	Star Wars: Episode IV - A New Hope (1977)	1	4	F	1	10
4	527	Schindler's List (1993)	1	5	F	1	10

In [8]:
```
Master_Data.isnull().any()
```

Out[8]:
```
MovieID       False
Title         False
UserID        False
Rating        False
Gender        False
Age           False
Occupation    False
dtype: bool
```

User Age Distribution

In [9]:
```
bins= [1,18,25,35,45,50,56,57]
labels = ['Under 18','18-24','25-34','35-44','45-49','50-55',"Above 56"]
Master_Data['Age_Updated'] = pd.cut(Master_Data['Age'], bins=bins, labels=labels, right=False)
Master_Data.tail()
```

Out[9]:

	MovieID	Title	UserID	Rating	Gender	Age	Occupation	Age_Updated
1000204	3513	Rules of Engagement (2000)	5727	4	M	25	4	25-34
1000205	3535	American Psycho (2000)	5727	2	M	25	4	25-34
1000206	3536	Keeping the Faith (2000)	5727	5	M	25	4	25-34
1000207	3555	U-571 (2000)	5727	3	M	25	4	25-34
1000208	3578	Gladiator (2000)	5727	5	M	25	4	25-34

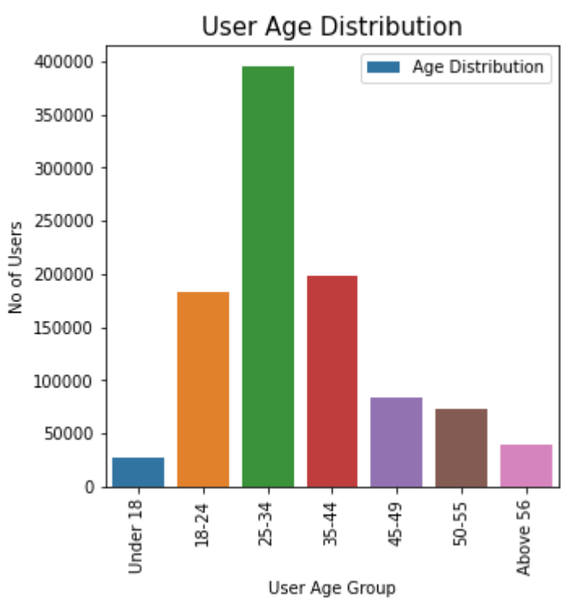
In [10]:
```
Age_Distribution = Master_Data['Age_Updated'].value_counts().to_frame().sort_index(ascending=True)
Age_Distribution
```

Out[10]:

	Age_Updated
Under 18	27211
18-24	183536
25-34	395556
35-44	199003
45-49	83633
50-55	72490
Above 56	38780

In [11]:
```
# Visualize the User Age Distribution
plt.figure(figsize=(5,5))
ax = sns.barplot(Age_Distribution.index,Age_Distribution['Age_Updated'],label='Age Distribution')
plt.tick_params(axis="x",labelrotation=90, labelsize = 10)
plt.xlabel("User Age Group", fontdict = {'fontsize' : 10})
plt.ylabel("No of Users", fontdict = {'fontsize' : 10})
plt.title("User Age Distribution", fontdict = {'fontsize' : 15})
plt.legend()
```

Out[11]: <matplotlib.legend.Legend at 0x2685c3a4348>



User rating of the movie “Toy Story”

In [12]:
```
ToyStory = Master_Data[Master_Data['MovieID'] == 1]
ToyStory.drop(['MovieID','UserID','Gender','Occupation'], axis=1)
```

Out[12]:

	Title	Rating	Age	Age_Updated
0	Toy Story (1995)	5	1	Under 18
53	Toy Story (1995)	4	50	50-55
124	Toy Story (1995)	4	25	25-34
263	Toy Story (1995)	5	25	25-34
369	Toy Story (1995)	5	35	35-44
...	...	...	...	...
575166	Toy Story (1995)	5	25	25-34
575214	Toy Story (1995)	5	25	25-34
575485	Toy Story (1995)	4	45	45-49
575589	Toy Story (1995)	4	25	25-34
575869	Toy Story (1995)	3	25	25-34

2077 rows × 4 columns

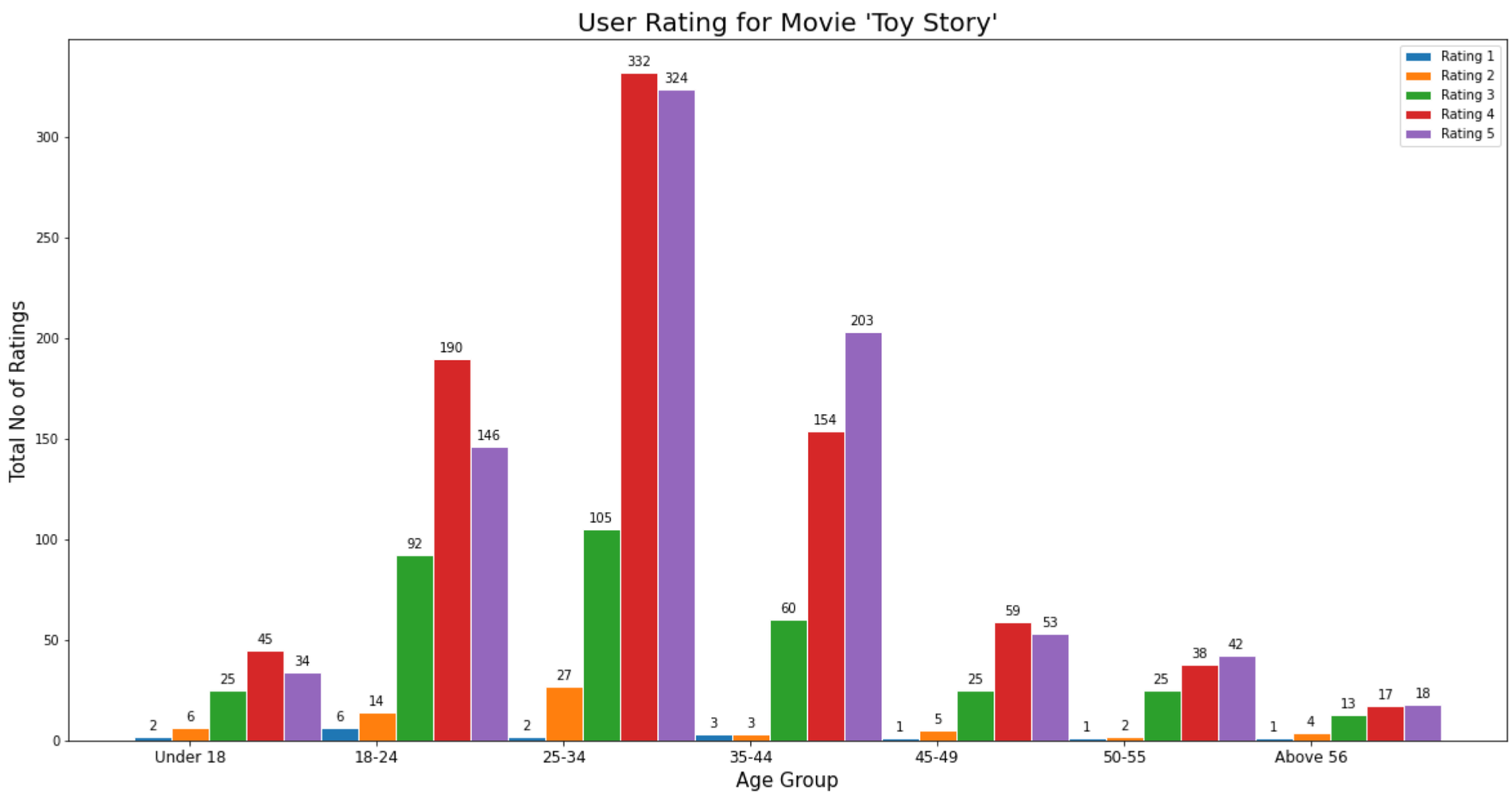
In [13]:
```
grouped_data = ToyStory.groupby(['Age_Updated','Rating'])
grouped_data = grouped_data.size().unstack()
```

In [14]:
```
grouped_data.rename(columns = {1:'One',2:'Two',3:'Three',4:'Four',5:'Five'}, inplace=True)
grouped_data
```

Out[14]:

Rating	One	Two	Three	Four	Five
Age_Updated					
Under 18	2	6	25	45	34
18-24	6	14	92	190	146
25-34	2	27	105	332	324
35-44	3	3	60	154	203
45-49	1	5	25	59	53
50-55	1	2	25	38	42
Above 56	1	4	13	17	18

In [15]:
```
# Define the position for the subsequent bar graph to be displayed
barWidth = 0.20
r1 = np.arange(len(grouped_data.One))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]
r5 = [x + barWidth for x in r4]


# Visualize Ratings provided by age group for Toy Story
fig, ax = plt.subplots(figsize=(20,10))
bar1 = plt.bar(r1, grouped_data.One, width=barWidth, edgecolor='white', label='Rating 1')
bar2 = plt.bar(r2, grouped_data.Two, width=barWidth, edgecolor='white', label='Rating 2')
bar3 = plt.bar(r3, grouped_data.Three, width=barWidth, edgecolor='white', label='Rating 3')
bar4 = plt.bar(r4, grouped_data.Four, width=barWidth, edgecolor='white', label='Rating 4')
bar5 = plt.bar(r5, grouped_data.Five, width=barWidth, edgecolor='white', label='Rating 5')

# Add xticks on the middle of the group bars
plt.xticks([r + barWidth for r in range(len(grouped_data.One))], grouped_data.index)
plt.tick_params(axis="x", labelsize = 12)
plt.xlabel("Age Group", fontdict = {'fontsize' : 15})
plt.ylabel("Total No of Ratings", fontdict = {'fontsize' : 15})
plt.title("User Rating for Movie 'Toy Story'", fontdict = {'fontsize' : 20})

def autolabel(rects):
    """Attach a text label above each bar in *rects*, displaying its height."""
    for rect in rects:
        height = rect.get_height()
        ax.annotate('{}'.format(height),
                    xy=(rect.get_x() + rect.get_width() / 2, height),
                    xytext=(0, 3),  # 3 points vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')


autolabel(bar1)
autolabel(bar2)
autolabel(bar3)
autolabel(bar4)
autolabel(bar5)

plt.legend()
plt.show()
```



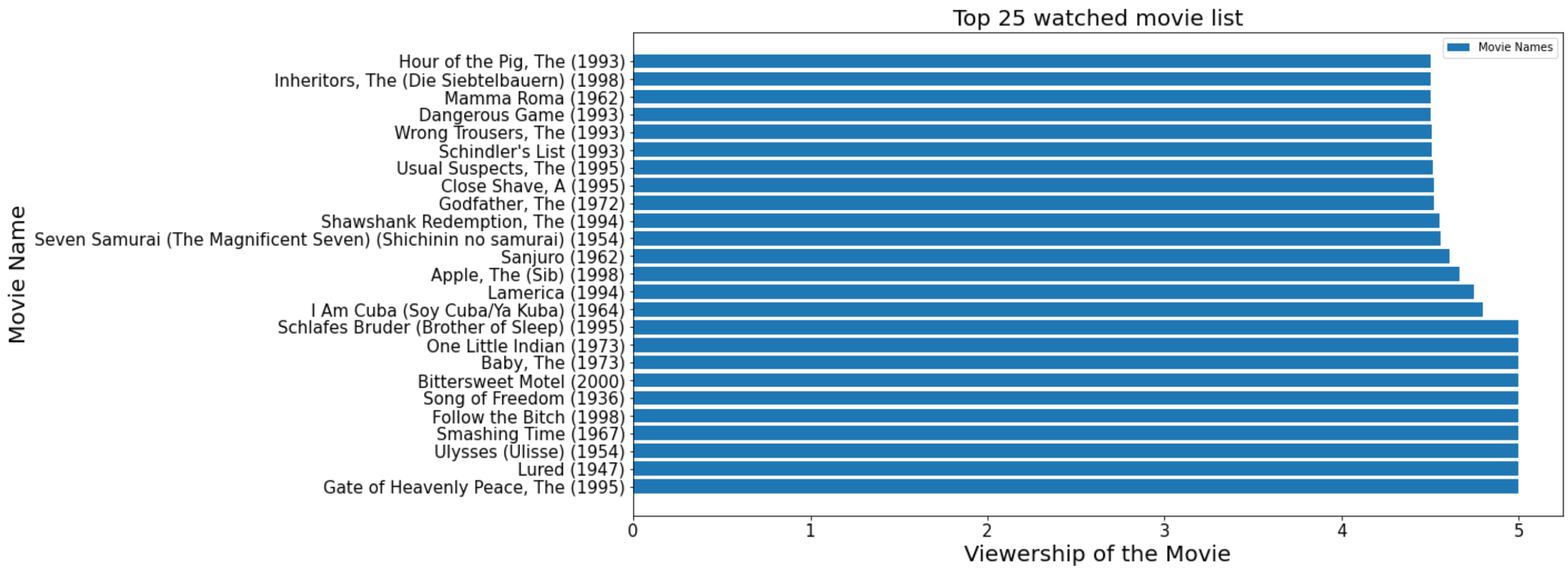
Top 25 movies by viewership rating

In [16]:
```
Top_Viewed_Movie = Master_Data.groupby("Title")["Rating"].agg("mean").sort_values(ascending=False).to_frame(name = 'Rate')
Top_Viewed_Movie = Top_Viewed_Movie.head(25)
Top_Viewed_Movie.head(10)
```

Out[16]:

Title	Rate
Gate of Heavenly Peace, The (1995)	5.0
Lured (1947)	5.0
Ulysses (Ulisse) (1954)	5.0
Smashing Time (1967)	5.0
Follow the Bitch (1998)	5.0
Song of Freedom (1936)	5.0
Bittersweet Motel (2000)	5.0
Baby, The (1973)	5.0
One Little Indian (1973)	5.0
Schlafes Bruder (Brother of Sleep) (1995)	5.0

In [17]:
```
# Visualize the Top 25 watched Movie
plt.figure(figsize=(15,8))
plt.barh(Top_Viewed_Movie.index,Top_Viewed_Movie.Rate,label='Movie Names')
plt.tick_params(axis="y", labelsize = 15)
plt.tick_params(axis="x", labelsize = 15)
plt.xlabel("Viewership of the Movie", fontdict = {'fontsize' : 20})
plt.ylabel("Movie Name", fontdict = {'fontsize' : 20})
plt.title("Top 25 watched movie list", fontdict = {'fontsize' : 20})
plt.legend()
plt.show()
```



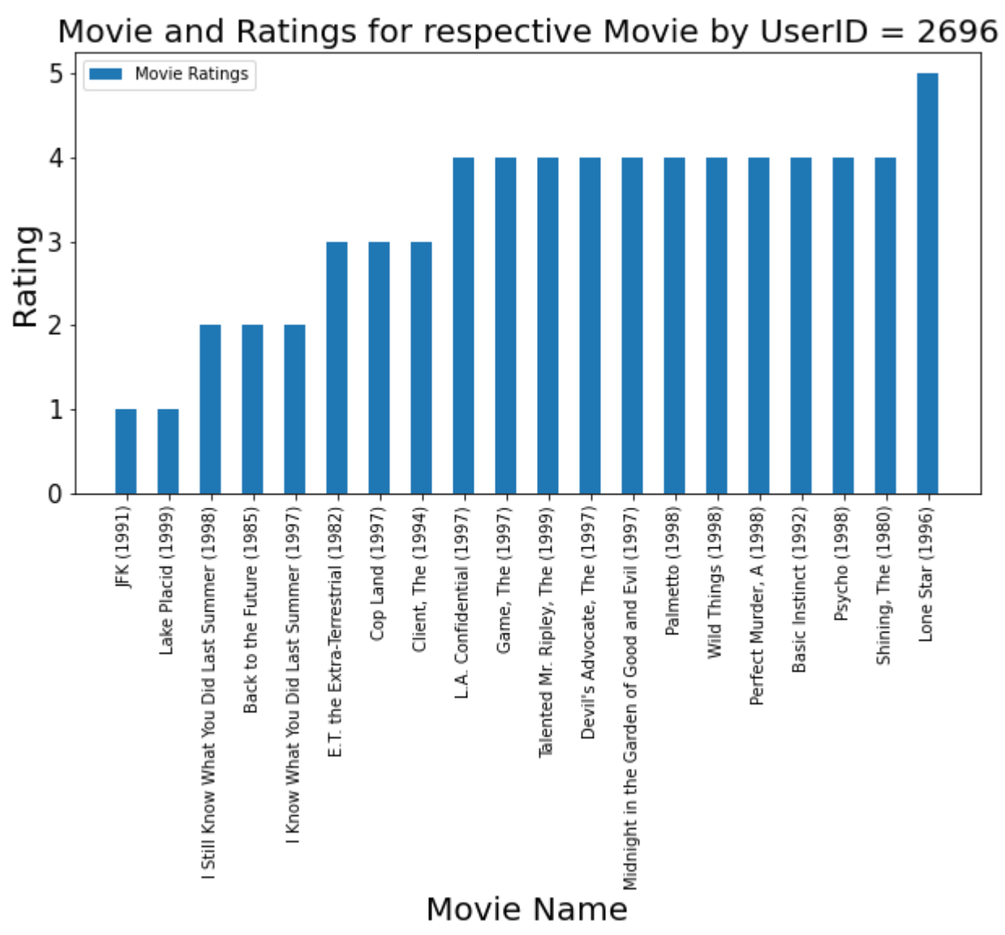
Ratings for all the movies reviewed by User ID=2696

In [18]:
```
User_ID_Movie = Master_Data[Master_Data['UserID'] == 2696]
User_ID_Movie = User_ID_Movie.sort_values(by='Rating')
User_ID_Movie
```

Out[18]:

	MovieID	Title	UserID	Rating	Gender	Age	Occupation	Age_Updated
991054	3386	JFK (1991)	2696	1	M	25	7	25-34
991052	2713	Lake Placid (1999)	2696	1	M	25	7	25-34
991050	2338	I Still Know What You Did Last Summer (1998)	2696	2	M	25	7	25-34
991040	1270	Back to the Future (1985)	2696	2	M	25	7	25-34
991044	1644	I Know What You Did Last Summer (1997)	2696	2	M	25	7	25-34
991038	1097	E.T. the Extra-Terrestrial (1982)	2696	3	M	25	7	25-34
991041	1589	Cop Land (1997)	2696	3	M	25	7	25-34
991035	350	Client, The (1994)	2696	3	M	25	7	25-34
991042	1617	L.A. Confidential (1997)	2696	4	M	25	7	25-34
991043	1625	Game, The (1997)	2696	4	M	25	7	25-34
991053	3176	Talented Mr. Ripley, The (1999)	2696	4	M	25	7	25-34
991045	1645	Devil's Advocate, The (1997)	2696	4	M	25	7	25-34
991046	1711	Midnight in the Garden of Good and Evil (1997)	2696	4	M	25	7	25-34
991047	1783	Palmetto (1998)	2696	4	M	25	7	25-34
991048	1805	Wild Things (1998)	2696	4	M	25	7	25-34
991049	1892	Perfect Murder, A (1998)	2696	4	M	25	7	25-34
991037	1092	Basic Instinct (1992)	2696	4	M	25	7	25-34
991051	2389	Psycho (1998)	2696	4	M	25	7	25-34
991039	1258	Shining, The (1980)	2696	4	M	25	7	25-34
991036	800	Lone Star (1996)	2696	5	M	25	7	25-34

In [19]:
```
# Visualize the movie and rating provided for the movie by UserID=2696
plt.figure(figsize=(10,5))
plt.bar(User_ID_Movie.Title,User_ID_Movie.Rating,label='Movie Ratings', width = 0.5)
plt.tick_params(axis="y", labelsize = 15)
plt.tick_params(axis="x",labelrotation=90, labelsize = 10)
plt.xlabel("Movie Name", fontdict = {'fontsize' : 20})
plt.ylabel("Rating", fontdict = {'fontsize' : 20})
plt.title("Movie and Ratings for respective Movie by UserID = 2696", fontdict = {'fontsize' : 20})
plt.legend()
plt.show()
```



FEATURE ENGINEERING - GENRE

In [20]:
```
Genre_Data = Meta_Data
Genre_Data.head()
```

Out[20]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10	48067
1	48	Pocahontas (1995)	Animation|Children's|Musical|Romance	1	5	978824351	F	1	10	48067
2	150	Apollo 13 (1995)	Drama	1	5	978301777	F	1	10	48067
3	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi	1	4	978300760	F	1	10	48067
4	527	Schindler's List (1993)	Drama|War	1	5	978824195	F	1	10	48067

In [21]:
```
Genre_Data.isnull().any()
```

Out[21]:
```
MovieID       False
Title         False
Genres        False
UserID        False
Rating        False
Timestamp     False
Gender        False
Age           False
Occupation    False
Zip-code      False
dtype: bool
```

In [22]:
```
Genre_Data['Genres'] = Genre_Data['Genres'].str.split("|", expand = False)
Genre_Data.head()
```

Out[22]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	Toy Story (1995)	[Animation, Children's, Comedy]	1	5	978824268	F	1	10	48067
1	48	Pocahontas (1995)	[Animation, Children's, Musical, Romance]	1	5	978824351	F	1	10	48067
2	150	Apollo 13 (1995)	[Drama]	1	5	978301777	F	1	10	48067
3	260	Star Wars: Episode IV - A New Hope (1977)	[Action, Adventure, Fantasy, Sci-Fi]	1	4	978300760	F	1	10	48067
4	527	Schindler's List (1993)	[Drama, War]	1	5	978824195	F	1	10	48067

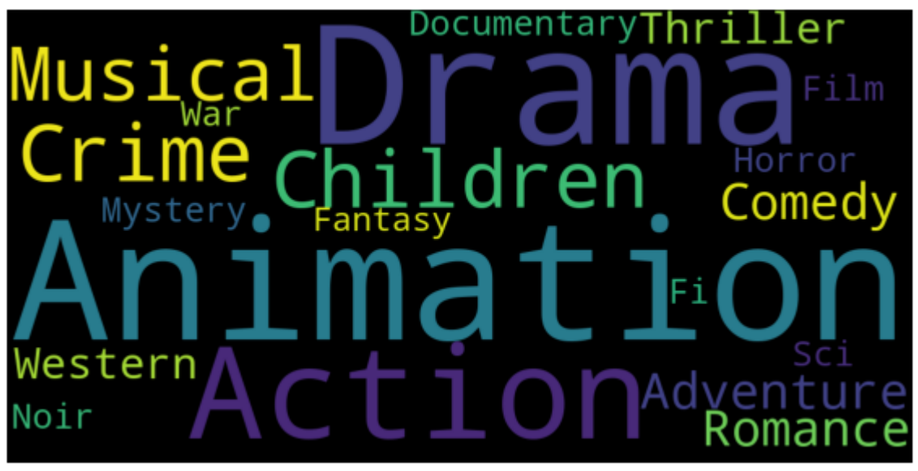
In [23]:
```
unique_list = []

# function to get unique values
def unique_value(list1):
    for x in list1:
        if x not in unique_list:
            unique_list.append(x)
        return unique_list

Genre_Data['Genres'].apply(unique_value)
```

Out[23]:
```
0          [Animation, Drama, Action, Children's, Crime, ...
1          [Animation, Drama, Action, Children's, Crime, ...
2          [Animation, Drama, Action, Children's, Crime, ...
3          [Animation, Drama, Action, Children's, Crime, ...
4          [Animation, Drama, Action, Children's, Crime, ...
                                 ...
1000204    [Animation, Drama, Action, Children's, Crime, ...
1000205    [Animation, Drama, Action, Children's, Crime, ...
1000206    [Animation, Drama, Action, Children's, Crime, ...
1000207    [Animation, Drama, Action, Children's, Crime, ...
1000208    [Animation, Drama, Action, Children's, Crime, ...
Name: Genres, Length: 1000209, dtype: object
```

In [24]:
```
print(unique_list, "\n")
print("Total no of genre available are: ",len(unique_list))
```

```
['Animation', 'Drama', 'Action', "Children's", 'Crime', 'Musical', 'Adventure', 'Comedy', 'Romance', 'Thriller', 'Western', 'Documentary', 'Sci-Fi', 'Horror', 'Film-Noir', 'Mystery', 'War', 'Fantasy']

Total no of genre available are:  18
```

In [25]:
```
unique_words=(" ").join(unique_list)
wordcloud = WordCloud(width = 1000, height = 500, max_words=100).generate(unique_words)
plt.figure(figsize=(10,10))
plt.imshow(wordcloud)
plt.axis("off")
plt.show()
```



In [26]:
```
Genre_Data
```

Out[26]:

MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	Toy Story (1995)	[Animation, Children's, Comedy]	1	5	978824268	F	1	10	48067
1	48	Pocahontas (1995)	[Animation, Children's, Musical, Romance]	1	5	978824351	F	1	10	48067
2	150	Apollo 13 (1995)	[Drama]	1	5	978301777	F	1	10	48067
3	260	Star Wars: Episode IV - A New Hope (1977)	[Action, Adventure, Fantasy, Sci-Fi]	1	4	978300760	F	1	10	48067
4	527	Schindler's List (1993)	[Drama, War]	1	5	978824195	F	1	10	48067
...	...	...	...	...	...	...	...	...	...	...
1000204	3513	Rules of Engagement (2000)	[Drama, Thriller]	5727	4	958489970	M	25	4	92843
1000205	3535	American Psycho (2000)	[Comedy, Horror, Thriller]	5727	2	958489970	M	25	4	92843
1000206	3536	Keeping the Faith (2000)	[Comedy, Romance]	5727	5	958489902	M	25	4	92843
1000207	3555	U-571 (2000)	[Action, Thriller]	5727	3	958490699	M	25	4	92843
1000208	3578	Gladiator (2000)	[Action, Drama]	5727	5	958490171	M	25	4	92843

1000209 rows × 10 columns

In [27]:
```
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()
encoded = pd.DataFrame(mlb.fit_transform(Genre_Data.Genres),
                  columns=mlb.classes_,
                  index=Genre_Data.index)
```

In [28]:
```
encoded
```

Out[28]:

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0
2	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0
4	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1000204	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0
1000205	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0
1000206	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
1000207	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1000208	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0

1000209 rows × 18 columns

In [29]:
```
Movie_Details_Table = pd.merge(Master_Data, encoded, left_index=True, right_index=True)
Movie_Details_Table = Movie_Details_Table.drop(["MovieID",'Title','UserID','Age_Updated'], axis=1)
Movie_Details_Table.head()
```

Out[29]:

	Rating	Gender	Age	Occupation	Action	Adventure	Animation	Children's	Comedy	Crime	...	Fantasy	Film-Noir	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western
0	5	F	1	10	0	0	1	1	1	0	...	0	0	0	0	0	0	0	0	0	0
1	5	F	1	10	0	0	1	1	0	0	...	0	0	0	1	0	1	0	0	0	0
2	5	F	1	10	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	4	F	1	10	1	1	0	0	0	0	...	1	0	0	0	0	0	1	0	0	0
4	5	F	1	10	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0

5 rows × 22 columns

In [30]:
```
age_encoded = pd.DataFrame(mlb.fit_transform(Movie_Details_Table.Gender),
                  columns=mlb.classes_,
                  index=Movie_Details_Table.index)
```

In [31]:
```
age_encoded.head()
```

Out[31]:

	F	M
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

In [32]:
```
Movie_Details = pd.merge(Movie_Details_Table, age_encoded, left_index=True, right_index=True)
Movie_Details.drop('Gender', axis = 1, inplace = True)
Movie_Details.head()
```

Out[32]:

	Rating	Age	Occupation	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	...	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western	F	M
0	5	1	10	0	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	1	0
1	5	1	10	0	0	1	1	0	0	0	...	0	1	0	1	0	0	0	0	1	0
2	5	1	10	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	1	0
3	4	1	10	1	1	0	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
4	5	1	10	0	0	0	0	0	0	0	...	0	0	0	0	0	0	1	0	1	0

5 rows × 23 columns

Build a model and perform prediction

In [33]:
```
X = Movie_Details.iloc[:, 1:]
Y = Movie_Details['Rating']
```

In [34]:
```
print(X.shape)
X.head()
```

```
(1000209, 22)
```

Out[34]:

	Age	Occupation	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	...	Horror	Musical	Mystery	Romance	Sci-Fi	Thriller	War	Western	F	M
0	1	10	0	0	1	1	1	0	0	0	...	0	0	0	0	0	0	0	0	1	0
1	1	10	0	0	1	1	0	0	0	0	...	0	1	0	1	0	0	0	0	1	0
2	1	10	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	1	0
3	1	10	1	1	0	0	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
4	1	10	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	1	0	1	0

5 rows × 22 columns

In [35]:
```
print(Y.shape)
Y.head()
```

```
(1000209,)
```

Out[35]:
```
0    5
1    5
2    5
3    4
4    5
Name: Rating, dtype: int64
```

In [36]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, Y)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[36]:
```
((750156, 22), (250053, 22), (750156,), (250053,))
```

LINEAR REGRESSION

In [37]:
```
from sklearn.linear_model import LinearRegression
classifier = LinearRegression()
classifier.fit(x_train, y_train)
```

Out[37]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [38]:
```
print(classifier.intercept_)
```

```
3.382246278202003
```

In [39]:
```
feature_wise_coeff = list(zip(x_train.columns,classifier.coef_))
feature_wise_coeff
```

Out[39]:
```
[('Age', 0.004024935406629382),
 ('Occupation', 0.0011759898640020527),
 ('Action', -0.09831115803438545),
 ('Adventure', 0.00874034338113728),
 ('Animation', 0.3719453156736318),
 ("Children's", -0.3254720870824206),
 ('Comedy', -0.014305072174852072),
 ('Crime', 0.1007199305264274),
 ('Documentary', 0.42633666341834625),
 ('Drama', 0.23082157610165147),
 ('Fantasy', 0.06967036093614526),
 ('Film-Noir', 0.4339351635406079),
 ('Horror', -0.291197130840607),
 ('Musical', 0.16241435010004523),
 ('Mystery', 0.010078956616775823),
 ('Romance', -0.010384775916202812),
 ('Sci-Fi', -0.024862859990976222),
 ('Thriller', 0.05677222473383271),
 ('War', 0.2944761170432372),
 ('Western', 0.105746231958863),
 ('F', 0.017626765912803877),
 ('M', -0.01762676591280385)]
```

In [40]:
```
y_pred = classifier.predict(x_test)
y_predicted = pd.DataFrame(y_pred, columns=['Predicted'])
y_predicted.head()
```

Out[40]:

	Predicted
0	3.733025
1	3.422763
2	3.834293
3	3.511179
4	3.727896

In [41]:
```
Observed_Predicted =pd.concat([y_test.reset_index(drop=True),y_predicted],axis=1)
Observed_Predicted.head()
```

Out[41]:

	Rating	Predicted
0	3	3.733025
1	3	3.422763
2	4	3.834293
3	4	3.511179
4	2	3.727896

In [42]:
```
#Finding Root Mean Squared Error
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(y_test,y_predicted))
rmse
```

Out[42]: 1.0949260496462583

DECISION TREE

In [43]:
```
from sklearn.tree import DecisionTreeClassifier
decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
```

Out[43]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

In [44]:
```
y_pred_dt = decision_tree.predict(x_test)
y_predicted_dt = pd.DataFrame(y_pred_dt, columns=['Predicted'])
y_predicted_dt.head()
```

Out[44]:

	Predicted
0	4
1	4
2	4
3	4
4	5

In [45]:
```
Observed_Predicted_DT =pd.concat([y_test.reset_index(drop=True),y_predicted_dt],axis=1)
Observed_Predicted_DT.head()
```

Out[45]:

	Rating	Predicted
0	3	4
1	3	4
2	4	4
3	4	4
4	2	5

In [46]:
```
acc_decision_tree = round(decision_tree.score(x_train, y_train) * 100, 2)
acc_decision_tree
```

Out[46]: 41.34

Inference: Among the two models used to predict the movie rating, Linear regression gives the better prediction