Project 02: Movielens Case Study

DESCRIPTION

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:
  - User Age Distribution
  - User rating of the movie "Toy Story"
  - Top 25 movies by viewership rating
  - Find the ratings for all the movies reviewed by for a particular user of user id = 2696
- Feature Engineering:

  Use column genres:

  - 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)
  - 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.
  - Determine the features affecting the ratings of any particular movie.
  - Develop an appropriate model to predict the movie ratings

Table of Contents

Import Libraries and the datasets

Import Libraries

In [1]:
```
#Importing all the necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from matplotlib.style import use

import warnings
warnings.filterwarnings('ignore')
```

Import Datasets

In [2]:
```
#loading the datasets
ratings_df = pd.read_csv("D:\\NIPUN_SC_REC\\3_Practice_Project\\Course_5_Data Science with Python\\Practice_pro
                         sep="::", header=None, names =['UserID','MovieID','Rating','Timestamp'])

users_df = pd.read_csv("D:\\NIPUN_SC_REC\\3_Practice_Project\\Course_5_Data Science with Python\\Practice_proje
                       sep="::", header=None, names =['UserID','Gender','Age','Occupation','Zip-code'])

movies_df = pd.read_csv("D:\\NIPUN_SC_REC\\3_Practice_Project\\Course_5_Data Science with Python\\Practice_proj
                        sep="::", header=None, names =['MovieID','Title','Genres'])
```

In [3]:
```
#View the first five observations in ratings dataframe
ratings_df.head()
```

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]:
```
#overview of the ratings dataframe
ratings_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000209 entries, 0 to 1000208
Data columns (total 4 columns):
 #   Column     Non-Null Count    Dtype
---  ------     --------------    -----
 0   UserID     1000209 non-null  int64
 1   MovieID    1000209 non-null  int64
 2   Rating     1000209 non-null  int64
 3   Timestamp  1000209 non-null  int64
dtypes: int64(4)
memory usage: 30.5 MB
```

In [5]:
```
#check the dimensionality of the DataFrame/(No.of rows and columns)
ratings_df.shape
```

Out[5]:
```
(1000209, 4)
```

In [6]:
```
#Check any empty or null values in columns
ratings_df.isnull().sum()
```

Out[6]:
```
UserID       0
MovieID      0
Rating       0
Timestamp    0
dtype: int64
```

The rating dataset has no empty or null columns.

In [7]:
```
#View the first five observations in users dataframe
users_df.head()
```

Out[7]:

	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

In [8]:
```
#overview of the users dataframe
users_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6040 entries, 0 to 6039
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   UserID      6040 non-null   int64
 1   Gender      6040 non-null   object
 2   Age         6040 non-null   int64
 3   Occupation  6040 non-null   int64
 4   Zip-code    6040 non-null   object
dtypes: int64(3), object(2)
memory usage: 188.8+ KB
```

In [9]:
```
#dimensionality of the DataFrame/(No.of rows and columns)
users_df.shape
```

Out[9]:
```
(6040, 5)
```

In [10]:
```
#Check any empty or null values in columns
users_df.isnull().sum()
```

Out[10]:
```
UserID        0
Gender        0
Age           0
Occupation    0
Zip-code      0
dtype: int64
```

The users dataset has no empty or null columns.

In [11]:
```
#View the first five observations in movies dataframe
movies_df.head()
```

Out[11]:

	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 [12]:
```
#overview of the movies dataframe
movies_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   MovieID  3883 non-null   int64
 1   Title    3883 non-null   object
 2   Genres   3883 non-null   object
dtypes: int64(1), object(2)
memory usage: 60.7+ KB
```

In [13]:
```
#check the dimensionality of the DataFrame/(No.of rows and columns)
movies_df.shape
```

Out[13]:
```
(3883, 3)
```

In [14]:
```
#Check any empty or null values in columns
movies_df.isnull().sum()
```

Out[14]:
```
MovieID    0
Title      0
Genres     0
dtype: int64
```

The movies dataset has no empty or null columns.

Create a new dataset [Master_Data]:

Create a new dataset [Master_Data] with the following columns MovieID, Title, UserID, Age, Gender and Occupation Rating.

(Hint: (i) Merge two tables at a time. (ii) Merge the tables using two primary keys MovieID & UserId)

In [15]:
```
#Merge the movies_df and ratings_df dataframes based on the key 'MovieID'.
Master_Data_df = pd.merge(movies_df, ratings_df, on = 'MovieID')
#Merge the Master_Data_df and users_df dataframes based on the key 'UserID'.
Master_Data_df = pd.merge(Master_Data_df, users_df, on = 'UserID')

#View the first five observations in the Master_Data_df dataframe.
Master_Data_df.head()
```

Out[15]:

	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 [16]:
```
#View the last five observations in the Master_Data_df dataframe.
Master_Data_df.tail()
```

Out[16]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
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

In [17]:
```
#View the overview of the Master_Data_df dataframe.
Master_Data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 10 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   MovieID     1000209 non-null  int64
 1   Title       1000209 non-null  object
 2   Genres      1000209 non-null  object
 3   UserID      1000209 non-null  int64
 4   Rating      1000209 non-null  int64
 5   Timestamp   1000209 non-null  int64
 6   Gender      1000209 non-null  object
 7   Age         1000209 non-null  int64
 8   Occupation  1000209 non-null  int64
 9   Zip-code    1000209 non-null  object
dtypes: int64(6), object(4)
memory usage: 68.7+ MB
```

In [18]:
```
#Number of movies in the Dataset.
len(Master_Data_df["Title"].explode().unique())
```

Out[18]:
```
3706
```

There are '3706' unique Movies in the dataset.

In [19]:
```
#Number of movies in the Dataset.
len(Master_Data_df["UserID"].explode().unique())
```

Out[19]:
```
6040
```

There are '6040' unique UserIDs in the dataset.

In [20]:
```
#View the type of the Master_Data_df dataframe.
type(Master_Data_df)
```

Out[20]:
```
pandas.core.frame.DataFrame
```

In [21]:
```
Master_Data_df.describe(include='all')
```

Out[21]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
count	1.000209e+06	1000209	1000209	1.000209e+06	1.000209e+06	1.000209e+06	1000209	1.000209e+06	1.000209e+06	1000209
unique	NaN	3706	301	NaN	NaN	NaN	2	NaN	NaN	3439
top	NaN	American Beauty (1999)	Comedy	NaN	NaN	NaN	M	NaN	NaN	94110
freq	NaN	3428	116883	NaN	NaN	NaN	753769	NaN	NaN	3802
mean	1.865540e+03	NaN	NaN	3.024512e+03	3.581564e+00	9.722437e+08	NaN	2.973831e+01	8.036138e+00	NaN
std	1.096041e+03	NaN	NaN	1.728413e+03	1.117102e+00	1.215256e+07	NaN	1.175198e+01	6.531336e+00	NaN
min	1.000000e+00	NaN	NaN	1.000000e+00	1.000000e+00	9.567039e+08	NaN	1.000000e+00	0.000000e+00	NaN
25%	1.030000e+03	NaN	NaN	1.506000e+03	3.000000e+00	9.653026e+08	NaN	2.500000e+01	2.000000e+00	NaN
50%	1.835000e+03	NaN	NaN	3.070000e+03	4.000000e+00	9.730180e+08	NaN	2.500000e+01	7.000000e+00	NaN
75%	2.770000e+03	NaN	NaN	4.476000e+03	4.000000e+00	9.752209e+08	NaN	3.500000e+01	1.400000e+01	NaN
max	3.952000e+03	NaN	NaN	6.040000e+03	5.000000e+00	1.046455e+09	NaN	5.600000e+01	2.000000e+01	NaN

In [22]:
```
Master_Data_df.shape
```

Out[22]:
```
(1000209, 10)
```

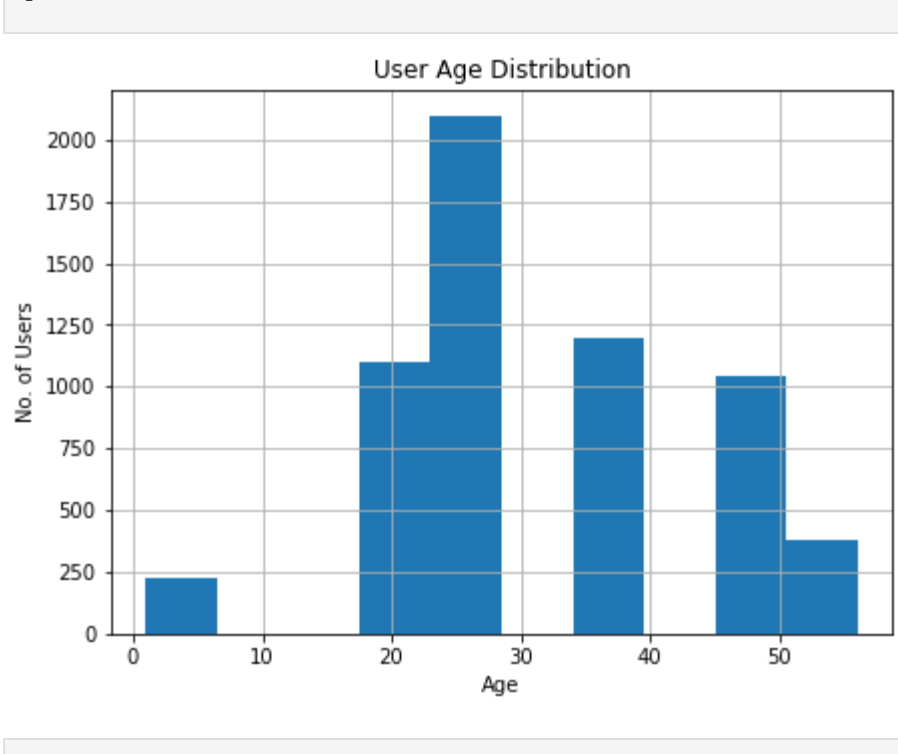
Explore the datasets using visual representations

Visualizing User Age Distribution

In [23]:
```
#Use view_counts() to see the age distribution
users_df['Age'].value_counts()
```

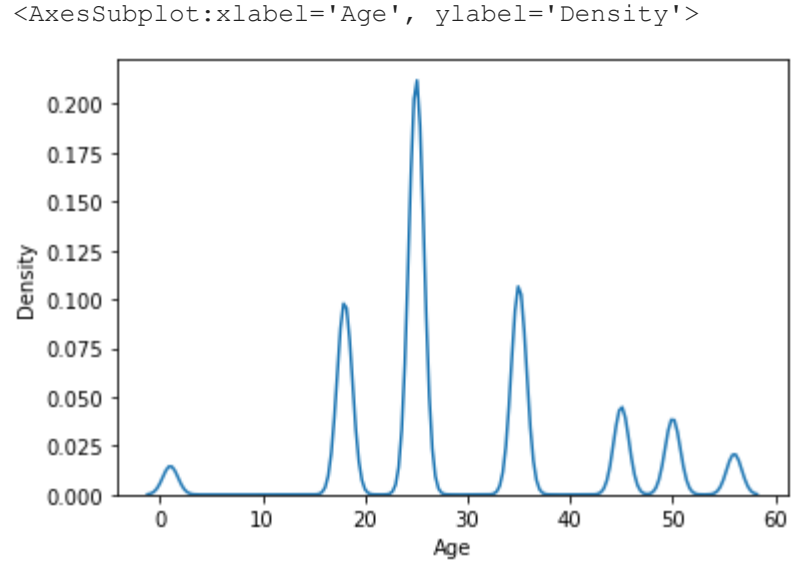
Out[23]:
```
25    2096
35    1193
18    1103
45     550
50     496
56     380
1      222
Name: Age, dtype: int64
```

In [24]:
```
#plot the age distribution in the form of histogram
plt.figure(figsize=(7,5))
users_df['Age'].hist()
plt.title("User Age Distribution")
plt.xlabel("Age")
plt.ylabel("No. of Users")
plt.show()
```



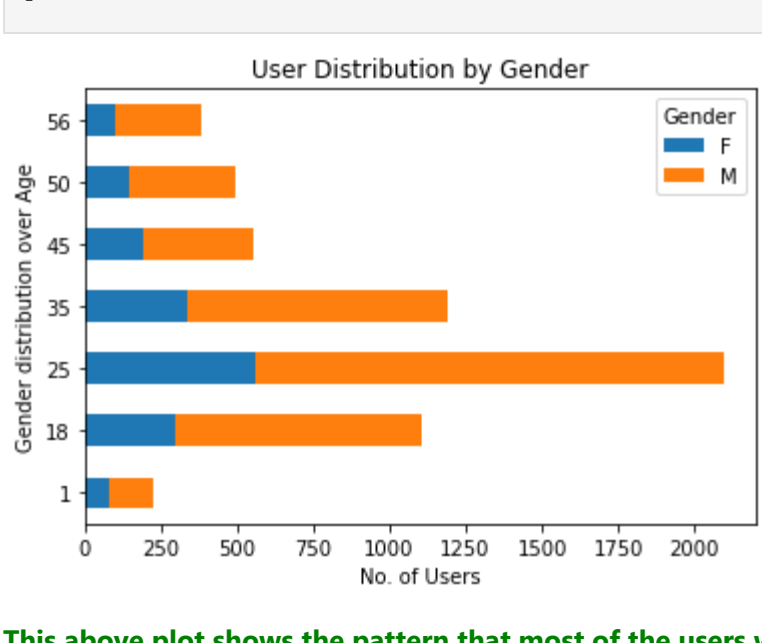
In [25]:
```
#Plotting the distribution curve.
sns.distplot(Master_Data_df["Age"], hist=False)
```

Out[25]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```



From the above plot, it seems most the users are from age group of 25, which inturn means 25 to 34 range.

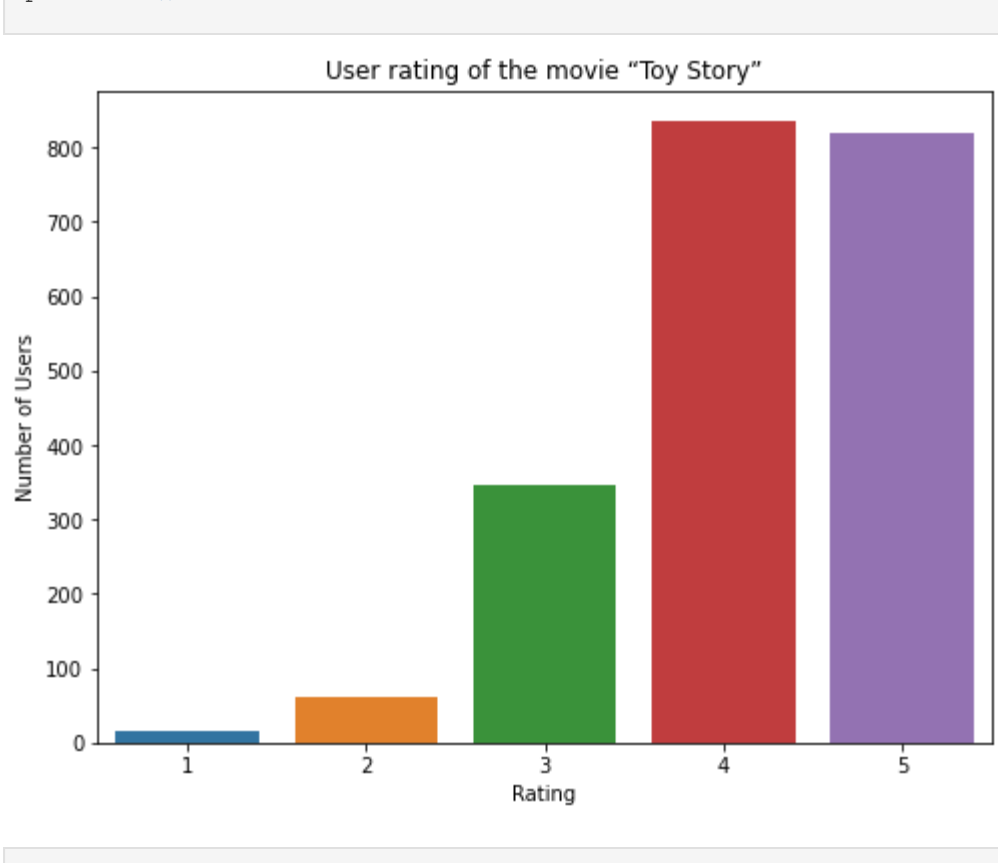
In [26]:
```
#Plotting a cross tabulation barplot to see Gender distribution over Age.
pd.crosstab(users_df["Age"],users_df["Gender"]).plot(kind="barh",stacked=True)
plt.title("User Distribution by Gender")
plt.xlabel("No. of Users")
plt.ylabel("Gender distribution over Age")
plt.show()
```



This above plot shows the pattern that most of the users were from Male gender.

Visualizing User rating of the movie "Toy Story"

In [27]:
```
#Plot the ToyStory rating distribution by No. of users.
plt.figure(figsize=(8,6))
Group_by_title_df = Master_Data_df.groupby('Title')
ToyStory_df = Group_by_title_df.get_group('Toy Story (1995)')
#Countplot the User rating
sns.countplot(ToyStory_df.Rating)
plt.title('User rating of the movie "Toy Story"')
plt.xlabel('Rating')
plt.ylabel('Number of Users')
plt.show()
```



In [28]:
```
#User ratings of ToyStory in numbers
ToyStory_df.Rating.value_counts()
```

Out[28]:
```
4    835
5    820
3    345
2     61
1     16
Name: Rating, dtype: int64
```

In [29]:
```
#Calculate the overall Average rating.
ToyStory_df.Rating.mean()
```

Out[29]:
```
4.146846413095811
```

The Movie Toy Story has got maximum number of 4-Stars ratings. It has got '835' 4-Star rating, '820' 5-Star ratings. On an average Toy Story has got 4.14 ratings.

Top 25 movies by viewership rating

Top 25 movies by number of Viewers

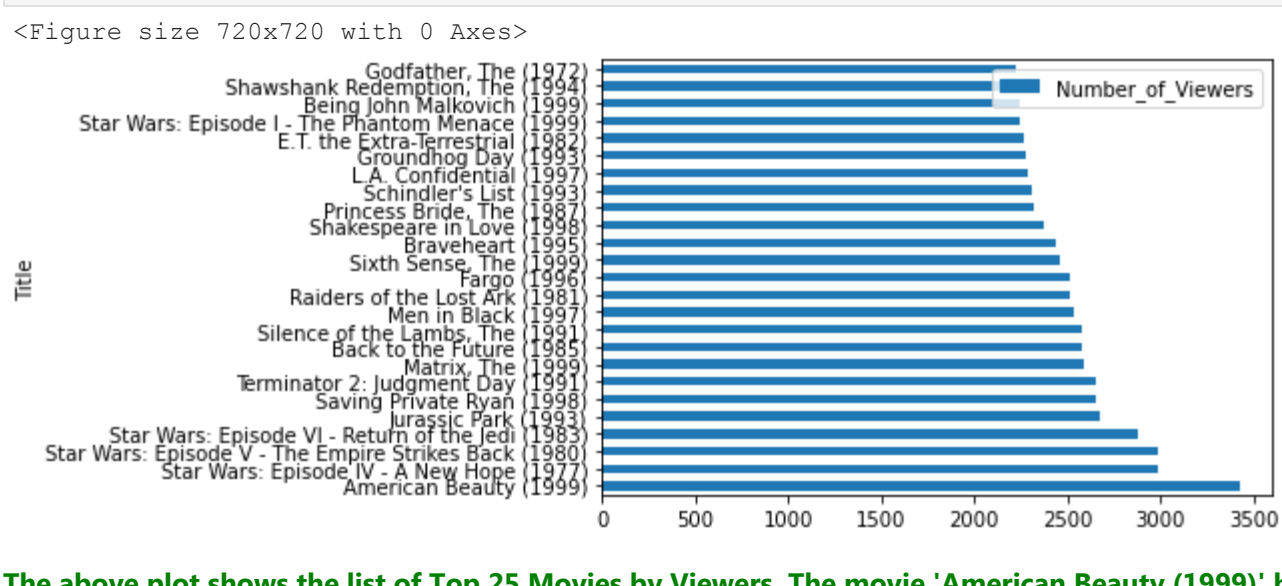
In [30]:
```
Top25_by_viewers = pd.DataFrame(Master_Data_df.groupby('Title')['Rating']
                               .count().sort_values(ascending=False).head(25))

Top25_by_viewers.rename(columns = {'Rating':'Number_of_Viewers'}, inplace = True)
Top25_by_viewers
```

Out[30]:

Title	Number_of_Viewers
American Beauty (1999)	3428
Star Wars: Episode IV - A New Hope (1977)	2991
Star Wars: Episode V - The Empire Strikes Back (1980)	2990
Star Wars: Episode VI - Return of the Jedi (1983)	2883
Jurassic Park (1993)	2672
Saving Private Ryan (1998)	2653
Terminator 2: Judgment Day (1991)	2649
Matrix, The (1999)	2590
Back to the Future (1985)	2583
Silence of the Lambs, The (1991)	2578
Men in Black (1997)	2538
Raiders of the Lost Ark (1981)	2514
Fargo (1996)	2513
Sixth Sense, The (1999)	2459
Braveheart (1995)	2443
Shakespeare in Love (1998)	2369
Princess Bride, The (1987)	2318
Schindler's List (1993)	2304
L.A. Confidential (1997)	2288
Groundhog Day (1993)	2278
E.T. the Extra-Terrestrial (1982)	2269
Star Wars: Episode I - The Phantom Menace (1999)	2250
Being John Malkovich (1999)	2241
Shawshank Redemption, The (1994)	2227
Godfather, The (1972)	2223

In [31]:
```
#plot Top 25 Movies based on Maximum Viewership.
plt.figure(figsize=(10,10))
Top25_by_viewers.plot(kind='barh')
plt.show()
```

```
<Figure size 720x720 with 0 Axes>
```



The above plot shows the list of Top 25 Movies by Viewers. The movie 'American Beauty (1999)' has got the highest number of viewers, which is '3428'.

Let's try to rearrange the Top 25 movies based on Average user ratings and number of viewers greater than 2000 viewers.

In [32]:
```
#Calculating Average movie rating for all movies in the dataset.
Movies_Avg_Ratings = pd.DataFrame(Master_Data_df.groupby('Title')['Rating']
                                  .mean().sort_values(ascending=False))

Movies_Avg_Ratings.rename(columns = {'Rating':'Average_Ratings'}, inplace = True)
Movies_Avg_Ratings
```

Out[32]:

Title	Average_Ratings
Ulysses (Ulisse) (1954)	5.0
Lured (1947)	5.0
Follow the Bitch (1998)	5.0
Bittersweet Motel (2000)	5.0
Song of Freedom (1936)	5.0
...	...
Fantastic Night, The (La Nuit Fantastique) (1949)	1.0
Cheetah (1989)	1.0
Torso (Corpi Presentano Tracce di Violenza Carnale) (1973)	1.0
Mutters Courage (1995)	1.0
Windows (1980)	1.0

3706 rows × 1 columns

In [33]:
```
#Merge the Top25_by_viewers and Movies_Avg_Ratings dataframes based on Movie Title.
Top25_Movies = pd.merge(Top25_by_viewers, Movies_Avg_Ratings, on='Title')

#New Dataframe created with Movie title, number of viewers and average ratings.
Top25_Movies
```

Out[33]:

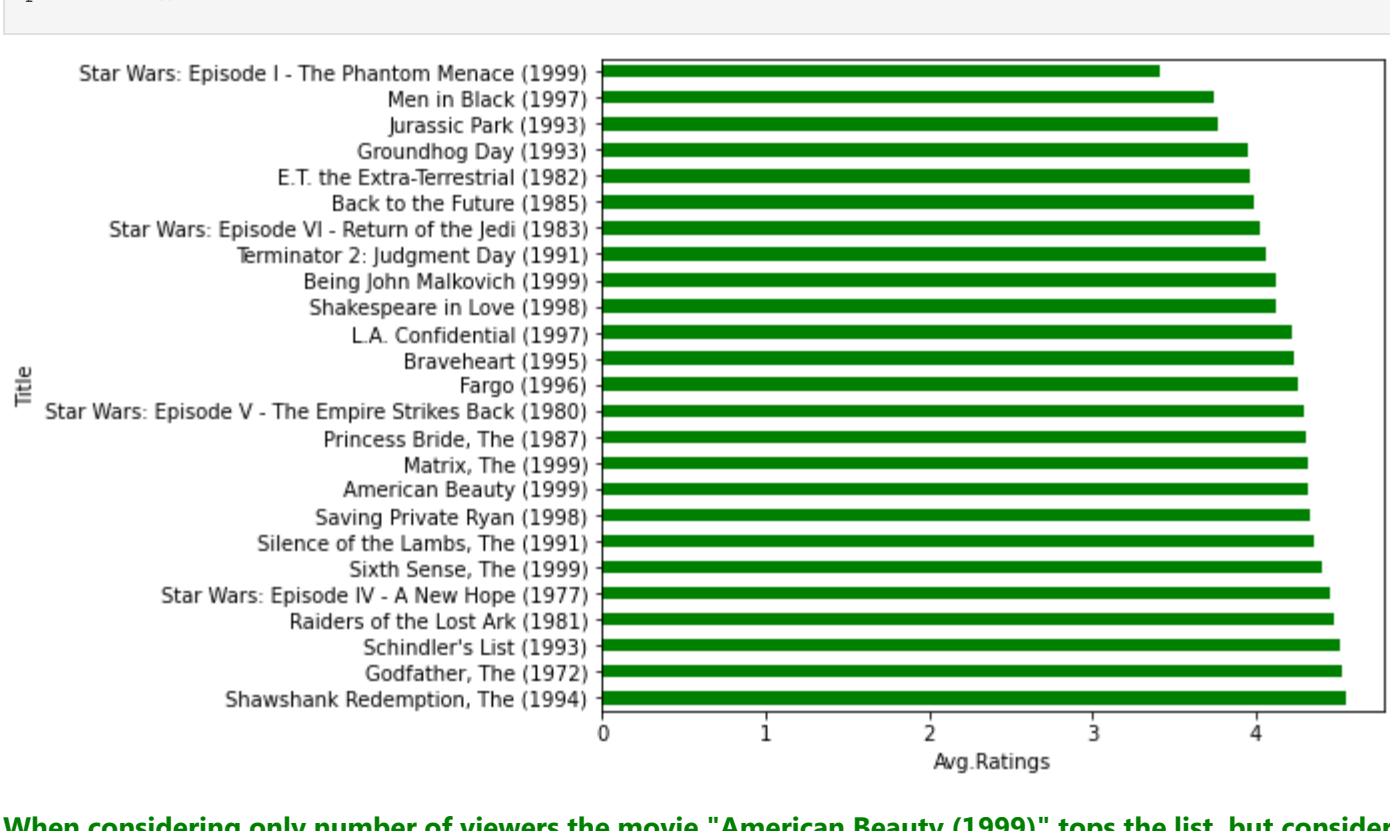
Title	Number_of_Viewers	Average_Ratings
American Beauty (1999)	3428	4.317386
Star Wars: Episode IV - A New Hope (1977)	2991	4.453694
Star Wars: Episode V - The Empire Strikes Back (1980)	2990	4.292977
Star Wars: Episode VI - Return of the Jedi (1983)	2883	4.022893
Jurassic Park (1993)	2672	3.763847
Saving Private Ryan (1998)	2653	4.337354
Terminator 2: Judgment Day (1991)	2649	4.058513
Matrix, The (1999)	2590	4.315830
Back to the Future (1985)	2583	3.990321
Silence of the Lambs, The (1991)	2578	4.351823
Men in Black (1997)	2538	3.739953
Raiders of the Lost Ark (1981)	2514	4.477725
Fargo (1996)	2513	4.254676
Sixth Sense, The (1999)	2459	4.406263
Braveheart (1995)	2443	4.234957
Shakespeare in Love (1998)	2369	4.127480
Princess Bride, The (1987)	2318	4.303710
Schindler's List (1993)	2304	4.510417
L.A. Confidential (1997)	2288	4.219406
Groundhog Day (1993)	2278	3.953029
E.T. the Extra-Terrestrial (1982)	2269	3.965183
Star Wars: Episode I - The Phantom Menace (1999)	2250	3.409778
Being John Malkovich (1999)	2241	4.125390
Shawshank Redemption, The (1994)	2227	4.554558
Godfather, The (1972)	2223	4.524966

Rearranging Top 25 Movies based on Average ratings.

In [34]:
```
#Sort the newly created dataframe based on rating.
Top25_Viewership_Rating = Top25_Movies.Average_Ratings.sort_values(ascending=False)
Top25_Viewership_Rating
```

Out[34]:
```
Title
Shawshank Redemption, The (1994)                        4.554558
Godfather, The (1972)                                   4.524966
Schindler's List (1993)                                 4.510417
Raiders of the Lost Ark (1981)                          4.477725
Star Wars: Episode IV - A New Hope (1977)               4.453694
Sixth Sense, The (1999)                                 4.406263
Silence of the Lambs, The (1991)                        4.351823
Saving Private Ryan (1998)                              4.337354
American Beauty (1999)                                  4.317386
Matrix, The (1999)                                      4.315830
Princess Bride, The (1987)                              4.303710
Star Wars: Episode V - The Empire Strikes Back (1980)   4.292977
Fargo (1996)                                            4.254676
Braveheart (1995)                                       4.234957
L.A. Confidential (1997)                                4.219406
Shakespeare in Love (1998)                              4.127480
Being John Malkovich (1999)                             4.125390
Terminator 2: Judgment Day (1991)                       4.058513
Star Wars: Episode VI - Return of the Jedi (1983)       4.022893
Back to the Future (1985)                               3.990321
E.T. the Extra-Terrestrial (1982)                       3.965183
Groundhog Day (1993)                                    3.953029
Jurassic Park (1993)                                    3.763847
Men in Black (1997)                                     3.739953
Star Wars: Episode I - The Phantom Menace (1999)        3.409778
Name: Average_Ratings, dtype: float64
```

In [35]:
```
plt.figure(figsize=(7,6))
Top25_Viewership_Rating.plot(kind='barh',color='green')
plt.xlabel("Avg.Ratings")
plt.show()
```



When considering only number of viewers the movie "American Beauty (1999)" tops the list, but considering Average user rating as-well-as the number of viewers watched(above 2000 viewers), the movie "The Shawshank Redemption" seems to be the top movie based on viewership ratings.

Find the ratings for all the movies reviewed by for a particular user of user id = 2696

In [36]:
```
#Create a dataframe with movies reviewed by User ID 2696
UserID_2696_df = Master_Data_df[Master_Data_df['UserID'] == 2696]
UserID_2696_df.head()
```

Out[36]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
991035	350	Client, The (1994)	Drama|Mystery|Thriller	2696	3	973308886	M	25	7	24210
991036	800	Lone Star (1996)	Drama|Mystery	2696	5	973308842	M	25	7	24210
991037	1092	Basic Instinct (1992)	Mystery|Thriller	2696	4	973308886	M	25	7	24210
991038	1097	E.T. the Extra-Terrestrial (1982)	Children's|Drama|Fantasy|Sci-Fi	2696	3	973308690	M	25	7	24210
991039	1258	Shining, The (1980)	Horror	2696	4	973308710	M	25	7	24210

In [37]:
```
type(UserID_2696_df)
```

Out[37]:
```
pandas.core.frame.DataFrame
```

In [38]:
```
#Number of Movies rated by USedID_2629
len(UserID_2696_df["Title"].explode().unique())
```

Out[38]:
```
20
```

UserID_2696 has reviewed 20 Movies in total.

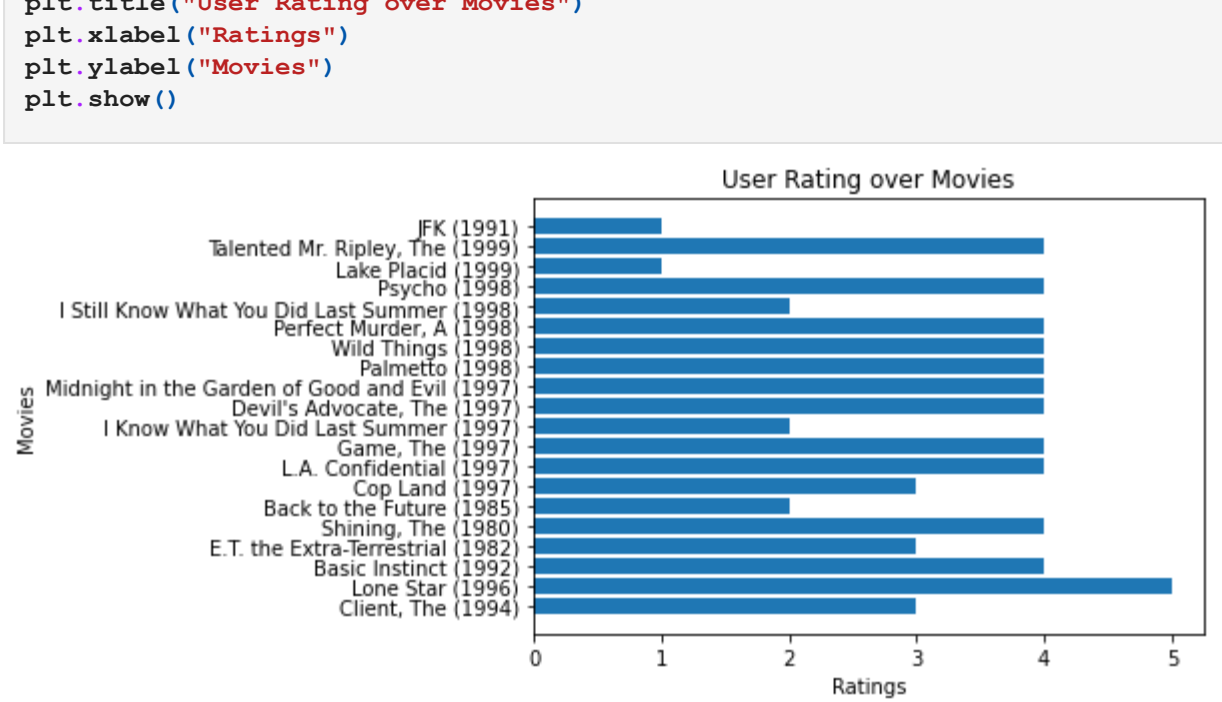
In [39]:
```
#List of Movies reviewed by UserID_2696
list(UserID_2696_df.Title)
```

Out[39]:
```
['Client, The (1994)',
 'Lone Star (1996)',
 'Basic Instinct (1992)',
 'E.T. the Extra-Terrestrial (1982)',
 'Shining, The (1980)',
 'Back to the Future (1985)',
 'Cop Land (1997)',
 'L.A. Confidential (1997)',
 'Game, The (1997)',
 'I Know What You Did Last Summer (1997)',
 "Devil's Advocate, The (1997)",
 'Midnight in the Garden of Good and Evil (1997)',
 'Palmetto (1998)',
 'Wild Things (1998)',
 'Perfect Murder, A (1998)',
 'I Still Know What You Did Last Summer (1998)',
 'Psycho (1998)',
 'Lake Placid (1999)',
 'Talented Mr. Ripley, The (1999)',
 'JFK (1991)']
```

In [40]:
```
#Most reviewed Movie-Genres by UserID_2696
UserID_2696_df.Genres.value_counts()
```

Out[40]:
```
Mystery|Thriller                     3
Drama|Mystery|Thriller               2
Horror|Mystery|Thriller              2
Drama|Mystery                        2
Crime|Drama|Mystery|Thriller         1
Comedy|Crime|Drama|Mystery           1
Horror                               1
Crime|Horror|Mystery|Thriller        1
Film-Noir|Mystery|Thriller           1
Crime|Drama|Mystery                  1
Comedy|Sci-Fi                        1
Crime|Horror|Thriller                1
Children's|Drama|Fantasy|Sci-Fi      1
Horror|Thriller                      1
Crime|Film-Noir|Mystery|Thriller     1
Name: Genres, dtype: int64
```

In [41]:
```python
#Plotting UserID_2696 Rating distribution over Movies.
plt.barh(UserID_2696_df["Title"],UserID_2696_df["Rating"])
plt.title("User Rating over Movies")
plt.xlabel("Ratings")
plt.ylabel("Movies")
plt.show()
```



From the above plot, we can say that, the highest rated movie by UserID_2696 is 'Lone Star(1996)'.

Feature Engineering:

Use column genres:

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)

In [42]:
```python
#Elements in Genres column are in a combined form.
Master_Data_df['Genres']
```

Out[42]:
```
0                   Animation|Children's|Comedy
1          Animation|Children's|Musical|Romance
2                                         Drama
3                  Action|Adventure|Fantasy|Sci-Fi
4                                     Drama|War
                          ...
1000204                          Drama|Thriller
1000205                  Comedy|Horror|Thriller
1000206                          Comedy|Romance
1000207                         Action|Thriller
1000208                            Action|Drama
Name: Genres, Length: 1000209, dtype: object
```

Elements in Genres column are in combined form. Let's split the genres as a list and add them as new columns in Master_Data itself.

In [43]:
```python
#Create a new column and add the Genres column data in the form of list by spliting the combined data.
Master_Data_df['Genres_List'] = Master_Data_df['Genres'].str.split('|')
#View the observation to check the new column created.
Master_Data_df.head()
```

Out[43]:

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

In [44]:
```python
Master_Data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 11 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   MovieID      1000209 non-null  int64
 1   Title        1000209 non-null  object
 2   Genres       1000209 non-null  object
 3   UserID       1000209 non-null  int64
 4   Rating       1000209 non-null  int64
 5   Timestamp    1000209 non-null  int64
 6   Gender       1000209 non-null  object
 7   Age          1000209 non-null  int64
 8   Occupation   1000209 non-null  int64
 9   Zip-code     1000209 non-null  object
 10  Genres_List  1000209 non-null  object
dtypes: int64(6), object(5)
memory usage: 72.5+ MB
```

In [45]:
```python
#Transform list-like element in 'Genres_List' column to find the unique genres.
nd_unique_genres=Master_Data_df['Genres_List'].explode().unique()
nd_unique_genres
```

Out[45]:
```
array(['Animation', "Children's", 'Comedy', 'Musical', 'Romance', 'Drama',
       'Action', 'Adventure', 'Fantasy', 'Sci-Fi', 'War', 'Crime',
       'Thriller', 'Western', 'Horror', 'Mystery', 'Documentary',
       'Film-Noir'], dtype=object)
```

In [46]:
```python
type(nd_unique_genres)
```

Out[46]:
```
numpy.ndarray
```

The ndarray 'nd_unique_genres' shows all the unique categories of genres in the dataset such as 'Animation', "Children's", 'Comedy', 'Musical', 'Romance', 'Drama', 'Action', 'Adventure', 'Fantasy', 'Sci-Fi', 'War', 'Crime', 'Thriller', 'Western', 'Horror', 'Mystery', 'Documentary', 'Film-Noir'.

Create a separate column for each genre category with a one-hot encoding

Data Preprocessing

Convert a multiple list of the categorical data into one hot encoded Numerical type using MultiLabelBinarizer().

MultiLabelBinarizer

MultiLabelBinarizer can be used to transform between iterable of iterables and a multilabel format. Since our Column Genres_List consist of multiple list, we can use MultiLabelBinarizer for one hot encoding.

In [47]:
```python
from sklearn.preprocessing import MultiLabelBinarizer

MultiLabel_Binarizer = MultiLabelBinarizer()

# use mlb to create a new dataframe of the genres from the list for each row from the original data
Encoded_Genres_df = pd.DataFrame(MultiLabel_Binarizer.fit_transform(Master_Data_df.Genres_List),
                                 columns=MultiLabel_Binarizer.classes_,
                                 index=Master_Data_df.Genres_List.index)
Encoded_Genres_df
```

Out[47]:

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

1000209 rows × 18 columns

In [48]:
```python
type(Encoded_Genres_df)
```

Out[48]:
```
pandas.core.frame.DataFrame
```

The Encoded_Genres_df dataframe contains columns of genres with one-hot encoded values. This will be merged with filtered Master data for creating test and training datasets.

Determine the features affecting the ratings of any particular movie.

Create a new filtered dataframe with the movie data features that are relevant to rating outcome:

```
* zip code
* gender
* age
* occupation
```

In [49]:
```python
#Create a dataframe with MovieID, Rating, Gender, Age, Occupation and Zip-code
Filtered_Master_Data_df = Master_Data_df.iloc[:, [0,4,6,7,8,9]]
Filtered_Master_Data_df.head()
```

Out[49]:

	MovieID	Rating	Gender	Age	Occupation	Zip-code
0	1	5	F	1	10	48067
1	48	5	F	1	10	48067
2	150	5	F	1	10	48067
3	260	4	F	1	10	48067
4	527	5	F	1	10	48067

In [50]:
```python
Filtered_Master_Data_df.dtypes
```

Out[50]:
```
MovieID        int64
Rating         int64
Gender        object
Age            int64
Occupation     int64
Zip-code      object
dtype: object
```

We need to change the datatype of elements in Gender and Zip-code columns.

Encode the Gender column by simply replacing 'F' and 'M' category to '0' and '1'.

In [51]:
```python
#The Gender column has categorical data of nominal type.
#Convert Gender category F and M to binary values '0's and '1's respectively.
Filtered_Master_Data_df['Gender'].replace(['F','M'],[0,1],inplace=True)
Filtered_Master_Data_df
```

Out[51]:

	MovieID	Rating	Gender	Age	Occupation	Zip-code
0	1	5	0	1	10	48067
1	48	5	0	1	10	48067
2	150	5	0	1	10	48067
3	260	4	0	1	10	48067
4	527	5	0	1	10	48067
...	...	...	...	...	...	...
1000204	3513	4	1	25	4	92843
1000205	3535	2	1	25	4	92843
1000206	3536	5	1	25	4	92843
1000207	3555	3	1	25	4	92843
1000208	3578	5	1	25	4	92843

1000209 rows × 6 columns

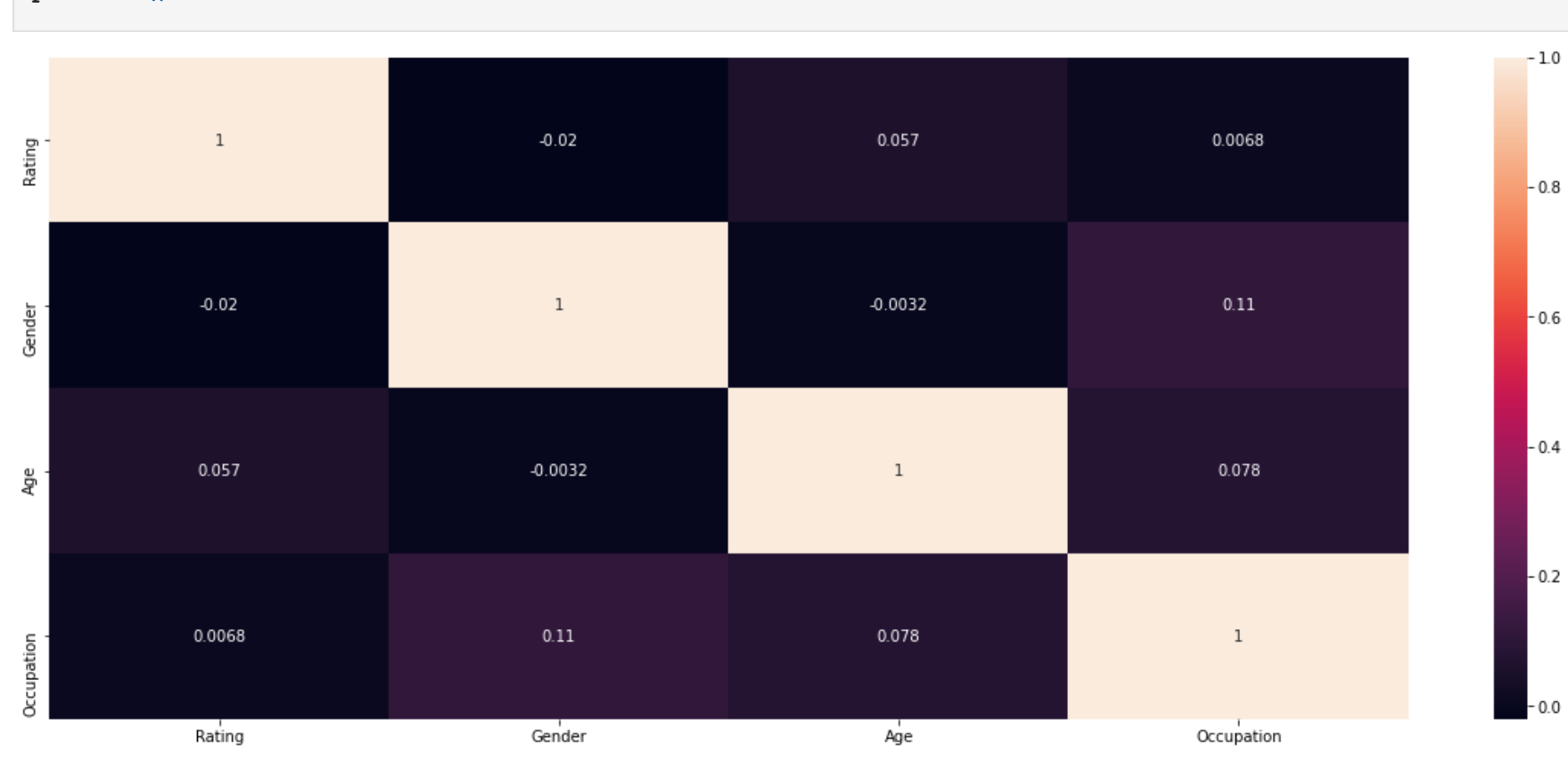
In [52]:
```python
#Also change 'zip code' into numerical datatype
Filtered_Master_Data_df['Zip-code'] = Filtered_Master_Data_df['Zip-code'].str[:5]
pd.to_numeric(Filtered_Master_Data_df['Zip-code'])
```

Out[52]:
```
0          48067
1          48067
2          48067
3          48067
4          48067
           ...
1000204    92843
1000205    92843
1000206    92843
1000207    92843
1000208    92843
Name: Zip-code, Length: 1000209, dtype: int64
```

In [53]:
```python
Filtered_Master_Data_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 6 columns):
 #   Column      Non-Null Count    Dtype
---  ------      --------------    -----
 0   MovieID     1000209 non-null  int64
 1   Rating      1000209 non-null  int64
 2   Gender      1000209 non-null  int64
 3   Age         1000209 non-null  int64
 4   Occupation  1000209 non-null  int64
 5   Zip-code    1000209 non-null  object
dtypes: int64(5), object(1)
memory usage: 49.6+ MB
```

In [54]:
```python
#plot the heatmap to obtain the correlation between rating over Age, Occupation and Gender.
plt.figure(figsize=(20,8))
correlation=Filtered_Master_Data_df.drop(['MovieID','Zip-code'],axis=1).corr()
sns.heatmap(correlation,xticklabels=correlation.columns.values,
            yticklabels=correlation.columns.values,annot=True)
plt.show()
```



In [55]:
```python
#Correlation in numbers
Filtered_Master_Data_df[Filtered_Master_Data_df.columns[1:]].corr()['Rating'][:]
```

Out[55]:
```
Rating        1.000000
Gender       -0.019861
Age           0.056869
Occupation    0.006753
Name: Rating, dtype: float64
```

We see that Age has the highest correlation over movie ratings. From the results we can say that Age is the most affecting feature over rating of movies.

Gender has Negative or inverse correlation over Ratings, which describes that these two variables tend to move in opposite size and direction from one another, such that when one increases the other variable decreases, and vice-versa.

Merging 'Filtered_Master_Data_df' with 'Encoded_Genres_df'.

In [56]:
```python
#Create new dataframe by Merging 'Filtered_Master_Data_df' with 'Encoded_Genres_df'
Master_Features_df = pd.merge(Filtered_Master_Data_df, Encoded_Genres_df, left_index=True, right_index=True)
Master_Features_df.head()
```

Out[56]:

	MovieID	Rating	Gender	Age	Occupation	Zip-code	Action	Adventure	Animation	Children's	...	Fantasy	Film-Noir	Horror	Musical	Myster
0	1	5	0	1	10	48067	0	0	1	1	...	0	0	0	0	
1	48	5	0	1	10	48067	0	0	1	1	...	0	0	0	1	
2	150	5	0	1	10	48067	0	0	0	0	...	0	0	0	0	
3	260	4	0	1	10	48067	1	1	0	0	...	1	0	0	0	
4	527	5	0	1	10	48067	0	0	0	0	...	0	0	0	0	

5 rows × 24 columns

In [57]:
```python
Master_Features_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 24 columns):
 #   Column       Non-Null Count    Dtype
---  ------       --------------    -----
 0   MovieID      1000209 non-null  int64
 1   Rating       1000209 non-null  int64
 2   Gender       1000209 non-null  int64
 3   Age          1000209 non-null  int64
 4   Occupation   1000209 non-null  int64
 5   Zip-code     1000209 non-null  object
 6   Action       1000209 non-null  int32
 7   Adventure    1000209 non-null  int32
 8   Animation    1000209 non-null  int32
 9   Children's   1000209 non-null  int32
 10  Comedy       1000209 non-null  int32
 11  Crime        1000209 non-null  int32
 12  Documentary  1000209 non-null  int32
 13  Drama        1000209 non-null  int32
 14  Fantasy      1000209 non-null  int32
 15  Film-Noir    1000209 non-null  int32
 16  Horror       1000209 non-null  int32
 17  Musical      1000209 non-null  int32
 18  Mystery      1000209 non-null  int32
 19  Romance      1000209 non-null  int32
 20  Sci-Fi       1000209 non-null  int32
 21  Thriller     1000209 non-null  int32
 22  War          1000209 non-null  int32
 23  Western      1000209 non-null  int32
dtypes: int32(18), int64(5), object(1)
memory usage: 142.5+ MB
```

Train Model For Logistic Regression

- **Create the X_feature dataframe, include all the independent variables, drop the Ratings column since this is our dependent variable.**
- **Also drop the zip-code and MovieID columns, since it has no major influence on the outcome of the movie ratings.**
- **Let's make a smaller dataset with only relevant features.**

In [58]:
```python
X_feature = Filtered_Master_Data_df.drop(['Zip-code'], axis=1)
X_feature.head()
```

Out[58]:

	MovieID	Rating	Gender	Age	Occupation
0	1	5	0	1	10
1	48	5	0	1	10
2	150	5	0	1	10
3	260	4	0	1	10
4	527	5	0	1	10

Since we're dealing with a large dataset, let's just train and test on a smaller portion of the dataset. There are '3706' Movies in the dataset. Let's grab only the first '500' unique movie IDs. We have to make sure both our X and Y vectors are the same number of rows.

In [59]:
```python
X_feature_small = X_feature[X_feature['MovieID'] < 500]
X_feature_small.head()
```

Out[59]:

	MovieID	Rating	Gender	Age	Occupation
0	1	5	0	1	10
1	48	5	0	1	10
2	150	5	0	1	10
3	260	4	0	1	10
53	1	4	0	50	9

The training data includes only the features we feel would influence the outcome of the movie rating.

I have chosen that Occupation, Age and Gender are the most influential.

We will get rid of the MovieID and Rating column as this will conflict with our regression.

In [60]:
```python
#Sampled X_Feature
X_feature_small_trimmed = X_feature_small.drop(['MovieID','Rating'], axis=1)
```

In [61]:
```python
X_feature_small_trimmed.shape
```

Out[61]:
```
(135309, 3)
```

For '500' Movies, there are '135309' rating enrties from different users is available in the trimmed dataset.

In [62]:
```python
X_feature_small_trimmed.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 135309 entries, 0 to 994618
Data columns (total 3 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   Gender      135309 non-null  int64
 1   Age         135309 non-null  int64
 2   Occupation  135309 non-null  int64
dtypes: int64(3)
memory usage: 4.1 MB
```

In [63]:
```python
X_feature_small_trimmed.head()
```

Out[63]:

	Gender	Age	Occupation
0	0	1	10
1	0	1	10
2	0	1	10
3	0	1	10
53	0	50	9

In [64]:
```python
plt.figure(figsize=(7,5))
corr=X_feature_small_trimmed.corr()
sns.heatmap(corr,xticklabels=corr.columns.values,yticklabels=corr.columns.values,annot=True)
plt.show()
```

The target feature we are trying to predict is the dependent variable "Rating", in other words the 'y' of the linear equation.

In [65]:
```python
#Target variable
Y_target = Master_Features_df['Rating'][Master_Features_df['MovieID']< 500]
```

In [66]:
```python
Y_target.shape
```

Out[66]:
```
(135309,)
```

Develop an appropriate model to predict the movie ratings

Split the data into training and testing data. By default this is 75% training and 25% testing

In [67]:
```python
#Import train_test_split from sklearn model selection
from sklearn.model_selection import train_test_split

#Assign the feature and target to split the training and testing dataset
x_train, x_test, y_train, y_test = train_test_split(X_feature_small_trimmed,Y_target,random_state=1)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[67]:
```
((101481, 3), (33828, 3), (101481,), (33828,))
```

Logistic regression is best used for predicting categorical data. We need to do logistic regression on the training data so we can see how well our test data does the prediction.

In [68]:
```python
#import Logistic Regression model from sklearn linear model
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(solver = 'liblinear')

#Fit the training dataset
logreg.fit(x_train,y_train)
```

Out[68]:
```
LogisticRegression(solver='liblinear')
```

Let's predict the outcome based on test set

In [69]:
```python
#Y_predict
y_pred = logreg.predict(x_test)
```

Let's get the accuracy score of the test data.

In [70]:
```python
#Import metrics from sklearn
from sklearn import metrics
print("Accuracy Score:",metrics.accuracy_score(y_test,y_pred))
print("Confusion Matrix:\n",metrics.confusion_matrix(y_test, y_pred))
print("Classification Report:\n",metrics.classification_report(y_test, y_pred))
```

```
Accuracy Score: 0.3398663828780892
Confusion Matrix:
 [[    0     0     0  1992     0]
 [    0     0     0  4109     0]
 [    0     0     0  9267     0]
 [    0     0     0 11497     0]
 [    0     0     0  6963     0]]
Classification Report:
               precision    recall  f1-score   support

           1       0.00      0.00      0.00      1992
           2       0.00      0.00      0.00      4109
           3       0.00      0.00      0.00      9267
           4       0.34      1.00      0.51     11497
           5       0.00      0.00      0.00      6963

    accuracy                           0.34     33828
   macro avg       0.07      0.20      0.10     33828
weighted avg       0.12      0.34      0.17     33828
```

Accuracy score acheived is 34%.

In [71]:
```python
# print the first 30 true and predicted responses
print ('actual:    ', y_test.values[0:30])
print ('predicted: ', y_pred[0:30])
```

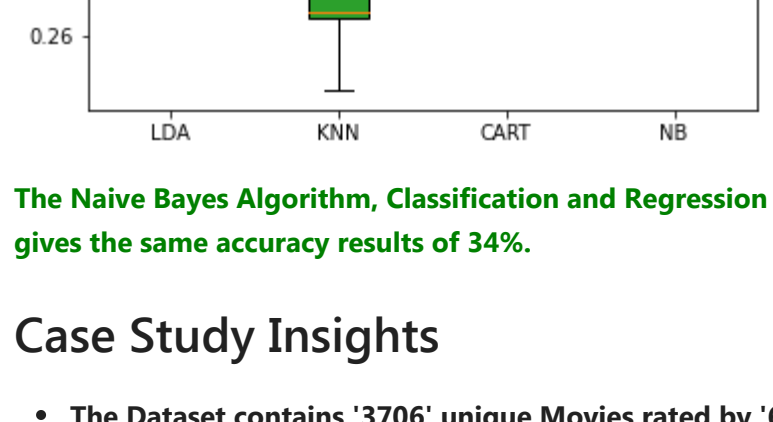
```
actual:     [2 1 3 5 3 1 3 4 4 3 3 3 4 4 2 1 3 4 2 2 3 3 4 3 4 4 3 3 3 4]
predicted:  [4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4]
```

Spot-Check with other Algorithms

In [72]:
```python
# Spot-Check Algorithms
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

seed = 7
models = []
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
# evaluate each model in turn
results = []
names = []
for name, model in models:
    kfold = KFold(n_splits=10, random_state=seed)
    cv_results = cross_val_score(model, x_train, y_train, cv=kfold, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LDA: 0.338911 (0.003931)
KNN: 0.267321 (0.012119)
CART: 0.340004 (0.003806)
NB: 0.338911 (0.003931)
```

In [73]:
```python
#Compare Algorithms by plotting the boxplot
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results, widths=0.35, patch_artist=True, boxprops=dict(facecolor="C2"))
ax.set_xticklabels(names)
plt.show()
```



The Naive Bayes Algorithm, Classification and Regression Trees(CART) and Linear Discriminant Analysis(LDA) algorithms also gives the same accuracy results of 34%.

Case Study Insights

- **The Dataset contains '3706' unique Movies rated by '6040' users. Most of the users are from age group of 25 to 34 range. The Gender distribution shows that there are more Male viewers compared to Female viewers. As per the Analysis, Age is the most affecting feature over movie rating. Also Gender has negative or inverse correlation over movie ratings.**
- **There are some unique categories of genres found in the dataset. such as, 'Animation', "Children's", 'Comedy', 'Musical', 'Romance', 'Drama', 'Action', 'Adventure', 'Fantasy', 'Sci-Fi', 'War', 'Crime', 'Thriller', 'Western', 'Horror', 'Mystery', 'Documentary', 'Film-Noir'.**
- **In the case of Movie ToyStory(1995), the movie has got '835' 4-Star rating and '820' 5-Star ratings. On an average ToyStory has got 4.14 ratings.**
- **The movie 'American Beauty (1999)' tops the top 25 movies by viewers, which is '3428' viewers. When considering average user rating as-well-as considerable number of viewers(above 2000 viewers), the movie "The Shawshank Redemption" tops the list, followed by The Godfather(1972).**
- **In the case of UserID_2696, the user has reviewed 20 movies in total. The highest rating given by the UserID_2696 is '5' for the movie "Lone Star(1996)".**
- **Logistic Regression model is chosen for predicting the movie ratings. since, the dataset contains catagorical data. The accuracy score obtained by Logistic regression model is 34%. Spot checking the training and testing dataset with different algorithm results in simillar accuracy of 34%.**