Project 02: Movielens Case Study

Submitted By Ms. Ankita Kale in September 2020

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

Dataset Description :
These files contain 1,000,209 anonymous ratings of approximately 3,900 movies made by 6,040 MovieLens users who joined MovieLens in 2000.

In [1]:

```
%config IPCompleter.greedy=True
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# machine learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

In [2]:

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

Import the three datasets

In [3]:

```
#Load the data
movies = pd.read_csv('data/movies.dat', sep="::" , header=None, names=["MovieID","Title","Genres"],
engine='python')
ratings = pd.read_csv("data/ratings.dat" , sep='::' , header=None, names =['UserID','MovieID','Rati
ng','Timestamp'] , engine='python')
users = pd.read_csv("data/users.dat", sep='::' , header=None, names =['UserID','Gender','Age','Occu
pation','Zip-code'] , engine='python')
```

In [4]:

```
movies.head()
```

Out[4]:

	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 [5]:

```
movies.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: 91.1+ KB
```

In [6]:

```
movies.describe()
```

Out[6]:

	MovieID
count	3883.000000
mean	1986.049446
std	1146.778349
min	1.000000
25%	982.500000
50%	2010.000000
75%	2980.500000
max	3952.000000

In [7]:

```
movies['Title'].describe()
```

Out[7]:

```
count                          3883
```

```
count                          3883
unique                         3883
top       Psycho Beach Party (2000)
freq                              1
Name: Title, dtype: object
```

In [8]:

```
ratings.head()
```

Out[8]:

	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 [10]:

```
ratings.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 [11]:

```
ratings.groupby(['UserID','MovieID']).count()
```

Out[11]:

		Rating	Timestamp
UserID	**MovieID**		
1	**1**	1	1
	48	1	1
	150	1	1
	260	1	1
	527	1	1
...	**...**	...	...
6040	**3683**	1	1
	3703	1	1
	3735	1	1
	3751	1	1
	3819	1	1

1000209 rows × 2 columns

In [12]:

```
ratings['Rating'].describe()
```

Out[12]:

```
count    1.000209e+06
mean     3.581564e+00
std      1.117102e+00
min      1.000000e+00
25%      3.000000e+00
50%      4.000000e+00
75%      4.000000e+00
max      5.000000e+00
Name: Rating, dtype: float64
```

In [13]:

```
users.head()
```

Out[13]:

	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 [14]:

```
users.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: 236.1+ KB
```

In [15]:

```
users['Age'].describe()
```

Out[15]:

```
count    6040.000000
mean       30.639238
std        12.895962
min         1.000000
25%        25.000000
50%        25.000000
75%        35.000000
max        56.000000
Name: Age, dtype: float64
```

In [16]:

```
len(set(users['UserID']))
```

Out[16]:

```
6040
```

In [17]:

```
6040*3883
```

Out[17]:

23453320

In [18]:

```
# Merging
tempDataset = pd.merge(ratings,movies,on='MovieID')
tempDataset.head()
```

Out[18]:

	UserID	MovieID	Rating	Timestamp	Title	Genres
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama
1	2	1193	5	978298413	One Flew Over the Cuckoo's Nest (1975)	Drama
2	12	1193	4	978220179	One Flew Over the Cuckoo's Nest (1975)	Drama
3	15	1193	4	978199279	One Flew Over the Cuckoo's Nest (1975)	Drama
4	17	1193	5	978158471	One Flew Over the Cuckoo's Nest (1975)	Drama

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

Here I created finalDF

In [19]:

```
finalDF = pd.merge(tempDataset, users, on='UserID')
finalDF.head()
```

Out[19]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F	1	10	48067
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F	1	10	48067
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F	1	10	48067
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F	1	10	48067
4	1	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy	F	1	10	48067

In [20]:

```
# Counting values based on Movie Rating
finalDF.groupby(['Rating']).sum()
```

Out[20]:

	UserID	MovieID	Timestamp	Age	Occupation
Rating					
1	167800234	110817755	54622277176071	1539828	437705
2	320205562	208381312	104631794087106	3076948	858971
3	789368925	501107422	254012338022416	7765396	2098727
4	1056656983	654499954	339307570467704	10523013	2826659

	UserID	MovieID	Timestamp	Age	Occupation
5	691[illegible]	39[illegible]	21987[illegible]	6839[illegible]	[illegible]

Explore the datasets using visual representations (graphs or tables), also include your comments

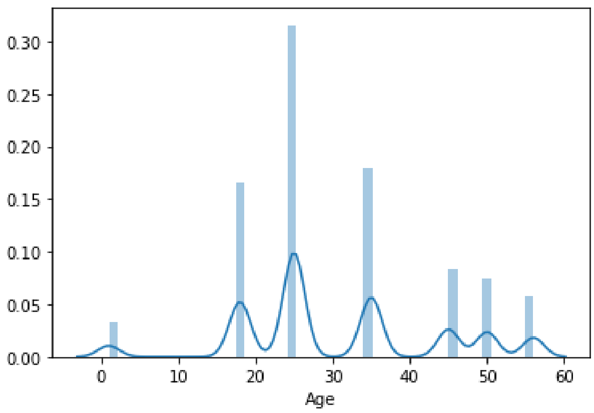
1. User Age Distribution

In [21]:

```
#User Age Distribution
sns.distplot(users.Age)
```

Out[21]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bab12a0a88>
```

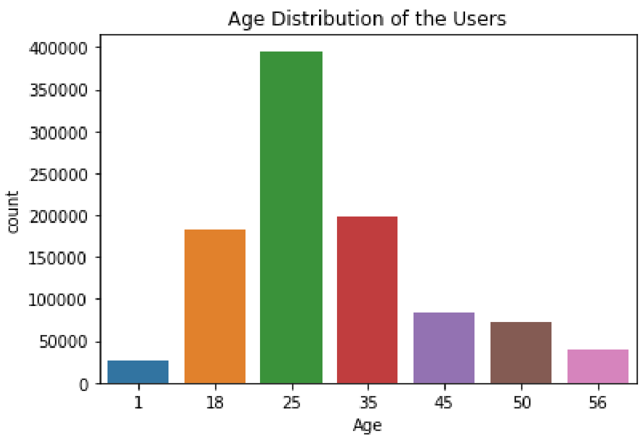


In [22]:

```
sns.countplot(x='Age',data = finalDF).set_title('Age Distribution of the Users')
```

Out[22]:

```
Text(0.5, 1.0, 'Age Distribution of the Users')
```



2. User rating of the movie “Toy Story”

In [23]:

```
finalDF[finalDF.Title.str.contains("Toy Story")]['Title'].unique()
```

Out[23]:

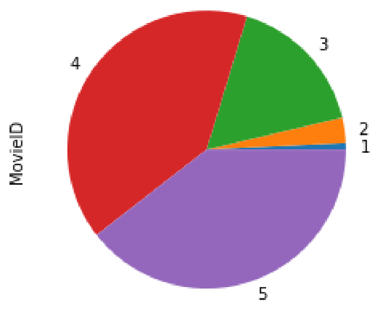
```
array(['Toy Story (1995)', 'Toy Story 2 (1999)'], dtype=object)
```

In [24]:

```
finalDF[finalDF.Title == 'Toy Story (1995)'].groupby('Rating')['MovieID'].count().plot(kind="pie")
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bab329b648>
```

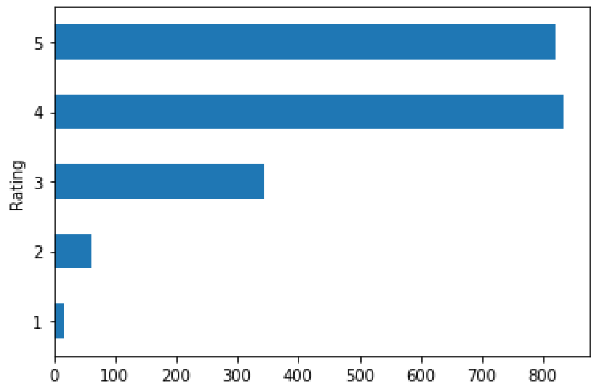


In [25]:

```
finalDF[finalDF.Title == 'Toy Story (1995)'].groupby('Rating')['MovieID'].count().plot(kind="barh"
)
```

Out[25]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bab32bb908>
```



It seems that lot of people rared 4 and 5 for the movie 'Toy Story (1995)'

In [26]:

```
# User rating of the movie "Toy Story"
toystoryRating = finalDF[finalDF['Title'].str.contains('Toy Story') == True]
toystoryRating
```

Out[26]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
40	1	1	5	978824268	Toy Story (1995)	Animation|Children's|Comedy	F	1	10	48067
50	1	3114	4	978302174	Toy Story 2 (1999)	Animation|Children's|Comedy	F	1	10	48067
417	17	3114	5	978159386	Toy Story 2 (1999)	Animation|Children's|Comedy	M	50	1	95350
634	18	1	4	978154768	Toy Story (1995)	Animation|Children's|Comedy	F	18	3	95825
938	19	1	5	978555994	Toy Story (1995)	Animation|Children's|Comedy	M	1	10	48073
...	...	...	...	...	...	...	...	...	...	...
994256	1025	3114	5	975002777	Toy Story 2 (1999)	Animation|Children's|Comedy	M	25	16	34677
994289	1898	3114	5	974699028	Toy Story 2 (1999)	Animation|Children's|Comedy	M	25	12	91101
994315	1970	3114	4	974686535	Toy Story 2 (1999)	Animation|Children's|Comedy	M	50	13	89052

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
994367	4741	3114	4	963267233	Toy Story 2 (1999)	Animation|Children's|Comedy	M	35	7	15203
994389	5713	3114	4	958512082	Toy Story 2 (1999)	Animation|Children's|Comedy	F	50	7	91362

3662 rows × 10 columns

In [27]:

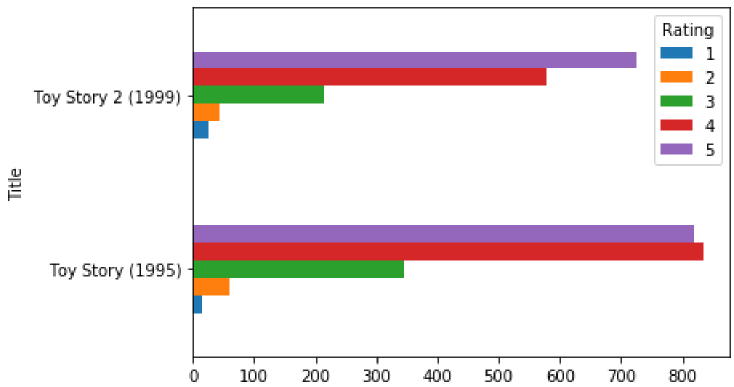
```
#User rating of the movie "Toy Story"
toystoryRating.groupby(["Title","Rating"]).size()
```

Out[27]:

```
Title               Rating
Toy Story (1995)    1          16
                    2          61
                    3         345
                    4         835
                    5         820
Toy Story 2 (1999)  1          25
                    2          44
                    3         214
                    4         578
                    5         724
dtype: int64
```

In [28]:

```
#Visual representations of User rating of the movie "Toy Story"
toystoryRating.groupby(["Title","Rating"]).size().unstack().plot(kind='barh',stacked=False,legend=
True)
plt.show()
```



It seems that lot of people rared 4 and 5 for the movie 'Toy Story (1995)' and 'Toy Story 2 (1999)'

3. Top 25 movies by viewership rating

In [29]:

```
#top_25 = print(finalDF.groupby('Title').Rating.count().nlargest(25))
top_25 = finalDF.groupby('Title').size().sort_values(ascending=False)[:25]
top_25
```

Out[29]:

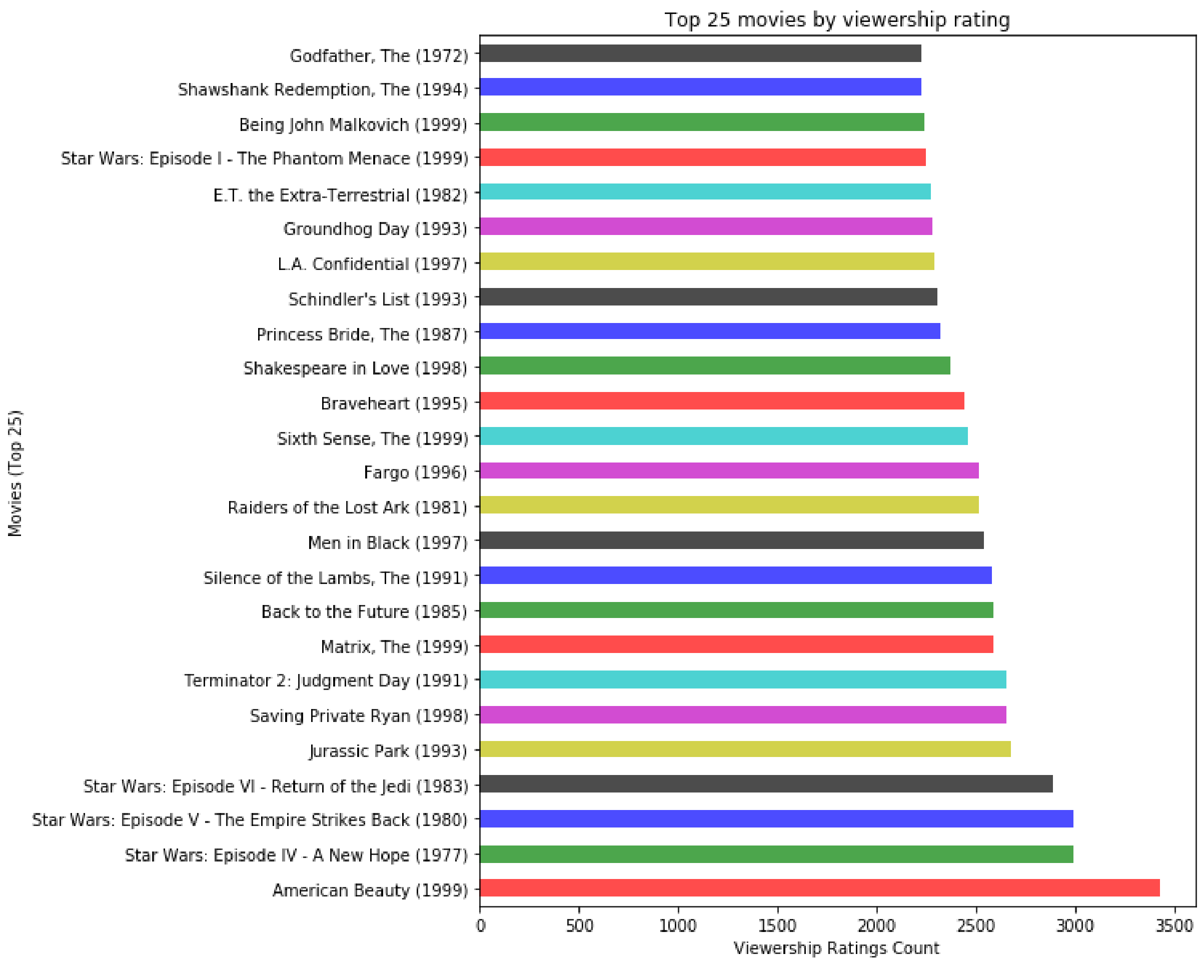
```
Title
American Beauty (1999)                                   3428
Star Wars: Episode IV - A New Hope (1977)                2991
Star Wars: Episode V - The Empire Strikes Back (1980)    2990
Star Wars: Episode VI - Return of the Jedi (1983)        2883
Jurassic Park (1993)                                     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
dtype: int64
```

In [30]:

```python
# Plotting Bar chart for Top 25 movies
#finalDF.groupby('Title').Rating.count().nlargest(25).plot(kind='barh')

top_25.plot(kind='barh', color=list('rgbkymc'), alpha=0.7, figsize=(8,10),stacked=False)
plt.xlabel("Viewership Ratings Count")
plt.ylabel("Movies (Top 25)")
plt.title("Top 25 movies by viewership rating")
plt.show()
```



In [ ]:

```python
#finalDF.groupby('Title').Rating.count().nsmallest(5)
```

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

In [31]:

```
# Subset the dataset wehere the UserId = 2696.
user_2696 = finalDF[finalDF['UserID']==2696]
user_2696
```

Out[31]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
953847	2696	1270	2	973308676	Back to the Future (1985)	Comedy|Sci-Fi	M	25	7	24210
953848	2696	1097	3	973308690	E.T. the Extra-Terrestrial (1982)	Children's|Drama|Fantasy|Sci-Fi	M	25	7	24210
953849	2696	1617	4	973308842	L.A. Confidential (1997)	Crime|Film-Noir|Mystery|Thriller	M	25	7	24210
953850	2696	800	5	973308842	Lone Star (1996)	Drama|Mystery	M	25	7	24210
953851	2696	3386	1	973308842	JFK (1991)	Drama|Mystery	M	25	7	24210
953852	2696	3176	4	973308865	Talented Mr. Ripley, The (1999)	Drama|Mystery|Thriller	M	25	7	24210
953853	2696	1711	4	973308904	Midnight in the Garden of Good and Evil (1997)	Comedy|Crime|Drama|Mystery	M	25	7	24210
953854	2696	1589	3	973308865	Cop Land (1997)	Crime|Drama|Mystery	M	25	7	24210
953855	2696	1783	4	973308865	Palmetto (1998)	Film-Noir|Mystery|Thriller	M	25	7	24210
953856	2696	1892	4	973308904	Perfect Murder, A (1998)	Mystery|Thriller	M	25	7	24210
953857	2696	1625	4	973308842	Game, The (1997)	Mystery|Thriller	M	25	7	24210
953858	2696	1644	2	973308920	I Know What You Did Last Summer (1997)	Horror|Mystery|Thriller	M	25	7	24210
953859	2696	1645	4	973308904	Devil's Advocate, The (1997)	Crime|Horror|Mystery|Thriller	M	25	7	24210
953860	2696	2389	4	973308710	Psycho (1998)	Crime|Horror|Thriller	M	25	7	24210
953861	2696	1805	4	973308886	Wild Things (1998)	Crime|Drama|Mystery|Thriller	M	25	7	24210
953862	2696	1092	4	973308886	Basic Instinct (1992)	Mystery|Thriller	M	25	7	24210
953863	2696	2713	1	973308710	Lake Placid (1999)	Horror|Thriller	M	25	7	24210
953864	2696	1258	4	973308710	Shining, The (1980)	Horror	M	25	7	24210
953865	2696	2338	2	973308920	I Still Know What You Did Last Summer (1998)	Horror|Mystery|Thriller	M	25	7	24210
953866	2696	350	3	973308886	Client, The (1994)	Drama|Mystery|Thriller	M	25	7	24210

In [32]:

```
user_2696.shape
```

Out[32]:

```
(20, 10)
```

In [33]:

```
#Find the ratings for all the movies reviewed by for a particular user of user id = 2696
#print(finalDF[finalDF.UserID == 2696].groupby('Rating')['MovieID'].count())
print(user_2696.groupby('Rating')['MovieID'].count())
```

```
Rating
1     2
2     3
3     3
4    11
5     1
Name: MovieID, dtype: int64
```

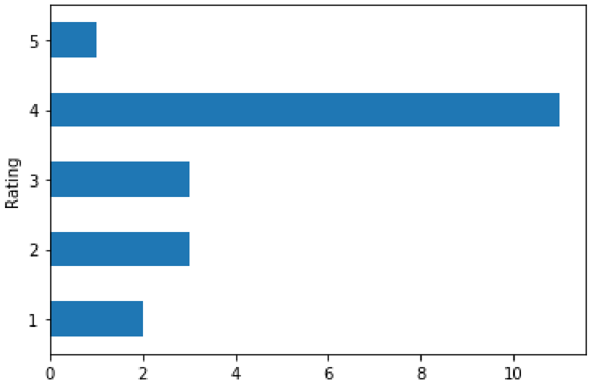
In [34]:

```
# Bar plot for above solution
```

```
# Bar plot for above solution
#finalDF[finalDF.UserID == 2696].groupby('Rating')['MovieID'].count().plot(kind='barh')
user_2696.groupby('Rating')['MovieID'].count().plot(kind='barh')
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2bac533dac8>
```

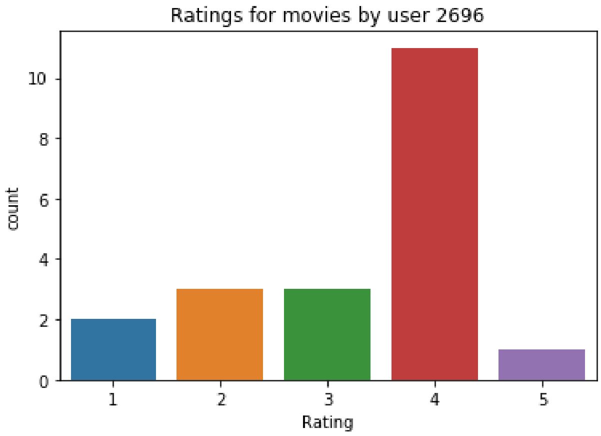


In [35]:

```
# Plotting the ratings given by the user 2696
sns.countplot(x='Rating',data = user_2696).set_title('Ratings for movies by user 2696')
```

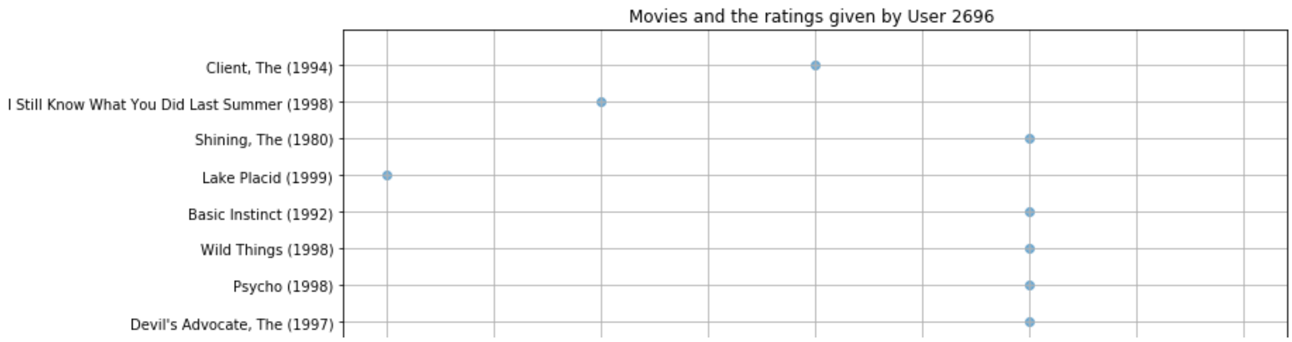
Out[35]:

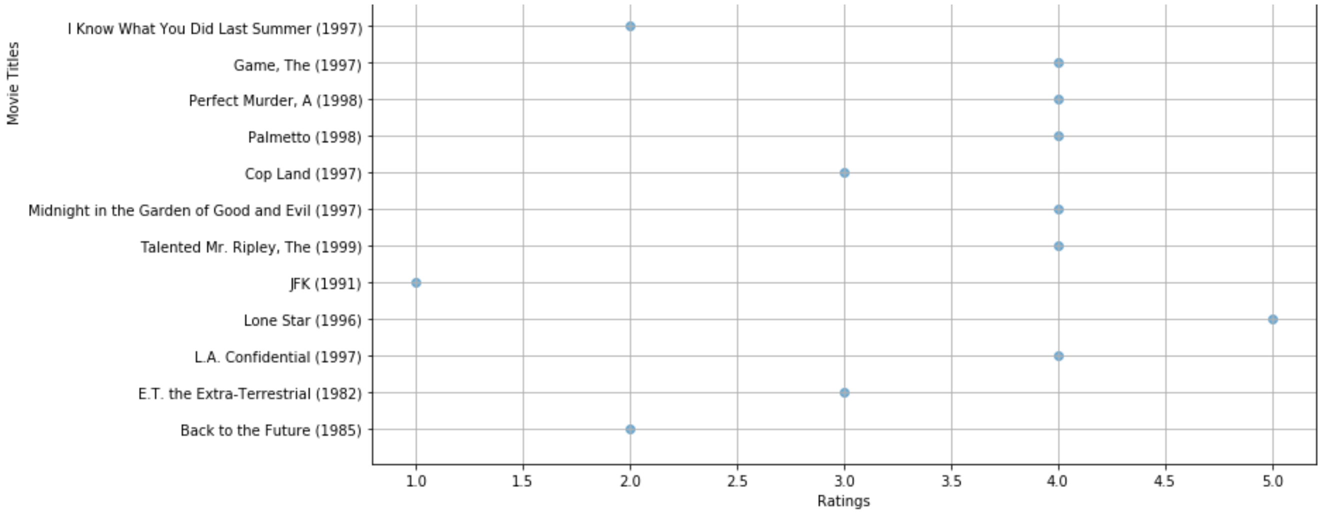
```
Text(0.5, 1.0, 'Ratings for movies by user 2696')
```



In [36]:

```
# Creating a scatter plot for the movies reviewed by the user 2696.
plt.figure(figsize=(12,10))
plt.scatter(user_2696.Rating,user_2696.Title, alpha=.55)
plt.title("Movies and the ratings given by User 2696 ")
plt.ylabel("Movie Titles")
plt.xlabel("Ratings")
plt.grid(b=True,which='major')
plt.show()
```





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)

In [38]:

```
finalDF.head()
```

Out[38]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F	1	10	48067
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F	1	10	48067
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F	1	10	48067
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F	1	10	48067
4	1	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy	F	1	10	48067

In [39]:

```
# split the data in column genre making a list
finalDF.Genres.str.split('|').tolist()[:5]
```

Out[39]:

```
[['Drama'],
 ['Animation', "Children's", 'Musical'],
 ['Musical', 'Romance'],
 ['Drama'],
 ['Animation', "Children's", 'Comedy']]
```

In [40]:

```
#Find out all the unique genres
list1 = finalDF.Genres.str.split('|').tolist()
finalList = []
for i in list1:
    for j in i:
        finalList.append(j)


list(set(finalList))
```

Out[40]:

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

In [41]:

```
finalDF.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 10 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 
 4   Title       1000209 non-null  object
 5   Genres      1000209 non-null  object
 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: 83.9+ MB
```

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.

In [42]:

```
#Create a separate column for each genre category with a one-hot encoding ( 1 and 0)
finalOHEDF = pd.concat([finalDF.Genres.str.get_dummies('|') , finalDF.iloc[:,[0,1,3,4,5,6,7,8,9]]] ,
axis=1)
```

In [43]:

```
finalOHEDF.head()
```

Out[43]:

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	...	Western	UserID	MovieID	Tin
0	0	0	0	0	0	0	0	1	0	0	...	0	1	1193	97
1	0	0	1	1	0	0	0	0	0	0	...	0	1	661	97
2	0	0	0	0	0	0	0	0	0	0	...	0	1	914	97

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	...	Western	UserID	MovieID	Tin
3	0	0	0	0	0	0	0	1	0	0	...	0	1	3408	97
4	0	0	1	1	1	0	0	0	0	0	...	0	1	2355	97

5 rows × 27 columns

3. Determine the features affecting the ratings of any particular movie.

In [44]:

```
#Determine the features affecting the ratings of any particular movie.
#
# Hint: Perform Chi-sqaure test between Xfeature v/s ratings ---- To do feature elimination
#
# and finalize your feature
#
# final numpy array called 'feature'

from scipy.stats import chi2_contingency

ctTitle = pd.crosstab(finalDF.Title,finalDF.Rating)
ctGenres = pd.crosstab(finalDF.Genres,finalDF.Rating)
ctGender = pd.crosstab(finalDF.Gender,finalDF.Rating)
ctAge = pd.crosstab(finalDF.Age,finalDF.Rating)
ctOccupation = pd.crosstab(finalDF.Occupation,finalDF.Rating)
ctZipCode = pd.crosstab(finalDF['Zip-code'],finalDF.Rating)
```

In [45]:

```
ctTitle.index.name
```

Out[45]:

```
'Title'
```

In [46]:

```
from scipy.stats import chi2_contingency

list1 = [ctTitle,ctGenres,ctGender,ctAge,ctOccupation,ctZipCode]

for i in list1:
    stat,pvalue,dof,expected_R = chi2_contingency(i)
    if pvalue <= 0.05:
        print("Alternate Hypothesis passed. {} and Rating have Relationship".format(i.index.name))
    else:
        print("Null hypothesis passed. {} and Profit doesnot have  Relationship".format(i.index.nam
e))
```

```
Alternate Hypothesis passed. Title and Rating have Relationship
Alternate Hypothesis passed. Genres and Rating have Relationship
Alternate Hypothesis passed. Gender and Rating have Relationship
Alternate Hypothesis passed. Age and Rating have Relationship
Alternate Hypothesis passed. Occupation and Rating have Relationship
Alternate Hypothesis passed. Zip-code and Rating have Relationship
```

In [48]:

```
finalDF.head()
```

Out[48]:

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	One Flew Over the Cuckoo's Nest (1975)	Drama	F	1	10	48067
1	1	661	3	978302109	James and the Giant Peach (1996)	Animation|Children's|Musical	F	1	10	48067

(1996)

	UserID	MovieID	Rating	Timestamp	Title	Genres	Gender	Age	Occupation	Zip-code
2	1	914	3	978301968	My Fair Lady (1964)	Musical|Romance	F	1	10	48067
3	1	3408	4	978300275	Erin Brockovich (2000)	Drama	F	1	10	48067
4	1	2355	5	978824291	Bug's Life, A (1998)	Animation|Children's|Comedy	F	1	10	48067

In [49]:

```
finalDF.groupby(['Title','Rating']).size()
```

Out[49]:

```
Title                  Rating
$1,000,000 Duck (1971) 1           3
                       2           8
                       3          15
                       4           7
                       5           4
                                 ...
eXistenZ (1999)        1          43
                       2          61
                       3         109
                       4         142
                       5          55
Length: 16912, dtype: int64
```

In [50]:

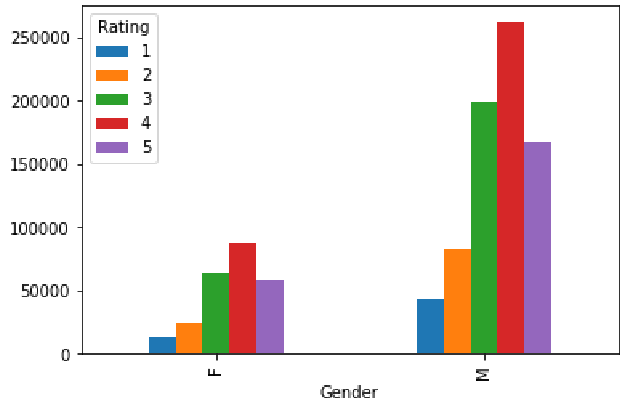
```
dfGenderAffecting = finalDF.groupby('Gender').size().sort_values(ascending=False)[:25]
dfGenderAffecting
```

Out[50]:

```
Gender
M    753769
F    246440
dtype: int64
```

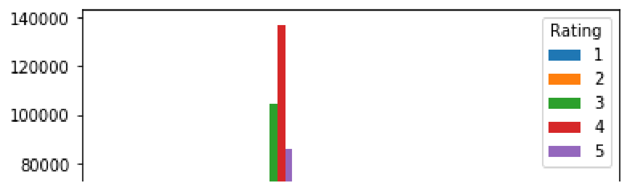
In [51]:

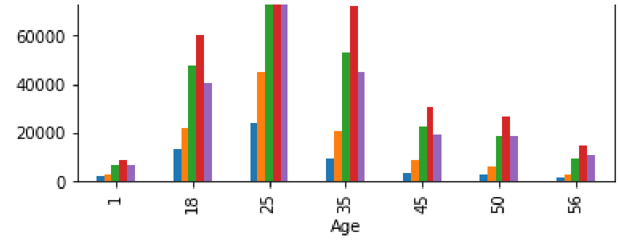
```
finalDF.groupby(['Gender','Rating']).size().unstack().plot(kind='bar',stacked=False,legend=True)
plt.show()
```



In [52]:

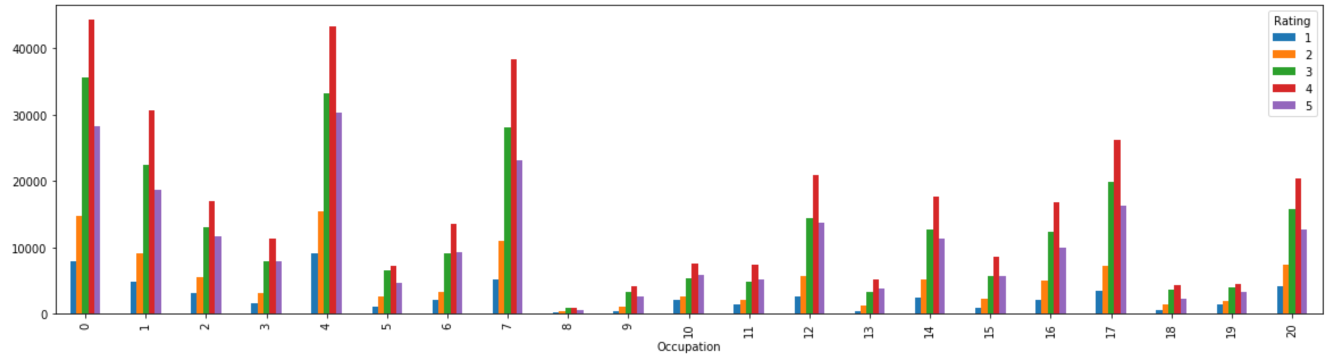
```
finalDF.groupby(["Age","Rating"]).size().unstack().plot(kind='bar',stacked=False,legend=True)
plt.show()
```





In [53]:

```
finalDF.groupby(["Occupation","Rating"]).size().unstack().plot(kind='bar',stacked=False,legend=Tru
e, figsize=(20,5))
plt.show()
```



4. Develop an appropriate model to predict the movie ratings

In [54]:

```
# using above found feature numpy array and rating as label

features = finalDF.iloc[:,[1,6,7,8]]
label = finalDF.Rating
```

In [55]:

```
features.head()
```

Out[55]:

	MovieID	Gender	Age	Occupation
0	1193	F	1	10
1	661	F	1	10
2	914	F	1	10
3	3408	F	1	10
4	2355	F	1	10

In [56]:

```
from sklearn.preprocessing import LabelEncoder

stateLabelEncoder = LabelEncoder()
features.iloc[:,0] = stateLabelEncoder.fit_transform(features.iloc[:,0])
features.iloc[:,1] = stateLabelEncoder.fit_transform(features.iloc[:,1])
```

from sklearn.preprocessing import LabelEncoder from sklearn.preprocessing import OneHotEncoder titleLe = LabelEncoder() genderLe = LabelEncoder() features[:,0] = titleLe.fit_transform(features[:,0]) features[:,1] = genderLe.fit_transform(features[:,1]) features ohe = OneHotEncoder(categorical_features=[0,1]) features = ohe.fit_transform(features).toarray()

In [57]:

```
features.head()
```

Out[57]:

	MovieID	Gender	Age	Occupation
0	1104	0	1	10
1	639	0	1	10
2	853	0	1	10
3	3177	0	1	10
4	2162	0	1	10

In [70]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(features,
                                                 label,
                                                 test_size=0.2,
                                                 random_state = 1)
```

In [71]:

```
# K Nearest Neighbors Classifier

from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors = 3)
knn_model.fit(X_train,y_train)
knn_predictions = knn_model.predict(X_test)
acc_knn = round(knn_model.score(X_train,y_train) * 100, 2)
acc_knn
```

Out[71]:

```
47.16
```

In [73]:

```
# creating a confusion matrix
from sklearn.metrics import confusion_matrix

cm_knn = confusion_matrix(y_test, knn_predictions)
```

In [74]:

```
knn_predictions
```

Out[74]:

```
array([5, 5, 4, ..., 1, 1, 3], dtype=int64)
```

In [75]:

```
cm_knn
```

Out[75]:

```
array([[ 3021,  2560,  2962,  1979,   708],
       [ 3773,  4659,  6466,  4901,  1681],
       [ 5861,  9404, 16237, 14733,  5964],
       [ 5190,  9688, 20280, 22353, 12206],
       [ 2402,  4719, 11112, 13939, 13244]], dtype=int64)
```

In [76]:

```
# Decision Tree
```

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train,y_train)
dt_predict= decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train,y_train) * 100, 2)
acc_decision_tree
```

Out[76]:

59.77

In [ ]:

```
# Random Forest

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train,y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train,y_train)
acc_random_forest = round(random_forest.score(X_train,y_train) * 100, 2)
acc_random_forest
```

In [78]:

```
# Logistic Regression

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, y_train) * 100, 2)
acc_log
```

Out[78]:

34.9

In [ ]:

```
# Support Vector Machines

svc = SVC()
svc.fit(X_train, y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, y_train) * 100, 2)
acc_svc
```

In [ ]:

```
# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, y_train) * 100, 2)
acc_gaussian
```

In [ ]:

```
# Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, y_train) * 100, 2)
acc_perceptron
```

In [ ]:

```
# Linear SVC

linear_svc = LinearSVC()
linear_svc.fit(X_train, y_train)
```

```
linear_svc.fit(X_train, y_train)
Y_pred = linear_svc.predict(X_test)
acc_linear_svc = round(linear_svc.score(X_train, y_train) * 100, 2)
acc_linear_svc
```

In [ ]:

```
# Stochastic Gradient Descent

sgd = SGDClassifier()
sgd.fit(X_train, y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train, y_train) * 100, 2)
acc_sgd
```

In [ ]:

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression', 'Random Forest', 'Naive Baye
s', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC', 'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron, acc_sgd, acc_linear_svc,
acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

In [ ]:

```

```

----- Thank You -----