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

In [1]:
```
# importing libraries for pandas and numpy
import pandas as pd
import numpy as np
```

Import the three datasets

In [2]:
```
movies = pd.read_csv('movies.dat',sep='::',header=None,engine='python',names=['MovieID','Title','Genres'])
# MovieID::Title::Genres
```

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

Out[3]:

	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 [4]:
```
rating = pd.read_csv('ratings.dat',sep='::',header=None,engine='python',names=['UserID','MovieID','Rating','Timestamp'])
#UserID::MovieID::Rating::Timestamp
```

In [5]:
```
rating.head()
```

Out[5]:

	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 [6]:
```
users = pd.read_csv('users.dat',sep='::',header=None,engine='python',names=['UserID','Gender','Age','Occupation','Zip-code'])
# UserID::Gender::Age::Occupation::Zip-code
```

In [7]:
```
users.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]:
```
# shape of the datasets

print(f'Movie:{movies.shape}, Rating:{rating.shape}, Users:{users.shape}')
```

```
Movie:(3883, 3), Rating:(1000209, 4), Users:(6040, 5)
```

Checking NaN is datasets

In [9]:
```
movies.isnull().sum()
```

Out[9]:
```
MovieID    0
Title      0
Genres     0
dtype: int64
```

In [10]:
```
rating.isnull().sum()
```

Out[10]:
```
UserID       0
MovieID      0
Rating       0
Timestamp    0
dtype: int64
```

In [11]:
```
users.isnull().sum()
```

Out[11]:
```
UserID        0
Gender        0
Age           0
Occupation    0
Zip-code      0
dtype: int64
```

In [12]:
```
# checking for duplicated values

print(f'Movie duplicates:{movies.duplicated().sum()}, Rating duplicates:{rating.duplicated().sum()}, User duplicates:{users.duplicated().sum()}')
```

```
Movie duplicates:0, Rating duplicates:0, User duplicates:0
```

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)

In [13]:
```
one = pd.merge(movies,rating,on='MovieID')
```

In [14]:
```
two = pd.merge(one,users,on='UserID')
```

In [15]:
```
two.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]:
```
two.shape
```

Out[16]:
```
(1000209, 10)
```

In [17]:
```
master_data = two.copy()
```

Master Data after merging two datasets

In [18]:
```
master_data.head()
```

Out[18]:

	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

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
```

In [19]:
```
# changing the columns into lower case

master_data.columns = master_data.columns.str.lower()
```

In [20]:
```
master_data['age']
```

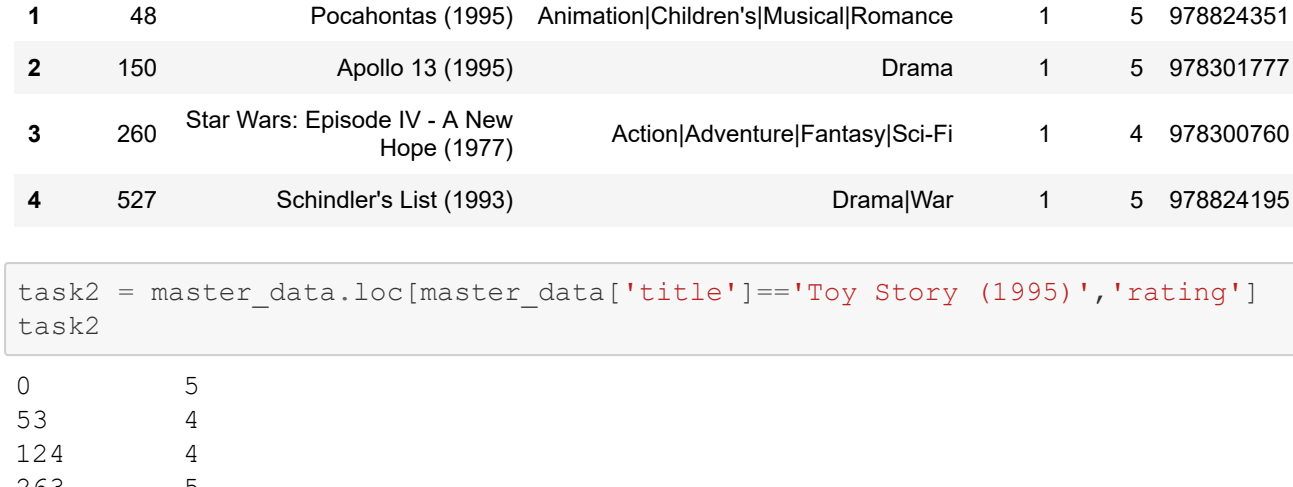
Out[20]:
```
0          1
1          1
2          1
3          1
4          1
          ..
1000204   25
1000205   25
1000206   25
1000207   25
1000208   25
Name: age, Length: 1000209, dtype: int64
```

In [21]:
```
pd.set_option('mode.chained_assignment', None)
import matplotlib.pyplot as plt
import seaborn as sns
```

```
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The text.latex.preview rcparam was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The mathtext.fallback_to_cm rcparam was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle: Support for setting the 'mathtext.fallback_to_cm' rcParam is deprecated since 3.3 and will be removed two minor releases later; use 'mathtext.fallback : 'cm' instead.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The validate_bool_maybe_none function was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The savefig.jpeg_quality rcparam was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The keymap.all_axes rcparam was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The animation.avconv_path rcparam was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
In C:\Users\Naveen\anaconda3\lib\site-packages\matplotlib\mpl-data\stylelib\_classic_test.mplstyle:
The animation.avconv_args rcparam was deprecated in Matplotlib 3.3 and will be removed two minor releases later.
```

User Age Distribution

In [22]:
```
plt.figure(figsize=(10,4))
sns.countplot(x='age',data=master_data,saturation=1)
plt.show()
```



Insights:

```
- From the countplot, it is seen that users of age around 25 has seen many movies
```

User rating of the movie "Toy Story"

In [23]:
```
master_data.head()
```

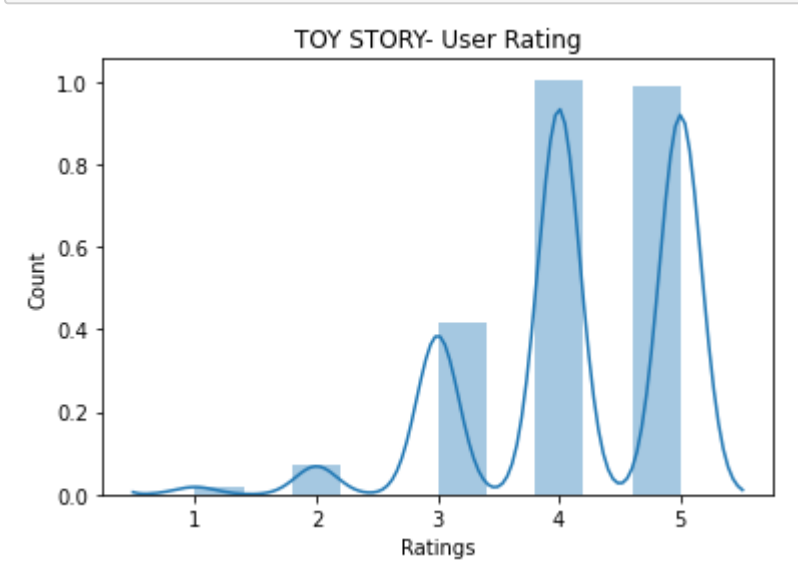
Out[23]:

	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 [24]:
```
task2 = master_data.loc[master_data['title']=='Toy Story (1995)','rating']
task2
```

Out[24]:
```
0         5
53        4
124       4
263       5
369       5
         ..
575166    5
575214    5
575485    4
575589    4
575869    3
Name: rating, Length: 2077, dtype: int64
```

In [25]:
```
sns.distplot(task2,bins=10)
plt.xlabel('Ratings')
plt.ylabel('Count')
plt.title('TOY STORY- User Rating')
plt.show()
```



Comments:

```
- 'Histogram' has been used above.
- 'Rating 4' is most given by user for the movie ToyStory
```

Top 25 movies by viewership rating

In [26]:
```
master_data.head()
```

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

In [27]:
```
task3 =master_data.groupby('title')['rating'].mean()
task3.head()
```

Out[27]:
```
title
$1,000,000 Duck (1971)           3.027027
'Night Mother (1986)             3.371429
'Til There Was You (1997)        2.692308
'burbs, The (1989)               2.910891
...And Justice for All (1979)    3.713568
Name: rating, dtype: float64
```

In [28]:
```
task3_25 = task3.sort_values(ascending=False)
```

In [29]:
```
task3_frame = task3_25.to_frame()[:25]
task3_frame
```

Out[29]:

title	rating
Gate of Heavenly Peace, The (1995)	5.000000
Lured (1947)	5.000000
Ulysses (Ulisse) (1954)	5.000000
Smashing Time (1967)	5.000000
Follow the Bitch (1998)	5.000000
Song of Freedom (1936)	5.000000
Bittersweet Motel (2000)	5.000000
Baby, The (1973)	5.000000
One Little Indian (1973)	5.000000
Schlafes Bruder (Brother of Sleep) (1995)	5.000000
I Am Cuba (Soy Cuba/Ya Kuba) (1964)	4.800000
Lamerica (1994)	4.750000
Apple, The (Sib) (1998)	4.666667
Sanjuro (1962)	4.608696
Seven Samurai (The Magnificent Seven) (Shichinin no samurai) (1954)	4.560510
Shawshank Redemption, The (1994)	4.554558
Godfather, The (1972)	4.524966
Close Shave, A (1995)	4.520548
Usual Suspects, The (1995)	4.517106
Schindler's List (1993)	4.510417
Wrong Trousers, The (1993)	4.507937
Dangerous Game (1993)	4.500000
Mamma Roma (1962)	4.500000
Inheritors, The (Die Siebtelbauern) (1998)	4.500000
Hour of the Pig, The (1993)	4.500000

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

In [30]:
```
task4 = master_data.loc[master_data['userid']==2696,['title','rating']]
task4.head(25)
```

Out[30]:

	title	rating
991035	Client, The (1994)	3
991036	Lone Star (1996)	5
991037	Basic Instinct (1992)	4
991038	E.T. the Extra-Terrestrial (1982)	3
991039	Shining, The (1980)	4
991040	Back to the Future (1985)	2
991041	Cop Land (1997)	3
991042	L.A. Confidential (1997)	4
991043	Game, The (1997)	4
991044	I Know What You Did Last Summer (1997)	2
991045	Devil's Advocate, The (1997)	4
991046	Midnight in the Garden of Good and Evil (1997)	4
991047	Palmetto (1998)	4
991048	Wild Things (1998)	4
991049	Perfect Murder, A (1998)	4
991050	I Still Know What You Did Last Summer (1998)	2
991051	Psycho (1998)	4
991052	Lake Placid (1999)	1
991053	Talented Mr. Ripley, The (1999)	4
991054	JFK (1991)	1

Feature Engineering

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 [31]:
```
un_genre = master_data['genres'].copy()
```

In [32]:
```
un_genre.duplicated().sum()
```

Out[32]:
```
999908
```

In [33]:
```
un_genre.drop_duplicates(keep='first')
```

Out[33]:
```
0                    Animation|Children's|Comedy
1            Animation|Children's|Musical|Romance
2                                          Drama
3                 Action|Adventure|Fantasy|Sci-Fi
4                                      Drama|War
                          ...
79242                            Romance|Western
80659                      Drama|Romance|Western
296412                Comedy|Film-Noir|Thriller
303913                          Film-Noir|Horror
920103                                   Fantasy
Name: genres, Length: 301, dtype: object
```

In [34]:
```
from itertools import combinations
from collections import Counter
```

In [35]:
```
count = Counter()
```

In [36]:
```
for row in un_genre:
    row_list = row.split('|')
    count.update(Counter(combinations(row_list,1)))
print(count)
```

```
Counter({('Comedy',): 356580, ('Drama',): 354529, ('Action',): 257457, ('Thriller',): 189680, ('Sci-Fi',): 157294, ('Romance',): 147523, ('Adventure',): 133953, ('Crime',): 79541, ('Horror',): 76386, ("Children's",): 72186, ('War',): 68527, ('Animation',): 43293, ('Musical',): 41533, ('Mystery',): 40178, ('Fantasy',): 36301, ('Western',): 20683, ('Film-Noir',): 18261, ('Documentary',): 7910})
```

In [37]:
```
ms= count.most_common()
```

In [38]:
```
for key,value in count.most_common():
    print(key)
```

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

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 [39]:
```
dummy = master_data['genres'].str.get_dummies()
dummy
```

Out[39]:

	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	Fantasy	Film-Noir	Horror	Musical	Mystery	Ro
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

Determine the features affecting the ratings of any particular movie.

In [40]:
```
master_data.head()
```

Out[40]:

	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 [41]:
```
master_data.dtypes
```

Out[41]:
```
movieid        int64
title         object
genres        object
userid         int64
rating         int64
timestamp      int64
gender        object
age            int64
occupation     int64
zip-code      object
dtype: object
```

In [42]:
```
new_df = pd.merge(master_data,dummy,on=master_data.index)
new_df
```

Out[42]:

	key_0	movieid	title	genres	userid	rating	timestamp	gender	age	occupation	...	Fan
0	0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10	...	
1	1	48	Pocahontas (1995)	Animation|Children's|Musical|Romance	1	5	978824351	F	1	10	...	
2	2	150	Apollo 13 (1995)	Drama	1	5	978301777	F	1	10	...	
3	3	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Fantasy|Sci-Fi	1	4	978300760	F	1	10	...	
4	4	527	Schindler's List (1993)	Drama|War	1	5	978824195	F	1	10	...	
...	...	...	...	...	...	...	...	...	...	...	...	
1000204	1000204	3513	Rules of Engagement (2000)	Drama|Thriller	5727	4	958489970	M	25	4	...	
1000205	1000205	3535	American Psycho (2000)	Comedy|Horror|Thriller	5727	2	958489970	M	25	4	...	
1000206	1000206	3536	Keeping the Faith (2000)	Comedy|Romance	5727	5	958489902	M	25	4	...	
1000207	1000207	3555	U-571 (2000)	Action|Thriller	5727	3	958490699	M	25	4	...	
1000208	1000208	3578	Gladiator (2000)	Action|Drama	5727	5	958490171	M	25	4	...	

1000209 rows × 29 columns

In [43]:
```
new_df.drop(columns='key_0',inplace=True)
```

In [44]:
```
new_df.columns
```

Out[44]:
```
Index(['movieid', 'title', 'genres', 'userid', 'rating', 'timestamp', 'gender',
       'age', 'occupation', 'zip-code', 'Action', 'Adventure', 'Animation',
       'Children's', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Fantasy',
       'Film-Noir', 'Horror', 'Musical', 'Mystery', 'Romance', 'Sci-Fi',
       'Thriller', 'War', 'Western'],
      dtype='object')
```

In [45]:
```
new_df.dtypes
```

Out[45]:
```
movieid         int64
title          object
genres         object
userid          int64
rating          int64
timestamp       int64
gender         object
age             int64
occupation      int64
zip-code       object
Action          int64
Adventure       int64
Animation       int64
Children's      int64
Comedy          int64
Crime           int64
Documentary     int64
Drama           int64
Fantasy         int64
Film-Noir       int64
Horror          int64
Musical         int64
Mystery         int64
Romance         int64
Sci-Fi          int64
Thriller        int64
War             int64
Western         int64
dtype: object
```

In [46]:
```
feature_affecting  =new_df.copy()
```

In [47]:
```
feature_affecting.drop(columns=['title','genres','zip-code'],inplace=True)
```

In [48]:
```
feature_affecting['gender'].replace('F',0,inplace=True)
```

In [49]:
```
feature_affecting['gender'].replace('M',1,inplace=True)
```

In [50]:
```
feature_affecting.head()
```

Out[50]:

	movieid	userid	rating	timestamp	gender	age	occupation	Action	Adventure	Animation	...	Fantasy	Film-Noir	Horror	Musical	Mys
0	1	1	5	978824268	0	1	10	0	0	1	...	0	0	0	0	
1	48	1	5	978824351	0	1	10	0	0	1	...	0	0	0	1	
2	150	1	5	978301777	0	1	10	0	0	0	...	0	0	0	0	
3	260	1	4	978300760	0	1	10	1	1	0	...	1	0	0	0	
4	527	1	5	978824195	0	1	10	0	0	0	...	0	0	0	0	

5 rows × 25 columns

In [51]:
```
from statsmodels.formula.api import ols
```

In [52]:
```
feature_affecting.rename(columns={"Children's" : 'Childrens'},inplace=True)
```

In [53]:
```
feature_affecting.rename(columns={'Sci-Fi':'SciFi'},inplace=True)
```

In [54]:
```
feature_affecting.rename(columns={'Film-Noir':'FilmNoir'},inplace=True)
```

In [55]:
```
from sklearn.model_selection import train_test_split as split
from sklearn.linear_model import LinearRegression
```

In [56]:
```
lr= LinearRegression()
```

In [57]:
```
train,test = split(feature_affecting,test_size=0.30,random_state = 12)
```

In [58]:
```
x_train = train.drop(columns='rating')
y_train = train['rating']
```

In [59]:
```
lr.fit(x_train,y_train)
```

Out[59]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [60]:
```
feature_affecting.columns
```

Out[60]:
```
Index(['movieid', 'userid', 'rating', 'timestamp', 'gender', 'age',
       'occupation', 'Action', 'Adventure', 'Animation', 'Childrens', 'Comedy',
       'Crime', 'Documentary', 'Drama', 'Fantasy', 'FilmNoir', 'Horror',
       'Musical', 'Mystery', 'Romance', 'SciFi', 'Thriller', 'War', 'Western'],
      dtype='object')
```

In [61]:
```
ft = '+'.join(train.drop(columns=['rating','timestamp']).columns)
mod = ols('rating~'+ft,data=train).fit()
```

In [62]:
```
mod.summary()
```

Out[62]:

OLS Regression Results

Dep. Variable:	rating	R-squared:	0.040
Model:	OLS	Adj. R-squared:	0.040
Method:	Least Squares	F-statistic:	1281.
Date:	Fri, 04 Sep 2020	Prob (F-statistic):	0.00
Time:	23:10:43	Log-Likelihood:	-1.0568e+06
No. Observations:	700146	AIC:	2.114e+06
Df Residuals:	700122	BIC:	2.114e+06
Df Model:	23		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	3.4981	0.007	523.862	0.000	3.485	3.511
movieid	-5.542e-05	1.23e-06	-45.034	0.000	-5.78e-05	-5.3e-05
userid	5.377e-06	7.58e-07	7.094	0.000	3.89e-06	6.86e-06
gender	-0.0343	0.003	-11.106	0.000	-0.040	-0.028
age	0.0041	0.000	36.718	0.000	0.004	0.004
occupation	0.0010	0.000	4.947	0.000	0.001	0.001
Action	-0.0951	0.004	-25.387	0.000	-0.102	-0.088
Adventure	-0.0019	0.004	-0.424	0.672	-0.011	0.007
Animation	0.3838	0.008	46.499	0.000	0.368	0.400
Childrens	-0.3516	0.007	-51.816	0.000	-0.365	-0.338
Comedy	-0.0086	0.003	-2.500	0.012	-0.015	-0.002
Crime	0.0823	0.005	16.220	0.000	0.072	0.092
Documentary	0.3792	0.015	25.053	0.000	0.350	0.409
Drama	0.2222	0.003	63.503	0.000	0.215	0.229
Fantasy	0.0752	0.008	9.838	0.000	0.060	0.090
FilmNoir	0.4239	0.010	40.762	0.000	0.403	0.444
Horror	-0.2860	0.005	-53.690	0.000	-0.296	-0.276
Musical	0.1389	0.007	19.426	0.000	0.125	0.153
Mystery	0.0054	0.007	0.763	0.446	-0.008	0.019
Romance	-0.0348	0.004	-8.979	0.000	-0.042	-0.027
SciFi	-0.0287	0.004	-7.089	0.000	-0.037	-0.021
Thriller	0.0448	0.004	11.504	0.000	0.037	0.052
War	0.2714	0.005	50.008	0.000	0.261	0.282
Western	0.1036	0.009	11.045	0.000	0.085	0.122

Omnibus:	32561.184	Durbin-Watson:	2.000
Prob(Omnibus):	0.000	Jarque-Bera (JB):	34801.819
Skew:	-0.522	Prob(JB):	0.00
Kurtosis:	2.677	Cond. No.	4.54e+04

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.54e+04. This might indicate that there are strong multicollinearity or other numerical problems.

In [ ]:

In [63]:
```
from sklearn import metrics
import numpy as np
from math import sqrt
```

In [64]:
```
x_test = test.drop(columns='rating')
y_test = test['rating']
```

In [65]:
```
y_pred = lr.predict(x_test)
```

In [66]:
```
np.sqrt(metrics.mean_squared_error(y_test,y_pred))
```

Out[66]:
```
1.092131762701962
```

In [67]:
```
metrics.mean_squared_error(y_test,y_pred)
```

Out[67]:
```
1.1927517871024944
```

Features Affecting

In [68]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
```

In [69]:
```
from sklearn.tree import DecisionTreeClassifier, export_graphviz
```

In [70]:
```
train_gc , test_gc = split(feature_affecting,test_size = 0.25, random_state = 12 )
```

In [71]:
```
dt = DecisionTreeClassifier()
```

In [72]:
```
x_dc = train_gc.drop(columns='rating')
y_dc = train_gc.rating
```

In [73]:
```
bestfeatures = SelectKBest(score_func=chi2, k=10)
fit = bestfeatures.fit(x_dc,y_dc)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x_dc.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Features','Score']  #naming the dataframe columns
print(featureScores.nlargest(10,'Score'))
```

```
     Features         Score
2   timestamp  1.243374e+08
0     movieid  2.436390e+06
1      userid  1.302944e+05
4         age  1.379895e+04
13      Drama  7.192148e+03
16     Horror  6.917901e+03
22        War  4.779410e+03
15   FilmNoir  2.873781e+03
20      SciFi  1.388519e+03
9   Childrens  1.227259e+03
```

Conclusion:

```
- Above are the constructed top 10 features affecting
```

Develop an appropriate model to predict the movie ratings

In [74]:
```python
feature_affecting.head()
```

Out[74]:

	movieid	userid	rating	timestamp	gender	age	occupation	Action	Adventure	Animation	...	Fantasy	FilmNoir	Horror	Musical
0	1	1	5	978824268	0	1	10	0	0	1	...	0	0	0	0
1	48	1	5	978824351	0	1	10	0	0	1	...	0	0	0	1
2	150	1	5	978301777	0	1	10	0	0	0	...	0	0	0	0
3	260	1	4	978300760	0	1	10	1	1	0	...	1	0	0	0
4	527	1	5	978824195	0	1	10	0	0	0	...	0	0	0	0

5 rows × 25 columns

Decision Tree Classifier

In [75]:
```python
dt.fit(train_gc.drop('rating', axis = 1), train_gc.rating)
```

Out[75]:
```
DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                       random_state=None, splitter='best')
```

In [76]:
```python
dt.predict(test_gc.drop('rating', axis = 1))
```

Out[76]:
```
array([3, 5, 3, ..., 3, 5, 4], dtype=int64)
```

In [77]:
```python
dt.predict_proba(test_gc.drop('rating', axis = 1))
```

Out[77]:
```
array([[0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 1., 0., 0.],
       ...,
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0.]])
```

In [78]:
```python
dt.score(x_dc,y_dc)
```

Out[78]:
```
1.0
```

In [79]:
```python
x_test  = test_gc.drop(columns='rating')
y_test = test_gc.rating
```

In [80]:
```python
y_pred = dt.predict(x_test)
```

In [81]:
```python
from sklearn import metrics
```

In [82]:
```python
metrics.accuracy_score(y_test,y_pred)
```

Out[82]:
```
0.31946027442182257
```

Insights:

```
- Using DecisionTreeClassifier here, we have obtained 31% accuracy from the model
```

KNN

In [83]:
```python
from sklearn.neighbors import KNeighborsClassifier
```

In [84]:
```python
knn = KNeighborsClassifier(n_neighbors = 7).fit(x_dc, y_dc)
knnaccuracy = knn.score(x_dc, y_dc)
knn_predictions = knn.predict(x_test)
print("KNN accuracy: {:.4f}".format(knnaccuracy))
```

```
KNN accuracy: 0.5230
```

Conclusion:

```
- From KNN we got 52% accuracy_score
```

Gaussian Naive Bayes

In [85]:
```python
from sklearn.naive_bayes import GaussianNB
```

In [86]:
```python
gnb = GaussianNB().fit(x_dc, y_dc)
gnbaccuracy = gnb.score(x_dc, y_dc)
gnb_predictions = gnb.predict(x_test)
print("GNB accuracy: {:.4f}".format(gnbaccuracy))
```

```
GNB accuracy: 0.3479
```

Here we got 34% score

In [ ]: