Importing required python libraries

In [1]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import seaborn as sns
import re
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from lightgbm import LGBMClassifier
import xgboost
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
```

Supressing warnings

In [2]:
```
import warnings
warnings.filterwarnings('ignore')
```

Importing the given three datasets

In [3]:
```
rating_columns = ['UserID','MovieID','Rating','Timestamp']
rating_df = pd.read_csv('D:\\AI Engineer Masters\\Project 01\\ratings.dat',header=None,
rating_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]:
```
user_columns = ['UserID','Gender','Age','Occupation','Zip-code']
user_df = pd.read_csv('D:\\AI Engineer Masters\\Project 01\\users.dat',header=None,deli
user_df.head()
```

Out[4]:

	UserID	Gender	Age	Occupation	Zip-code
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

In [5]:
```
movie_columns = ['MovieID','Title','Genres']
movie_df = pd.read_csv('D:\\AI Engineer Masters\\Project 01\\movies.dat',header=None,de
movie_df.head()
```

Out[5]:

	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

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

In [6]:
```
#Merge two tables at a time using two primary keys MovieID & UserId
master_df=user_df.merge(rating_df,how='outer',on='UserID')
master_df=master_df.merge(movie_df,how='outer',on='MovieID')
#Drop unwanted columns
master_df.drop(['Timestamp','Zip-code'],axis=1,inplace=True)
master_df.head()
```

Out[6]:

	UserID	Gender	Age	Occupation	MovieID	Rating	Title	Genres
0	1.0	F	1.0	10.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	Drama
1	2.0	M	56.0	16.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	Drama
2	12.0	M	25.0	12.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	Drama
3	15.0	M	25.0	7.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	Drama
4	17.0	M	50.0	1.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	Drama

Cleaning the dataset by removing nulls

In [7]:
```
#Finding count of null values for Ratings
print('Rows with missing Ratings:', master_df['Rating'].isnull().values.sum())
#Dropping the nulls
master_df.dropna(subset=['Rating'],axis=0,inplace=True)
print('Rows with missing Ratings:', master_df['Rating'].isnull().values.sum())
```

```
Rows with missing Ratings: 177
Rows with missing Ratings: 0
```

Exploring the dataset further

In [8]:
```
# checking number of values and type of columns
master_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 8 columns):
 #   Column      Non-Null Count    Dtype  
---  ------      --------------    -----  
 0   UserID      1000209 non-null  float64
 1   Gender      1000209 non-null  object 
 2   Age         1000209 non-null  float64
 3   Occupation  1000209 non-null  float64
 4   MovieID     1000209 non-null  int64  
 5   Rating      1000209 non-null  float64
 6   Title       1000209 non-null  object 
 7   Genres      1000209 non-null  object 
dtypes: float64(4), int64(1), object(3)
memory usage: 68.7+ MB
```

In [9]:
```
# Getting stas of the numeric fields
master_df.describe().T
```

Out[9]:

	count	mean	std	min	25%	50%	75%	max
UserID	1000209.0	3024.512348	1728.412695	1.0	1506.0	3070.0	4476.0	6040.0
Age	1000209.0	29.738314	11.751983	1.0	25.0	25.0	35.0	56.0
Occupation	1000209.0	8.036138	6.531336	0.0	2.0	7.0	14.0	20.0
MovieID	1000209.0	1865.539898	1096.040689	1.0	1030.0	1835.0	2770.0	3952.0
Rating	1000209.0	3.581564	1.117102	1.0	3.0	4.0	4.0	5.0

In [10]:
```
#Seeing correlation of numeric fields
master_df.corr()
```

Out[10]:

	UserID	Age	Occupation	MovieID	Rating
UserID	1.000000	0.034688	-0.026698	-0.017739	0.012303
Age	0.034688	1.000000	0.078371	0.027575	0.056869
Occupation	-0.026698	0.078371	1.000000	0.008585	0.006753
MovieID	-0.017739	0.027575	0.008585	1.000000	-0.064042
Rating	0.012303	0.056869	0.006753	-0.064042	1.000000

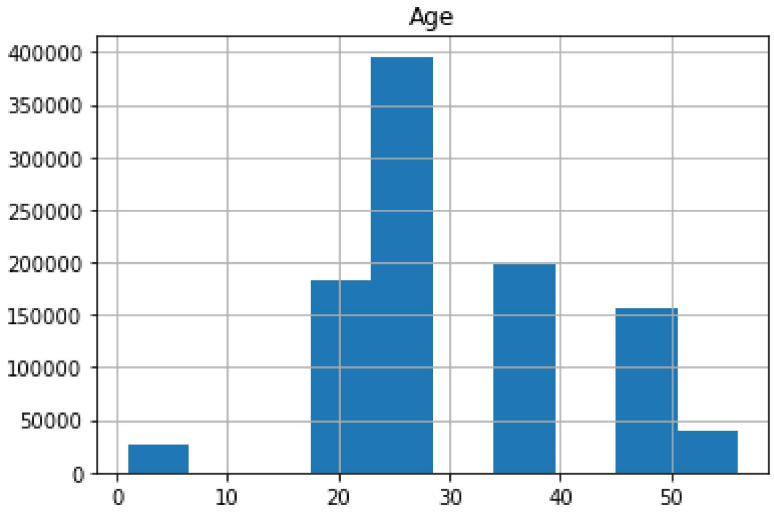
In [11]:
```
# Visualizing correlations
sns.heatmap(master_df.corr(),annot= True,linewidths=0.5)
```

Out[11]: <AxesSubplot:>

1.User Age Distribution

In [12]:
```
master_df.hist('Age')
```

Out[12]:
```
array([[<AxesSubplot:title={'center':'Age'}>]], dtype=object)
```



2.User rating of the movie “Toy Story”

In [13]:
```
#Note: Using ( in search to avoid the listing of Toy Story 2
master_df[master_df.Title.str.lower().str.contains('toy story \(')][['Title','Rating']]
```

Out[13]:

	Title	Rating
0	Toy Story (1995)	5.0
1	Toy Story (1995)	4.0
2	Toy Story (1995)	4.0
3	Toy Story (1995)	5.0
4	Toy Story (1995)	5.0
...	...	...
2072	Toy Story (1995)	5.0
2073	Toy Story (1995)	5.0

	Title	Rating
2074	Toy Story (1995)	4.0
2075	Toy Story (1995)	4.0
2076	Toy Story (1995)	3.0

2077 rows × 2 columns

3. Top 25 movies by viewership rating

In [14]:
```
# This will give the top 25 movies which are rated by the viewers in terms of ratings c
master_df.groupby('Title')[['Rating']].agg('count').sort_values(by='Rating',ascending=F
```

Out[14]:

Title	Rating
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

	Rating
Title	
Shawshank Redemption, The (1994)	2227
Godfather, The (1972)	2223

In [15]:
```
# The above ranking of movies is not actually fair as it is not taking care of the give
# So, it is advisable to do a more sophisticated ranking by the use of Bayesian adjuste
# Bayes Mean = (product_ratings_count * product_ratings_average + m * C) / (product_rat
# where m is the arithmetic mean of all the product's rating values and C is the 25 per

#caluclating Bayes constants
C=master_df.value_counts(['Title']).quantile(.25)
print('C =',C)
m=master_df['Rating'].mean()
print('m =',m)
```

```
C = 33.0
m = 3.581564453029317
```

In [16]:
```
#preparing a dataframe with ratings count & avergae for each movie
top_rating_df=master_df.groupby(['Title']).Rating.agg(['mean','count']).reset_index()
top_rating_df
```

Out[16]:

	Title	mean	count
0	$1,000,000 Duck (1971)	3.027027	37
1	'Night Mother (1986)	3.371429	70
2	'Til There Was You (1997)	2.692308	52
3	'burbs, The (1989)	2.910891	303
4	...And Justice for All (1979)	3.713568	199
...	...	...	...
3701	Zed & Two Noughts, A (1985)	3.413793	29
3702	Zero Effect (1998)	3.750831	301
3703	Zero Kelvin (Kjærlighetens kjøtere) (1995)	3.500000	2
3704	Zeus and Roxanne (1997)	2.521739	23
3705	eXistenZ (1999)	3.256098	410

3706 rows × 3 columns

In [17]:
```
# Preapring bayes mean as a new column = (product_ratings_count * product_ratings_avera
top_rating_df['Bayes Mean'] = (top_rating_df['count'] * top_rating_df['mean'] + m * C)
top_rating_df
```

Out[17]:

	Title	mean	count	Bayes Mean
0	$1,000,000 Duck (1971)	3.027027	37	3.288452
1	'Night Mother (1986)	3.371429	70	3.438754

	Title	mean	count	Bayes Mean
2	'Til There Was You (1997)	2.692308	52	3.037549
3	'burbs, The (1989)	2.910891	303	2.976761
4	...And Justice for All (1979)	3.713568	199	3.694791
...	...	...	...	...
3701	Zed & Two Noughts, A (1985)	3.413793	29	3.503091
3702	Zero Effect (1998)	3.750831	301	3.734107
3703	Zero Kelvin (Kjærlighetens kjøtere) (1995)	3.500000	2	3.576904
3704	Zeus and Roxanne (1997)	2.521739	23	3.146279
3705	eXistenZ (1999)	3.256098	410	3.280342

3706 rows × 4 columns

In [18]:
```
#This gives the top 25 movies sorted by Bayes adjusted ranking.
top_rating_df.sort_values(by=['Bayes Mean'],ascending=False).reset_index(drop=True)[:25
```

Out[18]:

	Title	mean	count	Bayes Mean
0	Shawshank Redemption, The (1994)	4.554558	2227	4.540350
1	Seven Samurai (The Magnificent Seven) (Shichin...	4.560510	628	4.511636
2	Godfather, The (1972)	4.524966	2223	4.511167
3	Usual Suspects, The (1995)	4.517106	1783	4.500106
4	Schindler's List (1993)	4.510417	2304	4.497301
5	Close Shave, A (1995)	4.520548	657	4.475640
6	Wrong Trousers, The (1993)	4.507937	882	4.474526
7	Raiders of the Lost Ark (1981)	4.477725	2514	4.466114
8	Rear Window (1954)	4.476190	1050	4.448930
9	Star Wars: Episode IV - A New Hope (1977)	4.453694	2991	4.444177
10	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	4.491489	470	4.431792
11	Dr. Strangelove or: How I Learned to Stop Worr...	4.449890	1367	4.429423
12	Casablanca (1942)	4.412822	1669	4.396705
13	To Kill a Mockingbird (1962)	4.425647	928	4.396661
14	Third Man, The (1949)	4.452083	480	4.396085
15	Sixth Sense, The (1999)	4.406263	2459	4.395342
16	One Flew Over the Cuckoo's Nest (1975)	4.390725	1725	4.375536
17	Maltese Falcon, The (1941)	4.395973	1043	4.370996
18	Lawrence of Arabia (1962)	4.401925	831	4.370592
19	Double Indemnity (1944)	4.415608	551	4.368479

	Title	mean	count	Bayes Mean
20	Wallace & Gromit: The Best of Aardman Animatio...	4.426941	438	4.367710
21	Citizen Kane (1941)	4.388889	1116	4.365702
22	North by Northwest (1959)	4.384030	1315	4.364385
23	Paths of Glory (1957)	4.473913	230	4.361945
24	Bridge on the River Kwai, The (1957)	4.386994	938	4.359621

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

In [19]:
```
master_df[master_df.UserID==2696][['UserID','Title','Rating']].reset_index(drop=True)
```

Out[19]:

	UserID	Title	Rating
0	2696.0	Back to the Future (1985)	2.0
1	2696.0	E.T. the Extra-Terrestrial (1982)	3.0
2	2696.0	L.A. Confidential (1997)	4.0
3	2696.0	Lone Star (1996)	5.0
4	2696.0	JFK (1991)	1.0
5	2696.0	Talented Mr. Ripley, The (1999)	4.0
6	2696.0	Midnight in the Garden of Good and Evil (1997)	4.0
7	2696.0	Cop Land (1997)	3.0
8	2696.0	Palmetto (1998)	4.0
9	2696.0	Perfect Murder, A (1998)	4.0
10	2696.0	Game, The (1997)	4.0
11	2696.0	I Know What You Did Last Summer (1997)	2.0
12	2696.0	Devil's Advocate, The (1997)	4.0
13	2696.0	Psycho (1998)	4.0
14	2696.0	Wild Things (1998)	4.0
15	2696.0	Basic Instinct (1992)	4.0
16	2696.0	Lake Placid (1999)	1.0
17	2696.0	Shining, The (1980)	4.0
18	2696.0	I Still Know What You Did Last Summer (1998)	2.0
19	2696.0	Client, The (1994)	3.0

Feature Engineering

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 [20]:
```
# Getting the genres split into a list of lists
genres_list_of_list=master_df.Genres.str.split('|').tolist()
# Flattenign the list of lists
genres_list=list(np.concatenate(genres_list_of_list))
# Removing dups
genres_list_unique=set(genres_list)
print('Number of unique Genres :',len(genres_list_unique))
genres_list_unique
```

```
Number of unique Genres : 18
```

Out[20]:
```
{'Action',
 'Adventure',
 'Animation',
 "Children's",
 'Comedy',
 'Crime',
 'Documentary',
 'Drama',
 'Fantasy',
 'Film-Noir',
 'Horror',
 'Musical',
 'Mystery',
 'Romance',
 'Sci-Fi',
 'Thriller',
 'War',
 'Western'}
```

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 [21]:
```
# pandas get_dummies will give us a one hot encoding result easily.
# It has '|' as the default separator and we do not need to do any separate split proce
# drop function is used to remove the original Genres column as it is no logner needed
# concat function is used to concatenate both the dataframes into one.
genre_one_hot_df=pd.concat([master_df.drop(['Genres'],axis=1),master_df.Genres.str.get_
genre_one_hot_df.head()
```

Out[21]:

	UserID	Gender	Age	Occupation	MovieID	Rating	Title	Action	Adventure	Animation	...	Fa
0	1.0	F	1.0	10.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	...	
1	2.0	M	56.0	16.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	...	

	UserID	Gender	Age	Occupation	MovieID	Rating	Title	Action	Adventure	Animation	...	F
2	12.0	M	25.0	12.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	...	
3	15.0	M	25.0	7.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	...	
4	17.0	M	50.0	1.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	...	

5 rows × 25 columns

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

In [22]:
```
print('Features of original data:', list(master_df.columns))
# Obviously UserID, MovieID & Title shouldn't be affecting the Rating.
# Even though the title will not affect, the year of the movie may affect the ratings.
# So, it will be good to extract it from title and use.
print('Features that affects:', ['Gender', 'Age', 'Occupation','Genres', 'Year'])
```

```
Features of original data: ['UserID', 'Gender', 'Age', 'Occupation', 'MovieID', 'Ratin
g', 'Title', 'Genres']
Features that affects: ['Gender', 'Age', 'Occupation', 'Genres', 'Year']
```

In [23]:
```
# Creating new column for year by extracting digits enclosed with () and takes just the
genre_one_hot_df['Year']=genre_one_hot_df.Title.str.findall(r'(\d+)').str[0]
genre_one_hot_df['Year']=genre_one_hot_df['Year'].astype('int')
genre_one_hot_df[['Title','Year']].head()
```

Out[23]:

	Title	Year
0	One Flew Over the Cuckoo's Nest (1975)	1975
1	One Flew Over the Cuckoo's Nest (1975)	1975
2	One Flew Over the Cuckoo's Nest (1975)	1975
3	One Flew Over the Cuckoo's Nest (1975)	1975
4	One Flew Over the Cuckoo's Nest (1975)	1975

In [24]:
```
# The Gender is  a character column and will not do good in a model. So btter to conver
refined_df=pd.concat([genre_one_hot_df.drop(['Gender'],axis=1),genre_one_hot_df.Gender.
refined_df.head()
```

Out[24]:

	UserID	Age	Occupation	MovieID	Rating	Title	Action	Adventure	Animation	Children's	...
0	1.0	1.0	10.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	...
1	2.0	56.0	16.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	...
2	12.0	25.0	12.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	...
3	15.0	25.0	7.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	...
4	17.0	50.0	1.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	...

5 rows × 27 columns

In [25]:
```
# Even though the occupation field is numeric, it has a ordinal relationship now which
# So, changing it to nominal by a one hot encoding
refined_df=pd.concat([refined_df.drop(['Occupation'],axis=1),pd.get_dummies(refined_df.
refined_df.head()
```

Out[25]:

	UserID	Age	MovieID	Rating	Title	Action	Adventure	Animation	Children's	Comedy	...	Oc
0	1.0	1.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	0	...	

	UserID	Age	MovieID	Rating	Title	Action	Adventure	Animation	Children's	Comedy	...	Oc
1	2.0	56.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	0	...	
2	12.0	25.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	0	...	
3	15.0	25.0	1193	4.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	0	...	
4	17.0	50.0	1193	5.0	One Flew Over the Cuckoo's Nest (1975)	0	0	0	0	0	...	

5 rows × 47 columns

In [26]:
```
# Remove unwanted features for prediction
final_df=refined_df.drop(['UserID','MovieID','Title'],axis=1)
final_df.head()
```

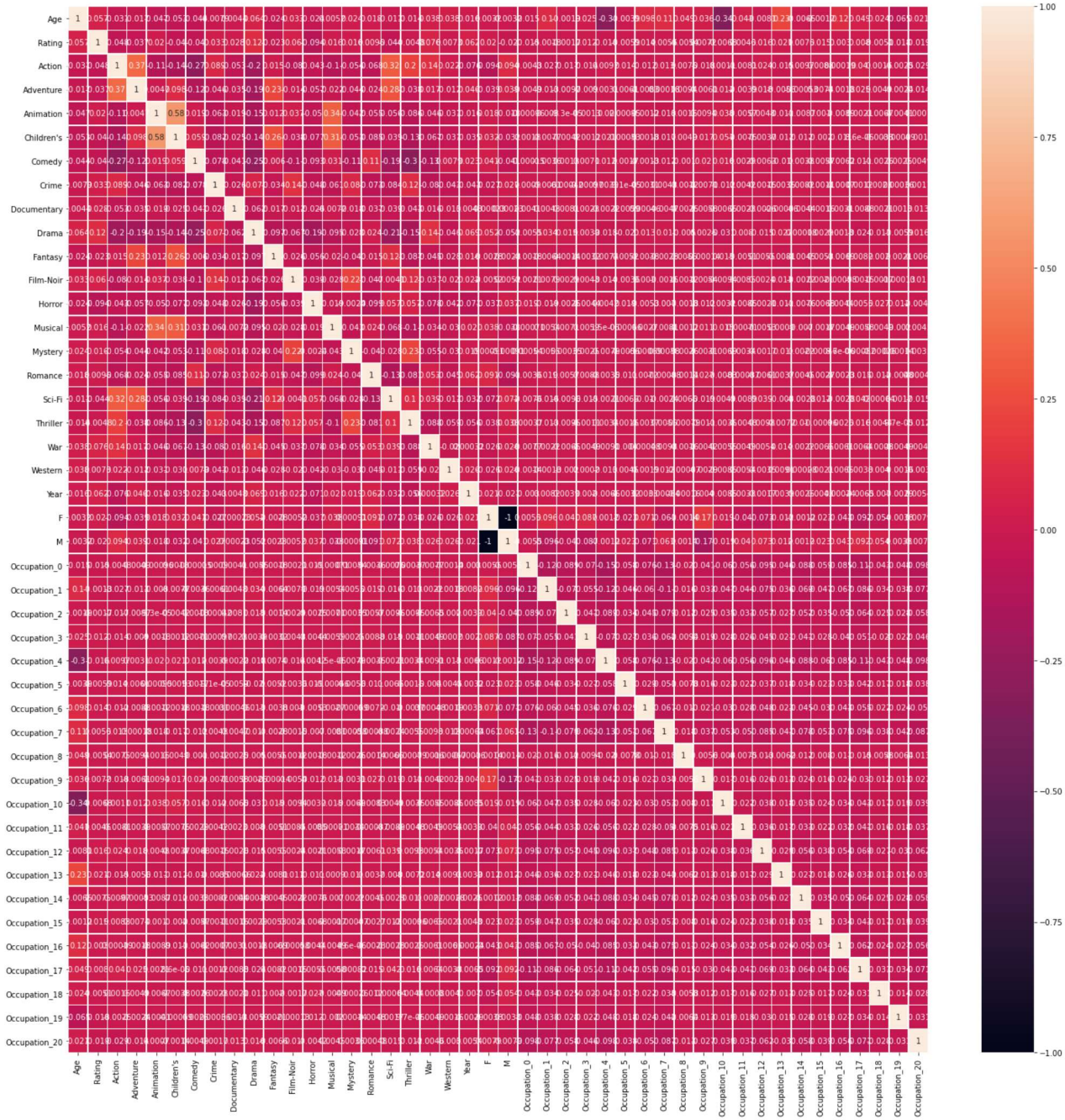
Out[26]:

	Age	Rating	Action	Adventure	Animation	Children's	Comedy	Crime	Documentary	Drama	...
0	1.0	5.0	0	0	0	0	0	0	0	1	...
1	56.0	5.0	0	0	0	0	0	0	0	1	...
2	25.0	4.0	0	0	0	0	0	0	0	1	...
3	25.0	4.0	0	0	0	0	0	0	0	1	...
4	50.0	5.0	0	0	0	0	0	0	0	1	...

5 rows × 44 columns

In [27]:
```
# Visualizing correlations
plt.figure(figsize=(25,25))
sns.heatmap(final_df.corr(),annot= True,linewidths=0.5)
```

Out[27]:
```
<AxesSubplot:>
```



4. Develop an appropriate model to predict the movie ratings

In [28]:
```
# Preparing the train & test split datasets
X=final_df.drop(['Rating'],axis=1)
y=final_df[['Rating']]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,random_state = 10
```

LGBMClassifier

In [29]:
```
start = time.time()
clf_lgb = LGBMClassifier(boosting_type = 'gbdt',n_jobs= -1,objective='multiclass')
clf_lgb.fit(X_train,y_train)
y_pred_lgb = clf_lgb.predict(X_test)
acc_lgb = accuracy_score(y_test,y_pred_lgb)*100
print('LGBM accuracy score is : ', acc_lgb)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
LGBM accuracy score is :  38.26546425250697
Execution Time: 0.4835 Min
```

Random Forest Classifier

In [30]:
```
start = time.time()
clf_RFC = RandomForestClassifier(min_samples_leaf=20,n_jobs=-1)
clf_RFC.fit(X_train,y_train)
y_pred_RFC = clf_RFC.predict(X_test)
acc_RFC = accuracy_score(y_test,y_pred_RFC)*100
print('Random Forest accuracy score is : ', acc_RFC)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
Random Forest accuracy score is :  38.58239769648374
Execution Time: 1.8584 Min
```

Decision Tree Classifier

In [31]:
```
start = time.time()
clf_dtree = DecisionTreeClassifier()
# This modelrequires more GPU to train and for time being I am keeping a smaller subset
clf_dtree.fit(X_train,y_train)
y_pred_dtree = clf_dtree.predict(X_test)
acc_dtree = accuracy_score(y_test,y_pred_dtree)*100
print('Decision Tree accuracy score is : ', acc_dtree)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
Decision Tree accuracy score is :  34.49625578628488
Execution Time: 0.2393 Min
```

Naive Bayes Classifier

In [32]:
```
start = time.time()
clf_gnb = GaussianNB()
clf_gnb.fit(X_train,y_train)
y_pred_gnb = clf_gnb.predict(X_test)
acc_gnb = accuracy_score(y_test,y_pred_gnb)*100
print('Naive Bayes accuracy score is : ', acc_gnb)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
Naive Bayes accuracy score is :  26.72838703872187
Execution Time: 0.0426 Min
```

XGBoost Classifier

In [33]:
```
start = time.time()
clf_xgb = xgboost.XGBClassifier(n_jobs=-1)
clf_xgb.fit(X_train,y_train)
y_pred_xgb = clf_xgb.predict(X_test)
acc_xgb = accuracy_score(y_test,y_pred_xgb)*100
print('XGB accuracy score is : ', acc_xgb)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
[10:27:37] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.4.0/src/lea
rner.cc:1095: Starting in XGBoost 1.3.0, the default evaluation metric used with the obj
ective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly set eval_met
ric if you'd like to restore the old behavior.
XGB accuracy score is :  38.66388058507713
Execution Time: 8.1454 Min
```

KNN Classifier

In [34]:
```
start = time.time()
#Roughly the k value will be the square root of total records in training set
# This modelrequires more GPU to train and for time being I am keeping a smaller subset
k = int(np.sqrt(len(X_train[:10000])))
clf_KNN = KNeighborsClassifier(n_neighbors=k,n_jobs=-1)
clf_KNN.fit(X_train[:10000],y_train[:10000])
y_pred_KNN = clf_KNN.predict(X_test[:10000])
acc_KNN = accuracy_score(y_test[:10000],y_pred_KNN)*100
print('KNN accuracy score is : ', acc_KNN)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
KNN accuracy score is :  33.72
Execution Time: 0.0961 Min
```

SVC Bagging Classifier

In [35]:
```
start = time.time()
n_estimators = 10
clf_bag_svc = OneVsRestClassifier(BaggingClassifier(SVC(kernel='linear', probability=Tr
# This modelrequires more GPU to train and for time being I am keeping a smaller subset
clf_bag_svc.fit(X_train[:1000],y_train[:1000])
y_pred_bag_svc = clf_bag_svc.predict(X_test[:1000])
acc_bag_svc = accuracy_score(y_test[:1000],y_pred_bag_svc)*100
print('Bagging SVC accuracy score is : ', acc_bag_svc)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
Bagging SVC accuracy score is :  29.4
Execution Time: 0.4829 Min
```

SVC Single Classifier

In [36]:
```
start = time.time()
clf_single_svc = OneVsRestClassifier(SVC(kernel='linear', probability=True, class_weigh
# This modelrequires more GPU to train and for time being I am keeping a smaller subset
clf_single_svc.fit(X_train[:1000],y_train[:1000])
y_pred_single_svc = clf_single_svc.predict(X_test[:1000])
acc_single_svc = accuracy_score(y_test[:1000],y_pred_single_svc)*100
print('Single SVC accuracy score is : ', acc_single_svc)
end = time.time()
print('Execution Time:',round((end - start)/60,4),'Min')
```

```
Single SVC accuracy score is :  26.5
Execution Time: 0.5665 Min
```