In [1]:

```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
%matplotlib inline
import re
import sys
import time
import datetime
from sklearn import metrics
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import warnings
warnings.filterwarnings("ignore")
print ('All Libraries Imported')
```

```
All Libraries Imported
```

In [2]:

```
# Import Ratings Dataset
df1=pd.read_csv('D:/Python/data science python/Datasets for project/movielens/ratings.d
               names=['UserID','MovieID','Rating','Timestamp'])
# Import Users Dataset
df2=pd.read_csv('D:/Python/data science python/Datasets for project/movielens/users.dat
               names=['UserID','Gender','Age','Occupation','Zip-code'])
# Import Movies Dataset
df3=pd.read_csv('D:\Python\data science python\Datasets for project\movielens\movies.da
               names=['MovieID','Title','Genres'])
```

In [3]:

```
df1.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

	UserID	MovieID	Rating	Timestamp
4	1	2355	5	978824291

In [4]:
```
df2.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]:
```
df3.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

In [6]:
```
print(df1.shape)
print(df2.shape)
print(df3.shape)
```

```
(1000209, 4)
(6040, 5)
(3883, 3)
```

In [7]:
```
# Merging data sets on the basis of common feature 'UserID'
df4=pd.merge(df1,df2,on='UserID')
df4.head()
```

Out[7]:

	UserID	MovieID	Rating	Timestamp	Gender	Age	Occupation	Zip-code
0	1	1193	5	978300760	F	1	10	48067
1	1	661	3	978302109	F	1	10	48067
2	1	914	3	978301968	F	1	10	48067
3	1	3408	4	978300275	F	1	10	48067
4	1	2355	5	978824291	F	1	10	48067

In [8]:
```
#Merge data sets with respect to 'MovieID'
Master_Data=pd.merge(df3,df4,on='MovieID')
Master_Data.head()
```

Out[8]:

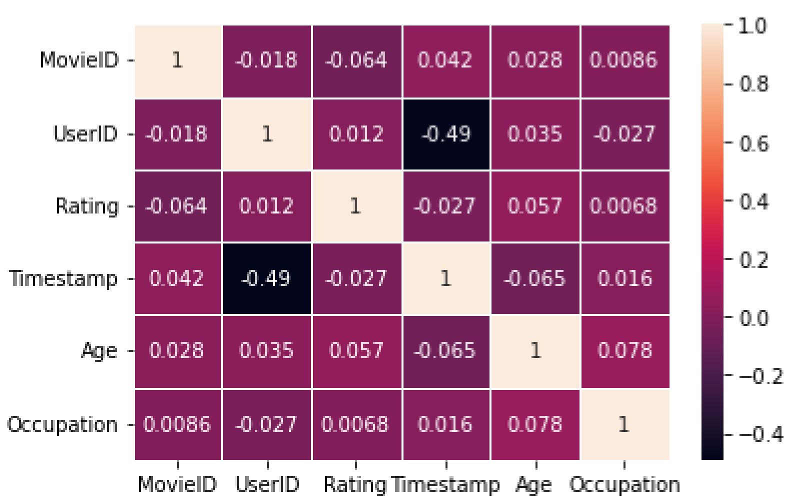
	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008	F	50	9
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496	M	25	12
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952	M	25	17
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474	F	35	1

In [9]:
```
Master_Data.shape
```

Out[9]:
```
(1000209, 10)
```

In [10]:
```
corr = Master_Data.corr()
sns.heatmap(corr,annot= True,linewidths=0.5)
```

Out[10]:
```
<AxesSubplot:>
```



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

Out[11]:
```
0
```

In [12]:
```
Master_Data.duplicated().sum()
```

Out[12]:
```
0
```

In [13]:
```
duplicate = Master_Data[Master_Data.duplicated(['Title', 'UserID'])]

print("Duplicate Rows based on Title and UserID :")

# Print the resultant Dataframe
duplicate
```

```
Duplicate Rows based on Title and UserID :
```

Out[13]:

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

In [14]:
```
# now lets add a column called rating_year which depicts the year when the rating was g
import datetime
year_lambda = lambda x: int(datetime.datetime.fromtimestamp(x).strftime('%Y'))
Master_Data['Rating_year'] = Master_Data['Timestamp'].apply(year_lambda)
Master_Data.head()
```

Out[14]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008	F	50	9
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496	M	25	12
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952	M	25	17
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474	F	35	1

In [15]:
```
# now lets create a new data frame which contains number of ratings given on each year
Ratings_per_year = Master_Data.groupby(['Rating_year'])['Rating_year'].count()
Ratings_per_year.head(5)
```

Out[15]:
```
Rating_year
2000    904175
2001     68628
2002     24053
```

```
2003      3353
Name: Rating_year, dtype: int64
```

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

In [16]:

```
Master_Data.to_csv("D:\Python\data science python\Datasets for project\movielens\Master
#Writing the file on system so can be accessed offline too
```

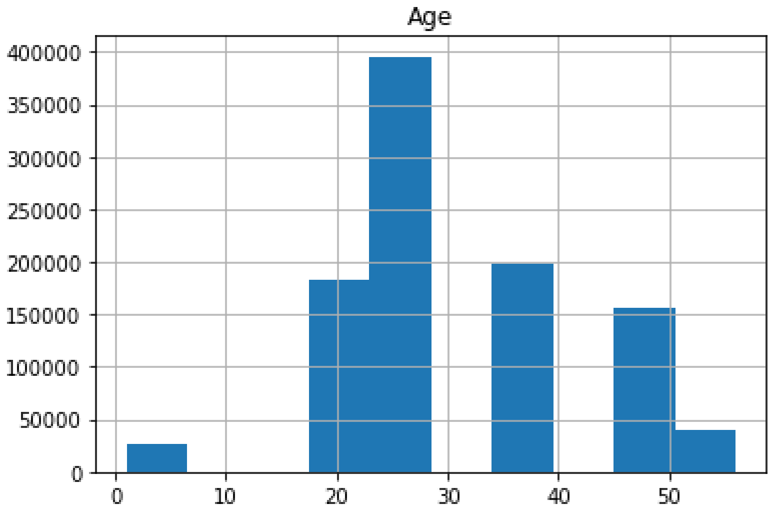
Problem Statement 1:

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

User Age Distribution

In [17]:

```
# Users with Different Age Groups
Master_Data.hist(column='Age')
plt.show()
```

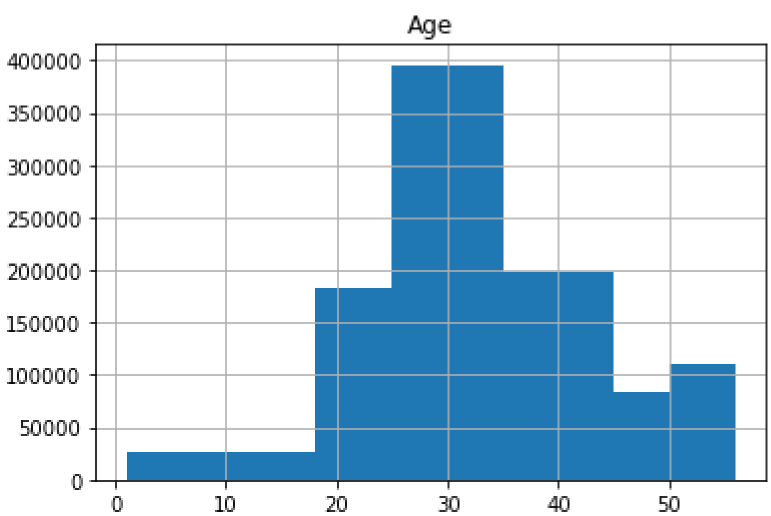


In [18]:

```
#Young people of the age group between 20 t0 30 are the most users
#As age increases the number of users decreases
#In the user data set, we have ages mapped to categorical numbers using this table from
#Value Description
#1 = "Under 18"
#18 = "18-24"
#25 = "25-34"
#35 = "35-44"
#45 = "45-49"
#50 = "50-55"
#56 = "56+"
# Redraw our histogram based on our newly mapped bin values
```

In [19]:

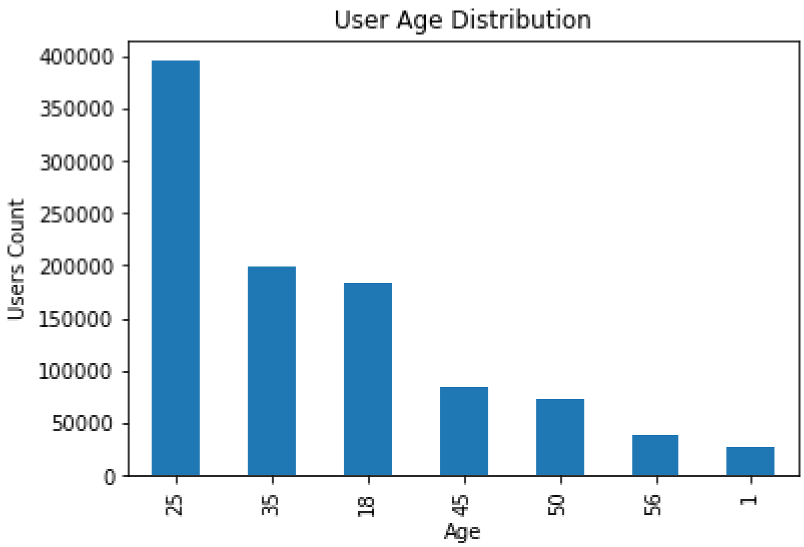
```
bins_list = [1, 18, 25, 35, 45, 50, 56]
Master_Data.hist(column='Age', bins = bins_list)
plt.show()
```



In [20]:

```
Master_Data['Age'].value_counts().plot(kind='bar')
plt.xlabel("Age")
plt.title("User Age Distribution")
plt.ylabel('Users Count')
```

```
plt.show()
#The graph emphasizes the result of the above graph that young users are most users whi
```



In [21]:

```
# 75% Users are between Age 18 to 35 years
```

In [22]:

```
#Gender Distribution
gender_group = Master_Data['Gender']
gender_group.value_counts()
```

Out[22]:

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

In [23]:

```
#Most Users are males
```

In [24]:

```
#Problem Statement 2
#Finding the rating of the movie 'Toy Story'
# Toy Story Rating
toystoryRating = Master_Data[Master_Data['Title'].str.contains('Toy Story') == True]
toystoryRating.head()
```

Out[24]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008	F	50	9
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496	M	25	12

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952	M	25	17
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474	F	35	1

In [25]:

```
movieTitles = Master_Data.Title.unique()
toyMovie = []
for i in movieTitles:
    if i.startswith("Toy Story") == True:
        toyMovie.append(i)

toyMovie
```

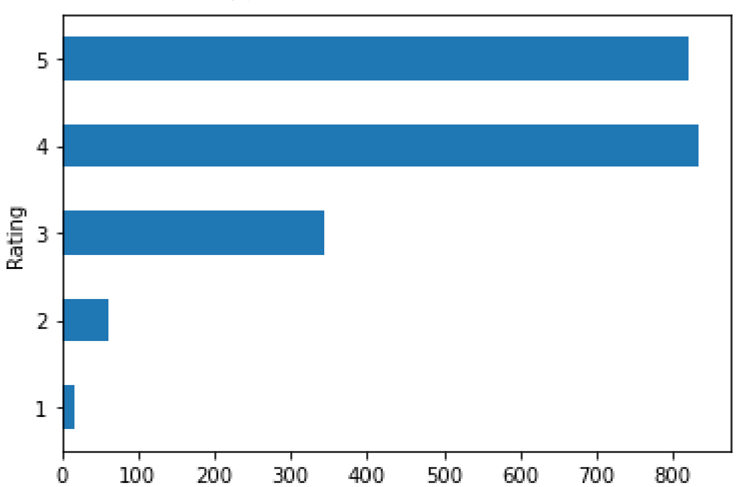
Out[25]:

```
['Toy Story (1995)', 'Toy Story 2 (1999)']
```

In [26]:

```
filterDf=Master_Data[Master_Data['Title']=='Toy Story (1995)']
print(filterDf.groupby('Rating')['UserID'].count())
filterDf.groupby('Rating')['UserID'].count().plot(kind='barh')
plt.show()
```

```
Rating
1     16
2     61
3    345
4    835
5    820
Name: UserID, dtype: int64
```



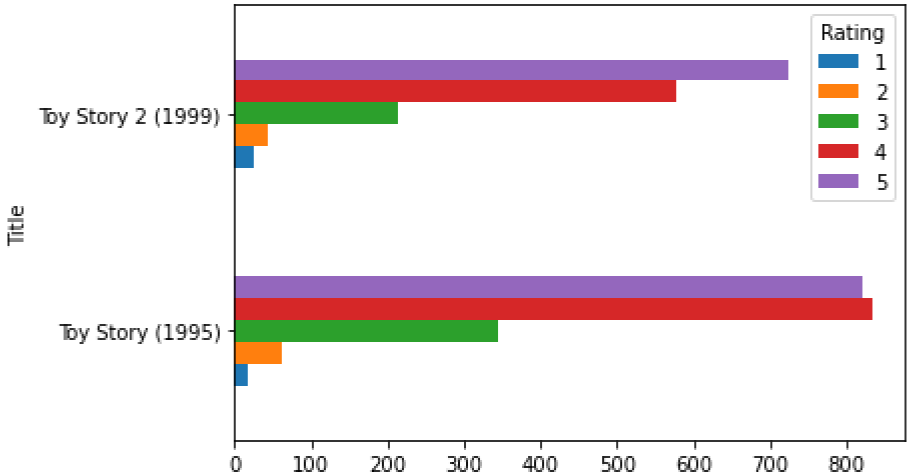
In [27]:

```
# Most Users gave rating of 4 or 5
Master_Data[Master_Data['MovieID'] == 1].Rating.mean()
```

Out[27]:

```
4.146846413095811
```

In [28]:
```
toystoryRating.groupby(["Title","Rating"]).size().unstack().plot(kind='barh',stacked=Fa
plt.show()
```



The most popular version was 'Toy Story(1995)'

In [29]:
```
#Finding ratings for the popular version
user_rating = Master_Data[Master_Data.Title == "Toy Story (1995)"]

user_rating
```

Out[29]:

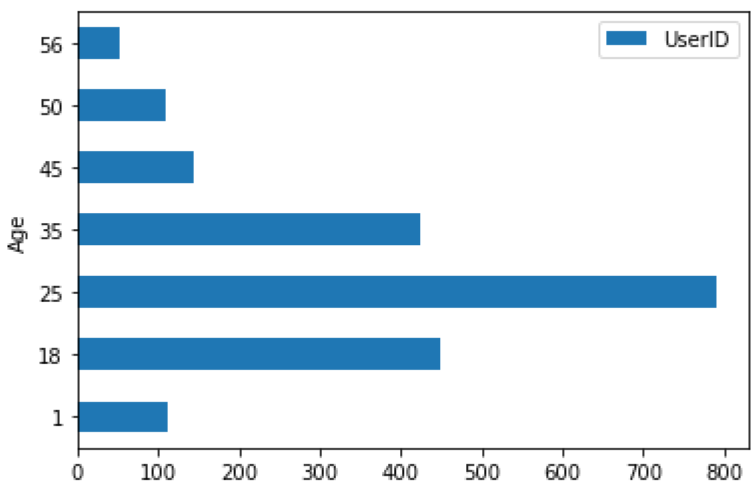
	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupati
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008	F	50	
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496	M	25	
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952	M	25	
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474	F	35	
...	...	...	...	...	...	...	...	...	
2072	1	Toy Story (1995)	Animation|Children's|Comedy	6022	5	956755763	M	25	
2073	1	Toy Story (1995)	Animation|Children's|Comedy	6025	5	956812867	F	25	

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupati
2074	1	Toy Story (1995)	Animation|Children's|Comedy	6032	4	956718127	M	45	
2075	1	Toy Story (1995)	Animation|Children's|Comedy	6035	4	956712849	F	25	
2076	1	Toy Story (1995)	Animation|Children's|Comedy	6040	3	957717358	M	25	

2077 rows × 11 columns

In [30]:
```
#Visualizing the Viewership groupby 'Age' for the movie 'Toy Story(1995)'
print(user_rating.groupby('Age')['UserID'].count())
user_rating.groupby('Age')['UserID'].count().plot(kind='barh',stacked=False,legend=True
plt.show()
```

```
Age
1      112
18     448
25     790
35     423
45     143
50     108
56      53
Name: UserID, dtype: int64
```



In [31]:
```
#The above plot shows that the Toystory movie is more popular for viewers between Age g
```

In [32]:
```
#How many movies have 5 star rating?
Master_Data[Master_Data['Rating'] == 5].Rating.count()
```

Out[32]: 226310

In [33]:
```
#Task 3: Top 25 movies by viewership rating
top_25movies=pd.DataFrame(Master_Data.groupby('Title')['Rating'].agg('mean')).sort_valu
top_25movies
```

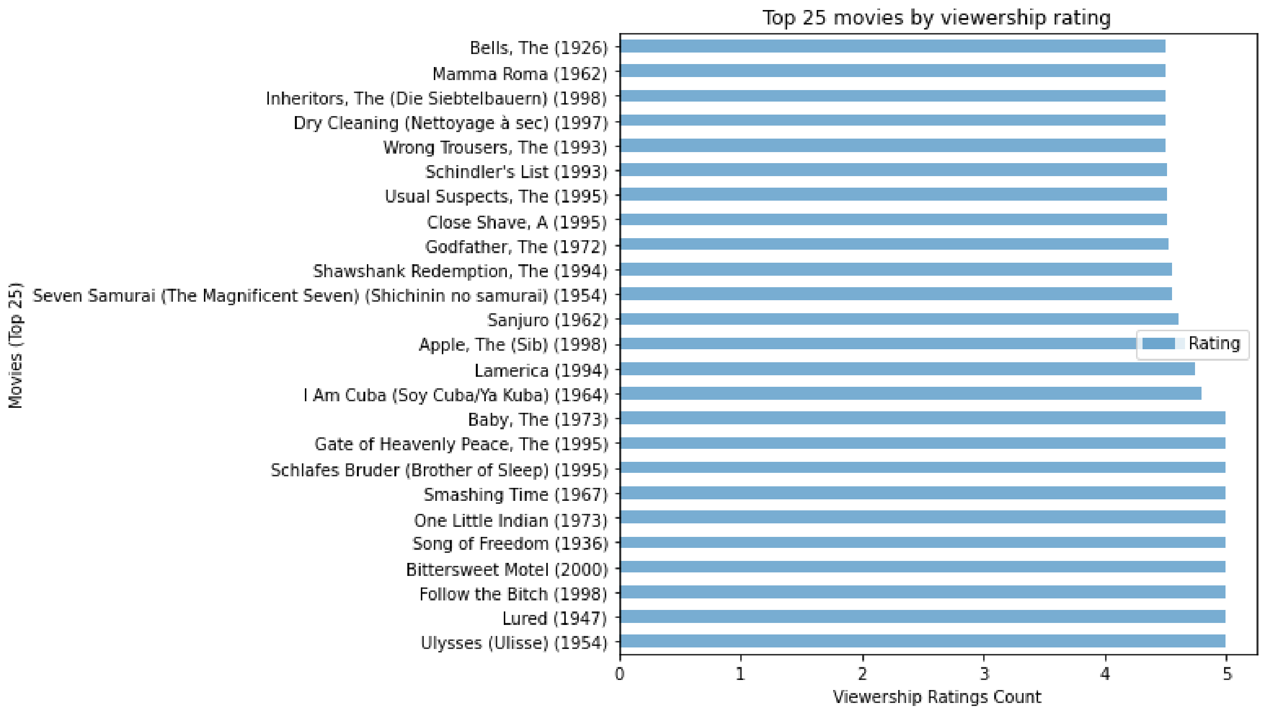
Out[33]:

Title	Rating
Ulysses (Ulisse) (1954)	5.000000
Lured (1947)	5.000000
Follow the Bitch (1998)	5.000000
Bittersweet Motel (2000)	5.000000
Song of Freedom (1936)	5.000000
One Little Indian (1973)	5.000000
Smashing Time (1967)	5.000000
Schlafes Bruder (Brother of Sleep) (1995)	5.000000
Gate of Heavenly Peace, The (1995)	5.000000
Baby, The (1973)	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
Dry Cleaning (Nettoyage à sec) (1997)	4.500000
Inheritors, The (Die Siebtelbauern) (1998)	4.500000
Mamma Roma (1962)	4.500000
Bells, The (1926)	4.500000

In [34]:
```
#Out of top 25 movies, top 10 movies has avg. Rating 5
```

In [35]:
```
top_25movies.plot(kind='barh',alpha=0.6,figsize=(7,7))
plt.xlabel("Viewership Ratings Count")
plt.ylabel("Movies (Top 25)")
```

```
plt.title("Top 25 movies by viewership rating")
plt.show()
```



In [36]:

```
#Task 4: Find the ratings for all the movies reviewed by for a particular user of user
userId = 2696
userRatingById = Master_Data[Master_Data["UserID"] == userId]
userRatingById = userRatingById.sort_values('Rating',ascending=False, ignore_index=True
userRatingById[['MovieID','Rating','Title']]
```

Out[36]:

	MovieID	Rating	Title
0	800	5	Lone Star (1996)
1	1645	4	Devil's Advocate, The (1997)
2	1783	4	Palmetto (1998)
3	1092	4	Basic Instinct (1992)
4	3176	4	Talented Mr. Ripley, The (1999)
5	1258	4	Shining, The (1980)
6	2389	4	Psycho (1998)
7	1892	4	Perfect Murder, A (1998)
8	1617	4	L.A. Confidential (1997)
9	1625	4	Game, The (1997)
10	1805	4	Wild Things (1998)
11	1711	4	Midnight in the Garden of Good and Evil (1997)
12	350	3	Client, The (1994)
13	1589	3	Cop Land (1997)

	MovieID	Rating	Title
14	1097	3	E.T. the Extra-Terrestrial (1982)
15	1644	2	I Know What You Did Last Summer (1997)
16	2338	2	I Still Know What You Did Last Summer (1998)
17	1270	2	Back to the Future (1985)
18	2713	1	Lake Placid (1999)
19	3386	1	JFK (1991)

In [37]:

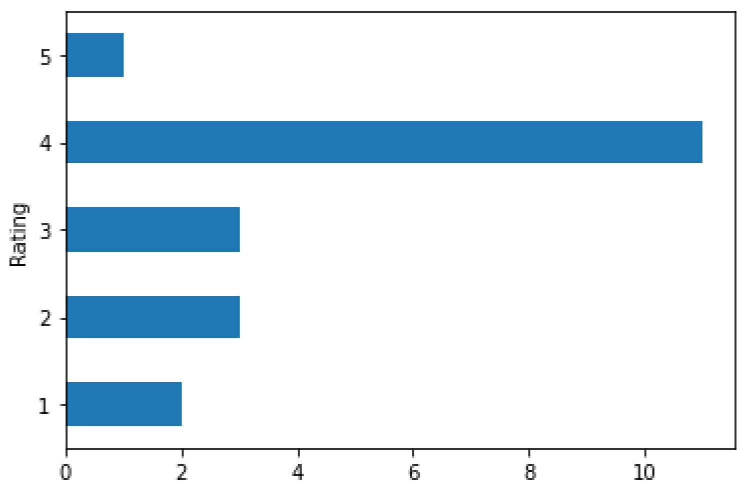
```
#Userid no 2696 have given maximum Rating of 5 to movie 'Lone Star (1996)'

#& most of user's rating is 4
```

In [38]:

```
print(Master_Data[Master_Data['UserID']==2696].groupby('Rating')['UserID'].count())
Master_Data[Master_Data['UserID']==2696].groupby('Rating')['UserID'].count().plot(kind=
plt.show()
#Most Ratings of the user with UserID 2696 is 4
```

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



Feature Engineering:

In [39]:

```
#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)
#Transforming a column to variable column
movies_genres = df3['Genres'].str.split('|')
movies_genres
```

Out[39]:
```
0          [Animation, Children's, Comedy]
1         [Adventure, Children's, Fantasy]
2                        [Comedy, Romance]
3                          [Comedy, Drama]
4                                 [Comedy]
                      ...
3878                              [Comedy]
3879                               [Drama]
3880                               [Drama]
3881                               [Drama]
3882                     [Drama, Thriller]
Name: Genres, Length: 3883, dtype: object
```

In [40]:
```
listGenres = set()
for genre in movies_genres:
    listGenres = listGenres.union(set(genre))
print(listGenres)
```

```
{"Children's", 'Horror', 'Animation', 'Thriller', 'Mystery', 'Sci-Fi', 'Crime', 'Musica
l', 'Film-Noir', 'Adventure', 'Fantasy', 'Action', 'Documentary', 'Western', 'Romance',
'Comedy', 'War', 'Drama'}
```

In [41]:
```
print('Total number of unique Genres are :', len(listGenres)  )
```

```
Total number of unique Genres are : 18
```

In [42]:
```
#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 [43]:
```
#create dummy variables for one-hot encoding in binary 0 or 1. Thus converting categori
#making copy of the data set first and creating a new variable
new_df=Master_Data[['Gender',
    'Age',
    'Occupation',
    'Rating',
    'Genres']].copy()
new_df.head(5)
```

Out[43]:

	Gender	Age	Occupation	Rating	Genres
0	F	1	10	5	Animation|Children's|Comedy
1	F	50	9	4	Animation|Children's|Comedy
2	M	25	12	4	Animation|Children's|Comedy
3	M	25	17	5	Animation|Children's|Comedy
4	F	35	1	5	Animation|Children's|Comedy

In [44]:
```
Genre = new_df['Genres']
Genre = Genre.str.get_dummies().add_prefix('Genres_') #converting categorical to numeri
movie_ratings_genres_df = pd.concat(
    [new_df.drop(
        ['Genres'],
        axis=1
```

```
    ),
     Genre],
    axis=1,

)
movie_ratings_genres_df.head()
```

Out[44]:

	Gender	Age	Occupation	Rating	Genres_Action	Genres_Adventure	Genres_Animation	Genres_Child
0	F	1	10	5	0	0	1	
1	F	50	9	4	0	0	1	
2	M	25	12	4	0	0	1	
3	M	25	17	5	0	0	1	
4	F	35	1	5	0	0	1	

5 rows × 22 columns

In [45]:

```
movie_ratings_genres_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1000209 entries, 0 to 1000208
Data columns (total 22 columns):
 #   Column              Non-Null Count    Dtype 
---  ------              --------------    ----- 
 0   Gender              1000209 non-null  object
 1   Age                 1000209 non-null  int64 
 2   Occupation          1000209 non-null  int64 
 3   Rating              1000209 non-null  int64 
 4   Genres_Action       1000209 non-null  int64 
 5   Genres_Adventure    1000209 non-null  int64 
 6   Genres_Animation    1000209 non-null  int64 
 7   Genres_Children's   1000209 non-null  int64 
 8   Genres_Comedy       1000209 non-null  int64 
 9   Genres_Crime        1000209 non-null  int64 
 10  Genres_Documentary  1000209 non-null  int64 
 11  Genres_Drama        1000209 non-null  int64 
 12  Genres_Fantasy      1000209 non-null  int64 
 13  Genres_Film-Noir    1000209 non-null  int64 
 14  Genres_Horror       1000209 non-null  int64 
 15  Genres_Musical      1000209 non-null  int64 
 16  Genres_Mystery      1000209 non-null  int64 
 17  Genres_Romance      1000209 non-null  int64 
 18  Genres_Sci-Fi       1000209 non-null  int64 
 19  Genres_Thriller     1000209 non-null  int64 
 20  Genres_War          1000209 non-null  int64 
 21  Genres_Western      1000209 non-null  int64 
dtypes: int64(21), object(1)
memory usage: 175.5+ MB
```

In [46]:

```
#need to convert Gender to int
```

In [47]:
```
df = pd.get_dummies(
    movie_ratings_genres_df,
    columns=['Gender']
)
```

In [48]:
```
df.head()
```

Out[48]:

	Age	Occupation	Rating	Genres_Action	Genres_Adventure	Genres_Animation	Genres_Children's	Ge
0	1	10	5	0	0	1	1	
1	50	9	4	0	0	1	1	
2	25	12	4	0	0	1	1	
3	25	17	5	0	0	1	1	
4	35	1	5	0	0	1	1	

5 rows × 23 columns

Determine the features affecting the ratings of any particular movie.

In [49]:
```
Master_Data.head()
```

Out[49]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	F	1	10
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008	F	50	9
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496	M	25	12
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952	M	25	17
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474	F	35	1

In [50]:
```
Master_Data.shape
```

Out[50]: (1000209, 11)

In [51]:
```
#Create a smaller features dataframe of the movie data features that are relevant to ra
#zip code
#gender
#age
#occupation
Master_Data['Gender'].replace(['F','M'],[0,1],inplace=True)
```

In [52]:
```
Master_Data.head()
```

Out[52]:

	MovieID	Title	Genres	UserID	Rating	Timestamp	Gender	Age	Occupation
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268	0	1	10
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008	0	50	9
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496	1	25	12
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952	1	25	17
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474	0	35	1

In [53]:
```
new_df = Master_Data.iloc[:, [0,4,6,7,8,9]]
```

In [54]:
```
new_df.head()
```

Out[54]:

	MovieID	Rating	Gender	Age	Occupation	Zip-code
0	1	5	0	1	10	48067
1	1	4	0	50	9	55117
2	1	4	1	25	12	11413
3	1	5	1	25	17	61614
4	1	5	0	35	1	95370

In [55]:
```
#Check the type of data
new_df.dtypes
```

Out[55]:
```
MovieID        int64
Rating         int64
Gender         int64
Age            int64
Occupation     int64
Zip-code      object
dtype: object
```

In [56]:
```
#Changing zipcode to numerical data
new_df['Zip-code'] = new_df['Zip-code'].str[:5]
pd.to_numeric(new_df['Zip-code'])
```

Out[56]:
```
0          48067
1          55117
2          11413
3          61614
4          95370
           ...  
1000204    92120
1000205    92120
1000206    60607
1000207    10003
1000208    61820
Name: Zip-code, Length: 1000209, dtype: int64
```

In [57]:
```
#Find the correlation of movieID

new_df[new_df.columns[1:]].corr()['Rating'][:]
```

Out[57]:
```
Rating        1.000000
Gender       -0.019861
Age           0.056869
Occupation    0.006753
Name: Rating, dtype: float64
```

We see that Age has the highest correlation and also Occupation has correlation to Rating of a movie

Actually it is a classification problem since Ratings is not continuous and has unique values & we will run different classification models

In [58]:
```
new_df.Rating.unique()
```

Out[58]:
```
array([5, 4, 3, 2, 1], dtype=int64)
```

In [59]:
```
new_df = new_df.drop(['Zip-code'], axis=1)
```

In [60]:
```
new_df = new_df.drop(['Zip-code'], axis=1)
```

Out[60]:

	MovieID	Rating	Gender	Age	Occupation
0	1	5	0	1	10
1	1	4	0	50	9
2	1	4	1	25	12
3	1	5	1	25	17
4	1	5	0	35	1

In [61]:

```
new_df.shape
```

Out[61]: (1000209, 5)

In [62]:

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

from sklearn import metrics

lineReg = LinearRegression(
    copy_X=True,
    fit_intercept=True,
    n_jobs=1,
    normalize=False
)
```

In [63]:

```
sample_df = new_df.sample(
    n=50000,
    random_state=0
)
sample_df.head()
```

Out[63]:

	MovieID	Rating	Gender	Age	Occupation
324271	1220	3	0	45	0
818637	3052	2	1	50	18
148677	541	5	1	56	13
778790	2906	1	1	25	11
525489	1957	4	1	45	17

In [64]:

```
x = sample_df.drop('Rating',axis=1)
y = sample_df['Rating']
```

In [65]:

```
x_train, x_test, y_train, y_test = train_test_split(
    x,
    y,
    test_size=0.30,
```

```
    random_state=0
)
```

In [66]:
```
linear_reg = LinearRegression()
```

In [67]:
```
linear_reg.fit(x_train, y_train)
```

Out[67]:
```
LinearRegression()
```

In [68]:
```
y_pred = linear_reg.predict(x_test)
```

In [69]:
```
# Evaluation: Finding out which features affect most to the Ratings of the movie
from sklearn.metrics import r2_score
y_train_pred= linear_reg.predict(x_train)
round(r2_score(y_train, y_train_pred)*100,2)
```

Out[69]:
```
0.78
```

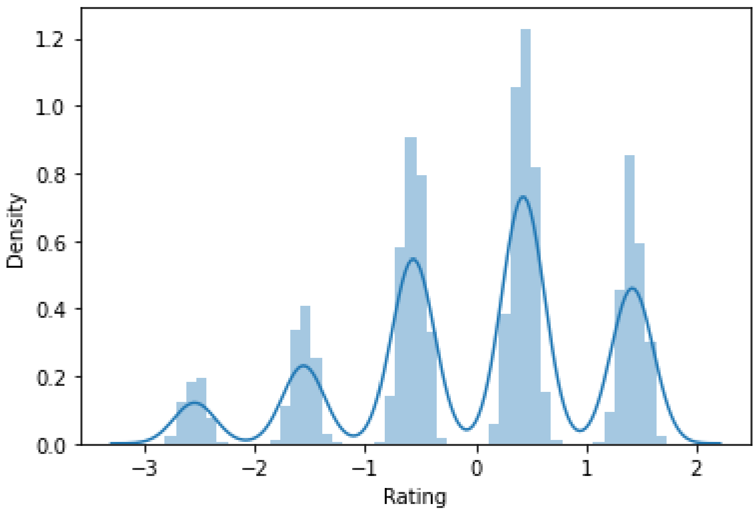
In [70]:
```
print(
    'y-intercept: ',
    linear_reg.intercept_
)
print(
    'Beta coefficients: ',
    linear_reg.coef_
)
print(
    'Mean Abs Error  MAE: ',
    metrics.mean_absolute_error(y_test, y_pred)
)
print(
    'Mean Sq Error  MSE: ',
    metrics.mean_squared_error(y_test, y_pred)
)
print(
    'Root Mean Sq Error RMSE:',
    np.sqrt(metrics.mean_squared_error(y_test, y_pred))
)
print(
    'r2 value: ',
    metrics.r2_score(y_test, y_pred)
)
```

```
y-intercept:  3.563670438396085
Beta coefficients:  [-7.11343162e-05 -2.82717683e-02  5.00934063e-03  1.31311952e-03]
Mean Abs Error  MAE:  0.927783889252895
Mean Sq Error  MSE:  1.2386698838145336
Root Mean Sq Error RMSE: 1.1129554725210409
r2 value:  0.005945834274465933
```

In [71]:
```
# RMSE value higher and r2 value on test data lower indicates the model is not good fit
```

In [72]:
```
residual = y_test - y_pred
sns.distplot(residual)
```

Out[72]: <AxesSubplot:xlabel='Rating', ylabel='Density'>



In [73]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

le.fit(sample_df['Age'])
x_age = le.transform(sample_df['Age'])
x_age
```

Out[73]:
```
array([4, 5, 6, ..., 2, 1, 1], dtype=int64)
```

In [74]:
```
le.fit(sample_df['Occupation'])
x_occ = le.transform(sample_df['Occupation'])
x_occ
```

Out[74]:
```
array([ 0, 18, 13, ...,  3,  1,  4], dtype=int64)
```

In [75]:
```
le.fit(sample_df['Gender'])
x_gen = le.transform(sample_df['Gender'])
x_gen
```

Out[75]:
```
array([0, 1, 1, ..., 0, 1, 1], dtype=int64)
```

In [76]:
```
le.fit(sample_df['MovieID'])
x_movieid = le.transform(sample_df['MovieID'])
x_movieid
```

Out[76]:
```
array([ 941, 2418,  468, ..., 2142,  124,  545], dtype=int64)
```

In [77]:
```
sample_df['New Age'] = x_age
sample_df['New Occupation'] = x_occ
sample_df['New Gender'] = x_gen
sample_df['New MovieID'] = x_movieid
```

In [78]:
```
# Feature Selection
x_input = sample_df[['New Age','New Occupation','New Gender','New MovieID']]
y_target = sample_df['Rating']
```

In [79]:
```
x_input.head()
```

Out[79]:

	New Age	New Occupation	New Gender	New MovieID
324271	4	0	0	941
818637	5	18	1	2418
148677	6	13	1	468
778790	2	11	1	2297
525489	4	17	1	1492

In [80]:
```
y_target.head()
```

Out[80]:
```
324271    3
818637    2
148677    5
778790    1
525489    4
Name: Rating, dtype: int64
```

In [81]:
```
# Split-out validation dataset
x_train, x_test, y_train, y_test = train_test_split(x_input, y_target, test_size=0.30)
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[81]:
```
((35000, 4), (15000, 4), (35000,), (15000,))
```

In [82]:
```
from sklearn.linear_model import LogisticRegression
#Logistic regression is best used for predicting categorical data
#need to do logistic regression on the training data so we can see how well our test da
```

In [83]:
```
logreg = LogisticRegression(max_iter=100000)
```

In [84]:
```
logreg.fit(x_train,y_train)
```

Out[84]:
```
LogisticRegression(max_iter=100000)
```

In [85]:
```
y_pred = logreg.predict(x_test)
```

In [86]:
```
from sklearn import metrics
round(metrics.accuracy_score(y_test,y_pred)*100,2)
```

Out[86]: 34.92

In [87]:
```
# print the first 30 true and predicted responses
print ('actual:    ', y_test.values[0:30])
print ('predicted: ', y_pred[0:30])
```

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

In [88]:
```
prediction_df = pd.DataFrame({'Test': y_test, 'Prediction': y_pred})
prediction_df.head()
```

Out[88]:

	Test	Prediction
631262	4	4
314953	2	4
141099	3	4
343175	5	4
183115	3	4

In [89]:
```
#KNN

from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import plot_confusion_matrix
knn = KNeighborsClassifier(n_neighbors = 8).fit(x_train, y_train)

# accuracy on x_test
accuracy = round(knn.score(x_test, y_test) *100,2)

# creating a confusion matrix
knn_predictions = knn.predict(x_test)
print (accuracy)
```

32.59

In [90]:
```
#Naive Bayes classifier

from sklearn.naive_bayes import GaussianNB
GN = GaussianNB().fit(x_train, y_train)
GN_predictions = GN.predict(x_test)

# accuracy on X_test
accuracy = GN.score(x_test, y_test)


round(accuracy*100,2)
```

Out[90]: 34.92

In [95]:
```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [96]:

```
#Performing LDA
#Linear Discriminant Analysis works by reducing the dimensionality of the dataset, proj
#Then it combines these points into classes based on their distance from a chosen point
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n_components=1)
x_train = lda.fit_transform(x_train, y_train)
x_test = lda.transform(x_test)
```

In [97]:

```
#Training & Making predictions
from sklearn.ensemble import RandomForestClassifier

classifier = RandomForestClassifier(max_depth=2, random_state=0)

classifier.fit(x_train, y_train)
y_pred = classifier.predict(x_test)
```

In [102…

```
#Evaluating the performances
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

cm = confusion_matrix(y_test, y_pred)
print(cm)
print('Accuracy' + str(accuracy_score(y_test, y_pred)))
```

```
[[   0    0    0  883    0]
 [   0    0    0 1646    0]
 [   0    0    0 3935    0]
 [   0    0    0 5238    0]
 [   0    0    0 3298    0]]
Accuracy0.3492
```

In [103…

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       883
           2       0.00      0.00      0.00      1646
           3       0.00      0.00      0.00      3935
           4       0.35      1.00      0.52      5238
           5       0.00      0.00      0.00      3298

    accuracy                           0.35     15000
   macro avg       0.07      0.20      0.10     15000
weighted avg       0.12      0.35      0.18     15000
```

In [ ]:

```
#F1 score very much away from 1, so we need to fine tune the model and the performance
```

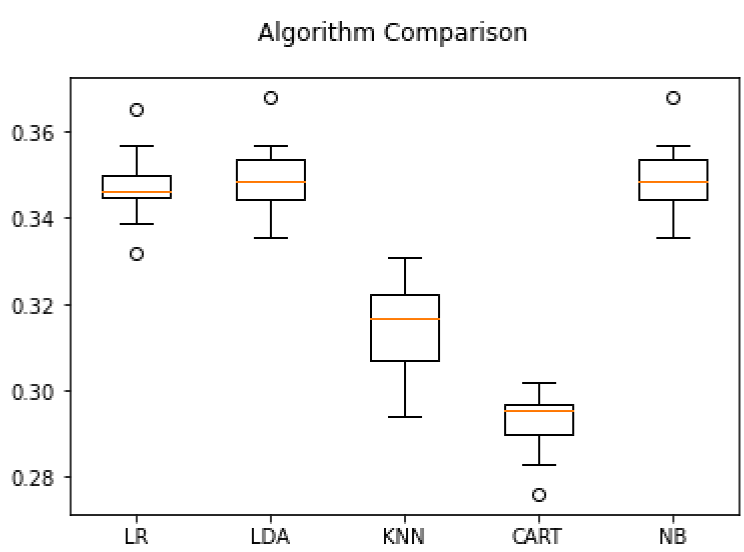
In [ ]:
```
#We can check all classifiers together for comparison purpose
```

In [91]:
```
# Spot-Check Algorithms
seed = 7
models = []
models.append(('LR', LogisticRegression()))
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)
    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)
```

```
LR: 0.347257 (0.008604)
LDA: 0.348971 (0.008853)
KNN: 0.314657 (0.010421)
CART: 0.292857 (0.007732)
NB: 0.348971 (0.008853)
```

In [92]:
```
# Compare Algorithms
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



In [93]:
```
# NB & LDA are better models than others
# The model has poor performance and so we need to further fine tune the model with xgb
```