Task 1: Import the three datasets

Reading Movies.data with separator '::' and adding columns names as 'MovieID', 'Title', 'Genres'

In [ ]:
```
import pandas as pd
```

In [2]:
```
movie_data = pd.read_csv('E:\\Simplilean\\DS with Python\\Data science with Pytho
```

```
C:\Python\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling bac
k to the 'python' engine because the 'c' engine does not support regex separato
rs (separators > 1 char and different from '\s+' are interpreted as regex); you
can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Printing columns

In [3]:
```
movie_data.columns
```

Out[3]:
```
Index(['MovieID', 'Title', 'Genres'], dtype='object')
```

Printing First 10 records of data frame

In [4]:
```
movie_data.head(10)
```

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
5	6	Heat (1995)	Action|Crime|Thriller
6	7	Sabrina (1995)	Comedy|Romance
7	8	Tom and Huck (1995)	Adventure|Children's
8	9	Sudden Death (1995)	Action
9	10	GoldenEye (1995)	Action|Adventure|Thriller

Using info() to check datatyps and memory usage

In [5]:
```
movie_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3883 entries, 0 to 3882
Data columns (total 3 columns):
MovieID    3883 non-null int64
Title      3883 non-null object
Genres     3883 non-null object
dtypes: int64(1), object(2)
memory usage: 91.1+ KB
```

Using describe to check Data exploration analaysis

In [6]:
```
movie_data.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

Reading Ratings.dat file with separator '::' and adding columns names as 'UserID', 'MovieID', 'Rating','Timestamp'

In [7]:
```
rating_data = pd.read_csv('E:\\Simplilean\\DS with Python\\Data science with Pyth
```

```
C:\Python\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling bac
k to the 'python' engine because the 'c' engine does not support regex separato
rs (separators > 1 char and different from '\s+' are interpreted as regex); you
can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Reading users.dat file with separator '::' and adding columns names as 'UserID', 'Genere', 'Age','Occupation','Zip-code'

In [8]:
```
users_data = pd.read_csv('E:\\Simplilean\\DS with Python\\Data science with Pytho
```

```
C:\Python\lib\site-packages\ipykernel_launcher.py:1: ParserWarning: Falling bac
k to the 'python' engine because the 'c' engine does not support regex separato
rs (separators > 1 char and different from '\s+' are interpreted as regex); you
can avoid this warning by specifying engine='python'.
  """Entry point for launching an IPython kernel.
```

Print rating data of first 10 records

In [9]:
```
rating_data.head(10)
```

Out[9]:

	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
5	1	1197	3	978302268
6	1	1287	5	978302039
7	1	2804	5	978300719
8	1	594	4	978302268
9	1	919	4	978301368

Check the size of rating data frame

In [10]:
```
rating_data.shape
```

Out[10]:
```
(1000209, 4)
```

Print users data of first 10 records

In [11]:
```
users_data.head(10)
```

Out[11]:

	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
5	6	F	50	9	55117
6	7	M	35	1	06810
7	8	M	25	12	11413
8	9	M	25	17	61614
9	10	F	35	1	95370

Check size of users dataframe

In [12]:
```
users_data.shape
```

Out[12]:
```
(6040, 5)
```

Task 2 : Create a new dataset [Master_Data] with the following columns MovieID Title UserID Age Gender Occupation

movie_rating_data :: Merge movies dataframe with rating datafrme on fetaure MovieID and print first 10 records to check. Note: as default join is inner for megre, we're not passing 'how'.

In [13]:
```
movie_rating_data= pd.merge(movie_data,rating_data, on=['MovieID'])
movie_rating_data.head(10)
```

Out[13]:

	MovieID	Title	Genres	UserID	Rating	Timestamp
0	1	Toy Story (1995)	Animation|Children's|Comedy	1	5	978824268
1	1	Toy Story (1995)	Animation|Children's|Comedy	6	4	978237008
2	1	Toy Story (1995)	Animation|Children's|Comedy	8	4	978233496
3	1	Toy Story (1995)	Animation|Children's|Comedy	9	5	978225952
4	1	Toy Story (1995)	Animation|Children's|Comedy	10	5	978226474
5	1	Toy Story (1995)	Animation|Children's|Comedy	18	4	978154768
6	1	Toy Story (1995)	Animation|Children's|Comedy	19	5	978555994
7	1	Toy Story (1995)	Animation|Children's|Comedy	21	3	978139347
8	1	Toy Story (1995)	Animation|Children's|Comedy	23	4	978463614
9	1	Toy Story (1995)	Animation|Children's|Comedy	26	3	978130703

In [14]:
```

rating_data.shape
```

Out[14]: (1000209, 4)

user_rating :: Merge users and rating datafrmes on fetaure UserID and print first 10 records to check. Note: as default join is inner for megre, we're not passing 'how'.

In [15]:
```
user_rating_df = pd.merge(users_data,rating_data, on=['UserID'])
user_rating_df.head(100)
```

Out[15]:

	UserID	Gender	Age	Occupation	Zip-code	MovieID	Rating	Timestamp
0	1	F	1	10	48067	1193	5	978300760
1	1	F	1	10	48067	661	3	978302109
2	1	F	1	10	48067	914	3	978301968
3	1	F	1	10	48067	3408	4	978300275
4	1	F	1	10	48067	2355	5	978824291
...	...	...	...	...	...	...	...	...
95	2	M	56	16	70072	2490	3	978299966
96	2	M	56	16	70072	1834	4	978298813
97	2	M	56	16	70072	3471	5	978298814
98	2	M	56	16	70072	589	4	978299773
99	2	M	56	16	70072	1690	3	978300051

100 rows × 8 columns

print first 10 recors to confim

In [16]:
```
user_rating_df.head(100)
```

Out[16]:

	UserID	Gender	Age	Occupation	Zip-code	MovieID	Rating	Timestamp
0	1	F	1	10	48067	1193	5	978300760
1	1	F	1	10	48067	661	3	978302109
2	1	F	1	10	48067	914	3	978301968
3	1	F	1	10	48067	3408	4	978300275
4	1	F	1	10	48067	2355	5	978824291
...	...	...	...	...	...	...	...	...
95	2	M	56	16	70072	2490	3	978299966
96	2	M	56	16	70072	1834	4	978298813
97	2	M	56	16	70072	3471	5	978298814
98	2	M	56	16	70072	589	4	978299773
99	2	M	56	16	70072	1690	3	978300051

100 rows × 8 columns

master_data :: Merge user_rating and movie_ranting on UserID, MovieID, Rating and projections as MovieID Title

UserID Age Gender Occupation

In [17]:
```
merged_data= pd.merge(user_rating_df,movie_rating_data,
                      on=['UserID', 'MovieID', 'Rating'])

master_data = merged_data[['MovieID', 'Title', 'UserID', 'Age', 'Gender', 'Occupa
```

In [18]:
```
master_data.head(10)
```

Out[18]:

	MovieID	Title	UserID	Age	Gender	Occupation	Rating
0	1193	One Flew Over the Cuckoo's Nest (1975)	1	1	F	10	5
1	661	James and the Giant Peach (1996)	1	1	F	10	3
2	914	My Fair Lady (1964)	1	1	F	10	3
3	3408	Erin Brockovich (2000)	1	1	F	10	4
4	2355	Bug's Life, A (1998)	1	1	F	10	5
5	1197	Princess Bride, The (1987)	1	1	F	10	3
6	1287	Ben-Hur (1959)	1	1	F	10	5
7	2804	Christmas Story, A (1983)	1	1	F	10	5
8	594	Snow White and the Seven Dwarfs (1937)	1	1	F	10	4
9	919	Wizard of Oz, The (1939)	1	1	F	10	4

Task 3: Explore the datasets using visual representations (graphs or tables)

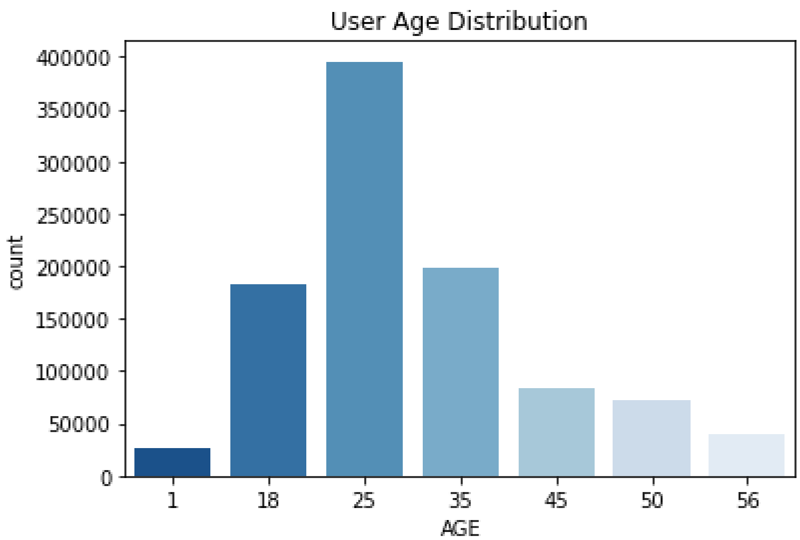
In [19]:
```
master_data['Age']
```

Out[19]:
```
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
```

3.1 : User Age Distribution

In [20]:
```
import matplotlib.pyplot as plt
import seaborn as sn
%matplotlib inline
ax = sn.countplot(x='Age', data=master_data, palette='Blues_r')
plt.xlabel('AGE')
plt.title('User Age Distribution')
```

Out[20]: Text(0.5, 1.0, 'User Age Distribution')



3.2 . User rating of the movie “Toy Story”

Step1:: Grouping master_data by MovieID.

Step2:: Then get group of 1 which is Toy Story

Step3:: Use result dataframe in for visualtion of 'User rating of the movie Toy Story'

In [21]:
```
movies_grouped= master_data.groupby('MovieID')
len(movies_grouped)
```

Out[21]: 3706

In [22]:
```
user_rating_for_ToyStory= movies_grouped.get_group(1)
```

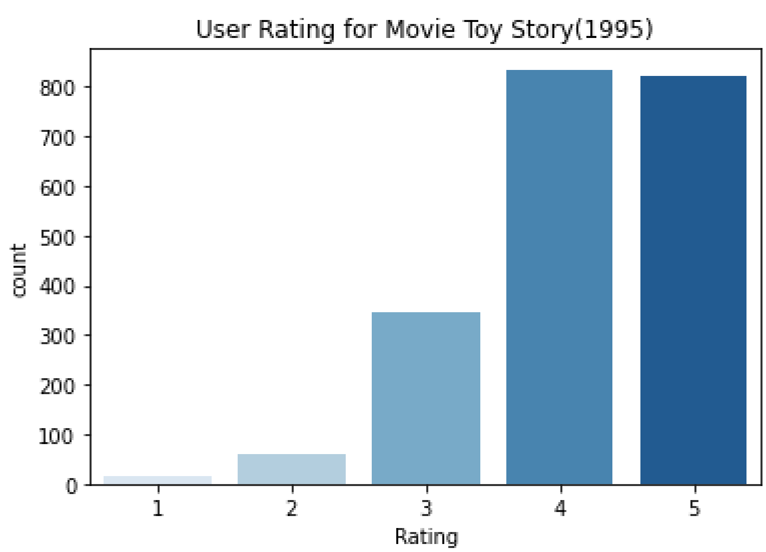
In [23]:
```
user_rating_for_ToyStory.head(10)
```

Out[23]:

	MovieID	Title	UserID	Age	Gender	Occupation	Rating
40	1	Toy Story (1995)	1	1	F	10	5
469	1	Toy Story (1995)	6	50	F	9	4
581	1	Toy Story (1995)	8	25	M	12	4
711	1	Toy Story (1995)	9	25	M	17	5
837	1	Toy Story (1995)	10	35	F	1	5
1966	1	Toy Story (1995)	18	18	F	3	4
2276	1	Toy Story (1995)	19	1	M	10	5
2530	1	Toy Story (1995)	21	18	M	16	3
2870	1	Toy Story (1995)	23	35	M	0	4
3405	1	Toy Story (1995)	26	25	M	7	3

In [24]:
```
ax = sn.countplot(x='Rating', data=user_rating_for_ToyStory, palette='Blues')
plt.xlabel('Rating')
plt.title('User Rating for Movie Toy Story(1995)')
```

Out[24]: Text(0.5, 1.0, 'User Rating for Movie Toy Story(1995)')



3.3 Top 25 movies by viewership rating

Step1: Group maser_data with rating and reset index

Step2: Check rating wise movies to take top 25 rating movies.

Step3: take top 25 , 5 rated movies

Step4: Plot the data

In [25]:
```
movies_rating_grouped = master_data.groupby(['Rating']).size().reset_index()
```

In [26]:
```
movies_rating_grouped
```

Out[26]:

	Rating	0
0	1	56174
1	2	107557
2	3	261197
3	4	348971
4	5	226310

In [27]:
```
top25movies= master_data['Title'][master_data['Rating']==5].value_counts().head(2
```

In [28]:
```
top25movies
```

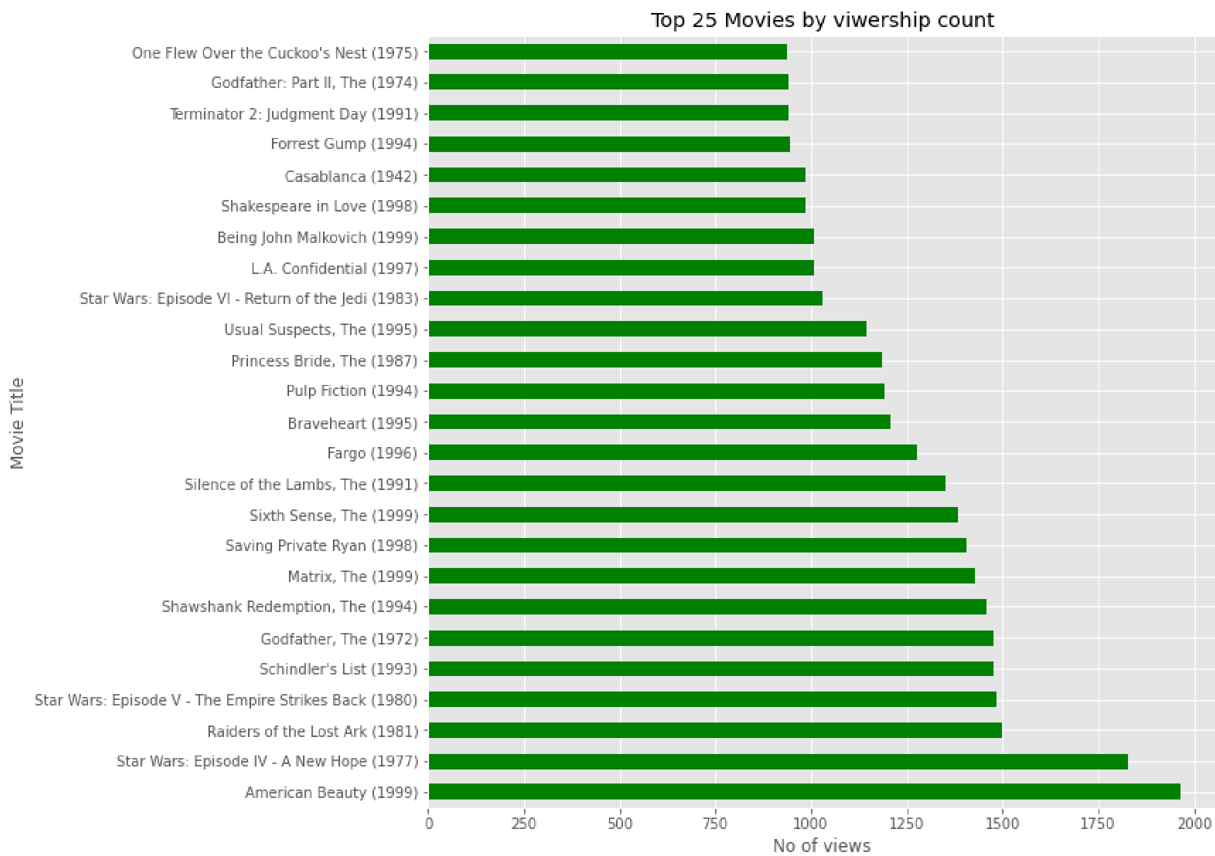
Out[28]:
```
American Beauty (1999)                                   1963
Star Wars: Episode IV - A New Hope (1977)                1826
Raiders of the Lost Ark (1981)                           1500
Star Wars: Episode V - The Empire Strikes Back (1980)    1483
Schindler's List (1993)                                  1475
Godfather, The (1972)                                    1475
Shawshank Redemption, The (1994)                         1457
Matrix, The (1999)                                       1430
Saving Private Ryan (1998)                               1405
Sixth Sense, The (1999)                                  1385
Silence of the Lambs, The (1991)                         1350
Fargo (1996)                                             1278
Braveheart (1995)                                        1206
Pulp Fiction (1994)                                      1193
Princess Bride, The (1987)                               1186
Usual Suspects, The (1995)                               1144
Star Wars: Episode VI - Return of the Jedi (1983)        1028
L.A. Confidential (1997)                                 1009
Being John Malkovich (1999)                              1007
Shakespeare in Love (1998)                                987
Casablanca (1942)                                         984
Forrest Gump (1994)                                       945
Terminator 2: Judgment Day (1991)                         942
Godfather: Part II, The (1974)                            941
One Flew Over the Cuckoo's Nest (1975)                    937
Name: Title, dtype: int64
```

In [29]:
```
from matplotlib import style

style.use('ggplot')

plt.figure(figsize=(10,10))
plt.ylabel("Movie Title")
plt.xlabel("No of views")
plt.title("Top 25 Movies by viwership count")
top25movies.plot(kind="barh" , color='g')
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x13b530f0>



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

Step1: Group maser_data with UserID

Step2: Check Userid 2526 using get_group

Step4: Plot the data

In [30]:
```
movies_Userid_grouped = master_data.groupby(['UserID'])
```

In [31]:
```
user_2696_movies_ratings= movies_Userid_grouped.get_group(2696)
```

In [32]:
```
user_2696_movies_ratings.head(10)
```

Out[32]:

	MovieID	Title	UserID	Age	Gender	Occupation	Rating
440667	1258	Shining, The (1980)	2696	25	M	7	4
440668	1270	Back to the Future (1985)	2696	25	M	7	2
440669	1617	L.A. Confidential (1997)	2696	25	M	7	4
440670	1625	Game, The (1997)	2696	25	M	7	4
440671	1644	I Know What You Did Last Summer (1997)	2696	25	M	7	2
440672	1645	Devil's Advocate, The (1997)	2696	25	M	7	4
440673	1805	Wild Things (1998)	2696	25	M	7	4
440674	1892	Perfect Murder, A (1998)	2696	25	M	7	4
440675	800	Lone Star (1996)	2696	25	M	7	5
440676	2338	I Still Know What You Did Last Summer (1998)	2696	25	M	7	2

In [33]:
```
master_data.columns
```

Out[33]:
```
Index(['MovieID', 'Title', 'UserID', 'Age', 'Gender', 'Occupation', 'Rating'],
dtype='object')
```

4.1 Feature Engineeing : Find out all the unique genres

Step1: Split geners using '|'

Step2: Create new columns/features using Generes split.

Step3: Append same Master data and check.

In [34]:
```
master_data.head(10)
```

Out[34]:

	MovieID	Title	UserID	Age	Gender	Occupation	Rating
0	1193	One Flew Over the Cuckoo's Nest (1975)	1	1	F	10	5
1	661	James and the Giant Peach (1996)	1	1	F	10	3
2	914	My Fair Lady (1964)	1	1	F	10	3
3	3408	Erin Brockovich (2000)	1	1	F	10	4
4	2355	Bug's Life, A (1998)	1	1	F	10	5
5	1197	Princess Bride, The (1987)	1	1	F	10	3
6	1287	Ben-Hur (1959)	1	1	F	10	5
7	2804	Christmas Story, A (1983)	1	1	F	10	5
8	594	Snow White and the Seven Dwarfs (1937)	1	1	F	10	4
9	919	Wizard of Oz, The (1939)	1	1	F	10	4

In [35]:
```
feature_task_master_data= merged_data[['Genres', 'Age', 'Gender', 'Rating']]
```

In [36]:
```
feature_task_master_data.head(10)
```

Out[36]:

	Genres	Age	Gender	Rating
0	Drama	1	F	5
1	Animation|Children's|Musical	1	F	3
2	Musical|Romance	1	F	3
3	Drama	1	F	4
4	Animation|Children's|Comedy	1	F	5
5	Action|Adventure|Comedy|Romance	1	F	3
6	Action|Adventure|Drama	1	F	5
7	Comedy|Drama	1	F	5
8	Animation|Children's|Musical	1	F	4
9	Adventure|Children's|Drama|Musical	1	F	4

In [37]:
```
feature_task_master_data['Genres'].str.split("|", expand = True)
```

Out[37]:

	0	1	2	3	4	5
0	Drama	None	None	None	None	None
1	Animation	Children's	Musical	None	None	None
2	Musical	Romance	None	None	None	None
3	Drama	None	None	None	None	None
4	Animation	Children's	Comedy	None	None	None
...	...	...	...	...	...	...
1000204	Comedy	None	None	None	None	None
1000205	Drama	Romance	War	None	None	None
1000206	Comedy	Drama	None	None	None	None
1000207	Drama	None	None	None	None	None
1000208	Children's	Drama	Fantasy	Sci-Fi	None	None

1000209 rows × 6 columns

In [38]:
```
feature_task_master_data[['Genres1','Genres2','Genres3','Genres4','Genres5','Genr
```

```
C:\Python\lib\site-packages\pandas\core\frame.py:3509: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydat
a.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-cop
y)
  self[k1] = value[k2]
```

In [39]:
```
feature_task_master_data.pop('Genres')
feature_task_master_data.head()
```

Out[39]:

	Age	Gender	Rating	Genres1	Genres2	Genres3	Genres4	Genres5	Genres6
0	1	F	5	Drama	None	None	None	None	None
1	1	F	3	Animation	Children's	Musical	None	None	None
2	1	F	3	Musical	Romance	None	None	None	None
3	1	F	4	Drama	None	None	None	None	None
4	1	F	5	Animation	Children's	Comedy	None	None	None

4.2. Feature Engineering : 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.

Step1: Check all unique genres in Data frame

Step2: Pass Genres columns to get_dummies in pandas to get encoded.

Step3: Check the data for confirmation

In [40]:
```
pd.unique(feature_task_master_data[['Genres1','Genres2','Genres3','Genres4','Genr
```

Out[40]:
```
['Drama',
 None,
 'Animation',
 "Children's",
 'Musical',
 'Romance',
 'Comedy',
 'Action',
 'Adventure',
 'Fantasy',
 'Sci-Fi',
 'War',
 'Thriller',
 'Crime',
 'Mystery',
 'Western',
 'Horror',
 'Film-Noir',
 'Documentary']
```

In [41]:
```
one_hot_columns=['Genres1','Genres2','Genres3','Genres4','Genres5','Genres6','Ger

one_hot_encoding_feature_task_master_data = pd.get_dummies(feature_task_master_da
```

In [42]:
```
one_hot_encoding_feature_task_master_data
```

Out[42]:

	Age	Rating	Genres1_Adventure	Genres1_Animation	Genres1_Children's	Genres1_Come
0	1	5	0	0	0	
1	1	3	0	1	0	
2	1	3	0	0	0	
3	1	4	0	0	0	
4	1	5	0	1	0	
...	...	...	...	...	...	
1000204	25	1	0	0	0	
1000205	25	5	0	0	0	
1000206	25	5	0	0	0	
1000207	25	4	0	0	0	
1000208	25	4	0	0	1	

1000209 rows × 67 columns

4.3 Feature Engineering: Determine the features affecting the ratings of any particular movie.

Step1: Check all columns in new encoded data frame

Step2: Create X, y from data frames

Step3: Split train and test using train_test_split

Step4: Import PCA lib

Step5: Create PCA and fit transform the X_train,y_train

Step6: now transform on X_test also

Step7: Print explained_variance_ratio and components_ to check the variance

In [43]:
```
one_hot_encoding_feature_task_master_data.columns
```

Out[43]:
```
Index(['Age', 'Rating', 'Genres1_Adventure', 'Genres1_Animation',
       'Genres1_Children's', 'Genres1_Comedy', 'Genres1_Crime',
       'Genres1_Documentary', 'Genres1_Drama', 'Genres1_Fantasy',
       'Genres1_Film-Noir', 'Genres1_Horror', 'Genres1_Musical',
       'Genres1_Mystery', 'Genres1_Romance', 'Genres1_Sci-Fi',
       'Genres1_Thriller', 'Genres1_War', 'Genres1_Western',
       'Genres2_Animation', 'Genres2_Children's', 'Genres2_Comedy',
       'Genres2_Crime', 'Genres2_Documentary', 'Genres2_Drama',
       'Genres2_Fantasy', 'Genres2_Film-Noir', 'Genres2_Horror',
       'Genres2_Musical', 'Genres2_Mystery', 'Genres2_Romance',
       'Genres2_Sci-Fi', 'Genres2_Thriller', 'Genres2_War', 'Genres2_Western',
       'Genres3_Children's', 'Genres3_Comedy', 'Genres3_Crime',
       'Genres3_Drama', 'Genres3_Fantasy', 'Genres3_Film-Noir',
       'Genres3_Horror', 'Genres3_Musical', 'Genres3_Mystery',
       'Genres3_Romance', 'Genres3_Sci-Fi', 'Genres3_Thriller', 'Genres3_War',
       'Genres3_Western', 'Genres4_Comedy', 'Genres4_Crime', 'Genres4_Drama',
       'Genres4_Fantasy', 'Genres4_Horror', 'Genres4_Musical',
       'Genres4_Mystery', 'Genres4_Romance', 'Genres4_Sci-Fi',
       'Genres4_Thriller', 'Genres4_War', 'Genres4_Western', 'Genres5_Musical',
       'Genres5_Romance', 'Genres5_Sci-Fi', 'Genres5_Thriller', 'Genres5_War',
       'Gender_M'],
      dtype='object')
```

In [44]:
```
one_hot_encoding_feature_task_master_data.head()
```

Out[44]:

	Age	Rating	Genres1_Adventure	Genres1_Animation	Genres1_Children's	Genres1_Comedy	Ge
0	1	5	0	0	0	0	
1	1	3	0	1	0	0	
2	1	3	0	0	0	0	
3	1	4	0	0	0	0	
4	1	5	0	1	0	0	

5 rows × 67 columns

Spliting data sets as X, y for classification problem

In [45]:
```
y = one_hot_encoding_feature_task_master_data['Rating']
X=one_hot_encoding_feature_task_master_data.drop('Rating', axis=1)
```

In [46]:
```
X
```

Out[46]:

	Age	Genres1_Adventure	Genres1_Animation	Genres1_Children's	Genres1_Comedy	Genr
0	1	0	0	0	0	
1	1	0	1	0	0	
2	1	0	0	0	0	
3	1	0	0	0	0	
4	1	0	1	0	0	
...	...	...	...	...	...	
1000204	25	0	0	0	1	
1000205	25	0	0	0	0	
1000206	25	0	0	0	1	
1000207	25	0	0	0	0	
1000208	25	0	0	1	0	

1000209 rows × 66 columns

In [47]:
```
y
```

Out[47]:
```
0          5
1          3
2          3
3          4
4          5
          ..
1000204    1
1000205    5
1000206    5
1000207    4
1000208    4
Name: Rating, Length: 1000209, dtype: int64
```

Spliting train and test on 70,30 %

In [48]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y, test_size=0.3, random_state=4
```

Applying PCA on X, y

In [49]:
```
from sklearn.decomposition import PCA
pca= PCA(n_components=2)
X_train= pca.fit_transform(X_train)
X_test = pca.transform(X_test)
explained_variance_ratio = pca.explained_variance_ratio_
```

In [50]:
```
explained_variance_ratio
```

Out[50]:
```
array([0.98745495, 0.00183069])
```

In [51]:
```
pca.components_
```

Out[51]:
```
array([[-9.99995840e-01, -1.03840736e-05,  7.06964790e-04,
         2.60203002e-04,  8.43302187e-04, -5.73936968e-05,
        -3.16484393e-05, -2.13609710e-03,  8.84093551e-07,
        -2.15101456e-04,  2.53097849e-04, -2.25916528e-04,
        -1.48419310e-04, -3.86549053e-05, -9.75234717e-05,
        -4.83283012e-05, -6.38830962e-05, -1.90867957e-04,
         9.27599592e-05,  8.01652518e-04,  4.50400260e-04,
         7.91421088e-05,  8.89302438e-07, -3.84228341e-04,
         8.07383357e-05, -1.27872339e-04,  2.07039159e-04,
        -1.25042586e-04, -2.71497217e-04, -4.30172900e-04,
         1.04702314e-04,  2.14976156e-04, -4.72311641e-04,
        -1.64527100e-04,  7.75905876e-05,  4.08406801e-04,
         1.09229618e-04, -8.01351528e-05,  2.50971580e-04,
        -4.15755891e-05,  2.19528882e-05,  1.57205927e-04,
         5.12146133e-06, -1.21095539e-04,  1.93575959e-04,
         2.61226708e-04, -3.41166573e-04, -7.01429669e-05,
         4.17185281e-05,  3.58813857e-05,  4.41521224e-05,
         4.33900058e-05,  7.37419807e-05,  6.72657817e-05,
        -6.12449301e-06,  2.88193358e-05,  1.24967530e-04,
         3.10561656e-05, -7.70040331e-07, -2.78911663e-05,
         1.39169706e-05,  2.65680772e-05,  4.53542612e-05,
         2.17854061e-05,  3.16384215e-05,  1.34943851e-04],
       [ 1.61708662e-03, -2.44933589e-02, -2.68083477e-02,
        -9.03053197e-03,  7.92778231e-01, -1.33692454e-03,
        -3.03889252e-03, -5.37679468e-01, -4.56168458e-04,
        -6.05058611e-03, -3.12185947e-02, -2.55829587e-03,
        -7.91091154e-03, -1.42933413e-03, -7.94763855e-03,
        -8.75961038e-03, -4.36680029e-04, -2.61278128e-03,
        -3.28256565e-03, -3.46303111e-02, -3.05923223e-02,
         1.94604144e-02,  2.90441660e-03,  2.32146973e-01,
         1.32305834e-02, -4.55234377e-03,  2.15263533e-02,
         6.49582805e-03, -2.32526176e-02,  3.45190488e-02,
        -2.79449659e-02, -1.07036829e-01, -3.75302043e-02,
         3.33490133e-03, -1.95222865e-03, -2.06065071e-02,
        -5.13623785e-03, -7.27756852e-03, -5.59411250e-03,
        -3.49607802e-04, -4.36723601e-04,  1.08912688e-03,
         5.78060085e-05,  5.21817070e-02, -3.20493489e-02,
        -2.57546645e-02, -6.83747854e-03,  3.73519300e-03,
        -8.91910598e-04, -1.46118386e-03, -3.45072028e-03,
        -2.41044039e-03, -1.80531260e-03, -5.46415874e-03,
         1.60968521e-03, -3.38139922e-03, -9.98763829e-04,
        -1.35203546e-02, -3.31904287e-03, -2.14792239e-03,
        -2.87612253e-04, -1.59158318e-03, -1.70616970e-03,
        -5.94025201e-04, -2.52715784e-03, -4.41551505e-02]])
```

4.4 Feature Engineering: Develop an appropriate model to predict the movie ratings

As to be predicted data is ratings, this will be multi class classification problem (Classes : 1,2,3,4,5)

So , Will apply different classifcation models and checks for optimsed model using f1 score

Logistic Regression

SVC with default kernel = rbf

Linear SVC

Gaussin Naive bayes

Decicision Tree classfiier

Randome Forest classifier

In [55]:
```
#Logistic regression

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[55]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [56]:
```
y_pred = classifier.predict(X_test)
y_pred
```

Out[56]:
```
array([4, 4, 4, ..., 4, 4, 4], dtype=int64)
```

Checking Confusion matrix for prection

In [57]:
```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm
```

Out[57]:
```
array([[     0,      0,      0,  17015,      0],
       [     0,      0,      0,  32061,      0],
       [     0,      0,      0,  78351,      0],
       [     0,      0,      0, 104518,      0],
       [     0,      0,      0,  68118,      0]], dtype=int64)
```

Finding F1 Score Logistic regression

In [58]:
```
from sklearn.metrics import f1_score
logistic_classfier_score = round(f1_score(y_test, y_pred , average='weighted'),2)
logistic_classfier_score
```

Out[58]: 0.18

Support vector classififier with default kernel rbf

In [63]:
```
from sklearn.svm import SVC
```

In [ ]:
```
# Support vector classififier

svc= SVC()
svc.fit(X_train,y_train)
svc.score(X_test,y_test)
svc_y_predict= svc-predict(X_test)
svc_classfier_score = round(f1_score(y_test, svc_y_predict , average='weighted'),
svc_classfier_score
```

LinearSVC Classifier

In [ ]:
```
# Support vector classififier :: Linear
svc_linear= SVC(kernel='linear')
svc_linear.fit(X_train,y_train)
svc_liner_y_pred = svc_linear.predict(X_test)
svc_linera_score = round(f1_score(y_test, svc_liner_y_pred , average='weighted'),
svc_linera_score
```

Naive Bayes Classifier

In [59]:
```
# Naive Bayes
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)
# accuracy on X_test
nb_y_pred = gnb.predict(X_test)
snb_classfier_score = round(f1_score(y_test, nb_y_pred , average='weighted'),2)
snb_classfier_score
```

Out[59]: 0.18

Decision Tree Classifier

In [60]:
```
# Decision Tree Classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
dt = DecisionTreeClassifier(random_state=1)
dt.fit(X_train,y_train)
dt_y_predict= dt.predict(X_test)
dt_classfier_score = round(f1_score(y_test, dt_y_predict , average='weighted'),2)
dt_classfier_score
```

Out[60]: 0.29

Random forest classifier

In [62]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators=10)
rfc.fit(X_train, y_train)
rf_y_predict= rfc.predict(X_test)
rft_classfier_score = round(f1_score(y_test, rf_y_predict , average='weighted'),2
rft_classfier_score
```

Out[62]: 0.29

Print All F1 scores to find suitable model

In [ ]:
```
prrint('Random forest classifier::',rft_classfier_score )
prrint('Decision Tree classifier::',dt_classfier_score )
prrint('Gaussian Naive Bayes classifier::',snb_classfier_score )
prrint('Logistic regression::',logistic_classfier_score )
prrint('Support vector classifier::',svc_classfier_score )
prrint('Linera Support Vector classifier::',svc_linera_score )
```

In [ ]: