Assignment 1C - Question 1

Clustering and Recommendations

In [1]:

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from scipy.spatial import distance
import matplotlib.pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.cluster import AgglomerativeClustering
from matplotlib import cm
from datetime import datetime
from sklearn.feature_extraction.text import CountVectorizer
import pickle
import ast
```

In [2]:

```
links = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/links.csv')
movies_metadata = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/movies.csv')
ratings = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/ratings.csv')
tags = pd.read_csv('CAB420_Assessment_1C_Data/Data/Q1/tags.csv')
```

In [3]:

```
movies_metadata.columns
```

Out[3]:

```
Index(['movieId', 'title', 'genres'], dtype='object')
```

In [4]:

```
print(movies_metadata)
```

```
      movieId                                     title  \
0           1                         Toy Story (1995)
1           2                           Jumanji (1995)
2           3                  Grumpier Old Men (1995)
3           4                 Waiting to Exhale (1995)
4           5       Father of the Bride Part II (1995)
...       ...                                       ...
9737   193581  Black Butler: Book of the Atlantic (2017)
9738   193583              No Game No Life: Zero (2017)
9739   193585                             Flint (2017)
9740   193587       Bungo Stray Dogs: Dead Apple (2018)
9741   193609       Andrew Dice Clay: Dice Rules (1991)

                                       genres
0     Adventure|Animation|Children|Comedy|Fantasy
1                      Adventure|Children|Fantasy
2                                Comedy|Romance
3                          Comedy|Drama|Romance
4                                        Comedy
...                                         ...
9737              Action|Animation|Comedy|Fantasy
9738                     Animation|Comedy|Fantasy
9739                                       Drama
9740                           Action|Animation
9741                                     Comedy

[9742 rows x 3 columns]
```

Filter ratings for 4+

In [5]:

```
print(ratings)
```

```
        userId  movieId  rating   timestamp
0            1        1     4.0   964982703
1            1        3     4.0   964981247
2            1        6     4.0   964982224
3            1       47     5.0   964983815
4            1       50     5.0   964982931
...        ...      ...     ...         ...
100831     610   166534     4.0  1493848402
100832     610   168248     5.0  1493850091
100833     610   168250     5.0  1494273047
100834     610   168252     5.0  1493846352
100835     610   170875     3.0  1493846415

[100836 rows x 4 columns]
```

In [6]:

```
ratings_4 = ratings[ratings['rating'] >= 4.0]
```

In [7]:

```
movies_list = np.unique(ratings['movieId'])[:200]
ratings = ratings.loc[ratings['movieId'].isin(movies_list)]
print('Shape of ratings dataset is: ',ratings.shape, '\n')
print('Max values in dataset are \n',ratings.max(), '\n')
print('Min values in dataset are \n',ratings.min(), '\n')
```

```
Shape of ratings dataset is:  (6351, 4)

Max values in dataset are
 userId       6.100000e+02
movieId      2.330000e+02
rating       5.000000e+00
timestamp    1.537799e+09
dtype: float64

Min values in dataset are
 userId               1.0
movieId              1.0
rating               0.5
timestamp    828124615.0
dtype: float64
```

In [8]:

```
users_list = np.unique(ratings['userId'])[:315]
ratings = ratings.loc[ratings['userId'].isin(users_list)]
print('Shape of ratings dataset is: ',ratings.shape, '\n')
print('Max values in dataset are \n',ratings.max(), '\n')
print('Min values in dataset are \n',ratings.min(), '\n')
print('Total Users: ', np.unique(ratings['userId']).shape[0])
print('Total Movies which are rated by 100 users: ', np.unique(ratings['movieId']).shap
e[0])
```

```
Shape of ratings dataset is:  (3505, 4)

Max values in dataset are
 userId       3.480000e+02
movieId      2.330000e+02
rating       5.000000e+00
timestamp    1.537158e+09
dtype: float64

Min values in dataset are
 userId               1.0
movieId              1.0
rating               0.5
timestamp    829322340.0
dtype: float64

Total Users:  315
Total Movies which are rated by 100 users:  187
```

In [9]:

```
users_fav_movies = ratings.loc[:, ['userId', 'movieId']]
```

In [10]:

```
users_fav_movies = ratings.reset_index(drop = True)
```

In [11]:

```
users_fav_movies.T
```

Out[11]:

	0	1	2	3	4	5
userId	1.0	1.0	1.0	1.0	1.0	1.0
movieId	1.0	3.0	6.0	47.0	50.0	70.0
rating	4.0	4.0	4.0	5.0	5.0	3.0
timestamp	964982703.0	964981247.0	964982224.0	964983815.0	964982931.0	964982400.0

4 rows × 3505 columns

In [12]:

```
def moviesListForUsers(users, users_data):
    # users = a list of users IDs
    # users_data = a dataframe of users favourite movies or users watched movies
    users_movies_list = []
    for user in users:
        users_movies_list.append(str(list(users_data[users_data['userId'] == user]['movieId'])).split('[')[1].split(']')[0])
    return users_movies_list
```

In [13]:

```
users = np.unique(users_fav_movies['userId'])
print(users.shape)
```

```
(315,)
```

In [14]:

```
users_movies_list = moviesListForUsers(users, users_fav_movies)
print('Movies list for', len(users_movies_list), ' users')
print('A list of first 10 users favourite movies: \n', users_movies_list[:10])
```

```
Movies list for 315  users
A list of first 10 users favourite movies:
 ['1, 3, 6, 47, 50, 70, 101, 110, 151, 157, 163, 216, 223, 231', '31', '2
1, 32, 45, 47, 52, 58, 106, 125, 126, 162, 171, 176, 190, 215, 222, 232',
'1, 21, 34, 36, 39, 50, 58, 110, 150, 153, 232', '2, 3, 4, 5, 6, 7, 8, 10,
11, 13, 15, 16, 17, 19, 21, 22, 24, 25, 26, 27, 31, 32, 34, 36, 41, 43, 4
5, 46, 47, 50, 54, 60, 61, 62, 65, 66, 76, 79, 86, 87, 88, 89, 92, 93, 95,
100, 102, 104, 105, 110, 112, 113, 126, 135, 140, 141, 145, 146, 150, 151,
153, 158, 159, 160, 161, 163, 165, 168, 170, 171, 174, 177, 179, 180, 181,
185, 186, 189, 191, 195, 196, 201, 204, 205, 207, 208, 209, 210, 212, 216,
217, 218, 219, 222, 224, 225, 230, 231', '1, 50, 58, 150, 165', '2, 10, 1
1, 21, 32, 34, 39, 47, 50, 110, 141, 150, 153, 185, 186, 208, 231', '41, 1
87, 223', '6, 10, 36, 44, 95, 110, 150, 153, 165, 170, 208', '39, 168, 22
2']
```

In [15]:

```
def prepSparseMatrix(list_of_str):
    # list_of_str = A list, which contain strings of users favourite movies separate by
comma ",".
    # It will return us sparse matrix and feature names on which sparse matrix is defin
ed
    # i.e. name of movies in the same order as the column of sparse matrix
    cv = CountVectorizer(token_pattern = r'[^\,\ ]+', lowercase = False)
    sparseMatrix = cv.fit_transform(list_of_str)
    return sparseMatrix.toarray(), cv.get_feature_names()
```

In [16]:

```
sparseMatrix, feature_names = prepSparseMatrix(users_movies_list)
```

In [17]:

```
df_sparseMatrix = pd.DataFrame(sparseMatrix, index = users, columns = feature_names)
df_sparseMatrix
```

Out[17]:

	1	10	100	101	102	104	105	106	107	11	...	87	88	89	9	92	93	94	95	97
1	1	0	0	1	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
4	0	0	0	0	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	(
5	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
6	0	1	1	0	1	1	1	0	0	1	...	1	1	1	0	1	1	0	1	(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	..
344	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
345	0	0	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	(
346	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(
347	1	1	0	0	0	0	0	0	0	1	...	0	0	0	0	0	0	0	0	(
348	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	(

315 rows × 187 columns

In [18]:

```
first_6_users_SM = users_fav_movies[users_fav_movies['userId'].isin(users[:6])].sort_values('userId')
first_6_users_SM.T
```

Out[18]:

	0	13	12	10	9	8
userId	1.0	1.0	1.0	1.0	1.0	1.0
movieId	1.0	231.0	223.0	163.0	157.0	151.0
rating	4.0	5.0	3.0	5.0	5.0	5.0
timestamp	964982703.0	964981179.0	964980985.0	964983650.0	964984100.0	964984041.0

4 rows × 145 columns

In [19]:

```
df_sparseMatrix.loc[np.unique(first_6_users_SM['userId']), list(map(str, np.unique(first_6_users_SM['movieId'])))]
```

Out[19]:

	1	2	3	4	5	6	7	8	10	11	...	217	218	219	222	223	224	225	230	231	232
1	1	0	1	0	0	1	0	0	0	0	...	0	0	0	0	1	0	0	0	1	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0	0	0	0	0	1
5	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	1
6	0	1	1	1	1	1	1	1	1	1	...	1	1	1	1	0	1	1	1	1	0
7	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

6 rows × 113 columns

In [20]:

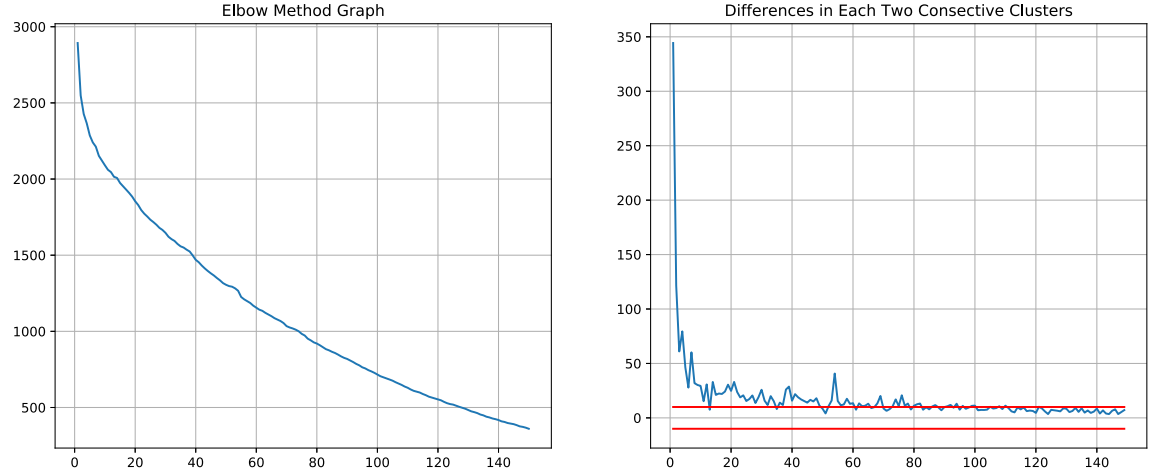
```
class elbowMethod():
    def __init__(self, sparseMatrix):
        self.sparseMatrix = sparseMatrix
        self.wcss = list()
        self.differences = list()
    def run(self, init, upto, max_iterations = 300):
        for i in range(init, upto + 1):
            kmeans = KMeans(n_clusters=i, init = 'k-means++', max_iter = max_iterations
, n_init = 10, random_state = 0)
            kmeans.fit(sparseMatrix)
            self.wcss.append(kmeans.inertia_)
        self.differences = list()
        for i in range(len(self.wcss)-1):
            self.differences.append(self.wcss[i] - self.wcss[i+1])
    def showPlot(self, boundary = 500, upto_cluster = None):
        if upto_cluster is None:
            WCSS = self.wcss
            DIFF = self.differences
        else:
            WCSS = self.wcss[:upto_cluster]
            DIFF = self.differences[:upto_cluster - 1]
        plt.figure(figsize=(15, 6))
        plt.subplot(121).set_title('Elbow Method Graph')
        plt.plot(range(1, len(WCSS) + 1), WCSS)
        plt.grid(b = True)
        plt.subplot(122).set_title('Differences in Each Two Consective Clusters')
        len_differences = len(DIFF)
        X_differences = range(1, len_differences + 1)
        plt.plot(X_differences, DIFF)
        plt.plot(X_differences, np.ones(len_differences)*boundary, 'r')
        plt.plot(X_differences, np.ones(len_differences)*(-boundary), 'r')
        plt.grid()
        plt.show()
```

In [21]:

```
elbow_method = elbowMethod(sparseMatrix)
```

In [22]:

```
elbow_method.run(1, 150)
elbow_method.showPlot(boundary = 10)
```



In [23]:

```
kmeans = KMeans(n_clusters=82, init = 'k-means++', max_iter = 300, n_init = 10, random_
state = 0)
clusters = kmeans.fit_predict(sparseMatrix)
```

In [24]:

```
users_cluster = pd.DataFrame(np.concatenate((users.reshape(-1,1), clusters.reshape(-1,1
)), axis = 1), columns = ['userId', 'Cluster'])
users_cluster.T
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	3
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	3
Cluster	13	0	74	77	7	5	68	0	24	0	...	3	9	15	0	5	38	25	32	

2 rows × 315 columns

In [25]:

```
def clustersMovies(users_cluster, users_data):
    clusters = list(users_cluster['Cluster'])
    each_cluster_movies = list()
    for i in range(len(np.unique(clusters))):
        users_list = list(users_cluster[users_cluster['Cluster'] == i]['userId'])
        users_movies_list = list()
        for user in users_list:
            users_movies_list.extend(list(users_data[users_data['userId'] == user]['mov
ieId']))
        users_movies_counts = list()
        users_movies_counts.extend([[movie, users_movies_list.count(movie)] for movie i
n np.unique(users_movies_list)])
        each_cluster_movies.append(pd.DataFrame(users_movies_counts, columns=['movieId'
, 'Count']).sort_values(by = ['Count'], ascending = False).reset_index(drop=True))
    return each_cluster_movies
cluster_movies = clustersMovies(users_cluster, users_fav_movies)
```

In [26]:

```
cluster_movies[1].T
```

Out[26]:

	0	1	2	3	4	5	6	7	8	9	...	23	24	25	26	27	28	29
movieId	1	34	231	110	165	153	32	47	10	208	...	173	170	163	161	158	29	3
Count	5	5	5	4	4	4	4	4	4	3	...	1	1	1	1	1	1	1

2 rows × 33 columns

In [27]:

```
for i in range(42):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    print('Users in Cluster ' + str(i) + ' -> ', len_users)
```

```
Users in Cluster 0 ->  52
Users in Cluster 1 ->  5
Users in Cluster 2 ->  1
Users in Cluster 3 ->  7
Users in Cluster 4 ->  8
Users in Cluster 5 ->  34
Users in Cluster 6 ->  1
Users in Cluster 7 ->  1
Users in Cluster 8 ->  1
Users in Cluster 9 ->  10
Users in Cluster 10 ->  1
Users in Cluster 11 ->  1
Users in Cluster 12 ->  1
Users in Cluster 13 ->  23
Users in Cluster 14 ->  5
Users in Cluster 15 ->  27
Users in Cluster 16 ->  1
Users in Cluster 17 ->  1
Users in Cluster 18 ->  1
Users in Cluster 19 ->  1
Users in Cluster 20 ->  1
Users in Cluster 21 ->  1
Users in Cluster 22 ->  1
Users in Cluster 23 ->  1
Users in Cluster 24 ->  4
Users in Cluster 25 ->  10
Users in Cluster 26 ->  1
Users in Cluster 27 ->  1
Users in Cluster 28 ->  1
Users in Cluster 29 ->  3
Users in Cluster 30 ->  1
Users in Cluster 31 ->  1
Users in Cluster 32 ->  16
Users in Cluster 33 ->  1
Users in Cluster 34 ->  1
Users in Cluster 35 ->  1
Users in Cluster 36 ->  1
Users in Cluster 37 ->  4
Users in Cluster 38 ->  21
Users in Cluster 39 ->  1
Users in Cluster 40 ->  1
Users in Cluster 41 ->  1
```

In [28]:

```
def getMoviesOfUser(user_id, users_data):
    return list(users_data[users_data['userId'] == user_id]['movieId'])
```

In [29]:

```
def fixClusters(clusters_movies_dataframes, users_cluster_dataframe, users_data, smalle
st_cluster_size = 11):
    # clusters_movies_dataframes: will be a list which will contain each dataframes of
 each cluster movies
    # users_cluster_dataframe: will be a dataframe which contain users IDs and their cl
uster no.
    # smallest_cluster_size: is a smallest cluster size which we want for a cluster to
 not remove
    each_cluster_movies = clusters_movies_dataframes.copy()
    users_cluster = users_cluster_dataframe.copy()
    # Let convert dataframe in each_cluster_movies to list with containing only movies
 IDs
    each_cluster_movies_list = [list(df['movieId']) for df in each_cluster_movies]
    # First we will prepair a list which containt lists of users in each cluster -> [[C
luster 0 Users], [Cluster 1 Users], ... ,[Cluster N Users]]
    usersInClusters = list()
    total_clusters = len(each_cluster_movies)
    for i in range(total_clusters):
        usersInClusters.append(list(users_cluster[users_cluster['Cluster'] == i]['userI
d']))
    uncategorizedUsers = list()
    i = 0
    # Now we will remove small clusters and put their users into another list named "un
categorizedUsers"
    # Also when we will remove a cluster, then we have also bring back cluster numbers
 of users which comes after deleting cluster
    # E.g. if we have deleted cluster 4 then their will be users whose clusters will be
5,6,7,..,N. So, we'll bring back those users cluster number to 4,5,6,...,N-1.
    for j in range(total_clusters):
        if len(usersInClusters[i]) < smallest_cluster_size:
            uncategorizedUsers.extend(usersInClusters[i])
            usersInClusters.pop(i)
            each_cluster_movies.pop(i)
            each_cluster_movies_list.pop(i)
            users_cluster.loc[users_cluster['Cluster'] > i, 'Cluster'] -= 1
            i -= 1
        i += 1
    for user in uncategorizedUsers:
        elemProbability = list()
        user_movies = getMoviesOfUser(user, users_data)
        if len(user_movies) == 0:
            print(user)
        user_missed_movies = list()
        for movies_list in each_cluster_movies_list:
            count = 0
            missed_movies = list()
            for movie in user_movies:
                if movie in movies_list:
                    count += 1
                else:
                    missed_movies.append(movie)
            elemProbability.append(count / len(user_movies))
            user_missed_movies.append(missed_movies)
        user_new_cluster = np.array(elemProbability).argmax()
        users_cluster.loc[users_cluster['userId'] == user, 'Cluster'] = user_new_cluste
r
        if len(user_missed_movies[user_new_cluster]) > 0:
            each_cluster_movies[user_new_cluster] = each_cluster_movies[user_new_cluste
r].append([{'movieId': new_movie, 'Count': 1} for new_movie in user_missed_movies[user_
```

```
new_cluster]], ignore_index = True)
    return each_cluster_movies, users_cluster
```

In [30]:

```
movies_df_fixed, clusters_fixed = fixClusters(cluster_movies, users_cluster, users_fav_
movies, smallest_cluster_size = 6)
```

In [31]:

```
j = 0
for i in range(15):
    len_users = users_cluster[users_cluster['Cluster'] == i].shape[0]
    if len_users < 6:
        print('Users in Cluster ' + str(i) + ' -> ', len_users)
        j += 1
print('Total Cluster which we want to remove -> ', j)
```

```
Users in Cluster 1 ->  5
Users in Cluster 2 ->  1
Users in Cluster 6 ->  1
Users in Cluster 7 ->  1
Users in Cluster 8 ->  1
Users in Cluster 10 ->  1
Users in Cluster 11 ->  1
Users in Cluster 12 ->  1
Users in Cluster 14 ->  5
Total Cluster which we want to remove ->  9
```

In [32]:

```
print('Length of total clusters before fixing is -> ', len(cluster_movies))
print('Max value in users_cluster dataframe column Cluster is -> ', users_cluster['Clus
ter'].max())
print('And dataframe is following')
users_cluster.T
```

```
Length of total clusters before fixing is ->  82
Max value in users_cluster dataframe column Cluster is ->  81
And dataframe is following
```

Out[32]:

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	3
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	3
Cluster	13	0	74	77	7	5	68	0	24	0	...	3	9	15	0	5	38	25	32	

2 rows × 315 columns

In [33]:

```
print('Length of total clusters after fixing is -> ', len(movies_df_fixed))
print('Max value in users_cluster dataframe column Cluster is -> ', clusters_fixed['Clu
ster'].max())
print('And fixed dataframe is following')
clusters_fixed.T
```

```
Length of total clusters after fixing is ->  11
Max value in users_cluster dataframe column Cluster is ->  10
And fixed dataframe is following
```

Out[33]:

	0	1	2	3	4	5	6	7	8	9	...	305	306	307	308	309	310	311	312	313
userId	1	3	4	5	6	7	8	9	11	12	...	339	340	341	342	343	344	345	346	347
Cluster	5	0	8	10	5	3	4	0	5	0	...	1	4	6	0	3	9	7	8	4

2 rows × 315 columns

In [34]:

```
print('Users cluster dataFrame for cluster 11 before fixing:')
users_cluster[users_cluster['Cluster'] == 11].T
```

```
Users cluster dataFrame for cluster 11 before fixing:
```

Out[34]:

	195
userId	219
Cluster	11

In [35]:

```
print('Users cluster dataFrame for cluster 4 after fixing which should be same as 11th
 cluster before fixing:')
clusters_fixed[clusters_fixed['Cluster'] == 4].T
```

```
Users cluster dataFrame for cluster 4 after fixing which should be same as
11th cluster before fixing:
```

Out[35]:

	6	11	23	43	53	54	60	75	99	107	...	124	125	132	134	139	162	216
userId	8	14	26	46	56	57	63	81	107	116	...	134	135	142	145	151	179	242
Cluster	4	4	4	4	4	4	4	4	4	4	...	4	4	4	4	4	4	4

2 rows × 22 columns

In [36]:

```
for i in range(len(movies_df_fixed)):
    len_users = clusters_fixed[clusters_fixed['Cluster'] == i].shape[0]
    print('Users in Cluster ' + str(i) + ' -> ', len_users)
```

```
Users in Cluster 0 ->  55
Users in Cluster 1 ->  8
Users in Cluster 2 ->  8
Users in Cluster 3 ->  35
Users in Cluster 4 ->  22
Users in Cluster 5 ->  82
Users in Cluster 6 ->  37
Users in Cluster 7 ->  10
Users in Cluster 8 ->  26
Users in Cluster 9 ->  21
Users in Cluster 10 ->  11
```

In [37]:

```
class saveLoadFiles:
    def save(self, filename, data):
        try:
            file = open('CAB420_Assessment_1C_Data/Data/' + filename + '.pkl', 'wb')
            pickle.dump(data, file)
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            file.close()
            return [False, err]
        else:
            file.close()
            return [True]
    def load(self, filename):
        try:
            file = open('CAB420_Assessment_1C_Data/Data/' + filename + '.pkl', 'rb')
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            file.close()
            return [False, err]
        else:
            data = pickle.load(file)
            file.close()
            return data
    def loadClusterMoviesDataset(self):
        return self.load('clusters_movies_dataset')
    def saveClusterMoviesDataset(self, data):
        return self.save('clusters_movies_dataset', data)
    def loadUsersClusters(self):
        return self.load('users_clusters')
    def saveUsersClusters(self, data):
        return self.save('users_clusters', data)
```

In [38]:

```
saveLoadFile = saveLoadFiles()
print(saveLoadFile.saveClusterMoviesDataset(movies_df_fixed))
print(saveLoadFile.saveUsersClusters(clusters_fixed))
```

```
[True]
[True]
```

In [39]:

```
load_movies_list, load_users_clusters = saveLoadFile.loadClusterMoviesDataset(), saveLo
adFile.loadUsersClusters()
print('Type of Loading list of Movies dataframes of 5 Clusters: ', type(load_movies_lis
t), ' and Length is: ', len(load_movies_list))
print('Type of Loading 100 Users clusters Data: ', type(load_users_clusters), ' and Sha
pe is: ', load_users_clusters.shape)
```

```
Type of Loading list of Movies dataframes of 5 Clusters:  <class 'list'>
and Length is:  11
Type of Loading 100 Users clusters Data:  <class 'pandas.core.frame.DataFr
ame'>  and Shape is:  (315, 2)
```

In [40]:

```
class userRequestedFor:
    def __init__(self, user_id, users_data):
        self.users_data = users_data.copy()
        self.user_id = user_id
        # Find User Cluster
        users_cluster = saveLoadFiles().loadUsersClusters()
        self.user_cluster = int(users_cluster[users_cluster['userId'] == self.user_id][
'Cluster'])
        # Load User Cluster Movies Dataframe
        self.movies_list = saveLoadFiles().loadClusterMoviesDataset()
        self.cluster_movies = self.movies_list[self.user_cluster] # dataframe
        self.cluster_movies_list = list(self.cluster_movies['movieId']) # list
    def updatedFavouriteMoviesList(self, new_movie_Id):
        if new_movie_Id in self.cluster_movies_list:
            self.cluster_movies.loc[self.cluster_movies['movieId'] == new_movie_Id, 'Co
unt'] += 1
        else:
            self.cluster_movies = self.cluster_movies.append([{'movieId':new_movie_Id,
'Count': 1}], ignore_index=True)
        self.cluster_movies.sort_values(by = ['Count'], ascending = False, inplace= Tru
e)
        self.movies_list[self.user_cluster] = self.cluster_movies
        saveLoadFiles().saveClusterMoviesDataset(self.movies_list)

    def recommendMostFavouriteMovies(self):
        try:
            user_movies = getMoviesOfUser(self.user_id, self.users_data)
            cluster_movies_list = self.cluster_movies_list.copy()
            for user_movie in user_movies:
                if user_movie in cluster_movies_list:
                    cluster_movies_list.remove(user_movie)
            return [True, cluster_movies_list]
        except KeyError:
            err = "User history does not exist"
            print(err)
            return [False, err]
        except:
            err = 'Error: {0}, {1}'.format(exc_info()[0], exc_info()[1])
            print(err)
            return [False, err]
```

In [41]:

```
movies_metadata = movies_metadata.loc[
    movies_metadata['movieId'].isin(list(map(str, np.unique(users_fav_movies['movieId'
])))))].reset_index(drop=True)
print('Let take a look at movie metadata for all those movies which we were had in our
 dataset')
movies_metadata
```

```
Let take a look at movie metadata for all those movies which we were had i
n our dataset
```

Out[41]:

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
182	229	Death and the Maiden (1994)	Drama|Thriller
183	230	Dolores Claiborne (1995)	Drama|Thriller
184	231	Dumb & Dumber (Dumb and Dumber) (1994)	Adventure|Comedy
185	232	Eat Drink Man Woman (Yin shi nan nu) (1994)	Comedy|Drama|Romance
186	233	Exotica (1994)	Drama

187 rows × 3 columns

In [42]:

```
print(list(movies_metadata.loc[movies_metadata['movieId'] == 39]['title']))
```

```
['Clueless (1995)']
```

In [43]:

```
user314Movies = getMoviesOfUser(314, users_fav_movies)
for movie in user314Movies:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['movieId'] == mo
vie]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(end = '\b\b]')
        print('')
```

```
Movie title:  ['Toy Story (1995)'] , Genres: [['Adventure|Animation|Childr
en|Comedy|Fantasy'] ]
Movie title:  ['Heat (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Sabrina (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['GoldenEye (1995)'] , Genres: [['Action|Adventure|Thrille
r'] ]
Movie title:  ['American President, The (1995)'] , Genres: [['Comedy|Drama
|Romance'] ]
Movie title:  ['Cutthroat Island (1995)'] , Genres: [['Action|Adventure|Ro
mance'] ]
Movie title:  ['Sense and Sensibility (1995)'] , Genres: [['Drama|Romanc
e'] ]
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Copycat (1995)'] , Genres: [['Crime|Drama|Horror|Mystery|T
hriller'] ]
Movie title:  ['Assassins (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Othello (1995)'] , Genres: [['Drama'] ]
Movie title:  ['Dangerous Minds (1995)'] , Genres: [['Drama'] ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres: [['M
ystery|Sci-Fi|Thriller'] ]
Movie title:  ['Clueless (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Richard III (1995)'] , Genres: [['Drama|War'] ]
Movie title:  ['Mortal Kombat (1995)'] , Genres: [['Action|Adventure|Fanta
sy'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thrille
r'] ]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Th
riller'] ]
Movie title:  ['Mighty Aphrodite (1995)'] , Genres: [['Comedy|Drama|Romanc
e'] ]
Movie title:  ['Indian in the Cupboard, The (1995)'] , Genres: [['Adventur
e|Children|Fantasy'] ]
Movie title:  ["Mr. Holland's Opus (1995)"] , Genres: [['Drama'] ]
Movie title:  ['Broken Arrow (1996)'] , Genres: [['Action|Adventure|Thrill
er'] ]
Movie title:  ['City Hall (1996)'] , Genres: [['Drama|Thriller'] ]
Movie title:  ['Muppet Treasure Island (1996)'] , Genres: [['Adventure|Chi
ldren|Comedy|Musical'] ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] ]
Movie title:  ['Flirting With Disaster (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Apollo 13 (1995)'] , Genres: [['Adventure|Drama|IMAX'] ]
Movie title:  ['Rob Roy (1995)'] , Genres: [['Action|Drama|Romance|War'] ]
Movie title:  ['Batman Forever (1995)'] , Genres: [['Action|Adventure|Come
dy|Crime'] ]
Movie title:  ['Crimson Tide (1995)'] , Genres: [['Drama|Thriller|War'] ]
Movie title:  ['Desperado (1995)'] , Genres: [['Action|Romance|Western'] ]
Movie title:  ['Die Hard: With a Vengeance (1995)'] , Genres: [['Action|Cr
ime|Thriller'] ]
Movie title:  ['Johnny Mnemonic (1995)'] , Genres: [['Action|Sci-Fi|Thrill
er'] ]
Movie title:  ['Judge Dredd (1995)'] , Genres: [['Action|Crime|Sci-Fi'] ]
Movie title:  ['Lord of Illusions (1995)'] , Genres: [['Horror'] ]
Movie title:  ['Net, The (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Species (1995)'] , Genres: [['Horror|Sci-Fi'] ]
Movie title:  ['To Wong Foo, Thanks for Everything! Julie Newmar (1995)']
, Genres: [['Comedy'] ]
Movie title:  ['Waterworld (1995)'] , Genres: [['Action|Adventure|Sci-Fi']
]
Movie title:  ['Boys on the Side (1995)'] , Genres: [['Comedy|Drama'] ]
Movie title:  ['Don Juan DeMarco (1995)'] , Genres: [['Comedy|Drama|Romanc
```

```
e'] ]
Movie title:  ['Drop Zone (1994)'] , Genres: [['Action|Thriller'] ]
```

In [44]:

```
user314Recommendations = userRequestedFor(314, users_fav_movies).recommendMostFavourite
Movies()[1]
for movie in user314Recommendations[:15]:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['id'] == str(mov
ie)]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(']', end = '')
        print()
```

```
Movie title:  ['Dumb & Dumber (Dumb and Dumber) (1994)'] , Genres: [['Adve
nture|Comedy'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy']
, ]
Movie title:  ['Ace Ventura: When Nature Calls (1995)'] , Genres: [['Comed
y'] , ]
Movie title:  ['Happy Gilmore (1996)'] , Genres: [['Comedy'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres: [['Action|Comedy|Ho
rror|Thriller'] , ]
Movie title:  ['Babe (1995)'] , Genres: [['Children|Drama'] , ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] , ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ['Smoke (1995)'] , Genres: [['Comedy|Drama'] , ]
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] ,
]
Movie title:  ['Disclosure (1994)'] , Genres: [['Drama|Thriller'] , ]
Movie title:  ['Father of the Bride Part II (1995)'] , Genres: [['Comedy']
, ]
Movie title:  ['First Knight (1995)'] , Genres: [['Action|Drama|Romance']
, ]
Movie title:  ['Billy Madison (1995)'] , Genres: [['Comedy'] , ]
Movie title:  ['Grumpier Old Men (1995)'] , Genres: [['Comedy|Romance'] ,
]
```

In [45]:

```
user4Movies = getMoviesOfUser(4, users_fav_movies)
for movie in user4Movies:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['movieId'] == mo
vie]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(end = '\b\b]')
        print('')
```

```
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres: [['M
ystery|Sci-Fi|Thriller'] ]
Movie title:  ['To Die For (1995)'] , Genres: [['Comedy|Drama|Thriller'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thrille
r'] ]
Movie title:  ['Mighty Aphrodite (1995)'] , Genres: [['Comedy|Drama|Romanc
e'] ]
Movie title:  ['Postman, The (Postino, Il) (1994)'] , Genres: [['Comedy|Dr
ama|Romance'] ]
Movie title:  ['Nobody Loves Me (Keiner liebt mich) (1994)'] , Genres:
[['Comedy|Drama'] ]
Movie title:  ['Flirting With Disaster (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['NeverEnding Story III, The (1994)'] , Genres: [['Adventure
|Children|Fantasy'] ]
Movie title:  ['Crumb (1994)'] , Genres: [['Documentary'] ]
Movie title:  ['Jeffrey (1995)'] , Genres: [['Comedy|Drama'] ]
Movie title:  ['Living in Oblivion (1995)'] , Genres: [['Comedy'] ]
Movie title:  ['Safe (1995)'] , Genres: [['Thriller'] ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] ]
Movie title:  ['Circle of Friends (1995)'] , Genres: [['Drama|Romance'] ]
Movie title:  ['Eat Drink Man Woman (Yin shi nan nu) (1994)'] , Genres:
[['Comedy|Drama|Romance'] ]
```

In [46]:

```
user4Recommendations = userRequestedFor(4, users_fav_movies).recommendMostFavouriteMovi
es()[1]
for movie in user4Recommendations[:15]:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['id'] == str(mov
ie)]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(']', end = '')
        print()
```

```
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] ,
]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Th
riller'] , ]
Movie title:  ['Leaving Las Vegas (1995)'] , Genres: [['Drama|Romance'] ,
]
Movie title:  ['City of Lost Children, The (Cité des enfants perdus, La)
(1995)'] , Genres: [['Adventure|Drama|Fantasy|Mystery|Sci-Fi'] , ]
Movie title:  ['Dead Man Walking (1995)'] , Genres: [['Crime|Drama'] , ]
Movie title:  ['Bottle Rocket (1996)'] , Genres: [['Adventure|Comedy|Crime
|Romance'] , ]
Movie title:  ['Toy Story (1995)'] , Genres: [['Adventure|Animation|Childr
en|Comedy|Fantasy'] , ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] , ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] , ]
Movie title:  ['Beauty of the Day (Belle de jour) (1967)'] , Genres: [['Dr
ama'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy']
, ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] , ]
Movie title:  ["Things to Do in Denver When You're Dead (1995)"] , Genres:
[['Crime|Drama|Romance'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres: [['Action|Comedy|Ho
rror|Thriller'] , ]
Movie title:  ['American President, The (1995)'] , Genres: [['Comedy|Drama
|Romance'] , ]
```

In [47]:

```
user42Movies = getMoviesOfUser(42, users_fav_movies)
for movie in user42Movies:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['movieId'] == mo
vie]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(end = '\b\b]')
        print('')
```

```
Movie title:  ['Grumpier Old Men (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Sabrina (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['GoldenEye (1995)'] , Genres: [['Action|Adventure|Thrille
r'] ]
Movie title:  ['American President, The (1995)'] , Genres: [['Comedy|Drama
|Romance'] ]
Movie title:  ['Casino (1995)'] , Genres: [['Crime|Drama'] ]
Movie title:  ['Ace Ventura: When Nature Calls (1995)'] , Genres: [['Comed
y'] ]
Movie title:  ['Get Shorty (1995)'] , Genres: [['Comedy|Crime|Thriller'] ]
Movie title:  ['Copycat (1995)'] , Genres: [['Crime|Drama|Horror|Mystery|T
hriller'] ]
Movie title:  ['Seven (a.k.a. Se7en) (1995)'] , Genres: [['Mystery|Thrille
r'] ]
Movie title:  ['Usual Suspects, The (1995)'] , Genres: [['Crime|Mystery|Th
riller'] ]
Movie title:  ['White Squall (1996)'] , Genres: [['Action|Adventure|Dram
a'] ]
Movie title:  ['Broken Arrow (1996)'] , Genres: [['Action|Adventure|Thrill
er'] ]
Movie title:  ['Happy Gilmore (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Braveheart (1995)'] , Genres: [['Action|Drama|War'] ]
Movie title:  ['Boomerang (1992)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Down Periscope (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Birdcage, The (1996)'] , Genres: [['Comedy'] ]
Movie title:  ['Brothers McMullen, The (1995)'] , Genres: [['Comedy'] ]
Movie title:  ['Apollo 13 (1995)'] , Genres: [['Adventure|Drama|IMAX'] ]
Movie title:  ['Batman Forever (1995)'] , Genres: [['Action|Adventure|Come
dy|Crime'] ]
Movie title:  ['Crimson Tide (1995)'] , Genres: [['Drama|Thriller|War'] ]
Movie title:  ['Desperado (1995)'] , Genres: [['Action|Romance|Western'] ]
Movie title:  ['Die Hard: With a Vengeance (1995)'] , Genres: [['Action|Cr
ime|Thriller'] ]
Movie title:  ['First Knight (1995)'] , Genres: [['Action|Drama|Romance']
]
Movie title:  ['Judge Dredd (1995)'] , Genres: [['Action|Crime|Sci-Fi'] ]
Movie title:  ['Net, The (1995)'] , Genres: [['Action|Crime|Thriller'] ]
Movie title:  ['Nine Months (1995)'] , Genres: [['Comedy|Romance'] ]
Movie title:  ['Showgirls (1995)'] , Genres: [['Drama'] ]
Movie title:  ['Something to Talk About (1995)'] , Genres: [['Comedy|Drama
|Romance'] ]
Movie title:  ['Clerks (1994)'] , Genres: [['Comedy'] ]
Movie title:  ['Disclosure (1994)'] , Genres: [['Drama|Thriller'] ]
Movie title:  ['Drop Zone (1994)'] , Genres: [['Action|Thriller'] ]
Movie title:  ['Dumb & Dumber (Dumb and Dumber) (1994)'] , Genres: [['Adve
nture|Comedy'] ]
```

In [48]:

```
user42Recommendations = userRequestedFor(42, users_fav_movies).recommendMostFavouriteMo
vies()[1]
for movie in user42Recommendations[:15]:
    title = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['title'])
    if title != []:
        print('Movie title: ', title, ', Genres: [', end = '')
        genre = list(movies_metadata.loc[movies_metadata['movieId'] == movie]['genres'
])
        if genre != []:
            print(genre, ', ', end = '')
        #genres = ast.literal_eval(movies_metadata.loc[movies_metadata['id'] == str(mov
ie)]['genres'].values[0].split('[')[1].split(']')[0])
        #for genre in genres:
            #print(genre['name'], ', ', end = '')
        print(']', end = '')
        print()
```

```
Movie title:  ['Toy Story (1995)'] , Genres: [['Adventure|Animation|Childr
en|Comedy|Fantasy'] , ]
Movie title:  ['Twelve Monkeys (a.k.a. 12 Monkeys) (1995)'] , Genres: [['M
ystery|Sci-Fi|Thriller'] , ]
Movie title:  ['Heat (1995)'] , Genres: [['Action|Crime|Thriller'] , ]
Movie title:  ['Clueless (1995)'] , Genres: [['Comedy|Romance'] , ]
Movie title:  ['Jumanji (1995)'] , Genres: [['Adventure|Children|Fantasy']
, ]
Movie title:  ['Sense and Sensibility (1995)'] , Genres: [['Drama|Romanc
e'] , ]
Movie title:  ['From Dusk Till Dawn (1996)'] , Genres: [['Action|Comedy|Ho
rror|Thriller'] , ]
Movie title:  ['Babe (1995)'] , Genres: [['Children|Drama'] , ]
Movie title:  ['Before Sunrise (1995)'] , Genres: [['Drama|Romance'] , ]
Movie title:  ["Mr. Holland's Opus (1995)"] , Genres: [['Drama'] , ]
Movie title:  ['Smoke (1995)'] , Genres: [['Comedy|Drama'] , ]
Movie title:  ['Mortal Kombat (1995)'] , Genres: [['Action|Adventure|Fanta
sy'] , ]
Movie title:  ['Taxi Driver (1976)'] , Genres: [['Crime|Drama|Thriller'] ,
]
Movie title:  ['Father of the Bride Part II (1995)'] , Genres: [['Comedy']
, ]
Movie title:  ['Billy Madison (1995)'] , Genres: [['Comedy'] , ]
```

In [ ]:

```
```