Weekend movie trip

September 28, 2020

1 Weekend movie trip

1.0.1 1. INDUSTRY

Movies industry is assigned.

1.0.2 2. DATA SETS

2.1. SOURCE: The dataset is from Grouplens in this link.

DESCRIPTION: The four datasets containing related to movies, user rating, tags and imdb ratings will be combined. The following attributes from the datasets will be used for analysis.

Attribute	Datatype
movieId	int64
userId	int64
rating	float64
tag	object

1.0.3 3. IDEAS

3.1. To suggest movies for users using tags and ratings with k-means clustering.

3.2. To suggest movies for users using tags and ratings with DBSCAN clustering.

1.0.4 4. LOADING THE DATASETS

Load the libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Import the csv files of dataset

[2]:
```
movies_data=pd.read_csv("movies.csv")
movies_data.head()
```

[2]:
```
   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)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
```

[3]:
```python
links_data=pd.read_csv("links.csv")
links_data.head()
```

[3]:
```
   movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0
3        4  114885  31357.0
4        5  113041  11862.0
```

[4]:
```python
rating_data=pd.read_csv("ratings.csv")
rating_data.head()
```

[4]:
```
   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
```

[5]:
```python
tags_data=pd.read_csv("tags.csv")
tags_data.head()
```

[5]:
```
   userId  movieId              tag   timestamp
0       2    60756            funny  1445714994
1       2    60756  Highly quotable  1445714996
2       2    60756     will ferrell  1445714992
3       2    89774     Boxing story  1445715207
4       2    89774              MMA  1445715200
```

1.0.5 5. DATA PREPARATION

5.1 DATA CLEANING

Shape of all the data

[6]:
```
print("Movies: "+str(movies_data.shape))
print("Links: "+str(links_data.shape))
print("Ratings: "+str(rating_data.shape))
print("Tags: "+str(tags_data.shape))
```

```
Movies: (9742, 3)
Links: (9742, 3)
Ratings: (100836, 4)
Tags: (3683, 4)
```

Drop the NaN rows

[7]:
```
movies_data=movies_data.dropna()
movies_data.head()
```

[7]:
```
   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)

                                        genres
0  Adventure|Animation|Children|Comedy|Fantasy
1                   Adventure|Children|Fantasy
2                               Comedy|Romance
3                         Comedy|Drama|Romance
4                                       Comedy
```

[8]:
```
links_data=links_data.dropna()
links_data.head()
```

[8]:
```
   movieId  imdbId   tmdbId
0        1  114709    862.0
1        2  113497   8844.0
2        3  113228  15602.0
3        4  114885  31357.0
4        5  113041  11862.0
```

[9]:
```
rating_data=rating_data.dropna()
rating_data.head()
```

[9]:
```
   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

[10]:
```
tags_data=tags_data.dropna()
tags_data.head()
```

[10]:

	userId	movieId	tag	timestamp
0	2	60756	funny	1445714994
1	2	60756	Highly quotable	1445714996
2	2	60756	will ferrell	1445714992
3	2	89774	Boxing story	1445715207
4	2	89774	MMA	1445715200

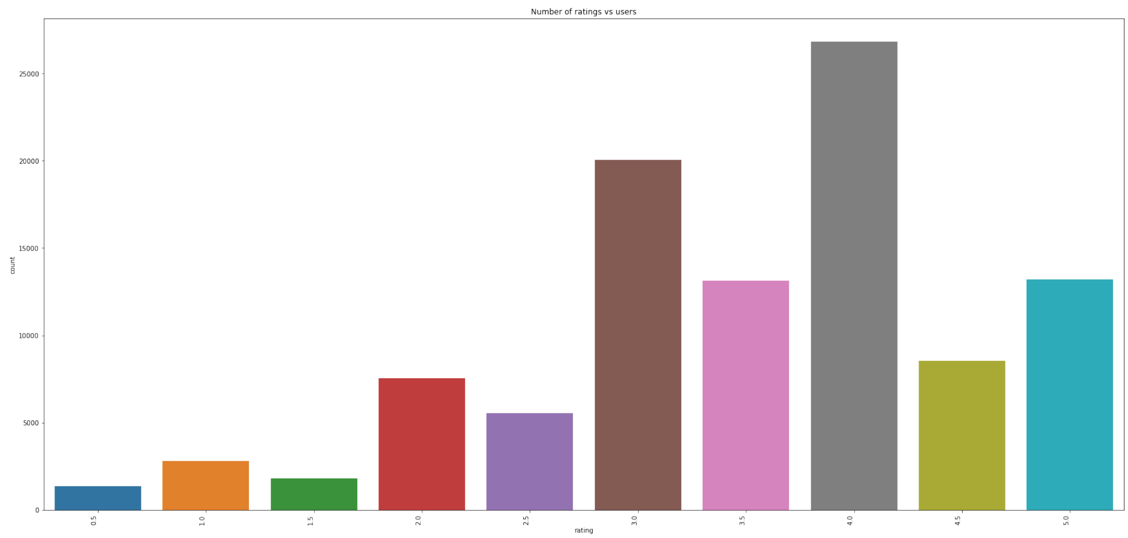
Shape of all the data

[11]:
```
print("Movies: "+str(movies_data.shape))
print("Links: "+str(links_data.shape))
print("Ratings: "+str(rating_data.shape))
print("Tags: "+str(tags_data.shape))
```

```
Movies: (9742, 3)
Links: (9734, 3)
Ratings: (100836, 4)
Tags: (3683, 4)
```

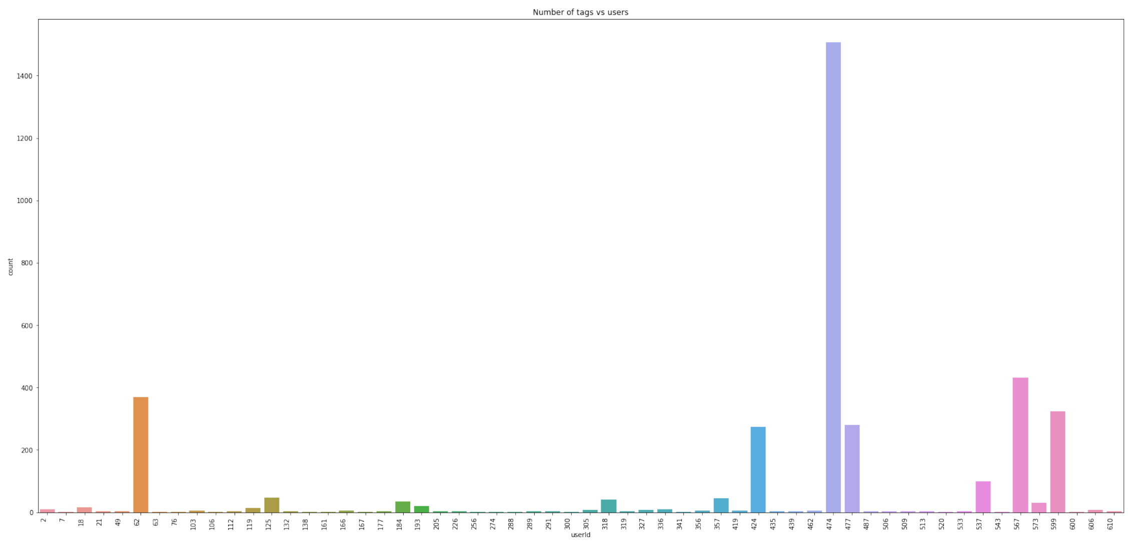
Rating distribution

[12]:
```
plt.figure(figsize=(30,14))
g = sns.countplot(x=rating_data['rating'], data=rating_data)
g.set_xticklabels(g.get_xticklabels(), rotation=90, ha="right");
g.set_title('Number of ratings vs users');
```



Tags by each user

[13]:
```
plt.figure(figsize=(30,14))
g = sns.countplot(x=tags_data['userId'], data=tags_data)
g.set_xticklabels(g.get_xticklabels(), rotation=90, ha="right");
g.set_title('Number of tags vs users');
```



5.2 FORMATTING

Combining the tag word from tag table for each user and movie

[26]:
```
# new column for storing tag
rating_data['tag'] = ''
# assigning tags for each row with respect to the player
for i in range(len(rating_data)):
        movie_id=rating_data['movieId'].values[i]
        user_id=rating_data['userId'].values[i]
        for j in range(len(tags_data)):
            if ((tags_data['movieId'].values[j] == movie_id) &␣
 ↪(tags_data['userId'].values[j]==user_id)):
                tag= tags_data['tag'].values[j]
                rating_data['tag'].values[i] = tag
rating_data.head()
```

[26]:

	userId	movieId	rating	timestamp	tag
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
```

Dropping the rows with no tags

[27]:
```
rating_data['tag'].replace('', np.nan, inplace=True)
rating_data=rating_data.dropna()
rating_data.head()
```

[27]:

	userId	movieId	rating	timestamp	tag
241	2	60756	5.0	1445714980	will ferrell
250	2	89774	5.0	1445715189	Tom Hardy
254	2	106782	5.0	1445714966	Martin Scorsese
1019	7	48516	1.0	1169687318	way too long
1808	18	431	4.0	1462138790	mafia

Reset the index and drop the old index column

[28]:
```
rating_data=rating_data.reset_index()
del rating_data['timestamp']
rating_data.head()
```

[28]:

	index	userId	movieId	rating	tag
0	241	2	60756	5.0	will ferrell
1	250	2	89774	5.0	Tom Hardy
2	254	2	106782	5.0	Martin Scorsese
3	1019	7	48516	1.0	way too long
4	1808	18	431	4.0	mafia

Printing the dimension of the dataset

[29]:
```
del rating_data['index']
rating_data.shape
```

[29]: (1635, 4)

Attributes and datatypes of the dataset

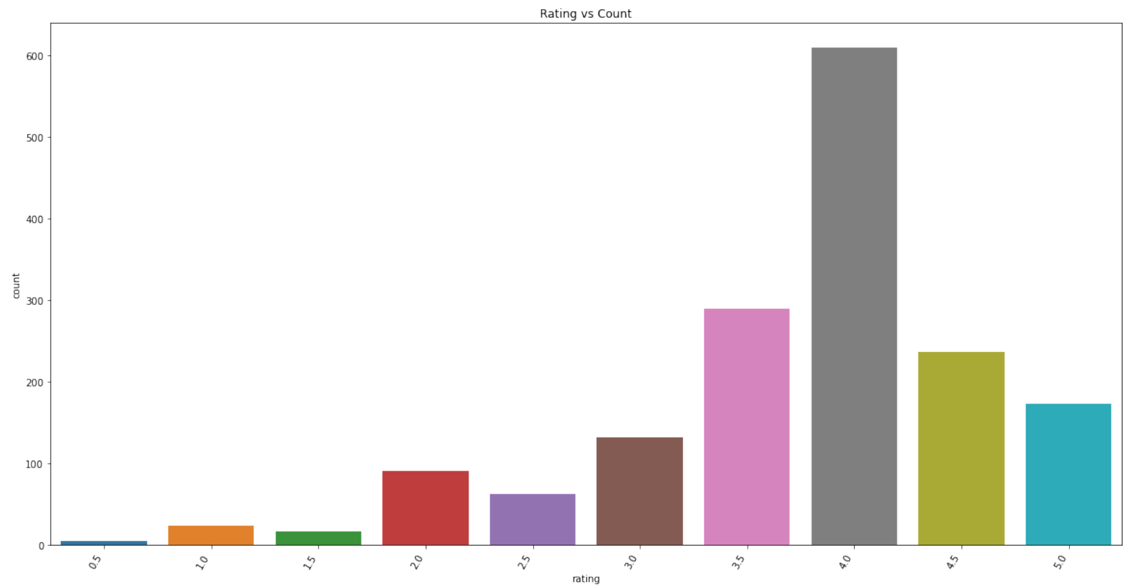
[30]:
```
for column in rating_data.columns:
    print(column, " is ", rating_data[column].dtype.name)
```

```
userId  is  int64
movieId  is  int64
rating  is  float64
tag  is  object
```

5.3 VISUALIZATION

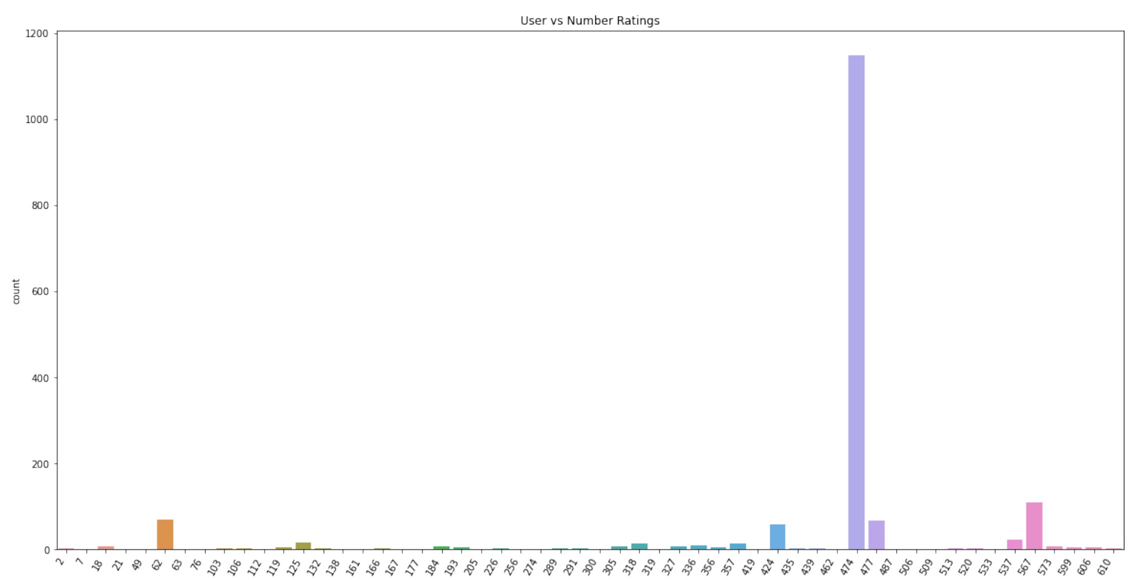
Barplot for ratings

```
[31]: plt.figure(figsize=(20,10))
      g = sns.countplot(x=rating_data['rating'], data=rating_data)
      g.set_xticklabels(g.get_xticklabels(), rotation=60, ha="right");
      g.set_title('Rating vs Count');
```



Barplot for user vs number of ratinngs

```
[32]: plt.figure(figsize=(20,10))
      g = sns.countplot(x=rating_data['userId'], data=rating_data)
      g.set_xticklabels(g.get_xticklabels(), rotation=60, ha="right");
      g.set_title('User vs Number Ratings');
```



Deleting PlayerLinernumber to release memory

[33]:
```
del tags_data
```

5.4 FEATURE ENGINEERING

Rearranging columns

[34]:
```
column_names = ["movieId", "userId", "tag","rating"]

rating_data = rating_data.reindex(columns=column_names)

rating_data.head()
```

[34]:

	movieId	userId	tag	rating
0	60756	2	will ferrell	5.0
1	89774	2	Tom Hardy	5.0
2	106782	2	Martin Scorsese	5.0
3	48516	7	way too long	1.0
4	431	18	mafia	4.0

Encoding the tag column using label encoder

[35]:
```
from sklearn.preprocessing import LabelEncoder

lb_make = LabelEncoder()
rating_data['tag_encode'] = lb_make.fit_transform(rating_data['tag'])
rating_data.head()
```

[35]:

	movieId	userId	tag	rating	tag_encode
0	60756	2	will ferrell	5.0	774
1	89774	2	Tom Hardy	5.0	268
2	106782	2	Martin Scorsese	5.0	141
3	48516	7	way too long	1.0	762
4	431	18	mafia	4.0	498

Rearranging the columns

[36]:
```
del rating_data['tag']

column_names = ["movieId", "userId","rating", "tag_encode"]

rating_data = rating_data.reindex(columns=column_names)

rating_data.head()
```

[36]:

	movieId	userId	rating	tag_encode
0	60756	2	5.0	774
1	89774	2	5.0	268
2	106782	2	5.0	141
3	48516	7	1.0	762
4	431	18	4.0	498

Normalizing the rating for each user from 0 to 5

[37]:

```
user_data=rating_data.userId.unique()
list1 = []
list2 =[]
for i in user_data:
    user_bar=rating_data.groupby(['userId']).get_group(i)
    min_val= user_bar.rating.min()
    max_val= user_bar.rating.max()
    OldRange = (max_val - min_val)
    NewRange = (5 - 0)
    if min_val != max_val:
        for j in range(len(rating_data)):
            if (rating_data['userId'].values[j] == user_bar['userId'].
 ↪unique()[0]):
                rating_data['rating'].values[j]= (((rating_data['rating'].
 ↪values[j] - min_val) * NewRange) / OldRange) + 0
rating_data
```

[37]:

	movieId	userId	rating	tag_encode
0	60756	2	5.000000	774
1	89774	2	5.000000	268
2	106782	2	5.000000	141
3	48516	7	1.000000	762
4	431	18	2.500000	498
…	…	…	…	…
1630	5694	606	0.000000	6
1631	6107	606	3.333333	295
1632	7382	606	5.000000	423
1633	3265	610	5.000000	453
1634	168248	610	5.000000	95

[1635 rows x 4 columns]

Drop data values less than 3.5 rating

[67]:

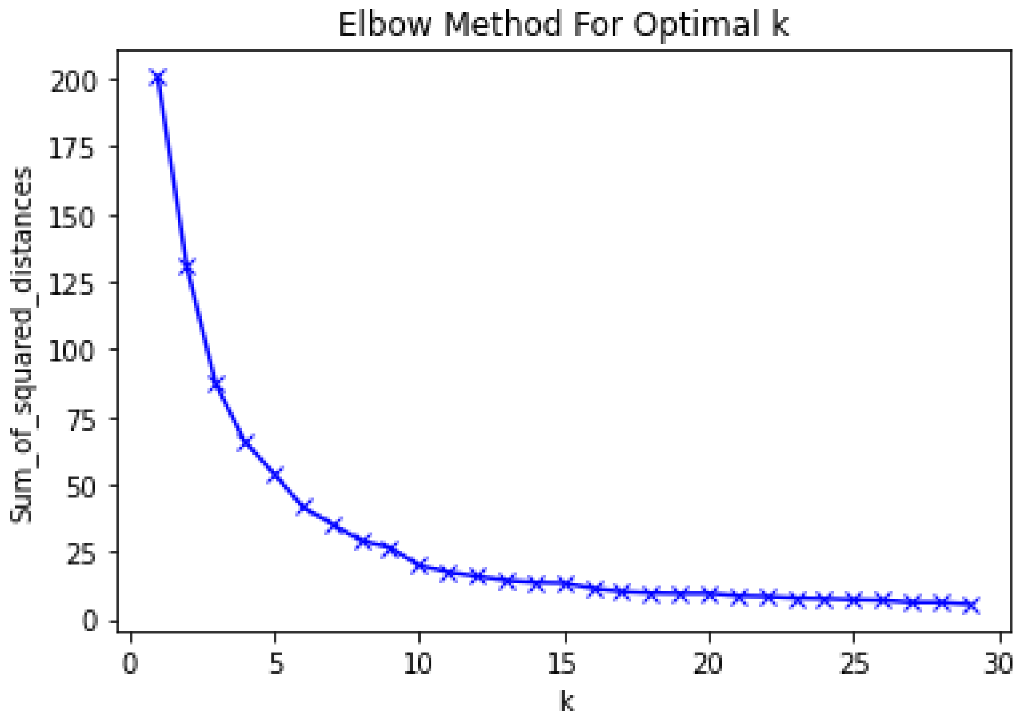
```
ratings_data = rating_data[rating_data['rating'] >= 3.5]
ratings_data = ratings_data.reset_index()
del ratings_data['index']
```

5.5 CLUSTERING

5.5.1 To cluster the movies based on tags and rating.

Find the best k from Elbow method

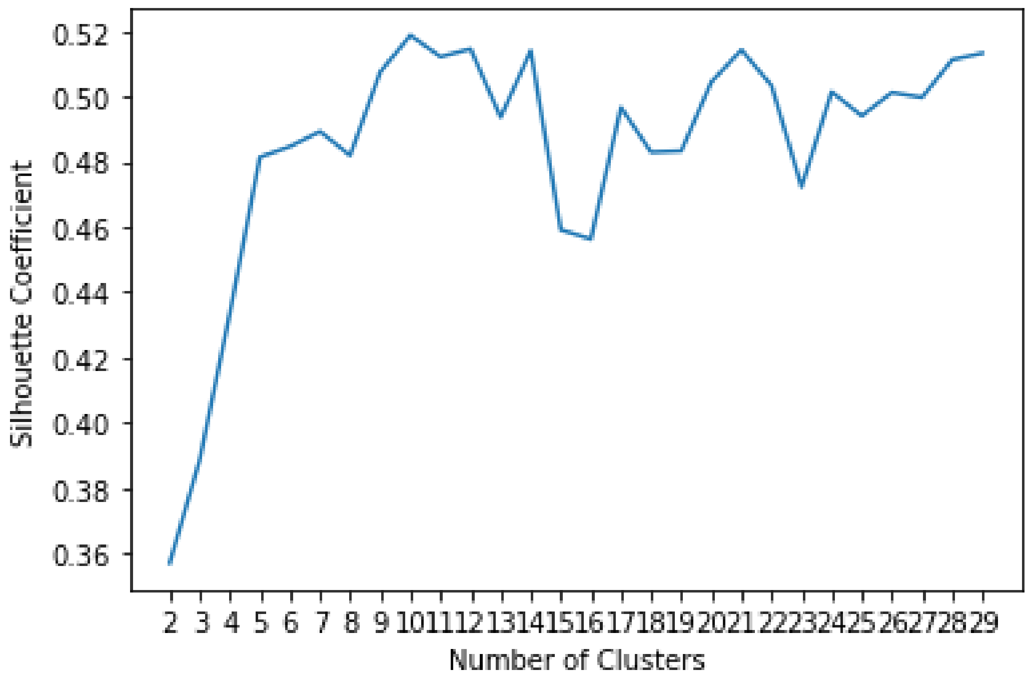
[74]:
```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
X = ratings_data.values[:,1:4]
X = np.nan_to_num(X)
mms = MinMaxScaler()
mms.fit(X)
data_transformed = mms.transform(X)
Sum_of_squared_distances = []
K = range(1,30)
kmeans_kwargs = {
   "init": "random",
   "n_init": 10,
   "max_iter": 300,
   "random_state": 42,
   }
for k in K:
    km = KMeans(n_clusters=k, **kmeans_kwargs)
    km = km.fit(data_transformed)
    Sum_of_squared_distances.append(km.inertia_)
plt.plot(K, Sum_of_squared_distances, 'bx-')
plt.xlabel('k')
plt.ylabel('Sum_of_squared_distances')
plt.title('Elbow Method For Optimal k')
plt.show()
```



Find the best k from Silhoutte

[75]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_score
kmeans_kwargs = {
   "init": "random",
   "n_init": 10,
   "max_iter": 300,
   "random_state": 42,
   }
X = ratings_data.values[:,1:4]
X = np.nan_to_num(X)
scaler = StandardScaler()
scaled_features = scaler.fit_transform(X)
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
# Notice you start at 2 clusters for silhouette coefficient
for k in range(2, 30):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(scaled_features)
    score = silhouette_score(scaled_features, kmeans.labels_)
    silhouette_coefficients.append(score)
plt.plot(range(2, 30), silhouette_coefficients)
```

```
plt.xticks(range(2, 30))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.show()
```



Preprocessing

[76]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets.samples_generator import make_blobs
%matplotlib inline
from sklearn.preprocessing import StandardScaler
Clus_dataSet = StandardScaler().fit_transform(X)
Clus_dataSet
```

[76]:
```
array([[-3.8961775 ,  1.8048161 ,  1.55903736],
       [-3.8961775 ,  1.8048161 , -0.61332554],
       [-3.8961775 ,  1.8048161 , -1.15856287],
       …,
       [ 1.44401517,  1.8048161 ,  0.05212159],
       [ 1.47938068,  1.8048161 ,  0.18091781],
       [ 1.47938068,  1.8048161 , -1.35605041]])
```

Performing k-means

[77]:
```python
clusterNum =10
k_means = KMeans(init = "k-means++", n_clusters = clusterNum, n_init = 50)
k_means.fit(X)
labels = k_means.labels_
print(labels)
```

```
[2 5 5 5 2 2 2 2 5 2 5 2 2 5 5 5 5 5 2 2 2 5 5 2 2 5 5 5 5 5 5 5 2 2 2 2 2
 5 5 5 2 2 5 2 2 5 2 2 2 2 2 5 5 5 2 2 6 2 2 2 5 5 5 2 5 2 5 2 6 5 5 2 2 2
 2 5 2 2 6 6 6 4 6 6 3 7 4 6 6 6 6 1 7 6 4 6 6 4 4 3 3 6 6 4 3 1 3 3 3 3 1
 3 1 3 3 0 3 3 1 3 0 3 3 4 3 3 1 3 4 1 4 3 1 1 3 3 1 9 3 3 3 9 4 3 1 9 9 1
 7 7 7 3 9 1 7 9 0 9 4 3 1 7 7 0 4 7 7 3 7 7 0 1 7 0 3 7 0 7 4 3 7 1 7 4 7
 7 1 7 7 4 7 0 7 4 4 7 9 3 7 7 9 7 7 4 7 7 7 7 0 4 1 4 0 0 3 1 7 4 0 3 3 7
 9 9 4 0 1 3 7 4 1 4 1 1 7 9 1 1 1 9 4 0 3 0 7 7 7 1 7 1 1 7 0 3 4 4 1 4 7
 7 9 7 7 4 4 9 3 9 0 4 1 9 3 3 4 9 1 1 4 0 1 1 1 3 1 1 9 7 1 7 1 1 0 4 7 1
 1 9 1 9 4 3 4 3 1 0 0 7 3 7 7 7 1 4 9 0 4 9 9 7 0 7 7 7 3 0 1 3 7 3 3 7 1
 7 4 0 1 4 9 4 4 3 3 4 4 1 3 7 7 0 7 4 1 7 3 4 9 4 7 7 3 7 0 4 0 7 0 9 4 7
 4 7 7 9 9 3 4 3 7 7 4 9 0 4 1 9 4 7 7 9 1 0 0 3 7 7 7 0 3 1 1 0 0 4 9 0 0
 3 9 9 3 0 7 0 0 1 7 4 0 7 9 3 7 1 1 0 1 7 0 9 7 7 3 3 0 1 4 4 0 3 3 0 3 3
 3 7 4 1 3 9 0 3 0 9 1 9 7 3 4 0 7 7 9 7 7 0 4 0 9 9 0 0 3 0 0 3 0 4 3 4 9
 4 0 9 1 0 9 4 9 1 0 7 0 1 4 7 7 3 0 1 7 7 3 4 4 9 3 9 9 3 9 3 9 7 1 9 0 0
 4 1 1 7 0 3 4 0 9 9 7 7 3 9 1 0 7 1 3 9 3 4 4 0 4 3 1 0 7 0 0 4 0 3 0 4 3
 4 0 4 1 9 1 4 1 7 7 9 9 3 7 3 9 1 4 4 0 1 7 4 0 9 4 0 0 9 1 0 9 3 4 3 9 0
 3 3 7 4 3 7 4 0 9 0 9 3 9 3 4 4 7 9 7 9 0 7 7 4 9 9 3 1 7 4 4 0 7 3 1 0 4
 3 4 9 4 7 0 4 4 7 3 9 7 7 9 4 4 3 9 4 0 7 4 4 9 4 4 0 7 7 3 9 7 1 3 9 9 1
 9 1 7 9 7 0 9 7 4 9 0 1 3 4 3 4 9 0 4 7 7 7 3 0 7 0 9 0 4 0 3 9 7 7 4 7 3
 4 3 9 1 3 3 7 0 4 9 7 7 9 7 9 0 0 7 7 0 3 9 4 7 0 7 0 7 4 4 9 9 7 0 4 3 4
 0 4 9 7 0 7 4 9 7 7 4 0 4 0 1 4 3 7 3 9 7 4 7 3 1 4 9 9 9 9 9 3 7 4 0 7 0
 9 9 9 4 0 4 4 1 4 7 7 3 7 0 0 1 9 7 1 3 0 4 3 9 3 9 4 4 3 1 7 0 0 0 9 9 7
 0 7 4 7 9 4 0 9 7 3 4 0 9 0 4 4 1 0 9 7 0 9 0 9 3 4 9 7 7 9 4 9 7 7 4 7 4
 7 1 4 7 7 7 7 3 1 1 1 3 9 0 9 4 3 3 3 9 3 7 0 4 3 4 3 0 9 3 3 9 9 3 4 0 3
 3 0 3 0 1 7 8 9 9 8 9 8 7 8 8 8 4 1 1 8 4 8 9 8 4 8 0 8 8 8 8 8 0 8 8 9 8
 8 8 4 0 8 8 8 0 8 8 8 0 9 9 8 8 8 8 8 8 9 9 0 8 9 4 4 8 8 8 8 8 8 0 0 7]
```

Clustered column added to the data

[78]:
```python
ratings_data["Clus_km"] = labels
ratings_data.head(5)
```

[78]:

	movieId	userId	rating	tag_encode	Clus_km
0	60756	2	5.00	774	2
1	89774	2	5.00	268	5
2	106782	2	5.00	141	5
3	1221	18	5.00	136	5
4	5995	18	3.75	725	2

View the centroids

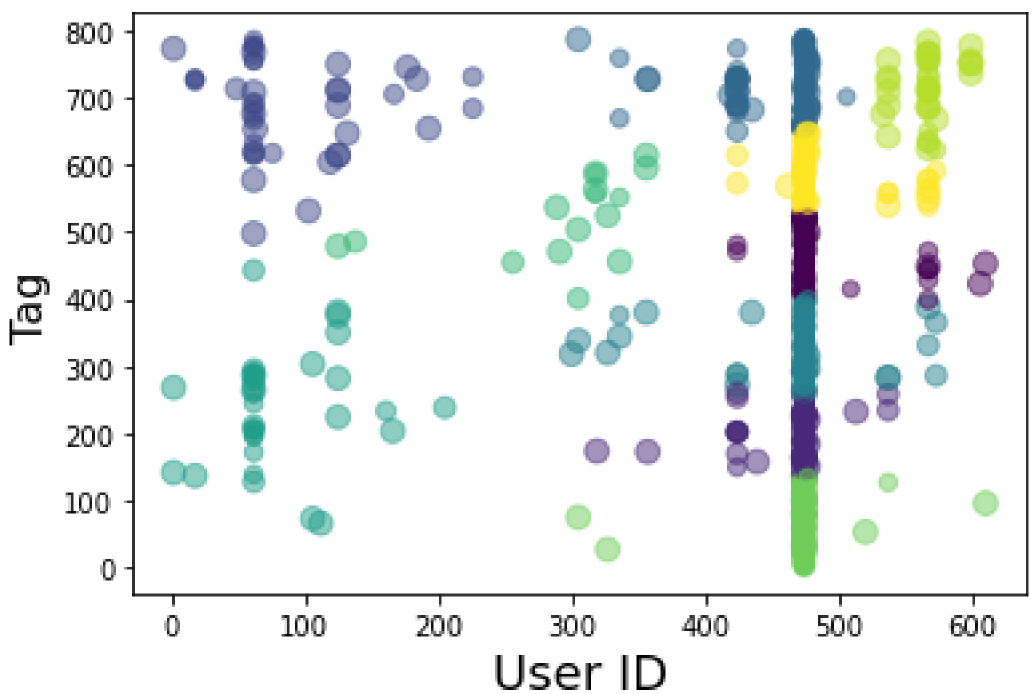
[79]:
```
ratings_data.groupby('Clus_km').mean()
```

[79]:

Clus_km	movieId	userId	rating	tag_encode
0	10464.185185	479.748148	4.048560	470.244444
1	6613.020202	468.616162	4.313272	197.282828
2	59928.512195	90.219512	4.417683	686.585366
3	12524.934307	461.941606	4.166869	711.708029
4	9858.293333	470.326667	4.140741	329.366667
5	55843.400000	80.142857	4.403571	240.257143
6	32844.187500	293.250000	4.664062	523.312500
7	7678.373563	473.609195	4.190613	73.051724
8	48138.384615	565.923077	4.629630	716.076923
9	14456.977778	479.570370	4.091049	587.725926

Clustered data

[80]:
```
area = np.pi * ( X[:, 1])**2
plt.scatter(X[:, 0], X[:, 2], s=area, c=labels.astype(np.float), alpha=0.5)
plt.xlabel('User ID', fontsize=18)
plt.ylabel('Tag', fontsize=16)
plt.show()
```



Plotting 3D

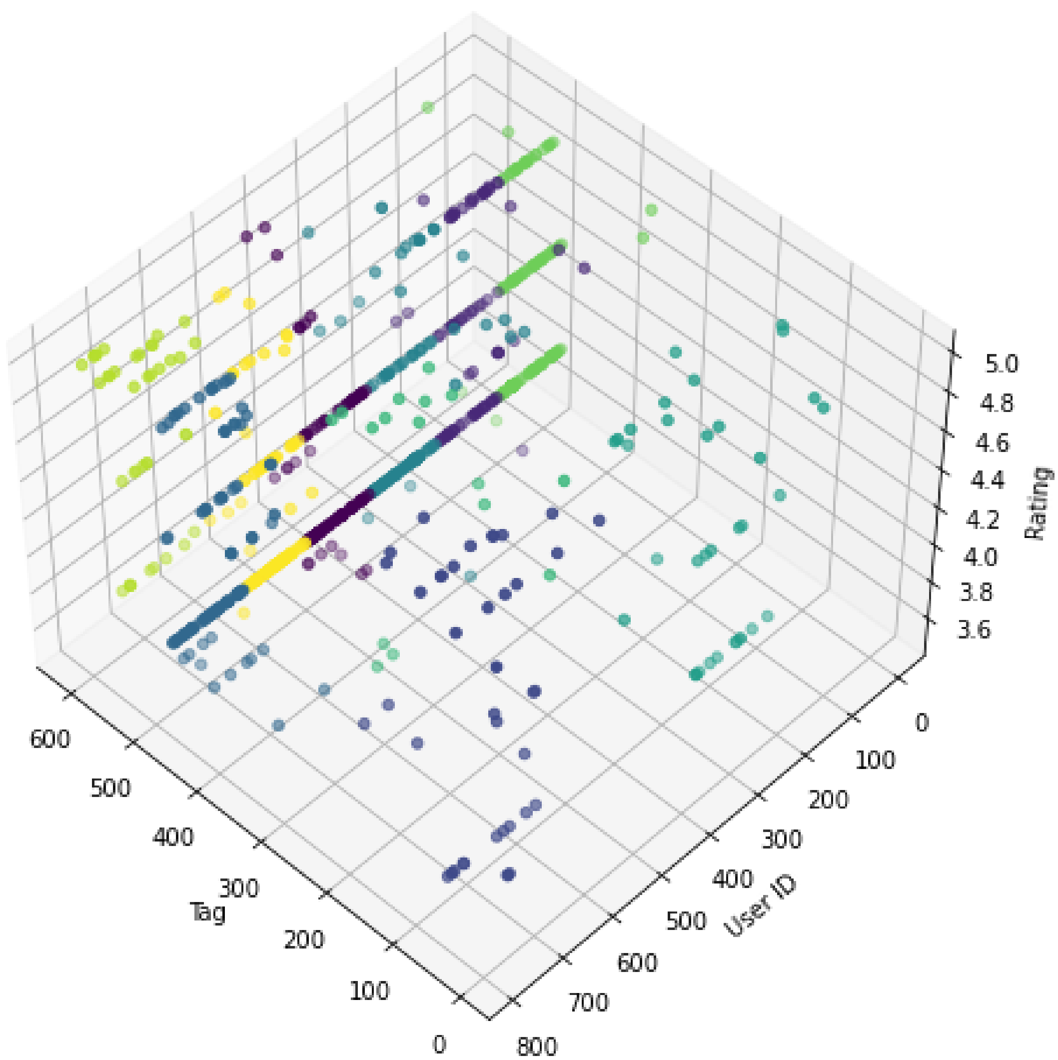
[81]:
```
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(1, figsize=(8, 6))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()

ax.set_xlabel('Tag')
ax.set_ylabel('User ID')
ax.set_zlabel('Rating')

ax.scatter(X[:, 0], X[:, 2], X[:, 1], c= labels.astype(np.float))
```

[81]:
```
<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x7f5374123250>
```



Finding movie for movieId using movie dataset

```
[82]: # new column for movie name
ratings_data['Movie'] = ''
# assigning title for each row with respect to the movieId
for i in range(len(ratings_data)):
        movie_id=ratings_data['movieId'].values[i]
        for j in range(len(movies_data)):
            if (movies_data['movieId'].values[j] == movie_id):
                movie= movies_data['title'].values[j]
                ratings_data['Movie'].values[i] = movie
ratings_data.head()
```

```
[82]:    movieId  userId  rating  tag_encode  Clus_km  \
0    60756       2    5.00         774        2
1    89774       2    5.00         268        5
2   106782       2    5.00         141        5
3     1221      18    5.00         136        5
4     5995      18    3.75         725        2

                            Movie
0            Step Brothers (2008)
1                  Warrior (2011)
2  Wolf of Wall Street, The (2013)
3   Godfather: Part II, The (1974)
4             Pianist, The (2002)
```

Printing the movies suggestion

```
[90]: user_data=ratings_data.userId.unique()
for i in user_data:
    user_bar=ratings_data.groupby(['userId']).get_group(i)
    print("UserID: "+str(user_bar.userId.unique())+"Cluster: "+str(user_bar.
 ↪Clus_km.unique()))
    for j in range(len(user_bar.Clus_km.unique())):
        clu=user_bar.Clus_km.unique()[j]
        movies_bar=ratings_data.groupby(['Clus_km']).get_group(clu)
        print(str(movies_bar.Movie.values[:3]))
```

```
UserID: [2]Cluster: [2 5]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [18]Cluster: [5 2]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [49]Cluster: [2]
```

```
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [62]Cluster: [5 2]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [63]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [76]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [103]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [106]Cluster: [5]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [112]Cluster: [5]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [119]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [125]Cluster: [2 6 5]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [132]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [138]Cluster: [6]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
UserID: [161]Cluster: [5]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [166]Cluster: [5]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [167]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [177]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [184]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [193]Cluster: [2]
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [205]Cluster: [5]
['Warrior (2011)' 'Wolf of Wall Street, The (2013)'
 'Godfather: Part II, The (1974)']
UserID: [226]Cluster: [2]
```

```
['Step Brothers (2008)' 'Pianist, The (2002)' 'Lucky Number Slevin (2006)']
UserID: [256]Cluster: [6]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
UserID: [289]Cluster: [6]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
UserID: [291]Cluster: [6]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
UserID: [300]Cluster: [4]
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
UserID: [305]Cluster: [6 3 7 4]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
UserID: [318]Cluster: [6]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
UserID: [319]Cluster: [1]
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
UserID: [327]Cluster: [7 6 4]
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
UserID: [336]Cluster: [6 4 3]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
UserID: [356]Cluster: [6 4]
['Going Places (Valseuses, Les) (1974)' 'Forbidden Kingdom, The (2008)'
 'The DUFF (2015)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
UserID: [357]Cluster: [3 1]
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
UserID: [419]Cluster: [3]
```

```
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
UserID: [424]Cluster: [3 1 0 4 9]
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
UserID: [435]Cluster: [4 3]
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
UserID: [439]Cluster: [1]
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
UserID: [462]Cluster: [9]
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
UserID: [474]Cluster: [9 1 7 3 0 4]
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
UserID: [477]Cluster: [3 1 9 0 4 7]
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
UserID: [506]Cluster: [3]
['28 Days Later (2002)' 'Wedding Crashers (2005)' 'Lord of War (2005)']
UserID: [509]Cluster: [0]
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
UserID: [513]Cluster: [1]
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
```

```
UserID: [520]Cluster: [7]
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
UserID: [533]Cluster: [8]
['Forrest Gump (1994)' 'Catch Me If You Can (2002)' 'Bank Job, The (2008)']
UserID: [537]Cluster: [9 8 7 4 1]
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
['Forrest Gump (1994)' 'Catch Me If You Can (2002)' 'Bank Job, The (2008)']
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
['Lion King, The (1994)' 'Jezebel (1938)' 'Kids (1995)']
UserID: [567]Cluster: [8 9 4 0]
['Forrest Gump (1994)' 'Catch Me If You Can (2002)' 'Bank Job, The (2008)']
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
UserID: [573]Cluster: [8 9 4]
['Forrest Gump (1994)' 'Catch Me If You Can (2002)' 'Bank Job, The (2008)']
['Inglourious Basterds (2009)' 'Babadook, The (2014)'
 'Who Killed Chea Vichea? (2010)']
['Lost in Translation (2003)' 'Tucker & Dale vs Evil (2010)'
 'Anchorman: The Legend of Ron Burgundy (2004)']
UserID: [599]Cluster: [8]
['Forrest Gump (1994)' 'Catch Me If You Can (2002)' 'Bank Job, The (2008)']
UserID: [606]Cluster: [0]
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
UserID: [610]Cluster: [0 7]
["One Flew Over the Cuckoo's Nest (1975)" 'Good Will Hunting (1997)'
 'Usual Suspects, The (1995)']
['Elite Squad (Tropa de Elite) (2007)'
 "There's Something About Mary (1998)" 'Sense and Sensibility (1995)']
```