# Importing All Required Libraries  
import pandas as pd  
print('Pandas version: ', pd.\_\_version\_\_)  
  
import numpy as np  
print('NumPy version: ', np.\_\_version\_\_)  
  
import matplotlib  
print('Matplotlib version: ', matplotlib.\_\_version\_\_)  
  
from matplotlib import pyplot as plt  
  
import sklearn  
print('Scikit-Learn version: ', sklearn.\_\_version\_\_)  
  
from sklearn.feature\_extraction.text import CountVectorizer  
  
from sklearn.cluster import KMeans  
  
  
import pickle  
print('Pickle version: ', pickle.format\_version)  
  
import sys  
print('Sys version: ', sys.version[0:5])  
  
from sys import exc\_info  
  
import ast  
import time  
  
# Data Processing.  
  
ratings = pd.read\_csv('./Prepairing Data/From Data/ratings.csv', usecols = ['userId', 'movieId','rating'])  
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')  
# Filtering data for only 4+ ratings  
ratings = ratings[ratings['rating'] >= 4.0]  
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')  
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')  
users\_list = np.unique(ratings['userId'])[:100]  
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']).shape[0])  
users\_fav\_movies = ratings.reset\_index(drop = True)  
users\_fav\_movies.to\_csv('./Prepairing Data/From Data/filtered\_ratings.csv')  
#Data Featuring  
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  
users = np.unique(users\_fav\_movies['userId'])  
print(users.shape)  
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 100 users  
A list of first 10 users favourite movies:   
 ['147', '64, 79', '1, 47', '1, 150', '150, 165', '34', '1, 16, 17, 29, 34, 47, 50, 82, 97, 123, 125, 150, 162, 175, 176, 194', '6', '32, 50, 111, 198', '81']  
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 defined   
 # 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()  
first\_6\_users\_SM = users\_fav\_movies[users\_fav\_movies['userId'].isin(users[:6])].sort\_values('userId')  
first\_6\_users\_SM.T  
   
# Clustering Model  
  
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()  
# Fitting data on Model  
kmeans = KMeans(n\_clusters=15, init = 'k-means++', max\_iter = 300, n\_init = 10, random\_state = 0)  
clusters = kmeans.fit\_predict(sparseMatrix)  
users\_cluster = pd.DataFrame(np.concatenate((users.reshape(-1,1), clusters.reshape(-1,1)), axis = 1), columns = ['userId', 'Cluster'])  
users\_cluster.T  
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]['movieId']))  
 users\_movies\_counts = list()  
 users\_movies\_counts.extend([[movie, users\_movies\_list.count(movie)] for movie in 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)  
for i in range(15):  
 len\_users = users\_cluster[users\_cluster['Cluster'] == i].shape[0]  
 print('Users in Cluster ' + str(i) + ' -> ', len\_users)  
Fixing Small Clusters  
  
def getMoviesOfUser(user\_id, users\_data):  
 return list(users\_data[users\_data['userId'] == user\_id]['movieId'])  
def fixClusters(clusters\_movies\_dataframes, users\_cluster\_dataframe, users\_data, smallest\_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 cluster 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 -> [[Cluster 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]['userId']))  
 uncategorizedUsers = list()  
 i = 0  
 # Now we will remove small clusters and put their users into another list named "uncategorizedUsers"  
 # 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\_cluster  
 if len(user\_missed\_movies[user\_new\_cluster]) > 0:  
 each\_cluster\_movies[user\_new\_cluster] = each\_cluster\_movies[user\_new\_cluster].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  
movies\_df\_fixed, clusters\_fixed = fixClusters(cluster\_movies, users\_cluster, users\_fav\_movies, smallest\_cluster\_size = 6)  
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)  
print('Length of total clusters before fixing is -> ', len(cluster\_movies))  
print('Max value in users\_cluster dataframe column Cluster is -> ', users\_cluster['Cluster'].max())  
print('And dataframe is following')  
users\_cluster.T  
print('Length of total clusters after fixing is -> ', len(movies\_df\_fixed))  
print('Max value in users\_cluster dataframe column Cluster is -> ', clusters\_fixed['Cluster'].max())  
print('And fixed dataframe is following')  
clusters\_fixed.T  
print('Users cluster dataFrame for cluster 11 before fixing:')  
users\_cluster[users\_cluster['Cluster'] == 11].T  
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  
print('Size of movies dataframe after fixing -> ', len(movies\_df\_fixed))  
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)  
for i in range(len(movies\_df\_fixed)):  
 print('Total movies in Cluster ' + str(i) + ' -> ', movies\_df\_fixed[i].shape[0])  
class saveLoadFiles:  
 def save(self, filename, data):  
 try:  
 file = open('datasets/' + 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('datasets/' + 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)  
saveLoadFile = saveLoadFiles()  
print(saveLoadFile.saveClusterMoviesDataset(movies\_df\_fixed))  
print(saveLoadFile.saveUsersClusters(clusters\_fixed))  
load\_movies\_list, load\_users\_clusters = saveLoadFile.loadClusterMoviesDataset(), saveLoadFile.loadUsersClusters()  
print('Type of Loading list of Movies dataframes of 5 Clusters: ', type(load\_movies\_list), ' and Length is: ', len(load\_movies\_list))  
print('Type of Loading 100 Users clusters Data: ', type(load\_users\_clusters), ' and Shape is: ', load\_users\_clusters.shape)  
#Recommendations For Users  
  
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, 'Count'] += 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= True)  
 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]  
movies\_metadata = pd.read\_csv(  
 './Prepairing Data/From Data/movies\_metadata.csv',   
 usecols = ['id', 'genres', 'original\_title'])  
  
movies\_metadata = movies\_metadata.loc[  
 movies\_metadata['id'].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  
user12Movies = getMoviesOfUser(12, users\_fav\_movies)  
for movie in user12Movies:  
 title = list(movies\_metadata.loc[movies\_metadata['id'] == str(movie)]['original\_title'])  
 if title != []:  
 print('Movie title: ', title, ', Genres: [', end = '')  
 genres = ast.literal\_eval(movies\_metadata.loc[movies\_metadata['id'] == str(movie)]['genres'].values[0].split('[')[1].split(']')[0])  
 for genre in genres:  
 print(genre['name'], ', ', end = '')  
 print(end = '\b\b]')  
 print('')  
user12Recommendations = userRequestedFor(12, users\_fav\_movies).recommendMostFavouriteMovies()[1]  
for movie in user12Recommendations[:15]:  
 title = list(movies\_metadata.loc[movies\_metadata['id'] == str(movie)]['original\_title'])  
 if title != []:  
 print('Movie title: ', title, ', Genres: [', end = '')  
 genres = ast.literal\_eval(movies\_metadata.loc[movies\_metadata['id'] == str(movie)]['genres'].values[0].split('[')[1].split(']')[0])  
 for genre in genres:  
 print(genre['name'], ', ', end = '')  
 print(']', end = '')  
 print()