**FOUNDATION OF DATA SCIENCE**

**ASSIGNMENT - 2**

**MUSIC RECOMMENDATION SYSTEM**

**CODE**

# -\*- coding: utf-8 -\*-

"""Music Recommendation System.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1vH4Hwhfy34eFBMM9jNEtAduec1XGoanF

"""

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

s = pd.Series([1.1, 2.3])

a = np.array(s)

print(a) # [1.1 2.3]

from sklearn.preprocessing import OrdinalEncoder

from sklearn.cluster import KMeans

from scipy import stats

from sklearn.preprocessing import StandardScaler

from sklearn.metrics.pairwise import cosine\_similarity

pd.options.mode.chained\_assignment = None

Spotify = pd.read\_csv('SpotifyFeatures.csv')

Spotify.head()

indexi = Spotify[['track\_name', 'artist\_name']]

attributes = Spotify.drop(['track\_id', 'time\_signature','track\_name', 'artist\_name', 'key'], axis = 1)

attributes.head()

ordinalencoder = OrdinalEncoder()

object\_cols = ['mode']

attributes[object\_cols] = ordinalencoder.fit\_transform(attributes[object\_cols])

attributes = pd.get\_dummies(attributes)

attributes.insert(loc=0, column='track\_name', value=indexi.track\_name)

attributes.insert(loc=1, column = 'artist\_name', value = indexi.artist\_name)

genres\_names = ['genre\_A Capella', 'genre\_Alternative', 'genre\_Anime', 'genre\_Blues',

"genre\_Children's Music", "genre\_Children’s Music", 'genre\_Classical',

'genre\_Comedy', 'genre\_Country', 'genre\_Dance', 'genre\_Electronic',

'genre\_Folk', 'genre\_Hip-Hop', 'genre\_Indie', 'genre\_Jazz',

'genre\_Movie', 'genre\_Opera', 'genre\_Pop', 'genre\_R&B', 'genre\_Rap',

'genre\_Reggae', 'genre\_Reggaeton', 'genre\_Rock', 'genre\_Ska',

'genre\_Soul', 'genre\_Soundtrack', 'genre\_World']

Spotify = Spotify.loc[:,~Spotify.columns.duplicated()]

Spotify.columns.duplicated().any()

attributes.columns.duplicated().any()

attributes = attributes.loc[:,~attributes.columns.duplicated()]

genre = attributes.groupby(['track\_name', 'artist\_name'])[genres\_names].sum()

column\_names = ['track\_name', 'artist\_name']

for i in genres\_names:

column\_names.append(i)

genre.reset\_index(inplace=True)

genre.columns = column\_names

attributes = attributes.drop(genres\_names, axis = 1)

atts\_cols = attributes.drop(['track\_name', 'artist\_name'], axis = 1).columns

scaler = StandardScaler()

attributes[atts\_cols] = scaler.fit\_transform(attributes[atts\_cols])

song = pd.merge(genre, attributes, how = 'inner', on = ['track\_name', "artist\_name"])

song = song.drop\_duplicates(['track\_name', 'artist\_name']).reset\_index(drop = True)

song.head()

sse={}

DF = pd.DataFrame(song.drop(['track\_name', 'artist\_name'], axis = 1))

np.isfinite(DF.all())

DF.replace([np.inf, -np.inf], np.nan, inplace=True)

DF.fillna(999, inplace=True)

for k in range(1, 30):

kmeans = KMeans(n\_clusters=k, max\_iter=1000).fit(DF)

DF["clusters"] = kmeans.labels\_

sse[k] = kmeans.inertia\_

plt.figure()

plt.plot(list(sse.keys()), list(sse.values()))

plt.title("Elbow method")

plt.xlabel("Number of cluster")

plt.show()

np.isfinite(song.all())

song.replace([np.inf, -np.inf], np.nan, inplace=True)

song.fillna(999, inplace=True)

DF = pd.DataFrame(song.drop(['track\_name', 'artist\_name'], axis = 1))

kmeans = KMeans(n\_clusters=17)

song['Cluster'] = kmeans.fit\_predict(DF)

def find\_song\_Database(name, artist, songs):

result = songs[(songs.artist\_name == str(artist)) & (songs.track\_name == str(name))]

if len(result) == 0:

return None

return result.drop(['track\_name', 'artist\_name', 'Cluster'], axis = 1)

def find\_similar(name, artist, songs, top\_n = 5):

Database = songs[songs.popularity > 0.5].reset\_index(drop = True)

indexi\_names = Database[['track\_name', 'artist\_name', 'Cluster']]

songs\_train = Database.drop(['track\_name', 'artist\_name', 'Cluster'], axis = 1)

song = find\_song\_Database(str(name), str(artist), Database)

if type(song) != type(None):

indexi\_song = song.index

cos\_dists = cosine\_similarity(songs\_train, songs\_train)

indexi\_names.loc[:,['result']] = cos\_dists[indexi\_song[0]]

indexi\_names = indexi\_names.sort\_values(by = ['result'], ascending = False)

return indexi\_names[1:top\_n].reset\_index(drop = True)

else:

print("Song not found")

return None

def playlist\_song(name, artist, songs, n\_songs = 10):

list\_songs = find\_similar(str(name), str(artist), song)

if type(list\_songs) != type(None):

print('Playlist based on "' + str(name) + '" by ' + str(artist))

print()

for i in np.arange(0,len(list\_songs)):

track\_name = list\_songs.track\_name[i]

artist\_name = list\_songs.artist\_name[i]

print(str(track\_name) + ' - ' + str(artist\_name))

return None

dists = find\_similar('Our Song', 'Taylor Swift', song)

playlist\_song('All Of The Lights', 'Kanye West', song, 10)