In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import spotipy
import playlist
import config
import time
import create_dataframes
from sklearn.metrics import pairwise
from sklearn.preprocessing import MinMaxScaler
from spotipy.oauth2 import SpotifyOAuth
```

In [2]:
```
import importlib
```

In [3]:
```
importlib.reload(playlist)
```

Out[3]:
```
<module 'playlist' from '/Users/marvazqu8/mv/fall23/dsc478/dsc478_final_projec
t/dsc478_final/playlist.py'>
```

In [4]:
```
# #create spotipy object to interact with spotify web API
# from spotipy.oauth2 import SpotifyClientCredentials
# sp = spotipy.Spotify(auth_manager=SpotifyClientCredentials(client_id=config.S
#                                                            client_secret=conf

# def get_user_playlist():
#     '''Ask the user for a link to a Spotify playlist'''

#     ask = True
#     playlist_len = 0

#     playlist_url = input('Enter a link to a Spotify playlist:')

#     while ask==True:
#         try:
#             playlist_len = int(input('How long would you like your final play
#             ask = False
#         except:
#             print('Please be sure to use an INTEGER when specifying how long

#     return (playlist_url, playlist_len)


# #playlist_link, final_playlist_len = get_user_playlist()
# playlist_link, final_playlist_len = ('https://open.spotify.com/playlist/3LNpQ
# playlist1 = playlist.Playlist(playlist_link, sp)
# artist_counts, artist_id_counts = playlist1.get_artist_counts()
# avg_audio_values = pd.DataFrame(playlist1.normalized_audio_features.mean(axis
# # playlist1's df_playlist is the FULL playlist with audio data2
```

In [5]:
```
#create spotipy object to interact with spotify web API
from spotipy.oauth2 import SpotifyClientCredentials
sp = spotipy.Spotify(
    auth_manager=SpotifyClientCredentials
        (client_id=config.SPOTIFY_CLIENT_ID,
```

```
        client_secret=config.SPOTIFY_SECRET
        ))
```

In [7]:
```
# playlist_link = create_dataframes.get_user_playlist()
playlist_link = 'https://open.spotify.com/playlist/37i9dQZEVXcNDMTzEMNNE1'
playlist1 = playlist.Playlist(playlist_link, sp)
artist_counts, artist_id_counts = playlist1.get_artist_counts()
avg_audio_values = pd.DataFrame(playlist1.normalized_audio_features.mean(axis=0
```

```
/var/folders/tc/pt7m46h93wxcsmrvn_xnjczm0000gn/T/ipykernel_32407/2365521695.p
y:5: FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise TypeEr
ror.  Select only valid columns before calling the reduction.
  avg_audio_values = pd.DataFrame(playlist1.normalized_audio_features.mean(axi
s=0))
```

In [8]:
```
artists_search = create_dataframes.artist_search_results(artist_counts.columns,
recs = playlist1.get_recommendations(artists_search, avg_audio_values, k=5)
#create_dataframes.create_new_playlist(recs, final_playlist_len)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File ~/Library/Python/3.8/lib/python/site-packages/pandas/core/indexes/base.p
y:3621, in Index.get_loc(self, key, method, tolerance)
   3620 try:
-> 3621     return self._engine.get_loc(casted_key)
   3622 except KeyError as err:

File pandas/_libs/index.pyx:136, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/index.pyx:163, in pandas._libs.index.IndexEngine.get_loc()

File pandas/_libs/hashtable_class_helper.pxi:5198, in pandas._libs.hashtable.P
yObjectHashTable.get_item()

File pandas/_libs/hashtable_class_helper.pxi:5206, in pandas._libs.hashtable.P
yObjectHashTable.get_item()

KeyError: 0

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Cell In [8], line 1
----> 1 artists_search = create_dataframes.artist_search_results(artist_counts
.columns, artist_id_counts)
      2 recs = playlist1.get_recommendations(artists_search, avg_audio_values,
k=5)

File ~/mv/fall23/dsc478/dsc478_final_project/dsc478_final/create_dataframes.p
y:52, in artist_search_results(artist_list, artist_id_list)
     49 for i in range(0, num_artists):
     51     artist = artist_list[i]
---> 52     artist_id = artist_id_list[i]
     54     song_names = []
     55     song_ids = []

File ~/Library/Python/3.8/lib/python/site-packages/pandas/core/frame.py:3505,
 in DataFrame.__getitem__(self, key)
   3503 if self.columns.nlevels > 1:
   3504     return self._getitem_multilevel(key)
-> 3505 indexer = self.columns.get_loc(key)
   3506 if is_integer(indexer):
   3507     indexer = [indexer]

File ~/Library/Python/3.8/lib/python/site-packages/pandas/core/indexes/base.p
y:3623, in Index.get_loc(self, key, method, tolerance)
   3621     return self._engine.get_loc(casted_key)
   3622 except KeyError as err:
-> 3623     raise KeyError(key) from err
   3624 except TypeError:
   3625     # If we have a listlike key, _check_indexing_error will raise
   3626     #  InvalidIndexError. Otherwise we fall through and re-raise
   3627     #  the TypeError.
   3628     self._check_indexing_error(key)

KeyError: 0
```

In [ ]:

In [ ]:
```
playlist1.audio_features
```

In [ ]:
```
playlist1.numeric_features
```

In [ ]:
```
playlist1.numeric_features.iloc[:,1:]
```

In [ ]:
```
tmp =  playlist1.normalize_audio_features(MinMaxScaler, playlist1.numeric_featu
tmp
```

In [ ]:
```
playlist1.normalized_audio_features = tmp
playlist1.normalized_audio_features
```

In [ ]:
```
artist_counts, artist_id_counts = playlist1.get_artist_counts()
```

In [ ]:
```
avg_audio_values
```

In [ ]:
```
# def artist_search_results(artist_list):
#     artist_search_res = {}

#     for artist in artist_list:
#         result = sp.search(artist, type=['track'], limit = 50) #limit is 50 b
#         song_names = []
#         song_ids = []

#         for song in result['tracks']['items']:
#             song_names.append(song['name'])
#             song_ids.append(song['id'])

#         d = {'track':song_names,
#             'track_id':song_ids}

#         songs_search_res = pd.DataFrame(d, index=None)

#         artist_search_res[artist] = songs_search_res

#     return artist_search_res
```

In [ ]:
```
artists_search = create_dataframes.artist_search_results(artist_counts.columns)
```

In [ ]:
```
# def knn_search(instance, data, K, measure):
#     """ find K nearest neighbors of an instance x among the instances in D "'
#     n_songs = data.shape[0]
#     sims = []
#     for j in range(0,n_songs) :
#         try:
#             sims.append(measure(np.array(instance).reshape(1, -1), np.array(d

#         except Exception as e:
#             print(e)
#             sims.append([[-1]])
```

```
#     sims = [x[0][0] for x in sims]
#     idx = np.argsort(sims)[::-1] # sorting in descending order bc were deali

#     # return the indexes of K nearest neighbors
#     return idx[:K], sims
```

artists_search is a dictionary where the KEY is the artist, and the VALUE is a pd.DataFrame with songs that come up when you search their name. Need to find the audio features for each of these tracks, normalize them, and find the cosine distance between each song & the average for the playlist (also try to compare to the song from the original playlist that the artist came from)

In [ ]:
```
# def get_recommendations(search_res, comparison_values, k):
#     '''Get song song recommendations for a list of artists'''

#     recommended_songs = pd.DataFrame(index=None, columns=['artist', 'track',

#     for key in search_res.keys():
#         songs_search_res = artists_search[key]
#         search_res_audio_features = playlist1.get_audio_features(songs_search
#         scaled_search_res_audio_features = playlist1.normalize_audio_features

#         scaled_audio_features =  np.array(scaled_search_res_audio_features)
#         avg_audio_values = np.array(comparison_values)

#         idxs, sims = knn_search(comparison_values, scaled_audio_features, k,

#         tmp = pd.DataFrame({'artist': key,
#             'track' : songs_search_res.iloc[idxs,:].track,
#             'track_id' : songs_search_res.iloc[idxs,:].track_id,
#             'similarity' : [sims[x] for x in idxs]}, index=None)
#         recommended_songs = pd.concat([recommended_songs, tmp], axis=0, ignor

#     return recommended_songs
```

In [ ]:
```
recs
```

In [ ]:
```
recs.index
```

use playlist_add_items to add all tracks to the playlist once its created

In [ ]:
```
# def get_playlist_id(playlist_name, spotify):
#     playlists = spotify.current_user_playlists()
#     for playlist in playlists['items']:
#         if playlist['name'] == playlist_name:
#             print(playlist.keys())
#             return playlist['id']
```

In [ ]:
```
# def build_playlist(tracks, playlist_id, final_len, spotify):
#     '''Uses the spotify API to put songs onto a new playlist'''

#     tracks_to_add = tracks.iloc[:final_len,:]

#     user_id = sp.me()['id']
```

```
#     spotify.user_playlist_add_tracks(user_id, playlist_id, tracks_to_add.trac
```

In [ ]:
```
# def create_new_playlist(recs, playlist_length):
#     '''Creates a new playlist by sorting recommendations in descending order

#     SCOPE = "playlist-modify-public"
#     sp = spotipy.Spotify(auth_manager=SpotifyOAuth(scope=SCOPE,
#                                                    client_id=SPOTIFY_CLIENT_I
#                                                    client_secret=SPOTIFY_SECR
#                                                    redirect_uri=REDIRECT_URI)
#                          )

#     recs = recs.sort_values(by='similarity', ascending=False)
#     rec_tracks_idx = recs.index[:final_playlist_len, ]
#     recommendations = recs.loc[rec_tracks_idx, :]

#     playlist_name = input('Choose a name for your new playlist:')
#     playlist_desc = input('Add a description for your new playlist (OPTIONAL)

#     name = playlist_name if (len(playlist_name) > 0) else f'New Playlist (DSC
#     desc = playlist_desc if (len(playlist_desc) > 0) else f'Playlist generate


#     user_id = sp.me()['id']
#     sp.user_playlist_create(user_id, name, public=True, description=desc)

#     playlist_id = get_playlist_id(name, sp)

#     build_playlist(recommendations, playlist_id, playlist_length, sp)

#     playlist_url = sp.playlist(playlist_id)['external_urls']['spotify']

#     print('DONE!')
#     print(f'Here is the link to your new playlist: {playlist_url}')
```

In [ ]:
```
# SCOPE = "playlist-modify-public"
# sp2 = spotipy.Spotify(auth_manager=SpotifyOAuth(scope=SCOPE,
#                                                 client_id=config.SPOTIFY_C
#                                                 client_secret=config.SPOTI
#                                                 redirect_uri=config.REDIRE
```

Using standardscaler

In [ ]:
```
# playlist_link2, final_playlist_len2 = create_dataframes.get_user_playlist()
# playlist2 = playlist.Playlist(playlist_link2, sp, StandardScaler)
# artist_counts2, artist_id_counts2 = playlist2.get_artist_counts()
# avg_audio_values2 = pd.DataFrame(playlist2.normalized_audio_features.mean(axi
# artists_search2 = create_dataframes.artist_search_results(artist_counts2.colu
# recs2 = playlist2.get_recommendations(artists_search2, avg_audio_values, k=5)
# #create_dataframes.create_new_playlist(recs, final_playlist_len)
```

In [ ]:
```
```

the issue is that when scaling audio features some are set to NaN. Handle with a try-except

block because im not too sure how often it is going to happen

so i have 5 songs for each artist that appears in the playlist which are 'simlilar' to the songs that are already in the playlist based on audio features.

- now build a new playlist to return to the user:
  - look at all of the recommended songs, and can choose the top X song that are most similar to the playlist audio features average value
  - MAKE SURE that the songs in the new playlist are NOT in the old playlist
  - number of songs by a given artist in the new playlist should be influenced by frequencies of artists in the original playlist

11/14 commented functions out & made sure that they can be imported by py files (consider renaming)

now try to do PCA on the features & run the code on that data

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
m = len(playlist1.df_playlist.popularity)
```

In [ ]:
```
std_scaled_popularity = StandardScaler().fit_transform(np.array(playlist1.df_pl
```

In [ ]:
```
minmax_scaled_popularity = MinMaxScaler().fit_transform(np.array(playlist1.df_p
```

try adding each of std & minmax scaled artist popularity to the data. (BUT WE HAVE TO GET POPULARITY IN THE SEARCH RESULTS AS WELL) perform PCA

In [ ]:
```
norm_features = playlist1.normalized_audio_features
norm_features
```

In [ ]:
```
norm_features.loc[:,13] = std_scaled_popularity
norm_features
```

In [ ]:
```
from sklearn import decomposition
```

In [ ]:
```
pca = decomposition.PCA(svd_solver='randomized')
pca_minmax_playlist_track_audio_features = pca.fit_transform(norm_features)
np.set_printoptions(precision=2,suppress=True, edgeitems=5, linewidth=120)
print(pca.explained_variance_ratio_)
```

In [ ]:
```
pca.explained_variance_ratio_[:5].sum()
```

In [ ]:
```
varPercentage = pca.explained_variance_ratio_*100
fig = plt.figure(figsize=(12,8))
ax = fig.add_subplot(111)
```

```
ax.plot(range(14), varPercentage[:], marker='^')
plt.xlabel('Principal Component Number')
plt.ylabel('Percentage of Variance')
plt.xticks(range(14))
plt.show()
```

In [ ]:
```
pca_transform_data = pca_minmax_playlist_track_audio_features[:,:5]
pca_transform_data
```

In [ ]:
```
# norm_features.loc[:,13] = minmax_scaled_popularity
# norm_features
```

In [ ]:
```
# pca = decomposition.PCA(svd_solver='randomized')
# pca_minmax_playlist_track_audio_features = pca.fit_transform(norm_features)
# np.set_printoptions(precision=2,suppress=True, edgeitems=5, linewidth=120)
# print(pca.explained_variance_ratio_)
```

In [ ]:
```
# varPercentage = pca.explained_variance_ratio_*100
# fig = plt.figure(figsize=(12,8))
# ax = fig.add_subplot(111)
# ax.plot(range(13), varPercentage[:], marker='^')
# plt.xlabel('Principal Component Number')
# plt.ylabel('Percentage of Variance')
# plt.xticks(range(13))
# plt.show()
```

In [ ]:
```
# norm_features.loc[:,13] = std_scaled_popularity
# norm_features
```

In [ ]:
```
playlist1.normalized_audio_features = pd.DataFrame(pca_transform_data)
playlist1.normalized_audio_features
```

In [ ]:
```
artist_counts, artist_id_counts = playlist1.get_artist_counts()
avg_audio_values = pd.DataFrame(playlist1.normalized_audio_features.mean(axis=0
artists_search = create_dataframes.artist_search_results(artist_counts.columns)
recs = playlist1.get_recommendations(artists_search, avg_audio_values, k=5)
create_dataframes.create_new_playlist(recs, final_playlist_len)
```

try clustering the playlist

In [ ]:
```
#get original data back not PCA yet
playlist_link, final_playlist_len = create_dataframes.get_user_playlist()
playlist1 = playlist.Playlist(playlist_link, sp)
artist_counts, artist_id_counts = playlist1.get_artist_counts()
avg_audio_values = pd.DataFrame(playlist1.normalized_audio_features.mean(axis=0
artists_search = create_dataframes.artist_search_results(artist_counts.columns)
recs = playlist1.get_recommendations(artists_search, avg_audio_values, k=5)
#create_dataframes.create_new_playlist(recs, final_playlist_len)
```