Notebook 02: Data acquisition notebook

Goal:

The goal of this notebook is to use functional abstraction to create three pandas data frames, which conatain spotify data for all the users we have tokens for, without running into any connection issues. The first data frame, the playlists data frame, will include information on the users' playlists (playlist ID, playlist name). The second frame, the tracks data frame, will include information on the tracks within the users' playlists (track name, track ID, playlist ID). The third, the audio data frame, will contain the audio data on the tracks in the tracks data frame. The tracks data frame can be thought of as the linking table between the playlists, and audio data frames. We will use this data to conduct on audio analysis on our sampled users' playlists.

Start and Refresh Function:

In [28]:

```
#set-up
import requests
from requests_oauthlib import OAuth2Session
import importlib
import json
import pprint
import pandas as pd
pp = pprint.PrettyPrinter(indent=2)
```

In [146]:

```
def start(user):
    '''
    the purpose of this function is to start a session for a user with the spotify api. The function re-freshes
    the users token if necessary.
    parameters:
        -user: a string that specifies, which user the function should start a session for
    return values:
        -keychain: a .txt file that acts as a user's keychain with an up to date token
        -token: a dictionary that contains the user's token
        -current_user: a string that specifies the user of a given session with the spotify api.
    '''
    current_user = user
    with open('keychain.txt') as json_file: #loads keychain
        keychain = json.load(json_file)
    token = keychain['spotify']['owners'][user]
    client_id = keychain['spotify']['client_id']
    redirect = keychain['spotify']['redirect_uris'][1]
    session = OAuth2Session(client_id, token=token) #creates session
    D = {}
    D = token['access_token']
    url = 'https://api.spotify.com/v1/me'
    response = requests.get(url,params=token)
    if response.status_code != 200: #checks if the connection isn't valid
        keychain,token = refresh(keychain,token,session,user) #calls the refresh function to update token
        return keychain, token, current_user
    else:
        return keychain, token, current_user
```

Additional Comments:

- I think the biggest challenge of getting these functions to work was conceptualize the idea of loading, and writing to the text file, which contained the users tokens. Additionally, we wanted a function that could start a session for any user, but also handle refreshing, thus we included the refresh function within the start function. Thus if any connection fails, only the start function needs be called, effectively restarting any sesssion.

In [96]:
```
def refresh(keychain,token,session,user):
    '''
    the purpose of this function is to refresh the token of the user, who is attempting to start a session
    with the spotify api.
    parameters:
        -keychain: a .txt file that acts as a user's keychain with an expired token
        -token: the user's expired token
        -session: an oAuth object that represent the session that is trying to be established.
        -user: a string that specifies, which user the function should start a session for
    return values:
        -keychain: a .txt file that acts as a user's keychain with an up to date token
        -token: a dictionary that contains the user's up to date token
    '''
    refresh_url = keychain['spotify']['token_uri']
    token = session.refresh_token(refresh_url, #refreshes token
            client_id=keychain['spotify']['client_id'],
            client_secret=keychain['spotify']['client_secret'])
    keychain['spotify']['owners'][user] = token
    with open('keychain.txt', 'w') as outfile: #writes to keychain
        json.dump(keychain, outfile)
    with open('keychain.txt') as json_file:  #re-loads keychain
        keychain = json.load(json_file)

    return keychain, token
```

Frame Creation for a single user

In [97]:
```
def userIO(token,current_user):
    '''
    the purpose of this function is to access the current user's id info, which is necessary for various queries
    to the spotify API.
    parameters:
        -token: a dictionary that contains the user's token
        -current_user: a string that specifies the user of a given session with the spotify api.
    return values:
        -user_info: a string containing the current user's spotify id information
    '''
    D = {}
    D['access_token'] = token['access_token']
    url = 'https://api.spotify.com/v1/me'
    response = requests.get(url, params=D)
    if response != 200: #restart session if the token has expired
        start(current_user)
    user_info = json.loads(response.text)
    return user_info['id']

user_id = userIO(token,current_user)
```

In [187]:
```
def playlists(user_id,current_user,token):
    '''
    The purpose of this function is twofold. First the function compiles a list of playlist ids
    for the current user's playlists, which will be used in the tracks function to get information about the track.
    Second, the function creates a pandas data frame of all of the user's playlists.
    parameters:
        -user_id: a string containing the current user's spotify id information, returned from userIO function
        -token: a dictionary that contains the user's token
        -current_user: a string that specifies the user of a given session with the spotify api.
    return values:
        -playlistDf: a pandas data frame of the user's playlist names and ids as well as the current user's
        id, which will serve as a kind of foreign key.
        -playlists_id: a list of the playlist_ids, which will be used to find track info.
    '''
    D = {}
    playlist_table = [] #list of dictionaries for all playlists
    playlists_id = []
    D['access_token'] = token['access_token']
    url = 'https://api.spotify.com/v1/users/{}/playlists'
    response = requests.get(url.format(user_id), params=D)
    try:
        start(current_user) #makes sure session is valid
        playlists = json.loads(response.text)
        for item in range(len(playlists['items'])):
            table_dic = {} #builds a dictionary containing essential data for a single playlist
            table_dic['user_id'] = user_id
            table_dic['playlist_id'] = playlists['items'][item]['id']
            table_dic['name'] = playlists['items'][item]['name']
            playlist_table.append(table_dic)
            playlist_id = playlists['items'][item]['id']
            playlists_id.append(playlist_id) #appends dictionary to list
    except: #some of the playlist data is prohibted, so this except lets us skip over the prohibited data
        pass
    playlistDf = pd.DataFrame(playlist_table)
    return playlistDf, tracks_url
```

In [199]:

```
def tracks(user_id, tracks_url,current_user,token):
    '''
    The purpose of this function is twofold. First the function compiles a list of lists of the track ids,
    which will be used in the audio function to get the songs audio information. Second, the function creates
    a pandas data frame of all of the names and ids of all of the tracks in the user's playlists, along with
    the current user's id.
    parameters:
        -user_id: a string containing the current user's spotify id information, returned from userIO function
        -token: a dictionary that contains the user's token
        -current_user: a string that specifies the user of a given session with the spotify api.
        -tracks_url
    return values:
        -track_df: a pandas data frame of all of the names and ids for all of the tracks in the user's playlists,
        along with the current user's id.
        -track_ids: a list of lists of the track ids.
    '''
    D = {}
    D['access_token'] = token['access_token']
    url = 'https://api.spotify.com/v1/users/{}/playlists/{}/tracks'
    track_ids = [] #stores track ids
    track_table = []
    for item in tracks_url:
        response = requests.get(url.format(user_id,item), params=D)
        try:
            start(current_user)
            tracks = json.loads(response.text) #loads playlists track urls
            playlist_tracks = []
            for items in range(len(tracks['items'])): #parses through playlists ids
                playlist_dict = {}
                append_track = tracks['items'][items]['track']['id']
                playlist_tracks.append(append_track)
                playlist_dict['track_id'] = tracks['items'][items]['track']['id']
                playlist_dict['song_name'] = tracks['items'][items]['track']['name']
                playlist_dict['user_id'] =  user_id
                playlist_dict['playlist_id'] = item
                track_table.append(playlist_dict)
            track_ids.append(playlist_tracks)
            track_table.append(playlist_dict)
        except:
            pass
    track_df = pd.DataFrame(track_table)
    return track_ids, track_df #creates data frame, and passes along track ids to the audio function.
```

In [20]:

```
def audio(user_id, track_ids,current_user,token):
    '''
    The purpose of this function is to create a data frame of all of the audio data for the tracks in the user's
    playlists.
    parameters:
        -user_id: a string containing the current user's spotify id information, returned from userIO function
        -token: a dictionary that contains the user's token
        -current_user: a string that specifies the user of a given session with the spotify api.
        -track_ids: a list of lists of the track ids.
    return values:
       -songsDf: a pandas data frame of all of the audio data for the tracks in the user's playlists.
    '''
    D = {}
    D['access_token'] = token['access_token']
    url = 'https://api.spotify.com/v1/audio-features/?ids={}'
    song_table = []
    for items in track_ids:
        '''
        transforms list of track urls into a single string of comma seperated track urls, which allows the
        function to get audio data for multiple tracks, which is much more efficient than getting them one by one.
        '''
        filter_track = list(filter(None, items))
        track_url = ",".join(filter_track)
        response = requests.get(url.format(track_url), params=D)
        try:
            start(current_user)
            songs = json.loads(response.text)
            for item in range(len(songs['audio_features'])):
                append_dic = songs['audio_features'][item]
                song_table.append(append_dic)
        except:
            pass
    songsDf = pd.DataFrame(song_table)
    return songsDf
```

Additional Comments:

- The biggest challenge with these function was making sure that each function was able to grab the correct data from the spotify endpoint, create the appropriate data frame, and pass a part of that data to the next function. For example, originally the playlist function passed a list of urls that linked directly to the track information to the track function. However this made it near impossible to get the playlist ids in the tracks data frame, which was crucial to how the data frames related to eachother. In honesty, the tracks and playlist functions should probably be split into two. Lastly, we had to figure out how to do a get request on the audio data for mutiple tracks at a time otherwise the function took 5 minutes to run (see comment in code).

Build Function: Creating Frames for multiple users

In [200]:
```
def builder(users):
    '''
    The purpose of this function is to iterate over the previously defined functions to get the spotify data
    for each of the users we have tokens for, and concatenate their data together.
    parameters:
        -users: a list of some, or all the users we have tokens for
    return values:
        -playlist_main: the data frame created by the playlists function, but contains all users info.
        -track_main: the data frame created by the tracks function, but contains all users info.
        -audio_main: the data frame created by the audio function, but contains all users info.
    '''
    playlist_merge = []
    track_merge = []
    audio_merge = []
    for user in users:
        keychain, token, current_user = start(user)
        user_id = userIO(token,current_user)

        playlistDf, tracks_url = playlists(user_id,current_user,token)
        playlist_merge.append(playlistDf)

        track_ids,track_df = tracks(user_id, tracks_url,current_user,token)
        track_merge.append(track_df)

        audioframe = audio(user_id,track_ids,current_user,token)
        audio_merge.append(audioframe)

    playlist_main = pd.concat(playlist_merge) #concatenates dataframes together
    track_main = pd.concat(track_merge)
    audio_main = pd.concat(audio_merge)

    return playlist_main,track_main,audio_main
```

Additional Comments:

- This function was pretty straight forward to create. It simply generalizes the steps for a single session to multiple. The hardest part was making sure the data frame function were working properly, and efficiently. We probably made this function a little too earlier.

Checking Results

In [246]:
```
playlist_df, track_df, audio_df = builder(['eric','miranda','gezim','luke','ben'])
```

In [237]:
```
#rename ideas for analysis
track_df['user_id'] = track_df['user_id'].replace('1266353543','eric')
track_df['user_id'] = track_df['user_id'].replace('mirandaarina','miranda')
track_df['user_id'] = track_df['user_id'].replace('1216177234','gezim')
track_df['user_id'] = track_df['user_id'].replace('125346134','luke')
track_df['user_id'] = track_df['user_id'].replace('benliepert','ben')
track_df['user_id'].unique()
```

Out[237]:
```
array(['eric', 'miranda', 'gezim', 'luke', 'ben'], dtype=object)
```

In [247]:
```
print(len(playlist_df))
playlist_df.head()
```

```
72
```

Out[247]:

	name	playlist_id	user_id
0	Your Top Songs 2017	37i9dQZF1E9UBfOE5yO2Rg	1266353543
1	The Ones That Got Away	37i9dQZF1Eak9mtZUi93hp	1266353543
2	Passed	0CluAklouKBTc4lDdaumFh	1266353543
3	DJ	5xQcD38XyjsbveWoXufZ04	1266353543
4	House	2ueo7tfdlg5DF08XxLFLEQ	1266353543

In [233]:
```
print(len(track_df))
track_df.head()
```

```
2148
```

Out[233]:

	playlist_id	song_name	track_id	user_id
0	0CluAklouKBTc4lDdaumFh	A Face In The Crowd	4tSZr210OTY6upjNYfEYUI	1266353543
1	0CluAklouKBTc4lDdaumFh	Run Of The Mill	3S574gsoQJl826YjsuRqSr	1266353543
2	0CluAklouKBTc4lDdaumFh	Out On The Weekend - Remastered Album Version	7DqktFsRwJa0XDFPMjV1xJ	1266353543
3	0CluAklouKBTc4lDdaumFh	Strangers	7obb4s6A7gf0Lc2AGxodMy	1266353543
4	0CluAklouKBTc4lDdaumFh	Stephanie Says - Original Mix	7brL0ZuueQZUgpDgheNcqs	1266353543

In [236]:
```
print(len(audio_df))
audio_df.head()
```

```
2092
```

Out[236]:

	acousticness	analysis_url	danceability	duration_ms	energy	id	instrumentalness	key	liveness	loudness
0	0.1100	https://api.spotify.com/v1/audio-analysis/4tSZ...	0.699	239307	0.610	4tSZr210OTY6upjNYfEYUI	0.001880	4	0.0600	-13.372
1	0.0303	https://api.spotify.com/v1/audio-analysis/3S57...	0.542	171693	0.643	3S574gsoQJl826YjsuRqSr	0.007070	11	0.1050	-7.864
2	0.0967	https://api.spotify.com/v1/audio-analysis/7Dqk...	0.673	271933	0.265	7DqktFsRwJa0XDFPMjV1xJ	0.033600	9	0.0809	-15.711
3	0.2530	https://api.spotify.com/v1/audio-analysis/7obb...	0.470	198373	0.397	7obb4s6A7gf0Lc2AGxodMy	0.000013	0	0.1090	-9.337
4	0.8510	https://api.spotify.com/v1/audio-analysis/7brL...	0.556	169560	0.284	7brL0ZuueQZUgpDgheNcqs	0.000024	0	0.0965	-14.921

In [239]:
```
#exports to csv
playlist_df.to_csv('Playlists.csv')
track_df.to_csv('Tracks.csv')
audio_df.to_csv('Audio.csv')
```

Additional Comments:

- The lengths of the tracks and audio data frames are different, which shouldn't be the case. Our guess is that some of the users songs are personally uploaded to spotify, and thus we can't get the data on them. Nontheless, the frames should still relate okay. We chose to use csvs, instead of a SQL connection. However, our frames are developed in a somewhat relation style, and would translate easily to an SQL databse.

In [ ]:
```

```