Big Data Systems Assignment

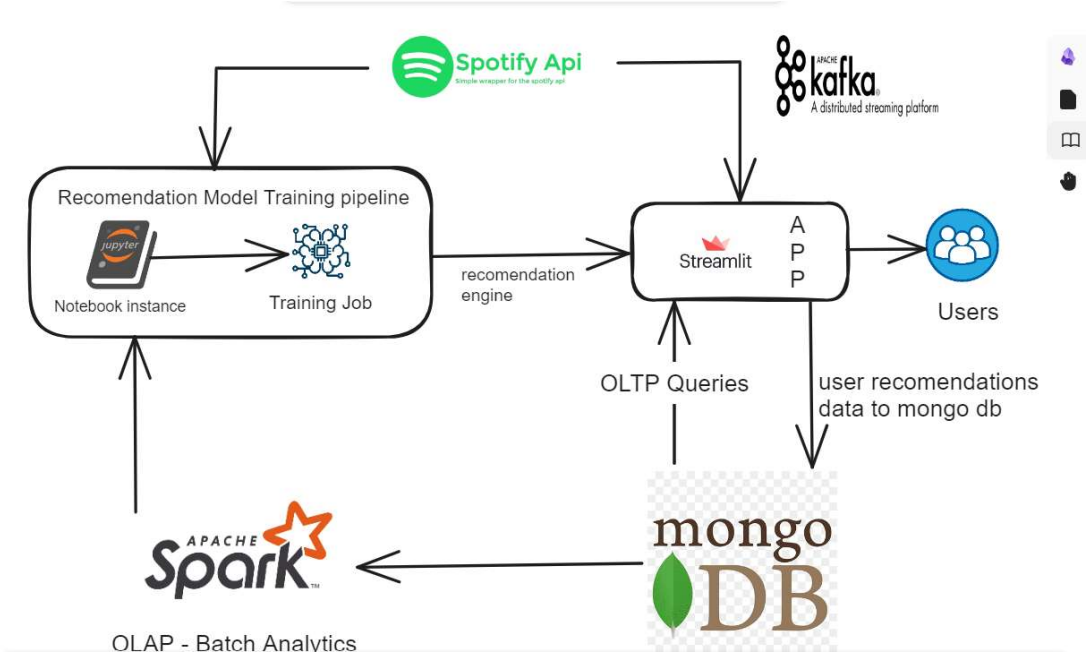
Spotify Music Recomendation System and Analytics

Assignment Submitted by:

Member Details:

1. Sumanta Kumar Patel - 2022OG04032

2. Rahul Khandpur - 2022OG04037



Explanation:

1. Spotify API: The system interacts with the Spotify API to fetch track data and audio features. This API provides access to a wide range of information about Spotify tracks.
2. Data Processing: The data obtained from the Spotify API is processed to extract relevant audio features and create a dataset suitable for music recommendations. This component performs real-time stream processing for immediate user interactions and batch analytics processing for historical data analysis.
3. Real-time Stream Processing: This component handles real-time user interactions and song recommendations. It continuously updates the music recommendation algorithm based on these interactions. The real-time NoSQL database stores and retrieves the necessary data efficiently for real-time processing.
4. Batch Analytics Processing: This component analyzes historical user data to improve the recommendation algorithm over time. It processes large volumes of data in batches, allowing for in-depth analysis and algorithm refinement. The NoSQL database stores and retrieves the historical data efficiently for batch analytics processing.

Rationale:
• **CAP Theorem:** The system prioritizes availability and partition tolerance. Consistency can be relaxed to provide a better user experience and handle system scalability. A

NoSQL database like MongoDB, which offers eventual consistency, is a suitable choice. Partition tolerance is crucial to handle large volumes of data and distributed processing.

• **Tech Choices:**

Spotify API is chosen for comprehensive audio features. **MongoDB** is selected as the NoSQL database due to its flexibility, scalability, and ability to handle both **real-time** and **batch processing**. The use of separate databases for real-time stream processing and batch analytics allows for optimized data management and processing for each use case.

Overall, the architecture enables real-time music recommendations based on user preference and batch analytics to continuously improve the recommendation algorithm. The use of NoSQL databases ensures efficient storage and retrieval of data, facilitating seamless integration with the rest of the system.

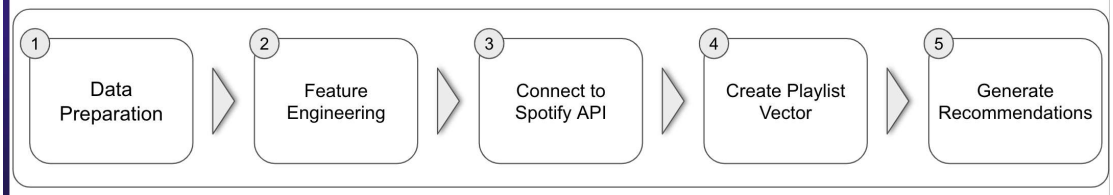
Dependencies

In [ ]:
```
import pandas as pd
import numpy as np
import json
import re
import sys
import itertools

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt


import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from spotipy.oauth2 import SpotifyOAuth
import spotipy.util as util

import warnings
warnings.filterwarnings("ignore")
```

In [ ]:
```
%matplotlib inline
```

In [ ]:
```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
```

How To Generate Recommendations For A Spotify Playlist



Details:

1. **Algorithm Type:** Content Based Recommendation System
2. **Limitations:**
   a. Kaggle Dataset doesn't consist of *all* of the songs in my playlists
   b. Only genre metadata was provided

1. Data Exploration/Preparation

In [ ]:
```
spotify_df = pd.read_csv('Spotify_Track_Genre.csv')
```

In [ ]:
```
spotify_df.head(5)
```

Out[ ]:

	Unnamed: 0	track_id	artists	album_name	track_name	po
0	0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	
1	1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	
2	2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	
3	3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	
4	4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	

5 rows × 21 columns

Note: From the overview of the code, it is noticed that there is an additional column "Unnamed:0" similar to row index which is not required in further analysis, so column "Unnamed:0" is dropped below.

In [ ]:
```
spotify_df.drop(columns = ['Unnamed: 0'], axis=1, inplace=True)
```

In [ ]:
```
spotify_df.head(5)
```

Out[ ]:

	track_id	artists	album_name	track_name	popularity	du
0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	
1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	
2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	
3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	
4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	

In [ ]:
```
spotify_df.rename(columns = {'track_id':'id'}, inplace = True)
```

In [ ]:
```
spotify_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 114000 entries, 0 to 113999
Data columns (total 20 columns):
 #   Column            Non-Null Count   Dtype  
---  ------            --------------   -----  
 0   id                114000 non-null  object 
 1   artists           113999 non-null  object 
 2   album_name        113999 non-null  object 
 3   track_name        113999 non-null  object 
 4   popularity        114000 non-null  int64  
 5   duration_ms       114000 non-null  int64  
 6   explicit          114000 non-null  bool   
 7   danceability      114000 non-null  float64
 8   energy            114000 non-null  float64
 9   key               114000 non-null  int64  
 10  loudness          114000 non-null  float64
 11  mode              114000 non-null  int64  
 12  speechiness       114000 non-null  float64
 13  acousticness      114000 non-null  float64
 14  instrumentalness  114000 non-null  float64
 15  liveness          114000 non-null  float64
 16  valence           114000 non-null  float64
 17  tempo             114000 non-null  float64
 18  time_signature    114000 non-null  int64  
 19  track_genre       114000 non-null  object 
dtypes: bool(1), float64(9), int64(5), object(5)
memory usage: 16.6+ MB
```

In [ ]:
```
spotify_df.isnull().sum()
```

Out[ ]:
```
id                  0
artists             1
album_name          1
track_name          1
popularity          0
duration_ms         0
explicit            0
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
time_signature      0
track_genre         0
dtype: int64
```

In [ ]:
```
spotify_df = spotify_df.dropna()
```

2. Feature Engineering

- Normalize float variables

- OHE for Popularity

- Create TF-IDF features off of artist genres

In [ ]:
```
float_cols = spotify_df.dtypes[spotify_df.dtypes == 'float64'].index.values
```

In [ ]:
```
float_cols
```

Out[ ]:
```
array(['danceability', 'energy', 'loudness', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo'],
      dtype=object)
```

In [ ]:
```
ohe_cols = 'popularity'
```

In [ ]:
```
spotify_df['popularity'].describe()
```

Out[ ]:
```
count    113999.000000
mean         33.238827
std          22.304959
min           0.000000
25%          17.000000
50%          35.000000
75%          50.000000
max         100.000000
Name: popularity, dtype: float64
```

In [ ]:
```
# create 5 point buckets for popularity
spotify_df['popularity_red'] = spotify_df['popularity'].apply(lambda x: int(x/5)
```

In [ ]:
```
spotify_df.head(5)
```

Out[ ]:

	id	artists	album_name	track_name	popularity	du
0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	
1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	
2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	
3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	
4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	

5 rows × 21 columns

In [ ]:
```
# Check for missing values in 'track_genre' column
missing_values = spotify_df['track_genre'].isnull().sum()
print("Missing values in 'track_genre' column:", missing_values)

# Inspect unique values in 'track_genre' column
unique_genres = spotify_df['track_genre'].explode().unique()
print("Unique genres:", unique_genres)
print("Number of unique genres:", len(unique_genres))
print("Data Type in track_genre:", spotify_df['track_genre'].dtype)
```

```
Missing values in 'track_genre' column: 0
Unique genres: ['acoustic' 'afrobeat' 'alt-rock' 'alternative' 'ambient' 'anime'
 'black-metal' 'bluegrass' 'blues' 'brazil' 'breakbeat' 'british'
 'cantopop' 'chicago-house' 'children' 'chill' 'classical' 'club' 'comedy'
 'country' 'dance' 'dancehall' 'death-metal' 'deep-house' 'detroit-techno'
 'disco' 'disney' 'drum-and-bass' 'dub' 'dubstep' 'edm' 'electro'
 'electronic' 'emo' 'folk' 'forro' 'french' 'funk' 'garage' 'german'
 'gospel' 'goth' 'grindcore' 'groove' 'grunge' 'guitar' 'happy'
 'hard-rock' 'hardcore' 'hardstyle' 'heavy-metal' 'hip-hop' 'honky-tonk'
 'house' 'idm' 'indian' 'indie-pop' 'indie' 'industrial' 'iranian'
 'j-dance' 'j-idol' 'j-pop' 'j-rock' 'jazz' 'k-pop' 'kids' 'latin'
 'latino' 'malay' 'mandopop' 'metal' 'metalcore' 'minimal-techno' 'mpb'
 'new-age' 'opera' 'pagode' 'party' 'piano' 'pop-film' 'pop' 'power-pop'
 'progressive-house' 'psych-rock' 'punk-rock' 'punk' 'r-n-b' 'reggae'
 'reggaeton' 'rock-n-roll' 'rock' 'rockabilly' 'romance' 'sad' 'salsa'
 'samba' 'sertanejo' 'show-tunes' 'singer-songwriter' 'ska' 'sleep'
 'songwriter' 'soul' 'spanish' 'study' 'swedish' 'synth-pop' 'tango'
 'techno' 'trance' 'trip-hop' 'turkish' 'world-music']
Number of unique genres: 114
Data Type in track_genre: object
```

In [ ]:
```
# Convert lists of genres to strings for TF-IDF vectorization
spotify_df['track_genre_str'] = spotify_df['track_genre'].apply(lambda x: ' '.jo
```

In [ ]:
```
spotify_df.head()
```

Out[ ]:

	id	artists	album_name	track_name	popularity	du
0	5SuOikwiRyPMVoIQDJUgSV	Gen Hoshino	Comedy	Comedy	73	
1	4qPNDBW1i3p13qLCt0Ki3A	Ben Woodward	Ghost (Acoustic)	Ghost - Acoustic	55	
2	1iJBSr7s7jYXzM8EGcbK5b	Ingrid Michaelson;ZAYN	To Begin Again	To Begin Again	57	
3	6lfxq3CG4xtTiEg7opyCyx	Kina Grannis	Crazy Rich Asians (Original Motion Picture Sou...	Can't Help Falling In Love	71	
4	5vjLSffimiIP26QG5WcN2K	Chord Overstreet	Hold On	Hold On	82	

5 rows × 22 columns

In [ ]:
```
# Convert lists of genres to strings for TF-IDF vectorization
spotify_df['track_genre_str'] = spotify_df['track_genre'].apply(lambda x: ''.joi
```

In [ ]:
```
print(spotify_df['track_genre_str'].head())
```

```
0    acoustic
1    acoustic
2    acoustic
3    acoustic
4    acoustic
Name: track_genre_str, dtype: object
```

In [ ]:
```
# function to create OHE features
# this gets passed later on
def ohe_prep(df, column, new_name):
    tf_df = pd.get_dummies(df[column])
    feature_names = tf_df.columns
    tf_df.columns = [new_name + "|" + str(i) for i in feature_names]
    tf_df.reset_index(drop = True, inplace = True)
    return tf_df
```

TF-IDF Overview

TF-IDF automatically emphasizes (i.e. weights) metadata terms based on how often they appear across our entire catalog.

Example:

Song 1

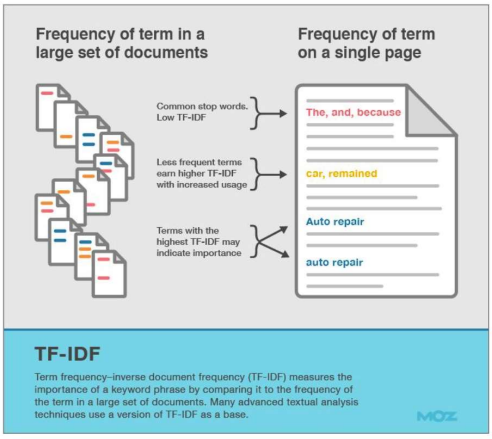
["Rock", "Metal"]

Rock	Pop	Metal
0.5	0	1.0

Song 2

["Rock", "Pop"]

Rock	Pop	Metal
0.5	1.0	0



In [ ]:
```
#function to build entire feature set
def create_feature_set(df, float_cols):

    # Process spotify df to create a final set of features that will be used to

    #tfidf genre lists
    tfidf = TfidfVectorizer()
    tfidf_matrix =  tfidf.fit_transform(df['track_genre_str'])
    genre_df = pd.DataFrame(tfidf_matrix.toarray())
    genre_df.columns = ['genre' + "|" + i for i in tfidf.get_feature_names_out()
    genre_df.reset_index(drop = True, inplace=True)

    popularity_ohe = ohe_prep(df, 'popularity_red','pop') * 0.15

    #scale float columns
    floats = df[float_cols].reset_index(drop = True)
    scaler = MinMaxScaler()
    floats_scaled = pd.DataFrame(scaler.fit_transform(floats), columns = floats.

    #concanenate all features
    final = pd.concat([genre_df, floats_scaled, popularity_ohe], axis = 1)

    #add song id
    final['id']=df['id'].values

    return final
```

In [ ]:
```
complete_feature_set = create_feature_set(spotify_df, float_cols=float_cols)#.me
```

In [ ]:
```
complete_feature_set.head()
```

Out[ ]:

	genre|acoustic	genre|afrobeat	genre|age	genre|alt	genre|alternative	genre|ambier
0	1.0	0.0	0.0	0.0	0.0	0.
1	1.0	0.0	0.0	0.0	0.0	0.
2	1.0	0.0	0.0	0.0	0.0	0.
3	1.0	0.0	0.0	0.0	0.0	0.
4	1.0	0.0	0.0	0.0	0.0	0.

5 rows × 145 columns

3. Connect to Spotify API

Useful links:

1. https://developer.spotify.com/dashboard/
2. https://spotipy.readthedocs.io/en/2.16.1/

In [ ]:
```
#client id and secret for my application
client_id = '7a832874ab9243239180d6c34dc02f02'
client_secret= '5a95c79436ee4f90a3d73958b19f2de0'
```

In [ ]:
```
scope = 'user-library-read'

if len(sys.argv) > 1:
    username = sys.argv[1]
else:
    print("Usage: %s username" % (sys.argv[0],))
    sys.exit()
```

In [ ]:
```
auth_manager = SpotifyClientCredentials(client_id=client_id, client_secret=clien
sp = spotipy.Spotify(auth_manager=auth_manager)
```

In [ ]:
```
token = util.prompt_for_user_token(scope, client_id= client_id, client_secret=cl
```

In [ ]:
```
sp = spotipy.Spotify(auth=token)
```

In [ ]:
```
sp.current_user_playlists()
```

Out[ ]:

```
{'href': 'https://api.spotify.com/v1/users/31wwhhgevitsxuew6moremo6fkia/playlists?offset=0&limit=50',
 'items': [{'collaborative': False,
   'description': '',
   'external_urls': {'spotify': 'https://open.spotify.com/playlist/0MMPRGkEnd1gp0YnINmqn5'},
   'href': 'https://api.spotify.com/v1/playlists/0MMPRGkEnd1gp0YnINmqn5',
   'id': '0MMPRGkEnd1gp0YnINmqn5',
   'images': [{'height': 640,
     'url': 'https://i.scdn.co/image/ab67616d0000b273f4a2ccbe20d6d52f16816812',
     'width': 640}],
   'name': 'Motivational',
   'owner': {'display_name': 'Sumanta Patel',
    'external_urls': {'spotify': 'https://open.spotify.com/user/31wwhhgevitsxuew6moremo6fkia'},
    'href': 'https://api.spotify.com/v1/users/31wwhhgevitsxuew6moremo6fkia',
    'id': '31wwhhgevitsxuew6moremo6fkia',
    'type': 'user',
    'uri': 'spotify:user:31wwhhgevitsxuew6moremo6fkia'},
   'primary_color': None,
   'public': True,
   'snapshot_id': 'Myw4MmI5Y2Q5ZjljMDljM2YzMjUwYWVkNTY2MDNkZTdhNDJhZmM0YWZm',
   'tracks': {'href': 'https://api.spotify.com/v1/playlists/0MMPRGkEnd1gp0YnINmqn5/tracks',
    'total': 1},
   'type': 'playlist',
   'uri': 'spotify:playlist:0MMPRGkEnd1gp0YnINmqn5'},
  {'collaborative': False,
   'description': '',
   'external_urls': {'spotify': 'https://open.spotify.com/playlist/5uG7tystxnVHpySvvK2aYg'},
   'href': 'https://api.spotify.com/v1/playlists/5uG7tystxnVHpySvvK2aYg',
   'id': '5uG7tystxnVHpySvvK2aYg',
   'images': [{'height': 640,
     'url': 'https://i.scdn.co/image/ab67616d0000b273a2fc41b0dd6ce4f0d16a4c46',
     'width': 640}],
   'name': 'Sleep_Time',
   'owner': {'display_name': 'Sumanta Patel',
    'external_urls': {'spotify': 'https://open.spotify.com/user/31wwhhgevitsxuew6moremo6fkia'},
    'href': 'https://api.spotify.com/v1/users/31wwhhgevitsxuew6moremo6fkia',
    'id': '31wwhhgevitsxuew6moremo6fkia',
    'type': 'user',
    'uri': 'spotify:user:31wwhhgevitsxuew6moremo6fkia'},
   'primary_color': None,
   'public': True,
   'snapshot_id': 'NSw2MTZkZWUyMmFkNGY1YTcwMTkwNmRkNzE4Nzk3MzhkZWJhODhlZGFk',
   'tracks': {'href': 'https://api.spotify.com/v1/playlists/5uG7tystxnVHpySvvK2aYg/tracks',
    'total': 3},
   'type': 'playlist',
   'uri': 'spotify:playlist:5uG7tystxnVHpySvvK2aYg'},
  {'collaborative': False,
   'description': '',
   'external_urls': {'spotify': 'https://open.spotify.com/playlist/5Awp8pMG1Adde6B0PuLJZl'},
   'href': 'https://api.spotify.com/v1/playlists/5Awp8pMG1Adde6B0PuLJZl',
   'id': '5Awp8pMG1Adde6B0PuLJZl',
   'images': [{'height': 640,
     'url': 'https://i.scdn.co/image/ab67616d0000b273ffb343926530168be4724dd4',
```

```
     'width': 640}],
   'name': 'EDM',
   'owner': {'display_name': 'Sumanta Patel',
    'external_urls': {'spotify': 'https://open.spotify.com/user/31wwhhgevitsxue
w6moremo6fkia'},
    'href': 'https://api.spotify.com/v1/users/31wwhhgevitsxuew6moremo6fkia',
    'id': '31wwhhgevitsxuew6moremo6fkia',
    'type': 'user',
    'uri': 'spotify:user:31wwhhgevitsxuew6moremo6fkia'},
   'primary_color': None,
   'public': True,
   'snapshot_id': 'NSw2YmU2NGE4ODFlNTEyNDQ4NmJjNzYzMzdhMjc3YTZiMGVjNDg5YzFm',
   'tracks': {'href': 'https://api.spotify.com/v1/playlists/5Awp8pMG1Adde6B0PuL
JZl/tracks',
    'total': 3},
   'type': 'playlist',
   'uri': 'spotify:playlist:5Awp8pMG1Adde6B0PuLJZl'},
  {'collaborative': False,
   'description': '',
   'external_urls': {'spotify': 'https://open.spotify.com/playlist/74PBiFGMu2f8
L6Q3oIw28C'},
   'href': 'https://api.spotify.com/v1/playlists/74PBiFGMu2f8L6Q3oIw28C',
   'id': '74PBiFGMu2f8L6Q3oIw28C',
   'images': [{'height': 640,
     'url': 'https://i.scdn.co/image/ab67616d0000b2732166b28714239e31b48aeb17',
     'width': 640}],
   'name': 'Epic',
   'owner': {'display_name': 'Sumanta Patel',
    'external_urls': {'spotify': 'https://open.spotify.com/user/31wwhhgevitsxue
w6moremo6fkia'},
    'href': 'https://api.spotify.com/v1/users/31wwhhgevitsxuew6moremo6fkia',
    'id': '31wwhhgevitsxuew6moremo6fkia',
    'type': 'user',
    'uri': 'spotify:user:31wwhhgevitsxuew6moremo6fkia'},
   'primary_color': None,
   'public': True,
   'snapshot_id': 'MywxYTQ1ZWJkNzMwMmI2ZjY1MjEzY2Y3ZDM4YmIzZWQ3NWRjMGJmZmEy',
   'tracks': {'href': 'https://api.spotify.com/v1/playlists/74PBiFGMu2f8L6Q3oIw
28C/tracks',
    'total': 1},
   'type': 'playlist',
   'uri': 'spotify:playlist:74PBiFGMu2f8L6Q3oIw28C'},
  {'collaborative': False,
   'description': '',
   'external_urls': {'spotify': 'https://open.spotify.com/playlist/0qRCsMTBitoA
zoSZSSM1FI'},
   'href': 'https://api.spotify.com/v1/playlists/0qRCsMTBitoAzoSZSSM1FI',
   'id': '0qRCsMTBitoAzoSZSSM1FI',
   'images': [{'height': 640,
     'url': 'https://mosaic.scdn.co/640/ab67616d0000b273178419da701ab7c7f693c9a
cab67616d0000b27321ebf49b3292c3f0f575f0f5ab67616d0000b273ad185265c9a1d05ae3b662
0bab67616d0000b273cf390065f5a3336f12143e16',
     'width': 640},
    {'height': 300,
     'url': 'https://mosaic.scdn.co/300/ab67616d0000b273178419da701ab7c7f693c9a
cab67616d0000b27321ebf49b3292c3f0f575f0f5ab67616d0000b273ad185265c9a1d05ae3b662
0bab67616d0000b273cf390065f5a3336f12143e16',
     'width': 300},
    {'height': 60,
     'url': 'https://mosaic.scdn.co/60/ab67616d0000b273178419da701ab7c7f693c9ac
```

```
ab67616d0000b27321ebf49b3292c3f0f575f0f5ab67616d0000b273ad185265c9a1d05ae3b6620
bab67616d0000b273cf390065f5a3336f12143e16',
     'width': 60}],
   'name': '80s_Rock',
   'owner': {'display_name': 'Sumanta Patel',
    'external_urls': {'spotify': 'https://open.spotify.com/user/31wwhhgevitsxue
w6moremo6fkia'},
    'href': 'https://api.spotify.com/v1/users/31wwhhgevitsxuew6moremo6fkia',
    'id': '31wwhhgevitsxuew6moremo6fkia',
    'type': 'user',
    'uri': 'spotify:user:31wwhhgevitsxuew6moremo6fkia'},
   'primary_color': None,
   'public': True,
   'snapshot_id': 'MTIsZGRhMTJkNWExZmJhZDliM2IwNmQ5NDBmZmM3MjM3MjE4NDBmMjg3Nw=
=',
   'tracks': {'href': 'https://api.spotify.com/v1/playlists/0qRCsMTBitoAzoSZSSM
1FI/tracks',
    'total': 10},
   'type': 'playlist',
   'uri': 'spotify:playlist:0qRCsMTBitoAzoSZSSM1FI'}],
 'limit': 50,
 'next': None,
 'offset': 0,
 'previous': None,
 'total': 5}
```


In [ ]:
```python
user_playlist_info = {}
list_photo = {}

for playlist in sp.current_user_playlists()['items']:
    playlist_name = playlist['name']
    playlist_uri = playlist['uri'].split(':')[2]

    # Check if images list is not empty before accessing the URL
    if playlist.get('images') and len(playlist['images']) > 0:
        playlist_image_url = playlist['images'][0]['url']
    else:
        # Handle case when images list is empty or doesn't exist
        playlist_image_url = "No Image Available"  # Placeholder or default valu

    user_playlist_info[playlist_name] = playlist_uri
    list_photo[playlist_uri] = playlist_image_url
```

In [ ]:
```python
user_playlist_info
```

Out[ ]:
```
{'Motivational': '0MMPRGkEnd1gp0YnINmqn5',
 'Sleep_Time': '5uG7tystxnVHpySvvK2aYg',
 'EDM': '5Awp8pMG1Adde6B0PuLJZl',
 'Epic': '74PBiFGMu2f8L6Q3oIw28C',
 '80s_Rock': '0qRCsMTBitoAzoSZSSM1FI'}
```

In [ ]:
```python
# defined the create_necessary_outputs function
def create_necessary_outputs(playlist_name, id_dic, df):
    playlist = pd.DataFrame()

    for ix, track_info in enumerate(sp.playlist(id_dic[playlist_name])['tracks']
        if 'track' in track_info:
            track = track_info['track']
            if 'id' in track and track['id'] in df['id'].values:
                playlist.loc[ix, 'artist'] = track['artists'][0]['name']
```

```
                playlist.loc[ix, 'name'] = track['name']
                playlist.loc[ix, 'id'] = track['id']

                if 'album' in track and 'images' in track['album'] and len(track
                    playlist.loc[ix, 'url'] = track['album']['images'][1]['url']

                if 'added_at' in track_info:
                    playlist.loc[ix, 'date_added'] = track_info['added_at']

    playlist['date_added'] = pd.to_datetime(playlist.get('date_added'))  # Conve
    #playlist = playlist.dropna(subset=['id'])  # Drop rows with NaN values in '

    return playlist
```

In [ ]:
```
playlist_80s_Rock = create_necessary_outputs('80s_Rock', user_playlist_info,spot
```

In [ ]:
```
playlist_80s_Rock
```

Out[ ]:

	artist	name	id	
1	Guns N' Roses	Sweet Child O' Mine	7snQQk1zcKl8gZ92AnueZW	https://i.scdn.co/image/ab67616d00001
4	Bon Jovi	You Give Love A Bad Name	0rmGAIH9LNJewFw7nKzZnc	https://i.scdn.co/image/ab67616d00001
6	Scorpions	Rock You Like a Hurricane - 2011	46QazXxQS0B31CnbRCy8CV	https://i.scdn.co/image/ab67616d00001
7	Bryan Adams	Summer Of '69	0GONea6G2XdnHWjNZd6zt3	https://i.scdn.co/image/ab67616d0000
8	Guns N' Roses	Knockin' On Heaven's Door	4JiEyzf0Md7KEFFGWDDdCr	https://i.scdn.co/image/ab67616d00001
9	Guns N' Roses	Paradise City	6eN1f9KNmiWEhpE2RhQqB5	https://i.scdn.co/image/ab67616d00001

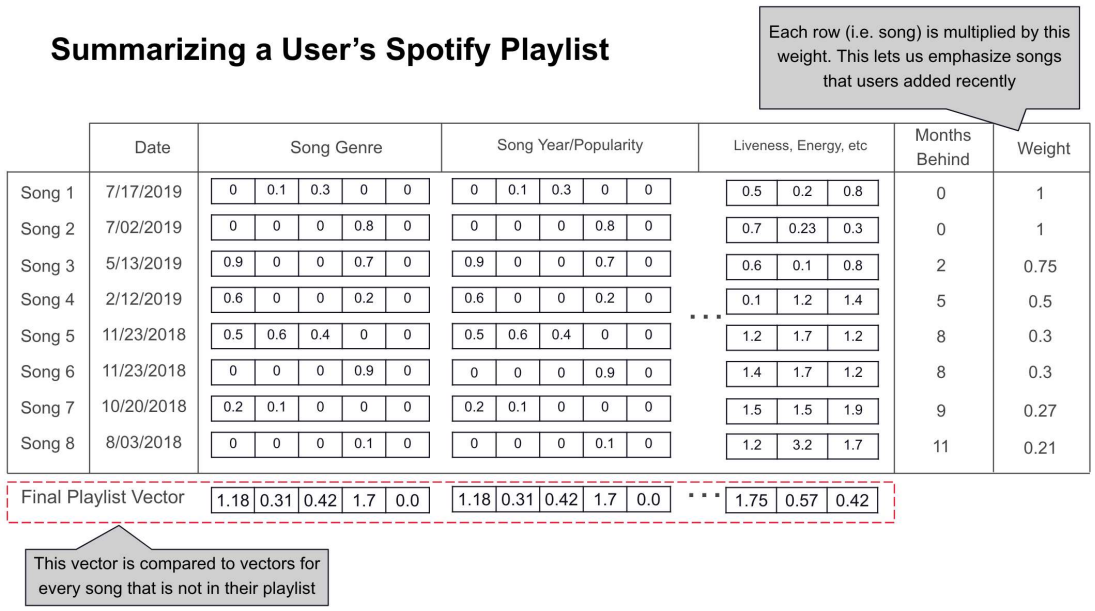
In [ ]:
```
"""
# Iterate through playlist_info dictionary and process each playlist
playlist_data = {}
for playlist_name, playlist_id in playlist_info.items():
    playlist_data[playlist_name] = create_necessary_outputs(playlist_name, playl
"""
```

Out[ ]:
```
'\n# Iterate through playlist_info dictionary and process each playlist\nplayli
st_data = {}\nfor playlist_name, playlist_id in playlist_info.items():\n    pla
ylist_data[playlist_name] = create_necessary_outputs(playlist_name, playlist_in
fo, spotify_df)\n'
```

In [ ]:
```python
from skimage import io
import matplotlib.pyplot as plt

def visualize_songs(df):
    """
    Visualize cover art of the songs in the inputted dataframe

    Parameters:
        df (pandas dataframe): Playlist Dataframe
    """

    temp = df['url'].values
    plt.figure(figsize=(15,int(0.625 * len(temp))))
    columns = 5

    for i, url in enumerate(temp):
        plt.subplot(len(temp) // columns + 1, columns, i + 1)  # Use // for inte

        try:
            image = io.imread(url)
            plt.imshow(image)
            plt.xticks(color='w', fontsize=0.1)
            plt.yticks(color='w', fontsize=0.1)
            plt.xlabel(df['name'].values[i], fontsize=12)
            plt.tight_layout(h_pad=0.4, w_pad=0)
            plt.subplots_adjust(wspace=None, hspace=None)
        except Exception as e:
            #print(f"Unable to load image {url}: {e}")
            plt.text(0.5, 0.5, "Image not available", horizontalalignment='cente
                     verticalalignment='center', transform=plt.gca().transAxes)
            plt.axis('off')  # Hide axis for cases where image cannot be loaded

    plt.show()
```

In [ ]:
```python
visualize_songs(playlist_80s_Rock)
```

4. Create Playlist Vector



In [ ]:
```
def generate_playlist_feature(complete_feature_set, playlist_df, weight_factor):
    complete_feature_set_playlist = complete_feature_set[complete_feature_set['i
    complete_feature_set_playlist = complete_feature_set_playlist.merge(playlist
    complete_feature_set_nonplaylist = complete_feature_set[~complete_feature_se

    playlist_feature_set = complete_feature_set_playlist.sort_values('date_added

    most_recent_date = playlist_feature_set.iloc[0,-1]

    for ix, row in playlist_feature_set.iterrows():
        playlist_feature_set.loc[ix,'months_from_recent'] = int((most_recent_dat

    playlist_feature_set['weight'] = playlist_feature_set['months_from_recent'].

    playlist_feature_set_weighted = playlist_feature_set.copy()
    #print(playlist_feature_set_weighted.iloc[:,:-4].columns)
    playlist_feature_set_weighted.update(playlist_feature_set_weighted.iloc[:,:-
    playlist_feature_set_weighted_final = playlist_feature_set_weighted.iloc[:,
    #playlist_feature_set_weighted_final['id'] = playlist_feature_set['id']

    return playlist_feature_set_weighted_final.sum(axis = 0), complete_feature_s
```

In [ ]:
```
complete_feature_set_playlist_vector_EDM, complete_feature_set_nonplaylist_EDM =
```

In [ ]:
```
complete_feature_set_playlist_vector_EDM.shape
```

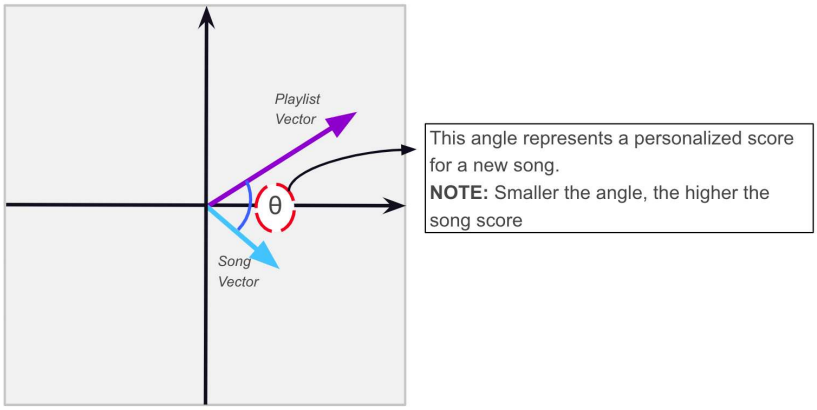
Out[ ]:
```
(144,)
```

5. Generate Recommendations

Calculating Scores for New Songs

Playlist Vector is compared to individual Song Vectors using *cosine similarity* to generate recommendations:

Cosine Similarity Explained:



In [ ]:
```
def generate_playlist_recos(df, features, nonplaylist_features):
    non_playlist_df = df[df['id'].isin(nonplaylist_features['id'].values)]
    non_playlist_df['sim'] = cosine_similarity(nonplaylist_features.drop('id', a
    non_playlist_df_top_40 = non_playlist_df.sort_values('sim',ascending = False
    non_playlist_df_top_40['url'] = non_playlist_df_top_40['id'].apply(lambda x:

    return non_playlist_df_top_40
```

In [ ]:
```
edm_top40 = generate_playlist_recos(spotify_df, complete_feature_set_playlist_ve
```

In [ ]:
```
edm_top40.head(5)
```

Out[ ]:

	id	artists	album_name	track_name	popularity	dur
47208	6KTv0Z8BmVqM7DPxbGzpVC	KISS	Dressed To Kill	Rock And Roll All Nite	77	
47328	1hlveB9M6ijHZRbzZ2teyh	Twisted Sister	Stay Hungry	We're Not Gonna Take It	75	
47165	2R6UrJ8uWbSIiHWmvRQvN8	Metallica	Garage Inc.	Whiskey In The Jar	75	
47560	6Nm8h73ycDG2saCnZV8poF	Rob Zombie	Hellbilly Deluxe	Dragula	76	
47225	7N3PAbqfTjSEU1edb2tY8j	Van Halen	1984 (Remastered)	Jump - 2015 Remaster	78	

5 rows × 24 columns

Important functions dumped below for application building

In [ ]:
```
import pickle
pickle.dump(generate_playlist_recos,open('generate_palylist_recos.pkl','wb'))
pickle.dump(spotify_df,open('spotify_df.pkl','wb'))
```

OLTP Queries - to be execured from the Streamlit App

Find one

```
{ "query_type": "find_one",  "collection_name": "Music_Recomendatio.user_songs_recomended",  "filter": {  "artists": "Dean Martin"  }  }
```

Find Query

```
{ "query_type": "find", "collection_name": "Music_Recomendatio.user_songs_recomended", "filter": { "field_name": "value" } }
```

Count Query

```
{ "query_type": "count" }
```