# Music Recommendation System

## PROBLEM STATEMENT :

To Develop a music recommendation system using machine learning to suggest personalized music tracks based on user preferences and listening history.

## DESCRIPTION :

This mini project aims to develop a personalized music recommendation system using machine learning techniques. The system will analyze user preferences and listening habits to suggest relevant and enjoyable music tracks. By leveraging machine learning algorithms, such as collaborative filtering and content-based filtering, the system will be able to provide accurate and tailored recommendations. The project will involve collecting and preprocessing music data, designing and training the recommendation model, and implementing a user-friendly interface for users to interact with the system.

The music recommendation system will enhance user experience by offering curated playlists and song suggestions, thereby increasing user engagement and satisfaction. Additionally, the project will explore the integration of various features such as genre preferences, mood detection, and contextual information to further improve the quality of recommendations. Through this project, we aim to demonstrate the practical application of machine learning in the field of music recommendation, benefiting both music enthusiasts and streaming platforms alike.

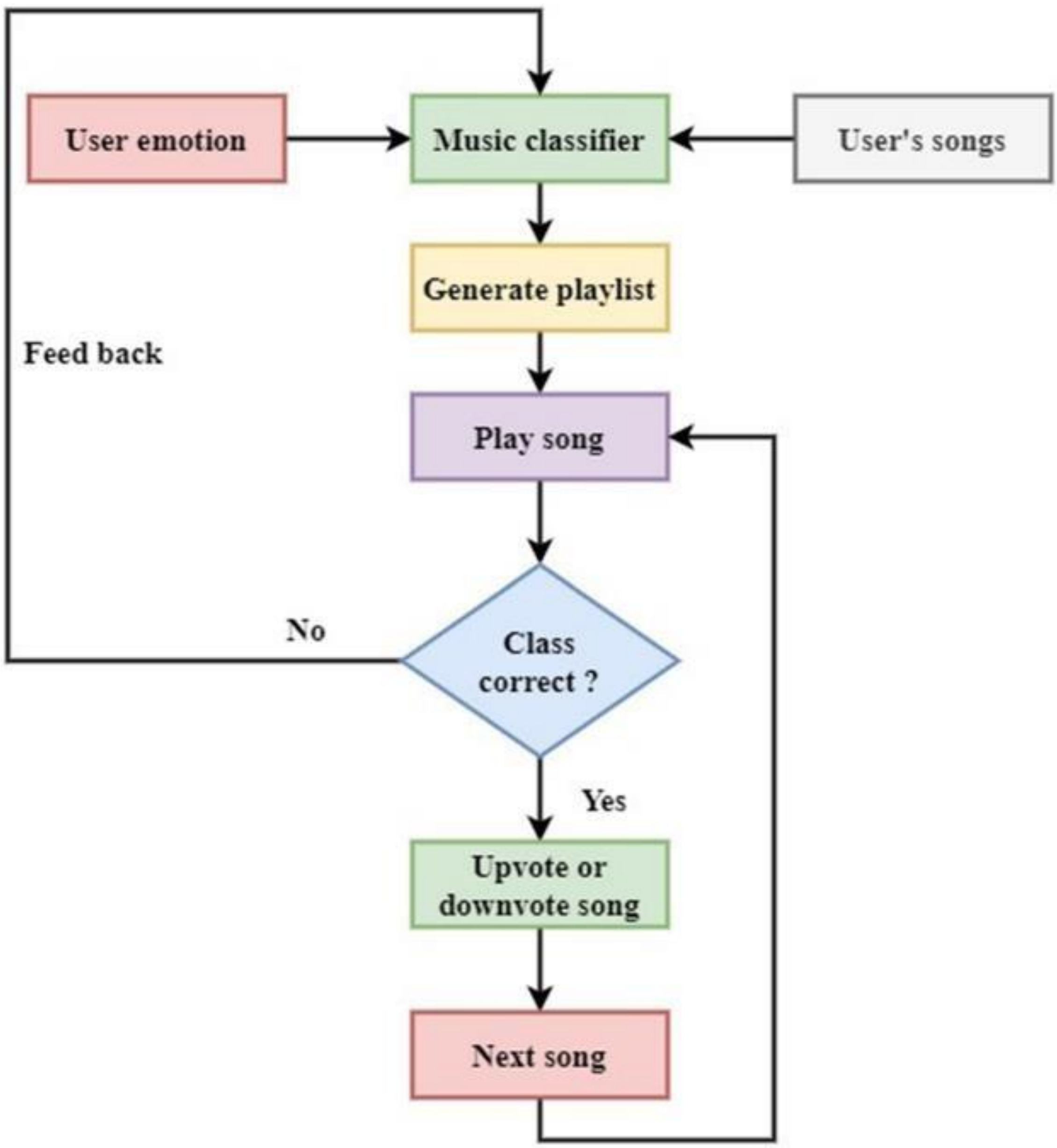
## APPLICATIONS :

- **Personalized Music Streaming:** Tailor music suggestions to each user's taste, boosting engagement.
- **Playlist Curation:** Create playlists based on mood, activity, or genre preferences, simplifying music selection.
- **Music Discovery:** Introduce users to new tracks, encouraging exploration and diversity in music.
- **Improved User Retention:** Keep users engaged with relevant music suggestions, increasing platform loyalty.
- **Targeted Marketing:** Use listening insights to tailor music promotions, enhancing marketing effectiveness.

## SOFTWARE REQUIREMENTS :

- Database Management System
- Machine Learning Libraries
- Web Development Framework
- User Interface Design Tools

## FLOW-CHART :



# CODING :

## Main.py :

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'spotify-
dataset:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-
sets%2F1800580%2F2936818%2Fbundle%2Farchive.zip%3FX-Goog-
Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-
com%2540kaggle-
161607.iam.gserviceaccount.com%252F20240316%252Fauto%252Fstorage%252Fgoog4_re
quest%26X-Goog-Date%3D20240316T102818Z%26X-Goog-Expires%3D259200%26X-
Goog-SignedHeaders%3Dhost%26X-Goog-
Signature%3D2174d140aab12ea64a650fe99043d1fe618e3993f25cf3bee9bdca00ee436d24fd
2f0df8ddf058ac47c8ea3036be38cf882f25e4124eb1cee7a3704a25e9d8797b89d856b0543655
232a5629ab5453f46171e523e58f28da3b531c0a74f1bb48bd35621b93d357cf884b52b5060a4
a86b38289718ad321a20837ca1de998f545b79a22beb38616ca5e28e88c8eca5c2da538cb1fc33
10573a051dd5a9b93438537d9bc5ba9d557166eedf1ec5313b59d5c33cb7954e60e466dc41e1f
7a8fb777bb1cb4916f4f89bc511aef3a780a109092ce09e8054b2ea41b920046d3d89e72aa21a8
4eeee082b9013a8eba859bda33e67d838d7aa1a98b6d039015a5021903,-spotify-tracks-
dataset:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-
sets%2F2570056%2F4372070%2Fbundle%2Farchive.zip%3FX-Goog-
Algorithm%3DGOOG4-RSA-SHA256%26X-Goog-Credential%3Dgcp-kaggle-
com%2540kaggle-
161607.iam.gserviceaccount.com%252F20240316%252Fauto%252Fstorage%252Fgoog4_re
quest%26X-Goog-Date%3D20240316T102818Z%26X-Goog-Expires%3D259200%26X-
Goog-SignedHeaders%3Dhost%26X-Goog-
Signature%3D0dc032c3787baf0c0c05f67fa69a326a629b01b37fd2d9529d79c00ba67ebfa4be
228c45e85b8249cd6806090a592c468e3705c63880cf49a178dcf01d1aebc2b445bfec9f681386
f95cf48b437e761a68640eba21be3416a177ea0b4cdd397e86845ce2eabbafdac761b41f60cac3e
904ed8774741db4d87e6fb6743727201eabba429c6781fb73b10c10647cdc9f08d2a7bd1fe4ccb
0f918fb199ed37ea5e09e84a733a4603a6e894a8553e89a6ac0f1d5060104f179f362697db592b
70f7b01045d66ecf590716096f11719f632939c060e25e327de34108379783fb69fed0222e179f
36abccd0786c91017e84e1b2511dbd0162112e11aee2f2c3ecf3d0c'
```

```
KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'),
target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
```

```
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')


import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics.pairwise import cosine_similarity

import warnings
warnings.filterwarnings('ignore')

#Main Raw Dataframe
df = pd.read_csv("/kaggle/input/-spotify-tracks-dataset/dataset.csv")
df.drop(columns='Unnamed: 0',inplace=True)

#Dataframe for getting year feature of songs
dfYear = pd.read_csv("/kaggle/input/spotify-dataset/data/data.csv")
dfYear = dfYear[['id','year']]
dfYear['track_id'] = dfYear['id']
dfYear.drop(columns='id',inplace=True)

#Merge 2 Dataframe
df = pd.merge(df,dfYear,on='track_id')
display(df.info(),df.head())

# Duplicate Check
df[df.duplicated('track_id')==True]

df[df['track_id']=='6Vc5wAMmXdKIAM7WUoEb7N']

# Crosstab Genre and Song

xtab_song = pd.crosstab(
    df['track_id'],
    df['track_genre']
)

xtab_song = xtab_song*2

display(xtab_song.head(),len(xtab_song))
```

```
# Concatenate the encoded genre columns with the original dataframe

dfDistinct = df.drop_duplicates('track_id')
dfDistinct = dfDistinct.sort_values('track_id')
dfDistinct = dfDistinct.reset_index(drop=True)

xtab_song.reset_index(inplace=True)
data_encoded = pd.concat([dfDistinct, xtab_song], axis=1)
display(data_encoded.head(),len(data_encoded))

numerical_features = ['explicit', 'danceability', 'energy', 'loudness', 'speechiness',
'acousticness', 'instrumentalness', 'liveness', 'valence','year']
scaler = MinMaxScaler()
data_encoded[numerical_features] = scaler.fit_transform(data_encoded[numerical_features])

# Select the relevant columns for computing item similarities
calculatied_features = numerical_features +
list(xtab_song.drop(columns='track_id').columns)

cosine_sim = cosine_similarity(data_encoded[calculatied_features],
data_encoded[calculatied_features])


def get_recommendations(title, N=5):
    indices = pd.Series(data_encoded.index,
index=data_encoded['track_name']).drop_duplicates()

    try:
        idx = indices[title]
        try:
            len(idx)
            temp = 2
        except:
            temp = 1
    except KeyError:
        return "Song not found in the dataset."

    if temp == 2:
        idx = indices[title][0]
    else:
        idx = indices[title]
    sim_scores = list(enumerate(cosine_sim[idx]))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:N+1]
    song_indices = [i[0] for i in sim_scores]
    recommended_songs = data_encoded[['track_name', 'artists',
```

```
'album_name']].iloc[song_indices]

    sim_scores_list = [i[1] for i in sim_scores]
    recommended_list = recommended_songs.to_dict(orient='records')
    for i, song in enumerate(recommended_list):
        song['similarity_score'] = sim_scores_list[i]

    return recommended_list

# Get the recommendations
recommended_songs = get_recommendations("Time", N=5)
if isinstance(recommended_songs, str):
    print(recommended_songs)
else:
    print("Recommended Songs:")
    for song in recommended_songs:
        print(f"Title: {song['track_name']}")
        print(f"Artist: {song['artists']}")
        print(f"Album: {song['album_name']}")
        print(f"Similarity Score: {song['similarity_score']:.2f}")
        print()
```

**SAMPLE OUTPUT:**

```
Recommended Songs:
Title: Suite: Judy Blue Eyes - 2005 Remaster
Artist: Crosby, Stills & Nash
Album: Crosby, Stills & Nash
Similarity Score: 1.00

Title: Red House
Artist: Jimi Hendrix
Album: Are You Experienced
Similarity Score: 1.00

Title: Heroes And Villains - Remastered 2001
Artist: The Beach Boys
Album: Smiley Smile (Remastered)
Similarity Score: 1.00

Title: The Wind Cries Mary
Artist: Jimi Hendrix
Album: Are You Experienced
Similarity Score: 0.99

Title: The Trial
Artist: Pink Floyd
Album: The Wall
Similarity Score: 0.99
```