Electronic Assignment Cover sheet

Students Number:	**20020071**	**20005553**
Name:	**Bodapati Abhiteja**	**Nikant Sharma**

Course Title: M.Sc. Cyber Security

Lecturer Name: Swati Dongre

Module/Subject Title: Advanced Programming Techniques

Assignment Title: CA_TWO_(70%)

MUSIC RECOMMENDED SYSTEM USING PYTHON

Table of contents:

Introduction:

Nowadays, music serves as both a soundtrack for our lives and a source of amusement in the digital age. Finding new songs that suit our tastes has never been easier given the vast amount of music that is readily available to us. This is where technology comes into play, and in this study we explore how to create and use a Python-based music recommender System.

The primary aim of this report is to provide a comprehensive overview of the conception, construction, and application of a Python-based music recommender system. We will delve into the essential data processing techniques, user engagement models, and underlying algorithms that form the foundation of this system, shedding light on the intricate relationship between computational efficiency and personalized music curation.

This report will lead you through the creation of a music recommendation system, starting with data collection and preparation. We'll examine various recommendation algorithms like collaborative filtering, content-based filtering, and hybrids, while also discussing challenges encountered and strategies employed to overcome them. This provides a valuable understanding of refining software and machine learning models iteratively.

We will assess the effectiveness of our music recommendation system by conducting thorough testing and validation. This evaluation will measure its recommendation accuracy, diversity, and ability to provide serendipitous discoveries. By analysing actual user interactions and feedback, our goal is to gauge the practical usefulness and user satisfaction of our Python-based music recommender.

Problem Statement and Requirements:
The objective is to develop an effective music recommendation system that utilizes web application development, machine learning, and data processing to provide users with personalized song suggestions aligned with their music preferences.

Data processing:

- The system should effectively manage a vast music dataset, including crucial details like artist names, song titles, URLs, and lyrical content.
- Prior to analysis, data pre-processing should meticulously refine the dataset by eliminating redundant columns and artifacts to ensure precision.
- Textual information, especially song lyrics, needs to be converted into numerical representations using methods such as TF-IDF vectorisation.

Machine Learning Techniques:

- Machine learning algorithms should be utilized by the system to determine the likeness between songs, focusing on their lyrical content.
- Utilization of the cosine similarity metric is essential to gauge the resemblance between song vectors accurately.
- Recommendations ought to be formulated based on songs sharing considerable similarity with those already preferred by the user, thus augmenting personalization.

Web Application Development:

- Deployment of the music recommendation system as a web application is imperative to offer users an intuitive interface.
- Utilization of Streamlit, a Python library designed for crafting dynamic web applications, is recommended for constructing the user-facing interface.
- Users must have the capability to input their music preferences, such as preferred artists or genres, and obtain tailored song recommendations promptly.

User Experience:

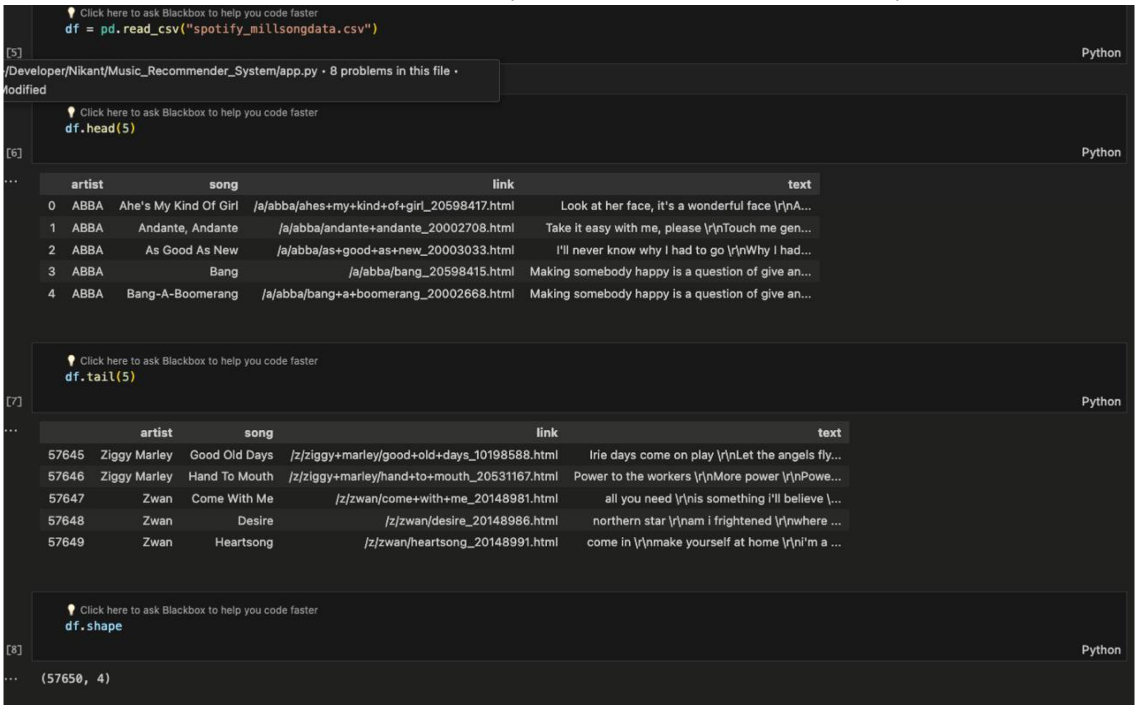
- The web application should provide a smooth and user-friendly experience, featuring transparent guidance and adaptable design.
- Users should be empowered to explore suggested songs, listen to previews, and offer feedback to refine recommendations further.

Scalability and Performance:

- The system should be scalable to handle increasing amounts of data and user interactions over time.
- Metrics related to performance, including the accuracy of recommendations, diversity, and response time, need to be closely observed and enhanced to ensure an exceptional user experience.

The steps involved in building the Music recommended system:

- Download the Dataset: The system utilizes a music dataset sourced from Kaggle, comprising a vast collection of 45 million songs. This dataset consists of four primary columns: artist, song, link, and lyrics. Accessing this dataset from Kaggle provides a rich and diverse pool of musical data to draw recommendations from. (Link will be in the References)



```
Click here to ask Blackbox to help you code faster
df = pd.read_csv("spotify_millsongdata.csv")
[5]                                                                 Python
/Developer/Nikant/Music_Recommender_System/app.py • 8 problems in this file •
Modified

Click here to ask Blackbox to help you code faster
df.head(5)
[6]                                                                 Python
```

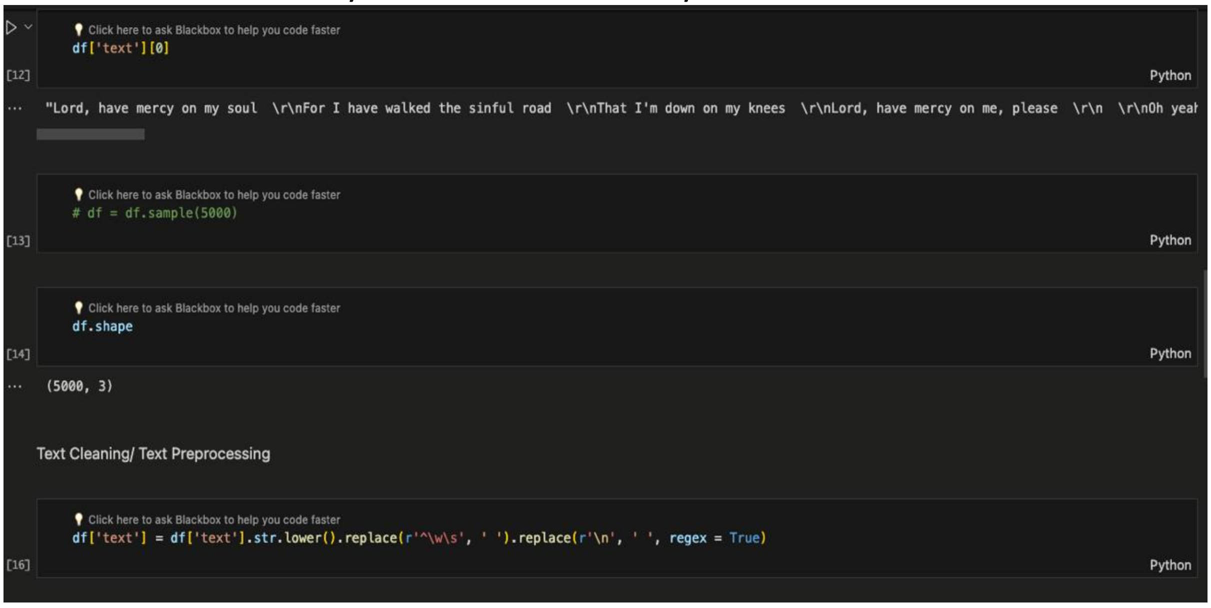
	artist	song	link	text
0	ABBA	Ahe's My Kind Of Girl	/a/abba/ahes+my+kind+of+girl_20598417.html	Look at her face, it's a wonderful face \r\nA...
1	ABBA	Andante, Andante	/a/abba/andante+andante_20002708.html	Take it easy with me, please \r\nTouch me gen...
2	ABBA	As Good As New	/a/abba/as+good+as+new_20003033.html	I'll never know why I had to go \r\nWhy I had...
3	ABBA	Bang	/a/abba/bang_20598415.html	Making somebody happy is a question of give an...
4	ABBA	Bang-A-Boomerang	/a/abba/bang+a+boomerang_20002668.html	Making somebody happy is a question of give an...

```
Click here to ask Blackbox to help you code faster
df.tail(5)
[7]                                                                 Python
```

	artist	song	link	text
57645	Ziggy Marley	Good Old Days	/z/ziggy+marley/good+old+days_10198588.html	Irie days come on play \r\nLet the angels fly...
57646	Ziggy Marley	Hand To Mouth	/z/ziggy+marley/hand+to+mouth_20531167.html	Power to the workers \r\nMore power \r\nPowe...
57647	Zwan	Come With Me	/z/zwan/come+with+me_20148981.html	all you need \r\nis something i'll believe \...
57648	Zwan	Desire	/z/zwan/desire_20148986.html	northern star \r\nam i frightened \r\nwhere ...
57649	Zwan	Heartsong	/z/zwan/heartsong_20148991.html	come in \r\nmake yourself at home \r\ni'm a ...

```
Click here to ask Blackbox to help you code faster
df.shape
[8]                                                                 Python
...  (57650, 4)
```

- Data Pre-processing: Upon obtaining the dataset, the next step involves pre-processing the data. This stage involves cleaning and refining the dataset by removing unnecessary columns and refining the textual content, specifically the lyrics. This cleaning process typically involves removing extraneous characters, extra spaces, backslashes, and any other artifacts that may hinder accurate analysis.

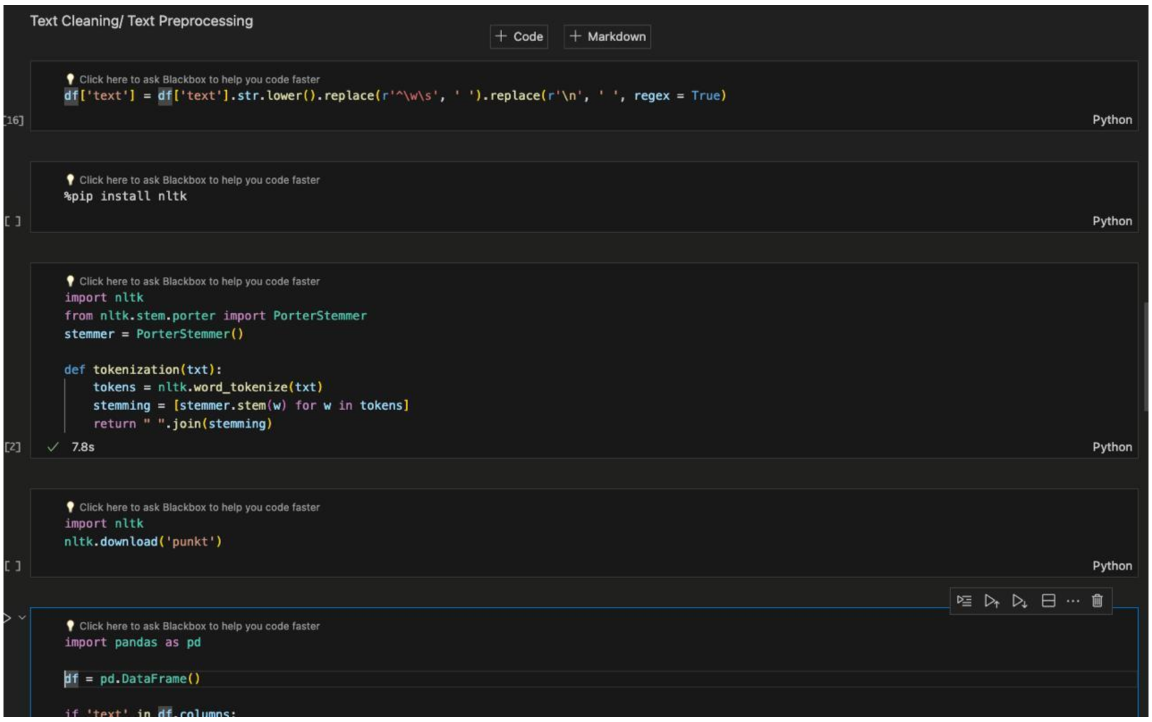


```
Click here to ask Blackbox to help you code faster
df['text'][0]
[12]                                                                Python
...  "Lord, have mercy on my soul  \r\nFor I have walked the sinful road  \r\nThat I'm down on my knees  \r\nLord, have mercy on me, please  \r\n  \r\nOh yeah

Click here to ask Blackbox to help you code faster
# df = df.sample(5000)
[13]                                                                Python

Click here to ask Blackbox to help you code faster
df.shape
[14]                                                                Python
...  (5000, 3)
```

Text Cleaning/ Text Preprocessing

```
Click here to ask Blackbox to help you code faster
df['text'] = df['text'].str.lower().replace(r'^\w\s', ' ').replace(r'\n', ' ', regex = True)
[16]                                                                Python
```

- Text Vectorization: With the data cleaned and organized, the system then proceeds to convert the textual data, specifically the song lyrics, into numerical features that are interpretable by machine learning models. To accomplish this, the system employs TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, a popular technique in natural language processing (NLP) for converting text into numerical representations while considering the importance of each word within the document.


Text Cleaning/ Text Preprocessing

```
[16]
df['text'] = df['text'].str.lower().replace(r'^\w\s', ' ').replace(r'\n', ' ', regex = True)
```

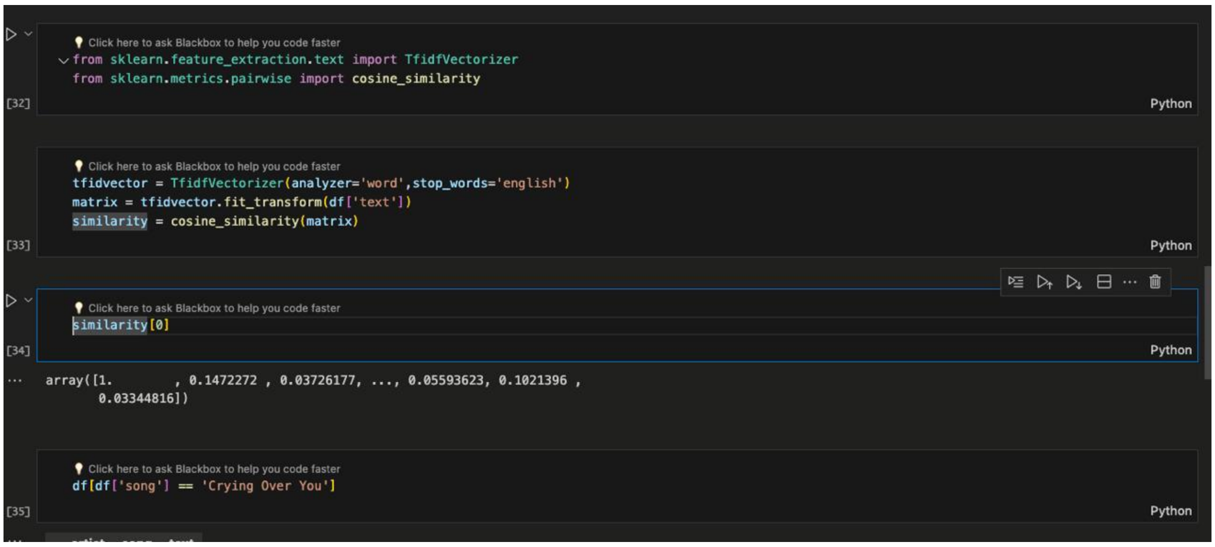
```
%pip install nltk
```

```
[2]
import nltk
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()

def tokenization(txt):
    tokens = nltk.word_tokenize(txt)
    stemming = [stemmer.stem(w) for w in tokens]
    return " ".join(stemming)
✓ 7.8s
```

```
import nltk
nltk.download('punkt')
```

```
import pandas as pd

df = pd.DataFrame()

if 'text' in df.columns:
```

- Calculate Cosine Similarity: Once the lyrics are vectorized, the system calculates the cosine similarity between songs based on their respective lyrical content. Cosine similarity is a metric used to measure the similarity between two vectors, in this case, the numerical representations of song lyrics. By computing the cosine similarity, the system can identify songs with similar lyrical themes or content.



```
[32]
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

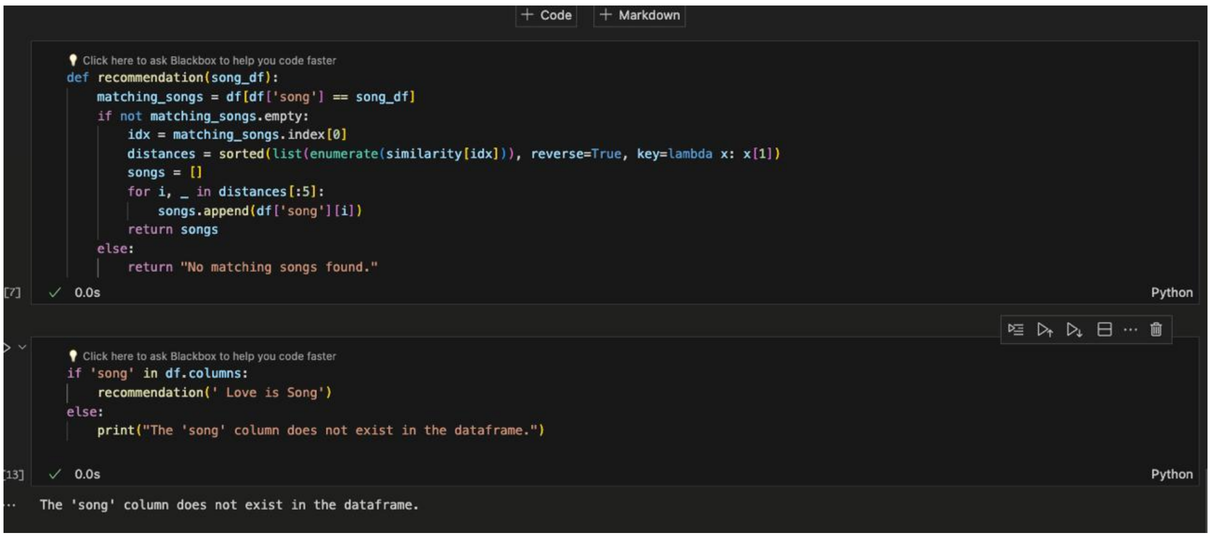
```
[33]
tfidvector = TfidfVectorizer(analyzer='word',stop_words='english')
matrix = tfidvector.fit_transform(df['text'])
similarity = cosine_similarity(matrix)
```

```
[34]
similarity[0]
```

```
array([1.        , 0.1472272 , 0.03726177, ..., 0.05593623, 0.1021396 ,
       0.03344816])
```

```
[35]
df[df['song'] == 'Crying Over You']
```

- Recommend Songs: Leveraging the computed cosine similarities, the system generates recommendations for users based on their preferences. By identifying songs that exhibit high similarity to those already enjoyed by the user, the system suggests tracks that align closely with their musical tastes, thereby enhancing the user's listening experience.

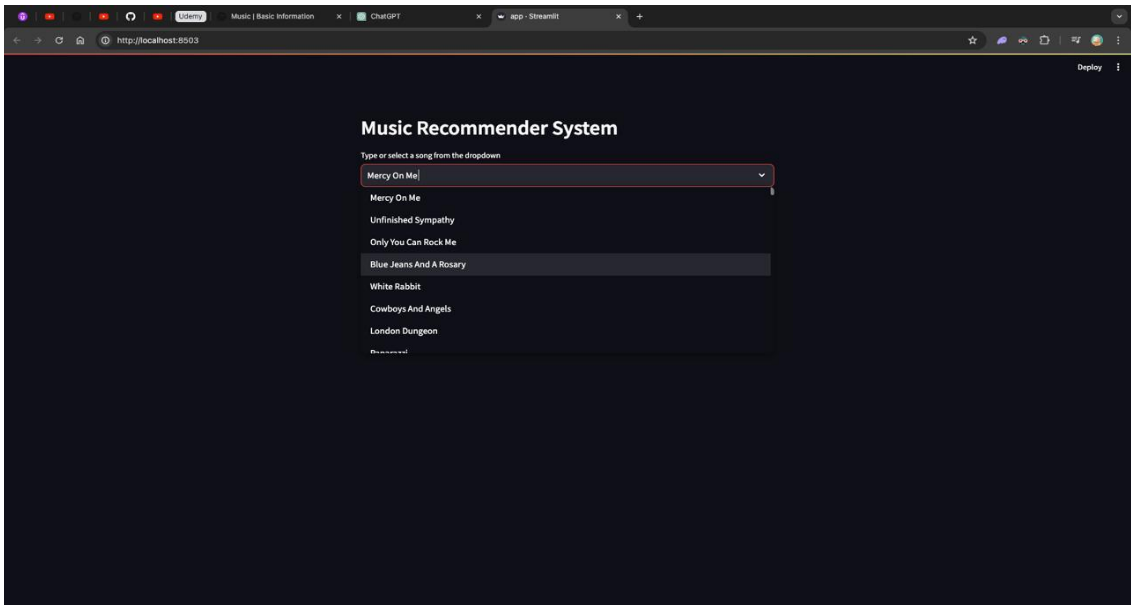


```
def recommendation(song_df):
    matching_songs = df[df['song'] == song_df]
    if not matching_songs.empty:
        idx = matching_songs.index[0]
        distances = sorted(list(enumerate(similarity[idx])), reverse=True, key=lambda x: x[1])
        songs = []
        for i, _ in distances[:5]:
            songs.append(df['song'][i])
        return songs
    else:
        return "No matching songs found."
```

```
if 'song' in df.columns:
    recommendation(' Love is Song')
else:
    print("The 'song' column does not exist in the dataframe.")
```

```
The 'song' column does not exist in the dataframe.
```

- Web Application Deployment: Finally, the music recommendation system is deployed as a web application using Streamlit, a popular Python library for creating interactive web applications. This deployment enables users to access the recommendation system through a user-friendly interface, where they can input their preferences and receive personalized song recommendations in real-time.



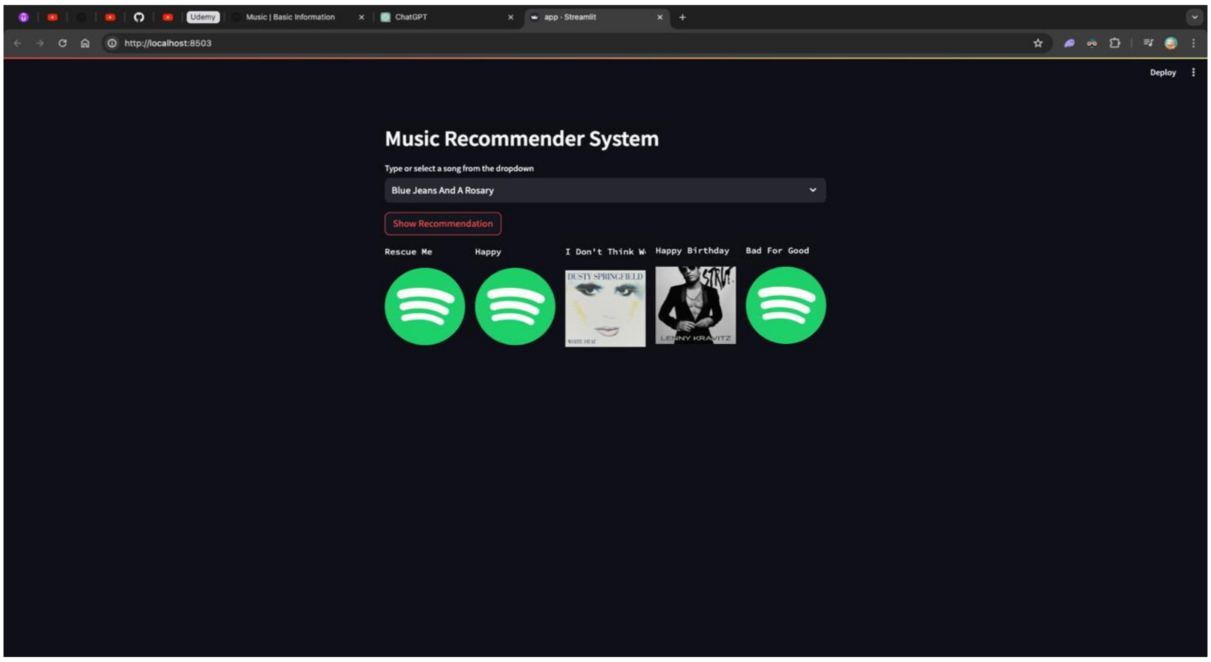
Music Recommender System

Type or select a song from the dropdown

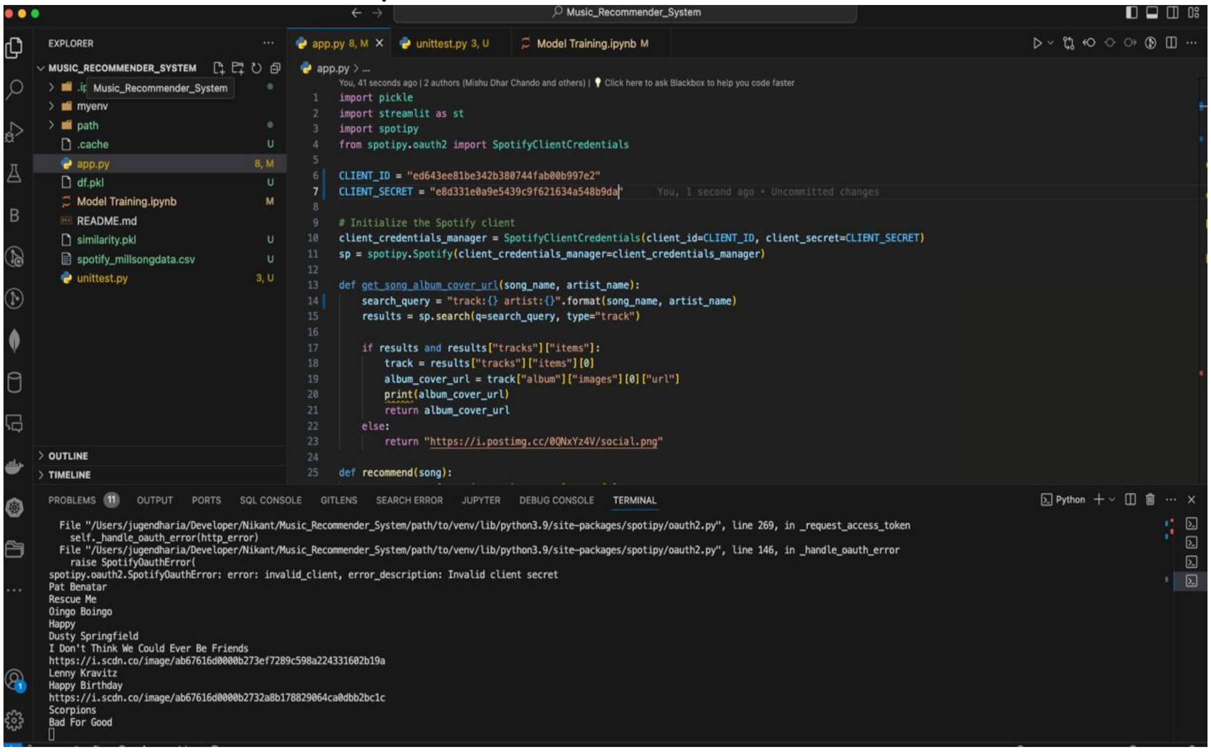
Mercy On Me

- Mercy On Me
- Unfinished Sympathy
- Only You Can Rock Me
- Blue Jeans And A Rosary
- White Rabbit
- Cowboys And Angels
- London Dungeon

The NLP based model responded with 5 song recommendations with the closest cosine similarity value.



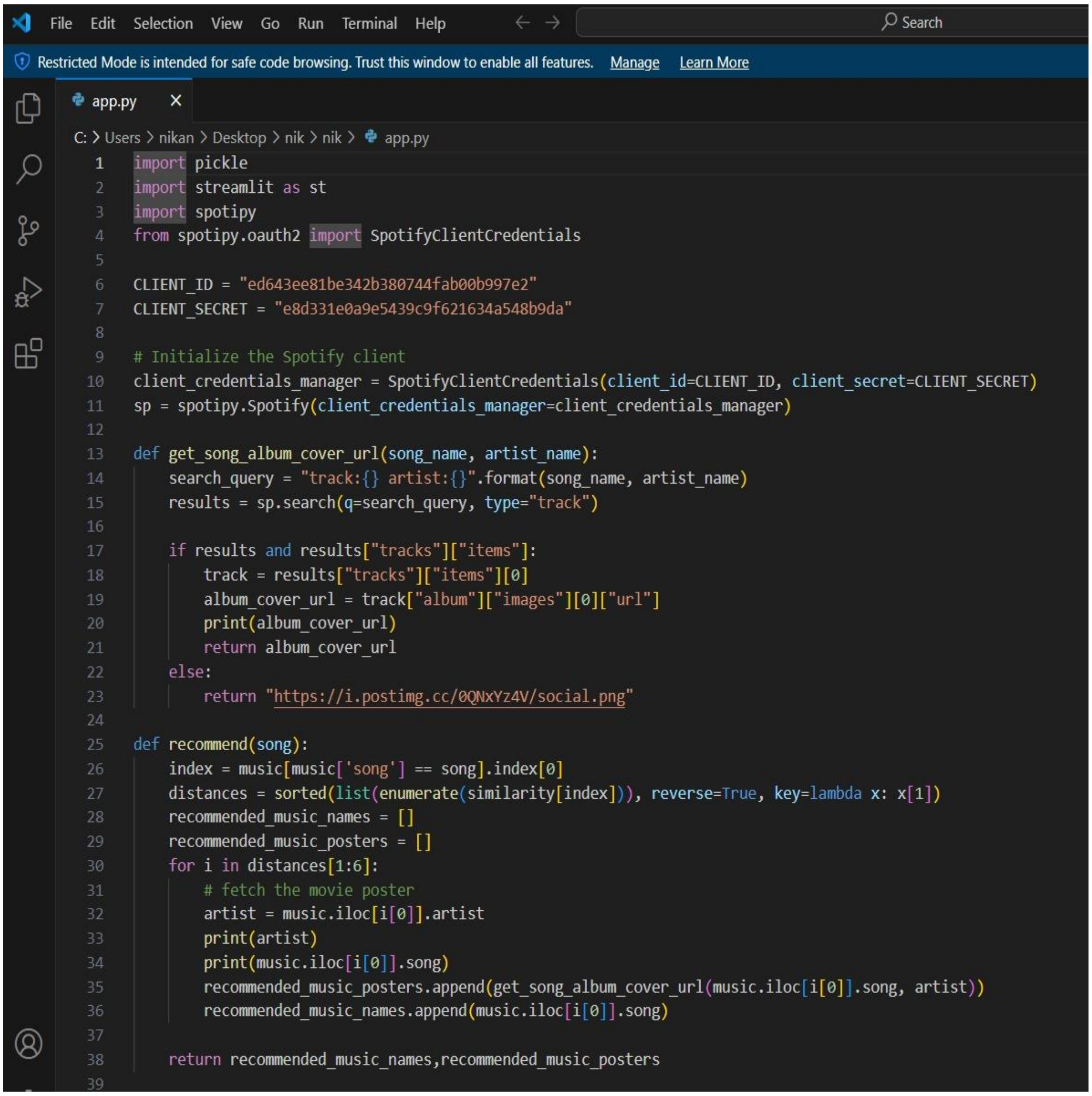
Recommendation output on terminal



```python
import pickle
import streamlit as st
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

CLIENT_ID = "ed643ee81be342b380744fab00b997e2"
CLIENT_SECRET = "e8d331e0a9e5439c9f621634a548b9da"

# Initialize the Spotify client
client_credentials_manager = SpotifyClientCredentials(client_id=CLIENT_ID, client_secret=CLIENT_SECRET)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

def get_song_album_cover_url(song_name, artist_name):
    search_query = "track:{} artist:{}".format(song_name, artist_name)
    results = sp.search(q=search_query, type="track")

    if results and results["tracks"]["items"]:
        track = results["tracks"]["items"][0]
        album_cover_url = track["album"]["images"][0]["url"]
        print(album_cover_url)
        return album_cover_url
    else:
        return "https://i.postimg.cc/0QNxYz4V/social.png"

def recommend(song):
```

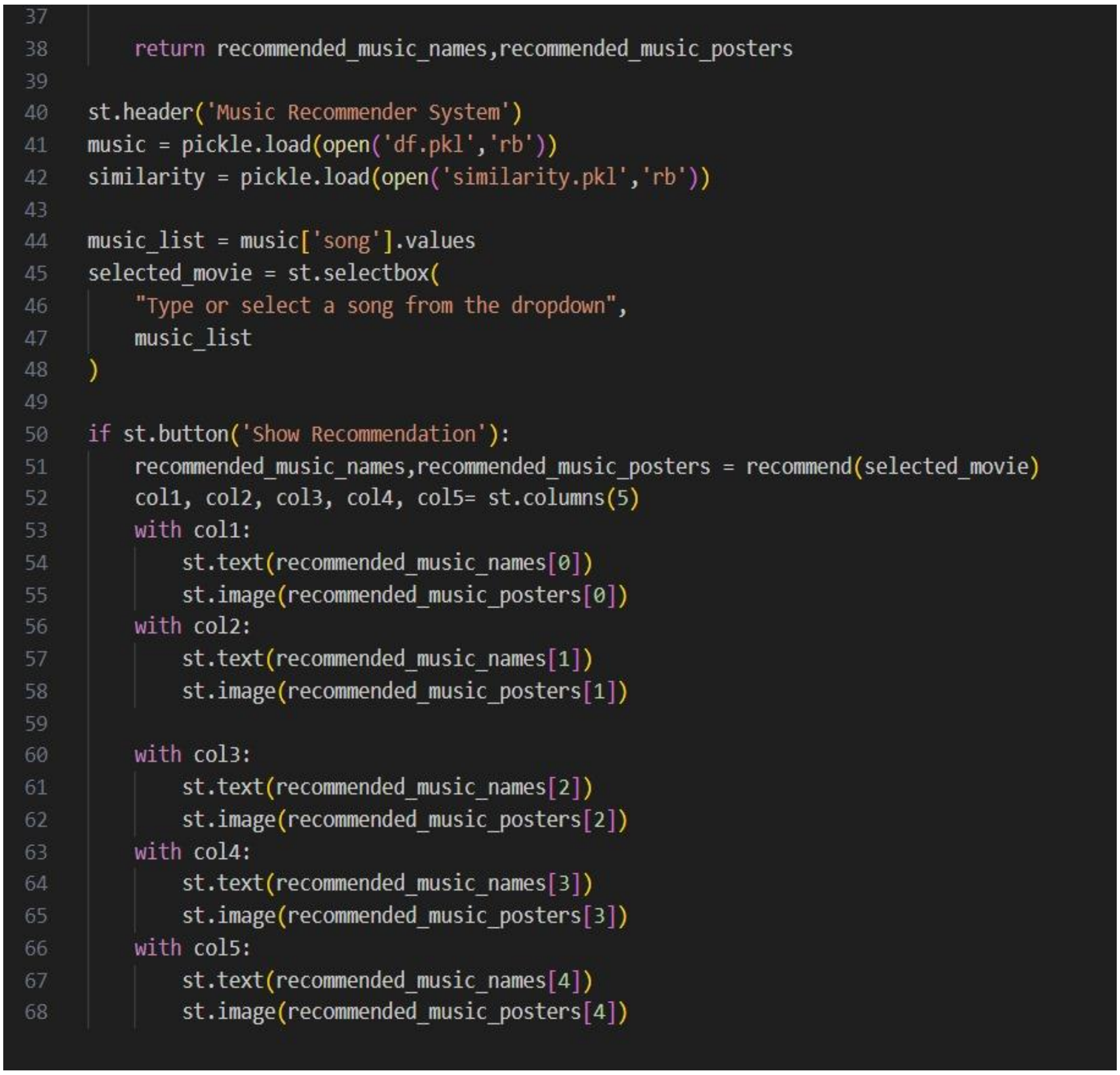
```
  File "/Users/jugendharia/Developer/Nikant/Music_Recommender_System/path/to/venv/lib/python3.9/site-packages/spotipy/oauth2.py", line 269, in _request_access_token
    self._handle_oauth_error(http_error)
  File "/Users/jugendharia/Developer/Nikant/Music_Recommender_System/path/to/venv/lib/python3.9/site-packages/spotipy/oauth2.py", line 146, in _handle_oauth_error
    raise SpotifyOauthError(
spotipy.oauth2.SpotifyOauthError: error: invalid_client, error_description: Invalid client secret
Pat Benatar
Rescue Me
Oingo Boingo
Happy
Dusty Springfield
I Don't Think We Could Ever Be Friends
https://i.scdn.co/image/ab67616d0000b273ef7289c598a224331602b19a
Lenny Kravitz
Happy Birthday
https://i.scdn.co/image/ab67616d0000b2732a8b178829064ca0dbb2bc1c
Scorpions
Bad For Good
```

Code:



File Edit Selection View Go Run Terminal Help
Search
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
app.py
C: > Users > nikan > Desktop > nik > nik > app.py

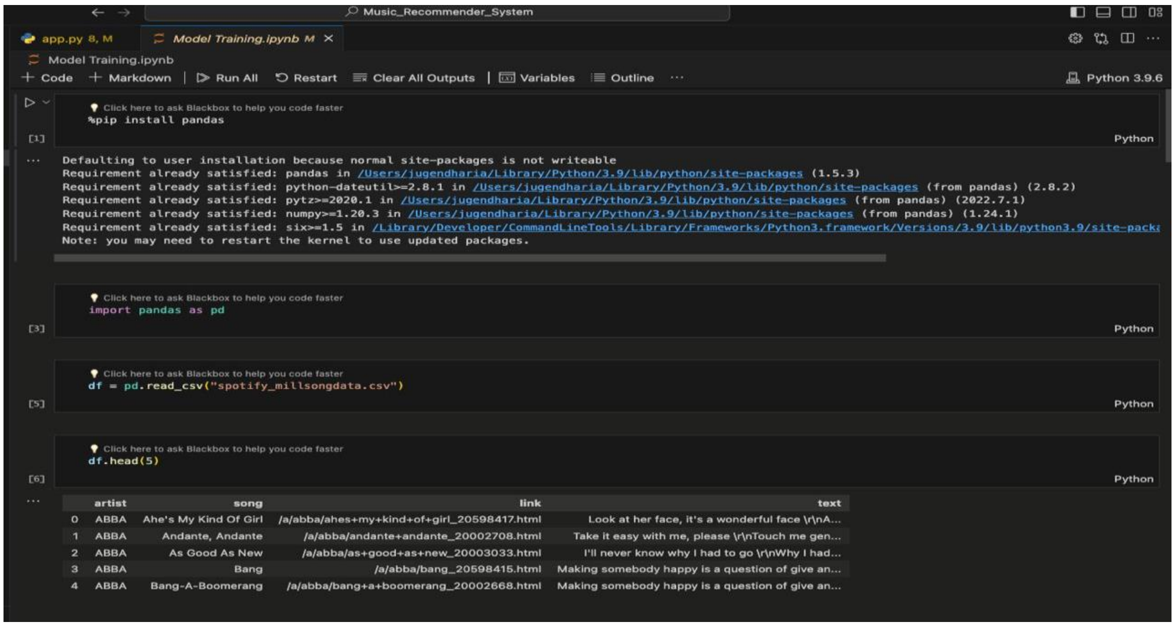
```python
import pickle
import streamlit as st
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials

CLIENT_ID = "ed643ee81be342b380744fab00b997e2"
CLIENT_SECRET = "e8d331e0a9e5439c9f621634a548b9da"

# Initialize the Spotify client
client_credentials_manager = SpotifyClientCredentials(client_id=CLIENT_ID, client_secret=CLIENT_SECRET)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)

def get_song_album_cover_url(song_name, artist_name):
    search_query = "track:{} artist:{}".format(song_name, artist_name)
    results = sp.search(q=search_query, type="track")

    if results and results["tracks"]["items"]:
        track = results["tracks"]["items"][0]
        album_cover_url = track["album"]["images"][0]["url"]
        print(album_cover_url)
        return album_cover_url
    else:
        return "https://i.postimg.cc/0QNxYz4V/social.png"

def recommend(song):
    index = music[music['song'] == song].index[0]
    distances = sorted(list(enumerate(similarity[index])), reverse=True, key=lambda x: x[1])
    recommended_music_names = []
    recommended_music_posters = []
    for i in distances[1:6]:
        # fetch the movie poster
        artist = music.iloc[i[0]].artist
        print(artist)
        print(music.iloc[i[0]].song)
        recommended_music_posters.append(get_song_album_cover_url(music.iloc[i[0]].song, artist))
        recommended_music_names.append(music.iloc[i[0]].song)

    return recommended_music_names,recommended_music_posters

```



```

    return recommended_music_names,recommended_music_posters

st.header('Music Recommender System')
music = pickle.load(open('df.pkl','rb'))
similarity = pickle.load(open('similarity.pkl','rb'))

music_list = music['song'].values
selected_movie = st.selectbox(
    "Type or select a song from the dropdown",
    music_list
)

if st.button('Show Recommendation'):
    recommended_music_names,recommended_music_posters = recommend(selected_movie)
    col1, col2, col3, col4, col5= st.columns(5)
    with col1:
        st.text(recommended_music_names[0])
        st.image(recommended_music_posters[0])
    with col2:
        st.text(recommended_music_names[1])
        st.image(recommended_music_posters[1])

    with col3:
        st.text(recommended_music_names[2])
        st.image(recommended_music_posters[2])
    with col4:
        st.text(recommended_music_names[3])
        st.image(recommended_music_posters[3])
    with col5:
        st.text(recommended_music_names[4])
        st.image(recommended_music_posters[4])
```

Data Structures and algorithms used:

The **Pandas DataFrame** serves as the primary data structure for storing the song dataset in the music recommendation system. With its tabular format and labelled axes, DataFrames in Pandas provide an efficient means of organizing and manipulating the dataset. This structure facilitates easy management of essential columns such as artist, song title, and lyrics, allowing for seamless data handling and analysis within the system.



```
%pip install pandas
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: pandas in /Users/jugendharia/Library/Python/3.9/lib/python/site-packages (1.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /Users/jugendharia/Library/Python/3.9/lib/python/site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/jugendharia/Library/Python/3.9/lib/python/site-packages (from pandas) (2022.7.1)
Requirement already satisfied: numpy>=1.20.3 in /Users/jugendharia/Library/Python/3.9/lib/python/site-packages (from pandas) (1.24.1)
Requirement already satisfied: six>=1.5 in /Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/lib/python3.9/site-packa
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
```

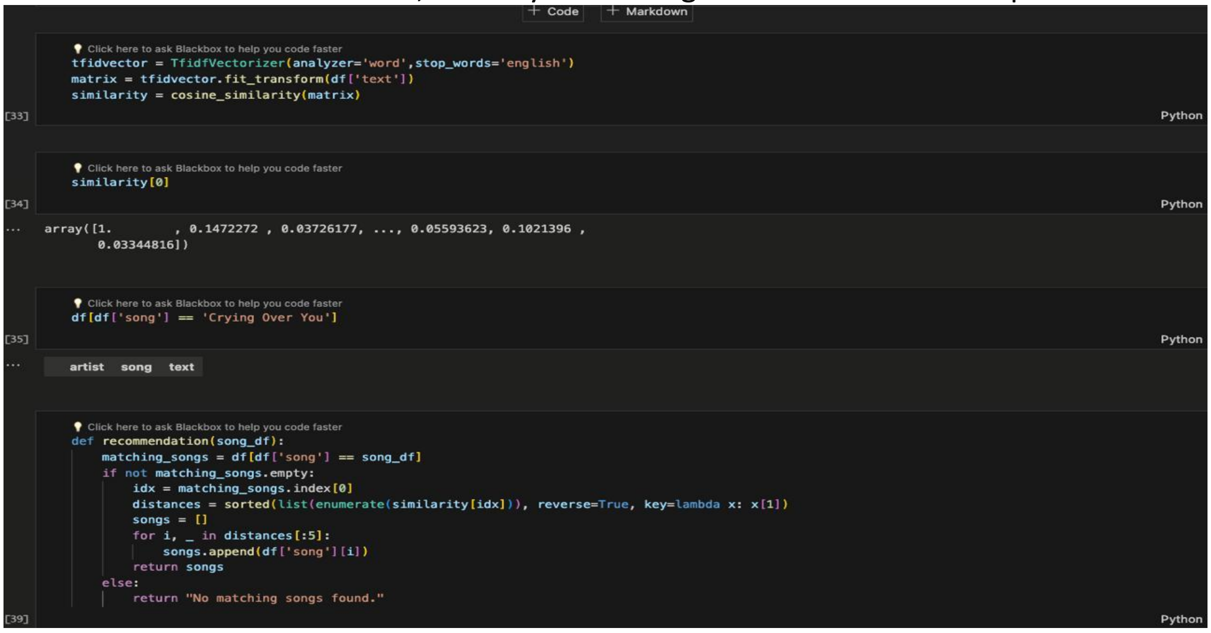
```
df = pd.read_csv("spotify_millsongdata.csv")
```

```
df.head(5)
```

	artist	song	link	text
0	ABBA	Ahe's My Kind Of Girl	/a/abba/ahes+my+kind+of+girl_20598417.html	Look at her face, it's a wonderful face \r\nA...
1	ABBA	Andante, Andante	/a/abba/andante+andante_20002708.html	Take it easy with me, please \r\nTouch me gen...
2	ABBA	As Good As New	/a/abba/as+good+as+new_20003033.html	I'll never know why I had to go \r\nWhy I had...
3	ABBA	Bang	/a/abba/bang_20598415.html	Making somebody happy is a question of give an...
4	ABBA	Bang-A-Boomerang	/a/abba/bang+a+boomerang_20002668.html	Making somebody happy is a question of give an...

Algorithms:

TF-IDF Vectorizer serves as a crucial tool for converting textual data, such as song lyrics, into numerical features essential for machine learning models' comprehension. This method captures the significance of words within the context of individual songs and across the entire dataset. By assigning weights to words based on their frequency and importance, TF-IDF vectorization enables the model to discern songs with similar thematic content, thereby enhancing the recommendation process.



```
tfidvector = TfidfVectorizer(analyzer='word',stop_words='english')
matrix = tfidvector.fit_transform(df['text'])
similarity = cosine_similarity(matrix)
```

```
similarity[0]
```

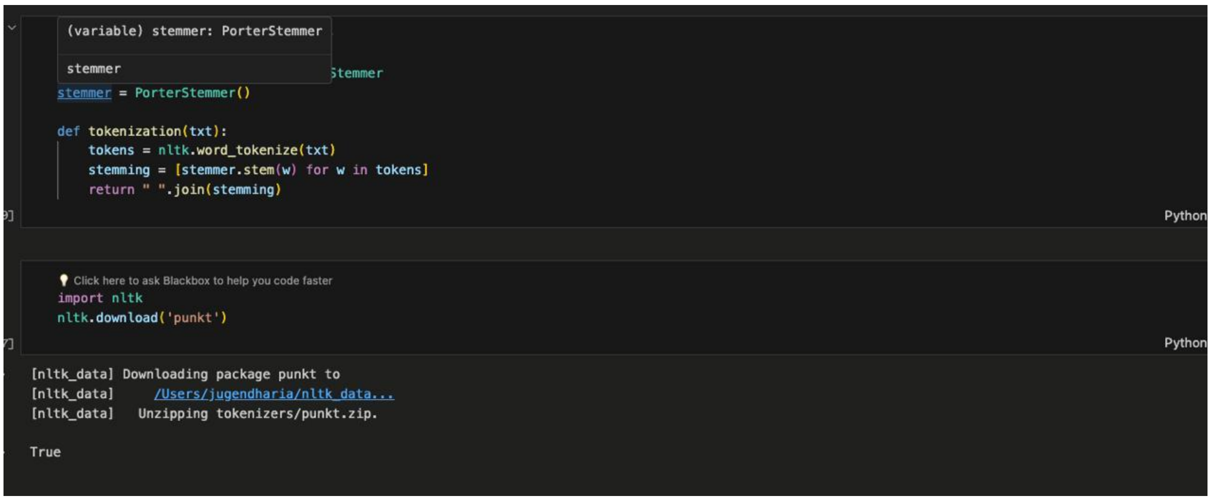
```
array([1.        , 0.1472272 , 0.03726177, ..., 0.05593623, 0.1021396 ,
       0.03344816])
```

```
df[df['song'] == 'Crying Over You']
```

```
artist  song  text
```

```
def recommendation(song_df):
    matching_songs = df[df['song'] == song_df]
    if not matching_songs.empty:
        idx = matching_songs.index[0]
        distances = sorted(list(enumerate(similarity[idx])), reverse=True, key=lambda x: x[1])
        songs = []
        for i, _ in distances[:5]:
            songs.append(df['song'][i])
        return songs
    else:
        return "No matching songs found."
```

Cosine Similarity emerges as a natural choice for comparing TF-IDF vectors, representing the semantic meaning of song lyrics. This metric effectively measures the similarity between two songs based on their lyrical content. By calculating the cosine of the angle between two song vectors, cosine similarity enables the recommender system to identify songs sharing similar themes, thus facilitating accurate and personalized recommendations to users.



```
(variable) stemmer: PorterStemmer

stemmer
stemmer = PorterStemmer()

def tokenization(txt):
    tokens = nltk.word_tokenize(txt)
    stemming = [stemmer.stem(w) for w in tokens]
    return " ".join(stemming)
```

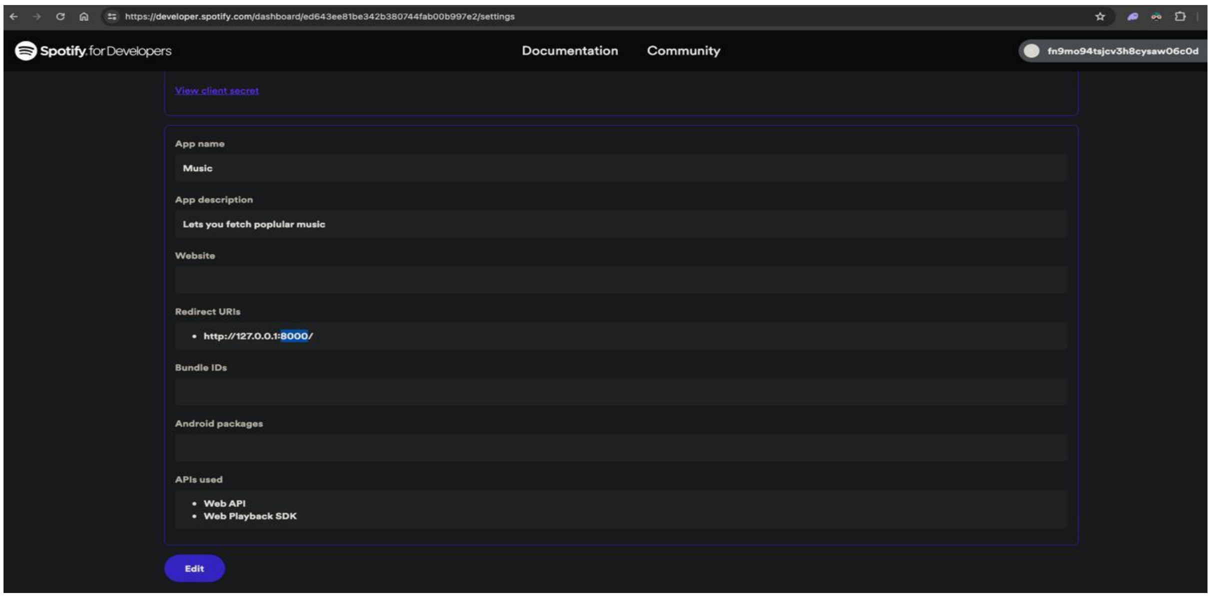
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/jugendharia/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.

True
```

API'S used in the system:

Spotify API:

The Spotify API grants access to its extensive music library, user playlists, and audio attributes. Incorporating the Spotify API enables the recommendation system to utilize user listening habits, preferences, and collaborative filtering techniques to deliver customized suggestions. Moreover, the API facilitates smooth playback of song previews directly within the application.



https://developer.spotify.com/dashboard/ed643ee81be342b380744fab00b997e2/settings
Spotify for Developers
Documentation
Community
fn9mo94tsjcv3h8cysaw06c0d
View client secret
Edit

App name
Music

App description
Lets you fetch poplular music

Website

Redirect URIs
- http://127.0.0.1:8000/

Bundle IDs

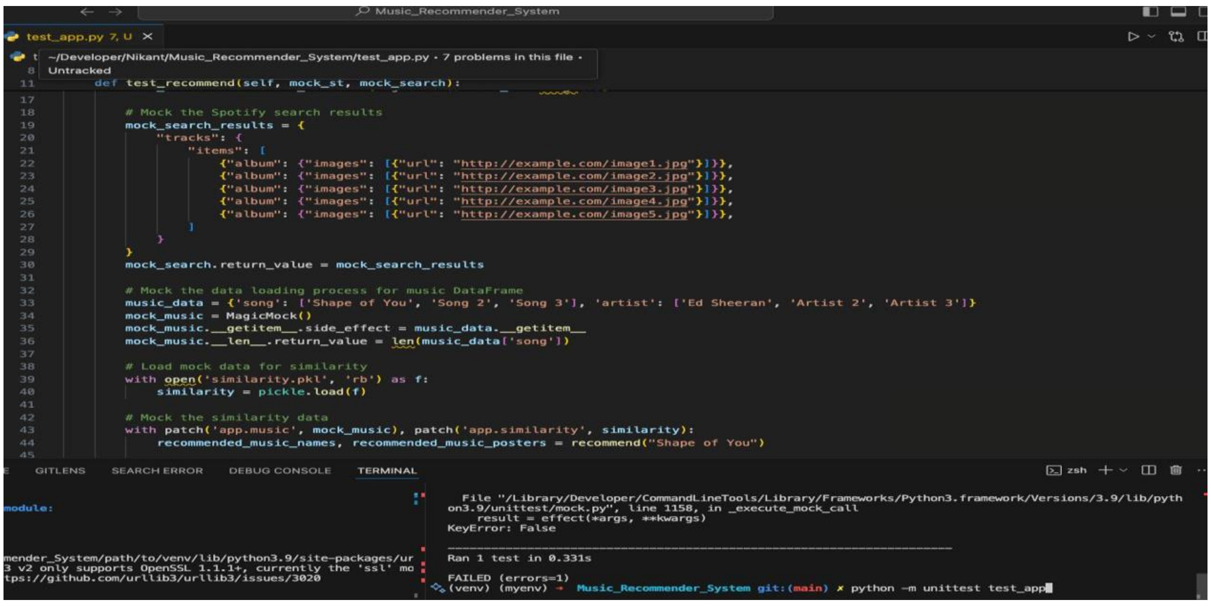
Android packages

APIs used
- Web API
- Web Playback SDK

:

Unit Testing within terminal:

Unit test, if the data doesn't match it will give an error I.e. we can use the data and compare it with the Spotify data if there is mismatching the data it will give an error if no output is there means the function has failed also write.



```python
    def test_recommend(self, mock_st, mock_search):

        # Mock the Spotify search results
        mock_search_results = {
            "tracks": {
                "items": [
                    {"album": {"images": [{"url": "http://example.com/image1.jpg"}]}},
                    {"album": {"images": [{"url": "http://example.com/image2.jpg"}]}},
                    {"album": {"images": [{"url": "http://example.com/image3.jpg"}]}},
                    {"album": {"images": [{"url": "http://example.com/image4.jpg"}]}},
                    {"album": {"images": [{"url": "http://example.com/image5.jpg"}]}},
                ]
            }
        }
        mock_search.return_value = mock_search_results

        # Mock the data loading process for music DataFrame
        music_data = {'song': ['Shape of You', 'Song 2', 'Song 3'], 'artist': ['Ed Sheeran', 'Artist 2', 'Artist 3']}
        mock_music = MagicMock()
        mock_music.__getitem__.side_effect = music_data.__getitem__
        mock_music.__len__.return_value = len(music_data['song'])

        # Load mock data for similarity
        with open('similarity.pkl', 'rb') as f:
            similarity = pickle.load(f)

        # Mock the similarity data
        with patch('app.music', mock_music), patch('app.similarity', similarity):
            recommended_music_names, recommended_music_posters = recommend("Shape of You")
```

```
module:

mender_System/path/to/venv/lib/python3.9/site-packages/ur
3 v2 only supports OpenSSL 1.1.1+, currently the 'ssl' mo
tps://github.com/urllib3/urllib3/issues/3020
```

```
  File "/Library/Developer/CommandLineTools/Library/Frameworks/Python3.framework/Versions/3.9/lib/pyth
on3.9/unittest/mock.py", line 1158, in _execute_mock_call
    result = effect(*args, **kwargs)
KeyError: False

----------------------------------------------------------------------
Ran 1 test in 0.331s

FAILED (errors=1)
(venv) (myenv) → Music_Recommender_System git:(main) ✗ python -m unittest test_app
```

References:

Kaggle dataset: Click Here for dataset
Streamlit: Reference link for setup & installation
Pandas DataFrame: Click here for the link
TF-IDF vectorization: Click here for the link
Spotify Library: Click Here for the link

THANK YOU!!