CHAPTER-1

INTRODUCTION

1.1 PROBLEM DEFINATION

On the Internet, where the number of choices is overwhelming, there is need to filter, prioritize and efficiently deliver relevant information in order to alleviate the problem of information overload, which has created a potential problem to many Internet users. Recommender systems solve this problem by searching through large volume of dynamically generated information to provide users with personalized content and services. This project explores the different characteristics and potentials of different prediction techniques in recommendation systems in order to serve as a compass for research and practice in the field of recommendation systems.

Recommendation Definition:

Let **C** be set of all users and let **S** be set of all possible recommendable items. Let **u** be a utility function measuring the usefulness of item **s** to user **c**, i.e., **u: C X S→R**, where **R** is a totally ordered set. For each user **c**ϵ**C**, we want to choose items **s**ϵ**S** that maximize **u**.

The main task is to estimate a utility function (**u**) that automatically predicts how a user will like an item. Based on Past behavior, Relations to other users, Item similarity, Context etc.

1.2 PROJECT OVERVIEW

With the rise of digital content distribution, people now have access to music collections on an unprecedented scale. Commercial music libraries easily exceed 15 million songs, which vastly exceeds the listening capability of any single person. With millions of songs to choose from, people sometimes feel overwhelmed. Thus, an efficient music recommender system is necessary in the interest of both music service providers and customers. Users will have no more pain to make decisions on what to listen while music companies can maintain their user group and attract new users by improving user's satisfaction.

Objectives

- The main objective of this project is to provide personalized recommendation.
- To increase the level of satisfaction among the users.
- To understand the user's tastes.
- To allow users to Play, Favorite, Download the songs.
- To allow users to create playlist.
- To helps the users to search the songs in effective and responsive manner.

1.3 HARDWARE REQUIREMENT:

Processor: i5

RAM: 4.00 GB

System Type: 64-bit Operating System

Hard Disk: 10GB HDD

1.4 SOFTWARE REQUIREMENT:

Operating System: Windows 10

Developing Languages: Python, JavaScript, HTML, AJAX, Jquery

RDBMS: SQLite

Framework: Django

Modules: Sklearn, Pandas, Numpy, Matplotlib

CHAPTER -2

LITERATURE SURVEY

2.1 EXISTING SYSTEM:

The explosive growth in the amount of available digital information and the number of visitors to the Internet have created a potential challenge of information overload which hinders timely access to items of interest on the Internet. Information retrieval systems, such as Google, DevilFinder and Altavista have partially solved this problem but prioritization and personalization (where a system maps available content to user's interests and preferences) of information were absent.

This has increased the demand for recommender systems more than ever before. Recommender systems are beneficial to both service providers and users. They reduce costs of finding and selecting items in an Online Music Portal. Recommendation systems have also proved to improve decision making process and quality. In Online Music Portal, recommender systems enhance revenues, for the fact that they are effective means of selling more products. In scientific libraries, recommender systems support users by allowing them to move beyond catalog searches. Therefore, the need to use efficient and accurate recommendation techniques within a system that will provide relevant and dependable recommendations for users cannot be over-emphasized.

2.2 PROPOSED SYSTEM:

Music Recommendation System uses Popularity Based, Collaborative Filtering Based and Content Based Recommender to find correlations between users and songs. Each of these recommender have certain advantages and disadvantages. This project is try to make a unified recommender which is combination of all these recommender to provide better recommendations.

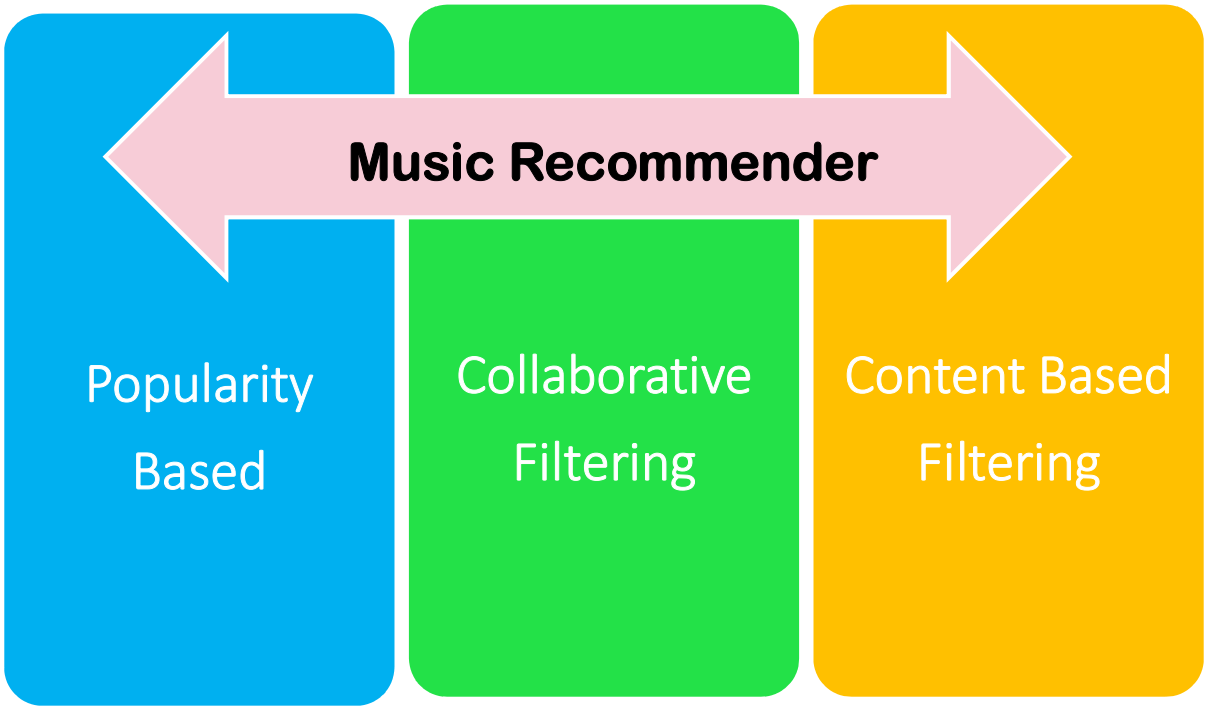

Figure. 2.1

Project approach

We propose following algorithms for our task:

1. **Popularity Based :**

   The most trivial recommendation algorithm is to simply present each song in descending order of its popularity skipping those songs already consumed by the user, regardless of the user's taste profile.

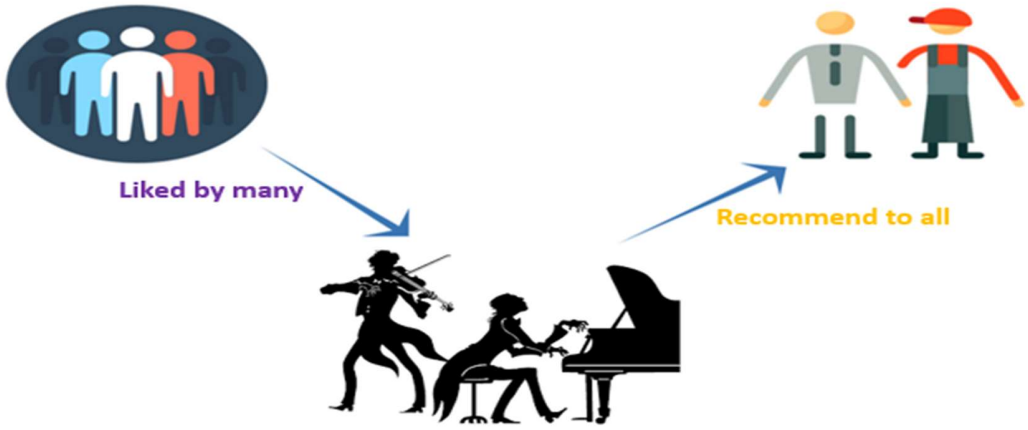

Figure. 2.2

2. **Content Based Filtering:**

   Use features of both products as well as users in order to predict whether a user will like a product or not.

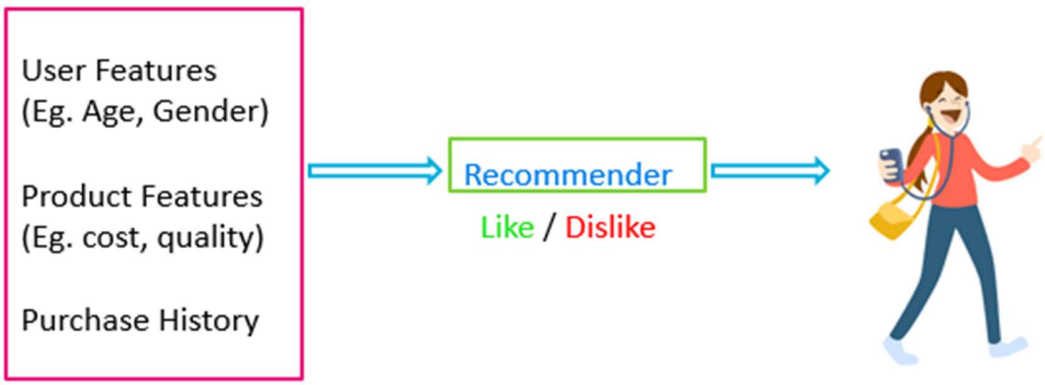

Figure. 2.3

3. **Collaborative Filtering :**

It can be either user-based or item-based. In user-based recommendation, users who listen to the same songs in the past tend to have similar interests and will probably listen to the same songs in future. In the item-based recommendation strategy, songs that are often listened by the same user tend to be similar and are more likely to be listened together in future by some other user.

User-Based Collaborative Filtering

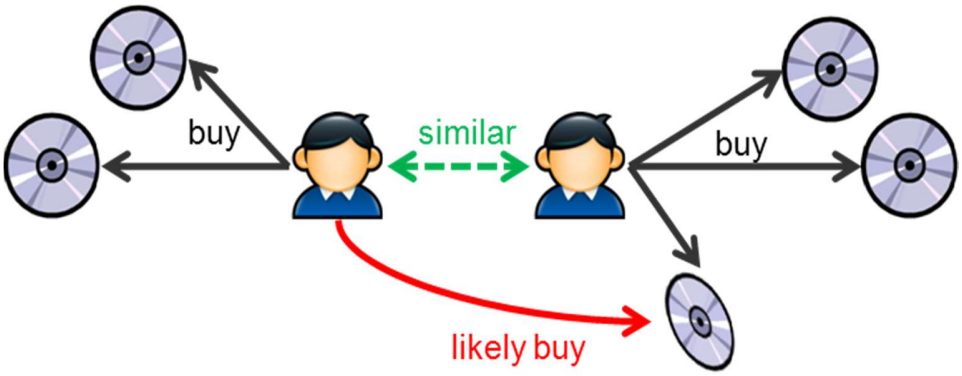

Figure. 2.4

Item-Based Collaborative Filtering

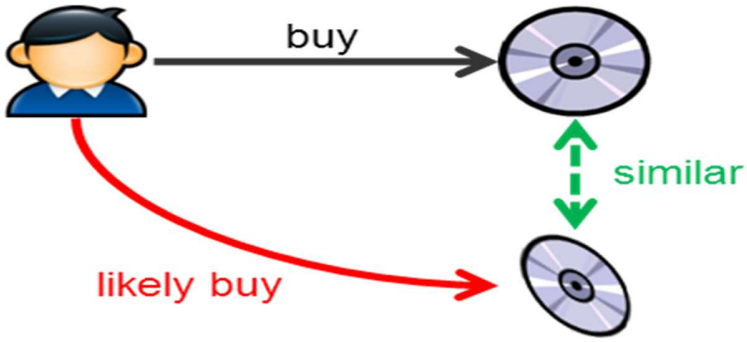

Figure. 2.5

2.3 FEASIBILITY STUDY:

Introduction:

The feasibility study of any system is mainly intended to study and analyze the proposed system and to decide whether the system under consideration will be viable or not after implementation. That it determines the usability of the project after deployment. To come to result a set of query is answered keeping the efficiency of the software and its impact on the domain for which it was developed.

Its main emphasis is on the following three questions elucidated below as:

- **What are the user's requirements and how does a candidate system meet them?**
- **What resources are available for the proposed systems? Is it worth solving the problem?**
- **What is the likely impact of the proposed system on the organization? i.e. how does the proposed system fit in the organization?**

Thus since the feasibility study may lead to commitment of large resources, it becomes necessary that it should be conducted competently and no fundamental errors of judgment are made.

Technical Feasibility:

In technical feasibility, we study all technical issues regarding the proposed system. It is mainly concerned with the specifications of the equipment and the software, which successfully satisfies and the end user's requirement.

The technical needs of the system may vary accordingly but include:

- **The feasibility to produce outputs in a given time.**
- **Response time under certain conditions.**
- **Ability to process a certain volume of transaction at a particular speed.**
- **Facility to communicate data.**

Under this analysis process questions like:

i. **Does the compatible platform exist within our domain or can we procure it?**
ii. **Does the proposed equipment have the technical capacity to hold the data required using the new system? Both at the development site and at server where we will be hiring the space for the website, and also the database.**
iii. **Would it be possible to upgrade the system after it is developed and implemented, if necessary?**
iv. **Would the recommended technology guarantee the reliability, accuracy and data security?**

This analysis process requires more emphasis on system configuration or system configuration given more importance rather than actual hardware specifications.

Tangibility refers to the ease with which costs and benefits can be measured. An outlay of cash for a specific item or activity is referred to as a tangible cost. They are shown as disbursements on the books. The purchases of the hardware or software, personnel training are examples of tangible costs. They are readily identified and measured.

Direct or Indirect Costs & Benefits:

Direct Costs are those with which a monetary figure can be directly associated in a project. They are applied directly to the operation. Direct benefits can also be specified to the given project.

Indirect Costs are the results of operation that are not directly associated with the given system or activity. They are often referred to as overhead. Light, air conditioning, maintenance, protection of the computer center is all tangible costs, but it is difficult to determine the proportion of each attribute to a specific activity. Indirect benefits are realized as a by-product of another activity or system.

Fixed and Variable Costs and Benefits:

Fixed Costs: Some costs and benefits are constant, regardless of how well a system is used. They are constant and do not change. Once encountered they will not reappear.

In contrast, **Variable Costs** are incurred on a regular basis. They are proportion to work volume and continue as long as is in operation. Fixed Benefits are also constant and do not change. The variable benefits on the other hand, are realized on a regular basis. For example, the system will save several hours of employees in preparing each report, which otherwise was prepared manually. Another examples are error free data entry, help about different topics and ease of use.

Operational Feasibility:

Proposed projects are beneficial only if they can be turned into information system that will meet the operating requirements for which they are designed and developed. It is mainly related to human organizational and political aspects. People are resistant to change and computers have been known to facilitate change.

Findings of operational feasibility analysis can be summarized as given below:

- **Since the audiences are millions of citizen who use it for information retrieval. So there will be many who will be benefited with this project.**
- **End users are always welcome to have the technical support on the site.**

CHAPTER – 3

SYSTEM DESIGN & ANALYSIS

3.1 REQUIREMENT SPECIFICATION:

The *software requirement specification*(SRS) is very important part of the software building process, which describes the actual user level requirement from technical point of view i.e., what the user actually wants? or for what purpose we are making everything? The objective of preparing the software requirement specification is to represent the requirements of the software in such a manner that ultimately leads to successful software implementation. It is the result of the analysis process of the software development. It should contain all the data the software is going to process, the function it will provide, and the behavior it will exhibit. The software requirements specifications (SRS) is defined in IEEE Std. 830-1993, IEEE Recommended Practice for Software Requirements Specifications. The document is organized in the following structure:

- Introduction
- Information Description
- Functional Description
- Behavior Description
- Validation Criteria
- Bibliography
- Appendix

a) Introduction: The introduction section describes the goals and objective of the software undergoing development in context of computer based system. It mainly deals with the software scope i.e. will bind the software application domain.

b) Information Description: This section of the SRS provides a detail of the problem that the software must solve. It should describe the core of the software- i.e. *The Data.* The

data or information the software is going to work on is the most basic part of the software. The description of each data or information entity is described here. It also give details of the relationships between the data elements of the software. The information description helps the software designers in their designing purpose.

c) Functional Description: This section of the SRS describes the each functions required to solve the problem. It emphasizes on the core of the software on which the data will be processed –i.e. *The Function*. It also includes the process specification of each function, design constraints, and performance characteristics. The DFD or any other graphical diagram can also be added to describe the functionality of the system.

d) Bibliography: This section contains references to all the related documents related with the software. This may include any technical document, standards document or software engineering paper.

e) Appendix: This section is supplementary and can include the matters that are important for the software development. It may include the statistical data, graphs or algorithm details.

Preliminary Investigation

Once the request for the information system development is received, the first system activity "Preliminary Investigation" begins. This activity is independent of the software-engineering paradigm. That means whether a system will be developed by means of the system development life cycle (SDLC), a prototyping strategy, or the structured analysis method, or a combination of these methods, preliminary investigation must be performed first.

This activity has three parts:

1. Request Clarification
2. Study
3. Approval.

It may be possible that the request that has been received is not clearly defined and it is difficult to figure out what for the request is. So, in that case, the person who has made the request is contacted and asked to explain the request. If the request is appropriate enough then in that case feasibility study is carried out for that request, the request must be defined and determined precisely. After request clarification is feasibility study is

performed. If the request for the information system is feasible and the request is approved, then starts the real analysis phase called "Requirement study". The data that are collected from the organization as the part of preliminary investigation is follows three mechanisms also called "**Fact Finding Techniques**":

- Reviewing existing Documents material and existing systems (web sites)
- Conducting Interviews and
- On-site Observation

To conduct preliminary investigation, we came to know that CRC is having their websites with the organization information purposes. The very first impression validates the request for the automation. We used all the three fact-finding techniques for data collection. We collected various papers and documents related to the activities, we analyzed every aspect of the online customer care system, and conducted marketing surveys to know the interest among the people associated and attached with different organization. It was surprising to know the most of them were very interested in computers and has the better understanding of computer & information systems. Both were very helpful in passing the exact information we were looking for. Reviewing the traditional one and having own experience in the field helped us immensely in the knowing the data items being processed by the system.

Information Content:

It represents the individual data and control objects that constitute some larger collection of information transformed by the software. In our case the object, which is composite of a number of important pieces of data as project, package and item with its category. So the contents are defined by the necessary attributes required to create it. During the analysis of the information domain the inter relationship between objects is also defined.

Information Flow:

It represents the manner in which data and control changes as each move through a system. Input objects are transformed into intermediate information, which is further transformed to output. Here, additional information is also supplied and transformations are some engineering functions/formulae.

System under Study:

In order to understand the need of the proposed system first we will have to understand the demand and supply of such system accordingly in the real world, ad in nowadays context.

With more than 1 billion population of the country like ours, where everything is changing fast and figures complete change in the mod of human interaction changing the direct dependability between man & machine through the fast growing electronics mode of communication opening a new door for internet technology to act as a interactive media for a professional interaction, this system deals directly with customer interaction with the respective companies for the products and service regarding to that products. The interaction response time can be decreased through a viable internet based response, request and query management system.

With the increasing of wed broadcast network using optical fiber cable the system being developed is feasible and cannot act as a trusted mode of communication between companies and their customers.

Engulfing the vast possibility of growth in the customer care industry at home or at a communication hub like cyber cafes. One cannot deny the possibility of good customer care industry through the net called online customer care system.

The manual system:

In the manual system the companies selling their products have to provide services through channels and mode of operations existing through different hierarchy level of problem solving stages and vice-versa to customer.

This is a tedious process and a person at higher level in the hierarchy is not having the complete control over the people who work at the lower level or the direct interaction between the customers and help desk officer. The system being proposed will only automate and facilitate the manual form as well as will provide blend.

3.2 SYSTEM FLOW CHART:

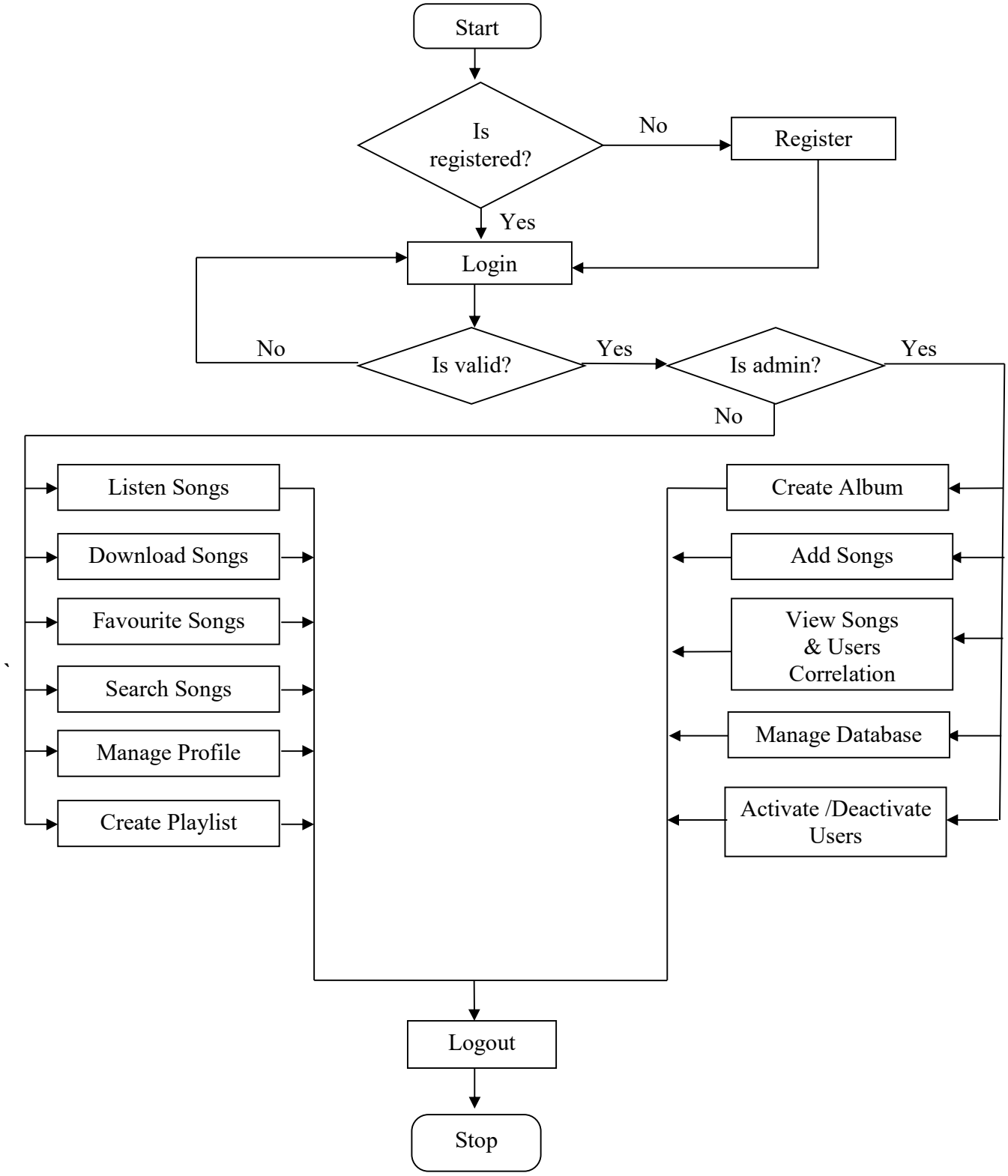

Figure. 3.1

3.3 USE CASE DIAGRAM:

A **user case diagram** shows you some of the use cases in your system, some of the actors in your system, and the relationships between them. A use case is a high- level piece of functionality that the system will provide. An actor is anyone or anything that interacts with the system being built. The purposes of use case diagrams can be as follows:

- Used to gather requirements of a system.
- Used to get an outside view of a system.
- Identify external and internal factors influencing the system.
- Show the interacting among the requirements are actors

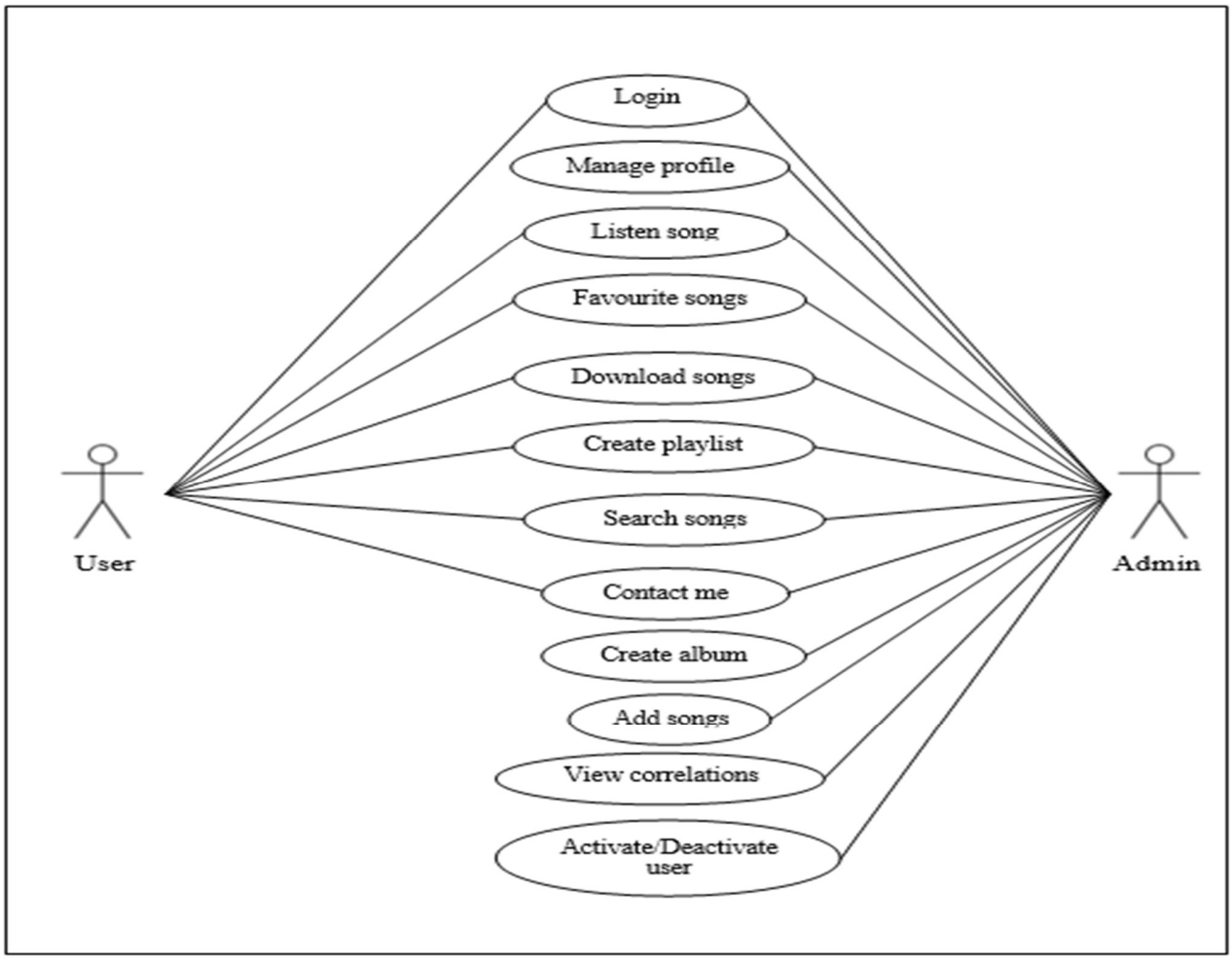

Figure. 3.2

3.4 DATA FLOW DIAGRAM:

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system, modeling its *process* aspects. Often they are a preliminary step used to create an overview of the system which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kinds of data will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of processes, or information about whether processes will operate in sequence or in parallel.

The data flow diagram is analogous to a road map . It is a network model of all posibilities with different detail shown on different hierarchial levels. This process of representing different level is called " levelling" or "partitionaning" by some data flow diagrams advocates. Details is not shown on different levels of the data flow diagram.Data flow diagram also known as "bubble chart" is used to clarify systems requirements. Following are the symbols of drawing a DFD:

- Ellipses representing processes which take data as input validate and process it and output it.
- Rectangle representing external entities, which are sources of destination of data.
- Arrows representing the data flows, which can be any type of data.

3.4.1 Zero Level DFD:

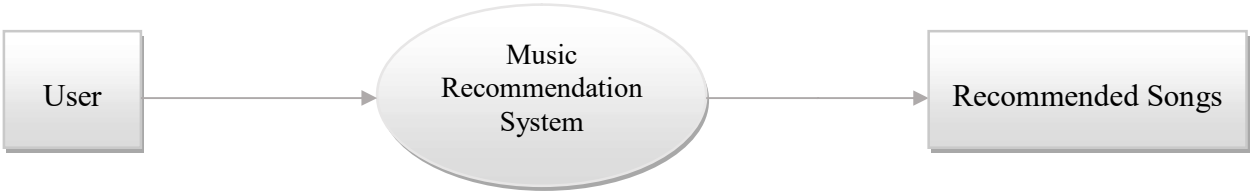

Figure. 3.3

3.4.2 First Level DFD:

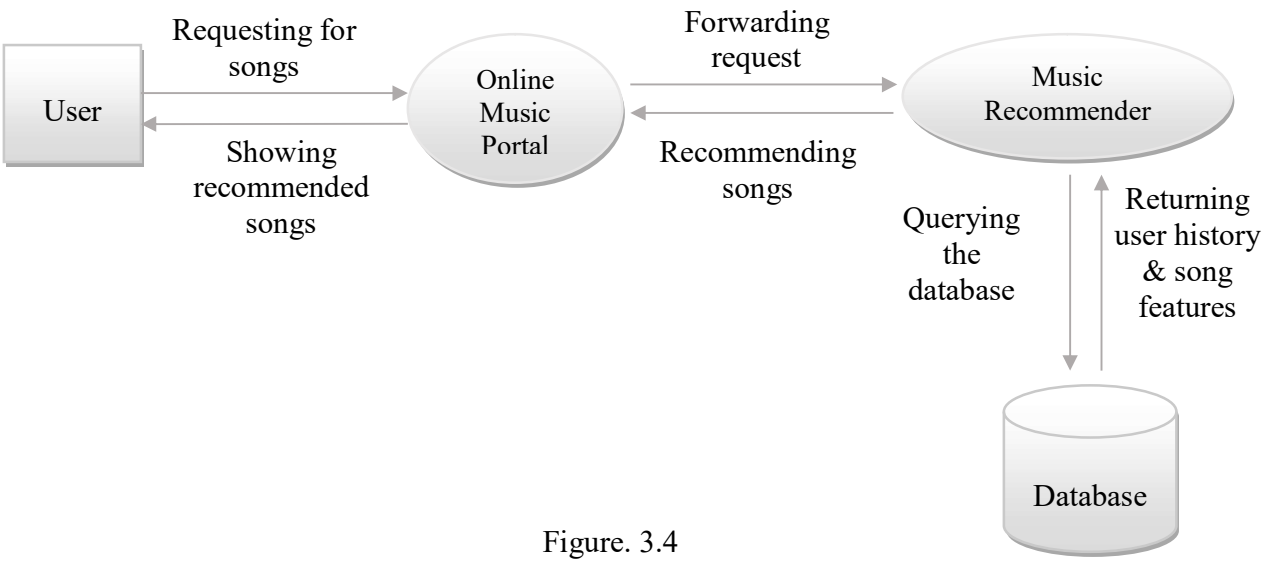

Figure. 3.4

3.4.3 Second Level DFD:

3.4.3.1 User Login

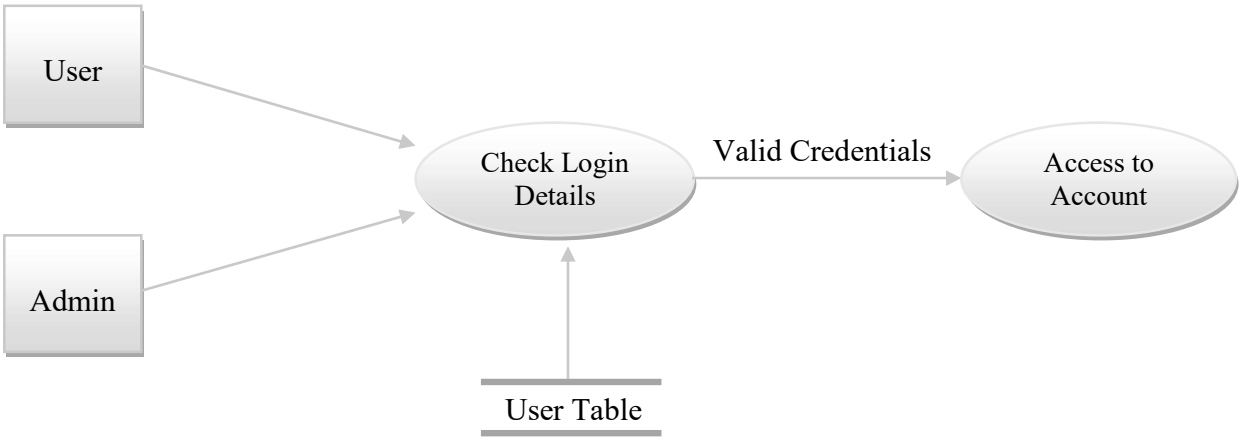

Figure. 3.5

3.4.3.2 Music recommender

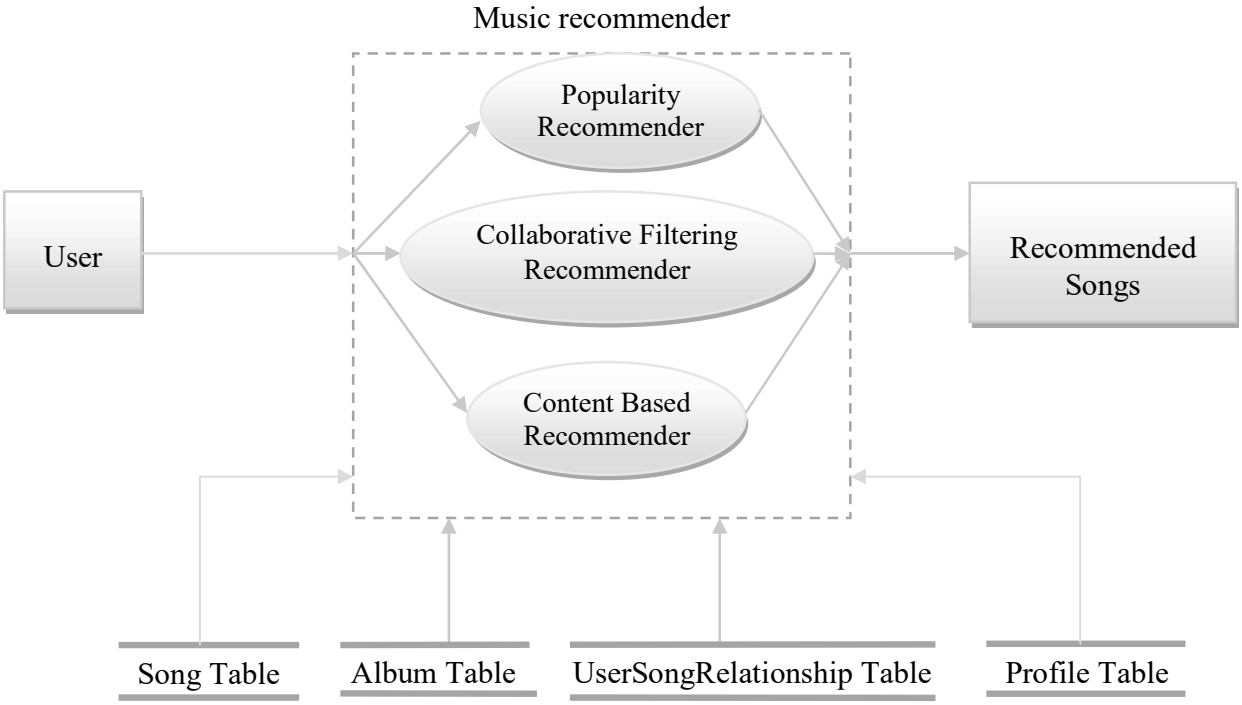

Figure. 3.6

3.5 ENTITY RELATIONSHIP DIAGRAM:

An **entity-relationship diagram** is a data modeling technique that creates a graphical representation of the entities, and the relationships between entities, within an information system. (View diagram.)

The three main components of an ERD are:

- The ***entity*** is a person, object, place or event for which data is collected.
- The ***relationship*** is the interaction between the entities.
- The ***cardinality*** defines the relationship between the entities in terms of numbers. The three main cardinal relationships are: one-to-one, expressed as 1 : 1; one-to-many, expressed as 1 : M; and many-to-many, expressed as M : M.

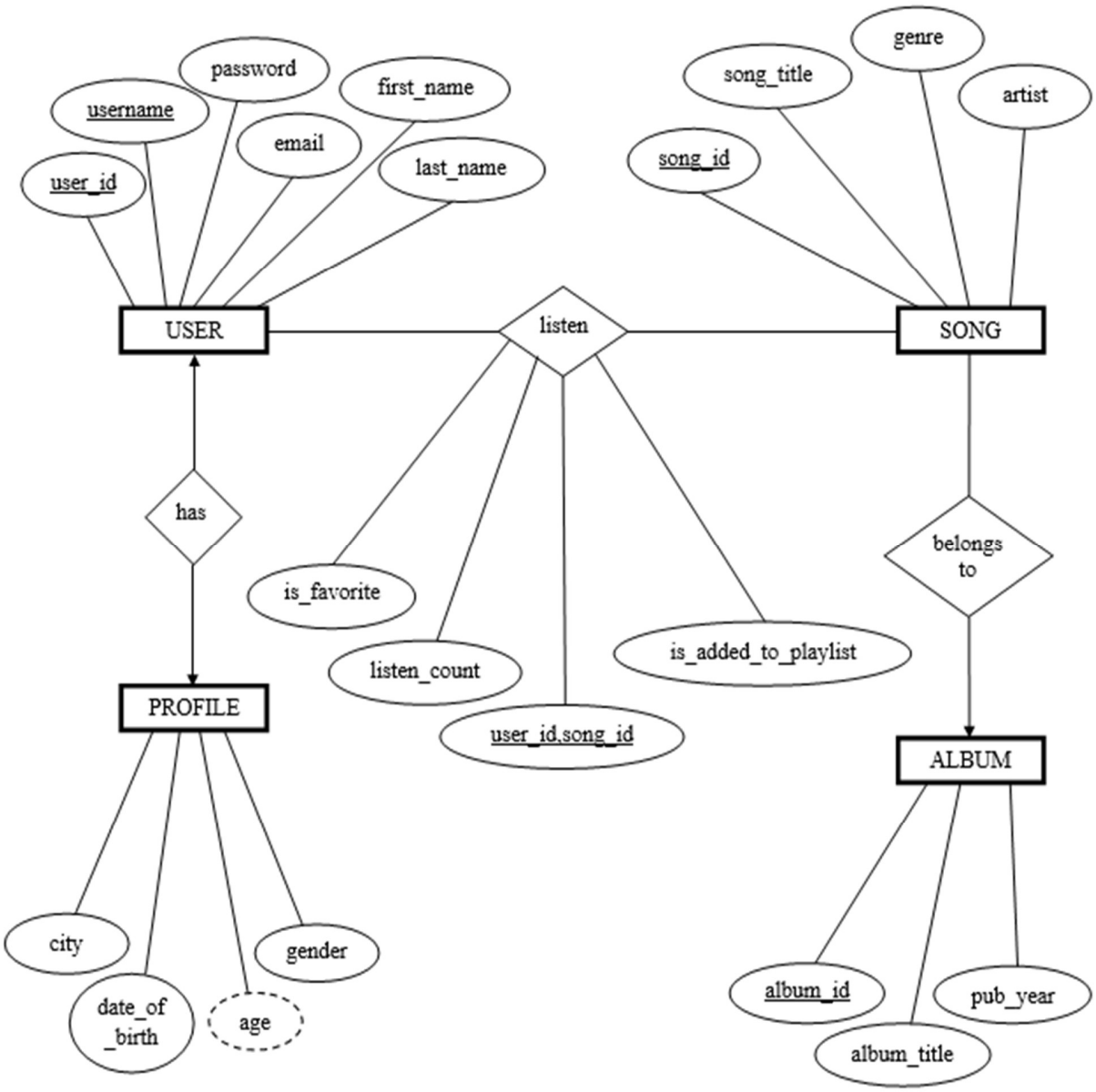
password
first_name
username
email
user_id
last_name
genre
song_title
artist
song_id
USER
listen
SONG
has
belongs
to
is_favorite
listen_count
is_added_to_playlist
user_id,song_id
PROFILE
ALBUM
city
gender
date_of
_birth
age
album_id
pub_year
album_title

Fig. 3.7

3.6 CLASS DIAGRAM:

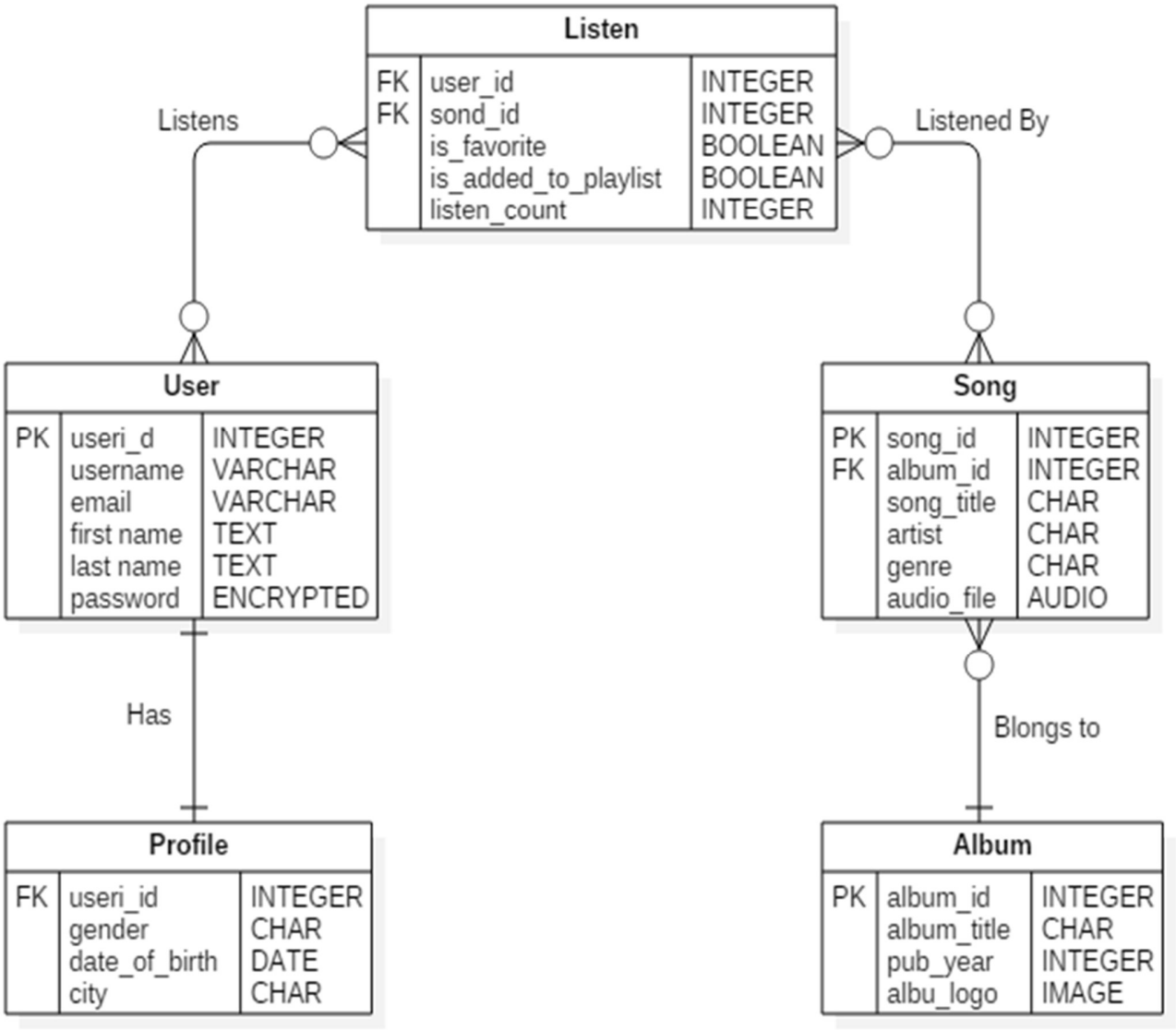

Figure. 3.8

3.7 DESIGN:

Homepage

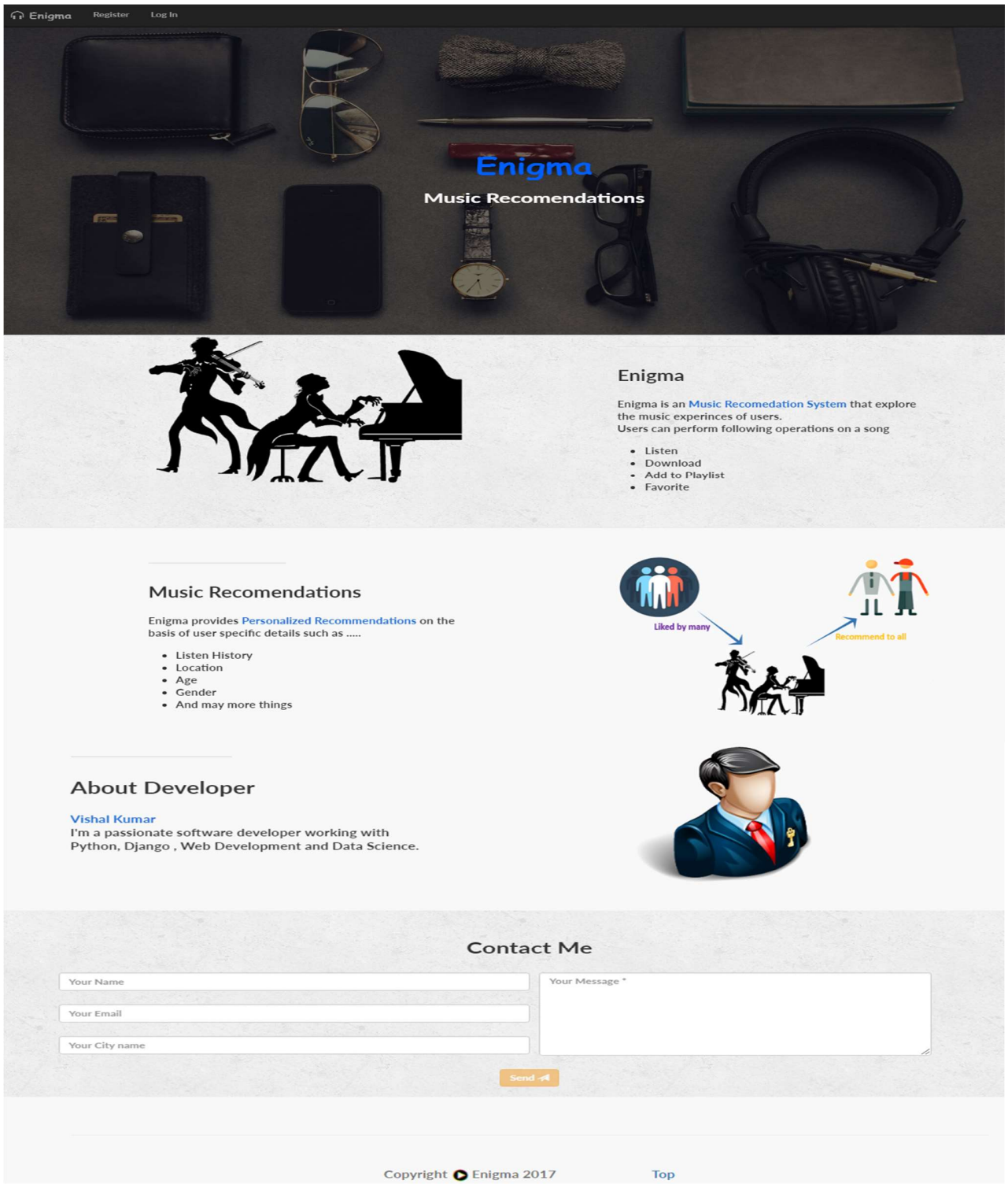

Figure. 3.9

User Registration

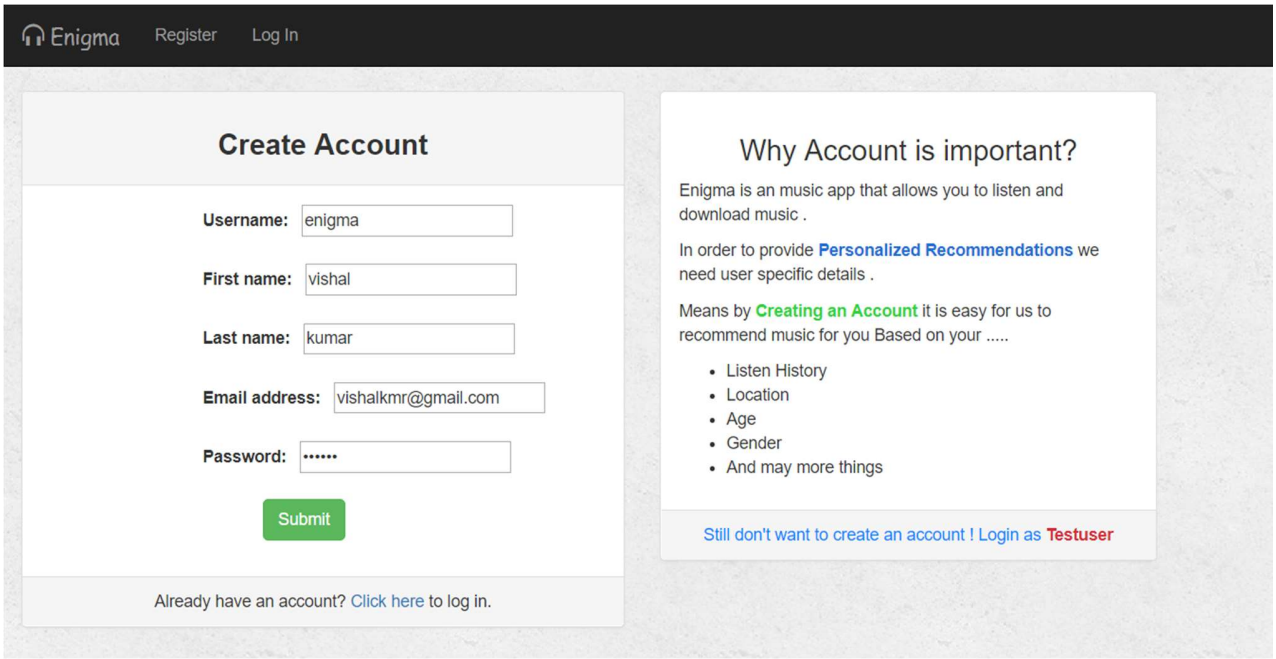

Figure. 3.10

User Login

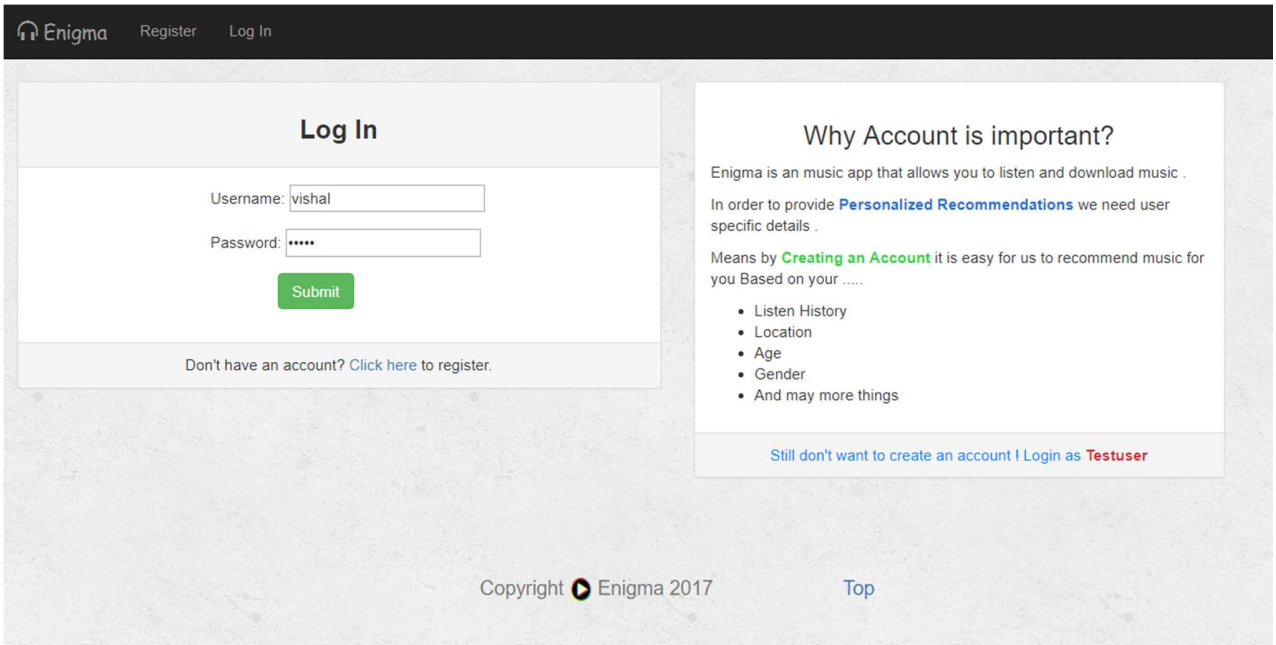

Fig. 3.11

Songs

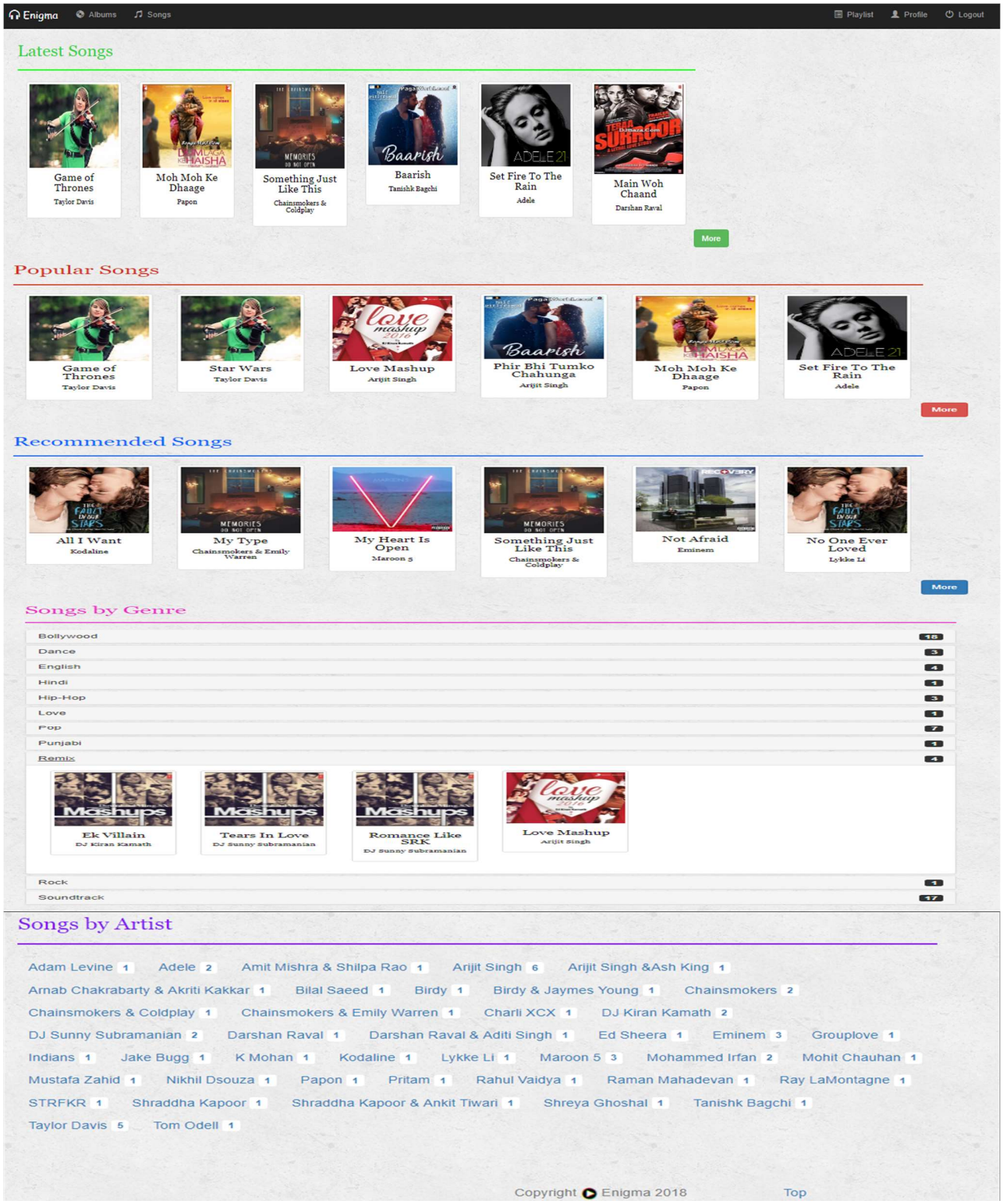

Figure. 3.12

Simmilar Songs

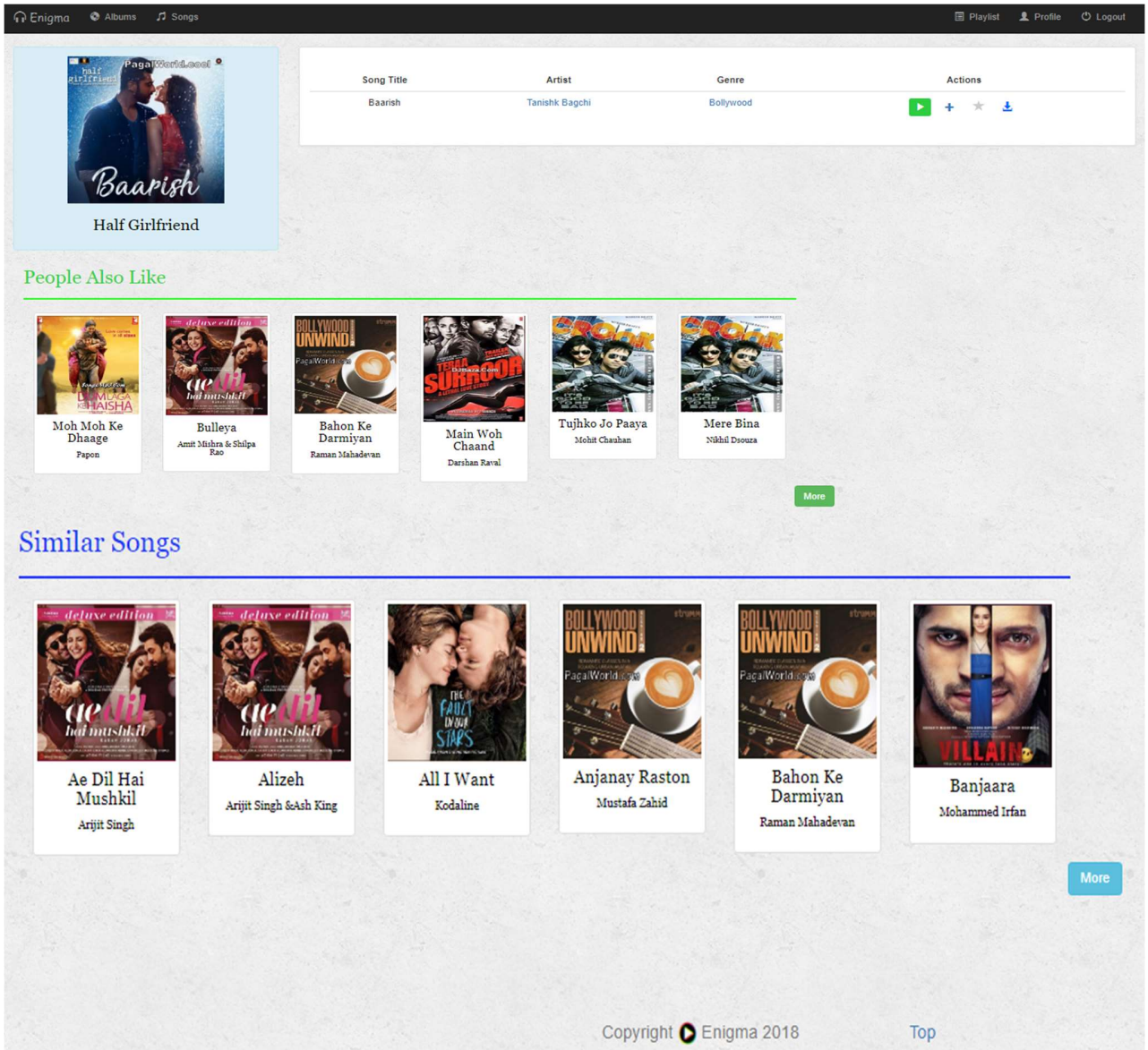

Figure. 3.13

Albums

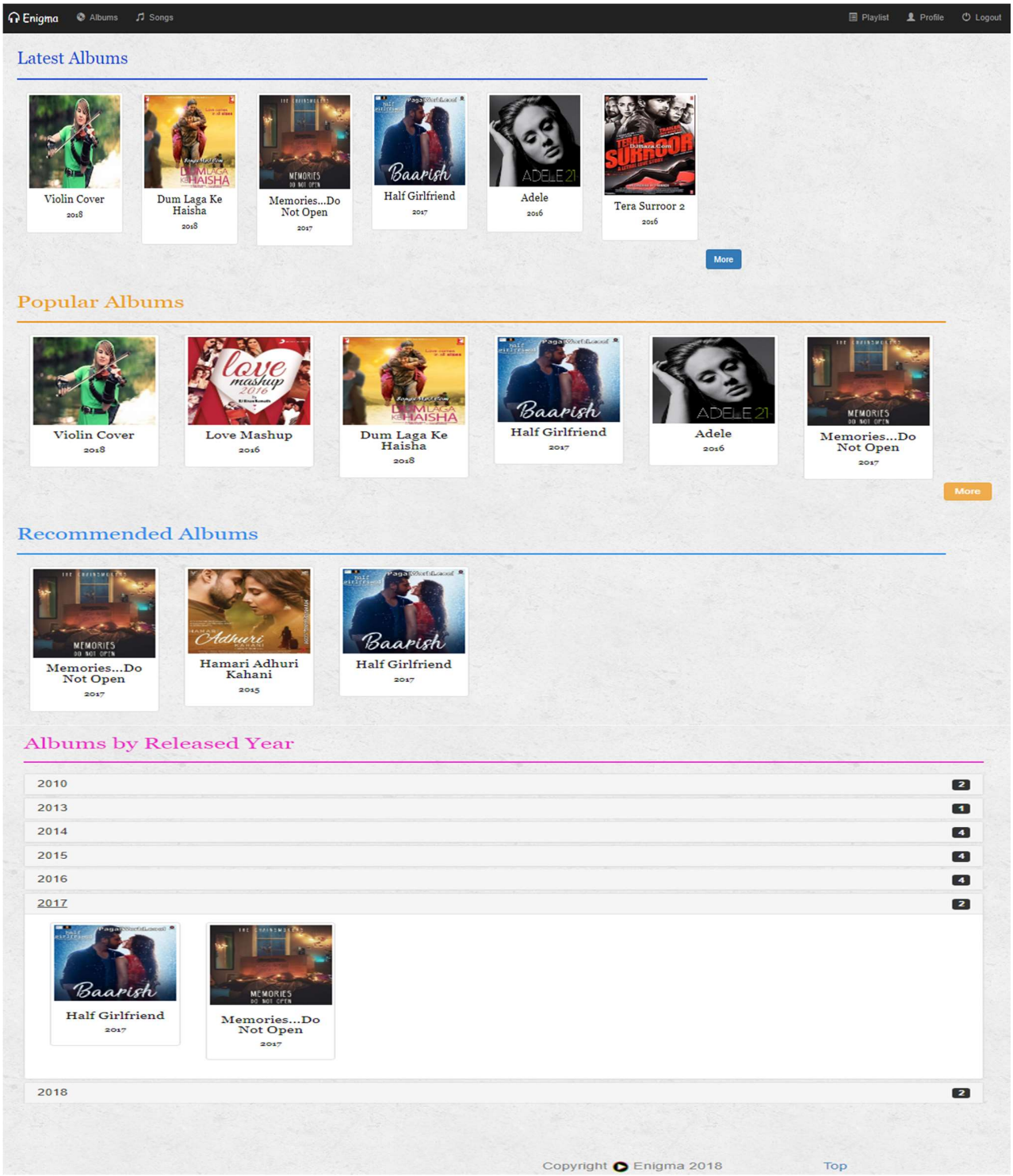

Figure. 3.14

Similar Albums

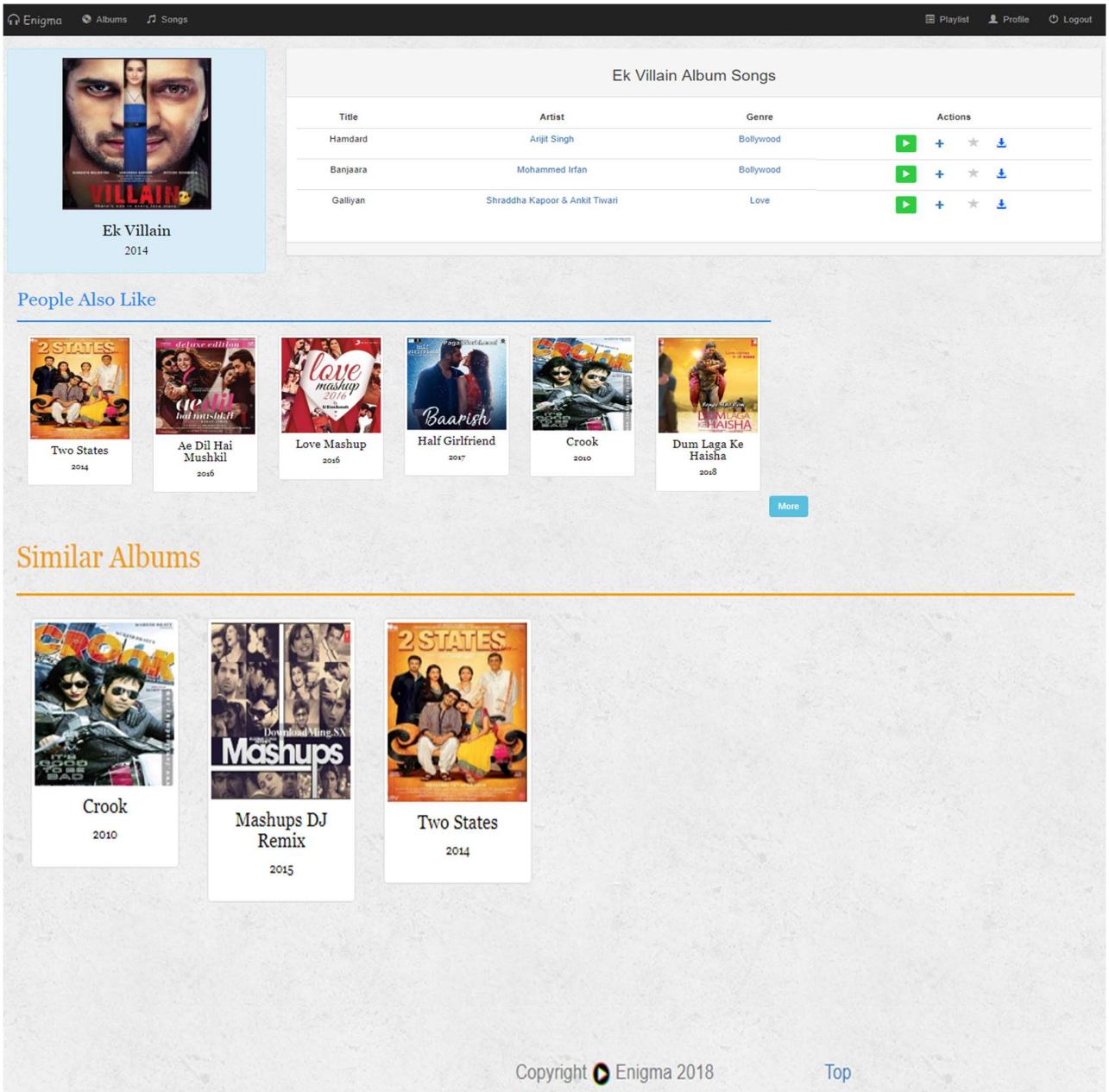

Figure. 3.15

Search

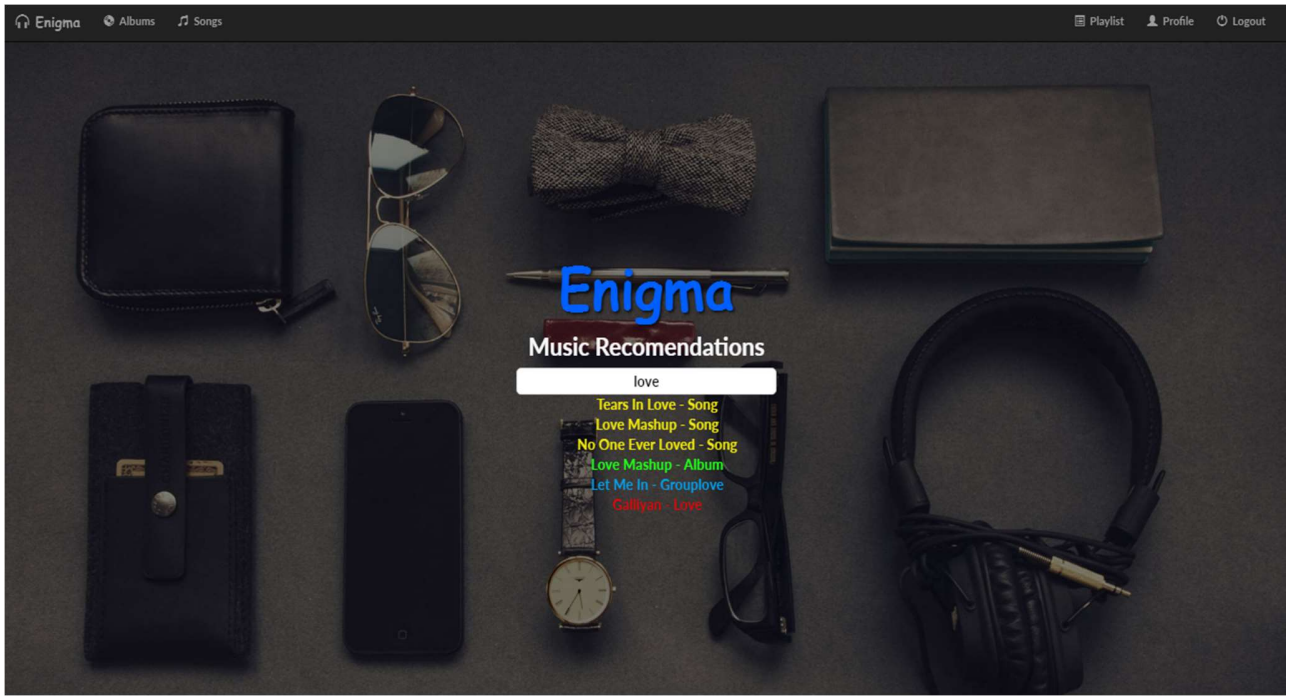

Figure. 3.16

User Playlist

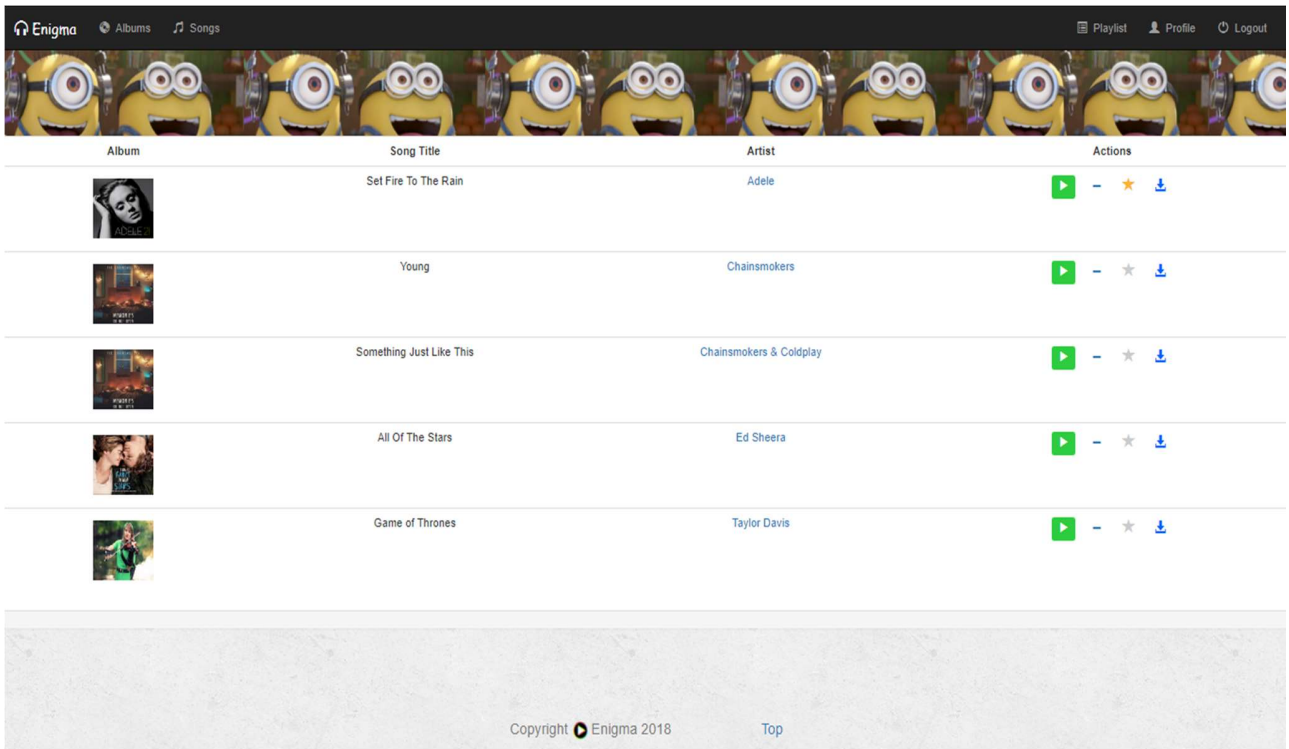

Figure. 3.17

User Profile

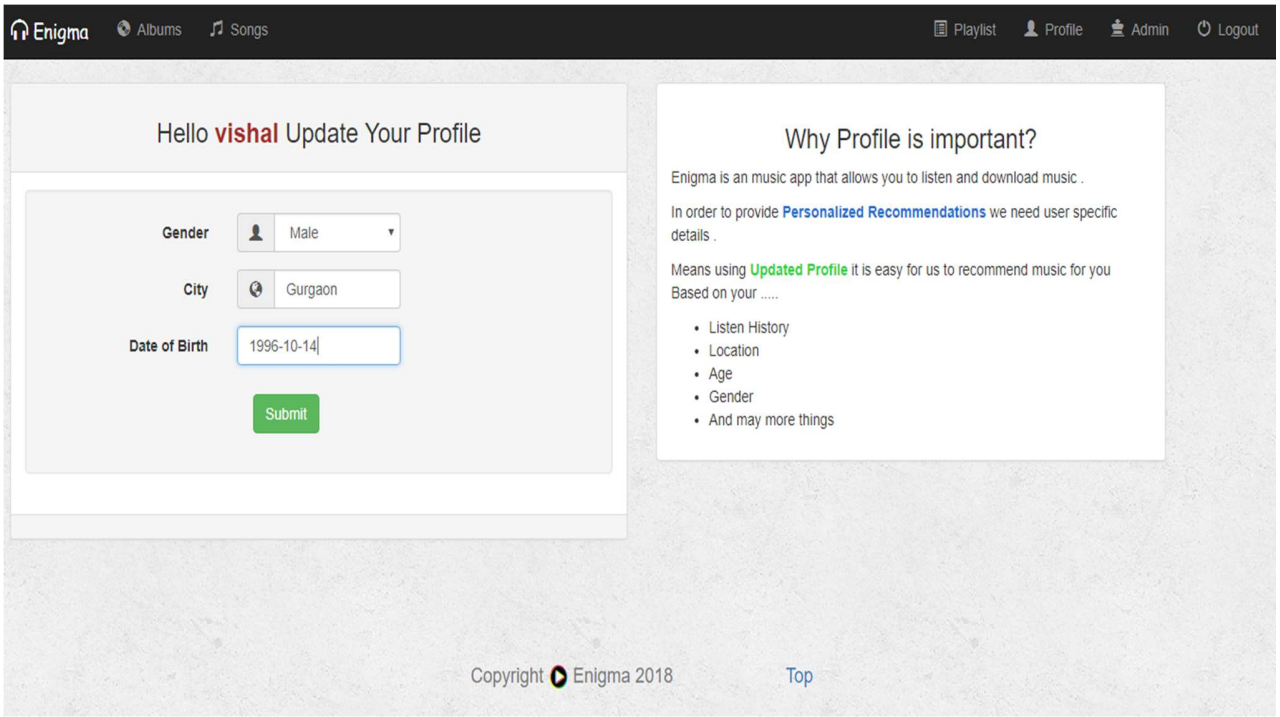

Figure. 3.18

Contact Me From

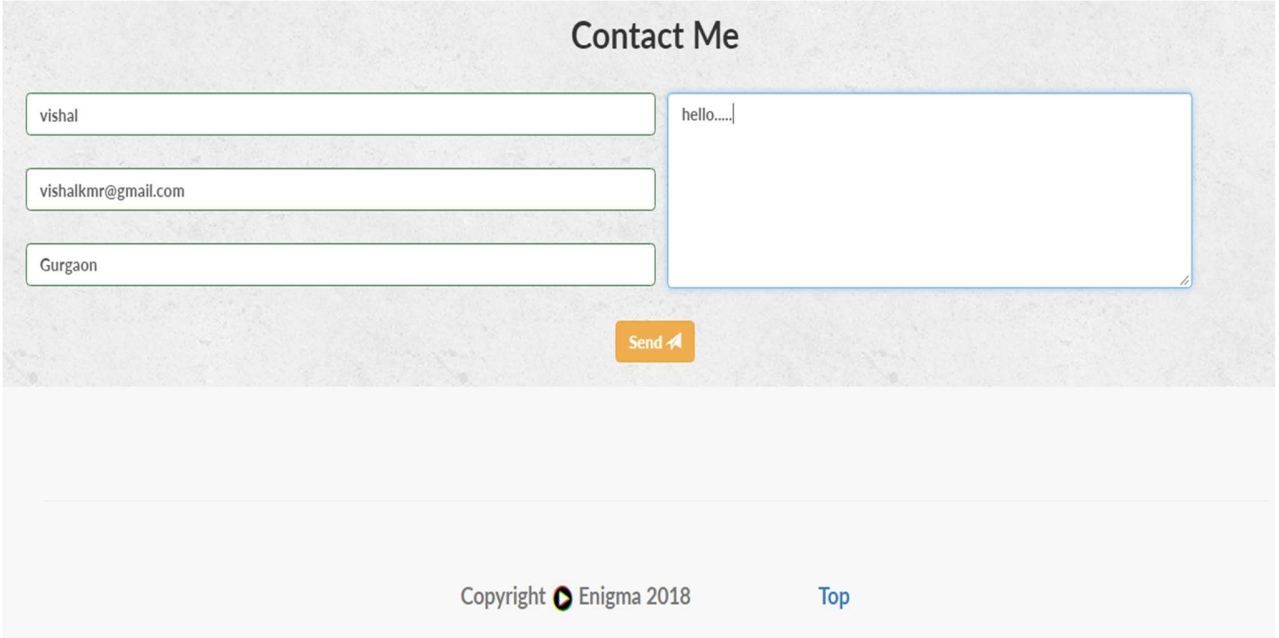

Figure. 3.19

Admin Panel

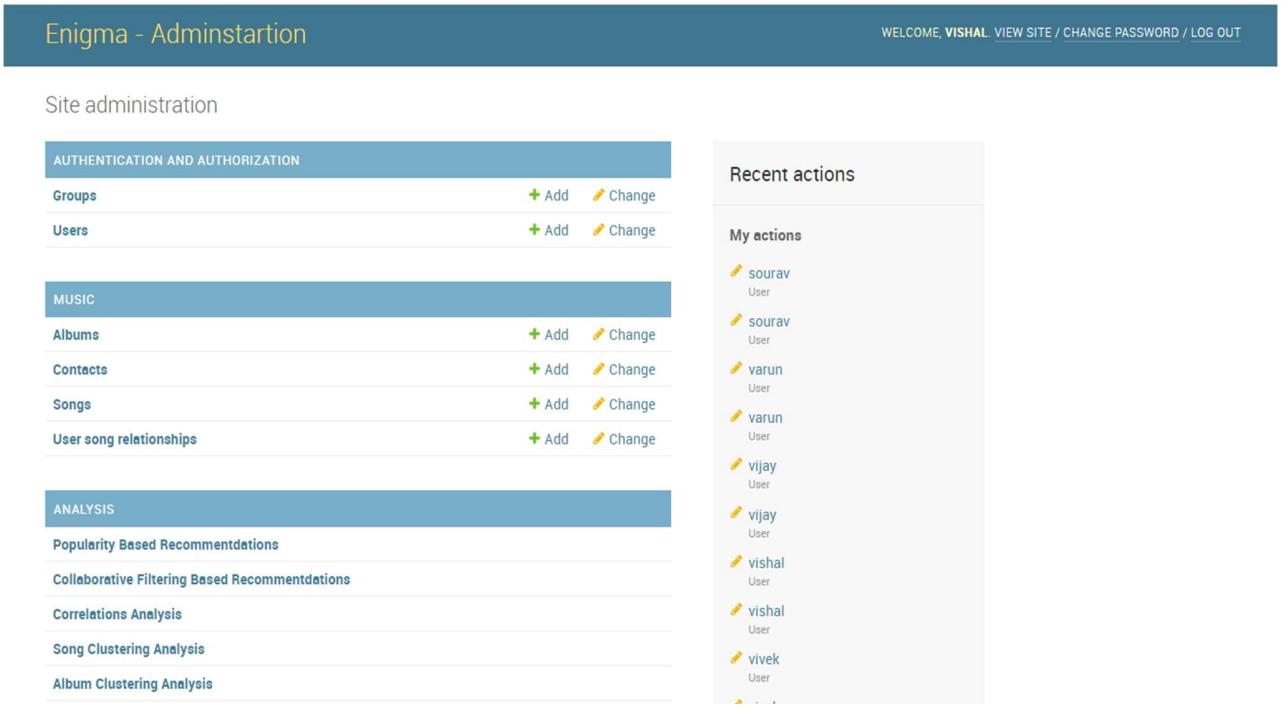

Figure. 3.20

User Table Admin View

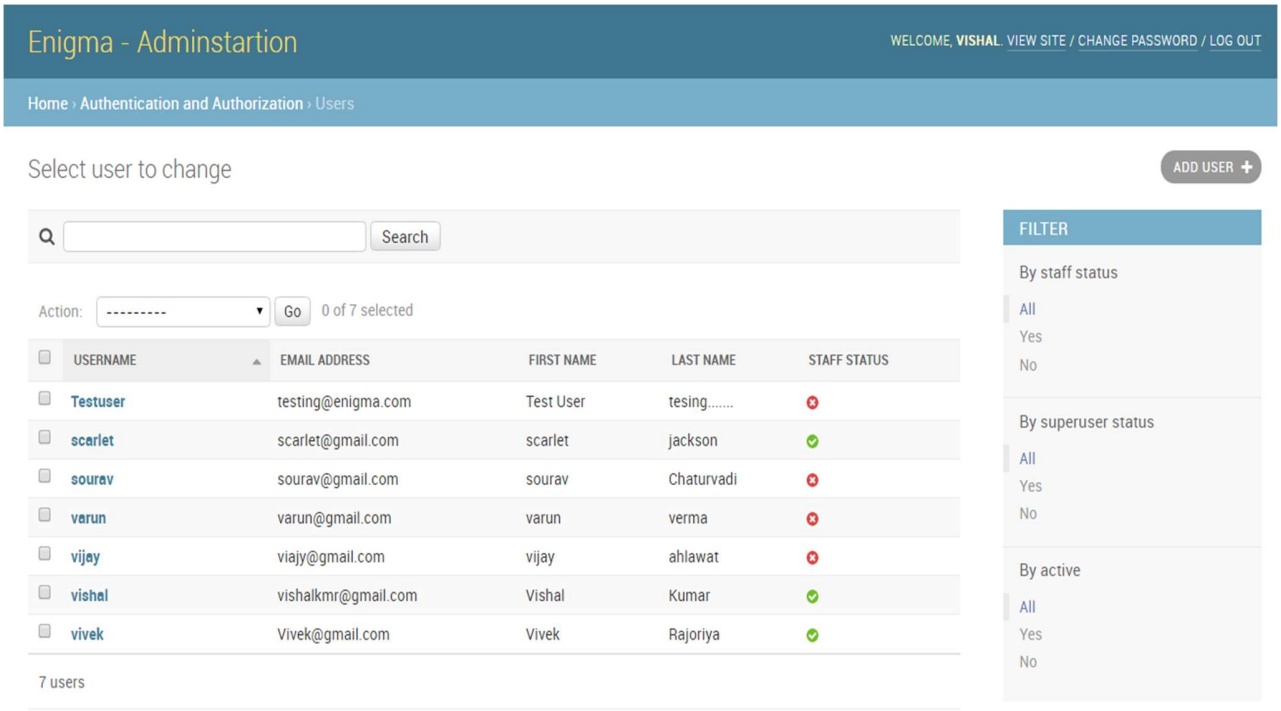

Figure. 3.21

Song Table Admin View

Enigma - Adminstartion

WELCOME, **VISHAL**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Music › Songs

Select song to change

ADD SONG +

Action: --------- Go 0 of 64 selected

SONG TITLE	ALBUM	ARTIST	GENRE
Young	Memories...Do Not Open 2017	Chainsmokers	Pop
Without Words	The Fault In Our Stars 2014	Ray LaMontagne	Soundtrack
While I'm Alive	The Fault In Our Stars 2014	STRFKR	Soundtrack
Tujhko Jo Paaya	Crook 2010	Mohit Chauhan	Bollywood
To Life	Recovery 2010	Eminem	Hip-Hop
Tears In Love	Mashups DJ Remix 2015	DJ Sunny Subramanian	Remix
Star Wars	Violin Cover 2018	Taylor Davis	Soundtrack
Space Bound	Recovery 2010	Eminem	Hip-Hop
Something Just Like This	Memories...Do Not Open 2017	Chainsmokers & Coldplay	Dance

Figure. 3.22

Album Table Admin View

Enigma - Adminstartion

WELCOME, **VISHAL**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Music › Albums

Select album to change

ADD ALBUM +

Action: --------- Go 0 of 19 selected

ALBUM TITLE	PUB YEAR
Adele	2016
Ae Dil Hai Mushkil	2016
Billboard Top 25 Rock Songs	2013
Bollywood Unwind	2015
Crook	2010
Dum Laga Ke Haisha	2018
Ek Villain	2014
Half Girlfriend	2017
Hamari Adhuri Kahani	2015
Kaash	2015
Love Mashup	2016

Figure. 3.23

UserSongRelationship Table Admin View

Enigma - Adminstartion

WELCOME, **VISHAL**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Music › User song relationships

Select user song relationship to change

ADD USER SONG RELATIONSHIP +

Action: --------- Go 0 of 100 selected

USER	SONG	IS FAVORITE 3	IS ADDED TO PLAYLIST 2	LISTEN COUNT 1
varun	Love Mashup	✓	✓	6
vivek	Love Mashup	✗	✓	3
vishal	Set Fire To The Rain	✓	✓	3
vijay	Ek Villain	✓	✓	3
sourav	Game of Thrones	✓	✓	3
vishal	Game of Thrones	✓	✓	3
vivek	Mere Bina	✗	✓	2
varun	Phir Bhi Tumko Chahunga	✗	✓	2
vivek	Banjaara	✓	✓	2

Figure. 3.24

Contact Table Admin View

Enigma - Adminstartion

WELCOME, **VISHAL**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Music › Contacts

Select contact to change

ADD CONTACT +

Action: --------- Go 0 of 2 selected

NAME	EMAIL	CITY	MESSAGE
vivek	vivek@gmail.com	Faridabad	hello
vishal	vishal@enigma.com	Gurgaon	Message Testing.....

2 contacts

Figure. 3.25

3.8 ALGORITHMS AND ANALYSIS:

3.8.1 Popularity Based Recommender

Popularity Based Recommender Class

```
class Popularity():
    def __init__(self):
        albums=pd.read_csv('albums.csv')
        songs=pd.read_csv('songs.csv')
        users=pd.read_csv('users.csv')
        usersongrelationships=pd.read_csv('usersongrelationships.csv')
        df1=pd.merge(songs, albums, on="album_id")
        df2=pd.merge(df1,usersongrelationships, on="song_id")
        df=pd.merge(df2, users, on="user_id")
        self.df=df
        self.unique_users_weight=1
        self.non_unique_users_weight=.25

    def song_age(self,song_title):
        x=str(self.df[self.df['song_title']==song_title]['pub_year'].unique())
        x=x.split('-')[0]
        time=localtime[0]-int(x[1:-1])
        if time==0:
            return 0.5
        else:
            return time

    def album_age(self,album_title):
        x=str(self.df[self.df['album_title']==album_title]['pub_year'].unique())
        x=x.split('-')[0]
        time=localtime[0]-int(x[1:-1])
        if time==0:
            return 0.5
        else:
            return time

    #Showing the most popular songs in the dataset
    def recommend_song(self):
        #df contains songs with atleat one listen_count
        df=self.df[self.df['listen_count']>0]
        grouped_data = self.df.groupby(['song_title']).sum()
        songs=grouped_data.index
        ################################
        #Listen Count history is Empty
        ################################
        if not len(df):
            print("*"*35+"Popular Songs"+'*'*50)
            print('\tListen Count history is Empty')
            print('\tDisplying random songs as popular songs')
            print("*"*100)
            return self.df
        # unique users for the given songs list
        u=[]
        for i in songs:
```

```
            u.append(df[df['song_title']==i]['username'].nunique())

        recommendations=grouped_data
        recommendations['unique_users']=u
        x=pd.DataFrame(recommendations.index)
        x.reset_index()
        y=x['song_title'].apply(self.song_age)
        recommendations['song_age']=[y[i] for i in range(0,len(y))]
        #Evaluating score for songs
        recommendations['score']=((recommendations['listen_count']-
        (recommendations['unique_users']))*self.non_unique_users_weight+
        recommendations['unique_users']*self.unique_users_weight)/
        recommendations['song_age']
        #sort the recommendations based upon score
        recommendations=recommendations.sort_values(by='score')[::-1]
        #removing Nan which may occures if unique users for song is 0
        recommendations=recommendations.dropna()
        recommendations= recommendations   [['listen_count','unique_users',
        'song_age','score']]
        recommendations.reset_index(inplace=True)
        recommendations.index.name='Rank'
        #removing the items whoes corelation is less than 0
        recommendations=recommendations[recommendations['score']>0]
        return recommendations

    #Showing the most popular albums in the dataset
    def recommend_album(self):
        #df contains albums with atleat one listen_count
        df=self.df[self.df['listen_count']>0]
        grouped_data = self.df.groupby(['album_title']).sum()
        albums=grouped_data.index
        ################################
        #Listen Count history is Empty
        ################################
        if not len(df):
            print("*"*35+"Popular Albums"+'*'*50)
            print('\tListen Count history is Empty')
            print('\tDisplying random albums as popular albums')
            print("*"*100)
            #returning random unique albums
            df=pd.DataFrame(self.df['album_title'].unique() )
            df.columns=['album_title']
            return df
        # unique songs for the given album
        s=[]
        for i in albums:
            s.append(df[df['album_title']==i]['song_title'].nunique())
        # unique users for the given songs list
        u=[]
        for i in albums:
            u.append(df[df['album_title']==i]['username'].nunique())
        recommendations =grouped_data
        recommendations['no_of_songs']=s
        recommendations['unique_users']=u
        x=pd.DataFrame(recommendations.index)
```

```
        x.reset_index()
        y=x['album_title'].apply(self.album_age)
        recommendations['album_age']=[y[i] for i in range(0,len(y))]
        #Evaluating score for albums
        recommendations['score']=((recommendations['listen_count']-
        recommendations['unique_users'])*self.non_unique_users_weight+
        recommendations['unique_users']*self.unique_users_weight)/
        (recommendations['no_of_songs']*recommendations['album_age'])
        #sort the recommendations based upon score
        recommendations=recommendations.sort_values(by='score')[::-1]
        #removing Nan which may occures if unique users for song is 0
        recommendations=recommendations.dropna()
        recommendations=recommendations[['listen_count','no_of_songs','unique_users'
        ,'album_age','score']]
        recommendations.reset_index(inplace=True)
        recommendations.index.name='Rank'
        #removing the items whoes corelation is less than 0
        recommendations=recommendations[recommendations['score']>0]
        return recommendations
```

Popularity Based Recommender Analysis

Figure showing the correlation between features for song

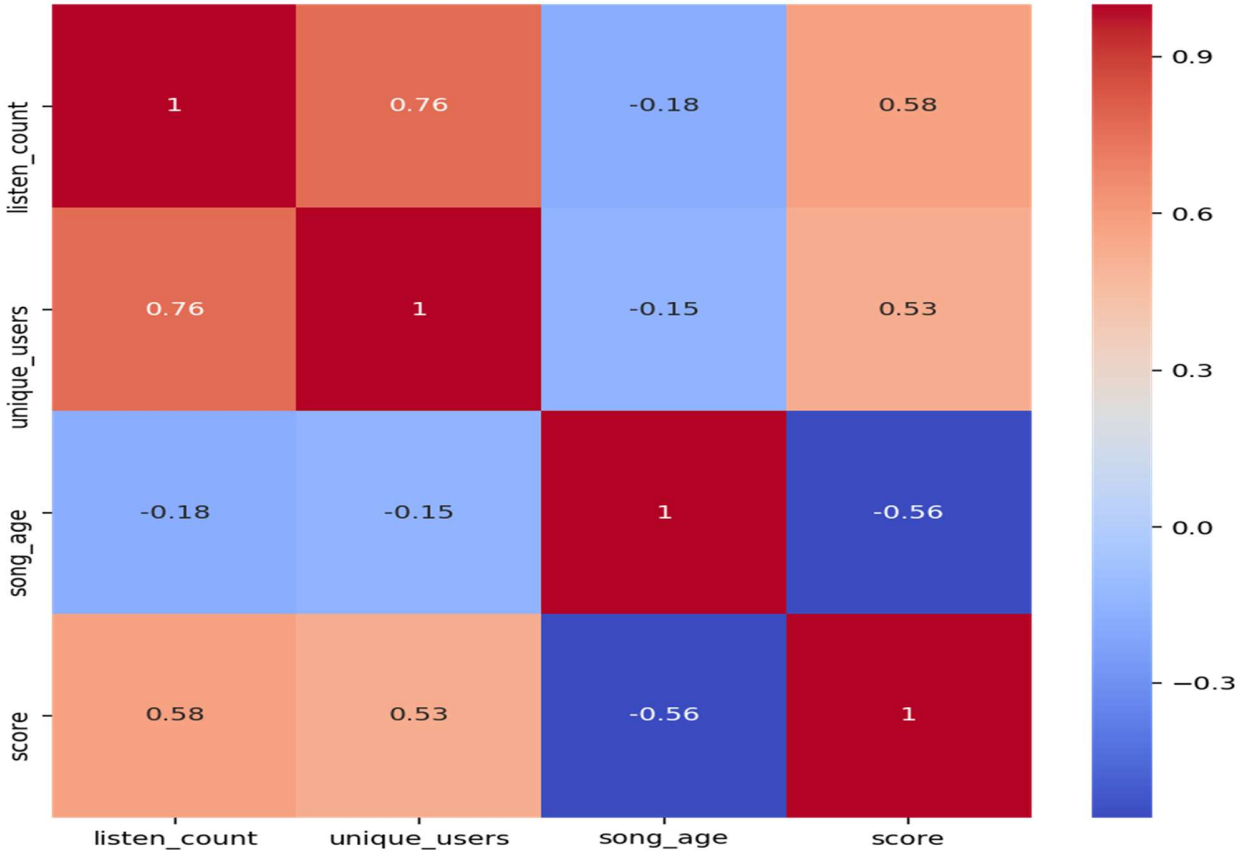

Figure. 3.26

Popular Songs Scoring Formula

Score = ([non_unique_user_listen_count * non_unique_users_weight] + [unique_user_listen_count * unique_users_weight]) / song_ age

Where unique_users_weight : 1 non_unique_users_weight : 0.25

Example of Popular Songs Sore evaluation:

Table. 3.1

song_title	listen_count	unique_users	song_age	score
Set Fire To The Rain	5	3	2	1.75

Figure showing the values of various features of corresponding songs

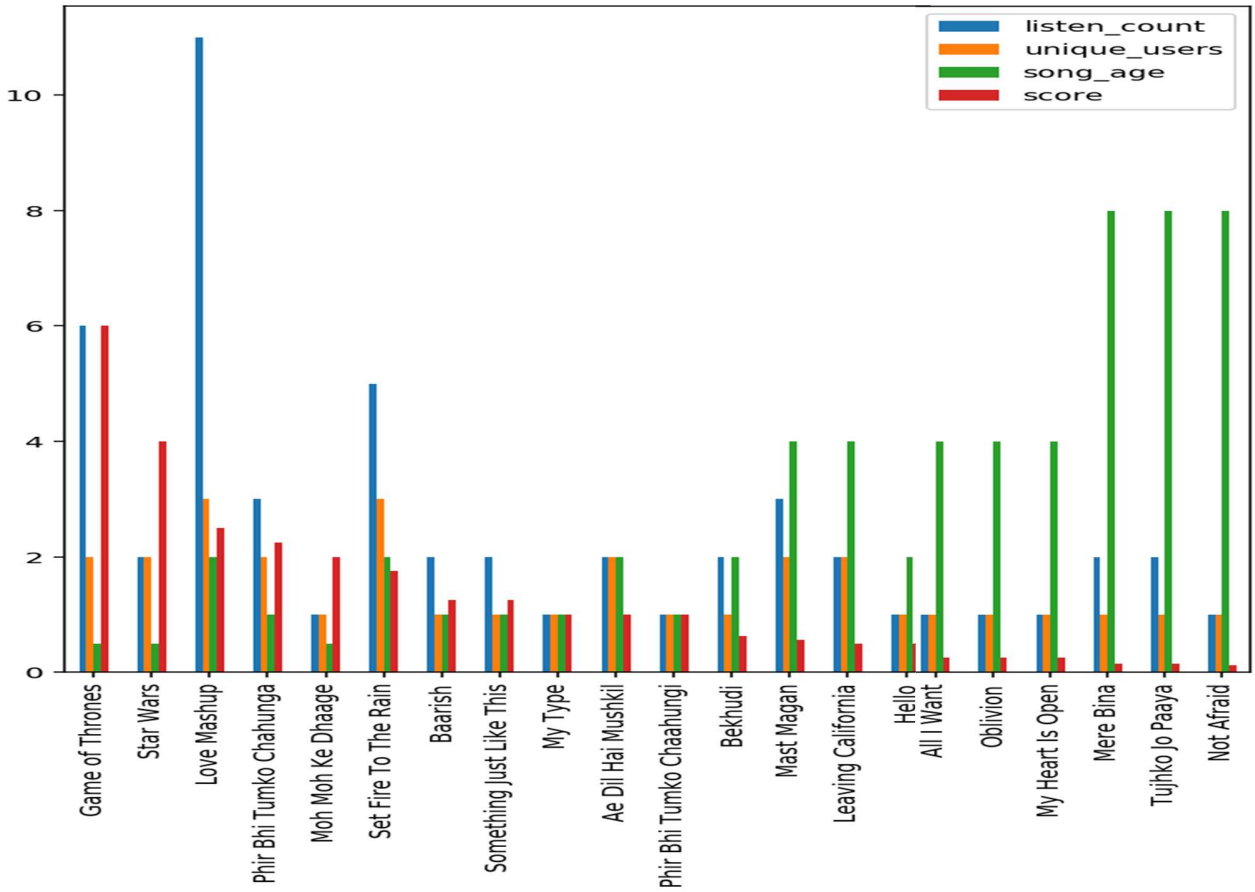

Figure. 3.27

3.8.2 Collaborative Filtering Based Recommender

Collaborative Filtering Recommender Class

```
class  Collaborative_Filtering (object):
    def __init__(self,username):
        albums=pd.read_csv('albums.csv')
        songs=pd.read_csv('songs.csv')
        users=pd.read_csv('users.csv')
        usersongrelationships=pd.read_csv('usersongrelationships.csv')
        df1=pd.merge(songs, albums, on="album_id")
        df2=pd.merge(df1,usersongrelationships, on="song_id")
        df=pd.merge(df2, users, on="user_id")
        self.df=df
        self.username=username
        self.all_users=self.df['username'].unique() #unique users
        self.all_songs=self.df['song_title'].unique() #unique songs
        self.all_albums=self.df['album_title'].unique() #unique albums
        self.user_songs= self.df[(self.df['listen_count']>0) &
        (self.df['username']==self.username)]['song_title'].unique()
        self.non_user_songs=self.df[(self.df['listen_count']==0) &
        (self.df['username']==self.username)]['song_title'].unique()
        self.user_albums= self.df[(self.df['listen_count']>0) &
        (self.df['username']==self.username)]['album_title'].unique()
        self.non_user_albums= []
        for album in self.all_albums:
            if album  not in self.user_albums:
                self.non_user_albums.append(album)
    #Recommends songs the (no. of user listen both song i and j)/(no. of user listen
    either song i or j)
    def songs_commmon_listen_count(self,i,j):
        c=0
        d=0
        for u in self.all_users:
            if (u !=self.username ):
                a=np.array(self.df[(self.df['listen_count']>0) &
                (self.df['username']==u)]['song_title'])
                if i in a and j in a:
                    c=c+1
                if i in a or j in a:
                    d=d+1
                    # d=0
        # for normalization
        if d:
            return c/d
        else:
            return c

    def construct_song_cooccurence_matrix(self):
        ##############################################
        #Construct cooccurence matrix dataframe of size
        #len(user_songs) X len(non_user_songs)
        ##############################################
        self.song_cooccurence_matrix=pd.DataFrame(np.matrix(np.zeros(shape=(len(
        self.user_songs), len(self.non_user_songs))), float)
```

```
        ,self.user_songs,self.non_user_songs)
        for i in self.user_songs:
            for j in self.non_user_songs:
                self.song_cooccurence_matrix.loc[i,j]=
                self.songs_commmon_listen_count(i,j)

        self.song_cooccurence_matrix.dropna(inplace=True)
        recommendations=pd.DataFrame(self.song_cooccurence_matrix.sum().
        sort_values()[::-1])
        self.s_song_cooccurence_matrix=pd.DataFrame(self.song_cooccurence_matrix.
        sum())
        self.s_song_cooccurence_matrix.columns=['Summation']
        #removing the items whoes corelation is less than 0
        recommendations=recommendations[recommendations[0]>0]
        self.n_song_cooccurence_matrix=recommendations
        self.n_song_cooccurence_matrix.columns=['Summation']
        recommendations=recommendations.reset_index()
        recommendations.index.name='Rank'
        recommendations.columns=['song_title','score']
        self.song_recommendations=recommendations

    #Recommends songs to login user
    def recommend_song(self):
        ##############################
        #User history is not present
        #Collaborative Filtering not possible
        #Recommending random songs
        ##############################
        if len(self.user_songs)==0:
            temp=pd.DataFrame(self.df['song_title'].unique())
            temp.columns=['song_title']
            self.song_recommendations =temp
            return self.song_recommendations
        else :
            self.construct_song_cooccurence_matrix()
            return self.song_recommendations

    #returning (the no. of user listen both album i and j)/(no. of user listen either
    album i or j)
    def albums_commmon_listen_count(self,i,j):
        c=0
        d=0
        for u in self.all_users:
            if (u !=self.username ) :
                a=np.array(self.df[(self.df['listen_count']>0) &
                (self.df['username']==u)]['album_title'].unique())
                if i in a and j in a:
                    c=c+1
                if i in a or j in a:
                    d=d+1
        # for normalization
        if d:
            return c/d
        return c
    def construct_album_cooccurence_matrix(self):
```

```
        ##############################################
        #Construct cooccurence matrix dataframe of size
        #len(user_albums) X len(non_user_albums)
        ##############################################
        self.album_cooccurence_matrix=pd.DataFrame(np.matrix(np.zeros(shape=
        (len(self.user_albums), len(self.non_user_albums))), float) ,
        self.user_albums,self.non_user_albums)

        for i in self.user_albums:
            for j in self.non_user_albums:
                self.album_cooccurence_matrix.loc[i,j]=
                 self.albums_commmon_listen_count(i,j)
        self.album_cooccurence_matrix.dropna(inplace=True)
        recommendations=pd.DataFrame(self.album_cooccurence_matrix.sum().
        sort_values()[::-1]
        self.s_album_cooccurence_matrix=pd.DataFrame(self.album_cooccurence_matrix.
        sum())
        self.s_album_cooccurence_matrix.columns=['Summation']
        #removing the items whoes corelation is less than 0
        recommendations=recommendations[recommendations[0]>0]
        self.n_album_cooccurence_matrix=recommendations
        self.n_album_cooccurence_matrix.columns=['Summation']
        recommendations=recommendations.reset_index()
        recommendations.index.name='Rank'
        recommendations.columns=['album_title','score']
        self.album_recommendations=recommendations

    #Recommends albums to login user
    def recommend_album(self):
        if len(self.user_albums)==0:
            print("*"*35+" Collaborative Filtering Recommended Albums "+'*'*40)
            print('\tUser history is not present ')
            print('\tCollaborative Filtering not possible ')
            print('\tRecommending random albums ')
            print('*'*100)
            temp=pd.DataFrame(self.df['album_title'].unique())
            temp.columns=['album_title']
            self.album_recommendations =temp
            return self.album_recommendations
        else :
            self.construct_album_cooccurence_matrix()
            return self.album_recommendations

    #gives similar songs correspoding to givrn song using cooccurence_matrix method
    def similar_song(self,song_title):
        ##############################################
        #Construct cooccurence matrix dataframe of size
        #1 X len(non_user_songs)
        ##############################################
        song=self.df[(self.df['listen_count']>0) &
        (self.df['song_title']==song_title)]
        song=song['song_title'].unique()
        self.similar_song_cooccurence_matrix=pd.DataFrame(np.array(np.zeros(shape=
        (len(song), len(self.non_user_songs))), float) ,song,
        self.non_user_songs)
```

```
        for j in self.non_user_songs:
            self.similar_song_cooccurence_matrix.loc[song_title,j]=
            self.songs_commmon_listen_count(song_title,j)
        self.similar_song_cooccurence_matrix.dropna(inplace=True)
        similar_songs=pd.DataFrame(self.similar_song_cooccurence_matrix.sum().
        sort_values()[::-1])

        #removing the current song from similar item (it may be present if user not
        listen that song  already)
        try:
            similar_songs.drop(song_title,axis=0,inplace=True)
        except:
            pass
        similar_songs.columns=[song_title]
        #removing the items whoes corelation is less than 0
        similar_songs=similar_songs[similar_songs[song_title]>0]
        self.n_similar_song_cooccurence_matrix=similar_songs
        self.similar_songs=similar_songs.reset_index()
        self.similar_songs.index.name='Rank'
        self.similar_songs.columns=['song_title','score']
        return self.similar_songs

    #gives similar albums corresponding to given album using
    cooccurence_matrix(Collaborative Filtering) method
    def similar_album(self,album_title):
        ##############################################
        #Construct cooccurence matrix dataframe of size
        #1 X len(non_user_albums)
        ##############################################
        album=self.df[(self.df['listen_count']>0) &
        (self.df['album_title']==album_title)]
        album=album['album_title'].unique()
        self.similar_album_cooccurence_matrix=pd.DataFrame(np.array(np.zeros(shape=
        (len(album), len(self.non_user_albums))), float) ,album,self.non_user_albums)
        for j in self.non_user_albums:
            self.similar_album_cooccurence_matrix.loc[album_title,j]=
             self.albums_commmon_listen_count(album_title,j)
        self.similar_album_cooccurence_matrix.dropna(inplace=True)
        similar_albums=pd.DataFrame(self.similar_album_cooccurence_matrix.sum().
         sort_values()[::-1])
        #removing the current album from similar item  (it may be present if user
        not listen that album  already)
        try:
            similar_albums.drop(album_title,axis=0,inplace=True)
        except:
            pass
        similar_albums.columns=[album_title]
        #removing the items whoes corelation is less than 0
        similar_albums=similar_albums[similar_albums[album_title]>0]
        self.n_similar_album_cooccurence_matrix=similar_albums
        self.similar_albums=similar_albums.reset_index()
        self.similar_albums.index.name='Rank'
        self.similar_albums.columns=['album_title','score']
        return similar_albums
```

Collaborative Filtering Recommender Analysis

Figure showing the Song Co-occurrence Matrix

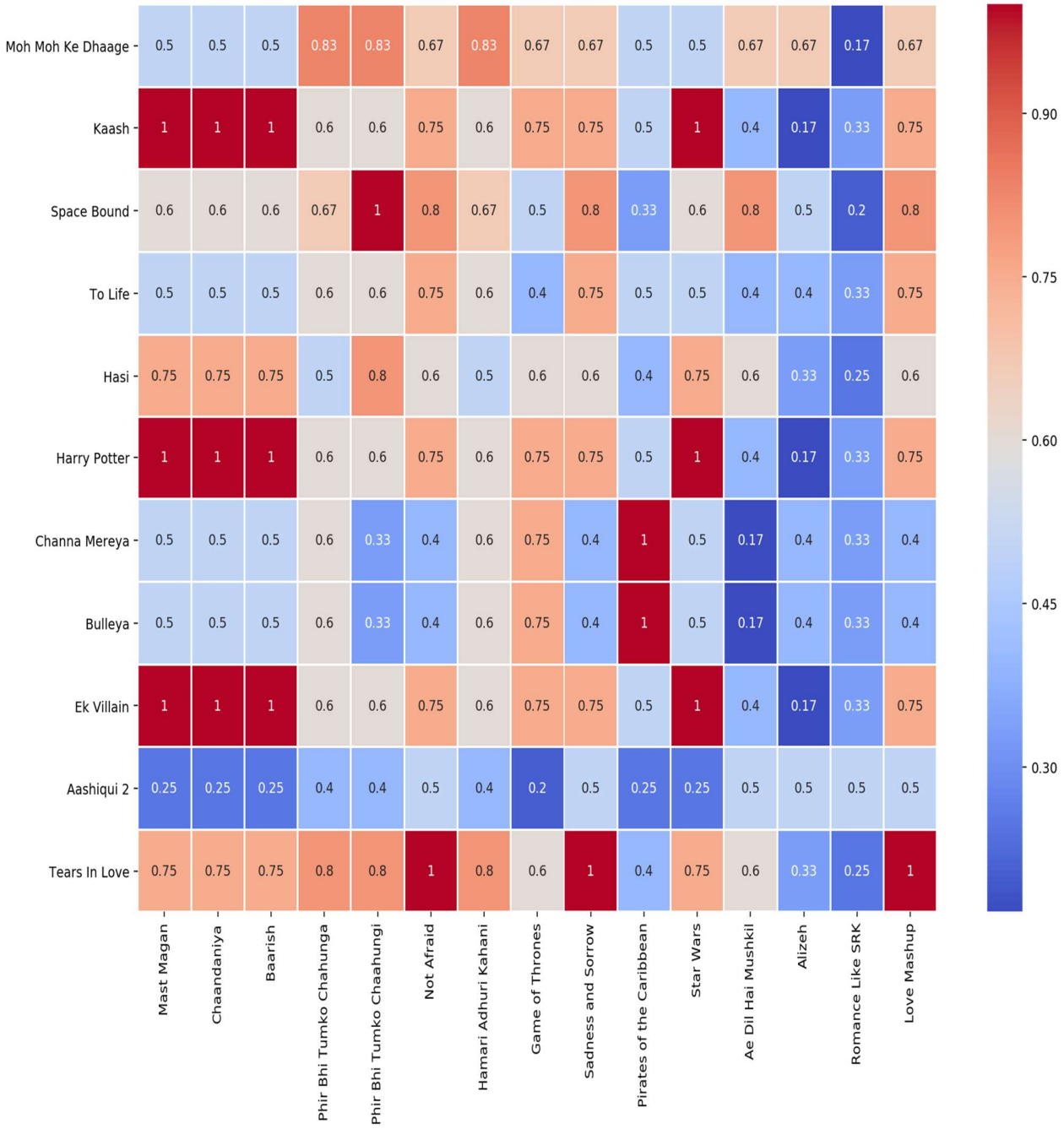

Figure. 3.28

Figure showing the Summation across columns in Song Co-occurrence Matrix

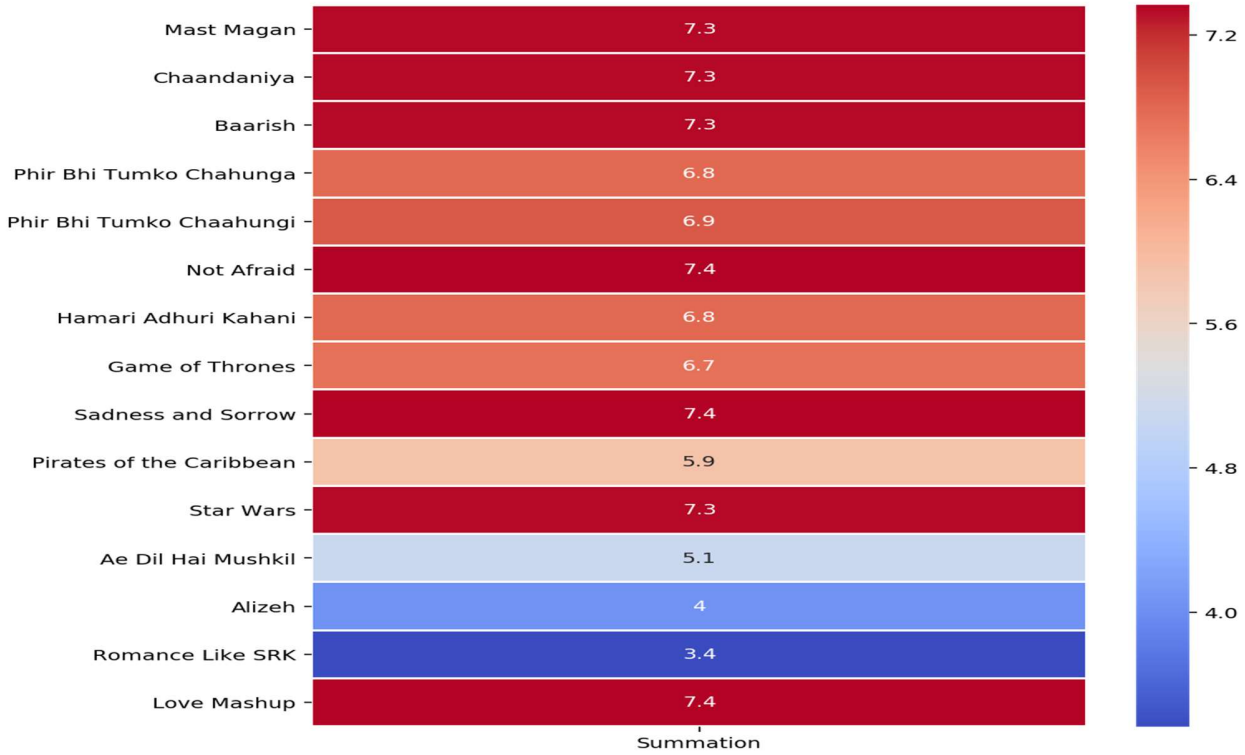

Figure. 3.29

Figure showing the Normalized Song Recommendations

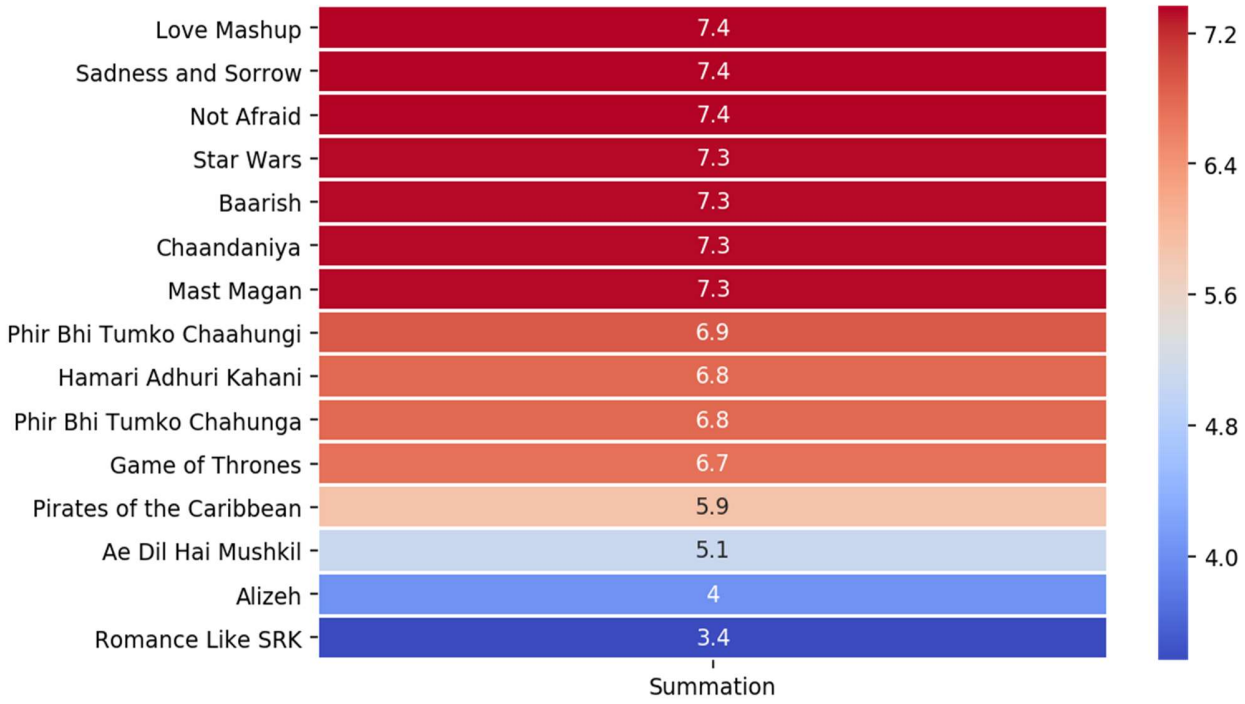

Figure. 3.30

3.8.3 Clustering Based Recommender

Clustering Recommender Class

```
class Clustering():
    def __init__(self,username):
        albums=pd.read_csv('albums.csv')
        songs=pd.read_csv('songs.csv')
        users=pd.read_csv('users.csv')
        usersongrelationships=pd.read_csv('usersongrelationships.csv')
        df1=pd.merge(songs, albums, on="album_id")
        df2=pd.merge(df1,usersongrelationships, on="song_id")
        df=pd.merge(df2, users, on="user_id")
        self.df=df
        self.username=username
        self.all_users=self.df['username'].unique() #unique users
        self.all_songs=self.df['song_title'].unique() #unique songs
        self.all_albums=self.df['album_title'].unique() #unique albums
        self.user_songs= self.df[(self.df['listen_count']>0) &
        (self.df['username']==self.username)]['song_title'].unique()
        self.non_user_songs=self.df[(self.df['listen_count']==0) &
        (self.df['username']==self.username)]['song_title'].unique()
        self.user_albums= self.df[(self.df['listen_count']>0) &
        (self.df['username']==self.username)]['album_title'].unique()
        self.non_user_albums= []
        for album in self.all_albums:
            if album  not in self.user_albums:
                self.non_user_albums.append(album)

    def song_year(self,song_title):
        x=str(self.df[self.df['song_title']==song_title]['pub_year'].unique())
        x=x.split('-')[0]
        return int(x[1:-1])

    def album_year(self,album_title):
        x=str(self.df[self.df['album_title']==album_title]['pub_year'].unique())
        x=x.split('-')[0]
        return int(x[1:-1])
/
    def genre(self,song_title):
        x=str(self.df[self.df['song_title']==song_title]['genre'].unique())
        return x[2:-2]

    #gives simialr songs correspoding to given song using Clustering method
    def similar_song(self,song_title):
        u=self.df[['song_title','listen_count','is_favorite','is_added_to_playlist']]
        u=u.groupby('song_title').sum()
        x=pd.DataFrame(u.index)
        x.reset_index()
        y=x['song_title'].apply(self.song_year)
        g=x['song_title'].apply(self.genre)
        u['genre']=[g[i] for i in range(0,len(g))]
        u['pub_year']=[y[i] for i in range(0,len(y))]
        X = u.iloc[:, :].values
```

```
        # Encoding categorical data
        from sklearn.preprocessing import LabelEncoder, OneHotEncoder
        labelencoder_X = LabelEncoder()
        X[:, -2] = labelencoder_X.fit_transform(X[:, -2])
        onehotencoder = OneHotEncoder(categorical_features = [-2])
        X = onehotencoder.fit_transform(X).toarray()
        # Feature Scaling
        sc_X = StandardScaler()
        SX = sc_X.fit_transform(X)
        # Fitting K-Means to the dataset
        kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 0)
        y_sc_kmeans = kmeans.fit_predict(SX)
        su=pd.DataFrame(u.index)
        su['cluster']=y_sc_kmeans
        su.set_index('song_title',inplace=True)
        similar_songs=pd.DataFrame(su[su['cluster']==su[su.index==song_title]
        ['cluster'][0]].index)
        similar_songs.set_index(np.arange( 1, len(similar_songs)+1 ),inplace=True)

        #removing items which are already listened by user
        for s in self.user_songs:
          try:
              similar_songs.drop(similar_songs[similar_songs['song_title']==s].
              index[0],axis=0,inplace=True)

          except:
              pass

        #removing the current song from similar item
        try:
            similar_songs.drop(similar_songs[similar_songs['song_title']==
            song_title].index[0],axis=0,inplace=True)
        except:
            pass
        similar_songs.set_index(np.arange( 1, len(similar_songs)+1 ),inplace=True)
        print("*"*35+" Similar Songs Based on Clustering"+'*'*30)
        print('Cluster  : '+str(su[su.index==song_title]['cluster'][0]))
        print(similar_songs)
        print('*'*100)
        return similar_songs

    #gives simialr songs correspoding to given song using Clustering method
    def similar_album(self,album_title):
        u=self.df[['username','gender','album_title','date_of_birth',
        'song_title','artist','listen_count','is_favorite','is_added_to_playlist']]
        u=u.groupby('album_title').sum()
        x=pd.DataFrame(u.index)
        x.reset_index()
        y=x['album_title'].apply(self.album_year)
        u['pub_year']=[y[i] for i in range(0,len(y))]
        X = u.iloc[:, :].values
        # Feature Scaling
        from sklearn.preprocessing import StandardScaler
        sc_X = StandardScaler()
        SX = sc_X.fit_transform(X)
```

```
        # Fitting K-Means to the dataset
        kmeans = KMeans(n_clusters = 4, init = 'k-means++', random_state = 0)
        y_kmeans = kmeans.fit_predict(SX)
        u['cluster']=y_kmeans
        similar_albums=pd.DataFrame(u[u['cluster']==u[u.index==album_title]
        ['cluster'][0]].index)
        similar_albums.set_index(np.arange( 1, len(similar_albums)+1 ),inplace=True)

        #removing items which are already listened by user
        for s in self.user_albums:
          try:
            similar_albums.drop(similar_albums[similar_albums['album_title']==s].
            index[0],axis=0,inplace=True)
          except:
              pass
        #removing the current album from similar item
        try:
            similar_albums.drop(similar_albums[similar_albums['album_title']==
            album_title].index[0],axis=0,inplace=True)
        except:
            pass
        similar_albums.set_index(np.arange( 1, len(similar_albums)+1),inplace=True)
        print("*"*35+" Similar Albums Based on Clustering "+'*'*30)
        print('Cluster  : '+str(u[u.index==album_title]['cluster'][0]))
        print(similar_albums)
        print('*'*100)
        return similar_albums
```

Clustering Recommender Analysis

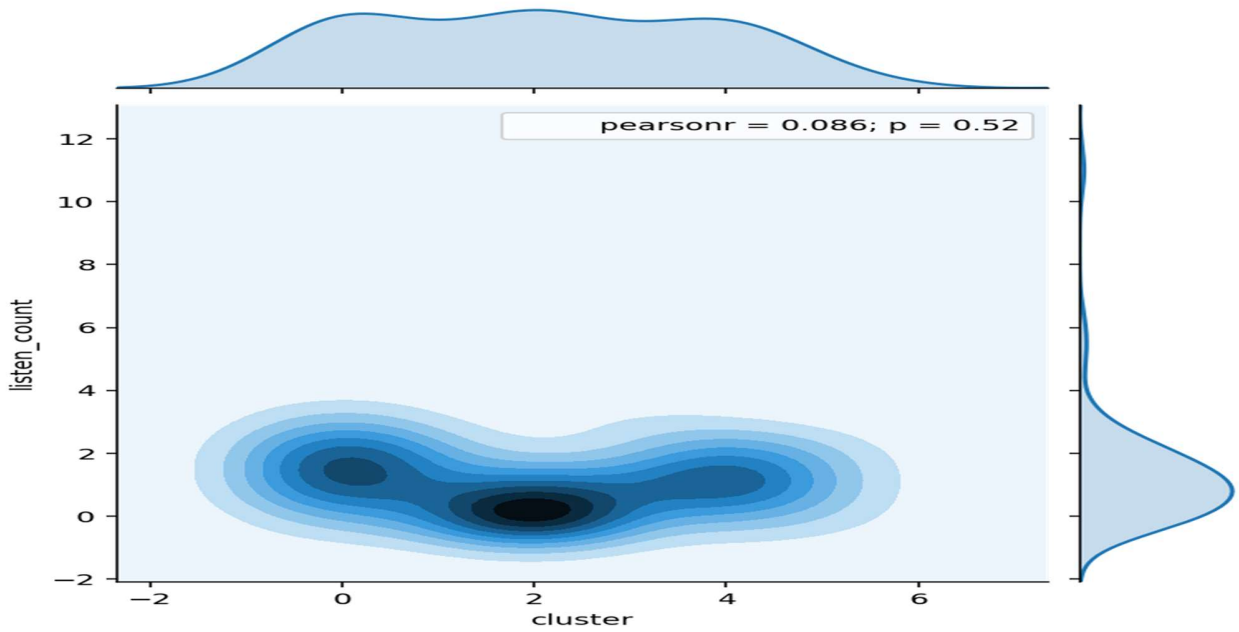

Figure. 3.31

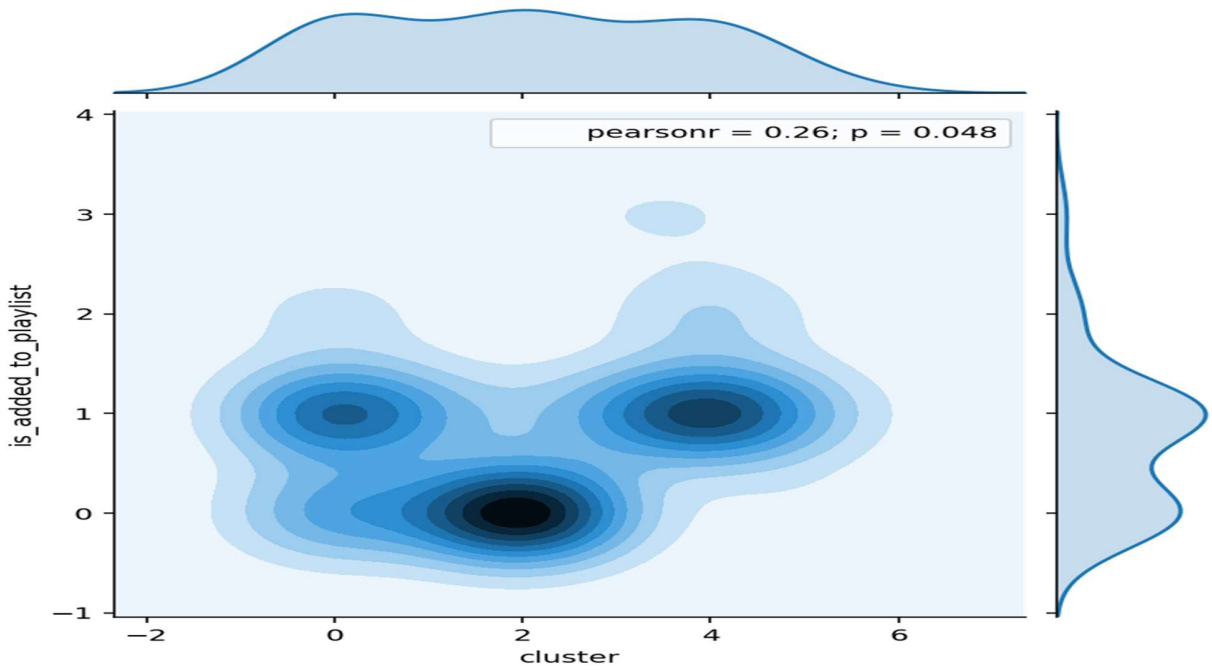

Figure. 3.32

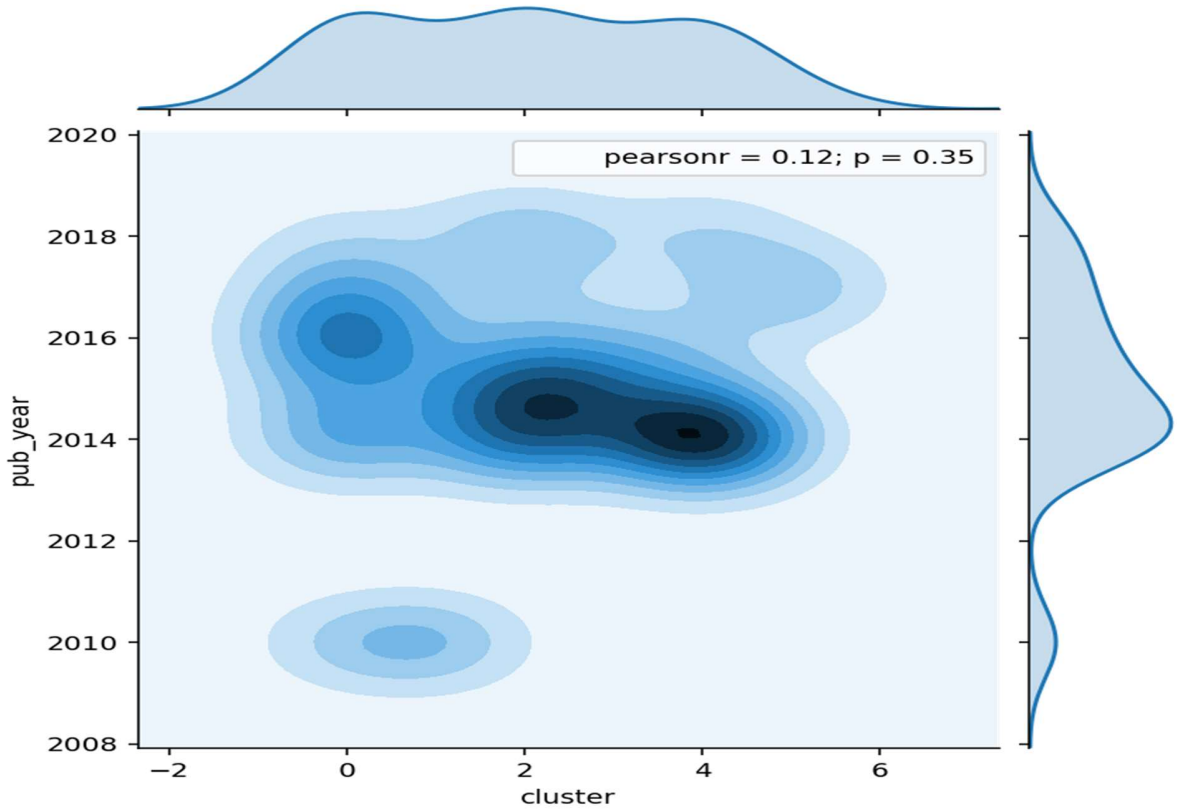

Figure. 3.33

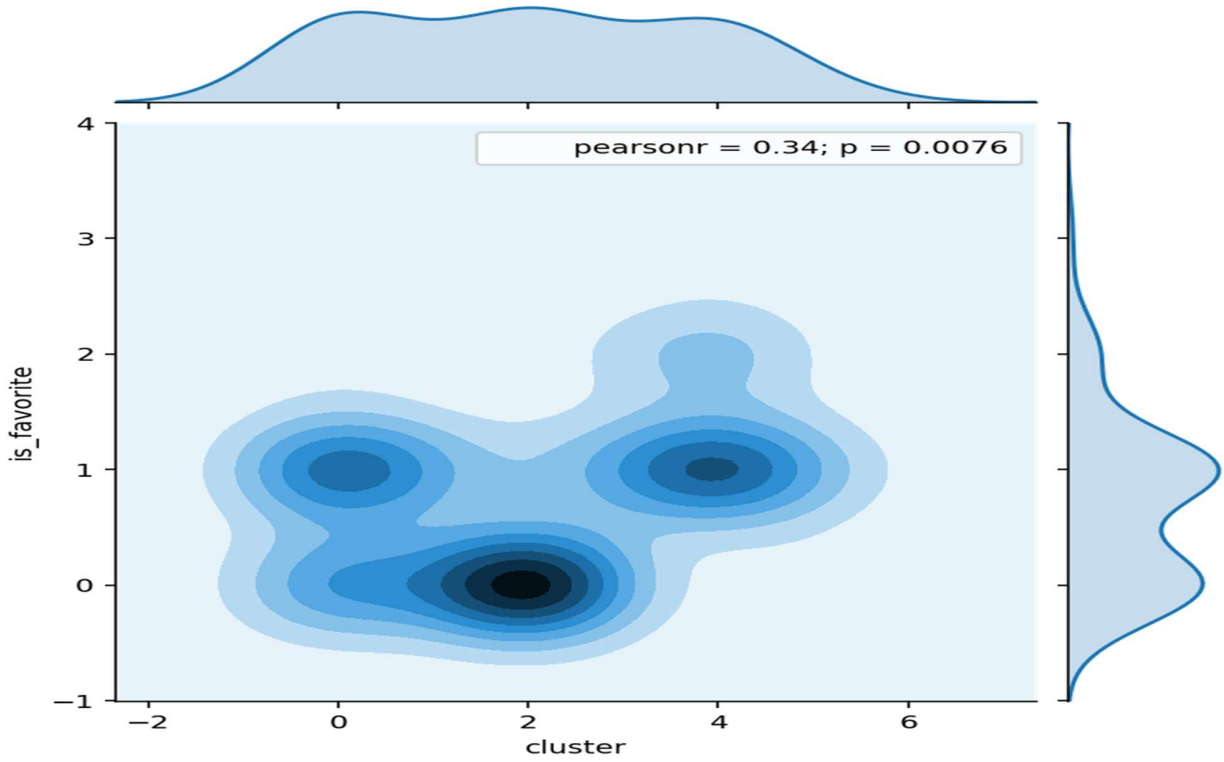

Figure. 3.34

3.9 TESTING PROCESS:

Software testing is a critical phase of SDLC that helps to achieve software quality assurance and represents the ultimate review of specification, design and coding.

Testing is the process of exercising the software item to detect the differences between its behavior and the desired behavior as spitulated by the requirement specifications- 'what is' and 'what should be'. Testing is a destructive process.

We begin the testing process by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are used. The process verifies that the application meets the requirements specified in the system requirements document and is bug free. At the end of each testing day, the team prepares a summary of completed and failed tests. Our programmers address any identified issues, and the application is resubmitted to the testing team until every item is resolved. All changes and retesting are tracked through spreadsheets available to both the testing and programming teams. Applications are not allowed to launch until all identified problems are fixed. A report is prepared at the end of testing to show exactly what was tested and to list the final outcomes.

To achieve a very high standard in quality of delivery, a comprehensive and planned testing will be carried out during project execution. The testing phase requires completing various tests to ensure the accuracy of programmed code, the inclusion of expected functionality, and the interoperability of applications and other network components. Thorough testing is critical to ensuring systems meet end-user requirements.

Organizations should review and complete user, operator, and maintenance manuals during the testing phase. Additionally, they should finalize conversion, implementation and training plans.

Primary Test Includes:

Acceptance Testing – End users perform acceptance tests to assess the overall functionality and interoperability of an application.

End-to-End Testing – End users and system technicians perform end-to-end tests to assess the interoperability of an application and other system components such as databases, hardware, software, or communication devices.

Functional Testing – End users perform functional tests to assess the operability of a program against predefined requirements. Functional tests include black box tests, which assess the operational functionality of a feature against predefined expectations, or white box tests, which assess the functionality of a feature's code.

Integration Testing – End users and system technicians perform integration tests to assess the interfaces of integrated software components.

Parallel Testing – End users perform parallel tests to compare the output of a new application against a similar, often the original, application.

Regression Testing – End user Retest applications to assess functionality after programmers make code changes to previously tested application.

Stress Testing – Technicians perform stress tests to assess the maximum limits of an application.

String Testing – Programmers perform string tests to assesss the functionality of related code modules.

System Testing – Technicians perform system tests to assess the functionality of an entire system.

Unit Testing – Programmers perform unit tests to assess the functionality of small modules code.

White Box/ Black Box Testing

Black box testing is testing without knowledge of the internal working of the item being tested. For example, when black box testing is applied to software engineering, the tester would only know the "legal" inputs and what the expected outputs should be, but not how the program actually arrives at those outputs. It is because of this that black box testing can be considered testing with respect to the specifications, no other knowledge of the program is necessary. For this reason, the tester and the programmer can be independent of one another, avoiding programmer bias toward his own work. For this testing, test groups are often used, "Test groups are sometimes called professional idiots. People who are good at designing incorrect data."

Test Cases and Results

Table. 3.2

Test ID	Test Description	Test Data	Expected Result	Actual Result	Status	Figure NO
TC01	Leave all fields empty while registration	All fields are blank	Message should be displayed " Please enter fields"	The message is displayed correctly	Pass	3.35
TC02	Enter invalid email id while registration	Email-id: vishal.com	Message should be displayed "Incorrect email id."	The message is displayed correctly.	Pass	3.36
TC03	Username already exist	Username field contains username which is already registered	Message should be displayed "Username is already exists"	The message is displayed correctly.	Pass	3.37
TC04	Enter invalid username/password while logging in.	Invalid username/password	Message should be displayed "Invalid Login Credentials"	The message is displayed correctly.	Pass	3.38
TC05	Leave all fields empty while filling contact me form	All fields are blank	Message should be displayed " Please enter fields"	The message is displayed correctly.	Pass	3.39
TC06	Message field is too short while filling contact me form	Message field contains only 1 characters	Message should be displayed " Message is too short"	The message is displayed correctly.	Pass	3.40
TC07	User enter incorrect URL explicitly	Incorrect URL	Message should be displayed " Page not found"	The message is displayed correctly.	Pass	3.41

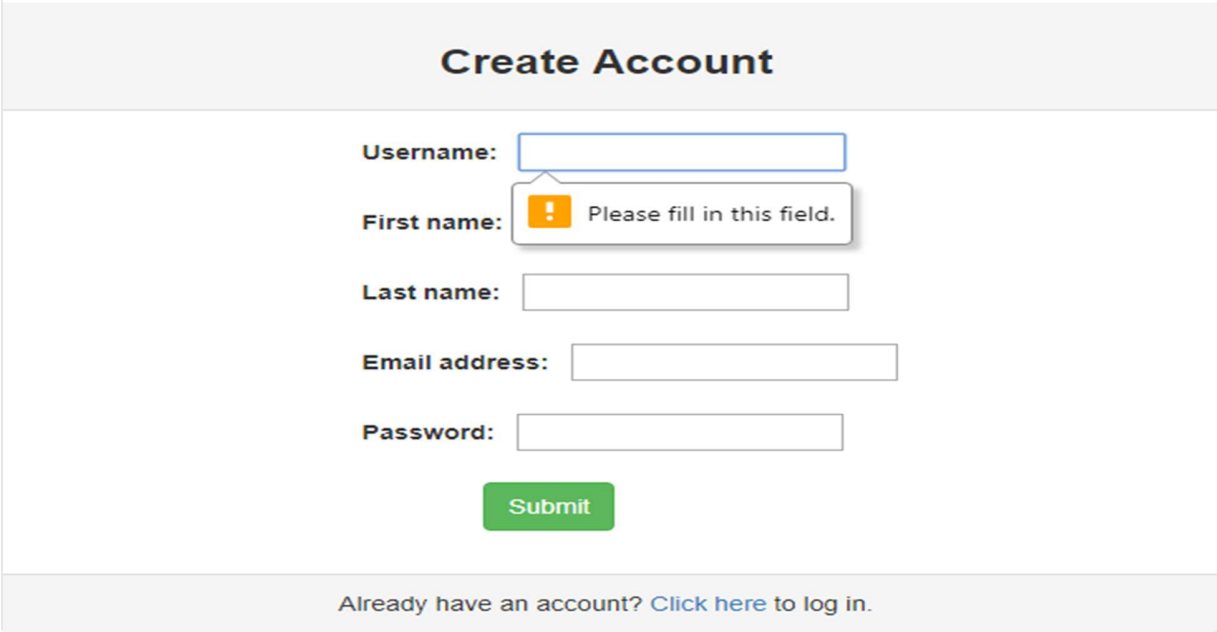

Figure. 3.35

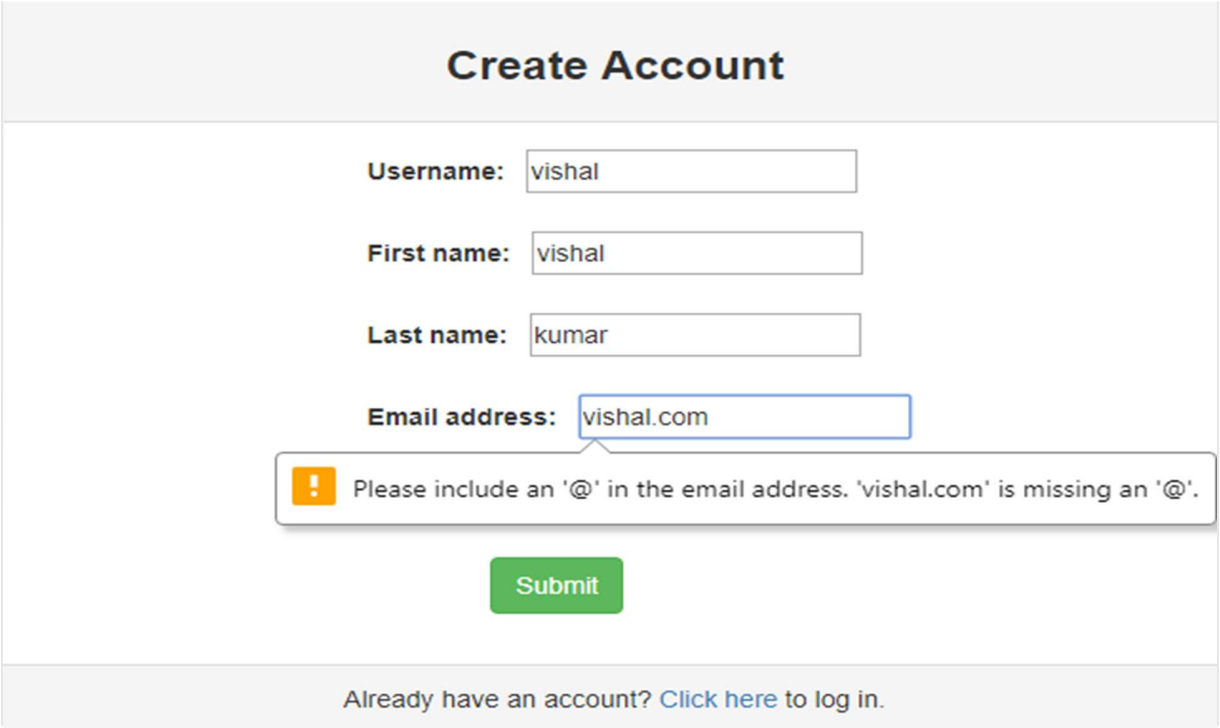

Figure. 3.36

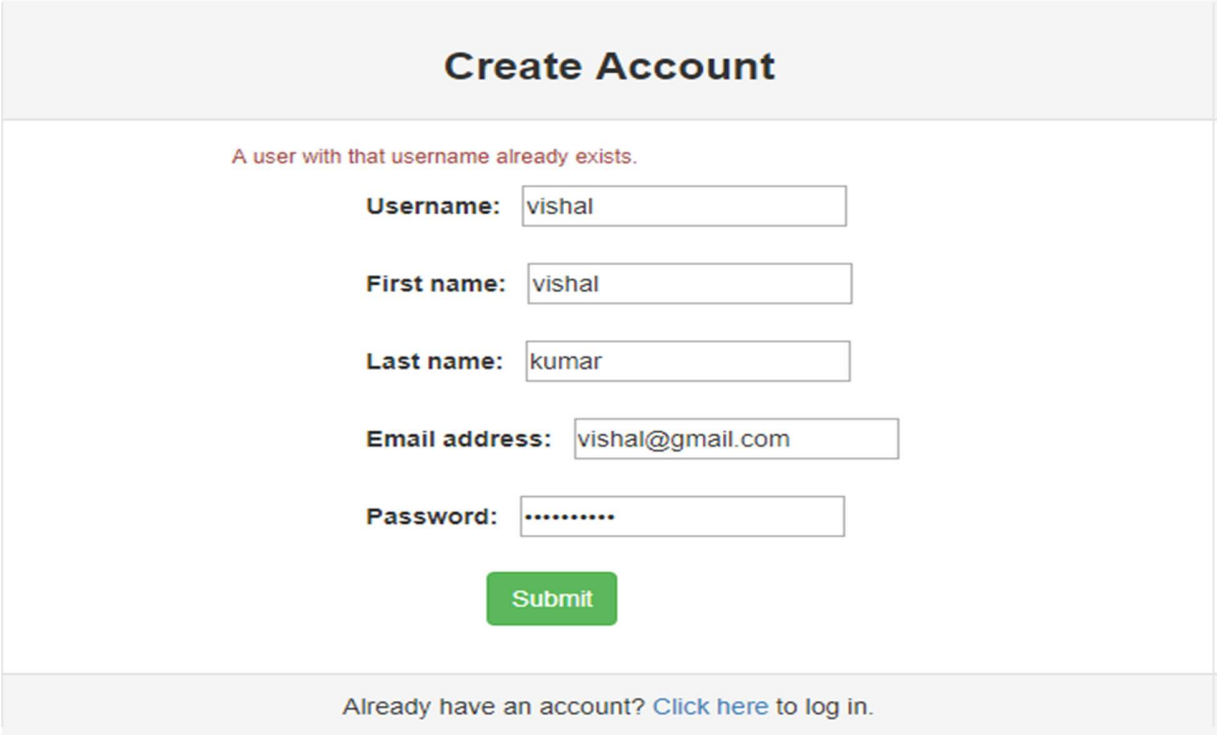

Figure. 3. 37

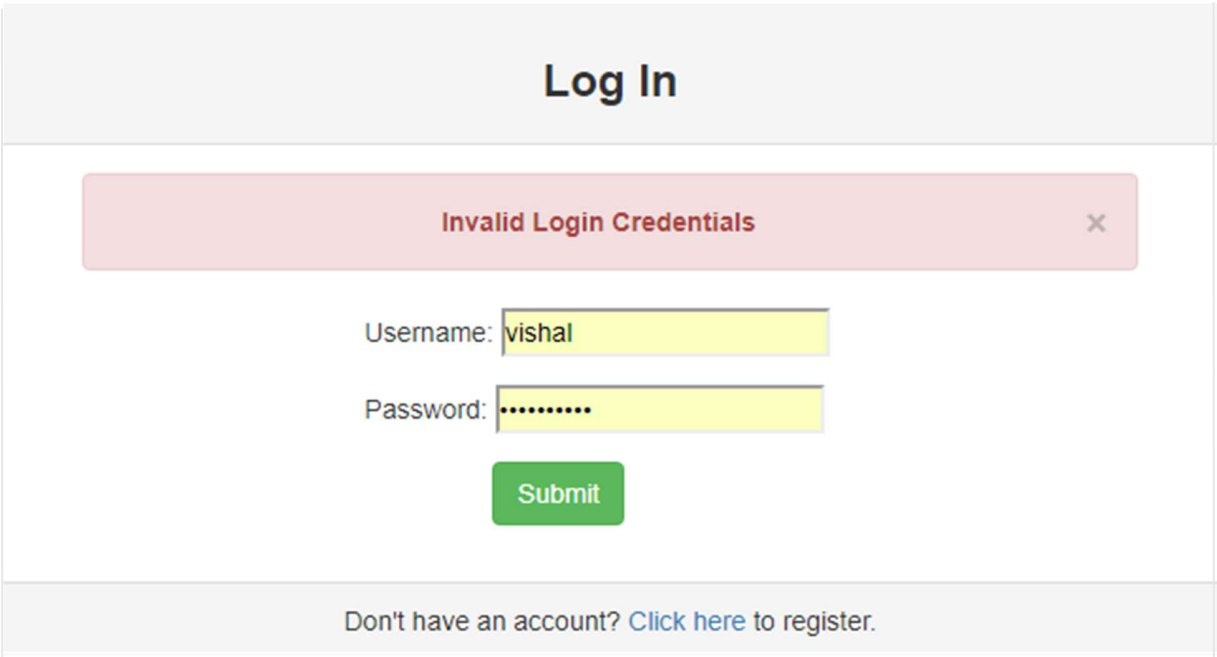

Figure. 3. 38

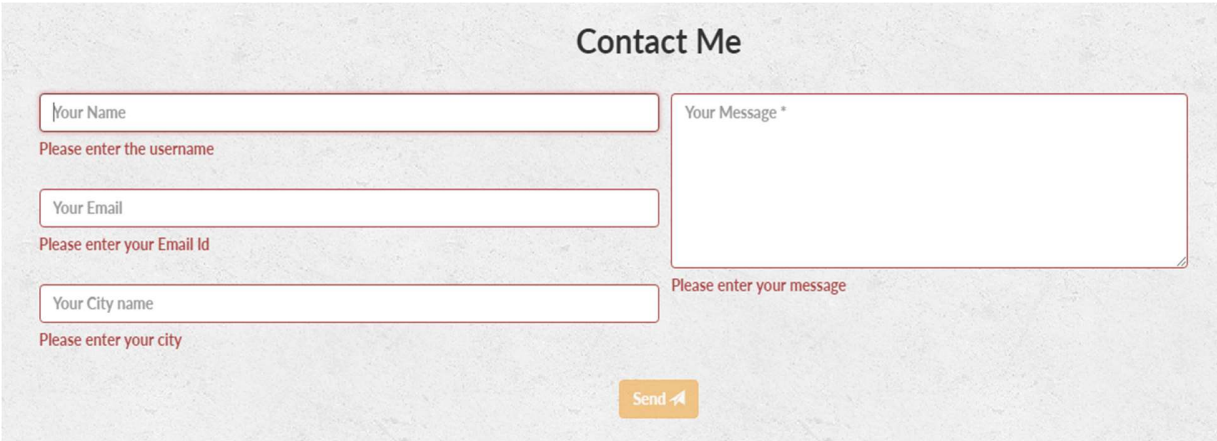

Figure. 3.39

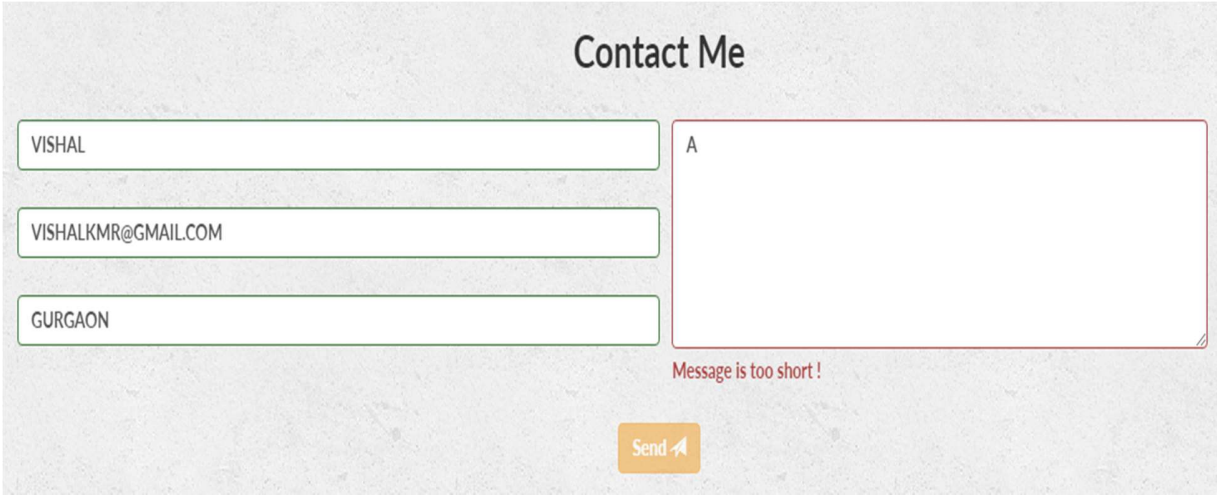

Figure. 3.40

Figure. 3.41

CHAPTER-4

RESULTS/OUTPUTS

INPUT TO THE SYSTEM

1. Registration Module
2. Login Module
3. Profile Module
4. Contact Module
5. Song & Album Module
6. Recommender Module

OUTPUT FROM THE SYSTEM

User Correlation

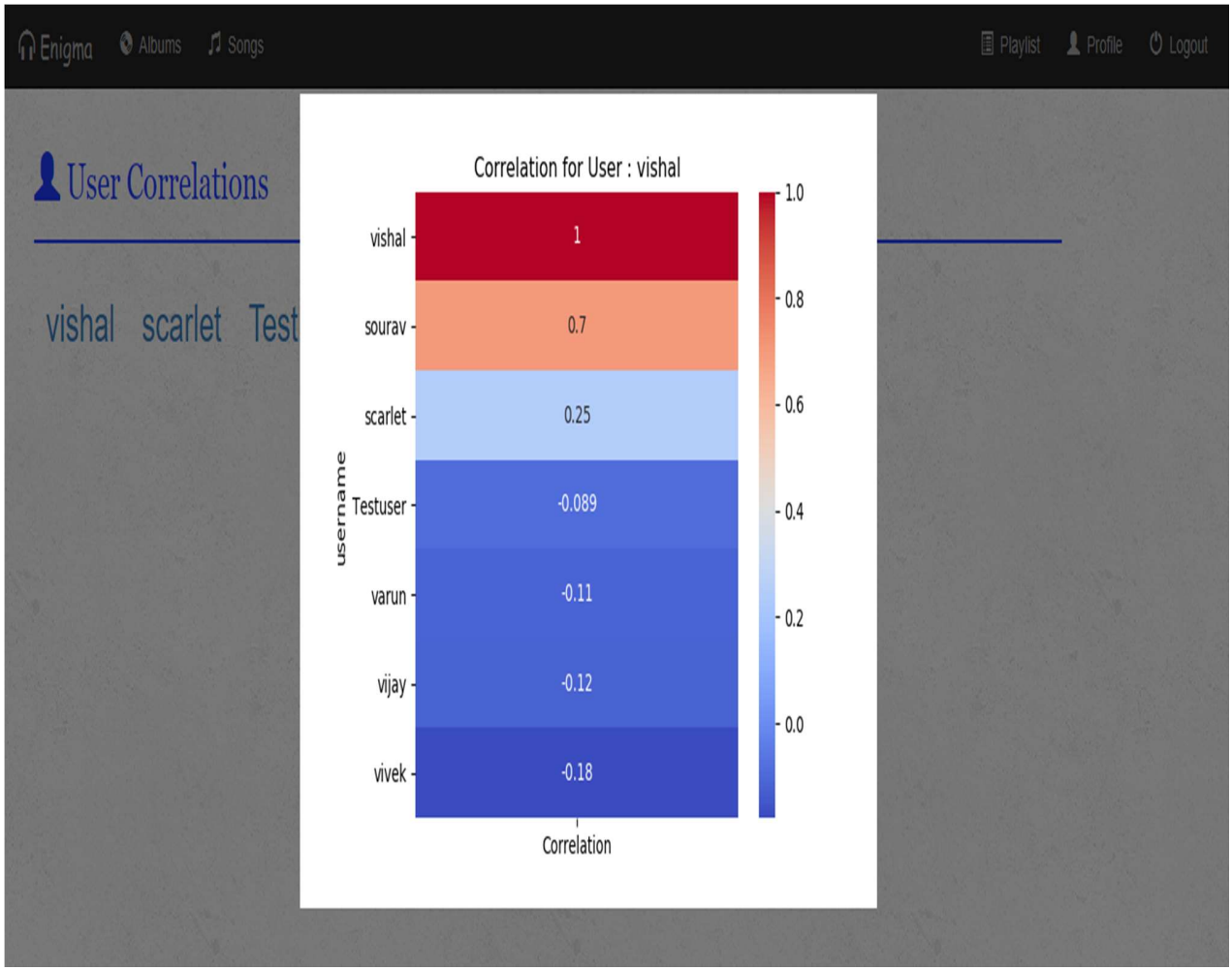

Figure. 4.1

Album Correlation

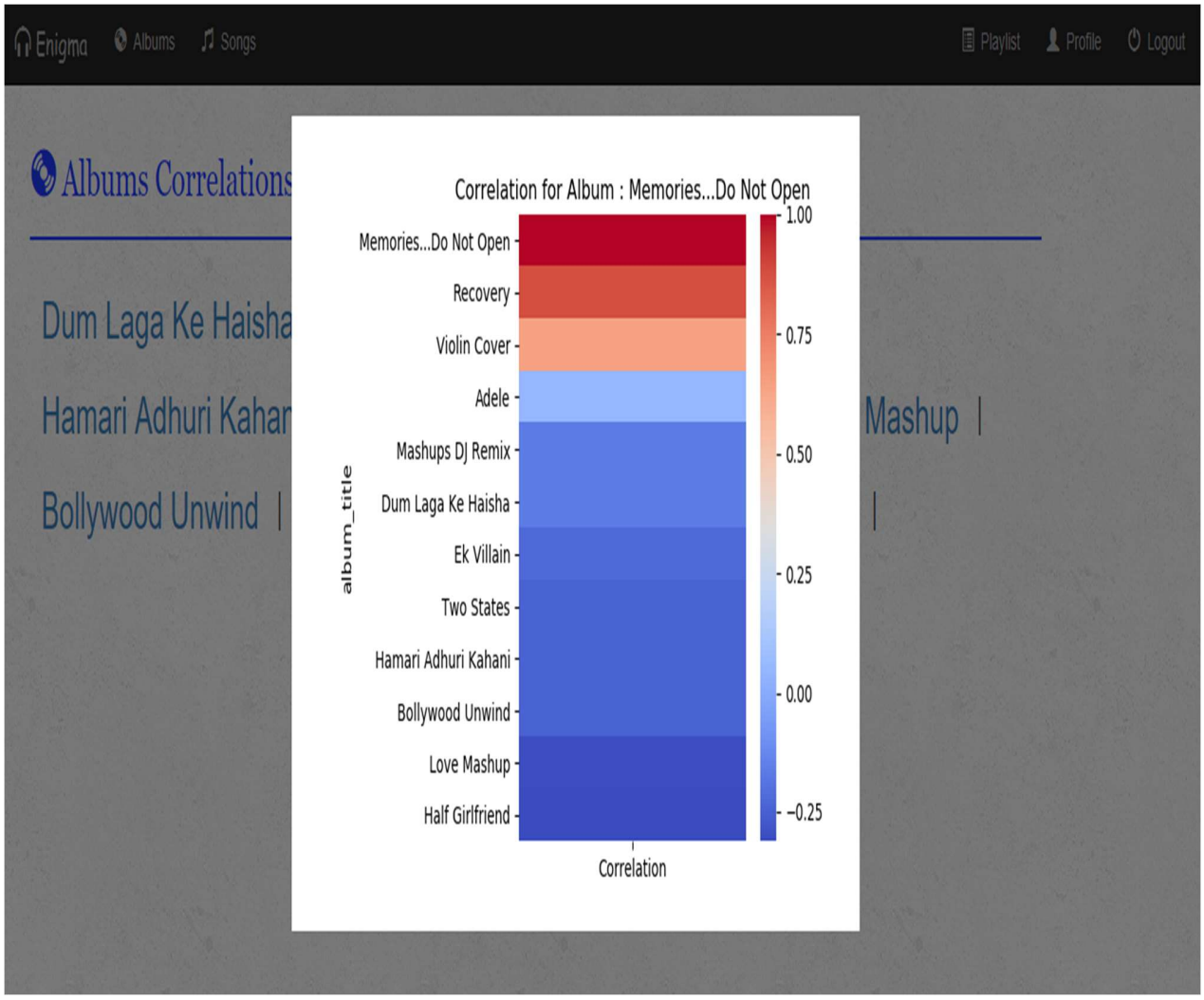

Figure. 4.2

Songs Correlation

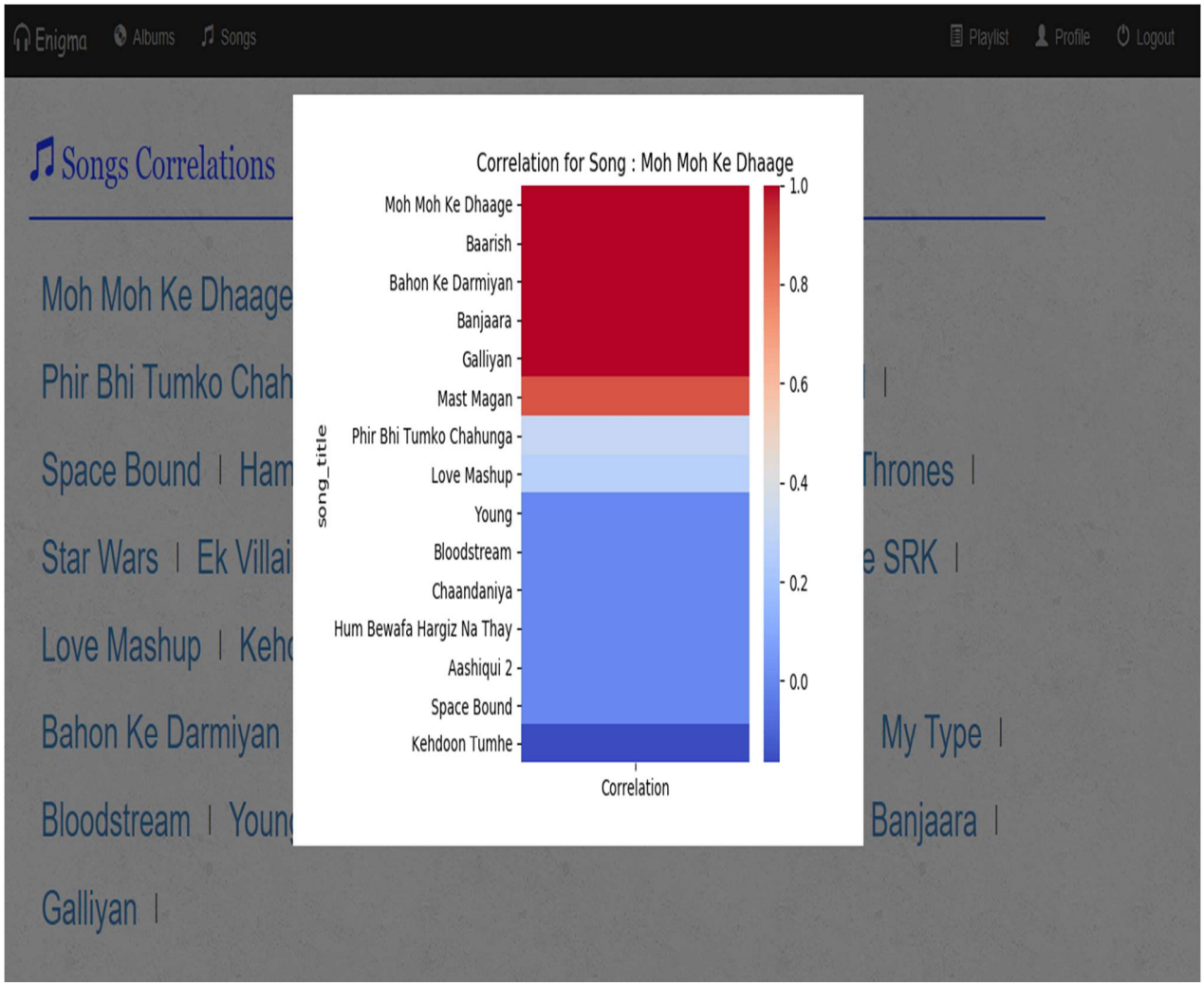

Figure. 4.3

CHAPTER-5

CONCLUSION

This is a project of Artificial Intelligence and Machine Learning course. We find it is very good as we got a chance to practice theories that we have learnt in the course, to do some implementation and to try to get a better understanding of a real artificial intelligence problem: Music Recommender System. There are many different approaches to this problem and we get to know some algorithms in detail and especially the models that we have explained in the project. By manipulating the data set, changing the learning set and testing set, changing some parameters of the problem and analyzing the result, we earn a lot practicing skills. We have faced a lot of problem in dealing with huge dataset, how to explore it in a better way and we also had difficulties in some programming details. However, with lot of efforts, we have overcame all of these. In terms of research, we still have a lot to do to make our studies a better one.

Music Recommender System is such a wide, open and complicated subject that we can take some initiatives and do a lot more tests in future. We also got to realize that building a recommender system is not a trivial task. The fact that large scale dataset makes it difficult in many aspects. Firstly, recommending relevant songs out large dataset for different users is not an easy task. Secondly, the metadata includes huge information and when exploring it, it is difficult to extract relevant features for song. Thirdly, technically speaking, processing such a huge dataset is memory and CPU intensive. All these difficulties due to the data and to the system itself make it more challenging and also more attractive. We hope that we will get other opportunities in the future to work in the domain of artificial intelligence. We are certain that we can do a better job.

CHAPTER-6

REFERENCES

1. Book: Recommender Systems An Introduction by Dietmar Jannach
2. Recommender Systems Books Webpage: www. recommenderbook.net
3. www.djangoproject.com
4. seaborn.pydata.org
5. sklearn.org
6. Coursera course on Recommender Systems, by University of Washington
7. Udemy course: Machine Learning A-Z
8. Udemy course: Machine-Learning-Bootcamp

CHAPTER-7

APPENDICES

Software changes with change in requirement and environment to maintain the utility adaptive maintenance technique will be followed and more process area will be considered under the scope of the system as and when occurred.

- The software Scope is limited by user clarity about requirement specification in the software.
- The software is partial implementation of computer based information system.
- At present The Software is partially dealing with the schedule of classes held in the institute it.
- Software is dealing with employee information partially.