MACHINE LEARNING BASED MUSIC GENRE CLASSIFICATION ON SPOTIFY DATA

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE

MAJOR PROJECT

BY

G. VARUN

G. RAJU

N. SUSHMA

A. SHARVANI

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Overview

Music is like a mirror, and it tells people a lot about who you are and what you care about, whether you like it or not. We love to say “you are what you stream”.

Companies now-a-days use music classification, either to be able to place recommendations to their customers (such as Spotify, Soundcloud) or simply as a product (for example Shazam). Determining music genres is the first step in that direction. Machine Learning techniques have proved to be quite successful in extracting trends and patterns from a large pool of data. The same principles are applied in Music Analysis also.

1.2 Purpose

Throughout the course we mainly focused around computer vision tasks and a little bit of NLP. I decided to reach out a new field and use the machine learning techniques l learned on the field of sound processing. This paper discusses the task of classifying the music genre of a sound sample.

2. LITERATURE SURVEY

2.1 Existing problem

You'll be able to understand the problem to classify if it is regression or a classification kind of problem. You will be able to know how to pre-process/clean the data using different data preprocessing techniques. You will able to analyze or get insights into data through visualization. Applying different algorithms according to the dataset and based on visualization. You will able to know how to find the accuracy of the model. You will be able to know how to build a web application using the Flask framework.

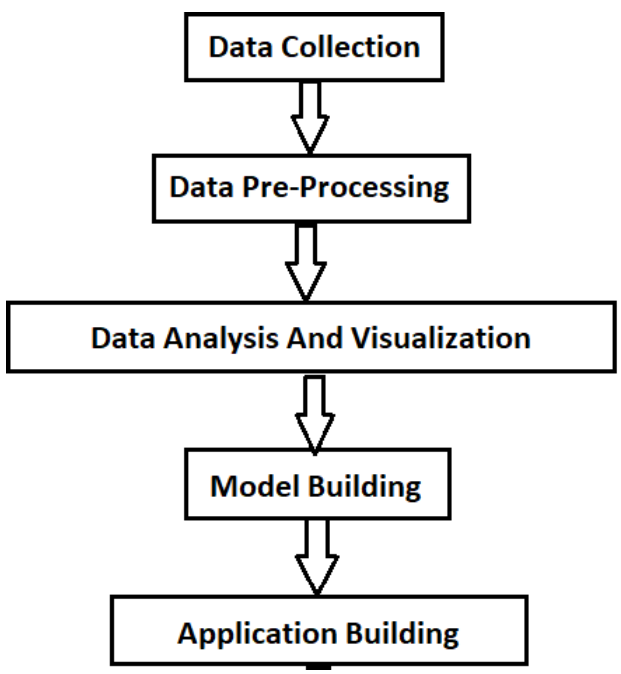
2.2 Proposed solution

By doing this project we classify if it is a regression or a classification kind of problem. And also able to analyze or get insights into data. In this project we are using the different algorithms according to the dataset and based on visualization.

3.THEORETIC ANALYSIS

3.1 Block Diagram

A **block diagram** is a drawing illustration of a system whose major parts or components are represented by blocks. These blocks are joined by lines to display the relationship between subsequent blocks. We use block diagrams to visualize the functional view of a system. It uses blocks connected with lines to represent components of a system. With a block diagram, you can easily illustrate the essential parts of a software design or engineering system and depict the data flow in a process flow chart.



3.2 UML Diagrams

UML, which stands for Unified Modeling Language, is a way to visually represent the architecture, design, and implementation of complex software systems. When you're writing code, there are thousands of lines in an application, and it's difficult to keep track of the relationships and hierarchies within a software system. UML diagrams divide that software system into components and subcomponents. The UML diagrams are categorized into structural diagrams, behavioral diagrams, and also interaction overview diagrams.

1.Use Case Diagram: It represents the functionality of a system by utilizing actors and use cases. It encapsulates the functional requirement of a system and its association with actors. It portrays the use case view of a system.

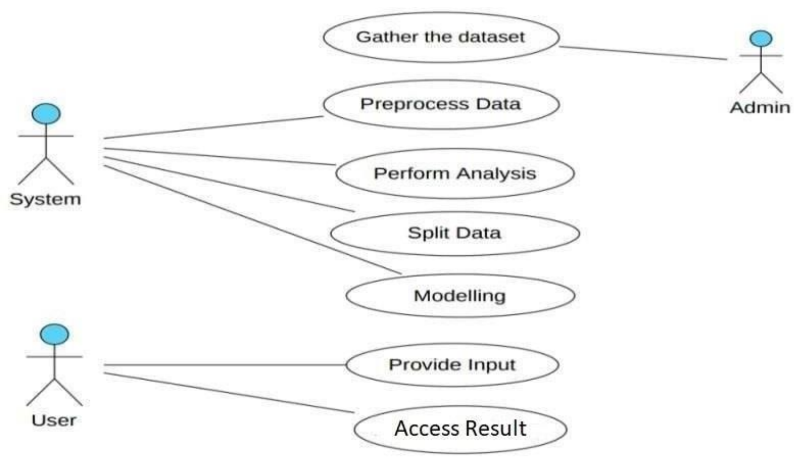


Fig.1. Use Case Diagram

2.Sequence Diagram: It shows the interactions between the objects in terms of messages exchanged over time. It delineates in what order and how the object functions are in a system.

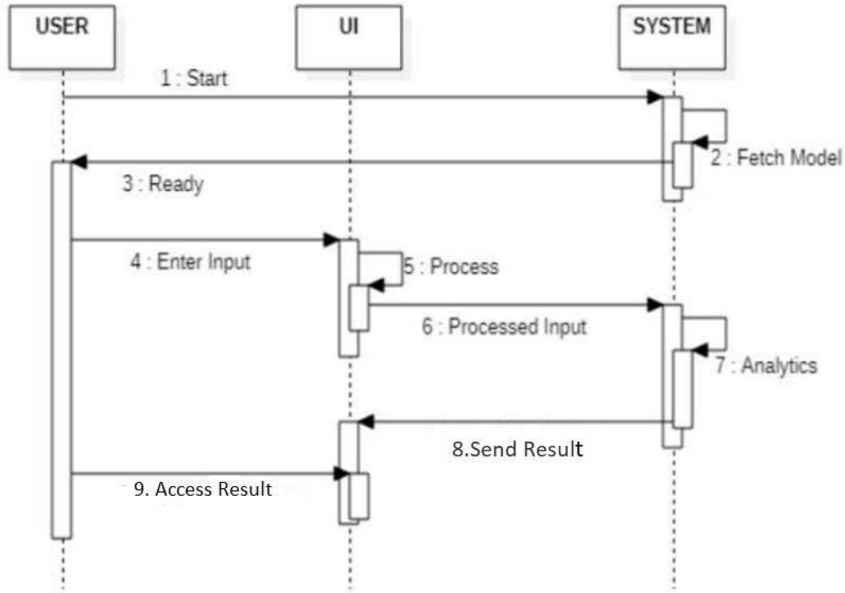


Fig.2. Sequence Diagram

3.Activity Diagram: It models the flow of control from one activity to the other. With the help of an activity diagram, we can model sequential and concurrent activities. It visually depicts the workflow as well as what causes an event to occur.

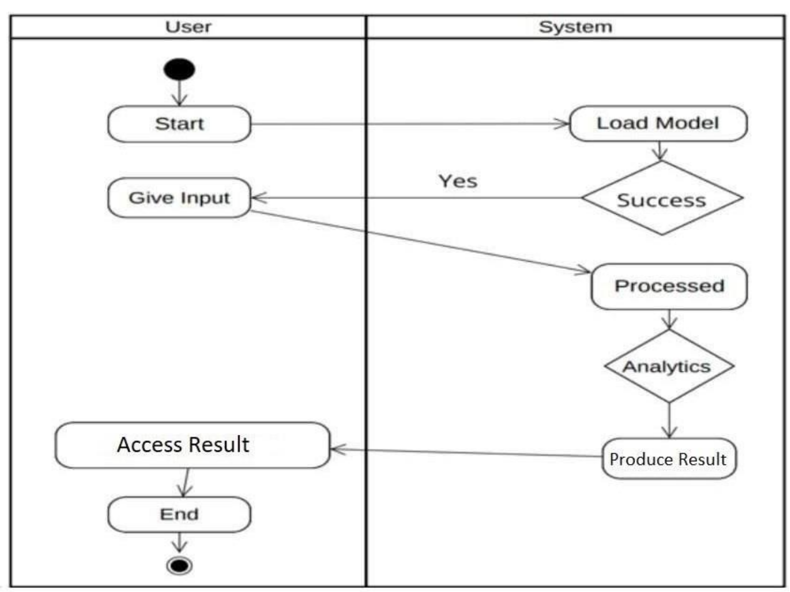


Fig.3. Activity Diagram

3.2 Modules:

PANDAS:

Python Pandas module is basically an open-source Python module. It has a wide scope of use in the field of computing, data analysis, statistics, etc.

Pandas module uses the basic functionalities of the NumPy module.

Label Encoder:

The LabelEncoder module in Python's sklearn is used to encode the target labels into categorical integers (e.g. 0, 1, 2, ...).

One-hot Encoder:

One-hot encoding is used to convert categorical variables into a format that can be readily used by machine learning algorithms. The basic idea of one-hot encoding is to create new variables that take on values 0 and 1 to represent the original categorical values.

Python Seaborn:

Python Seaborn module serves the purpose of Data Visualization at an ease with higher efficiency. In order to represent the variations in a huge data set, data visualization is considered as the best way to depict and analyze the data.

Joblib:

joblib is basically a wrapper library that uses other libraries for running code in parallel. It also lets us choose between multi-threading and multi-processing. joblib is ideal for a situation where you have loops and each iteration through loop calls some function that can take time to complete.

3.3 Hardware/Software Designing

Recommended System Requirements:

Processors: Intel® Core™ i5 processor 4300M at 2.60 GHz or 2.59 GHz (1 socket, 2 cores, 2 threads per core), 8 GB of DRAMIntel® Xeon® processor E5-2698 v3 at 2.30 GHz (2 sockets, 16 cores each, 1 thread per core), 64 GB of DRAMIntel® Xeon Phi™ processor 7210 at 1.30 GHz (1 socket, 64 cores, 4 threads per core), 32 GB of DRAM, 16 GB of MCDRAM (flat mode enabled).

- Disk space: 2 to 3 GB.
- Operating systems: Windows® 10, macOS*, and Linux*.

Minimum System Requirements:

- Processors: Intel Atom® processor or Intel® Core™ i3 processor.
- Disk space: 1 GB.
- Operating systems: Windows* 7 or later, macOS, and Linux.
- Python* versions: 3.9.

Software requirements:

Anaconda navigator: Anaconda is an open-source distribution for python and R. It is used for data science, machine learning, deep learning, etc. With the availability of more than 300 libraries for data science, it becomes fairly optimal for any programmer to work on anaconda for data science.

Spyder:

Spyder is a free and open source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

Jupyter:

Jupyter is the latest web-based interactive development environment for notebooks, code and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.

4. EXPERIMENTAL ANALYSIS

Milestone 1: Data Collection

ML depends heavily on data, without data, a machine can't learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

You can collect datasets from different open sources like kaggle.com, data.gov; UCI machine learning repository etc. The dataset used for this project was obtained from Kaggle.

Milestone 2: Data Pre-processing

Data Pre-processing includes the following main task

- Importing the libraries.
- Importing the dataset.
- Analyse the data.
- Taking care of Missing Data.
- Data Visualisation.
- Splitting Data into Train and Test.

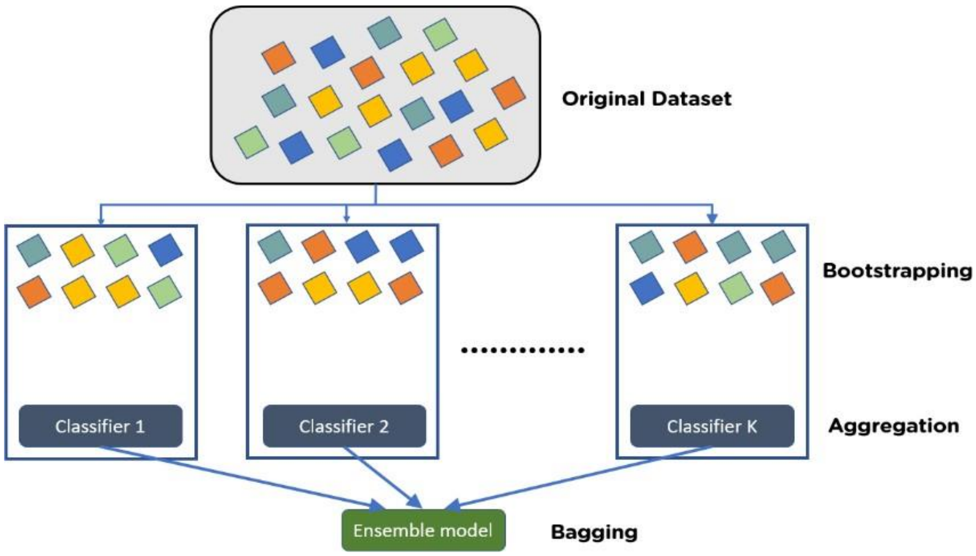
Milestone 3: Model Building

The model building process involves setting up ways of collecting data, understanding and paying attention to what is important in the data to answer the questions you are asking, finding a statistical, mathematical or a simulation model to gain understanding and make predictions. Model Building Includes:

- Import the model building libraries.
- Initialising the model.
- Training the model.
- Model Evaluation.
- Save the Model.

Milestone 4: Building Bagging Classifier

Bagging, also known as Bootstrap aggregating, is **an ensemble learning technique that helps to improve the performance and accuracy of machine learning algorithms**. It is used to deal with bias-variance trade-offs and reduces the variance of a prediction model.



- Consider there are n observations and m features in the training set. You need to select a random sample from the training dataset without replacement.
- A subset of m features is chosen randomly to create a model using sample observations.
- The feature offering the best split out of the lot is used to split the nodes.
- The tree is grown, so you have the best root nodes.

Milestone 5: Application Building

- build an HTML Page.
- Build the python code Flask application
- Run the application in local browser.
- Show casting the prediction on UI.

5. TECHNICAL ARCHITECTURE

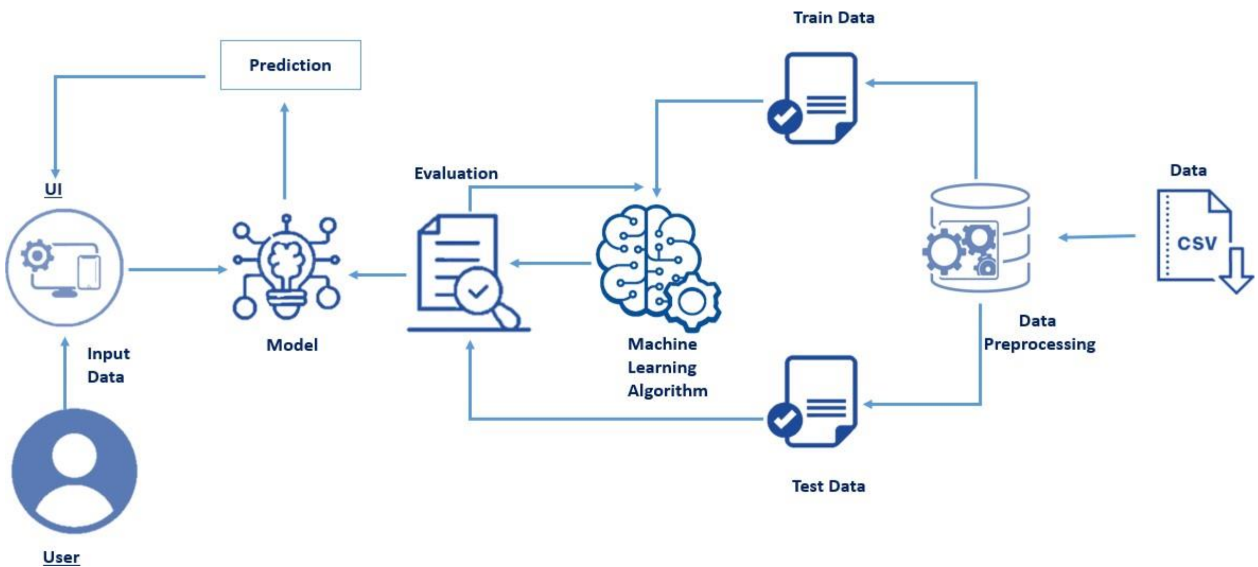
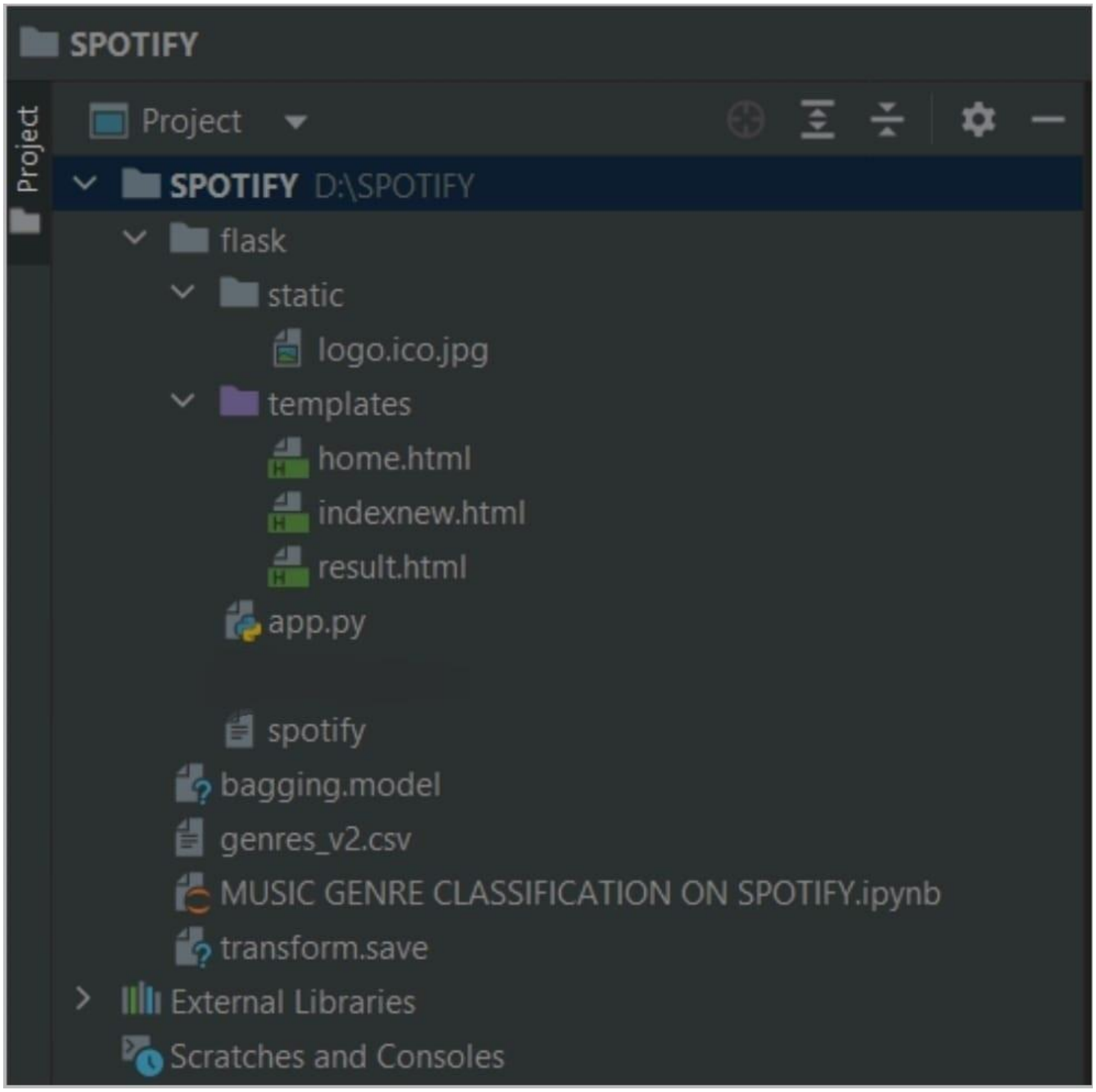


Fig.4 Technical Architecture

6. PROJECT STRUCTURE



7.ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

1. The main thing to identify and divide the audio into different features is amplitude and frequency that changes within a short span of time.
2. We can visualize the audio frequency wave of amplitude and frequency with respect to time in form of a wave plot that can be easily plotted using librosa.
3. MFCCs total provides 39 features related to frequency and amplitude. In that 12 parameters are related to the amplitude of frequencies. It means it provides us with enough frequency channels to analyze audio and this is the reason MFCCs are used everywhere for feature extraction in audios.
4. The key working of MFCC is to remove vocal excitation (pitch information) by dividing audio into frames, make extracted features independent, adjust the loudness, and frequency of sound according to humans, and capture the context.

DISADVANTAGES:

1. Genres are very helpful for music discovery. They bond fans and listeners and facilitate shared experiences. But as anything with a boundary or classification, it can eventually end up being manipulated, create division and a lot of unconscious choices.
2. The model is insufficiently robust to apply the training results to previously unknown musical data
3. Some audio characteristics derived from the raw audio signals were left out.

8. APPLICATIONS

Mall:

Music is played continuously in the malls, and selection of right music to be played is hectic as well as time consuming work. So here, our system helps to choose the song according to any occasion or event.

Restaurant:

In a restaurant, choosing the right music is an important task when it comes to various occasions as per customer's demand; our system will help to choose a particular genre song for the same.

Airport:

Music is played in the airports for the entertainment of people as they wait for hours due to various reasons, so our system will help to choose the song as per the requirements.

9. CONCLUSION

Automatic Genre classification is a difficult and problematic task that none the less has important value in terms of both pure research and commercial application. Continuing research in automatic genre classification has much to offer, as does parallel research involving other aspects of musical similarity. Automatic genre classification performance appears to have fallen into a local maximum recently, and serious modifications to the approaches used are needed in order to realize further improvements.

10. FUTURE SCOPE

If the limitations set by us were not present, these models can definitely perform a way better as it were. Forfurther improvement we can also use the python speech recognition package to read and analyze the song lyrics and include another layer of classification according to lyrics.

This will also greatly impact the accuracy. The song can also be decomposed and the instruments used in the song can be isolated and recognized separately giving us another layer of classification.

We can also use a pipeline of basic machine learning classification algorithms to serially input data and analyze it while giving its output to the next algorithm in the pipeline. The algorithms used could be KNN, Naive Bayes, SVM, XGBoost, Decision tree, etc. This will also ensure excellent accuracy.

11. CODE SNIPPETS

11.1 MODEL CODE :

Data Preprocessing

1. Handling the null values.
2. Handling the categorial values if any.
3. Normalize the data if required
4. Identify the dependent and independent variables.
5. Split the dataset into train and test sets.

```
import numpy as np
import pandas as pd
import os
import matplotlib.pyplot as plt
```

```
data=pd.read_csv("C:/Users/varun/OneDrive/Desktop/spotify/genres_v2.csv")
data.head()
```

```
C:\Users\varun\AppData\Local\Temp\ipykernel_19388\96829336.py:1: DtypeWarning: Columns (19) have mixed types. Specify dtype opt
ion on import or set low_memory=False.
  data=pd.read_csv("C:/Users/varun/OneDrive/Desktop/spotify/genres_v2.csv")
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	id	
0	0.831	0.814	2	-7.364	1	0.4200	0.0598	0.013400	0.0556	0.3890	...	2Vc6NJ9PW9gD9q343XFRKx	spotify:track:2Vc6N
1	0.719	0.493	8	-7.230	1	0.0794	0.4010	0.000000	0.1180	0.1240	...	7pgJBLVz5VmnL7uGHmRj6p	spotify:track:7pgJ
2	0.850	0.893	5	-4.783	1	0.0623	0.0138	0.000004	0.3720	0.0391	...	0vSWgAlfpye0WCGeNmuNhy	spotify:track:0vSW
3	0.476	0.781	0	-4.710	1	0.1030	0.0237	0.000000	0.1140	0.1750	...	0VSXnJqQkwuH2ei1nOQ1nu	spotify:track:0VSX
4	0.798	0.624	2	-7.668	1	0.2930	0.2170	0.000000	0.1660	0.5910	...	4jCeguq9rMTlbMmPHuO7S3	spotify:track:4jCe

Figure 1: .ipynb code describing importing libraries and displaying the few rows from the data set.

```
data.tail(1562)
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	id	
40743	0.265	0.970	1	-3.676	0	0.1130	0.005620	0.145000	0.2910	0.149	...	7Am832dA5akvZPZ7Ptjw9R	spotify:tra
40744	0.510	0.927	1	-5.377	1	0.0483	0.000557	0.007590	0.3790	0.337	...	2HgoDy6BkfGOAdjXQg1Mn2	spotify:tra
40745	0.481	0.969	1	-2.969	1	0.0479	0.011200	0.973000	0.2160	0.429	...	2Dk9uOLRqa9teix9utL6GC	spotify:t
40746	0.474	0.899	6	-5.748	1	0.0552	0.000399	0.282000	0.3450	0.315	...	22sLNUFrEUP8mPOJ5XUAMZ	spotify:track
40747	0.491	0.914	5	-4.468	0	0.0413	0.000367	0.000110	0.1450	0.281	...	30XY1oyrObfjv3OHUud5R2	spotify:tr
...	...	...	...	...	...	...	...	...	...	...	...	...	
42300	0.528	0.693	4	-5.148	1	0.0304	0.031500	0.000345	0.1210	0.394	...	46bXU7Sgj7104ZoXxzz9tM	spotify:tr
42301	0.517	0.768	0	-7.922	0	0.0479	0.022500	0.000018	0.2050	0.383	...	0he2ViGMUO3ajKTxLOfWVT	spotify:tra
42302	0.361	0.821	8	-3.102	1	0.0505	0.026000	0.000242	0.3850	0.124	...	72DAt9Lbpy9EUS29OzQLob	spotify:tra
42303	0.477	0.921	6	-4.777	0	0.0392	0.000551	0.029600	0.0575	0.488	...	6HXgExFVuE1c3cq9QjFCcU	spotify:tra
42304	0.529	0.945	9	-5.862	1	0.0615	0.001890	0.000055	0.4140	0.134	...	6MAAMZImxcvYhRnxDLTufD	spotify:tra

Figure 2: .ipynb code for displaying last rows of the dataset by using data.tail() method

```
data.columns
```

```
Index(['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
       'type', 'id', 'uri', 'track_href', 'analysis_url', 'duration_ms',
       'time_signature', 'genre', 'song_name', 'Unnamed: 0', 'title'],
      dtype='object')
```

```
data.iloc[0,:]
```

```
danceability                                                 0.831
energy                                                       0.814
key                                                              2
loudness                                                    -7.364
mode                                                             1
speechiness                                                   0.42
acousticness                                                0.0598
instrumentalness                                            0.0134
liveness                                                    0.0556
valence                                                      0.389
tempo                                                      156.985
type                                                audio_features
id                                          2Vc6NJ9PW9gD9q343XFRKx
uri                           spotify:track:2Vc6NJ9PW9gD9q343XFRKx
track_href        https://api.spotify.com/v1/tracks/2Vc6NJ9PW9gD...
analysis_url      https://api.spotify.com/v1/audio-analysis/2Vc6...
duration_ms                                                 124539
time_signature                                                   4
genre                                                    Dark Trap
song_name                                      Mercury: Retrograde
Unnamed: 0                                                     NaN
title                                                          NaN
Name: 0, dtype: object
```

Figure 3: .ipynb code for displaying the columns and specific value in a row by using data.columns and data.iloc[] methods.

```
data=data.drop(['title','Unnamed: 0', 'id', 'uri', 'track_href', 'analysis_url','type','song_name'],axis=1)
print(data.columns)
data.head()
```

```
Index(['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
       'duration_ms', 'time_signature', 'genre'],
      dtype='object')
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature	genre
0	0.831	0.814	2	-7.364	1	0.4200	0.0598	0.013400	0.0556	0.3890	156.985	124539	4	Dark Trap
1	0.719	0.493	8	-7.230	1	0.0794	0.4010	0.000000	0.1180	0.1240	115.080	224427	4	Dark Trap
2	0.850	0.893	5	-4.783	1	0.0623	0.0138	0.000004	0.3720	0.0391	218.050	98821	4	Dark Trap
3	0.476	0.781	0	-4.710	1	0.1030	0.0237	0.000000	0.1140	0.1750	186.948	123661	3	Dark Trap
4	0.798	0.624	2	-7.668	1	0.2930	0.2170	0.000000	0.1660	0.5910	147.988	123298	4	Dark Trap

Figure 4: .ipynb code for dropping the unnecessary columns and displaying the remaining columns in the dataset by using data.drop() method.

```
data.isnull().sum()
```

```
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_ms         0
time_signature      0
genre               0
dtype: int64
```

```
data.shape
```

```
(42305, 14)
```

Figure 5: .ipynb code describes returning the number of missing values in the dataset and shape of data set.

```python
#identifying all the numeric columns
numeric = data._get_numeric_data()
genre = data['genre']
print(numeric.head())
print("Numeric columns: ",end=" ")
print(numeric.columns)
print(len(numeric.columns))


numeric.describe()
```

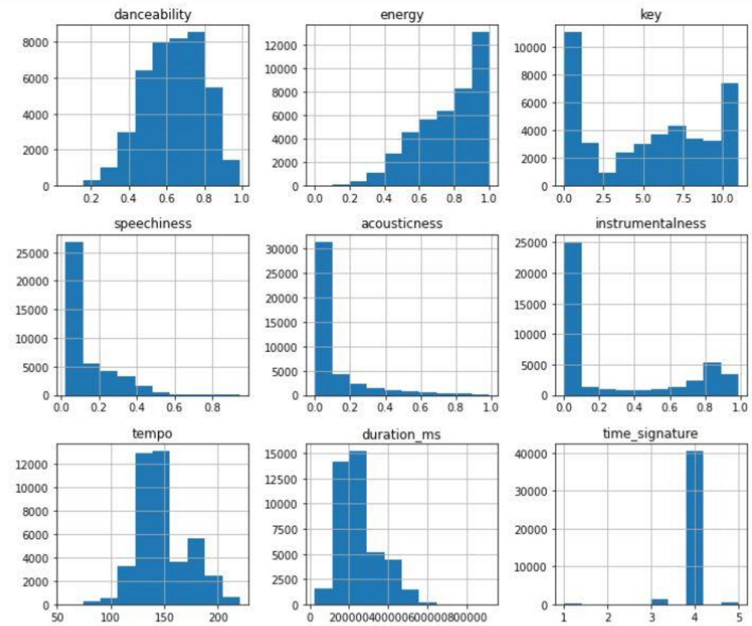
```
   danceability  energy  key  loudness  mode  speechiness  acousticness  \
0         0.831   0.814    2    -7.364     1       0.4200        0.0598
1         0.719   0.493    8    -7.230     1       0.0794        0.4010
2         0.850   0.893    5    -4.783     1       0.0623        0.0138
3         0.476   0.781    0    -4.710     1       0.1030        0.0237
4         0.798   0.624    2    -7.668     1       0.2930        0.2170

   instrumentalness  liveness  valence    tempo  duration_ms  time_signature
0          0.013400    0.0556   0.3890  156.985       124539               4
1          0.000000    0.1180   0.1240  115.080       224427               4
2          0.000004    0.3720   0.0391  218.050        98821               4
3          0.000000    0.1140   0.1750  186.948       123661               3
4          0.000000    0.1660   0.5910  147.988       123298               4
Numeric columns:  Index(['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo',
       'duration_ms', 'time_signature'],
      dtype='object')
13
```

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	
count	42305.000000	42305.000000	42305.000000	42305.000000	42305.000000	42305.000000	42305.000000	42305.000000	42305.000000	42305.000000	4230
mean	0.639364	0.762516	5.370240	-6.465442	0.549462	0.136561	0.096160	0.283048	0.214079	0.357101	14
std	0.156617	0.183823	3.666145	2.941165	0.497553	0.126168	0.170827	0.370791	0.175576	0.233200	2
min	0.065100	0.000243	0.000000	-33.357000	0.000000	0.022700	0.000001	0.000000	0.010700	0.018700	5
25%	0.524000	0.632000	1.000000	-8.161000	0.000000	0.049100	0.001730	0.000000	0.099600	0.161000	12
50%	0.646000	0.803000	6.000000	-6.234000	1.000000	0.075500	0.016400	0.005940	0.135000	0.322000	14
75%	0.766000	0.923000	9.000000	-4.513000	1.000000	0.193000	0.107000	0.722000	0.294000	0.522000	16
max	0.988000	1.000000	11.000000	3.148000	1.000000	0.946000	0.988000	0.989000	0.988000	0.988000	22

Figure 6: .ipynb code for identifying all the numeric columns in the data set.

```
#histogram of numerical columns in the dataset
num_hist = numeric.hist(layout=(3,5),figsize=(20,10))
plt.show()
```




```
#Identifying no of unique genre
np.unique(genre)
```

```
array(['Dark Trap', 'Emo', 'Hiphop', 'Pop', 'Rap', 'RnB', 'Trap Metal',
       'Underground Rap', 'dnb', 'hardstyle', 'psytrance', 'techhouse',
       'techno', 'trance', 'trap'], dtype=object)
```

Figure 7: .ipynb code for histogram of numerical columns and identifying unique genre in the dataset.

```
grouped_genre = data.groupby('genre')

for col in numeric.columns:
    fig,ax = plt.subplots()

    for i, d in grouped_genre:
        d[col].hist(alpha=0.4, ax=ax, label=i,figsize=(10,4))
        ax.set_title(col)

    ax.legend()
    plt.show()
```

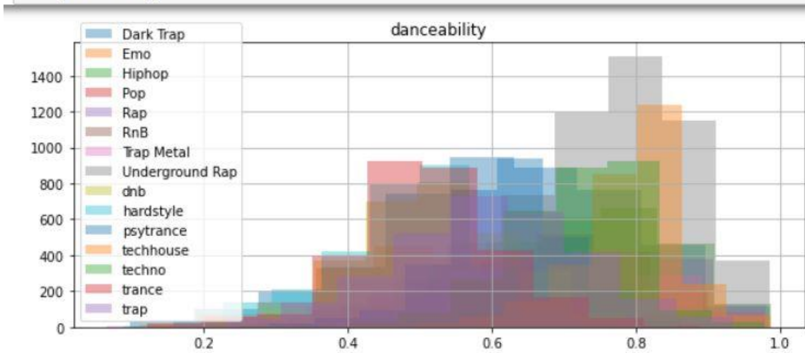


Figure 8: .ipynb code for grouping the data according to categories and applying function on it.

```
#Standard deviation of different features in a Genre
grouped_genre.std()
```

genre	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
Dark Trap	0.161002	0.180810	3.638048	3.156933	0.498722	0.123217	0.214746	0.366365	0.144774	0.206191	27.201544	60377.140798
Emo	0.126014	0.220404	3.477168	2.598227	0.464369	0.075109	0.260326	0.077927	0.145681	0.202451	27.545931	43426.023963
Hiphop	0.142183	0.162416	3.679533	2.827193	0.499312	0.131501	0.219940	0.093075	0.162457	0.222957	28.188804	60884.353006
Pop	0.119175	0.155379	3.644698	1.947846	0.498153	0.081074	0.198390	0.085986	0.140379	0.215627	32.360009	35368.133282
Rap	0.127670	0.137444	3.705862	2.353116	0.499643	0.134830	0.178520	0.058052	0.132247	0.215302	29.106285	59281.310692
RnB	0.140207	0.164788	3.638515	2.518220	0.499084	0.112993	0.231904	0.060419	0.133160	0.221235	29.820987	50866.054759
Trap Metal	0.169564	0.174147	3.748826	3.038965	0.450189	0.159220	0.166583	0.177845	0.186634	0.215788	26.672863	45075.031628
Underground Rap	0.128180	0.155994	3.777848	2.755624	0.486962	0.144794	0.186051	0.116380	0.148016	0.224521	26.671957	54752.469716
dnb	0.106605	0.097925	3.589637	1.946063	0.497165	0.076611	0.051514	0.359274	0.167220	0.185296	1.141970	51728.231272
hardstyle	0.099230	0.082729	3.534666	1.589293	0.480284	0.090424	0.070447	0.253083	0.189537	0.157019	1.917755	58437.289366
psytrance	0.086189	0.093983	3.502445	1.585368	0.490295	0.027955	0.023667	0.140714	0.234006	0.181496	4.450475	68718.084176
techhouse	0.073436	0.131920	3.782393	1.899141	0.495520	0.041437	0.040741	0.326697	0.142550	0.233905	1.655647	91206.405796
techno	0.088053	0.141080	3.570704	2.383438	0.493223	0.036655	0.097297	0.133036	0.128944	0.169857	3.852044	69792.810230
trance	0.102201	0.107369	3.486528	2.210131	0.496802	0.052799	0.047673	0.372017	0.209879	0.153906	4.437328	103359.717162
trap	0.128669	0.098066	3.801735	1.946665	0.489228	0.146996	0.060583	0.273226	0.201278	0.177189	4.760100	41354.219861

```
#Min values of different features in a Genre
grouped_genre.min()
```

genre	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signatu
Dark Trap	0.0979	0.000243	0	-25.222	0	0.0242	0.000001	0.000000	0.0307	0.0235	75.418	42133	
Emo	0.1110	0.014800	0	-32.929	0	0.0232	0.000001	0.000000	0.0210	0.0358	87.018	50720	
Hiphop	0.1970	0.027900	0	-24.694	0	0.0227	0.000017	0.000000	0.0219	0.0352	95.622	38333	
Pop	0.2090	0.173000	0	-16.423	0	0.0232	0.000068	0.000000	0.0215	0.0383	106.960	121143	
Rap	0.2410	0.144000	0	-19.720	0	0.0271	0.000151	0.000000	0.0221	0.0362	57.967	77500	
RnB	0.1910	0.060900	0	-29.478	0	0.0239	0.000081	0.000000	0.0235	0.0338	91.560	62213	
Trap Metal	0.0651	0.000243	0	-33.357	0	0.0242	0.000001	0.000000	0.0221	0.0206	74.716	52963	
Underground Rap	0.2410	0.134000	0	-21.657	0	0.0251	0.000018	0.000000	0.0221	0.0294	95.622	49227	
dnb	0.1380	0.349000	0	-17.088	0	0.0265	0.000003	0.000000	0.0189	0.0253	169.857	35862	
hardstyle	0.0891	0.464000	0	-16.475	0	0.0253	0.000008	0.000000	0.0153	0.0318	143.803	91617	
psytrance	0.2900	0.368000	0	-16.694	0	0.0300	0.000002	0.000924	0.0228	0.0228	128.008	108000	
techhouse	0.3680	0.231000	0	-22.714	0	0.0269	0.000004	0.000000	0.0107	0.0242	119.900	67431	
techno	0.2540	0.188000	0	-26.172	0	0.0276	0.000001	0.000000	0.0222	0.0187	120.001	119629	
trance	0.1210	0.262000	0	-19.708	0	0.0259	0.000003	0.000000	0.0159	0.0199	125.000	48667	
trap	0.1500	0.205000	0	-17.419	0	0.0272	0.000006	0.000000	0.0241	0.0237	135.019	25600	

Figure 9: .ipynb code for calculating standard deviation and min values of different features in a genre.

```
#Max values of different features in a Genre
grouped_genre.max()
```

genre	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms	time_signature
Dark Trap	0.985	0.998	11	1.646	1	0.946	0.984	0.989	0.958	0.968	220.290	534857	5
Emo	0.926	0.995	11	-0.946	1	0.729	0.988	0.952	0.943	0.971	208.951	548253	5
Hiphop	0.988	0.978	11	-0.067	1	0.944	0.982	0.947	0.973	0.975	206.247	723573	5
Pop	0.935	0.977	11	-2.058	1	0.463	0.948	0.947	0.790	0.966	210.796	484147	5
Rap	0.981	0.980	11	-1.181	1	0.833	0.883	0.951	0.892	0.970	205.895	728413	5
RnB	0.978	0.974	11	0.175	1	0.786	0.986	0.939	0.965	0.979	215.669	602297	5
Trap Metal	0.985	0.999	11	3.148	1	0.908	0.986	0.963	0.962	0.976	214.034	474157	5
Underground Rap	0.985	0.997	11	2.499	1	0.914	0.947	0.964	0.962	0.980	207.982	636213	5
dnb	0.855	0.999	11	3.108	1	0.906	0.783	0.968	0.981	0.970	177.100	514629	5
hardstyle	0.774	0.999	11	0.101	1	0.774	0.562	0.983	0.976	0.935	157.687	566495	5
psytrance	0.925	0.999	11	-1.464	1	0.473	0.321	0.964	0.988	0.917	157.033	913052	5
techhouse	0.988	0.999	11	-1.318	1	0.469	0.563	0.967	0.923	0.988	130.044	596191	5
techno	0.978	1.000	11	-1.927	1	0.422	0.911	0.972	0.966	0.950	142.031	894386	5
trance	0.887	1.000	11	-1.093	1	0.785	0.939	0.985	0.975	0.934	146.009	705131	5
trap	0.935	1.000	11	1.851	1	0.863	0.985	0.970	0.975	0.970	162.148	392683	5

Figure 10: .ipynb code for calculating Max values of different features in a genre.

```
#Songs count for all the genre's
genre_count = {}
for gen in np.unique(genre):
    genre_count[gen] = len(data[data['genre'] == gen])
genre_count
```

```
{'Dark Trap': 4578,
 'Emo': 1680,
 'Hiphop': 3028,
 'Pop': 461,
 'Rap': 1848,
 'RnB': 2099,
 'Trap Metal': 1956,
 'Underground Rap': 5875,
 'dnb': 2966,
 'hardstyle': 2936,
 'psytrance': 2961,
 'techhouse': 2975,
 'techno': 2956,
 'trance': 2999,
 'trap': 2987}
```

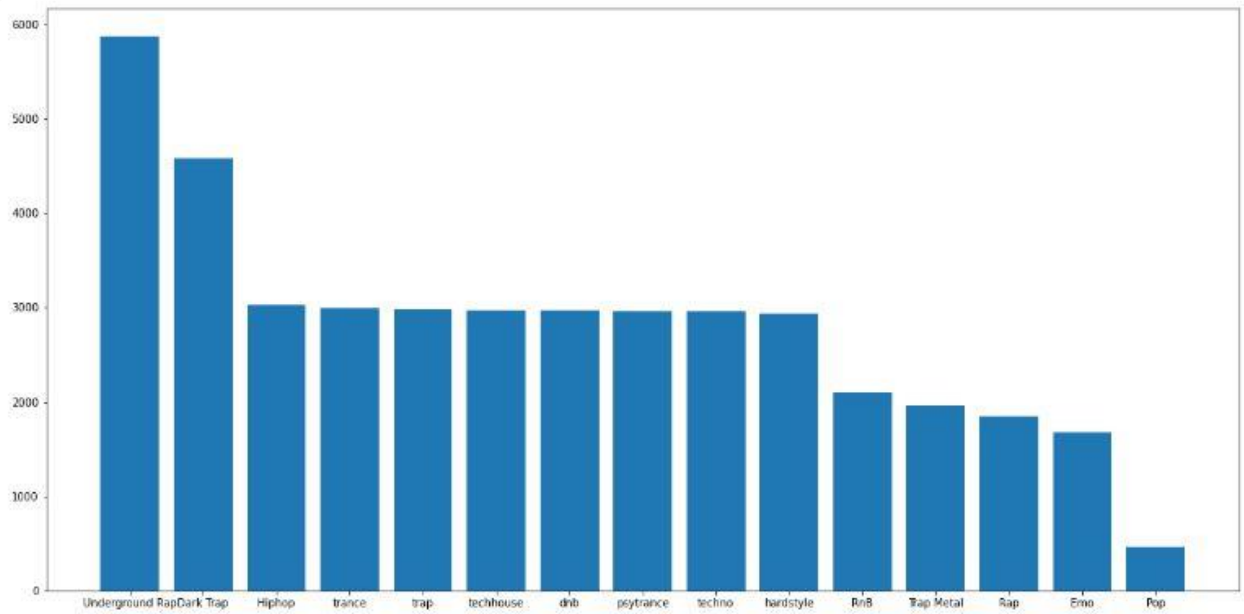
```
data['genre'].value_counts()
```

```
Underground Rap    5875
Dark Trap          4578
Hiphop             3028
trance             2999
trap               2987
techhouse          2975
dnb                2966
psytrance          2961
techno             2956
hardstyle          2936
RnB                2099
Trap Metal         1956
Rap                1848
Emo                1680
Pop                 461
Name: genre, dtype: int64
```

```
org_genre=data['genre']
```

Figure 11: .ipynb code describing songs count for all the genres.

```
fig = plt.figure(figsize=(20,10))
plt.bar(height=data['genre'].value_counts().values,x=data['genre'].value_counts().index)
plt.show()
```




```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,OneHotEncoder,StandardScaler
from imblearn.over_sampling import SMOTE
import seaborn as sns
import joblib

data['genre'] = LabelEncoder().fit_transform(data['genre'])
corr = data.corr()
sns.heatmap(data=corr)
```

```
<AxesSubplot:>
```

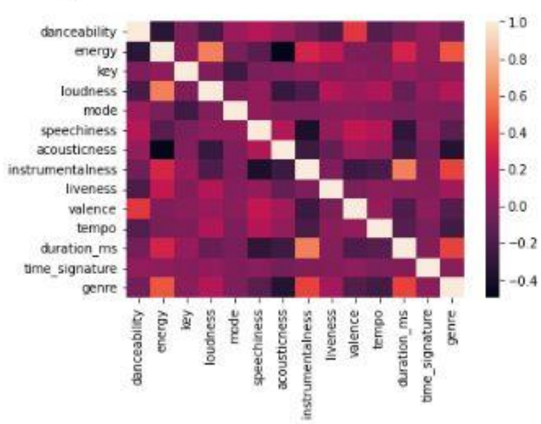


Figure 12: .ipynb code for displaying graphs based on genres.

```
corr['genre']
```

```
danceability        -0.052687
energy               0.471327
key                  0.027398
loudness             0.160771
mode                -0.019531
speechiness         -0.144596
acousticness        -0.356286
instrumentalness     0.414434
liveness             0.107690
valence             -0.170698
tempo               -0.231731
duration_ms          0.412508
time_signature       0.019943
genre                1.000000
Name: genre, dtype: float64
```

```
features = data.drop(['key','mode','time_signature','danceability','genre'],axis=1)
features
```

	energy	loudness	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_ms
0	0.814	-7.364	0.4200	0.059800	0.013400	0.0556	0.3890	156.985	124539
1	0.493	-7.230	0.0794	0.401000	0.000000	0.1180	0.1240	115.080	224427
2	0.893	-4.783	0.0623	0.013800	0.000004	0.3720	0.0391	218.050	98821
3	0.781	-4.710	0.1030	0.023700	0.000000	0.1140	0.1750	186.948	123661
4	0.624	-7.668	0.2930	0.217000	0.000000	0.1660	0.5910	147.988	123298
...	...	...	...	...	...	...	...	...	...
42300	0.693	-5.148	0.0304	0.031500	0.000345	0.1210	0.3940	150.013	269208
42301	0.768	-7.922	0.0479	0.022500	0.000018	0.2050	0.3830	149.928	210112
42302	0.821	-3.102	0.0505	0.026000	0.000242	0.3850	0.1240	154.935	234823
42303	0.921	-4.777	0.0392	0.000551	0.029600	0.0575	0.4880	150.042	323200
42304	0.945	-5.862	0.0615	0.001890	0.000055	0.4140	0.1340	155.047	162161

42305 rows × 9 columns

```
from IPython.display import display
pd.set_option('display.max_rows', 42304)
display(org_genre)
```

```
0         Dark Trap
1         Dark Trap
2         Dark Trap
3         Dark Trap
4         Dark Trap
            ...
42300     hardstyle
42301     hardstyle
42302     hardstyle
42303     hardstyle
42304     hardstyle
Name: genre, Length: 42305, dtype: object
```

Figure 13: .ipynb code describing correlation between genres and displaying maximum rows.

```
: from sklearn.model_selection import cross_val_score
  from sklearn.model_selection import RepeatedStratifiedKFold
  from sklearn.ensemble import BaggingClassifier

  model = BaggingClassifier()

  cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
  n_scores = cross_val_score(model, xtrain, ytrain, scoring='accuracy',\
                             cv=cv, n_jobs=-1, error_score='raise')
```

```
: n_scores
```

```
: array([0.76170213, 0.75390071, 0.75560284, 0.76368794, 0.7458156 ,
         0.75787234, 0.76553191, 0.75730496, 0.75574468, 0.75319149,
         0.76085106, 0.75319149, 0.76638298, 0.76312057, 0.75602837,
         0.75758865, 0.75617021, 0.7541844 , 0.74964539, 0.75546099,
         0.76425532, 0.76382979, 0.76241135, 0.75148936, 0.75985816,
         0.76184397, 0.7564539 , 0.74992908, 0.75177305, 0.75262411])
```

```
: #Average cro
  n_scores.mean()
```

```
: 0.7572482269503547
```

```
: model = BaggingClassifier()
  model.fit(xtrain,ytrain)
  pred = model.predict(xtest)
  pred
```

```
: array([ 2,  9,  7, ...,  9,  0, 11])
```

```
: #Accuracy of the model
  from sklearn.metrics import accuracy_score,f1_score,confusion_matrix
  accuracy_score(ytest,pred)
```

```
: 0.7649361702127659
```

Figure 14: .ipynb code describing importing bagging classifier and finding accuracy of the model.

```
f1_score(ytest,pred,average='weighted')
```

```
0.7601259806642734
```

```
joblib.dump(model,"bagging.model")
```

```
['bagging.model']
```

```
xtest[1563]
```

```
array([ 1.10841235,  0.12537504, -0.70206696, -0.5522838 ,  1.4185984 ,
       -0.71251307, -0.77712456, -0.31372699,  2.81788752])
```

Figure 15: .ipynb code describing xtest and ytest.

11.2 HTML AND PYTHON CODE :

1. app.py code

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Sep  9 11:41:10 2020

@author: Adminr
"""
# importing the necessary dependencies
from flask import Flask,request,render_template
import numpy as np
import joblib

app=Flask(__name__)# initializing a flask app
model=joblib.load('bagging.model')
sc=joblib.load('transform.save')

@app.route('/')# route to display the home page
def home():
    return render_template('home.html') #rendering the home page
@app.route('/Prediction',methods=['POST','GET'])
def prediction(): # route which will take you to the prediction page
    return render_template('indexnew.html')
@app.route('/Home',methods=['POST','GET'])
def my_home():
    return render_template('home.html')

@app.route('/predict',methods=["POST","GET"])# route to show the predictions in a web UI
def predict():
    #  reading the inputs given by the user
    input_feature=[float(x) for x in request.form.values() ]
    x=[np.array(input_feature)]
    x= sc.transform(x)
    print(x)
     # predictions using the loaded model file
    prediction=model.predict(x)
    labels=['Dark Trap','Emo','Hiphop','Pop','Rap','Rnb','Trap Metal','Underground Rap','dnb','hardstyle','psytrance','techhouse','techno','trance','trap']
    print("Prediction is:",labels[prediction[0]])
     # showing the prediction results in a UI
    return render_template("resultnew.html",prediction=labels[prediction[0]])
if __name__=="__main__":

    app.run(host='0.0.0.0', port=8000,debug=True)    # running the app

```

Figure 16: .python code used for rendering all the HTML pages.

2. home.html code

```html

<form action="/Prediction" method="[POST,GET]">
<header>
    <div class="wrapper">
        <div class="logo">
          <img src="static/logo.ico" class="w3-round" alt="Norway">
        </div>
        <link rel="stylesheet" type="text/css" href="style.css">
        <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='style.css') }}">

<ul class="nav-area">

<li><a href="#">Home</a></li>

    <input  type="submit" value="Prediction">

</ul>
</div>
<div class="welcome-text">
        <h1>Music Genre Prediction for Spotify</h1>
 </div>
</header>


</form>
```

Figure 17: home.html is the code for home page of our Web Application.

3. index.html code

```
<html>
<style>
.idiv{
width:60%;
margin:auto;
background-color:#008a91;
text-align:center;
margin-top:2%;
border-radius:10px;


}
body{
font-family:segoe ui;
background: linear-gradient(rgba(0,0,0,0.8),rgba(0,0,0,0.8)),url(https://ak.picdn.net/shutterstock/videos/2754836/thumb/2.jpg);
    height: 100vh;
    -webkit-background-size: cover;
    background-size: cover;
    background-position: center center;
    position: relative;
}

input{
font-size:1.3em;
width:80%;
text-align:center;
border-color: white;


}
::placeholder { /* Chrome, Firefox, Opera, Safari 10.1+ */
  color: olive;
  opacity: 1; /* Firefox */
}
input placeholder{
text-align:center;
}
button{
outline:0;
border:0;
background-color:darkred;
color:white;
width:100px;
height:40px;
```

```
border:solid 1px black;
}
select {
font-size:1.3em;
width:80%;
text-align:center;
    text-indent: 31%;
     border: 2px solid balck;

     color: #808000;

    }
hr{
border-top: 1px dashed black;
}
</style>
<head>
<title>-- Music Genre Prediction for Spotify -- </title>
</head>
<body>
<div class='idiv'>

<br/>
<h1>Music Genre Prediction</h1>
<hr/>
<br/>
<form action="/predict" method="POST">

 <input class="form-input" type="text" name="Energy" placeholder="Enter the energy in music"><br>
 <input class="form-input" type="text" name="Loudness" placeholder="Enter the Loudness in music"><br>
 <input class="form-input" type="text" name="Speechiness" placeholder="Enter the speechiness in music"><br>
 <input class="form-input" type="text" name="Acousticness" placeholder="Enter the acousticness in music"><br>
 <input class="form-input" type="text" name="Instrumentalness" placeholder="Enter the instrumentalness in music"><br>
 <input class="form-input" type="text" name="Liveness" placeholder="Enter the liveness in music"><br>
 <input class="form-input" type="text" name="Valence" placeholder="Enter the valence in music"><br>
 <input class="form-input" type="text" name="Tempo" placeholder="Enter the tempo in music"><br>
 <input class="form-input" type="text" name="Duration" placeholder="Enter the duration of music in millisec"><br>
<button type="submit" class="btn btn-primary btn-block btn-large">Predict</button>
</form>
<br/>
<br/>
<br/>
</div>

</body>
</html>
```

Figure 18: index.html is the code for index page of our Web Application

4. result new.html code

```
<html>
<style>
.idiv{
width:60%;
margin:auto;
background-color:black;
text-align:center;
margin-top:2%;
border-radius:10px;
background-image:url("https://api.time.com/wp-content/uploads/2018/04/listening-to-music-headphones.jpg?quality=85&w=1024&h=512&crop=1");

  background-repeat: no-repeat;



margin-top:2%;
}
body{
background-color:black;
font-family:segoe ui;
background: linear-gradient(rgba(0,0,0,0.8),rgba(0,0,0,0.8)),url(https://9b16f79ca967fd0708d1-2713572fef44aa49ec323e813b06d2d9.ssl.cf2.rackcdn.com/1140x_a10-7_cTC/NS-WKMAG0730-1595944356.jpg);
    height: 100vh;
    -webkit-background-size: cover;
    background-size: cover;
    background-position: center center;
    position: relative;

}
input{
font-size:1.3em;
width:80%;
text-align:center;
}
input placeholder{
text-align:center;
}


button{
outline:0;
border:0;
background-color:darkred;
color:white;
width:100px;
height:40px;
}
button:hover{
background-color:brown;
border:solid 1px black;
}
h1{
color:red;
}
h2{
color:olive;

}
</style>
<head>
<title>--Music Genre Prediction -- </title>
</head>
<body>
<div class='idiv'>

<br/>
<h1>Music Genre Prediction</h1>
<br/>
<h2> Genre Predicted is {{prediction}}</h2>

<br/>
<br/>
<br/>
</div>

</body>
</html>
```

Figure 19: resultnew.html is the code for displaying result.

12. CONCLUSION

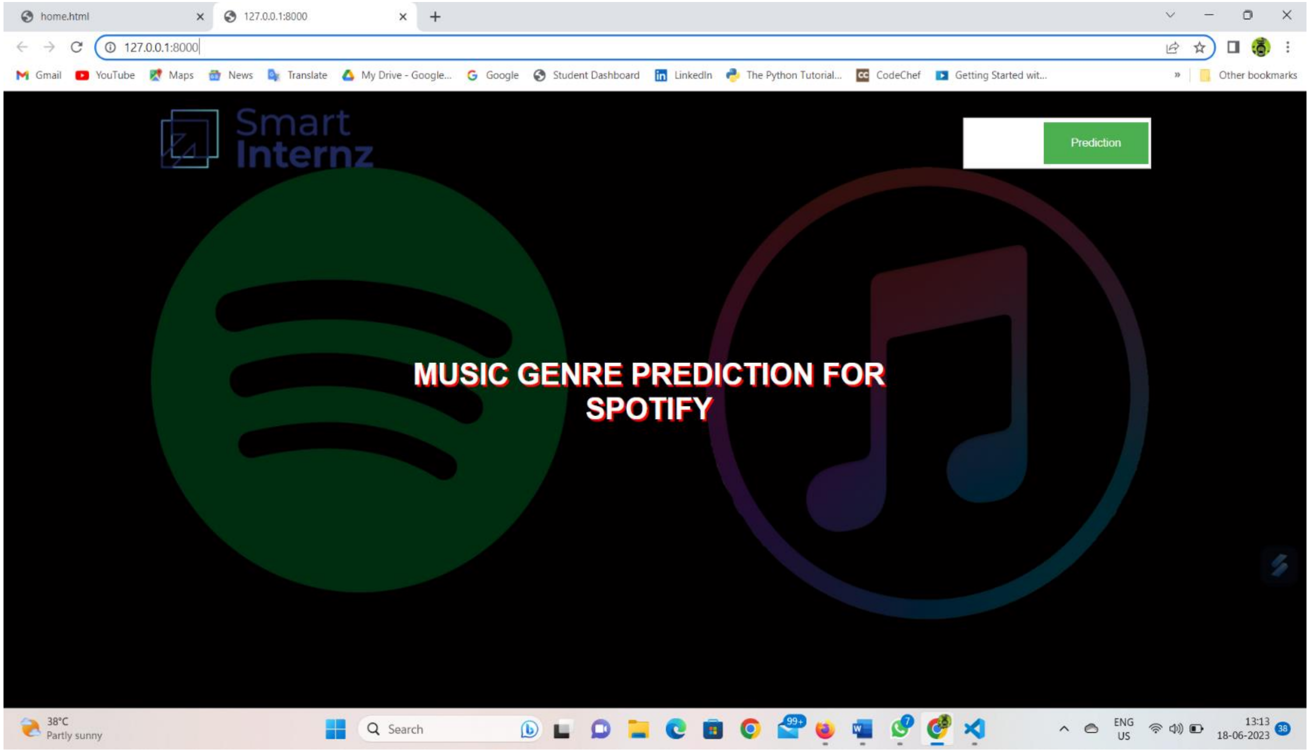


Figure 20: Home Page (Which has Prediction button on top right of the Display)

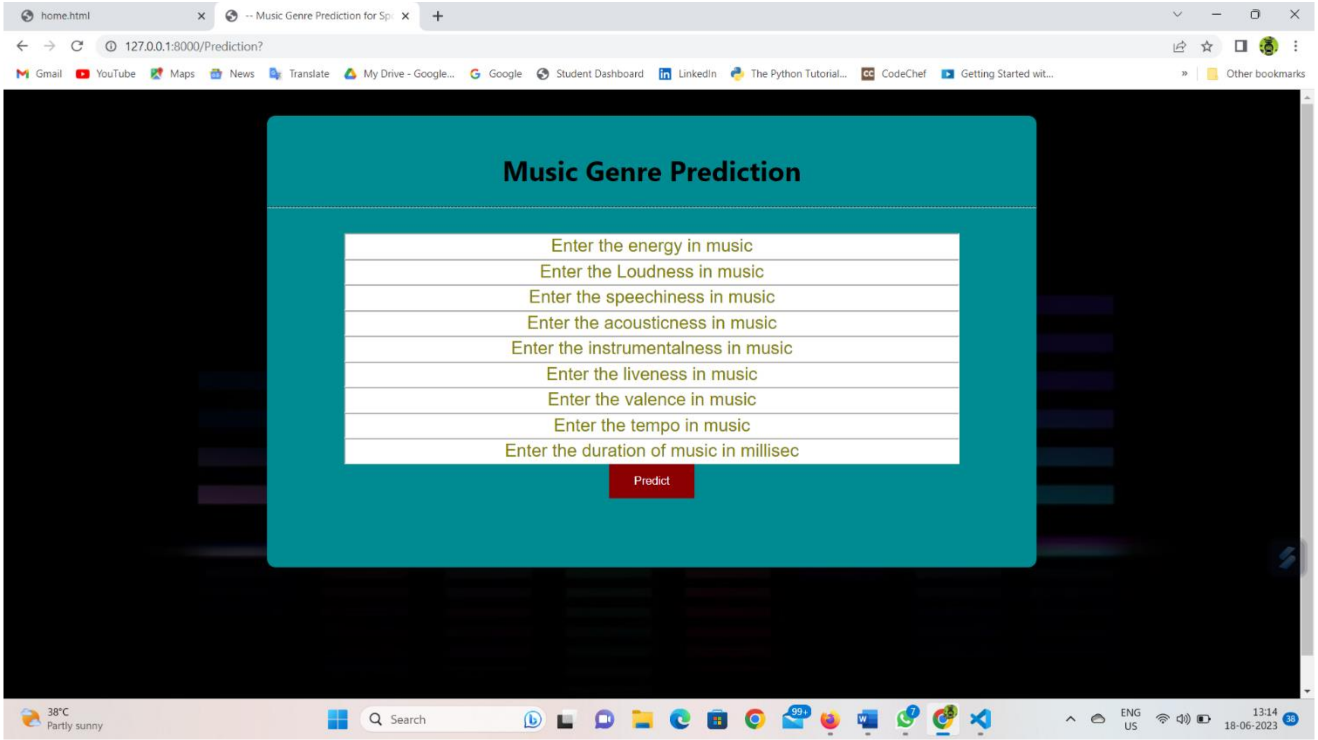


Figure 21: Input page (Which takes input from the user).

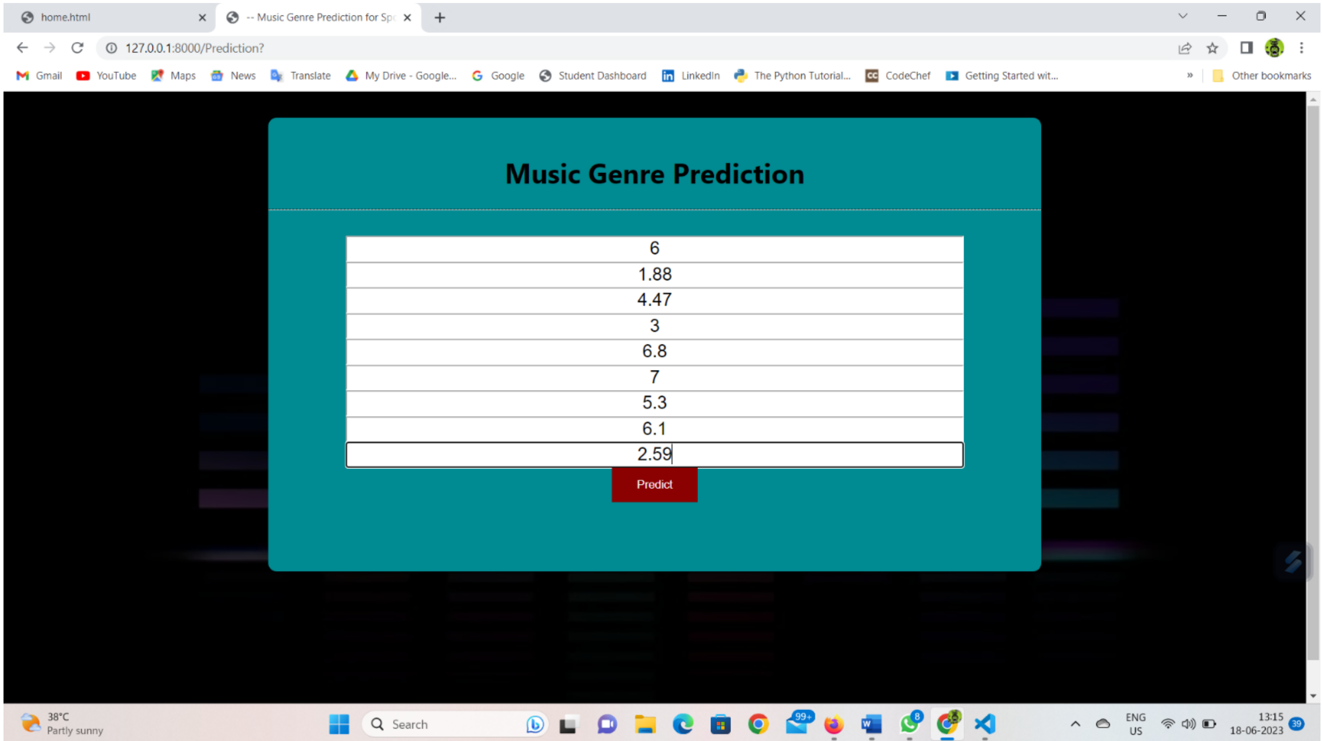


Figure 22: Input Page (where inputs are given by the User).

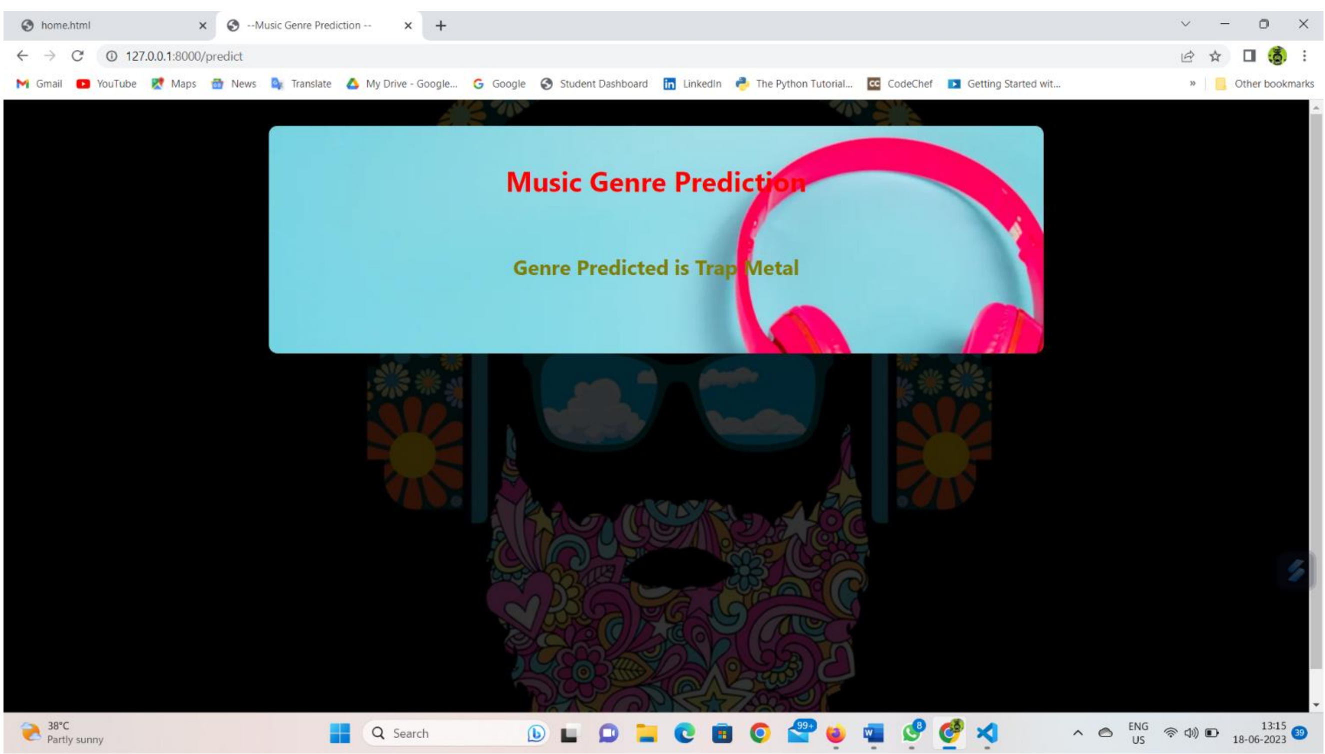


Figure 23: Output Page (Displays the type of the Music Genre).

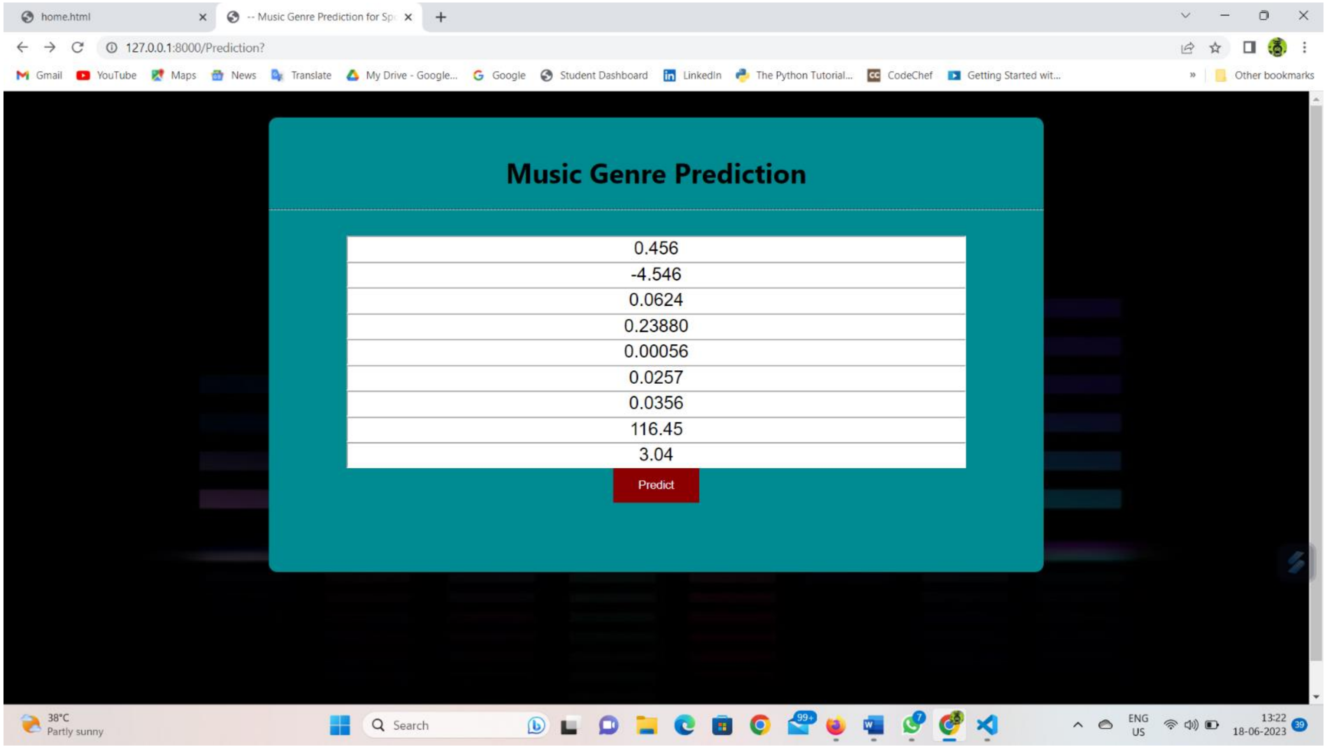


Figure 24: Input Page (where inputs are given by the User).

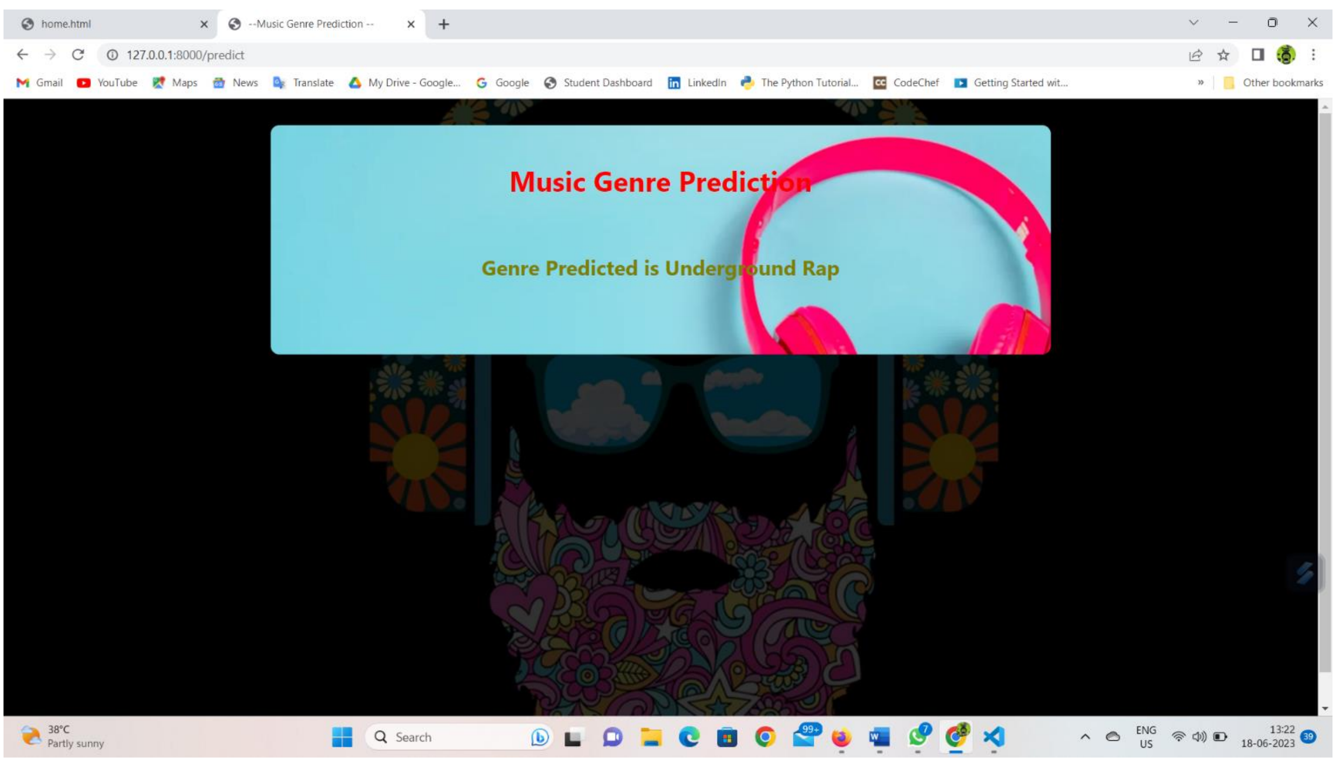


Figure 25: Output Page (Displays the type of the Music Genre).

13.HELP FILE

PROJECT EXECUTION:

STEP-1: Go to **Start**, search and launch **ANACONDA NAVIGATOR.**

STEP-2: After launching of **ANACONDA NAVIGATOR**, launch **JUPYTER NOTEBOOK.**

STEP-3: Open **“Music Genre Classification” IPYNB file.**

STEP-4: Then run all the cells.

STEP-5: All the **data preprocessing, training and testing, model building, accuracy** of the model can be showcased.

STEP-6: And a pickle file will be generated.

STEP-7: Create a Folder named **FLASK** on the **DESKTOP**. Extract the pickle file into this Flask Folder.

STEP-8: Extract all the html files (home.html, index.html, resultsnew.html) and python file(app.py) into the **FLASK Folder.**

STEP-9: Then go back to **ANACONDA NAVIGATOR** and the launch the **SPYDER.**

STEP-10: After launching Spyder, give the path of **FLASK FOLDER** which you have created on the DESKTOP.

STEP-11: Open all the app.py and html files present in the Flask Folder.

STEP-12: After running of the app.py, open **ANACONDA PROMPT** and follow the below steps:

cd File Path click enter

python app.py click enter (We could see running of files).

STEP-13: Then open **BROWSER**, at the URL area type “**localhost:5000”.**

STEP-14: Home page of the project will be displayed.

STEP-15: Click on **“Prediction”.** Directly it will be navigated to index page.

STEP-16: A index page will be displayed where the user needs to give the inputs and then click on **“Predict”.** Output will be generated showing the type of the Music Genre.