VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEM FACULTY

SUBJECT: DATA MINING

FINAL PROJECT REPORT

TOPIC:

PREDICT THE MUSIC GENRE OF A SONG

Lecturer: Mrs Cao Thi Nhan

Mr Vu Minh Sang

Class: IS252.M21.HTCL

Group: Team 11

Student performance:

Bui Duc Duy	20521228
Nguyen Thi Cam Van	20522145
Ho Bao An	20520876
Luu Thao Linh	20521532

Ho Chi Minh City, June 2023

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEM FACULTY

SUBJECT: DATA MINING

FINAL PROJECT REPORT

TOPIC:

PREDICT THE MUSIC GENRE OF A SONG

Lecturer: Mrs Cao Thi Nhan

Mr Vu Minh Sang

Class: IS252.M21.HTCL

Group: Team 11

Student performance:

Bui Duc Duy	20521228
Nguyen Thi Cam Van	20522145
Ho Bao An	20520876
Luu Thao Linh	20521532

Ho Chi Minh City, June 2023

TEACHER'S COMMENTS

ACKNOWLEDGEMENT

In fact, there is no success that is not tied to the support and help, whether more or less, directly or indirectly from others. With the deepest gratitude, first of all, our group would like to express our sincere thanks to the teachers of the University of Information Technology - Vietnam National University, Ho Chi Minh City and the teachers of the Faculty of Information Systems helped the group to have the basic knowledge as a basis to carry out this topic.

In particular, our team would like to express our sincere thanks to Mrs. Cao Thi Nhan - theoretical lecturer and Mr. Vu Minh Sang -practical lecturer of Data Mining who wholeheartedly helped, directly instructed and guided the group throughout the process a project.

Thanks to that, we have gained a lot of useful knowledge in applying as well as project-making skills. Without the guidance and teachings of the teacher, our group thinks this project of the group would be very difficult to complete. Once again, I sincerely thank teacher. In addition, for the project to be completed, it is impossible to thank the people who did it, thank you to the team members who worked hard and completed the task on schedule.

Finally, thank you to all the team members who worked at their best to complete their thesis well. Sincerely thank!

TABLE OF CONTENTS

CHAPTER I : INTRODUCTION

1.1 Problems

Music is everywhere around us. The impact of music on human beings cannot be expressed in words. Through its perennial journey, the face of music has seen a lot of diversity and change in its form. The world around us is not homogenous and there exists a vast diversity in the lifestyle of people, their thinking and their culture. With this diversity it is apparent that music too will be different and will have diversity that matches to the interests of different people. Keeping this diversity in mind there is tremendous scope for audio streaming and media services companies to automatically classify the genre of a particular song in order to cater similar songs to potential listeners. With the current boom in Data Science practices and predictive modelling and with the availability of structured/unstructured data online such tasks have become convenient and are at the forefront of large companies.

In this online assesment the given task at hand aims to categorize songs into their respective categories viz. (Electronic, Rap, Hip-Hop, Pop) based on some certain parameters or features given in a tabular data format on the basis of which we classify which category the songs belong to.

The first and foremost task would be to import all the necessary libraries.

1.2 Project goal

- Building a data system on natural language, using machine learning to train machines to make highly reliable predictions and information for humans.

- Music genre prediction helps recommend songs, albums or playlists based on genres that the user has liked before. This helps users discover new music that might match their taste

1.3 Developer tools & Technology

In the process of implementation, the group used a number of software for researching and developing the topic:

- Information collection and analysis using the python library and programming language.

- Data sources: [Prediction of music genre | Kaggle]

All of the above software is installed and used by the team on Microsoft Windows 10 operating system. The compatibility of the above software with other operating systems is not within the scope of this study.

CHAPTER II: DATA PREPROCESSING

2.1 Description of original data

2.1.1 Data Sources

Author: VICSUPERMAN

2.1.2 Data file

Total data rows: 50005

2.1.3 Attribute number and value

Total columns: 18

Dataset characteristics: Multivariable

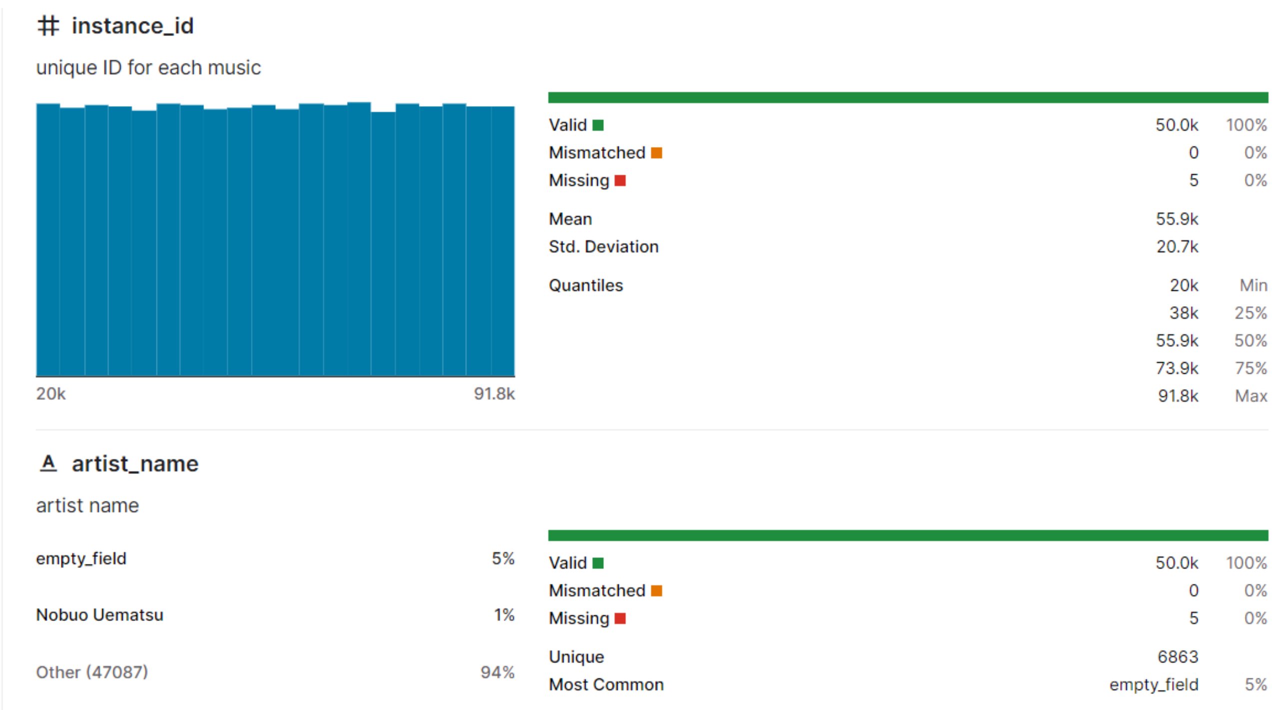
Attribute number characteristics: Characters, real numbers, integers

Lost value: None

2.1.4 Statistics of attribute values

Symbol: # -number, -character

Sources: [Prediction of music genre | Kaggle]



A track_name

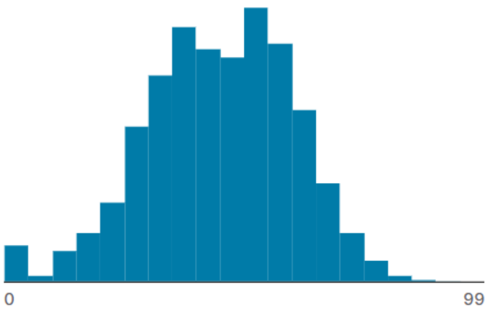
track name

41700
unique values

Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Unique	41.7k	
Most Common	Home	0%

popularity

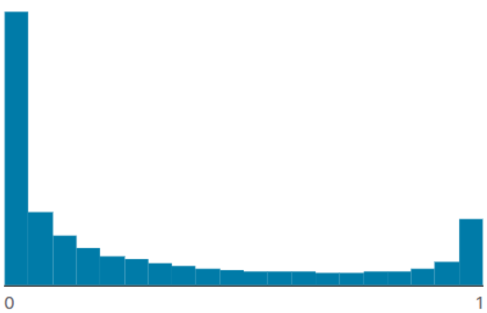
how popular of this music



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	44.2	
Std. Deviation	15.5	
Quantiles	0	Min
	34	25%
	45	50%
	56	75%
	99	Max

acousticness

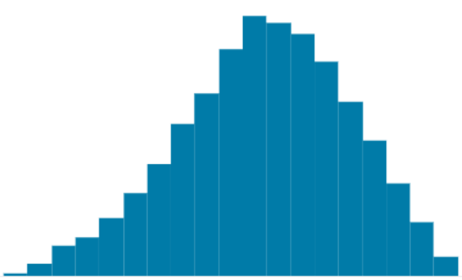
acousticness



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.31	
Std. Deviation	0.34	
Quantiles	0	Min
	0.02	25%
	0.14	50%
	0.55	75%
	1	Max

danceability

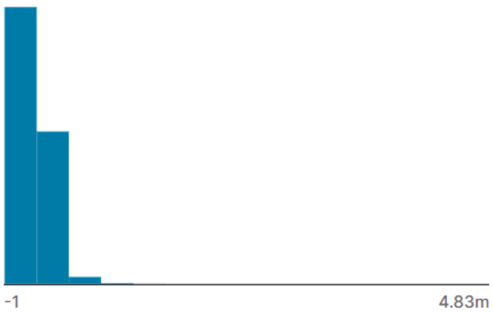
danceability



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.56	
Std. Deviation	0.18	
Quantiles	0.06	Min
	0.44	25%
	0.57	50%

duration_ms

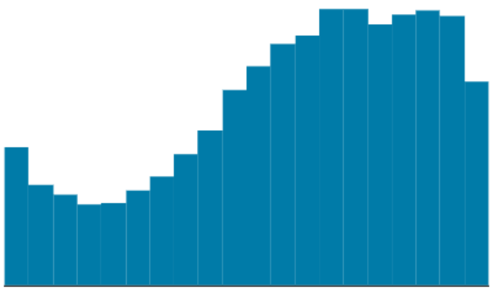
the duration of the music in ms



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	221k	
Std. Deviation	129k	
Quantiles	-1	Min
	175k	25%
	219k	50%
	269k	75%
	4.83m	Max

energy

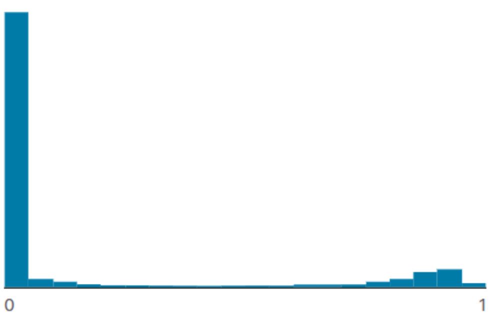
energy



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.6	
Std. Deviation	0.26	
Quantiles	0	Min
	0.43	25%
	0.64	50%
	0.81	75%

instrumentalness

instrumentalness



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.18	
Std. Deviation	0.33	
Quantiles	0	Min
	0	25%
	0	50%
	0.15	75%
	1	Max

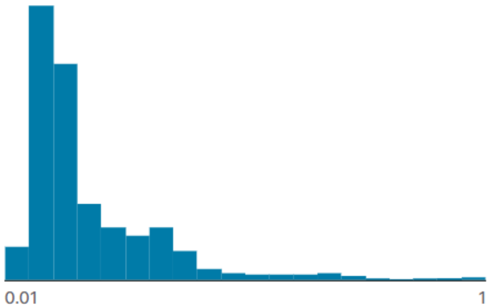
A key

music key

G	11%
C	11%
Other (38756)	78%

Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Unique	12	
Most Common	G	11%

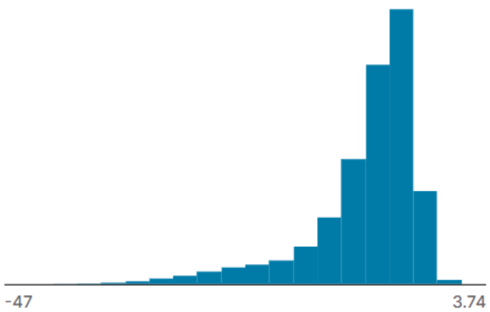
liveness



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.19	
Std. Deviation	0.16	
Quantiles	0.01	Min
	0.1	25%
	0.13	50%
	0.24	75%
	1	Max

loudness

loudness



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	-9.13	
Std. Deviation	6.16	
Quantiles	-47	Min
	-10.9	25%
	-7.28	50%
	-5.17	75%
	3.74	Max

A mode

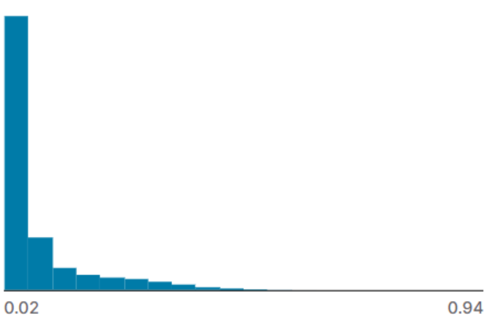
mode

Major	64%
Minor	36%
Other (5)	0%

Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Unique	2	
Most Common	Major	64%

speechiness

speechiness



Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.09	
Std. Deviation	0.1	
Quantiles	0.02	Min
	0.04	25%
	0.05	50%
	0.1	75%
	0.94	Max

tempo

tempo

?	10%
120.0	0%
Other (45008)	90%

Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Unique	29.4k	
Most Common	?	10%

obtained_date

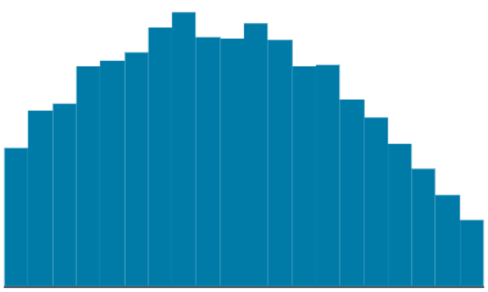
date

1Apr01 1Apr05

Valid	50.0k	100%
Mismatched	1	0%
Missing	5	0%
Minimum	1Apr01	
Mean	28Feb04	
Maximum	1Apr05	

valence

valence



0 0.99

Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Mean	0.46	
Std. Deviation	0.25	
Quantiles	0	Min
	0.26	25%
	0.45	50%
	0.65	75%
	0.99	Max

music_genre

our Class

11
unique values

Valid	50.0k	100%
Mismatched	0	0%
Missing	5	0%
Unique	10	
Most Common	Electronic	10%

Attribute statistics table:

STT	Attribute	Attribute meaning	Value of property	Average value	Median value	Mode
1	**instance_id**	Serial number of the song				
2	**artist_name**	Name of the artist				Empty_field
3	**track_name**	Title of the song				Home
4	**popularity**	Score assigned to the song	From [0-100]	44.220	48.0	52.0
5	**acousticness**	Acoustic a song	From [0.0 – 1.0]	0.3063	0.0171	0.995
6	**danceability**	Danceability	From [0.0 – 1.0]	0.558	0.591	0.529
7	**duration_ms**	Duration in milliseconds		221252.602	219360.0	-1.0
8	**energy**	Energetic the song	From [0 - 1]	0.5997	0.674	0.805
9	**instrumentalness**	The amount of vocals	From [0.0 – 1.0]	0.1816	0.000159	0.0
10	**key**	Group of pitches or scale				G
11	**liveness**	Probability that the song was recorded with a live audience		0.1938	0.33	0.11
12	**loudness**	Loud the song		-9.1337	-6.74	-5.443
13	**mode**	Major and Minor scales that the song				Major
14	**speechiness**	Presence of spoken words in a track		0.0935	0.26899	0.0332
15	**tempo**	Speed at which the song is being played				?
16	**obtained_date**	The date at which the song metadata was retrieved				4-Apr

17	**valence**	The musical positiveness conveyed by a track	From [0.0 – 1.0]	0.4562	0.505	0.3379
18	**music_genre**	The actual category to which the song belongs				Electronic

2.1.5 Subclass Number

Subclass Attributes: **instance_id, popularity, acousticness, danceability, duration_ms, energy, instrumentalness, liveness, loudness, speechiness, valence.**

2.2 Data preprocessing

Purpose:

- DataTransformation
- Data Collection
- Data visualization and comments

2.2.1 Import Library

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

✓ 0.6s Python

2.2.2 Import Dataset

```
data = pd.read_csv('music_genre.csv')
data
```

✓ 0.5s Python

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mode
0	32894.0	Röyksopp	Röyksopp's Night Out	27.0	0.00468	0.652	-1.0	0.941	0.79200	A#	0.115	-5.201	Minor
1	46652.0	Thievery Corporation	The Shining Path	31.0	0.01270	0.622	218293.0	0.890	0.95000	D	0.124	-7.043	Minor
2	30097.0	Dillon Francis	Hurricane	28.0	0.00306	0.620	215613.0	0.755	0.01180	G#	0.534	-4.617	Major
3	62177.0	Dubloadz	Nitro	34.0	0.02540	0.774	166875.0	0.700	0.00253	C#	0.157	-4.498	Major
4	24907.0	What So Not	Divide & Conquer	32.0	0.00465	0.638	222369.0	0.587	0.90900	F#	0.157	-6.266	Major
...	...	...	...	...	...	...	...	...	...	...	...	...	...
50000	58878.0	BEXEY	GO GETTA	59.0	0.03340	0.913	-1.0	0.574	0.00000	C#	0.119	-7.022	Major
50001	43557.0	Roy Woods	Drama (feat. Drake)	72.0	0.15700	0.709	251860.0	0.362	0.00000	B	0.109	-9.814	Major
50002	39767.0	Berner	Lovin' Me (feat. Smiggz)	51.0	0.00597	0.693	189483.0	0.763	0.00000	D	0.143	-5.443	Major
50003	57944.0	The-Dream	Shawty Is Da Shit	65.0	0.08310	0.782	262773.0	0.472	0.00000	G	0.106	-5.016	Minor
50004	63470.0	Naughty By Nature	Hip Hop Hooray	67.0	0.10200	0.862	267267.0	0.642	0.00000	F#	0.272	-13.652	Minor

50005 rows × 18 columns

2.2.3 Check data type, information

```
# Check data type
data.dtypes
```

✓ 0.2s Python

```
instance_id          float64
artist_name           object
track_name            object
popularity           float64
acousticness         float64
danceability         float64
duration_ms          float64
energy               float64
instrumentalness     float64
key                   object
liveness             float64
loudness             float64
mode                  object
speechiness          float64
tempo                 object
obtained_date         object
valence              float64
music_genre           object
dtype: object
```

```python
data.info()
```

✓ 0.2s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50005 entries, 0 to 50004
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   instance_id       50000 non-null  float64
 1   artist_name       50000 non-null  object 
 2   track_name        50000 non-null  object 
 3   popularity        50000 non-null  float64
 4   acousticness      50000 non-null  float64
 5   danceability      50000 non-null  float64
 6   duration_ms       50000 non-null  float64
 7   energy            50000 non-null  float64
 8   instrumentalness  50000 non-null  float64
 9   key               50000 non-null  object 
 10  liveness          50000 non-null  float64
 11  loudness          50000 non-null  float64
 12  mode              50000 non-null  object 
 13  speechiness       50000 non-null  float64
 14  tempo             50000 non-null  object 
 15  obtained_date     50000 non-null  object 
 16  valence           50000 non-null  float64
 17  music_genre       50000 non-null  object 
dtypes: float64(11), object(7)
```

2.2.4 Overview of the data

- Data is a combination of string and integer values.

- **instance_id :** Serial number of the song in the dataset.

- **artist_name** : Name of the artist of the song.

- **track_name** : Title of the song.

- **popularity** : An arbitrary score assigned to the song in the range of 0-100 with 100 being most popular and 0 being least.

- **acousticness** : This value describes how acoustic a song is. A score of 1.0 means the song is most likely to be an acoustic one.

- **danceability** : Danceability describes how suitable a track is for dancing based on a combination of musical elements. A value of 0.0 is least danceable and 1.0 is most danceable.

- **duration_ms** : Is the duration in milliseconds of the song.

- **energy** : Represents how energetic the song is. The range of this field is between [0-1] with 1 being song with highest energy and 0 with lowest.

- **instrumentalness** : This value represents the amount of vocals in the song. The closer it is to 1.0, the more instrumental the song is.

- **key** : Key of a piece is the group of pitches, or scale, that forms the basis of a music composition.

- **liveness** : This value describes the probability that the song was recorded with a live audience.

- **loudness** : Column representing how loud the song is.

- **mode :** Major and Minor scales that the song is based upon.

- **speechiness** : Speechiness detects the presence of spoken words in a track.

- **tempo** : Speed at which the song is being played.

- **obtained_date** : The date at which the song metadata was retrieved.

- **valence** : A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive.

- **music_genre** : The actual category to which the song belongs. This is our target variable.

2.2.5 Description of dataset information

```python
#Information description of numeric data
data.describe().T
```

✓ 0.3s Python

	count	mean	std	min	25%	50%	75%	max
instance_id	50000.0	55888.396360	20725.256253	20002.000000	37973.5000	55913.500000	73863.250000	91759.000
popularity	50000.0	44.220420	15.542008	0.000000	34.0000	45.000000	56.000000	99.000
acousticness	50000.0	0.306383	0.341340	0.000000	0.0200	0.144000	0.552000	0.996
danceability	50000.0	0.558241	0.178632	0.059600	0.4420	0.568000	0.687000	0.986
duration_ms	50000.0	221252.602860	128671.957157	-1.000000	174800.0000	219281.000000	268612.250000	4830606.000
energy	50000.0	0.599755	0.264559	0.000792	0.4330	0.643000	0.815000	0.999
instrumentalness	50000.0	0.181601	0.325409	0.000000	0.0000	0.000158	0.155000	0.996
liveness	50000.0	0.193896	0.161637	0.009670	0.0969	0.126000	0.244000	1.000
loudness	50000.0	-9.133761	6.162990	-47.046000	-10.8600	-7.276500	-5.173000	3.744
speechiness	50000.0	0.093586	0.101373	0.022300	0.0361	0.048900	0.098525	0.942
valence	50000.0	0.456264	0.247119	0.000000	0.2570	0.448000	0.648000	0.992

```python
# Information description of string data
data.describe(include=['O'])
```

✓ 0.4s Python

	artist_name	track_name	key	mode	tempo	obtained_date	music_genre
count	50000	50000	50000	50000	50000	50000	50000
unique	6863	41699	12	2	29394	5	10
top	empty_field	Home	G	Major	?	4-Apr	Electronic
freq	2489	16	5727	32099	4980	44748	5000

We can see that there are 17 features and one label column (music_genre). Out of the features, 12 are numerical (one of which, tempo, is missclassified and will be dealt with later), and 5 are categorical.

We can also already see hints to hidden missing values in 3 features **('tempo', 'artist_name' and 'duration_ms').** Those will be dealt with shortly one by one.

2.2.6 Handle the null data

Check the null data

```python
# Check the null data
data[data.isnull().any(axis=1)]
```

✓ 0.2s Python

	instance_id	artist_name	track_name	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	key	liveness	loudness	mod
10000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
10001	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
10002	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
10003	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
10004	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na

There are only 5 rows that contain NaN values. We'll remove them

```
# The drop the rows with null data
data.dropna(inplace=True)
data.reset_index(drop=True, inplace=True)
data.info()
```
✓ 0.4s Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 18 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   instance_id       50000 non-null  float64
 1   artist_name       50000 non-null  object 
 2   track_name        50000 non-null  object 
 3   popularity        50000 non-null  float64
 4   acousticness      50000 non-null  float64
 5   danceability      50000 non-null  float64
 6   duration_ms       50000 non-null  float64
 7   energy            50000 non-null  float64
 8   instrumentalness  50000 non-null  float64
 9   key               50000 non-null  object 
 10  liveness          50000 non-null  float64
 11  loudness          50000 non-null  float64
 12  mode              50000 non-null  object 
 13  speechiness       50000 non-null  float64
 14  tempo             50000 non-null  object 
 15  obtained_date     50000 non-null  object 
 16  valence           50000 non-null  float64
 17  music_genre       50000 non-null  object 
dtypes: float64(11), object(7)
memory usage: 6.9+ MB
```

2.2.7 Check the balance data of the predictor attribute

```
# Check if the data is balanced
data['music_genre'].value_counts()
```
✓ 0.1s Python

```
Electronic     5000
Anime          5000
Jazz           5000
Alternative    5000
Country        5000
Rap            5000
Blues          5000
Rock           5000
Classical      5000
Hip-Hop        5000
Name: music_genre, dtype: int64
```

There are 10 different genres with equal distribution (balanced data). This means the accuracy score will be a good metric to use.

2.2.8 Data preprocessing and visualization with the attributes one by one

1. Instance_id column

This is just an index. We'll drop it.

```
data = data.drop(columns=['instance_id'])
```
✓ 0.3s Python

2. artist_name column

Print out the unique values

```
print(f"There are {data['artist_name'].nunique()} unique artists in the set")
```
✓ 0.1s Python

```
There are 6863 unique artists in the set
```

Attribute description

```
data['artist_name'].describe()
```
Python

```
count            50000
unique            6863
top        empty_field
freq              2489
Name: artist_name, dtype: object
```

Find the missing data

```
missing_artist = data[data['artist_name'] == 'empty_field']
missing_artist.head()
```
Python

	artist_name	track_name	popularity	acousticness	danceability	duration_ms	energy
19	empty_field	7th Sevens	50.0	0.0281	0.656	307328.0	0.65
25	empty_field	Revolution	34.0	0.0236	0.715	221050.0	0.97
44	empty_field	World (The Price Of Love) - [Radio Edit] [2015...	31.0	0.0035	0.595	222147.0	0.90
128	empty_field	Down With Me - VIP	32.0	0.0139	0.498	-1.0	0.94
135	empty_field	Olvidela Compa	44.0	0.1530	0.792	265133.0	0.54

Print out the percentage of data that is missed

```python
# Print out the percentage of data that is missed
print(f"Percent of missing artist names: {(missing_artist.shape[0]/data.shape[0])
```

✓ 0.1s Python

```
Percent of missing artist names: 4.978%
```

Nearly 5% of the observations are missing the artist's names **(marked as 'empty_field'**), but these entries are still valid otherwise. we will not drop these observations.

```python
data[data['artist_name'] != 'empty_field'].groupby('artist_name')['music_genre'].
```

erver (Ctrl+Alt+D)

Python

```
1    0.799767
2    0.171087
3    0.027834
4    0.001312
Name: music_genre, dtype: float64
```

For the entries that do contain an artist's name, it seems that a song that comes from a particular artist has an ~80% chance of belonging to one specific genre.

However, in it's current form it's not helpful for classifying songs from artists outside the training set. We'll need to extract more general features, starting with the simplest - name length.

Find the length of the **artists names.**

```python
# Find the length of the artists names
data['length_name'] = data['artist_name'].str.len()
```

Python

Statistics the data between the two columns are artist_name and music_genre.

```python
data[data['artist_name'] != 'empty_field'].groupby('music_genre')['length_name'].
```
✓ 0.4s Python

music_genre	count	mean	std	min	25%	50%	75%	max
Alternative	4728.0	11.397631	4.898169	1.0	8.0	11.0	14.0	46.0
Anime	4728.0	12.198604	5.419303	2.0	8.0	12.0	15.0	35.0
Blues	4745.0	13.544573	5.425815	1.0	10.0	12.0	16.0	40.0
Classical	4734.0	16.336502	4.489369	4.0	13.0	15.0	20.0	52.0
Country	4779.0	12.656623	3.277890	3.0	11.0	12.0	14.0	41.0
Electronic	4777.0	9.734561	4.199621	2.0	7.0	9.0	12.0	26.0
Hip-Hop	4755.0	9.376025	3.986559	2.0	6.0	9.0	12.0	36.0
Jazz	4770.0	12.504193	4.777946	3.0	10.0	12.0	15.0	51.0
Rap	4737.0	9.645134	4.028050	2.0	7.0	9.0	12.0	38.0
Rock	4758.0	12.005885	5.026811	2.0	9.0	11.0	14.0	46.0

Remove the empty field data

```python
data2 = data.drop(data[data["artist_name"] == "empty_field"].index)
```
✓ 0.1s Python

Get top 20 artists

```python
artists = data2["artist_name"].value_counts()[:20].sort_values(ascending = True)
artists
```

✓ 0.3s Python

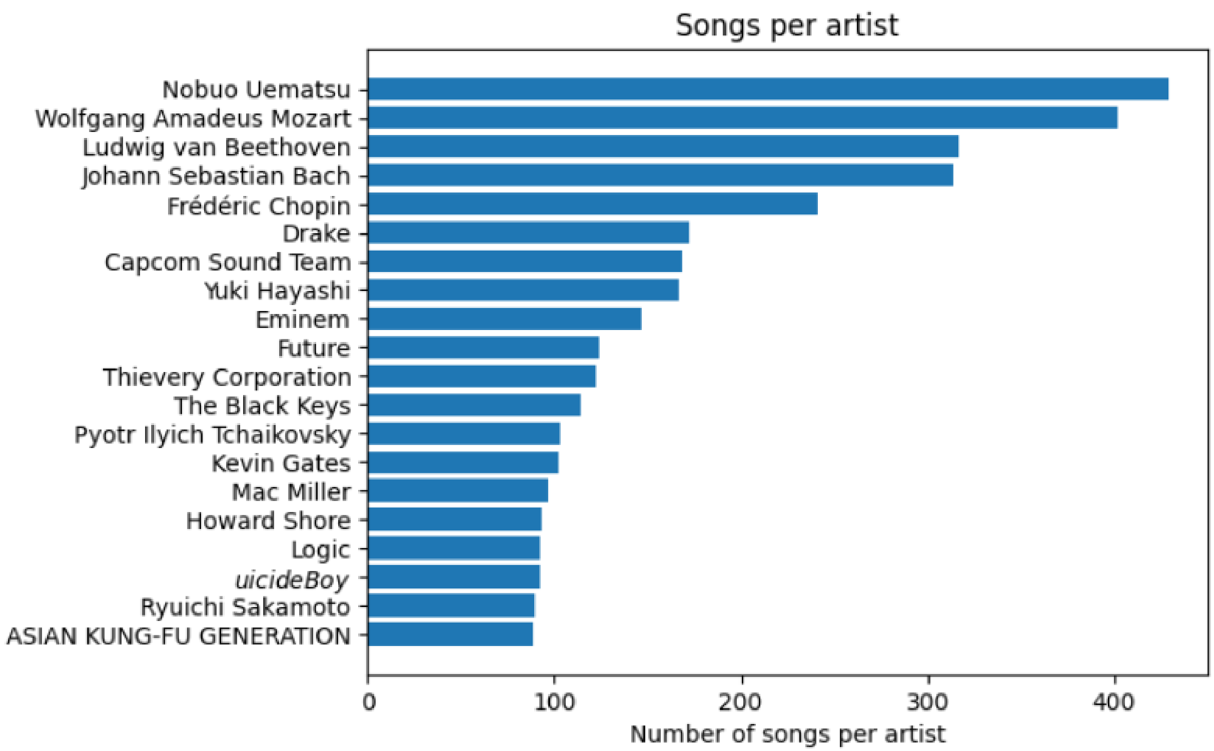
```
ASIAN KUNG-FU GENERATION      89
Ryuichi Sakamoto              90
$uicideBoy$                   92
Logic                         92
Howard Shore                  93
Mac Miller                    97
Kevin Gates                  102
Pyotr Ilyich Tchaikovsky     103
The Black Keys               114
Thievery Corporation         122
Future                       124
Eminem                       147
Yuki Hayashi                 167
Capcom Sound Team            169
Drake                        172
Frédéric Chopin              241
Johann Sebastian Bach        314
Ludwig van Beethoven         317
Wolfgang Amadeus Mozart      402
Nobuo Uematsu                429
Name: artist_name, dtype: int64
```

artist_name holds information about the singers' name. The code line below proves the dataset contains information about 6863 artists. In general, this feature could be used for predicting music genre - no one expects to see Beatles in folk charts, or Mozart - in rock top 20s.

Draw barh chart

```python
plt.barh(artists.index, artists)
plt.xlabel("Number of songs per artist")
plt.title("Songs per artist")
plt.show()
```

✓ 0.3s Python



It seems the dataset was compiled by Japanese authors or in Japan since several artists in top20 are from the Land of the Rising Sun. Furthermore, many composers (e.g., Mozart, Beethoven, etc.) also found their place in this list. Now, to avoid large number of features, the artist_name and length_name is removed.

```python
data = data.drop(columns=['artist_name'])
```

✓ 0.8s Python

```python
# drop 'length_name' feature
data = data.drop(columns = ['length_name'])
```

✓ 0.1s Python

3. track_name column

Attribute description

```
data['track_name'].describe()
```

✓ 0.2s Python

```
count      50000
unique     41699
top         Home
freq          16
Name: track_name, dtype: object
```

track_name contains information about title of the song. The line of code below proves that the dataset contains information about 41699 songs because it has too many unique attributes and is not helpful for classification, otherwise it will cause an error for the algorithm so I will remove it.

```
data = data.drop(columns=['track_name'])
```

✓ 0.1s Python

4. popularity column

See the unique attributes

```
data.popularity.unique()
```

✓ 0.7s Python

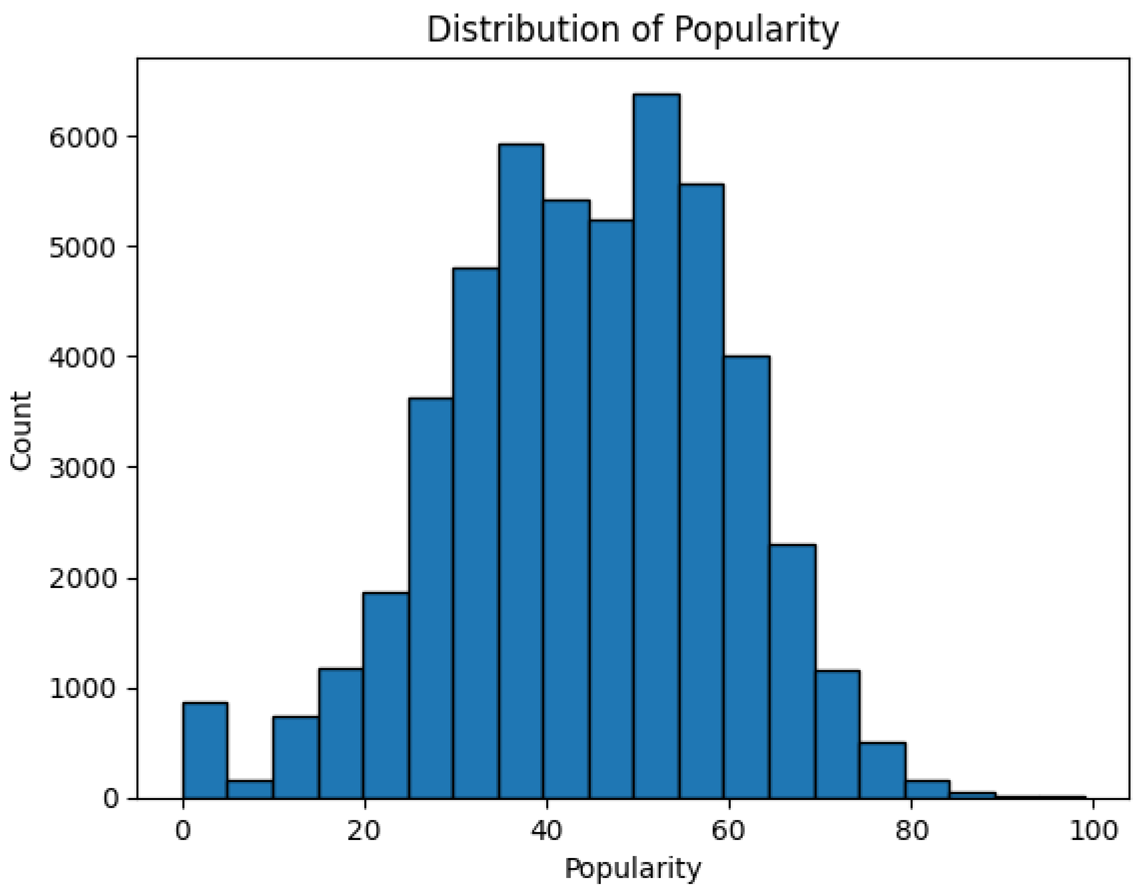
```
array([27., 31., 28., 34., 32., 47., 46., 43., 39., 22., 30., 50., 59.,
       29., 35., 44., 33., 56., 21., 48., 45., 53., 63., 25., 36., 37.,
       51., 55., 49., 41., 38., 52., 24., 42., 26., 96., 40., 23., 61.,
       54., 66., 70., 67., 60., 58., 65., 69., 72., 64., 62., 57.,  0.,
       76., 20., 74., 71., 84., 68., 18., 82.,  3., 11., 17., 15., 12.,
       10., 13., 16., 14.,  9., 19.,  8.,  7.,  4.,  2.,  1.,  5.,  6.,
       79., 73., 75., 78., 83., 81., 80., 77., 85., 97., 88., 87., 86.,
       99., 89., 93., 90., 94., 91., 95., 92.])
```

Has 100 unique properties that range from 0 – 99.

Next, let's see the distribution of the attributes in the dataset

```
plt.hist(data['popularity'], bins=20, edgecolor='black')
# Đặt tên cho trục x và y
plt.xlabel('Popularity')
plt.ylabel('Count')
# Đặt tiêu đề cho biểu đồ
plt.title('Distribution of Popularity')
# Hiển thị biểu đồ
plt.show()
```

✓ 0.3s Python



Attribute description

```python
data['popularity'].describe()
```

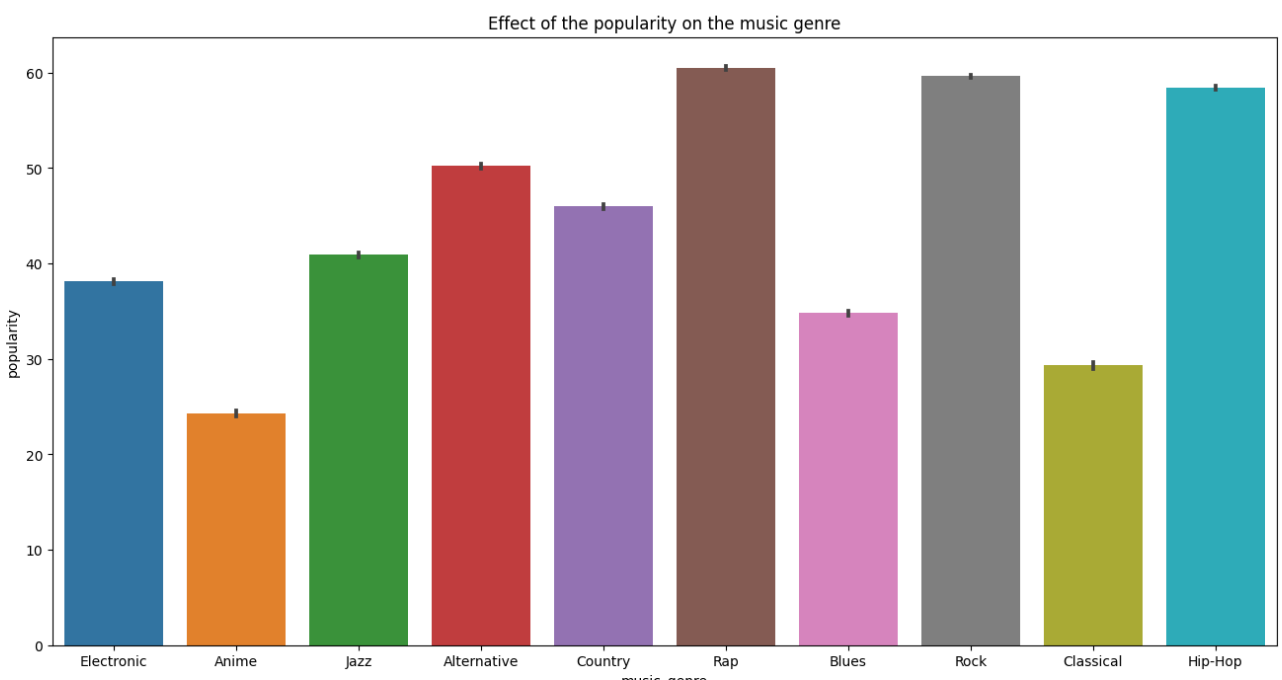
✓ 0.1s Python

```
count    50000.000000
mean        44.220420
std         15.542008
min          0.000000
25%         34.000000
50%         45.000000
75%         56.000000
max         99.000000
Name: popularity, dtype: float64
```

The histogram shows the distribution of the data's values

```python
plt.figure(figsize=(16,8))
sns.barplot(x = 'music_genre', y = 'popularity', data = data)
plt.title('Effect of the popularity on the music genre')
plt.show()
```

✓ 1.2s Python



This feature shows a nice spread of distributions for the different genres. Could definitely be useful for classification.

Rap, Hip-Hop and Rock seem to be the most popular genres, while Anime, Blues and Classical are the least popular. The other 4 genres are somewhere in between.

5. acousticness column

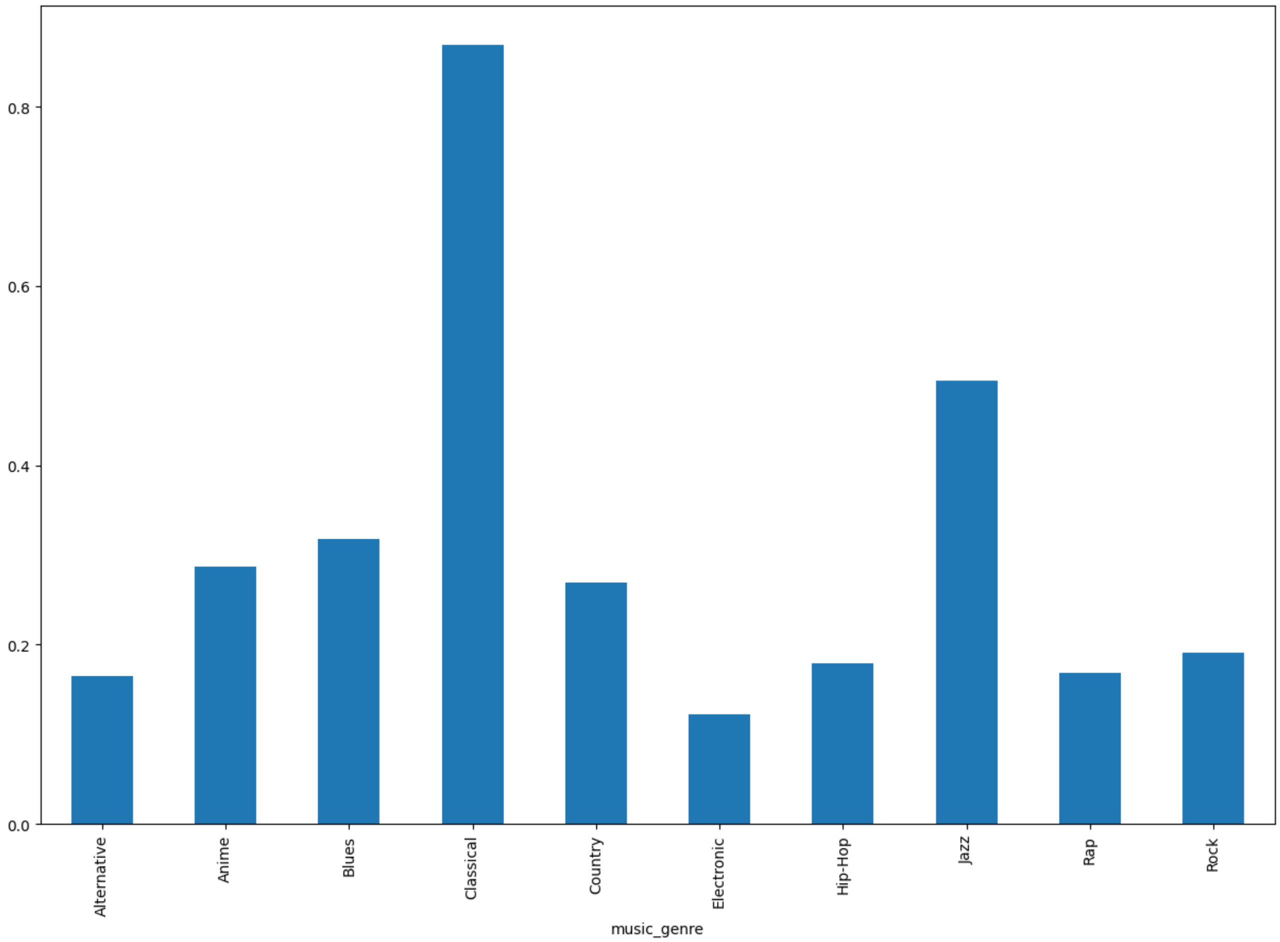
Statistics the data of acousticness

```
data['acousticness'].describe()
```
✓ 0.7s Python

```
count    50000.000000
mean         0.306383
std          0.341340
min          0.000000
25%          0.020000
50%          0.144000
75%          0.552000
max          0.996000
Name: acousticness, dtype: float64
```

The histogram shows the distribution of the data's values

```
df2=data.groupby(["music_genre"]).mean()
df2["acousticness"].plot(kind='bar', figsize=(15,10))
```
✓ 0.7s Python

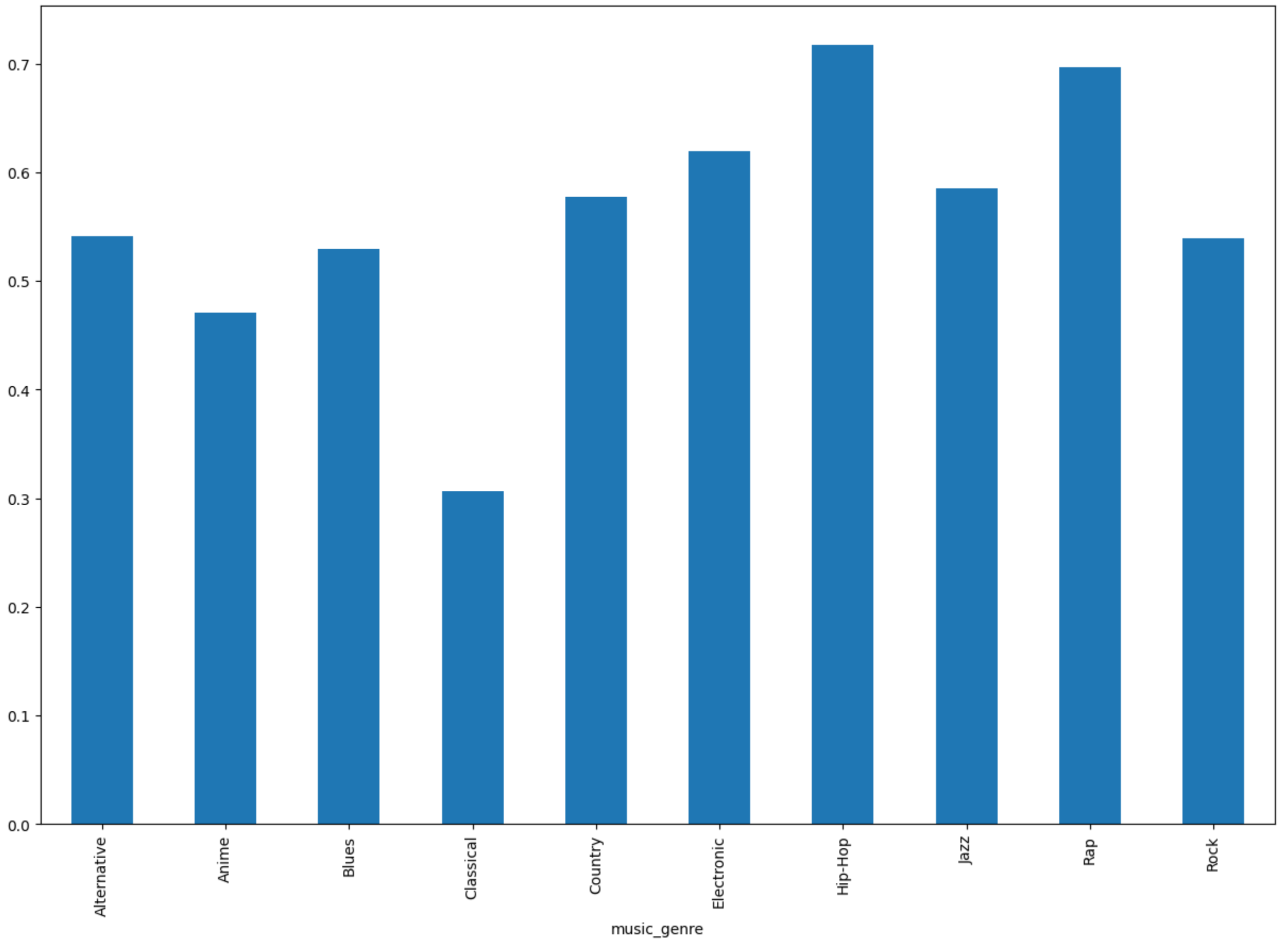


Interestingly, classical and jazz music has a higher percentage of acousticness than the rest.

6. danceability column

```
df2=data.groupby(["music_genre"]).mean()
df2["danceability"].plot(kind='bar', figsize=(15,10))
```

✓ 0.4s Python



Classical music sticks out again, but Rap and Hip-Hop can also be distinguished from the rest (they seem to go together often).

7. duration_ms column

Statistics the data between the two columns are duration_ms and music_genre.

```
data.groupby('music_genre')['duration_ms'].describe()
```
✓ 0.1s Python

music_genre	count	mean	std	min	25%	50%	75%	
Alternative	5000.0	210404.8078	90366.002886	-1.0	185994.75	219561.5	255583.00	6
Anime	5000.0	208880.8290	105275.946177	-1.0	146250.25	230205.5	272993.25	12
Blues	5000.0	229301.0962	131931.237262	-1.0	171910.00	221140.0	280033.25	20
Classical	5000.0	278014.3464	219698.722816	-1.0	148770.25	241640.0	362070.25	31
Country	5000.0	195556.0686	77252.730320	-1.0	179328.25	207073.5	233839.25	5
Electronic	5000.0	244553.3832	164125.128913	-1.0	192655.50	236888.0	300000.00	48
Hip-Hop	5000.0	198395.9458	86465.775861	-1.0	167707.00	209056.0	248627.00	7
Jazz	5000.0	238092.4468	133485.337615	-1.0	170491.00	236040.0	304278.50	13
Rap	5000.0	196508.7920	85618.048197	-1.0	168339.00	207509.5	244454.25	5
Rock	5000.0	212818.3128	94290.281753	-1.0	186396.75	218793.0	257249.25	8

-1.0 is obvously not a valid time measurement. These are missing values.

```
miss_duration = data[data['duration_ms'] == -1].shape[0]
num_obs_tot = data.shape[0]
print(f"There are {miss_duration} missing values, which accounts for {(miss_durat
```
✓ 0.1s Python

```
There are 4939 missing values, which accounts for 9.878% of the data points.
```

Nearly 10% of entries are missing the duration and comparing with the statistics most of the tracks are about the same which is not valid for classification so we will proceed to remove the feature

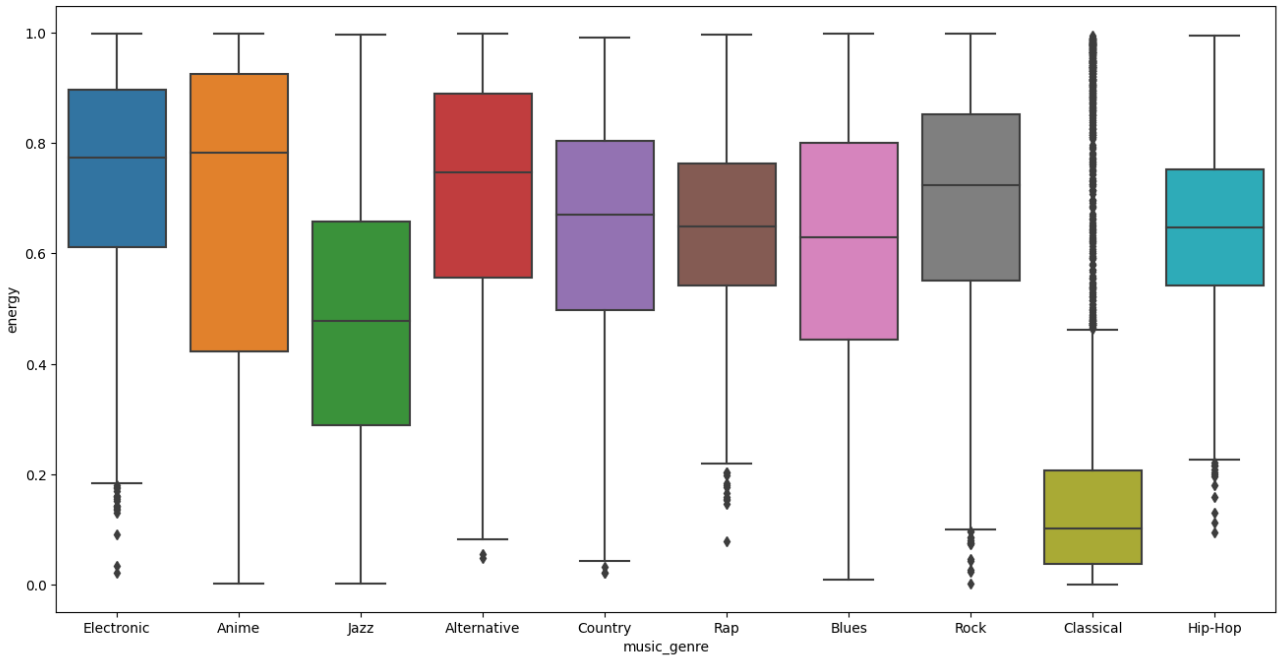
```
# drop 'duration_ms' feature
data = data.drop(columns = ['duration_ms'])
```
✓ 0.1s Python

8. energy column

The histogram shows the distribution of the data's values

```python
plt.figure(figsize=(16,8))
sns.boxplot(data=data, x='music_genre', y='energy')
plt.show()
```

✓ 0.6s Python



As usual, classical music stands out (and Jazz to a much lesser degree). Rap and Hip-Hop still match each other.

9. instrumentalness column

See the distribution of the data's values.

```python
data['instrumentalness'].value_counts()
```

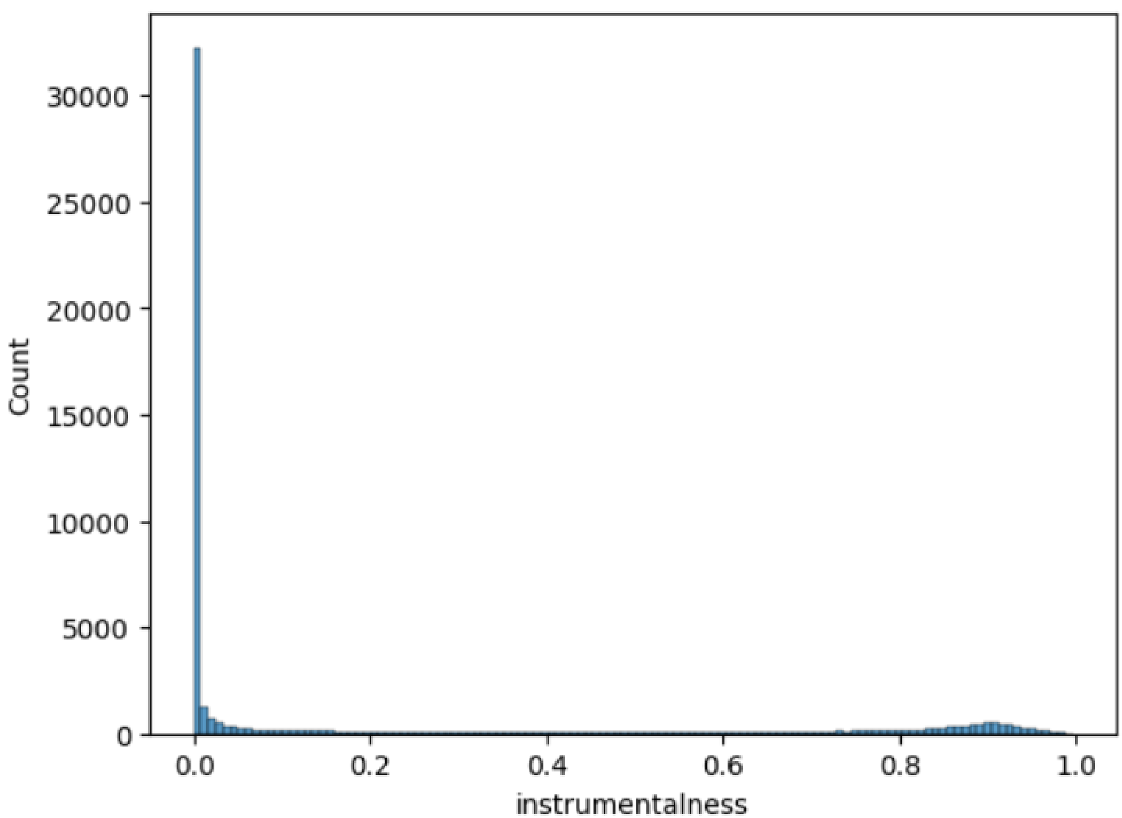
✓ 0.2s Python

```
0.000000    15001
0.898000       70
0.902000       69
0.897000       66
0.912000       66
            ...
0.000049        1
0.000876        1
0.000094        1
0.000787        1
0.000926        1
Name: instrumentalness, Length: 5131, dtype: int64
```

The histogram shows the distribution of the data's values

```python
sns.histplot(x='instrumentalness',data=data);
```

✓ 1.5s Python



Such a large number of 0.0 entries likely indicates missing values rather than real data points, we won't fill in missing values. Instead, we'll discard this feature entirely.

```
data = data.drop(columns=['instrumentalness'])
```
✓ 0.2s Python

10. key column

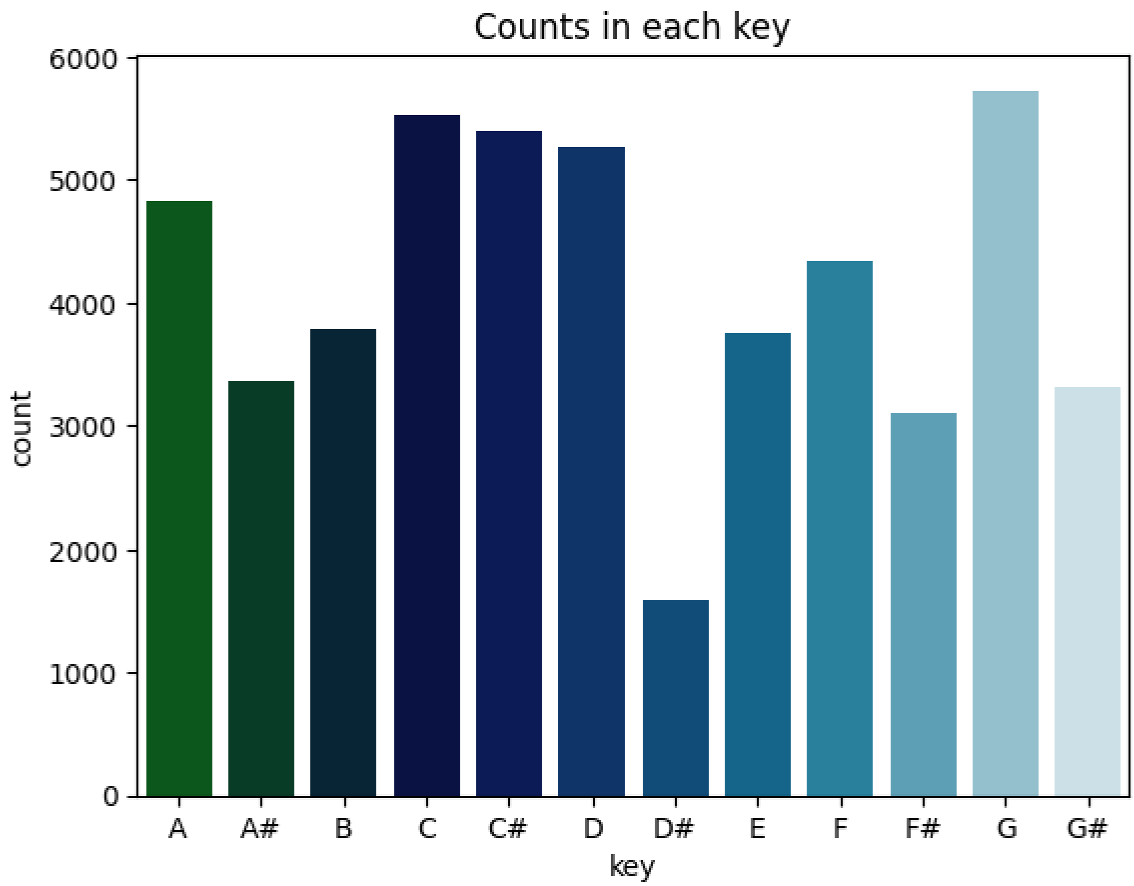
Check the unique attributes

```
data['key'].unique()
```
✓ 0.9s Python

```
array(['A#', 'D', 'G#', 'C#', 'F#', 'B', 'G', 'F', 'A', 'C', 'E', 'D#'],
      dtype=object)
```

The histogram shows the distribution of the data's values

```
def plot_counts(feature, order = None):
    sns.countplot(x = feature, data = data, palette = "ocean", order = order)
    plt.title(f"Counts in each {feature}")
    plt.show()
plot_counts("key", ["A", "A#", "B", "C", "C#", "D", "D#", "E", "F", "F#", "G", "G#
```
✓ 0.2s Python

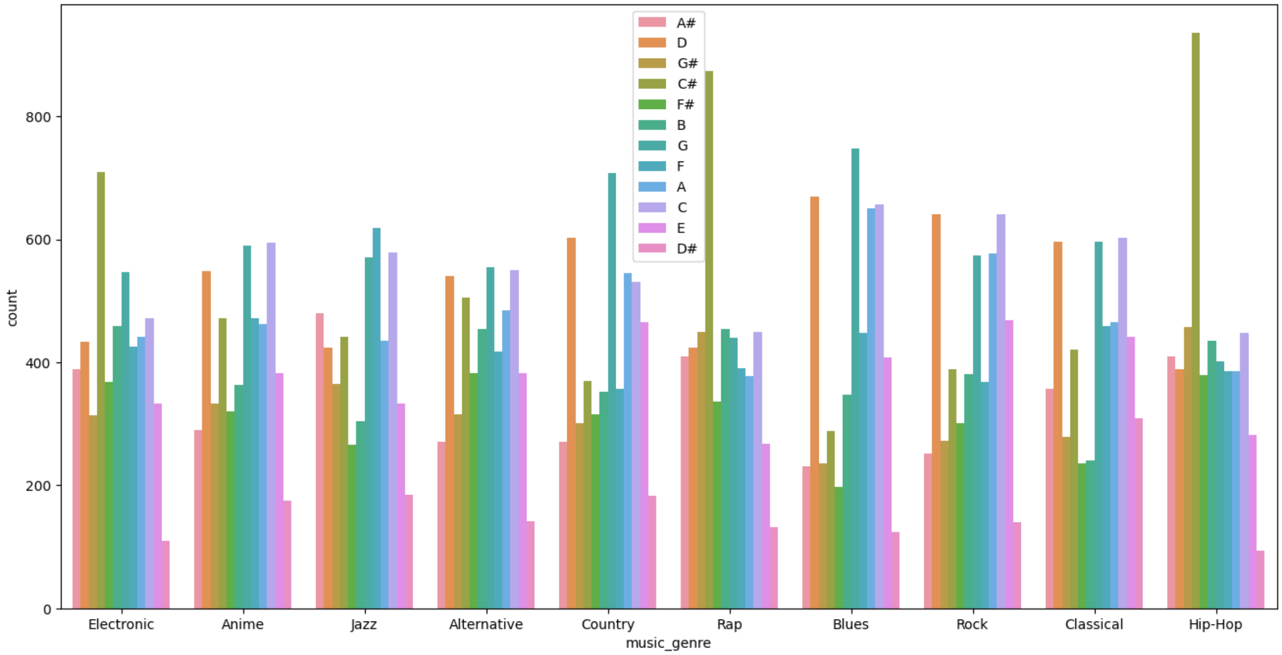


Different genres have noticeably different spreads.

Draw a histogram showing the distribution of key attribute values with music_genre

```
plt.figure(figsize=(16,8))
sns.countplot(x="music_genre", hue="key",data=data)
plt.legend(loc=0)
plt.show()
```

✓ 0.8s Python



- We can see with some genres like electronic, rap, hiphop, key = C# is the main
- Country and blues, key = D or key = G is the majority
- Key = D# is quite low in all genres
- The rest of the distribution is quite even
- We'll keep this feature, but use encode to make it useful.

```
# Encode key feature
from sklearn.preprocessing import LabelEncoder
key_encoder = LabelEncoder()
data["key"] = key_encoder.fit_transform(data["key"])
```

✓ 0.2s Python

Data before encoder

```python
# Data before encoder
data['key'].value_counts()
```

✓ 0.1s Python

```
G      5727
C      5522
C#     5405
D      5265
A      4825
F      4341
B      3789
E      3760
A#     3356
G#     3319
F#     3101
D#     1590
Name: key, dtype: int64
```

Data after encoder

```python
# Data after encoder
data['key'].value_counts()
```

✓ 0.9s Python

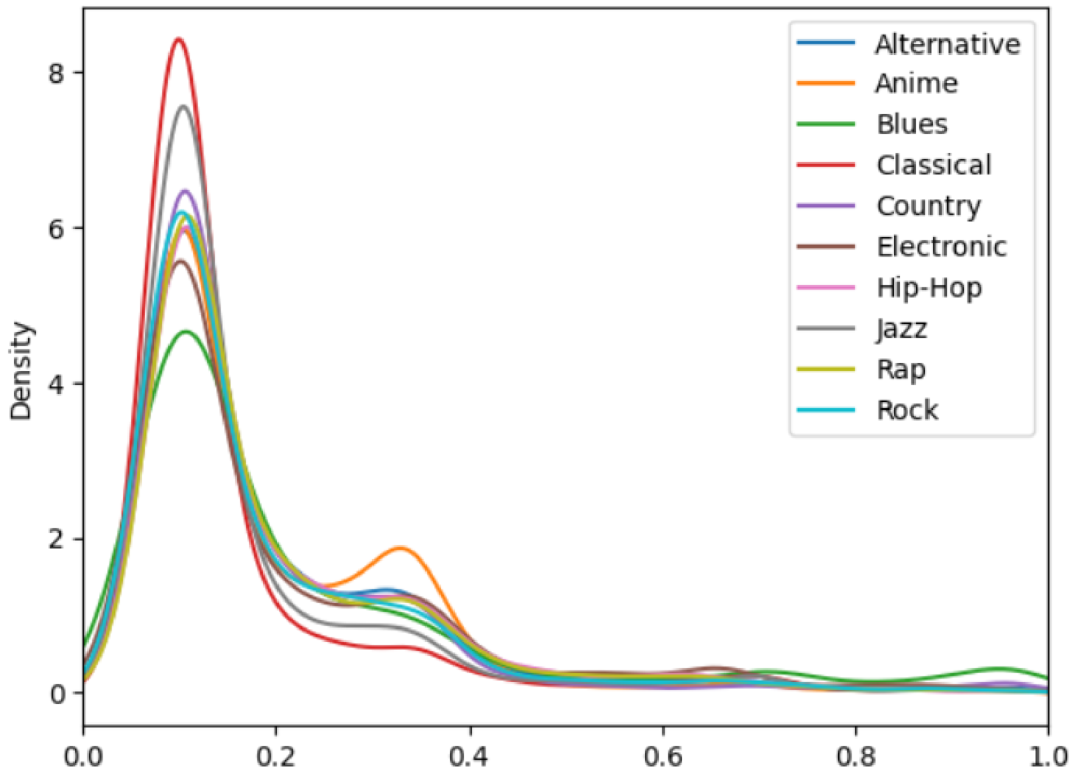
```
10     5727
3      5522
4      5405
5      5265
0      4825
8      4341
2      3789
7      3760
1      3356
11     3319
9      3101
6      1590
```

11. liveness column

The histogram shows the distribution of the data's values

```
data.groupby('music_genre')['liveness'].plot.kde()
plt.legend()
plt.xlim([0,1])
plt.show()
```

✓ 3.6s Python



The distributions seem similarly skewed for all genres, so this feature will likely not contribute much to the model. We'll discard this feature entirely.

```
data = data.drop(columns=['liveness'])
```

✓ 0.1s Python

12. loudness column

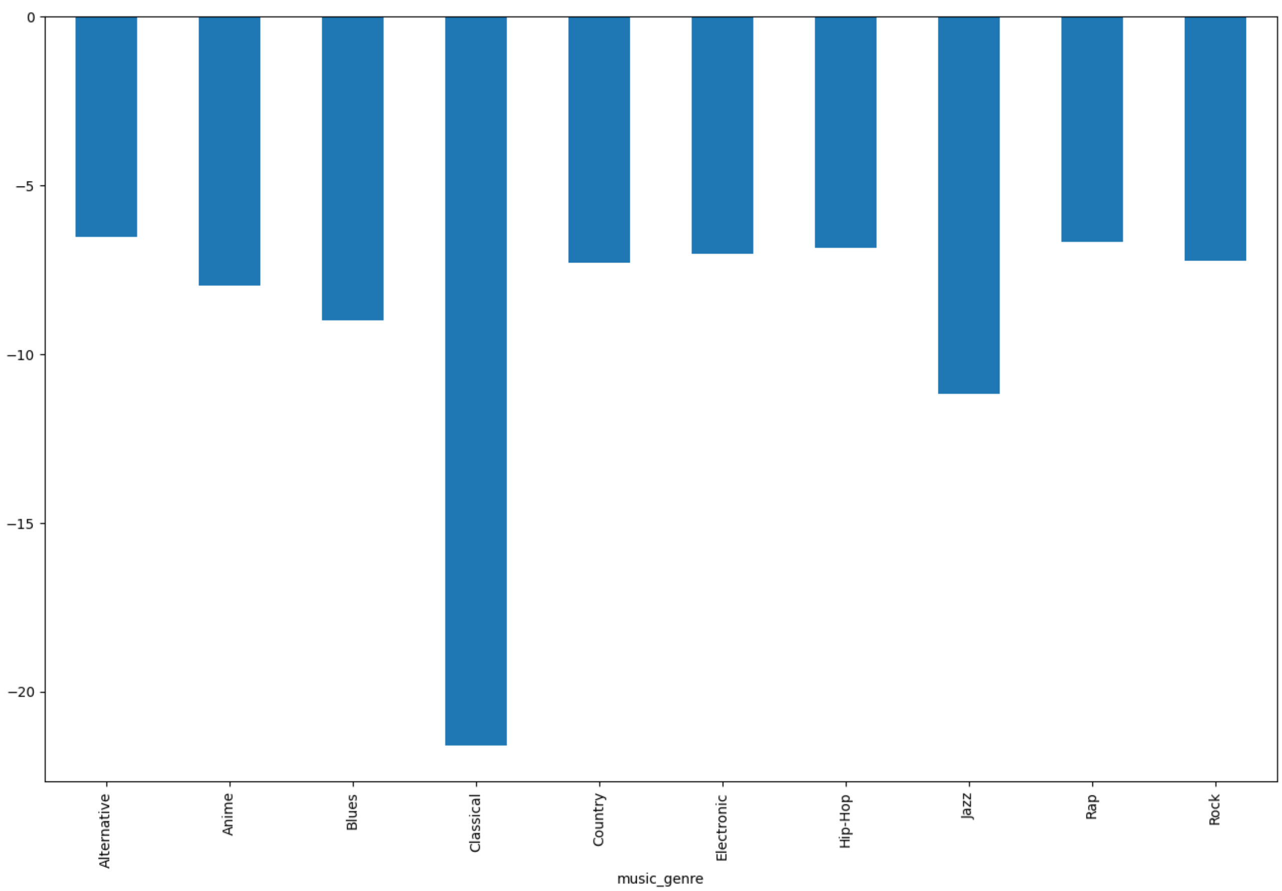
Attribute description

```
data['loudness'].describe()
```
✓ 0.9s Python

```
count    50000.000000
mean        -9.133761
std          6.162990
min        -47.046000
25%        -10.860000
50%         -7.276500
75%         -5.173000
max          3.744000
Name: loudness, dtype: float64
```

The histogram shows the distribution of the data's values

```
df1=data.groupby(["music_genre"]).mean()
df1["loudness"].plot(kind='bar', figsize=(16,10))
```
✓ 0.4s Python



As usual, classical music is far from the rest, with Jazz (and Blues) also differing from the rest somewhat.

13. mode column

Check the unique values

```
data['mode'].unique()
```

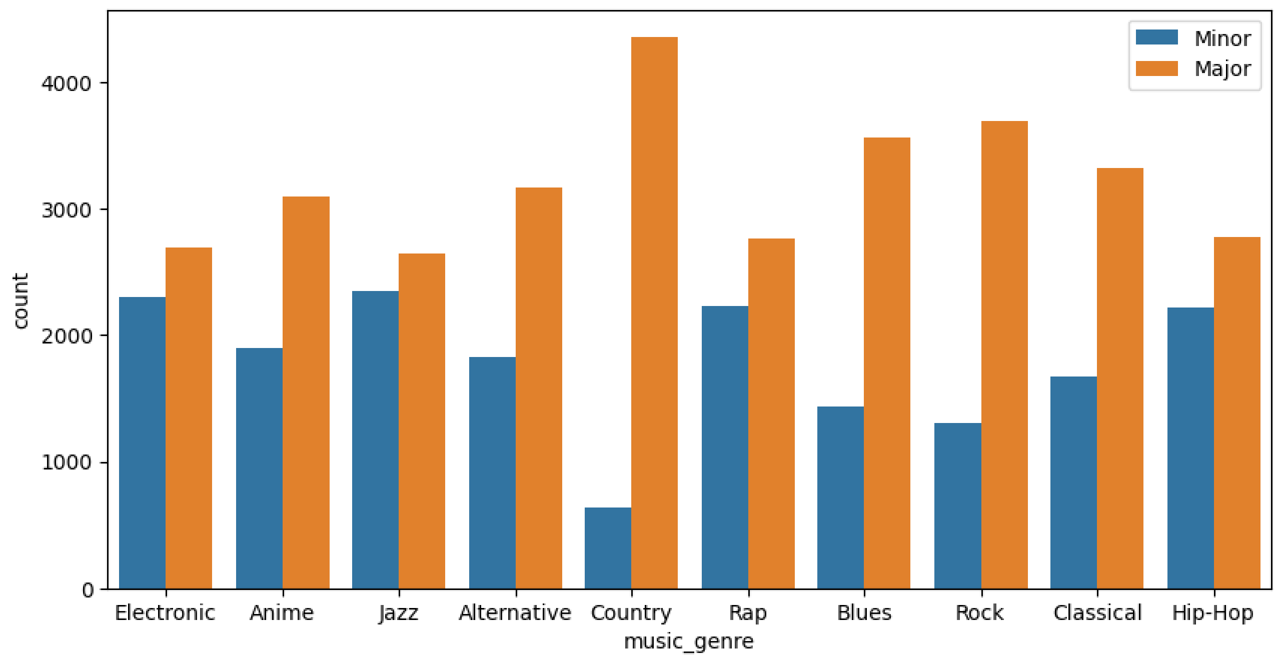
✓ 0.1s Python

```
array(['Minor', 'Major'], dtype=object)
```

The histogram shows the distribution of the data's values

```
plt.figure(figsize=(10,5))
sns.countplot(x="music_genre", hue="mode",data=data)
plt.legend(loc=0)
plt.show()
```

✓ 0.5s Python



All genres seem to have a prefererence for the "Major" mode, but to different degrees. It is the most pronounced in the Country genre. We'll use this feature after encoding.

```
# Encode mode feature
key_encoder = LabelEncoder()
data["mode"] = key_encoder.fit_transform(data["mode"])
```
✓ 0.1s Python

Data before encoder

```
# Data before encoder
data['mode'].value_counts()
```
✓ 0.1s Python

```
Major    32099
Minor    17901
Name: mode, dtype: int64
```

Data after encoder

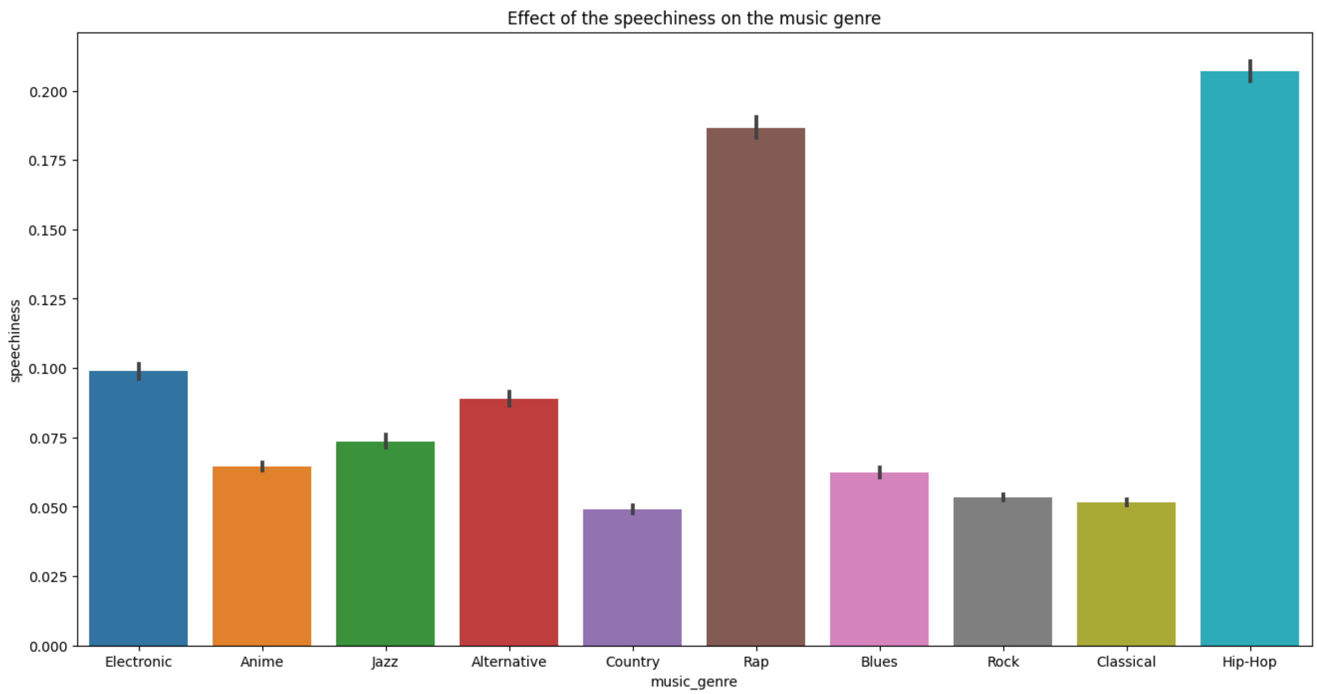
```
# Data after encoder
data['mode'].value_counts()
```
✓ 0.1s Python

```
0    32099
1    17901
```

14. speechiness column

The lineplot histogram shows the distribution of the data's values

```
plt.figure(figsize=(16,8))
sns.barplot(x = 'music_genre', y = 'speechiness', data = data)
plt.title('Effect of the speechiness on the music genre')
plt.show()
```
Python



This feature should contribute especially to identifying Hip-Hop and Rap.

15. tempo column

Statistics the data between the two columns are tempo and music_genre

```python
data.groupby('music_genre')['tempo'].describe()
```

✓ 0.1s Python

music_genre	count	unique	top	freq
Alternative	5000	4236	?	505
Anime	5000	4134	?	503
Blues	5000	4376	?	530
Classical	5000	4380	?	500
Country	5000	4233	?	514
Electronic	5000	3644	?	534
Hip-Hop	5000	4115	?	480
Jazz	5000	4302	?	479
Rap	5000	4076	?	496
Rock	5000	4341	?	439

This feature should be numeric. The "?" is a missing value.

```
print(f"This feature contains {(data[data['tempo'] == '?'].shape[0]/data.shape[0])
```
✓ 0.1s Python

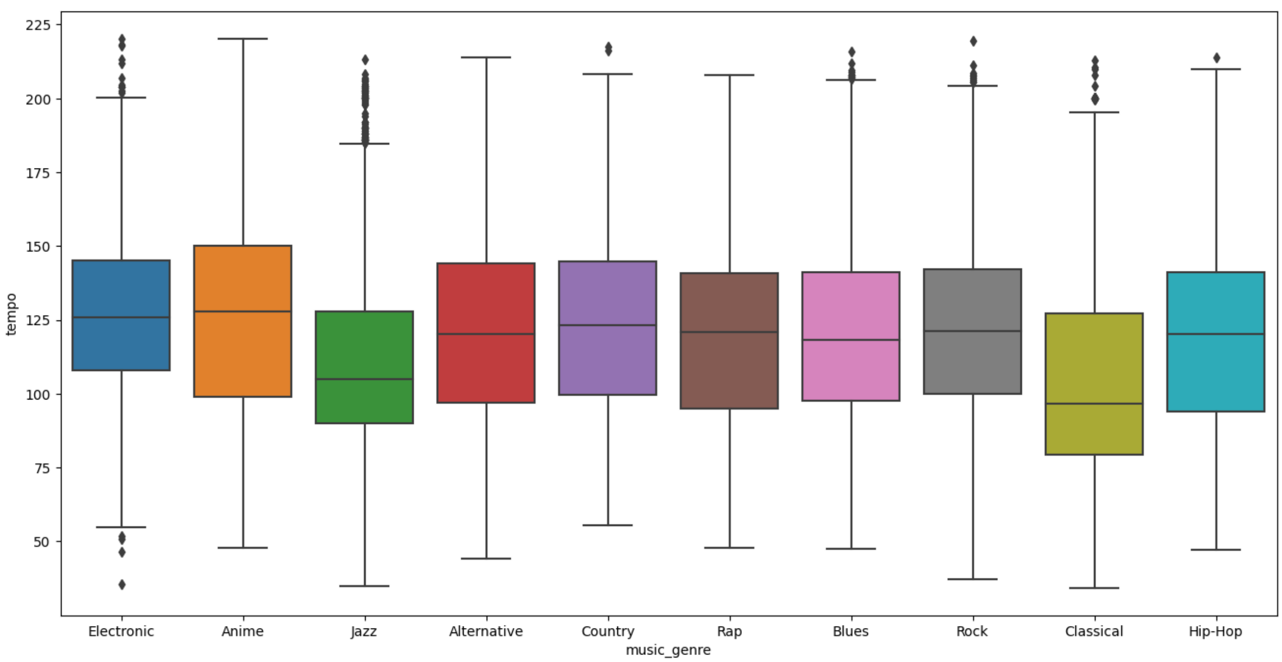
```
This feature contains 9.96% missing values
```

Replace "?" with np.nan and correctly classify the feature.

```
# replace "?" with np.nan and correctly classify the feature:
data.loc[data['tempo'] == '?', 'tempo'] = np.nan
data = data.astype({'tempo': np.float64})
data["tempo"] = np.around(data["tempo"], decimals = 2)
```
✓ 0.9s Python

The boxplot histogram shows the distribution of the data's values

```
plt.figure(figsize=(16,8))
sns.boxplot(data=data, x='music_genre', y='tempo')
plt.show()
```
✓ 0.5s Python



The variation between genres is not great so we'll drop the feature.

```
data = data.drop(columns=['tempo'])
```
✓ 0.2s Python

16. obtained_date column

Check the unique values

```
data['obtained_date'].unique()
```
✓ 0.1s Python

```
array(['4-Apr', '3-Apr', '5-Apr', '1-Apr', '0/4'], dtype=object)
```

Only gives the 4 dates at which the data was obtained. Not useful to us, so we'll drop it.

```
data = data.drop(columns=['obtained_date'])
```
✓ 0.8s Python

17. valence column

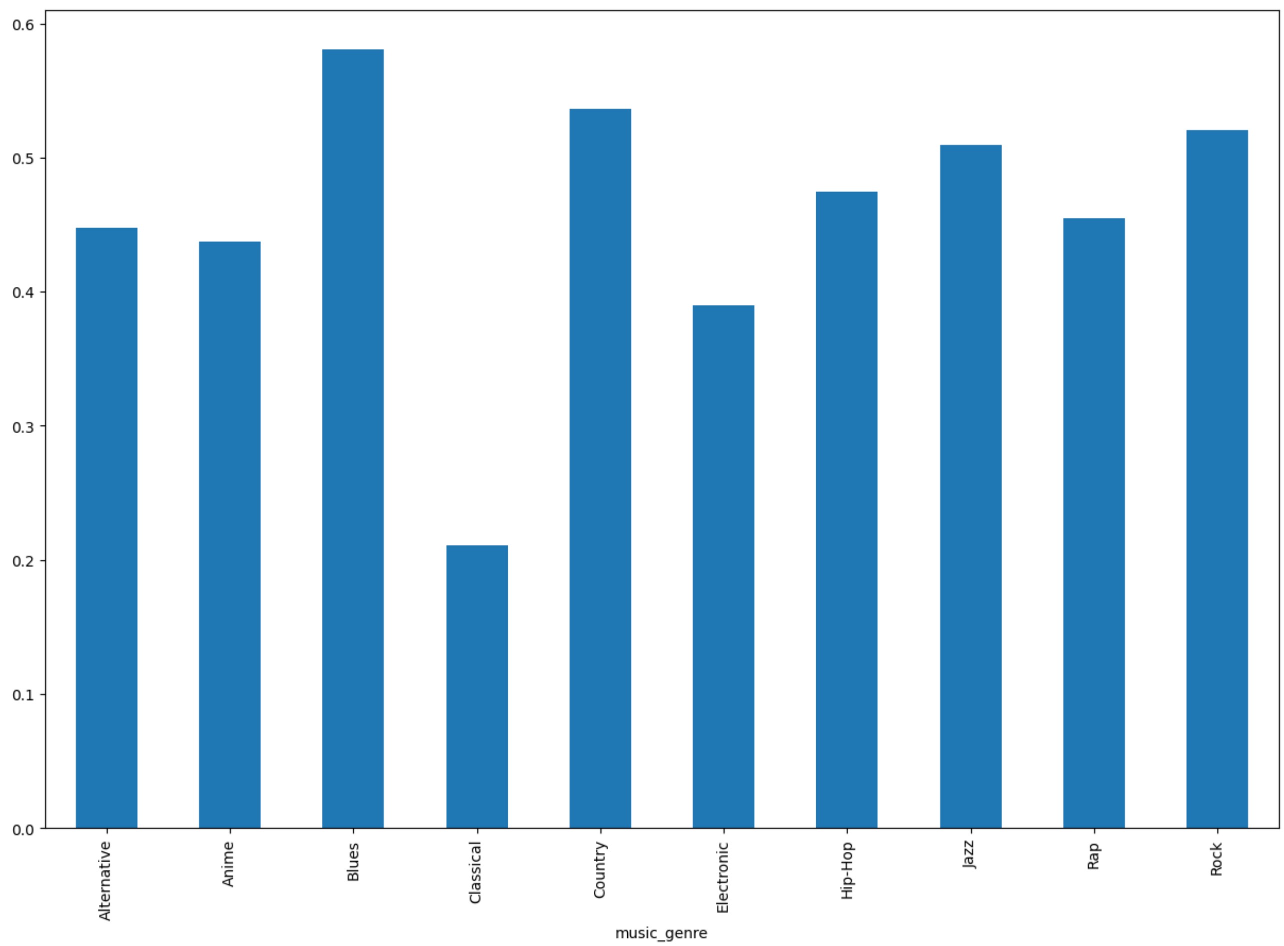
Statistics the data between the two columns are valence and music_genre

```
data.groupby('music_genre')['valence'].describe()
```
Python

	count	mean	std	min	25%	50%	75%	max
music_genre								
Alternative	5000.0	0.447513	0.216445	0.0342	0.28300	0.4285	0.60000	0.983
Anime	5000.0	0.437670	0.248353	0.0000	0.23200	0.4390	0.63200	0.990
Blues	5000.0	0.580788	0.224741	0.0315	0.41000	0.5900	0.76000	0.985
Classical	5000.0	0.210523	0.197650	0.0000	0.05650	0.1400	0.30525	0.982
Country	5000.0	0.536732	0.221114	0.0396	0.36100	0.5270	0.71625	0.977
Electronic	5000.0	0.389884	0.239673	0.0205	0.18900	0.3585	0.55900	0.992
Hip-Hop	5000.0	0.474927	0.220622	0.0336	0.30300	0.4735	0.64300	0.979
Jazz	5000.0	0.509248	0.251076	0.0289	0.29675	0.5150	0.71100	0.985
Rap	5000.0	0.454999	0.213480	0.0336	0.28900	0.4460	0.61200	0.970
Rock	5000.0	0.520361	0.233627	0.0277	0.34000	0.5160	0.70300	0.985

The bar histogram shows the distribution of the data's values

```
df1=data.groupby(["music_genre"]).mean()
df1["valence"].plot(kind='bar', figsize=(15,10))
```
✓ 0.3s Python



Again, only classical music truly stands out from the rest.

18. music_genre column

Group some genres with the same parameters together

```python
# Group some genres with the same parameters together
data.music_genre = data.music_genre.replace('Rock', 'Rock, Alternative or Country')
data.music_genre = data.music_genre.replace('Alternative', 'Rock, Alternative or Country'
data.music_genre = data.music_genre.replace('Country', 'Rock, Alternative or Country')
data.music_genre = data.music_genre.replace('Rap', 'Rap or Hip-Hop')
data.music_genre = data.music_genre.replace('Hip-Hop', 'Rap or Hip-Hop')
data.music_genre = data.music_genre.replace('Jazz', 'Jazz, Blues or Electronic')
data.music_genre = data.music_genre.replace('Blues', 'Jazz, Blues or Electronic')
data.music_genre = data.music_genre.replace('Electronic', 'Jazz, Blues or Electronic')
```

✓ 0.1s Python

Review properties after grouping

```python
data['music_genre'].value_counts()
```

✓ 0.1s Python

```
Jazz, Blues and Electronic       15000
Rock, Alternative and Country    15000
Rap and Hip-Hop                  10000
Anime                             5000
Classical                         5000
Name: music_genre, dtype: int64
```

CHAPTER III: ALGORITHMS AND EXPERIMENTS

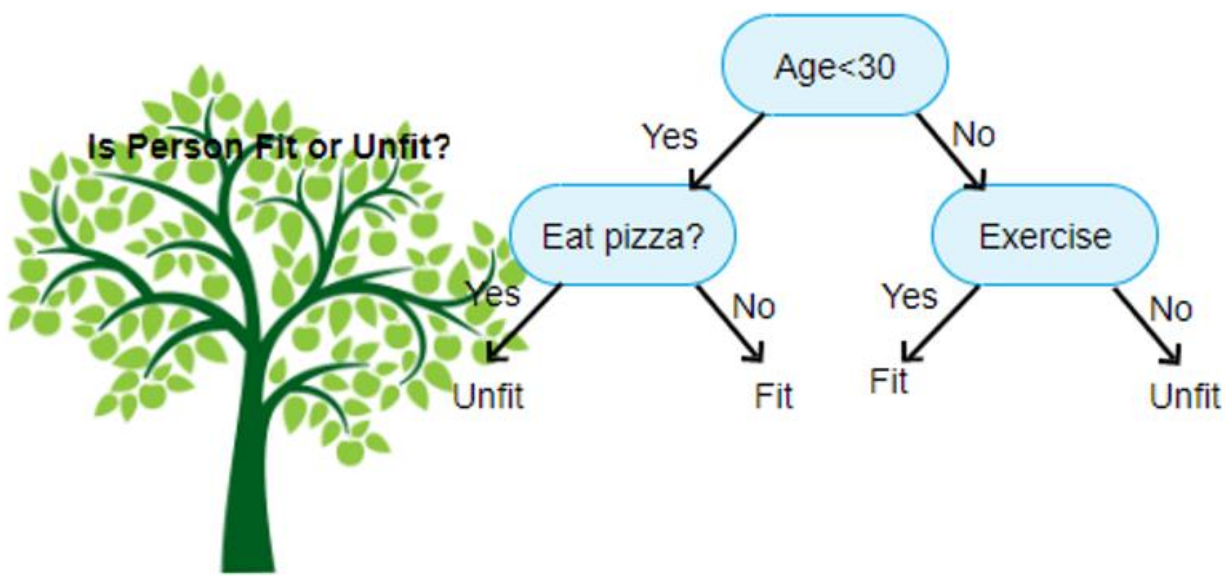
3.1 Algorithm used

3.1.1 Decision Tree

- A decision tree is a tree structure such that:

- Each node in the network corresponds to a test on an attribute.
- Each branch represents the test result.
- Leaf nodes represent classes or class distributions.
- The highest node in the tree is the root node.

- Decision tree shap:



- Basic strategy:

- Start from single node showing all samples.
- If the samples belong to the same class, the node becomes a leaf node and is labeled with that class.

- In contrast, using the attribute measure to select the attribute will best separate the samples into classes.
- A branch is created for each value of the selected attribute and the samples are partitioned by use the same process recursively to create a decision tree.
- The process ends only if any of the following conditions are true.

- All templates for a given node belong to the same class

- There are no more attributes that the sample can rely on for further partitioning

- No samples left at node

- ID3 is an algorithm used in decision trees. This algorithm uses information gain to build a decision tree. The largest Information Gain attribute will be selected as the root node

- Information Gain:

$$Info(S) = E(S) = -\sum_{i=1}^{n} f_S(A_i) \log_2 f_S(A_i)$$

- Amount of information needed to classify an element in S based on attribute A: InfoA(S)

$$Info_A(S) = -\sum_{i=1}^{v} \frac{|S_j|}{|S|} Info(S_i)$$

- Information gain is the difference between the original Info(S) information value (before partitioning) and the new InfoA(S) information value (after partitioning with A)

$$Gain(A) = Info(S) - Info_A(S)$$

CART: Unlike ID3 which uses Information Cain formula, Cart algorithm uses Gini formula. The attribute with the smallest Gini value will be the root node

- Gini index of the set S:

$$Gini(S) = 1 - \sum_{j} p(j|S)^2$$

P(j|S) is the frequency of j in S

- Gini of attribute:

$$Gini_A(S) = \sum_{i=1}^{k} \frac{n_i}{n} Gini(i)$$

In case: ni is the number of samples in note I, n is the number of samples in note A

3.1.2 Random Forest

3.1.2.1 Random Forest explain

Random Forest is one of the most popular and commonly used algorithms by Data Scientists. **Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems**. It builds decision trees on different samples and takes their majority vote for classification and average in case of regression.

One of the most important features of the Random Forest Algorithm is that it can handle the data set containing continuous variables, as in the case of regression, and categorical variables, as in the case of classification. It performs better for classification and regression tasks.

❖ **Steps Involved in Random Forest Algorithm**

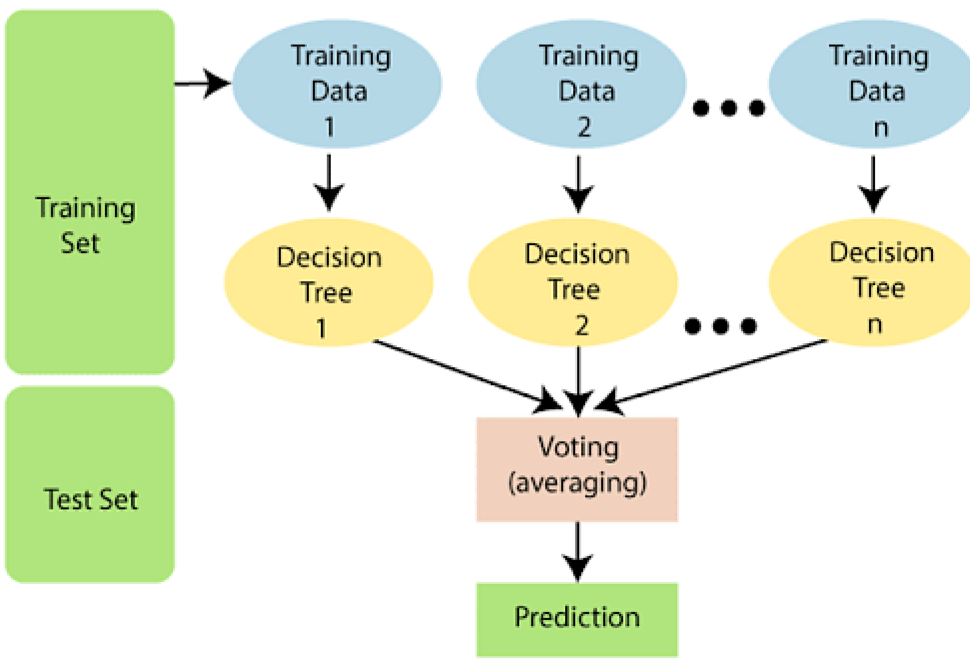
Step 1: In the Random forest model, a subset of data points and a subset of features is selected for constructing each decision tree. Simply put, n random records and m features are taken from the data set having k number of records.

Step 2: Individual decision trees are constructed for each sample.

Step 3: Each decision tree will generate an output.

Step 4: Final output is considered based on Majority Voting or Averaging for Classification and regression, respectively.

3.1.2.2 Random Forests Algorithm



The following steps explain the working Random Forest Algorithm:

Step 1: Select random samples from a given data or training set.

Step 2: This algorithm will construct a decision tree for every training data.

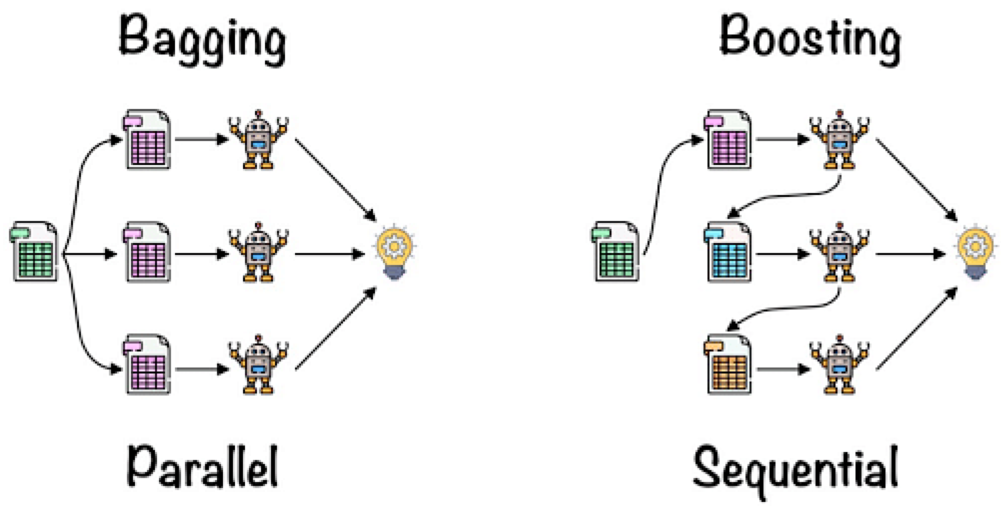
Step 3: Voting will take place by averaging the decision tree.

Step 4: Finally, select the most voted prediction result as the final prediction result.

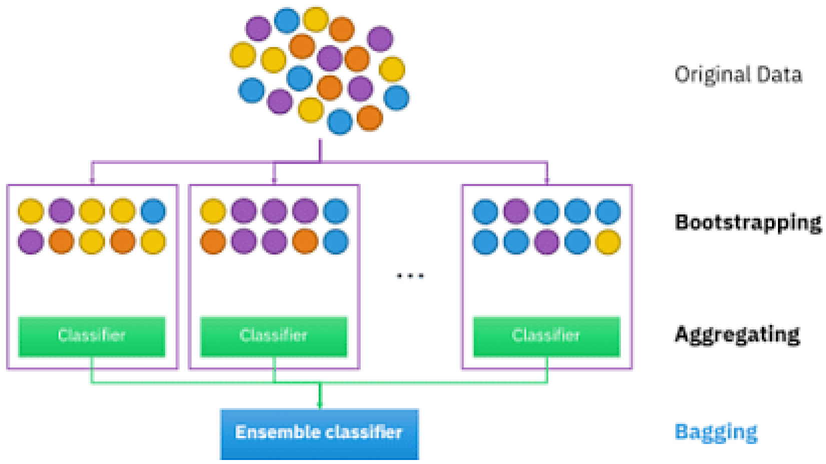
This combination of multiple models is called Ensemble. Ensemble uses two methods:

- Bagging: Creating a different training subset from sample training data with replacement is called Bagging. The final output is based on majority voting.

- Boosting: Combing weak learners into strong learners by creating sequential models such that the final model has the highest accuracy is called Boosting. Example: ADA BOOST, XG BOOST.



BaggingBagging is also known as Bootstrap Aggregation used by random forest. The process begins with any original random data. After arranging, it is organised into samples known as Bootstrap Sample. This process is known as Bootstrapping.Further, the models are trained individually, yielding different results known as Aggregation. In the last step, all the results are combined, and the generated output is based on majority voting. This step is known as Bagging and is done using an Ensemble Classifier.



❖ Essential Features of Random Forest

- Miscellany: Each tree has a unique attribute, variety and features concerning other trees. Not all trees are the same.

- Immune to the curse of dimensionality: Since a tree is a conceptual idea, it requires no features to be considered. Hence, the feature space is reduced.

- Parallelization: We can fully use the CPU to build random forests since each tree is created autonomously from different data and features.

- Train-Test split: In a Random Forest, we don't have to differentiate the data for train and test because the decision tree never sees 30% of the data.

- Stability: The final result is based on Bagging, meaning the result is based on majority voting or average.

❖ How Random Forest is applied?

Random Forest has a wide range of applications across various domains due to its versatility and robustness.

- Classification Problems: Random Forest is often used for classification tasks, such as spam detection, sentiment analysis, customer churn prediction, disease diagnosis, and image classification. Its ability to handle both numerical and categorical features makes it suitable for diverse datasets.

- Regression Problems: Random Forest can be applied to regression tasks, including predicting housing prices, stock market trends, energy consumption, and demand forecasting. It can capture complex non-linear relationships between input variables and the target variable.

- Feature Importance: Random Forest provides a measure of feature importance, which can be utilized for feature selection in data preprocessing. This information helps identify the most relevant features for prediction and can guide feature engineering efforts.

- Anomaly Detection: Random Forest can be used for anomaly detection by training on normal data and identifying instances that deviate significantly from the learned patterns. This is useful in fraud detection, network intrusion detection, and detecting unusual behaviors in various domains.

- Ensemble Learning: Random Forest is a type of ensemble learning method, which combines multiple models to improve overall performance. It can be used as a base learner in ensemble techniques such as bagging, boosting, and stacking to further enhance predictive accuracy.

- Imputation of Missing Values: Random Forest can handle missing data effectively. It can be used to impute missing values in a dataset by using the available features to predict missing values, making it valuable for data preprocessing tasks.

- Recommender Systems: Random Forest can be employed in building recommendation systems to suggest relevant products, movies, or content to users based on their preferences and behavior.

- Bioinformatics and Genomics: Random Forest finds applications in analyzing DNA sequences, gene expression data, and protein-protein interactions. It can be used for tasks like gene expression classification, protein structure prediction, and identifying disease biomarkers.

3.1.3 Naive Bayes

3.1.3.1 Bayes Theorem

- Bayes' Theorem (Bayes' Theorem) is a mathematical theorem that calculates the probability of a random event A, given that the related event B has occurred.

- This theorem is named after the 18th century English mathematician Thomas Bayes.

- This is one of the extremely useful tools, a close friend of Data Scientists who work in data science.

- Bayes theorem allows to calculate the probability of a random event A given that related event B has occurred. This probability is denoted P(A|B) and read as "the probability of A if there is B". This quantity is called conditional probability or posterior probability because it is derived from a given value of B or depends on that value.

- According to Bayes' theorem, the probability that A occurs when B is known will depend on 3 factors:

- **The probability that A occurs on its own, regardless of B**. It is denoted by P(A) and read as the probability of A. This is called the marginal probability or a priori probability, it is "a priori" " in the sense that it is not interested in any information about B.
- **Probability of occurring B on its own, regardless of A**. It is denoted by P(B) and read as "probability of B". This quantity is also called a normalizing constant because it is always the same, regardless of the event A is trying to know.

- Probability of B happening when A is known. It is denoted by P(B|A) and read as "probability of B if there is A". This quantity is called the likelihood that B will occur, given that A has occurred. Pay attention not to confuse the probability that B will occur when A is known and the probability that A will occur when B is known.

- We can restate it with the following formula: The probability that A and B occur at the same time is:

$$P(A,B) = P(A)\ P(B)$$

- In case:

- P(A)P(A) is the probability of a distinct A occurring.
- P(B)P(B) is the probability that B occurs separately.

- If A and B are two related events, and the probability that event B occurs is greater than 0, we can define the probability that A will occur, given that B occurs as follows:

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

- Bayes theorem is based on the definition of conditional probability above, expressed in the form of a formula as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

The symbol ¬A is not A (or A's complement). We have P(A)+P(¬A) = 1

From there: P(B) =P(B,A) + P(B,¬A) = P(B|A)P(A) + P(B|¬A)P(¬A)

Bayes' theorem is written in variant form as follows:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|\neg A)P(\neg A)}$$

3.1.3.2 naïve Bayes classification algorithm

- Naive Bayes Classification (NBC) is a classification algorithm based on probability calculation applying Bayes theorem. This algorithm belongs to the group of supervised learning algorithms.

- Each data sample is represented by X=(x1, x2,..., xn) with attributes A1, A2,..., An

- Grades C1, C2, ..., Cm. Given an unknown sample X.

- Subclassing Naive Bayes will determine that X belongs to class Ci if and only if:

$$P(C_i|X) > P(C_j|X), \text{ với mọi } 1 \le i, j \le m, j \neq i$$

$$P(C_i|X) = \frac{P(X|Ci) \times P(Ci)}{P(X)}$$

❖ **According to Bayes' theorem**

Since P(X) is constant for all classes, **only the maximum P(X|Ci) x P(Ci) is needed. If P(Ci) is not known, we need to assume P(C1)=P(C2)=...= P(Cm)** and **we will maximize P(X|Ci). Otherwise, we maximize P(X|Ci) x P(Ci)**

However, the problem of calculating P(X|Ci) is impossible!

Admit Naive: assume attribute independence

$$P(X|C_i) = \prod_{k=1}^{n} P(X_k|C_i)$$

It is possible to approximate P(x1|Ci),…,P(xn|Ci) from the training samples.

If Ak is a qualitative attribute, then P(xk|Ci) = sik/si where sik is the number of training samples of Ci with the value xk for Ak and si is the number of samples belonging to class Ci

If Ak is continuous, then it is assumed to have a Gaussian distribution:

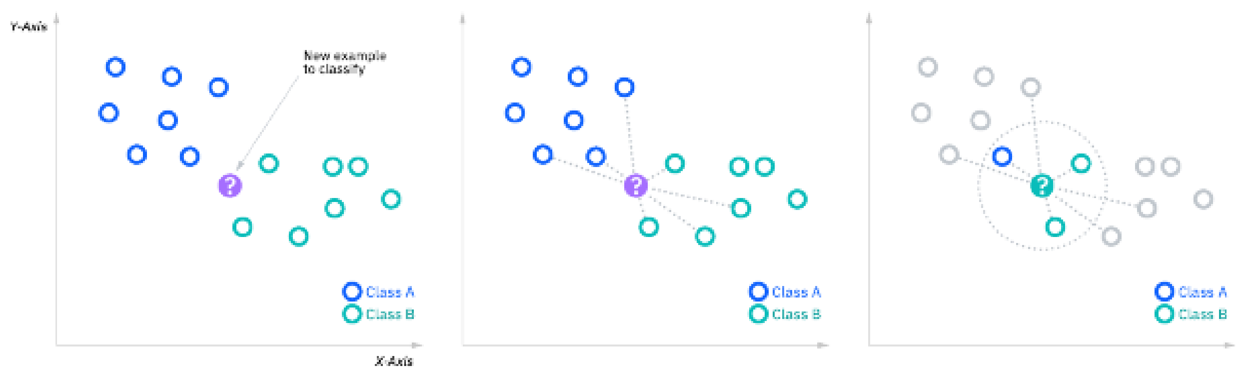
$$P(X_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}) = \frac{1}{\sqrt{2\pi\sigma_{C_i}}} e^{-\frac{(x_k - \mu_{C_i})^2}{2\sigma^2 C_i}}$$

3.1.4 K-Nearest Neighbors

3.1.4.1 K-Nearest Neighbors explain

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.

For classification problems, a class label is assigned on the basis of a majority vote—i.e. the label that is most frequently represented around a given data point is used. While this is technically considered "plurality voting", the term, "majority vote" is more commonly used in literature. The distinction between these terminologies is that "majority voting" technically requires a majority of greater than 50%, which primarily works when there are only two categories. When you have multiple classes—e.g. four categories, you don't necessarily need 50% of the vote to make a conclusion about a class; you could assign a class label with a vote of greater than 25%.



3.1.4.2 K-nearest neighbors Algorithm

Compute KNN: distance metrics

To recap, the goal of the k-nearest neighbor algorithm is to identify the nearest neighbors of a given query point, so that we can assign a class label to that point. In order to do this, KNN has a few requirements.

- Determine your distance metrics

In order to determine which data points are closest to a given query point, the distance between the query point and the other data points will need to be calculated. These distance metrics help to form decision boundaries, which partitions query points into different regions. You commonly will see decision boundaries visualized with Voronoi diagrams.

While there are several distance measures that you can choose from, this article will only cover the following:

- Euclidean distance (p=2): This is the most commonly used distance measure, and it is limited to real-valued vectors. Using the below formula, it measures a straight line between the query point and the other point being measured.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (y_i - x_i)^2}$$

- Manhattan distance (p=1): This is also another popular distance metric, which measures the absolute value between two points. It is also referred to as taxicab distance or city block distance as it is commonly visualized with a grid, illustrating how one might navigate from one address to another via city streets.

-

$$\text{Manhattan Distance} = d(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i|\right)$$

- Minkowski distance: This distance measure is the generalized form of Euclidean and Manhattan distance metrics. The parameter, p, in the formula below, allows for the creation of other distance metrics. Euclidean distance is represented by this formula when p is equal to two, and Manhattan distance is denoted with p equal to one.

$$\text{Minkowski Distance} = \left(\sum_{i=1}^{n} |x_i - y_i|\right)^{1/p}$$

- **Hamming distance:** This technique is used typically used with Boolean or string vectors, identifying the points where the vectors do not match. As a result, it has also been referred to as the overlap metric. This can be represented with the following formula:

$$\text{Hamming Distance} = D_H = \left(\sum_{i=1}^{k} |x_i - y_i| \right)$$

$$x=y \qquad D=0$$
$$x \neq y \qquad D \neq 1$$

3.1.5 Support Vector Machine

3.1.5.1 Support Vector Machine explain

Support Vector Machine (SVM) is a robust classification and regression technique that maximizes the predictive accuracy of a model without overfitting the training data. **SVM is particularly suited to analyzing data with very large numbers (for example, thousands) of predictor fields.**

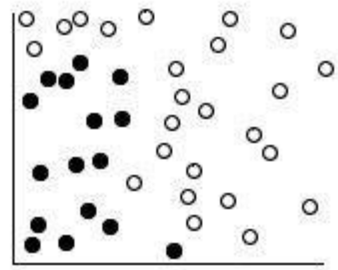
SVM has applications in many disciplines, including customer relationship management (CRM), facial and other image recognition, bioinformatics, text mining concept extraction, intrusion detection, protein structure prediction, and voice and speech recognition.

❖ **How Support Vector Machine Models works?**

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

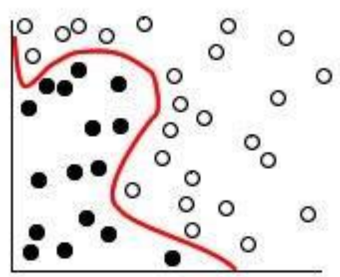
For example, consider the following figure, in which the data points fall into two different categories.

Figure 1. Data with a preliminary model



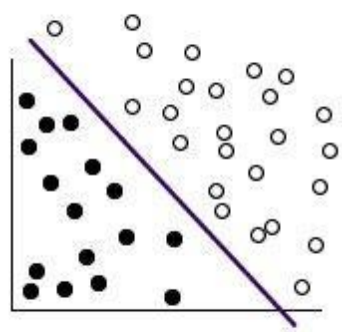
The two categories can be separated with a curve, as shown in the following figure.

Figure 2. Data with separator added



After the transformation, the boundary between the two categories can be defined by a hyperplane, as shown in the following figure.

Figure 3. Transformed data



The mathematical function used for the transformation is known as the **kernel** function.

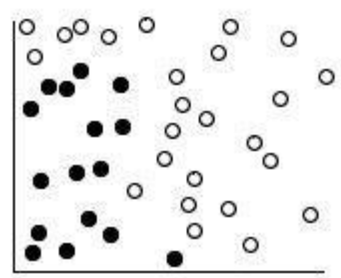
A linear kernel function is recommended when linear separation of the data is straightforward. In other cases, one of the other functions should be used. You will need to experiment with the different functions to obtain the best model in each case, as they each use different algorithms and parameters.

❖ How Support Vector Machine Models works?

SVM works by mapping data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. A separator between the categories is found, then the data are transformed in such a way that the separator could be drawn as a hyperplane. Following this, characteristics of new data can be used to predict the group to which a new record should belong.

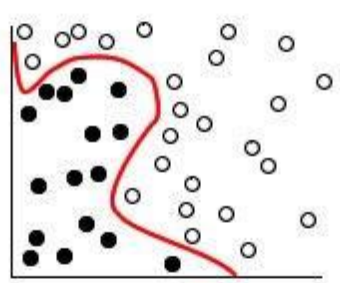
For example, consider the following figure, in which the data points fall into two different categories.

Figure 1. Data with a preliminary model



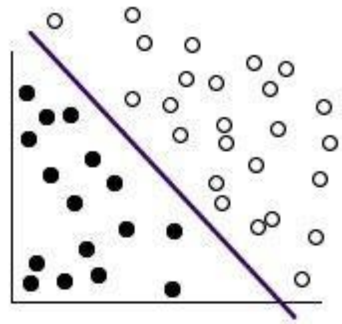
The two categories can be separated with a curve, as shown in the following figure.

Figure 2. Data with separator added



After the transformation, the boundary between the two categories can be defined by a hyperplane, as shown in the following figure.

Figure 3. Transformed data



The mathematical function used for the transformation is known as the **kernel** function.

A linear kernel function is recommended when linear separation of the data is straightforward. In other cases, one of the other functions should be used. You will need to experiment with the different functions to obtain the best model in each case, as they each use different algorithms and parameters.

3.2 Experiments on Jupyter Notebook

- Graph, count, and view label ratios to get an overview of a song's musical genre.
- Build decision properties, with the decision property as music_genre.
- Pre-processing before putting in train.

3.2.1 Replace categorical attribute values with numerical

```
type_music = {'Jazz, Blues or Electronic': 0, 'Anime':1,
     'Rock, Alternative or Country':2 , 'Rap or Hip-Hop':3, 'Classical':4}
df.music_genre = df.music_genre.map(type_music)
```

✓ 0.8s Python

3.2.2 Split the decision property column to a separate column

```
# # Tách cột thuộc tính quyết định ra 1 cột riêng
label = df.music_genre
df = df.drop('music_genre', axis=1)
```

3.2.3 Separating train and test data

Train data accounts for 70%, test accounts for 30%

```
# Tách dữ liệu train và test
X_train, X_test, y_train, y_test = train_test_split(df, label, test_size=0.3,random_state=10)
```

3.2.4 Decision Trees Algorithm

```
from sklearn import tree
clf1 = tree.DecisionTreeClassifier(criterion="gini", random_state=0)

start_cart = time.time()
cart_pred = clf1.fit(X_train, y_train).predict(X_test)
end_cart = time.time()
times_tree_cart = timedelta(seconds=round(end_cart - start_cart,4)).total_seconds()
print("Time decision tree (CART)",times_tree_cart)
cart_score = round(metrics.accuracy_score(y_test, cart_pred)*100,2)
accuracy_tree_cart = cart_score
print("Accuracy",accuracy_tree_cart)
print("Report",metrics.classification_report(y_test,cart_pred))
```

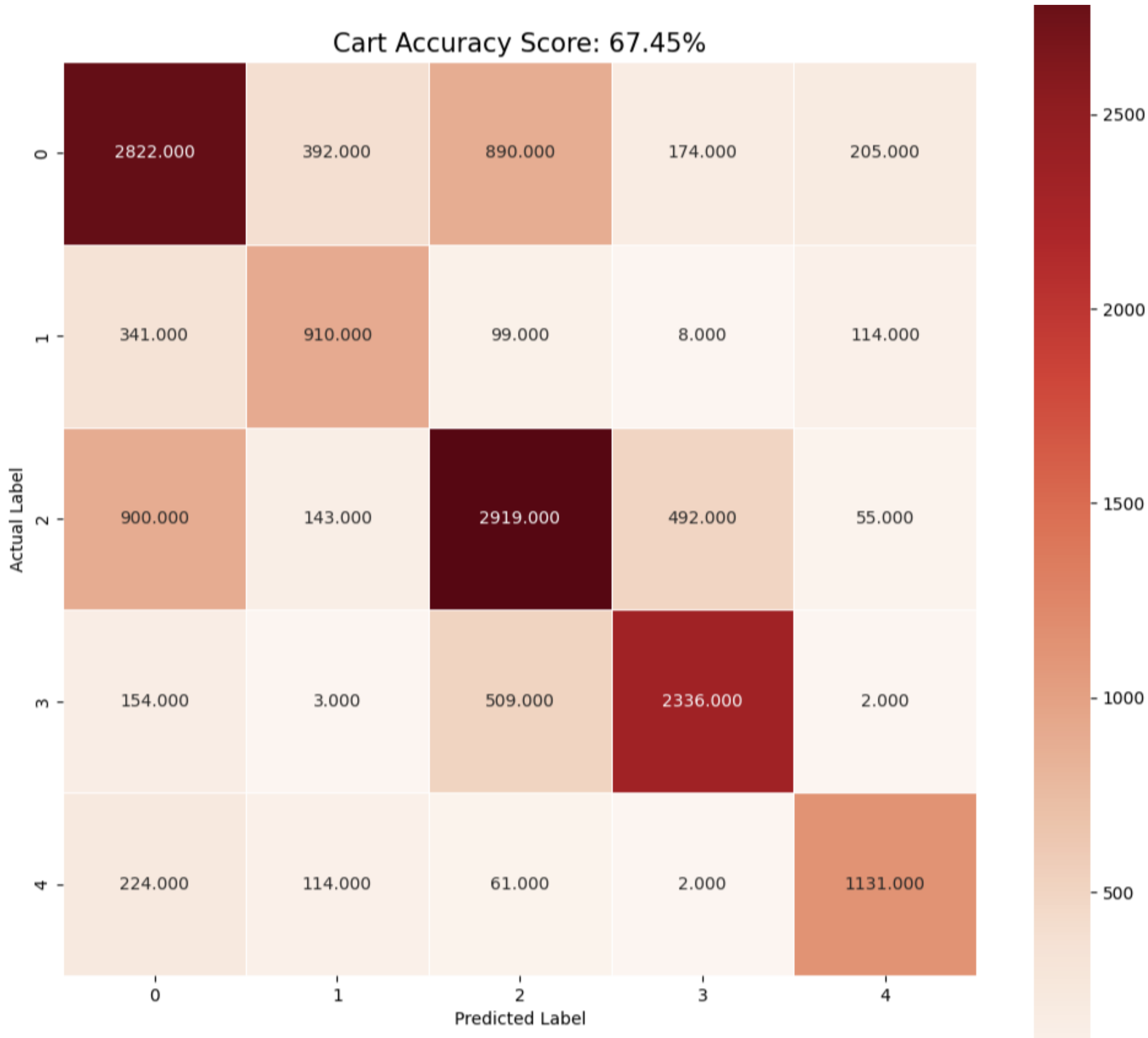
```
Time decision tree (CART) 0.378
Accuracy 67.45
Report               precision    recall  f1-score   support

           0       0.64      0.63      0.63      4483
           1       0.58      0.62      0.60      1472
           2       0.65      0.65      0.65      4509
           3       0.78      0.78      0.78      3004
           4       0.75      0.74      0.74      1532

    accuracy                           0.67     15000
   macro avg       0.68      0.68      0.68     15000
weighted avg       0.68      0.67      0.67     15000
```

- Algorithm accuracy: 67.45%
- Algorithm runtime: 0.378s

```
# Vẽ ma trận nhầm lẫn cho thuật toán Decision Tree (CART)
cart_cm = metrics.confusion_matrix(y_test, cart_pred)
plt.figure(figsize=(12,12))
ax=sns.heatmap(cart_cm, annot=True, fmt=".3f", linewidth=.5, square=True, cmap='Reds')
ax.set_ylabel('Actual Label')
ax.set_xlabel('Predicted Label')
title = 'Cart Accuracy Score: {0}%'.format(cart_score)
plt.title(title,size=15)
plt.show()
```



Through the confused matrix of the Decision Tree algorithm picture tissue (CART), we know:

➢ Precision of the algorithmic Picture tissue: 67.45%

3.2.5 Random Forest algorithm

```
# Thực hiện thuật toán Ramdom Forest
rfc = RandomForestClassifier(criterion="gini", random_state=0)
start_rf =time.time()
rf_pred = rfc.fit(X_train, y_train).predict(X_test)
end_rf= time.time()
times_rf = timedelta(seconds=round(end_rf-start_rf,4)).total_seconds()
print ("time", times_rf)

rf_score = round(metrics.accuracy_score(y_test, rf_pred)*100,2)
accuracy_rf = rf_score
print("Accuracy", accuracy_rf,"%")
print("Report", metrics.classification_report(y_test, rf_pred))
```

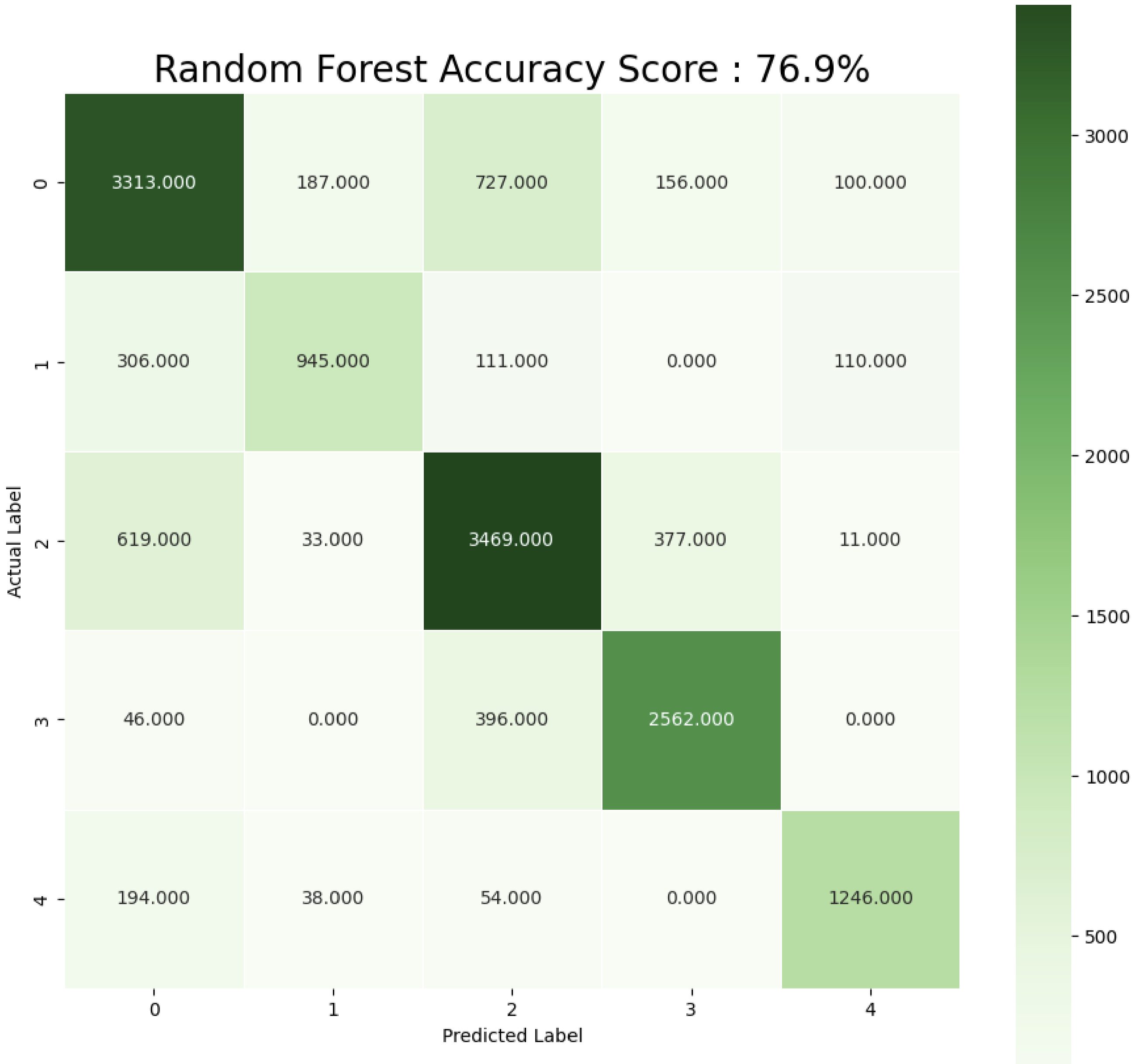
```
time 12.3143
Accuracy 76.9 %
Report               precision    recall  f1-score   support

           0       0.74      0.74      0.74      4483
           1       0.79      0.64      0.71      1472
           2       0.73      0.77      0.75      4509
           3       0.83      0.85      0.84      3004
           4       0.85      0.81      0.83      1532

    accuracy                           0.77     15000
   macro avg       0.79      0.76      0.77     15000
weighted avg       0.77      0.77      0.77     15000
```

- Algorithm accuracy: 76.9%
- Algorithm runtime: 12.3143s

```
# Vẽ ma trận nhầm lẫn cho mô hình thuật toán Random Forest
rf_cm = metrics.confusion_matrix(y_test, rf_pred)
plt.figure(figsize=(11,11))
ax =sns.heatmap(rf_cm, annot =True, fmt =".3f",linewidths = .5, square =True, cmap= 'Greens')
ax.set_ylabel('Actual Label')
ax.set_xlabel('Predicted Label')
title = 'Random Forest Accuracy Score : {0}%'.format(rf_score)
plt.title(title, size =20)
```



Through the confused matrix of random forest algorithm picture tissue, we know:

- Precision of the algorithmic Picture tissue: 67.45%

3.2.6 Naive Bayes Algorithm

```
# Thực hiện thuật toán Naive Bayes
from sklearn.naive_bayes import GaussianNB
nv = GaussianNB()
start_nv = time.time()
nv_pred = nv.fit(X_train, y_train).predict(X_test)
end_nv = time.time()
times_nv = timedelta(seconds=round(end_nv - start_nv,4)).total_seconds()
print("Time Naive Bayes",times_nv)
nv_score = round(metrics.accuracy_score(y_test, nv_pred)*100,2)
accuracy_nv = nv_score
print("Accuracy",accuracy_nv)
print("Report",metrics.classification_report(y_test,nv_pred))
```

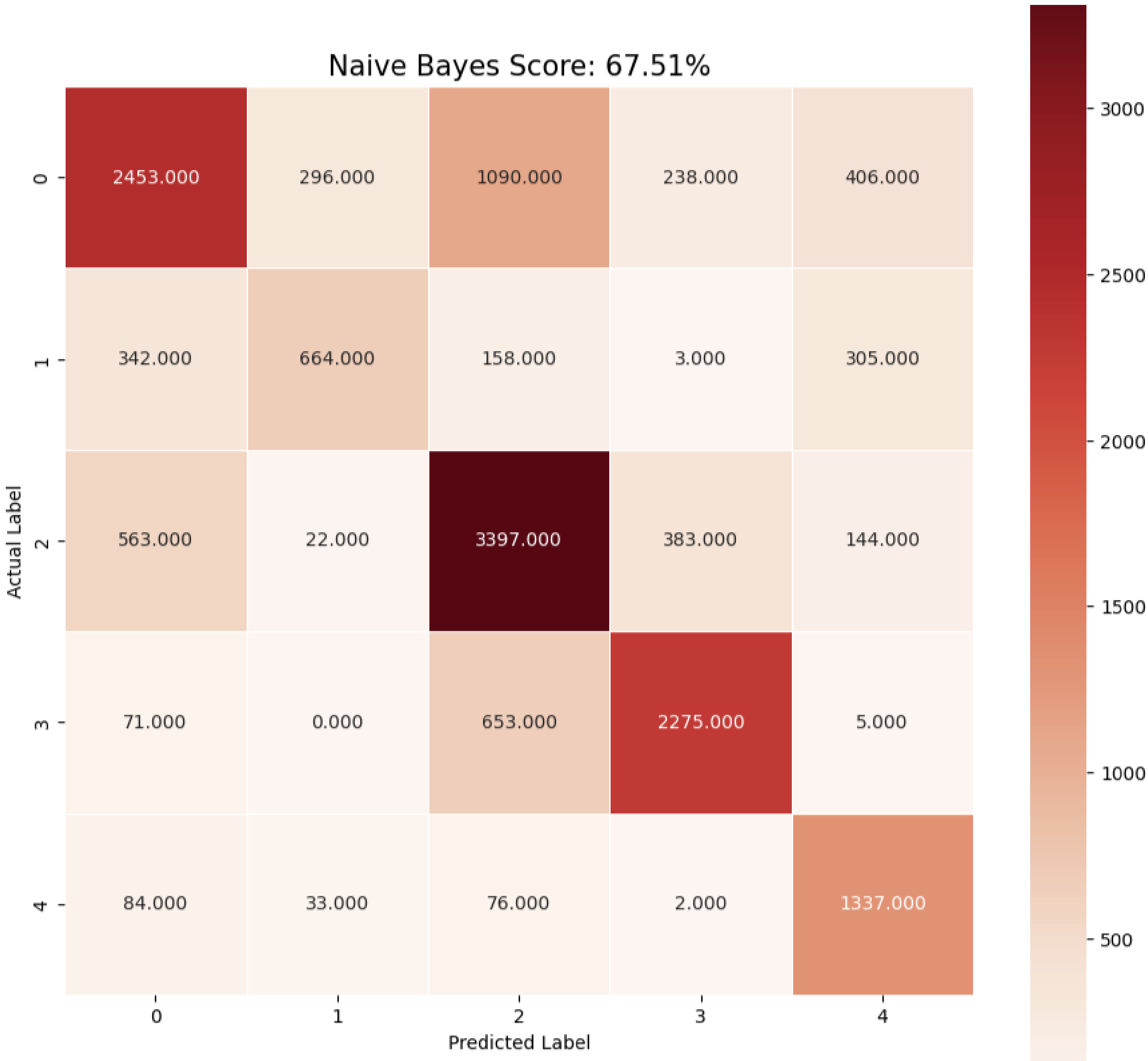
```
Time Naive Bayes 0.039
Accuracy 67.51
Report               precision    recall  f1-score   support

           0       0.70      0.55      0.61      4483
           1       0.65      0.45      0.53      1472
           2       0.63      0.75      0.69      4509
           3       0.78      0.76      0.77      3004
           4       0.61      0.87      0.72      1532

    accuracy                           0.68     15000
   macro avg       0.68      0.68      0.66     15000
weighted avg       0.68      0.68      0.67     15000
```

- Algorithm accuracy: 67.51%
- Algorithm runtime: 0.039s

```
# Vẽ ma trận nhầm lẫn cho thuật toán Naive Bayes
nv_cm = metrics.confusion_matrix(y_test, nv_pred)
plt.figure(figsize=(11,11))
ax=sns.heatmap(nv_cm, annot=True, fmt=".3f", linewidth=.5, square=True, cmap='Reds')
ax.set_ylabel('Actual Label')
ax.set_xlabel('Predicted Label')
title = 'Naive Bayes Score: {0}%'.format(nv_score)
plt.title(title,size=15)
plt.show()
```



Through the confused matrix of the Naive Bayes algorithmic Picture tissue, we learn :

- Precision of the algorithmic Picture tissue: 67.51%

3.2.7 K-Nearest Neighbors Algorithm (KNN)

```
# Thực hiện thuật toán KNN
from sklearn.neighbors import KNeighborsClassifier
import time
from sklearn.metrics import classification_report
from datetime import timedelta
import matplotlib.pyplot as plt
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
start_knn = time.time()
knn_scores =  []

for i in range (1,12):
    knc = KNeighborsClassifier(i)
    knn_pred = knc.fit(X_train, y_train).predict(X_test)
    knn_scores.append(metrics.accuracy_score(y_test, knn_pred))
    max_knn_score = max (knn_scores)
knn_score_ind = [i for i, v in enumerate(knn_scores) if v == max_knn_score]
end_knn =time.time()
times_knn = timedelta(seconds=round(end_knn - start_knn,4)).total_seconds()
print('Highest Accuracy Score : {}% with k = {}'.format(max_knn_score*100, list(map(lambda x: x + 1, knn_score_ind))))
print ('Time', times_knn)
knn_score = round(max_knn_score*100,2)
accuracies_max_knn = knn_score
print("Accuracy", accuracies_max_knn,"%")
print("Report", metrics.classification_report(y_test, knn_pred))
```

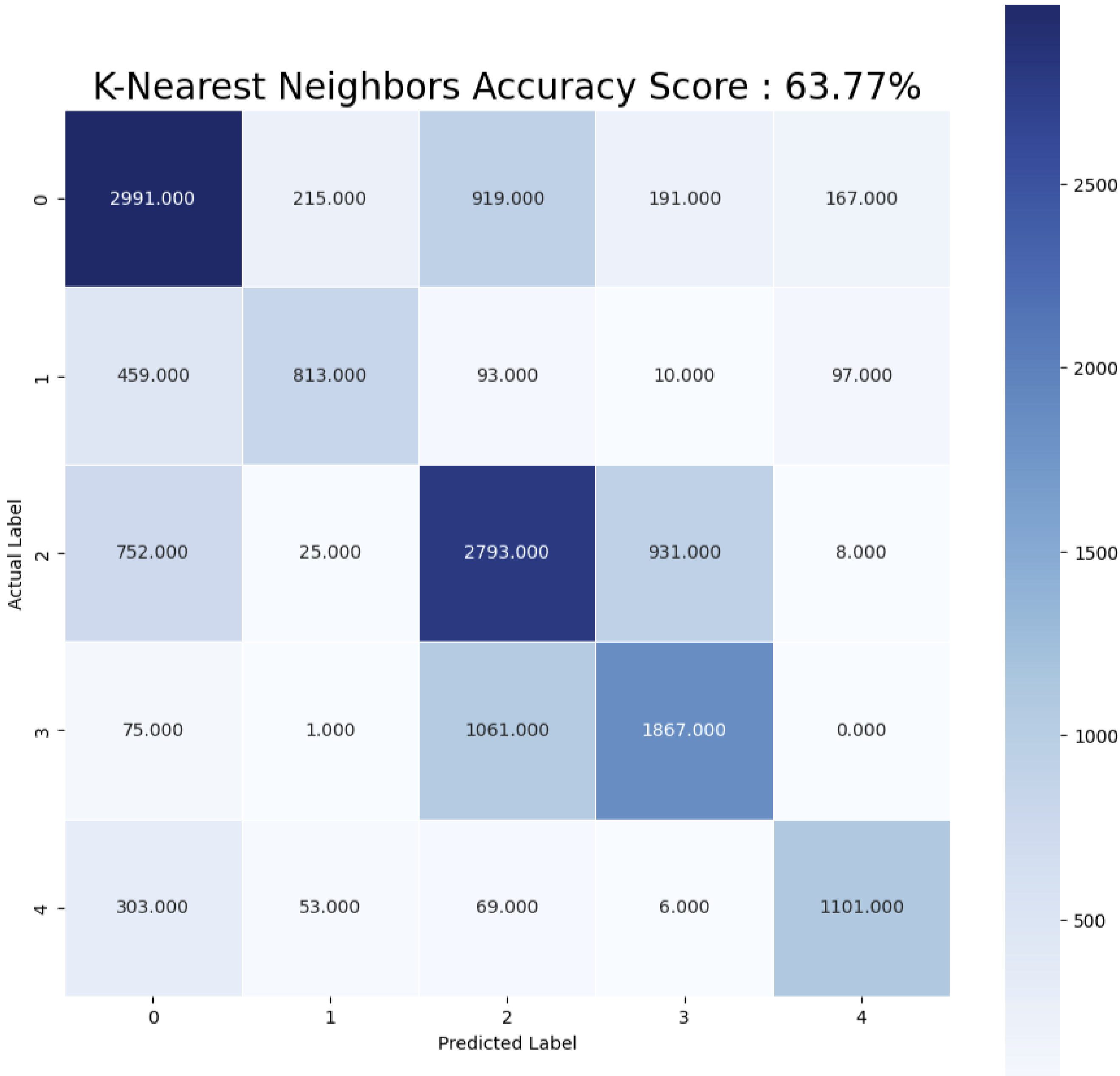
```
Highest Accuracy Score : 63.766666666666667% with k = [11]
Time 6.5999
Accuracy 63.77 %
Report               precision    recall  f1-score   support

           0       0.65      0.67      0.66      4483
           1       0.73      0.55      0.63      1472
           2       0.57      0.62      0.59      4509
           3       0.62      0.62      0.62      3004
           4       0.80      0.72      0.76      1532

    accuracy                           0.64     15000
   macro avg       0.68      0.64      0.65     15000
weighted avg       0.64      0.64      0.64     15000
```

- Algorithm accuracy: 63.77%
- Algorithm runtime: 6.5999s

```
# Tiếp tục thực hiện thuật toán KNN
# Vẽ ma trận nhầm lẫn
knn_cm = metrics.confusion_matrix(y_test, knn_pred)
plt.figure(figsize=(11,11))
ax = sns.heatmap(knn_cm, annot =True, fmt =".3f",linewidths = .5, square =True, cmap= 'Blues')
ax.set_ylabel('Actual Label')
ax.set_xlabel('Predicted Label')
title = 'K-Nearest Neighbors Accuracy Score : {0}%'.format(knn_score)
plt.title(title, size =20)
```



Through the confusion matrix of the KNN algorithm Picture model, we know:

- Precision of the algorithmic Picture tissue: 63.77%

3.2.8 Support Vector Machine

```
from sklearn import svm
# Thực hiện thuật toán Support Vector Machine
SVM_model = svm.SVC(kernel='linear')
start_svm =time.time()
svm_pred = SVM_model.fit(X_train, y_train).predict(X_test)
end_svm= time.time()
times_svm = timedelta(seconds=round(end_svm-start_svm,4)).total_seconds()
print ("time", times_svm)

svm_score = round(metrics.accuracy_score(y_test, svm_pred)*100,2)
accuracy_svm = svm_score
print("Accuracy", accuracy_svm,"%")
print("Report", metrics.classification_report(y_test, svm_pred))
```

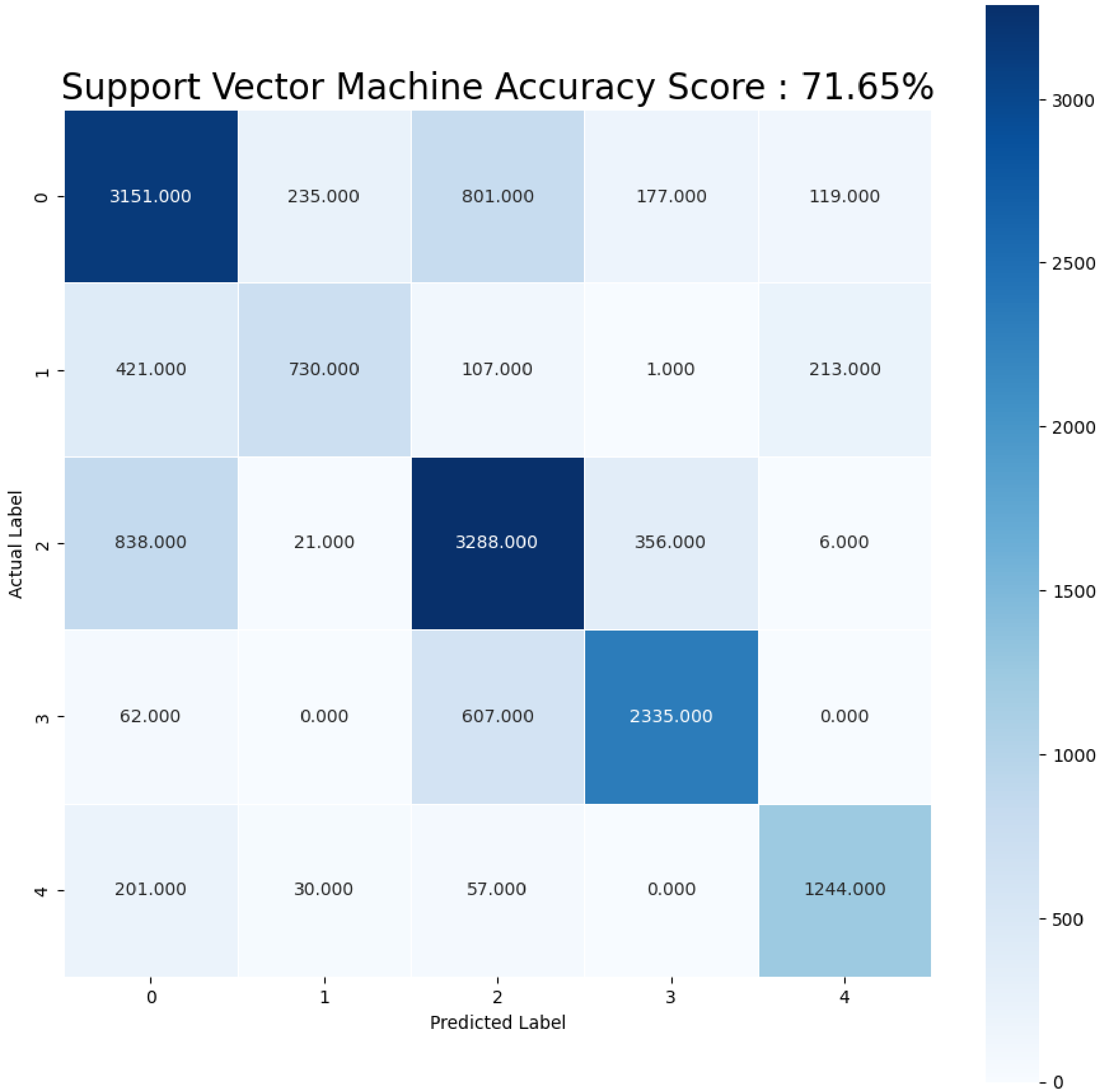
```
time 153.1277
Accuracy 71.65 %
Report               precision    recall  f1-score   support

           0       0.67      0.70      0.69      4483
           1       0.72      0.50      0.59      1472
           2       0.68      0.73      0.70      4509
           3       0.81      0.78      0.80      3004
           4       0.79      0.81      0.80      1532

    accuracy                           0.72     15000
   macro avg       0.73      0.70      0.71     15000
weighted avg       0.72      0.72      0.72     15000
```

- Algorithm accuracy: 71.65%
- Algorithm runtime: 153.1277s

```
# Tiếp tục thực hiện thuật toán SVM
# Vẽ ma trận nhầm lẫn
svm_cm = metrics.confusion_matrix(y_test, svm_pred)
plt.figure(figsize=(11,11))
ax = sns.heatmap(svm_cm, annot =True, fmt =".3f",linewidths = .5, square =True, cmap= 'Blues')
ax.set_ylabel('Actual Label')
ax.set_xlabel('Predicted Label')
title = 'K-Nearest Neighbors Accuracy Score : {0}%'.format(svm_score)
plt.title(title, size =20)
```



Through the confused matrix of Support Vector Machine algorithmic Picture tissue, we learn .

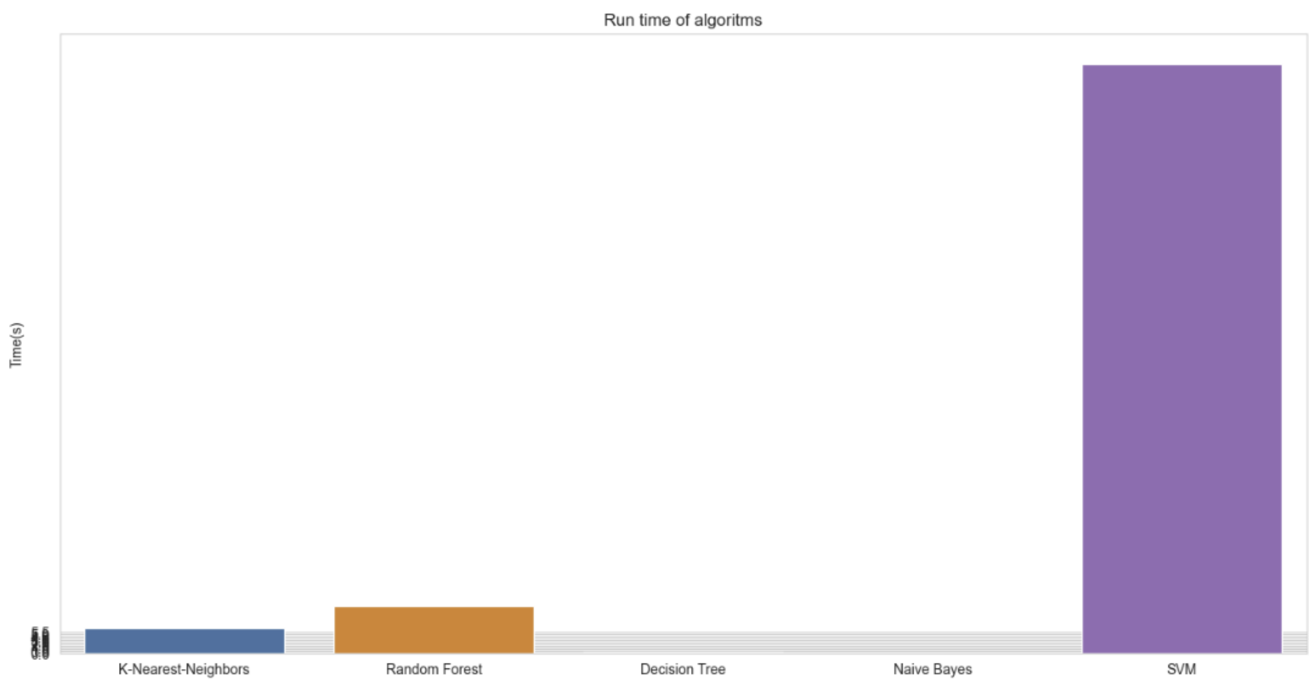
- Precision of the algorithmic Picture tissue: 71.65%

3.2.9 Comparison, Evaluation

Use the BarPlot graph to get an overview of runtime and accuracy between algorithms.

Draw a chart comparing the running time of algorithms

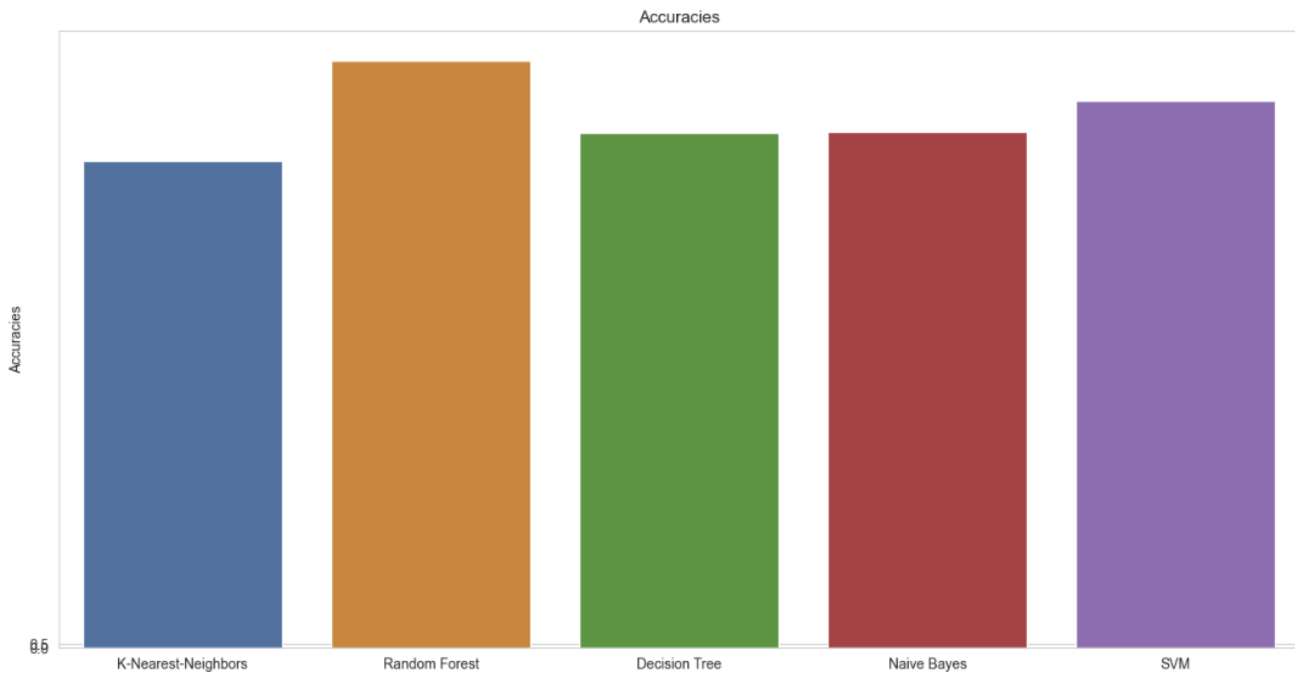
```
colors = ['blue', 'green', 'red', 'bwr', 'purple']
sns.set_style('whitegrid')
plt.figure(figsize=(16,8))
plt.yticks(np.arange(0,6,0.5))
plt.ylabel('Time(s)')
plt.title('Run time of algoritms')
sns.barplot(x=list(['K-Nearest-Neighbors','Random Forest','Decision Tree','Naive Bayes','SVM']),
            y= list([times_knn, times_rf, times_tree_cart, times_nv, times_svm]))
plt.show()
```



Conclusion on the runtime chart:

- ❖ The Naïve Bayes algorithm is the algorithm with the fastest running time for datasets. With only 0.039s.
- ❖ SVM algorithm is slowest with 153.1277s.
- ❖ Draw a chart comparing the accuracy of algorithms:

```
# colors = ['blue', 'green', 'red', 'bwr', 'purple']
sns.set_style('whitegrid')
plt.figure(figsize=(16,8))
plt.yticks(np.arange(0,1,0.5))
plt.ylabel('Accuracies')
plt.title('Accuracies')
sns.barplot(x=list(['K-Nearest-Neighbors','Random Forest','Decision Tree','Naive Bayes', 'SVM']),
            y= list([accuracies_max_knn, accuracy_rf, accuracy_tree_cart, accuracy_nv, accuracy_svm]))
plt.show()
```



Conclusion on the accuracy chart:

- ❖ The algorithms all give very high accuracy results, balanced with each other, most of which is 67~72% accuracy.
- ❖ The K-Nearest Neighbors algorithm has the lowest accuracy of the five algorithms, with an accuracy 63.77%.

```
results = pd.DataFrame({
    'Model': ['K-Nearest-Neighbors', 'Random Forest', 'Decision Tree (CART)','Naive Bayes',
              'Support Vector Machine'],
    'Score': [ accuracies_max_knn, accuracy_rf,accuracy_tree_cart, accuracy_nv, accuracy_svm]})
result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Model')
result_df
```

Model	Score
Random Forest	76.90
Support Vector Machine	71.65
Naive Bayes	67.51
Decision Tree (CART)	67.45
K-Nearest-Neighbors	63.77

CHAPTER IV: PREDICTIVE SOFTWARE

4.1 Software overview

4.1.1 Algorithms used

Based on the results obtained in the previous section, the team decided to use random forest algorithm for this software. According to the comparison results, this algorithm, although it has a bad speed, but it gives the highest accuracy.

4.1.2 Properties used to make predictions

```
#tim thuoc tính có độ tin cậy cao
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
clf.fit(X_train,y_train)
feature_imp = pd.Series(clf.feature_importances_,index=df.columns).sort_values(ascending=False)
feature_imp
```

```
popularity      0.277134
speechiness     0.147151
loudness        0.133212
danceability    0.126343
acousticness    0.098869
energy          0.090419
valence         0.077356
key             0.035506
mode            0.014010
dtype: float64
```

```
from sklearn.feature_selection import SelectFromModel
sel = SelectFromModel(RandomForestClassifier(n_estimators=100))
sel.fit(X_train,y_train)
```

```
SelectFromModel(estimator=RandomForestClassifier())
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook. On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
selected_feat = X_train.columns[(sel.get_support())]
len(selected_feat)
```

```
4
```

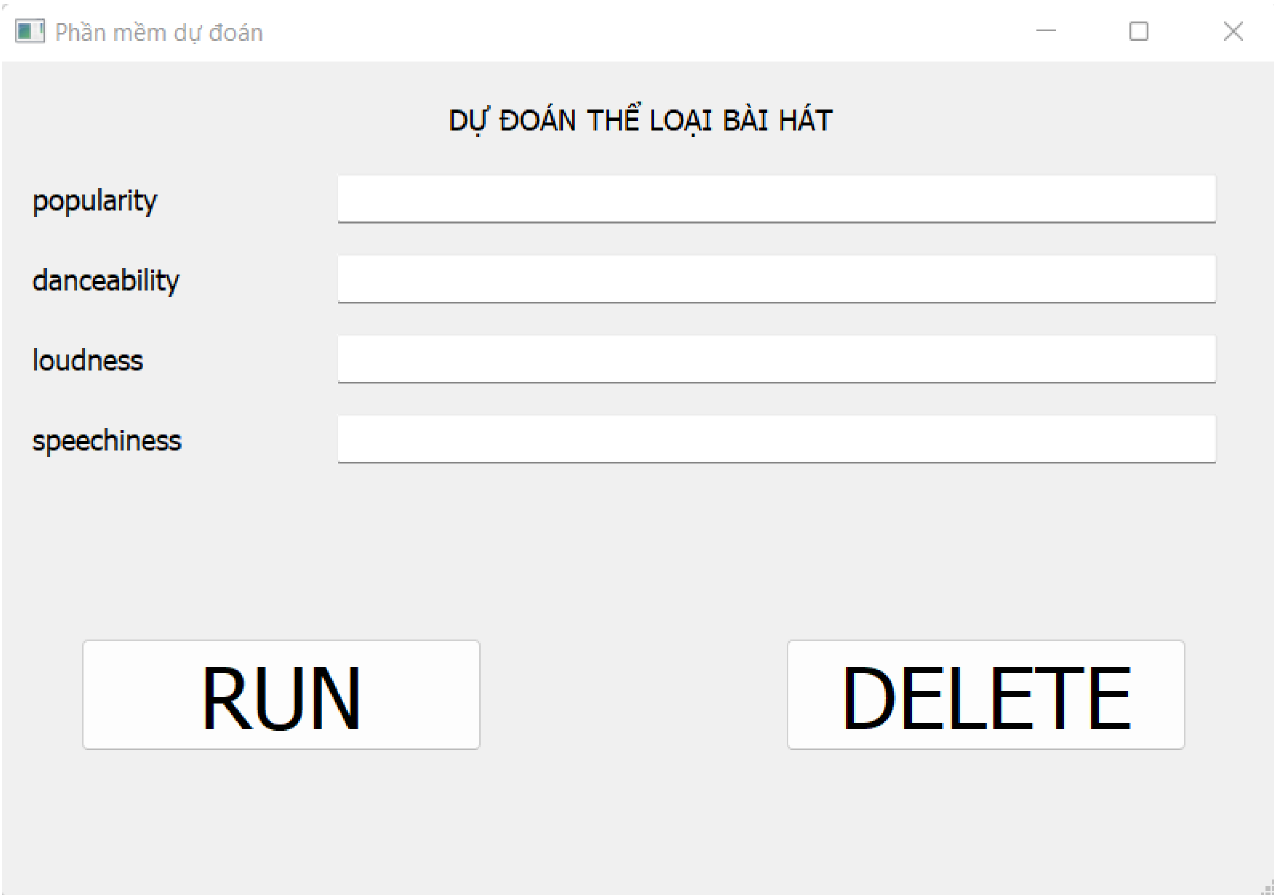
```
print(selected_feat)
```

```
Index(['popularity', 'danceability', 'loudness', 'speechiness'], dtype='object')
```

According to the important attribute classification results, I will take 4 attributes **'popularity', 'danceability', 'loudness', 'speechiness'** to make predictions.

4.1.3 Interface and Testing

4.1.3.1 Interface



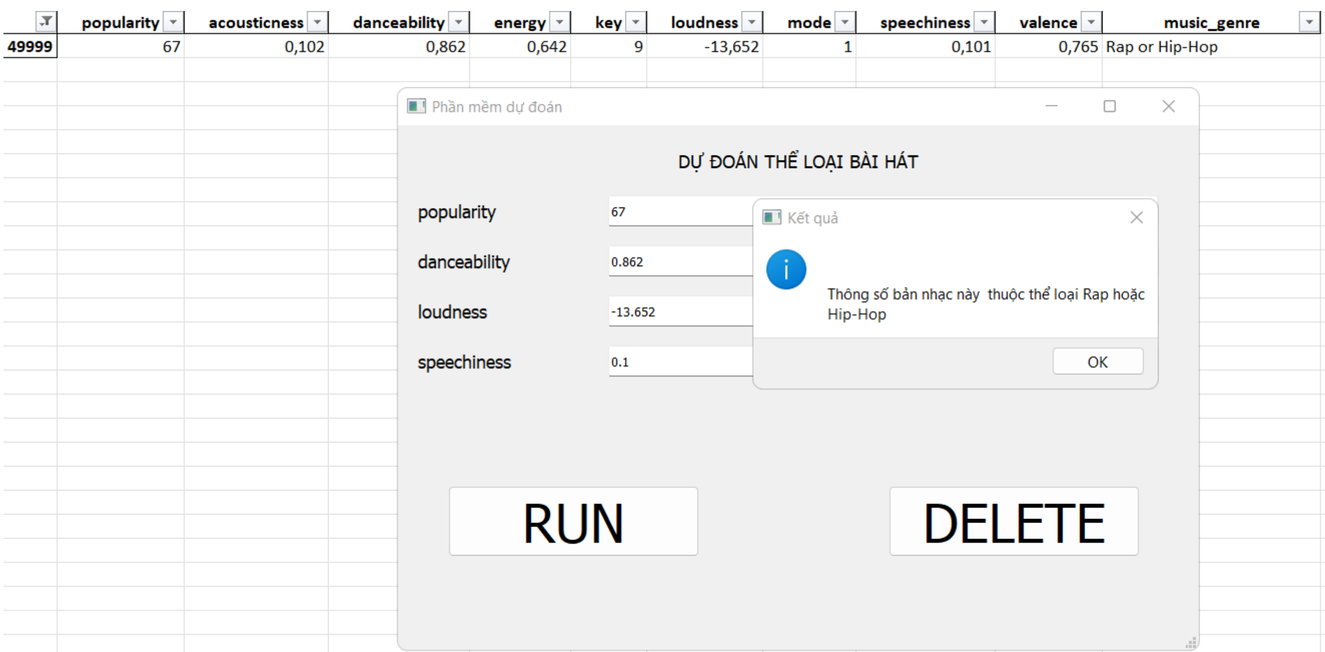
4.1.3.2 Testing

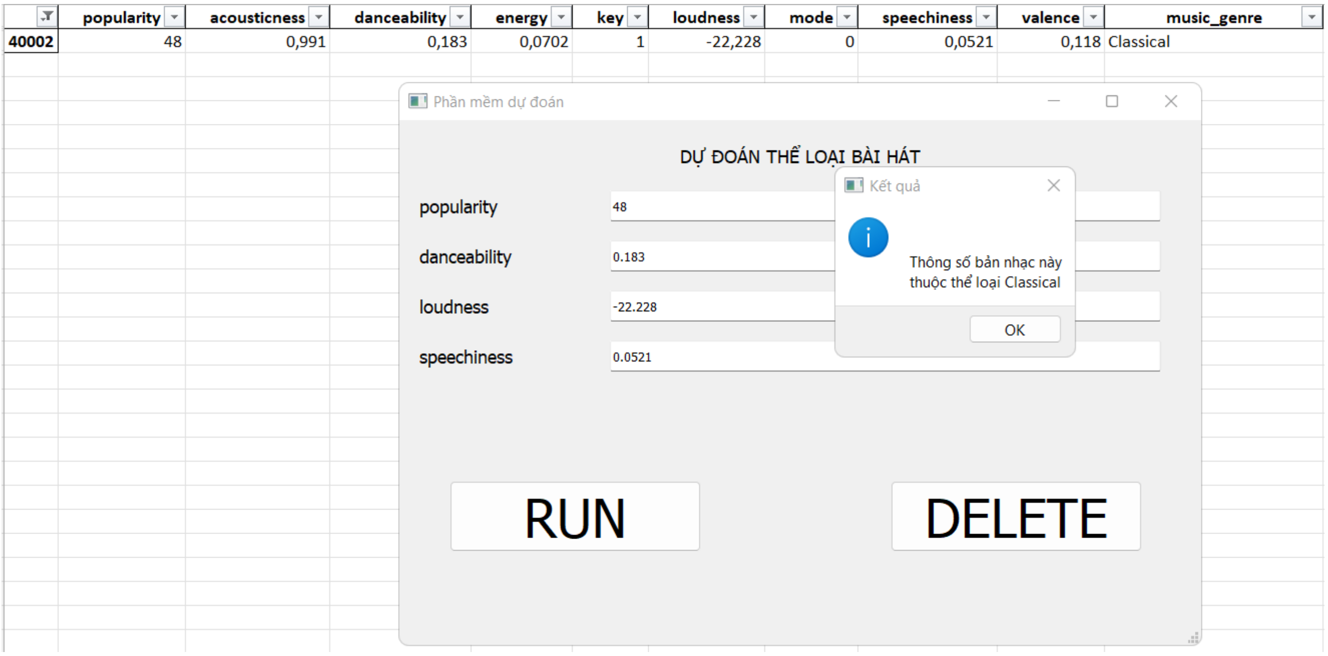
Test Dataset

```
df2 = pd.read_csv('test_v4.csv')
df2
```

	Unnamed: 0	popularity	acousticness	danceability	energy	key	loudness	mode	speechiness	valence	music_genre
0	0	27.0	0.00468	0.652	0.941	1	-5.201	1	0.0748	0.759	Jazz, Blues and Electronic
1	1	31.0	0.01270	0.622	0.890	5	-7.043	1	0.0300	0.531	Jazz, Blues and Electronic
2	2	28.0	0.00306	0.620	0.755	11	-4.617	0	0.0345	0.333	Jazz, Blues and Electronic
3	3	34.0	0.02540	0.774	0.700	4	-4.498	0	0.2390	0.270	Jazz, Blues and Electronic
4	4	32.0	0.00465	0.638	0.587	9	-6.266	0	0.0413	0.323	Jazz, Blues and Electronic
...	...	...	...	...	...	...	...	...	...	...	...
49995	49995	59.0	0.03340	0.913	0.574	4	-7.022	0	0.2980	0.330	Rap and Hip-Hop
49996	49996	72.0	0.15700	0.709	0.362	2	-9.814	0	0.0550	0.113	Rap and Hip-Hop
49997	49997	51.0	0.00597	0.693	0.763	5	-5.443	0	0.1460	0.395	Rap and Hip-Hop
49998	49998	65.0	0.08310	0.782	0.472	10	-5.016	1	0.0441	0.354	Rap and Hip-Hop
49999	49999	67.0	0.10200	0.862	0.642	9	-13.652	1	0.1010	0.765	Rap and Hip-Hop

The software results come out in line with the original data.





4.2 Software code

4.2.1 Interface section code

```
from PyQt5 import QtCore, QtGui, QtWidgets
class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 524)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(30, 10, 741, 51))
        font = QtGui.QFont()
        font.setPointSize(32)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
```

```python
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(20, 120, 170, 31))
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setKerning(False)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")

        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(20, 170, 170, 31))
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setKerning(False)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
```

```python
        self.label_4 = QtWidgets.QLabel(self.centralwidget)
        self.label_4.setGeometry(QtCore.QRect(20, 220, 170, 31))
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setKerning(False)
        self.label_4.setFont(font)
        self.label_4.setObjectName("label_4")

```

```python
        self.label_1 = QtWidgets.QLabel(self.centralwidget)
        self.label_1.setGeometry(QtCore.QRect(20, 70, 170, 31))
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setKerning(False)
        self.label.setFont(font)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        font = QtGui.QFont()
        font.setPointSize(11)
        font.setKerning(False)
        self.label_1.setFont(font)
        self.label_1.setObjectName("label_1")
        self.lineEdit_2 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_2.setGeometry(QtCore.QRect(210, 120, 550, 31))
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.lineEdit_3 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_3.setGeometry(QtCore.QRect(210, 170, 550, 31))
        self.lineEdit_3.setObjectName("lineEdit_3")
        self.lineEdit_4 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_4.setGeometry(QtCore.QRect(210, 220, 550, 31))
        self.lineEdit_4.setObjectName("lineEdit_4")
        self.lineEdit_1 = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_1.setGeometry(QtCore.QRect(210, 70, 550, 31))
        self.lineEdit_1.setObjectName("lineEdit_1")
        self.pushButton = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(50, 360, 251, 71))
```

```python

        font = QtGui.QFont()
        font.setPointSize(32)
        self.pushButton.setFont(font)
        self.pushButton.setObjectName("pushButton")
        self.pushButton_2 = QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(490,360, 251, 71))
        font = QtGui.QFont()
        font.setPointSize(32)
        self.pushButton_2.setFont(font)
        self.pushButton_2.setObjectName("pushButton_2")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)
```

```
        self.pushButton.clicked.connect(self.Crun)
        self.pushButton_2.clicked.connect(self.Clr)
        # train()
    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "Phần mềm dự đoán"))
        self.label.setText(_translate("MainWindow", "DỰ ĐOÁN THỂ LOẠI BÀI HÁT"))
        self.label_1.setText(_translate("MainWindow", "popularity"))
        self.label_2.setText(_translate("MainWindow", "danceability"))
        self.label_3.setText(_translate("MainWindow", "loudness"))
        self.label_4.setText(_translate("MainWindow", "speechiness"))
        self.pushButton.setText(_translate("MainWindow", "RUN"))
        self.pushButton_2.setText(_translate("MainWindow", "DELETE"))
        'popularity', 'danceability', 'loudness', 'speechiness'
    def Clr(self) -> None:
        self.lineEdit_1.clear()
        self.lineEdit_2.clear()
        self.lineEdit_3.clear()
        self.lineEdit_4.clear()
    def Crun(self) -> None:
        my_dict =    {"popularity":float(self.lineEdit_1.text()),
                      "danceability":float(self.lineEdit_2.text()),
                      "loudness":float(self.lineEdit_3.text())
        , "speechiness":float(self.lineEdit_4.text())}
        t=str('Thông số bản nhạc này')
        print(my_dict)
```

```
        # Xác định đường dẫn tuyệt đối của tệp tin "R.pkl"
        file_path = os.path.abspath("R.pkl")

        # Sử dụng đường dẫn tệp tin trong hàm find_data_file()
        output = check_input(my_dict, file_path)
        print(output)

        msg = QtWidgets.QMessageBox()
        msg.setIcon(QtWidgets.QMessageBox.Information)
        a = ""
```

```
        if output == 0:
            a=" thuộc thể loại"
            msg.setInformativeText("{} {} Jazz, Blues hoặc Electronic".format(t,str(a)))
        elif output == 1:
            a=" thuộc thể loại"
            msg.setInformativeText("{} {} Anime".format(t,str(a)))
        elif output == 2:
            a=" thuộc thể loại"
            msg.setInformativeText("{} {} Rock, Alternative hoặc Country".format(t,str(a)))
        elif output == 3:
            a=" thuộc thể loại"
            msg.setInformativeText("{} {} Rap hoặc Hip-Hop".format(t,str(a)))
        elif output == 4:
            a=" thuộc thể loại"
            msg.setInformativeText("{} {} Classical".format(t,str(a)))
        msg.setWindowTitle("Kết quả")
        msg.exec_()
```

```
    # from sklearn.metrics import accuracy_score

if __name__ == '__main__':
        train()
        app = QtWidgets.QApplication(sys.argv)
        MainWindow = QtWidgets.QMainWindow()
        ui = Ui_MainWindow()
        ui.setupUi(MainWindow)
        MainWindow.show()
        sys.exit(app.exec_())

```

4.2.2 Processing section code

```
from cgi import test
from PyQt5 import QtCore, QtGui, QtWidgets

#  phần xử lí
import pandas as pd
from sklearn.metrics import accuracy_score
import os
import sys
import pickle
import numpy as np
```

```
#For training
def train() -> None:
    with open('test_v4_software.csv') as f:
    # with open('test_v4.csv') as f:
        df = pd.read_csv(f)
    df_filtered = df.replace('Unnamed: 0',np.nan)
    df_filtered.dropna(inplace=True)
    df_filtered.reset_index(drop=True, inplace=True)
    dataset = df_filtered.copy()
    from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    for col in dataset.columns[ [i == object for i in dataset.dtypes] ]:
        dataset.loc[:,col] = le.fit_transform(dataset[col])
    dataset = dataset[['popularity', 'danceability', 'loudness', 'speechiness', 'music_genre']]

    x = dataset.iloc[:, :-1].values
    y = dataset.iloc[:, -1].values

    from sklearn.compose import ColumnTransformer
    ct = ColumnTransformer(transformers=[], remainder='passthrough' )
    x = np.array(ct.fit_transform(x))

    from sklearn.preprocessing import LabelEncoder
    le = LabelEncoder()
    y = le.fit_transform(y)
```

```
#train test split
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42)
    from sklearn.ensemble import RandomForestClassifier
    classifier = RandomForestClassifier(n_estimators =10, criterion='gini', random_state=0)
    classifier.fit(x_train, y_train)
    R= classifier.fit(x_train,y_train)

#Save Model As Pickle File
    with open('R.pkl','wb') as m:
        pickle.dump(R,m)
    test(x_test,y_test)
```

```
#Test accuracy of the model
def test(X_test,Y_test):
    with open('R.pkl','rb') as mod:
        p=pickle.load(mod)
    pre=p.predict(X_test)
    print (accuracy_score(Y_test,pre)) #Prints the accuracy of the model

def find_data_file(filename):
    if getattr(sys, "frozen", False): # The application is frozen.
        datadir = os.path.dirname(sys.executable)
    else:
# The application is not frozen.
        datadir = os.path.dirname( __file__)
    return os.path.join(datadir, filename)
```

```
def check_input(data, file_path):
    df = pd.DataFrame(data=data, index=[0])
    with open(file_path, 'rb') as model:
        p = pickle.load(model)
    op = p.predict(df)
    return op
```

CHAPTER V: CONCLUSION

5.1 Advantages and limitations of each algorithm

5.1.1 Decision Tree

❖ Advantages

- The algorithm is simple, intuitive, not too complicated to understand the first time.

- The training dataset doesn't have to be too large to build an analytical model.

- Some decision tree algorithms are capable of processing missing data and faulty data without applying methods such as "imputing missing values" or removing. Less affected by the exception data.

- There is no need to make initial assumptions about the laws of distribution as in statistics, and as a result the results of the analysis obtained are the most objective, "natural".

- It can help us classify data objects according to multi-layered, multi-class classifications, especially if the target variable is a complex quantitative distortion.

- Can be applied flexibly to target variables, target variables.

- Delivers highly accurate forecast results, easy to implement, fast in training, no need to switch variables.

- Easy to interpret or explain to listeners, viewers who want to understand the results of analysis but have no knowledge of data science.

- Articulate the connection between variables and data attributes in the most intuitive way.

- In addition to economics, finance, decision tree algorithms can be applied in the fields of health, agriculture, biology.

❖ Limitations

- The decision tree algorithm works effectively on a simple dataset that has few data variables that relate to each other, and vice versa if applied to complex datasets.

- When applied with complex datasets, many different variables and attributes can lead to overfitting patterns, which are too consistent with training data leading to the problem of not giving accurate classification results when applied to test data, and new data.

- The variance value is high, when there is a small change in the dataset can affect the structure of the model.

- The tree algorithm decides to apply only to classification trees if misclassification can lead to serious mistakes.

- The tree algorithm decides whether it is likely to be "biased" or biased if the dataset is not balanced.

- Training and testing datasets must be perfectly prepared, good quality must be balanced in layers, groups in target variables.

- There is no technical "support" or "reverse query" capability.

5.1.2 Random Forest

❖ **Advantages**

Random Forest has several advantages that contribute to its popularity and effectiveness in machine learning tasks.

- **Robustness:** Random Forest is robust against overfitting, which occurs when a model performs well on training data but fails to generalize to new, unseen data. The algorithm's ensemble approach, combined with bootstrapping and feature randomness, helps to reduce overfitting and improve generalization performance.

- **Versatility**: Random Forest can handle a wide range of data types, including numerical and categorical features. It can also handle missing values and outliers without requiring extensive data preprocessing.

- **Feature Importance**: Random Forest provides a measure of feature importance, indicating the relative contribution of each feature to the model's predictions. This information can be used for feature selection, dimensionality reduction, and gaining insights into the underlying data.

- **Non-linearity and Interactions**: Random Forest can capture non-linear relationships between features and the target variable. It can also handle interactions between features, allowing for more complex and expressive modeling capabilities.

- Scalability: Random Forest can efficiently handle large datasets with a high number of features. The algorithm's parallelizability makes it suitable for parallel and distributed computing environments, enabling faster training and prediction times.

- Resistance to Noise: Random Forest is less affected by noisy data compared to individual decision trees. By aggregating predictions from multiple trees, the impact of noise is mitigated, leading to more reliable results.

❖ Limitations:

- Model Interpretability: Random Forest models can be less interpretable compared to individual decision trees. As the algorithm combines multiple trees, understanding the exact decision-making process can be more challenging.

- Computationally Intensive: Training a Random Forest model can be computationally expensive, especially for large datasets with numerous features. The algorithm constructs multiple decision trees, which increases the overall training time and memory requirements.

- Memory Usage: Random Forest models consume more memory compared to simpler algorithms. As each decision tree is stored separately, the memory footprint of the model can become substantial, particularly when dealing with a large number of trees or complex datasets.

- Overfitting in Noisy Data: While Random Forest is generally robust to overfitting, it can still be affected by noise or outliers in the data. Noisy features can lead to overfitting, where individual trees capture spurious patterns in the data**.**

- Biased Class Distribution: Random Forest may struggle with imbalanced class distributions, where one class is significantly more prevalent than others. The algorithm can have a bias towards the majority class, resulting in poorer performance for minority classes.

- Training Time Sensitivity: Random Forest training can be sensitive to the choice of hyperparameters, such as the number of trees or the depth of each tree. Finding the optimal set of hyperparameters can require some experimentation and tuning.

5.1.3 Naïve Bayes

❖ Advantages

- Independent assumptions: works well for multiple problems/data domains and applications.

- Simple but good enough to solve many problems such as text layering, spam filtering

- Easy to use and fast when it comes to guessing the label of test data. It's pretty good in multi-class prediction (test later).

- When assuming that the features of the data are independent of each other, Naive Bayes runs better than other algorithms such as logistic regression and also needs less data.

- Allows the succession of prior knowledge and obeserved data.

- It is good that there is a numerical difference between the classification classes.

- Model training (parameter estimation) is easy and fast.

❖ **Limitations:**

- The accuracy of Naive Bayes compared to other algorithms is not high.

- In the real world, it is almost impossible when the features of test data are independent of each other.

- Problem zero (stated how to solve it above).

- The model is not trained by a strong and rigorous optimization method.

- The parameters of the model are estimates of the probability of single conditions. Do not take into account the interaction between these estimates.

5.1.4 K–Nearest Neighbors

❖ **Advantages**

- **Easy to implement:** Given the algorithm’s simplicity and accuracy, it is one of the first classifiers that a new data scientist will learn.

- **Adapts easily**: As new training samples are added, the algorithm adjusts to account for any new data since all training data is stored into memory.

- **Few hyperparameters**: KNN only requires a k value and a distance metric, which is low when compared to other machine learning algorithms.

❖ **Limitations**

- Does not scale well: Since KNN is a lazy algorithm, it takes up more memory and data storage compared to other classifiers. This can be costly from both a time and money perspective. More memory and storage will drive up business expenses and more data can take longer to compute. While different data structures, such as Ball-Tree, have been created to address the computational inefficiencies, a different classifier may be ideal depending on the business problem.

- Curse of dimensionality: The KNN algorithm tends to fall victim to the curse of dimensionality, which means that it doesn't perform well with high-dimensional data inputs. This is sometimes also referred to as the peaking phenomenon (PDF, 340 MB), where after the algorithm attains the optimal number of features, additional features increases the amount of classification errors, especially when the sample size is smaller.

- Prone to overfitting: Due to the "curse of dimensionality", KNN is also more prone to overfitting. While feature selection and dimensionality reduction techniques are leveraged to prevent this from occurring, the value of k can also impact the model's behavior. Lower values of k can overfit the data, whereas higher values of k tend to "smooth out" the prediction values since it is averaging the values over a greater area, or neighborhood. However, if the value of k is too high, then it can underfit the data.

5.1.5 Support Vector Machine

❖ Advantages

- Effective in high dimensional spaces.

- Still effective in cases where number of dimensions is greater than the number of samples.

- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

❖ Limitations

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.

- SVM do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation.

REFERENCES

1. https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

2. https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

3. http://myweb.sabanciuniv.edu/rdehkharghani/files/2016/02/The-Morgan-Kaufmann-Series-in-Data-Management-Systems-Jiawei-Han-Micheline-Kamber-Jian-Pei-Data-Mining.-Concepts-and-Techniques-3rd-Edition-Morgan-Kaufmann-2011.pdf

4. https://www.ibm.com/docs/en/spss-modeler/saas?topic=nodes-support-vector-machine-models

5. https://scikit-learn.org/stable/modules/svm.html