IT2362-02 Predictive Modelling Assignment Report

BY:
TEY JIA YING

Table of Contents

1. Introduction

For this project, the aim is to classify a piece of music based on the characteristic or features of the music. I will be using a pre-selected dataset for this Data2021.csv .The file contains information about music and its classification. Each row in the file contains characteristics of a piece of music like the loudness, tempo etc. The music pieces are classified into 11 different classes (0 – 10). Each of these classes correspond to a type of music like Folk, Blue, HipHop, Metal Pop.

2. Data Understanding

The first stage of any data analytics project is to have a good understanding of the data. Hence, I decided to understand the data quantity, data types and characteristics of each column to have a good gauge on what I could do with the data in terms of analysis, preparation, and modelling.

2.1 Data Quantity

```
print("Rows, columns: " + str(df.shape))
```

```
Rows, columns: (17996, 17)
```

I used them. shape and found out that the dataset has 17996 rows and 17 columns which would be considered a good amount of data for analysis.

2.2 Data Types

```
df=pd.read_csv('Data2021.csv')
df.dtypes
```

```
Artist Name            object
Track Name             object
Popularity            float64
danceability          float64
energy                float64
key                   float64
loudness              float64
mode                    int64
speechiness           float64
acousticness          float64
instrumentalness      float64
liveness              float64
valence               float64
tempo                 float64
duration_in min/ms    float64
time_signature          int64
Class                   int64
dtype: object
```

Using the .dtypes function, we can see that most of the rows are of float or int type while only the columns containing Name are of object type.

2.3 Explanation of Dataset and Data Characteristics

This dataset contains information about songs and its music characteristics.
After some research on each of the characteristics, I managed to get the definition of each characteristic.

Explanation of Dataset:

This dataset consists of **13 music charcteristics** that is used to predict the target variable:

Music Characteristics	Description
Danceability	Describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity.
Valence	Describes the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).
Energy	Represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale.
Tempo	The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece, and derives directly from the average beat duration.
Loudness	The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks.
Speechiness	This detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value.
Instrumentalness	Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal".
Liveness	Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live.
Acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic.
Key	The estimated overall key of the track. Integers map to pitches using standard Pitch Class notation . E.g. 0 = C, 1 = C♯/D♭, 2 = D, and so on.
Mode	Indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0.
Duration	The duration of the track in milliseconds.
Time Signature	An estimated overall time signature of a track. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure).

The other non-music characteristic columns are popularity, Class, Track Name, Artist Name.

- Popularity represents the amount of fame/support for each song.
- The target variable would be Class where it has a total 11 unique classes that represents different type of music such as Folk, Blue, HipHop, Metal Pop.
- The **2 other rows - Artist Name and Track Name** that is a unique identifier of each row.

3. Data Preparation

After we have gotten a good understanding of our data, let us prepare the data for analysis and modelling. My methodology for this stage will be first, to replace missing values, then remove duplicates, followed by converting duration to minutes and lastly after all invalid/missing data is handled. I will be removing outliers

3.1 Replacing Missing Values

```
# Import the INNImputer
from sklearn.impute import KNNImputer

# The dfKNN data frame will only have the 3 columns with missing values
dfKNN = df[['Popularity',"instrumentalness","key"]]

# Create a kNNImputer object and set k=1
imputer = KNNImputer(n_neighbors=5)

# We use the fit_transform() method to perform the imputation
# We also create another DataFrame from the results returned
# by the fit_transform function
dfKNN = pd.DataFrame(imputer.fit_transform(dfKNN))

# We merge back with the rest of the columns (Year and Method)
df = pd.concat([df[["Artist Name","Track Name","danceability","energy","loudness","mode",
                    "speechiness","acousticness","liveness","valence","tempo","duration_in min/ms",
                    "time_signature","Class"]], dfKNN], axis=1, join="inner")
df
```

I chose not to drop the null values and instead replace it with KNN imputation as total number of missing values for popularity, instrumentalness and key were ranging from 400 to 4300. If I were to drop all it will affect our dataset as we do not have much information on these 3 music characteristics to predict the class.

3.2 Removing Duplicates

Let's explore the number of duplicates in the whole dataset.

```
df.shape
```

```
(17996, 17)
```

```
print("Number of duplicates in whole dataset:",df.duplicated(keep='first').sum())
```

```
Number of duplicates in whole dataset: 0
```

Looks like there are no duplicates. However, as I scanned through the whole dataset, I realised that there are rows that have duplicated values across all columns except for Class. This would mean that there are 2 exactly songs having the same music characteristic but they of 2 different classes. Hence, let us drop data with similar values across all columns except for class

```
print("Number of duplicates across all columns(except for class):",df[['Artist Name', 'Track Name', 'danceability', 'energy', 'lo
       'mode', 'speechiness', 'acousticness', 'liveness', 'valence', 'tempo',
       'duration_in min/ms', 'time_signature', 'Popularity',
       'instrumentalness', 'key']].duplicated(keep='first').sum())
```

```
Number of duplicates across all columns(except for class): 1673
```

```
df_no_class=df[['Artist Name', 'Track Name', 'danceability', 'energy', 'loudness',
       'mode', 'speechiness', 'acousticness', 'liveness', 'valence', 'tempo',
       'duration_in min/ms', 'time_signature', 'Popularity',
       'instrumentalness', 'key']].drop_duplicates(keep=False)
```

```
df=df[df.index.isin(df_no_class.index)]
```

```
df.shape
```

```
(14838, 17)
```

Now, that we have dropped duplicates, we are left with 14838 rows.

3.3 Converting Duration from Milliseconds to Minutes

After scanning through the dataset, I realised that the longest duration is minutes is 30. Hence, I will be converting the values above 30 from milliseconds to minutes.

```
def convert_to_min(time):
    if time>30:
        return time/(1000*60)
    else:
        return time

df['duration_in_min']=df['duration_in min/ms'].apply(convert_to_min)
```

```
<ipython-input-231-09a8aa8de0ae>:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df['duration_in_min']=df['duration_in min/ms'].apply(convert_to_min)
```

```
df.drop(['duration_in min/ms'],axis=1,inplace=True)
```

3.4 Removing Outliers

```
###Use a new dataframe for numerical columns
df1=df[["danceability","energy","loudness","mode","speechiness","acousticness","liveness","valence","tempo","duration_in_min","ti
z_scores = stats.zscore(df1)
abs_z_scores = np.abs(z_scores)
filtered_entries = (abs_z_scores < 3).all(axis=1)
print("Number of Outliers:",df.shape[0]-np.count_nonzero(filtered_entries))
```

```
Number of Outliers: 1354
```

In this stage, I first checked for the number of outliers using z score and realised that there were 1354 outliers which stands around 8% of the whole dataset. Hence, I decided to remove the outliers as there is enough data to spare.

```
df= df[filtered_entries]
df
```

	Artist Name	Track Name	danceability	energy	loudness	mode	speechiness	acousticness	liveness	valence	tempo	duration_in min/ms	time_signature	Clas
0	Bruno Mars	That's What I Like (feat. Gucci Mane)	0.854	0.564	-4.964	1	0.0485	0.017100	0.0849	0.8990	134.071	234596.0	4	
1	Boston	Hitch a Ride	0.382	0.814	-7.230	1	0.0406	0.001100	0.1010	0.5690	116.454	251733.0	4	1
2	The Raincoats	No Side to Fall In	0.434	0.614	-8.334	1	0.0525	0.486000	0.3940	0.7870	147.681	109667.0	4	
3	Deno	Lingo (feat. J.I & Chunkz)	0.853	0.597	-6.528	0	0.0555	0.021200	0.1220	0.5690	107.033	173968.0	4	
4	Red Hot Chili Peppers	Nobody Weird Like Me - Remastered	0.167	0.975	-4.279	1	0.2160	0.000169	0.1720	0.0918	199.060	229960.0	4	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
17991	Green-House	Find Home	0.166	0.109	-17.100	0	0.0413	0.993000	0.0984	0.1770	171.587	193450.0	3	
17992	Micatone	All Gone	0.638	0.223	-10.174	0	0.0329	0.858000	0.0705	0.3350	73.016	257067.0	4	
17993	Smash Hit Combo	Peine perdue	0.558	0.981	-4.683	0	0.0712	0.000030	0.6660	0.2620	105.000	216222.0	4	
17994	Beherit	Salomon's Gate	0.215	0.805	-12.757	0	0.1340	0.001290	0.2560	0.3550	131.363	219693.0	4	
17995	The Raconteurs	Broken Boy Soldier	0.400	0.853	-5.320	0	0.0591	0.006040	0.3340	0.3770	138.102	182227.0	4	1

16481 rows × 17 columns

Now that I have removed outliers, it will allow for a better model performance in the later stage.

4. Exploratory Data Analysis

After we have cleaned the data, we will now move on to Exploratory Data Analysis (EDA) to explore and visualize some data to get some meaningful insights. For this stage, I will be exploring these few areas for analysis:

- Descriptive Statistical Analysis
- Correlation Analysis
- Song Class Analysis
- Further Analysis of Music Characteristic and Class

4.1 Descriptive Statistical Analysis

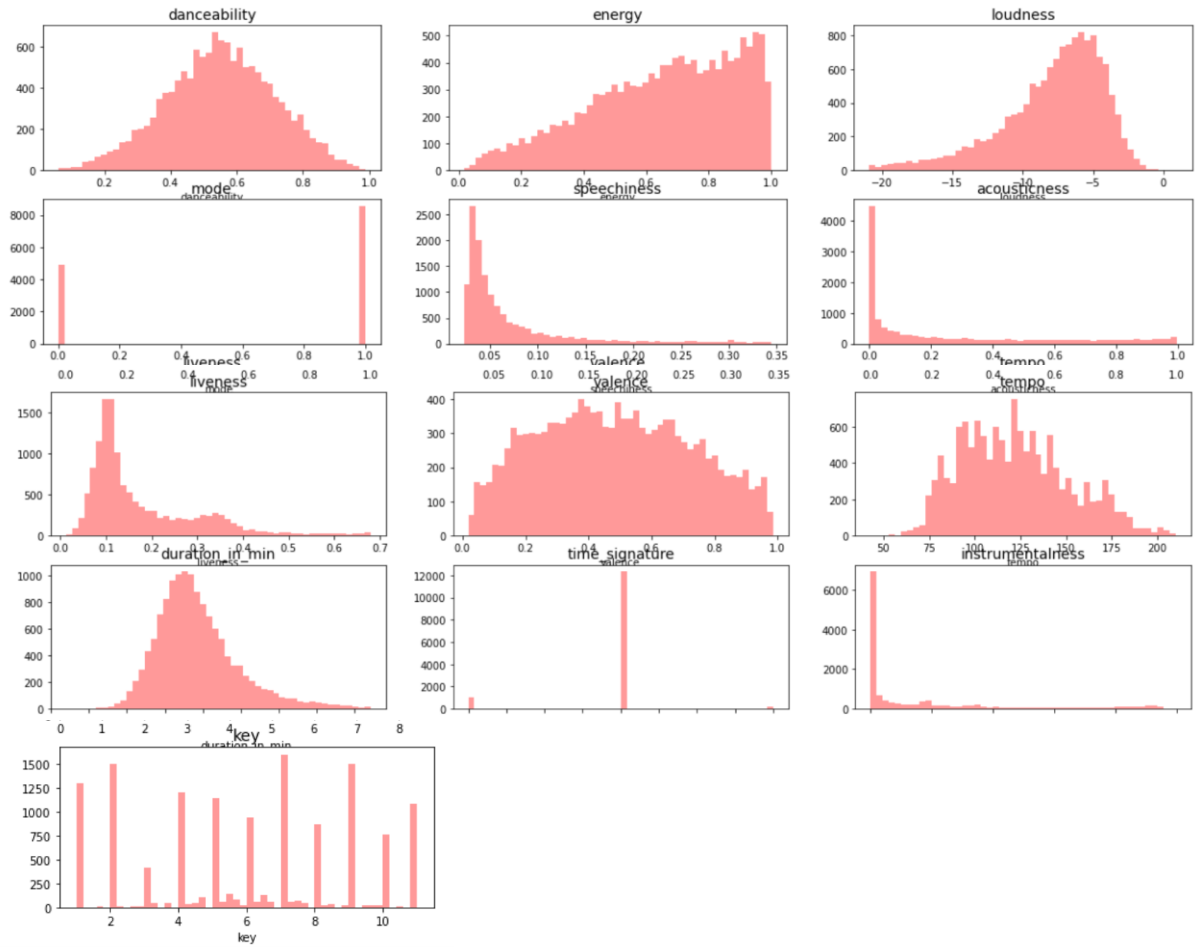
4.1.1 Statistical Summary of the Numerical Variables

I used the describe () function once again to look at the statistical summary of the numerical variables.

	danceability	energy	loudness	mode	speechiness	acousticness	liveness	valence	tempo	time_signature	Class	Popularity	instrumentalness	key	duration_in_min
count	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000	13484.000000
mean	0.551126	0.652734	-7.830060	0.635494	0.071960	0.265530	0.178511	0.484164	122.106857	3.938891	6.857980	43.952996	0.167114	5.934295	3.855722
std	0.163310	0.233155	3.594211	0.481309	0.063338	0.314791	0.122955	0.240459	29.584804	0.283498	3.155246	17.341515	0.266135	3.074491	1.080495
min	0.059900	0.016300	-20.892000	0.000000	0.022500	0.000000	0.011900	0.018300	42.956000	3.000000	0.000000	1.000000	0.000001	1.000000	0.388667
25%	0.441000	0.488000	-9.677500	0.000000	0.034300	0.005940	0.096900	0.292000	98.812750	4.000000	5.000000	32.000000	0.000227	4.000000	3.136667
50%	0.552000	0.684500	-7.084000	1.000000	0.045900	0.105000	0.125000	0.475000	119.990500	4.000000	8.000000	43.000000	0.015972	6.000000	3.673175
75%	0.666000	0.851000	-5.205000	1.000000	0.079800	0.480000	0.235000	0.670000	141.014000	4.000000	10.000000	56.000000	0.215106	9.000000	4.364504
max	0.989000	1.000000	0.943000	1.000000	0.344000	0.996000	0.680000	0.986000	209.905000	5.000000	10.000000	95.000000	0.996000	11.000000	8.429550

From this data frame, we may not be able to visualize at one glance the distribution of all characteristics. Hence, I have decided to plot the distribution of each characteristic below

4.1.2 Distribution of each Music Characteristics



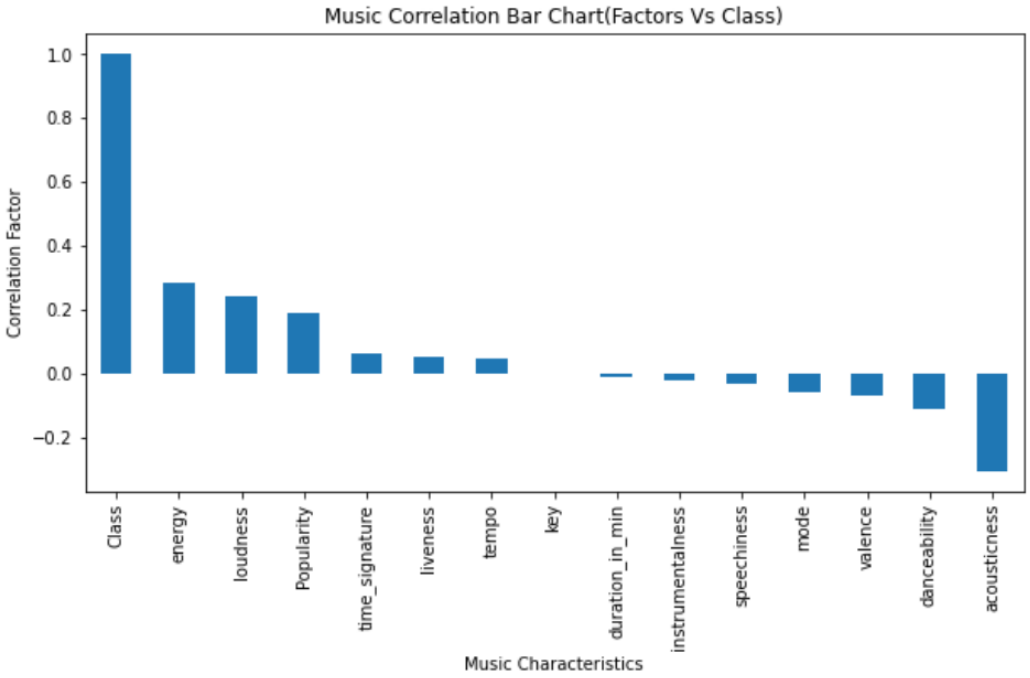
Insights:

Distribution	Music Characteristics
Symmetrical	1. Danceability 2. Valence
Left skewed	1. Energy 2. Loudness
Right skewed	1. Speechiness 2. Liveness 3. Acousticness 4. Instrumentalness 5. Duration in min
Multimodal	1. Key 2. Tempo
Have only 1 or 2 bars in their distribution	1. Mode 2. Time Signature

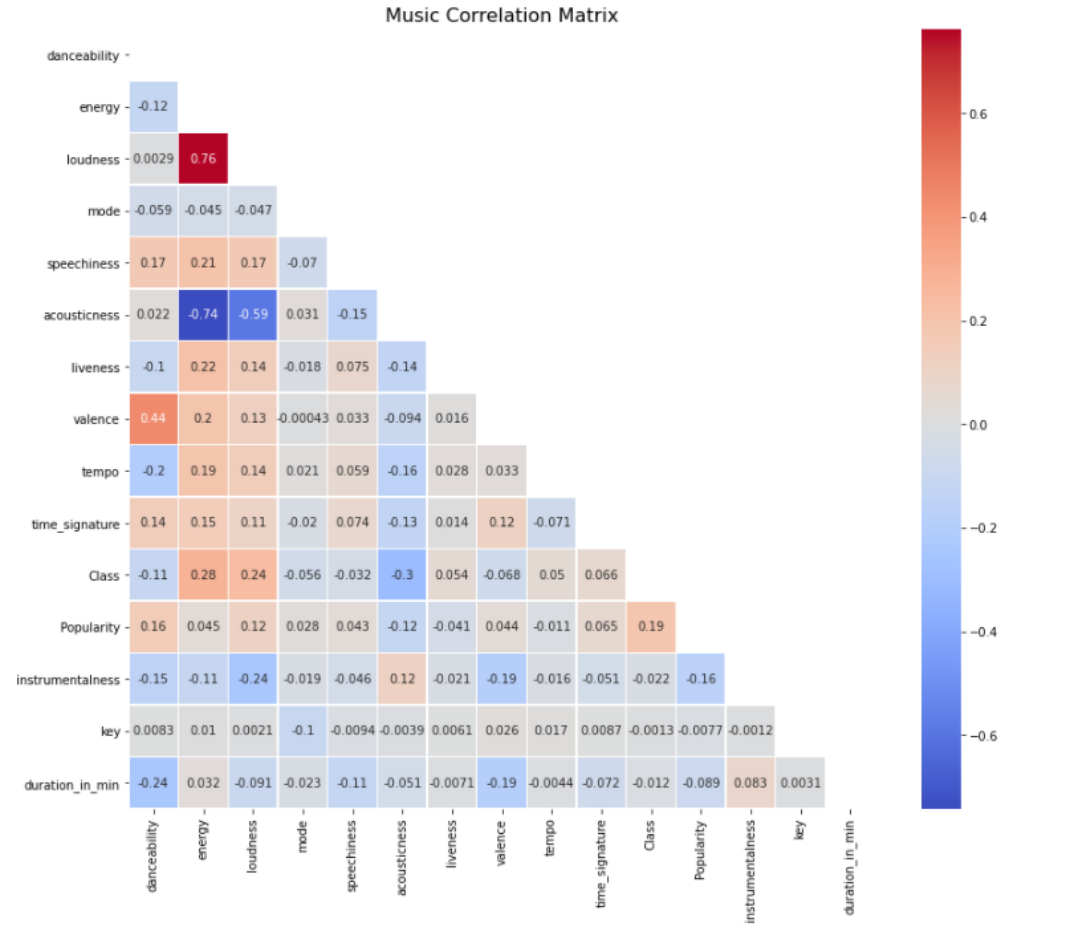
4.2 Correlation Analysis

Next, I did a Correlation Analysis to better understand the relationship between the 14 key factors affecting the class of the music

4.2.1 Music Correlation Bar Chart



4.2.2 Music Correlation Matrix



Insights:

Correlation between music characteristics and class
Overall correlation between the 13 variables and the Class is not very strong. However, I have identified four variables which had significant correlation with the Class

1. Acousticness Vs Class(-0.3)
2. Energy Vs Class(0.28)
3. Loudness Vs Class(0.24)

Correlation among music characteristics

High Correlation exist among these music characteristics

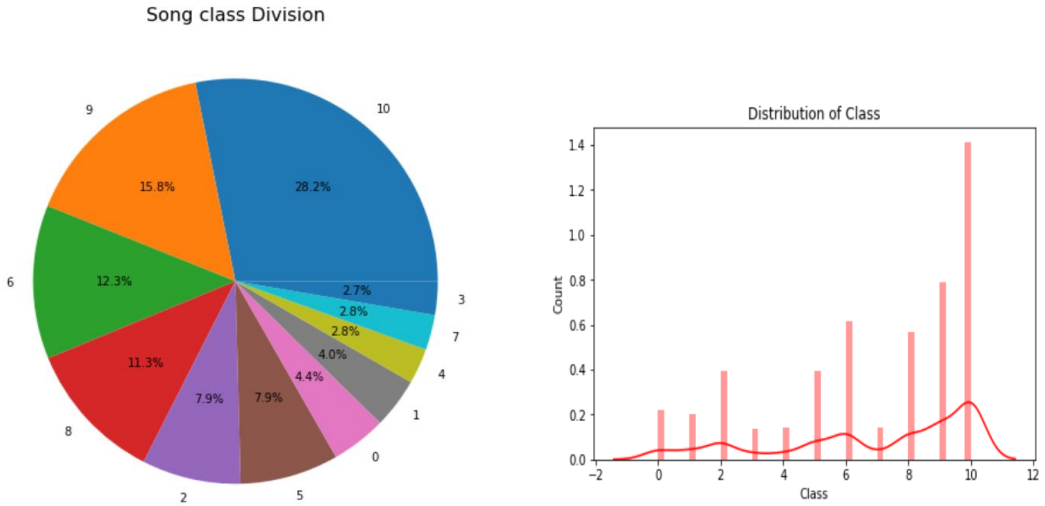
1. Loudness Vs Energy (0.76)
2. Acousticness Vs Energy (-0.74)
3. Acousticness Vs Loudness (-0.59)

Hence, for the modelling stage, I will be doing feature selection to avoid multicollinearity

4.3 Song Class Analysis

Next, I will be analysing the class of the music to get a better understanding of it.

4.3.1 Distribution and Proportion of Song Class



As we can see from the above pie chart, Class 10 stands the highest percentage of 28.2% followed by class 9 then 6. The classes with the lower data examples are classes 4,3,7,0. They have around 2% to 4% of data examples which is around 360 to 540.

There is uneven distribution between 11 classes where Class 10 hits the highest percentage of 28.2%. There are many minority classes such as class 7,3,4,0 where their population is less than 5%. Hence, during modelling stage, I will be balancing the data.

4.4 Further Analysis of Music Characteristics and Class

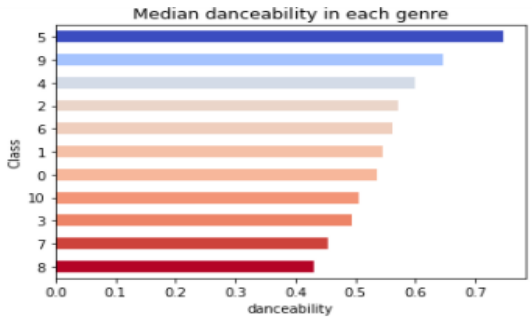
4.4.1 Median Music Characteristic across All Classes

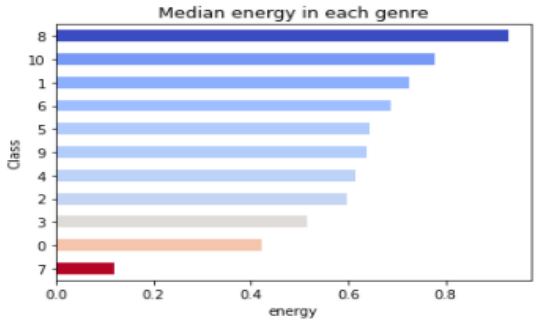
```
def plot_genre_horizontal_bar(col, title=None):
    data = df.groupby('Class')[col].median().sort_values()

    cmap = plt.cm.coolwarm_r
    norm = plt.Normalize(vmin=data.min(), vmax=data.max())
    colors = [cmap(norm(value)) for value in data]

    data.plot.barh(color=colors)
    plt.xlabel(col)
    plt.title(title, fontdict={'size': 13})
    plt.show()
```

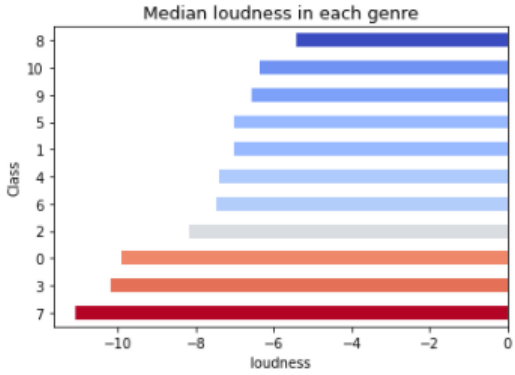
```
labels =["danceability","energy","loudness","speechiness","acousticness","liveness","valence","tempo","duration_in min","ins
for i in range(len(labels)):
    plot_genre_horizontal_bar(labels[i],title="Median "+labels[i]+"in each genre")
```

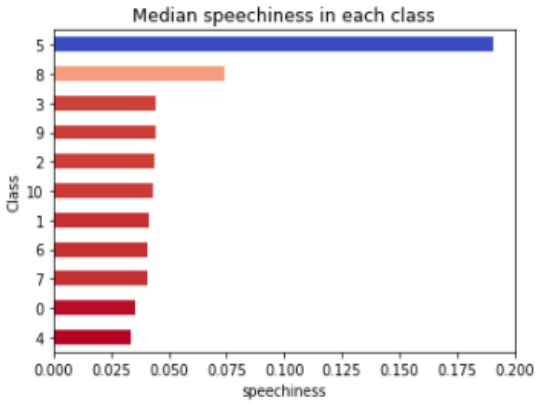
Next, I create a plot_genre_horizontal_bar () function to group the classes and sort it by the median value of the music characteristic. I then plotted each music characteristic across each class of songs out. Take note that I normalized the value of the music characteristic for fairer comparison.

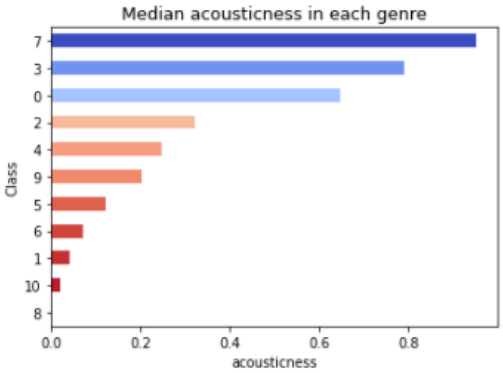
***Graphs will be displayed first followed by insights.**

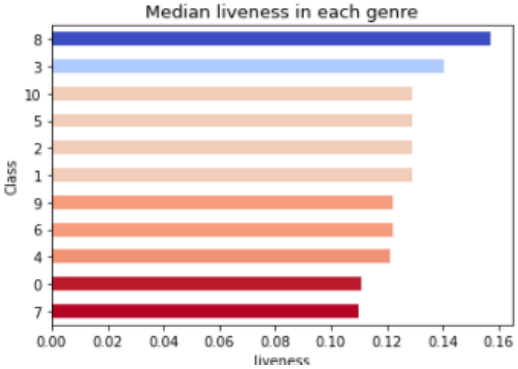


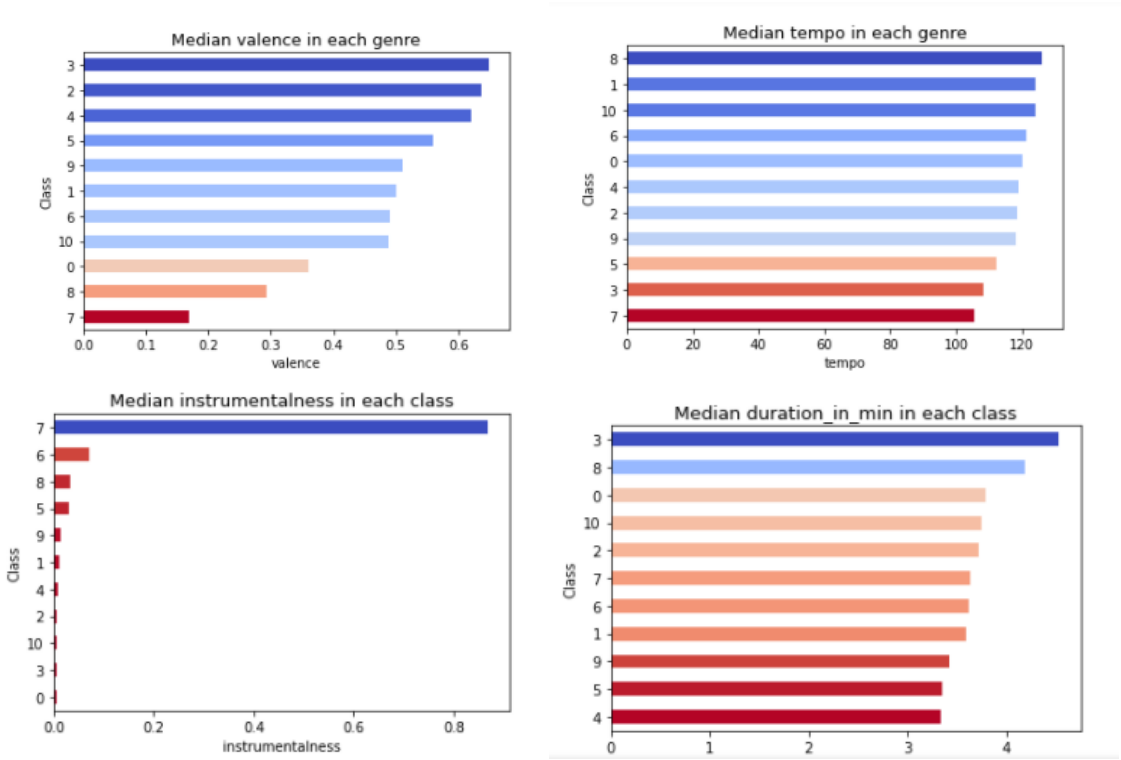












Insights on music characteristics across different music genres/classes:

Class with highest danceability: **Class 5**

Class with highest energy: **Class 8**

Class with highest volume(loudness): **Class 8**

Class with highest speechiness: **Class 5**

Class with highest acousticness**: Class 7**

Class with highest liveness: **Class 8**

Class with highest valence: **Class 3**

Class with highest tempo: **Class 8**

Class with highest duration: **Class 3**

Class with highest instrumentalness: **Class 7**

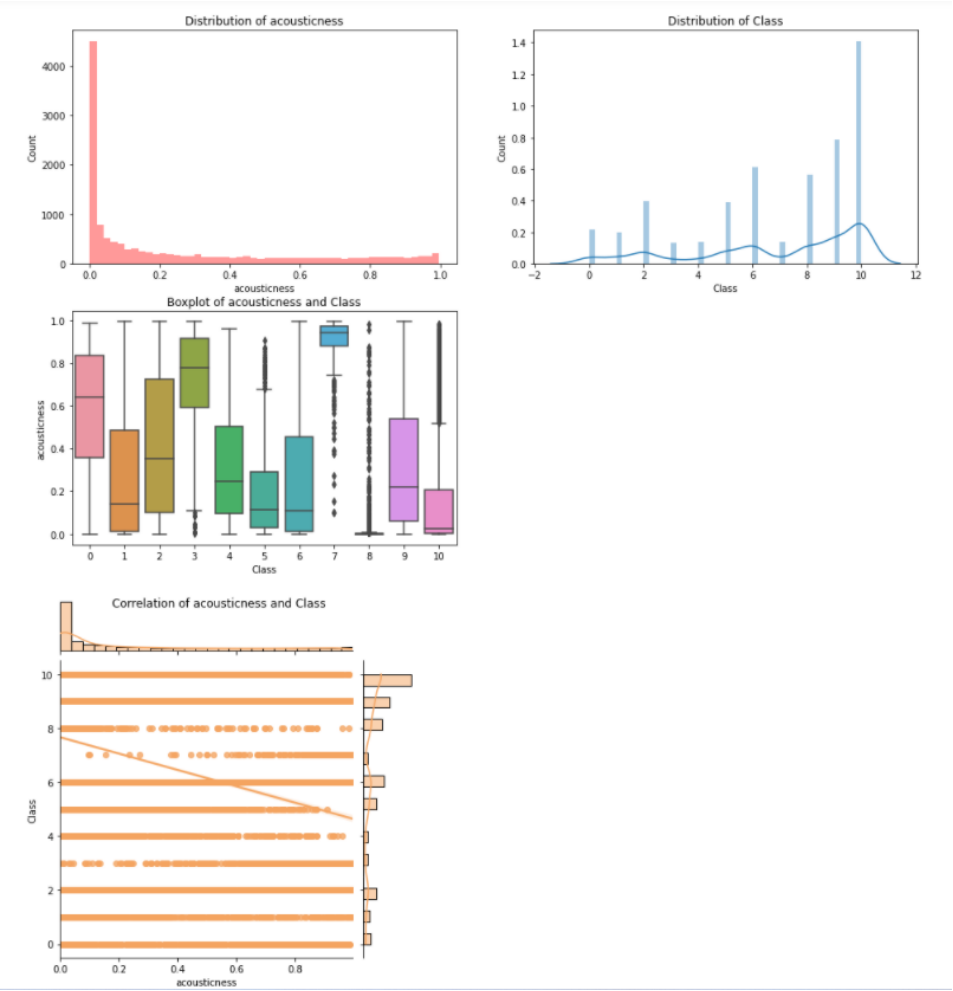
Further Interesting Insights:

1. Songs of class 8 have the highest value for most music characteristics.
2. For instrumentalness in each class, all classes except for class 7 have near to 0.0 instrumentalness.

4.4.2 Deeper Dive into Analysis between Music Characteristic and Class

In this section, we will only focus on the 3 most significant variables(Acousticness, Energy and Loudness) as mentioned in the correlation matrix above. I will dive deeper to analyse the relationship between each of the 3 factors and the Class to produce some insights from it.

4.4.2.1 Deeper Dive into Analysis between Acousticness and Class



Insights:

a) **<u>Distribution Chart of acousticness:</u>**
The distribution chart of **acousticness** is right skewed which means that the quantity of acousticness is quite less in this dataset. The quantity of acousticness ranges between 0.0 to 1.0 g/dm^3 where the distribution mainly clusters around 0.0 to 0.2.

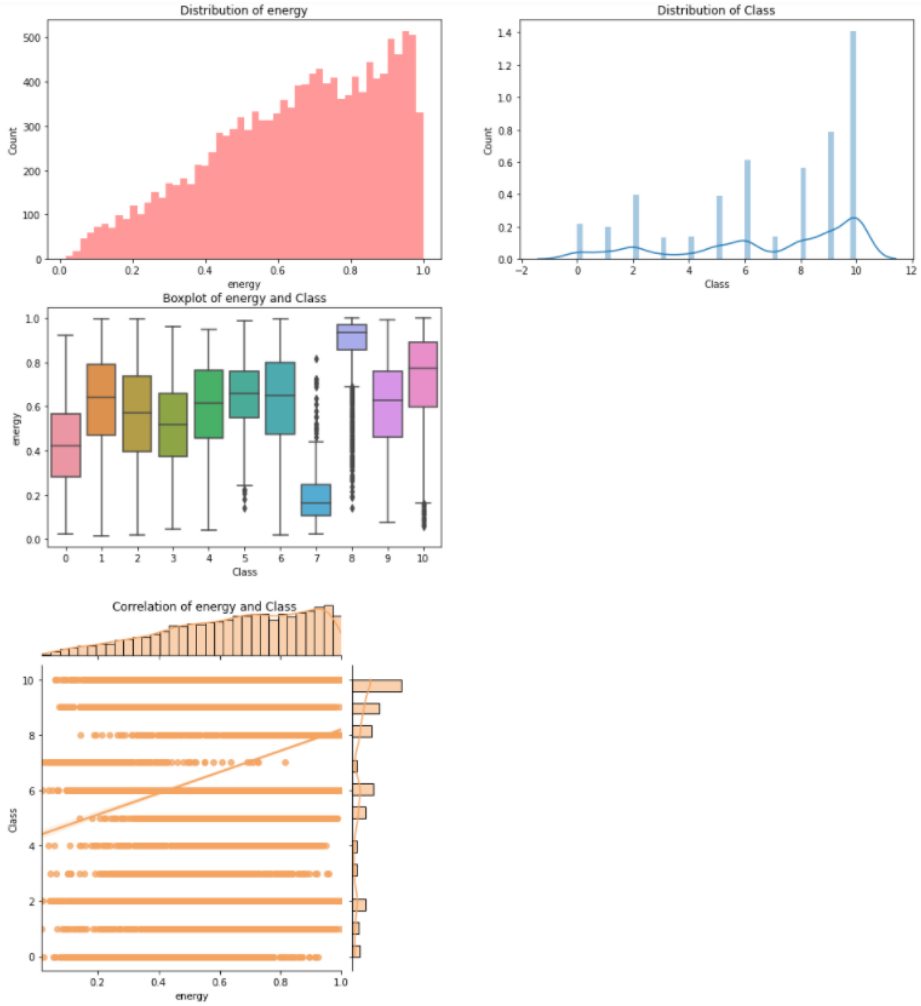
b) **<u>Distribution Chart of class:</u>**
We can see that the chart is left-skewed where most of the class is class 10.

c) **<u>Boxplot for acousticness:</u>**
From the boxplot, we can see that Classes 0,3,7 has a higher value of acousticness where the median value of acousticness is above 0.6. Whereas for classes 1, 2, 4, 5, 6, 8, 9 and 10, song in this class have a lower value of acousticness where median value is above 0.4.

d) **<u>Joint Plot for acousticness and class:</u>**
We have combined the two distribution charts together to see their relationship. We learned that there is a negative relationship between acousticness and class as the orange line shows a negative linear graph. This evidences the results from the boxplot where the increase in acousticness results in a

decrease in class number. The graph is quite steeped up which shows that the relationship between acousticness and class is quite strong.

4.4.2.2 Deeper Dive into Analysis between Energy and Class



Insights:

a) **Distribution Chart of Energy:**
The distribution chart of energy is very left skewed which means that the quantity of energy is quite a lot in this dataset. The quantity of energy ranges between 0.0 to 1.0 where the distribution is higher when value of energy increases

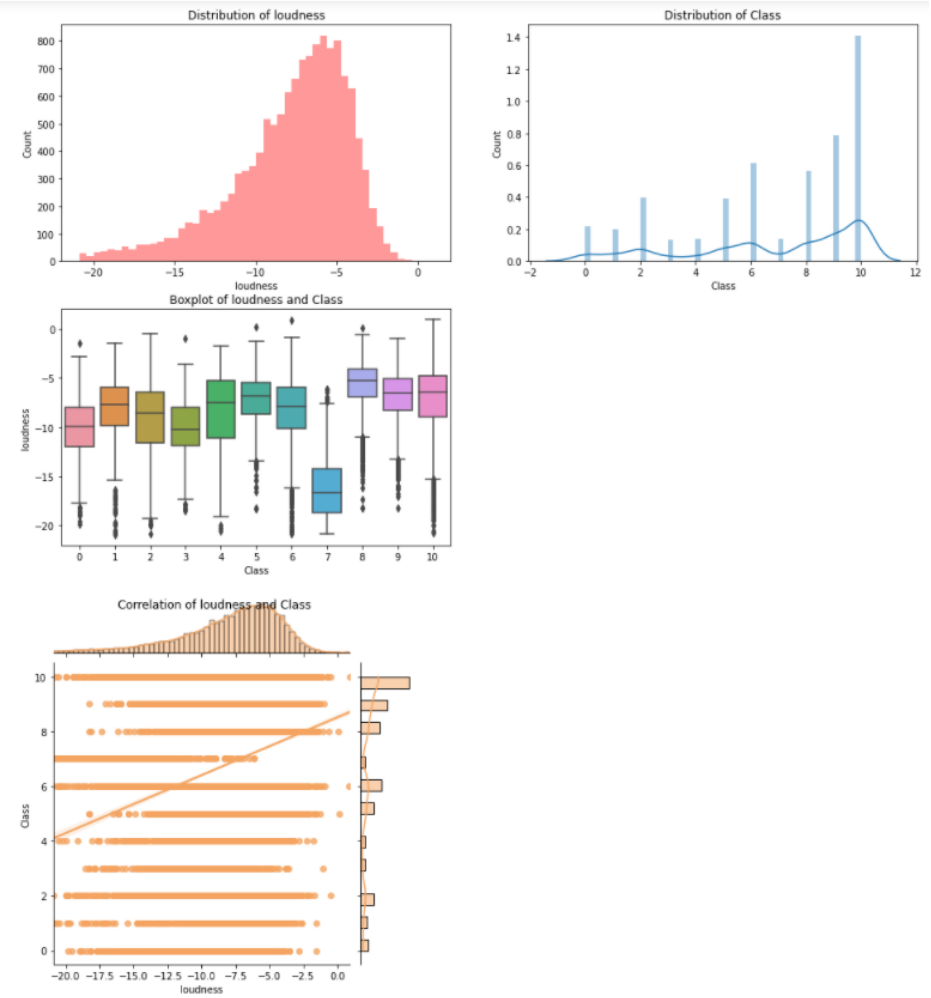
b) **Distribution Chart of class:**
We can see that the chart is left-skewed where most of the class is class 10.

c) **Boxplot for energy:**
From the boxplot, we can see that for most classes, their energy level is quite high for most classes the median value of energy is 0.5 and above. For Class 7, the energy level is very low as compared to other Classes with the median value of energy at 0.1~.

d) **Joint Plot for energy and class:**
We have combined the two distribution charts together to see their relationship. We learned that there is a positive relationship between energy and class as the orange line shows a positive linear graph. This evidences the results from the boxplot where the increase in energy results in an increase in

class number. The graph is quite steeped up which shows that the relationship between energy and class is quite strong.

4.4.2.3 Deeper Dive into Analysis between Loudness and Class



Insights:

a) **Distribution Chart of loudness:**
The distribution chart of loudness is very left-skewed which means that the quantity of loudness is quite a lot in this dataset. The quantity of loudness is higher when value is higher (around -10 to -5) 。

b) **Distribution Chart of class:**
We can see that the chart is left-skewed where most of the class is class 10.

c) **Boxplot for loudness:**
From the boxplot, we can see that for most classes, their loudness level is quite high for most classes the median value of loudness is -10 and above. For Class 7, the loudness level is very low as compared to other Classes with the median value of loudness at -17~.

d) **Joint Plot for loudness and class:**
We have combined the two distribution charts together to see their relationship. We learned that there is a positive relationship between loudness and class as the orange line shows a positive linear graph. This evidences the results from the boxplot where the increase in loudness results in an increase in

class number. The graph is quite steeped up which shows that the relationship between loudness and class is quite strong.

5. Data Modelling, Evaluation and Prediction

5.1 Preparing data for modelling

For the preparation of data for modelling, my methodology would be:

A) Drop fields that are not used for classification
B) Split Input Features and Label
C) Data Balancing
D) Feature Scaling
E) Feature Selection
F) Train Test Split

*Note: I have tried another **failed approach** which is to:*

A) Drop fields that are not used for classification
B) Split Input Features and Label
C) Train Test Split
D) Data Balancing
E) Feature Scaling
F) Feature Selection

In this failed approach, I train test split the data before balancing and scaling it. This would mean that only the train data is balanced and scaled. Test data was not. The model could not perform well as the unseen data or test data was in a different format from the training data. Hence, this approach turns out to give minimal accuracy and precision of 10% to 40%.

After seeking out some online resource and consults, I will be following the first methodology that I mentioned instead of the failed one.

5.1.1 Dropping fields not used for classification

Let us first drop columns that are not definitely involved in the prediction of class such as popularity, song name and track name. Popularity is not a music characteristic and should not be used for predicting classes. Whereas the song name and track name is just a unique identifier for each row.

```
df_modelling=df.drop(['Artist Name',"Track Name","Popularity"],axis=1)
```

```
df_modelling
```

	danceability	energy	loudness	mode	speechiness	acousticness	liveness	valence	tempo	duration_in min/ms	time_signature	Class	instrumentalness	key
0	0.854	0.564	-4.964	1	0.0485	0.017100	0.0849	0.8990	134.071	234596.0	4.0	5	0.251237	1.0
1	0.382	0.814	-7.230	1	0.0406	0.001100	0.1010	0.5690	116.454	251733.0	4.0	10	0.004010	3.0
2	0.434	0.614	-8.334	1	0.0525	0.486000	0.3940	0.7870	147.681	109667.0	4.0	6	0.000196	6.0
3	0.853	0.597	-6.528	0	0.0555	0.021200	0.1220	0.5690	107.033	173968.0	4.0	5	0.057497	10.0
4	0.167	0.975	-4.279	1	0.2160	0.000169	0.1720	0.0918	199.060	229960.0	4.0	10	0.016100	2.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
17991	0.166	0.109	-17.100	0	0.0413	0.993000	0.0984	0.1770	171.587	193450.0	3.0	6	0.824000	7.0
17992	0.638	0.223	-10.174	0	0.0329	0.858000	0.0705	0.3350	73.016	257067.0	4.0	2	0.000016	11.0
17993	0.558	0.981	-4.683	0	0.0712	0.000030	0.1290	0.2620	105.000	216222.0	4.0	8	0.000136	4.0
17994	0.215	0.805	-12.757	0	0.1340	0.001290	0.2560	0.3550	131.363	219693.0	4.0	8	0.017450	6.0
17995	0.400	0.853	-5.320	0	0.0591	0.006040	0.3340	0.3770	138.102	182227.0	4.0	10	0.212000	4.0

17996 rows × 15 columns

5.1.2 Split Input Features and label

```
X=df_modelling[["danceability","energy","loudness","mode","speechiness","acousticness","liveness","valence","tempo","duration_in
y=df_modelling[["Class"]]
```

```
print(X.shape)
print(y.shape)
```

```
(17996, 13)
(17996, 1)
```

5.1.3 Data Balancing

I then balanced the whole dataset (X,y) using imblearn.over_sampling.

```
#Enter your codes here
y.Class.value_counts()
```

```
10    3800
9     2125
6     1654
8     1524
2     1065
5     1062
0      592
1      543
4      380
7      378
3      361
Name: Class, dtype: int64
```

```
from imblearn.over_sampling import SMOTE
smote=SMOTE()
X,y=smote.fit_resample(X,y)
```

```
#Enter your codes here
y.Class.value_counts()
```

```
0     3800
1     3800
2     3800
3     3800
4     3800
5     3800
6     3800
7     3800
8     3800
9     3800
10    3800
Name: Class, dtype: int64
```

5.1.4 Feature Scaling

I then did feature scaling using sklearn.preprocessing.MinMaxScaler () method to scale the data/normalize the data to a range of 0 to 1 for fairer comparison.
Noticed that I only feature scaled non-categorical variable. For encoded categorical variable such as "mode", “time_signature" and "key", I did not scale it because it is already encoded. (E.g., Key=B# is 9)

```
scaler = preprocessing.MinMaxScaler()
X[['danceability', 'energy', 'loudness', 'acousticness',
       'speechiness', 'liveness', 'valence', 'tempo', 'duration_in_min', 'instrumentalness']]
=scaler.fit_transform(X[['danceability', 'energy', 'loudness', 'acousticness',
       'speechiness', 'liveness', 'valence', 'tempo', 'duration_in_min', 'instrumentalness']])
```

5.1.5 Feature Selection

Note that for feature selection, I will be carrying it out in my first model(Decision Tree) as I will be able to compare the results before and after selecting the features needed to increase model performances. I would need to develop a model first to carry out this process.
** *Refer to 5.2.2.1- Feature selection stage*

5.1.6 Train Test Split

Lastly, after preparing the data for modelling, I then performed train_test_split

```
#Enter your codes to split the data into X_train, y_train, X_test and y_test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify=y)
```

5.2 Modelling

From our analysis above, we now have a better understanding on the relationship between each music's characteristics and classes.

5.2.1 Methodology

5.2.1.1 Selecting Classification Models

From our analysis above, we now have a better understanding on the relationship between each music's characteristics and classes. As our objective is to better understand what factors affect the class (type of music), we will need to use classification model.

There are many types of classification model:

1. Decision Tree Classifier
2. K-Neighbors Classifier
3. Random Forest Classifier
4. XGBoost Classifier
5. Gradient Boosting Classifier
6. Multinomial Naive Bayes
7. Stochastic Gradient Descent(SGD) Classifier
8. One-Vs-Rest Classifier
9. Neural Network
10. *Logistic Regression (only for binary class classification)*
11. *Support vector Machine (only for binary class classification)*

However, I will omitting **Logistic Regression and SVM** as both can only are for binary classification but in this case, we have 11 classes.

5.2.1.2 How will I be carrying out this process?

1. Try out each model
2. Fine tune parameters to give the BEST results for each model
3. Compare the best results of each model
4. Derive the model that gives the BEST results out of all models

5.2.1.3 Metrics used - Why I use cross validation?

I will be using cross validation rather than classification report of precision, recall, f1-score and accuracy, as it allows a better representation assessment of the model performance as it test each and every portion of the dataset for each model. This is because our dataset. Also, given that the dataset is small, it is better to get a good representative of each portion of our dataset to get a good and reliable test result.

5.2.1.4 Metrics used - What area of metric will I be focusing on?

However, I will be focusing more on precision in this project. This is because we are not targeting on health-related classification or prediction where false negatives are important. In this case, more false negatives are not as costly. Hence, we be focusing on false positive, preciseness of our classification.

5.2.2 Models

5.2.2.1 Model I - Decision Tree Classifier

The first model I will be using would be using is decision tree classifier.

Step A: Building Model - Decision Tree

```
#Enter your codes here to train a DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier
tr = DecisionTreeClassifier(random_state=42)
tr.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
#Enter your codes to print out the depth of the treeI will be generating the classif

tr.get_depth()
```

```
39
```

I imported the DecisionTreeClassifier function from the sklearn.tree library. Take note that this model I build includes no parameters, only a random state.

Step B: Model Evaluation

We can see that results for **classification report performance are lower than our cross-validation performance**. Hence, in the next few models, **I will be using cross validation** as it allows me to test and train every portion of the dataset. Also, given that the dataset is small, it is better to get a good representative of each portion of our dataset to get a good and reliable test result.

```
from sklearn.metrics import classification_report
y_pred = tr.predict(X_test)
print(y_pred)
print(classification_report(y_test,y_pred))
```

```
[ 9  8 10 ...  3  7  8]
              precision    recall  f1-score   support

           0       0.62      0.64      0.63       760
           1       0.54      0.56      0.55       760
           2       0.53      0.52      0.52       760
           3       0.79      0.85      0.82       760
           4       0.69      0.76      0.72       760
           5       0.76      0.72      0.74       760
           6       0.39      0.38      0.39       760
           7       0.90      0.90      0.90       760
           8       0.76      0.72      0.74       760
           9       0.43      0.41      0.42       760
          10       0.35      0.33      0.34       760

    accuracy                           0.62      8360
   macro avg       0.61      0.62      0.62      8360
weighted avg       0.61      0.62      0.62      8360
```

Cross Validation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(tr, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.6368440614708794
Average Precision: 0.6336196536059664
Average Recall: 0.636840042961163
Average F1: 0.634243373293953
```

I then performed cross validation of 10-folds to get a reliable precision.
At this stage, the model performance is at an **average performance**. We can see that accuracy is 0.64, precision is 0.63, recall is 0.64 and F1 is 0.63.

Feature Selection

I will be using mlxtend.feature_selection to do **forward feature selection** to find the best subset of features that gives highest precision.

```
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
sfsl=SFS(tr,k_features=13,forward=True,scoring="precision_macro",cv=10)
sfsl.fit(X,y)
sfsl.subsets_
```

```
    'duration_in_min',
    'instrumentalness',
    'key')},
 12: {'feature_idx': (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12),
  'cv_scores': array([0.60750589, 0.60128516, 0.61191168, 0.63031825, 0.63883637,
         0.64906892, 0.64467488, 0.63550017, 0.66396102, 0.65680919]),
  'avg_score': 0.633987153755126,
  'feature_names': ('danceability',
   'energy',
   'mode',
   'loudness',
   'acousticness',
   'speechiness',
   'liveness',
   'valence',
   'tempo',
   'duration_in_min',
   'instrumentalness',
   'key')},
```

Looking at the results, **subset 12** which contains 'danceability', 'energy', 'loudness', 'acousticness', ' speechiness ', 'valence', 'tempo', 'duration_in_min', 'time_signature', 'instrumentalness', 'key' **gives highest precision of 0.6339**.
Hence, we will be using this subset of features to build a new decision tree model and do cross validation to find out the performance of the model.

```
X=df_modelling[['danceability',
   'energy',
   'loudness',
   'acousticness',
   'speechiness',
   'valence',
   'tempo',
   'duration_in_min',
   'time_signature',
   'instrumentalness',
   'key']]
y=df_modelling[["Class"]]

from imblearn.over_sampling import SMOTE
smote=SMOTE()
X,y=smote.fit_resample(X,y)

scaler = preprocessing.MinMaxScaler()
X[['danceability', 'energy', 'loudness', 'acousticness',
       'speechiness', 'valence', 'tempo', 'duration_in_min', 'instrumentalness']]=scaler.fit_transform(X[['danceability', 'ener
       'speechiness', 'valence', 'tempo', 'duration_in_min', 'instrumentalness']])
#Enter your codes to split the data into X_train, y_train, X_test and y_test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,stratify=y)

tr = DecisionTreeClassifier(random_state=42)
tr.fit(X_train, y_train)
```

```
DecisionTreeClassifier(random_state=42)
```

```
result = cross_validate(tr,X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.639762482555988
Average Precision: 0.6361851740258518
Average Recall: 0.6397611166237712
Average F1: 0.6370620010225501
```

Looks like the results have improved after using subset 12 of features. It increase from 0.63 range to 0.64 range of each of the performance metric.
Hence, we will be using this **subset of features consisting of 'danceability', 'energy', 'loudness', 'acousticness', ' speechiness ', 'valence', 'tempo', 'duration_in_min', 'time_signature', 'instrumentalness', 'key' for the rest of our models.** ¶

Step C: Tuning Parameters

How will I be tuning the parameters?

I will be using GridSearchCV to do hyper-parameter to find the best combination of parameters that gives highest precision

Hyper-parameter tuning – GridSearchCV

Now, I will be using Grid Search CV to find the best combination of parameters and the corresponding value.

```
tr = DecisionTreeClassifier(random_state=42)
```

```
from sklearn.model_selection import GridSearchCV
```

```
params = {
    'max_depth':list(range(1,50)),
     "min_samples_split":list(range(1,10)),
    "min_samples_leaf":list(range(1,10)),
    'criterion': ["gini", "entropy"]
}
```

```
# Instantiate the grid search model
grid_search = GridSearchCV(estimator=tr,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "precision_macro")
```

```
%%time

with tf.device('/GPU:0'):
    grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 7938 candidates, totalling 31752 fits
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:918: UserWarning: One or more of the test scores
are non-finite: [       nan 0.06636748 0.06636748 ... 0.5313526  0.5313526  0.5313526 ]
  warnings.warn(
```

```
Wall time: 25min 43s
```

```
tr_best=grid_search.best_estimator_
tr_best
```

```
DecisionTreeClassifier(criterion='entropy', max_depth=22, random_state=42)
```

```
tr_best=DecisionTreeClassifier(criterion='entropy', max_depth=22, random_state=42)
```

```
result = cross_validate(tr_best, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.6449539343449618
Average Precision: 0.6400327611905886
Average Recall: 0.6449605378884193
Average F1: 0.6416296859707229
```

We see that by using a combination of parameters [**criterion='entropy', max_depth=22, random_state=42**], the overall results did improve.

Step D: Model Insights

Now let me do a comparison of all the decision tree models I have fine-tuned.

After much tuning, let's view the summary of model results:

Tuning Methods	Model information	Accuracy	Precision	Recall	F1-score
Original Model (No Parameters)	DecisionTreeClassifier(random_state=42)	0.64	0.64	0.64	0.64
Tuned Model	DecisionTreeClassifier(criterion='entropy', max_depth=22, random_state=42)	0.65	0.64	0.65	0.64

From this table we can see that we can use the last model which is **DecisionTreeClassifier(criterion='entropy', max_depth=22, random_state=42)** as it **gives the higher precision, accuracy, recall and f1-score.**

Conclusion: Highest precision of tuned decision tree is 64%.

Step E: Predictions

Now, that we have derived the best decision tree model, we can use it to do some predictions on unseen data.

```
d = {'danceability': [0.473, 0.55], 'energy': [0.23, 0.33],"loudness":[-1,-9],
"acousticness":[0.33,0.01],"speechiness":[0.02,0.1],"valence":[0.32,0.90],"tempo":[0.10,0.90],
"duration_in_min":[7,9],"time_signature":[3,4],"instrumentalness":[0.23,0.78],"key":[3,4]}
df = pd.DataFrame(data=d)
tr_best.fit(X_train,y_train)
df['class']=tr_best.predict(df)
display(df)
```

	danceability	energy	loudness	acousticness	speechiness	valence	tempo	duration_in_min	time_signature	instrumentalness	key	class
0	0.473	0.23	-1	0.33	0.02	0.32	0.1	7	3	0.23	3	0
1	0.550	0.33	-9	0.01	0.10	0.90	0.9	9	4	0.78	4	4

5.2.2.2 Model II – K-Neighbors Classifier

The second model I will be using would be using is K-Neighbors classifier.

Step A: Building Model – K-Neighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier as KNC

classifier = KNC()
classifier.fit(X_train, y_train)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

```
KNeighborsClassifier()
```

I imported the KNeighborsClassifier function from the sklearn.neighbor's library. Take note that this model I build includes no parameters, only a random state.

Step B: Model Evaluation

Cross Validation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(classifier, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.6529678100578484
Average Precision: 0.6446341092518711
Average Recall: 0.6529785807311853
Average F1: 0.6346146245080541
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable accuracy. At this stage, the model performance is performing good with precision at 0.64 and accuracy and recall at 0.65.

Step C: Tuning Parameters

Now, I will be tuning the parameters using GridSearchCV.

```
knn_2 = KNC()
params = {
    'leaf_size': list(range(1,50)),
    'n_neighbors': list(range(1,100)),
    'p': [1,2]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=knn_2,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

```
%%time
grid_search.fit(X_train, y_train)
```

```
knc_best=grid_search.best_estimator_
knc_best
```

```
KNeighborsClassifier(leaf_size=25, n_neighbors=1, p=1)
```

```
knc_best=KNC(leaf_size=25, n_neighbors=1, p=1)
```

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(knc_best, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.7947866385158336
Average Precision: 0.7880893693831723
Average Recall: 0.7947811630522723
Average F1: 0.7864581015088197
```

Looks like the model performance has improved a lot. We see that by using a combination of paramet ers [leaf_size=25, n_neighbors=1, p=1] to give the **best precision of 0.79,** accuracy of 0.79, recall of 0.79 and f1-score=0.79.

Step D: Model Insights

Now let me do a comparison of all the K-Neighbors Classifier I have fine-tuned.

After much tuning, let's view the summary of model results:

Tuning Methods	Model information	Accuracy	Precision	Recall	f1-score
Original Model (No Parameters)	KNeighborsClassifier()	0.65	0.64	0.65	0.63
Grid search CV	KNeighborsClassifier(leaf_size=25, n_neighbors=1, p=1)	0.79	0.79	0.79	0.79

From this table we can see that we can use the last model which consist of **KNeighborsClassifier (**le af_size=25, n_neighbors=1, p=1**)** as it **gives best precision of 0.79, accuracy of 0.79, recall of 00.7 9 and f1-score=0.79.**

Conclusion: Highest precision of tuned K-Neighbors Classifier is 79%.

Step E: Predictions

Now, that we have derived the best K-Neighbors Classifier model, we can use it to do some predictions on unseen data.

```
d = {'danceability': [0.473, 0.55], 'energy': [0.23, 0.33],"loudness":[-1,-9],
"acousticness":[0.33,0.01],"speechiness":[0.02,0.1],"valence":[0.32,0.90],"tempo":[0.10,0.90],
"duration_in_min":[7,9],"time_signature":[3,4],"instrumentalness":[0.23,0.78],"key":[3,4]}
df = pd.DataFrame(data=d)
knc_best.fit(X_train,y_train)
df['class']=knc_best.predict(df)
display(df)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversionWarning: A column-vector y w
as passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

	danceability	energy	loudness	acousticness	speechiness	valence	tempo	duration_in_min	time_signature	instrumentalness	key	class
0	0.473	0.23	-1	0.33	0.02	0.32	0.1	7	3	0.23	3	3
1	0.550	0.33	-9	0.01	0.10	0.90	0.9	9	4	0.78	4	1

5.2.2.3 Model III – Random Forest Classifier

The third model I will be using would be using is Random Forest Classifier

Step A: Building Model – Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
RFC=RandomForestClassifier()
RFC.fit(X_train, y_train)
```

```
<ipython-input-371-8594d7246451>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please ch
ange the shape of y to (n_samples,), for example using ravel().
  RFC.fit(X_train, y_train)
```

```
RandomForestClassifier()
```

I imported the RandomForestClassifier () function from the sklearn.ensemble library. Take note that this model I build includes no parameters.

Step B: Model Evaluation

Cross Validation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(RFC, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.7978252904557233
Average Precision: 0.79187426913976
Average Recall: 0.7978170218104016
Average F1: 0.788701617264592
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable accuracy. At this stage, the model performance is performing good with precision at 0.79 and accuracy and recall at 0.80.

Step C: Tuning Parameters

Now, I will be tuning the parameters using GridSearchCV.

```
]:
with tf.device('/GPU:0'):
    from sklearn.model_selection import RepeatedStratifiedKFold
    RFC2 = RandomForestClassifier()
    n_estimators = [10, 100, 1000]
    max_features = ['sqrt', 'log2']
    # define grid search
    grid = dict(n_estimators=n_estimators,max_features=max_features)
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    grid_search = GridSearchCV(estimator=RFC2, param_grid=grid, n_jobs=-1, cv=cv, scoring='precision_macro',error_score=0)
    grid_result = grid_search.fit(X, y)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\model_selection\_search.py:880: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  self.best_estimator_.fit(X, y, **fit_params)
```

```
]: print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
```

```
Best: 0.795515 using {'max_features': 'sqrt', 'n_estimators': 1000}
```

```
]: RFC_best = RandomForestClassifier(max_features='sqrt',n_estimators=1000)
   RFC_best
```

```
]: RandomForestClassifier(max_features='sqrt', n_estimators=1000)
```

```
]: RFC_best=RandomForestClassifier(max_features='sqrt', n_estimators=1000)
```

```
]: #Enter your codes to use cross_validation here
   from sklearn.model_selection import cross_validate

   result = cross_validate(RFC_best, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
   print("Average Accuracy:", np.mean(result['test_accuracy']))
   print("Average Precision:", np.mean(result['test_precision_macro']))
   print("Average Recall:", np.mean(result['test_recall_macro']))
   print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.8050500324798812
Average Precision: 0.8004426710191744
Average Recall: 0.8050405544639864
Average F1: 0.7963544425675914
```

Looks like the model performance has by a little. We see that by using a combination of parameters {' max_features': 'sqrt', 'n_estimators': 1000}
to give the best precision of 0.80, accuracy and recall of 0.80

Step D: Model Insights

Now let me do a comparison of all the Random Forest Classifiers I have fine-tuned.

After much tuning, let's view the summary of model results:

Tuning Methods	Model information	Accuracy	Precision	Recall	f1-score
Original Model (No Parameters)	RandomForestClassifier()	0.80	0.79	0.80	0.79
Grid search CV	RandomForestClassifier(max_features='sqrt', n_estimators=1000)	0.80	0.80	0.80	0.80

From this table we can see that we can use the last model which consist of Random Forest Classifier (max_features='sqrt', n_estimators=1000) as it gives best precision of 0.80, accuracy and recall of 0.80.

Conclusion: Highest precision of tuned Random Forest Classifier is 80%.

Step E: Predictions

```
d = {'danceability': [0.473, 0.55], 'energy': [0.23, 0.33],"loudness":[-1,-9],
"acousticness":[0.33,0.01],"speechiness":[0.02,0.1],"valence":[0.32,0.90],"tempo":[0.10,0.90],
"duration_in_min":[7,9],"time_signature":[3,4],"instrumentalness":[0.23,0.78],"key":[3,4]}
df = pd.DataFrame(data=d)
RFC_best.fit(X_train,y_train)
df['class']=RFC_best.predict(df)
display(df)
```

```
<ipython-input-294-345b35654605>:5: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please ch
ange the shape of y to (n_samples,), for example using ravel().
  RFC_best.fit(X_train,y_train)
```

	danceability	energy	loudness	acousticness	speechiness	valence	tempo	duration_in_min	time_signature	instrumentalness	key	class
0	0.473	0.23	-1	0.33	0.02	0.32	0.1	7	3	0.23	3	2
1	0.550	0.33	-9	0.01	0.10	0.90	0.9	9	4	0.78	4	2

5.2.2.4 Model IV - XGBoost classifier

The fifth model I will be using would be using is XGBoost Classifier.

Step A: Building Model – XGBoost Classifier

```
from xgboost import XGBClassifier

xgb = XGBClassifier()
xgb.fit(X_train, y_train)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier is d
eprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encoder=F
alse when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..., [num
_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[11:03:22] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly s
et eval_metric if you'd like to restore the old behavior.
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=16,
              num_parallel_tree=1, objective='multi:softprob', predictor='auto',
              random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None,
              subsample=1, tree_method='exact', validate_parameters=1,
              verbosity=None)
```

I imported the XGBClassifier function from the xgboost library. Take note that this model I build includes no parameters.

Step B: Model Evaluation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(xgb, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[10:36:20] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explici
tly set eval_metric if you'd like to restore the old behavior.
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier i
s deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_enc
oder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
```

```
[10:36:33] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explici
tly set eval_metric if you'd like to restore the old behavior.
Average Accuracy: 0.7339262814887758
Average Precision: 0.7332907976461368
Average Recall: 0.7339384554559832
Average F1: 0.7269116476683986
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable result.
At this stage, the model performance is quite good. We can see that precision is 0.73, accuracy is 0.73, recall is 0.73 and f1-score is 0.73.

Step C: Tuning Parameters

I will just be using grid search CV to do hyper-parameter tuning.

```
params={
 "max_depth"        : [ 3, 4, 5, 6, 8],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.1, 0.2 , 0.3, 0.4 ]}
```

```
from sklearn.model_selection import GridSearchCV
xgb_2 = XGBClassifier()
xgb_2.fit(X_train, y_train)
grid_search = GridSearchCV(estimator=xgb_2,
                           param_grid=params,
                           cv=4, n_jobs=-1, verbose=1, scoring = "accuracy")
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[07:49:02] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explicitly s
et eval_metric if you'd like to restore the old behavior.
```

```
%%time
grid_search.fit(X_train, y_train)

score_df = pd.DataFrame(grid_search.cv_results_)
score_df.head()
```

```
Fitting 4 folds for each of 80 candidates, totalling 320 fits
```

```
xgb_best=grid_search.best_estimator_
xgb_best
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.7,
              enable_categorical=False, gamma=0.0, gpu_id=-1,
              importance_type=None, interaction_constraints='',
              learning_rate=0.2, max_delta_step=0, max_depth=15,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=16, num_parallel_tree=1,
              objective='multi:softprob', predictor='auto', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
: #Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(xgb_best, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[10:42:24] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explici
tly set eval_metric if you'd like to restore the old behavior.

C:\Users\JiaYi\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier i
s deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_enc
oder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2,
..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[10:42:51] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.
3.0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Explici
tly set eval_metric if you'd like to restore the old behavior.
Average Accuracy: 0.8123956063928868
Average Precision: 0.8122806728460841
Average Recall: 0.8123934075475209
Average F1: 0.8075725628072515
```

Looks like the model performance has improved. We see that by using a combination of parameters **[base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False, gamma=0.0, gpu_id=-1, importance_type=None, interaction_constraints='', learning_rate=0.2, max_delta_step=0, max_depth=15, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=16, num_parallel_tree=1, objective='multi:softprob', predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None]** to give the best precision of 0.81, accuracy of 0.81, recall of 0.81 and f1-score of 0.81.

Step D: Model Insights

Now let me do a comparison of all the XGBoost Classifier I have fine-tuned.

After much tuning, let's view the summary of model results:

Tuning Methods	Model information	Accuracy	Precision	Recall	f1-score
Original Model (No Parameters)	XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1, enable_categorical=False, gamma=0, gpu_id=-1, importance_type=None, interaction_constraints='', learning_rate=0.300000012, max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=16, num_parallel_tree=1, objective='multi:softprob', predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)	0.73	0.73	0.73	0.73
Grid search CV	XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False, gamma=0.0, gpu_id=-1, importance_type=None, interaction_constraints='', learning_rate=0.2, max_delta_step=0, max_depth=15, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=16, num_parallel_tree=1, objective='multi:softprob', predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)	0.81	0.81	0.81	0.81

From this table we can see that we can use the tuned model which consist **of XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False, gamma=0.0, gpu_id=-1, importance_type=None, interaction_constraints='', learning_rate=0.2, max_delta_step=0, max_depth=15, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=16, num_parallel_tree=1, objective='multi:softprob', predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)** as it **gives the** best precision of 0.81, accuracy of 0.81, recall of 0.81 and f1-score of 0.81.

Conclusion: Highest precision of tuned XGBoost Classifier is 81%.

Step E: Predictions

Now, that we have derived the XGBoost Classifier model, we can use it to do some predictions on unseen data.

```
d = {'danceability': [0.473, 0.55], 'energy': [0.23, 0.33],"loudness":[-1,-9],
"acousticness":[0.33,0.01],"speechiness":[0.02,0.1],"valence":[0.32,0.90],"tempo":[0.10,0.90],
"duration_in_min":[7,9],"time_signature":[3,4],"instrumentalness":[0.23,0.78],"key":[3,4]}
df = pd.DataFrame(data=d)
xgb_best.fit(X_train,y_train)
df['class']=xgb_best.predict(df)
display(df)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifi
eprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_en
alse when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ..
_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[11:24:16] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoos
0, the default evaluation metric used with the objective 'multi:softprob' was changed from 'merror' to 'mlogloss'. Expli
et eval_metric if you'd like to restore the old behavior.
```

	danceability	energy	loudness	acousticness	speechiness	valence	tempo	duration_in_min	time_signature	instrumentalness	key	class
0	0.473	0.23	-1	0.33	0.02	0.32	0.1	7	3	0.23	3	10
1	0.550	0.33	-9	0.01	0.10	0.90	0.9	9	4	0.78	4	2

5.2.2.5 Model V – Gradient Boosting Classifier

The fourth model I will be using would be using is Gradient Boosting Classifier.

Step A: Building Model – Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
GBC=GradientBoostingClassifier()
GBC.fit(X_train, y_train)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
GradientBoostingClassifier()
```

I imported the GradientBoostingClassifier () function from the sklearn.ensemble library. Take note that this model I build includes no parameters.

Step B: Model Evaluation

Cross Validation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(GBC, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.5865564065663692
Average Precision: 0.5756074345990829
Average Recall: 0.586559766476198
Average F1: 0.5743339638877611
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable accuracy. At this stage, the model performance is performing good with precision at 0.58 and accuracy and recall at 0.59.

Note: I will not be tuning this classifier as the base model already has a much lower performance than the base model of other algorithms. Hence, even with tuning, it may only increase a little bit but not enough to hit the 70% range of precision/accuracy/recall. Also, since accuracy is low, I will not be using this model to do any prediction.

Conclusion: Highest precision of tuned Gradient Boosting Classifier is 58%.

5.2.2.6 Model VI – Multinomial NB

The sixth model I will be using would be using is Multinomial NB Model.

Step A: Building Model – Multinomial NB

```
from sklearn.naive_bayes import MultinomialNB

MNB = MultinomialNB()
MNB.fit(X_train, y_train)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
MultinomialNB()
```

I imported the MultinomialNB () function from the sklearn.naive_bayes library. Take note that this model I build includes no parameters, only a random state.

Step B: Model Evaluation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(MNB, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
Average Accuracy: 0.4122731396913762
Average Precision: 0.3756781065570345
Average Recall: 0.4122744116096624
Average F1: 0.377663304597036
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable accuracy. At this stage, the model performance is performing below average as compared with previous model with precision at 0.37 and accuracy and recall at 0.41.

Note: I will not be tuning this classifier as the base model already has a much lower performance *(30 to 40% range)* than the base model of other algorithms. Hence, even with tuning, it may only increase a little bit but not enough to hit the 70% range of precision/accuracy/recall. Also, since accuracy is low, I will not be using this model to do any predictions.

Conclusion: Highest precision of tuned Multinomial Naive Bayes Classifier is 38%.

5.2.2.7 Model VII – SGD Classifier

The seventh model I will be using would be using is SGD Classifier Model.

Step A: Building Model –SGD Classifier

```
from sklearn.linear_model import SGDClassifier
SGD = SGDClassifier()
SGD.fit(X_train, y_train)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was passed
when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
SGDClassifier()
```

I imported the SGDClassifier () function from the sklearn. linear_model library. Take note that this model I build includes no parameters.

Step B: Model Evaluation

Cross Validation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(SGD, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision is ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision is ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision is ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision is ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1245: UndefinedMetricWarning: Precision is ill-de
fined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

Average Accuracy: 0.42311169921626757
Average Precision: 0.4356611694349228
Average Recall: 0.42311530449147117
Average F1: 0.36351728042316084
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable accuracy. At this stage, the model performance is performing below average as compared with previous model with precision at 0.44 and accuracy and recall at 0.42.

Note: I will not be tuning this classifier as the base model already has a much lower performance *(40% range)* than the base model of other algorithms. Hence, even with tuning, it may only increase a little bit but not enough to hit the 70% range of precision/accuracy/recall. Also, since accuracy is low, I will not be using this model to do any predictions

Conclusion: Highest precision of SGD Classifier is 44%.

5.2.2.8 Model VIII – One Vs Rest Classifier

The seventh model I will be using would be using is **One Vs Rest Classifier** Model.

Step A: Building Model – One Vs Rest Classifier

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import LinearSVC

ORC=OneVsRestClassifier(LinearSVC(random_state=0))
ORC.fit(X_train, y_train)
```

```
OneVsRestClassifier(estimator=LinearSVC(random_state=0))
```

I imported the OneVsRestClassifier () function from the sklearn.multiclass library. Take note that this model I build includes no parameters except for an estimator.

Step B: Model Evaluation

Cross Validation

```
#Enter your codes to use cross_validation here
from sklearn.model_selection import cross_validate

result = cross_validate(ORC, X, np.ravel(y), cv=14, scoring=['accuracy', 'f1_macro', 'precision_macro', 'recall_macro'])
print("Average Accuracy:", np.mean(result['test_accuracy']))
print("Average Precision:", np.mean(result['test_precision_macro']))
print("Average Recall:", np.mean(result['test_recall_macro']))
print("Average F1:", np.mean(result['test_f1_macro']))
```

```
ase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, incre
ase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, incre
ase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\svm\_base.py:985: ConvergenceWarning: Liblinear failed to converge, incre
ase the number of iterations.
  warnings.warn("Liblinear failed to converge, increase "

Average Accuracy: 0.4718432689072896
Average Precision: 0.4229064059627106
Average Recall: 0.4718399478207922
Average F1: 0.41048095739272616
```

I then did model evaluation by performing cross validation of 10-folds to get a reliable accuracy. At this stage, the model performance is performing below average as compared with previous model with precision at 0.42 and accuracy and recall at 0.47.

Note: I will not be tuning this classifier as the base model already has a much lower performance *(40% range)* than the base model of other algorithms. Hence, even with tuning, it may only increase a little bit but not enough to hit the 70% range of precision/accuracy/recall

Conclusion: Highest precision of One-Vs-Rest Classifier is 42%.

5.2.2.9 Model IX – Neural Network Model

Step A: Preparing Data for neural network model

Different from machine learning model, for my neural network model requires the encoded class column to be converted in to dummy variables. Hence, I encoded and converted the y_train and y_test into 0 and 1s.

```
encoder = LabelEncoder()
encoder.fit(y_train)
encoded_Y_train = encoder.transform(y_train)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y_train = np_utils.to_categorical(encoded_Y_train)
encoder = LabelEncoder()
encoder.fit(y_test)
encoded_Y_test = encoder.transform(y_test)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y_test = np_utils.to_categorical(encoded_Y_test)
```

```
C:\Users\JiaYi\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: D
when a 1d array was expected. Please change the shape of y to (n_samples, ),
  return f(*args, **kwargs)
```

```
dummy_y_train
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)
```

Step B: Building and Tuning Model – Deep Neural Network

Next, I started to build my deep neural network. I decided to build a sequential model.

I created 4 dense layer with relu activation function each. The first one being the input layer and followed by 2 dense layers being the middle layers. For the input and middle layers I decided to give each layer 512 neurons. For the last dense layer, it is output layer. For last layer, since there were 11 classes of songs, the number of output neurons would be set at 11.

Tuning Process:

At first, I set the neurons at the range of 16 to 32 but I turned out to give a lower accuracy. Also, at first, there was only 1 layer. However, I realised the more layers I add the higher performance it gave. However, I stopped at the 4th layer as the model performance was starting to decrease. In my training process, I also added a early stopping call back as it is a good method that allows me to specify an arbitrary large number of training epochs and stop training once the model performance stops improving on a hold out validation dataset. I also tested with differen batch size and realized the optimal batch size was 10.

```
model1 = Sequential()
model1.add(Dense(512, input_shape = (X_train.shape[1],)))
model1.add(Activation('relu'))
model1.add(Dropout(0.1))

model1.add(Dense(512, input_shape = (X_train.shape[1],)))
model1.add(Activation('relu'))
model1.add(Dropout(0.1))

model1.add(Dense(512, input_shape = (X_train.shape[1],)))
model1.add(Activation('relu'))
model1.add(Dropout(0.1))


model1.add(Dense(11))
model1.add(Activation('softmax'))

model1.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

import keras
from keras.callbacks import EarlyStopping

# early stopping callback
# This callback will stop the training when there is no improvement in
# the validation loss for 10 consecutive epochs.
es = keras.callbacks.EarlyStopping(monitor='val_loss', mode='min',
                                   patience=10,
                                   restore_best_weights=True) # important - otherwise you just return the last weigths...

# now we just update our model fit call
history = model1.fit(X_train,
                     dummy_y_train,
                     callbacks=[es],
                     epochs=8000000, # you can set this to a big number!
                     batch_size=10,
                     shuffle=True,
                     validation_split=0.2,
                     verbose=1)
```

```
model1.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 512)               6144

 activation_4 (Activation)   (None, 512)               0

 dropout_3 (Dropout)         (None, 512)               0

 dense_5 (Dense)             (None, 512)               262656

 activation_5 (Activation)   (None, 512)               0

 dropout_4 (Dropout)         (None, 512)               0

 dense_6 (Dense)             (None, 512)               262656

 activation_6 (Activation)   (None, 512)               0

 dropout_5 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 11)                5643

 activation_7 (Activation)   (None, 11)                0

=================================================================
Total params: 537,099
Trainable params: 537,099
Non-trainable params: 0
_________________________________________________________________
```

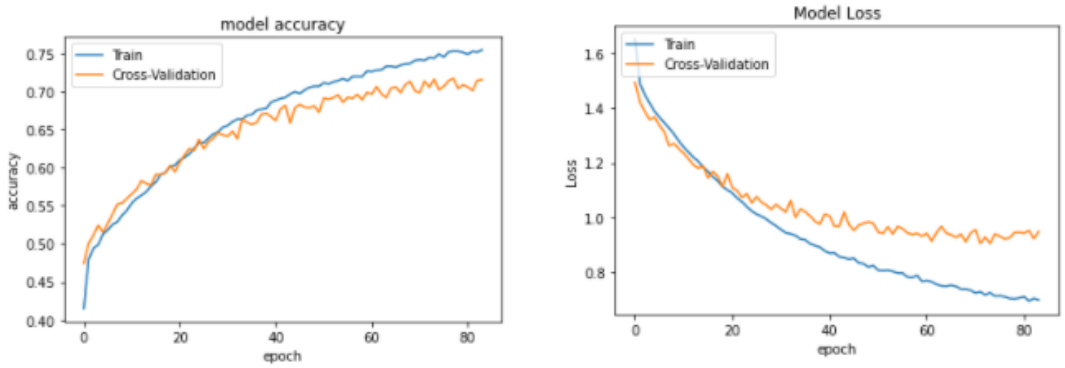
Step C: Model Evaluation

```
: from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

preds = model1.predict(X_test) # see how the model did!
print(classification_report(dummy_y_test.argmax(axis=1), preds.argmax(axis=1)))
```

```
              precision    recall  f1-score   support

           0       0.67      0.80      0.73       760
           1       0.69      0.67      0.68       760
           2       0.64      0.63      0.64       760
           3       0.88      0.95      0.91       760
           4       0.76      0.93      0.83       760
           5       0.75      0.86      0.80       760
           6       0.59      0.38      0.46       760
           7       0.93      0.97      0.95       760
           8       0.73      0.86      0.79       760
           9       0.57      0.49      0.53       760
          10       0.46      0.31      0.37       760

    accuracy                           0.71      8360
   macro avg       0.70      0.71      0.70      8360
weighted avg       0.70      0.71      0.70      8360
```

```
: accuracy = model1.evaluate(X_test, dummy_y_test, verbose=2)
print("Accuracy:",accuracy[1]*100)
```

```
262/262 - 0s - loss: 0.9021 - accuracy: 0.7145 - 158ms/epoch - 603us/step
Accuracy: 71.44736647605896
```



I then printed the classification report and realised that the performance is quite good with precision, accuracy, recall and f1-score at 0.70~. Also, looks like the model accuracy and loss for training and testing performance is quite close which is a good sign of the model performing well.

Note: Based on my research, neural network models take millions of data to train. Although the model performance is good, it may be better suited for applications where the dataset is sufficiently large. In this project, it is more exploratory in nature and to gain a better understanding of neural network models.

Conclusion: Highest precision of Tuned Deep Neural Network Model is 70%

6 Comparison of models

Now, let's compared to see which fine-tuned model performed the best. I have picked the best performing model of each algorithm.

Fine- Tuned model of algorithm	Model Information	Accuracy	Precision	Recall	F1-score
Decision Tree	DecisionTreeClassifier(criterion='entropy', max_depth=22, random_state=42)	0.65	0.64	0.65	0.64
K-Neighbors Classifier	KNeighborsClassifier(leaf_size=4, n_neighbors=1, p=1)	0.79	0.79	0.79	0.79
Random Forest Classifier	RandomForestClassifier(max_features='log2', n_estimators=1000)	0.80	0.80	0.80	0.80

XGBoost Classifier	XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1, colsample_bynode=1, colsample_bytree=0.7, enable_categorical=False, gamma=0.0, gpu_id=-1, importance_type=None, interaction_constraints='', learning_rate=0.2, max_delta_step=0, max_depth=15, min_child_weight=1, missing=nan, monotone_constraints='()', n_estimators=100, n_jobs=16, num_parallel_tree=1, objective='multi:softprob', predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=None, subsample=1, tree_method='exact', validate_parameters=1, verbosity=None)	**0.81**	**0.81**	**0.81**	**0.81**
Gradient Boosting Classifier	GradientBoostingClassifier(learning_rate= 0.001, max_depth=9, n_estimators=1000, subsample=0.5)	0.59	0.58	0.59	0.57
Multinomial Naive Bayes	MultinomialNB()	0.41	0.38	0.41	0.38
SGD Classifier	SGDClassifier()	0.42	0.44	0.42	0.36
One-Vs-Rest Classifier	OneVsRestClassifier(estimator=LinearSVC(random_state=0))	0.47	0.42	0.42	0.36

Neural Network	(Refer to model summary)
	`model1.summary()`
	`Model: "sequential_1"`
	`Layer (type) Output Shape Param #`
	`dense_4 (Dense) (None, 512) 6144`
	`activation_4 (Activation) (None, 512) 0`
	`dropout_3 (Dropout) (None, 512) 0`
	`dense_5 (Dense) (None, 512) 262656`
	`activation_5 (Activation) (None, 512) 0`
	`dropout_4 (Dropout) (None, 512) 0`
	`dense_6 (Dense) (None, 512) 262656`
	`activation_6 (Activation) (None, 512) 0`
	`dropout_5 (Dropout) (None, 512) 0`
	`dense_7 (Dense) (None, 11) 5643`
	`activation_7 (Activation) (None, 11) 0`
	`Total params: 537,099`
	`Trainable params: 537,099`
	`Non-trainable params: 0`
		0.70	0.70	0.71	0.70

Conclusion: The best model to be used is XGBoost Classifier with the highest accuracy of 81%, precision of 81%, recall 0f 81% and f1-score of 81%.

7 Conclusion

For this report, I have gone through the whole data pipeline from data understanding, preparation, exploratory data analysis, modelling, evaluation, prediction and finally comparison of models. To optimize the performance of the model, I have tried many methodologies and switched the order of the data preparations and modelling preparation steps. The insights derived from a wide variety of models also contributed greatly to the overall objective, which is to classify songs by music characteristics. It is evident that different models with different parameters can indeed affect model performance. Additionally, the more models tested, the higher probability of finding a good model for our predictions. I hope this report will provide great value to you and assist you to find the best model that will derive the most accurate and precise predictions.