BSD3533 DATA MINING

GROUP PROJECT 2023/2024

TITLE: MUSIC & MENTAL HEALTH SURVEY RESULTS

GROUP C

STUDENT NAME	STUDENT ID	SECTION	PICTURE
ELAINE TOK CHIA WEN	SD21001	02G	
LEE ZHI LIN	SD21022		
LIM KA QUAN	SD21033		
TEAN JIN HE	SD21063		

TABLE OF CONTENTS

1.0		Executive Summary	3
	1.1	Description of Project	3
	1.2	Problem to be Solved	4
	1.3	Description of Dataset	4
2.0		Summary of Project Context and Objectives	8
	2.1	Summary of Project Context	8
	2.2	Objectives	8
3.0		Methodology	9
4.0		Results and Discussion	11
5.0		Conclusion	25
6.0		References	26

1.0 Executive Summary

1.1 Description of Project

The project investigates the relationship between people's self-reported mental health and music preferences by conducting a thorough analysis of the MxMH dataset. The project explores a variety of genres within the dataset with the fundamental goal of providing insightful analysis to the field of music therapy, which is widely acknowledged as an evidence-based practice for enhancing mental health. Key information is described in great detail, including age, favorite genre, favorite streaming service, number of hours per day spent listening to music, and mental health indicators like OCD, depression, anxiety, and insomnia.

In order to improve people's mental health by providing them with a more sophisticated understanding of their musical flavors, the project places a strong emphasis on analytical goals. Its goal is to find patterns or correlations within the dataset. The project's contribution to evidence-based practices in music therapy is highlighted by the findings, which provide a spotlight on the complex relationship between musical preferences and mental well-being. It might provide fascinating new light on the fundamental relationship between music and the mind.

The project's commitment to improving people's mental health is emphasized by its overarching goal of advancing evidence-based practices in music therapy through data-driven insights. Beyond the field of music therapy, the project hopes to advance knowledge about the relationship between music and mental health. By placing itself within an extensive analytical framework that makes use of the dataset for pattern and correlation identification, it hopes to expand its findings and provide insights into the complex ways that music affects the mind. Through informed music therapy practices, the project's potential overall impact is intended to improve people's mental health and underscore the importance of understanding the role that music plays in mental health.

1.2 Problem to be Solved

The project aims to address the following problem:

- Explore correlations between individuals' music preferences and self-reported mental health indicators.
- Investigate how different musical elements, such as genres, streaming services, and habits, relate to mental health outcomes.
- Advancing evidence-based practices by understanding connections between music and mental health and contributing valuable information to the field of Music Therapy by uncovering specific influences of musical elements on mental well-being.

1.3 Description of Dataset

The "Music & Mental Health Survey Results" is a dataset that can be found on the Kaggle platform (https://www.kaggle.com/datasets/catherinerasgaitis/mxmh-survey-results). This dataset is provided by Catherine Rasgaitis and contains information on various attributes of the music on mental health surveys. The dataset is in CSV format and contains 33 columns and 732 rows. The data type of the dataset contains strings, numerical and float. This dataset can be used for frequent pattern mining, a data mining technique used to identify patterns in data that occur frequently. Predictive modeling can be used with this dataset to investigate trends and correlations regarding how respondents' mental health conditions are affected by music. The survey data can be analyzed using predictive modeling methods, which include algorithms like regression, decision trees, and neural networks. This strategy seeks to forecast the impact of music on mental health, offering insightful information about whether or not respondents' well-being is positively or negatively impacted by music. By pointing out trends and patterns in how music may affect people's psychological states, the analysis can advance knowledge of the connection between musical preferences and mental health outcomes. Below are the short explanation of each attributes:

Variables	Data Type	Definition
Timestamp	String	Date and time when form was submitted.
Age	Float	Respondent’s age.
Primary streaming service	String	Respondent's primary streaming service.
Hours per day	Float	Number of hours the respondent listens to music per day.
While working	String	Does the respondent listen to music while studying/working?
Instrumentalist	String	Does the respondent play an instrument regularly?
Composer	String	Does the respondent compose music?
Fav genre	String	Respondent's favorite or top genre.
Exploratory	String	Does the respondent actively explore new artists/genres?
Foreign languages	String	Does the respondent regularly listen to music with lyrics in a language they are not fluent in?
BPM	Float	Beats per minute of favorite genre.
Frequency [Classical]	String	How frequently the respondent listens to classical music.
Frequency [Country]	String	How frequently the respondent listens to country music.

Frequency [EDM]	String	How frequently the respondent listens to EDM music.
Frequency [Folk]	String	How frequently the respondent listens to folk music.
Frequency [Gospel]	String	How frequently the respondent listens to gospel music.
Frequency [Hip hop]	String	How frequently the respondent listens to hip hop music.
Frequency [Jazz]	String	How frequently the respondent listens to jazz music.
Frequency [K pop]	String	How frequently the respondent listens to k pop music.
Frequency [Latin]	String	How frequently the respondent listens to latin music.
Frequency [Lofi]	String	How frequently the respondent listens to lofi music.
Frequency [Metal]	String	How frequently the respondent listens to metal music.
Frequency [Pop]	String	How frequently the respondent listens to pop music.
Frequency [R&B]	String	How frequently the respondent listens to R&B music.
Frequency [Rap]	String	How frequently the respondent listens to rap music.

Frequency [Rock]	String	How frequently the respondent listens to rock music.
Frequency [Video game music]	String	How frequently the respondent listens to video game music.
Anxiety	Float	Self-reported anxiety, on a scale of 0-10.
Depression	Float	Self-reported depression, on a scale of 0-10.
Insomnia	Float	Self-reported insomnia, on a scale of 0-10.
OCD	Float	Self-reported OCD, on a scale of 0-10.
Music effects	String	Does music improve/worsen respondent's mental health conditions?
Permissions	String	Permissions to publicize data.

2.0 Summary of Project Context and Objectives

2.1 Summary of Project Context

As mental health becomes increasingly crucial, this project aims to create an innovative graphical user interface that explores the complex relationship between people's musical preferences and self-reported mental health indicators. The initiative's driving principle is to provide people with unique perspectives into their musical preferences, encouraging proactive mental health management via creative and user-friendly ways.

This project serves a wide range of stakeholders, including mental health professionals, organizations interested in multifaceted mental health, and individuals looking for individualized mental health support. The main goal is to explore the intricate relationship between music and mental health through utilizing graphical user interface. The interface incorporates mood tracking and guided mindfulness exercises to predict the effect of music on mental well-being based on user inputs about their preferred music. Issues with self-reported data are resolved through seamless integration with professional mental health services.

The goals are in line with learning more about the potential relationships between various music genres, streaming services, and music-related behaviors and mental health consequences. In order to shed light on how particular musical elements might affect mental well-being, the project aims to make a significant contribution to the field of music therapy by identifying potential correlations between individuals' music preferences and self-reported mental health indicators.

2.2 Objectives

The objectives of this project are:

- To identify potential correlations between individuals' music preferences and self-reported mental health indicators.
- To gain insights into how different music genres, streaming services, and music-related habits may relate to mental health outcomes.
- To identify valuable information to the field of Music Therapy by understanding how specific musical elements might influence mental well-being.

3.0 Methodology

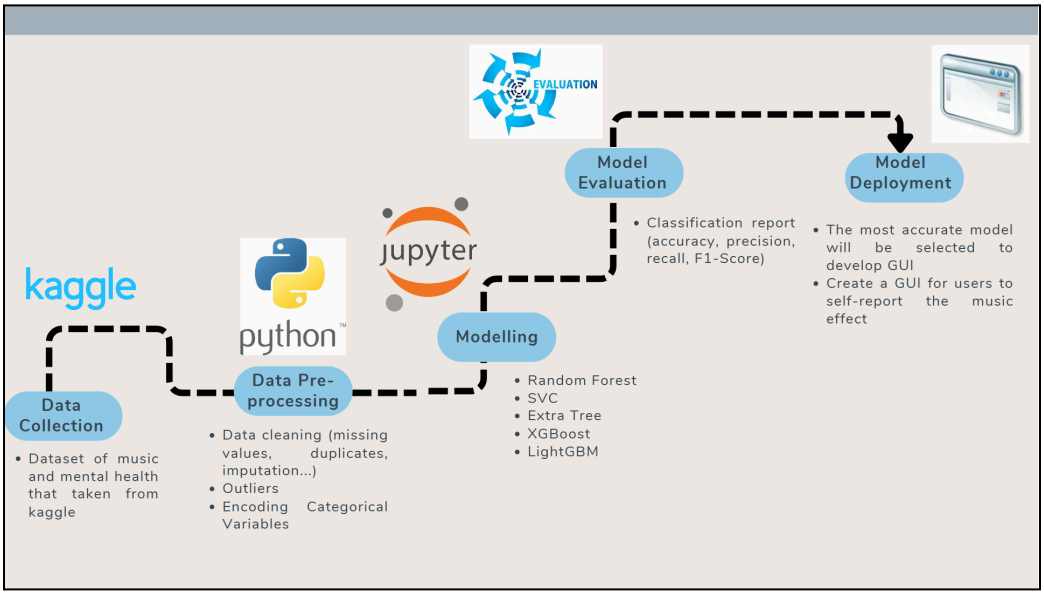


Figure 3.1: Pipeline of the project

First and foremost, the dataset is being extracted and downloaded from Kaggle and imported into the Jupyter. The dataset is about the survey results on music preference and self-reported mental health. The detailed description about the data and dataset are being explained in section 1.3 mentioned above. This project will utilize the dataset on the predictive model to predict the music effect towards each user.

Moreover, the data is being pre-processed before it can be utilized in the predictive model. The first step is to check the data types, null values, duplicated values and also the outliers. Since our output for dependent variables, music effect is multiclasses then we also need to check the balancing of class count.

The null values will be handled by applying imputation with mean and mode. Then, handling extreme values and capping the outliers is necessary due to the extreme values that will affect model assumption and performance of predictive models. The imbalance of class count needs to be dealt with by using the SMOTE balancing technique. This is because it generates synthetic samples for the minority class to improve pattern recognition and balance the dataset.

Additionally, drop unrelated columns to ensure the accuracy of predicting. Label encode and mapping are also being utilized to convert the categorical data into numeric. This is to ensure the user is easy to fill in during the self reported graphical user interface.

Apart from that, data partitioning where the data needs to split into train data and test data. In our case, we split 80% as a training set and 20% as a testing set. Scaling and accuracy checking are performed before fitting into the model. Next, the highest accuracy of the model will be chosen and combined to create a graphical user interface which helps the users to predict the music effect easily.

In conclusion, the extra trees model is the most accurate model in predicting the music effect and being used as the predictive models of the GUI. Users may enter age, hours per day, frequency of each genre and the level of mental health issues to predict the music effect on the mental health problems.

4.0 Results and Discussion

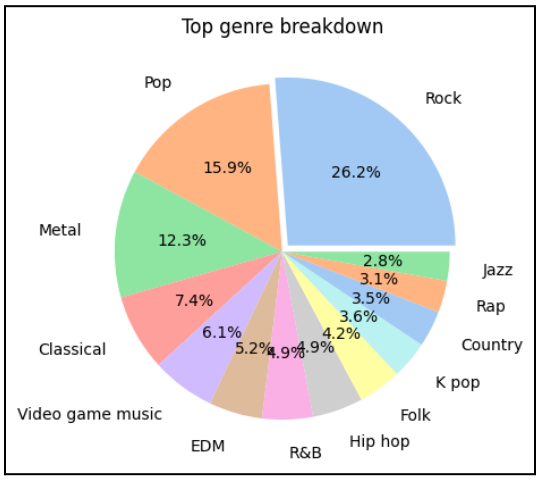


Figure 4.1: Top Genre Breakdown

The pie chart shows the breakdown of favorite genres among the respondents. Rock is the most popular genre with 26.2% of the votes, followed by pop with 15.9% of the votes and metal with 12.3% of the votes. The other genres are less popular, but still have a significant number of fans.

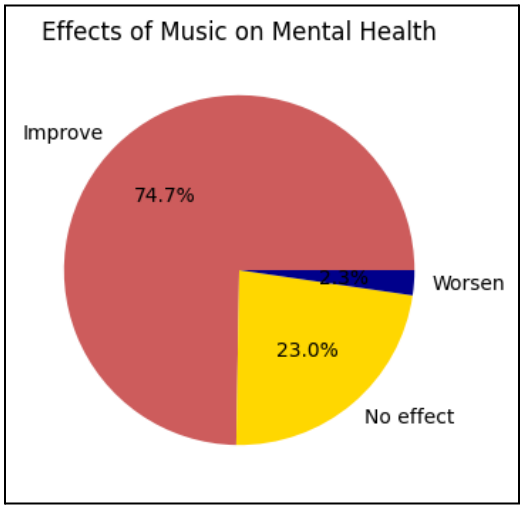


Figure 4.2: Effects of Music on Mental Health

The pie chart shows the effects of music on mental health. The survey was conducted among 736 people. The results show that almost 75% of the respondents believe that music improves their mental health, 23% of respondents believe that it has no effect, and 2% believe that it worsens their mental health.

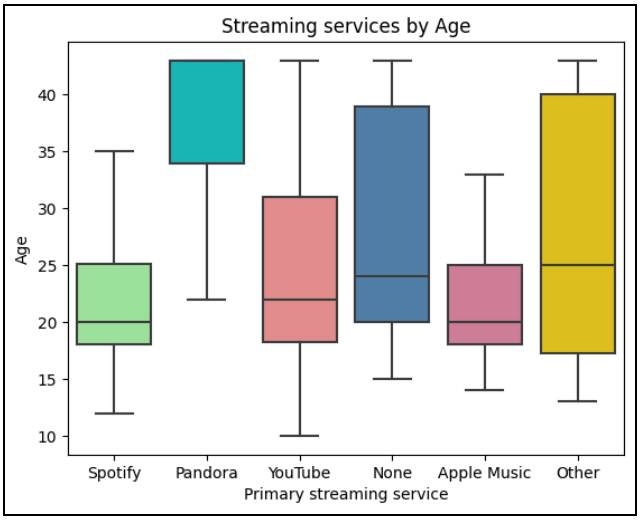


Figure 4.3: Streaming Services by Age

The boxplot shows the age distribution of users among different streaming services. Based on our observation, every streaming service median level is similar, which is between 20 and 25 years old except Pandora streaming service. Pandora users are majority older than users of other streaming services.

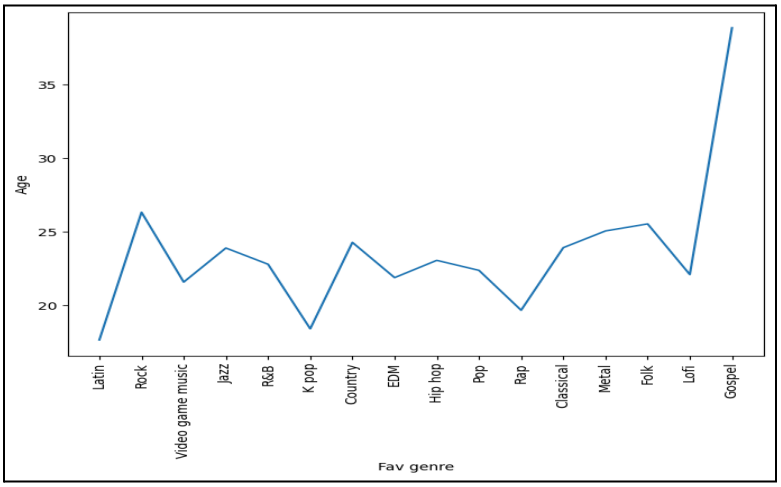


Figure 4.4: Age by Favourite Genre

The graph shows the relationship between age and favourite genre of music. As we can observe from the graph, it shows that people who like Gospel music are typically older than people who like other music genres. Most of the youngsters show a preference for Latin, K-pop and Rap.

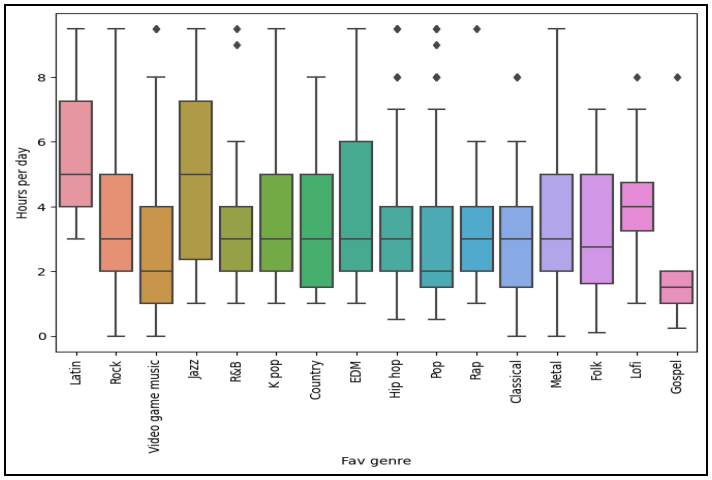


Figure 4.5: Hours per Day by Favourite Genre

The boxplot shows the distribution of daily music listening hours across different favorite music genres. Examining the plot, it becomes evident that individuals who favor Latin and Jazz music spend the most time on their music, reflected by a median of 7 hours per day. In contrast, those who prefer Gospel music tend to spend the least time listening, with a median around 2 hours.

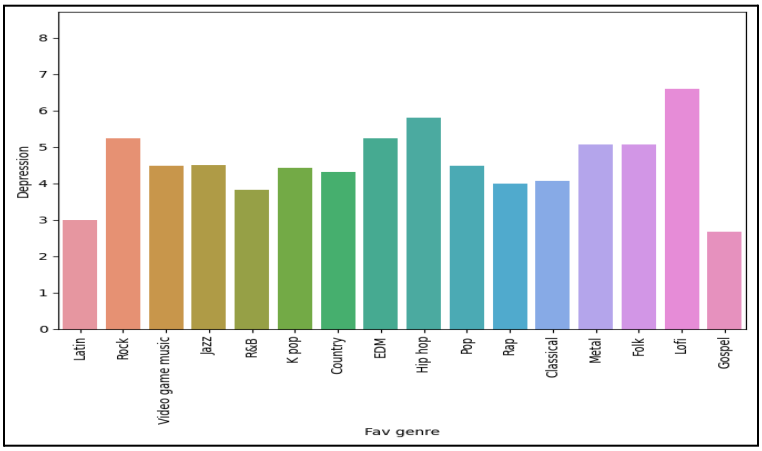


Figure 4.6: Depression by Favourite Genre

The graph shows the relationship between favorite genres of music and depression. The x-axis is the favorite genre of music, and the y-axis is the depression score. The data shows that people who listen to Lofi music have the highest depression scores, while people who listen to Latin music have the lowest depression scores. This may be because lofi music is often associated with feelings of sadness and isolation, while Latin music is often associated with feelings of happiness and energy.

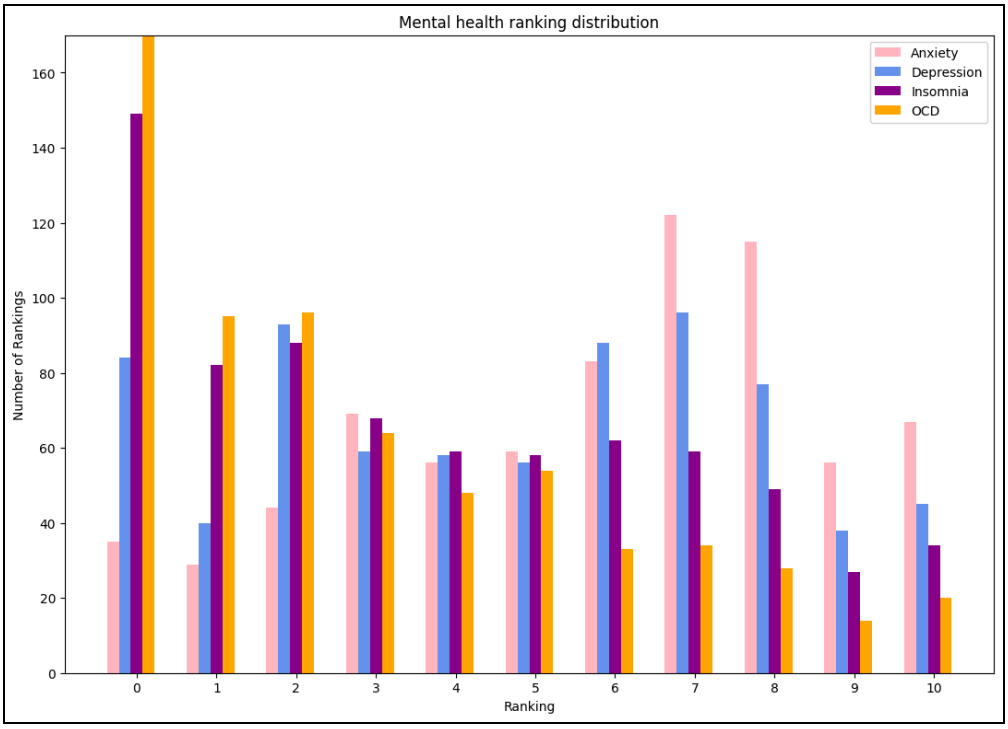


Figure 4.7: Mental Health Ranking Distribution

The bar chart provides a visual representation of how individuals ranked various mental health disorders on a scale from 0 to 10. On the x-axis, we have the ranking levels, ranging from 0 (indicating minimal impact) to 10 (reflecting a significant challenge). The y-axis displays the number of individuals corresponding to each ranking level. Notably, at rank 0, the chart indicates that a substantial majority, surpassing 160 individuals, reported no issues with Obsessive-Compulsive Disorder (OCD). Conversely, at rank 10, only a minority of approximately 20 individuals faced OCD challenges.

Contrastingly, when it comes to anxiety, the chart reveals a different pattern. Around 60 people reported facing anxiety challenges at rank 10, suggesting a considerable impact when listening to music. On the other hand, at rank 0, approximately 30 individuals indicated the absence of anxiety-related concerns. This shows that different mental health issues are affected differently by music.

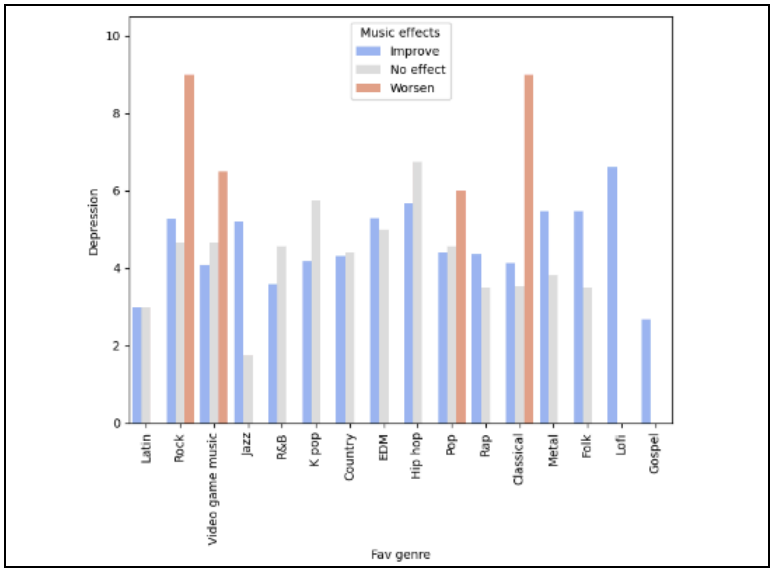


Figure 4.8: Music Effects by Depression and Favourite Genre

According to the graph, people who listen to rock, video game music, pop and classical are most likely to report that music worsens their depression. Meanwhile, people who listen to Lofi, Metal, Hip Hop and Folk music improve their depression.

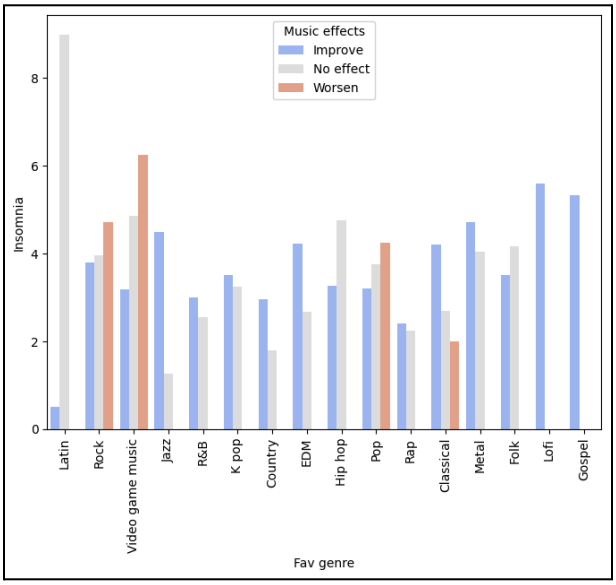


Figure 4.9: Music Effects by Insomnia and Favourite Genre

From the chart, we can see that people who listen to rock, video game music and pop are more likely to have insomnia. While people who listen to Lofi, Gospel, Jazz and Mental music are less likely to have insomnia.

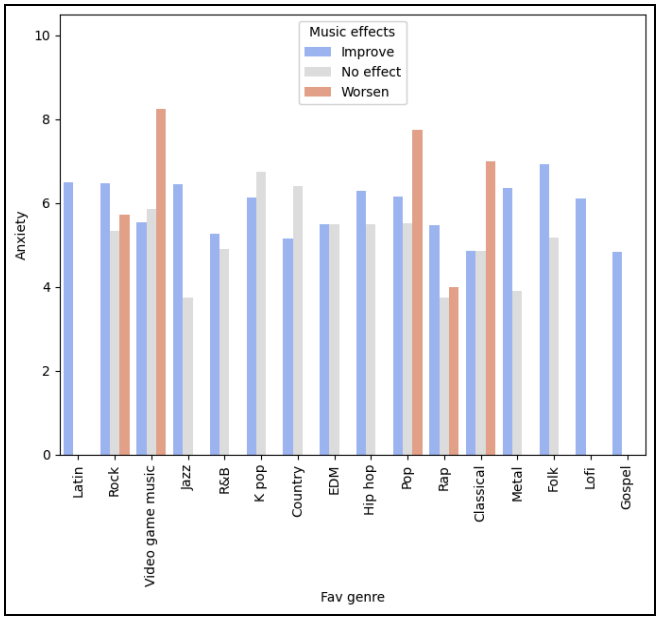


Figure 4.10: Music Effects by Anxiety and Favourite Genre

According to the chart, video game music and pop have a significant effect on worsening people's anxiety. Other genres of music have similar effects to improve people's anxiety.

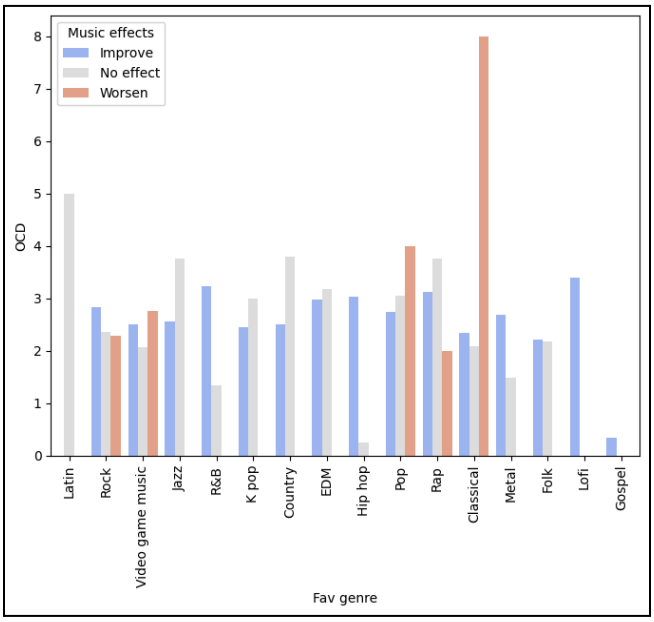


Figure 4.11: Music Effects by OCD and Favourite Genre

The graph reveals that classical music tends to significantly worsen Obsessive-Compulsive Disorder (OCD), followed by pop music. In contrast, other music genres either have a minor impact or no noticeable effect on OCD.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 736 entries, 0 to 735
Data columns (total 33 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Timestamp                       736 non-null    object
 1   Age                             735 non-null    float64
 2   Primary streaming service       735 non-null    object
 3   Hours per day                   736 non-null    float64
 4   While working                   733 non-null    object
 5   Instrumentalist                 732 non-null    object
 6   Composer                        735 non-null    object
 7   Fav genre                       736 non-null    object
 8   Exploratory                     736 non-null    object
 9   Foreign languages               732 non-null    object
 10  BPM                             629 non-null    float64
 11  Frequency [Classical]           736 non-null    object
 12  Frequency [Country]             736 non-null    object
 13  Frequency [EDM]                 736 non-null    object
 14  Frequency [Folk]                736 non-null    object
 15  Frequency [Gospel]              736 non-null    object
 16  Frequency [Hip hop]             736 non-null    object
 17  Frequency [Jazz]                736 non-null    object
 18  Frequency [K pop]               736 non-null    object
 19  Frequency [Latin]               736 non-null    object
 20  Frequency [Lofi]                736 non-null    object
 21  Frequency [Metal]               736 non-null    object
 22  Frequency [Pop]                 736 non-null    object
 23  Frequency [R&B]                 736 non-null    object
 24  Frequency [Rap]                 736 non-null    object
 25  Frequency [Rock]                736 non-null    object
 26  Frequency [Video game music]    736 non-null    object
 27  Anxiety                         736 non-null    float64
 28  Depression                      736 non-null    float64
 29  Insomnia                        736 non-null    float64
 30  OCD                             736 non-null    float64
 31  Music effects                   728 non-null    object
 32  Permissions                     736 non-null    object
dtypes: float64(7), object(26)
```

Figure 4.12: Data information

This project is done by using Jupyter Notebook. By using df.info(), we obtain 736 rows and 33 columns in our dataset which means there are 736 respondents and 33 variables that we can find interesting insight.

During preprocessing, we identified an extreme value in the Beats per minute (BPM) column, denoted as 999999999. Besides that, there are lots of null values in the column and it is quite for people to identify the actual BPM without specific tools. Thus, we decided to drop the column after deep consideration.

Then, we checked the correlations of variables using heatmap. We drop the features like Timestamp and Permissions since they are useless for prediction.

To ensure accurate analysis, we replaced categorical null value with the mode, and numerical null value with median. We also applied boundary capping to address outliers. This meticulous cleaning process laid the groundwork for a smooth and precise analysis.

Subsequently, we examined class imbalance, particularly in the 'music effect' variable, and opted to address it using the SMOTE (Synthetic Minority Over-sampling Technique) to balance the classes.

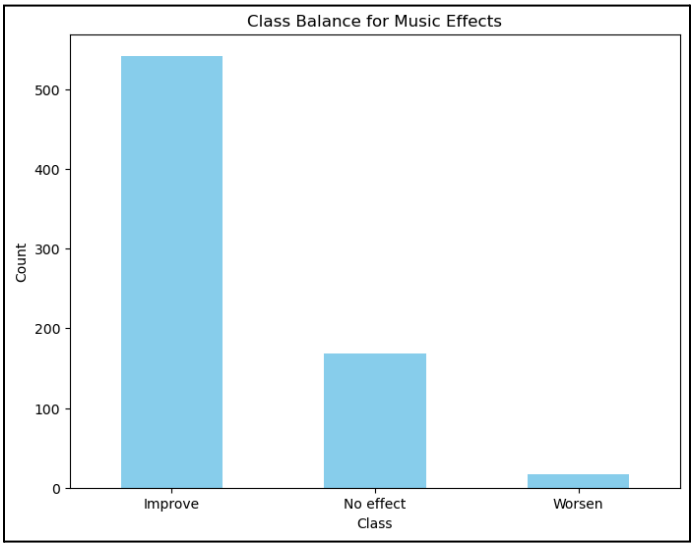


Figure 4.13: Class Distribution

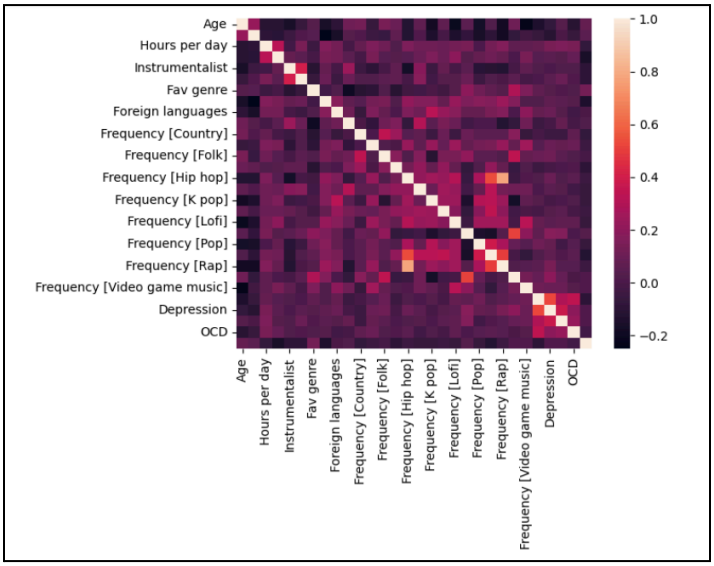


Figure 4.14: Check Correlations using Heatmap

After the data preprocessing, we decided to use the library LazyPredict to produce an outcome to guide our choice of 5 models. Below shows the outcome of LazyPredict:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score
XGBClassifier	0.89	0.89	None	0.89
LGBMClassifier	0.89	0.89	None	0.89
ExtraTreesClassifier	0.88	0.88	None	0.88
RandomForestClassifier	0.87	0.87	None	0.87
SVC	0.87	0.87	None	0.87
QuadraticDiscriminantAnalysis	0.86	0.85	None	0.86
NuSVC	0.85	0.85	None	0.85
BaggingClassifier	0.83	0.83	None	0.83
LabelPropagation	0.83	0.83	None	0.81
LabelSpreading	0.83	0.83	None	0.81
KNeighborsClassifier	0.79	0.79	None	0.77
DecisionTreeClassifier	0.77	0.77	None	0.77
ExtraTreeClassifier	0.77	0.77	None	0.77
LogisticRegression	0.74	0.74	None	0.74
LinearSVC	0.74	0.74	None	0.74
CalibratedClassifierCV	0.74	0.74	None	0.74
AdaBoostClassifier	0.73	0.73	None	0.73
SGDClassifier	0.72	0.71	None	0.72
LinearDiscriminantAnalysis	0.71	0.71	None	0.70
RidgeClassifierCV	0.71	0.71	None	0.70
RidgeClassifier	0.71	0.70	None	0.70
GaussianNB	0.67	0.66	None	0.65
Perceptron	0.65	0.65	None	0.66
PassiveAggressiveClassifier	0.63	0.63	None	0.64
NearestCentroid	0.61	0.61	None	0.61
BernoulliNB	0.60	0.60	None	0.60
DummyClassifier	0.31	0.33	None	0.14

Figure 4.15: Results of LazyPredict

This comprehensive approach ensures the reliability of our subsequent analyses and predictions related to the 'music effect.' Based on the result of LazyPredict, we choose 5 model namely LGBMClassifier, XGBClassifier, ExtraTreesClassifier, RandomForestClassifier and Support Vector Classifier (SVC) and put the data into training to get the model with highest accuracy.

Model Name	Accuracy	Precision	Recall	F1
LightGBM Classifier	0.8970	0.8977	0.8970	0.8969
XGBoost Classifier	0.8949	0.8965	0.8949	0.8945
Extra Trees Classifier	0.8869	0.8893	0.8869	0.8869
Random Forest Classifier	0.8828	0.8864	0.8828	0.8828
SVC	0.8747	0.8794	0.8747	0.8743

Figure 4.16: Table of Accuracy Score

Based on the above result, we observe that the actual accuracy of each model is similar and approximate to 90% thus we decide to develop a Graphical User Interface (GUI) based on Extra Trees classifier because it is a powerful and accurate algorithm that is well-suited for classification tasks. We selected the Extra Trees Classifier as the core of our GUI because it's a powerful tool for understanding the connection between music, mental health symptoms, and mental health conditions. It works by creating a bunch of decision trees and then combining their results to make predictions. This makes it really good at handling complex data, like the kind we're working with in our GUI.

The Extra Trees Classifier is also robust, which means it's not easily fooled by outliers or extreme values in the data. This is important because we want our GUI to be able to make accurate predictions even for people who have unusual music preferences or mental health symptoms. Finally, the Extra Trees Classifier is good at finding the most important features in the data. This helps us understand which music preferences and mental health symptoms are most strongly linked to mental health conditions. This information can be used to develop targeted interventions that can help people improve their mental health.

Overall, the Extra Trees Classifier is a great choice for our GUI because it's accurate, robust, and helps us understand the relationship between music, mental health symptoms, and mental health conditions.

To create a GUI based on the Extra Trees classifier, we will use the Python programming language and the tkinter library. We will create a simple GUI that allows users to select a music

genre and then input their mental health symptoms. To use the GUI, a user will need to input their age, primary streaming service, number of hours they listen to music per day, whether they are an instrumentalist or composer, their favorite genre, and their frequency of listening to various genres of music. The user will also need to input their mental health symptoms, including their levels of anxiety, depression, insomnia, and OCD. Once all of the information has been entered, the user will click the "Predict" button and the GUI will use the Extra Trees classifier to predict the user's mental health condition.

Figure 4.17: Windows of GUI

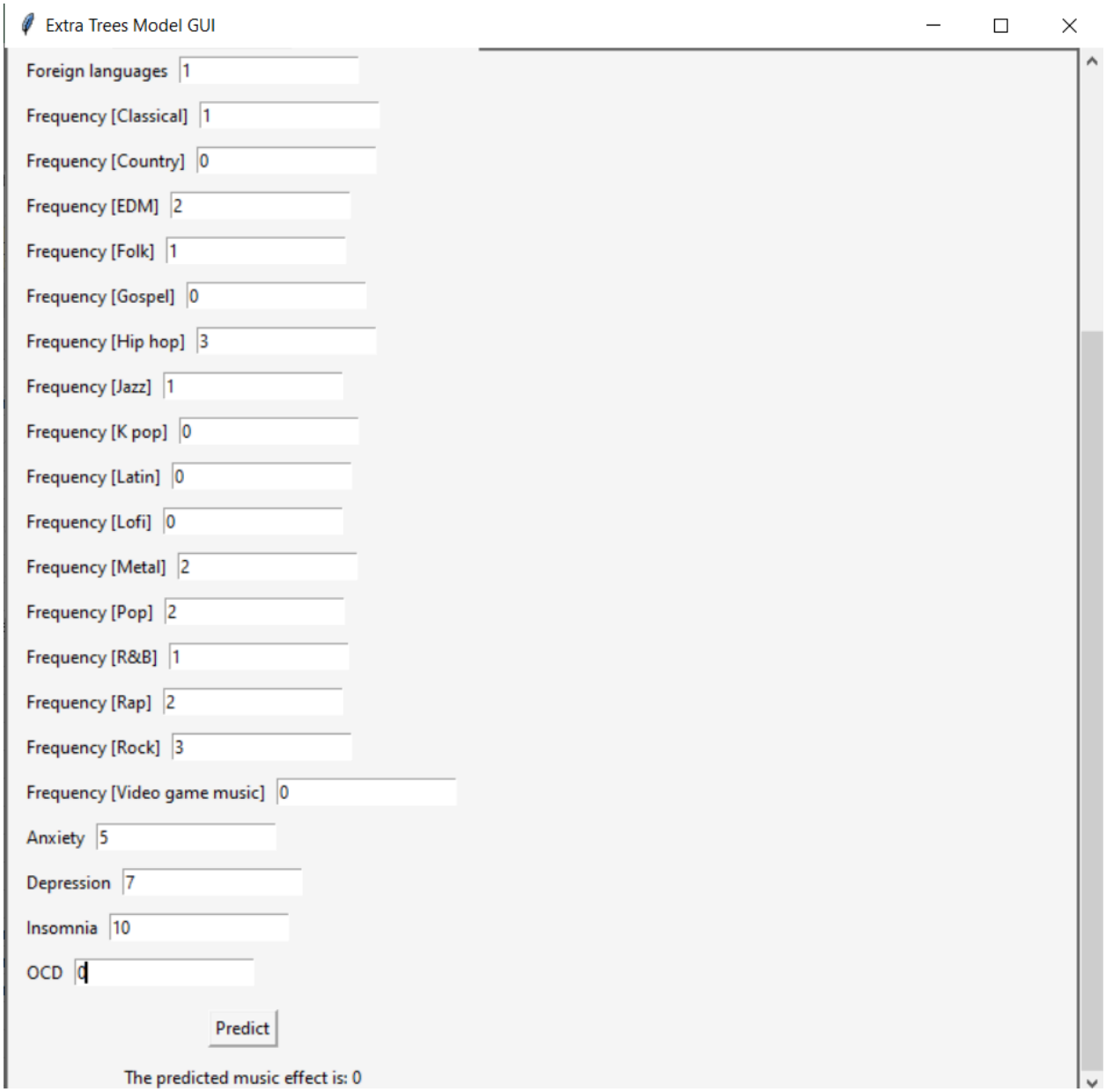


Figure 4.18: Results of Prediction for GUI Predict Improve

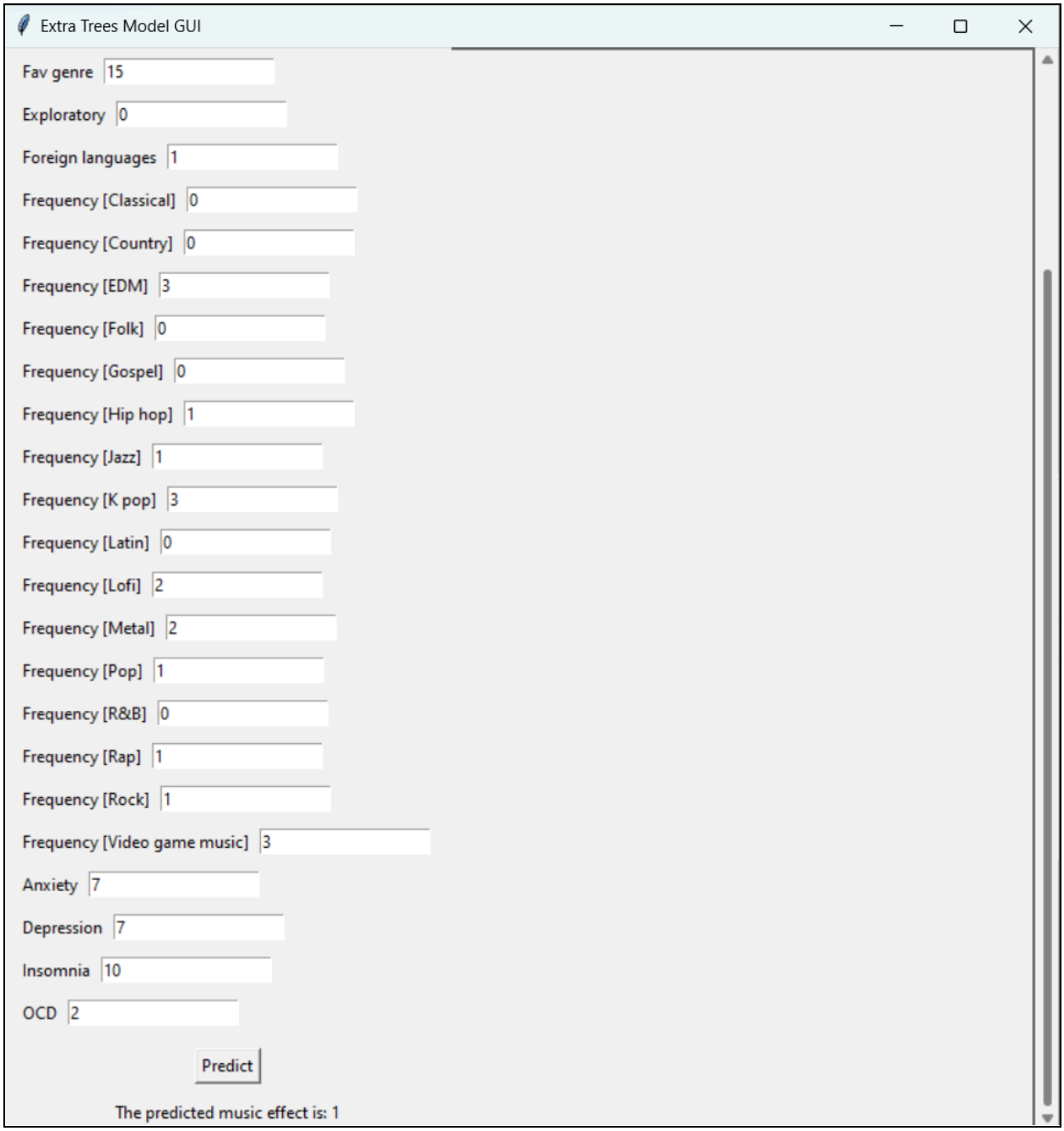


Figure 4.19: Results of Prediction for GUI Predict No Effect

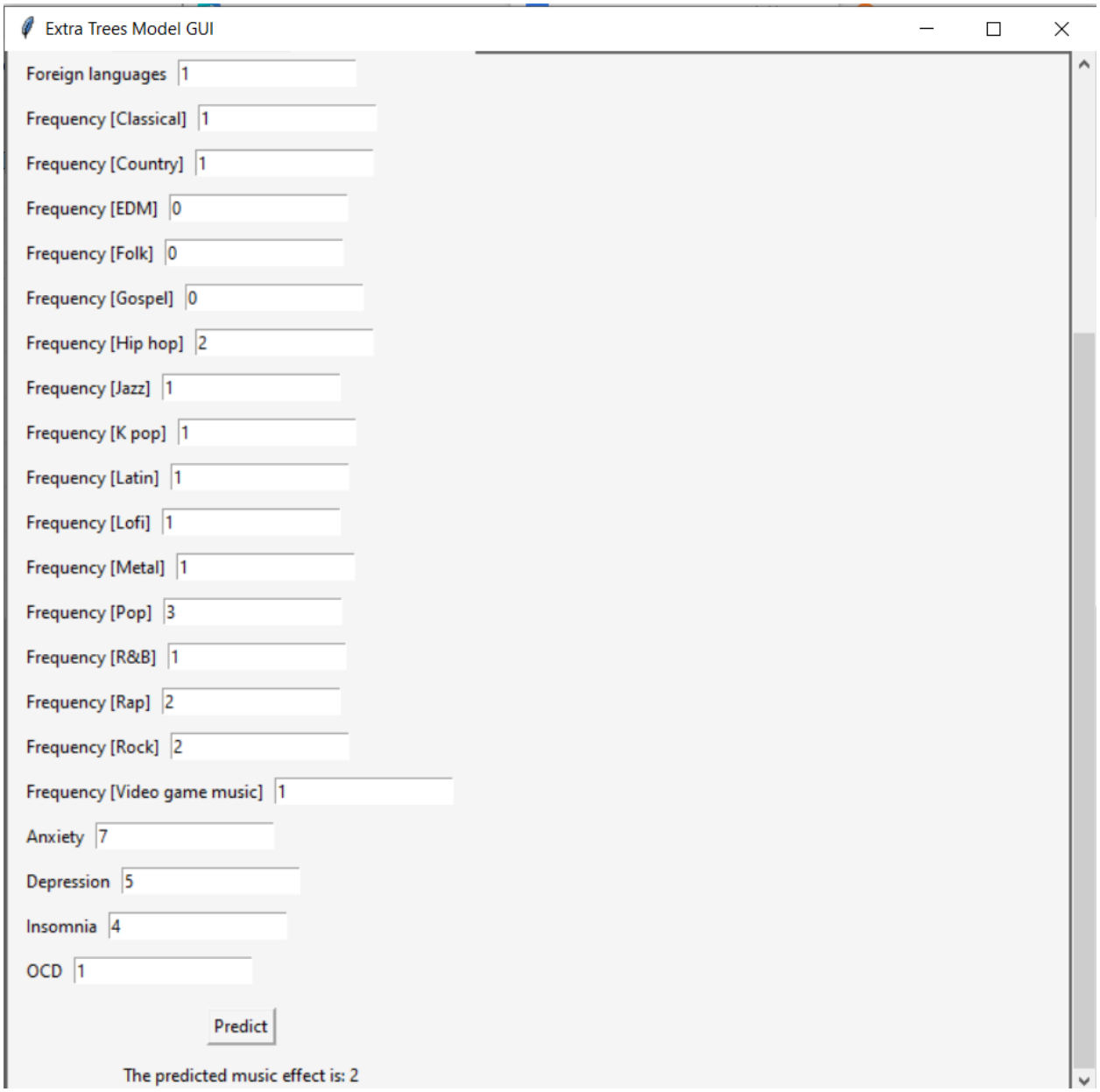


Figure 4.20: Results of Prediction for GUI Predict Worsen

5.0 Conclusion

In conclusion, while our model demonstrates commendable accuracy, there is acknowledgment of the need for enhancements in the user interface of the GUI to optimize user convenience. The project underscores the complex relationship between music and mental health, recognizing the nuanced effects that vary among individuals based on factors such as age, gender, and music genre preferences. It emphasizes the importance of carefully selecting music tailored to individual needs, acknowledging its potential as a tool for both enhancing mental well-being and aiding in the management of mental health conditions.

The project "Music & Mental Health Survey Results" greatly contributes to our collective knowledge of the complex interplay between musical tastes and mental well-being. Featuring an user-friendly interface designed to empower users, mental health practitioners, and researchers, the project offers insightful information for focused interventions that can promote better mental health outcomes.

In addition to its positive effects on individuals, the project indirectly addresses the negative effects of stress on people, which promotes environmental sustainability. It promotes data-driven methods for mental health interventions systemically and uses the GUI to highlight technological innovation in the field. This demonstrates the project's broad benefits across societal, environmental, and systemic dimensions of mental health and well-being. It also supports ongoing research and paves the way for future developments in user-friendly tools within mental health applications.

6.0 References

Chen, W., Edwards, E., & Shurtleff, D. (2022, September). *Music and Health: What You Need To Know*. NCCIH. https://www.nccih.nih.gov/health/music-and-health-what-you-need-to-know

Cherry, K. (2019, December 10). *How listening to music can have psychological benefits*. Verywell Mind. https://www.verywellmind.com/surprising-psychological-benefits-of-music-4126866

Gustavson, D. E., Coleman, P. L., Iversen, J. R., Maes, H. H., Gordon, R. L., & Lense, M. D. (2021). Mental health and music engagement: review, framework, and guidelines for future studies. *Translational Psychiatry*, *11*(1), 1–13. https://doi.org/10.1038/s41398-021-01483-8

MARTIN, G., CLARKE, M., & PEARCE, C. (1993). Adolescent Suicide: Music Preference as an Indicator of Vulnerability. *Journal of the American Academy of Child & Adolescent Psychiatry*, *32*(3), 530–535. https://doi.org/10.1097/00004583-199305000-00007

Dataset Link: https://www.kaggle.com/datasets/catherinerasgaitis/mxmh-survey-results

import necessary library

In [1]:
```
#import libraries
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer,SimpleImputer
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
```

read data

In [2]:
```
#import data
df = pd.read_csv("mxmh_survey_results.csv")
df
```

Out[2]:

	Timestamp	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Explo
0	8/27/2022 19:29:02	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	
1	8/27/2022 19:57:31	63.0	Pandora	1.5	Yes	No	No	Rock	
2	8/27/2022 21:28:18	18.0	Spotify	4.0	No	No	No	Video game music	
3	8/27/2022 21:40:40	61.0	YouTube Music	2.5	Yes	No	Yes	Jazz	
4	8/27/2022 21:54:47	18.0	Spotify	4.0	Yes	No	No	R&B	
...	...	...	...	...	...	...	...	...	
731	10/30/2022 14:37:28	17.0	Spotify	2.0	Yes	Yes	No	Rock	
732	11/1/2022 22:26:42	18.0	Spotify	1.0	Yes	Yes	No	Pop	
733	11/3/2022 23:24:38	19.0	Other streaming service	6.0	Yes	No	Yes	Rap	
734	11/4/2022 17:31:47	19.0	Spotify	5.0	Yes	Yes	No	Classical	
735	11/9/2022 1:55:20	29.0	YouTube Music	2.0	Yes	No	No	Hip hop	

736 rows × 33 columns

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 736 entries, 0 to 735
Data columns (total 33 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Timestamp                     736 non-null    object 
 1   Age                           735 non-null    float64
 2   Primary streaming service     735 non-null    object 
 3   Hours per day                 736 non-null    float64
 4   While working                 733 non-null    object 
 5   Instrumentalist               732 non-null    object 
 6   Composer                      735 non-null    object 
 7   Fav genre                     736 non-null    object 
 8   Exploratory                   736 non-null    object 
 9   Foreign languages             732 non-null    object 
 10  BPM                           629 non-null    float64
 11  Frequency [Classical]         736 non-null    object 
 12  Frequency [Country]           736 non-null    object 
 13  Frequency [EDM]               736 non-null    object 
 14  Frequency [Folk]              736 non-null    object 
 15  Frequency [Gospel]            736 non-null    object 
 16  Frequency [Hip hop]           736 non-null    object 
 17  Frequency [Jazz]              736 non-null    object 
 18  Frequency [K pop]             736 non-null    object 
 19  Frequency [Latin]             736 non-null    object 
 20  Frequency [Lofi]              736 non-null    object 
 21  Frequency [Metal]             736 non-null    object 
 22  Frequency [Pop]               736 non-null    object 
 23  Frequency [R&B]               736 non-null    object 
 24  Frequency [Rap]               736 non-null    object 
 25  Frequency [Rock]              736 non-null    object 
 26  Frequency [Video game music]  736 non-null    object 
 27  Anxiety                       736 non-null    float64
 28  Depression                    736 non-null    float64
 29  Insomnia                      736 non-null    float64
 30  OCD                           736 non-null    float64
 31  Music effects                 728 non-null    object 
 32  Permissions                   736 non-null    object 
dtypes: float64(7), object(26)
memory usage: 189.9+ KB
```

data preprocessing

checking duplicated

In [4]:
```
df.duplicated().sum()
```

Out[4]:
```
0
```

checking missing value

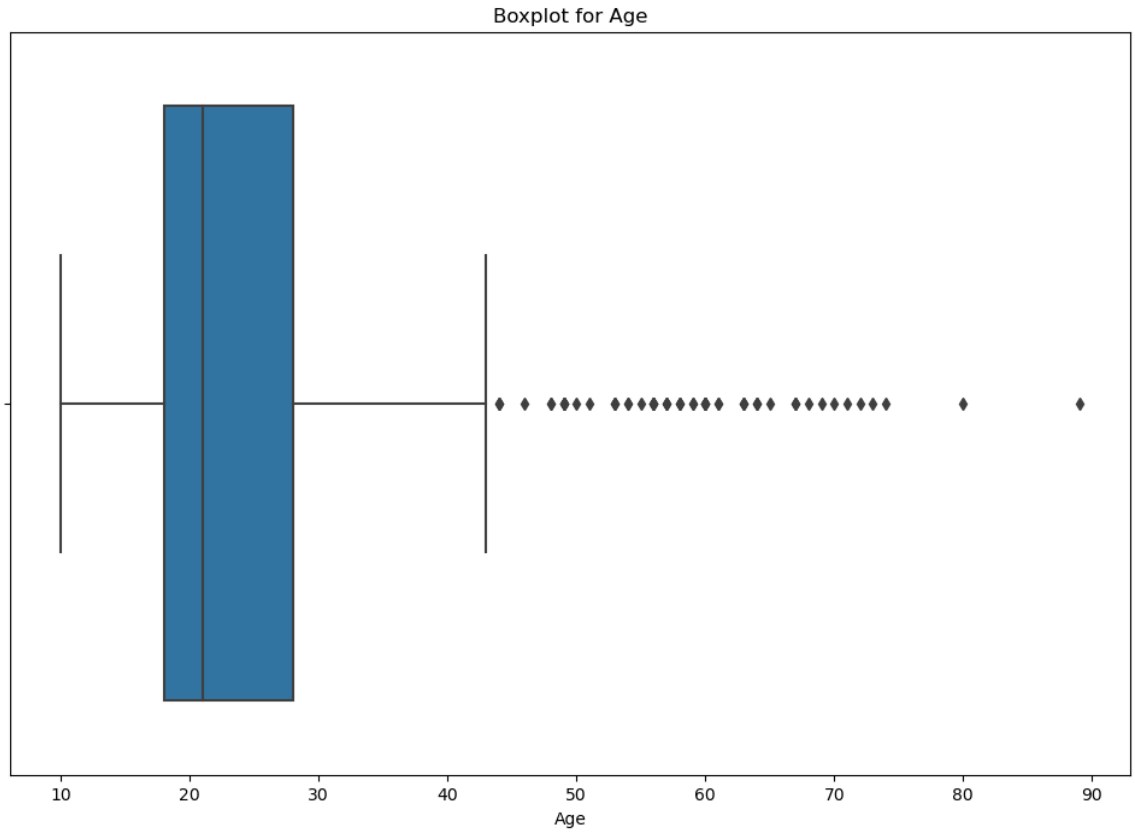
In [5]:
```
df.isnull().sum()
```

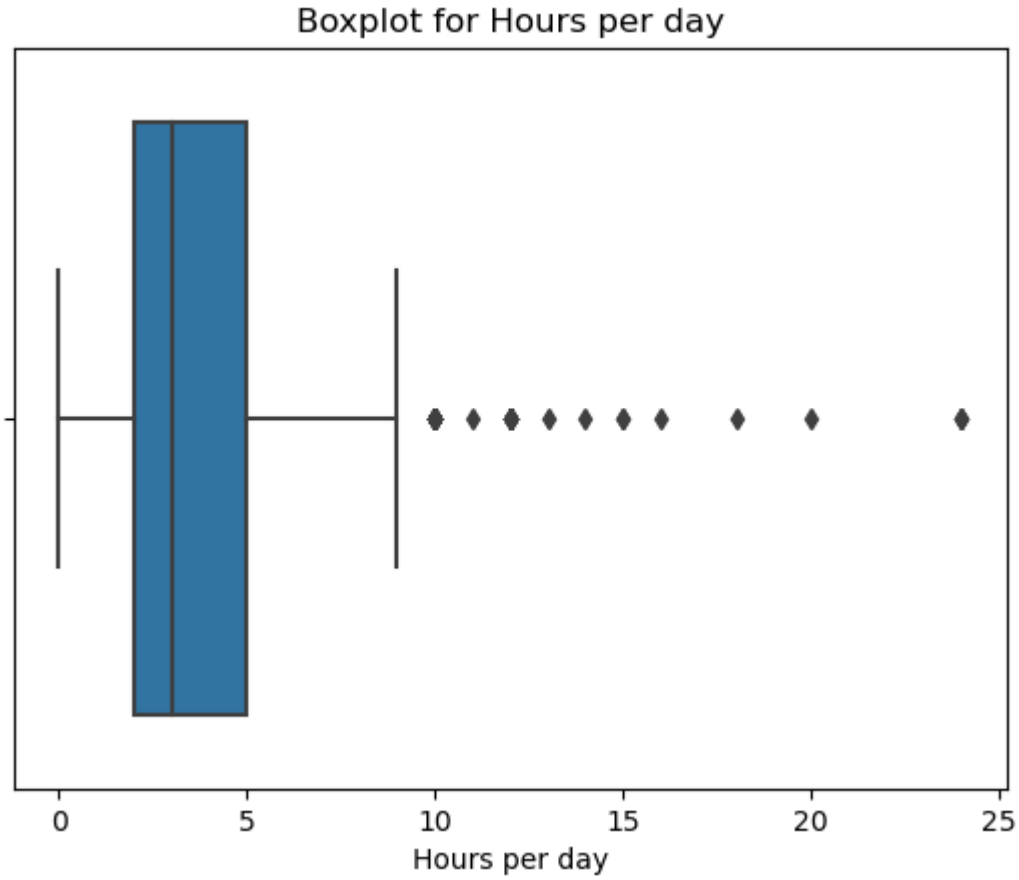
Out[5]:
```
Timestamp                         0
Age                               1
Primary streaming service         1
Hours per day                     0
While working                     3
Instrumentalist                   4
Composer                          1
Fav genre                         0
Exploratory                       0
Foreign languages                 4
BPM                             107
Frequency [Classical]             0
Frequency [Country]               0
Frequency [EDM]                   0
Frequency [Folk]                  0
Frequency [Gospel]                0
Frequency [Hip hop]               0
Frequency [Jazz]                  0
Frequency [K pop]                 0
Frequency [Latin]                 0
Frequency [Lofi]                  0
Frequency [Metal]                 0
Frequency [Pop]                   0
Frequency [R&B]                   0
Frequency [Rap]                   0
Frequency [Rock]                  0
Frequency [Video game music]      0
Anxiety                           0
Depression                        0
Insomnia                          0
OCD                               0
Music effects                     8
Permissions                       0
dtype: int64
```

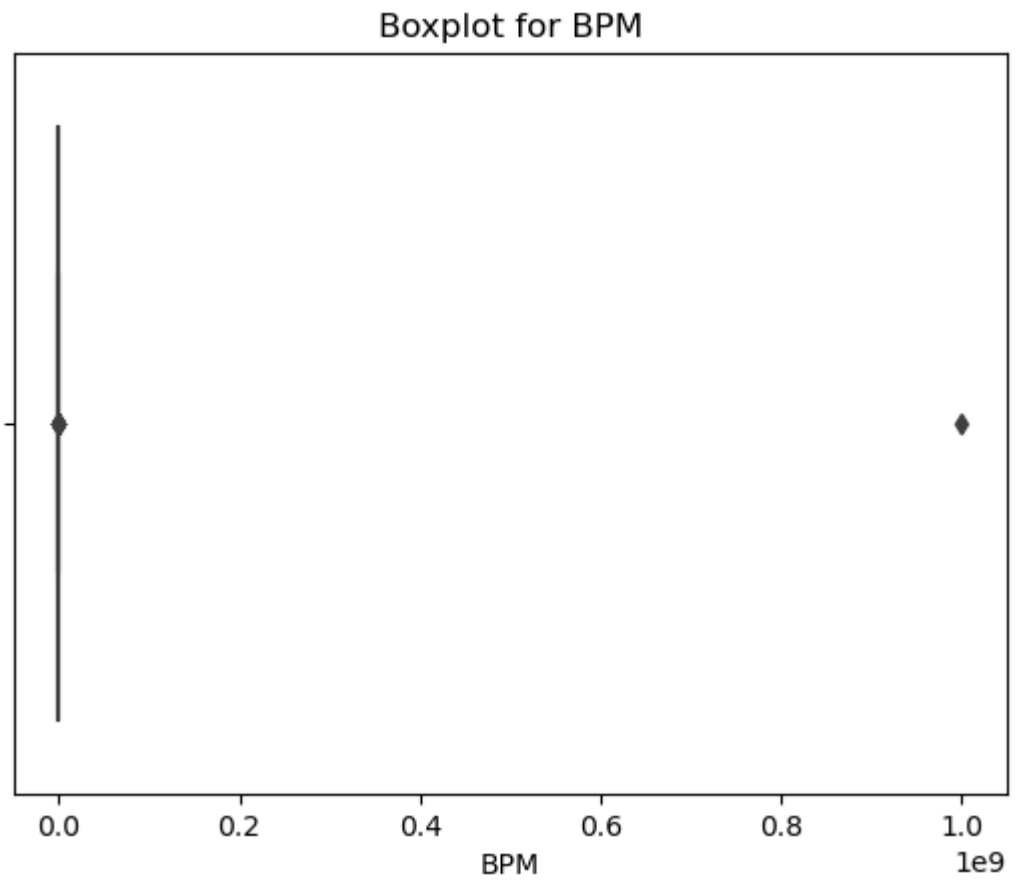
There are 8 attributes with missing values which are age, primary streaming service, while working, instrumentalist, composer, foreign language, BPM and Music effect. Since the missing value of bpm is too much (more than 1/7 observations) hence we decide to drop it.

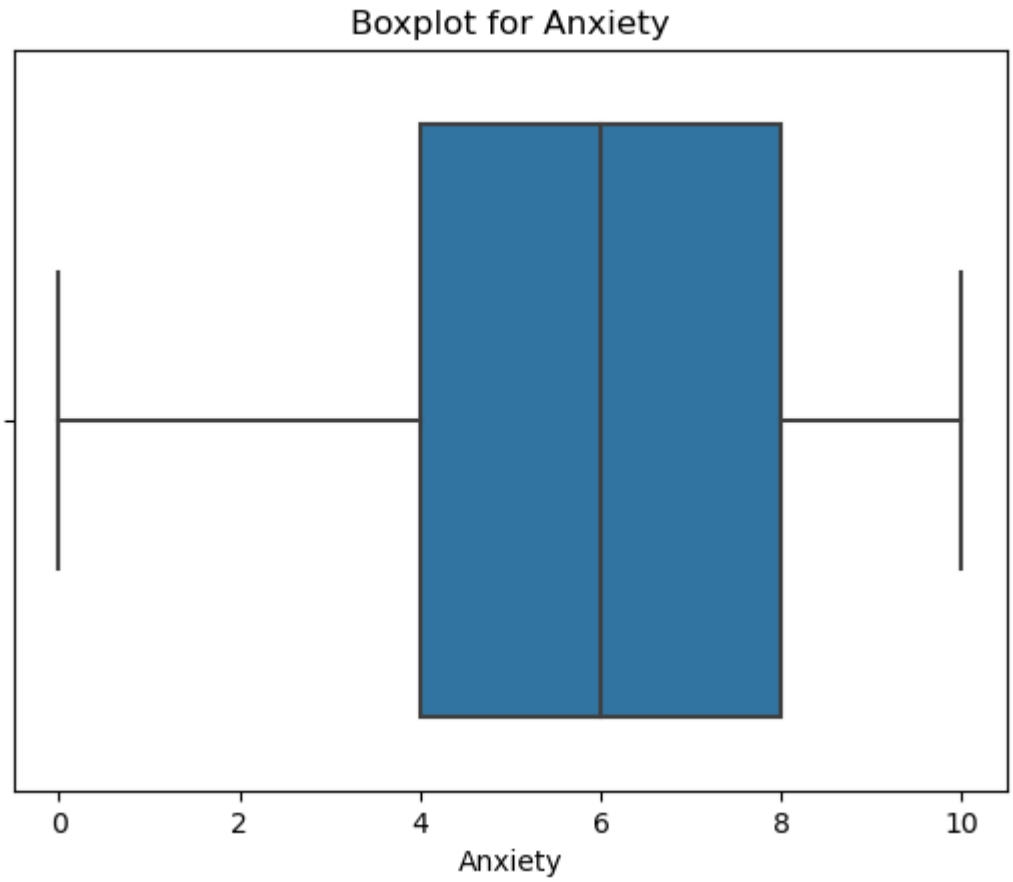
checking for outliers

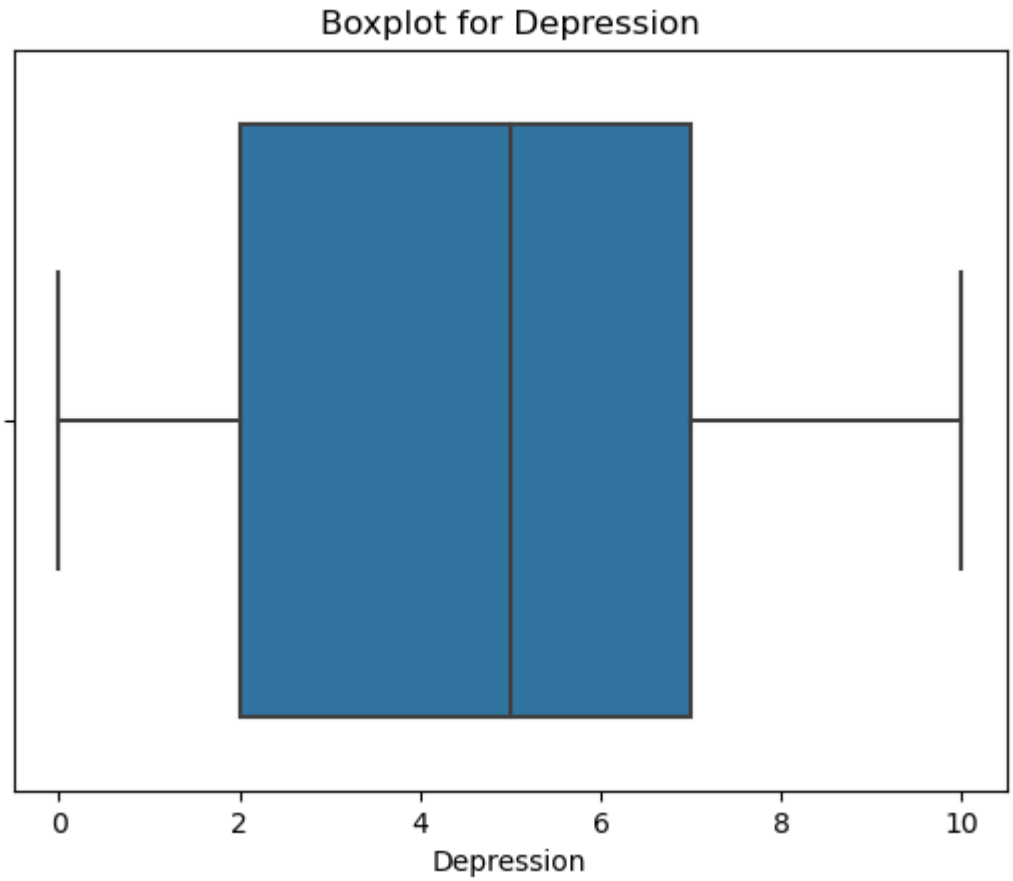
In [6]:
```
import seaborn as sns
import matplotlib.pyplot as plt


columns_of_interest = ["Age", "Hours per day", "BPM", "Anxiety", "Depressior

# Create a boxplot for each selected column
plt.figure(figsize=(12, 8))
for column in columns_of_interest:
    sns.boxplot(x=df[column])
    plt.title(f'Boxplot for {column}')
    plt.show()
```

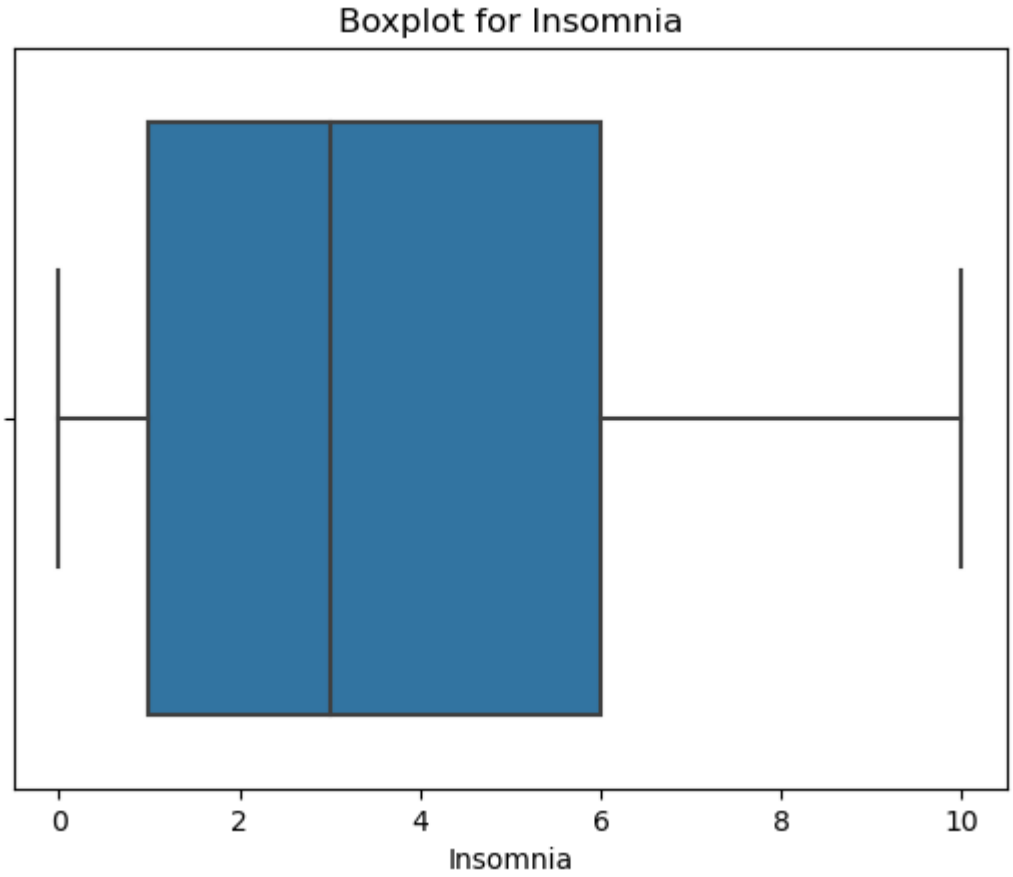


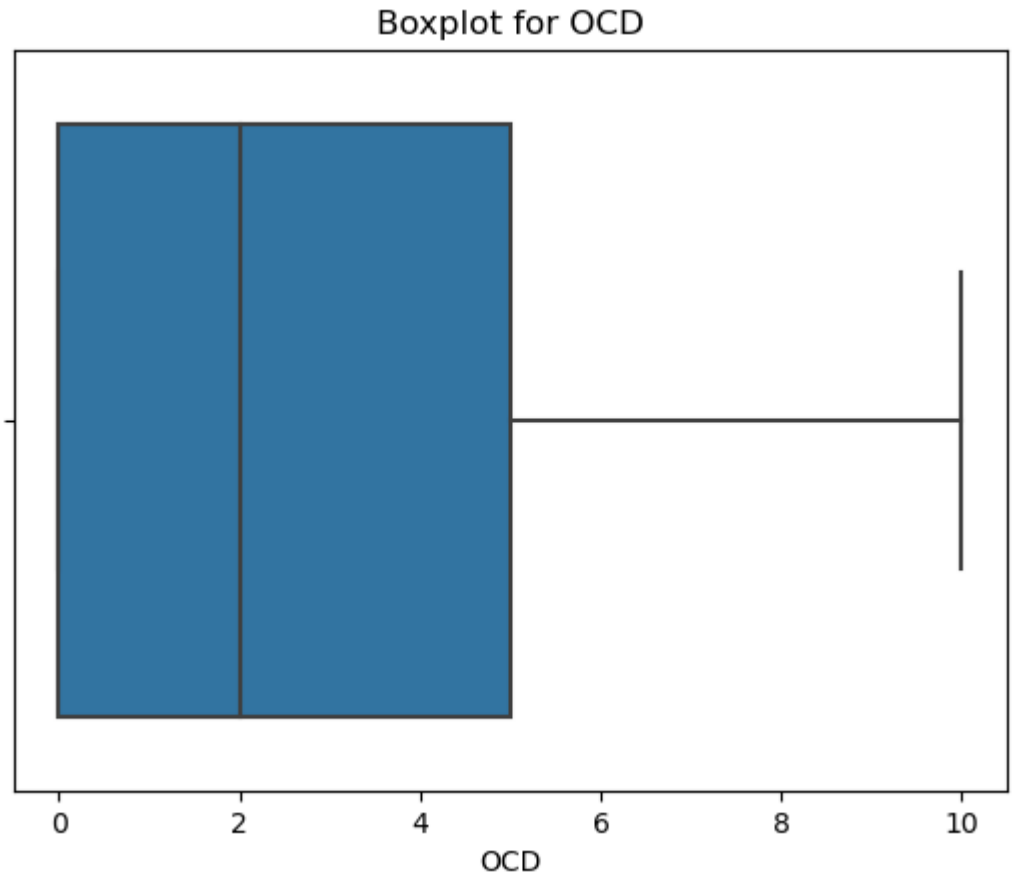

Boxplot for Hours per day
0
5
10
15
20
25
Hours per day


Boxplot for BPM
0.0
0.2
0.4
0.6
0.8
1.0
1e9
BPM


Boxplot for Anxiety
0
2
4
6
8
10
Anxiety


Boxplot for Depression
0
2
4
6
8
10
Depression





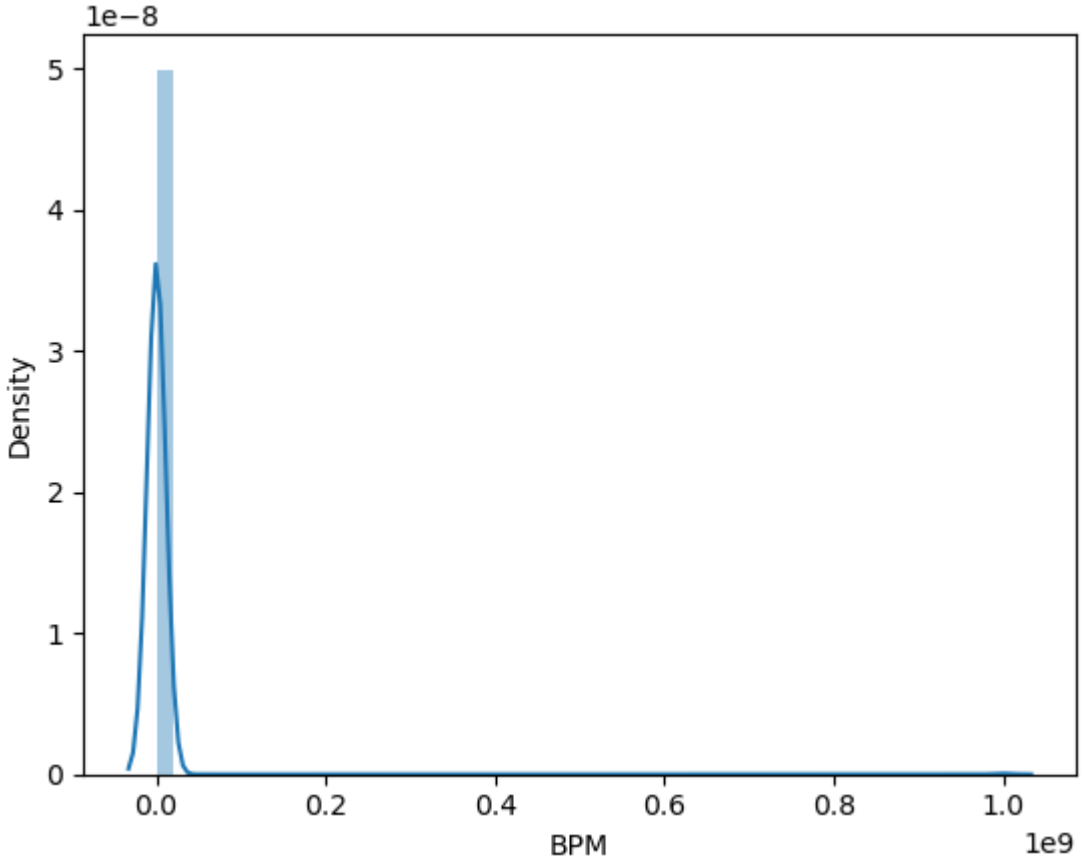
Hours per day exist outlier around 24 hours(1day)which is not reasonable, therefore we need to deal with it.

BPM exist extreme value thus need to deal with it

In [7]:
```
# Plotting Distribution Graph
sns.distplot(df['BPM'])
```

```
C:\Users\ACER\AppData\Local\Temp\ipykernel_16836\3687548621.py:2: UserWarn
ing:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['BPM'])
```

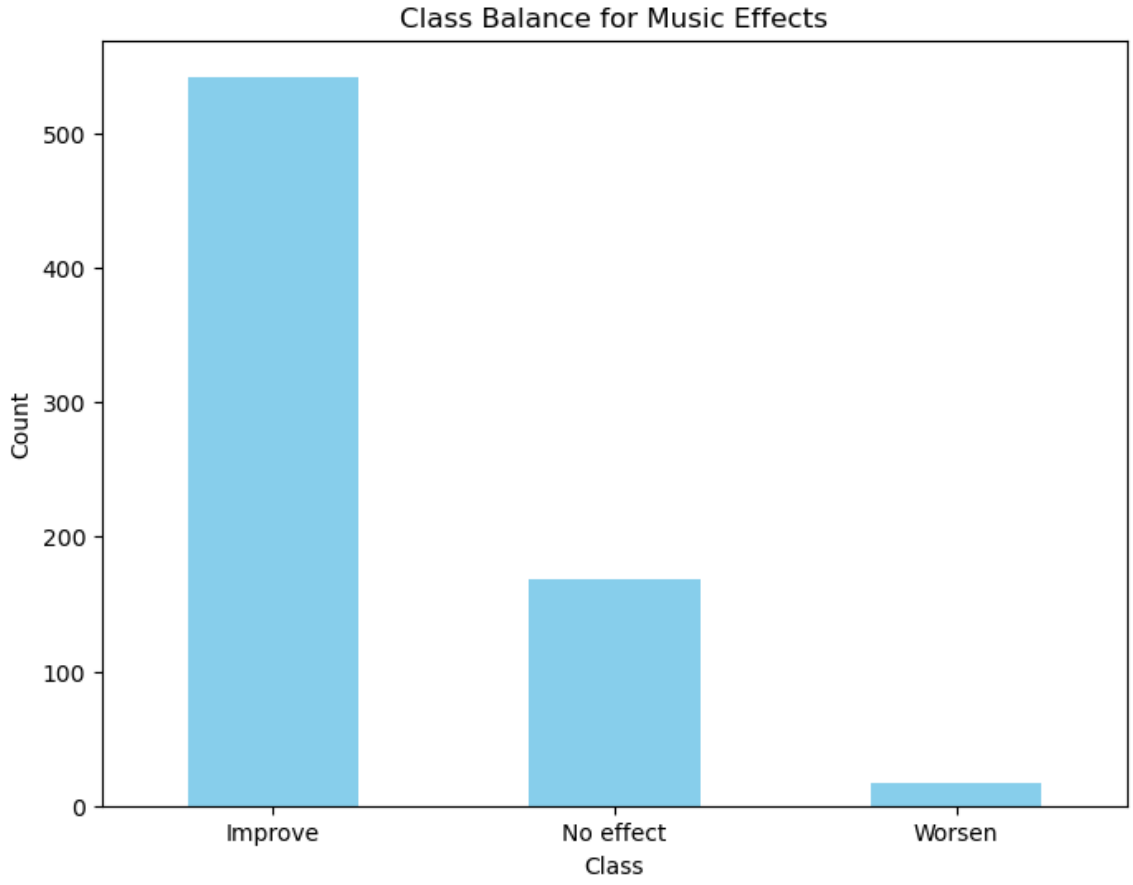
Out[7]: <AxesSubplot: xlabel='BPM', ylabel='Density'>



We found that there is an extreme large value(1.000000e+09) exist in BPM, thus we will check for it

checking for class balancing of response variable

In [8]:
```
class_balance = df['Music effects'].value_counts()

# Plotting
plt.figure(figsize=(8, 6))
class_balance.plot(kind='bar', color='skyblue')
plt.title('Class Balance for Music Effects')
plt.xlabel('Class')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```



In [9]:
```
class_balance = df['Music effects'].value_counts()
print(class_balance)
```

```
Improve      542
No effect    169
Worsen        17
Name: Music effects, dtype: int64
```

Imputation for missing value

In [10]:
```
#fill with mode since categorical data type
df['Primary streaming service'] = df['Primary streaming service'].fillna(df
df['While working'] = df['While working'].fillna(df['While working'].mode()
df['Instrumentalist'] = df['Instrumentalist'].fillna(df['Instrumentalist'].r
df['Composer'] = df['Composer'].fillna(df['Composer'].mode()[0])
df['Foreign languages'] = df['Foreign languages'].fillna(df['Foreign languag
df['Music effects'] = df['Music effects'].fillna(df['Music effects'].mode()
```

In [11]:
```
impute = KNNImputer()
simple_impute = SimpleImputer(missing_values='NAN', strategy='mean')
df['Age'] = impute.fit_transform(df['Age'].values.reshape(-1,1))
df['BPM'] = impute.fit_transform(df['BPM'].values.reshape(-1,1))
```

In [12]:
```
df.isnull().sum()
```

Out[12]:
```
Timestamp                      0
Age                            0
Primary streaming service      0
Hours per day                  0
While working                  0
Instrumentalist                0
Composer                       0
Fav genre                      0
Exploratory                    0
Foreign languages              0
BPM                            0
Frequency [Classical]          0
Frequency [Country]            0
Frequency [EDM]                0
Frequency [Folk]               0
Frequency [Gospel]             0
Frequency [Hip hop]            0
Frequency [Jazz]               0
Frequency [K pop]              0
Frequency [Latin]              0
Frequency [Lofi]               0
Frequency [Metal]              0
Frequency [Pop]                0
Frequency [R&B]                0
Frequency [Rap]                0
Frequency [Rock]               0
Frequency [Video game music]   0
Anxiety                        0
Depression                     0
Insomnia                       0
OCD                            0
Music effects                  0
Permissions                    0
dtype: int64
```

Handle extreme value and capping outliers

In [13]:

```
# Calculate the IQR (Interquartile Range)
Q1 = df['BPM'].quantile(0.25)
Q3 = df['BPM'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['BPM'] < lower_bound) | (df['BPM'] > upper_bound)]

# Display only the 'BPM' column of outliers
print("BPM column outliers:")
print(outliers['BPM'])
```

```
BPM column outliers:
10     1.589948e+06
12     1.589948e+06
15     1.589948e+06
30     1.589948e+06
32     1.589948e+06
           ...
688    1.589948e+06
700    1.589948e+06
706    1.589948e+06
712    1.589948e+06
717    1.589948e+06
Name: BPM, Length: 114, dtype: float64
```

In [14]:

```
# Calculate the median of the 'BPM' column
median_bpm = df['BPM'].median()

# Replace extreme values with the median
df['BPM'] = df['BPM'].apply(lambda x: median_bpm if x == 999999999 else x)

# Calculate the IQR (Interquartile Range) after the initial replacement
Q1 = df['BPM'].quantile(0.25)
Q3 = df['BPM'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for capping
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Apply capping to remaining outliers
df['BPM'] = df['BPM'].apply(lambda x: upper_bound if x > upper_bound else lo
```

In [15]:
```
# Calculate the IQR (Interquartile Range)
Q1 = df['Hours per day'].quantile(0.25)
Q3 = df['Hours per day'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identify outliers
outliers = df[(df['Hours per day'] < lower_bound) | (df['Hours per day'] > u

# Display only the 'hours_per_day' column of outliers
print("Hours per day column outliers:")
print(outliers['Hours per day'])
```

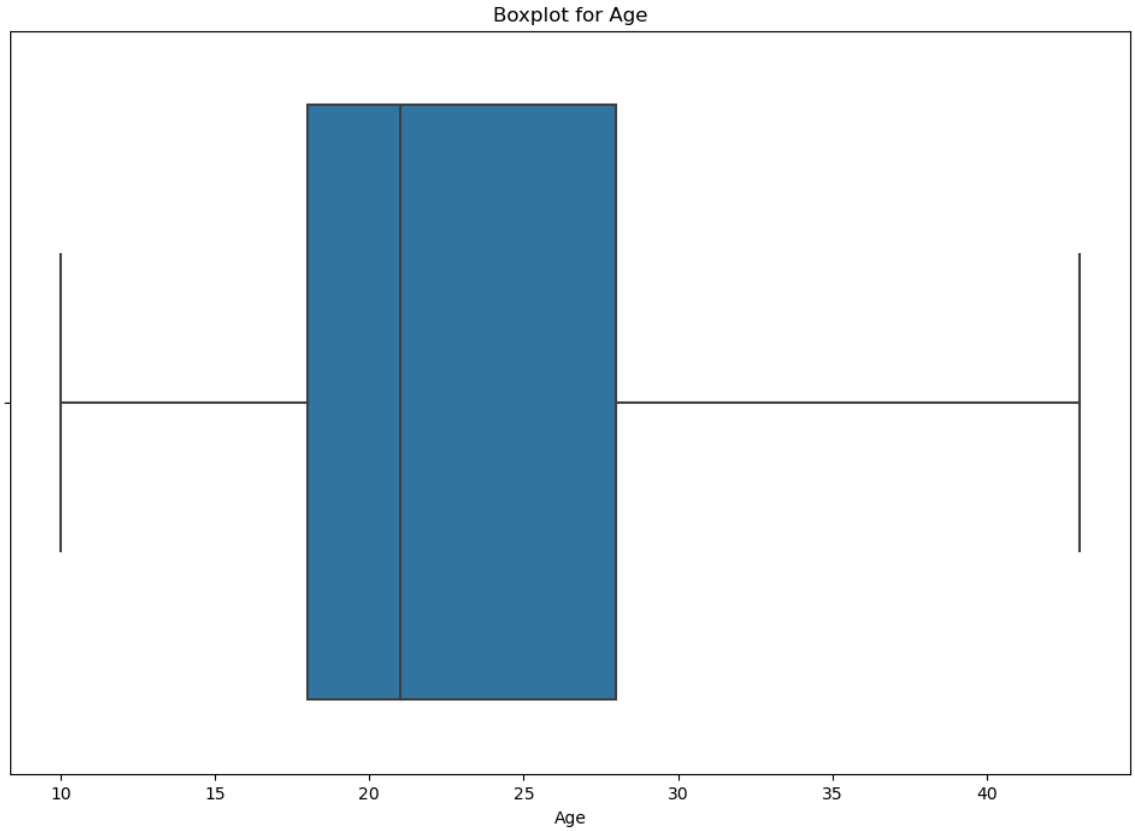
```
Hours per day column outliers:
17     12.0
18     24.0
26     12.0
53     12.0
77     10.0
95     10.0
125    10.0
142    10.0
164    10.0
223    12.0
257    10.0
280    10.0
290    20.0
320    10.0
336    10.0
341    10.0
347    16.0
357    10.0
359    15.0
366    24.0
407    14.0
417    12.0
420    10.0
426    13.0
464    10.0
465    10.0
466    10.0
485    15.0
586    10.0
587    10.0
589    10.0
598    12.0
611    12.0
638    10.0
655    12.0
659    12.0
672    11.0
673    10.0
695    24.0
726    18.0
Name: Hours per day, dtype: float64
```

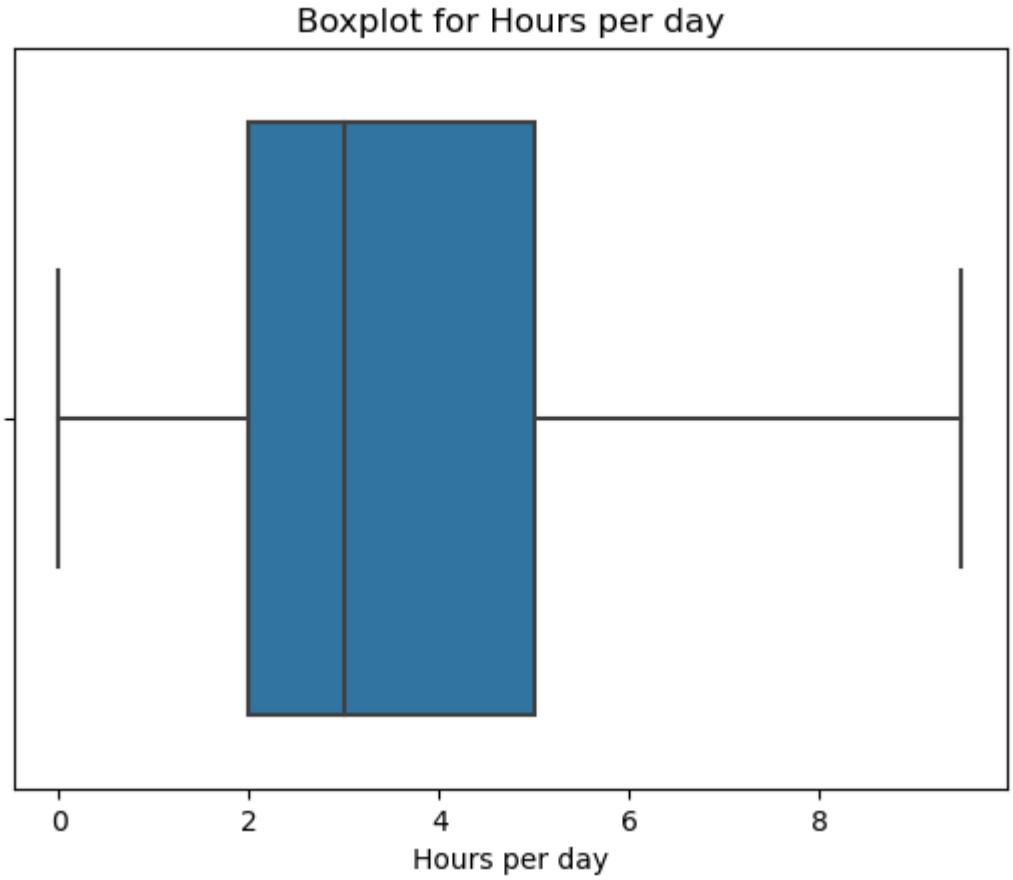
In [16]:
```
# Calculate the median of the 'hours_per_day' column
median_hours_per_day = df['Hours per day'].median()

# Replace extreme values with the median
df['Hours per day'] = df['Hours per day'].apply(lambda x: median_hours_per_d

# Apply capping to remaining outliers
df['Hours per day'] = df['Hours per day'].apply(lambda x: upper_bound if x >
```

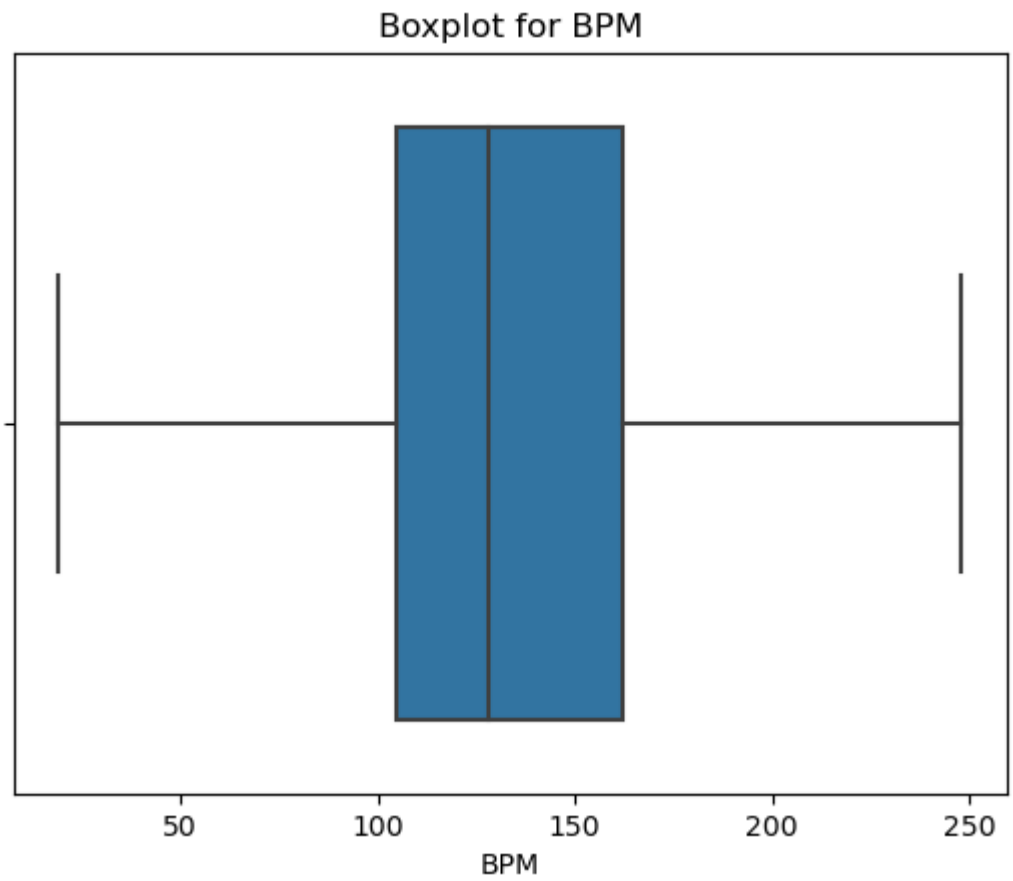
In [17]:
```
# Calculate the IQR (Interquartile Range)
Q1 = df['Age'].quantile(0.25)
Q3 = df['Age'].quantile(0.75)
IQR = Q3 - Q1

# Define the lower and upper bounds for capping
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Apply capping to remaining outliers
df['Age'] = df['Age'].apply(lambda x: upper_bound if x > upper_bound else lo
```

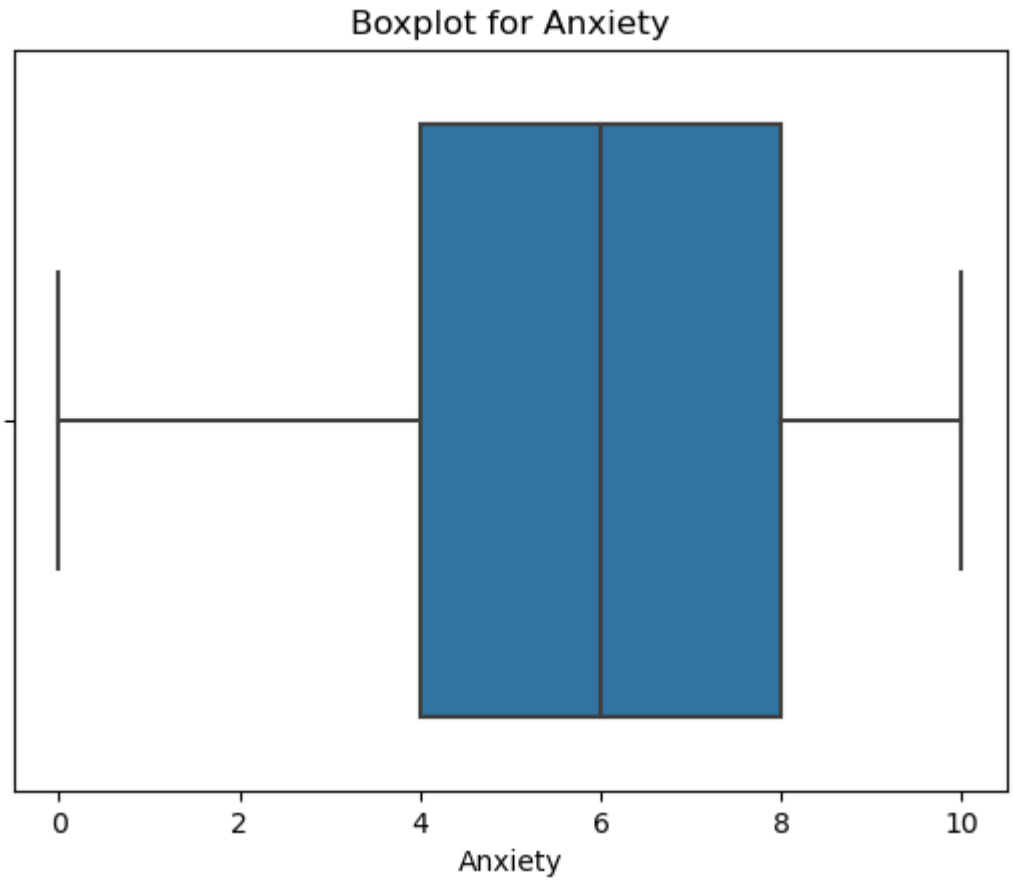
In [18]:

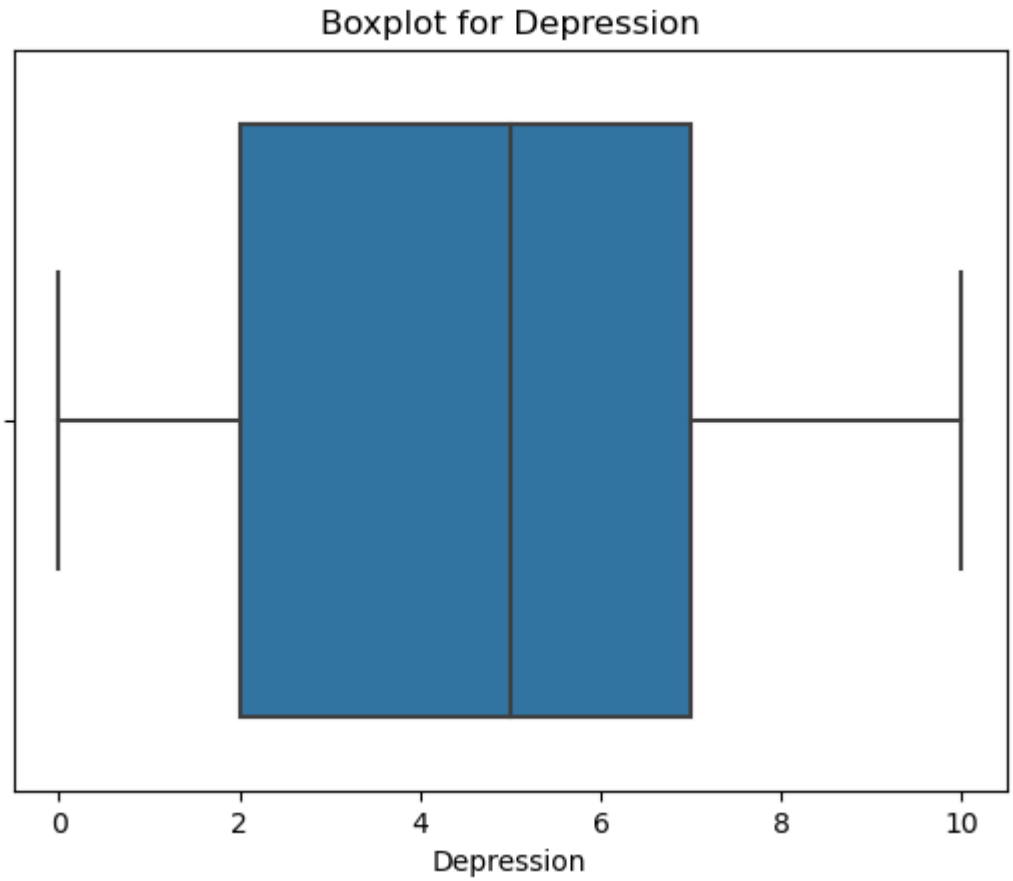
```
columns_of_interest = ["Age", "Hours per day", "BPM", "Anxiety", "Depressior

# Create a boxplot for each selected column
plt.figure(figsize=(12, 8))
for column in columns_of_interest:
    sns.boxplot(x=df[column])
    plt.title(f'Boxplot for {column}')
    plt.show()
```

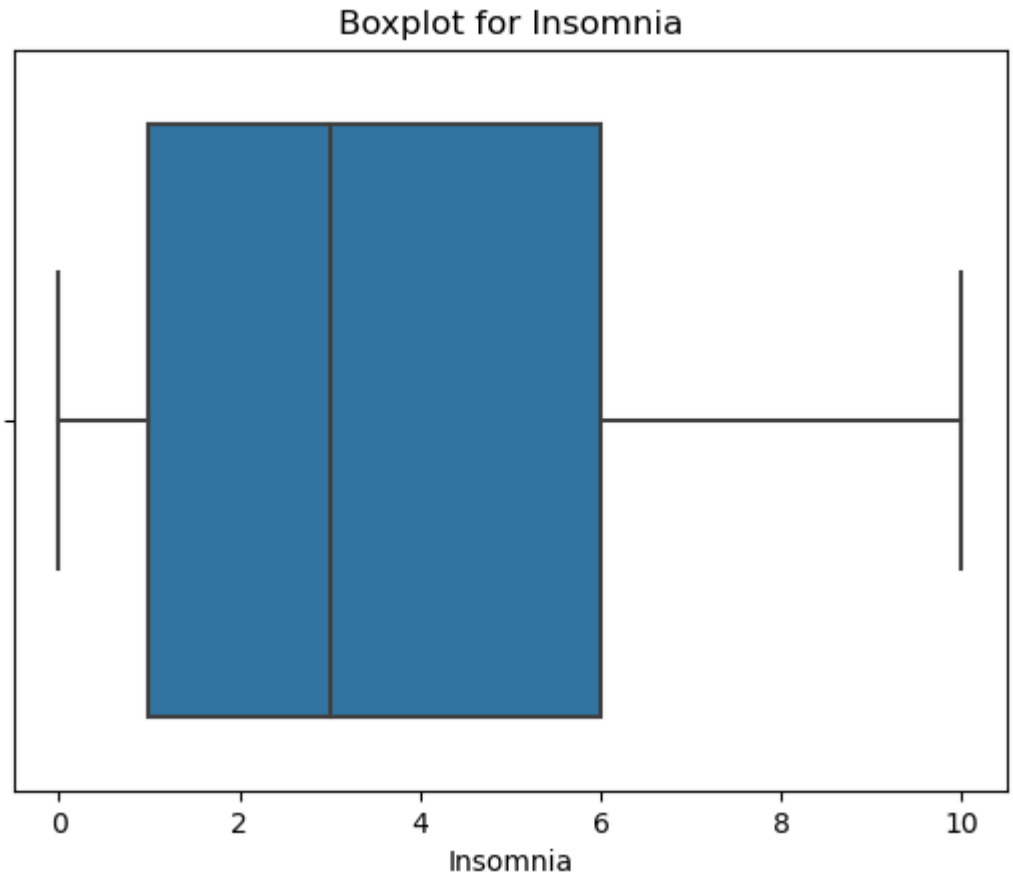


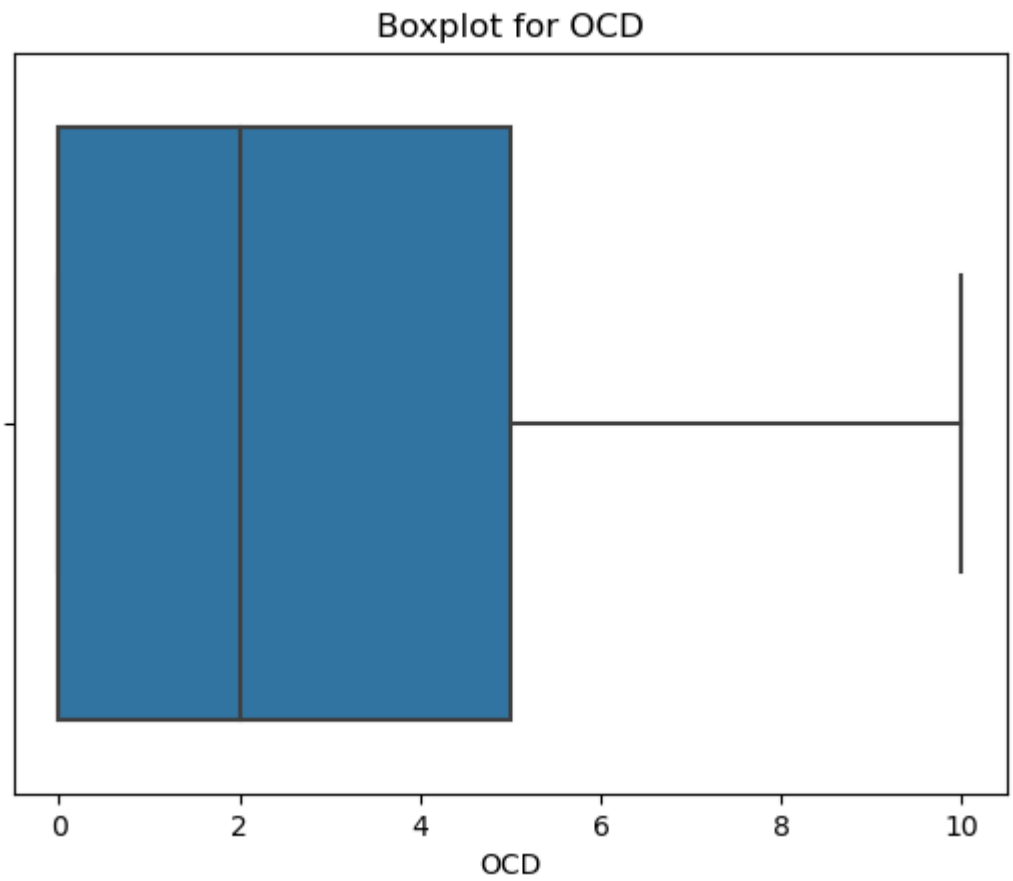

Boxplot for Hours per day
0
2
4
6
8
Hours per day


Boxplot for BPM
50
100
150
200
250
BPM


Boxplot for Anxiety
0
2
4
6
8
10
Anxiety


Boxplot for Depression
0
2
4
6
8
10
Depression

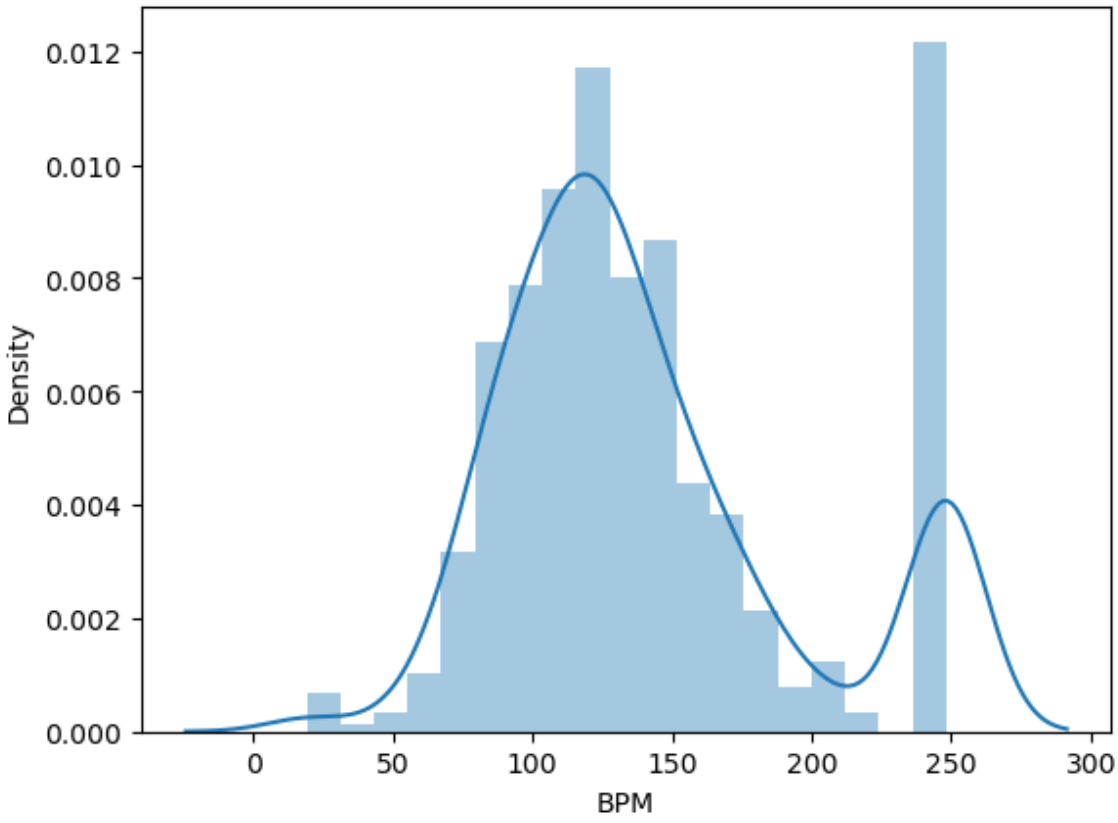

Boxplot for Insomnia
0
2
4
6
8
10
Insomnia


Boxplot for OCD
0
2
4
6
8
10
OCD

In [19]:
```
# Plotting Distribution Graph
sns.distplot(df['BPM'])
```

```
C:\Users\ACER\AppData\Local\Temp\ipykernel_16836\3687548621.py:2: UserWarn
ing:

`distplot` is a deprecated function and will be removed in seaborn v0.14.
0.

Please adapt your code to use either `displot` (a figure-level function wi
th
similar flexibility) or `histplot` (an axes-level function for histogram
s).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://
gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(df['BPM'])
```

Out[19]: <AxesSubplot: xlabel='BPM', ylabel='Density'>



In [20]:
```
df.head(5)
```

Out[20]:

	Timestamp	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Explorato
0	8/27/2022 19:29:02	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	Ye
1	8/27/2022 19:57:31	43.0	Pandora	1.5	Yes	No	No	Rock	Ye
2	8/27/2022 21:28:18	18.0	Spotify	4.0	No	No	No	Video game music	N
3	8/27/2022 21:40:40	43.0	YouTube Music	2.5	Yes	No	Yes	Jazz	Ye
4	8/27/2022 21:54:47	18.0	Spotify	4.0	Yes	No	No	R&B	Ye

5 rows × 33 columns

drop no related column

In [21]:
```
df = df.drop(["Timestamp","Permissions","BPM"],axis=1)
```

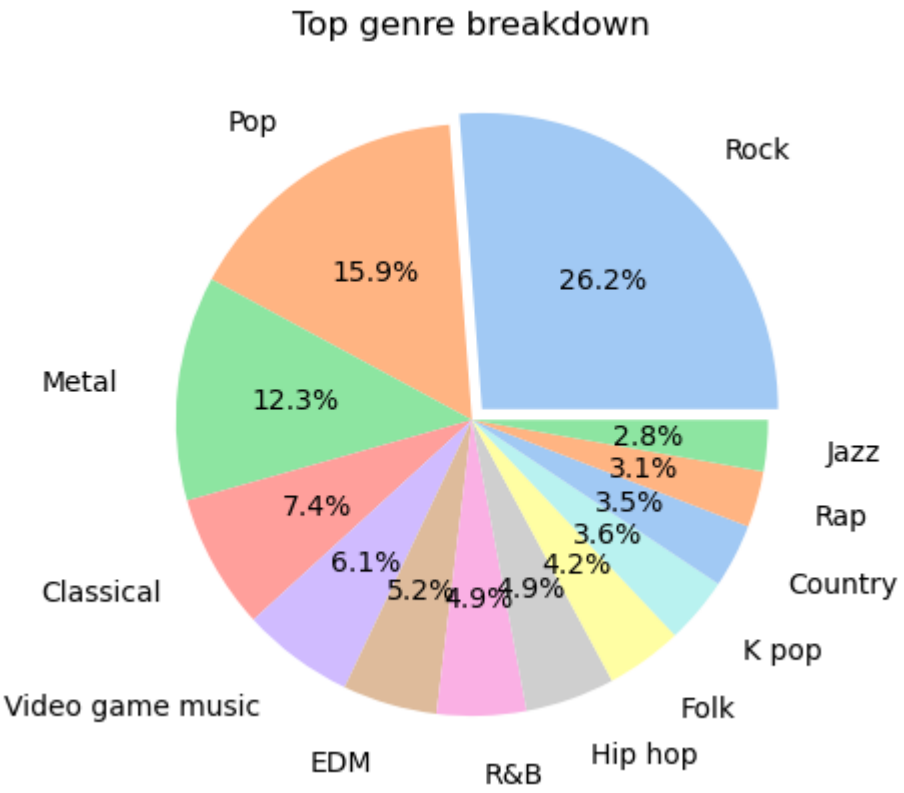
In [22]:
```
df
```

Out[22]:

	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Exploratory	Fc lang
0	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	Yes	
1	43.0	Pandora	1.5	Yes	No	No	Rock	Yes	
2	18.0	Spotify	4.0	No	No	No	Video game music	No	
3	43.0	YouTube Music	2.5	Yes	No	Yes	Jazz	Yes	
4	18.0	Spotify	4.0	Yes	No	No	R&B	Yes	
...	...	...	...	...	...	...	...	...	
731	17.0	Spotify	2.0	Yes	Yes	No	Rock	Yes	
732	18.0	Spotify	1.0	Yes	Yes	No	Pop	Yes	
733	19.0	Other streaming service	6.0	Yes	No	Yes	Rap	Yes	
734	19.0	Spotify	5.0	Yes	Yes	No	Classical	No	
735	29.0	YouTube Music	2.0	Yes	No	No	Hip hop	Yes	

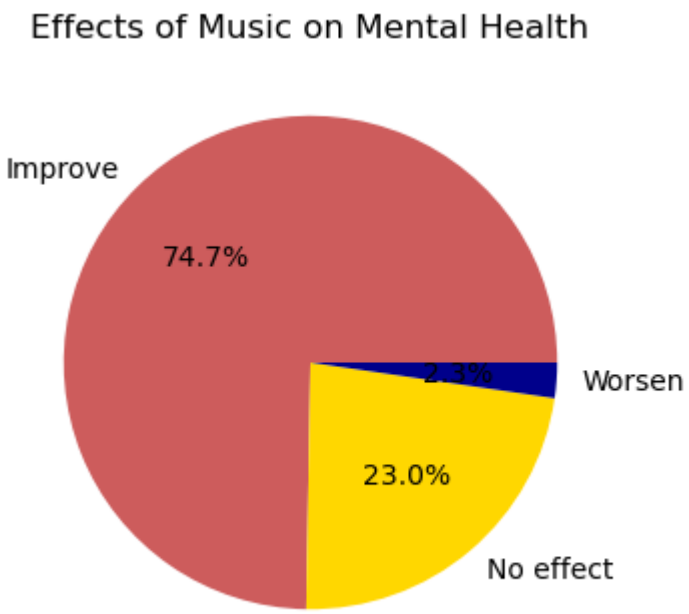
736 rows × 30 columns

In [23]:
```
genre = df["Fav genre"].value_counts().loc[lambda x: x > 10]
total_count = genre.sum()

# Calculate percentages
percentages = genre / total_count * 100

# Plot the pie chart
genre.plot(kind='pie', labeldistance=1.2, explode=[0.05, 0, 0, 0, 0, 0, 0, 0
           colors=sns.color_palette('pastel')[0:13], autopct='%1.1f%%')

plt.title('Top genre breakdown')
plt.ylabel("")
plt.show()
```

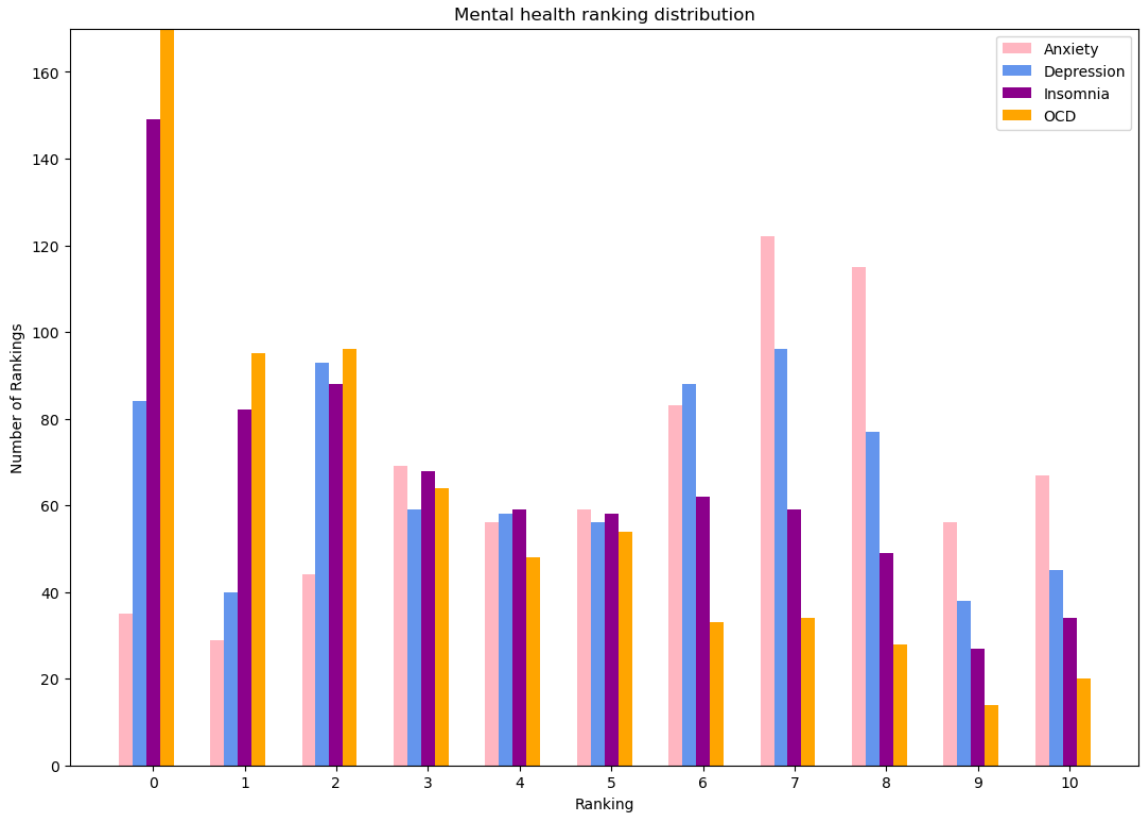


In [24]:

```
plt.figure(figsize=(5, 4))
plt.title('Effects of Music on Mental Health')

effects = df['Music effects'].value_counts()

# Calculate percentages
total_count = effects.sum()
percentages = effects / total_count * 100

# Plot the pie chart with percentages
effects.plot(kind='pie', colors=["indianred", "gold", "darkblue"], autopct=
plt.show()
```

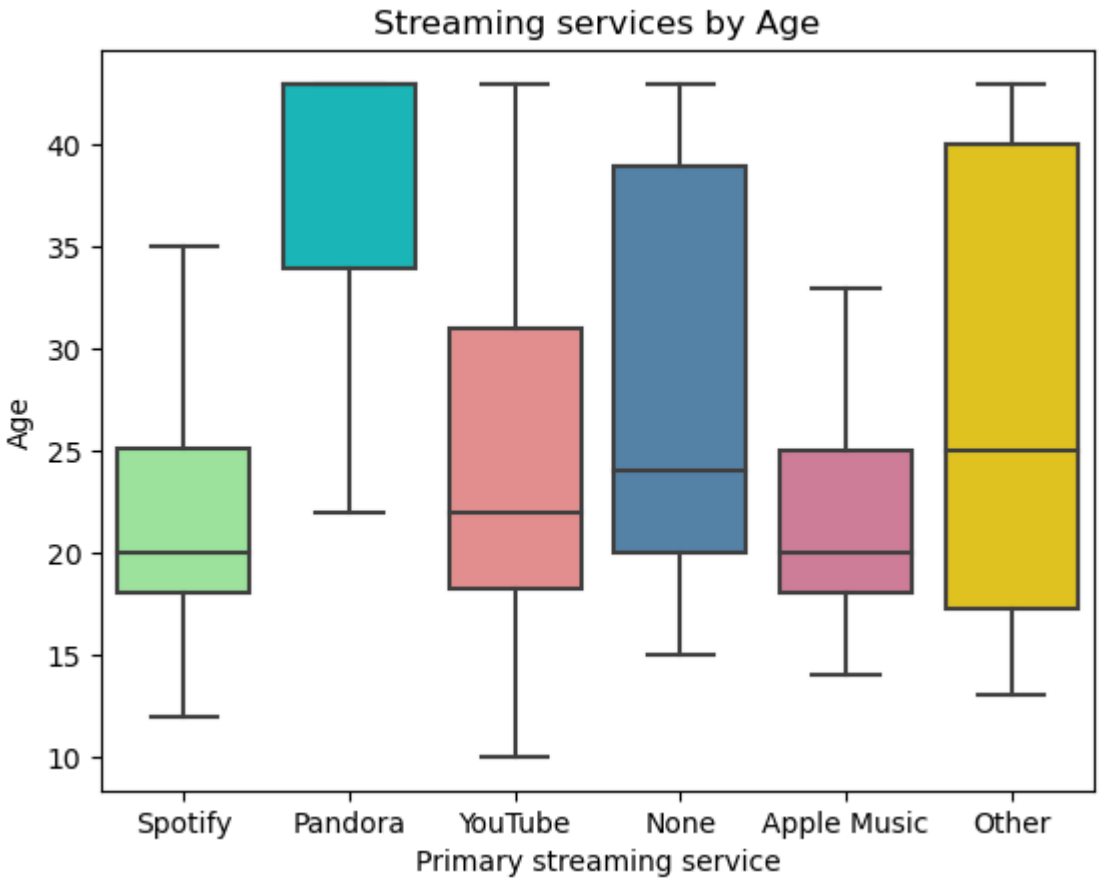


In [25]:
```
m_all = ["Anxiety", "Depression", "Insomnia", "OCD"]

mental_df = df[m_all]
mental_df.round(0).astype(int)

disorder_count = []
for disorder in m_all:
    x=0
    while x !=11:
        count =  (mental_df[disorder].values == x).sum()
        disorder_count.append(count)
        x +=1

labels = ['0','1', '2', '3', '4', '5', '6', '7', '8', '9', '10']
x = np.arange(len(labels))
width = 0.15

fig, ax = plt.subplots(figsize=(13, 9))

b1 = ax.bar(x-2*width, disorder_count[0:11], width, label="Anxiety", color =
b2 = ax.bar(x-width, disorder_count[11:22], width, label="Depression", colo
b3 = ax.bar(x, disorder_count[22:33], width, label="Insomnia", color = 'dar
b4 = ax.bar(x+width, disorder_count[33:], width, label="OCD", color = 'oran

ax.set_ylim([0, 170])
ax.set_ylabel('Number of Rankings')
ax.set_xlabel('Ranking')
ax.set_title('Mental health ranking distribution')
ax.set_xticks(x, labels)
ax.legend()

plt.show()
```



In [26]:
```
s_colors2 = ['lightgreen', 'darkturquoise', 'lightcoral', 'steelblue', 'pale

df.replace(['Other streaming service', 'I do not use a streaming service.',
                        ['Other', 'None', 'YouTube'], inplace=True)

bplot = sns.boxplot(data=df, x="Primary streaming service", y = "Age",
            showfliers = False,
            palette = s_colors2)

plt.title('Streaming services by Age')
```

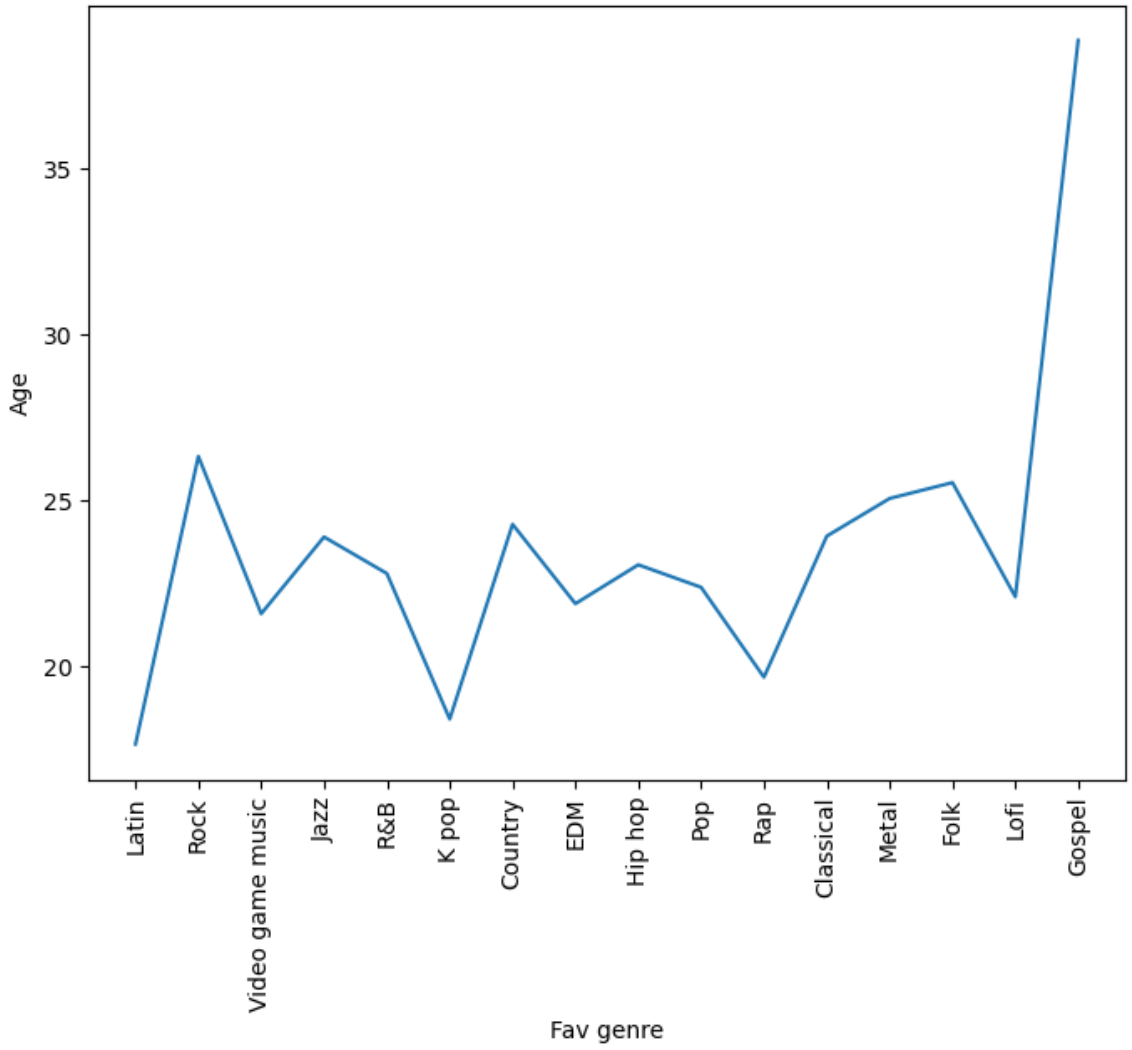
Out[26]: Text(0.5, 1.0, 'Streaming services by Age')



In [27]:
```
plt.figure(figsize=(8,6))
sns.lineplot(x=df['Fav genre'], y=df['Age'], ci=None)
plt.xticks(rotation=90)
```

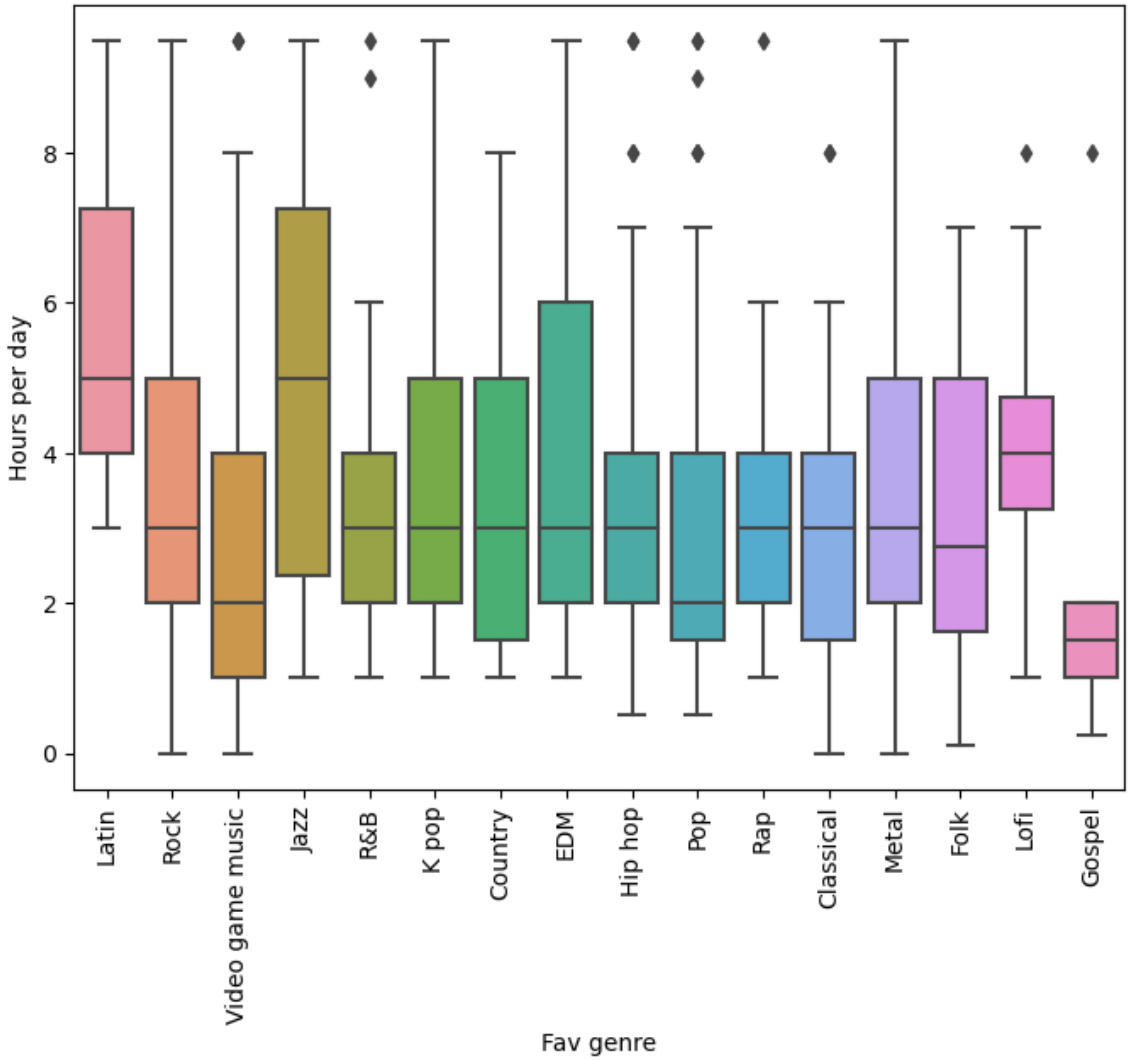
```
C:\Users\ACER\AppData\Local\Temp\ipykernel_16836\1472583737.py:2: FutureWa
rning:

The `ci` parameter is deprecated. Use `errorbar=None` for the same effect.

  sns.lineplot(x=df['Fav genre'], y=df['Age'], ci=None)
```

Out[27]:
```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



In [28]:
```
plt.figure(figsize=(8,6))
sns.boxplot(x=df['Fav genre'], y=df['Hours per day'])
plt.xticks(rotation=90)
```

Out[28]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```

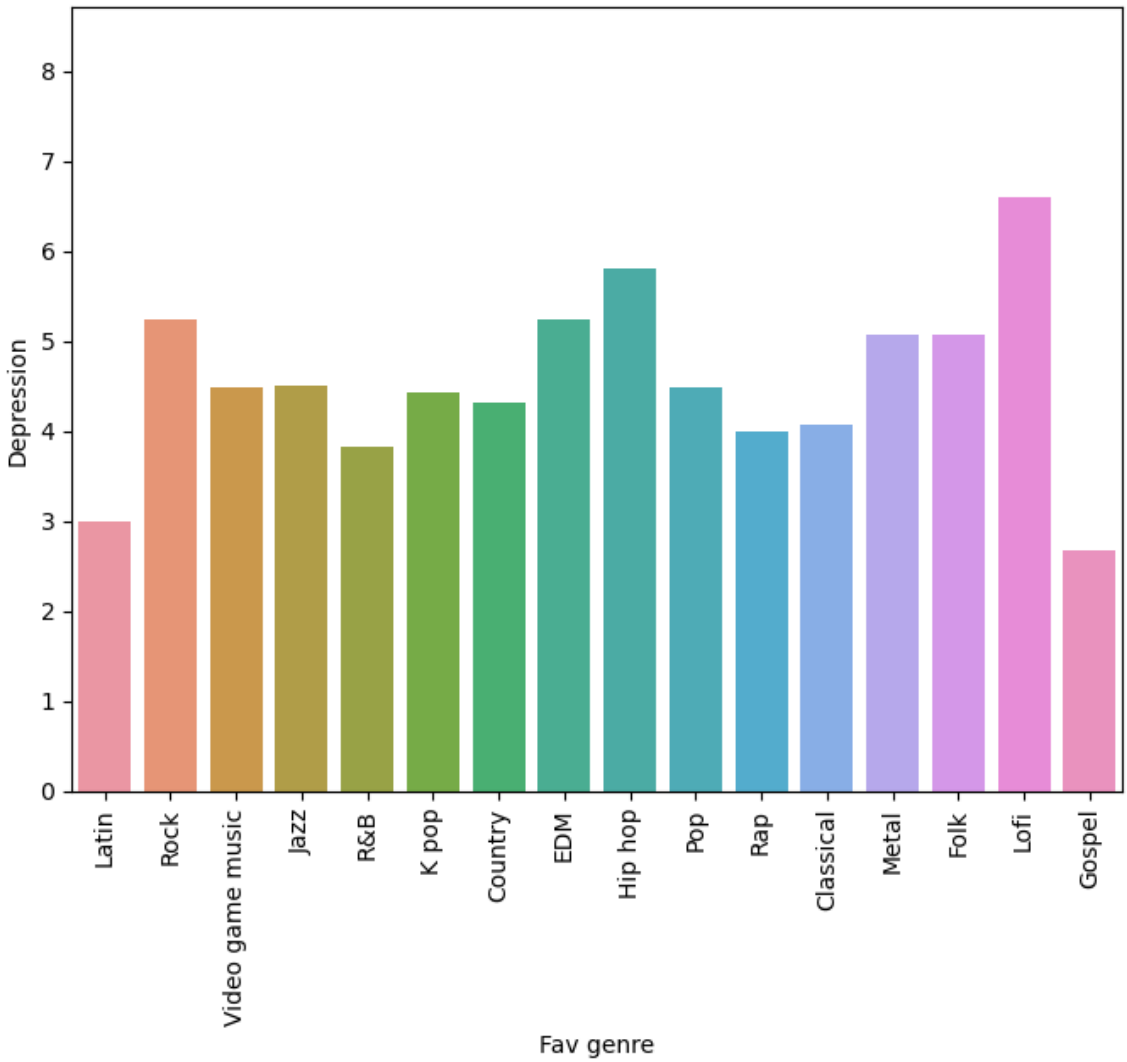


In [29]:
```
for disorder in m_all:
    d_avg = str(round(df[disorder].mean(), 2))
    print(disorder + ' average: ' + d_avg)
```

```
Anxiety average: 5.84
Depression average: 4.8
Insomnia average: 3.74
OCD average: 2.64
```

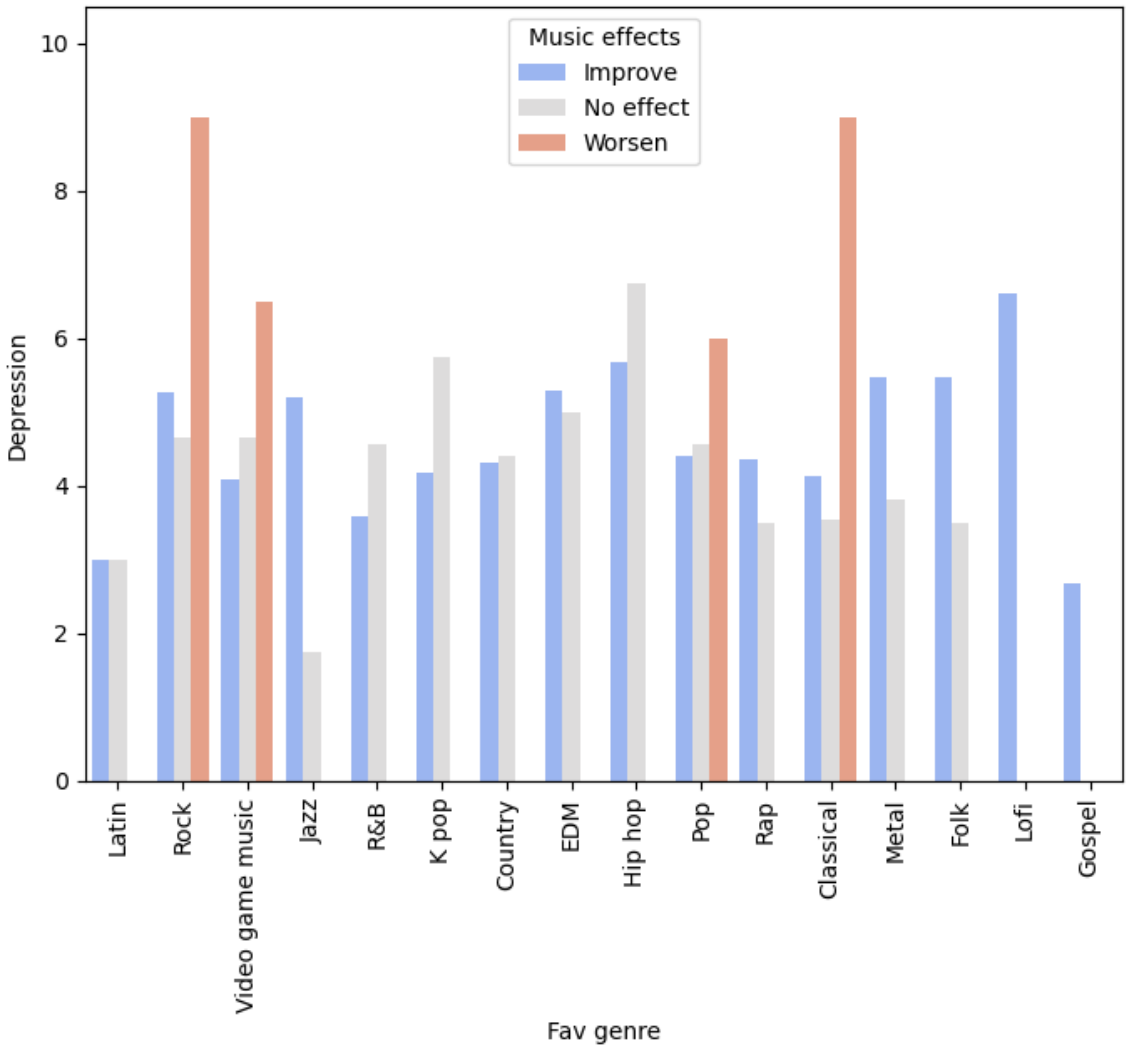
In [30]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['Depression'], errwidth=0)
plt.xticks(rotation=90)
```

Out[30]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



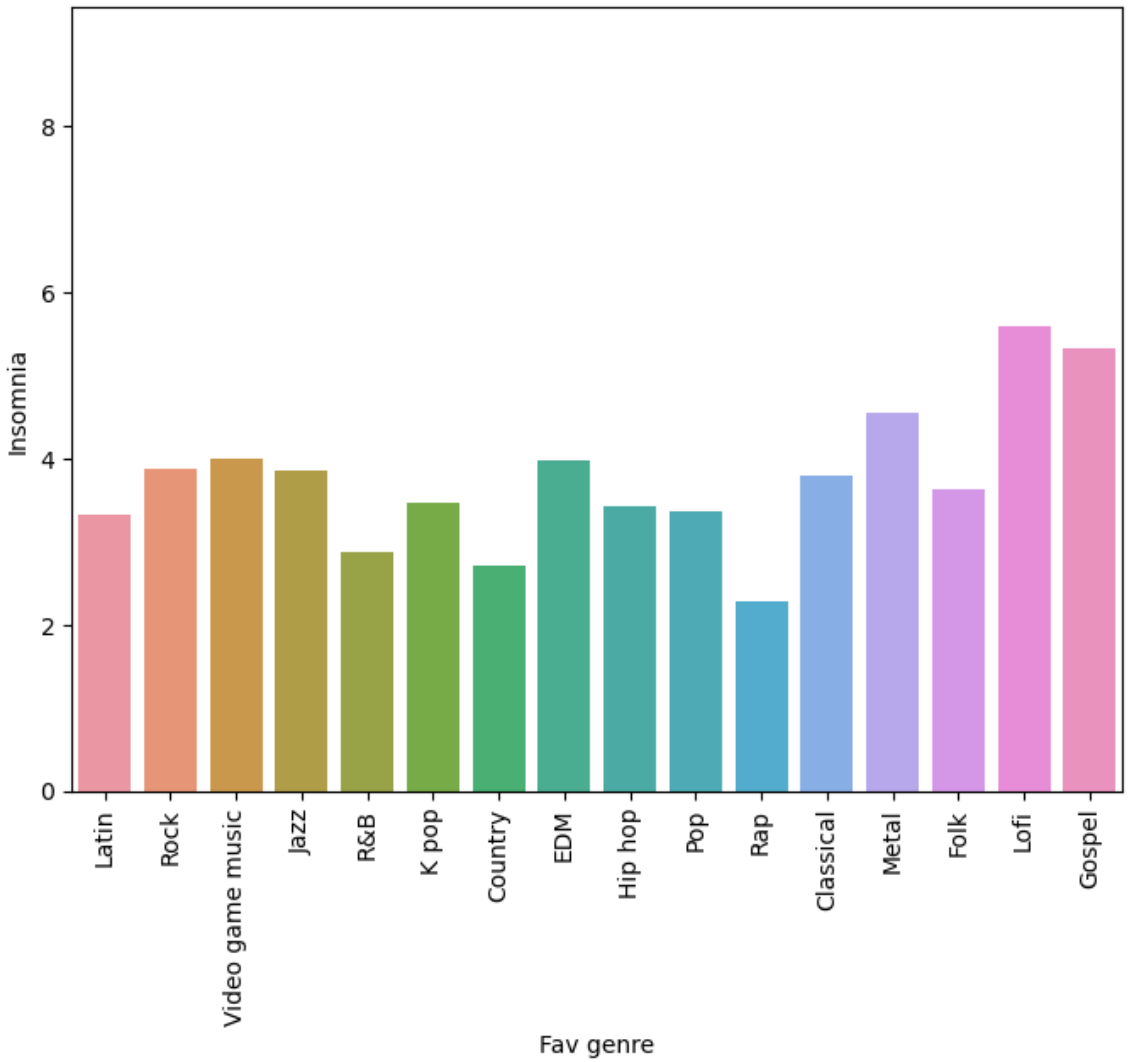
In [31]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['Depression'], hue=df['Music effects'],
plt.xticks(rotation=90)
```

Out[31]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



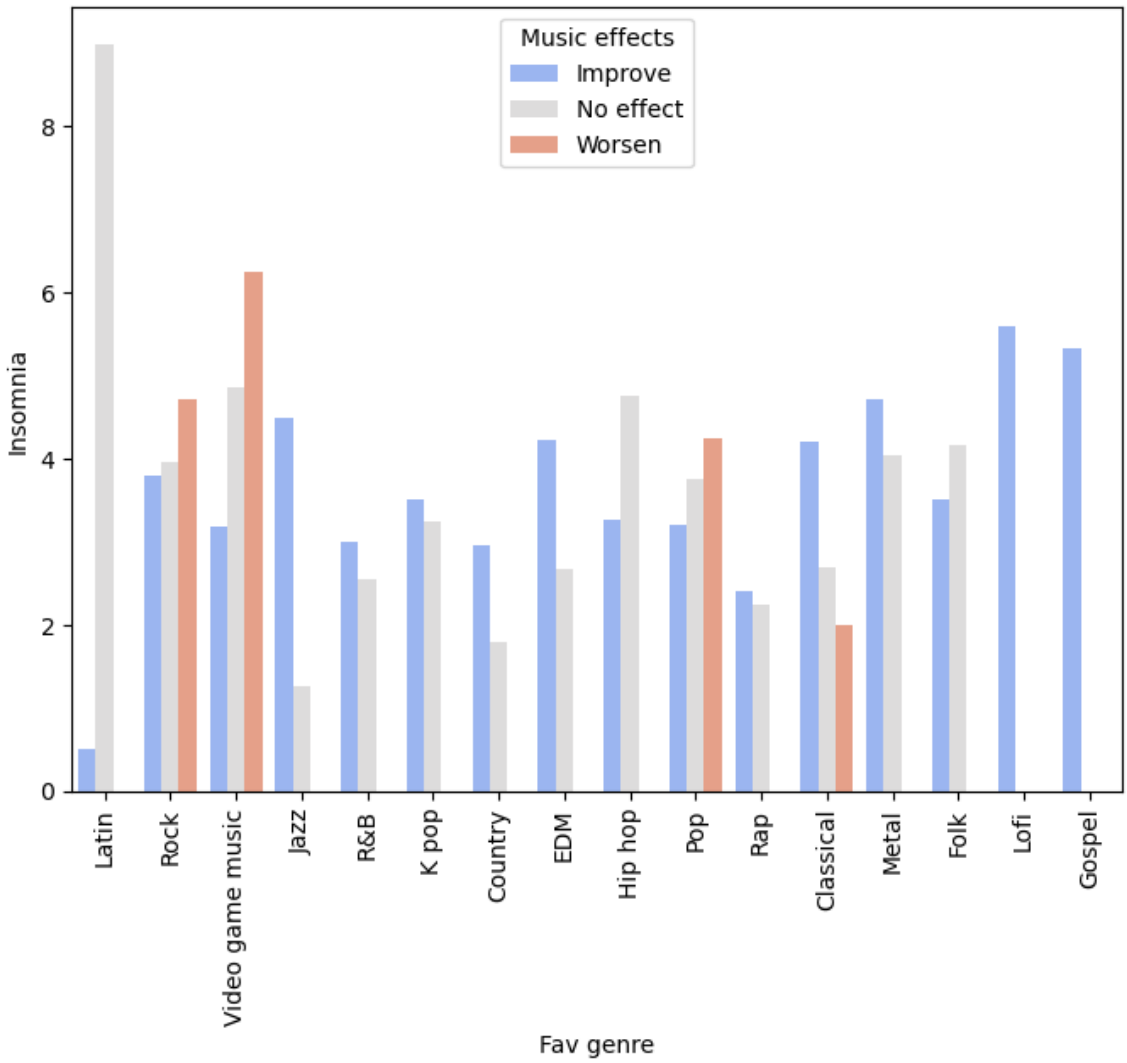
In [32]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['Insomnia'], errwidth=0)
plt.xticks(rotation=90)
```

Out[32]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



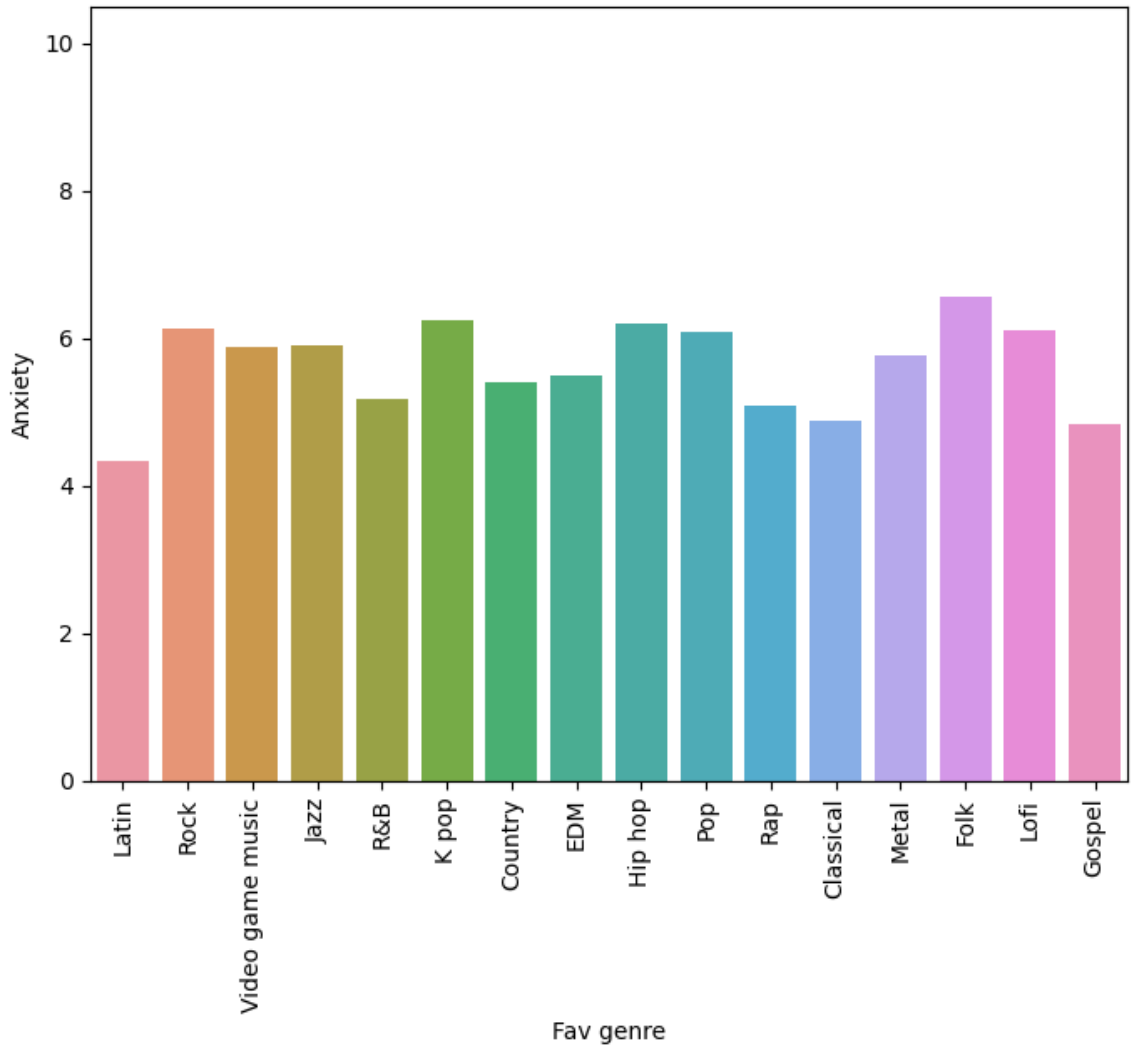
In [33]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['Insomnia'], hue=df['Music effects'], er
plt.xticks(rotation=90)
```

Out[33]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



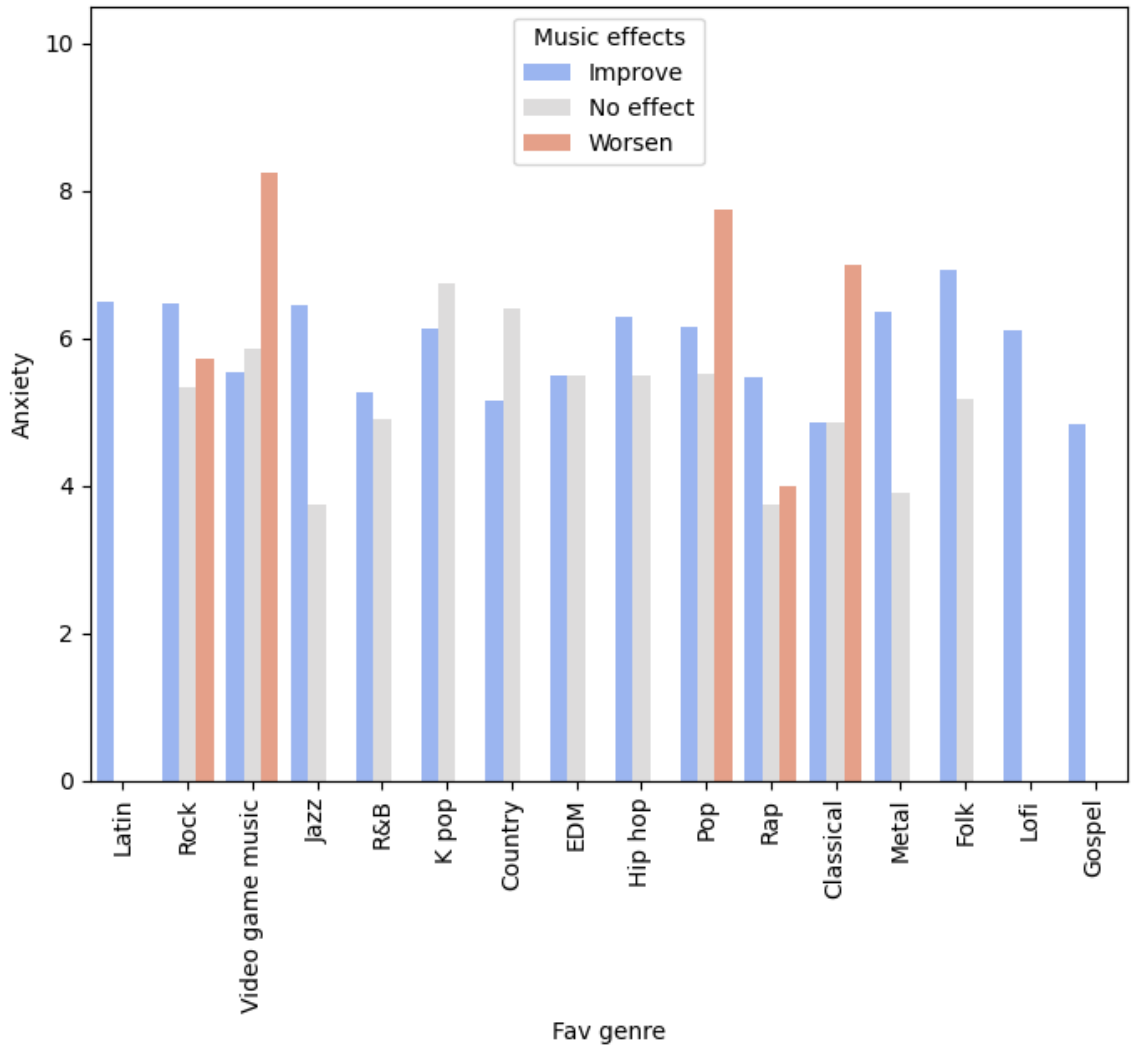
In [34]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['Anxiety'], errwidth=0)
plt.xticks(rotation=90)
```

Out[34]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



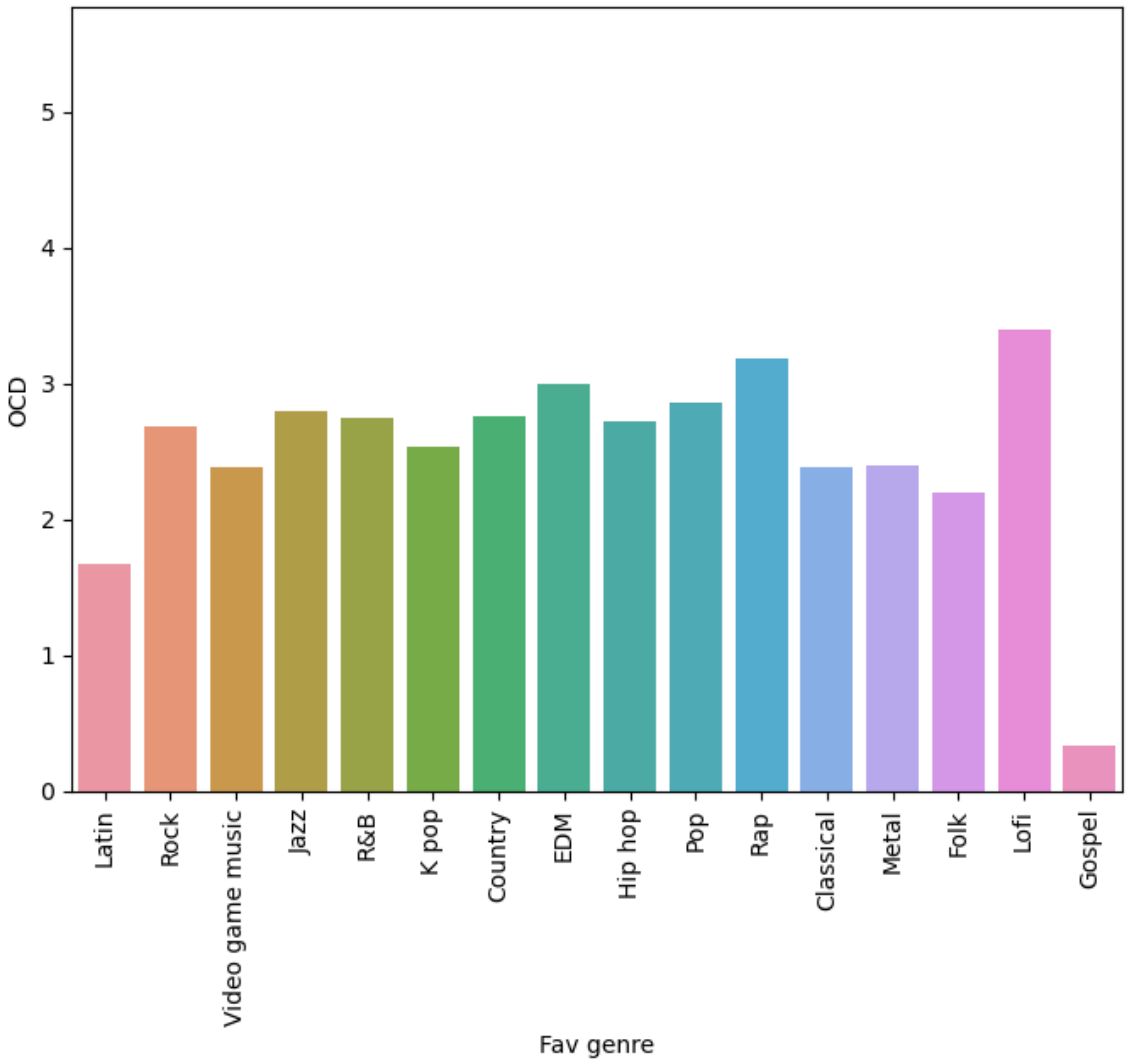
In [35]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['Anxiety'], hue=df['Music effects'], er
plt.xticks(rotation=90)
```

Out[35]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



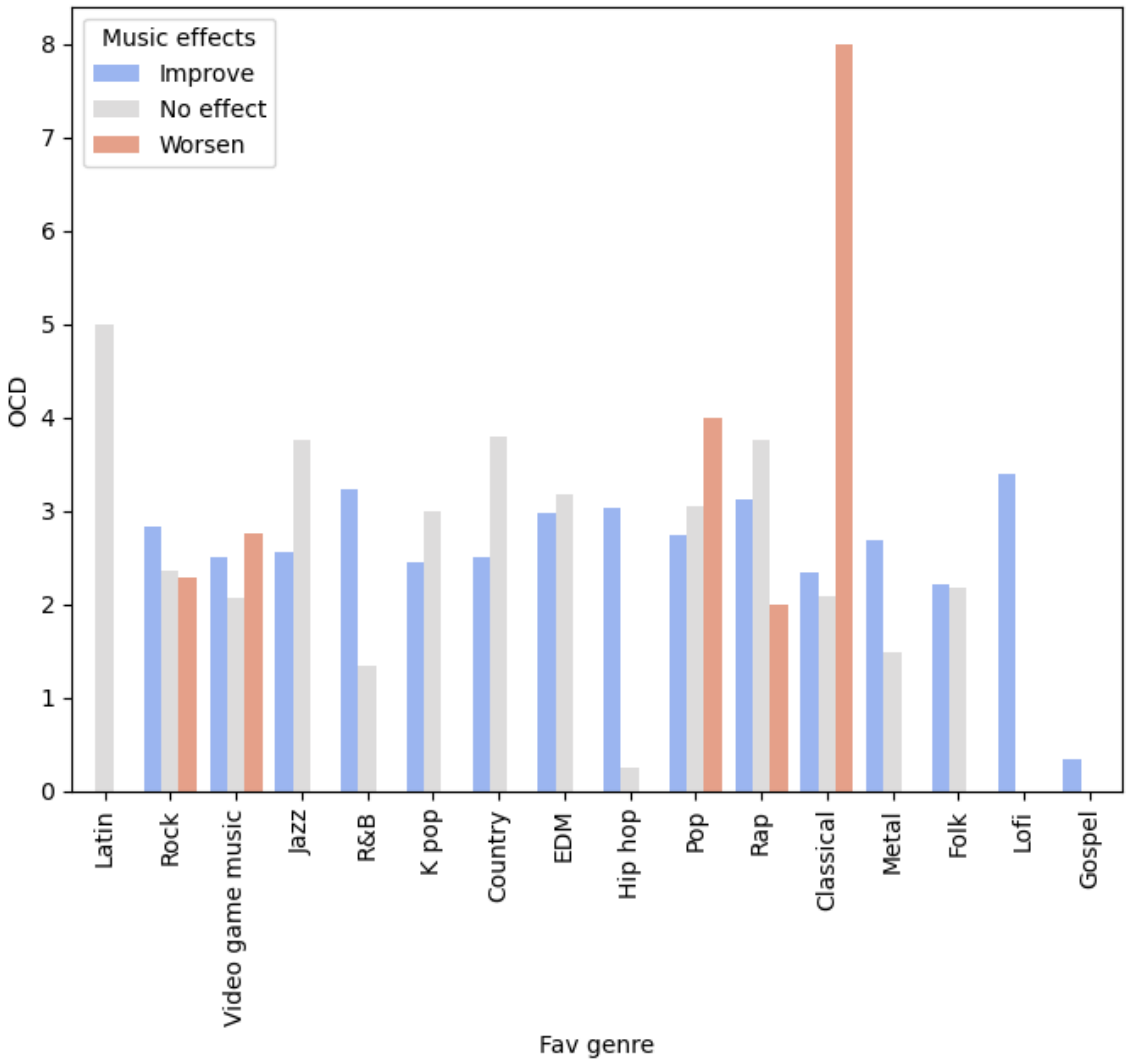
In [36]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['OCD'], errwidth=0)
plt.xticks(rotation=90)
```

Out[36]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```

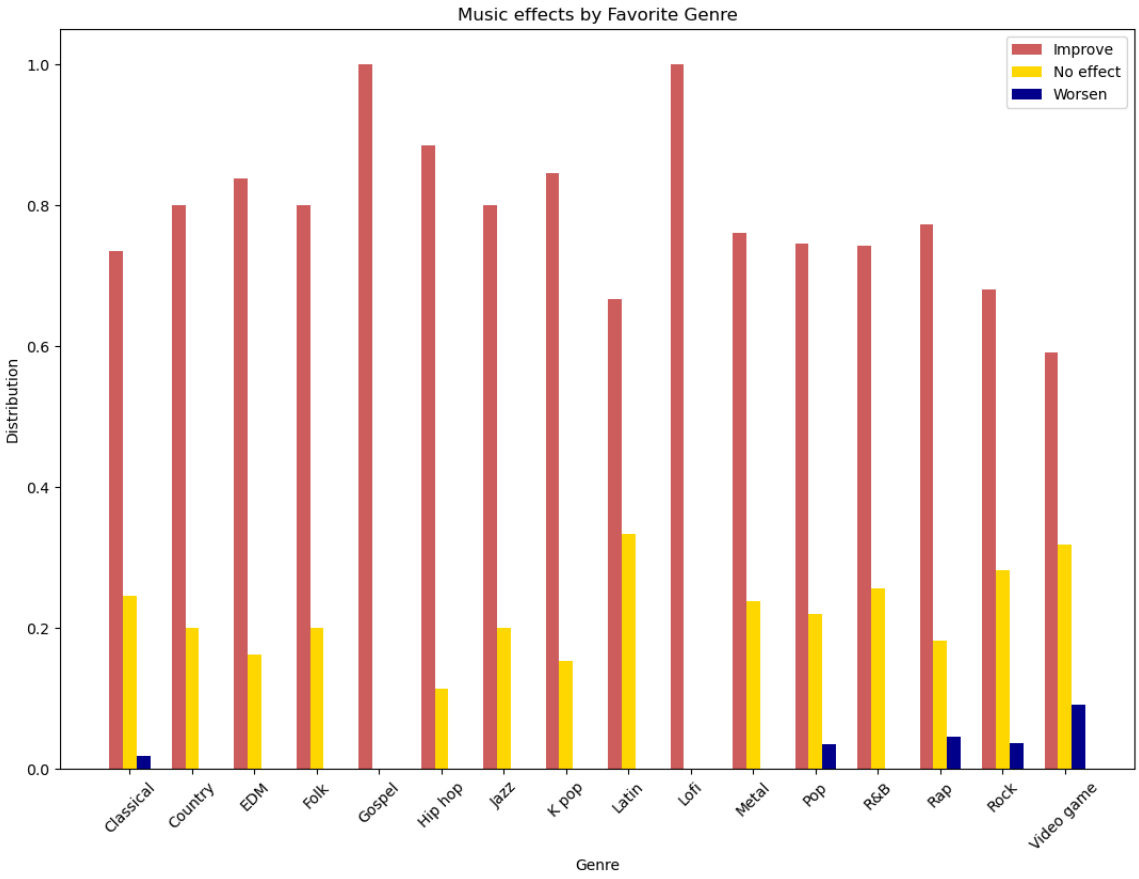


In [37]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=df['Fav genre'], y=df['OCD'], hue=df['Music effects'], errwidt
plt.xticks(rotation=90)
```

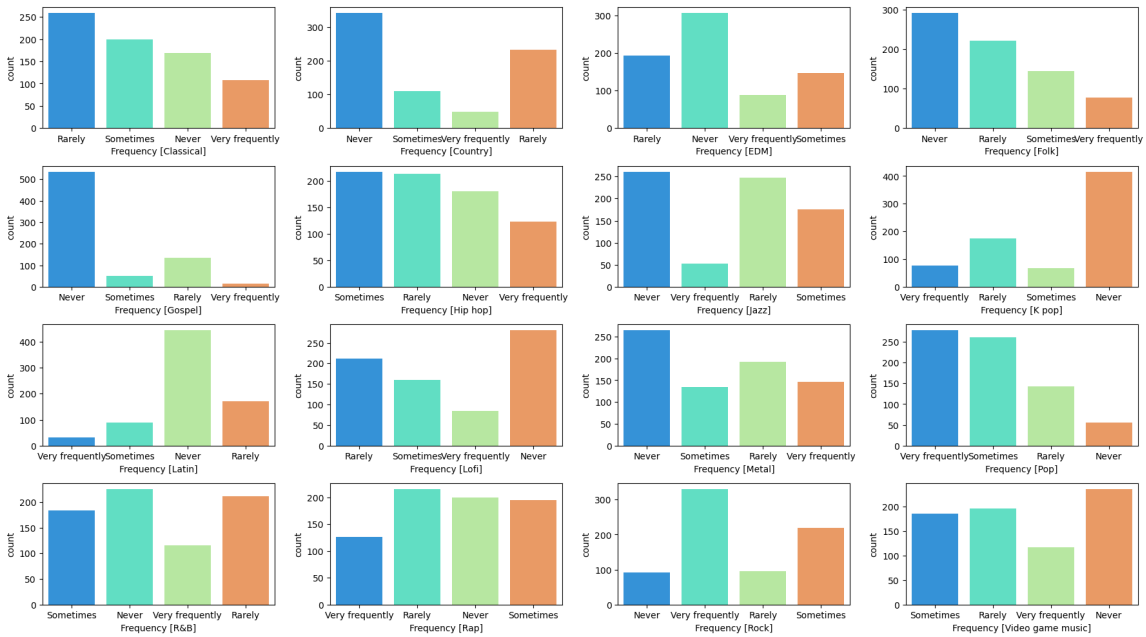
Out[37]:
```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15]),
 [Text(0, 0, 'Latin'),
  Text(1, 0, 'Rock'),
  Text(2, 0, 'Video game music'),
  Text(3, 0, 'Jazz'),
  Text(4, 0, 'R&B'),
  Text(5, 0, 'K pop'),
  Text(6, 0, 'Country'),
  Text(7, 0, 'EDM'),
  Text(8, 0, 'Hip hop'),
  Text(9, 0, 'Pop'),
  Text(10, 0, 'Rap'),
  Text(11, 0, 'Classical'),
  Text(12, 0, 'Metal'),
  Text(13, 0, 'Folk'),
  Text(14, 0, 'Lofi'),
  Text(15, 0, 'Gospel')])
```



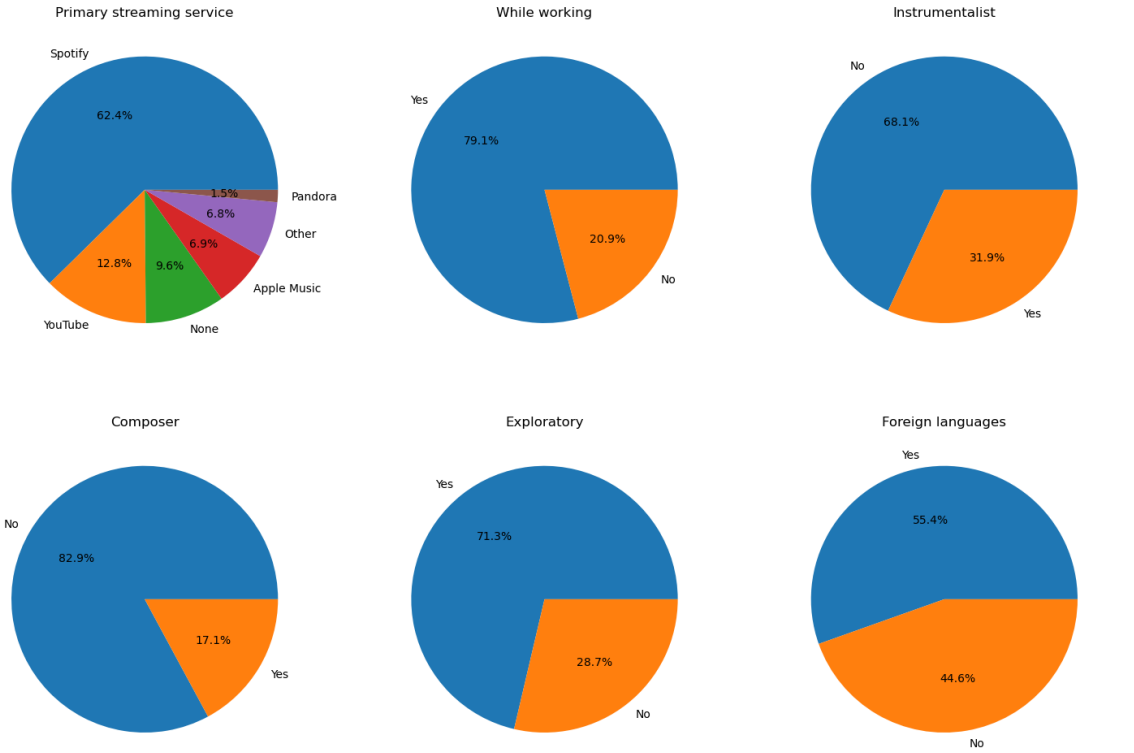
In [38]:
```
df.replace(['Video game music'],
                        ['Video game'], inplace=True)

g_all = df['Fav genre'].unique()
g_all.sort()
fg_df = df.groupby(['Fav genre'])
fg_dist = fg_df['Music effects'].value_counts(ascending=False, normalize=Tru

insert_indices = [5, 8, 11, 13, 14, 17, 20, 23, 26, 28, 29, 32, 38]
for i in range(len(insert_indices)):
    fg_dist.insert(insert_indices[i], 0)

imp_dist = fg_dist[0::3]
no_eff_dist = fg_dist[1::3]
wors_dist = fg_dist[2::3]

width = 0.22

x = np.arange(len(g_all))

fig, ax = plt.subplots(figsize=(13, 9))

b1 = ax.bar(x-width, imp_dist, width, label="Improve", color = 'indianred')
b2 = ax.bar(x, no_eff_dist, width, label="No effect", color = 'gold')
b3 = ax.bar(x+width, wors_dist, width, label="Worsen", color = 'darkblue')

plt.title("Music effects by Favorite Genre")
ax.set_ylabel('Distribution')
ax.set_xlabel('Genre')
ax.set_xticks(x, g_all, rotation = 45)
ax.legend()

plt.show()
```


Music effects by Favorite Genre
Improve
No effect
Worsen
Distribution
1.0
0.8
0.6
0.4
0.2
0.0
Classical
Country
EDM
Folk
Gospel
Hip hop
Jazz
K pop
Latin
Lofi
Metal
Pop
R&B
Rap
Rock
Video game
Genre

In [39]:
```
figure,axes=plt.subplots(4,4,figsize=(18,10))
sns.countplot(ax=axes[0,0],x=df['Frequency [Classical]'],palette="rainbow")
sns.countplot(ax=axes[0,1],x=df['Frequency [Country]'],palette="rainbow")

sns.countplot(ax=axes[0,2],x=df['Frequency [EDM]'],palette="rainbow")

sns.countplot(ax=axes[0,3],x=df['Frequency [Folk]'],palette="rainbow")

sns.countplot(ax=axes[1,0],x=df['Frequency [Gospel]'],palette="rainbow")

sns.countplot(ax=axes[1,1],x=df['Frequency [Hip hop]'],palette="rainbow")

sns.countplot(ax=axes[1,2],x=df['Frequency [Jazz]'],palette="rainbow")

sns.countplot(ax=axes[1,3],x=df['Frequency [K pop]'],palette="rainbow")

sns.countplot(ax=axes[2,0],x=df['Frequency [Latin]'],palette="rainbow")

sns.countplot(ax=axes[2,1],x=df['Frequency [Lofi]'],palette="rainbow")

sns.countplot(ax=axes[2,2],x=df['Frequency [Metal]'],palette="rainbow")

sns.countplot(ax=axes[2,3],x=df['Frequency [Pop]'],palette="rainbow")

sns.countplot(ax=axes[3,0],x=df['Frequency [R&B]'],palette="rainbow")

sns.countplot(ax=axes[3,1],x=df['Frequency [Rap]'],palette="rainbow")

sns.countplot(ax=axes[3,2],x=df['Frequency [Rock]'],palette="rainbow")

sns.countplot(ax=axes[3,3],x=df['Frequency [Video game music]'],palette="ra

plt.tight_layout()
```



In [40]:
```
#i used pie charts to visualize the distribution of some columns
pieColumnNames=['Primary streaming service', 'While working', 'Instrumentali
fig, axes = plt.subplots(2, 3, figsize=(18, 12))

for i in range(len(pieColumnNames)):
  columnName = pieColumnNames[i]
  valueCount = df[columnName].value_counts()
  axes[i//3, i%3].pie(valueCount, labels=valueCount.index, autopct='%1.1f%%
  axes[i//3, i%3].set_title(columnName)
```



library for modelling

In [41]:
```
#!pip install category_encoders
```

In [42]:
```
#! pip install catboost
```

In [43]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
import category_encoders as ce
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [44]:
```
df
```

Out[44]:

	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Exploratory	Fo lang
0	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	Yes	
1	43.0	Pandora	1.5	Yes	No	No	Rock	Yes	
2	18.0	Spotify	4.0	No	No	No	Video game	No	
3	43.0	YouTube	2.5	Yes	No	Yes	Jazz	Yes	
4	18.0	Spotify	4.0	Yes	No	No	R&B	Yes	
...	...	...	...	...	...	...	...	...	
731	17.0	Spotify	2.0	Yes	Yes	No	Rock	Yes	
732	18.0	Spotify	1.0	Yes	Yes	No	Pop	Yes	
733	19.0	Other	6.0	Yes	No	Yes	Rap	Yes	
734	19.0	Spotify	5.0	Yes	Yes	No	Classical	No	
735	29.0	YouTube	2.0	Yes	No	No	Hip hop	Yes	

736 rows × 30 columns

Encode

In [45]:
```
le = LabelEncoder()
df['Music effects'] = le.fit_transform(df['Music effects'])

# Access the mapping
label_mapping = dict(zip(le.classes_, le.transform(le.classes_)))

# Print the mapping
print("Label Mapping:")
print(label_mapping)

original_labels = le.inverse_transform([0, 1, 2])
print("Original Labels:")
print(original_labels)
```

```
Label Mapping:
{'Improve': 0, 'No effect': 1, 'Worsen': 2}
Original Labels:
['Improve' 'No effect' 'Worsen']
```

In [46]:
```
unique_values_service = df['Primary streaming service'].unique()
unique_values_service
```

Out[46]:
```
array(['Spotify', 'Pandora', 'YouTube', 'None', 'Apple Music', 'Other'],
      dtype=object)
```

In [47]:
```
unique_values_genre = df['Fav genre'].unique()
unique_values_genre
```

Out[47]:
```
array(['Latin', 'Rock', 'Video game', 'Jazz', 'R&B', 'K pop', 'Country',
       'EDM', 'Hip hop', 'Pop', 'Rap', 'Classical', 'Metal', 'Folk',
       'Lofi', 'Gospel'], dtype=object)
```

In [48]:
```
unique_values_frequency= df['Frequency [R&B]'].unique()
unique_values_frequency
```

Out[48]:
```
array(['Sometimes', 'Never', 'Very frequently', 'Rarely'], dtype=object)
```

In [49]:
```
# Mapping for categorical variables
service_mapping = {"Spotify": 0, "Pandora": 1, "YouTube": 2, "Apple Music":
binary_mapping = {"Yes": 1, "No": 0}
genre_mapping = {"Classical": 0, "Country": 1, "EDM": 2, "Folk": 3, "Gospel'
                 "Latin": 8, "Lofi": 9, "Metal": 10, "Pop": 11, "R&B": 12, '
frequency_mapping = {"Never": 0, "Rarely": 1, "Sometimes": 2, "Very frequent


# Mapping for specific columns
column_mappings = {
    "Primary streaming service": service_mapping,
    "While working": binary_mapping,
    "Instrumentalist": binary_mapping,
    "Composer": binary_mapping,
    "Exploratory": binary_mapping,
    "Foreign languages": binary_mapping,
    "Fav genre": genre_mapping,
    "Frequency [Classical]": frequency_mapping,
    "Frequency [Country]": frequency_mapping,
    "Frequency [EDM]": frequency_mapping,
    "Frequency [Folk]": frequency_mapping,
    "Frequency [Gospel]": frequency_mapping,
    "Frequency [Hip hop]": frequency_mapping,
    "Frequency [Jazz]": frequency_mapping,
    "Frequency [K pop]": frequency_mapping,
    "Frequency [Latin]": frequency_mapping,
    "Frequency [Lofi]": frequency_mapping,
    "Frequency [Metal]": frequency_mapping,
    "Frequency [Pop]": frequency_mapping,
    "Frequency [R&B]": frequency_mapping,
    "Frequency [Rap]": frequency_mapping,
    "Frequency [Rock]": frequency_mapping,
    "Frequency [Video game music]": frequency_mapping,

}

# Apply mappings to the dataset
df = df.replace(column_mappings)
```

In [50]:
```
df
```

Out[50]:

	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Exploratory	Fore langua
0	18.0	0	3.0	1	1	1	8	1	
1	43.0	1	1.5	1	0	0	14	1	
2	18.0	0	4.0	0	0	0	15	0	
3	43.0	2	2.5	1	0	1	6	1	
4	18.0	0	4.0	1	0	0	12	1	
...	...	...	...	...	...	...	...	...	
731	17.0	0	2.0	1	1	0	14	1	
732	18.0	0	1.0	1	1	0	11	1	
733	19.0	4	6.0	1	0	1	13	1	
734	19.0	0	5.0	1	1	0	0	0	
735	29.0	2	2.0	1	0	0	5	1	

736 rows × 30 columns

determine the variable that we want to investigate

In [51]:
```
df.corr()
```

Out[51]:

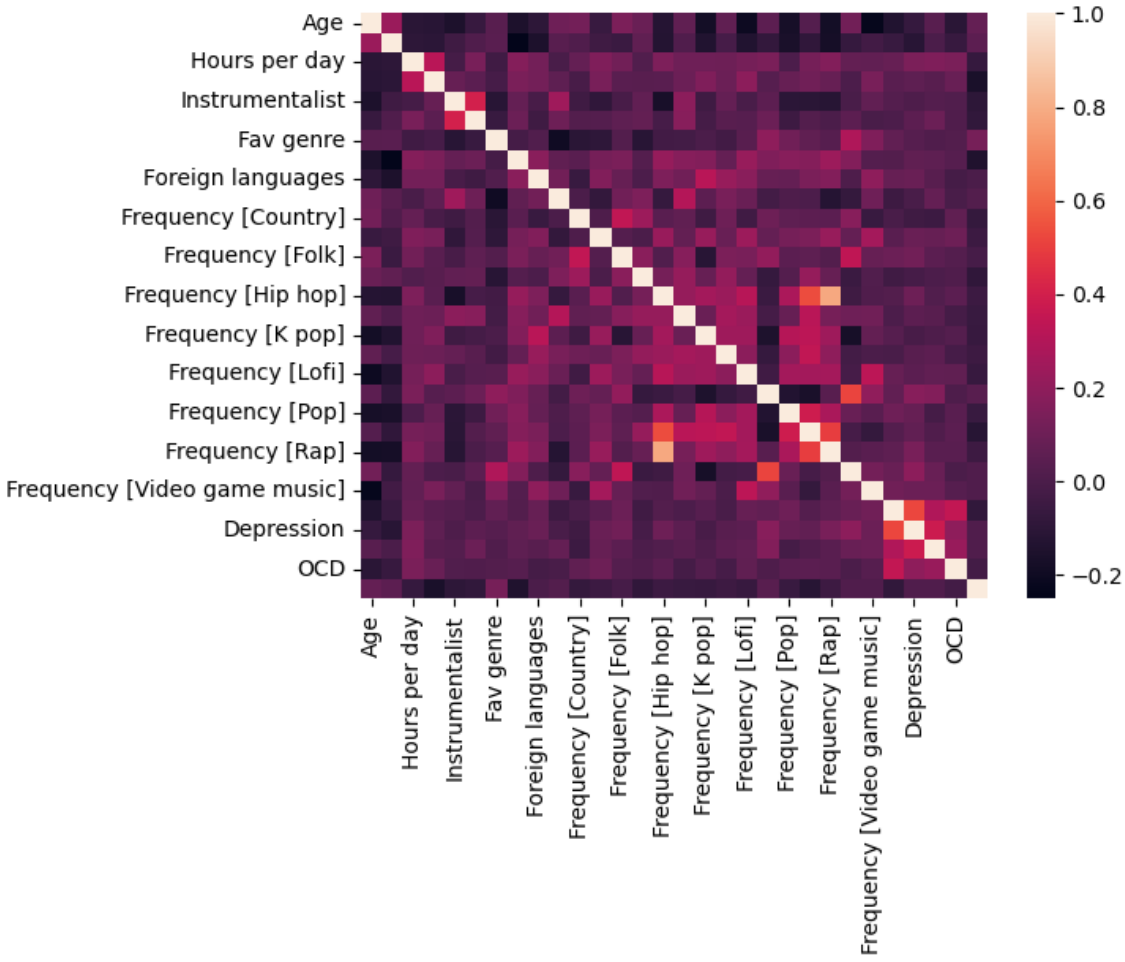
	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	F ger
Age	1.000000	0.230865	-0.112817	-0.116484	-0.156019	-0.064988	0.0348
Primary streaming service	0.230865	1.000000	-0.103071	-0.104266	-0.040526	0.013713	0.0417
Hours per day	-0.112817	-0.103071	1.000000	0.320574	-0.024016	0.135278	-0.0376
While working	-0.116484	-0.104266	0.320574	1.000000	0.080039	0.047567	-0.0177
Instrumentalist	-0.156019	-0.040526	-0.024016	0.080039	1.000000	0.400539	-0.1169
Composer	-0.064988	0.013713	0.135278	0.047567	0.400539	1.000000	-0.0615
Fav genre	0.034870	0.041773	-0.037677	-0.017798	-0.116912	-0.061589	1.0000
Exploratory	-0.156634	-0.249629	0.165419	0.139238	0.073282	0.096692	-0.0227
Foreign languages	-0.100835	-0.152661	0.117253	0.116729	-0.013320	-0.042438	0.0158
Frequency [Classical]	0.102223	0.035467	-0.009346	0.053030	0.245286	0.097554	-0.1968
Frequency [Country]	0.120277	0.010108	0.076452	-0.014630	-0.041773	0.023240	-0.1115
Frequency [EDM]	-0.063576	-0.040997	0.150655	0.135619	-0.090165	0.024398	-0.0953
Frequency [Folk]	0.142368	-0.056414	0.115333	0.042661	-0.014081	0.055280	0.0220
Frequency [Gospel]	0.088189	0.059943	0.016551	0.037146	0.063516	0.071488	-0.1132
Frequency [Hip hop]	-0.123495	-0.130940	0.147704	0.039577	-0.167813	-0.015565	-0.0327
Frequency [Jazz]	0.058921	0.012573	0.104831	0.093253	0.185238	0.179121	-0.0289
Frequency [K pop]	-0.182727	-0.138469	0.106270	0.153965	-0.039947	-0.020214	-0.0023
Frequency [Latin]	0.062461	-0.019426	0.099934	0.094825	0.075132	0.035381	-0.0330
Frequency [Lofi]	-0.209340	-0.141029	0.122447	0.193356	-0.005574	0.033149	0.0347
Frequency [Metal]	0.048675	-0.083067	0.138896	0.029153	0.049539	0.093096	0.1966
Frequency [Pop]	-0.174958	-0.162251	-0.000093	0.077994	-0.107145	-0.046093	0.1112
Frequency [R&B]	0.026167	-0.092492	0.124946	0.110416	-0.110172	0.021490	0.0590
Frequency [Rap]	-0.183473	-0.180439	0.163999	0.081465	-0.118637	-0.004403	0.0321
Frequency [Rock]	0.113664	-0.072881	0.070947	-0.000386	-0.010160	0.049197	0.2911
Frequency [Video game music]	-0.231361	-0.031379	0.064726	0.132567	0.063276	-0.008421	0.1543
Anxiety	-0.141911	-0.063767	0.080179	0.039487	0.027310	0.003176	0.0658

	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	F ger
Depression	-0.071882	-0.114605	0.141736	0.055855	0.006157	0.050861	0.0327
Insomnia	0.033543	-0.003179	0.153389	0.037034	0.025460	0.094605	0.0102
OCD	-0.110005	-0.064353	0.136029	0.083072	0.014111	0.012331	0.0198
Music effects	0.074917	0.050854	-0.073140	-0.165183	-0.098826	-0.085482	0.1299

30 rows × 30 columns

In [52]:
```
sns.heatmap(df.corr())
```

Out[52]: <AxesSubplot: >



In [53]:
```
X = df.drop(['Music effects'], axis=1)
y = df['Music effects']
```

In [54]:
```
## Apply SMOTE to solve imbalancing data
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

In [55]:
```
X_train,X_test,y_train,y_test = train_test_split(X_resampled,y_resampled,tes
```

determine need to scaling or not

In [56]:
```
df.describe()
```

Out[56]:

	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genr
count	736.000000	736.000000	736.000000	736.000000	736.000000	736.000000	736.00000
mean	23.925553	1.232337	3.380503	0.790761	0.319293	0.171196	9.42391
std	8.421012	1.770226	2.376208	0.407042	0.466520	0.376936	4.82985
min	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	18.000000	0.000000	2.000000	1.000000	0.000000	0.000000	5.00000
50%	21.000000	0.000000	3.000000	1.000000	0.000000	0.000000	11.00000
75%	28.000000	2.000000	5.000000	1.000000	1.000000	0.000000	14.00000
max	43.000000	5.000000	9.500000	1.000000	1.000000	1.000000	15.00000

8 rows × 30 columns

In [57]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [58]:
```
#!pip install lazypredict
```

In [59]:
```
from lazypredict.Supervised import LazyClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

In [60]:
```
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(X_train_scaled, X_test_scaled, y_train, y_test)

print(models)
```

```
 97%|█████████▋ | 28/29 [00:02<00:00, 14.06it/s]

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000749 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1649
[LightGBM] [Info] Number of data points in the train set: 1155, number of
used features: 29
[LightGBM] [Info] Start training from score -1.114319
[LightGBM] [Info] Start training from score -1.065404
[LightGBM] [Info] Start training from score -1.116961

100%|██████████| 29/29 [00:02<00:00,  9.71it/s]
```

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
LGBMClassifier	0.90	0.90	None	0.90	0.55
XGBClassifier	0.89	0.89	None	0.89	
ExtraTreesClassifier	0.88	0.88	None	0.88	
RandomForestClassifier	0.88	0.88	None	0.88	
SVC	0.87	0.87	None	0.87	
NuSVC	0.85	0.85	None	0.85	
QuadraticDiscriminantAnalysis	0.85	0.85	None	0.85	
BaggingClassifier	0.85	0.85	None	0.85	
LabelPropagation	0.83	0.83	None	0.82	
LabelSpreading	0.83	0.83	None	0.82	
KNeighborsClassifier	0.78	0.78	None	0.77	
DecisionTreeClassifier	0.77	0.77	None	0.77	
AdaBoostClassifier	0.77	0.76	None	0.77	
ExtraTreeClassifier	0.74	0.74	None	0.74	
LogisticRegression	0.74	0.74	None	0.74	
CalibratedClassifierCV	0.74	0.73	None	0.73	
LinearSVC	0.74	0.73	None	0.73	
SGDClassifier	0.72	0.71	None	0.71	
RidgeClassifierCV	0.71	0.71	None	0.70	
LinearDiscriminantAnalysis	0.71	0.71	None	0.70	
RidgeClassifier	0.71	0.70	None	0.70	
GaussianNB	0.67	0.66	None	0.65	
PassiveAggressiveClassifier	0.64	0.64	None	0.64	
Perceptron	0.62	0.62	None	0.63	
NearestCentroid	0.61	0.61	None	0.61	
BernoulliNB	0.59	0.59	None	0.60	
DummyClassifier	0.31	0.33	None	0.14	

```
XGBClassifier                    0.19
ExtraTreesClassifier             0.17
RandomForestClassifier           0.22
SVC                              0.08
NuSVC                            0.17
QuadraticDiscriminantAnalysis    0.01
BaggingClassifier                0.06
LabelPropagation                 0.13
LabelSpreading                   0.14
KNeighborsClassifier             0.05
DecisionTreeClassifier           0.02
AdaBoostClassifier               0.10
ExtraTreeClassifier              0.01
LogisticRegression               0.05
CalibratedClassifierCV           0.63
LinearSVC                        0.21
SGDClassifier                    0.02
RidgeClassifierCV                0.02
LinearDiscriminantAnalysis       0.03
RidgeClassifier                  0.01
GaussianNB                       0.01
PassiveAggressiveClassifier      0.02
Perceptron                       0.02
NearestCentroid                  0.02
BernoulliNB                      0.01
DummyClassifier                  0.01
```

In [61]:
```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import RidgeClassifier, LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import classification_report
import time
```

In [62]:
```
from lightgbm import LGBMClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
```

In [63]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
```

In [64]:
```
lgbm_classifier_best = LGBMClassifier(random_state=42)


# Train the model
lgbm_classifier_best.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_lgbm = lgbm_classifier_best.predict(X_test_scaled)


# Evaluate the model
accuracy_lgbm = accuracy_score(y_test, y_pred_lgbm)
print(f"Accuracy for LightGBM Classifier: {accuracy_lgbm:.2f}")


precision_lgbm = precision_score(y_test, y_pred_lgbm,average='weighted')
recall_lgbm = recall_score(y_test, y_pred_lgbm,average='weighted')
f1_lgbm = f1_score(y_test, y_pred_lgbm,average='weighted')

# Print classification report
print("Classification Report for LightGBM Classifier:")
print(classification_report(y_test, y_pred_lgbm))
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of
testing was 0.000261 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1649
[LightGBM] [Info] Number of data points in the train set: 1155, number of
used features: 29
[LightGBM] [Info] Start training from score -1.114319
[LightGBM] [Info] Start training from score -1.065404
[LightGBM] [Info] Start training from score -1.116961
Accuracy for LightGBM Classifier: 0.90
Classification Report for LightGBM Classifier:
              precision    recall  f1-score   support

           0       0.88      0.83      0.85       171
           1       0.82      0.86      0.84       152
           2       0.99      0.99      0.99       172

    accuracy                           0.90       495
   macro avg       0.89      0.90      0.89       495
weighted avg       0.90      0.90      0.90       495
```

In [65]:
```
import xgboost as xgb
from sklearn.model_selection import GridSearchCV

# Initialize the XGBoost Classifier
xgb_classifier = xgb.XGBClassifier(random_state=42)

# Define the parameter grid to search
param_grid_xgb = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'n_estimators': [50, 100, 150],
}

# Initialize GridSearchCV
grid_search_xgb = GridSearchCV(estimator=xgb_classifier, param_grid=param_gr

# Perform GridSearchCV on the training data
grid_search_xgb.fit(X_train_scaled, y_train)

# Get the best parameters and model
best_params_xgb = grid_search_xgb.best_params_
print("Best Parameters for XGBoost Classifier:", best_params_xgb)
```

```
Best Parameters for XGBoost Classifier: {'learning_rate': 0.2, 'max_dept
h': 7, 'n_estimators': 150}
```

In [66]:
```
xgb_classifier_best = xgb.XGBClassifier(learning_rate=0.2, max_depth=7, n_es

xgb_classifier_best.fit(X_train_scaled, y_train)
# Make predictions on the test set using the best model
y_pred_xgb = xgb_classifier_best.predict(X_test_scaled)

# Evaluate the best model
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
print(f"Best Model Accuracy for XGBoost Classifier: {accuracy_xgb:.2f}")

precision_xgb = precision_score(y_test, y_pred_xgb,average='weighted')
recall_xgb = recall_score(y_test, y_pred_xgb,average='weighted')
f1_xgb = f1_score(y_test, y_pred_xgb,average='weighted')


# Print classification report
print("Classification Report for XGBoost Classifier:")
print(classification_report(y_test, y_pred_xgb))
```

```
Best Model Accuracy for XGBoost Classifier: 0.89
Classification Report for XGBoost Classifier:
              precision    recall  f1-score   support

           0       0.90      0.81      0.85       171
           1       0.81      0.88      0.85       152
           2       0.97      0.99      0.98       172

    accuracy                           0.89       495
   macro avg       0.89      0.89      0.89       495
weighted avg       0.90      0.89      0.89       495
```

In [67]:
```
from sklearn.ensemble import ExtraTreesClassifier

# Initialize the Extra Trees Classifier
et_classifier = ExtraTreesClassifier(random_state=42)

param_grid_et = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

grid_search_et = GridSearchCV(estimator=et_classifier, param_grid=param_gri
grid_search_et.fit(X_train_scaled, y_train)

best_params_et = grid_search_et.best_params_
print("Best Parameters for Extra Trees Classifier:", best_params_et)
```

```
Best Parameters for Extra Trees Classifier: {'max_depth': 20, 'min_samples
_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150}
```

In [68]:
```
et_classifier_best = ExtraTreesClassifier(max_depth=20, min_samples_leaf=1,

et_classifier_best.fit(X_train_scaled, y_train)

# Make predictions on the test set using the best model
y_pred_et = et_classifier_best.predict(X_test_scaled)

# Evaluate the best model
accuracy_et = accuracy_score(y_test, y_pred_et)
print(f"Best Model Accuracy for Extra Trees Classifier: {accuracy_et:.2f}")

precision_et = precision_score(y_test, y_pred_et,average='weighted')
recall_et = recall_score(y_test, y_pred_et,average='weighted')
f1_et = f1_score(y_test, y_pred_et,average='weighted')


# Print classification report
print("Classification Report for Extra Trees Classifier:")
print(classification_report(y_test, y_pred_et))
```

```
Best Model Accuracy for Extra Trees Classifier: 0.89
Classification Report for Extra Trees Classifier:
              precision    recall  f1-score   support

           0       0.87      0.80      0.83       171
           1       0.79      0.87      0.83       152
           2       0.99      0.99      0.99       172

    accuracy                           0.89       495
   macro avg       0.89      0.89      0.88       495
weighted avg       0.89      0.89      0.89       495
```

In [69]:
```
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid to search
param_grid_rf = {
    'n_estimators': [50, 100, 150],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Initialize RandomForestClassifier
rf_classifier = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search_rf = GridSearchCV(estimator=rf_classifier, param_grid=param_gri

# Perform GridSearchCV on the training data
grid_search_rf.fit(X_train_scaled, y_train)

# Get the best parameters and model
best_params_rf = grid_search_rf.best_params_
print("Best Parameters for Random Forest Classifier:", best_params_rf)
```

```
Best Parameters for Random Forest Classifier: {'max_depth': 20, 'max_featu
res': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimator
s': 150}
```

In [70]:
```
rf_classifier_best = RandomForestClassifier(max_depth=20, max_features='log2
#rf_classifier_best = RandomForestClassifier(max_depth=None, max_features='l
# Train the model
rf_classifier_best.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_rf = rf_classifier_best.predict(X_test_scaled)

# Evaluate the best model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Best Model Accuracy for Random Forest Classifier: {accuracy_rf:.2f}"

precision_rf  = precision_score(y_test, y_pred_rf,average='weighted')
recall_rf  = recall_score(y_test, y_pred_rf,average='weighted')
f1_rf  = f1_score(y_test, y_pred_rf,average='weighted')



# Print classification report
print("Classification Report for Random Forest Classifier:")
print(classification_report(y_test, y_pred_rf))
```

```
Best Model Accuracy for Random Forest Classifier: 0.88
Classification Report for Random Forest Classifier:
              precision    recall  f1-score   support

           0       0.88      0.78      0.83       171
           1       0.78      0.87      0.82       152
           2       0.99      0.99      0.99       172

    accuracy                           0.88       495
   macro avg       0.88      0.88      0.88       495
weighted avg       0.89      0.88      0.88       495
```

In [71]:
```
from sklearn.svm import SVC

param_grid_svc = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf', 'poly'],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3, 4]
}

# Initialize SVC
svc_classifier = SVC(random_state=42)

# Initialize GridSearchCV
grid_search_svc = GridSearchCV(estimator=svc_classifier, param_grid=param_gr

# Perform GridSearchCV on the training data
grid_search_svc.fit(X_train_scaled, y_train)

# Get the best parameters
best_params_svc = grid_search_svc.best_params_
print("Best Parameters for SVC:", best_params_svc)
```

```
Best Parameters for SVC: {'C': 10, 'degree': 2, 'gamma': 'scale', 'kerne
l': 'rbf'}
```

In [72]:
```
svc_classifier_best = SVC(C=10, degree=2, gamma='scale', kernel='rbf',randor

# Train the model
svc_classifier_best.fit(X_train_scaled, y_train)

# Make predictions on the test set
y_pred_svc = svc_classifier_best.predict(X_test_scaled)

# Evaluate the best model
accuracy_svc = accuracy_score(y_test, y_pred_svc)
print(f"Best Model Accuracy for SVC: {accuracy_svc:.2f}")

precision_svc  = precision_score(y_test, y_pred_svc,average='weighted')
recall_svc  = recall_score(y_test, y_pred_svc, average='weighted')
f1_svc  = f1_score(y_test, y_pred_svc, average='weighted')




# Print classification report
print("Classification Report for SVC:")
print(classification_report(y_test, y_pred_svc))
```

```
Best Model Accuracy for SVC: 0.87
Classification Report for SVC:
              precision    recall  f1-score   support

           0       0.88      0.75      0.81       171
           1       0.76      0.88      0.82       152
           2       0.98      0.99      0.99       172

    accuracy                           0.87       495
   macro avg       0.87      0.87      0.87       495
weighted avg       0.88      0.87      0.87       495
```

In [73]:
```
import pandas as pd

models = [
    'LightGBM Classifier',
    'XGBoost Classifier',
    'Extra Trees Classifier',
    'Random Forest Classifier',
    'SVC'
]

accuracy_scores = [
    accuracy_lgbm,
    accuracy_xgb,
    accuracy_et,
    accuracy_rf,
    accuracy_svc
]

precision_scores = [
    precision_lgbm,
    precision_xgb,
    precision_et,
    precision_rf,
    precision_svc
]

recall_scores = [
    recall_lgbm,
    recall_xgb,
    recall_et,
    recall_rf,
    recall_svc
]

f1_scores = [
    f1_lgbm,
    f1_xgb,
    f1_et,
    f1_rf,
    f1_svc
]

# Create a DataFrame
results_df = pd.DataFrame({
    'Model Name': models,
    'Accuracy': accuracy_scores,
    'Precision': precision_scores,
    'Recall': recall_scores,
    'F1': f1_scores
})

# Set the display precision to 4 decimal places
pd.set_option('display.float_format', '{:.4f}'.format)

# Sort the DataFrame by 'Accuracy' in descending order
results_df = results_df.sort_values(by='Accuracy', ascending=False)

# Set the 'Model Name' column as the index
results_df = results_df.set_index('Model Name')

# Print the sorted and formatted results DataFrame
```

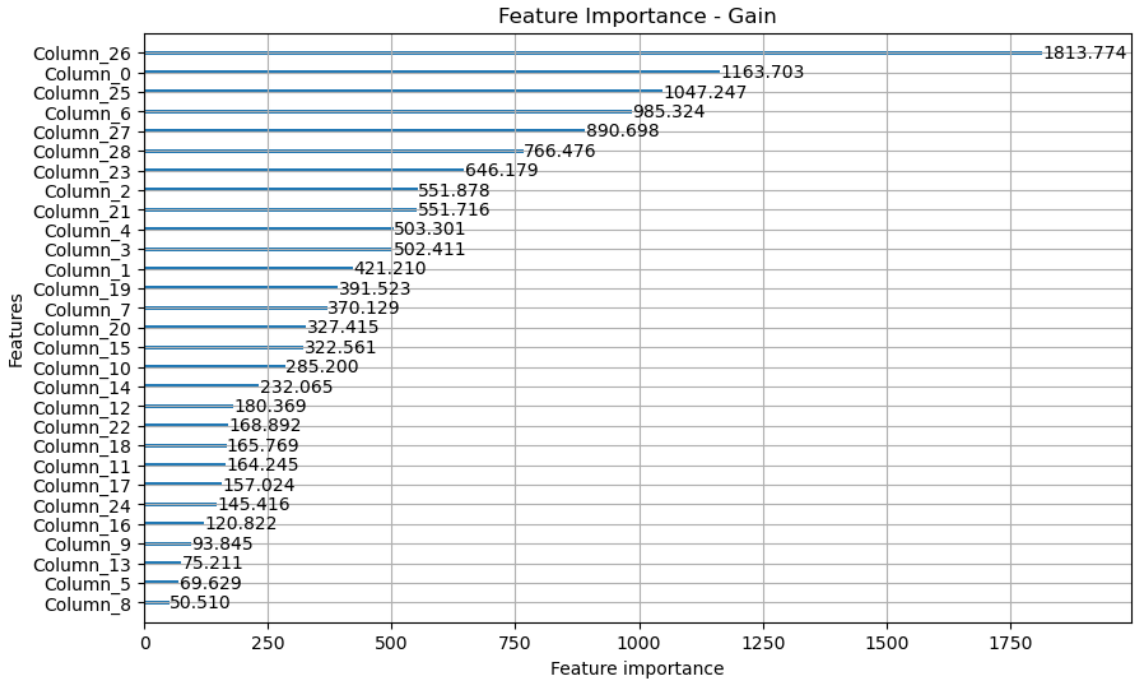
```
results_df
```

Out[73]:

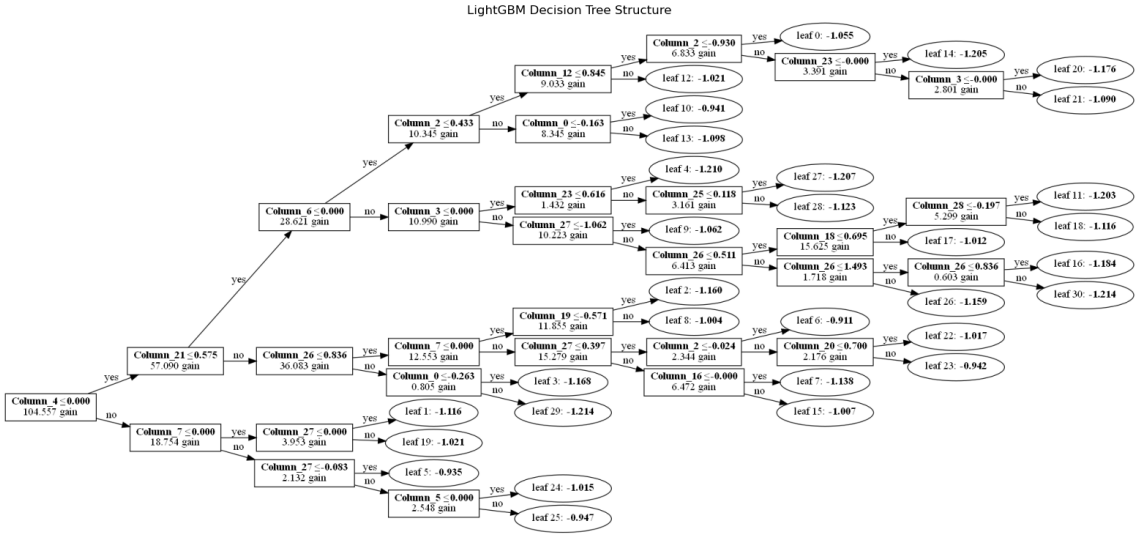
Model Name	Accuracy	Precision	Recall	F1
LightGBM Classifier	0.8970	0.8977	0.8970	0.8969
XGBoost Classifier	0.8949	0.8965	0.8949	0.8945
Extra Trees Classifier	0.8869	0.8893	0.8869	0.8869
Random Forest Classifier	0.8828	0.8864	0.8828	0.8828
SVC	0.8747	0.8794	0.8747	0.8743

In [74]:

```
import lightgbm as lgb
import matplotlib.pyplot as plt



# Plot feature importance
lgb.plot_importance(lgbm_classifier_best, figsize=(10, 6), importance_type=
plt.title('Feature Importance - Gain')
plt.show()
```



In [75]:
```
import lightgbm as lgb
import matplotlib.pyplot as plt

lgb.plot_tree(lgbm_classifier_best, tree_index=0, figsize=(20, 10), show_int
plt.title('LightGBM Decision Tree Structure')
plt.show()
```



While working, Instrumentalist, Composer, Exploratory, Foreign languages :
"Yes": 1, "No": 0

Fav genre :
"Classical": 0, "Country": 1, "EDM": 2, "Folk": 3, "Gospel": 4, "Hip hop": 5, "Jazz": 6, "K pop": 7, "Latin": 8, "Lofi": 9, "Metal": 10, "Pop": 11, "R&B": 12, "Rap": 13, "Rock": 14, "Video game music": 15

Frequency [Classical] , Frequency [Country], Frequency [EDM], Frequency [Folk], Frequency [Gospel], Frequency [Hip hop], Frequency [Jazz], Frequency [K pop], Frequency [Latin], Frequency [Lofi], Frequency [Metal], Frequency [Pop], Frequency [R&B], Frequency [Rap], Frequency [Rock], Frequency [Video game music] :
"Never": 0, "Rarely": 1, "Sometimes": 2, "Very frequently": 3

Primary streaming service :
"Spotify": 0, "Pandora": 1, "YouTube Music": 2, "Apple Music": 3, "Other streaming service": 4, "I do not use a streaming service.": 5

Extra Trees Model

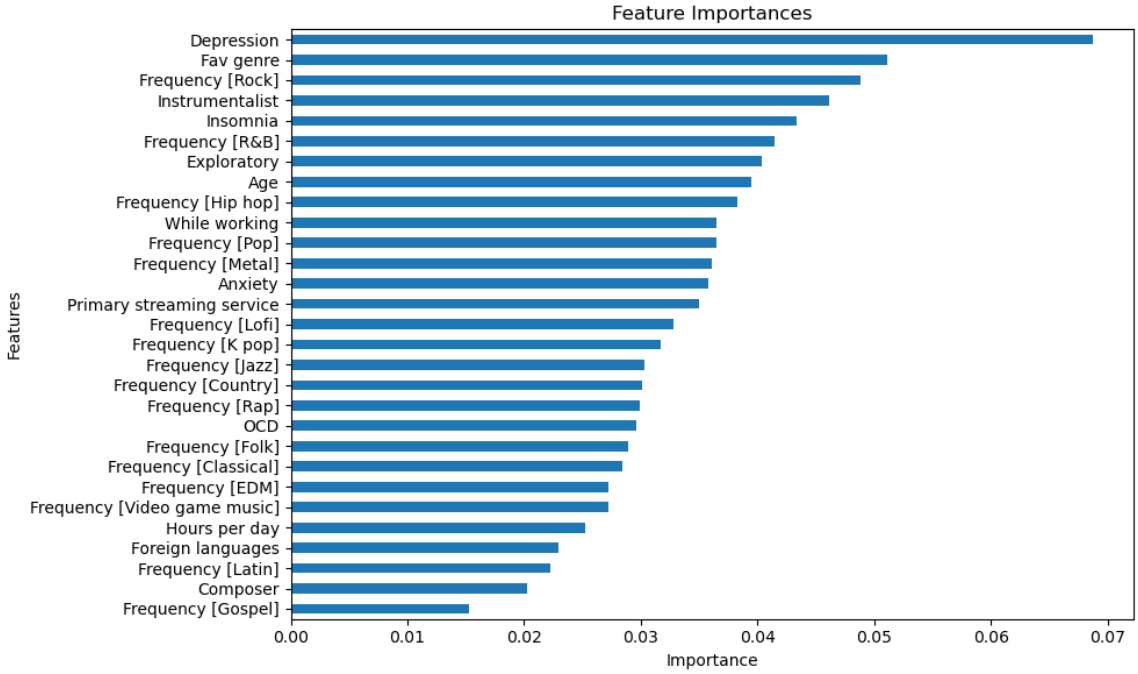
In [76]:
```
import tkinter as tk
from tkinter import messagebox
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_score, f

class ExtraTreesGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Extra Trees Model GUI")


        label_encoder = LabelEncoder()
        for column in df.columns:
            if df[column].dtype == 'object':
                df[column] = label_encoder.fit_transform(df[column].astype(s

        # Split the data into features and target variable
        X = df.drop('Music effects', axis=1)
        y = df['Music effects']

        # Apply SMOTE to handle class imbalance
        smote = SMOTE(random_state=42)
        X_resampled, y_resampled = smote.fit_resample(X, y)

        # Split the resampled data into training and testing sets
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_sp

        # Create and train the Extra Trees model
        self.model = ExtraTreesClassifier(n_estimators=150, max_depth=20, mi
        self.model.fit(self.X_train, self.y_train)

        # Create GUI components
        self.create_gui()

    def create_gui(self):
        # Create a canvas widget and a vertical scrollbar
        canvas = tk.Canvas(self.root)
        scrollbar = tk.Scrollbar(self.root, orient="vertical", command=canva
        # Create a frame inside the canvas and attach the scrollbar to the c
        scrollable_frame = tk.Frame(canvas)
        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)
        # Bind the configure event of the canvas to a function that updates
        canvas.bind("<Configure>", lambda e: canvas.configure(scrollregion=c
        # Pack the canvas and the scrollbar
        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        # Create entry fields for each feature in the scrollable frame
        for column in self.X_train.columns:
            frame = tk.Frame(scrollable_frame)
            frame.pack(side="top", fill="x", padx=10, pady=5)

            label = tk.Label(frame, text=column)
            label.pack(side="left")

            entry = tk.Entry(frame)
            entry.pack(side="left", padx=5)
```

```
            # Save entry in an instance variable for later access
            setattr(self, f"{column}_entry", entry)

        # Create predict button in the scrollable frame
        self.predict_button = tk.Button(scrollable_frame, text="Predict", co
        self.predict_button.pack(pady=10)

        # Result label in the scrollable frame
        self.result_label = tk.Label(scrollable_frame, text="")
        self.result_label.pack()

    def predict(self):
        # Get user input from entry fields
        user_input = [float(getattr(self, f"{column}_entry").get()) for colu

        # Make predictions using the trained model
        prediction = self.model.predict([user_input])[0]

        # Display the prediction
        self.result_label.config(text=f"The predicted music effect is: {pred


if __name__ == "__main__":
    root = tk.Tk()
    app = ExtraTreesGUI(root)
    root.mainloop()
```

Label Mapping: {'Improve': 0, 'No effect': 1, 'Worsen': 2} Original Labels: ['Improve' 'No effect' 'Worsen']

In [77]:
```
et_classifier_best = ExtraTreesClassifier(max_depth=20, min_samples_leaf=1,
et_classifier_best.fit(X_train_scaled, y_train)

# Make predictions on the test set using the best model
y_pred_et = et_classifier_best.predict(X_test_scaled)

# Evaluate the best model
accuracy_et = accuracy_score(y_test, y_pred_et)
print(f"Best Model Accuracy for Extra Trees Classifier: {accuracy_et:.2f}")

precision_et = precision_score(y_test, y_pred_et, average='weighted')
recall_et = recall_score(y_test, y_pred_et, average='weighted')
f1_et = f1_score(y_test, y_pred_et, average='weighted')

# Print classification report
print("Classification Report for Extra Trees Classifier:")
print(classification_report(y_test, y_pred_et))

# Plotting feature importances as a vertical bar chart
feature_importances = pd.Series(et_classifier_best.feature_importances_, inc
sorted_importances = feature_importances.sort_values(ascending=False)

plt.figure(figsize=(10, 6))
sorted_importances.plot(kind='barh')
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.gca().invert_yaxis()
plt.tight_layout()
plt.show()
```

```
Best Model Accuracy for Extra Trees Classifier: 0.89
Classification Report for Extra Trees Classifier:
              precision    recall  f1-score   support

           0       0.87      0.80      0.83       171
           1       0.79      0.87      0.83       152
           2       0.99      0.99      0.99       172

    accuracy                           0.89       495
   macro avg       0.89      0.89      0.88       495
weighted avg       0.89      0.89      0.89       495
```


Feature Importances
Depression
Fav genre
Frequency [Rock]
Instrumentalist
Insomnia
Frequency [R&B]
Exploratory
Age
Frequency [Hip hop]
While working
Frequency [Pop]
Frequency [Metal]
Anxiety
Primary streaming service
Frequency [Lofi]
Frequency [K pop]
Frequency [Jazz]
Frequency [Country]
Frequency [Rap]
OCD
Frequency [Folk]
Frequency [Classical]
Frequency [EDM]
Frequency [Video game music]
Hours per day
Foreign languages
Frequency [Latin]
Composer
Frequency [Gospel]
Features
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
Importance

In [ ]: