In [ ]:
```
# Importing Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# increase the maximum number of rows and columns displayed
pd.set_option('display.max_rows', 10000)
pd.set_option('display.max_columns', 1000)

# Load the data in a dataframe
Data=pd.read_csv('mxmh_survey_results.csv')
```

Data Exploration and Preparation

In [ ]:
```
# Print Dataframe
Data.head()
```

Out[ ]:

	Timestamp	Age	Primary streaming service	Hours per day	While working	Instrumentalist	Composer	Fav genre	Exploratory	Foreign languages	BPM	Frequency [Classical]	Frequency [Country]	Frequency [EDM]	Frec
0	8/27/2022 19:29:02	18.0	Spotify	3.0	Yes	Yes	Yes	Latin	Yes	Yes	156.0	Rarely	Never	Rarely	
1	8/27/2022 19:57:31	63.0	Pandora	1.5	Yes	No	No	Rock	Yes	No	119.0	Sometimes	Never	Never	
2	8/27/2022 21:28:18	18.0	Spotify	4.0	No	No	No	Video game music	No	Yes	132.0	Never	Never	Very frequently	
3	8/27/2022 21:40:40	61.0	YouTube Music	2.5	Yes	No	Yes	Jazz	Yes	Yes	84.0	Sometimes	Never	Never	
4	8/27/2022 21:54:47	18.0	Spotify	4.0	Yes	No	No	R&B	Yes	No	107.0	Never	Never	Rarely	

In [ ]:
```
# Look at the columns
Data.columns
```

Out[ ]:
```
Index(['Timestamp', 'Age', 'Primary streaming service', 'Hours per day',
       'While working', 'Instrumentalist', 'Composer', 'Fav genre',
       'Exploratory', 'Foreign languages', 'BPM', '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]',
       'Anxiety', 'Depression', 'Insomnia', 'OCD', 'Music effects',
       'Permissions'],
      dtype='object')
```

In [ ]:
```
Data.describe()
```

Out[ ]:

	Age	Hours per day	BPM	Anxiety	Depression	Insomnia	OCD
count	735.000000	736.000000	6.290000e+02	736.000000	736.000000	736.000000	736.000000
mean	25.206803	3.572758	1.589948e+06	5.837636	4.796196	3.738451	2.637228
std	12.054970	3.028199	3.987261e+07	2.793054	3.028870	3.088689	2.842017
min	10.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000
25%	18.000000	2.000000	1.000000e+02	4.000000	2.000000	1.000000	0.000000
50%	21.000000	3.000000	1.200000e+02	6.000000	5.000000	3.000000	2.000000
75%	28.000000	5.000000	1.440000e+02	8.000000	7.000000	6.000000	5.000000
max	89.000000	24.000000	1.000000e+09	10.000000	10.000000	10.000000	10.000000

In [ ]:
```
# check if the data is balanced
Data['Music effects'].value_counts(normalize=True).round(2)
```

Out[ ]:
```
Improve      0.74
No effect    0.23
Worsen       0.02
Name: Music effects, dtype: float64
```

In [ ]:
```
# No. of rows present within the data
len(Data)
```

Out[ ]:
```
736
```

In [ ]:
```
# Null values check
Data.isnull().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
# Print out all columns with missing values
for i in Data.columns:
    if Data[i].isna().sum()>0:
        print(i,'(',Data[i].dtype,')',':',Data[i].isna().sum())
```

```
Age ( float64 ) : 1
Primary streaming service ( object ) : 1
While working ( object ) : 3
Instrumentalist ( object ) : 4
Composer ( object ) : 1
Foreign languages ( object ) : 4
BPM ( float64 ) : 107
Music effects ( object ) : 8
```

In [ ]:
```
# Let's apply mean imputation for BPM since the % of missing values is high
Data['BPM'] = Data['BPM'].fillna(Data['BPM'].mean())

# Replacing nan with 0 for AGE
Data['Age']=Data['Age'].fillna(0)

# Run it again : Print out all columns with missing values
for i in Data.columns:
    if Data[i].isna().sum()>0:
        print(i,'(',Data[i].dtype,')',':',Data[i].isna().sum())
```

```
Primary streaming service ( object ) : 1
While working ( object ) : 3
Instrumentalist ( object ) : 4
Composer ( object ) : 1
Foreign languages ( object ) : 4
Music effects ( object ) : 8
```

In [ ]:
```
# Since the % of null values ain't significant, I will go ahead and drop them

Data=Data.dropna()

#Final check for null values
Data.isna().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
# From 736 rows to 719 rows after dropping na
len(Data)
```

Out[ ]:
```
719
```

In [ ]:
```
Data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 719 entries, 2 to 735
Data columns (total 33 columns):
 #   Column                         Non-Null Count  Dtype  
---  ------                         --------------  -----  
 0   Timestamp                      719 non-null    object 
 1   Age                            719 non-null    float64
 2   Primary streaming service      719 non-null    object 
 3   Hours per day                  719 non-null    float64
 4   While working                  719 non-null    object 
 5   Instrumentalist                719 non-null    object 
 6   Composer                       719 non-null    object 
 7   Fav genre                      719 non-null    object 
 8   Exploratory                    719 non-null    object 
 9   Foreign languages              719 non-null    object 
 10  BPM                            719 non-null    float64
 11  Frequency [Classical]          719 non-null    object 
 12  Frequency [Country]            719 non-null    object 
 13  Frequency [EDM]                719 non-null    object 
 14  Frequency [Folk]               719 non-null    object 
 15  Frequency [Gospel]             719 non-null    object 
 16  Frequency [Hip hop]            719 non-null    object 
 17  Frequency [Jazz]               719 non-null    object 
 18  Frequency [K pop]              719 non-null    object 
 19  Frequency [Latin]              719 non-null    object 
 20  Frequency [Lofi]               719 non-null    object 
 21  Frequency [Metal]              719 non-null    object 
 22  Frequency [Pop]                719 non-null    object 
 23  Frequency [R&B]                719 non-null    object 
 24  Frequency [Rap]                719 non-null    object 
 25  Frequency [Rock]               719 non-null    object 
 26  Frequency [Video game music]   719 non-null    object 
 27  Anxiety                        719 non-null    float64
 28  Depression                     719 non-null    float64
 29  Insomnia                       719 non-null    float64
 30  OCD                            719 non-null    float64
 31  Music effects                  719 non-null    object 
 32  Permissions                    719 non-null    object 
dtypes: float64(7), object(26)
memory usage: 191.0+ KB
```

In [ ]:
```
# Convert "timestamp" column to Timestamp Datatype
Data['Timestamp'] = pd.to_datetime(Data['Timestamp'])
Data['Timestamp'].dtype
```

Out[ ]:
```
dtype('<M8[ns]')
```

In [ ]:
```
Categorical_variables=[]
# Get value counts of only categorical data
for i in Data.columns:
    if Data[i].dtype==object:

        print(Data[i].value_counts())
```

```
        Categorical_variables.append(i)
        print()
```

```
Spotify                             451
YouTube Music                        90
I do not use a streaming service.    69
Apple Music                          50
Other streaming service              49
Pandora                              10
Name: Primary streaming service, dtype: int64

Yes    568
No     151
Name: While working, dtype: int64

No     490
Yes    229
Name: Instrumentalist, dtype: int64

No     595
Yes    124
Name: Composer, dtype: int64

Rock               184
Pop                114
Metal               87
Classical           51
Video game music    43
EDM                 36
R&B                 35
Hip hop             35
Folk                29
Country             24
Rap                 22
K pop               21
Jazz                20
Lofi                10
Gospel               6
Latin                2
Name: Fav genre, dtype: int64

Yes    515
No     204
Name: Exploratory, dtype: int64

Yes    396
No     323
Name: Foreign languages, dtype: int64

Rarely             254
Sometimes          193
Never              166
Very frequently    106
Name: Frequency [Classical], dtype: int64

Never              333
Rarely             230
Sometimes          108
Very frequently     48
Name: Frequency [Country], dtype: int64

Never              296
Rarely             191
Sometimes          144
Very frequently     88
Name: Frequency [EDM], dtype: int64

Never              284
Rarely             217
Sometimes          142
Very frequently     76
Name: Frequency [Folk], dtype: int64

Never              523
Rarely             132
Sometimes           50
Very frequently     14
Name: Frequency [Gospel], dtype: int64

Sometimes          214
Rarely             209
Never              174
Very frequently    122
Name: Frequency [Hip hop], dtype: int64

Never              252
Rarely             244
Sometimes          171
Very frequently     52
Name: Frequency [Jazz], dtype: int64

Never              407
Rarely             174
Very frequently     71
Sometimes           67
Name: Frequency [K pop], dtype: int64

Never              432
Rarely             171
Sometimes           85
```

```
Very frequently     31
Name: Frequency [Latin], dtype: int64

Never              273
Rarely             204
Sometimes          158
Very frequently     84
Name: Frequency [Lofi], dtype: int64

Never              256
Rarely             189
Very frequently    144
Sometimes          130
Name: Frequency [Metal], dtype: int64

Very frequently    271
Sometimes          256
Rarely             140
Never               52
Name: Frequency [Pop], dtype: int64

Never              220
Rarely             208
Sometimes          176
Very frequently    115
Name: Frequency [R&B], dtype: int64

Rarely             211
Never              193
Sometimes          190
Very frequently    125
Name: Frequency [Rap], dtype: int64

Very frequently    324
Sometimes          214
Rarely              94
Never               87
Name: Frequency [Rock], dtype: int64

Never              230
Rarely             194
Sometimes          180
Very frequently    115
Name: Frequency [Video game music], dtype: int64

Improve      536
No effect    166
Worsen        17
Name: Music effects, dtype: int64

I understand.    719
Name: Permissions, dtype: int64
```

In [ ]:
```
# Drop column called "Permission" as it gives no insights
Data = Data.drop('Permissions', axis=1)
```

In [ ]:
```
Categorical_variables.remove('Permissions')
for i in Categorical_variables:
    print(i,':',Data[i].unique())
    print(len(Data[i].unique()))
    print()
```

```
Primary streaming service : ['Spotify' 'YouTube Music' 'I do not use a streaming service.'
 'Apple Music' 'Other streaming service' 'Pandora']
6

While working : ['No' 'Yes']
2

Instrumentalist : ['No' 'Yes']
2

Composer : ['No' 'Yes']
2

Fav genre : ['Video game music' 'Jazz' 'R&B' 'K pop' 'Rock' 'Country' 'EDM' 'Hip hop'
 'Pop' 'Rap' 'Classical' 'Metal' 'Folk' 'Lofi' 'Gospel' 'Latin']
16

Exploratory : ['No' 'Yes']
2

Foreign languages : ['Yes' 'No']
2

Frequency [Classical] : ['Never' 'Sometimes' 'Rarely' 'Very frequently']
4

Frequency [Country] : ['Never' 'Sometimes' 'Very frequently' 'Rarely']
4

Frequency [EDM] : ['Very frequently' 'Never' 'Rarely' 'Sometimes']
4

Frequency [Folk] : ['Never' 'Rarely' 'Sometimes' 'Very frequently']
4

Frequency [Gospel] : ['Never' 'Sometimes' 'Rarely' 'Very frequently']
4

Frequency [Hip hop] : ['Rarely' 'Never' 'Very frequently' 'Sometimes']
4

Frequency [Jazz] : ['Rarely' 'Very frequently' 'Never' 'Sometimes']
4

Frequency [K pop] : ['Very frequently' 'Sometimes' 'Never' 'Rarely']
4

Frequency [Latin] : ['Never' 'Very frequently' 'Sometimes' 'Rarely']
4

Frequency [Lofi] : ['Sometimes' 'Very frequently' 'Rarely' 'Never']
4

Frequency [Metal] : ['Sometimes' 'Never' 'Rarely' 'Very frequently']
4

Frequency [Pop] : ['Rarely' 'Sometimes' 'Very frequently' 'Never']
4

Frequency [R&B] : ['Never' 'Sometimes' 'Very frequently' 'Rarely']
4

Frequency [Rap] : ['Rarely' 'Never' 'Very frequently' 'Sometimes']
4

Frequency [Rock] : ['Rarely' 'Never' 'Very frequently' 'Sometimes']
4

Frequency [Video game music] : ['Very frequently' 'Never' 'Rarely' 'Sometimes']
4

Music effects : ['No effect' 'Improve' 'Worsen']
3
```

In [ ]:
```python
# create dummy variables for all categorical columns
Data_clean = pd.get_dummies(Data[Categorical_variables])

# concatenate the dummy variables with the original DataFrame
Data_Final = pd.concat([Data.drop(Categorical_variables, axis=1), Data_clean], axis=1)
```

In [ ]:
```python
Data_Final.columns
```

Out[ ]:
```
Index(['Timestamp', 'Age', 'Hours per day', 'BPM', 'Anxiety', 'Depression',
       'Insomnia', 'OCD', 'Primary streaming service_Apple Music',
       'Primary streaming service_I do not use a streaming service.',
       ...
       'Frequency [Rock]_Rarely', 'Frequency [Rock]_Sometimes',
       'Frequency [Rock]_Very frequently',
       'Frequency [Video game music]_Never',
       'Frequency [Video game music]_Rarely',
       'Frequency [Video game music]_Sometimes',
       'Frequency [Video game music]_Very frequently', 'Music effects_Improve',
       'Music effects_No effect', 'Music effects_Worsen'],
      dtype='object', length=107)
```

In [ ]:
```
# The final cleant data
Data_Final.head()
```

Out[ ]:

	Timestamp	Age	Hours per day	BPM	Anxiety	Depression	Insomnia	OCD	Primary streaming service_Apple Music	Primary streaming service_I do not use a streaming service.	Primary streaming service_Other streaming service	Primary streaming service_Pandora	Primary streaming service_Spotify	servic
2	2022-08-27 21:28:18	18.0	4.0	132.0	7.0	7.0	10.0	2.0	0	0	0	0	1	
3	2022-08-27 21:40:40	61.0	2.5	84.0	9.0	7.0	3.0	3.0	0	0	0	0	0	
4	2022-08-27 21:54:47	18.0	4.0	107.0	7.0	2.0	5.0	9.0	0	0	0	0	1	
5	2022-08-27 21:56:50	18.0	5.0	86.0	8.0	8.0	7.0	7.0	0	0	0	0	1	
6	2022-08-27 22:00:29	18.0	3.0	66.0	4.0	8.0	6.0	0.0	0	0	0	0	0	

In [ ]:
```
len(Data_Final.columns)
```

Out[ ]:
```
107
```

In [ ]:
```
# Get value counts of only categorical data
for i in Data_Final.columns:
    if Data_Final[i].dtype==object:
        print(i)
```

In [ ]:
```
'''
# heatmap is very unclear because of the volumn of variables
import seaborn as sns
# Creating heatmap
corr_matrix = Data_Final.corr()
sns.heatmap(corr_matrix, annot=True, cmap="coolwarm")
plt.show()
'''
```

Out[ ]:
```
'\n# heatmap is very unclear because of the volumn of variables\nimport seaborn as sns\n# Creating heatmap\ncorr_matrix = Data_
Final.corr()\nsns.heatmap(corr_matrix, annot=True, cmap="coolwarm")\nplt.show()\n'
```

In [ ]:
```
corr_matrix[['Music effects_Improve','Music effects_No effect','Music effects_Worsen']].round(2)
```

Out[ ]:

	Music effects_Improve	Music effects_No effect	Music effects_Worsen
Age	-0.06	0.07	-0.03
Hours per day	0.03	-0.02	-0.04
BPM	-0.06	0.07	-0.01
Anxiety	0.12	-0.15	0.05
Depression	0.02	-0.07	0.12
Insomnia	0.00	-0.02	0.04
OCD	0.04	-0.05	0.03
Primary streaming service_Apple Music	-0.00	0.01	-0.01
Primary streaming service_I do not use a streaming service.	-0.07	0.07	0.01
Primary streaming service_Other streaming service	-0.02	0.04	-0.04
Primary streaming service_Pandora	0.07	-0.07	-0.02
Primary streaming service_Spotify	0.04	-0.06	0.04
Primary streaming service_YouTube Music	-0.01	0.02	-0.03
While working_No	-0.18	0.16	0.05
While working_Yes	0.18	-0.16	-0.05
Instrumentalist_No	-0.10	0.10	0.03
Instrumentalist_Yes	0.10	-0.10	-0.03
Composer_No	-0.09	0.08	0.02
Composer_Yes	0.09	-0.08	-0.02
Fav genre_Classical	-0.01	0.02	-0.01
Fav genre_Country	0.02	-0.01	-0.03
Fav genre_EDM	0.05	-0.03	-0.04
Fav genre_Folk	0.02	-0.01	-0.03
Fav genre_Gospel	0.05	-0.05	-0.01
Fav genre_Hip hop	0.07	-0.06	-0.04
Fav genre_Jazz	0.02	-0.01	-0.03
Fav genre_K pop	0.03	-0.02	-0.03
Fav genre_Latin	-0.03	0.03	-0.01
Fav genre_Lofi	0.07	-0.07	-0.02
Fav genre_Metal	0.01	0.01	-0.06
Fav genre_Pop	0.00	-0.01	0.03
Fav genre_R&B	-0.00	0.01	-0.04
Fav genre_Rap	0.01	-0.02	0.03
Fav genre_Rock	-0.08	0.06	0.06
Fav genre_Video game music	-0.08	0.04	0.12
Exploratory_No	-0.15	0.14	0.04
Exploratory_Yes	0.15	-0.14	-0.04
Foreign languages_No	-0.01	0.02	-0.01
Foreign languages_Yes	0.01	-0.02	0.01
Frequency [Classical]_Never	0.02	-0.02	0.00
Frequency [Classical]_Rarely	0.02	-0.02	-0.00
Frequency [Classical]_Sometimes	-0.03	0.03	0.01
Frequency [Classical]_Very frequently	-0.01	0.01	-0.01
Frequency [Country]_Never	-0.05	0.04	0.04
Frequency [Country]_Rarely	-0.01	0.01	-0.01
Frequency [Country]_Sometimes	0.06	-0.05	-0.04
Frequency [Country]_Very frequently	0.04	-0.04	-0.00
Frequency [EDM]_Never	-0.08	0.06	0.07
Frequency [EDM]_Rarely	0.10	-0.08	-0.05
Frequency [EDM]_Sometimes	-0.03	0.04	-0.03
Frequency [EDM]_Very frequently	0.02	-0.02	-0.00
Frequency [Folk]_Never	-0.00	-0.01	0.04
Frequency [Folk]_Rarely	0.02	-0.02	-0.00
Frequency [Folk]_Sometimes	-0.01	0.02	-0.03
Frequency [Folk]_Very frequently	-0.01	0.02	-0.02
Frequency [Gospel]_Never	-0.09	0.08	0.01
Frequency [Gospel]_Rarely	0.06	-0.07	0.02

	Music effects_Improve	Music effects_No effect	Music effects_Worsen
Frequency [Gospel]_Sometimes	0.01	0.01	-0.04
Frequency [Gospel]_Very frequently	0.08	-0.08	-0.02
Frequency [Hip hop]_Never	-0.09	0.11	-0.02
Frequency [Hip hop]_Rarely	0.02	-0.02	-0.02
Frequency [Hip hop]_Sometimes	0.07	-0.08	0.02
Frequency [Hip hop]_Very frequently	0.00	-0.01	0.03
Frequency [Jazz]_Never	-0.07	0.07	0.02
Frequency [Jazz]_Rarely	0.03	-0.04	0.02
Frequency [Jazz]_Sometimes	0.03	-0.01	-0.04
Frequency [Jazz]_Very frequently	0.03	-0.03	-0.01
Frequency [K pop]_Never	-0.05	0.06	-0.03
Frequency [K pop]_Rarely	0.00	-0.02	0.06
Frequency [K pop]_Sometimes	0.02	-0.03	0.01
Frequency [K pop]_Very frequently	0.05	-0.04	-0.05
Frequency [Latin]_Never	-0.05	0.05	-0.00
Frequency [Latin]_Rarely	0.00	0.00	-0.02
Frequency [Latin]_Sometimes	0.06	-0.06	-0.00
Frequency [Latin]_Very frequently	0.01	-0.04	0.06
Frequency [Lofi]_Never	-0.07	0.06	0.03
Frequency [Lofi]_Rarely	-0.00	0.01	-0.04
Frequency [Lofi]_Sometimes	0.06	-0.07	0.03
Frequency [Lofi]_Very frequently	0.03	-0.02	-0.03
Frequency [Metal]_Never	0.02	-0.01	-0.04
Frequency [Metal]_Rarely	0.00	-0.01	0.03
Frequency [Metal]_Sometimes	0.00	-0.01	0.02
Frequency [Metal]_Very frequently	-0.03	0.03	-0.01
Frequency [Pop]_Never	0.02	-0.03	0.03
Frequency [Pop]_Rarely	-0.10	0.11	-0.01
Frequency [Pop]_Sometimes	0.02	-0.01	-0.02
Frequency [Pop]_Very frequently	0.05	-0.06	0.01
Frequency [R&B]_Never	-0.08	0.08	0.02
Frequency [R&B]_Rarely	-0.06	0.04	0.06
Frequency [R&B]_Sometimes	0.09	-0.07	-0.07
Frequency [R&B]_Very frequently	0.07	-0.07	-0.02
Frequency [Rap]_Never	-0.04	0.04	-0.01
Frequency [Rap]_Rarely	-0.02	0.02	0.00
Frequency [Rap]_Sometimes	0.02	-0.02	0.01
Frequency [Rap]_Very frequently	0.04	-0.04	0.00
Frequency [Rock]_Never	0.02	-0.04	0.05
Frequency [Rock]_Rarely	0.01	-0.01	-0.01
Frequency [Rock]_Sometimes	0.00	-0.00	-0.00
Frequency [Rock]_Very frequently	-0.02	0.03	-0.03
Frequency [Video game music]_Never	-0.03	0.03	-0.01
Frequency [Video game music]_Rarely	0.01	0.00	-0.03
Frequency [Video game music]_Sometimes	0.06	-0.05	-0.03
Frequency [Video game music]_Very frequently	-0.04	0.01	0.08
Music effects_Improve	1.00	-0.94	-0.27
Music effects_No effect	-0.94	1.00	-0.09
Music effects_Worsen	-0.27	-0.09	1.00

In [ ]:
```
Data_Final=Data_Final.drop('Timestamp', axis=1)
```

In [ ]:
```
# # Data_Final.info()
# # Convert all columns to single datatype
# for i in Data_Final.columns:
#         if Data_Final[i].dtype!=float:
#                 #print(i)
#                 Data_Final[i].astype(float)
```

In [ ]:
```
# COnverting to single data type
Data_Final=Data_Final.astype(float)
```

In [ ]:
```
Data_Final.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 719 entries, 2 to 735
Columns: 106 entries, Age to Music effects_Worsen
dtypes: float64(106)
memory usage: 601.0 KB
```

Phase 1 : Implementing an ML model

In [ ]:
```
# No strong correlation seen above
# Using Decision Trees

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# identify the features and target variable
X = Data_Final.drop(['Music effects_Improve','Music effects_No effect','Music effects_Worsen'], axis=1)
y = Data_Final[['Music effects_Improve','Music effects_No effect','Music effects_Worsen']]

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [ ]:
```
# create a decision tree classifier object
clf = DecisionTreeClassifier()

# train the classifier on the training data
clf.fit(X_train, y_train)
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score
# make predictions on the testing data
y_pred = clf.predict(X_test)

# evaluate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
# precision = precision_score(y_test, y_pred)
# recall = recall_score(y_test, y_pred)

print("Accuracy:", accuracy.round(2)*100,'%')
# print("Precision",precision.round(2)*100,'%')
# print("Recall",recall.round(2)*100,'%')
```

```
Accuracy: 65.0 %
```

In [ ]:
```
# Implementing a simple neural network

import tensorflow as tf

# Define the model architecture using Sequential model
model = tf.keras.models.Sequential()
# Taking 50 units with 2 hidden layers
model.add(tf.keras.layers.Dense(50, activation='relu', input_dim=X_train.shape[1]))
model.add(tf.keras.layers.Dense(50, activation='relu'))
model.add(tf.keras.layers.Dense(3, activation='softmax'))
```

In [ ]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [ ]:
```
# Train the model
history = model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10, batch_size=32)
```

```
Epoch 1/10
18/18 [==============================] - 4s 122ms/step - loss: 4190.8755 - accuracy: 0.2383 - val_loss: 3765.2881 - val_accurac
y: 0.2569
Epoch 2/10
18/18 [==============================] - 0s 6ms/step - loss: 5517.1602 - accuracy: 0.4557 - val_loss: 3012.1226 - val_accuracy:
0.7292
Epoch 3/10
18/18 [==============================] - 0s 5ms/step - loss: 2159.6033 - accuracy: 0.6800 - val_loss: 186.6825 - val_accuracy:
0.8056
Epoch 4/10
18/18 [==============================] - 0s 5ms/step - loss: 13676.5146 - accuracy: 0.7113 - val_loss: 4330.6699 - val_accurac
y: 0.6389
Epoch 5/10
18/18 [==============================] - 0s 6ms/step - loss: 8022.1982 - accuracy: 0.5165 - val_loss: 6128.1133 - val_accuracy:
0.5833
Epoch 6/10
18/18 [==============================] - 0s 17ms/step - loss: 8534.3398 - accuracy: 0.6174 - val_loss: 5587.6362 - val_accurac
y: 0.7153
Epoch 7/10
18/18 [==============================] - 0s 12ms/step - loss: 7361.9390 - accuracy: 0.6678 - val_loss: 4638.9980 - val_accurac
y: 0.7431
Epoch 8/10
18/18 [==============================] - 0s 13ms/step - loss: 5930.5464 - accuracy: 0.6765 - val_loss: 3499.5837 - val_accurac
y: 0.7500
Epoch 9/10
18/18 [==============================] - 0s 10ms/step - loss: 4311.5415 - accuracy: 0.6730 - val_loss: 2385.7229 - val_accurac
y: 0.7500
Epoch 10/10
18/18 [==============================] - 0s 7ms/step - loss: 2752.3806 - accuracy: 0.6765 - val_loss: 1148.0195 - val_accuracy:
0.7500
```

Phase 3 : Improving the existing neural network

In [ ]:

```python
# Increaing the Capacity
# Define the model architecture using Sequential model
model_2 = tf.keras.models.Sequential()
# Taking 50 units with 2 hidden layers
model_2.add(tf.keras.layers.Dense(200, activation='relu', input_dim=X_train.shape[1]))
model_2.add(tf.keras.layers.Dense(100, activation='relu'))
model_2.add(tf.keras.layers.Dense(75, activation='relu'))
model_2.add(tf.keras.layers.Dense(55, activation='relu'))
model_2.add(tf.keras.layers.Dense(25, activation='relu'))
model_2.add(tf.keras.layers.Dense(3, activation='softmax'))

# Compile the model
model_2.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model_2.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=50, batch_size=32)
```

```
Epoch 1/50
18/18 [==============================] - 4s 120ms/step - loss: 15248.4834 - accuracy: 0.4661 - val_loss: 2189.7371 - val_accura
cy: 0.2292
Epoch 2/50
18/18 [==============================] - 1s 37ms/step - loss: 15336.5234 - accuracy: 0.4052 - val_loss: 4478.2476 - val_accurac
y: 0.2708
Epoch 3/50
18/18 [==============================] - 0s 11ms/step - loss: 7111.8271 - accuracy: 0.4035 - val_loss: 1402.4318 - val_accurac
y: 0.7292
Epoch 4/50
18/18 [==============================] - 0s 17ms/step - loss: 17720.8340 - accuracy: 0.3861 - val_loss: 8683.9453 - val_accurac
y: 0.2778
Epoch 5/50
18/18 [==============================] - 1s 36ms/step - loss: 10696.3330 - accuracy: 0.3670 - val_loss: 5432.7173 - val_accurac
y: 0.6042
Epoch 6/50
18/18 [==============================] - 0s 18ms/step - loss: 3059.9265 - accuracy: 0.6713 - val_loss: 372.9375 - val_accuracy:
0.7986
Epoch 7/50
18/18 [==============================] - 0s 15ms/step - loss: 693.0262 - accuracy: 0.6852 - val_loss: 82.8228 - val_accuracy:
0.8056
Epoch 8/50
18/18 [==============================] - 0s 7ms/step - loss: 4088.6616 - accuracy: 0.6157 - val_loss: 7196.1118 - val_accuracy:
0.3958
Epoch 9/50
18/18 [==============================] - 0s 6ms/step - loss: 10596.7012 - accuracy: 0.4539 - val_loss: 5589.9653 - val_accurac
y: 0.6667
Epoch 10/50
18/18 [==============================] - 0s 8ms/step - loss: 7098.9863 - accuracy: 0.6174 - val_loss: 5227.5894 - val_accuracy:
0.4167
Epoch 11/50
18/18 [==============================] - 0s 18ms/step - loss: 11073.8848 - accuracy: 0.5774 - val_loss: 7662.7207 - val_accurac
y: 0.7361
Epoch 12/50
18/18 [==============================] - 0s 14ms/step - loss: 6266.7300 - accuracy: 0.6713 - val_loss: 587.3619 - val_accuracy:
0.7500
Epoch 13/50
18/18 [==============================] - 0s 7ms/step - loss: 14063.4404 - accuracy: 0.4887 - val_loss: 11734.7969 - val_accurac
y: 0.6111
Epoch 14/50
18/18 [==============================] - 0s 10ms/step - loss: 11208.0752 - accuracy: 0.6435 - val_loss: 313.8983 - val_accurac
y: 0.8056
Epoch 15/50
18/18 [==============================] - 0s 6ms/step - loss: 12205.5654 - accuracy: 0.6939 - val_loss: 8406.3184 - val_accurac
y: 0.4306
Epoch 16/50
18/18 [==============================] - 0s 10ms/step - loss: 23681.0254 - accuracy: 0.5443 - val_loss: 21233.0039 - val_accura
cy: 0.7014
Epoch 17/50
18/18 [==============================] - 0s 14ms/step - loss: 27881.0918 - accuracy: 0.6539 - val_loss: 18398.2383 - val_accura
cy: 0.7361
Epoch 18/50
18/18 [==============================] - 0s 6ms/step - loss: 22515.7090 - accuracy: 0.6678 - val_loss: 14152.6299 - val_accurac
y: 0.7500
Epoch 19/50
18/18 [==============================] - 0s 10ms/step - loss: 15297.4590 - accuracy: 0.6748 - val_loss: 7744.6719 - val_accurac
y: 0.7500
Epoch 20/50
18/18 [==============================] - 0s 6ms/step - loss: 5255.3706 - accuracy: 0.6835 - val_loss: 763.7791 - val_accuracy:
0.8056
Epoch 21/50
18/18 [==============================] - 0s 11ms/step - loss: 1275.6613 - accuracy: 0.7096 - val_loss: 127.3769 - val_accuracy:
0.8056
Epoch 22/50
18/18 [==============================] - 0s 7ms/step - loss: 471.2910 - accuracy: 0.7217 - val_loss: 183.7133 - val_accuracy:
0.8056
Epoch 23/50
18/18 [==============================] - 0s 7ms/step - loss: 20999.5820 - accuracy: 0.5913 - val_loss: 22381.4980 - val_accurac
y: 0.7292
Epoch 24/50
18/18 [==============================] - 0s 11ms/step - loss: 23227.5469 - accuracy: 0.6626 - val_loss: 2189.0295 - val_accurac
y: 0.7431
Epoch 25/50
18/18 [==============================] - 0s 25ms/step - loss: 15341.2402 - accuracy: 0.5009 - val_loss: 12616.3906 - val_accura
cy: 0.7222
Epoch 26/50
18/18 [==============================] - 0s 10ms/step - loss: 12287.2207 - accuracy: 0.6730 - val_loss: 4509.1768 - val_accurac
y: 0.7500
Epoch 27/50
18/18 [==============================] - 0s 7ms/step - loss: 2493.2412 - accuracy: 0.6922 - val_loss: 543.9820 - val_accuracy:
0.8056
Epoch 28/50
18/18 [==============================] - 0s 6ms/step - loss: 26912.8887 - accuracy: 0.6243 - val_loss: 26026.9922 - val_accurac
y: 0.6181
Epoch 29/50
18/18 [==============================] - 0s 15ms/step - loss: 36532.9727 - accuracy: 0.6330 - val_loss: 23404.8438 - val_accura
cy: 0.7083
Epoch 30/50
18/18 [==============================] - 0s 9ms/step - loss: 25983.0352 - accuracy: 0.6626 - val_loss: 14712.1250 - val_accurac
y: 0.7431
Epoch 31/50
18/18 [==============================] - 0s 13ms/step - loss: 14757.0625 - accuracy: 0.6730 - val_loss: 6866.8574 - val_accurac
y: 0.7500
Epoch 32/50
18/18 [==============================] - 0s 8ms/step - loss: 5238.2837 - accuracy: 0.6835 - val_loss: 191.4386 - val_accuracy:
0.7986
Epoch 33/50
```

```
18/18 [==============================] - 0s 6ms/step - loss: 9031.2549 - accuracy: 0.6557 - val_loss: 10156.0459 - val_accurac
y: 0.6806
Epoch 34/50
18/18 [==============================] - 0s 5ms/step - loss: 16291.5283 - accuracy: 0.6643 - val_loss: 8928.7930 - val_accurac
y: 0.7222
Epoch 35/50
18/18 [==============================] - 0s 6ms/step - loss: 24762.3184 - accuracy: 0.6278 - val_loss: 6155.2490 - val_accurac
y: 0.1944
Epoch 36/50
18/18 [==============================] - 0s 7ms/step - loss: 16884.0723 - accuracy: 0.5913 - val_loss: 13414.4629 - val_accurac
y: 0.6667
Epoch 37/50
18/18 [==============================] - 0s 8ms/step - loss: 17629.4102 - accuracy: 0.6765 - val_loss: 10265.1338 - val_accurac
y: 0.7431
Epoch 38/50
18/18 [==============================] - 0s 20ms/step - loss: 11272.4717 - accuracy: 0.6748 - val_loss: 5048.3262 - val_accurac
y: 0.7500
Epoch 39/50
18/18 [==============================] - 0s 10ms/step - loss: 22309.8320 - accuracy: 0.6713 - val_loss: 1240.5448 - val_accurac
y: 0.6875
Epoch 40/50
18/18 [==============================] - 0s 13ms/step - loss: 15181.5498 - accuracy: 0.6713 - val_loss: 15159.5420 - val_accura
cy: 0.7500
Epoch 41/50
18/18 [==============================] - 0s 12ms/step - loss: 21095.0430 - accuracy: 0.6835 - val_loss: 14350.3242 - val_accura
cy: 0.7500
Epoch 42/50
18/18 [==============================] - 0s 8ms/step - loss: 19137.5684 - accuracy: 0.6748 - val_loss: 12845.7373 - val_accurac
y: 0.7431
Epoch 43/50
18/18 [==============================] - 1s 18ms/step - loss: 16893.3867 - accuracy: 0.6835 - val_loss: 10977.6377 - val_accura
cy: 0.7361
Epoch 44/50
18/18 [==============================] - 0s 8ms/step - loss: 14361.4502 - accuracy: 0.6817 - val_loss: 9092.6357 - val_accurac
y: 0.7361
Epoch 45/50
18/18 [==============================] - 0s 17ms/step - loss: 11719.3076 - accuracy: 0.6852 - val_loss: 7228.1113 - val_accurac
y: 0.7222
Epoch 46/50
18/18 [==============================] - 0s 8ms/step - loss: 9239.3457 - accuracy: 0.6800 - val_loss: 5332.4238 - val_accuracy:
0.7431
Epoch 47/50
18/18 [==============================] - 0s 6ms/step - loss: 6677.7510 - accuracy: 0.6800 - val_loss: 3395.4028 - val_accuracy:
0.7222
Epoch 48/50
18/18 [==============================] - 0s 7ms/step - loss: 3845.5227 - accuracy: 0.6661 - val_loss: 1337.8772 - val_accuracy:
0.7361
Epoch 49/50
18/18 [==============================] - 0s 8ms/step - loss: 1233.3602 - accuracy: 0.6922 - val_loss: 124.9256 - val_accuracy:
0.7847
Epoch 50/50
18/18 [==============================] - 0s 14ms/step - loss: 2165.2195 - accuracy: 0.7287 - val_loss: 1282.1060 - val_accurac
y: 0.7431
```

Conclusion:

1. Based on the correlation matrix above, didn't see any significant strong or week correlations
2. ML model had a similar accuracy as compared to neural networks. Improving the NN by increasing capacity didn't seem to make any different in accuracy