Exercise Day 7

Group Exercise

Advanced Regression Analysis and Feature Selection

Team:

Shunmugapriya Swamy – 100003722

Kavya Gopalaiah – 100001360

In this project, we focused on cleaning, preparing the *MXMH Survey Results* dataset for analysis. This dataset includes various features related to music habits and psychological factors, with challenges such as missing values, categorical data, and inconsistent scales. We used techniques like KNN Imputation, Random Forest modeling, feature scaling and Decision tree classification to address these issues systematically. This report outlines the step-by-step process we followed, providing insights into our decisions and the methods used to ensure the data was ready for analysis and modeling.

Dataset and Code Access

GitHub Repository:

1. *https://github.com/Shunmugapriya-1612/Statistics-and-Machine-Learning/tree/Development* - Folder name "**Exercise-Day_7-Group-Exercise**"
2. *Git Repository: Kavya Gopalaiah Day 7*

1. Importing Necessary Libraries

To begin, we imported all the libraries we needed for data handling, modeling, and visualization. Libraries like pandas and numpy were used for data manipulation, while scikit-learn provided tools for imputation, encoding, scaling, and modeling.

```
import pandas as pd
import numpy as np
from sklearn.impute import KNNImputer
from sklearn.ensemble import RandomForestRegressor
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import cross_val_score, KFold
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
import shap
from sklearn.tree import DecisionTreeClassifier
```

2. Loading the Dataset

We loaded the dataset using pandas and reviewed its structure to understand the features and identify missing values.

```
# dataset loaded
df = pd.read_csv('mxmh_survey_results.csv')
```

3. Handling Missing Values in the Age Feature

The Age feature had missing values, which we imputed using the KNN Imputer. This approach estimates missing values based on the nearest neighbors in the dataset, maintaining data relationships.

- Selected numeric columns such as [‘Age’, 'Hours per day', 'BPM', 'Anxiety', 'Depression', 'Insomnia', 'OCD’], as KNN requires numeric data.
- Initialized the imputer with k=5 neighbors.

- Imputed missing values and replaced the original Age column with the imputed values.

```
#---------------------------------Handling missing values of Age feature ------------------
# Selecting relevant numeric columns for imputation
numeric_cols = ['Age', 'Hours per day', 'BPM', 'Anxiety', 'Depression', 'Insomnia', 'OCD']
df_numeric = df[numeric_cols]

# Initializing KNN Imputer
imputer = KNNImputer(n_neighbors=5)

# Performing KNN imputation
imputed_data = imputer.fit_transform(df_numeric)

print("Imputed data based on prediction\n",imputed_data)
print("\n")
# Creating a new dataframe with imputed values
df_imputed = pd.DataFrame(imputed_data, columns=numeric_cols)


# Replacing the original 'Age' column with the imputed values
df['Age'] = df_imputed['Age']
df['Age'] = df['Age'].astype(int)
```

4. Handling Missing Values in Categorical Features

For categorical features like Primary streaming service, while working, Instrumentalist, and Composer, we opted for forward fill to fill missing values. As this method propagates the last valid observation forward, ensuring the continuity of responses.

```
#---------------------------------------Handling missing values of categorical features--
# Filling missing values using forward fill
df['Primary streaming service'] = df['Primary streaming service'].ffill()
df['While working']=df['While working'].ffill()
df['Instrumentalist'] =df['Instrumentalist'].ffill()
df['Composer']=df['Composer'].ffill()
df['Composer']=df['Composer'].ffill()
```

5. Handling Missing Values in the BPM Feature

The BPM feature (beats per minute) had missing values, so we predicted these using a Random Forest Regressor.

- Splitted the data into rows with and without missing BPM values.
- One-hot encoded categorical features (on beat based music genres).

- Trained a RandomForestRegressor on rows with complete BPM values.
- Predicted and imputed missing BPM values.

```
#-------------------------------------Handling missing value of BPM feature---------------------------
## 1. Spliting data into rows with and without missing values
train_data = df[df['BPM'].notnull()]
missing_data = df[df['BPM'].isnull()]

#Features used to predict values of BPM
categorical_columns = ['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]'
]

# One-hot encoding categorical columns for training and missing data
train_data_encoded = pd.get_dummies(train_data, columns=categorical_columns, drop_first=True)
missing_data_encoded = pd.get_dummies(missing_data, columns=categorical_columns, drop_first=True)

# Aligning columns in missing_data_encoded to match the columns in train_data_encoded
missing_data_encoded = missing_data_encoded.reindex(columns=train_data_encoded.columns, fill_value=0)

# Features which have less correlation with feature BPM
col = ['Timestamp','Age','Primary streaming service','Hours per day',
'While working','Instrumentalist','Composer',
'Fav genre','Exploratory','Foreign languages','BPM','Anxiety',
'Depression','Insomnia','OCD','Music effects']
```

```
# Splitting into features and target
X_train = train_data_encoded.drop(col, axis=1) # Features for training
y_train = train_data_encoded['BPM'] #Target (BPM)
X_missing = missing_data_encoded.drop(col, axis=1) # Features for missing data

# Training a RandomForestRegressor model
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Predicting the missing BPM values
predicted_values = model.predict(X_missing)

# Filling missing values in the original dataframe
df.loc[df['BPM'].isnull(), 'BPM'] = predicted_values
```

6. Handling Missing Values in the Music Effects Feature

For the Music effects column,

- Temporarily replaced missing values with 'Unknown'.
- Encoded it with a Label Encoder.

- Trained a RandomForestClassifier to predict missing values using related features like Age and mental health scores.

```
#----------------------------------Handling missing value of Music effects feature--------------------
# Filling missing values temporarily with 'Unknown' for encoding
df['Music effects']= df['Music effects'].fillna('Unknown')

# Encoding the 'Music effects' column
label_encoder = LabelEncoder()
df['Music effects'] = label_encoder.fit_transform(df['Music effects'])

# Selecting features for prediction (you can add more relevant features)
features = ['Age', 'Hours per day', 'Anxiety', 'Depression', 'Insomnia', 'OCD']

# Prepareing the data for training
known_data = df[df['Music effects'] != label_encoder.transform(['Unknown'])[0]]
unknown_data = df[df['Music effects'] == label_encoder.transform(['Unknown'])[0]]

X_known = known_data[features]
y_known = known_data['Music effects']
X_unknown = unknown_data[features]

# Training a Random Forest Classifier
X_train, X_test, y_train, y_test = train_test_split(X_known, y_known, test_size=0.2, random_state=42)
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

```
# Predicting missing 'Music effects' values if X_unknown is not empty
if not X_unknown.empty:
    y_pred_unknown = rf_model.predict(X_unknown)
    df.loc[df['Music effects'] == label_encoder.transform(['Unknown'])[0], 'Music effects'] = y_pred_unknown

# Decoding the 'Music effects' column back to original labels
df['Music effects'] = label_encoder.inverse_transform(df['Music effects'])
```

7. Removing Noise and Scaling Numerical Features

After handling missing values, we cleaned and normalized the numerical features. Specifically:

- Removed rows where BPM values exceeded 250 to eliminate Noise.
- Applied MinMaxScaler to scale hyperparameters like BPM and Hours per day to a [0, 1] range for consistent numerical comparisons.

```
#----------------------------------------removing noise and scaling numerical features
scaler = MinMaxScaler()

# Removing rows where BPM is above 200
df = df[df['BPM'] < 250]  # Removes rows where BPM is above 200

# Scaling features using Normalization
numeric_cols = ['BPM', 'Hours per day']
df.loc[:, numeric_cols] = scaler.fit_transform(df.loc[:, numeric_cols])
```

8. Encoding Categorical Features

To prepare the dataset for machine learning models, we encoded categorical features using one-hot encoding. This created binary columns for each category in categorical features.

```
#-----------------------------------------------Encoding categorical features-----------------------------------------------

categorical_cols = ['Primary streaming service', 'While working','Instrumentalist','Composer','Fav genre'
,'Exploratory','Foreign languages','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]','Music effects']

#encoding categorical features
EncodedDF = pd.get_dummies(df, columns=categorical_cols)
EncodedDF.drop(columns=['Timestamp'], inplace=True)
```

9. Feature Selection Using Decision Tree Classifier

We used a Decision Tree Classifier to identify the most important features that contributed to predicting the target variables (Music effects categories). This step ensured that only the most relevant features were used in subsequent models.

```
#-------------------------Decision tree model(feature selection)-----------------------------------------------------------
dropFeatures = ['Music effects_Improve','Music effects_No effect','Music effects_Worsen']
X = EncodedDF.drop(columns=dropFeatures)
y = EncodedDF[dropFeatures]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
dt_model = DecisionTreeClassifier(random_state=42)

# Fitting the model on the training data
dt_model.fit(X_train, y_train)

importances = dt_model.feature_importances_

# Creating a DataFrame to view feature importances
feature_importance_df = pd.DataFrame({
    'Feature': X.columns,
     'Coefficient': importances
}).sort_values(by='Coefficient', ascending=False)

print("Selected features based on decision tree\n\n",feature_importance_df['Feature'].head(10).tolist())
```

10. Linear Regression

Using the selected features, we trained a Linear Regression model to predict the target variables (Music effects categories).

```
#-----------------------------------------Linear Regression-----------------------------------
X_selected=X[feature_importance_df['Feature'].head(10)]
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.1, random_state=42)


model = LinearRegression()
model.fit(X_train, y_train)

# Making predictions
y_pred = model.predict(X_test)

# Evaluating the model
print("***Linear Regression****\n")
print("Root Mean Squared Error (MSE):", np.sqrt(mean_squared_error(y_test, y_pred)))
print("R^2 Score:", r2_score(y_test, y_pred))
print("\n")
```

11. Cross-Validation

To validate the model's performance, we used cross-validation with 5 folds to ensure the results were robust across different subsets of the data.

```
#--------------------------Cross-validation-----------------------------------------------------

rf_model = RandomForestClassifier(random_state=42)
cv_scores = cross_val_score(rf_model, X_selected, y, cv=5, scoring='accuracy')
print("***Cross Validation****\n")
# Printing out the accuracy for each fold
print("Cross-validation scores for each fold:", cv_scores)

# Computing the average accuracy across all folds
average_accuracy = cv_scores.mean()
print("***Cross fold validation****\n")
print(f"Average accuracy across all folds: {average_accuracy:.4f}")
print("\n")
```

12. Visualizing Feature Importances

A heatmap was created to visualize the coefficients of the selected features for each target variable.

```
#-------------------------------------feature importance heat map---------

coefficients_df = pd.DataFrame(model.coef_, columns=X_selected.columns)
coefficients_df.index = dropFeatures

plt.figure(figsize=(10, 6))
sns.heatmap(coefficients_df, annot=True, cmap="coolwarm", center=0)
plt.title("Feature Importance for Multiple Targets")
plt.xlabel("Features")
plt.ylabel("Target Variables")
plt.show()
```

13. Residual Analysis

Residual plots were generated for each target variable to assess model performance visually and identify patterns in prediction errors.

```
#-------------------------------residual plot--------------------------------
# Prediction
y_pred = model.predict(X_test)

# Calculating residuals for each target variable
residuals_improve = y_test['Music effects_Improve'] - y_pred[:, 0]
residuals_no_effect = y_test['Music effects_No effect'] - y_pred[:, 1]
residuals_worsen = y_test['Music effects_Worsen'] - y_pred[:, 2]

# Plotting residuals for each target variable
fig, axs = plt.subplots(3, 1, figsize=(10, 18))

# Residual Plot for 'Music effects_Improve'
sns.scatterplot(x=y_pred[:, 0], y=residuals_improve, ax=axs[0])
axs[0].axhline(y=0, color='r', linestyle='--')
axs[0].set_title("Residual Plot for 'Music effects_Improve'")
axs[0].set_xlabel("Predicted Values")
axs[0].set_ylabel("Residuals")

# Residual Plot for 'Music effects_No effect'
sns.scatterplot(x=y_pred[:, 1], y=residuals_no_effect, ax=axs[1])
axs[1].axhline(y=0, color='r', linestyle='--')
axs[1].set_title("Residual Plot for 'Music effects_No effect'")
axs[1].set_xlabel("Predicted Values")
axs[1].set_ylabel("Residuals")
```

```
# Residual Plot for 'Music effects_Worsen'
sns.scatterplot(x=y_pred[:, 2], y=residuals_worsen, ax=axs[2])
axs[2].axhline(y=0, color='r', linestyle='--')
axs[2].set_title("Residual Plot for 'Music effects_Worsen'")
axs[2].set_xlabel("Predicted Values")
axs[2].set_ylabel("Residuals")

# Adjusting layout and show plots
plt.tight_layout()
plt.show()

print("****Residual plots on categorical target****\n")
print("Root Mean Squared Error (RMSE) for 'Music effects_Improve':", np.sqrt(mean_squared_error(y_test['Music effects_Improve'], y_pred[:, 0])))
print("Root Mean Squared Error (RMSE) for 'Music effects_No effect':", np.sqrt(mean_squared_error(y_test['Music effects_No effect'], y_pred[:, 1])))
print("Root Mean Squared Error (RMSE) for 'Music effects_Worsen':", np.sqrt(mean_squared_error(y_test['Music effects_Worsen'], y_pred[:, 2])))
print("\n")
```

14. Regularization

Both L1 (Lasso) and L2 (Ridge) regularization methods were applied to prevent overfitting and improve the model's generalization.

```
#-----------------------------------------l2 Regularization-----------------------------------------
ridge_model = Ridge(alpha=1.0)

# Fiting the model to the training data
ridge_model.fit(X_train, y_train)

# Making predictions
y_pred_ridge = ridge_model.predict(X_test)

# Evaluating the model
print("***l2 Regularization****\n")
print("Ridge Model - Root Mean Squared Error:", np.sqrt(mean_squared_error(y_test, y_pred_ridge)))
print("Ridge Model - R^2 Score:", r2_score(y_test, y_pred_ridge))
print("\n")

#-------------------------------------------------l1 Regularization-----------------------------------
lasso_model = Lasso(alpha=0.01, positive=True)
lasso_model.fit(X_train, y_train)

y_pred_lasso = lasso_model.predict(X_test)
rmse_lasso = np.sqrt(mean_squared_error(y_test, y_pred_lasso))
r2_lasso = r2_score(y_test, y_pred_lasso)

print("***l1 Regularization****\n")
print(f"Lasso with Lower Alpha - RMSE: {rmse_lasso}, R^2: {r2_lasso}")
```

15. Polynomial Regression

A Polynomial Regression model was applied to capture potential nonlinear relationships in the data.

```
#-------------------------------------Polynomial Regression----------
poly = PolynomialFeatures(degree=1)
X_poly = poly.fit_transform(X_train)
poly_model = LinearRegression()
poly_model.fit(X_poly, y_train)

X_test_poly = poly.transform(X_test)
y_pred_poly = poly_model.predict(X_test_poly)
rmse_poly = np.sqrt(mean_squared_error(y_test, y_pred_poly))
r2_poly = r2_score(y_test, y_pred_poly)
print("***Polynomial Regression****\n")
print(f"Polynomial Regression - RMSE: {rmse_poly}, R^2: {r2_poly}")
print("\n")
```

16. Explaining Results with SHAP

Finally, we used the SHAP library to explain the model predictions and identify which features had the most influence.

```
#-----------------------------------------Shap package for reporting--------
model = LinearRegression()
model.fit(X_train, y_train)
explainer = shap.LinearExplainer(model, X_train)
shap_values = explainer(X_test)
sample_idx = 0  # Index of the sample to explain (change as needed)
for i, target in enumerate(['Music effects_Improve', 'Music effects_No effect', 'Music effects_Worsen']):
    print(f"Saving SHAP Force Plot for Target: {target}")
    force_plot = shap.force_plot(
        explainer.expected_value[i],
        shap_values.values[sample_idx, :, i],
        X_test.iloc[sample_idx],
        feature_names=X_test.columns
    )

    shap.save_html(f"force_plot_target_{i}.html", force_plot)
    print(f"Force plot for {target} saved as force_plot_target_{i}.html")
```

This comprehensive pipeline cleaned the dataset, handled missing values, encoded features, selected important predictors, and trained models. Each step enhanced our understanding and improved the performance of predictive models for the Music effects target variables.

Force Plots (for target – ‘Music effects - Improve’)

Force Plots (for target – ‘Music effects – No effects’)

Force Plots (for target – ‘Music effects – Worsen)

Explanation of Outputs

Imputed Data(‘Age’) Based on Predictions: After handling missing values and imputing them using predictive models, the processed dataset resembles the array shown. This ensures no missing data and a clean dataset for further analysis and modeling.

```
Imputed data based on prediction
 [[ 18.    3.  156.  ...   0.    1.    0. ]
 [ 63.    1.5 119.  ...   2.    2.    1. ]
 [ 18.    4.  132.  ...   7.   10.    2. ]
 ...
 [ 19.    6.  120.  ...   2.    2.    2. ]
 [ 19.    5.  170.  ...   3.    2.    1. ]
 [ 29.    2.   98.  ...   2.    2.    5. ]]
```

Selected Features Based on Decision Tree: ***Id***entified the top 10 features contributing most to the target variable prediction. These features include demographic, psychological, and musical preference variables.

```
Selected features based on decision tree

 ['Anxiety', 'Age', 'OCD', 'Depression', 'Frequency [Gospel]_Never', 'Hours per day', 'BPM', 'Frequency [Pop]_Never', 'Fav genre_Pop', 'Frequency [
```

Linear Regression Results: Was trained using the selected features and evaluated for its predictive performance:

Root Mean Squared Error (RMSE): 0.352, indicating moderate error in prediction.

R^2 Score: 0.035, showing that only 3.5% of the variability in the target variable is explained by the model, suggesting the need for further improvements or alternative modeling techniques.

```
***Linear Regression****

Root Mean Squared Error (MSE): 0.351975436365514
R^2 Score: 0.03521628805115029
```

Cross-Validation Accuracy: The model was cross-validated using 5-fold validation:

Accuracy scores for each fold: [0.7551, 0.7143, 0.7755, 0.7483, 0.7055].

Average accuracy: 0.7397 (or 73.97%), reflecting stable performance across folds.

```
***Cross Validation****

Cross-validation scores for each fold: [0.75510204 0.71428571 0.7755102  0.74829932 0.70547945]
```

```
***Cross fold validation****

Average accuracy across all folds: 0.7397
```

Residual Analysis for Categorical Targets: RMSE values were computed for each target class to understand prediction errors. Lower RMSE indicates better fit:

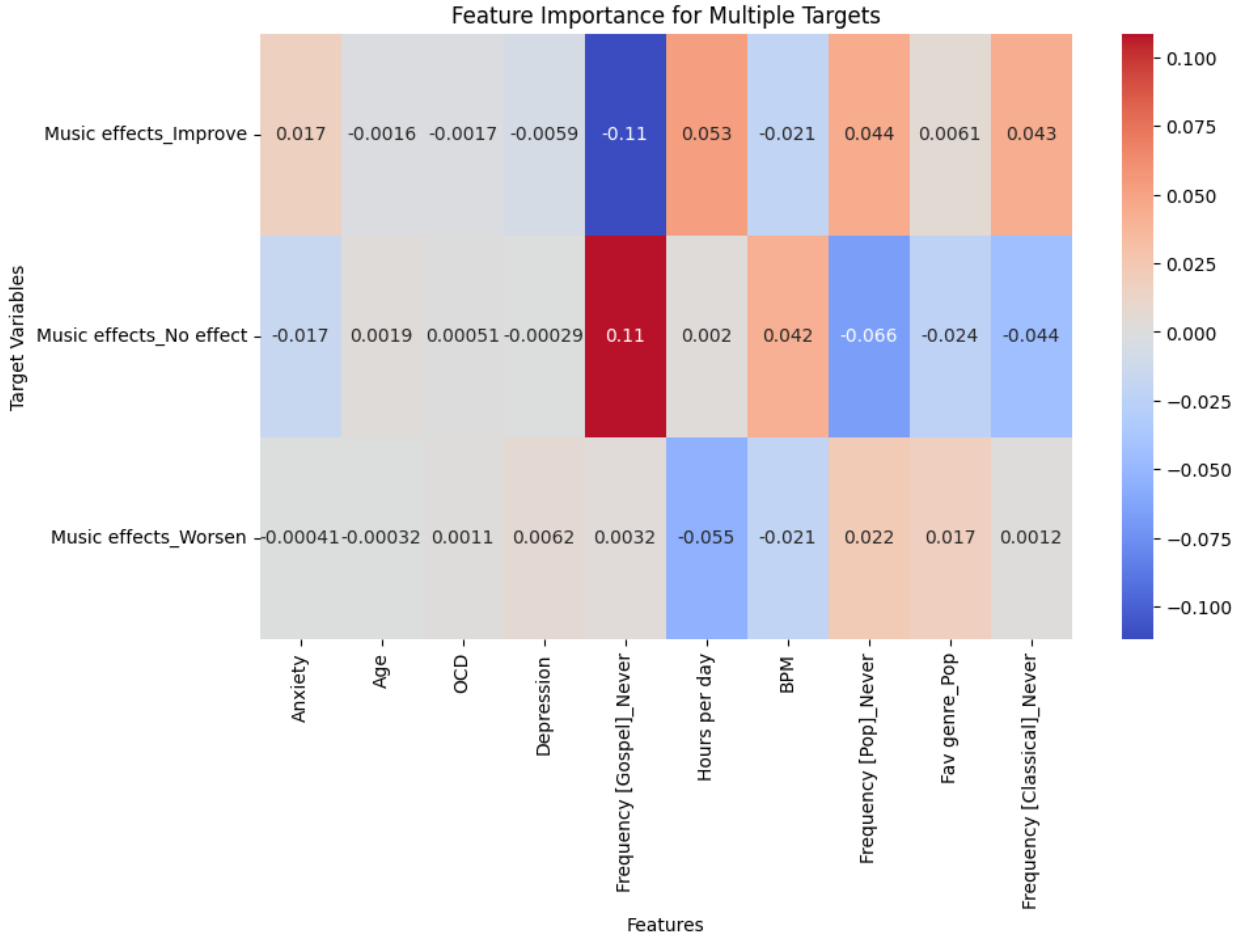
Music effects_Improve: RMSE = 0.426

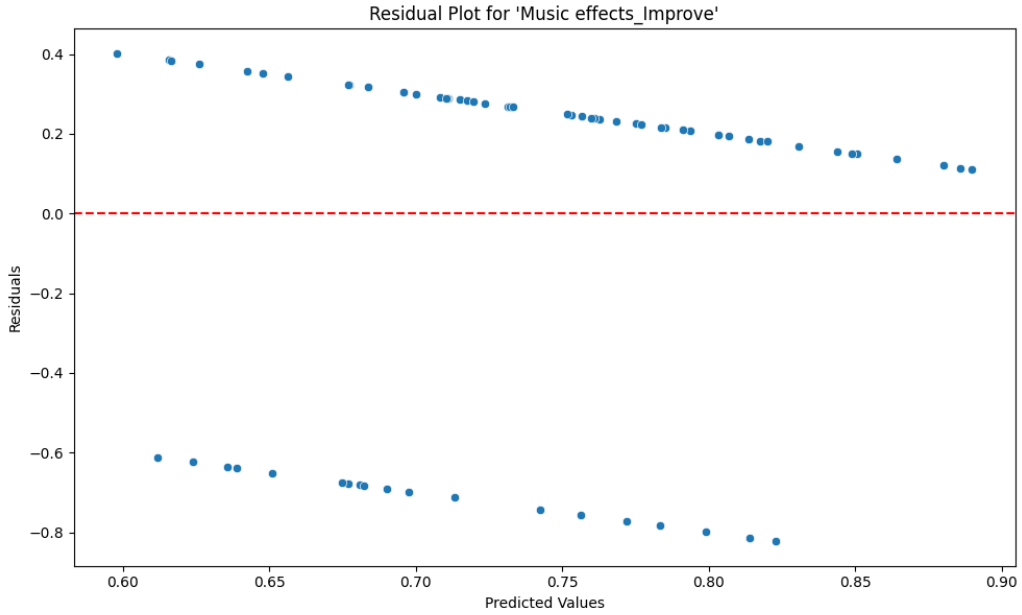
Music effects_No effect: RMSE = 0.421

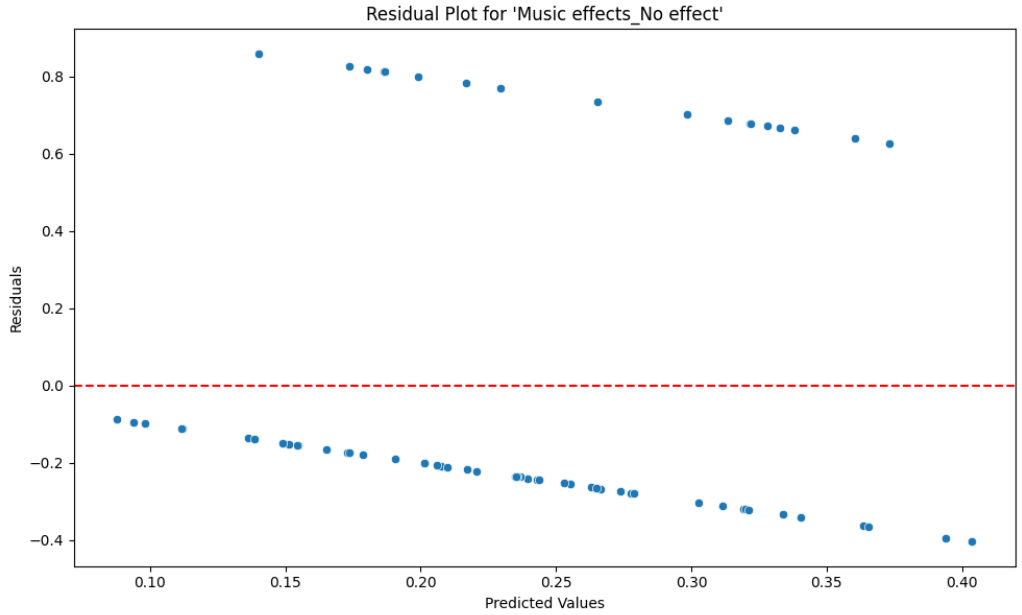
Music effects_Worsen: RMSE = 0.114

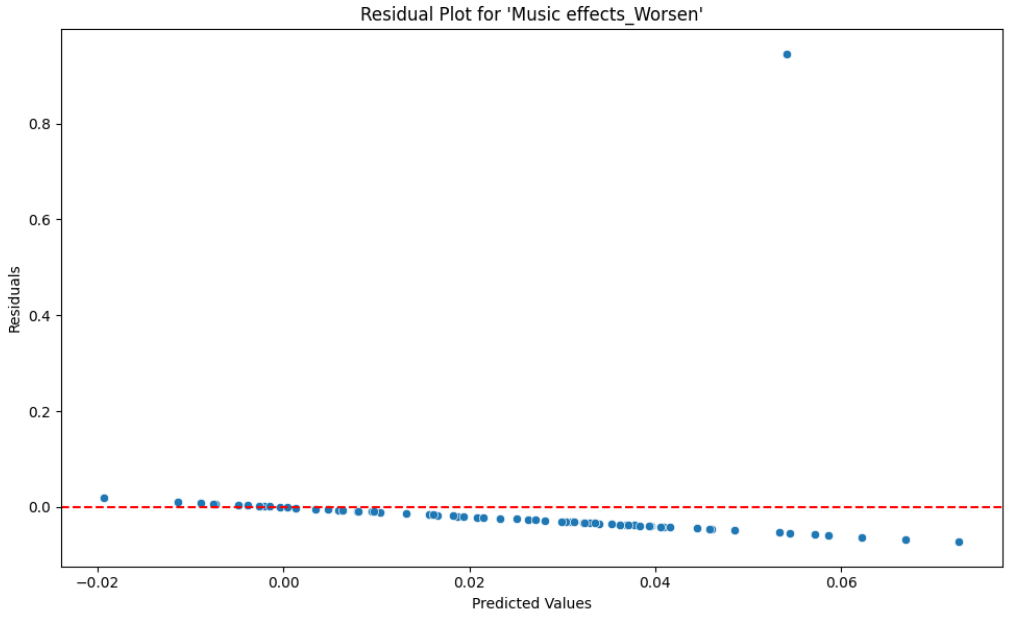
```
****Residual plots on categorical target****

Root Mean Squared Error (RMSE) for 'Music effects_Improve': 0.42603816739377237
Root Mean Squared Error (RMSE) for 'Music effects_No effect': 0.42079733271375136
Root Mean Squared Error (RMSE) for 'Music effects_Worsen': 0.11437310924701843
```




Residual Plot for 'Music effects_Improve'
Residuals
Predicted Values
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
0.60
0.65
0.70
0.75
0.80
0.85
0.90


Residual Plot for 'Music effects_No effect'
Residuals
Predicted Values
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
0.10
0.15
0.20
0.25
0.30
0.35
0.40


Residual Plot for 'Music effects_Worsen'
Residuals
Predicted Values
0.8
0.6
0.4
0.2
0.0
−0.02
0.00
0.02
0.04
0.06

L2 Regularization (Ridge Regression): Helped in preventing overfitting.

RMSE: 0.352

R^2: 0.035, similar to linear regression.

```
***l2 Regularization****

Ridge Model - Root Mean Squared Error: 0.35195819723618105
Ridge Model - R^2 Score: 0.03510207614192359
```

L1 Regularization (Lasso Regression): Used with a lower alpha to fine-tune the model.

RMSE: 0.354

R^2: 0.027, indicating slightly lower performance compared to Ridge regression.

```
***l1 Regularization****

Lasso with Lower Alpha - RMSE: 0.3544608903413442, R^2: 0.026615621949344376
```

Polynomial Regression: Applied to capture non-linear relationships, yielding:

RMSE: 0.352

R^2: 0.035, identical to linear regression, suggesting no significant non-linear relationships in the data.

```
***Polynomial Regression****

Polynomial Regression - RMSE: 0.351975436365514, R^2: 0.035216288051150325
```

SHAP Force Plots for Model Explanation: Was used to explain the impact of each feature on model predictions for each target variable.

Force Plot for Music effects_Improve: Explains how features influence predictions for the Improve category.

Force Plot for Music effects_No effect: Explains predictions for the No effect category.

Force Plot for Music effects_Worsen: Explains predictions for the Worsen category.

These force plots are saved as HTML files:

force_plot_target_0.html (Improve)

force_plot_target_1.html (No effect)

force_plot_target_2.html (Worsen).

The SHAP visualizations provide an interpretable summary of the model's predictions, making it easier to understand the significance of each feature in influencing the target variable.

THANK YOU