

# Final Project Report

By

**Muhammad Abdul Rafey**

# Hit or Miss?

Predicting Song Success on Spotify Using Machine Learning

ITCS-3156-001: Intro to Machine Learning

Professor: **Minwoo Lee**

Date: 12/05/2025

# Introduction

**Problem Statement:** The music streaming industry has reshaped the ways in which artists publish and make money from their work, with platforms like Spotify becoming the leading indicator of a song's popularity. The goal of this project is to predict the "Popularity Score" (0-100) of a track based on only its audio characteristics, including acousticness, danceability, energy, and duration. This is a regression supervised problem that analyzes over 114,000 tracks in order to see whether or not intrinsic audio features can predict commercial success reliably, or whether popularity is determined by external factors.

**Motivation and Challenges:** Record labels, producers, and independent artists looking to optimize their music for streaming algorithms must understand the drivers of popularity. The capability to predict popularity based on audio features means creators can "tune" their tracks for better reception in the market before the release of a track.

However, this task is extremely challenging. First, the dataset is "zero-inflated," in which many of the songs in the dataset have a popularity score of 0, likely due to a lack of marketing rather than poor audio quality. Second, musical taste is subjective and complex; there likely does not exist a linear relationship between a single feature such as "loudness" and popularity. Overcoming these challenges calls for models that can handle nonlinear patterns and feature interactions.

**Summary of Approach:** To solve this problem, I used a dataset of Spotify tracks that had a mix of genres and audio metrics. I then put into place a machine learning pipeline comparing two different approaches:

**1. Linear Models (OLS and Ridge Regression):** To establish a baseline and test for linear relationships between audio features and popularity, while using Ridge regularization to handle high correlation between features like energy and loudness.

**2. Non-Linear Model (Random Forest Regressor):** To capture complex, non-linear interactions between features.

Models were then evaluated using the RMSE along with R-squared ( $R^2$ ) metrics to identify which approach best models the chaotic nature of music popularity.

## Data

**Dataset Description:** The dataset used for this project is the "Spotify Tracks Dataset" taken from Kaggle by Maharshi Pandya. It includes about 114,000 unique tracks over 125 genres. The data consists of metadata such as track name, artist, and album, but most importantly, it has numerical audio features extracted directly from the Spotify API.

The target variable for this project is **popularity**, a score from 0 to 100 assigned by Spotify's algorithm based on total streams and recent stream frequency. The input features include technical audio attributes such as:

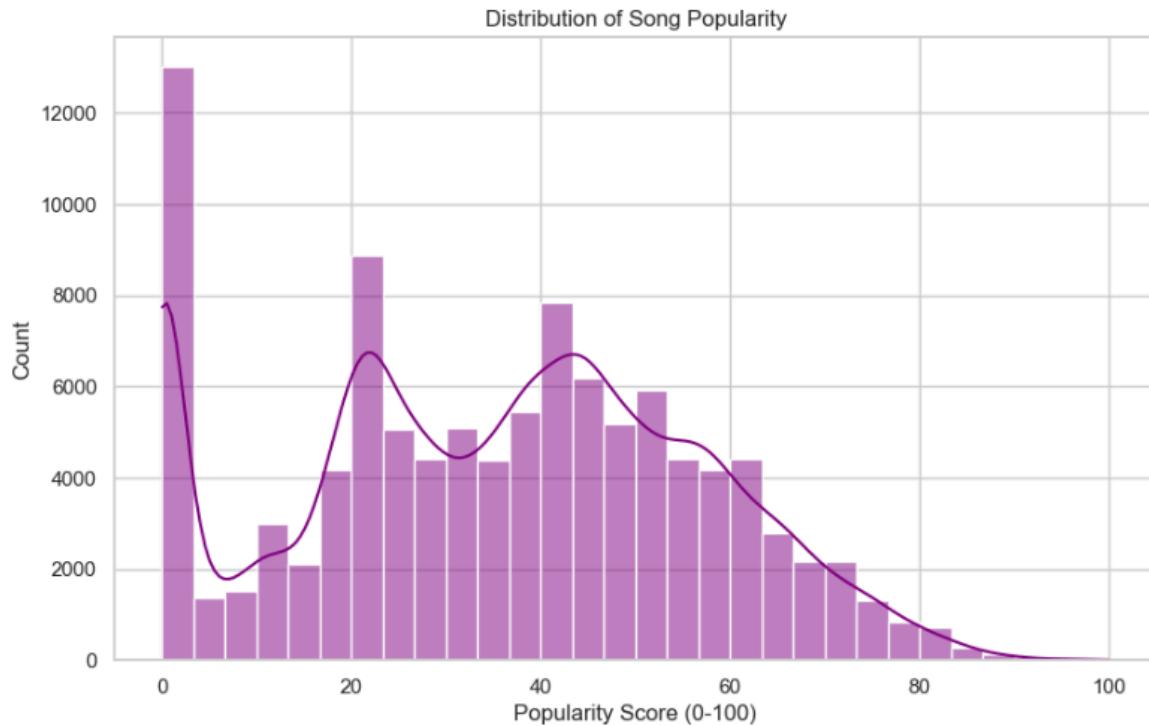
- **Acousticness:** Confidence measures from 0.0 to 1.0 of whether the track is acoustic.
- **Danceability:** How suitable a track is for dancing (0.0 to 1.0).
- **Energy:** A measure of intensity and activity.
- **Loudness:** The overall loudness of a track in decibels (dB).
- **Valence:** A measure of musical positiveness (e.g., happy, cheerful, euphoric).

**Data Preprocessing:** Before modeling, I performed a thorough data cleaning checklist to ensure quality:

1. Feature Reduction: Identifiers like *track\_id*, *album\_name*, and *track\_name* were removed as they do not provide any predictive value for a generalized regression model.
2. Duplicate Removal: Essentially, there were duplicate songs across multiple albums in the dataset. After carefully identifying the duplicate rows, approximately 7,000 I was able to remove them to avoid leakage between testing and training sets.
3. Handling Missing Values: I checked for null values and found the dataset clean, requiring no imputation.
4. Standardization: Since the features have vastly different ranges (e.g., *loudness* is in negative dB while *tempo* is 50-200 BPM), I applied *StandardScaler* to normalize the data. This step was critical for the Ridge Regression model, which is sensitive to the scale of input variables.

**Visual Analysis:** To understand the underlying patterns, I generated visualizations of the target distribution and feature correlations.

**Figure 1: Distribution of Song Popularity:**



As is evident from Figure 1, the popularity distribution is very skewed and "zero-inflated." A large portion of the data has a popularity score of 0, and thus regression models face difficulty with this data since it is not normally distributed; rather, there's a clear dichotomy between "unheard" songs and successful ones.

**Figure 2: Feature Correlation Heatmap**

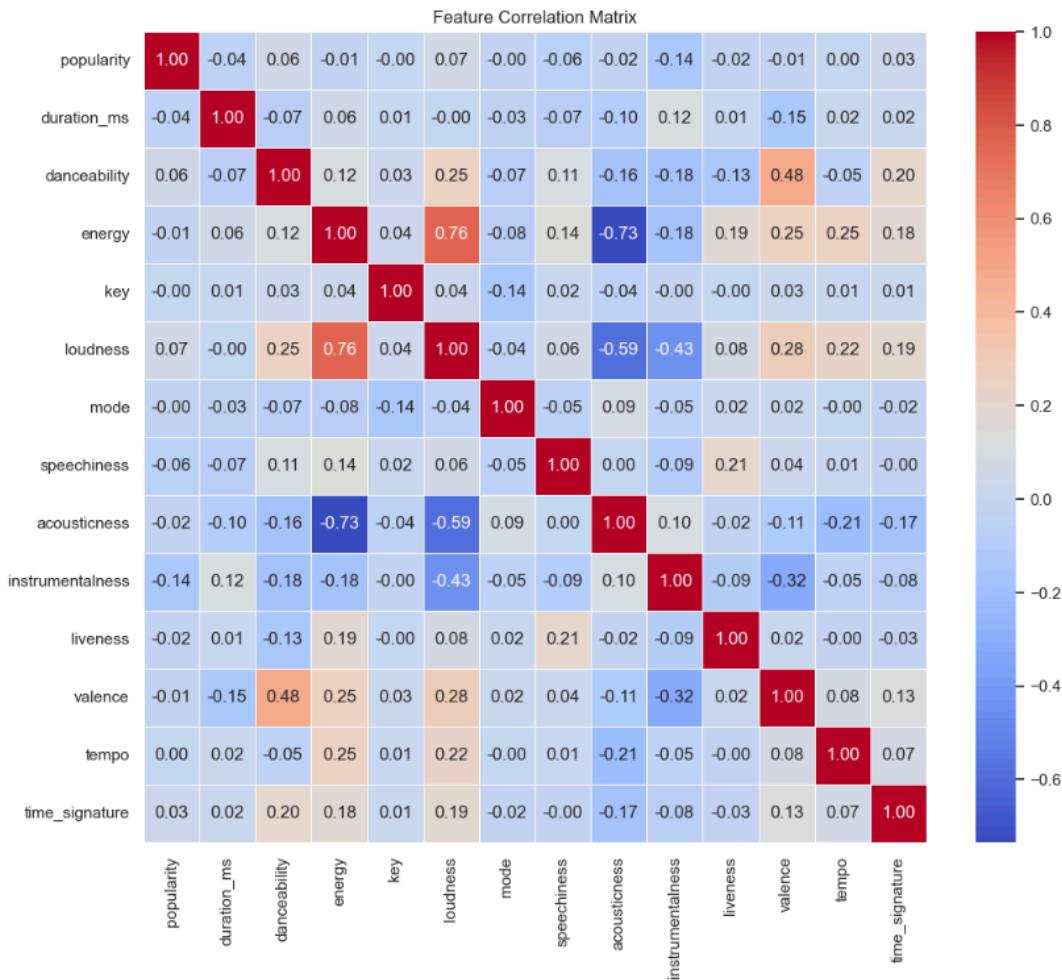


Figure 2 shows the correlation between numerical features. An important trend highlighted here is a very strong positive correlation of 0.76 between Energy and Loudness, with a strong negative correlation of -0.73 between Acousticness and Energy. This multicollinearity demonstrates why ordinary least squares regression is inappropriate and thus further justifies using Ridge Regression (since it uses L2 regularization), as standard linear models cannot handle highly correlated features.

## Methods

**Experimental Setup:** The work has been implemented in Python using the Scikit-Learn library. The dataset was divided into a train set of 80% and a test set of 20% to ensure that models were tested on unseen data.

- **Feature Scaling:** Since Ridge Regression is sensitive to the magnitude of coefficients, I applied *StandardScaler* to all numerical features. This ensures that features with large ranges like "Duration" (ms) and features with small ranges like "Speechiness" (0-1) are treated equally by the model.

**Machine Learning Algorithms:** I employed two distinct modeling approaches to compare linear vs. non-linear performance:

1. **Ridge Regression (Linear Approach):** I first established a baseline using linear models. The standard Ordinary Least Squares (OLS) model assumes a linear relationship expressed as:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \epsilon$$

However, there is a strong multicollinearity between 'Energy' and 'Loudness' - their correlation stands at 0.76. In such cases, standard OLS can become unstable. So, I decided to use Ridge Regression. Unlike OLS, which only minimized the Sum of Squared Errors (SSE), Ridge added an L2 regularization penalty to the loss function:

$$\text{Loss} = \sum_{i=1}^n (y_i - \hat{y}_i)^2 + \alpha \sum_{j=1}^p \beta_j^2$$

Here,  $\alpha$  is the regularization strength (hyperparameter). I used *GridSearchCV* to tune this  $\alpha$  parameter, testing values from 0.1 to 1000. This penalty shrinks the coefficients ( $\beta$ ) towards zero, reducing model variance and preventing overfitting.

2. **Random Forest Regressor (Non-Linear Approach):** Considering the hypothesis that the popularity of music is driven by complex, non-linear interactions of features, I implemented a Random Forest Regressor. The ensemble method constructs multiple decision trees at training time. For regression tasks, the final prediction is made through averaging the outputs of all the individual trees:

$$\hat{y} = \frac{1}{T} \sum_{t=1}^T h_t(x)$$

Where  $T$  is the number of trees (100 in this experiment) and  $h_t(x)$  is the prediction from the  $t$ -th tree. This method is robust to outliers and implicitly performs feature selection.

3. **Evaluation Metrics:** The models were evaluated using R-Squared ( $R^2$ ), which measures the proportion of variance in popularity explained by the model, and Root Mean Squared Error (RMSE), which measures the average error in prediction points.

## Results & Discussion

**Linear Model Performance (OLS & Ridge):** I began by training the Ordinary Least Squares (OLS) and Ridge Regression models. Despite applying standardization and tuning the Ridge regularization parameter ( $\alpha$ ), both linear models performed poorly.

- OLS  $R^2$  Score: 0.0369
- Ridge  $R^2$  Score: 0.0369 (Best  $\alpha$ : 100.0)
- RMSE: ~21.01

As the results illustrate, these linear models explain only ~3.7% of the variance in popularity, which suggests that the actual relationship between audio features and popularity is fundamentally non-linear. Also, just because a song is louder does not necessarily translate to its being more popular. It was also seen that Ridge Regression, even with strong regularization, produced results identical to OLS, and hence, the issue was not overfitting but that linear models are just too simple to capture the complexity of musical trends.

**Figure 3: Linear Model Coefficients vs. Ridge Coefficients:**

| --- Model Performance ---      |                  |           |             |
|--------------------------------|------------------|-----------|-------------|
| OLS RMSE: 21.0072              |                  |           |             |
| OLS R2 Score: 0.0369           |                  |           |             |
| -----                          |                  |           |             |
| Ridge RMSE: 21.0072            |                  |           |             |
| Ridge R2 Score: 0.0369         |                  |           |             |
| --- Coefficient Comparison --- |                  |           |             |
|                                | Feature          | OLS_Coeff | Ridge_Coeff |
| 0                              | duration_ms      | -0.758402 | -0.758391   |
| 1                              | danceability     | 1.628504  | 1.628466    |
| 2                              | energy           | -1.443664 | -1.443610   |
| 3                              | loudness         | 1.172673  | 1.172649    |
| 4                              | speechiness      | -1.639507 | -1.639483   |
| 5                              | acousticness     | -0.505627 | -0.505602   |
| 6                              | instrumentalness | -3.240451 | -3.240406   |
| 7                              | liveness         | 0.031094  | 0.031079    |
| 8                              | valence          | -2.198006 | -2.197950   |
| 9                              | tempo            | 0.199778  | 0.199768    |

**Random Forest Performance:** Moving to the Random Forest Regressor (with 100 estimators) resulted in a dramatic improvement in predictive power.

- **Random Forest R^2 Score:** 0.4586
- **RMSE:** 15.75

The Random Forest model explains around 46% of the variance, over a 12x improvement over the linear baseline. This supports our hypothesis that popularity is driven by specific, non-linear combinations of features (e.g., a track being both high energy and having a certain duration) rather than global linear trends.

**Feature Analysis:** The Random Forest model allows us to inspect which features drive these predictions.

**Figure 4: Feature Importance (Random Forest)**

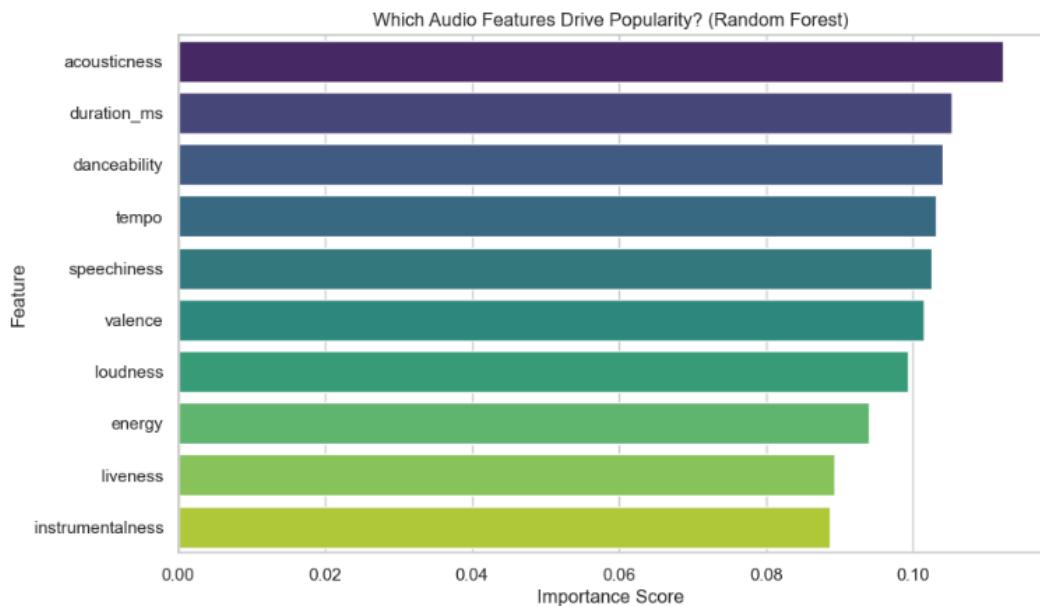


Figure 4 shows that Acousticness and Duration were the most influential predictors, followed by Danceability. Surprisingly, Loudness and Energy, which were highly intercorrelated in our heatmap, became secondary factors. That would suggest that the "mood" of a song - acoustic vs. electronic - along with the structure (duration) of the track, is more decisive for streaming success than raw volume or intensity.

**Discussion of Prediction Errors:** To understand where the model still struggles, I visualized the predicted vs. actual values.

**Figure 5: Actual vs. Predicted Popularity Scatter Plot**

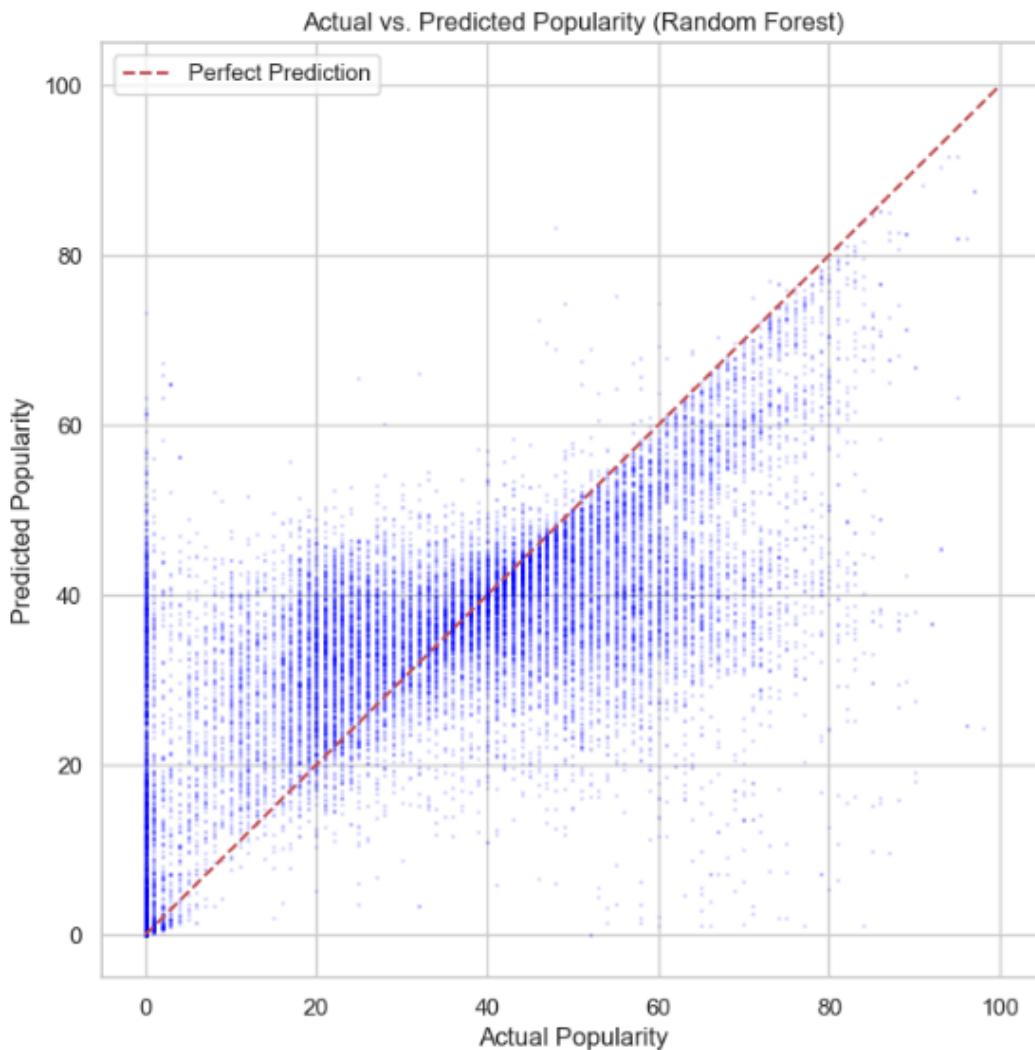


Figure 5 highlights an important insight: the dataset is zero-inflated. The solid vertical line of blue dots at Actual Popularity = 0 shows that the model often predicts a popularity score of 20-40 for songs that actually have 0 popularity.

- Interpretation: The model sees a song with "hit potential" audio features, such as correct length and high danceability, and thus predicts that it should be popular. However, the actual popularity is 0. Such a difference probably occurs for reasons external to what's available in this dataset, like fame of the artist, size of the marketing budget, or playlist position. A song can sound mathematically perfect yet utterly bomb if nobody actually hears it.

## Conclusion

In this project, I set out to predict the popularity of Spotify tracks using audio features. By comparing linear and non-linear machine learning models, I discovered that popularity is not a simple sum of a song's parts.

### Key Findings:

- Linearity is a Myth: The failure of the OLS and Ridge models ( $R^2 \sim 0.03$ ) demonstrates that there is no direct linear formula for a hit song. Simply making a track "louder" or "faster" does not guarantee popularity.
- Complexity Matters: The Random Forest model achieved an  $R^2$  of 0.46, a significant improvement. This would therefore suggest that the "recipe" for a popular song is more complex in nature. For instance, perhaps a high-energy song needs to be short (low duration) to make it a success, while a low-energy song requires high acousticness.
- The "Zero" Problem: A major limitation identified was the "zero-inflated" nature of the data. The model struggled to differentiate between "bad songs" and "good songs that just had no marketing."

**What I Learned:** Through this experiment, I learned the importance of model selection. While mathematically elegant, Ridge Regression handles the correlated features (such as Energy and Loudness) but cannot force a linear model to fit nonlinear data. I also learned that in datasets from real-world applications, external factors like marketing are often just as important to predict the outcome as the intrinsic features, such as audio quality.

## References & Acknowledgement

### References:

GeeksforGeeks. "Random Forest Regression in Python." *GeeksforGeeks*, 14 June 2019,

[www.geeksforgeeks.org/machine-learning/random-forest-regression-in-python/](https://www.geeksforgeeks.org/machine-learning/random-forest-regression-in-python/).

Accessed 5 Dec. 2025.

Pandya, Maharshi. "Spotify Tracks Dataset." *Www.kaggle.com*, 2022,

[www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset?resource=download](https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset?resource=download).

Accessed 5 Dec. 2025.

**AI Acknowledgement:** I, Muhammad Abdul Rafey, acknowledge the use of AI in completing this assignment and would like to provide a brief explanation of how I utilized AI, specifically ChatGPT, as a tool to support my work. It assisted me throughout the project including adding comments on my code and helped me fix grammar issues in my report writing. I also used it to debug code for me.

Prompt: I need help debugging this code:

Code:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
print("--- Tuning Ridge Regression ---")
param_grid = {'alpha': [0.1, 1.0, 10.0, 100.0, 1000.0]}
ridge_cv = GridSearchCV(Ridge(), param_grid, cv=5, scoring='neg_mean_squared_error')
ridge_cv.fit(X_train_scaled, y_train)
print(f"Best Ridge Alpha: {ridge_cv.best_params_['alpha']}")
best_ridge = ridge_cv.best_estimator_
y_pred_ridge_tuned = best_ridge.predict(X_test_scaled)
print(f"Tuned Ridge RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_ridge_tuned)):.4f}")
print(f"Tuned Ridge R2: {r2_score(y_test, y_pred_ridge_tuned)}")
print("\n--- Training Random Forest (Non-Linear Model) ---")
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1) #
n_jobs=-1 uses all CPU cores
rf_model.fit(X_train_scaled, y_train)
y_pred_rf = rf_model.predict(X_test_scaled)

print(f"Random Forest RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_rf)):.4f}")
print(f"Random Forest R2: {r2_score(y_test, y_pred_rf)}")
```

Error:

```
Cell In[5], line 17 print(f"Tuned Ridge R2: {r2_score(y_test, y_pred_ridge_tuned)):.4f}") ^  
SyntaxError: f-string: unmatched ')'
```

Response:

Your error is **purely a syntax issue** — there's an **extra parenthesis** in this line:  
`print(f"Tuned Ridge R2: {r2_score(y_test, y_pred_ridge_tuned)):.4f}")`

Specifically, this part is wrong:

```
r2_score(y_test, y_pred_ridge_tuned))  
          ^ extra parenthesis
```

 **Corrected Code**

Replace that line with:

```
print(f"Tuned Ridge R2: {r2_score(y_test, y_pred_ridge_tuned):.4f}")
```

 **Here is your entire corrected block**

```
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model_selection import GridSearchCV  
  
# --- Experiment 1: Tuning Ridge Regression ---  
print("--- Tuning Ridge Regression ---")  
param_grid = {'alpha': [0.1, 1.0, 10.0, 100.0, 1000.0]}  
ridge_cv = GridSearchCV(Ridge(), param_grid, cv=5, scoring='neg_mean_squared_error')  
ridge_cv.fit(X_train_scaled, y_train)  
  
print(f"Best Ridge Alpha: {ridge_cv.best_params_['alpha']}")  
best_ridge = ridge_cv.best_estimator_  
y_pred_ridge_tuned = best_ridge.predict(X_test_scaled)  
  
print(f"Tuned Ridge RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_ridge_tuned)):.4f}")  
print(f"Tuned Ridge R2: {r2_score(y_test, y_pred_ridge_tuned):.4f}")  
  
# --- Experiment 2: Random Forest Regressor ---  
print("\n--- Training Random Forest (Non-Linear Model) ---")  
rf_model = RandomForestRegressor(n_estimators=100, random_state=42, n_jobs=-1)  
rf_model.fit(X_train_scaled, y_train)  
y_pred_rf = rf_model.predict(X_test_scaled)  
  
print(f"Random Forest RMSE: {np.sqrt(mean_squared_error(y_test, y_pred_rf)):.4f}")  
print(f"Random Forest R2: {r2_score(y_test, y_pred_rf):.4f}")
```

 **After this fix, your code will run normally.**

Prompt: Help me understand this Error:

Code:

```
importance_df = pd.DataFrame({
    'Feature': feature_cols,
    'Importance': rf_model.feature_importances_
}).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
plt.title('Which Audio Features Drive Popularity? (Random Forest)')
plt.xlabel('Importance Score')
plt.show()

plt.figure(figsize=(8, 8))
plt.scatter(y_test, y_pred_rf, alpha=0.1, color='blue', s=1)
plt.plot([0, 100], [0, 100], '--r', label='Perfect Prediction') # Identity line
plt.xlabel('Actual Popularity')
plt.ylabel('Predicted Popularity')
plt.title('Actual vs. Predicted Popularity (Random Forest)')
plt.legend()
plt.show()
Error:
C:\Users\mabdu\AppData\Local\Temp\ipykernel_5908\836118973.py:9: FutureWarning: Passing
palette without assigning hue is deprecated and will be removed in v0.14.0. Assign the y
variable to hue and set legend=False for the same effect. sns.barplot(x='Importance',
y='Feature', data=importance_df, palette='viridis')
```

Response:

Here's the explanation — **your code is fine and will run**, but Seaborn is giving you a **FutureWarning**, meaning:

"This works right now, but in a future version of Seaborn, this will break unless you change it."

### ✓ Why the Warning Happens

You're doing:

```
sns.barplot(x='Importance', y='Feature', data=importance_df, palette='viridis')
```

Seaborn says:

You used a **palette**, but

You did **not** specify a **hue** argument.

In future versions, Seaborn wants palettes to be tied to a categorical **hue**. Otherwise, it doesn't know how to apply the palette safely.

## Two Correct Ways to Fix the Warning

### Fix Option 1 — Assign `Feature` to `hue` (Recommended)

This gives each bar its own color (works well for feature importance).

```
sns.barplot(  
    x='Importance',  
    y='Feature',  
    data=importance_df,  
    hue='Feature',  
    palette='viridis',  
    dodge=False,  
    legend=False  
)
```

- ✓ Uses the palette
- ✓ Removes the warning
- ✓ Colors each feature bar uniquely

### Fix Option 2 — Use a Single Color Instead of a Palette

If you don't want multicolored bars:

```
sns.barplot(  
    x='Importance',  
    y='Feature',  
    data=importance_df,  
    color='skyblue'  
)
```

- ✓ No warning
- ✓ Simpler visualization

## Why Seaborn Warns You

Starting in Seaborn v0.14+:

`palette=...` must be linked to a `hue=` argument.

Otherwise Seaborn doesn't know what the palette is intended to map to.

This is why it suggests:

"Assign the `y` variable to `hue` and set `legend=False` for the same effect."

### ⌚ What Should You Use for a Feature Importance Plot?

**Best practice:** Each feature gets its own color (Option 1).  
It becomes easier to visually separate features.

Use this:

```
sns.barplot(  
    x='Importance',  
    y='Feature',  
    data=importance_df,  
    hue='Feature',  
    palette='viridis',  
    dodge=False,  
    legend=False  
)
```

If you drop this into your code, the warning will disappear.

## Source Code

Github: <https://github.com/mabdulrafey/ITCS3156-Final-Project>