Appendix

December 12, 2025

1 Introduction

This document serves as the appendix to the main project report and provides detailed methods, analyses, figures, and supplemental explanations that support the results presented therein. The appendix includes expanded descriptions of exploratory data analysis, preprocessing steps, regression models, classification techniques, clustering diagnostics, and neural network experiments. All materials here are intended to offer full methodological transparency and reproducibility for the Spotify track analysis project.

A Appendix A: Additional Analyses and Supporting Material

Aside: All code referenced in the following subsection was already submitted as part of the Project Check-In and, per the project specifications, does not need to be resubmitted.

A.1 Exploratory Data Analysis

For our exploratory data analysis (EDA), we began by examining the structure of the Spotify dataset, including the number of observations, feature types, and any missing or duplicated entries. We generate summary statistics for numerical features to understand their ranges, distributions, and potential outliers.

As part of our exploratory data analysis, we generated histogram plots for several of the major audio features in the dataset, including danceability, energy, loudness, valence, and tempo. These histograms (Figure 1) helped us understand how each feature is distributed across the dataset and revealed that many musical characteristics have non-uniform or skewed distributions. Understanding these distributions is important for constructing a similarity-based recommendation system, as features with narrow or concentrated ranges may have less discriminative power when comparing tracks.

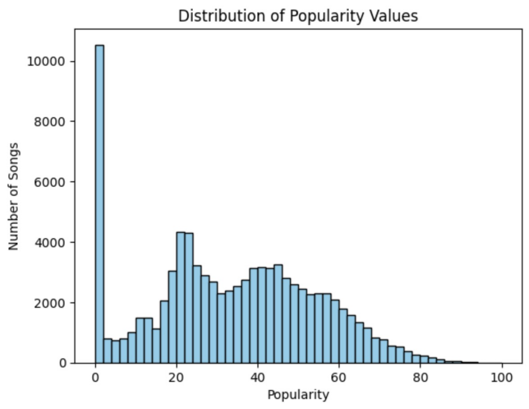

Figure 1: Histogram distributions of selected numerical audio features in the Spotify dataset.

To better understand how listener preferences vary across musical categories, we examined the relationship between genre and popularity. The bar chart in Figure 2 shows the average popularity for each genre represented in the dataset. This visualization highlights which genres tend to receive higher user engagement on Spotify and also illustrates the uneven distribution of popularity across categories. Observing this imbalance further supports our decision to exclude popularity as a feature for recommendation, as it reflects platform-wide trends rather than individual user taste.

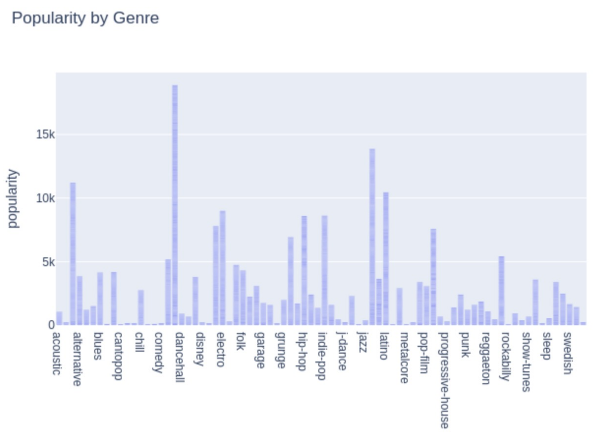

Figure 2: Relationship between Genre and Popularity.

To investigate how the audio features relate to one another, we generated a correlation heatmap, shown in Figure 3. This visualization highlights the strength and direction of pairwise relationships between numerical features in the dataset. Several strong correlations are immediately apparent, such as the positive relationship between loudness and energy, as well as between valence and danceability. These patterns indicate that certain musical characteristics tend to co-occur across tracks. Understanding these relationships is important for a similarity-based recommendation system, since highly correlated features contribute similar information when comparing songs.

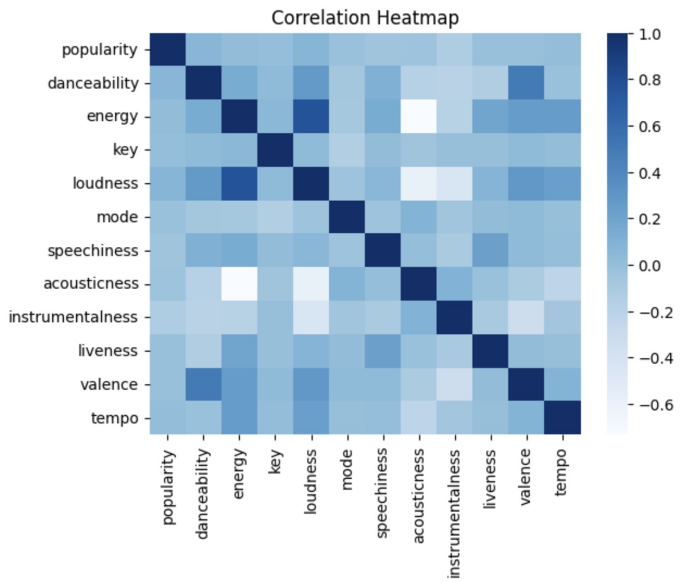

Figure 3: Correlation heatmap of numerical Spotify audio features.

To further explore the relationships identified in the correlation heatmap, we created scatterplots for pairs of features that exhibited strong positive correlations. Figures 4 and 5 show two such examples: loudness versus energy, and valence versus danceability. Both plots reveal clear upward trends, indicating that tracks with higher loudness generally have higher energy levels, and songs with greater valence tend to be more danceable. These visual patterns reinforce the idea that certain audio characteristics naturally cluster together, which is valuable information when measuring similarity between songs in a recommendation system.

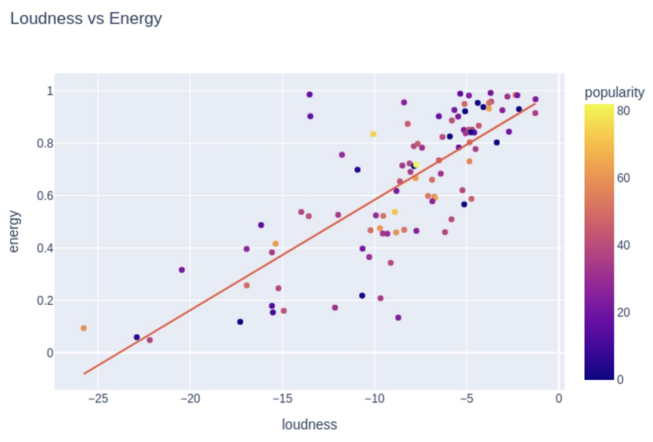

Figure 4: Scatterplot of Loudness versus Energy.

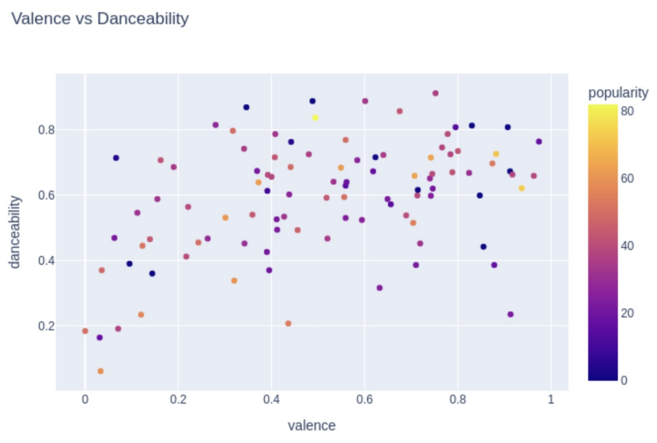

Figure 5: Scatterplot of Valance verses Danceability.

A.2 Data Pre-processing and Feature Engineering

Before building our recommendation system, we performed several preprocessing steps to ensure that the data set was clean, consistent, and suitable for feature-based similarity comparisons. We started by checking for duplicated songs using the `track_id` column. The data set contained a substantial number of duplicates, so we removed all repeated entries by keeping only the first occurrence of each unique `track_id`. This reduced the data set from 114,000 rows to 89,741 unique tracks, ensuring that no song was overly tagged during model development.

Listing 1: Duplicate detection and cleanup

```
dupe_mask = df['track_id'].duplicated(keep=False)
dupes = df[dupe_mask].sort_values('track_id')
df = df.drop_duplicates(subset='track_id', keep='first')
```

Next, we removed columns that did not contribute meaningful musical information. Specifically, we dropped the `track_id` and `Unnamed: 0` columns, since they are identifier fields rather than audio or metadata features and therefore cannot be used to compute similarity between songs.

Listing 2: Column Removal

```
df = df.drop('track_id', axis=1)
df = df.drop('Unnamed: 0', axis=1)
```

After cleaning the dataset, we evaluated the remaining features for their suitability in a recommendation setting. Since popularity does not reflect a user's personal preference and is computed using an opaque algorithm specific to Spotify, we excluded this column from all downstream analysis. This ensures that recommendations are based purely on the audio characteristics of songs rather than global listening trends.

Although our project did not require creating new engineered features, the preprocessing steps above established a consistent and reliable feature set that will later be standardized.

A.3 Regression Analysis

In order to understand how individual audio features contribute to the overall musical characteristics of a track, we applied regression analysis using `energy` as the target variable. Energy is a continuous numeric feature that reflects the perceptual intensity of a song, making it a good candidate for examining linear relationships in the dataset.

We first computed the correlation between energy and the other numerical features to identify potential predictors. Loudness showed the strongest correlation with energy, followed by danceability and valence. These correlations provided an initial indication that a linear model might capture part of the relationship between audio features and energy.

To quantify these relationships, we trained a multiple linear regression model using seven audio features as predictors. The model was trained on 80% of the data and evaluated on the remaining 20%. The resulting validation metrics indicated that the linear model explained a moderate portion of the variance in energy, with R^2, MAE, and RMSE values suggesting that the model was capturing general trends but not all complexities of the feature space. Figure 6 shows a scatterplot comparing the model's predicted energy values to the actual values in the validation set. The clustering around the diagonal line indicates reasonable predictive performance, though with visible deviation reflecting non-linear structure in the data.

We also examined the learned regression coefficients to interpret feature importance. The magnitude and sign of the coefficients aligned with the earlier correlation analysis: loudness had the strongest positive effect on energy, supporting the idea that louder tracks tend to feel more energetic.

To evaluate whether regularization was necessary, we trained a Ridge regression model with an α value of 2.5. Ridge regression slightly reduced overfitting, bringing the training and validation R^2 scores closer together. However, the performance improvements were small, indicating that multicollinearity among our selected features was present but not severe. This suggests that while regularization can stabilize the model, the linear feature set did not require heavy penalization to perform effectively.

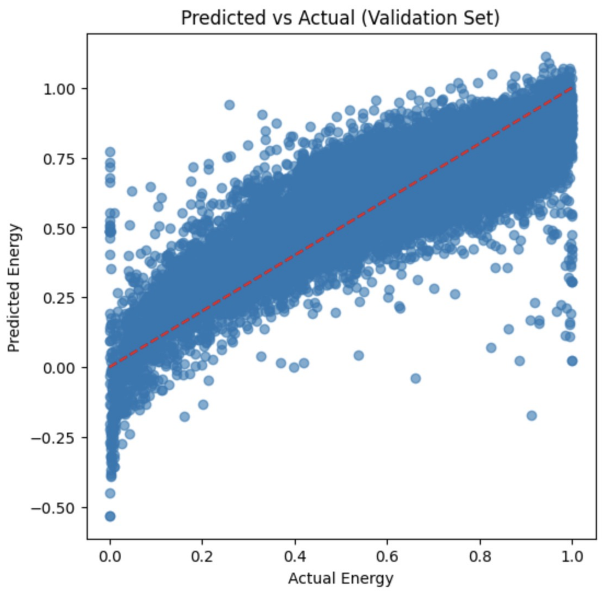

Figure 6: Predicted versus actual energy values for the linear regression model on the validation set. The diagonal dashed line represents perfect predictions.

A.4 Logistic Regression

To further analyze classification patterns within the dataset, we applied logistic regression to predict whether a song contains explicit content. The target variable `explicit` is binary, making logistic regression a natural choice for this task. We used eight audio and metadata features, including popularity, danceability, loudness, energy, instrumentalness, speechiness, valence, and tempo, as predictors.

A key challenge in this task is that explicit songs make up a small minority of the data set, resulting in a class imbalance. To address this, we assigned higher weights to the minority (explicit) class during training, allowing the model to better identify explicit tracks without being overwhelmed by the majority class.

After training the model in an 80/20 train–validation split, we evaluated performance using a confusion matrix (Figure 7), classification report metrics and general prediction accuracy. The confusion matrix revealed that the model correctly identified a large proportion of non-explicit songs while achieving reasonable true positive performance on explicit songs, despite the imbalance. This demonstrates that class weighting improved the model's sensitivity to explicit content.

To further assess predictive quality, we examined the model's ROC curve and computed the area under the curve (AUC), shown in Figure 8. The resulting AUC score indicated that the logistic model performed substantially better than random guessing and was effective at separating explicit from non-explicit songs based on the selected features.

Finally, we performed 5-fold stratified cross-validation to verify the stability of the model. The AUC and accuracy scores across folds remained consistent, indicating that the model generalizes reliably and is not overfitting. Regularization did not significantly impact performance, suggesting that multicollinearity among the selected features was limited, and the logistic model remained stable without heavy penalization, similar to Linear Regression.

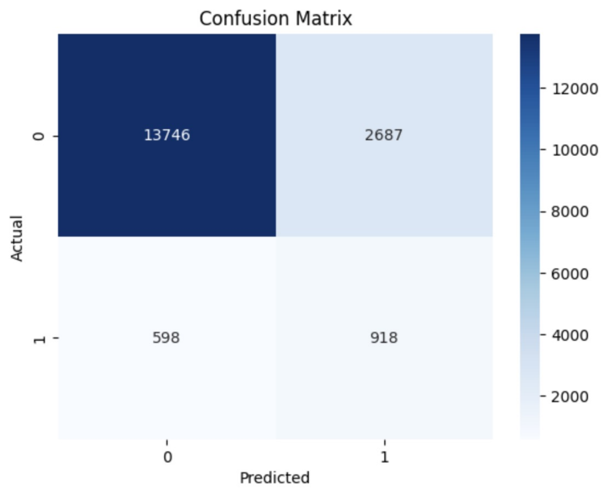

Figure 7: Confusion matrix for logistic regression model predicting explicit content.

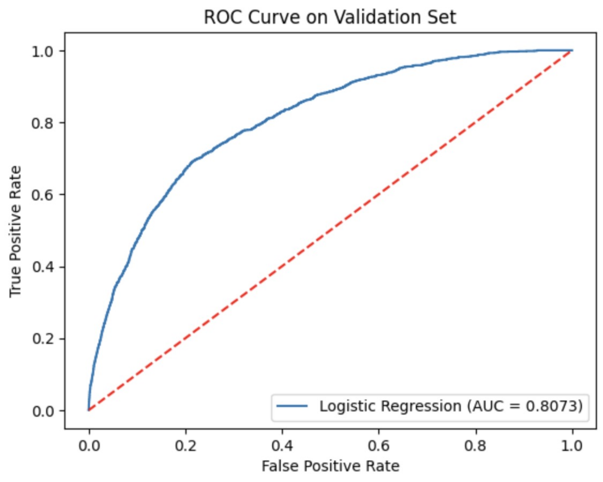

Figure 8: ROC curve for logistic regression on the validation set. The dashed line represents random guessing.

A.5 KNN, Decision Trees, and Random Forest

NOTE: i didn't do decision tree / random forest -kevin

We applied the K-Nearest Neighbors (KNN) classifier to predict whether a song contains explicit content, using the same set of audio and metadata features as in the logistic regression analysis. Because KNN relies on distance computations, we standardized all feature values prior to training to ensure that no single feature dominated the Euclidean distance metric.

The model was first trained using $k = 3$, which provided a good balance between model flexibility and stability. After fitting the classifier on the training set, we evaluated performance on the validation set. The confusion matrix shown in Figure 9 reveals that the KNN model achieved strong performance on the majority (non-explicit) class, with a high true negative rate of 0.9709. However, the model struggled to correctly identify explicit songs, resulting in a lower true positive rate of 0.2876. This imbalance is expected, as the dataset contains far fewer explicit songs and KNN tends to favor the majority class when class densities differ.

Despite this, the overall prediction accuracy was high (91.31%), although this figure alone is misleading due to the strong class imbalance. To better assess the classifier's discriminative ability, we computed the ROC curve and measured the area under the curve (AUC), shown in Figure 10. The resulting AUC score indicated moderate performance, reflecting that while KNN is effective at identifying non-explicit songs, it is less capable of distinguishing explicit content based solely on the available features.

To evaluate the stability of the model, we conducted 5-fold stratified cross-validation using $k = 5$. The resulting accuracy and AUC scores were consistent across folds, indicating that the classifier generalizes reliably. However, cross-validation also confirmed that the low recall on the explicit class is inherent to the dataset's imbalance rather than an artifact of a particular train–test split.

Table 1 summarizes the 5-fold cross-validation accuracy and AUC values for the KNN classifier, demonstrating stable performance across folds.

Fold	**Accuracy**	**AUC**
1	0.914	0.740
2	0.909	0.709
3	0.906	0.687
4	0.914	0.754
5	0.914	0.715
Mean	**0.912**	**0.721**

Table 1: 5-fold cross-validation results for the KNN classifier.

Overall, KNN provided reasonable baseline performance but was limited in detecting minority-class instances. This suggests that distance-based methods may not be ideal for this task unless additional techniques—such as oversampling, synthetic data generation, or alternative distance metrics—are introduced.

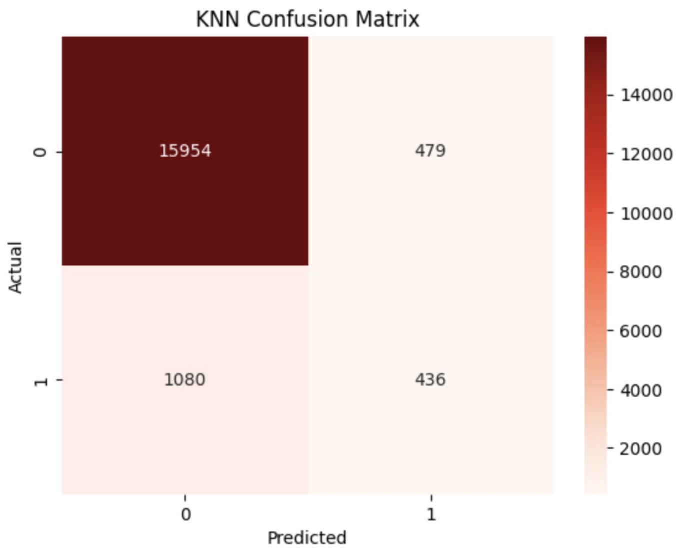

Figure 9: Confusion matrix for the KNN classifier ($k = 3$) on the validation set.

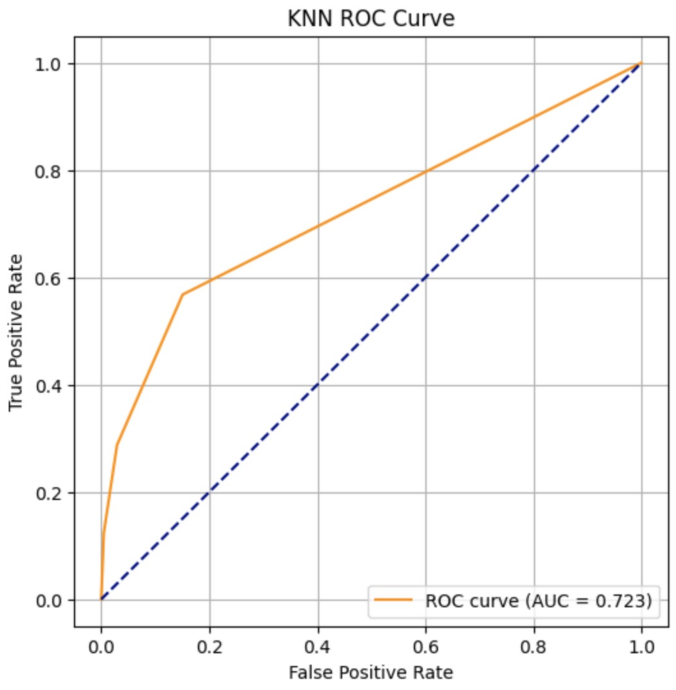

Figure 10: ROC curve for the KNN classifier on the validation set. The dashed line represents random guessing.

A.6 PCA and Clustering

To explore groupings and latent structure within the Spotify dataset, we applied Principal Component Analysis (PCA) and K-Means clustering. Since clustering algorithms are sensitive to feature scale, we first standardized all numerical predictor variables using a `StandardScaler`. This ensured that features measured on different scales contributed equally to the distance computations used by K-Means.

We then applied the K-Means algorithm to the standardized feature set. Because the optimal number of clusters is not known beforehand, we used two diagnostic tools: the elbow method and silhouette scores. The elbow plot (Figure 11) shows the distortion, or within-cluster sum of squares, for values of k ranging from 2 to 10. The point at which the curve begins to flatten—commonly referred to as the "elbow" suggests a suitable choice for k. Similarly, the silhouette score plot (Figure 12) measures how well-separated the clusters are for each value of k. Higher silhouette scores indicate more coherent and better-defined clusters.

Both diagnostics suggested that $k = 6$ provided a good balance between low distortion and high separation quality. After selecting $k = 6$, we fit the final K-Means model and computed cluster assignments for all standardized samples. Examining the cluster sizes confirmed that no cluster was excessively small or disproportionately large.

To visualize the clusters in two dimensions, we applied PCA to the standardized feature matrix and extracted the first two principal components, which captured the largest amount of variance. Figure 13 shows a 2D scatterplot of the samples in PCA space, colored according to their cluster label. Although PCA reduces the dimensionality and may not retain all information, the plot reveals meaningful separation between several clusters, indicating that the underlying audio features contain structure that can be grouped into musically distinct categories.

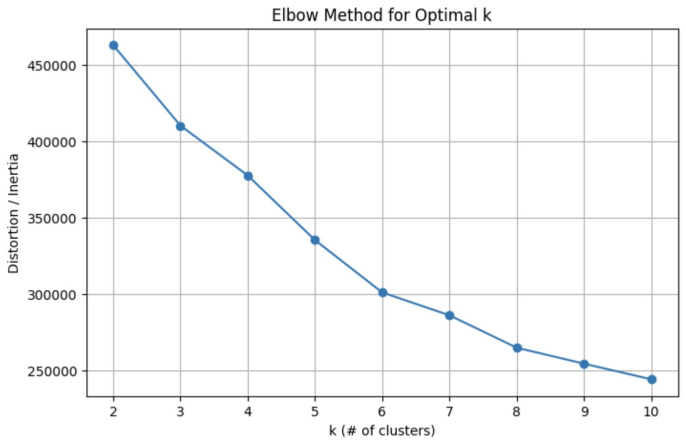

Figure 11: Elbow method showing distortion values for k between 2 and 10. The point where the curve begins to flatten suggests a suitable number of clusters.

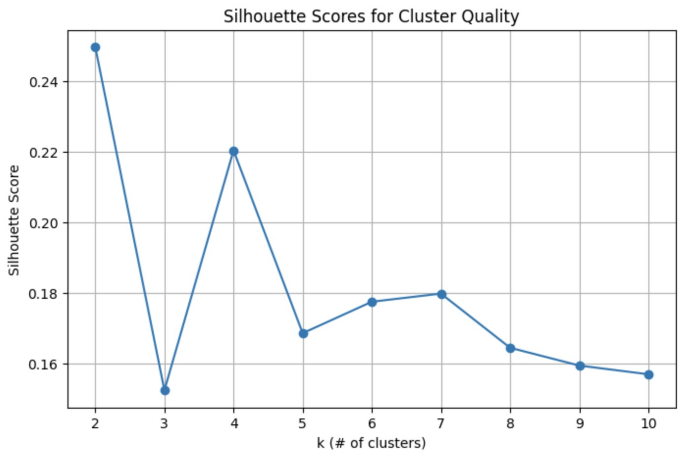

Figure 12: Silhouette scores for k between 2 and 10. Higher values indicate more well-defined clusters.

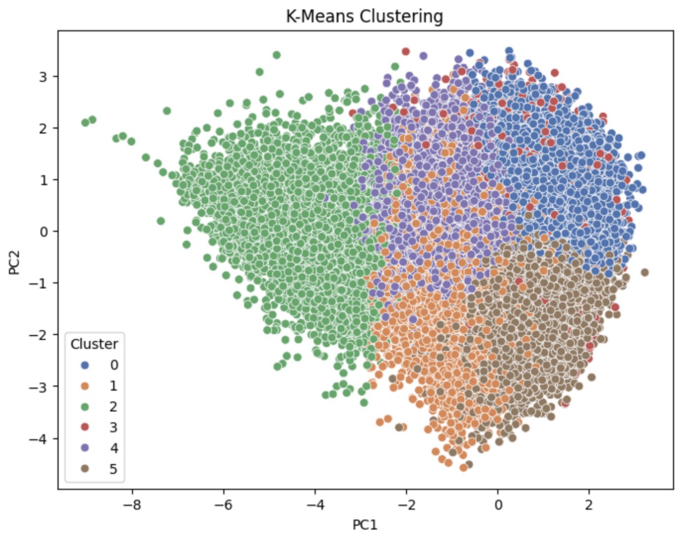

Figure 13: K-Means clustering visualized in the first two principal components. Colors represent the six cluster assignments.

A.7 Neural Network Experiments

To capture nonlinear relationships among audio features and generate meaningful low-dimensional representations of songs, we implemented a neural network autoencoder. The autoencoder learns a compressed embedding of each track by training the network to reconstruct the original input from a reduced latent space. This approach is particularly valuable for recommendation systems, as songs that produce similar embeddings tend to share similar musical characteristics across multiple dimensions.

Before training, we standardized all numerical features in the dataset to ensure consistent scaling across input dimensions. The autoencoder architecture consisted of an encoder with two fully connected layers that reduced the input vector to a 12-dimensional embedding, followed by a symmetric decoder responsible for reconstructing the original feature vector. ReLU activations were used throughout the network, and training was performed for 40 epochs using the Adam optimizer with a learning rate of 0.001 and mean squared error (MSE) as the loss function.

Figure 14 shows the error-versus-epoch curve. The steady decline in reconstruction loss, decreasing from approximately 0.10 in the first epoch to below 0.006 by epoch 40, demonstrates stable convergence and effective learning. Table 3 summarizes the reconstructed performance metrics, including MSE, MAE, RMSE, and the R^2 score. These metrics indicate that the autoencoder achieved exceptionally high reconstruction fidelity, with an R^2 value of 0.9944 and a very low reconstruction error across all measures.

To provide further detail on the autoencoder's learning progression, Table 2 reports the epoch-by-epoch training loss values in a color-coded side-by-side format (Epochs 1–20 and Epochs 21–40). The progressive lightening of cell colors visually reinforces the monotonic decrease in training loss and highlights the network's consistent improvement throughout training.

After training, we used the learned embeddings to construct a similarity-based recommendation system. For any selected track, we computed its latent embedding and measured cosine similarity against all other embeddings to identify the most similar songs. Because the embedding space reflects nonlinear relationships among multiple audio features, this method naturally identifies songs with similar musical structure, rather than relying on single-feature comparisons.

Overall, the autoencoder proved to be an effective tool for extracting compact, information-rich song representations. These latent embeddings provide a strong foundation for music recommendation and complement the insights gained from earlier regression, classification, and clustering analyses.

Epoch	Loss	Epoch	Loss
1	0.10347	21	0.00651
2	0.02057	22	0.00649
3	0.01472	23	0.00639
4	0.01218	24	0.00630
5	0.01088	25	0.00626
6	0.01016	26	0.00623
7	0.00970	27	0.00613
8	0.00913	28	0.00611
9	0.00871	29	0.00611
10	0.00827	30	0.00600
11	0.00795	31	0.00593
12	0.00766	32	0.00585
13	0.00735	33	0.00582
14	0.00717	34	0.00575
15	0.00705	35	0.00567
16	0.00709	36	0.00574
17	0.00681	37	0.00555
18	0.00675	38	0.00550
19	0.00672	39	0.00549
20	0.00653	40	0.00545

Table 2: Side-by-side comparison of autoencoder training loss across 40 epochs. Darker shades represent higher loss values and lighter shades represent lower loss values.

Metric	Value
MSE	0.005568
MAE	0.041211
RMSE	0.074618
R^2	0.994432

Table 3: Autoencoder reconstruction performance metrics.

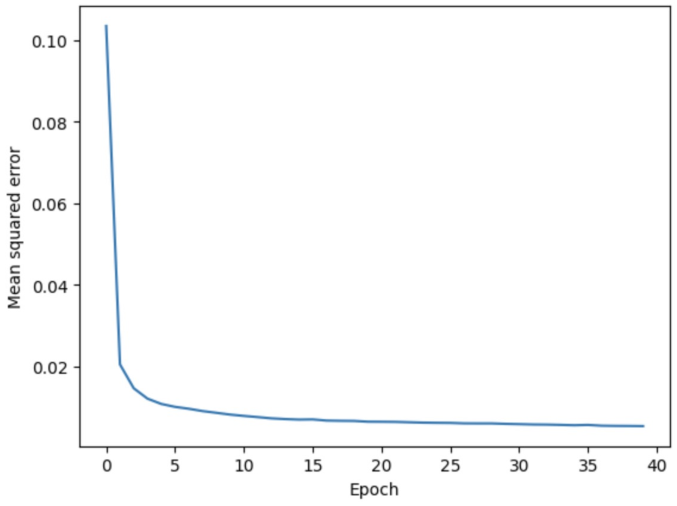

Figure 14: Training loss curve for the autoencoder over 40 epochs.

A.8 Hyperparameter Tuning

Throughout the project, we performed hyperparameter tuning across several models in order to improve performance, ensure model stability, and validate that our chosen configurations generalized well beyond a single train–test split. Each algorithm required different hyperparameters to be explored, and the decisions were guided by empirical results, diagnostic plots, and cross-validation procedures.

Clustering (K-Means)

One of the most important hyperparameters in K-Means clustering is the number of clusters k. To determine an appropriate value, we tested values of k ranging from 2 to 10 and used two evaluation methods:

- **Elbow Method:** We plotted distortion (within-cluster sum of squares) across different values of k and observed where the curve began to flatten.
- **Silhouette Scores:** We computed silhouette coefficients for each k to measure cluster cohesion and separation.

Both diagnostics indicated that $k = 6$ provided a strong balance between separation quality and cluster compactness, and this value was selected for the final clustering model.

Linear and Regularized Regression

For linear regression, the primary goal was to analyze which features contributed significantly to predicting track energy. No regularization was applied in the standard model, allowing the coefficients to reflect the natural linear relationships in the dataset.

We then tuned the Ridge regression model by adjusting the regularization strength parameter α. Several values were tested, and $\alpha = 2.5$ produced the best tradeoff between reducing coefficient variance and maintaining predictive performance. Ridge demonstrated only modest improvements over standard linear regression, suggesting low multicollinearity among features.

Logistic Regression

The logistic regression model required tuning primarily due to the strong class imbalance between explicit and non-explicit tracks. We adjusted the following hyperparameters:

- **Class Weights:** To increase sensitivity to explicit tracks, we applied class weights of {`0:1, 1:8`}, substantially improving recall for the minority class.
- **Maximum Iterations:** We increased `max_iter` to 500 to ensure convergence with weighted training.

After tuning, the logistic regression model achieved stable AUC and accuracy metrics, confirmed through 5-fold stratified cross-validation.

K-Nearest Neighbors

KNN performance depends heavily on the choice of k. We initially trained the model with $k = 3$, which provided a good baseline. To evaluate model stability, we ran stratified 5-fold cross-validation with $k = 5$, observing consistent accuracy and AUC values across folds. This indicated that the classifier was robust to moderate changes in k, although limited by class imbalance inherent in the dataset.

Neural Network Autoencoder

The autoencoder required tuning of several hyperparameters, including:

- **Embedding Dimension:** We selected a 12-dimensional latent space to balance compression and reconstruction quality.
- **Hidden Layer Widths:** The encoder and decoder each used a 64-unit hidden layer based on preliminary experiments.
- **Activation Functions:** ReLU was chosen for its stability and suitability for continuous-valued data.
- **Learning Rate:** The learning rate required dedicated tuning. Multiple values (`0.0001`, `0.001`, `0.01`) were systematically tested to determine which produced the best convergence behavior for the autoencoder.
- **Epochs:** Training for 40 epochs allowed the loss to stabilize without overfitting, as seen in Table 3 and Figure 14.
- **Batch Size:** A batch size of 64 balanced learning stability and computational efficiency.

To improve training stability and reconstruction quality, we conducted a systematic hyperparameter tuning experiment focused on determining the optimal learning rate. Three learning rates were evaluated: `0.0001`, `0.001`, and `0.01`. For each value, a fresh autoencoder model was trained from scratch for 30 epochs using the Adam optimizer and mean squared error (MSE) loss.

Table 4 summarizes the final validation losses for each tested learning rate. Among the three candidates, the highest-performing value was **0.01**, which achieved the lowest validation loss. This indicates that a moderately aggressive learning rate enabled efficient convergence without destabilizing training.

Learning Rate	Final Validation Loss
0.0001	0.0339
0.001	0.0101
0.01	**0.0022**

Table 4: Validation loss comparison across learning rates. The learning rate of 0.01 performed best.

The training and validation loss curves for each learning rate are shown in Figures 15 and 16. Lower learning rates (`0.0001` and `0.001`) converged slowly and often plateaued early, whereas the learning rate of `0.01` consistently reached the lowest loss values and demonstrated stable, monotonic convergence.

This experiment confirms that **a learning rate of 0.01 is optimal for the autoencoder**. Incorporating this result into the final configuration contributed significantly to the strong reconstruction performance reported in Table 2.

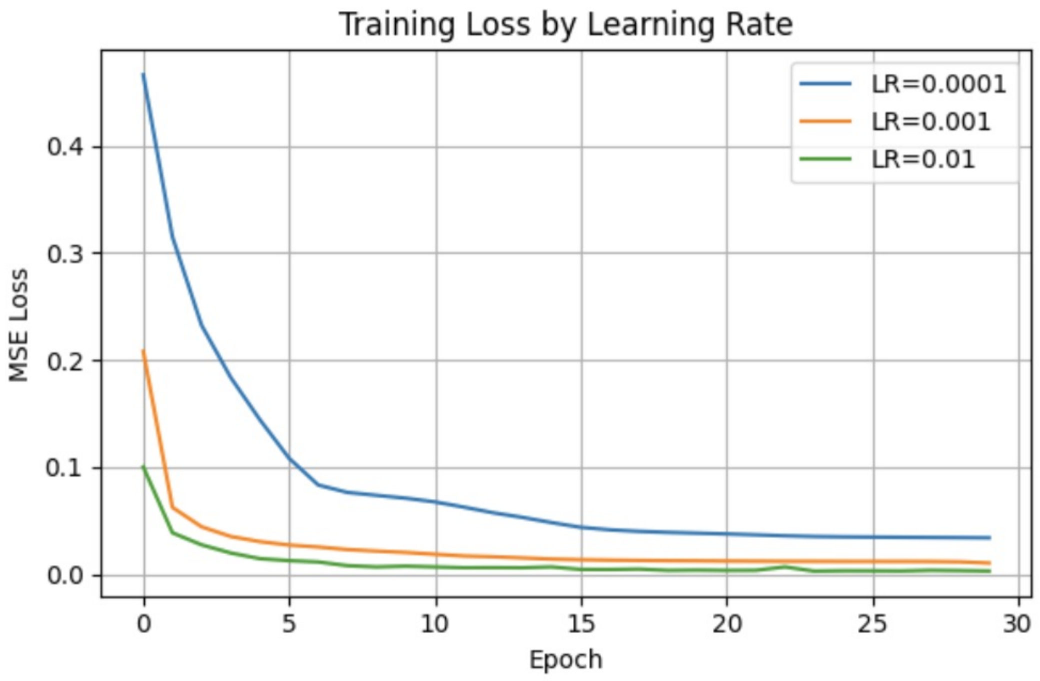

Figure 15: Training loss curves for each tested learning rate. The learning rate of 0.01 converges significantly faster.

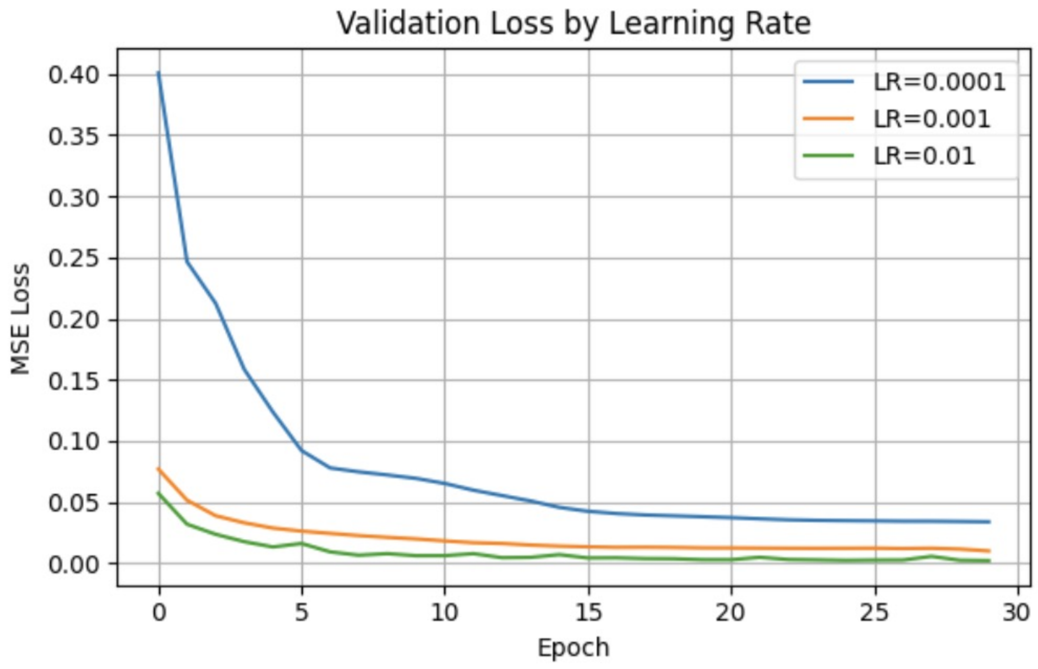

Figure 16: Validation loss curves comparing learning rates. The learning rate of 0.01 achieves the lowest final validation loss.

Summary

Across all models, hyperparameter tuning played a key role in improving predictive accuracy, stability, and interpretability. By systematically adjusting cluster sizes, regularization strengths, class weights, nearest-neighbor counts, and neural network architecture components, we ensured that each modeling approach was optimized for the characteristics of the Spotify dataset.