Spotify Songs Popularity Report

Data Set of Choice: Spotify

For this project, we analyzed the spotify-tracks-dataset provided by user maharshipandya. According to the Hugging Face platform, the data was collected and cleaned through Spotify's API and Python (not much further information on the method of collection was provided). The raw data set contains 114,000 entries. Each row corresponds to a song and its measured features. A link to the data can be found here.

The data set contains the following features:

- **track_id:** The Spotify ID for the track
- **artists**: The artists' names who performed the track. If there is more than one artist, they are separated by a ;
- **album_name:** The album name in which the track appears
- **track_name:** Name of the track
- **popularity:** The popularity of a track is a value between 0 and 100, with 100 being the most popular. The popularity is calculated by algorithm and is based, in the most part, on the total number of plays the track has had and how recent those plays are. Generally speaking, songs that are being played a lot now will have a higher popularity than songs that were played a lot in the past. Duplicate tracks (e.g. the same track from a single and an album) are rated independently. Artist and album popularity is derived mathematically from track popularity.
- **duration_ms:** The track length in milliseconds
- **explicit:** Whether or not the track has explicit lyrics (true = yes it does; false = no it does not OR unknown)
- **danceability:** Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable
- **energy:** Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale
- **key:** The key the track is in. Integers map to pitches using standard Pitch Class notation. E.g. 0 = C, 1 = C♯/D ♭ , 2 = D, and so on. If no key was detected, the value is -1
- **loudness:** The overall loudness of a track in decibels (dB)
- **mode:** Mode indicates the modality (major or minor) of a track, the type of scale from which its melodic content is derived. Major is represented by 1 and minor is 0
- **speechiness:** Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values

between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks

- **acousticness:** A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic
- **instrumentalness:** Predicts whether a track contains no vocals. "Ooh" and "aah" sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly "vocal". The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content
- **liveness:** Detects the presence of an audience in the recording. Higher liveness values represent an increased probability that the track was performed live. A value above 0.8 provides strong likelihood that the track is live
- **valence:** A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry)
- **tempo:** The overall estimated tempo of a track in beats per minute (BPM). In musical terminology, tempo is the speed or pace of a given piece and derives directly from the average beat duration
- **time_signature:** An estimated time signature. The time signature (meter) is a notational convention to specify how many beats are in each bar (or measure). The time signature ranges from 3 to 7 indicating time signatures of 3/4, to 7/4.
- **track_genre:** The genre in which the track belongs

Problem of Interest:

Given the data, our team was interested in addressing how the popularity of a song could be predicted by other features. To conduct this analysis, we performed exploratory data analysis to understand relationships between different variables. We then trained multiple models based on the concepts presented in class to determine which models would be able to address our problem.

Key Methodology that Addressed Problems and Why:

This report details our methodological approach and findings in developing predictive models for song popularity. Our analysis began with extensive exploratory data analysis (EDA), which revealed minimal direct correlations between individual features and popularity metrics, with maximum correlations of approximately 0.05. This early finding guided our subsequent modeling decisions, highlighting the need for sophisticated approaches capable of capturing complex feature interactions.

In response to these data characteristics, we implemented and evaluated multiple modeling approaches of increasing complexity. Our baseline K-Nearest Neighbors (KNN) implementation achieved 57.5% test accuracy, demonstrating the limitations of simple distance-based metrics in handling the high-dimensional nature of our dataset. Building upon these initial insights, we explored dimensionality reduction and clustering techniques through Principal Component Analysis (PCA). While these methods provided valuable perspectives on data variance and structure, they did not directly enhance our predictive capabilities due to the absence of clear popularity-related clustering patterns.

Our more advanced modeling efforts yielded significantly improved results. The Random Forest methodology demonstrated robust performance with 66% test accuracy and an AUC of 0.7085, effectively managing feature complexity through its ensemble approach of aggregated decision trees. However, the neural network architecture emerged as our most effective solution, achieving 77% test accuracy. This superior performance can be attributed to its sophisticated architecture, which incorporated ReLU activation functions in hidden layers and sigmoid activation in the output layer, enabling effective capture of non-linear relationships while maintaining strong generalization capabilities.

The comparative performance of these methodologies offers valuable insights into the nature of song popularity prediction. The neural network's success highlights the importance of modeling complex, non-linear feature interactions, while the Random Forest's performance demonstrates the value of ensemble methods in managing feature complexity. The limitations of simpler approaches underscore the inherent complexity of popularity prediction and the necessity for sophisticated modeling techniques.

Through the strategic implementation of these varied approaches, complemented by rigorous preprocessing protocols, we successfully developed a robust framework for predicting song popularity. The clear superiority of the neural network methodology suggests that future efforts in this domain should prioritize architectures capable of modeling complex feature relationships while maintaining computational efficiency and generalization capabilities.

Cross-Validation/Evaluation Metrics Used:

Results

In this section, we present the outcomes of the various models used to predict song popularity. These results highlight each method’s strengths and weaknesses and have guided our selection of the most effective approach.

Cross-validation

- For the KNN model, 5-fold cross-validation was used to validate model performance and generalizability.
- For the random forest and neural network models, we split the data into training, validation, and test sets.

Neural network results

- Test accuracy: 78%
  The neural network achieved the highest accuracy among all models tested, and it was able to effectively capture the non-linear relationships between features and popularity. The high accuracy suggests that the model was proficient in distinguishing between popular and non-popular songs.
- Training and validation loss:

The training and validation loss curves both showed a steady decrease over epochs, eventually converging. This shows that the model was learning effectively without overfitting and the model generalizes well to unseen data.
The decreasing loss trends also suggest that the model's learning rate (0.01) was well-tuned, and the network learned at an optimal pace. Had the learning rate been too high, we would have observed erratic loss curves, while a lower learning rate would have slowed convergence significantly.

- Model limitations:
  Neural networks are often considered black-box models, so it is difficult to interpret the exact contributions of each feature to the prediction.

Random forest results

- Validation accuracy: 66.5%
- Test accuracy: 66.0%
  The random forest model provided a solid performance, with an accuracy of 66% on the test set. This performance is notably better than KNN but falls short of the neural network.
- Test AUC: 0.7085
  The AUC score suggests that the model had fair predictive performance- it was reasonably good at distinguishing between popular and non-popular songs. The model's predictions are better than random guessing but there is still room for improvement.
- Model strengths:
  The random forest model was effective in handling the data's complexity and reducing the impact of outliers. It was able to aggregate the results of multiple decision trees, providing robustness against overfitting.
- Model weaknesses:
  The random forest model's accuracy was limited, possibly due to weak correlations between individual features and popularity. Its inability to capture more intricate patterns in the data prevented it from achieving higher accuracy.

KNN results

- Test accuracy: 56.8%
  The K-Nearest Neighbors (KNN) model struggled to accurately classify songs based on popularity, achieving a test accuracy of just 56.8%. This result is only marginally better than random guessing. KNN was not well-suited for the problem.
- 5-fold cross-validation AUC:
  - Scores: [0.5775, 0.5813, 0.5791, 0.5822, 0.5815]
  - Average AUC: 0.5803
    The low AUC scores indicate the model's limited ability to distinguish between popular and non-popular songs. There is poor classification performance.
- Confusion matrix for
  KNN: [[3883 4145]
  [3606 6314]]
  True positive rate: 63.6%
  True negative rate: 48.4%

The confusion matrix shows that KNN had difficulty distinguishing between the two classes- there was a high number of false positives and false negatives. The model struggles to identify non-popular songs accurately since it had a relatively low true negative rate.

- Model limitations:
  The KNN model's poor performance can be attributed to the Curse of Dimensionality and its reliance on distance metrics. It does not perform well when the data is high-dimensional or when features have a low correlation with the target variable.

Conclusions

Our project aims to predict a song's popularity based on its features. We leveraged various machine learning models including neural networks, random forests, and k-nearest neighbors (KNN).

The neural network model outperformed its counterparts, achieving a test accuracy of 77%. It was well-suited for capturing the non-linear relationships inherent in the data, which simpler models like KNN and random forest could not fully exploit. ReLU activation in the hidden layer and the sigmoid activation in the output layer helped the network learn complex patterns and produce reliable probabilistic outputs.

With a test accuracy of 66% and an AUC of 0.7085, the random forest model demonstrated a moderate ability to predict song popularity. While it handled non-linearities better than KNN, its performance was still limited by the weak correlations between individual features and popularity. The model's robustness to overfitting made it a reliable, albeit less accurate, alternative to the neural network.

The KNN model struggled with the task and had a test accuracy of 56.8% and an AUC of 0.5803. Its poor performance can be attributed to the Curse of Dimensionality and its reliance on distance metrics, which are less effective for high-dimensional data with weakly correlated features.

Predicting song popularity is a complex task that requires models capable of capturing non-linear interactions and subtle patterns within the data. The neural network's superior performance underscores the importance of using flexible and adaptive models for such tasks.

How to Use Code:

Steps to Run Code

1. Clone the github repository to run on your device locally.
2. Install dependencies.
3. Use the latest version of Python to avoid conflicts.
4. Run the code.

APPENDIX

Data Preprocessing and Cleaning

Our data preprocessing workflow implemented a comprehensive approach to ensure optimal data quality and compatibility with learning algorithms. The process consisted of several critical steps designed to address common data challenges while preserving valuable information.

Missing data handling required a nuanced approach. We identified and removed columns with minimal missingness, including 'artists', 'album_name', and 'track_name', as their contribution to the modeling process was determined to be negligible. For the remaining features, we employed strategic imputation methods—applying mean imputation for numeric variables to maintain their central tendencies, while categorical features received mode imputation to ensure consistent representation across the dataset.
Our duplicate management strategy operated on two distinct levels. The first level focused on removing exact duplicates where all column values matched, effectively eliminating redundant data points. The second level addressed track-specific duplicates by implementing a sophisticated aggregation process. For records sharing the same track_id, we combined genres using logical operations and calculated average values for numeric features such as popularity. This approach ensured the preservation of unique information while maintaining data integrity.

```
Num rows before removing duplicates: 114000
Num rows after removing exact duplicates: 113550
Num rows involved in ID duplicates (including original rows): 23809
Num rows after aggreagting inexact duplicates: 89741
```

To enhance data quality further, we implemented outlier detection and removal using the Z-score method, establishing a threshold of three standard deviations. This statistical approach effectively identified and addressed extreme values that could potentially compromise the accuracy of our analyses and machine learning models.

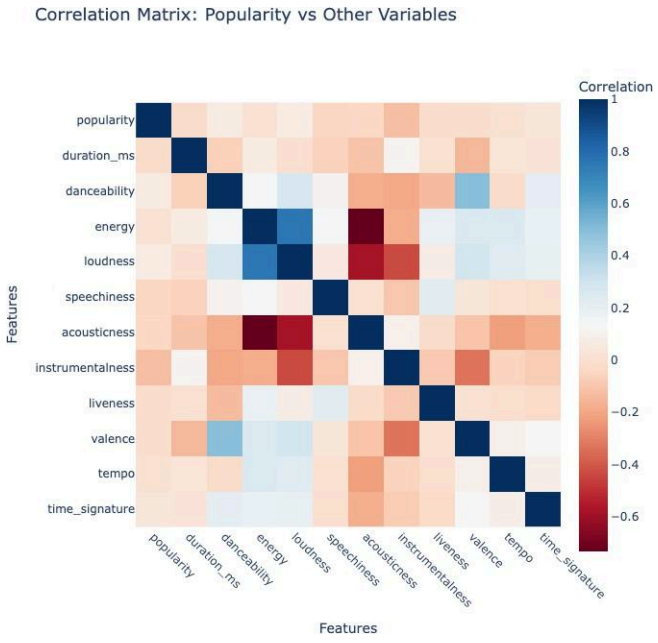
The final preprocessing phase focused on feature encoding to optimize our data for machine learning applications. We transformed categorical features (key, mode, track_genre) into binary columns through one-hot encoding, maximizing their utility while preserving interpretability. Additionally, we standardized boolean columns into integer representations (True → 1, False → 0) to ensure consistency across all features.

Exploratory Data Analysis

Our exploratory analysis revealed several key insights about relationships between musical features and their distributions within our dataset. Figure 1 presents our initial correlation analysis between track popularity and various audio features, showing notably weak correlations across all variables, with

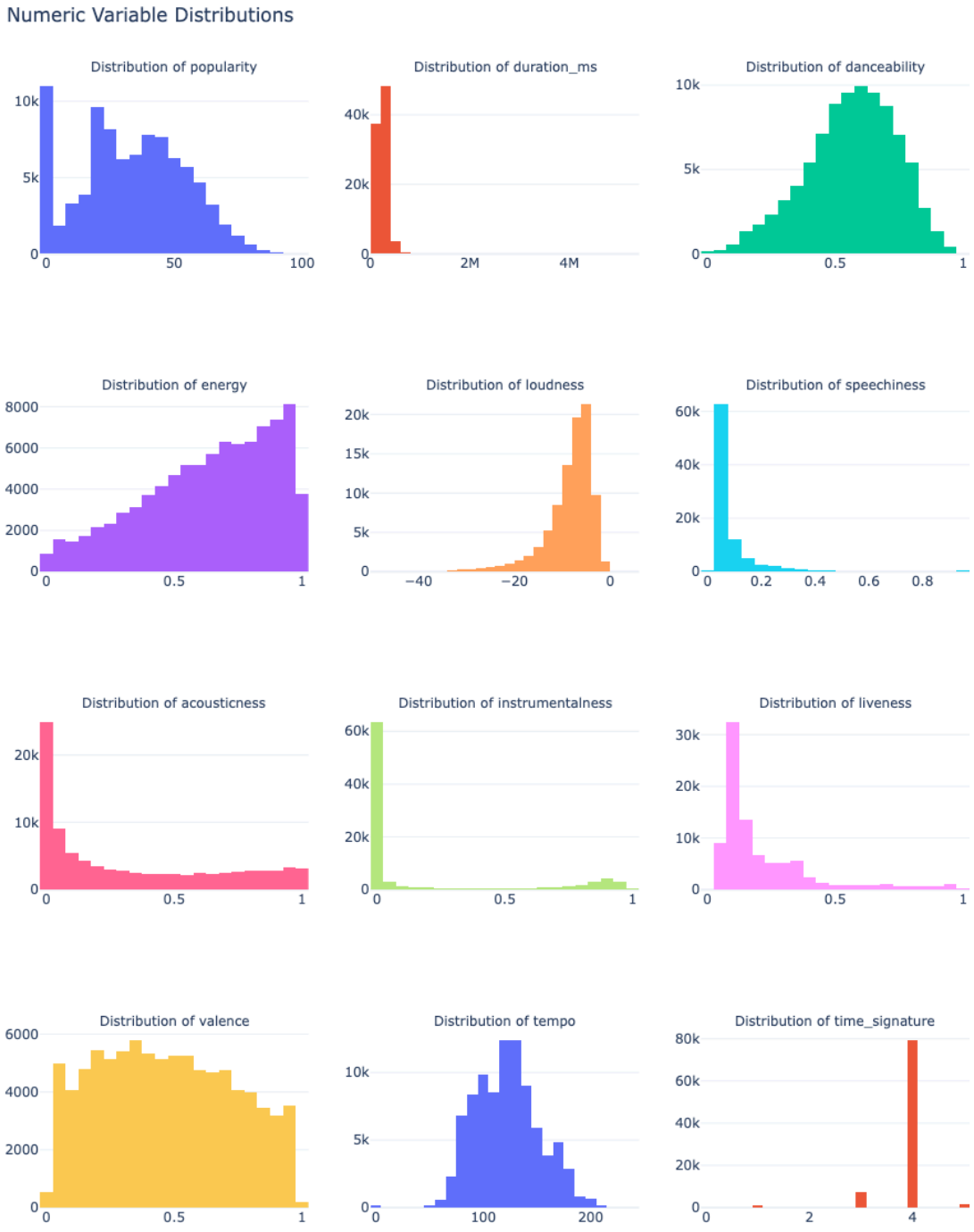
maximum absolute correlation values of approximately 0.05. This suggests that predicting track popularity may be more complex than simply analyzing audio characteristics.

The comprehensive correlation matrix visualization in Figure 2 provides additional insights into feature relationships beyond popularity. We observed strong negative correlations between acousticness and electronic music characteristics (energy, loudness), as well as notable positive correlations between related features such as valence/danceability. These relationships suggest underlying patterns in how musical elements combine in tracks.



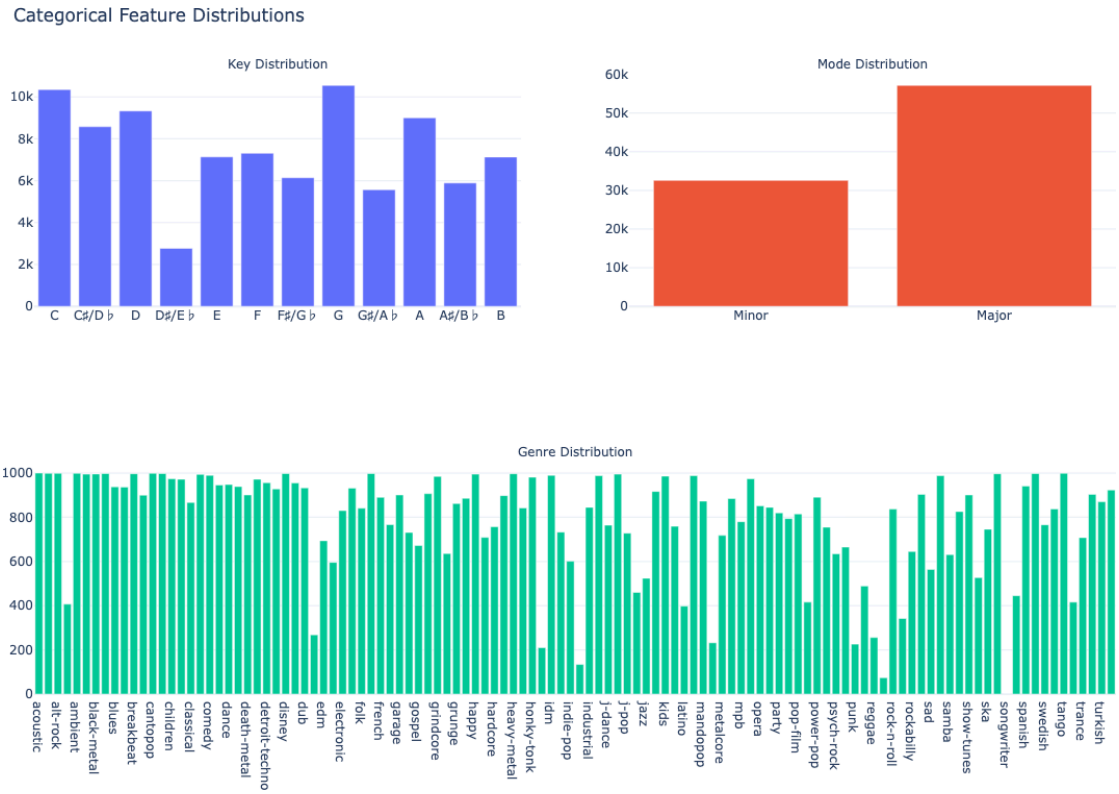
The analysis of numerical features, presented in Figure 3, revealed distinct distribution patterns across our variables. Several features exhibited clear normal distributions, including danceability, tempo, popularity, and loudness, suggesting natural clustering around central values for these characteristics. In contrast,

variables such as duration, speechiness, and instrumentalness showed heavily skewed distributions with significant outliers, indicating the presence of specialty tracks that deviate from typical patterns.



Our examination of categorical features in Figure 4 provided valuable context about the musical composition of our dataset. The mode distribution revealed a clear preference for major keys over minor

keys among tracks. The key distribution showed a relatively balanced representation across different musical keys, suggesting no strong bias toward particular tonalities. The genre distribution demonstrated remarkable diversity, with representation across numerous categories without overwhelming dominance by any single genre. This balance suggests our dataset provides a comprehensive view of the musical landscape.

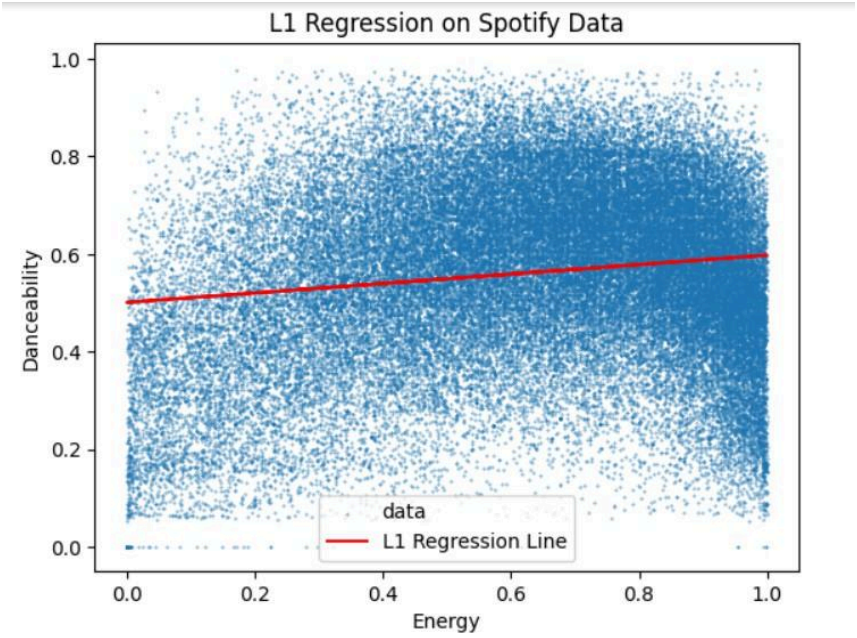


These analytical findings establish a strong foundation for subsequent modeling efforts, particularly highlighting the complexity of popularity prediction and the need to consider multiple feature interactions rather than individual characteristics in isolation. The balanced representation across musical categories and clear distribution patterns provide confidence in the dataset's ability to support robust analysis and modeling.

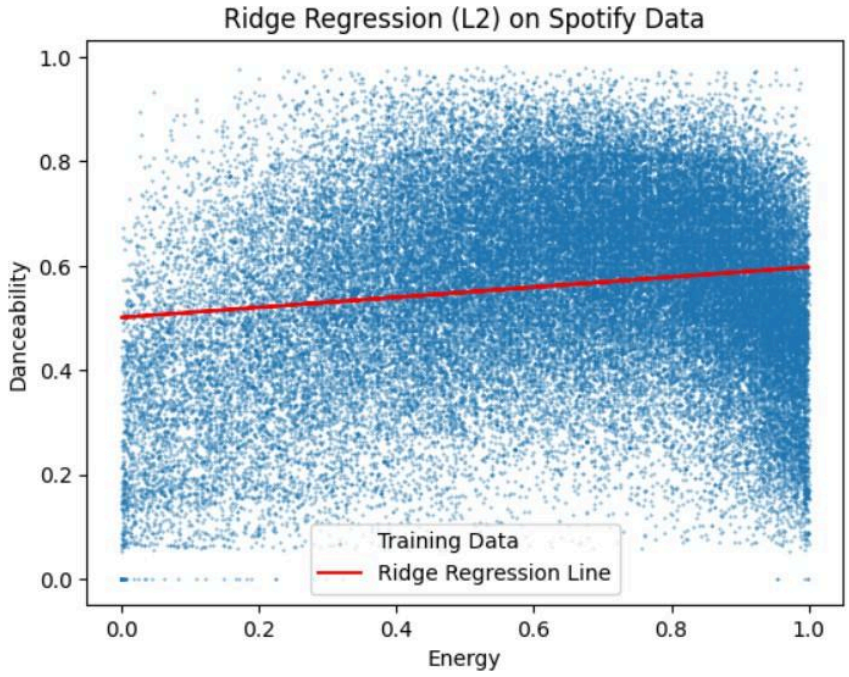
Regression Analysis

In the early stage of our project, we applied linear regression to model the relationship between energy and danceability. Our response and predictor variables were danceability and energy respectively. Our exploratory data analysis revealed that popularity has a weak correlation across different variables. So, we chose energy and danceability since they have a higher correlation than other variable pairs. Choosing popularity would most likely show a nonlinear relationship. We first trained an L1 Regression Model and an L2 Regression Model without using any regularization technique.

Our L1 model:



Our L2 model:



Both the L1 and L2 regression models with no regularization appear to be underfitting the data. The regression lines are almost flat. There seems to be a more complex, non-linear relationship between energy and danceability, and the regression lines fail to capture this. We didn't use this analysis to asssess feature importance.

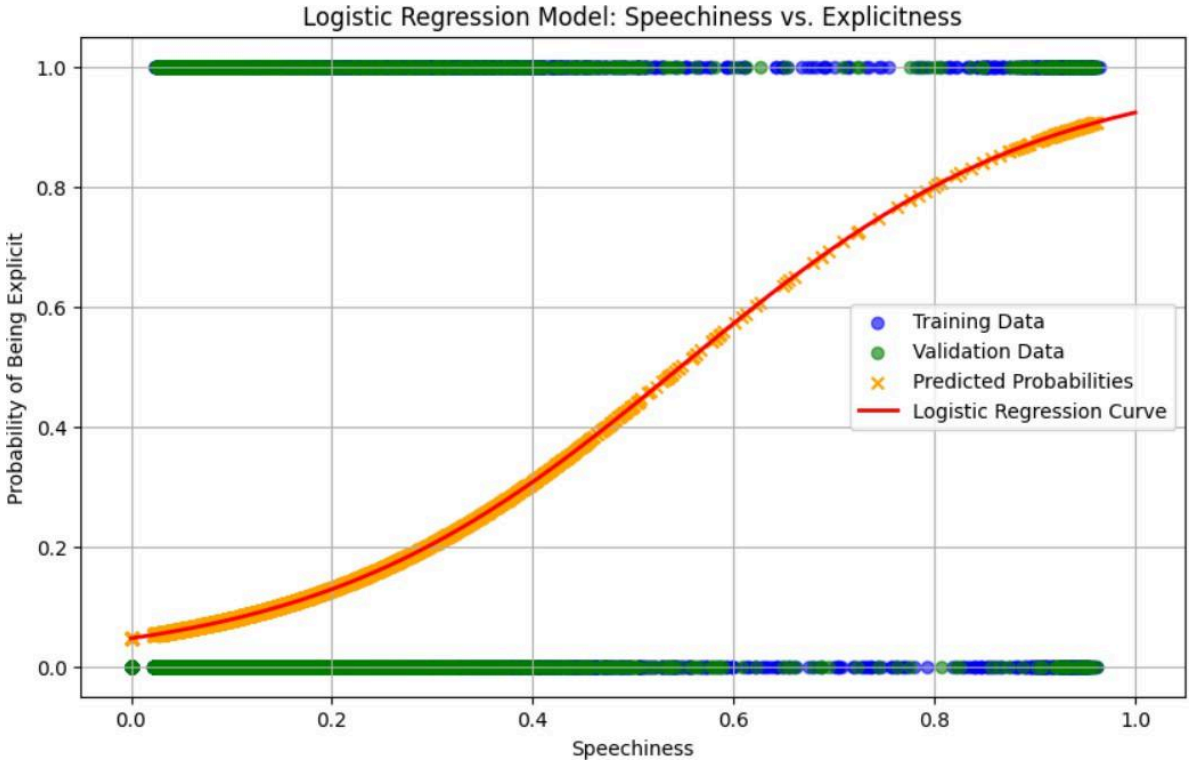
Since both L1 and L2 regression models appear to be underfitting the data and scatter plots suggest a non-linear relationship between energy and danceability, regularization is not necessary. Regularization is primarily a technique to prevent overfitting, not underfitting.

Logistic Regression Analysis

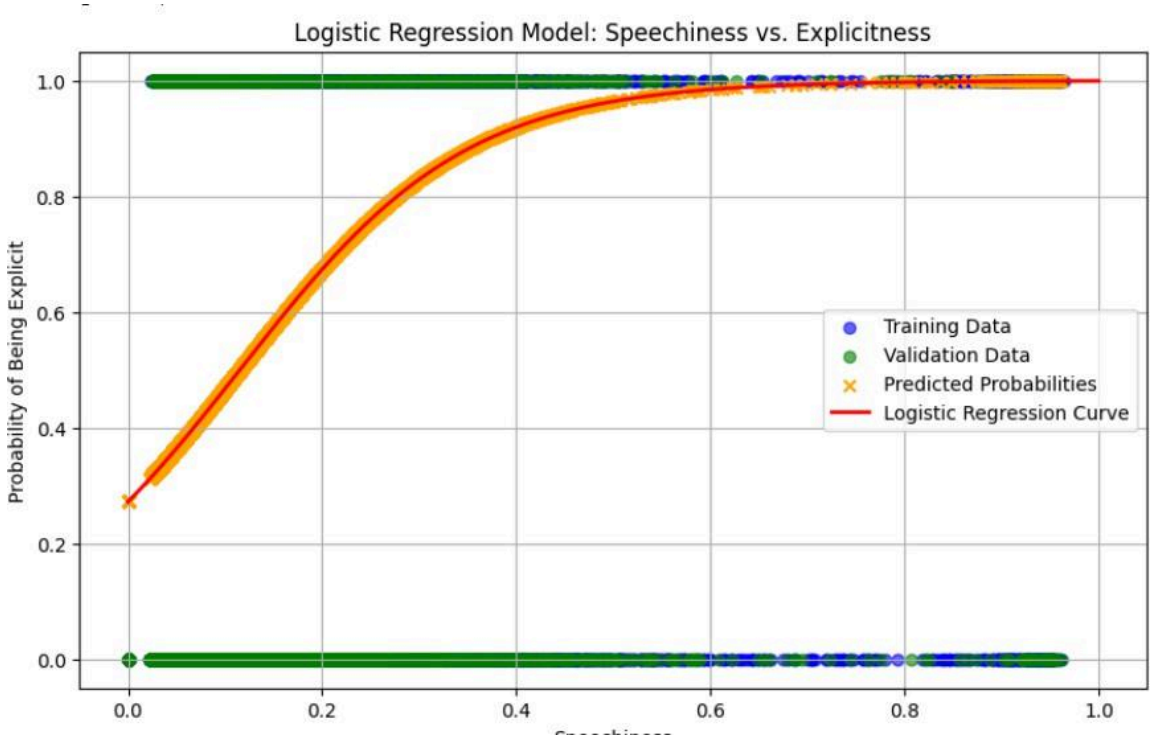
Logistic regression was used to predict whether a song is explicit based on the feature speechiness. The response variable, explicitness, was binary (1 for explicit, 0 for non-explicit).

The following steps were performed. We first split the dataset into training, validation, and test sets. Then, we used logistic regression with and without class weighting to account for class imbalance. We assessed performance through metrics such as accuracy, sensitivity, specificity, and the ROC curve. Our last step was optimizing the decision threshold to balance sensitivity and specificity.

Logistic regression model without class weighting:



Logistic regression model with class weighting:



We found out that the dataset contained significantly more non-explicit songs (91.4%) than explicit ones (8.6%). This imbalance affected model performance, making it biased toward predicting non-explicit songs.

The feature speechiness showed some ability to distinguish explicit from non-explicit songs, but it was not highly predictive on its own.

The initial sensitivity was low (~6%), indicating the model struggled to identify explicit songs when we didn't account for class imbalance. After using class_weight='balanced' and tuning the threshold, the sensitivity improved to ~68% with a reasonable specificity (~71%). In both models, we have an AUC score of 0.763 indicating that the models had fair predictive performance.

We did not use this analysis to assess feature importance.

Regularization was not applied in our logistic regression analysis. Since logistic regression was performed with only one predictor (speechiness), regularization wasn't critical. If more features were included, regularization could help prevent overfitting, improve generalization of the model and identify and reduce the influence of less important features.

KNN/Decision Trees/Random Forest

For the Spotify data, we wanted to classify whether a class was popular or not based on its features. We performed K-nearest neighbors and random forest as classification algorithms to determine whether songs could be accurately predicted to be popular or not. Songs were categorized as popular depending on a

popularity threshold value of 30. If a song had a popularity score of 30 or greater, then the song was considered popular (1). This number was preferred over the threshold of 50 because it produced a more even distribution between popular and non-popular songs.

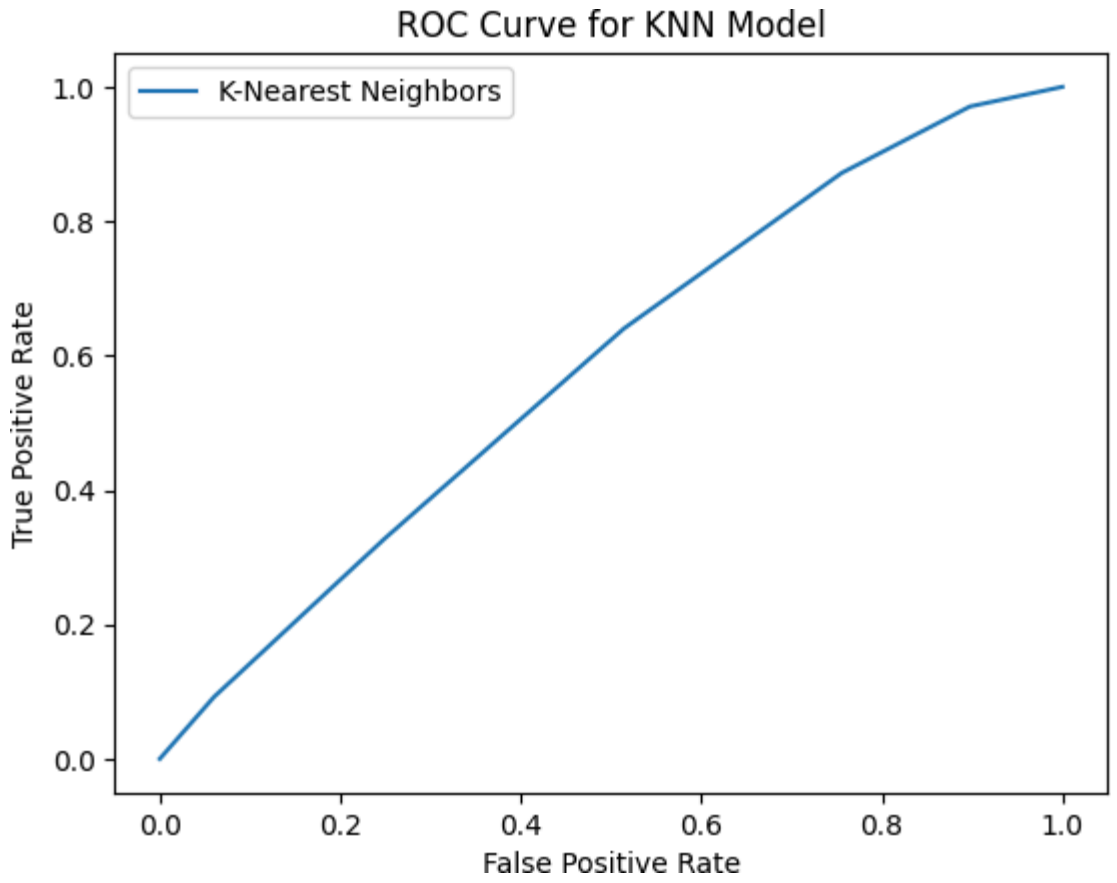
We wanted to determine whether popularity could be classified with KNN based on a point's danceability and speechiness. These features were chosen because based on domain knowledge, popular songs are often danced to. Also, danceability and speechiness had a low correlation when performing the correlation matrix so we wanted to study features that did not exhibit collinearity. The algorithm classifies a song as popular depending on its distance from danceability and speechiness.

To perform KNN, we tested whether popularity could be predicted based on its danceability and speechiness. We chose these metrics because they are relevant to popularity based on domain knowledge and because these two properties are not collinear themselves. We also chose not to include more than these two features to avoid the Curse of Dimensionality. However, the resulting test accuracy of 0.575 and average area under the curve (AUC) based on 5-fold cross-validation demonstrated that the model was not much better than random guessing.

Confusion Matrix:
[[3949 4180]
[3532 6288]]
Accuracy: 0.5703381804000223
Error: 0.42966181959997773
True Positive Rate: 0.640325865580448
True Negative Rate: 0.4857916102841678

AUC Score: 0.5850068172610746
5-Fold Cross-Validation AUC Scores: [0.57505427 0.57381161 0.57402936 0.57362206 0.57831236]
Average AUC Score: 0.5749659329743568

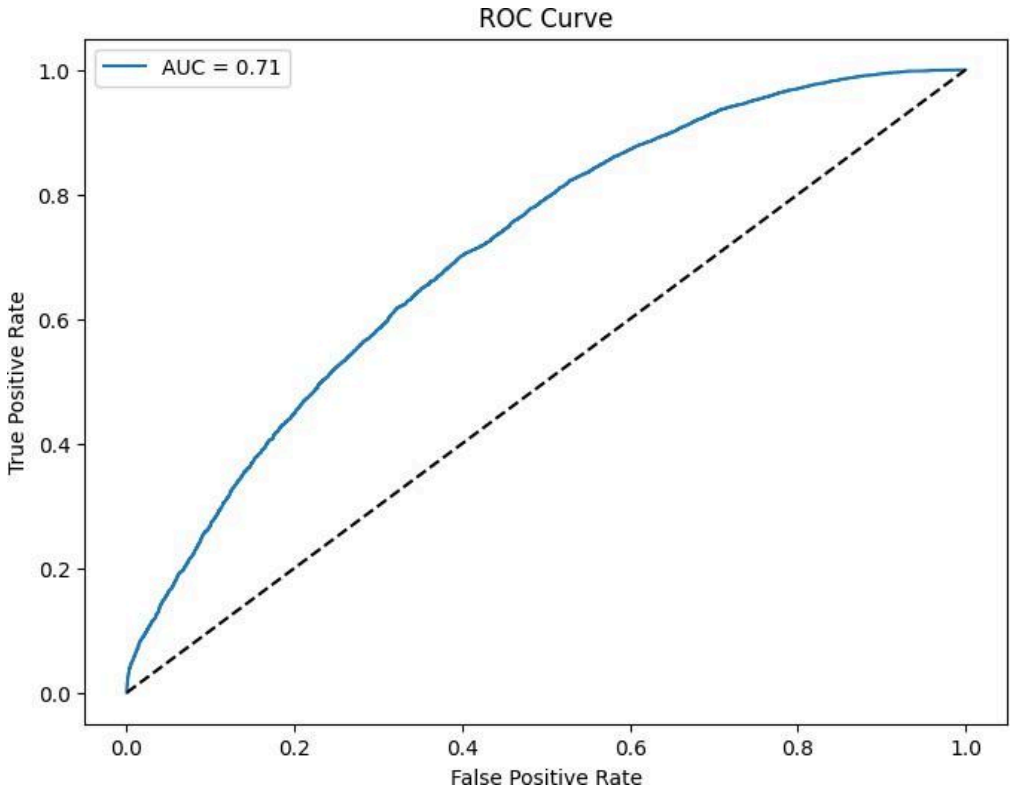
Figure 1: AUC curve of KNN



We then performed classification using the random forest algorithm. The random forest takes a majority vote from the classifications of the individual decision trees. The decision trees split the data based on the CART algorithm to classify songs as popular or not. The resulting test area under the curve for this model was 0.7085, demonstrating an improvement from KNN on classification of popularity.

Validation MSE: 0.3346891018497883
Test MSE: 0.3396478716291509
Validation Accuracy: 0.6653108981502117
Test Accuracy: 0.6603521283708491
Test AUC: 0.7085016344667884

Figure 2: AUC curve of random forest

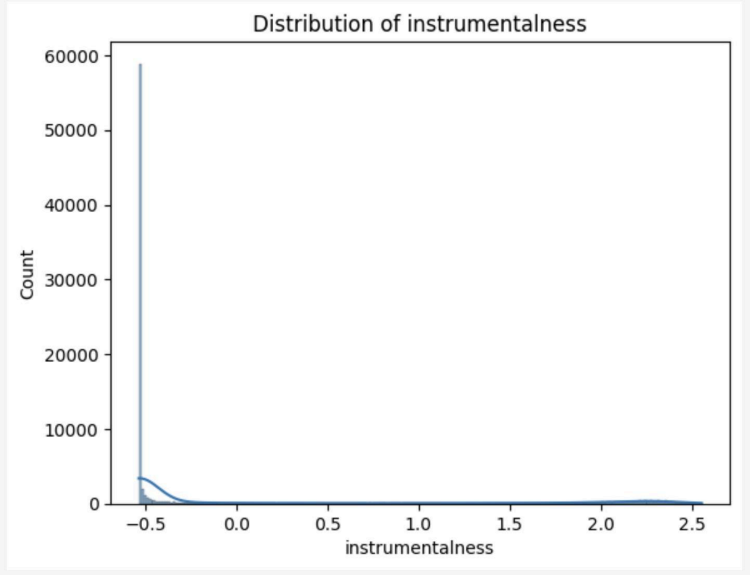


PCA & Clustering

Principal component analysis (PCA) is a dimensionality reduction technique which transforms correlated variables into uncorrelated variables to explain variance in the data. PCA and clustering did not directly address the problem of predicting a song's popularity based on its features. However, we were able to use PCA and clustering to see if there were certain songs that might have clustered together and if there was any relationship to its popularity. Based on our analysis, there were no clear clusters that depicted patterns in popularity.

For the Spotify dataset, we performed PCA on numerical variables including "speechiness", "danceability", "energy", "valence", "tempo", "loudness", and "acousticness." We excluded categorical variables such as "mode" and "time signature" because categorical variables do not have meaningful variance structure for PCA to analyze. We also excluded instrumentalness because it appeared to be very poorly distributed (Fig 1.). Prior to PCA, we standardized our variables because they were on different scales. To determine if a song is popular or not, we used a threshold of 50. If a song has a popularity score over 50, it is encoded as a 1 to indicate popularity. If it is below 50, then it is labeled with a 0, describing it as not popular.

Figure 1: Distribution of Instrumentalness



Once PCA was performed, we viewed the scree plot (Fig 2.) to determine which principal components to focus our attention on. We decided to choose the first two principal components because cumulatively, they describe 0.58 of the variance of the data. We then plotted PC2 against PC1 (Fig 3.).

Figure 2: Scree plot of PCA on standardized Spotify data

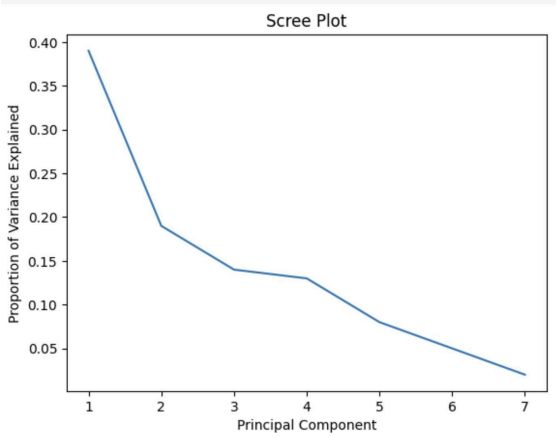
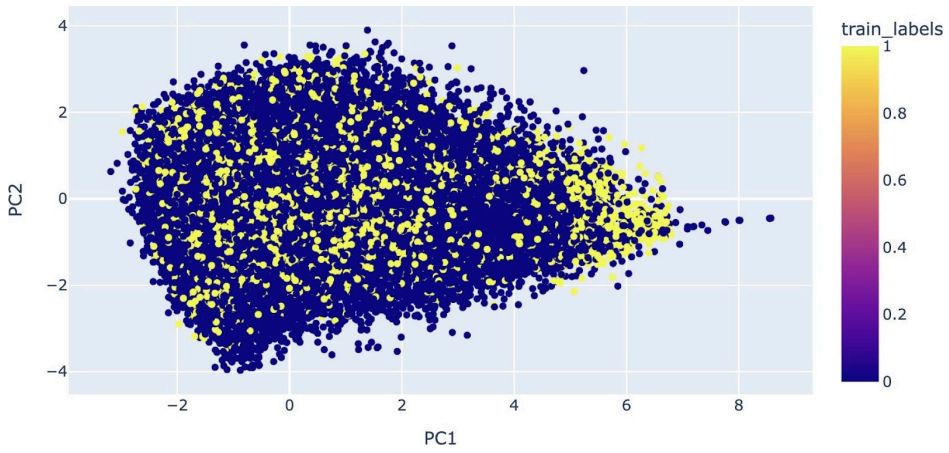


Figure 3: Plot of PC1 vs PC2 with popularity labels



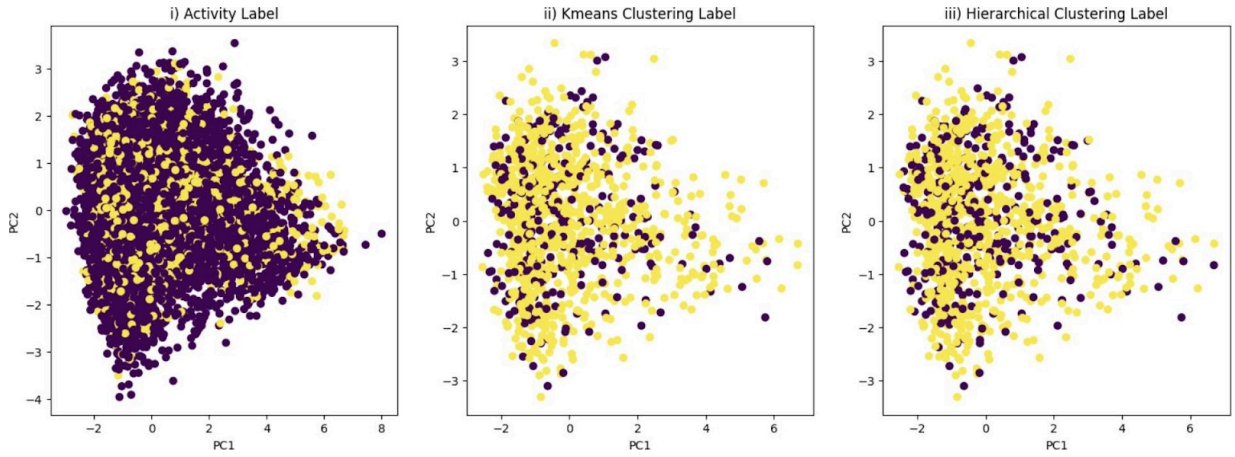
Because we standardized the data, we were able to gain meaningful insight into the variable loadings in both PC1 and PC2 to understand the data better. In PC1, energy contributed to the most variance with a loading of -0.539455 and in PC2, speechiness explained the most variance with a value of -0.958678. Both of these values were negative, suggesting that an increase in the aforementioned variables decreased the value of the PCs.

After performing PCA, we attempted both hierarchical and K-means clustering on the PCA transformed data. Due to the size of the data set, it was too computationally expensive to run clustering on the entire dataset.

To resolve this issue, we randomly sampled 10,000 data points from the original 89,740 point data set. This required an assumption that the random sample would be representative of the general trend in the data. Then, we performed hierarchical clustering and K-means clustering on this subsample to estimate how many clusters each method creates. When the number of clusters is not specified, hierarchical clustering creates two clusters while K-means clustering creates seven clusters.

Because we were looking at popularity, a binary variable, we chose to create two clusters. We performed both K-means clustering and hierarchical clustering on the subset of the data (n=10,000), generated silhouette scores for each method, and calculated the rand index to compare the two clustering methods. The silhouette-score for hierarchical clustering was 0.242 and the silhouette score for K-means was 0.294. The rand index was 0.784. This suggests that the clustering was not very descriptive of the patterns of the data across the clustering methods. However, the higher rand index indicates that both clustering methods were able to place the data points into similar clusters, suggesting that there may be another pattern we were not able to identify. Finally, we plotted the activity labels, K-means clustering labels, and hierarchical clustering to determine if there were any obvious clusters (Fig. 4). There were no obvious patterns in any of the clusters, showing that these methods were not particularly useful in addressing our overall goal.

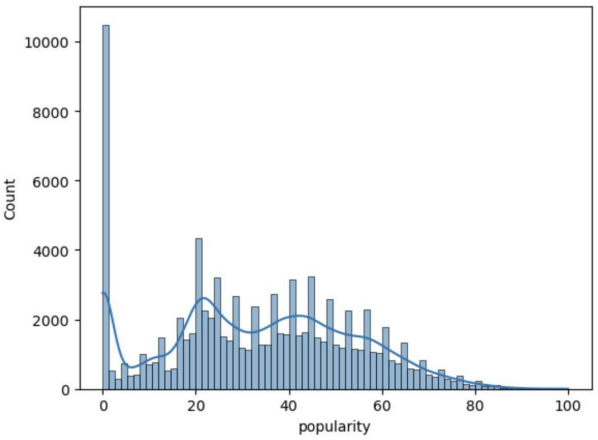
Figure 4: Subset of original data set plotted with their original labels (popular or not), kmeans labels, and hierarchical labels.

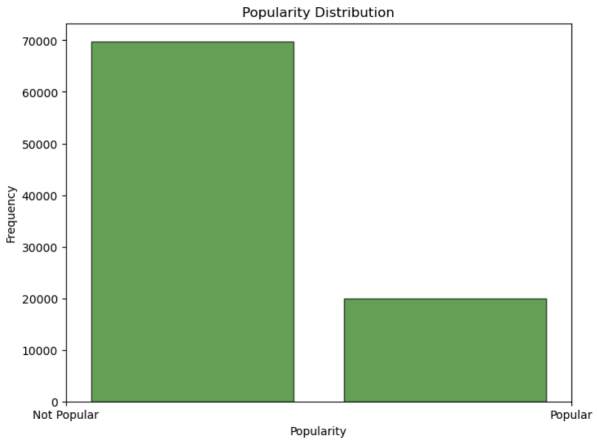


Neural Networks

In this section of our project, we trained our neural network. Training our neural network depends on backpropagation, which relies on the type of data/ layers, output, and activation function to generate the best model.

Our neural network model focused on converting popularity into a binary value, where 0 represents “not popular” and 1 represents “popular”. Since we had a range of popularity from 0-100, we set a threshold of 50 as we compared popularity distribution in comparison to its counts. These are represented through the following bar plots:





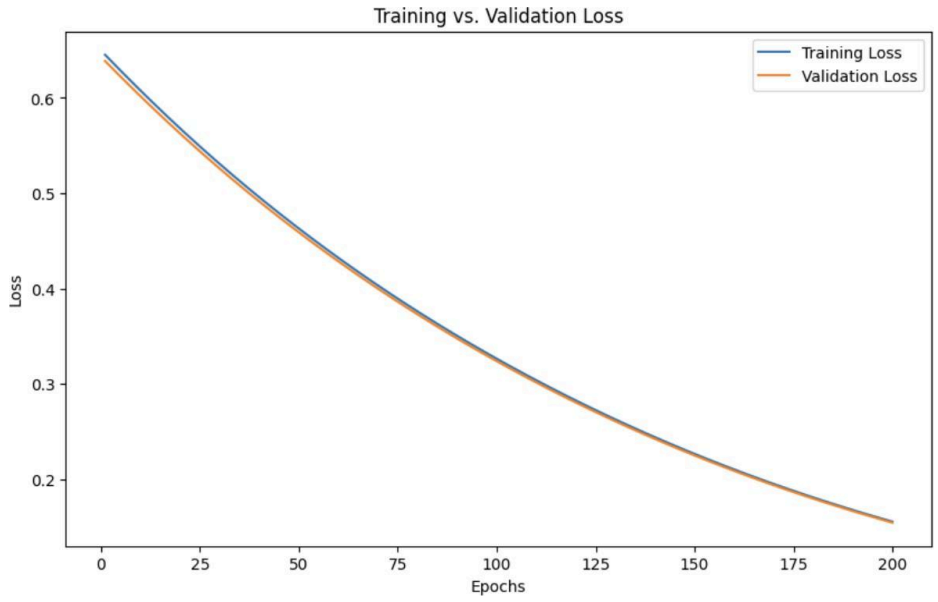
With this, we separated popularity into its own set of labels.

We split the data into its proper training and testing sets. Then, we standardized features to ensure that the training and testing were on the same scale since our feature variables were continuous prior to standardizing.

Using PyTorch, we implemented a neural network. Our model takes in continuous, standardized values as input where each feature is a dimension. Our hidden layer uses the ReLU activation function to introduce nonlinearity, allowing the network to learn patterns without it complicating the gradients. Since we want our output to be binary, using the sigmoid function was useful for the output layer in order to output values within the 0-1 range to be interpreted as probabilities. With this, we used a forward algorithm function with set hyperparameters to define our neural network. We initialized our model with our input & hidden layers and used an optimizer to set the learning rate. We also used Binary Cross Entropy Loss in order to track the loss function within our model.

Then, we trained the model through a ranging number of epochs. To train the model, we stored the losses in empty lists from both training and validation datasets at each epoch in order to address if the model is learning or overfitting. Storing a list helped us determine how the loss changes over time in order to further understand the model's learning behavior. After training the model, we set up an evaluation metric revealing the testing accuracy of our model.

For an improved model, we attempted to track the training loss in order to identify a decreased trend in the plot. This would determine that the model is learning to fit the data. The validation should follow a similar trajectory, decreasing alongside the training loss. We aimed to track if the model was overfitting, which would reveal if the validation loss plotline in our model would increase. If this were the case, we would have implemented regularization methods of neural networks that were discussed in lecture, such as dropout, norm penalties, and early stopping. Our results from training our model is revealed below:



Analyzing our plot, we can see that the model is not overfitting. The behavior of the training and validation loss is as expected. We can see the trends decreasing, ultimately reaching a convergence if we increase the number of epochs and allow the model to run for a longer period of time.

We set up a test accuracy function to determine the efficiency of our model. The model obtained a 78% test accuracy score, providing a proficient score to accept the model's results.

Since our model was not overfitting, it was not necessary to implement regularizations that could improve the trajectories of the model. However, to demonstrate our knowledge, we attempted to add norm penalties in the form of weight decays to the optimizer in order to see if results varied. Not much change occurred with this addition.

Hyperparameters

- The hyperparameters in our model are mainly influenced by learning rate, input layers, output layers, and the number of epochs.
- With a smaller learning rate, the trajectories of our training and validation loss plot were skewed. When we aimed to decrease the learning rate to 0.001, we found that the model's validation loss followed a linear trajectory, revealing that the learning rate was too low for the model. In our final neural network model, we set our learning rate to 0.01.
- We set our input layers to be the dimensions of the features, and set our hidden layers to be an arbitrary value, such as 16.
- We used the entire dataset to represent our batch size, relying on batch gradient descent in comparison to mini-batch or stochastic gradient descent.
- We also set our number of epochs to 200, as we could visually define convergence with this value.