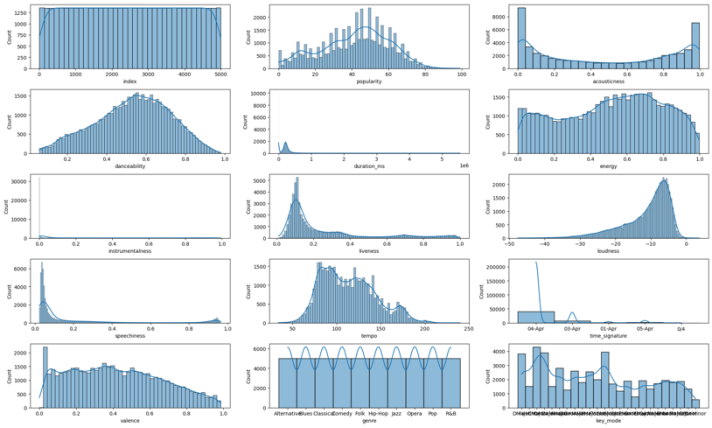
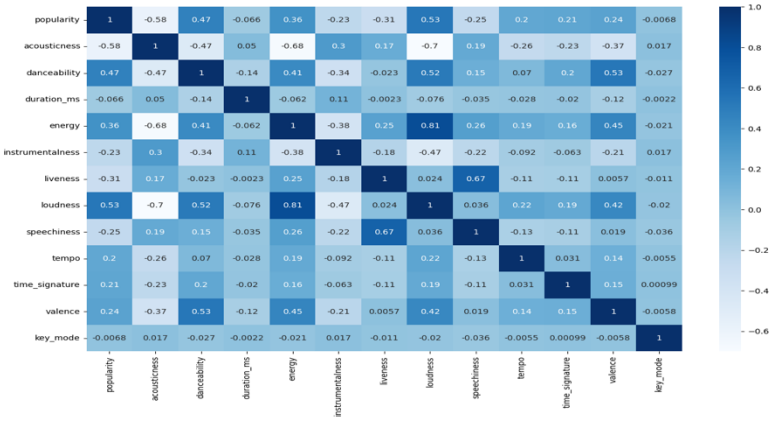
COMP309 – Assignment 3
greenthom – 300536064

Core

One of the first things noticed in the dataset was that key features such as tempo and duration_ms had missing values. These missing values would have negatively affected the model's ability to capture patterns within the data, leading to incomplete or distorted relationships between features and the target variable (such as popularity). For example, tempo had 7501 missing values marked with question marks, and duration_ms had 10114 missing values set to -1. Addressing these missing values through imputation or removal is essential to avoid introducing bias and to ensure that the model generalizes effectively to new data. In contrast, time_signature had a very low number of missing values (only 3 instances), so it was deemed unnecessary to modify it. However, unaddressed missing values in features like tempo and duration could lead to significant model inaccuracies if not handled properly.





Highly Correlated Features: Energy, Loudness, and Danceability: The correlation heatmap highlighted that energy, loudness, and danceability are highly correlated with one another. For example, loudness and energy showed a correlation of 0.81, the highest among any feature pairs in the dataset. This makes intuitive sense, as louder songs tend to convey more energy, and highly energetic songs tend to be more danceable. The strong correlations between these features mean that as one increases, the others are likely to follow. While this interaction is helpful for understanding the relationships between musical characteristics, it also introduces the risk of multicollinearity. High multicollinearity may increase variance in model coefficients and reduce model interpretability. To address this, it may be necessary to remove, combine, or apply Principal Component Analysis (PCA) to these correlated features to reduce redundancy and multicollinearity, potentially improving the model's performance. For instance, since loudness and energy are almost interchangeable in their effect on the data, it may be worth considering removing one of them.

Weak or Near-Zero Correlations with Duration, Mode, and Key: The features duration, mode, and key were found to have near-zero correlations with other features in the dataset, suggesting that changes in these values have little to no effect on other attributes. For instance, duration_ms had minimal correlation with features like danceability and energy, meaning that the length of a song doesn't significantly affect its energy or how danceable it is. Similarly, key and mode had weak correlations with most features, showing that the scale (major or minor) in which a song is composed has limited influence on musical attributes like tempo or loudness. The low predictive value of these features could potentially introduce noise into the model and increase the risk of overfitting. As such, it may be worth removing these features from the final model to improve accuracy and reduce overfitting risks.

Negative Correlation of Acousticness with Energy, Loudness, and Danceability: Acousticness showed strong negative correlations with energy (-0.68), loudness (-0.70), and danceability (-0.47). This aligns with the idea that highly acoustic songs, which typically feature natural and less electronically-enhanced soundscapes, tend to be quieter, less energetic, and less rhythmically driven. These negative correlations provide a valuable distinction for the model, as they suggest that acoustic tracks are inherently different from energetic, loud, and danceable songs. This pattern could help the model differentiate between genres like Classical, Folk, and Jazz, which often feature acoustic instrumentation, and genres like Pop and Hip-Hop, which tend to be louder and more energetic.

Relevance of Time Signature to Energy and Tempo: Although time_signature had very weak correlations with other features, one interesting observation was its slight correlation with energy (0.16). Time signature refers to how many beats occur per bar in a song. A 4/4-time signature, for instance, does not typically correlate with energy unless the note values (e.g., crotchets or quavers) are considered. While the correlation between time_signature and energy was small, it is important to note that certain time signatures might allow for more rapid or slower tempos depending on how fast notes are played within a bar. This slight correlation with energy could be explored further in models that focus on how rhythm and structure impact a song's overall energy, though it is likely unnecessary to prioritize this feature.

Low Contribution of Mode and Key: Lastly, mode (whether a song is in a major or minor scale) and key were found to have little relevance in the dataset due to their weak correlations with other features. As observed from the heatmap, changes in key or mode do not significantly affect other attributes such as tempo, valence, or popularity. This suggests that these features do not provide significant predictive value and may even introduce noise or cause overfitting in the model. Therefore, removing or deprioritizing key and mode in the model could improve model performance and streamline feature selection.

In conclusion, these seven patterns highlight key areas that could impact the accuracy and generalizability of the model. Addressing missing values in key features like tempo and duration_ms, handling multicollinearity between correlated features like energy and loudness, and removing features with low predictive value such as mode and key are all steps that can optimize the model and improve its predictions. Furthermore, the negative correlation between acousticness and other features like loudness and danceability will aid the model in distinguishing between genres based on their acoustic characteristics.

There was more EDA analysis on the dataset in my core.ipynb but these were the main features I observed when performing basic analysis on the dataset.

Completion

Discuss the initial design of your system, i.e. before you have submitted any predictions to the Kaggle competition. Justify each decision you made in its design, e.g. reference insight you gained in the Core part.

- Initial design of the system was centered around key decisions made from doing EDA. Missing values were identified in key features like tempo and duration_ms, with tempo having 7501 missing values (represented as '?') and duration_ms having 10114 missing values (represented as -1). Missing values in these features would have negatively affected the model's ability to capture meaningful patterns in the data, especially since these features are critical for understanding a song's rhythm and length. Therefore, a SimpleImputer was employed to fill missing values with the

mean of each feature. This imputation strategy ensures that the model retains complete data, preventing potential biases that could arise from missing values. Time signature, with only three missing values, was left unchanged, as its low count of missing values would not substantially affect the model.

- The dataset contains several categorical variables, including key, mode, and time_signature, which are not directly usable by machine learning algorithms. To address this, Label Encoding was applied to these categorical features to convert them into numerical values. Label encoding was chosen over one-hot encoding to reduce the dimensionality of the dataset and maintain simplicity, especially since features like key and mode have a limited number of distinct values (e.g., major/minor for mode). Label encoding also allows the model to treat these features ordinally if required, ensuring that the structure of the data is preserved.
- Features such as artist_name, track_name, instance_id, and track_id were removed from the dataset, as they are irrelevant for predicting musical attributes like genre. These features do not contribute to the relationships between the numerical data and the target variable (genre). Removing them reduces noise and ensures that the model focuses on the features that matter most, such as tempo, danceability, and energy.
- Since the dataset contains a mix of features with different ranges (e.g., loudness in decibels, duration_ms in milliseconds), it was necessary to scale the data. A MinMaxScaler was applied to ensure that all features are within a standard range, improving the performance of distance-based algorithms like KNeighborsClassifier (KNN). Scaling helps prevent features with large numerical ranges from disproportionately influencing the model, ensuring that all features are treated equally.
- Dimensionality Reduction Using PCA and ICA: To explore the possibility of reducing dimensionality while retaining the most important information, Principal Component Analysis (PCA) and Fast Independent Component Analysis (ICA) were applied. Both techniques were used to reduce the feature space to 14 components. PCA helps reduce multicollinearity between highly correlated features (such as energy, loudness, and danceability) and captures the variance of the dataset in fewer dimensions. ICA, on the other hand, separates the components in the dataset by maximizing statistical independence. The system was designed to compare the results of using both PCA and ICA, as they offer different approaches to dimensionality reduction. This allows the system to determine whether reducing the number of features improves model accuracy or generalization.
- Random Forest (RF), KNeighborsClassifier (KNN), and Gradient Boosting Classifier (GNB) algorithms were chosen to provide a balance between ensemble methods (RF, GNB) and a distance-based classifier (KNN), allowing for comparison of performance across different techniques. Random Forest is a strong baseline model known for its robustness in handling large datasets and feature importance. KNN was included to assess how well the scaled and transformed features would perform in a simple, non-parametric classification approach. Finally, Gradient Boosting was used as it provides iterative training and could potentially lead to better accuracy by focusing on misclassified examples.

Discuss the design of one or more of your intermediary systems. Justify the changes you made to the previous design based on its performance on the leaderboard, and from any other additional investigation you performed.

- In the previous design, missing values for numerical features such as tempo and duration were imputed using the mean however after doing research into imputation strategies for missing values for numerical data it was updated to use the median. This change was made after further

investigation showed that many numerical features, such as loudness and tempo had skewed distributions. By using the median instead of the mean, it handled the skewness in the data, by reducing the potential outliers to distort imputation and improve model accuracy. For this model I also did categorical and numerical encoding instead of using label encoding to convert categorical data to numerical data, encoded and imputed categorical and numerical data separately. The categorical features imputation strategy was most_frequent, replacing missing values with the most common value in the column. This decision was driven by the observation that these features had a small number of missing values(time_signature -> 0/4), and using the most frequent value was a simple and effective way to handle this.

- In the previous design, I used MinMaxScaler to scale/encode the numerical values in the datasets. Since I converted all the features to numerical features this was the only step I took for scaling/encoding. In this design, categorical features such as time_signature, key and mode were transformed using OneHotEncoder instead of LabelEncoder. OneHotEncoding was chosen to convert categorical variables which was good for the features since they were nominal features. This transformation prevents it from being described as ordinal which would reduce bias and allow to expand the categorical features into multiple binary features which helped the models being used to treat each categorical feature independently. Used StandardScaler instead of MinMaxScaler for the numerical features, ensuring that the features were on a similar scaler.
- In the previous design, I was using KNN, RF, and GNB which was retained for this model to assess performance on the new preprocessing techniques being used. Also added a Multilayer Perceptron Classifier (MLPC) and removed KNN. The MLPC was chosen due to the models ability to capture complex non-linear relationships in the data. During testing, the MLPC showed better accuracy than both Random Forest and Gradient Boosting, with an accuracy score of 0.6515 compared to 0.06276 for Random Forest and 0.6269 for Gradient Boosting.
- In the previous design, I also had mode and key as two separate values for encoding, which both got converted to numerical features. For this model, similar to my EDA, I concatenated the two models together into one feature. Based off EDA, I saw that both features (when separate, they are joined together in EDA) gave similar correlation values. In music typically these two features are described together as they define the scale the 'song' is played in. They are then treated as categorical features and undergo the same imputation and encoding as other categorical values like time signature.
- In addition to testing individual models, RF, GMB, MLPC, a StackingClassifier was introduced as an ensemble method which combined the predictions of the three models used after they were trained. The idea was to leverage the strengths of each algorithm to improve the overall performance. Although the stacking classifier initially yielded an accuracy score of 0.6288, slightly lower than MLPC, it performed more consistently across the various testing subsets. Further insight can go into how stacking different models together can improve the accuracy of this classifier. Did not do enough research into what models would help improve this but this could be further looked into.

Use your judgement to choose the best system you have developed – this may not necessarily be the most accurate system on the leaderboard. Make sure you select this submission as your final one on the competition page before the deadline. Explain why you chose this system and note any particularly novel/interesting parts of it. You should submit screen captures and/or the source and executable code required to run your chosen submission so that the tutors can verify its authenticity.

- Further improved my model from my intermediary, still uses same numerical, categorical encoding and imputing. Still running same models also. Changes were made more from the features and relationships observed in EDA.
- Several variations were tested to assess the impact of removing specific features such as duration_ms, key, and mode. These were identified as features that had very low correlation values observed in the initial EDA and typically didn't impact the model. Due to the size of the test data, making sure we aren't overfitting the models is vital which was one of the reasons for removing features, as well as having the model focus on more relative relationships of different features when determining the data instead of having features that don't contribute to those relationships at all. When duration_ms was removed, there was only a minor change in the model's accuracy, for example, RF accuracy changed from 0.6234 to 0.6233. However, removing key and mode resulted in a more noticeable improvement, for example, random forest accuracy increased to 0.6271, reinforcing the decision to drop these low-impact features. The experiments with feature removal helped confirm that some features, like key and mode, were not contributing significantly to the model's predictive power. Additionally, the relatively consistent accuracy of models with and without duration_ms suggests that duration_ms may not be as critical a feature as initially thought. MLPC had the best accuracy from these variations.
- For the final design, key and mode features were outright removed. Based off the insights from the correlation heatmap and observation through testing, it became evident that key and mode had near-zero correlations with other features and did not contribute significantly to the prediction accuracy. As they added little predictive value, they were removed.
- Based on the removal of features we observed MLPC classifier was scoring the best (0.6525 accuracy). This was the final model that was decided on to perform on the testing data.
- Added back in artist_name to the categorical features list to be encoded and added to the model. The artist_name feature was OneHotEncoded which would generate multiple binary features for each artist name. This improved the models performance with a training accuracy from 0.6525 to 0.7857 (0.1332 increase). However, the risk of doing this is due to OneHotEncoding this the amount of binary features this created could cause the model to overfit, resulting in the model not identifying important patterns when performing on unseen data. This however did improve my accuracy on running my test set and Kaggle submission. Any further attempt to improve the accuracy were redundant, so this is my final model.

Challenge

How easy is it to interpret your chosen machine learning model, i.e. how easy to comprehend why certain predictions have been made? If your model is difficult to interpret, do you see any problems with this? (e.g. whether users will trust your model? Whether it is difficult for the deployment of your solution or to use the model? And so forth.) How would it compare to a simpler model, e.g. a simple K-Nearest neighbor? If your model is easy to interpret, what are its limitations? (e.g. whether it can catch the underlying relationships in the data? Whether it can provide accurate predictions?) How would it compare to a more complex model, such as an ensemble method (e.g. random forest)?

- Multilayer Perceptron Classifier (MLPC) is a neural network-based model that is known for capturing complex, non-linear relationships in data. However, its interpretability is significantly lower compared to simpler models like K-Nearest Neighbors (KNN) or decision tree-based models such as Random forest. This lack of interpretability stems from the "black box" nature of neural networks, where the internal workings (such as weights, biases) are difficult to understand or

explain to users who are not familiar with deep learning techniques. In an MLPC, predictions are made based on how data passes through multiple layers of neurons. Each neuron applies a weight to the input it receives, transforms it with an activation function, and passes the result to the next layer. While this structure allows the model to recognize complex patterns, it also makes it difficult to track the specific reasons behind individual predictions. For example, if the model predicts a certain genre of music, it is challenging to determine exactly which features contributed to that prediction and how they interacted.

- Main issue with using a MLPC is the difficulty in explaining why certain predictions are made. Users are less likely to trust a model if they cannot understand or visualize how decisions are made. E.g. if a user questions why a particular song is recommended as a certain genre, the MLPC model would not be able to provide an easily interpretable explanation. If the MLPC performs poorly on certain predictions, it can be difficult to diagnose why. Unlike simpler models (e.g. decision trees), where you can examine decision paths and feature importance, neural networks do not provide an easy way to identify the cause of misclassification. This makes it harder to improve or fine-tune the model. The complex non-linear relationships captured by the MLPC are difficult to interpret in terms of the contributions of individual features.
- A simpler model like K-Nearest Neighbor (KNN) offers much higher interpretability compared to MLPC. KNN works by looking at the closest training data points to a new sample and making predictions based on the majority class of its neighbors. This process is transparent and easy to understand because it is based purely on similarity. KNN advantages are interpretability, where for each prediction, KNN can show the nearest neighbors and how similar they are to the sample being classified, which is intuitive and easy to explain. Another advantage is that it has no hidden layers or complex weight adjustments that makes it easier to visualize and understand how predictions/decisions are made. However, KNN does have its disadvantages where it is limited to simple relationships as it struggles with high-dimensional data and complex relationships between features. E.g. in music datasets, where non-linear relationships between tempo, danceability and energy exist, KNN might underperform.
- While MLPC can capture complex patterns and achieve higher accuracy it has limitations. As of above, MLPC has a lack of interpretability, which can affect user trust and transparency. Neural networks are also prone to overfitting, especially when trained on small datasets or when the model is too complex. Without careful tuning of hyperparameters such as the number of layers and neurons, the MLPC may memorize training data and perform poorly on new, unseen data. MLPC models are computationally more expensive to train and predict compared to simpler models. This can be a disadvantage in real-time applications or systems with limited computational resources.
- Compared to models like Random Forest, the MLPC offers higher accuracy in some cases but falls short in interpretability. Random Forest can provide feature importance scores, giving insight into which features are most influential in predictions. MLPC lacks this straightforward feature importance mechanism, making it harder to understand how it weighs different features like popularity, loudness, or danceability. There are some pros of random forest where you can see which features matter the most and while still complex, decision trees in random forest provide a clearer decision-making process. There are some pros with MLPC also where it is better at modeling non-linear relationships as well as in practice MLPC achieved a higher accuracy in the created model.

References

- ChatGPT (help with increasing performance of model, particularly fixing errors around preprocessing)
- https://ecs.wgtn.ac.nz/foswiki/pub/Courses/COMP309_2024T2/LectureSchedule/COMP309-W6-Tutorial-2024.pdf (followed EDA, preprocessing and machine learning information carrying out assignment)
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (understand fitting and parameters for lr)
- https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (understand fitting and parameters for mlpc)
- https://www.atlassian.com/data/charts/heatmap-complete-guide (how to read correlations in a heatmap chart)
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html (understand how LabelEncoding performs on categorical data)
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html (understand how OneHotEncoding performs on categorical data)