
# Data Science

## Classify Song Genres from Audio Feature

**01286663 DATA SCIENCE AND DATA ANALYTICS**

**Software Engineering Program**

By

65011381 Napatr Sapprasert
65011462 Phupa Denphatcharangkul
65011400 Natthawut Lin
65011558 Suvijuk Samitimata

# Abstract

**Summary**: This data science project explores genre classification of songs using audio features from the GTZAN dataset. Emphasis is placed on data collection, preprocessing, feature manipulation, and insights generation to improve classification accuracy. The outcome aims to enhance user experience by improving music recommendations.

# Table of Contents

# Introduction

**Problem Statement:** Accurate song classification by genre is critical in digital music services for delivering personalized recommendations. This project addresses the challenge of using audio features to distinguish genres.

**Objectives:** Leverage data science techniques to collect, preprocess, and analyze audio data for effective genre classification.

# Literature Review

## Previous Work:

Davis, S. and Mermelstein, P. (1980) Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech and Signal Processing, 28, 357-366.

Volkmann, J. & Stevens, S. & Newman, E.. (2005). A Scale for the Measurement of the Psychological Magnitude Pitch. The Journal of the Acoustical Society of America. 8. 208-208. 10.1121/1.1901999.

Gursimran Kour, Neha Mehan . Music Genre Classification using MFCC, SVM and BPNN. International Journal of Computer Applications. 112, 6 ( February 2015), 12-14. DOI=10.5120/19669-1119

## Research Gaps:

the research aims to help accurately classify songs based on genre to help with easy mass identification of collections of songs. This could open up use cases such as improved recommendations, curated playlists, Targeted content management, moderations, and marketing, etc.

# Methodology

**Data Collection / Sources:** This project uses the GTZAN dataset, widely used in genre classification tasks, containing various genre-labeled audio tracks.

**Data Preprocessing:** Audio preprocessing involves:

- **Feature Extraction**: Extracting Mel Frequency Cepstral Coefficients (MFCCs), chroma features, and other relevant audio attributes.
    - **Basic Track Information :**
        - **length:** Duration or length of the audio track.
    - **Spectral Features :**
        - **chroma_stft_mean / chroma_stft_var:** Chroma Short-Time Fourier Transform, showing the energy distribution across the 12 different pitch classes (like C, C#, etc.). Chroma features are helpful for capturing harmonic content.
        - **spectral_centroid_mean** / **spectral_centroid_var**: Represents the "center of mass" of the spectrum. Higher values typically indicate "brighter" sounds.
        - **spectral_bandwidth_mean** / **spectral_bandwidth_var**: Measures the width of the frequencies, indicating how spread out the spectrum is around the centroid. Higher values often mean more complex sounds.
        - **rolloff_mean** / **rolloff_var**: Spectral roll-off represents the frequency below which a specified percentage of the total spectral energy lies. It helps to differentiate high-frequency sounds.
        - **zero_crossing_rate_mean** / **zero_crossing_rate_var**: Counts how often the signal crosses zero, indicating noisiness or percussive elements in a track.
    - **Rhythmic Features :**
        - **tempo:** The estimated tempo or beats per minute (BPM) of the track.
    - **Harmonic Features :**
        - **harmony_mean / harmony_var:** These features represent harmonic content, helping to capture tonal and harmonic aspects.

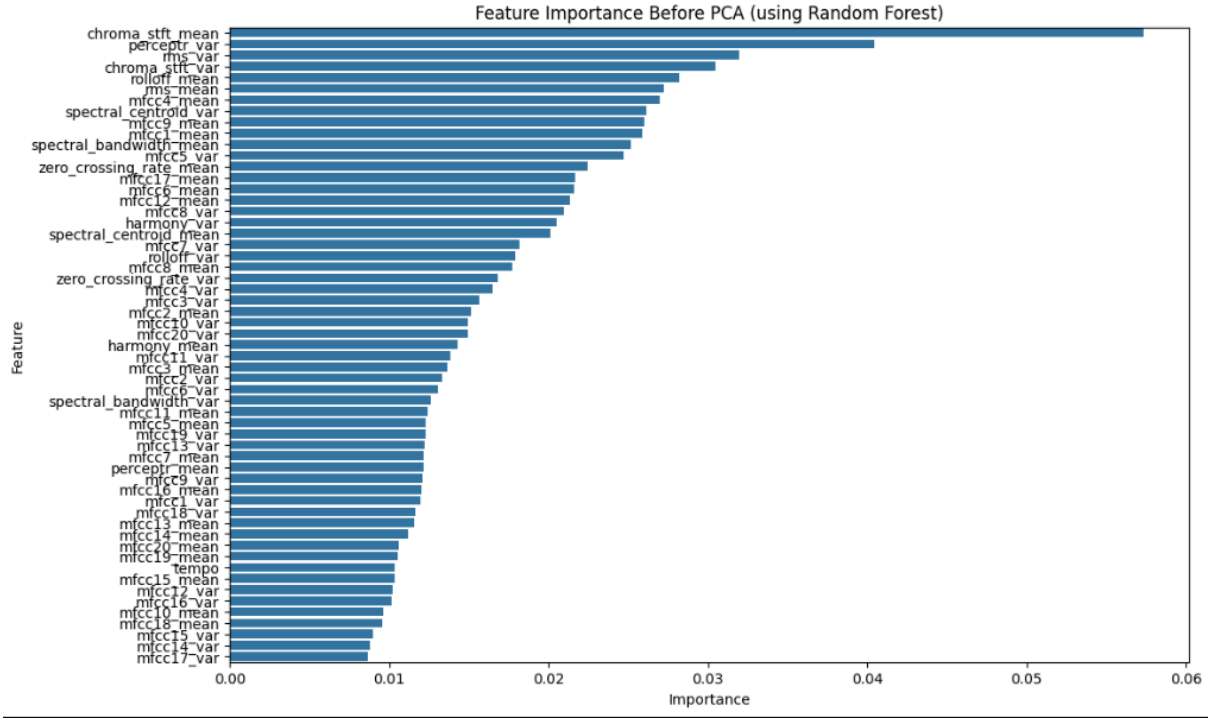
        - **perceptr_mean** / **perceptr_var**: Likely related to perceptual harmonicity, which captures harmony based on perceived sound quality.
    - **Mel-Frequency Cepstral Coefficients (MFCC) :**
        - **mfcc1_mean to mfcc20_mean and mfcc1_var to mfcc20_var:** Each MFCC coefficient captures different aspects of the sound's timbral quality, where lower MFCCs represent the general spectral shape, and higher coefficients capture more detailed textures.
    - **Label :**
        - **label:** The target variable, representing the genre category of each track.
- **Remove duplicate:** Use .drop_duplicates() from pandas DataFrame
- **Remove anomalies:** Use methods like the IQR (Interquartile Range) to identify and remove outliers.
- **Normalization**: Standardizing audio features to ensure uniformity across tracks.

**Data Manipulations:** Feature engineering and dimensionality reduction methods (like PCA) to identify and emphasize genre-distinguishing patterns.

**Algorithms or Methods Used for Insight Findings:**

- **Feature importance from Random forest**

We use random forest classifier to identify which features contribute the most to classification accuracy. This will help us understand which features are most influential in predicting genres.




The results indicate that the following five features have the highest influence.

| **Feature** | **Importance** |
|---|---|
| chroma_stft_mean | 0.057327 |
| perceptr_var | 0.040436 |
| rms_var | 0.031949 |
| chroma_stft_var | 0.030508 |
| rolloff_mean | 0.028247 |

To understand how features relate to genres, it would be easier to break down the meanings behind these features:

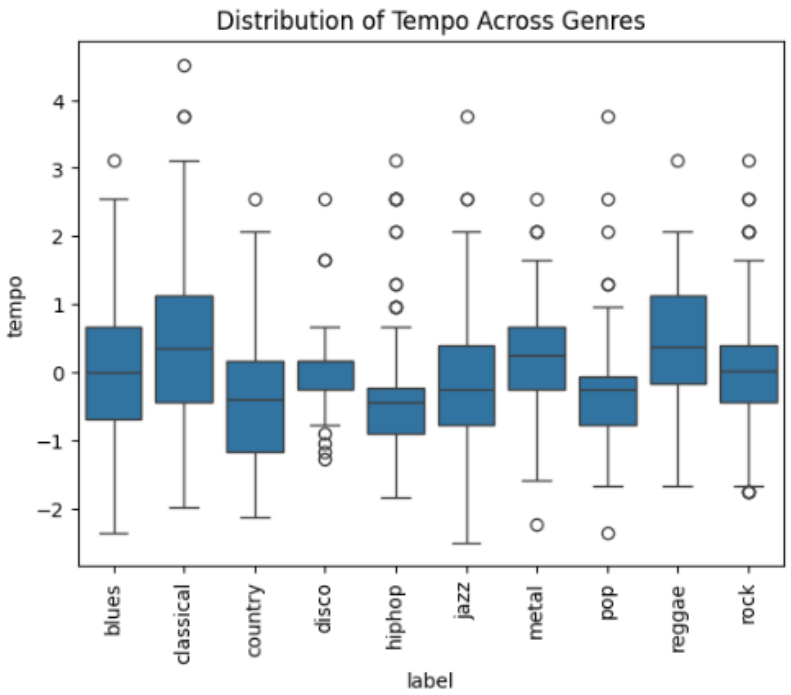
1. **Chroma Features**: These represent the twelve pitch classes (C, C#, D, etc.), capturing the pitch distribution in music.
    - **Chroma Feature Mean (chroma_mean)**: Shows the average intensity of each pitch class throughout a track. Higher or lower averages might suggest genre-specific pitch patterns.

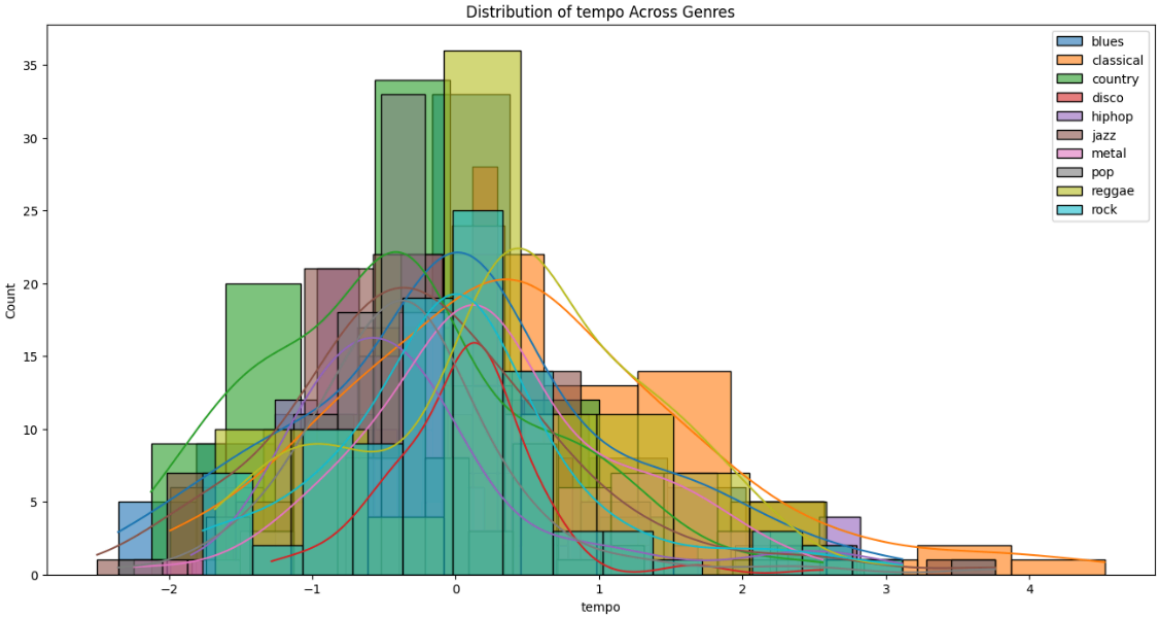
    - **Chroma Feature Variance (chroma_var)**: Measures how much pitch intensity fluctuates over time, which can indicate genre-specific dynamics in harmony or melody.
2. **Perceptual Variation (perceptr_var)**: Reflects how much the "feel" or "texture" of a track changes over time. A high perceptual variation suggests evolving intensity, common in genres with shifting textures (e.g., electronic music with drops). Lower perceptual variation is often seen in consistent genres, like classical piano.
3. **RMS (Root Mean Square)**: Measures a track's loudness or energy.
    - **RMS Variance (rms_var)**: Shows how much the loudness fluctuates. High RMS variance, indicating dynamic loudness changes, is common in genres like orchestral or rock. Low RMS variance, showing consistent loudness, is typical in ambient or minimalistic genres.
4. **Spectral Roll-Off (rolloff_mean)**: Captures the brightness of the sound by measuring the frequency below which a set portion of spectral energy lies.
    - Low roll-off values: Associated with bass-heavy, darker sounds, found in genres like dubstep or ambient.
    - High roll-off values: Indicate brighter, treble-heavy sounds, common in genres like pop or electronic music.

After we obtain the important features we can use them to see patterns or trends that help differentiate genres.

- **Exploratory Data Analysis**

- Boxplot
- Violin Plots
- Histplot

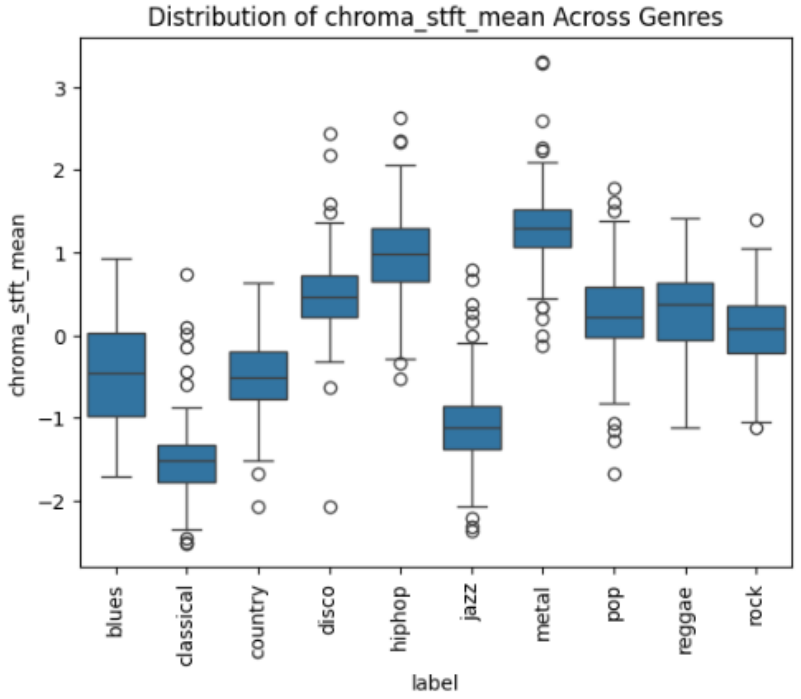


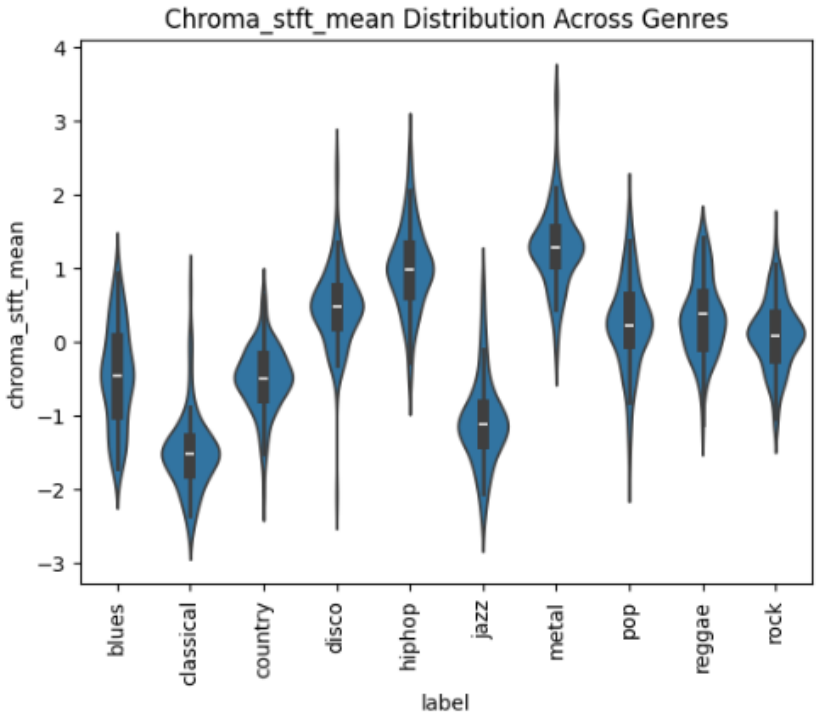



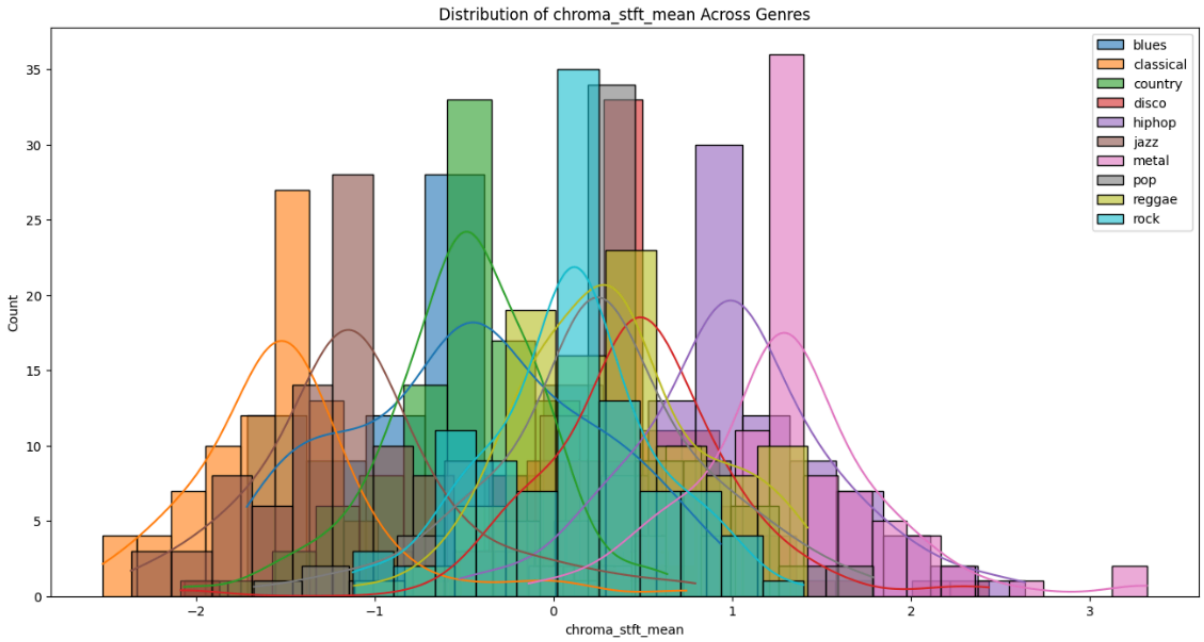

The graph indicates that the **mean tempo (average BPM)** across genres is quite similar, making it a weak indicator for distinguishing between genres. This lack of variance suggests that tempo alone may not significantly contribute to genre classification, as it doesn't capture unique characteristics that differ across musical styles.

-

In contrast, examining the distribution of chroma_stft_mean across genres reveals its effectiveness as a distinguishing feature. The variation in chroma_stft_mean between genres is more pronounced, indicating that it captures unique tonal characteristics specific to each genre. This makes it a highly significant feature for genre prediction, as its values differ enough to provide a reliable basis for classification across genres.





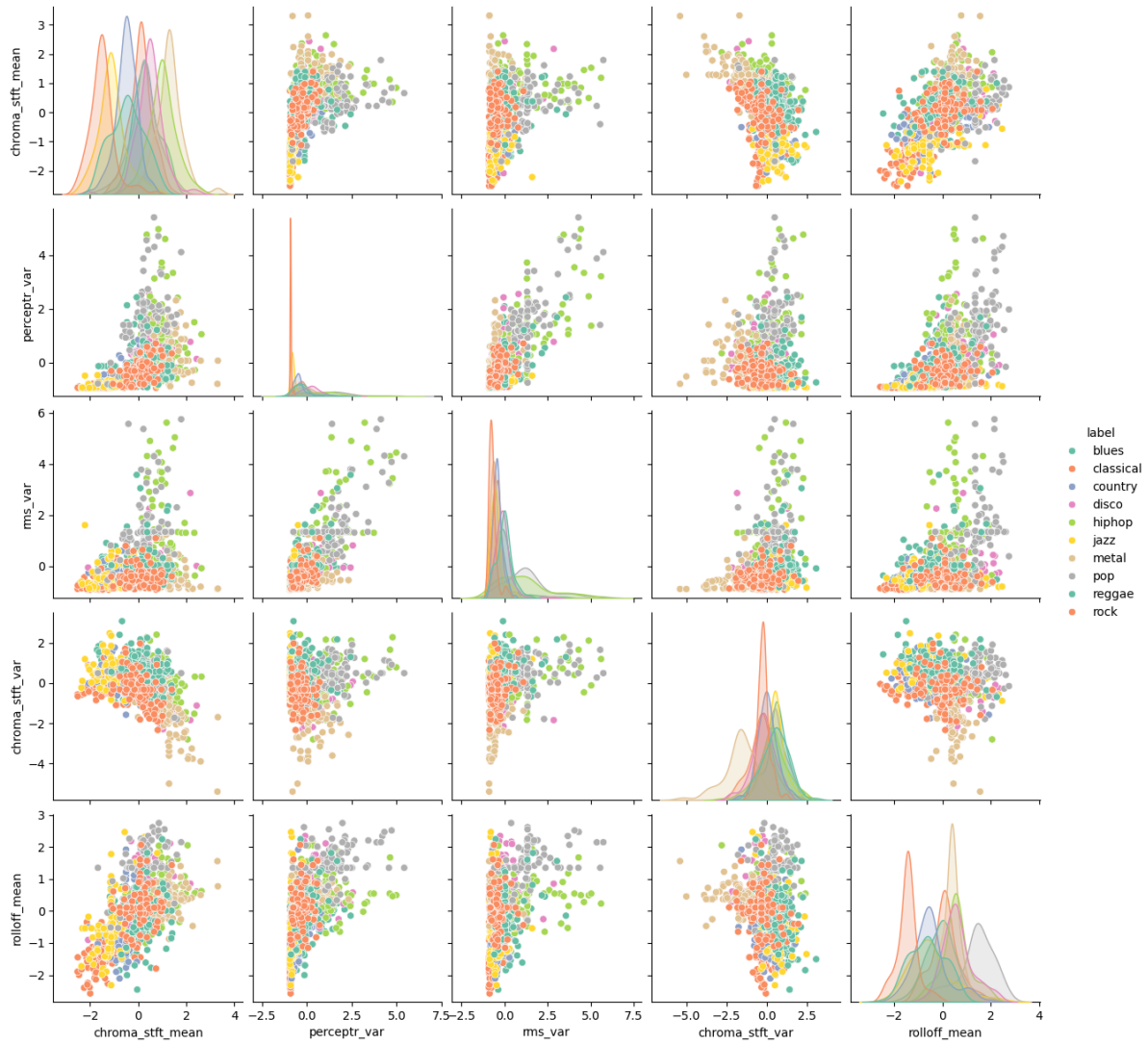

The distribution graph of chroma_stft_mean also highlights similarities between certain genres, such as country and blues, which display overlapping values. This suggests that these genres may share similar pitch or tonal characteristics, making them harder to differentiate based on this feature alone. Despite this overlap, chroma_stft_mean remains a valuable predictor for other genres where tonal distinctions are more apparent.

These plots can highlight patterns in the distribution of features (e.g., tempo, mfcc1_mean, spectral_centroid_mean) across different genres.

- **Feature Correlation and Visualization**

In the previous method, we examined each feature individually to identify which ones are most effective at differentiating between certain genres. On the other hand, to explore the relationships between features, we use the following tools

- Pair plots
- Radar charts



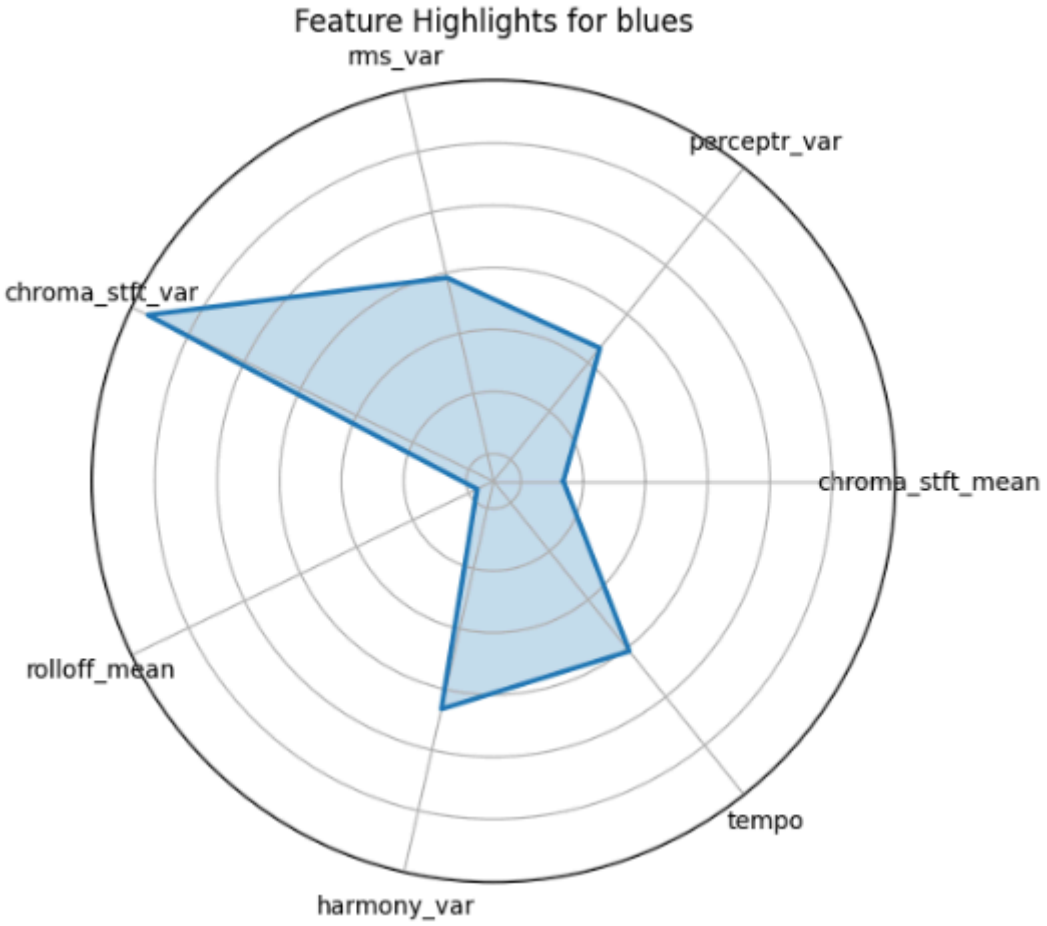

The pair plot graph between chroma_stft_mean and rms_var shows the most distinguishable clusters across genres indicates that these two features are highly effective for differentiating between them.

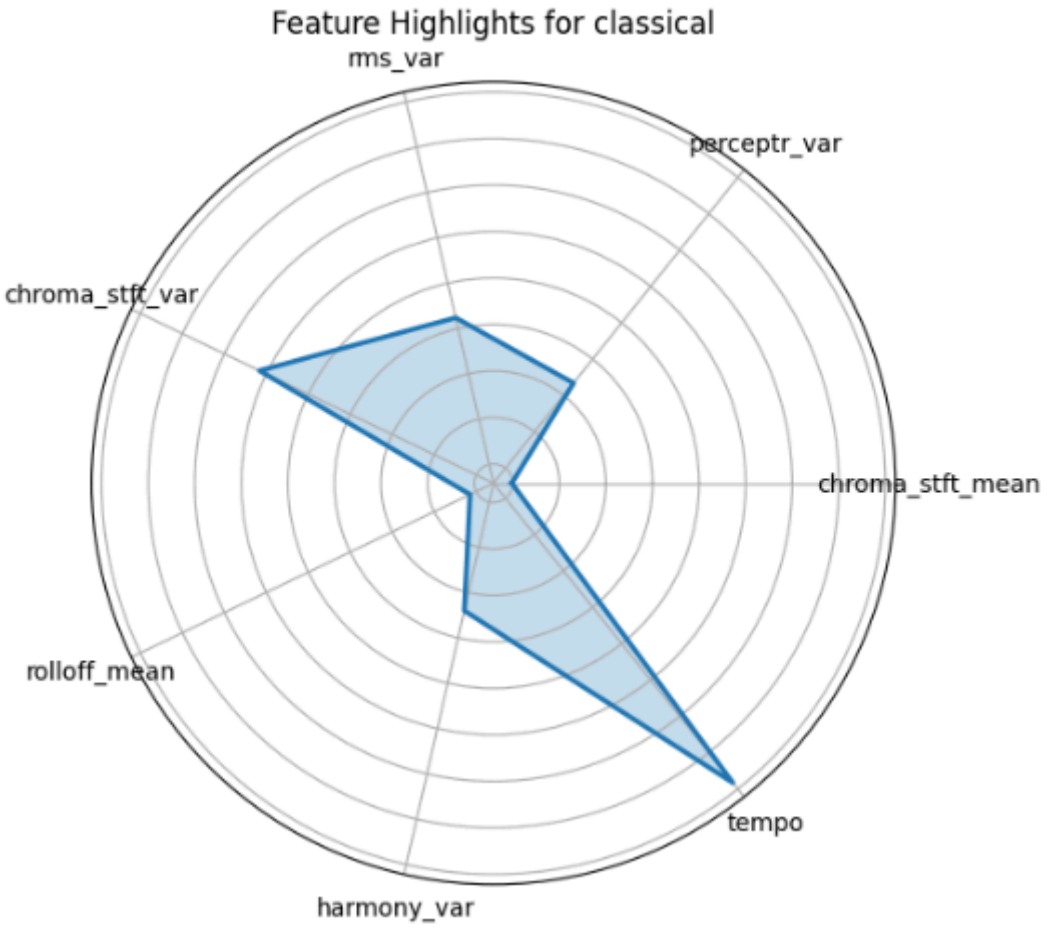
This suggests that certain genres may have unique combinations of pitch and dynamic range that are not shared by others.

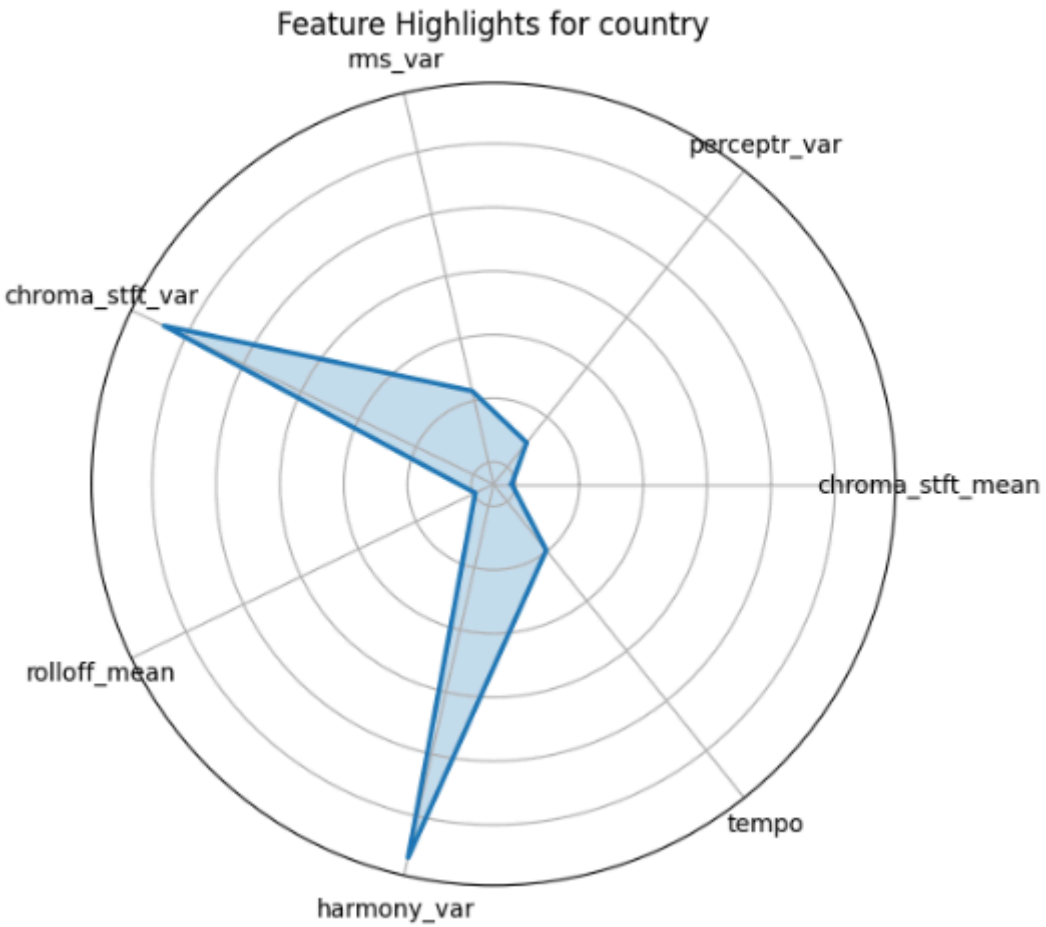
Genres with **higher chroma_stft_mean and lower rms_var** might represent more consistent, steady-pitched sounds with minimal loudness variation.

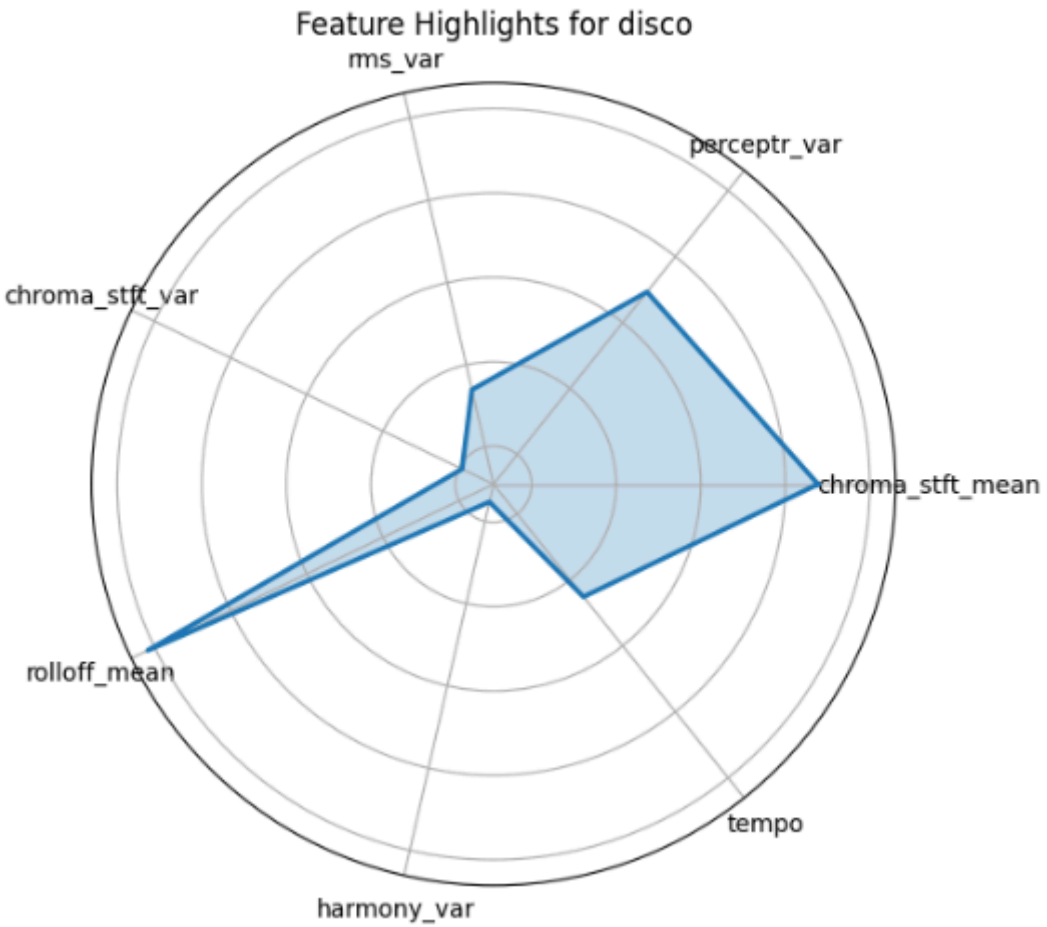
On the other hand, genres with **lower chroma_stft_mean and higher rms_var** might feature more dynamic range and pitch variability, aligning with genres that frequently change tone or energy, like jazz or rock.

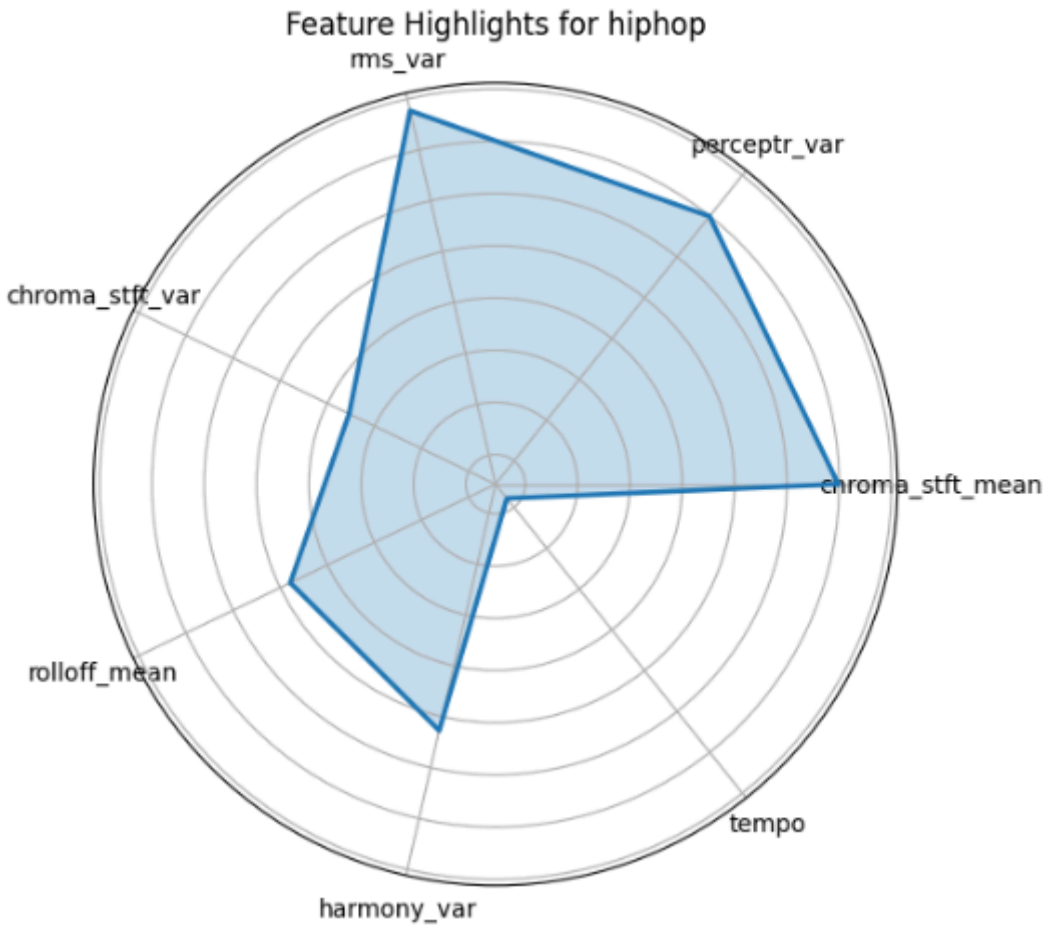
Additional relationships and characteristics can be visualized using the following radar charts.

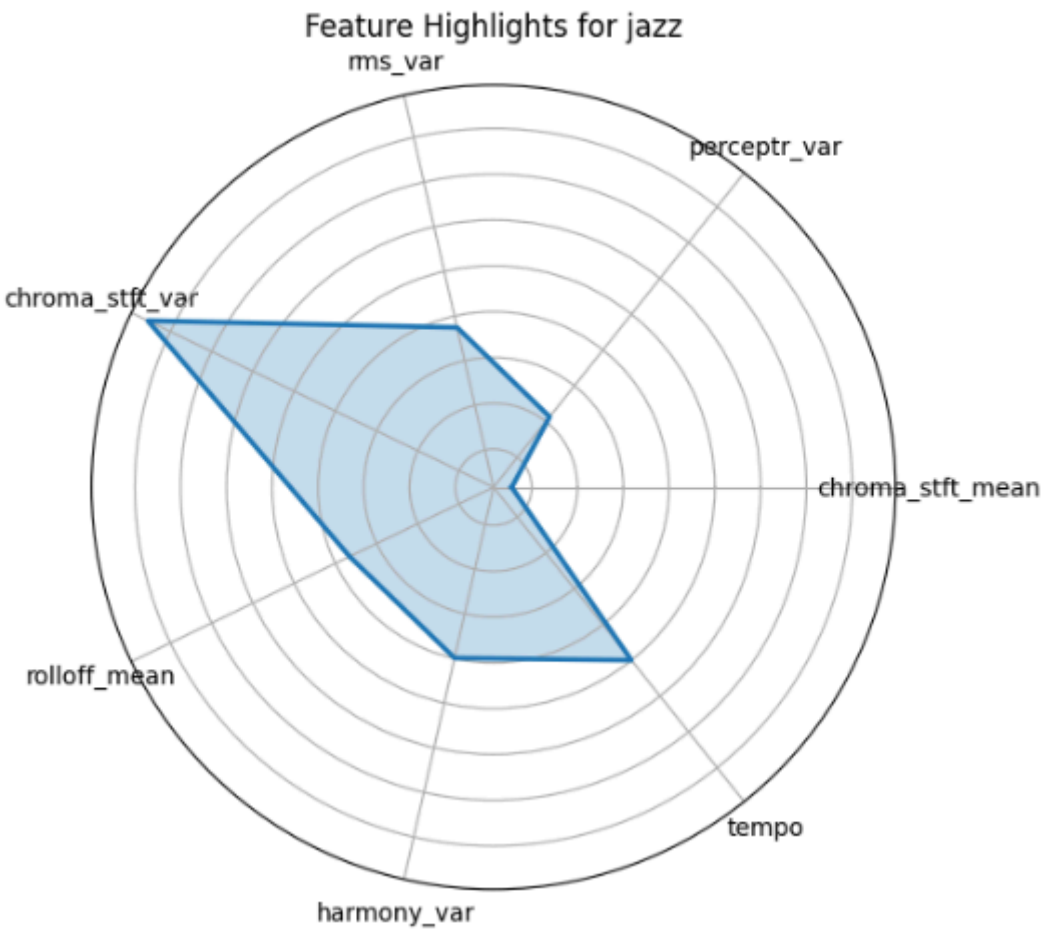


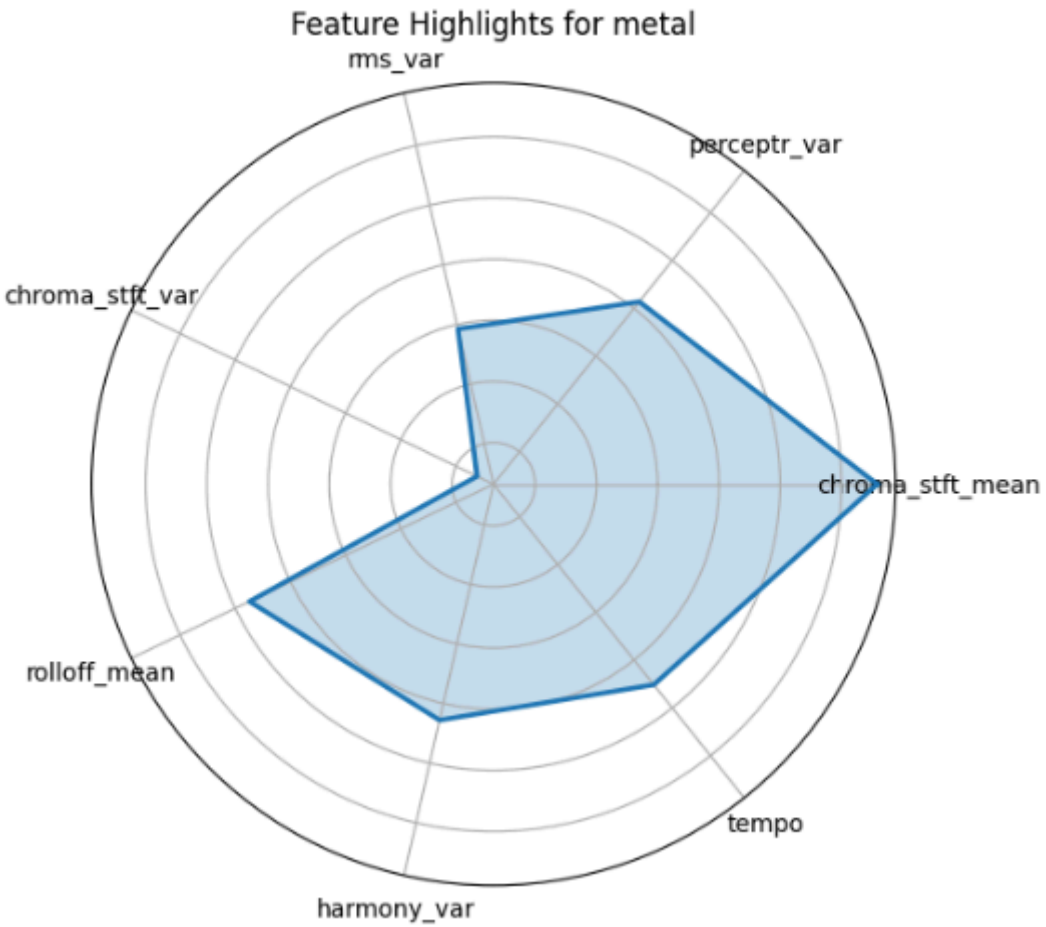

Feature Highlights for classical
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var

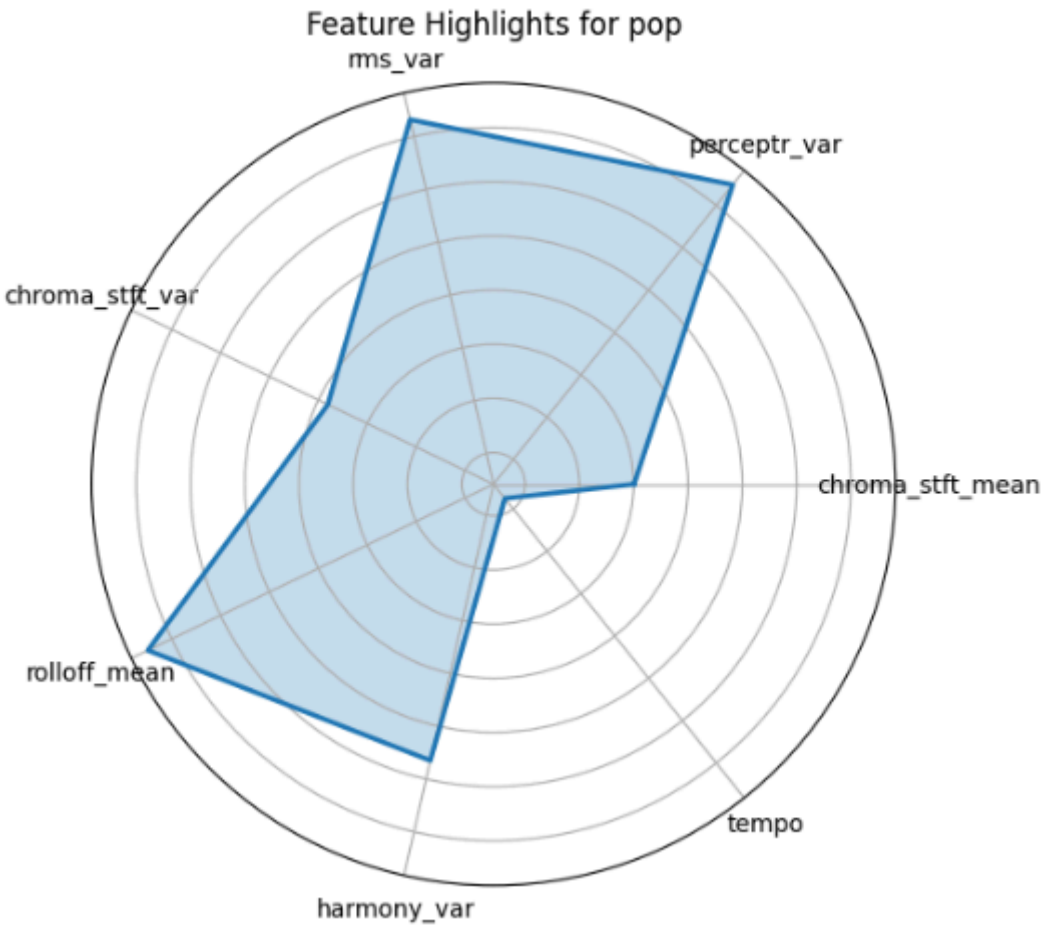

Feature Highlights for country
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var

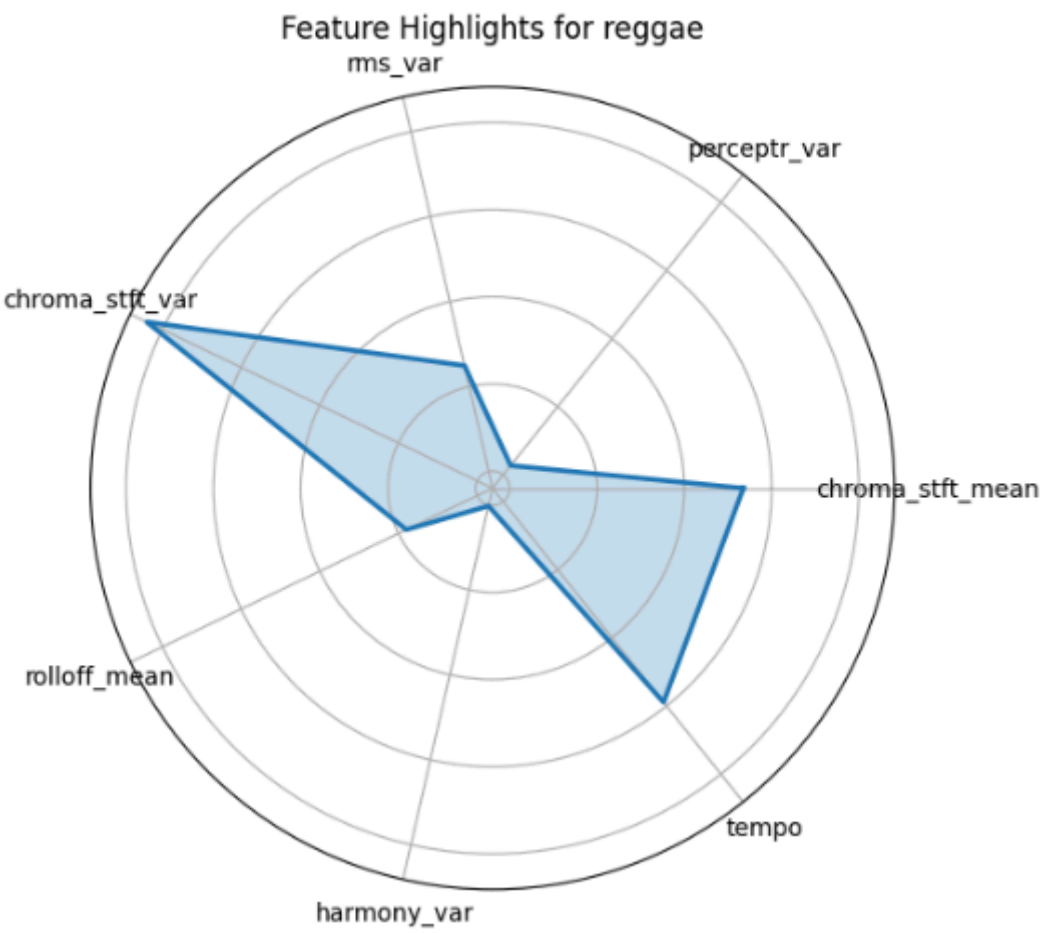

Feature Highlights for disco
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var

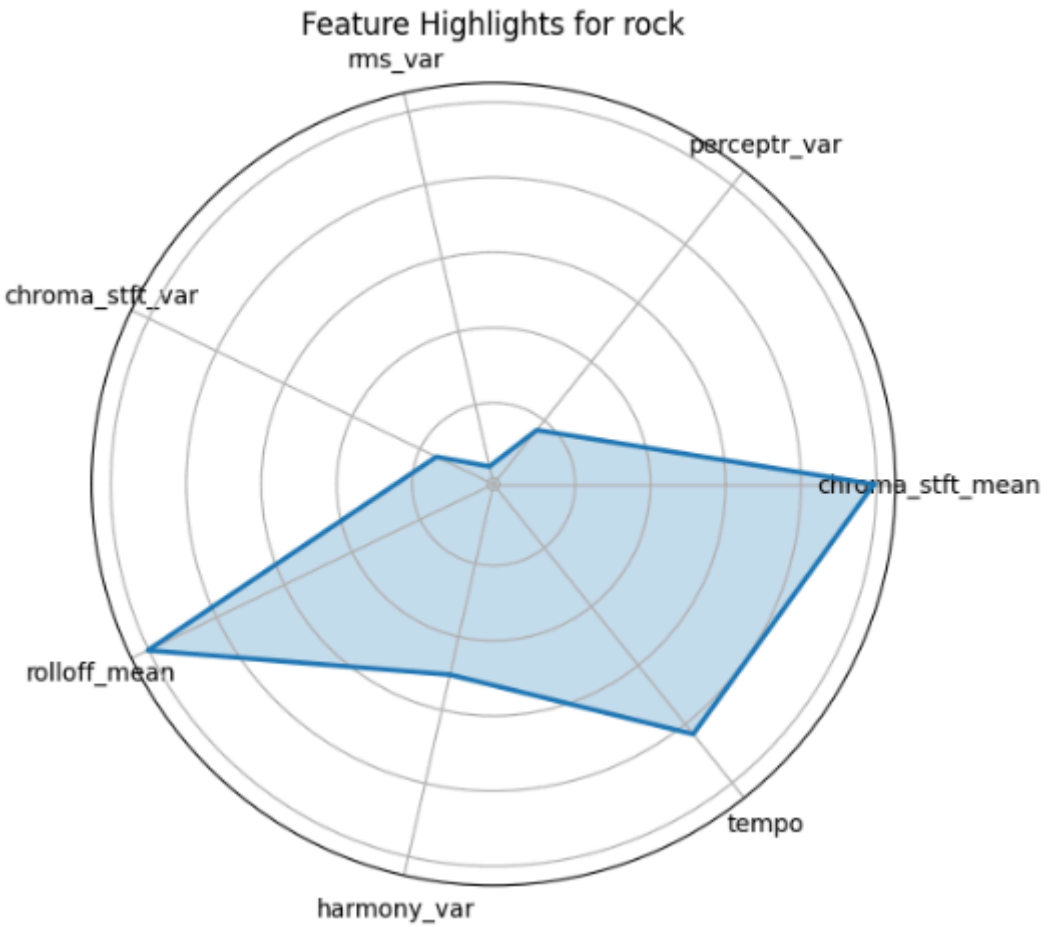

Feature Highlights for hiphop
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var


Feature Highlights for jazz
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var


Feature Highlights for metal
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var


Feature Highlights for pop
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var


Feature Highlights for reggae
rms_var
perceptr_var
chroma_stft_var
chroma_stft_mean
rolloff_mean
tempo
harmony_var




**Evaluation Metrics:** Measuring accuracy, F1 score, and confusion matrix analysis to assess the model's performance on genre classification.

See Result section for the final evaluation

# Implementation

## Tools and Software Used

### Platform and IDE :

Colab - We use Google Colab as our main platform because it allows us to run code in the cloud and share it easily. This choice provides access to powerful resources and simplifies collaboration, as everyone can view and work on the same code and results without needing special setups.

**Libraries** :

Numpy and Pandas - Mainly used for loading data and manipulating the data so that it is most effective for the machine to learn the pattern.

Matplotlib and seaborn - Use for visualizing data. In data preprocessing we use the two libraries to identify the outlier of our dataset so that we can remove those data with too high and too little impact on our result. Also we use it to visualize explained variance of PCA which further help us identify outliers for better accuracy.

Sklearn - Use for standardizing data ,hyperparameter tuning and machine learning. We use StandardScaler to first standardize the data before splitting the test. Then we use the standardized data for PCA and train_test_split. We also use the PCA function to reduce dimension so the machine has an easier time learning the pattern. With all data preprocessing and hyperparameter tuning we choose RandomForestClassifier and MLP model to classify our songs into different genres.

## Code Snippet

**Data loading using pandas** -

```
# Load data
df = pd.read_csv("features_30_sec.csv")
df = df.drop_duplicates()
```

**Data manipulation using pandas** - First, we drop the columns “length” since all the values in the columns are 30. Then we fill the nans with the average value of the feature with the same genre.

```
# Drop 'length' if it’s constant across all samples
df = df.drop(columns=['length'])
df.head()
#fill Nans with mean of label
df = df.groupby('label').apply(lambda x: x.fillna(x.mean(numeric_only=True)))
print(df.shape)
df = df.reset_index(drop=True)
df.head()
```

**Data manipulation** - before we split we will drop the labels columns

```
X = df.drop(columns=['label'])
y = df["label"]
```

**Data preprocessing** - We will select those with numerical values then remove those data which fall out of range between -1.5 x of the differences of the 5th and 95th percentile and 1.5 x of the differences of the 5th and 95th percentile.

Ex. 5th percentile value = 3 , 95th percentile value = 12

IQR = 12 - 3 = 9

lower bound = 5th percentile + ( -1.5 x IQR ) = 3 + (-13.5) = -10.5

upper bound = 95th percentile + ( 1.5 x IQR ) = 12 + 13.5 = 25.5

accept range = -10.5 to 25.5

```
# Select only numeric columns for outlier detection
numeric_cols = X.select_dtypes(include=[np.number]).columns
X_numeric = X[numeric_cols]

# Remove anomalies using IQR method
Q1 = X_numeric.quantile(0.05)
Q3 = X_numeric.quantile(0.95)
IQR = Q3 - Q1
threshold = 1.5
outliers = ((X_numeric < (Q1 - threshold * IQR)) | (X_numeric > (Q3 + threshold * IQR)))
X_clean = X[not outliers.any(axis=1)]
y_clean = y[not outliers.any(axis=1)]
```

**Data Preprocessing** - We use StandardScaler to standardize so that each column has the mean value of 0 and SD if 1. Then we split the dataset into a test set and train set.

```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_clean)
# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_clean, test_size=0.2, random_state=42)
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

**Dimensionality reduction + Hyperparameter tuning** + **Model Training** - We created a pipeline using the Pipeline function that standardizes the data and applies PCA. Then, we used the GridSearchCV to tune the hyperparameters including the  number of PCA components and Random Forest's number of trees and tree depths, through 5-fold cross-validation, selecting the best model based on accuracy.

```
# Fit PCA without reducing the number of components
pca = PCA()
pca.fit(X_scaled)

# Preprocessing pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Hyperparameter tuning with GridSearchCV
param_grid = {
    'pca__n_components': [i for i in range(40, 51)],
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
}

grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

**Evaluation** - This code evaluates the model by making predictions on the test set (X_test) using the best parameters from GridSearchCV. It prints the best parameters, the classification report (precision, recall, F1-score), and generates the confusion matrix to assess model performance.

```
# Evaluate the model
y_pred = grid_search.predict(X_test)
print("Best parameters:", grid_search.best_params_)
print("Classification report:\n", classification_report(y_test, y_pred))

# Generate confusion matrix
cm = confusion_matrix(y_test, y_pred)
```

**Alternative Model - Multilayer Perceptron**

as the random forest model seems to be underfit, as it cannot fully capture the complex nature of the dataset, we also used MLP, a more complex model to reduce underfitting.

**Data Transformation** - We use the LabelEncoder function to encode categorical labels in y into numeric values (y_encoded), and the PCA function with 41 components to reduce the dimensionality of the scaled features (X_scaled), resulting in the transformed data X_pca.

```
# Encode labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

pca = PCA(n_components=41)
X_pca = pca.fit_transform(X_scaled)
```

**Data Split and Model Training** - We use train_test_split to split the PCA-transformed data (X_pca) and cleaned labels (y_clean) into training and test sets. Then, we initialize the MLPClassifier with specified parameters and train it using the fit method on the training data (X_train_MLP, y_train_MLP).

```
X_train_MLP, X_test_MLP, y_train_MLP, y_test_MLP = train_test_split(X_pca, y_clean, test_size=0.2, random_state=42)

# Initialize the MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(256, 128, 64), max_iter=500, activation='relu', solver='adam', random_state=42)

# Train the model
mlp.fit(X_train_MLP, y_train_MLP)
```

**Model Evaluation** - We use the predict method to make predictions on the test set (X_test_MLP). Then, we calculate the accuracy using the accuracy_score function, print the classification report with the classification_report function, and generate the confusion matrix using the confusion_matrix function.

```
# Predict on the test set
y_pred_MLP = mlp.predict(X_test_MLP)

# Calculate accuracy
accuracy = accuracy_score(y_test_MLP, y_pred_MLP)
print("Classification report:\n", classification_report(y_test, y_pred_MLP))

# Calculate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
```

## Difficulties Encounter

We faced two significant challenges with our dataset:

1. **High Dimensionality and Outlier Management**: The dataset contains numerous features, making it challenging to identify which features are most impactful for model accuracy. With so many dimensions, selecting the right features without losing essential information is complex. Furthermore, the data included outliers that could skew results, but removing these outliers while preserving valuable data required careful handling.
2. **Feature Overlap Across Genres**: Another challenge was that many features were similar across different genres, which made it difficult for the model to distinguish patterns unique to each genre. This overlap led to frequent misclassifications, as the model struggled to find clear boundaries between genres based on the available features.

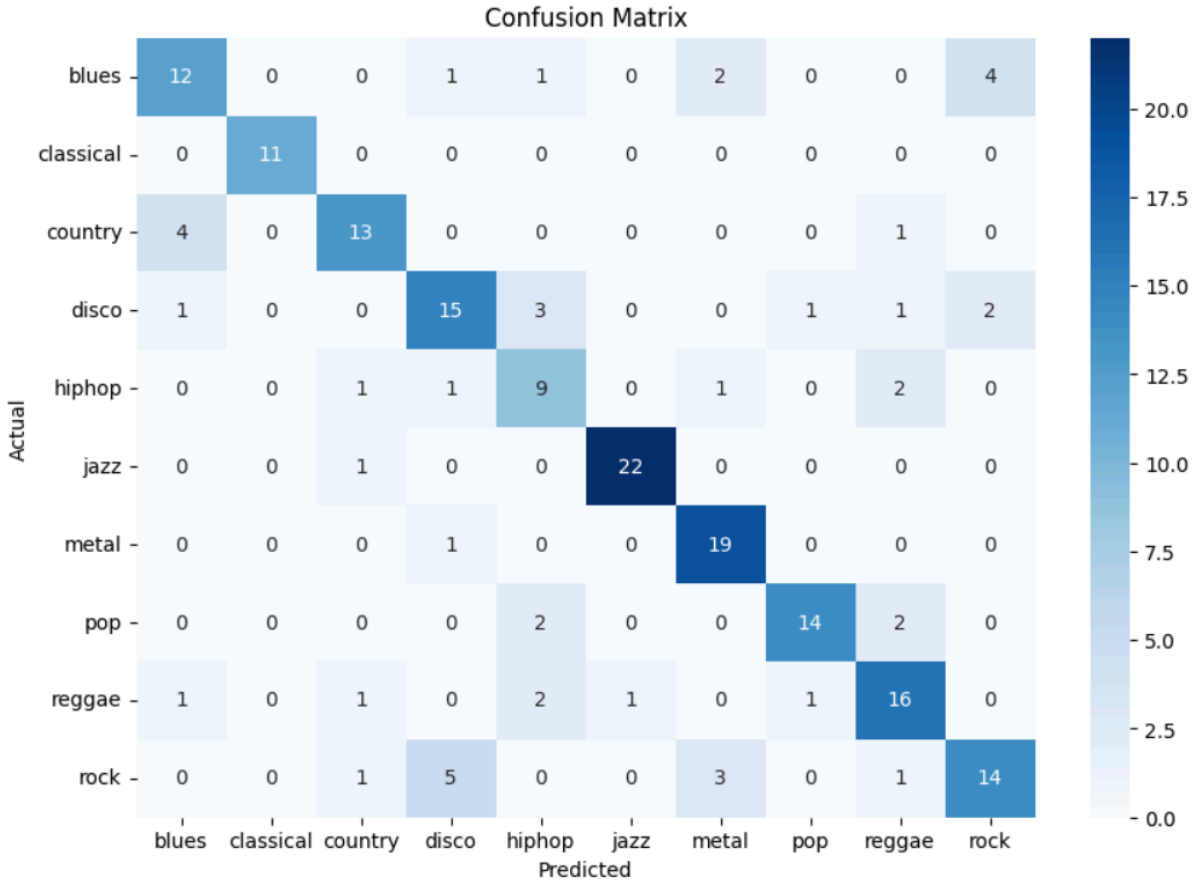
Together, these challenges complicated the data preprocessing steps, requiring us to implement techniques like PCA for dimensionality reduction and careful outlier analysis to ensure that our model could learn effectively and improve prediction accuracy.

# Results

## Confusion Matrix

## RandomForestClassifier

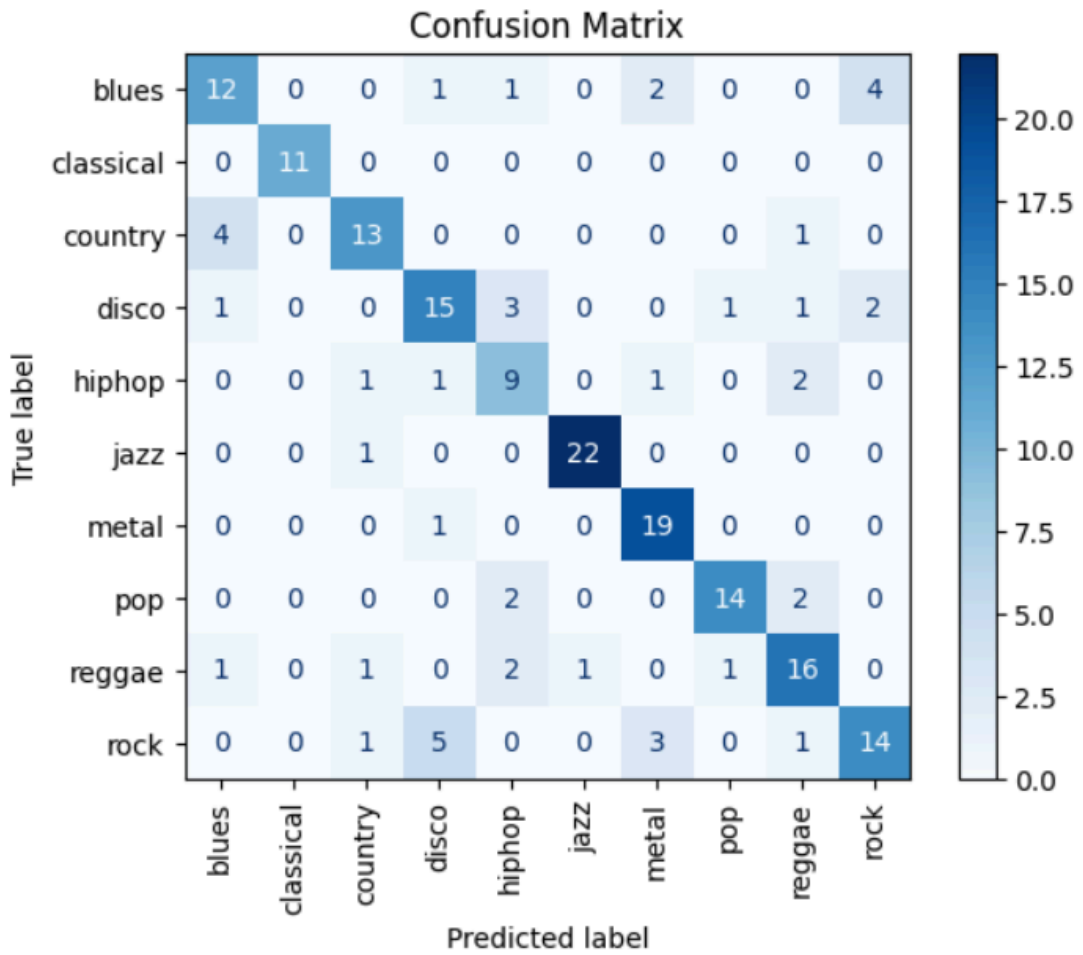
```
Best parameters: {'classifier__max_depth': 20, 'classifier__n_estimators': 300, 'pca__n_components': 41}
Classification report:
              precision    recall  f1-score   support

       blues       0.67      0.60      0.63        20
   classical       1.00      1.00      1.00        11
     country       0.76      0.72      0.74        18
       disco       0.65      0.65      0.65        23
      hiphop       0.53      0.64      0.58        14
        jazz       0.96      0.96      0.96        23
       metal       0.76      0.95      0.84        20
         pop       0.88      0.78      0.82        18
      reggae       0.70      0.73      0.71        22
        rock       0.70      0.58      0.64        24

    accuracy                           0.75       193
   macro avg       0.76      0.76      0.76       193
weighted avg       0.75      0.75      0.75       193
```



**Result : MLP**

Classification report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| blues | 0.85 | 0.85 | 0.85 | 20 |
| classical | 0.85 | 1.00 | 0.92 | 11 |
| country | 0.81 | 0.94 | 0.87 | 18 |
| disco | 0.83 | 0.65 | 0.73 | 23 |
| hiphop | 0.67 | 0.86 | 0.75 | 14 |
| jazz | 1.00 | 0.87 | 0.93 | 23 |
| metal | 0.87 | 1.00 | 0.93 | 20 |
| pop | 0.89 | 0.94 | 0.92 | 18 |
| reggae | 0.79 | 0.68 | 0.73 | 22 |
| rock | 0.86 | 0.79 | 0.83 | 24 |
| accuracy | | | 0.84 | 193 |
| macro avg | 0.84 | 0.86 | 0.85 | 193 |
| weighted avg | 0.85 | 0.84 | 0.84 | 193 |



## Result Interpretation

### Random Forest

### Best Performing Classes:

- **Classical:** Perfect performance with precision, recall, and F1-score all equal to 1.00. The model accurately classifies all classical genre instances.
- **Jazz:** High performance with precision, recall, and F1-score at 0.96, indicating strong classification.

### Moderate Performance:

- **Country:** Precision (0.76) and recall (0.72) are decent, with an F1-score of 0.74, showing the model is fairly accurate for this class.
- **Metal:** Strong precision (0.76) and high recall (0.95), resulting in a solid F1-score of 0.84.
- **Pop:** Precision (0.88) and recall (0.78) are fairly good, with an F1-score of 0.82.
- **Reggae:** Balanced performance with precision (0.70) and recall (0.73), leading to an F1-score of 0.71.

### Lower Performance:

- **Blues:** The model has lower precision (0.67) and recall (0.60), resulting in a low F1-score of 0.63, meaning it struggles to classify blues correctly.
- **Disco:** Precision (0.65) and recall (0.65) are both moderate, with an F1-score of 0.65, indicating average performance.
- **Hip-hop:** The model has lower precision (0.53) and recall (0.64), leading to an F1-score of 0.58, suggesting it's not as good at distinguishing hip-hop from other genres.
- **Rock:** Both precision (0.70) and recall (0.58) are low, resulting in a low F1-score of 0.64, showing the model has difficulty with rock genre classification.

**MLP**

**Best Performing Classes**:

- **Jazz** and **Classical**: These classes have high precision, recall, and F1-scores (close to 1.00). The model is very accurate in classifying these genres.
- **Metal**: Also performs very well, with high scores across precision, recall, and F1-score.

**Moderate Performance**:

- **Blues**, **Country**, **Pop**: These genres show decent performance with precision and recall around 0.85-0.89 and F1-scores above 0.85.

**Lower Performance**:

- **Disco** and **Reggae**: These genres have slightly lower recall values (0.65 and 0.68, respectively), meaning the model misses some instances of these genres, leading to lower F1-scores (0.73 and 0.73).
- **Hip-hop**: This genre has lower precision (0.67), suggesting that the model tends to misclassify other genres as hip-hop.

## Comparison

**Overall Accuracy :**

- **MLP:** 84%
- **Random Forest:** 75%

MLP outperforms Random Forest in overall accuracy, correctly predicting more test instances.

Macro and Weighted Averages :

- **MLP:** Macro average F1: 0.85, Weighted average F1: 0.84
- **Random Forest:** Macro average F1: 0.76, Weighted average F1: 0.75

MLP shows a better overall performance across all genres.

Comparison Conclusion:

- **MLP** performs better overall, with higher accuracy and better balance between precision, recall, and F1-score.
- **Random Forest** performs well in some genres but struggles with others, especially in recall for blues, hip-hop, and rock. MLP is the better choice for overall performance.

# Conclusion

## Summary of Finding

1. **High Dimensionality of Features**: The dataset contains a large number of features, which can lead to increased computational complexity and potential overfitting. This high dimensionality required us to apply dimensionality reduction techniques to simplify the data while retaining key information.
2. **Outlier Removal**: We identified and removed outliers to prevent them from disproportionately influencing the model's training process. Outliers can distort results and lead to inaccurate predictions, so their removal was a crucial step in ensuring model reliability.
3. **Data Normalization**: After outlier removal, we normalized the data using StandardScaler to ensure that each feature had a similar scale. Normalization was essential for algorithms sensitive to feature scales, such as PCA and neural networks, as it helps improve model performance and convergence during training.
4. **Selecting the Number of PCA Components**: Using PCA, we reduced the data's dimensionality by selecting only the most relevant components. We used explained variance to decide the number of components, choosing 41 as it captured the majority of the variance. This balance between dimensionality and information retention helped make the model more efficient without sacrificing important details.
5. **Model Training and Comparison**: We trained two separate models, Random Forest and MLP (Multilayer Perceptron), to assess which would perform better on this dataset. Random Forest, a robust ensemble model, served as a benchmark, while the MLP allowed for more complex, non-linear relationships in the data.
6. **Results and Conclusion**: Upon evaluation, the MLP model outperformed the Random Forest model, achieving higher accuracy and better classification metrics across most genres. This suggests that MLP was more effective at capturing the underlying patterns in our high-dimensional data, likely due to its capacity to model complex relationships.

Overall, our approach of outlier removal, normalization, dimensionality reduction with PCA, and model comparison led to the conclusion that MLP was better suited for this dataset than Random Forest.

**Future Work**

Given the high variability and complexity of features in our dataset, relying on manually tuned models like Random Forest and MLP may not fully capture the intricate patterns within the data. As a next step, we plan to explore deep learning architectures, which are more capable of handling complex, high-dimensional data and could potentially yield better accuracy.

Implementing deep learning models, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), could help identify deeper patterns that traditional machine learning models may overlook. However, deep learning also comes with trade-offs: these models require significantly more computational resources, including longer training times and access to powerful GPUs or TPUs. Additionally, they often demand careful tuning of hyperparameters and may be more challenging to interpret.

Despite these challenges, transitioning to deep learning is a promising path forward to achieve improved accuracy and more robust results. Leveraging our preprocessed data in this new context could help maximize the potential of our dataset and uncover valuable insights.

# References


- https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
- https://www.tutorialspoint.com/how-to-train-mfcc-using-machine-learning-algorithms

**Davis, S. and Mermelstein, P. (1980) Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. IEEE Transactions on Acoustics, Speech and Signal Processing, 28, 357-366.**

**Volkmann, J. & Stevens, S. & Newman, E.. (2005). A Scale for the Measurement of the Psychological Magnitude Pitch. The Journal of the Acoustical Society of America. 8. 208-208. 10.1121/1.1901999.**

**Gursimran Kour, Neha Mehan . Music Genre Classification using MFCC, SVM and BPNN. International Journal of Computer Applications. 112, 6 ( February 2015), 12-14. DOI=10.5120/19669-1119**

# Appendices

**Data Samples** - these missing values will be filled with the mean of respected columns with the same genre type

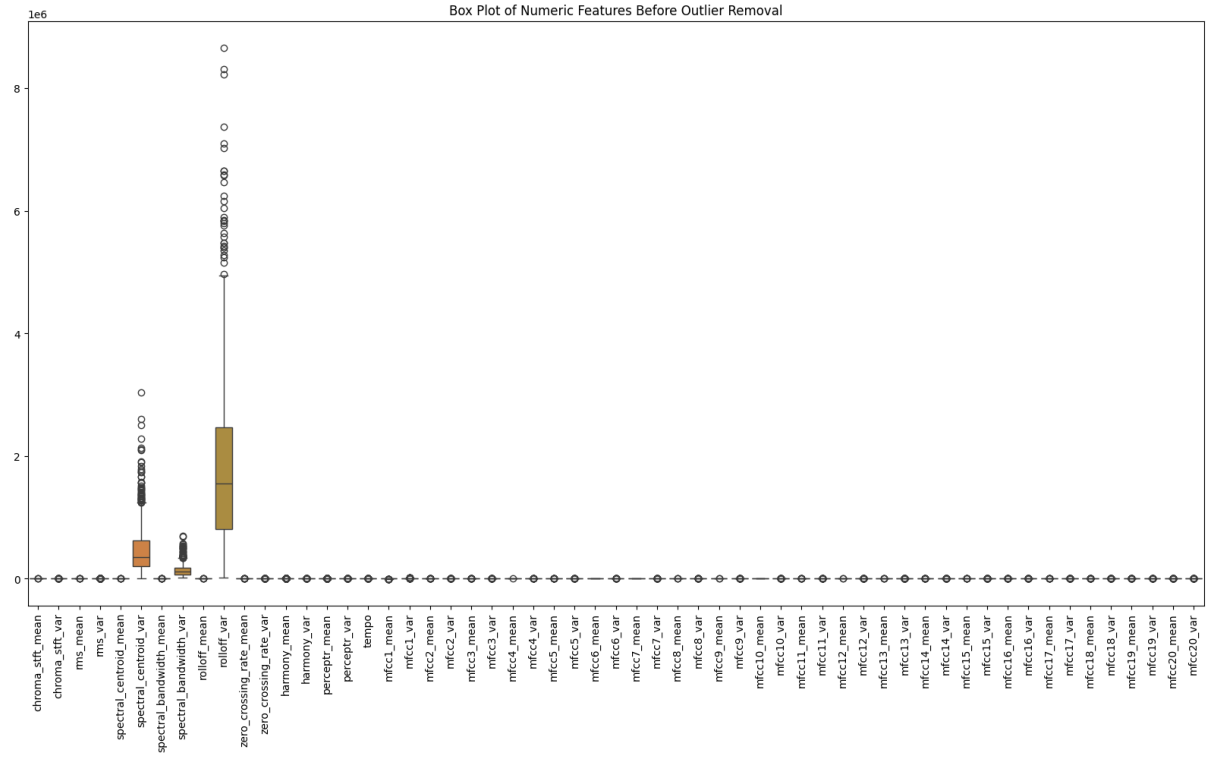
```
Missing Values:
 chroma_stft_mean           203
chroma_stft_var             169
rms_mean                    159
rms_var                     188
spectral_centroid_mean      191
spectral_centroid_var       187
spectral_bandwidth_mean     164
spectral_bandwidth_var      183
rolloff_mean                192
rolloff_var                 175
zero_crossing_rate_mean     179
zero_crossing_rate_var      192
harmony_mean                178
harmony_var                 175
perceptr_mean               187
perceptr_var                174
tempo                       173
mfcc1_mean                  167
mfcc1_var                   183
mfcc2_mean                  186
mfcc2_var                   168
mfcc3_mean                  170
mfcc3_var                   182
...
mfcc20_var                  196
label                         0
```

**Data Description** -

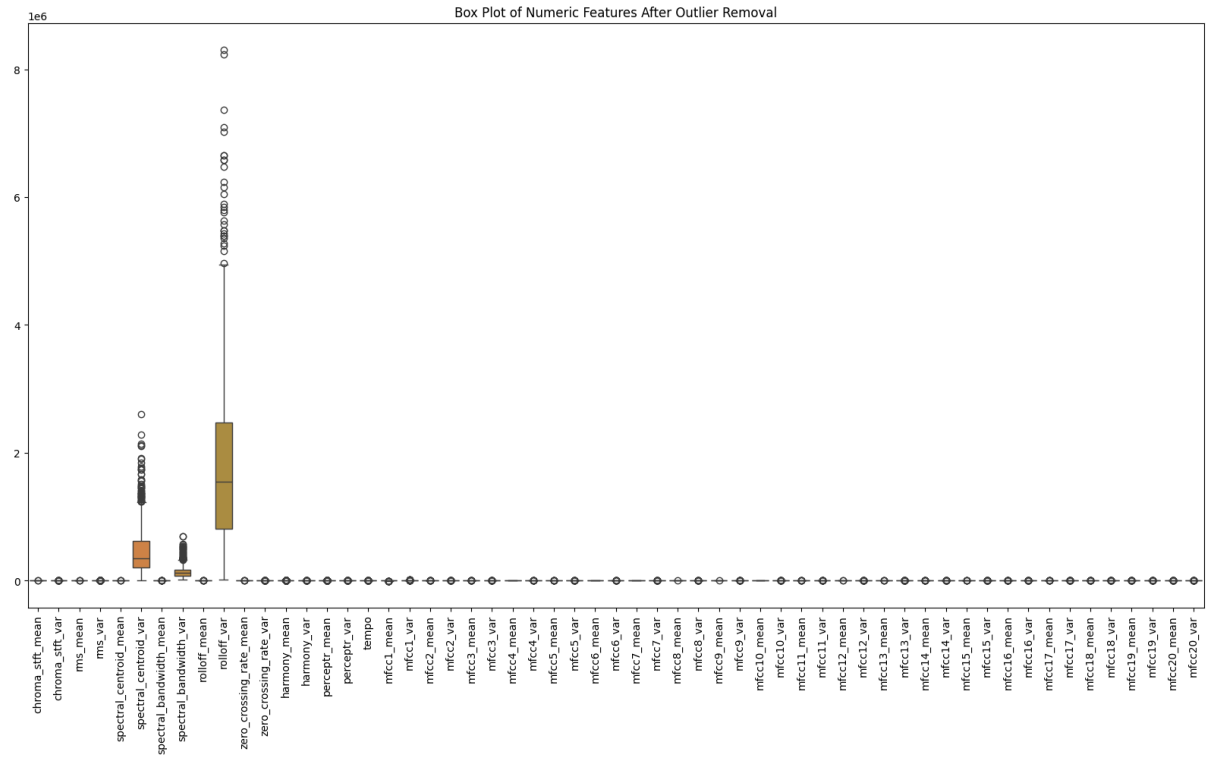
```
DataFrame Description:
       chroma_stft_mean  chroma_stft_var     rms_mean      rms_var  \
count       1000.000000      1000.000000  1000.000000  1000.000000
mean           0.379045         0.086346     0.132541     0.003042
std            0.078755         0.007219     0.062859     0.003461
min            0.171939         0.048010     0.005276     0.000017
25%            0.322776         0.082879     0.089494     0.001046
50%            0.386801         0.086726     0.129138     0.001827
75%            0.433039         0.090243     0.175939     0.003398
max            0.663685         0.108111     0.397973     0.027679

       spectral_centroid_mean  spectral_centroid_var  spectral_bandwidth_mean  \
count             1000.000000           1.000000e+03              1000.000000
mean              2202.874881           4.658462e+05              2241.064812
std                682.659325           3.816332e+05               501.978214
min                570.040355           7.911251e+03               898.066208
25%               1688.156076           2.055497e+05              1928.953538
50%               2240.651324           3.521423e+05              2230.696631
75%               2657.548935           6.205887e+05              2521.511421
max               4435.243901           3.036843e+06              3509.646417

       spectral_bandwidth_var  rolloff_mean   rolloff_var  ...  mfcc16_mean  \
count             1000.000000   1000.000000  1.000000e+03  ...  1000.000000
mean            137569.355509   4564.913302  1.836919e+06  ...     1.183186
...
max      14.694924   393.161987    15.369627   472.822266
```

## Outlier visualization :

Before removing the outlier




After outlier removal -

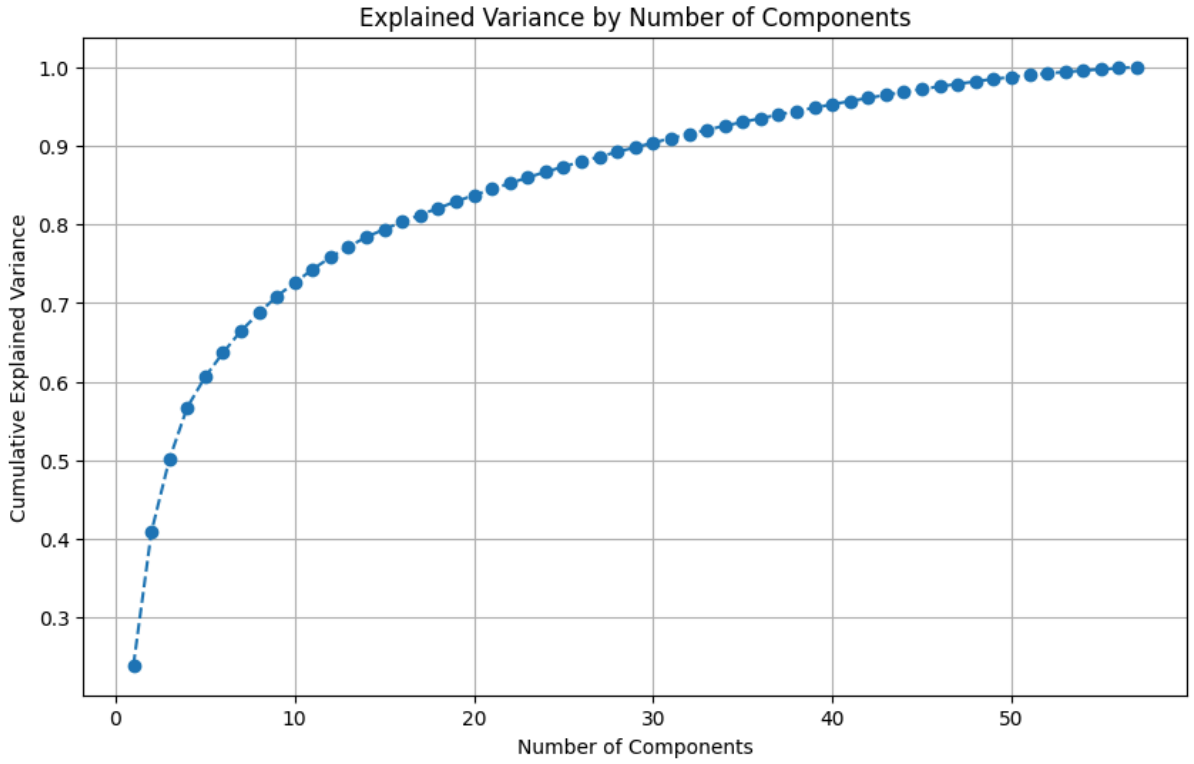


**Explained variance by number of components -**

**Cumulative Explained Variance** (y-axis) indicates the percentage of data variance captured as components are added.

**Number of Components** (x-axis) shows the count of principal components used.

Since the graph rises sharply at first and levels off around 30 components, we chose to use 41 PCA components. At this point, adding more components has minimal impact on explained variance, making it unnecessary to include additional components beyond 41. This choice balances dimensionality reduction with information retention efficiently.



# Acknowledgements

**Thanks to Dr. Chiawat Nuthong for instructions on the direction of the research. Thanks to user Andrada on Kaggle for providing the raw dataset. Special acknowledgement for Davis S., Mermelstein P. for being the pioneer of MFCC which is the basis of this research.**