Alexandra Przysucha
FML

FML Capstone

For this capstone, I did the classification project. Using the music_data.csv provided by Spotify's API, it reports the audio features of its 50,000 songs. Here, I used these audio features of the 50,000 randomly picked songs to predict the genre that the song belongs to.

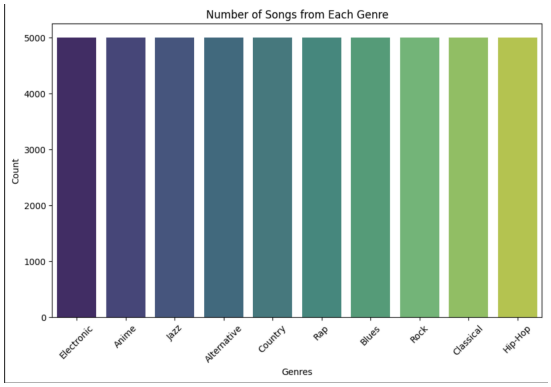
I set my random_state and random seed as my N-ID Number (16284294), to maintain individuality in my code.

```
seed = random.seed(16284294)
print(seed)
n_id_number = 16284294
```

For the dataset, first I preprocessed and cleaned it, which entailed handling the missing data. I dropped the 5 NaN rows.

```
data.dropna(inplace=True)
```

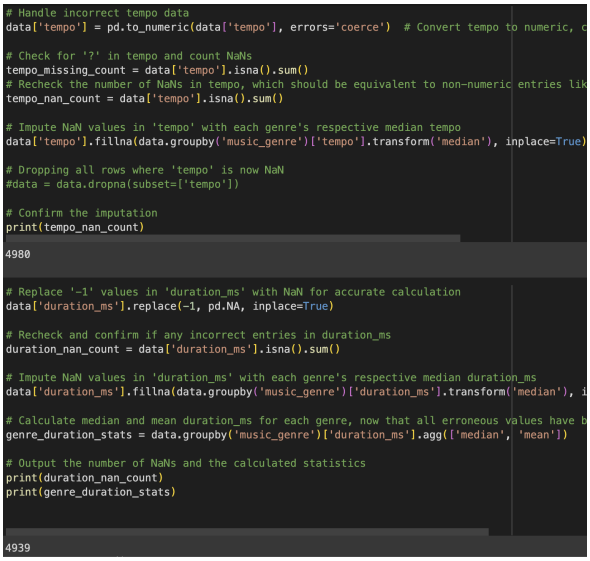
The dataset had 18 features for each song which are as follows: unique Spotify ID (instance_id), artist name (artist_name), song name (track_name), popularity (popularity) acousticness (acousticness), danceability (danceability), duration (duration_ms), energy (energy), instrumentality (instrumentalness), key (key), liveness (liveness), loudness (loudness), mode (mode), speechiness (speechiness), tempo (tempo), obtained date (obtained_date), valence (valence), and the last feature will be our target variable which is the genre of the song (music_genre), where there existed 10 different genres: (Electronic, Anime, Jazz, Alternative, Country, Rap, Blues, Rock, Classical, and Hip-Hop). The detailed descriptions for what each of the features represented and on what scale the values of each of the features spanned were stated in the spec sheet provided by Pascal, which gave me an understanding of what each feature was really measuring of each song.




```
#   Column
---  ------
0   instance_id
1   artist_name
2   track_name
3   popularity
4   acousticness
5   danceability
6   duration_ms
7   energy
8   instrumentalness
9   key
10  liveness
11  loudness
12  mode
13  speechiness
14  tempo
15  obtained_date
16  valence
17  music_genre
```

For the rows with a "?" in the tempo column of the dataset and for the rows with nonsensical values of "-1" in the duration_ms column of the dataset, since there were so many I

didn't find it logical nor reasonable to simply drop 10% of the dataset just because of that, so instead I decided for both the tempo and duartion_ms features, for any nonsensical values in those columns, I imputed them with their respective medians for each genre. What I Mean by this is that if there was a ? in the tempo column for a song that had its genre be jazz, I imputed the ? with the median value of tempo for all songs that are labeled to be jazz or be a part of the genre of jazz, not just the median value for tempo for songs in all genres. I also used median value imputation rather than mean value imputation because using the medians is a more robust method to make sure that the central tendency is maintained without being skewed by extreme values than using means.



```
# Handle incorrect tempo data
data['tempo'] = pd.to_numeric(data['tempo'], errors='coerce')  # Convert tempo to numeric, c

# Check for '?' in tempo and count NaNs
tempo_missing_count = data['tempo'].isna().sum()
# Recheck the number of NaNs in tempo, which should be equivalent to non-numeric entries lik
tempo_nan_count = data['tempo'].isna().sum()

# Impute NaN values in 'tempo' with each genre's respective median tempo
data['tempo'].fillna(data.groupby('music_genre')['tempo'].transform('median'), inplace=True)

# Dropping all rows where 'tempo' is now NaN
#data = data.dropna(subset=['tempo'])

# Confirm the imputation
print(tempo_nan_count)
```

```
4980
```

```
# Replace '-1' values in 'duration_ms' with NaN for accurate calculation
data['duration_ms'].replace(-1, pd.NA, inplace=True)

# Recheck and confirm if any incorrect entries in duration_ms
duration_nan_count = data['duration_ms'].isna().sum()

# Impute NaN values in 'duration_ms' with each genre's respective median duration_ms
data['duration_ms'].fillna(data.groupby('music_genre')['duration_ms'].transform('median'), i

# Calculate median and mean duration_ms for each genre, now that all erroneous values have b
genre_duration_stats = data.groupby('music_genre')['duration_ms'].agg(['median', 'mean'])

# Output the number of NaNs and the calculated statistics
print(duration_nan_count)
print(genre_duration_stats)
```

```
4939
```

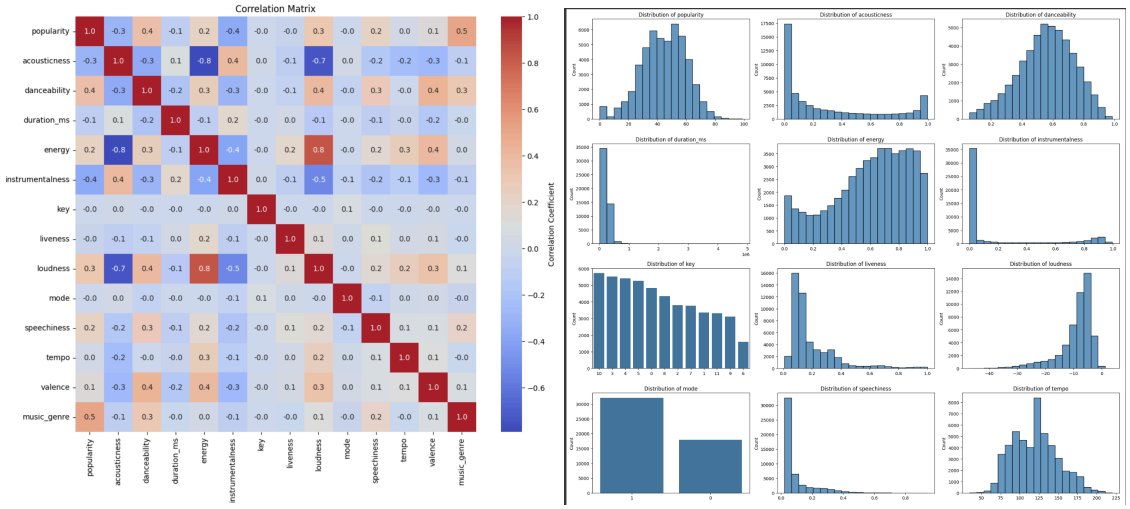
Additionally, I label encoded the music genres as follows so that they weren't string values and instead numerical: ('Alternative': 0, 'Anime': 1, 'Blues': 2, 'Classical': 3, 'Country': 4, 'Electronic':5, 'Hip-Hop':6, 'Jazz':7, 'Rap': 8, 'Rock': 9). Furthermore, I also label encoded the key variable so that it was no longer made up of string values and that it became ordinal, which the mapping is as follows: ('A': 0, 'A#': 1, 'B': 2, 'C': 3, 'C#': 4, 'D': 5, 'D#': 6, 'E': 7, 'F': 8, 'F#': 9, 'G': 10, 'G#': 11), and lastly I encoded the mode variable to be 0 if the song was in minor key or 1 if the song was in major key.

```
key_mapping = {'A': 0,'A#': 1,'B': 2,'C': 3,'C#': 4,'D': 5,'D#': 6,'E': 7,'F': 8,'F#': 9,'G': 10, 'G#': 11}
data['key'] = data['key'].map(key_mapping)
data['mode'] = data['mode'].map({'Major': 1, 'Minor': 0})

# Map and label encode 'music_genre'
genre_mapping = {
    'Alternative': 0, 'Anime': 1, 'Blues': 2, 'Classical': 3, 'Country': 4,
    'Electronic': 5, 'Hip-Hop': 6, 'Jazz': 7, 'Rap': 8, 'Rock': 9
}
data['music_genre'] = data['music_genre'].map(genre_mapping)

# Encode the target variable (genre)
label_encoder = LabelEncoder()
data['music_genre'] = label_encoder.fit_transform(data['music_genre'])
```

Ultimately, I also realized that the variables: instance_id, obtained_date, artist_name, and track_name, were all irrelevant features in terms of classification for predicting a song's genre since these features do not contain intrinsic information that correlates directly with the musical

characteristics or properties that define genres. Instead, they are really just identifiers and metadata. Even though they might be useful for organizing and referencing songs, they do not contribute to understanding the genre based on sound attributes. Thus, for the purposes of dimensionality reduction and the creation of the classification model, I dropped these features from the dataset. This helps the model focus on the acoustic and musical features that are statistically relevant and predictive of music genres.

```
# Drop irrelevant columns
data = data.drop(columns=['instance_id', 'artist_name', 'track_name', 'obtained_date'])
```



For the train-test split, I made sure to make my test set so that for each of the 10 genres, I used 500 randomly picked songs for the test set and the other 4500 songs from that genre for the training set. So, the complete test set was (5000,14), where 500 randomly picked songs were from one genre, and this was done for 10 genres. The 14 represents the 14 features left in our dataset after dropping the 4 irrelevant ones discussed previously. All of the other data was put and used in the training set to make sure there was no leakage, and thus there were 45000 songs in the train test and 14 features as well (45000, 14). Lastly, an important thing to note is that for all of the following sections and code, except for the evaluation of the classification models at the very end where the classification reports and the AUC scores are done and shown, the training set is being used. The test set is not touched nor being used at all for dimensionality reduction methods, clustering, hyperparameter tuning, or model creation.

```
train_data.shape
```

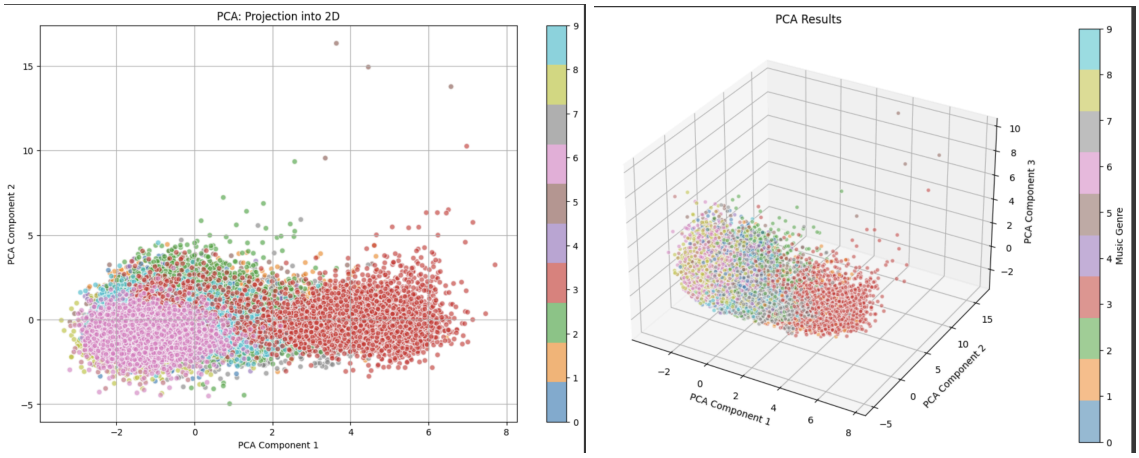
```
(45000, 14)
```

```
test_data.shape
```

```
(5000, 14)
```

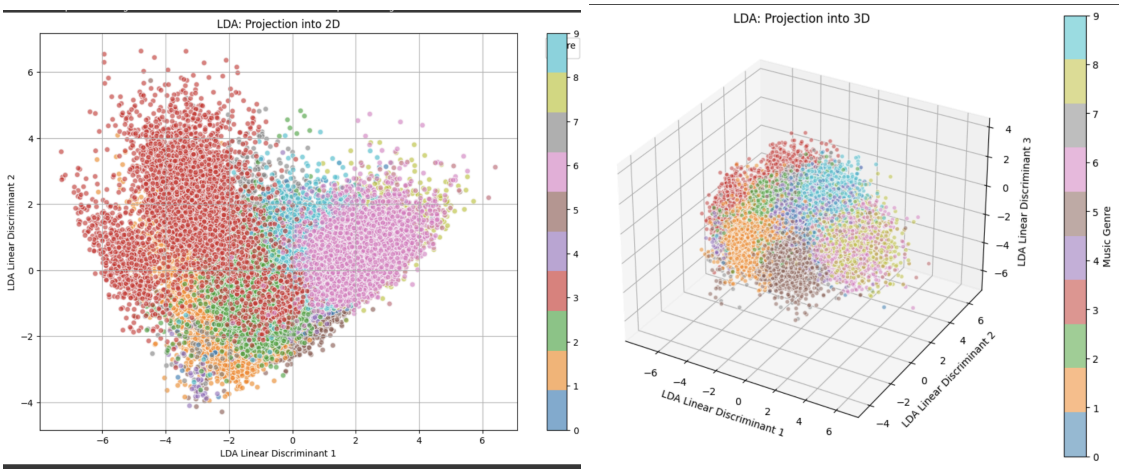
For the dimensionality reduction done prior to the creation of the classification model, I used two methods of dimensionality reduction which were PCA and LDA, and ultimately, the PCA was done to show why it is best to use LDA for this problem. First, since this is a supervised learning problem (because we have been given the music genre labels in the dataset

already), PCA is not the best since it best works with no labels. Additionally, PCA cannot handle ordinal data well, let alone it cannot reasonably and logically handle categorical data well, which both types are in this dataset after cleaning and preprocessing. Therefore, when I did do a PCA for dimensionality reduction for this dataset, I had to exclude the key and mode variables from the PCA and the standardization, because using StandardScaler for binary data like mode is illogical and nonsensical as well and because ordinal data does distort the PCA results, since PCA assumes interval data without any inherent ordering, and ordinal data is data that is in fact inherently ordered. When I did the PCA, the PCA determined that there were 3 meaningful principal components found above the Kaisen criterion which only accounted for and explained 55.64% of the variance in the data itself. If we were trying to maximize just the explained variance, then we would use 8 principal components since 8 were determined to explain at least 90% of the variance in the dataset. For visualization purposes below, I showed how the PCA was projected in 2D using 2 principal components and how the PCA was projected in 3D using the first and top 3 principal components.



As expected, the PCA visualizations did not help in terms of visualizing any groupings and this is because the PCA performed poorly due to the type of data we have for the goal of our genre classification. Therefore, the preferred and final method of dimensionality reduction that I did, and continued to use further on in the classification model was the Linear Discrimnant Analysis. LDA was chosen over PCA because it maximizes the genre class separation, and generally, it maximizes the separation between multiple classes which is good for multi-class classification problems. Additionally, for LDA, unlike in PCA, I was able to use all the variables in it and additionally, LDA is a method that uses class labels and is thus already more suitable for the music data that we are working with. For LDA, I showed 2 visualizations, similar to PCA, where the 2D plot showed the LDA projection in 2D using 2 discriminants, and the 3D plot showed the LDA projection in 3D using 3 discriminants. As seen in the 3D plot below, the LDA with 3 discriminants had a much better visualization, and thus better performance in dimensionality reduction, than all of the other visualizations (LDA with 2 discriminants, PCA with 2 components, and PCA with 3 components). Even though there were some genres that

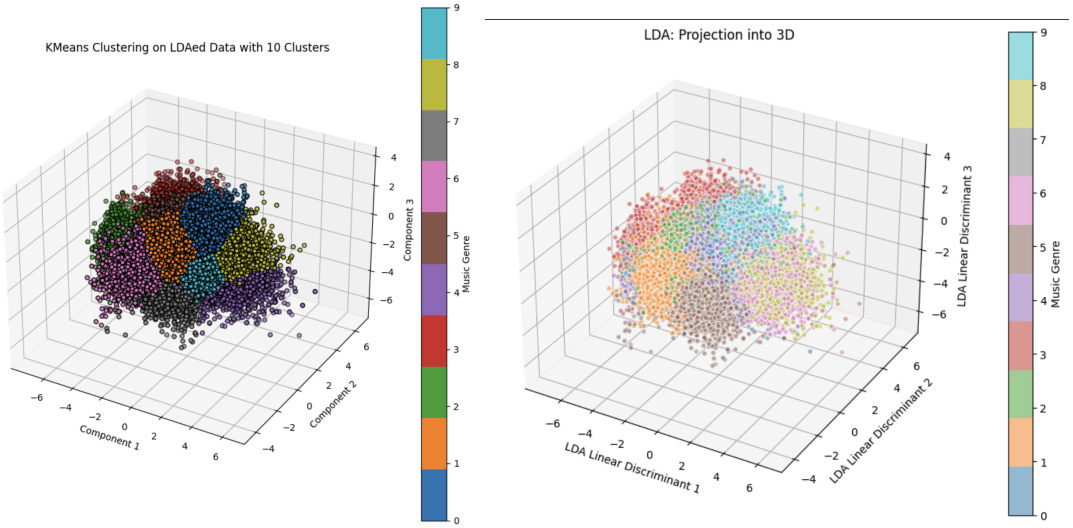
formed tight clusters, which is probably indicative of the distinct acoustic features in the data), others had lots of overlap, which highlights similarities in their audio features.



Ultimately, it is important to understand that even though nothing is cleanly clustered and separate from each other, this is a reflection of the data and what the data is, which is to say music is complicated and complex. Many genres have overlap with each other in terms of certain audio features and instrumentation. Additionally, many songs that can be seen in the duplicate song analysis done in the code and below, show that even if the same song with the same title and artist exists in the dataset twice, it might map to or be considered to be a part of 2 different genres, which is reasonable since some songs might be apart of multiple genres which is okay since music is not “black-or-white” when comes to determining what genre it is. In summary, the LDA performing this way is not due to bad LDA performance, but just the best we can do given the background behind the data we are working with.

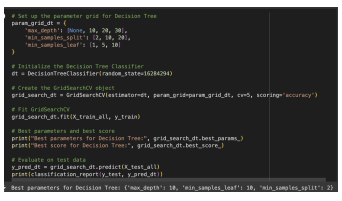
	artist_name	track_name	music_genre
3240	!!!	Even When The Water's Cold	5
17469	!!!	Even When The Water's Cold	0
28832	$uicideBoy$	2nd Hand	8
47807	$uicideBoy$	2nd Hand	6
26834	$uicideBoy$	Do You Believe In God?	8
46645	$uicideBoy$	Do You Believe In God?	6
25962	$uicideBoy$	Eclipse	8
46931	$uicideBoy$	Eclipse	6
28246	$uicideBoy$	Elysian Fields	8
47964	$uicideBoy$	Elysian Fields	6

Next, I performed clustering. Even though I know, given the labeled data, and the spec sheet provided, that we have 10 genres so there should be 10 clusters, I performed this K-means clustering on the LDA dimension reduced data just to try to see if the labels we already have are replicated by k means, where k represents the number of genres. Doing this, I plotted a 3D plot of the KMeans clustering where k =10 using the LDA-reduced data, and it came up with similar but not exactly the same clustering as was seen in the LDA 3D projection.

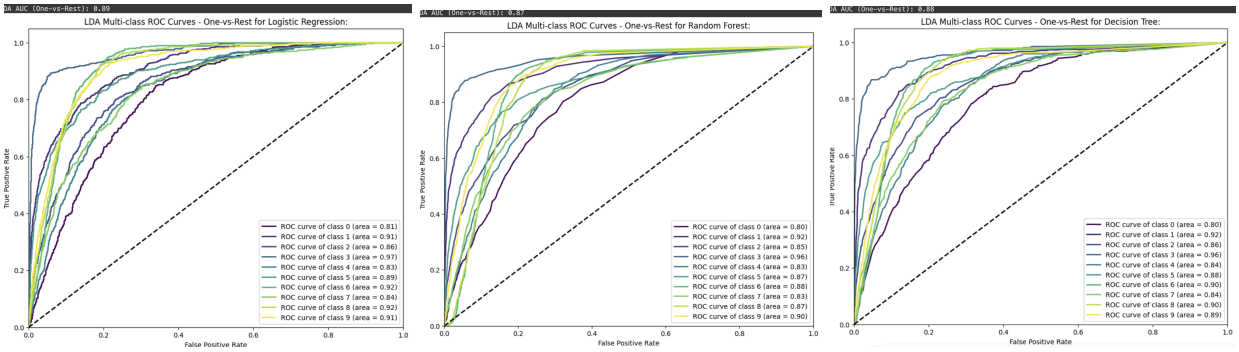


Thus, based on the conclusions made in the dimensionality reduction method section, I chose to use the LDA dimension-reduced data, which used 3 discriminants, for the creation of our classification model. I chose to review 3 classification models to see which performs the best in multi-class genre classification: Logistic Regression, Decision Tree, and Random Forest. I chose to do Decision Trees and Random Forests due to their ability to handle complex interactions between the features. In particular, Random Forests are able to provide improved generalization over Decision Trees since its method is averaging multiple decision trees on different portions of the dataset. I did Logistic Regression as well for simplicity and time complexity purposes but I knew that it would already perform slightly worse than the other two models since it is less suitable and less effective for multi-class classification problems, where it's not just two binary classes for outputs you are trying to predict.

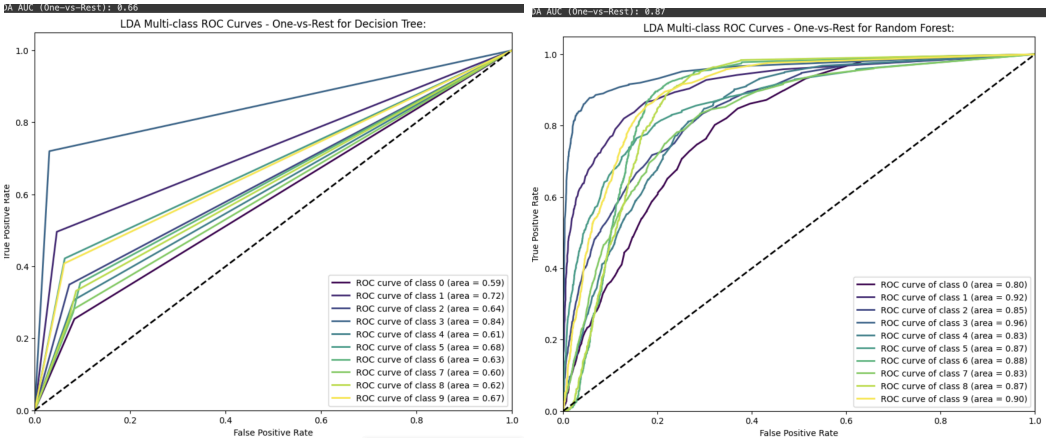
For the sake of time, the sake of my CPU not crashing, and the sake of my kernel not crashing and being unable to run my code from start to finish, I was only able to do hyperparameter tuning using GridSearchCV to find the optimal parameter settings for Decision Trees and not Random Forests due to the large number of parameters I was originally trying to test and tune for the RandomForest classifier. Using GridSearchCV, I varied the following parameters for the DecisionTree Classifier: max_depth, min_samples_split, and min_samples_leaf. These parameters control the complexity of the trees, helping to prevent overfitting while maximizing predictive power. After, doing the GridSearchCV, the best parameters that were found for the Decision Tree were the 'max_depth' to be 10, the 'min_samples_leaf' to be 10, and the 'min_samples_split' to be 2, and thus for the decision tree model performed, I would implement the following parameters into the model.



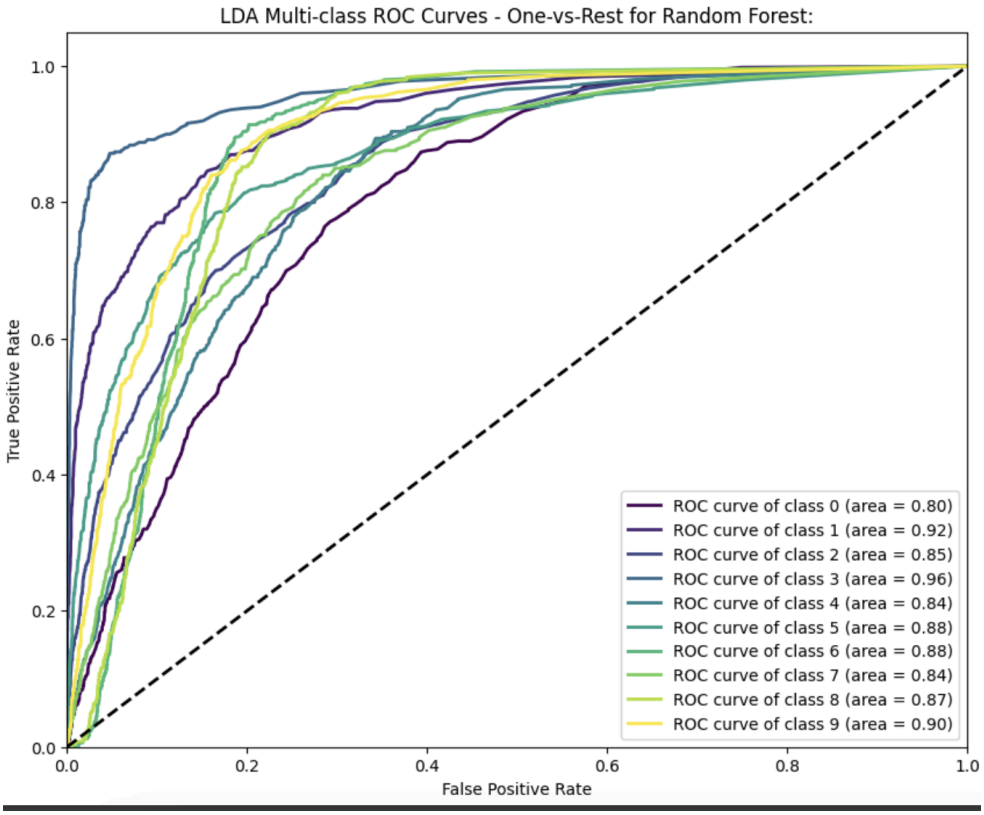
Finally, for the classification model creation, I created 3 models, using 3 different classifiers (Logistic Regression, Decision Tree, and Random Forest), all with random_states equal to my N-ID number (16284294), and for the decision tree classifier, I also implemented the best parameters found in the hyperparameter tuning. Additionally, the LDA dimensionality reduction method is used with 3 discriminants being used and recognized. Furthermore, since this is a multi-class classification problem for the ROC curves we are using a 'ovr' (One vs. Rest) approach to assess the models' performance for predicting the music genres classes, and for the AUC metric we are using the 'macro' average with the 'ovr' approach since our classes are perfectly balanced and thus macro is the appropriate average metric to use. Average = 'micro' would only be suitable to use if the classes were imbalanced. Ultimately, the AUC metrics for the models are as follows: {Logistic Regression: 0.89, RandomForest: 0.87, DecisionTree: 0.88}, but while looking at the ROC curves for each genre for all 3 models as well as the accuracy metrics for all 3 models, it becomes clear that the Logistic Regression is probably overfitting and thus the AUC being the highest for the model, and for the Random Forest, if we can overcome the obstacle with hyperparameter tuning for the Random Forest Model, it is likely it would perform even better than it already is which would ultimately be better than the Decision Tree classifier's performance after doing optimal hyperparameter tuning.



In fact, when not including the most optimal hyperparameters in the model classifier for DecisionTree, the AUC for Decision dropped to 0.66 and the ROC curves were worse than with the optimal hyperparameters as seen below in comparison to the not-hyperparameter-tuned RandomForest classifier, with an AUC of 0.87. This proves with optimal hyperparameter tuning the RandomForest classifier would absolutely perform the best out of all of the models and have the highest AUC metric if hyperparameter tuning requirements and performance were met.

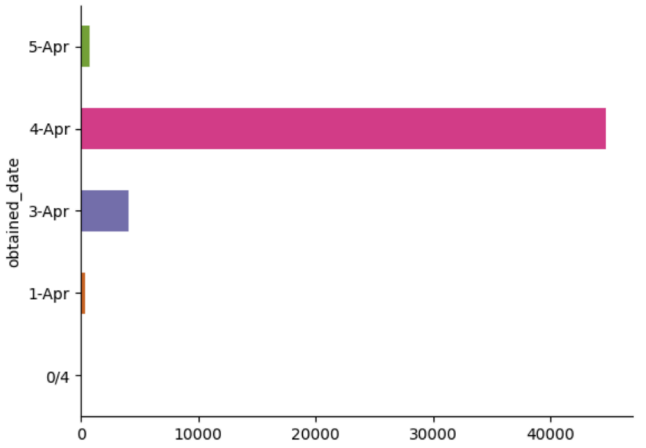


In conclusion, the best model (if we were able to perform our hyperparameter tuning to completion, which unfortunately was not able to be done, because of our kernel crashing) which is the RandomForest classifier for music genre classification with the ROC curves for each genre shown below and the **AUC metric for this model is 0.87.**

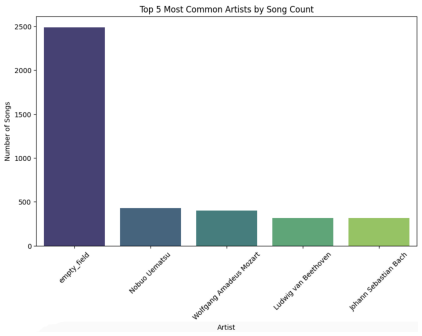


Extra Credit: Something I found interesting was in the obtained-date feature in the dataset when doing the EDA, I noticed that all of the data was obtained by Spotify within the first week of April, with a majority of the 50,000 songs being obtained on April 4th. Also, a very small percentage of the dataset is categorized to be obtained by Spotify on 0/4 which is likely a mistake in the imputation of data for the obtaiend_date feature. This is interesting because it brings up the possibility that if Spotify had obtained this music data during another week or

month, would there be more songs of a certain genre in the dataset than when compared to the dataset we have here which was all obtained in early April? It brings up the question of whether the model for predicting a song's genre would be better or worse.



Additionally, for the artists in the dataset, when I was doing the Exploratory Data Analysis (EDA), was that the "artist" that was the most common within the dataset was "empty_field" which I realized meant that a lot of genres have artist_names that are not "missing" as in NANs but just not actually correctly there. This brings up an interesting idea of whether we could somehow impute "empty_field" values in artist_name with the names of artists who have songs that have similar music features as the empty_field ones and then use artist_names in our classification.




```
music_genre
Anime          272
Alternative    272
Classical      266
Rap            263
Blues          255
Hip-Hop        245
Rock           242
Jazz           230
Electronic     223
Country        221
Name: count, dtype: int64
```