Capstone Project:

Data Preprocessing:

In this project, I aim to build an accurate classifier that retains high auc score. But before I train any models, I need to explore the missing values. In this dataset, there are different forms of missing values beside pure null values. For instance, when 'artist_name' == empty_field, 'instrumentalness' == 0, 'duration' ==-1, and when 'tempo' == ?. Since imputing missing values conditioning on music genre might cause data leakage(in the meantime, imputing without conditions can cause bias), I choose to simply drop the null values row-wisely to minimize biasing the original dataset.

```
music.dropna(inplace = True)
music.drop(music[(music['duration_ms'] == -1)].index, inplace = True)
music.drop(columns='obtained_date',inplace = True)
music.drop(music[(music['instrumentalness'] == 0)].index, inplace = True)
music.drop(music[(music['tempo'] == '?')].index, inplace = True)
music.dtypes
```

After dropping the missing values, I checked for the count of left over instances for each genre:

```
music.music_genre.value_counts()
```

```
Classical       3886
Electronic      3848
Jazz            3636
Blues           3402
Anime           2899
Alternative     2885
Rock            2880
Country         2002
Hip-Hop         1482
Rap             1465
Name: music_genre, dtype: int64
```

As shown in the image above, there exist a slight imbalance between classes, but I consider it falls into tolerable range.

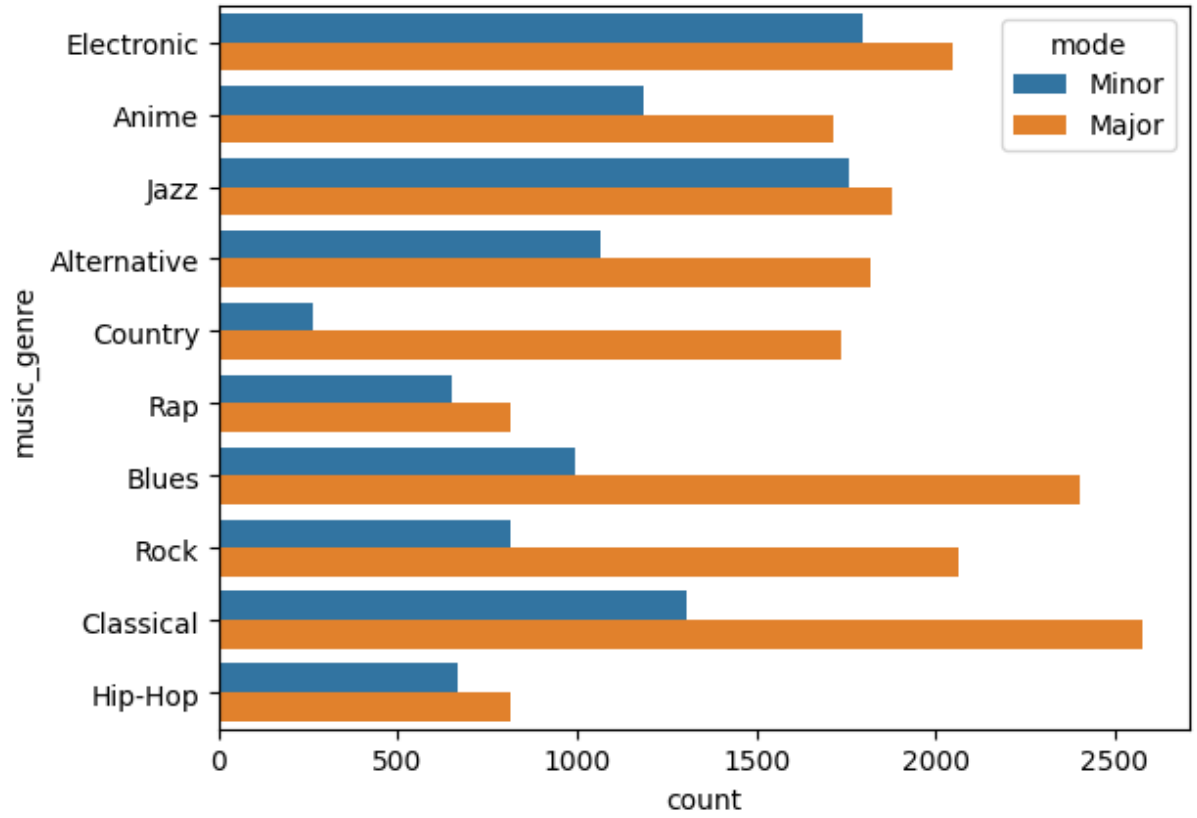
Next, I created a correlation table for feature selection:

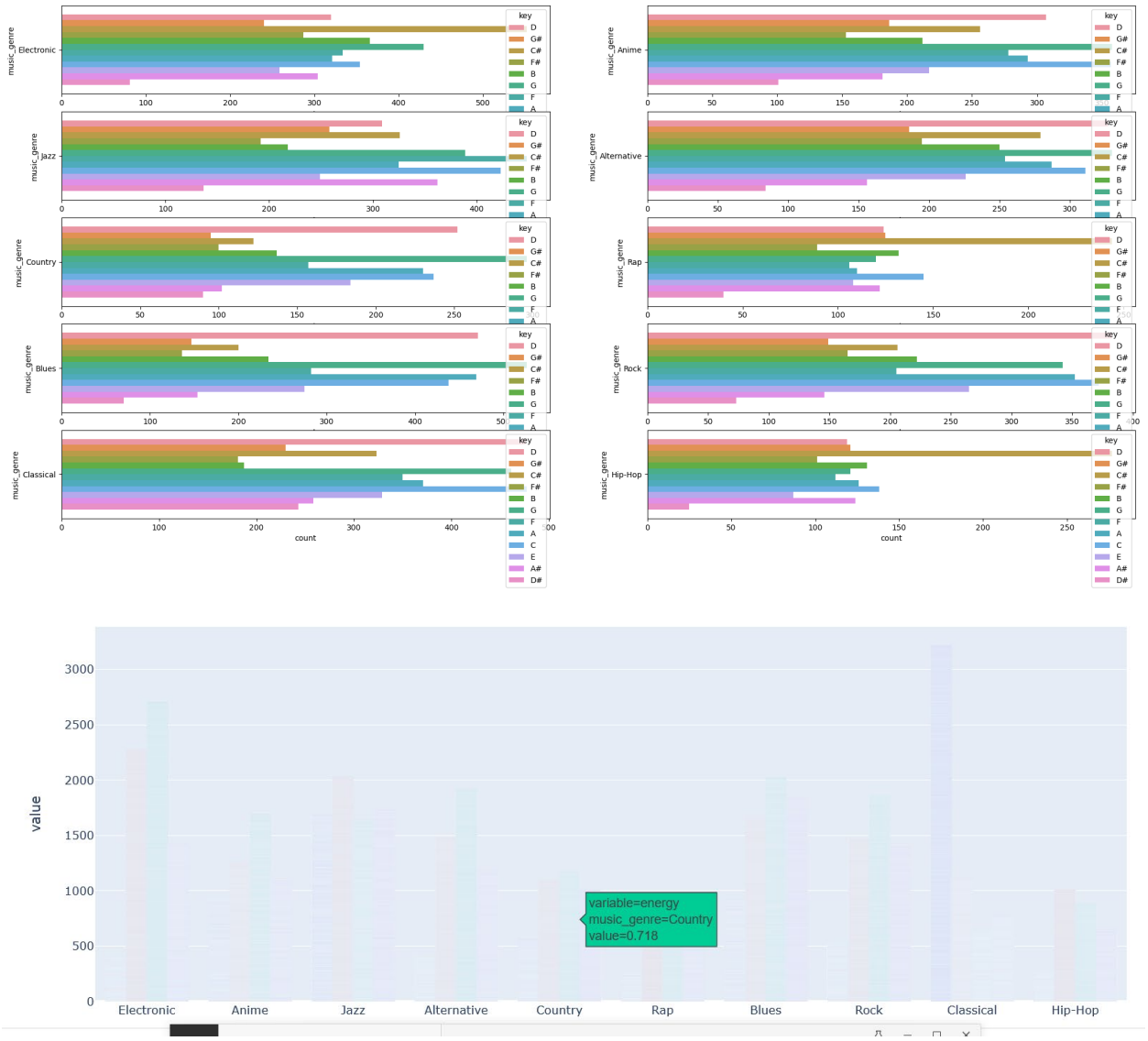
	instance_id	popularity	acousticness	danceability	duration_ms	energy	instrumentalness	liveness	loudness	speechiness	valence
instance_id	1.000000	0.003913	-0.005480	-0.004468	-0.006047	0.011046	-0.007170	0.010829	0.008806	-0.004809	0.003534
popularity	0.003913	1.000000	-0.282773	0.313461	-0.079826	0.230260	-0.348445	-0.043073	0.301922	0.143899	0.144475
acousticness	-0.005480	-0.282773	1.000000	-0.369528	0.079006	-0.819195	0.383602	-0.130092	-0.750736	-0.171699	-0.310949
danceability	-0.004468	0.313461	-0.369528	1.000000	-0.189224	0.316174	-0.267337	-0.043988	0.415549	0.224394	0.486244
duration_ms	-0.006047	-0.079826	0.079006	-0.189224	1.000000	-0.093820	0.130356	0.035686	-0.125702	-0.099594	-0.168024
energy	0.011046	0.230260	-0.819195	0.316174	-0.093820	1.000000	-0.390260	0.198878	0.850484	0.188956	0.411244
instrumentalness	-0.007170	-0.348445	0.383602	-0.267337	0.130356	-0.390260	1.000000	-0.105108	-0.505142	-0.166078	-0.270413
liveness	0.010829	-0.043073	-0.130092	-0.043988	0.035686	0.198878	-0.105108	1.000000	0.145697	0.111343	0.047025
loudness	0.008806	0.301922	-0.750736	0.415549	-0.125702	0.850484	-0.505142	0.145697	1.000000	0.177681	0.361822
speechiness	-0.004809	0.143899	-0.171699	0.224394	-0.099594	0.188956	-0.166078	0.111343	0.177681	1.000000	0.039982
valence	0.003534	0.144475	-0.310949	0.486244	-0.168024	0.411244	-0.270413	0.047025	0.361822	0.039982	1.000000

Most of these features are highly independent from each other, this is a good thing. Base on domain knowledge and common sense, I choose to drop the instance_id and popularity features. These two features are very unlikely to influence the genre prediction.

Next, I check for the non-numeral features:

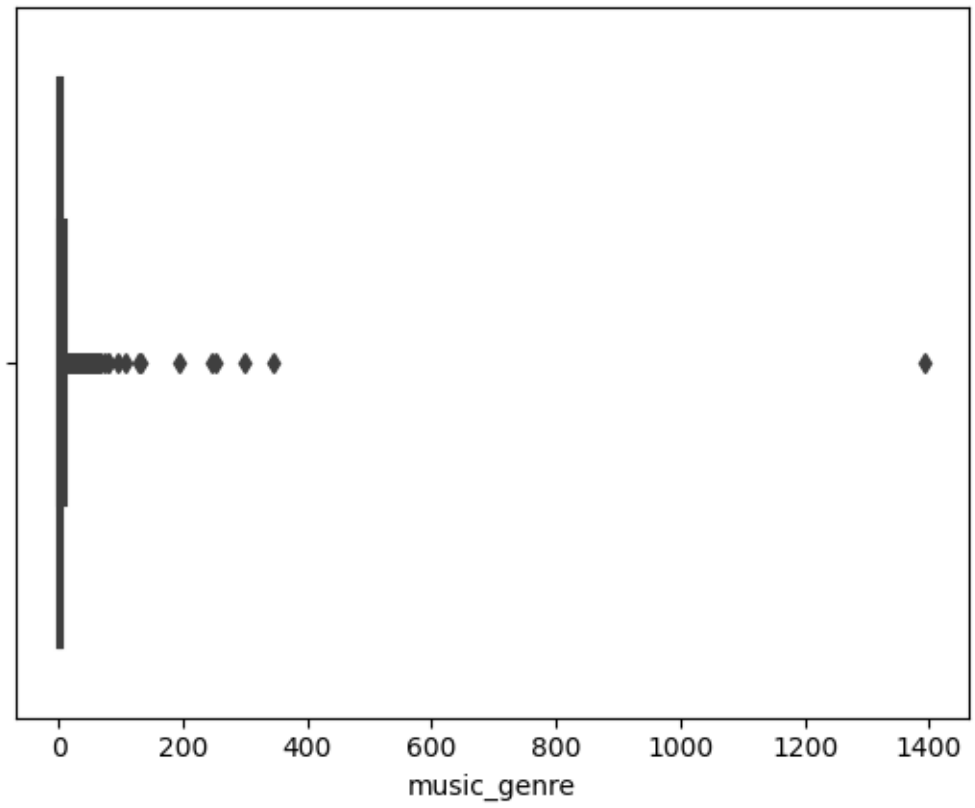
Base on my domain knowledge, mode and keys are also unlikely to influence the genre prediction. However, the visualizations propose support to an opposite opinion:



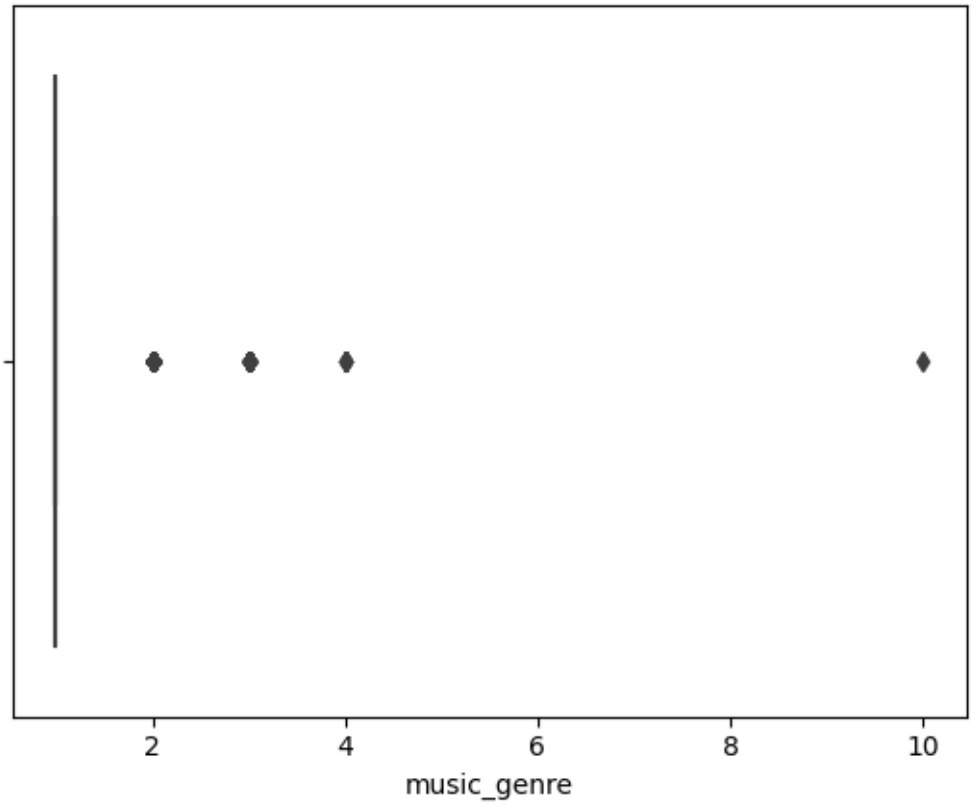


From these graphs, I failed to reject the hypothesis that these features have influence on genre. Especially in Country genre, Major music are significantly more likely to Country music than minor music. Therefore, we choose to retain these two features(mode and key).

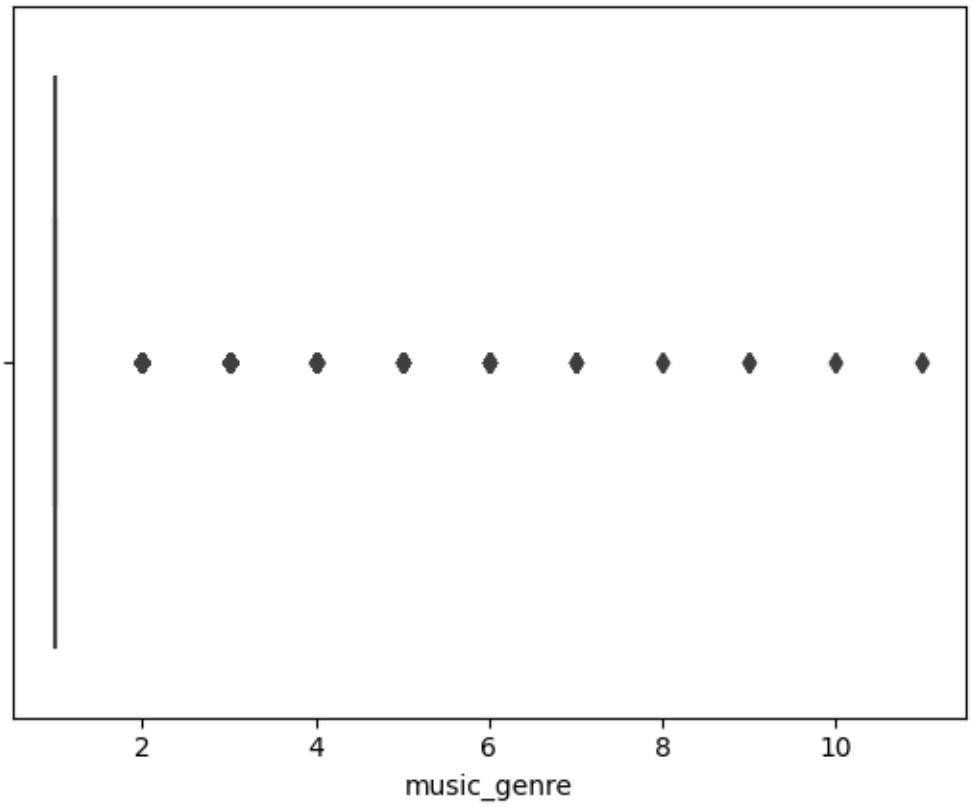
Next, I plot the distribution count of genres of an artist and distinct distribution in box plots. From these two plots, I conclude that most artist stick with one genre, therefore the feature of artist_name should significantly contribute classifying music genre. For instance, Wolfgang Amadeus Mozart is an indicator to Classical Genre. In addition, all artists in this dataset(after dropping null) compose music less than 4 genres.

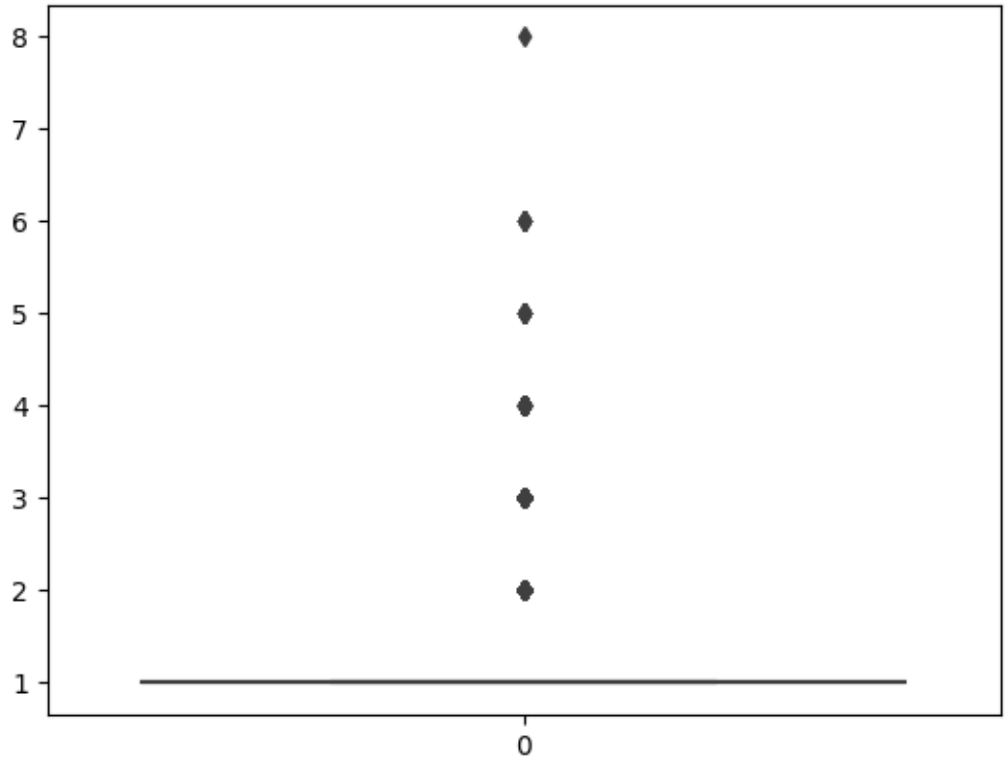


Distinct One:



I also plot the track name:





Although both track name and artist name tend to stick with one genre, there are essential difference between track name and artist name. From previous plots, I learned that most artists composed more than 10 pieces while most track contains less than 10 pieces of songs. Therefore, training with track name would make our model less robust in real world. So I choose to discard this feature.

track_name	music_genre
Summertime	11
Home	11
I Don't Know	10
Without You	10
Forever	9
...	...
Hood Cycle (Bonus)	1
Hoochie Coochie Man - Live	1
Hoochie Coochie Man	1
Honkytonk U	1
黒い雫	1

25180 rows × 1 columns

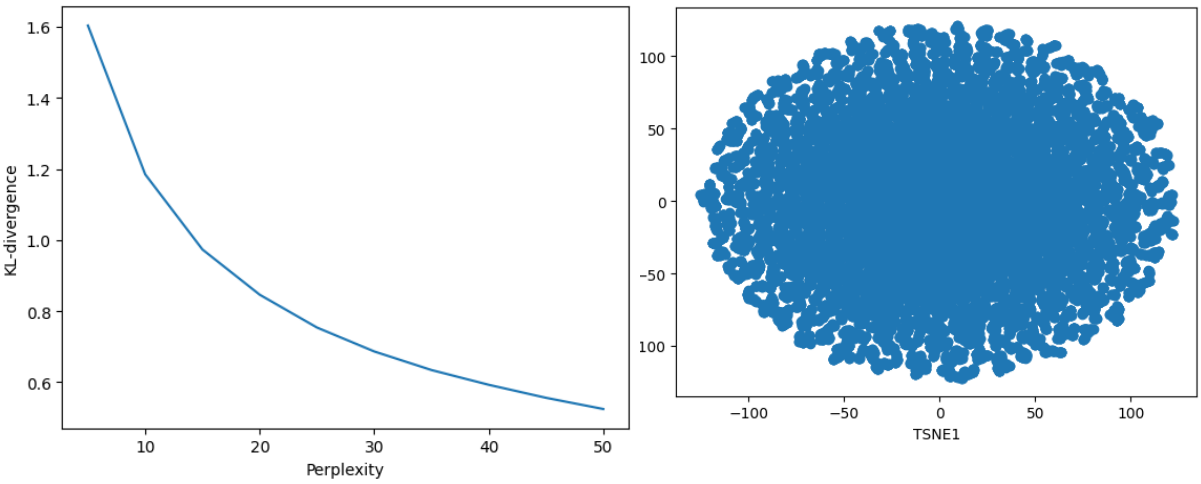
Nobuo Uematsu	346
Wolfgang Amadeus Mozart	299
Ludwig van Beethoven	253
Johann Sebastian Bach	248
...	...
Jim Capaldi	1
Beirut	1
Jidenna	1
Jibbs	1
Blue Six	1

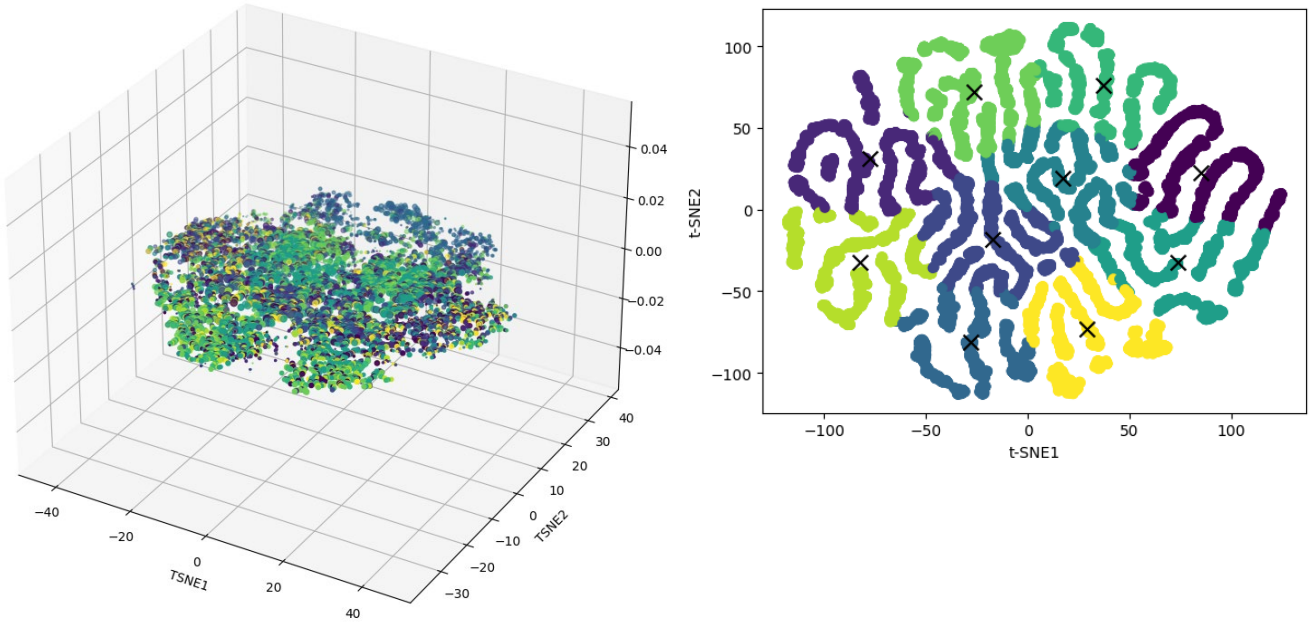
After selected these features, I performed label encoding on key, genre, and artist_name to prepare future dimension reductions. Then I standardized numerical features(not categorical features). In the end I performed a train test split. For *each* genre, I used 500 randomly picked songs for the test set and the other songs from that genre for the training set. So the complete test set will be 5000x1 randomly picked genres (one per song, 500 from each genre).

Dimension Reduction and Clustering:

I tried two dimension reduction approaches, t-SNE and PCA. PCA is a linear dimensionality reduction technique and preserving global structure well while t-SNE is a nonlinear dimensionality reduction technique that preserving local structure well.

For t-SNE, I first plot the perplexity vs. KL-divergence graph, then plot the t-SNE with different perplexity(5 vs 15) in 2D and 3D.





As shown in the images above, the result did not go well. It is really difficult to distinguish different genres. The clusters are not representative to genres. I test how it performs on different models, and compare with the performance of naïve model, none of models scored well.

```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(tsne_result, y_train)
pred = neigh.predict(X_test_transformed)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, neigh.predict_proba(X_test_transformed), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 10.68%
AUC Score: 0.51
```

```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
pred = neigh.predict(X_test)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, neigh.predict_proba(X_test), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 55.82%
AUC Score: 0.84
```

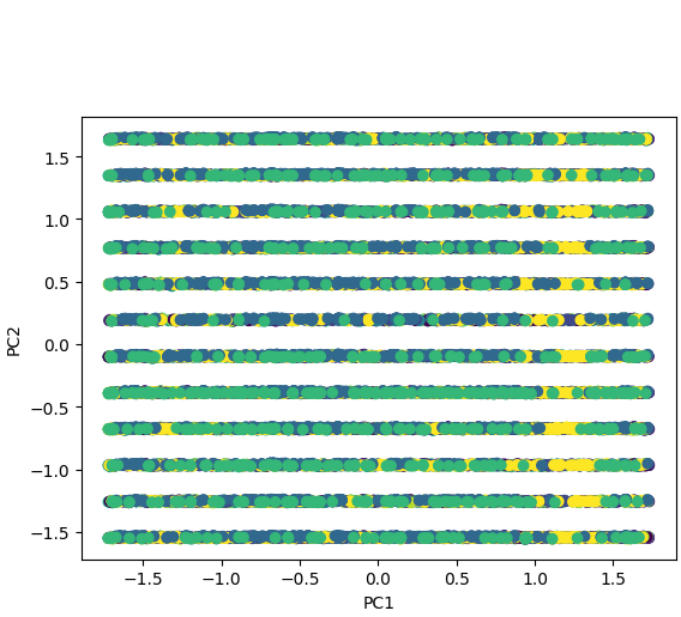
```
clf = LogisticRegression(random_state=0).fit(tsne_result, y_train)
pred = clf.predict(X_test_transformed)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, clf.predict_proba(X_test_transformed), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

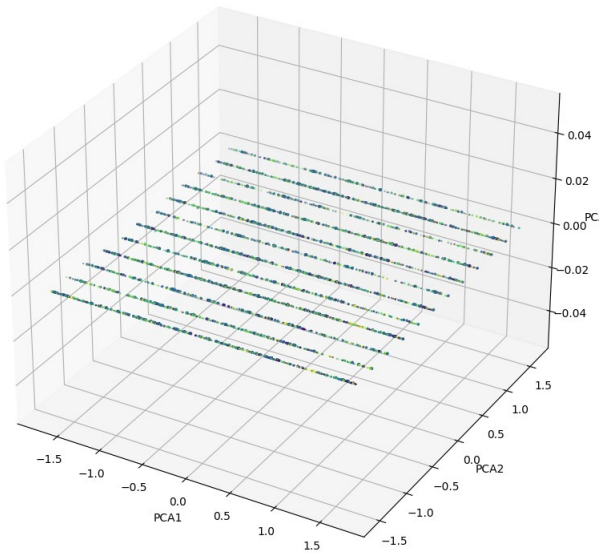
```
Accuracy: 10.70%
AUC Score: 0.52
```

```
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, clf.predict_proba(X_test), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 29.14%
AUC Score: 0.77
```

How about PCA?:



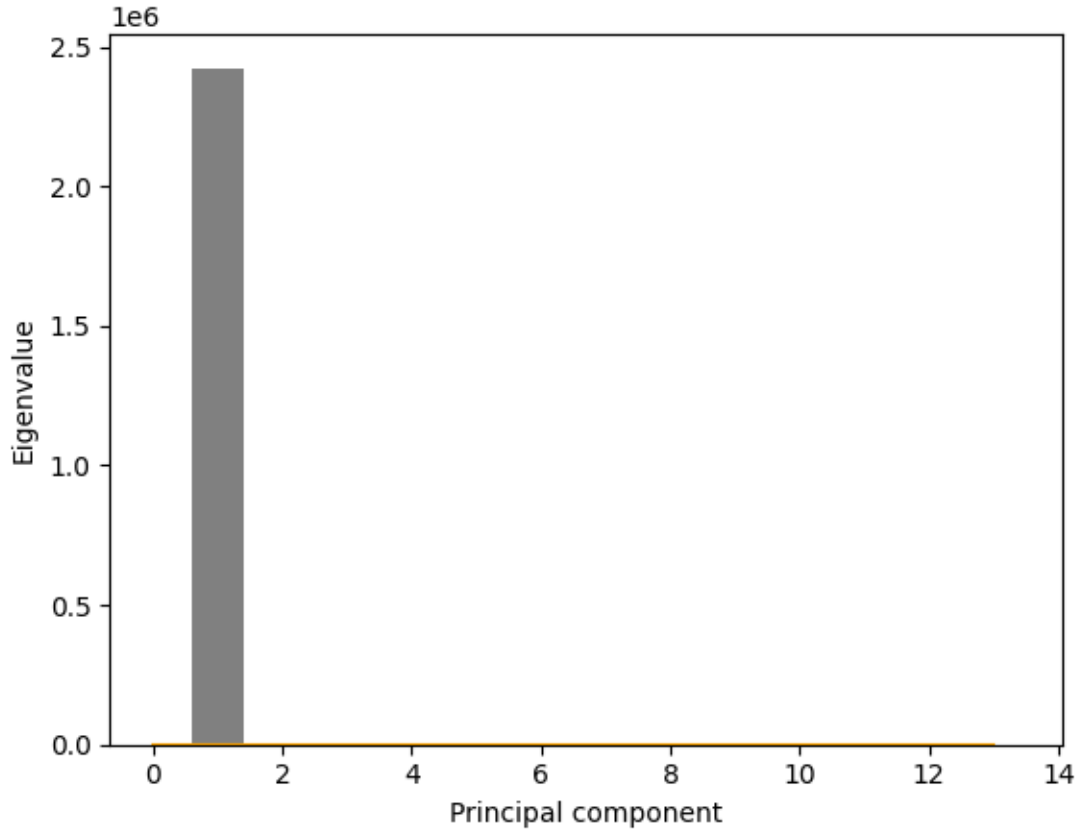


Surprisingly, PCA performs really well and it is way much faster than t-SNE in this large dataset. However, the clusters are still not representative for genres.

```
pca = PCA()
pca.fit(X_train)
eigenvalues = pca.explained_variance_
eigenvectors = pca.components_
n_eigenvalues_above_1 = np.sum(eigenvalues > 1)
print(f'Number of eigenvalues above 1: {n_eigenvalues_above_1}')
```

```
Number of eigenvalues above 1: 4
```

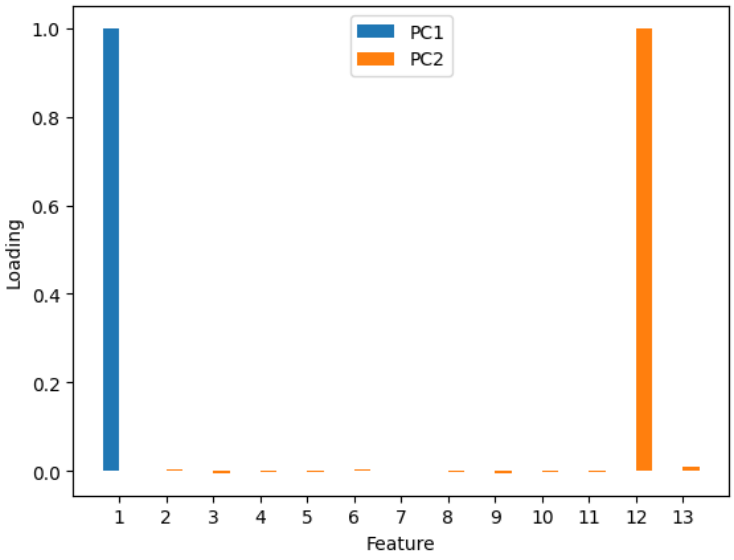
There are 4 eigenvalues above 1.




```
] variance_explained = np.sum(eigenvalues[:1]) / np.sum(eigenvalues) * 100
  print(f'Percentage of variance explained by first dimensions: {variance_explained:.2f}%')
```

```
Percentage of variance explained by first dimensions: 100.00%
```

But the PC1 is too large that it explains almost 100% of the variance. I plot the loading to see what features does the PC1 relies on:



It turns out that PC1 is highly relies on artist name. Perhaps one-hot encoding is a better approach for encoding the artist name here.

Let's evaluate its performance on different models and compare with naïve model:

```
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, clf_tree.predict_proba(X_pca_test[:,:1]), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 74.28%
AUC Score: 0.92
```

```
clf_tree = DecisionTreeClassifier(criterion="entropy",  random_state=42)
clf_tree.fit(X_train, y_train)
pred = clf_tree.predict(X_test)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, clf_tree.predict_proba(X_test), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 32.70%
AUC Score: 0.63
```

```
[ ]  neigh = KNeighborsClassifier(n_neighbors=3)
     neigh.fit(data_pca[:,:1], y_train)
     pred = neigh.predict(X_pca_test[:,:1])
     accuracy = accuracy_score(y_test, pred)
     auc_score = roc_auc_score(y_test, neigh.predict_proba(X_pca_test[:,:1]), multi_class='ovr')

     print("Accuracy: {:.2f}%".format(accuracy * 100))
     print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 67.26%
AUC Score: 0.90
```

```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)
pred = neigh.predict(X_test)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, neigh.predict_proba(X_test), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 55.82%
AUC Score: 0.84
```

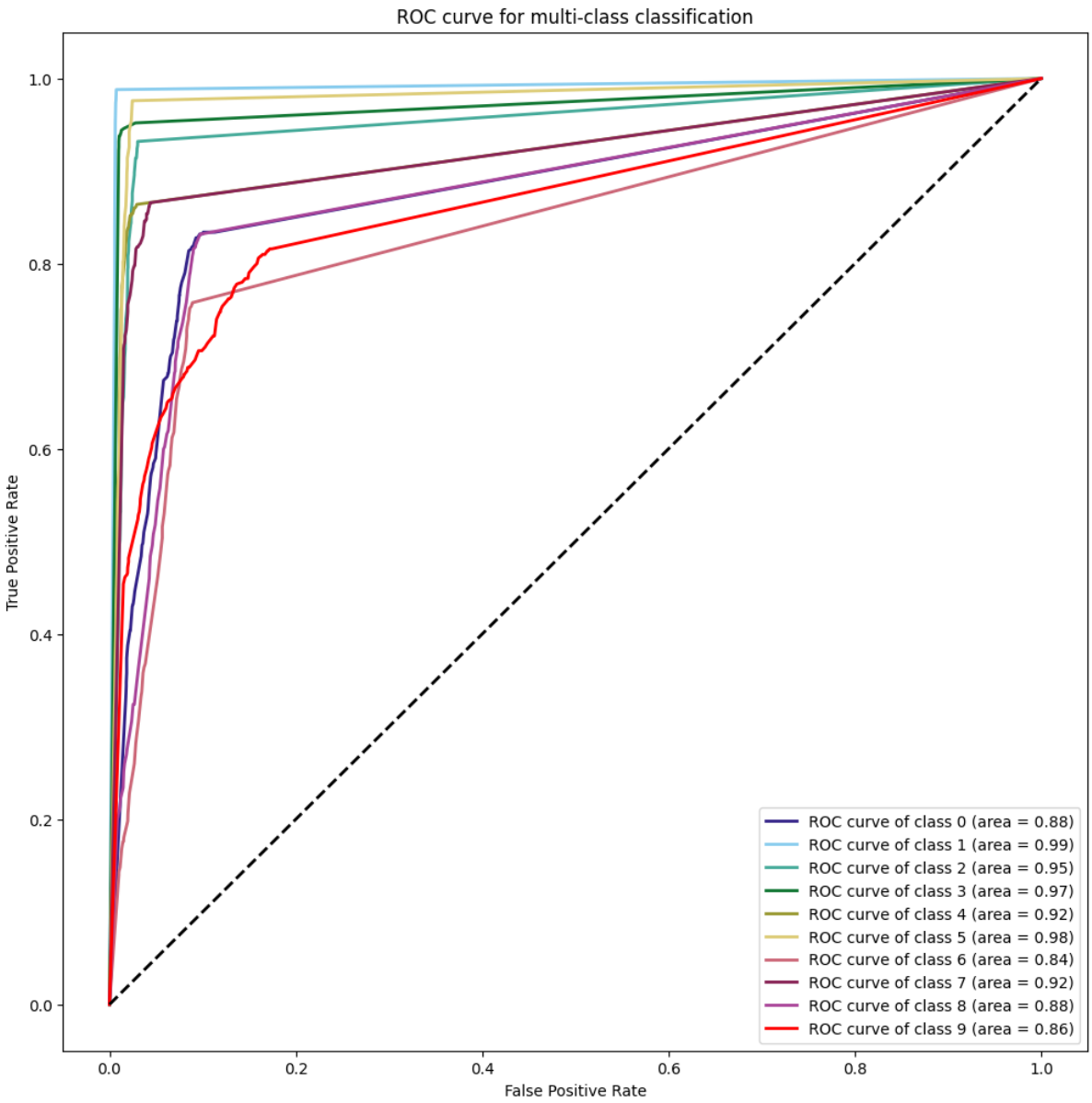
```
[ ] clf = LogisticRegression(random_state=10).fit(X_pca[:,1:], y_train)
    pred = clf.predict(X_pca_test[:,1:])
    accuracy = accuracy_score(y_test, pred)
    auc_score = roc_auc_score(y_test, clf.predict_proba(X_pca_test[:,1:]), multi_class='ovr')

    print("Accuracy: {:.2f}%".format(accuracy * 100))
    print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 38.36%
AUC Score: 0.83
```

```
clf = LogisticRegression(random_state=0).fit(X_train, y_train)
pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, pred)
auc_score = roc_auc_score(y_test, clf.predict_proba(X_test), multi_class='ovr')

print("Accuracy: {:.2f}%".format(accuracy * 100))
print("AUC Score: {:.2f}".format(auc_score))
```

```
Accuracy: 29.14%
AUC Score: 0.77
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning:
```

It performs really well in decision tree, KNN, but not Logistic Regression. I plot the ROC graph to evaluate decision tree model performance on different class in OVR(one vs rest) condition.



Since the decision tree model works pretty well, it also makes sense to try Adaboost and Random Forest.:

```
n_classes = len(np.unique(y_train))
n_estimators = 50  # number of decision trees to use
learning_rate = 1.0  # learning rate of the AdaBoost model

# train an AdaBoost model using decision tree as the base estimator
base_estimator = DecisionTreeClassifier(criterion="entropy",  random_state=42)
ada_boost = AdaBoostClassifier(base_estimator=base_estimator, n_estimators=n_estimators, learning_rate=learning_rat
ada_boost.fit(data_pca[:,:1], y_train)

# predict the class labels and probabilities for the test data
y_pred = ada_boost.predict(X_pca_test[:,:1])
y_pred_proba = ada_boost.predict_proba(X_pca_test[:,:1])

# evaluate the model using accuracy score and ROC AUC score
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred_proba, multi_class='ovr')
print('Accuracy:', accuracy)
print('ROC AUC score:', roc_auc)
```

```
Accuracy: 0.7428
ROC AUC score: 0.9110642444444444
```

```
rf = RandomForestClassifier(n_estimators=1000)
rf.fit(data_pca[:,:1], y_train)

# Predict on the test set
y_pred = rf.predict(X_pca_test[:,:1])

# Evaluate accuracy and ROC AUC
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, rf.predict_proba(X_pca_test[:,:1]), multi_class='ovr')

print('Accuracy:', accuracy)
print('ROC AUC:', roc_auc)
```

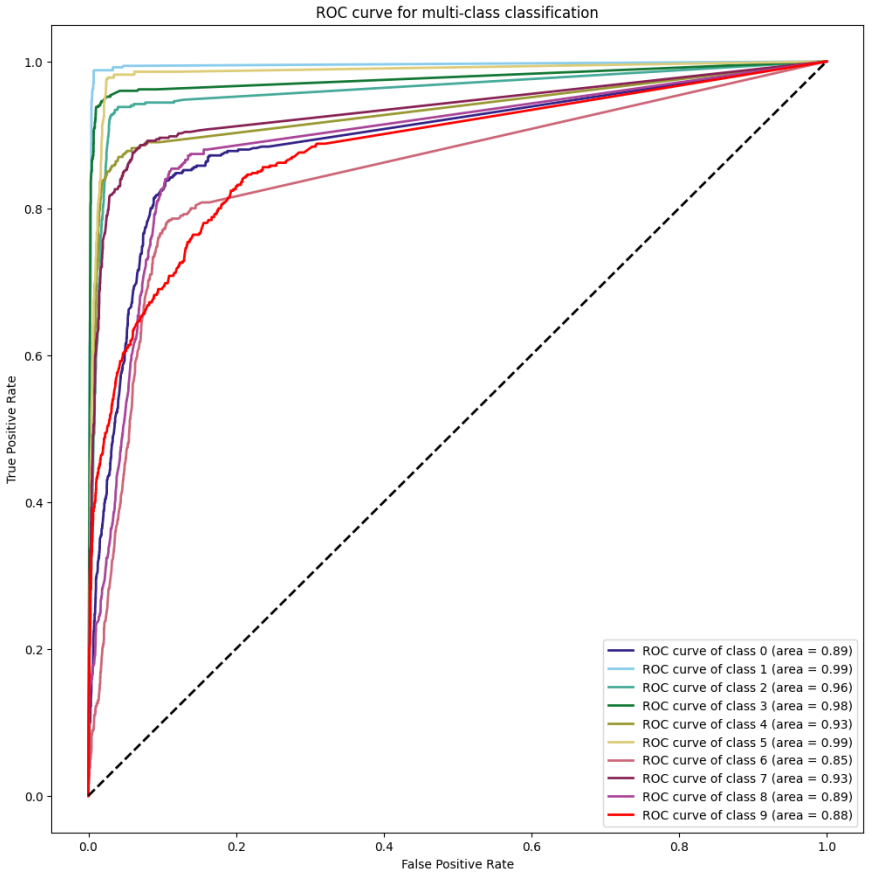
```
Accuracy: 0.7406
ROC AUC: 0.9300884222222221
```

As shown in the images, the random forest performs better than Adaboost, so I run a grid-search to optimize the hyperparameter and too further boost its performance.

```
grid_search = GridSearchCV(rf, param_grid=param_grid, scoring=scorer, n_jobs=-1)

# Fit the grid search object to the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and the corresponding model
best_params = grid_search.best_params_
best_rf = grid_search.best_estimator_

# Predict on the test set using the best model
y_pred = best_rf.predict(X_test)

# Evaluate accuracy and ROC AUC
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, best_rf.predict_proba(X_test), multi_class='ovr')

print('Best parameters:', best_params)
print('Accuracy:', accuracy)
print('ROC AUC:', roc_auc)
```

```
Best parameters: {'max_depth': 5, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Accuracy: 0.3548
```

In the end, we got a model with following ROC graph.



In conclusion, this study select 13 features (including artist name) from the dataset and run dimension reduction through PCA. Then train a random forest model with 1000 estimator and evaluate model performance on the test set. In the end, we got a model with 74% accuracy and 0.93 roc auc score on average.

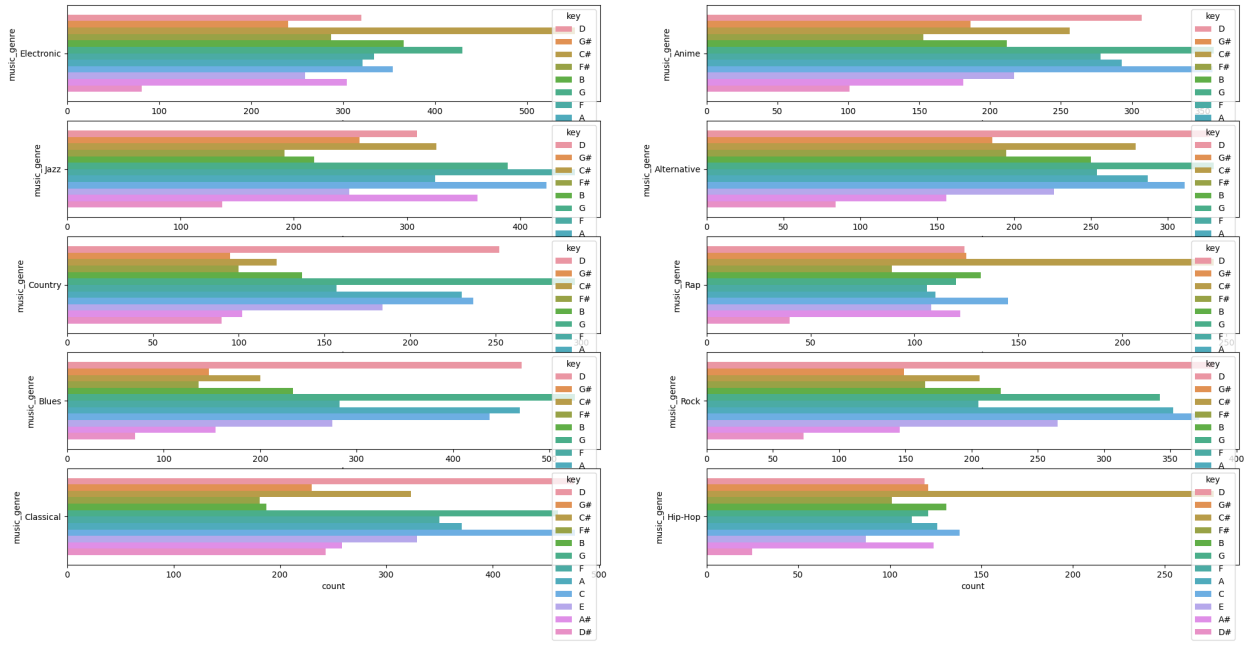
Something to stay cautious about here, this is model is very likely to not robust when facing new artists because it is highly rely on artist already appeared in the training set. In another words, although the model does not overfit the training set, it is overfitting the entire sample. It would not perform as well as this study when confronting strange artist. The artist name is the most important factor that underlies this classification success.

Additional info: model trained without artist name:

```
rf = RandomForestClassifier(max_depth=30, min_samples_leaf= 1, min_samples_split= 2, n_estimators=1000)
rf.fit(X_pca[:,1:], y_train)

# Predict on the test set
y_pred = rf.predict(X_pca_test[:,1:])

# Evaluate accuracy and ROC AUC
accuracy = accuracy_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, rf.predict_proba(X_pca_test[:,1:]), multi_class='ovr')

print('Accuracy:', accuracy)
print('ROC AUC:', roc_auc)
```

```
Accuracy: 0.3936
ROC AUC: 0.8369783777777778
```

Extra credit:



D# is the least popular key among all keys!