Big Three Project

Background

As a lover of both K-pop and data, I often wonder about the relationship between them and so for this project I decided to explore one aspect of that relationship using the Spotify API. K-pop artists are all under companies who usually have major creative control and frequently work with in-house or the same freelance producers, writers, choreographers, and marketers. Therefore certain K-pop companies have brand or vibe that is apparent in their artists and K-pop fans recongize. This is especially true in the case of the K-pop companies known as “the big three”: SM Entertainment, JYP Entertainment, and YG Entertainment. Long-time K-pop fans frequently claim to be able to identify which company a song is from with just a single listen. Therefore, I wanted to see if a computer could identify which of the big three companies a song came from using the Spotify features of that song.

Dataset

Ideally, I would have used every song released by all the artists under each company but for the sake of time and simplicity, I decide to just use a few artists from each. I wanted the groups I used to have a similar context to each other so I chose groups from the same generation and gender. I used the full discography of each artist, excluding remixes or the same songs in different languages (usually Japanese or English versions of Korean title tracks). I did not include any songs made by the artists after they left the company if they left but I did include solo songs released by members of my chosen groups as long as the soloist was under the company.

The groups I chose:

Stratifying factor	JYP	SM	YG
2nd Gen GGroup	Wonder Girls	Girls Generation	2NE1
2nd Gen BGroup			BIGBANG
3rd Gen GGroup	TWICE	Red Velvet	BLACKPINK
3rd Gen BGroup	GOT7	EXO	iKON
4th Gen BGroup	Stray Kids	WAYV	TREASURE

Initially, I did not have BIGBANG but my YG dataset was so much smaller than the other two that I added them for numerical balance although it messed up my stratification a bit. Also as a bit of a side project, I made another dataset that included every artist under all three companies and had various statistics for them like youtube followers and award show wins (https://github.com/mclainbrown/big3_project/blob/main/Datasets/big_3_artists.csv).

Data Collection

The first step in collecting the datasets was to make myself a Spotify for developers profile which allows you to read data from Spotify using Spotipy which is a python library of functions for interacting with Spotify. Next, I made playlists of each artist's full discography and manually excluded remixes and versions of the same songs in other languages. I had to split the playlists into multiple parts for a few artists as the playlist function would only fetch features for 100 songs at a time. The code for getting Spotify features from a playlist is in my code folder and is called playlist.py[1]. I then uploaded the files to my github and concatenated them, first by artist

[1] You might note that there is a line near the bottom of that code which creates a column called company and I would just manually change which company was listed when I changed artists, so make sure to remove that line if you want to use that code for a project of your own

and then by company. All the concatenated and unconcatenated datasets can be found in the Datasets folder. For my non-spotify dataset, I collected information across various platforms on all the artists and then manually entered it into a spreadsheet.

Data Exploration

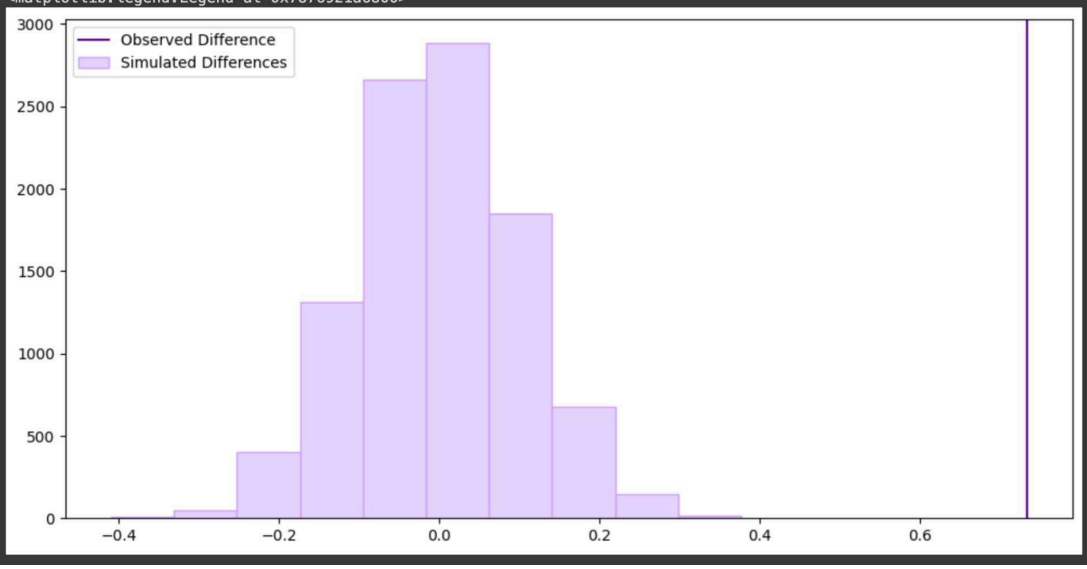
Once I had all my data, I made a few exploratory models to see what my dataset looked like. I made bar charts comparing the average score on each feature in all three companies and those graphs can be found in my graphs folder. I also conducted a permutation on loudness, which was one of the features where the companies differed the most to see if the difference was real or just due to chance.

```
[ ]  #observed test stat
     obsv_stat = np.average(JYP_df['loudness'])- np.average(YG_df['loudness'])
     print(obsv_stat)

     0.7335407134809717

     JYP_YG_df = JYP_df.append(YG_df)
     loudness = np.array(JYP_YG_df["loudness"].values)
     company = ['YG', 'JYP']
     sim_comp = np.random.choice(company, len(loudness))

[ ]  def compute_diff_once (company, loudness):
       JYP_loudness = loudness[company == 'JYP']
       YG_loudness = loudness[company == 'YG']
       difference = np.average(JYP_loudness) - np.average(YG_loudness)
       return difference

[ ]  num_sim = 10000
     test_stats = []
     for i in range(num_sim):
       sim_comp = np.random.choice(company, len(loudness))
       one_stat = compute_diff_once(sim_comp, loudness)
       test_stats.append(one_stat)
```

My permutation was significant as seen below:

```
[ ]  plt.figure(figsize = (12, 6))
     plt.hist(test_stats, color = '#E5D4FF', edgecolor = '#D0A2F7')
     plt.axvline(obsv_stat, color = '#5B0888')
     plt.legend(('Observed Difference', "Simulated Differences"))
```

```
<matplotlib.legend.Legend at 0x7876921a6800>
```



Classifier Models

First I imported my libraries and set my target

```
[4]  from sklearn.metrics import *
     from sklearn.model_selection import train_test_split, KFold
     from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
     from sklearn.model_selection import KFold, cross_val_score
     from sklearn.preprocessing import LabelEncoder
     import lightgbm as lgb
```

```
[5]  import imblearn
     from imblearn.over_sampling import SMOTE
```

```
[52] kpop_df = pd.concat([YG_df, SM_df, JYP_df])
     kpop_df = kpop_df.drop('Unnamed: 0', axis=1)
```

```
[53] #setting target
     kpop_df['target'] = [1 if i == 'YG' else 0 for i in kpop_df.company]
```

```
[ ]  kpop_df.head()
```

Next, I used smote while creating my training and validation to prevent overrepresentation

```
#dropped all qualitative variables
kpop_df = kpop_df.drop(['id','artist','title','company','date_released'], axis = 1)
```

```
[55] X_train, X_validation, y_train, y_validation = train_test_split(kpop_df.iloc[:,:-1], kpop_df.target, train_size=0.2)

     X_train_lgbm = pd.get_dummies(X_train)
     X_validation_lgbm = pd.get_dummies(X_validation)

     X_train_lgbm.reset_index(inplace = True)
     X_validation_lgbm.reset_index(inplace = True)
```

```
[56]
     sm = SMOTE()

     X_train_smote, y_train_smote = sm.fit_resample(X_train_lgbm, y_train.ravel())
     X_validation_smote, y_validation_smote = sm.fit_resample(X_validation_lgbm, y_validation.ravel())


     X_train_smote = pd.DataFrame(X_train_smote, columns = X_train_lgbm.columns).set_index('index')
     X_validation_smote = pd.DataFrame(X_validation_smote, columns = X_validation_lgbm.columns).set_index('index')
```

Then I built three different classifier models (LGBM, Gradient Boosting, and Random Forest)

```
lgb_c = lgb.LGBMClassifier(n_estimators=10000, objective='binary',
                           class_weight='balanced', learning_rate=0.01,
                           reg_alpha=0.1, reg_lambda=0.1, silent=True,
                           subsample=0.8, n_jobs=-1, random_state=50)

lgb_c.fit(X_train_smote, y_train_smote,
          eval_set=[(X_validation_smote, y_validation_smote)],
          eval_metric='auc',)
```

```
[26] gbc = GradientBoostingClassifier(n_estimators=1000, learning_rate=0.01)
     gbc.fit(X_train_smote, y_train_smote)

     validation_pred_proba_grad = gbc.predict_proba(X_validation_smote)
     roc_auc_score(y_validation_smote, validation_pred_proba_grad[:,1])
```

```
     0.6634645352185294
```

```
[27] rf = RandomForestClassifier()
     rf.fit(X_train_smote,y_train_smote)

     validation_pred_proba_grad = rf.predict_proba(X_validation_smote)
     roc_auc_score(y_validation_smote, validation_pred_proba_grad[:,1])
```

```
     0.6928390678847592
```

I did some K-fold cross-validation with my Random Forest model

```
[28] x = kpop_df.drop(['target'], axis = 1)
     y = kpop_df['target']
```

```
[29] #cross-validation
     k_folds = KFold(n_splits = 10, shuffle = True)

     scores = cross_val_score(rf, x, y, cv = k_folds)
     scores
```

```
array([0.7283237 , 0.68208092, 0.68208092, 0.65317919, 0.71098266,
       0.68786127, 0.69364162, 0.64739884, 0.65895954, 0.65697674])
```

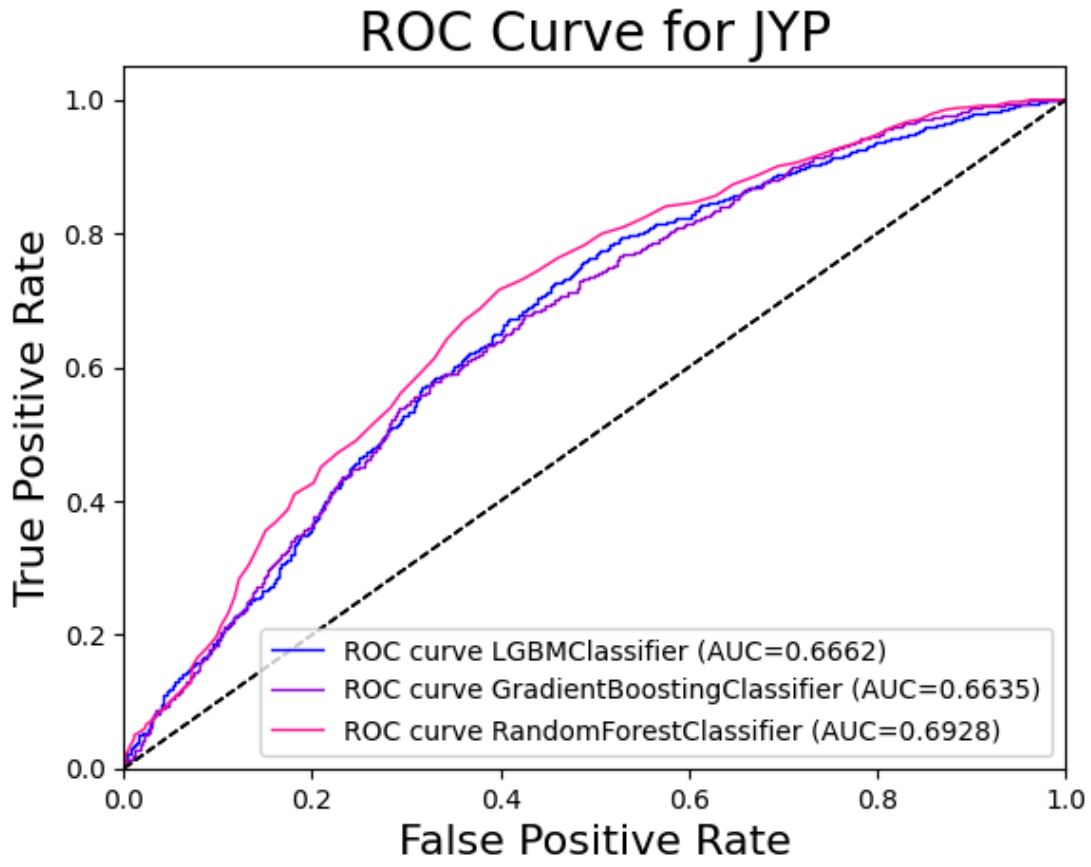
```
[30] #average score with cross-validation
     np.average(scores)
```

```
0.6801485414706278
```

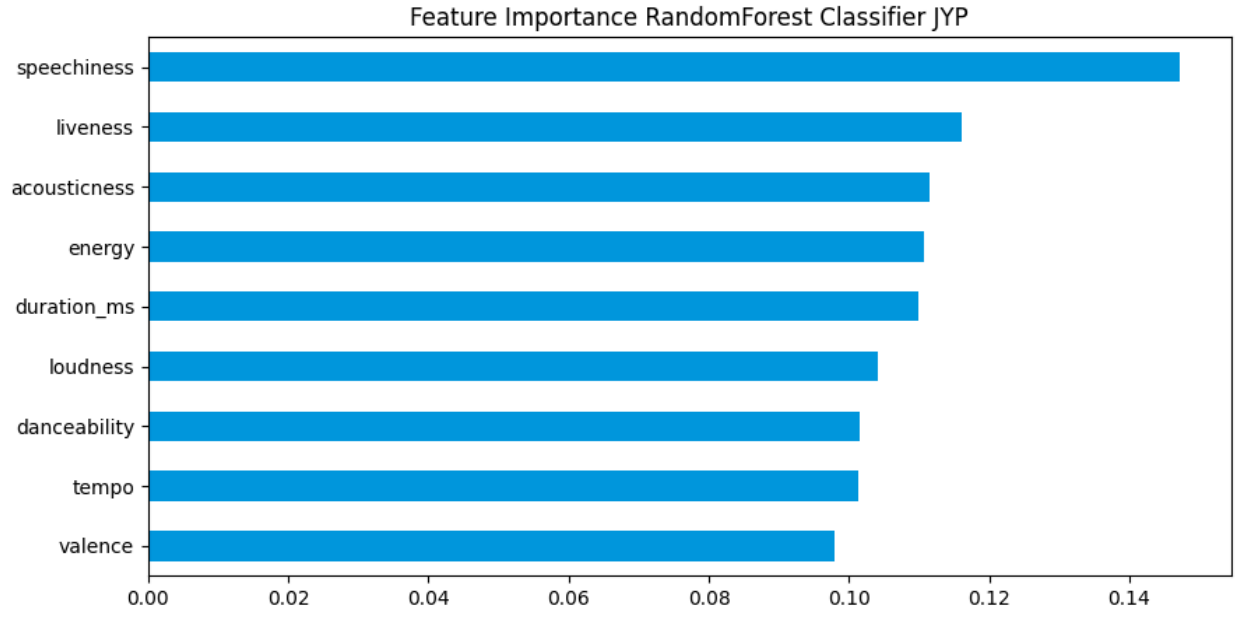
Finally I made an ROC graph using my three models

```
#dropped mode, instrumentalness, and key by using backwards selection
fig1 = plt.gcf()
def plot_roc(model, X_validation, y_validation,  color):
    validation_pred_proba = model.predict_proba(X_validation)
    fpr, tpr, thresholds = roc_curve(y_validation, validation_pred_proba[:,1])
    auc = roc_auc_score(y_validation, validation_pred_proba[:,1])

    plt.plot(fpr, tpr, color=color,lw=1, label='ROC curve {} (AUC={:0.4f})'.format(model.__class__.__name__,auc))
    plt.plot([0, 1], [0, 1], color='black', lw=1, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate', fontsize = '16')
    plt.ylabel('True Positive Rate',fontsize = '16')
    plt.title('ROC Curve for JYP', fontsize = '20')
    plt.legend(loc="lower right");

plot_roc(lgb_c, X_validation_smote, y_validation_smote, 'b')
plot_roc(gbc, X_validation_smote, y_validation_smote, 'darkviolet')
plot_roc(rf, X_validation_smote, y_validation_smote, 'deeppink')
plt.figure(figsize= (15,20))
```

My ROC curve showed that all of my models were able to predict JYP more than 50% of the time which is a success!

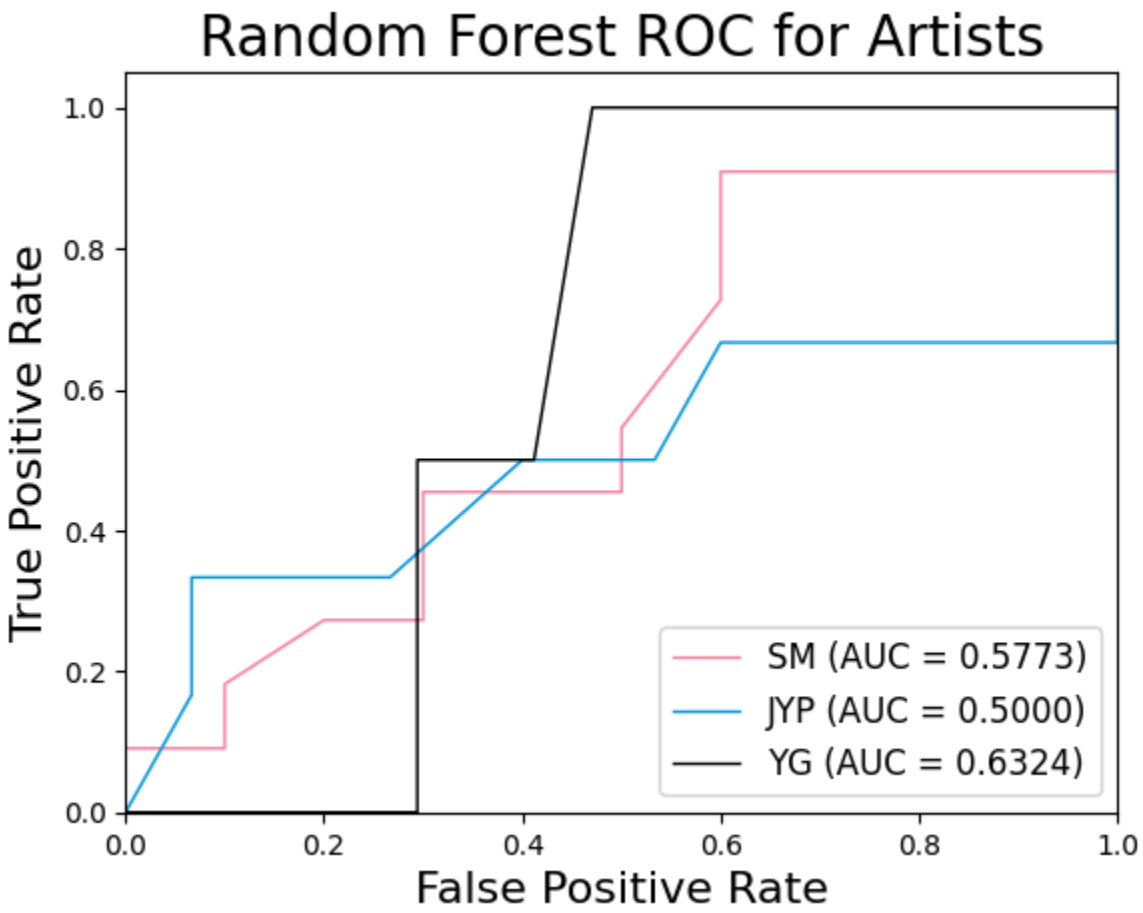


Lastly, I did feature importance graphs for all three models



The ROC and feature importance graphs for the other companies can be found in my graphs folder but all of my models did have an AUC above 50% so they were somewhat able to identify which company a song belonged to just by its Spotify features.

I also did a classifier for my other dataset but since it had much fewer points, it was not a great curve:



As shown in the ROC above, I was not really successful at predicting which company an artist belonged to based on their achievements which was not that surprising to me as each artist differs a good deal and there were not that many which makes classification very difficult. I had originally wanted to do classifier models with different aspects of the songs like frequency maps but I discovered that there was not really a fast/efficient method to gather those and it was maybe illegal since I have no rights to these songs so the best I could do was the artist statistics.

3D Model

Just for fun I made a 3D Model with my artist dataset which does not show particularly well thorough screenshot but was very cool and I would recommend

```
import plotly.express as px
fig = px.scatter_3d(df, x='music_show_awards', y='eoy_awards', z='physical_sales',
              color='company', symbol='gender', color_discrete_sequence=['#ffd3d9', 'black', '#0096df'], hover_name = 'artist',
              opacity=0.8)

fig.update_layout(margin=dict(l=0, r=0, b=0, t=0))
fig.show()
```