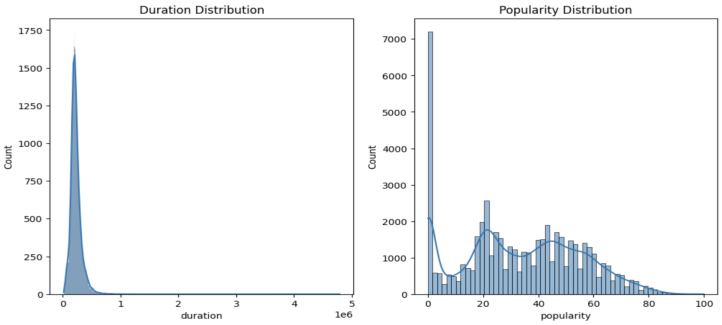
DS-GA 1001 Capstone Project

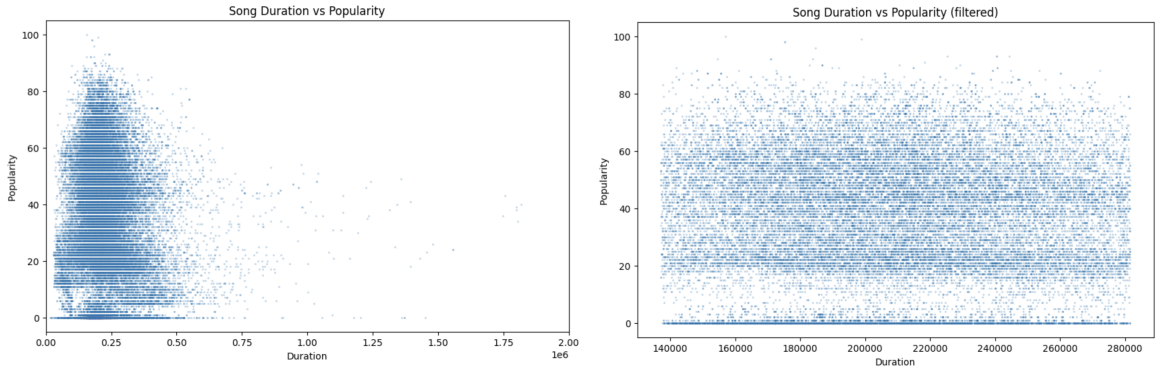
Yuanxin Zhang & Chenxin Gu

1) Is there a relationship between song length and popularity of a song? If so, is it positive or negative?



Since neither the duration distribution nor the popularity distribution is normally distributed, direct tests like the Pearson correlation coefficient will not work well here. We instead did **a linear regression** and got MSE 470.05 and R^2 0.0031. This result suggests that the duration of a song and popularity are extremely weakly related.

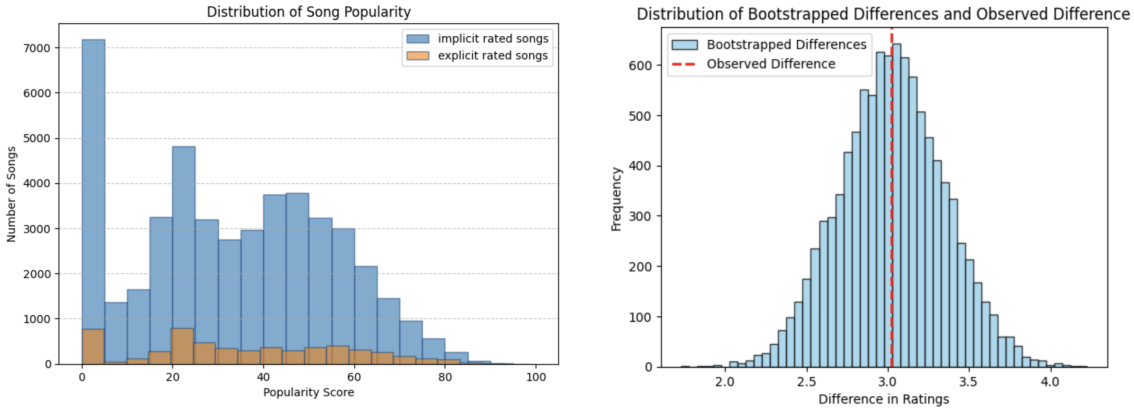
Since the duration distribution is highly skewed, we suspected that the low correlation is caused by the outliers. Thus, we repeated regression after filtering data only between the 10th and 80th quantile to decrease the influence of outliers. However, MSE is 494.57 and R^2 is: -1.04e-05, suggesting an even lower possibility that there exists any correlation between the two variables. Thus, we conclude that **there isn't a linear relationship between song length and popularity of a song**. We can also verify this conclusion from the below visualization.



2) Are explicitly rated songs more popular than songs that are not explicit?

We choose to conduct a hypothesis test to determine if explicitly rated songs are more popular than those that are not explicitly rated.

- **H0:** Explicitly rated songs are as popular as those are not explicitly rated.
- **H1:** Explicitly rated songs are more popular than those are not explicitly rated.



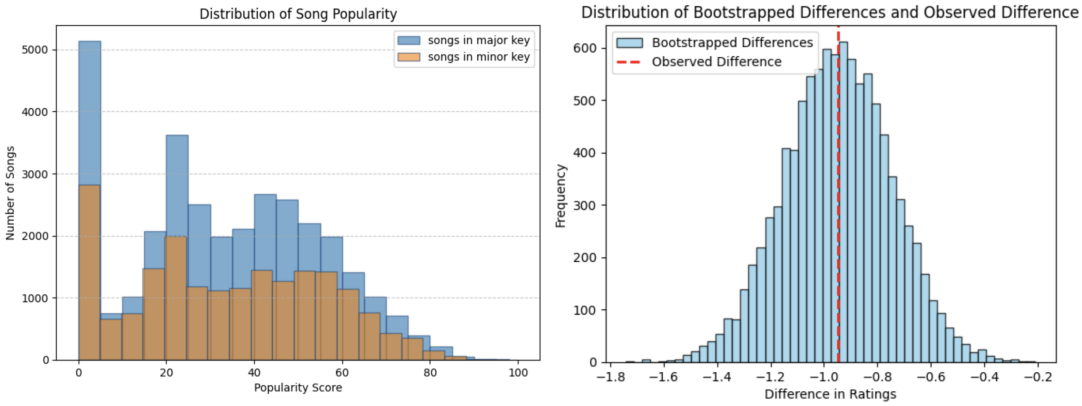
Since the two samples differ too much in size (5597 vs. 46403) and are not distributed normally, **we choose to use bootstrap resampling with differences in mean as test statistics**. Our observed difference in means is 3.0225 and the P-value for this difference is 0.5037, which is not statistically significant at the 0.005 level. Thus, we conclude that **explicitly rated songs are not more popular than those that are not explicitly rated**.

3) Are songs in major key more popular than songs in minor key?

Similar to question 2), here are the null hypothesis and alternative hypothesis for our hypothesis test.

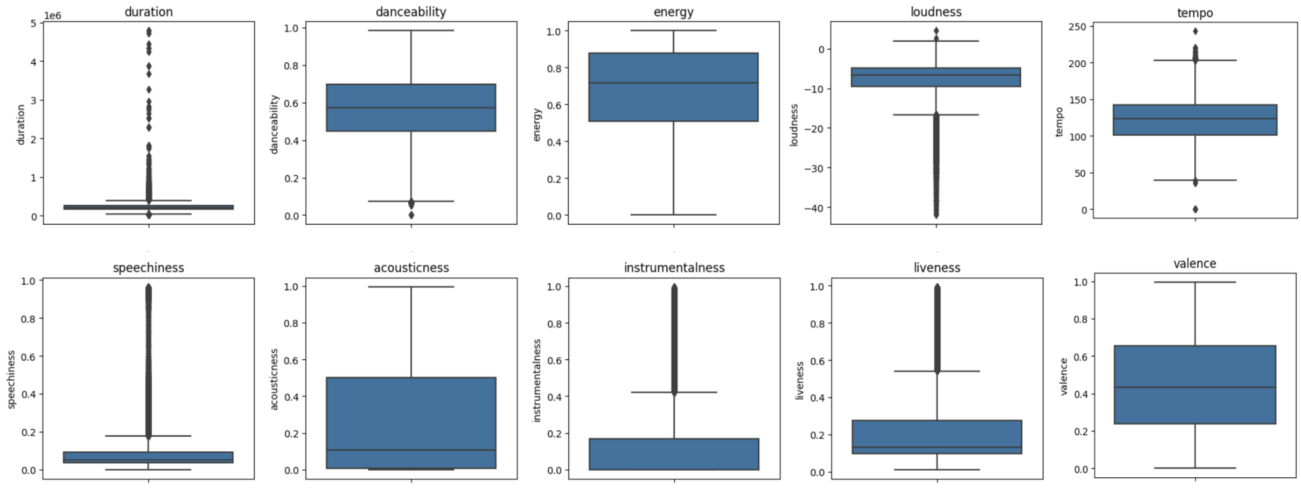
- **H0:** Songs in major key are as popular as songs in minor key.
- **H1:** Songs in major key are more popular than songs in minor key.

Again notice the high peak around zero is also valid data in this dataset and the two sample sizes are quite different, **bootstrap resampling method is still the best choice in this case**. We used difference in means as our test statistics, and got observed difference in means to be -0.9481, P-value for this difference 0.4985. The observed difference is not statistically significant at the 0.005 level, therefore we conclude that **songs in major key are not necessarily more popular than songs in minor key**.

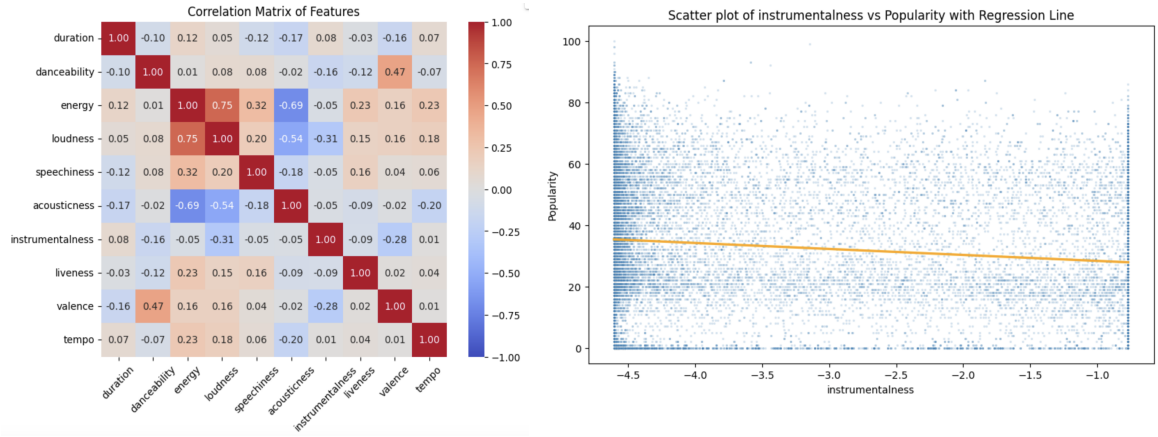


4) Which of the following 10 song features: duration, danceability, energy, loudness, speechiness, acousticness, instrumentalness, liveness, valence, and tempo predicts popularity best? How good is this model?

To determine which feature predicts popularity best, we consider doing **simple linear regression** for each feature concerning popularity, and **compare model behavior with COD evaluation**.



Since we already discovered that some features in this dataset have extremely skewed distribution in previous questions, we are motivated to check the distribution for all 10 targeted features. As shown above, half of the features have numerous outliers. We tested on parameters and chose to use data between 10th and 80th quantile for model building. Since 'instrumentalness' is still right-skewed, we further applied log transformation to normalize it.



Before we start model fitting, we checked multicollinearity between the 10 features so that there doesn't exist an obvious problem with multicollinearity. We fit the model using linear regression from the `sklearn` package and calculated COD for each model. '**Instrumentalness**', with a COD of 0.0206 higher than all other features, **is the best predictor** for popularity among the ten features. However, since this COD score is still very low. This means that **none of these ten features alone works well in terms of predicting popularity**.

*5) Building a model that uses *all* of the song features mentioned in question 4, how well can you predict popularity? How much (if at all) is this model improved compared to the model in question 4). How do you account for this? What happens if you regularize your model?*

Namely, we build a **multivariate linear regression** using all ten features mentioned in the previous question as X. Before we define our regression model, we normalized all ten features using `StandardScaler` from `sklearn.preprocessing` to unify the measure. Then, we defined multivariate regression model using the following code:

```
def build_multi_regression_model(song_features_10):
        # Splitting the dataset
        X = song_features_10.iloc[:, :-1].values
        y = song_features_10['popularity'].values

        # Building the model
        model = LinearRegression().fit(X, y)
        predictions = model.predict(X)
        R2 = r2_score(y, predictions)
        return R2
R2 = build_multi_regression_model(song_features_10)
```

The R^2 value generated is 0.0437, a little higher than the highest COD from our previous simple linear regression models, shows that **this multivariate model predicts popularity a bit better than using only one of the ten traits**. However, since 0.0437 is still a very small number for R^2 value, **the model is still almost unable to predict popularity well**.

Then, since we have already performed collinearity analysis before and concluded weak collinearity between any two variables, we chose to use ridge regression as a regularized model since it does not perform feature selection. For model building, we used a similar structure as before, but switched to `ridge_model = Ridge(alpha=1.0)` instead. The R^2 value of ridge regression is 0.0473, which is almost the same as the R^2 value from our multivariate regression model, suggesting that **these two models perform equally weakly on predicting popularity of songs**.

6) When considering the 10 song features in the previous question, how many meaningful principal components can you extract? What proportion of the variance do these principal components account for? Using these principal components, how many clusters can you identify? Do these clusters reasonably correspond to the genre labels in column 20 of the data?

We use eigenvalues with the k>1 method to find the meaningful principal components. By running this algorithm, we get **3** meaningful principal components among these 10 features.

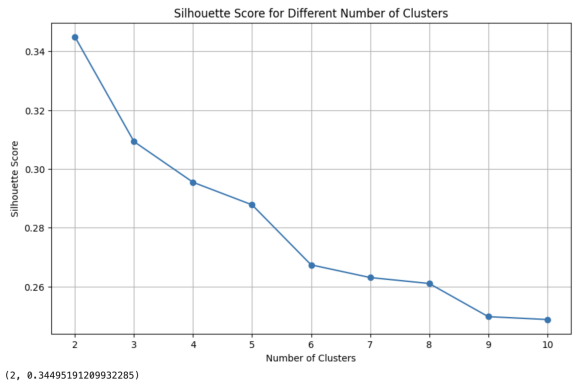
From the examplained_variance, we see that

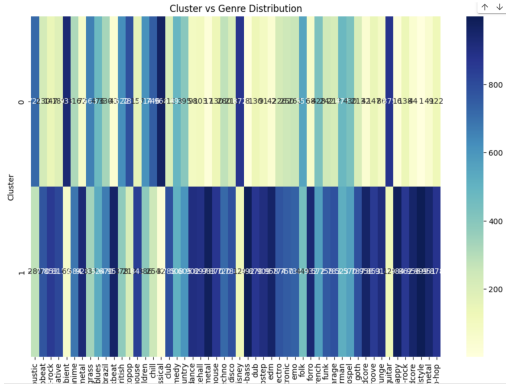
- The first principal component explains approximately 26.6% of the variance.
- The second principal component explains about 17.1% of the variance.
- The third component accounts for roughly 11.5% of the variance.

Cumulatively, the variance explained by the principal components is as follows:

The first three components together explain about 55.25% of the total variance. The first four components explain approximately 65.1% of the variance. To capture more than 75% of the variance, you would need to consider the first six components. For over 90% of the variance explanation, at least the first seven components should be included.

By both the elbow method and the Silhouette score, we identified **2** clusters among these features.



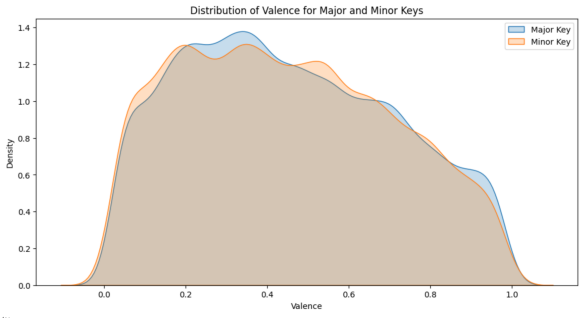


The left graph shows the Silhouette score.

In terms of the relationship between clusters the genre labels, we concluded that **for genres' categories, if it shows deep blue or shallow yellow, that means a good clustering between two labels**.

7) Can you predict whether a song is in a major or minor key from valence using logistic regression or a support vector machine? If so, how good is this prediction? If not, is there a better one?

First, we visualize the valence distribution for major and minor keys, the graph shows below.



The graph here shows that the data are balanced in terms of the target variables. So we don't need to do resampling.
Now we use SVM and logistic regression to see which algorithm predicts better. And we see **both of them don't perform well, since the roc score is around 50%**. However, we tried the decision tree model here and found a slightly better model roc score of 53%. So we conclude that **the decision tree is a better choice for this question**.

8) Can you predict genre by using the 10 song features from question 4 directly or the principal components you extracted in question 6 with a neural network? How well does this work?

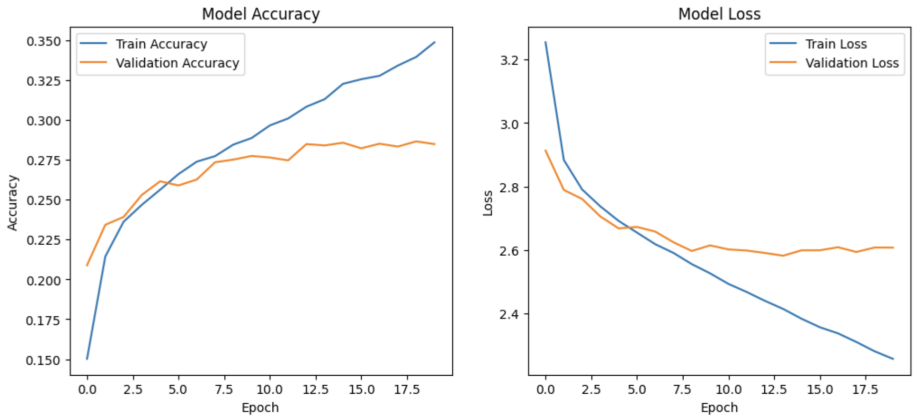
We chose to include all 10 features to predict genre in order to get a more precise result.

- **Data preprocessing:** We used two functions from sklearn.preprocessing, MinMaxScaler and labelencoder for preprocessing. Since the features are scaled differently, MinMaxScaler helps us normalize all data to [0, 1] such that each feature contributes proportionately to the model training and improves model convergence. We transformed the 52 labels in track_genre from text into numerical format, and then into a categorical format for classification algorithms to process.

```
scaler = MinMaxScaler()
song_features_10[features.columns] = scaler.fit_transform(features)


label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
y_categorical = to_categorical(y_encoded)
```

- **Model selection: LSTM** outperformed 1D CNN, and RNN on this task. Through experiments, we discovered that the LSTM, with its capability to remember information for long periods, outperformed the RNN, making it our choice for the final model.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.optimizers import Adam

model = Sequential([
        Bidirectional(LSTM(128, return_sequences=True, activation='tanh'),
        input_shape=(X_train.shape[1], 1)),
        Dropout(0.2),
        Bidirectional(LSTM(64, activation='tanh')),
        Dropout(0.2),
        Dense(y_categorical.shape[1], activation='softmax')
])

optimizer_adam = Adam(learning_rate=0.005)
model.compile(optimizer=optimizer_adam, loss='categorical_crossentropy',
metrics=['accuracy'])
```



- **Model Architecture and Hyperparameters Tuning:** we experimented with step length in LSTM to find the balance between learning efficiency and model accuracy. As shown in the above plot, overfitting becomes a problem in the second half of the training process, so we added a 20% dropout rate to reduce overfitting and enhance model generalization. Training for 20 epochs is quite enough for our case because validation accuracy becomes steady after 10 epochs.
- **Training:** Through 20 epochs of training, our model increased from 15.02% accuracy initially to 34.86% accuracy on the training set. On the validation set, accuracy improved from 20.89% to 28.49%, reflecting the model's gradually enhanced ability to recognize music genres.
- **Testing and conclusion: The model achieves a final accuracy of 29.43% on the testing set, demonstrating its ability when handling unseen data. Although the final accuracy is still moderately low, we can tell that the model is indeed working given the number of labels in our task**. Also, given the fact that distinguishing features across genres can be overlapping, an accuracy of around 30%, though still has potential room for improvement, meets our expectations for the task.

9) In recommender systems, the popularity-based model is an important baseline. We have a two-part question in this regard: a) Is there a relationship between popularity and average star rating for the 5k songs we have explicit feedback for? b) Which 10 songs are in the "greatest hits" (out of the 5k songs), based on the popularity-based model?

Here we use another dataset called "StarRating.csv". With 10000 user ratings regarding the first 5000 songs among the Spotify song data, we first try to do data preprocessing on the star rating document.

- **Data Preprocessing:** we filter all rows with 5 or more ratings to make sure the ratings are fair. And then update the analysis_data DataFrame to include only songs with at least 5 ratings. The new data frame has the music names and average ratings.
- **Statistical analysis:** we perform a correlation analysis between popularity and average star rating. The correlation between a song's popularity and its average star rating is approximately 0.57. This indicates a moderate positive relationship. In other words, there tends to be a tendency for songs with higher popularity scores to also have higher average star ratings, although this relationship is not extremely strong.

	songNumber	artists	track_name	popularity
2003	2003	The Neighbourhood	Sweater Weather	93
3003	3003	The Neighbourhood	Sweater Weather	93
2000	2000	The Neighbourhood	Daddy Issues	87
3000	3000	The Neighbourhood	Daddy Issues	87
3300	3300	Oliver Tree;Robin Schulz	Miss You	87
2002	2002	The Neighbourhood	Softcore	86
2106	2106	The Killers	Mr. Brightside	86
3002	3002	The Neighbourhood	Softcore	86
3004	3004	GAYLE	abcdefu	86
3257	3257	The Killers	Mr. Brightside	86

- **Top 10 "greatest hits":** Based on the popularity-based model, we use the metric "popularity" as a measurement. Here we get the top 10 greatest hits music without personalization:

You can see there are some repeated songs shown above, but they are not duplicates (we

test the duplication at the very beginning). That means they are different versions of the song. So we don't do any operation here.

Finally, we calculate the mean average precision score to compare this method with the ensuing ones on the recommendation precision. The popularity-based model has a map around 0.002, which is a bad recommendation model for the general recommendation.

10) You want to create a "personal mixtape" for all 10k users we have explicit feedback for. This mixtape contains individualized recommendations as to which 10 songs (out of the 5k) a given user will enjoy most. How do these recommendations compare to the "greatest hits" from the previous question and how good is your recommender system in making recommendations?

In this part, we will try the latent factor model and content-based model to compare which has a better recommendation.

Latent Factor Model:

- **Data Preprocessing:** Filling missing values in the dataset with zeros.

Splitting the data into training (80%) and testing (20%) sets.

- **Model Implementation:** Conversion of the datasets into matrix form. Application of Singular Value Decomposition (SVD) with 50 latent factors. Conversion of the sigma matrix into a diagonal matrix. Generation of predicted ratings using the SVD components.
- **Recommendation Generation:** Creation of a function to extract the top 10 song recommendations for each user. Transformation of song indices to song names for interpretability.

User ID	Top 10 Recommendations
0	[CASTLE OF GLASS, Everlong, Last Nite, Start a...
1	[My Type, Californication, Hero, The Kids Aren...
2	[Time to Pretend, Chop Suey!, Flawless, Alive,...
3	[Everybody Talks, Monster, I'm In Love With Yo...
4	[Sweater Weather, Smells Like Teen Spirit, You...

The output displays a variety of recommended songs for different users, indicating a personalized approach.

- **Evaluation of the Recommender System:** Implementation of the Average Precision at K (APK) metric to assess recommendation quality. Calculation of Mean Average Precision (MAP) across all users in the test set, which is around 0.03 for the Latent factor model. And this model is better than the popularity-based model (MAP = 0.002).

Content-based Model:

- **Data Preprocessing:** Normalization of song features (like popularity, duration, danceability, etc.) using MinMaxScaler.
- **User Profile Creation:** Calculation of user profiles based on their ratings and the normalized features of songs they have rated.
- **Prediction of Ratings:** Generation of predicted ratings for each user using cosine similarity between user profiles and song features.

- **Recommendation Generation:** Creation of a function to extract the top 10 song recommendations for each user. Conversion of song indices to song names for clarity and ease of understanding.
- **Results Compilation:** Construction of a DataFrame showcasing user IDs alongside their top 10 song recommendations.

User ID	Top 10 Recommendations
0	[Juan Ingaramo - Por Amarte, John Frusciante -...
1	[Graham Colton - Life's What You Make It, John...
2	[Graham Colton - Life's What You Make It, Rex ...
3	[Graham Colton - Life's What You Make It, John...
4	[Rex Williams - You Are My Heart, Graham Colto...

  The output displays a diverse range of recommended songs for different users, reflecting the content-based approach.
- **Evaluation of the Recommender System:** Utilization of the Average Precision at K (APK) metric to evaluate recommendation quality. Calculation of Mean Average Precision (MAP) for all users in the dataset, which is around 0.023.
- **Comparison with Latent Factor Model and Popularity based model:**
  Unlike the latent factor model, the content-based approach directly uses song features to generate recommendations**.** From the previous question, we see its Mean average precision is around 0.002; the latent factor model is 0.038; the content-based model is 0.023. So personalized recommendations are better than the popularity-based model.

11) Extra credit: Motivated by question 8, we believe that features can be used for genre classification. Since we suspect that low accuracy might be caused by overlaps of features between genres, we decided to perform a binary classification task on two extremely distinguishable genres and check the accuracy.

We manually selected two genres, namely ‘classical’ and ‘anime’, which in common sense sounds very different for binary classification. Given that we only have 2000 data in this case, we again included all 10 features to get a more precise result. We tried out both traditional machine learning models including SVM and decision tree, and neural networks in this section.

Data preprocessing

We used two functions from `sklearn.preprocessing`, `StandardScaler` and `LabelEncoder`. For SVM and decision tree model, we labeled two genres directly with 0 and 1, while for neural networks, we used one-hot encoding to meet the input requirement for LSTM. We also transformed x into 3D input in the format of `[samples, time steps, features]` in the latter case.

Model building and evaluation

Our model building is basically the same as those we used earlier for other questions. After adjusting some hyperparameters with various experiments, we noticed that:

- **SVM** performs quite well here and the accuracy score is around 92%, which is a pretty decent prediction compared to question 8.
- **Decision Tree Classification** performs a little weaker than **SVM**, and the f1 score is around 89%, but the performance is still way better than question 8.

- The accuracy of **Neural Network (LSTM)** increased from 77.42% accuracy initially to 94.45% accuracy on the training set through 30 epochs of training. On the validation set, accuracy improved from 81.25% to 90.62%, reflecting the model's gradually enhanced ability to recognize music genres.

Also, it is worth mentioning that SVM and decision tree classification is still way more efficient on this task than using neural networks. Given the current manageable dataset, the difference might not yet be a huge problem, but we should keep in mind to keep algorithm running time in control.

Testing and conclusion

The model achieves a final accuracy of 93.75% on the testing set, demonstrating its ability when handling unseen data. This final result shows a better performance than Question 8. It reminds us that the choice of target variable is significant.