CSCI 184 Applied Machine Learning
Music Genre Classifier

Names/Teammates: Wallace Hwang, Vincent Joaquin, Pranav Byakod
June 11, 2023

All variables and what they represent:

['danceability', 'energy', 'key', 'loudness', 'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness', 'valence', 'tempo', 'type', 'id', 'uri', 'track_href', 'analysis_url', 'duration_ms', 'time_signature', 'genre', 'song_name', 'Unnamed: 0', 'title']

=

[Suitability of track for dancing, Perceptual measure of intensity and activity, Key the track is in, Loudness of track in decibels, Modality of track, Presence of spoken words, Confidence measure of if track is acoustic, Prediction of if track has no vocals, Presence of audience, Musical positiveness, Beats per minute, Object type, Spotify ID for track, Spotify URI for track, Link to Web API endpoint, URL to full audio analysis, Duration of track in milliseconds, How many beats are in each bar, Genre of song, Song title, Unknown feature, Title of playlist track came from]

Background / Introduction:

Music genre classification is an important task in the field of machine learning and music analysis. Being able to automatically categorize songs into different genres has numerous applications, including music recommendation systems, playlist generation, and music streaming platforms. In this project, we aim to develop a music genre classifier using supervised machine learning techniques. To build our music genre classifier, we decided to divide the project among our team members. Pranav Byakod worked on implementing the K-Nearest Neighbors (KNN) model, Wallace Hwang focused on the Random Forest model, and Vincent Joaquin explored the Neural Network approach. Despite using different algorithms, we all utilized the same set of features and aimed to compare the performance of our models to identify the most effective approach.

Design:

Iteration #1:

Our group ended up deciding on splitting up the project into 3 different models: **K-Nearest Neighbors, Random Forest,** and **Neural Network**. Pranav Byakod would be implementing the K-NN model, Wallace Hwang would be implementing the Random Forest

model, and Vincent Joaquin would be working with a Neural Network. With the project, we will all be utilizing the same features and gathering our results to find which model would be best suited to completing the task at hand. We would all perform feature selection utilizing ANOVA correlation, and use our findings from that correlation to decide on which features would become the inputs of our models. Since we had an abundance of data points and available features from Kaggle, we all unanimously decided that **supervised learning** would be the most efficient means of classifying music genres by their provided features.

After finding the most suitable features to build our model upon, Pranav Byakod and Vincent Joaquin will be using a training, validation, and test set split to tune in on hyperparameters, as well as build their respective models to have the most efficient training and testing accuracy possible. With the K-NN model and SVM model, we will be evaluating those models utilizing metrics like accuracy, precision, recall, and F1 score. On the other hand, with the random forest model, the model will be evaluated with its OOB score and a final accuracy to evaluate it.

Iteration #2:

After attempting to build the model using SVM, Vincent Joaquin realized that the linear boundaries from SVM were making it difficult for the model to accurately predict the genres of a track based on its feature. Thus, instead of using an SVM, he decided on implementing a **Neural Network** for the model instead. Wallace Hwang and Pranav Byakod continued building upon the same models.

Implementation:

K-Nearest Neighbors - Pranav Byakod:

The following libraries are used to implement K-Nearest Neighbors on the genres_v2 dataset.

```
In [30]: import pandas as pd
         import numpy as np
         from sklearn.model_selection import train_test_split
         from sklearn.neighbors import KNeighborsClassifier
         from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
In [31]: df = pd.read_csv('genres_v2.csv', low_memory=False)
         df_updated = df.drop(columns=['type', 'id', 'uri', 'track_href', 'analysis_url', 'song_name', 'Unnamed: 0', 'title'])
         df.head()
```

Out[31]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	...	id	
0	0.831	0.814	2	-7.364	1	0.4200	0.0598	0.013400	0.0556	0.3890	...	2Vc6NJ9PW9gD9q343XFRKx	spotify:track:2Vc6N
1	0.719	0.493	8	-7.230	1	0.0794	0.4010	0.000000	0.1180	0.1240	...	7pgJBLVz5VmnL7uGHmRj6p	spotify:track:7pgJB
2	0.850	0.893	5	-4.783	1	0.0623	0.0138	0.000004	0.3720	0.0391	...	0vSWgAlfpye0WCGeNmuNhy	spotify:track:0vSWg
3	0.476	0.781	0	-4.710	1	0.1030	0.0237	0.000000	0.1140	0.1750	...	0VSXnJqQkwuH2ei1nOQ1nu	spotify:track:0VSX
4	0.798	0.624	2	-7.668	1	0.2930	0.2170	0.000000	0.1660	0.5910	...	4jCeguq9rMTlbMmPHuO7S3	spotify:track:4jCeg

Tue, Jun 6, 2023 4:27 PM

5 rows × 22 columns

After plotting the original dataframe, there were certain columns, such as type, id, url, trach_href, analysis_url, song_name, Unnamed:0, and title, that contained NaN or missing values. Those columns were subsequently removed from the updated dataset to make visualization smoother. Once the data frame was updated, we then searched for useful features using ANOVA classification (as the dataset values are numerical inputs and categorical outputs). The features obtained from this classification were then assigned to a new value X, with the target values set to a variable y, to begin the KNN process.

```
In [39]: X = df_updated[features]
         y = df_updated[target]
```

```
In [40]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [41]: knn = KNeighborsClassifier(n_neighbors=5)
         knn.fit(X_train, y_train)
         y_pred = knn.predict(X_test)
```

```
In [42]: accuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred, average='macro')
         recall = recall_score(y_test, y_pred, average='macro')
         f1 = f1_score(y_test, y_pred, average='macro')
```

Now for any supervised machine learning method, it's important to segregate the data into training and test sets. This helps reduce any data inconsistencies and allows us to test different portions of the dataset. Once that's complete, we set the train and test size options as 80/20 (that was the split the group decided as our base test) and begin classifying the dataset.

To start KNN-classification, we need to come up with a k value that helps improve the scores. Through the elbow method, it was found that a range between 3-7 would be best for this dataset, so we consequently set the number of neighbors to 5 for further analysis. After fitting the model and calculating the scores, we received the following values.

```
In [43]: print('Accuracy:', accuracy)
         print('Precision:', precision)
         print('Recall:', recall)
         print('F1 Score:', f1)

         Accuracy: 0.4011346176574873
         Precision: 0.3715630896382957
         Recall: 0.3727927023923636
         F1 Score: 0.3695606974363932
```

As you can see, KNN did not perform too well in terms of the scores for this dataset. It produced an average of around 0.4, and prediction, recall, and f1 scores of around 0.36-0.37. Increasing the neighbors to 7 does improve the precision, recall, and f1 scores to around 0.39-0.4, however the accuracy still remains stagnant at 0.41. This means that with any modification to the k value, it's unlikely to have a drastic effect on the final scores.

Random Forest - Wallace Hwang:

For my model, these are the following libraries I would be using:

```
In [1]: # Importing libraries and functions from libraries
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.feature_selection import f_classif
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score
```

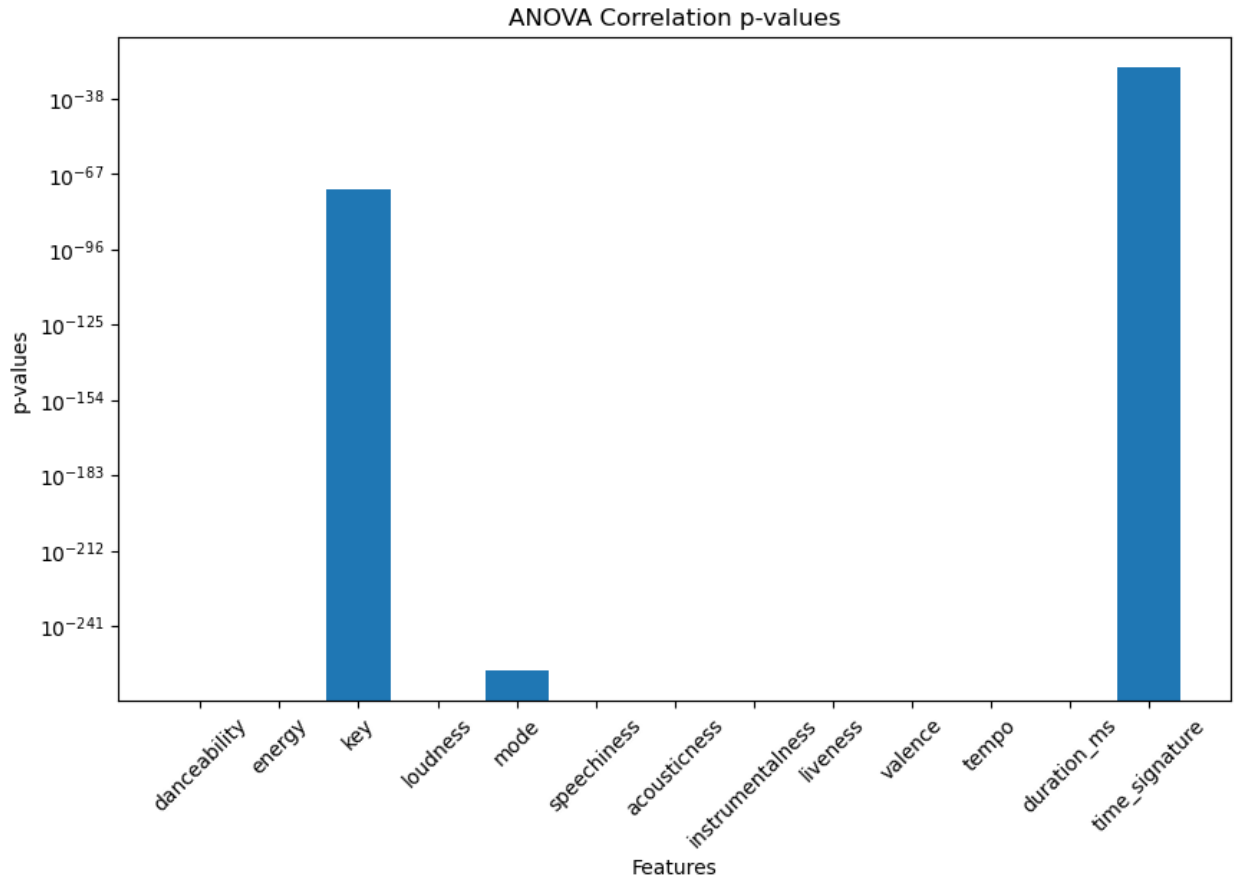
From the dataset we gained from Kaggle, they provided us with 42,305 data points with 22 features assigned to them (The shape of the data frame would be 42,305 columns x 22 rows). However, some of the features provided to us were clearly going to be irrelevant or not beneficial to the implementation of our model, so I eliminated the following features: 'type', 'id', 'uri', 'track_href', 'analysis_url', 'song_name', 'Unnamed: 0', 'title'. Upon dropping those features, the shape of the data frame changed to 42,305 columns x 14 rows.

From there forward, I split my data frame into the inputs and outputs that we would want to eventually train the model on. Since it was important that we know which features to test our model with, I referenced the lectures from camino to find that ANOVA correlation was the best suited filter method to find which features were most suitable for filtering inputs of our model.

```
In [3]: # Gaining information for feature selection on all remaining features vs. our target feature, 'genre'
        X = df.drop('genre', axis = 1)
        y = df['genre']

        anova_corr, p_values = f_classif(X, y)

        anova_corr_df = pd.DataFrame({'Feature': X.columns, 'ANOVA Correlation': anova_corr, 'p-value': p_values})

        anova_corr_df
```

The f_classif function from sklearn provided me with information on the ANOVA correlation for reach feature given that our output was ‘genre’, as well as their p-values (a measurement of the statistical probability that the current feature is correlated with our output feature). Here is a

visual of representation of the p-values for each feature:



To then filter those features, I used an alpha of 0.01 (a significance level of 1%) to choose features that are within that significance level. However, I found that all of our given features were within that range (p-values of less than 0.01), and therefore all remaining features are statistically likely to correlate with the ‘genre’ feature which we are testing for.

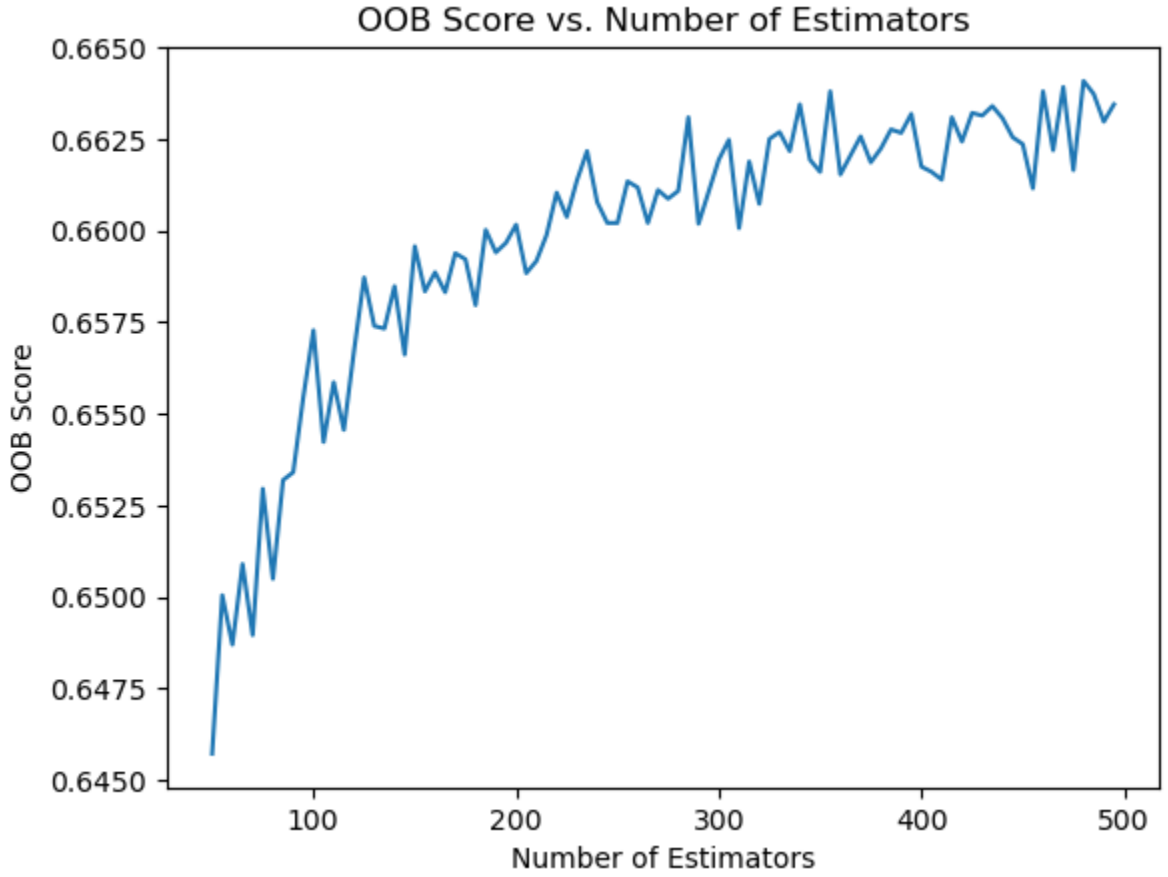
Since we can’t arbitrarily choose the number of trees within random forest, it was important that we test different values of the number of trees to eventually find the optimal number of trees for the model.

Iteration# 1:

I first utilized a train_test_split to divide the dataset into an 80% training set and 20% test set. I would then iterate from 50 to 500 trees with steps of 5 trees at once to find the optimal number of trees. Using the OOB method, I was able to find that the optimal number of trees was 365. However, after attempting to fit the test set to our trained model, I found that while the training accuracy of the model was quite good (98%), the test accuracy of the model was underperforming (66%). Thus, after consulting the professor, I was informed that it would be better to train the model using the entire dataset with the OOB method.

Iteration# 2:

In this iteration of the model, I used the entire dataset to train the model. With the same steps as before, using the OOB method, I was eventually able to find that 480 trees was most optimal to train our model upon. Here is a graph of the OOB scores when compared to the number of trees:



After having found the optimal number of trees, I trained the model using that value and was eventually able to get a quite accurate model. The final results of the model was an accuracy of 92.8%.

In [49]:
```
# Comparison for True Genre values vs Predicted Genre Values
comparison = pd.DataFrame({'Actual value for Train Set': y, 'Predicted value for Train Set': y_pred})
comparison
```

Out[49]:

	Actual value for Train Set	Predicted value for Train Set
0	Dark Trap	Dark Trap
1	Dark Trap	Dark Trap
2	Dark Trap	Dark Trap
3	Dark Trap	Trap Metal
4	Dark Trap	Underground Rap
...	...	...
42300	hardstyle	hardstyle
42301	hardstyle	hardstyle
42302	hardstyle	hardstyle
42303	hardstyle	hardstyle
42304	hardstyle	hardstyle

42305 rows × 2 columns

In [52]:
```
# Accuracy Score of Prediction values
accuracy = accuracy_score(y, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9277626758066423
```

Neural Network - Vincent Joaquin:
Implementation

To create a neural network, I referenced the lecture notes and some online resources to alter my model for multi classification. I used the following libraries:

```
import pandas as pd
import numpy as np
import keras as kr
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical
```

I then imported the dataset as a data frame and one-hot encoded the 15 genres in which the songs are to be classified. I used 13 numerical features: 'danceability', 'valence','tempo', 'energy', 'key', 'duration_ms', 'loudness', 'mode', 'time_signature', 'speechiness', 'acousticness', 'instrumentalness', 'liveness'. Using these features, I tried 3 different models of 1 hidden layer, 3 hidden layers, and 5 different layers. Each hidden layer has an activation function Rectified Linear Unit and the output layer uses “softmax.” My model would run up to 1000 epochs at most, but would stop if accuracy did not change for 300 epochs. The batch sizes were set to 64.

	precision	recall	f1-score	support
0	0.14	0.17	0.15	3667
1	0.00	0.00	0.00	1356
2	0.00	0.00	0.00	2395
3	0.00	0.00	0.00	373
4	0.00	0.00	0.00	1469
5	0.00	0.00	0.00	1697
6	0.00	0.00	0.00	1560
7	0.33	0.78	0.46	4679
8	0.57	0.82	0.67	2391
9	0.29	0.49	0.36	2324
10	0.75	0.72	0.74	2359
11	0.28	0.21	0.24	2349
12	0.56	0.80	0.66	2404
13	0.32	0.29	0.30	2413
14	0.10	0.05	0.07	2408
accuracy			0.36	33844
macro avg	0.22	0.29	0.24	33844
weighted avg	0.26	0.36	0.29	33844

	precision	recall	f1-score	support
0	0.13	0.42	0.20	3652
1	0.00	0.00	0.00	1324
2	0.00	0.00	0.00	2423
3	0.00	0.00	0.00	368
4	0.00	0.00	0.00	1511
5	0.00	0.00	0.00	1678
6	0.00	0.00	0.00	1581
7	0.30	0.55	0.38	4715
8	0.00	0.00	0.00	2372
9	0.00	0.00	0.00	2302
10	0.31	0.96	0.47	2331
11	0.00	0.00	0.00	2368
12	0.00	0.00	0.00	2394
13	0.07	0.15	0.09	2435
14	0.00	0.00	0.00	2390
accuracy			0.20	33844
macro avg	0.05	0.14	0.08	33844
weighted avg	0.08	0.20	0.11	33844

Evaluations for Single layer model and 3-layer model respectively

	precision	recall	f1-score	support
0	0.00	0.00	0.00	3647
1	0.00	0.00	0.00	1350
2	0.00	0.00	0.00	2452
3	0.00	0.00	0.00	375
4	0.00	0.00	0.00	1485
5	0.00	0.00	0.00	1661
6	0.00	0.00	0.00	1537
7	0.14	1.00	0.24	4694
8	0.00	0.00	0.00	2395
9	0.00	0.00	0.00	2303
10	0.00	0.00	0.00	2387
11	0.00	0.00	0.00	2391
12	0.00	0.00	0.00	2368
13	0.00	0.00	0.00	2396
14	0.00	0.00	0.00	2403
accuracy			0.14	33844
macro avg	0.01	0.07	0.02	33844
weighted avg	0.02	0.14	0.03	33844

Evaluation for 5-layer model

I initially thought that more hidden layers would be able to capture the complexity of the dataset, but my model with a single hidden layer performed the best at an accuracy of 36%. Changing the amount of layers seemed to have a negative impact on performance, and changing the number of neurons did not have any significant effect on the performance of the models.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	3647
1	0.00	0.00	0.00	1350
2	0.00	0.00	0.00	2452
3	0.00	0.00	0.00	375
4	0.00	0.00	0.00	1485
5	0.00	0.00	0.00	1661
6	0.00	0.00	0.00	1537
7	0.14	1.00	0.24	4694
8	0.00	0.00	0.00	2395
9	0.00	0.00	0.00	2303
10	0.00	0.00	0.00	2387
11	0.00	0.00	0.00	2391
12	0.00	0.00	0.00	2368
13	0.00	0.00	0.00	2396
14	0.00	0.00	0.00	2403
accuracy			0.14	33844
macro avg	0.01	0.07	0.02	33844
weighted avg	0.02	0.14	0.03	33844

Evaluation of Single Layer model with 50 hidden layer neurons

Results:

Neural Networks performed the worst at 36% accuracy.

KNN performed second worst with a 40% accuracy.

Random forests performed best at 92.8% accuracy.

Conclusions/Interpretations:

After conducting our project on music genre classification using different supervised machine learning algorithms, we have gained valuable insights and reached several conclusions regarding the performance and suitability of the implemented models. Here are the key conclusions and interpretations from our project:

1. Random Forest outperformed K-Nearest Neighbors and Neural Network models: Among the three models we implemented, Random Forest demonstrated the highest accuracy and performed the best in classifying music genres. It achieved an accuracy of 92.8%, indicating its effectiveness in accurately predicting the genre based on the given features. This highlights the power of ensemble learning and the ability of Random Forest to handle complex datasets.
2. K-Nearest Neighbors (KNN) model's performance was relatively lower: The KNN model achieved an accuracy of 40%, which was considerably lower compared to the other models. Despite tuning the hyperparameter for the number of neighbors and attempting to improve the model's performance, KNN struggled to capture the underlying patterns and relationships in the dataset effectively. It may not be the most suitable algorithm for music genre classification with the given features.
3. Neural Network (NN) model showed moderate performance: The Neural Network model achieved an accuracy of 36%, which was the lowest among the implemented models. This suggests that the complexity of the dataset and the selected features may not have been fully captured by the architecture and configuration of the Neural Network. Further experimentation and optimization of the model's structure and hyperparameters could potentially yield better results.
4. Feature selection through ANOVA correlation was effective: Utilizing ANOVA correlation, we identified the most relevant features for music genre classification. All the selected features showed statistically significant correlations with the target variable, indicating their importance in distinguishing different genres. This feature selection process helped streamline the models' inputs and potentially improved their performance by eliminating irrelevant or noisy features.

5. The impact of dataset size and quality: The dataset we used consisted of a substantial number of data points (42,305) and various features. However, it is essential to consider the quality and representativeness of the dataset for achieving higher accuracy. Further exploration with larger and more diverse datasets could provide additional insights and potentially enhance the performance of the models.
6. Model evaluation metrics: In addition to accuracy, we evaluated the models using other metrics such as precision, recall, F1 score, and OOB score (for Random Forest). These metrics provide a more comprehensive understanding of the models' performance by considering aspects such as true positive, false positive, and false negative rates. Analyzing these metrics can help identify specific areas where the models excel or struggle in classifying certain genres.
7. Future improvements and considerations: Although our models achieved reasonably high accuracy, there is always room for improvement. Experimenting with different algorithms, model architectures, and hyperparameters could lead to enhanced performance. Additionally, incorporating additional relevant features or exploring advanced techniques such as deep learning models might yield better results. It is also important to consider the generalizability of the models by testing them on diverse datasets and ensuring their robustness in classifying genres across different music domains.

In conclusion, our project on music genre classification using supervised machine learning techniques provided valuable insights into the performance and suitability of various models. The Random Forest model stood out as the most accurate and effective approach, while the K-Nearest Neighbors and Neural Network models showed room for improvement. The feature selection process, dataset quality, and evaluation metrics played crucial roles in evaluating and interpreting the models' performance. These findings contribute to the field of music analysis and provide a foundation for further research and advancements in music genre classification.

<u>Resources Used:</u>

Software and Tools we used and for what purpose they served:

- Jupyter Notebook/Google Colab
    - For writing the code in Python
- Python Libraries (pandas, numpy, scikit-learn, matplotlib)
    - To conduct data analysis
    - Documentation for libraries for referencing and better understanding of built-in functions
- Websites used to collect data:
    - Kaggle: https://www.kaggle.com/datasets/mrmorj/dataset-of-songs-in-spotify?select=genres_v2.csv

        - Used to collect data points of tracks and their respective features from spotify API
    - Spotify API: https://developer.spotify.com/documentation/web-api/reference/get-audio-features
        - Used to interpret features and their relevance to our models