University of Tehran

School of Mechanical Engineering

Artificial Intelligence

Home Work 3

Professor:

Dr. Masoud Shariat Panahi

Author:

Masoud Pourghavam

May, 2023

Table of Contents

List of Figures

List of Tables

1. Classification of plants

Smart greenhouse technologies, first of all, require automatic detection of the desired type of plants. This diagnosis is usually done with the help of the external characteristics of the plants. The attached data (iris.csv) contains five columns. The first four columns show the inputs (characteristics) of the plant, including sepal length and width, petal length and width, and the fifth column shows the output or plant class. The three different classes of iris plant are represented in Figure 1.

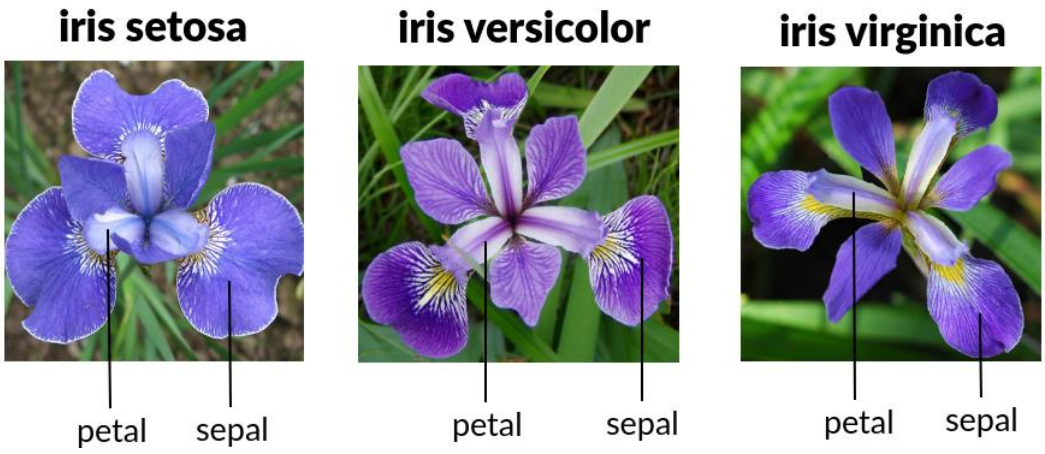

Fig 1. Different classes of iris plant

Through the Figures 2 and 3, the distribution of data in terms of Speal and Petal length and width are shown. For a better understanding of dataset, we use an overall scatter plot given in Figure 4 to show the class data in terms of different features.

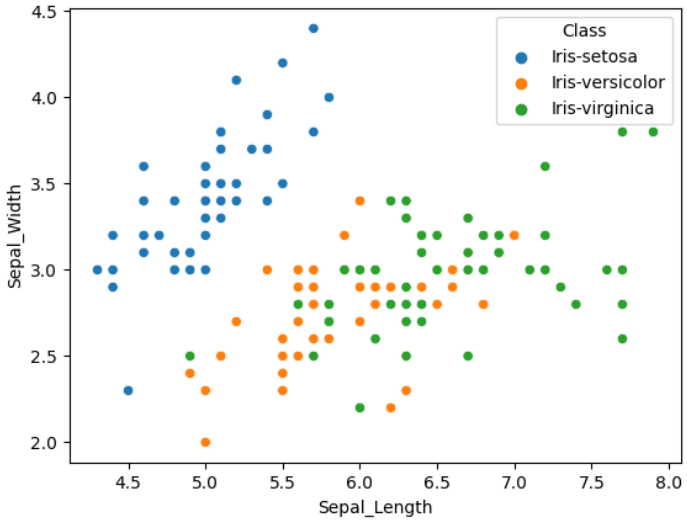

Fig 2. Scatter plot of plant classes for sepal dimension

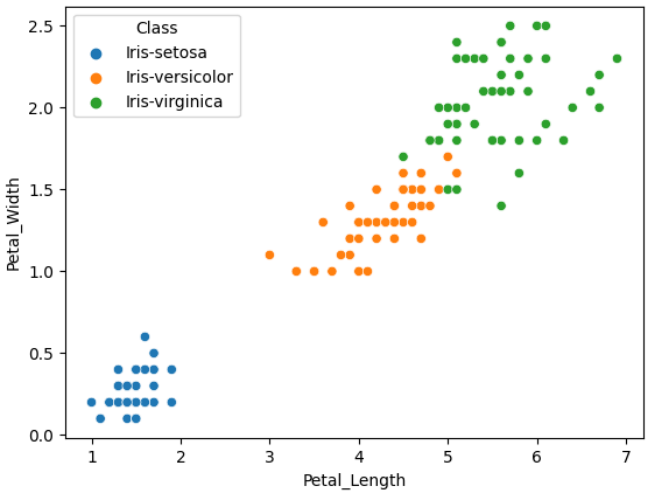

Fig 3. Scatter plot of plant classes for petal dimension

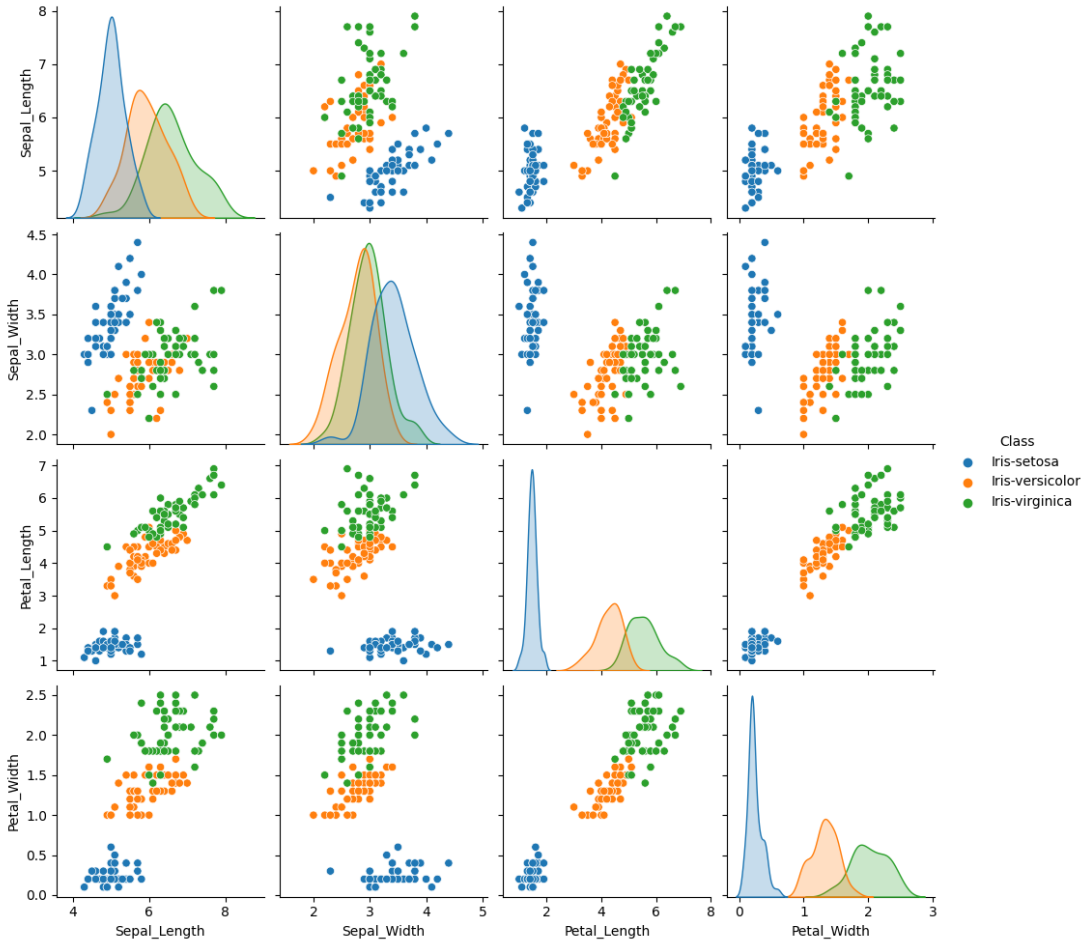

Fig 4. Seaborn pair plot of dataset

In this problem, we will use the following libraries shown in Table 1.

Table 1. Libraries used in section 1.

No.	Library title
1	pandas
2	numpy
3	matplotlib
4	scikit-learn
5	seaborn
6	warnings

We will use the above libraries for the following purposes:

- Pandas: Data manipulation and analysis library for tabular data.
- NumPy: Fundamental library for scientific computing with support for multi-dimensional arrays and mathematical functions.
- Matplotlib: Data visualization library with customizable plotting functions.
- Scikit-learn: Machine learning library with algorithms for classification, regression, clustering, and more.
- Seaborn: Statistical data visualization library built on top of Matplotlib.
- Warnings: Module for controlling warning messages in Python.

A. K-nearest neighbor algorithm (KNN)

i. Splitting the data into training and testing sets

First, we read the CSV dataset using the following code:

```
iris_df = pd.read_csv('Iris.csv')
```

Then, consider 80% of the data for training and the remaining 20% for testing using the following code:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

After that, we train the model for plant classification using KNN algorithm with K=3.

```
classifier=KNeighborsClassifier(n_neighbors =3, metric = 'minkowski',p = 2)
classifier.fit(X_train, y_train)
```

ii. Accuracy, recall, precision, and jaccard scores

Here, we calculate the accuracy, recall, precision and jaccard scores as shown in Tables 2 to 5 and the confusion matrix as represented in Figure 5.

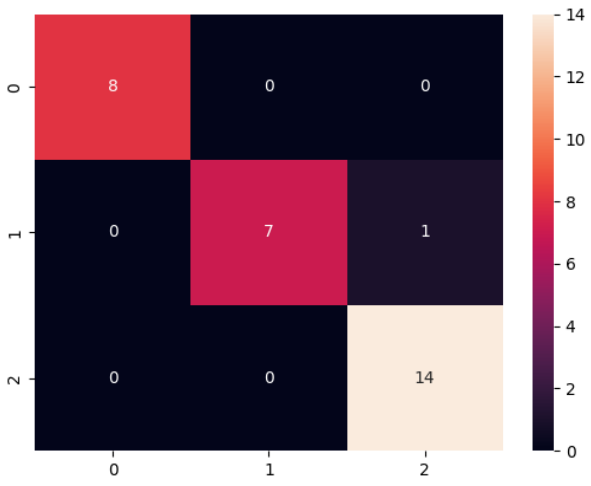

Fig 5. Confusion matrix without normalization

Table 2. Jaccrad similarity scores without normalization

Jaccard score	
Iris-setosa	1.000
Iris-versicolor	0.875
Iris-virginica	0.933
Macro average accuracy	**0.936**

Table 3. Recall scores without normalization

Recall score	
Iris-setosa	1.000
Iris-versicolor	0.875
Iris-virginica	1.000
Macro average accuracy	**0.958**

Table 4. Precision scores without normalization

Precision score	
Iris-setosa	1.000
Iris-versicolor	1.000
Iris-virginica	0.933
Macro average accuracy	**0.977**

Table 5. Accuracy scores without normalization

Accuracy score	
Iris-setosa	1.000
Iris-versicolor	0.937
Iris-virginica	0.966
Macro average accuracy	**0.968**

B. Normalization

After the standard normalization using following code, we will have the confusion matrix given in Figure 6. The scores with normalization are given in Tables 6 to 9.

```
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

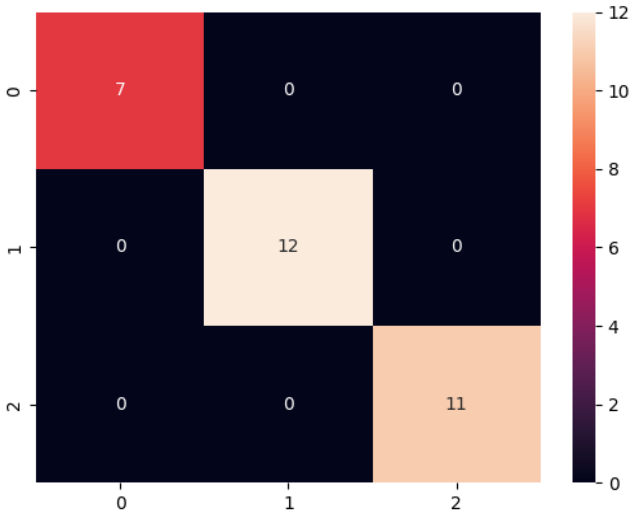

Fig 6. Confusion matrix with normalization

Table 6. Jaccrad similarity scores with normalization

Jaccard score	
Iris-setosa	1.000
Iris-versicolor	1.000
Iris-virginica	1.000
Macro average accuracy	**1.000**

Table 7. Recall scores with normalization

Recall score	
Iris-setosa	1.000
Iris-versicolor	1.000
Iris-virginica	1.000
Macro average accuracy	**1.000**

Table 8. Precision scores with normalization

Precision score	
Iris-setosa	1.000
Iris-versicolor	1.000
Iris-virginica	1.000
Macro average accuracy	**1.000**

Table 9. Accuracy scores with normalization

Accuracy score	
Iris-setosa	1.000
Iris-versicolor	1.000
Iris-virginica	1.000
Macro average accuracy	**1.000**

i. Comparison of results before and after the normalization

The results obtained from the classification using the KNN algorithm on the iris plant dataset are quite impressive, indicating a high level of accuracy and performance. The Jaccard score, also known as the Jaccard similarity coefficient, measures the similarity between the predicted and true classes. A score of 1 indicates that all instances are correctly classified and there are no false positives or false negatives. In the initial classification, the Jaccard score is 0.936, suggesting that the algorithm is performing well but with a small number of misclassifications. Recall, also called sensitivity or true positive rate, measures the ability of the classifier to correctly identify positive instances. A recall score of 0.958 indicates that 95.8% of the instances belonging to each class were correctly classified by the KNN algorithm.

Precision represents the accuracy of the classifier in correctly identifying positive instances out of the total instances predicted as positive. A precision score of 0.977 suggests that 97.7% of the instances predicted as positive were actually true positives. Accuracy measures the overall correctness of the classifier across all classes. The accuracy score of 0.968 implies that 96.8% of all instances in the dataset were correctly classified.

Now, let's discuss the impact of standard normalization on the metrics:

Standard normalization (also known as z-score normalization) scales the features of the dataset to have zero mean and unit variance. This normalization technique is often used to ensure that all features contribute equally to the classification process. After applying standard normalization to the data, we saw that the Jaccard, recall, precision, and accuracy scores will all be 1.

The increase in accuracy after the normalization process indicates that the standardization of the data has improved the performance of the KNN algorithm for classifying the iris plants. The normalization process typically helps algorithms by bringing the features onto the same scale, removing biases caused by different magnitudes or units in the original data. In the context of KNN classification, the algorithm determines the class membership of an instance based on the classes of its k nearest neighbors in the feature space. If the features have significantly different scales, certain features with larger magnitudes can dominate the distance calculation and influence the classification decision more than others. This can lead to biased or suboptimal results. By applying standard normalization to the data, the features are transformed to have zero mean and unit variance. This equalizes the impact of each feature and ensures that all features contribute equally to the distance calculations. Consequently, the KNN algorithm can make more accurate and unbiased predictions, resulting in a higher accuracy score.

The increase in accuracy suggests that the normalization process has successfully addressed any issues related to varying scales in the original data and has improved the KNN algorithm's ability to correctly classify the iris plants. It indicates that the algorithm can now make more reliable and robust predictions, leading to a higher overall accuracy rate.

2. Prediction of song genre and popularity

In order to increase its audience, a music site plans to rank songs released between 2000 and 2020 in terms of genre and popularity. For this purpose, features such as song title, artist, release year, energy, tempo, etc. are used. The desired data is provided in the musics.csv file. This file contains 18 columns, two columns are popularity and genre and 16 other columns show the inputs.

Table 10. Libraries used in section 2.

No.	Library title
1	pandas
2	numpy
3	matplotlib
4	scikit-learn
5	seaborn
6	warnings
7	collections

We will use the above libraries for the following purposes:

- Pandas: Data manipulation and analysis library for tabular data.
- NumPy: Fundamental library for scientific computing with support for multi-dimensional arrays and mathematical functions.
- Matplotlib: Data visualization library with customizable plotting functions.
- Scikit-learn: Machine learning library with algorithms for classification, regression, clustering, and more.
- Seaborn: Statistical data visualization library built on top of Matplotlib.
- Warnings: Module for controlling warning messages in Python.
- Collections: Provides specialized data structures such as Counter, defaultdict, namedtuple, and deque for efficient and convenient data processing tasks.

A. Initialization

i. Columns elimination

First, we read the CSV dataset using the following code:

```
df = pd.read_csv('musics.csv')
```

Then, we will delete the song, year, and artist columns in the dataset using the following code:

```
df.drop(df.columns[[0, 1, 4]], axis=1, inplace=True)
```

We deleted the mentioned columns because in some cases, the song, year, and artist columns may not be directly relevant to the analysis or task at hand. Removing these columns helps to focus on the features that are more directly related to the problem being addressed. On the other hand, removing unnecessary columns reduces the dimensionality of the dataset. High-dimensional datasets can be more challenging to work with, both in terms of computational complexity and interpretability. By reducing the number of columns, the dataset becomes more manageable.

Removing non-essential columns simplifies the data analysis process. It allows us to focus on the remaining relevant features, making it easier to understand patterns, relationships, and insights in the data. Also, the song, year, and artist columns may contain sensitive or personally identifiable information. By removing these columns, we can protect the privacy of individuals or comply with data protection regulations. Removing irrelevant or redundant columns can lead to better model performance. Unnecessary features can introduce noise or bias into the model, leading to overfitting or decreased generalization ability. Removing these columns focuses the model on the most important and informative features.

So, we can conclude that removing the song, year, and artist columns can streamline the dataset, improve model performance, simplify analysis, and potentially protect privacy or comply with regulations. The specific advantages will depend on the context and goals of the analysis or task.

ii. Popularity column

To replace the values in the popularity column of our dataset according to the specified ranges in the problem, we can use the map() function along with a dictionary to define the mapping of ranges to replacement values.

```
popularity_mapping = {range(0, 21): 1, range(21, 41): 2, range(41, 61): 3, range(61, 81): 4, range(81, 101): 5}
```

In the following code, the popularity_mapping dictionary defines the ranges as keys and the corresponding replacement values. The map() function is then used on the popularity column to replace the values based on the mapping.

```
df['popularity'] = df['popularity'].map({value: replacement for key, replacement in popularity_mapping.items() for value in key})
```

B. Qualitative to quantitative

The reason and advantage of changing data from qualitative to quantitative in the explicit and genre columns is that the most machine learning algorithms require numerical inputs. By converting qualitative data to quantitative form, we make it compatible with these algorithms, allowing us to apply various machine learning techniques for analysis, prediction, and classification tasks. On the other hand, quantitative data provides a more structured and standardized representation of information compared to qualitative data. It enables us to capture and encode specific patterns, relationships, and characteristics of the data, which can be leveraged by machine learning algorithms to make accurate predictions or classifications.

Quantitative data can improve the performance of machine learning models. Numerical representations allow models to apply mathematical operations, calculate distances, and estimate relationships between features more effectively. This can lead to better model performance, higher accuracy, and improved generalization ability. Quantitative data is often easier to explore and visualize compared to qualitative data. With quantitative data, we can leverage various statistical techniques and visualizations such as histograms, scatter plots, and correlation matrices to gain insights, identify patterns, and understand the underlying relationships between variables.

Quantitative data opens up opportunities for conducting statistical analysis, hypothesis testing, and deriving meaningful insights from the data. It allows us to apply statistical techniques such

as regression analysis, ANOVA, or correlation analysis to explore relationships and draw conclusions based on the numerical representations. Also, converting qualitative data to quantitative form introduces consistency and standardization across the dataset. It enables us to apply common data preprocessing techniques, handle missing values, perform scaling or normalization, and ensure uniformity in the representation of the data, facilitating easier data manipulation and analysis.

i. Explicit column

To replace the values in the explicit column of our dataset, we can use the replace() method from pandas.

```
df['explicit'].replace({'true': 0, 'false': 1}, inplace=True)
```

ii. Genre column

We create a new column in the dataset called new_genre.

```
df['new_genre'] = df['genre'].apply(lambda x: ', '.join(x.split(',')))
```

We find all available genres, then, determine the total number of available genres and create a one hot encoding for the new_genre column. The results can be seen in the "updated_dataset.csv" file as shown in Figure 7.

O	P	Q	R	S	T	U	V	W	X	Y
genre	new_genre	genre_ Da	genre_ Fol	genre_ R&	genre_ blu	genre_ cla	genre_ cou	genre_ eas	genre_ hip	genre_ jaz
['pop']	pop	0	0	0	0	0	0	0	0	0
['rock', ' pop']	rock, pop	0	0	0	0	0	0	0	0	0
['pop', ' country']	pop, country	0	0	0	0	0	1	0	0	0
['rock', ' metal']	rock, metal	0	0	0	0	0	0	0	0	0
['pop']	pop	0	0	0	0	0	0	0	0	0
['hip hop', ' pop', ' R&B']	hip hop, pop, R&B	0	0	1	0	0	0	0	0	0
['hip hop']	hip hop	0	0	0	0	0	0	0	0	0
['pop', ' rock']	pop, rock	0	0	0	0	0	0	0	0	0
['pop', ' R&B']	pop, R&B	0	0	1	0	0	0	0	0	0
['Dance/Electronic']	Dance/Electronic	0	0	0	0	0	0	0	0	0
['pop']	pop	0	0	0	0	0	0	0	0	0
['pop']	pop	0	0	0	0	0	0	0	0	0
['pop']	pop	0	0	0	0	0	0	0	0	0
['rock', ' pop']	rock, pop	0	0	0	0	0	0	0	0	0

Fig 7. One hot encoding of genre column

C. Data distribution

Analyzing the data distribution provides insights into the underlying patterns, characteristics, and behavior of the dataset. It helps in understanding the range, spread, central tendency, and shape of the data, which is crucial for making informed decisions and drawing meaningful conclusions. Data distribution analysis helps identify potential issues such as outliers, skewed distributions, or imbalances in the data. These insights guide data preprocessing steps, including outlier removal, data transformation, or handling imbalanced classes, to ensure the data is suitable for subsequent analysis or modeling tasks. In the Figure 8, the data distribution for each feature is shown. Through the Figure 9, the distribution of popularity levels with different colors are given.

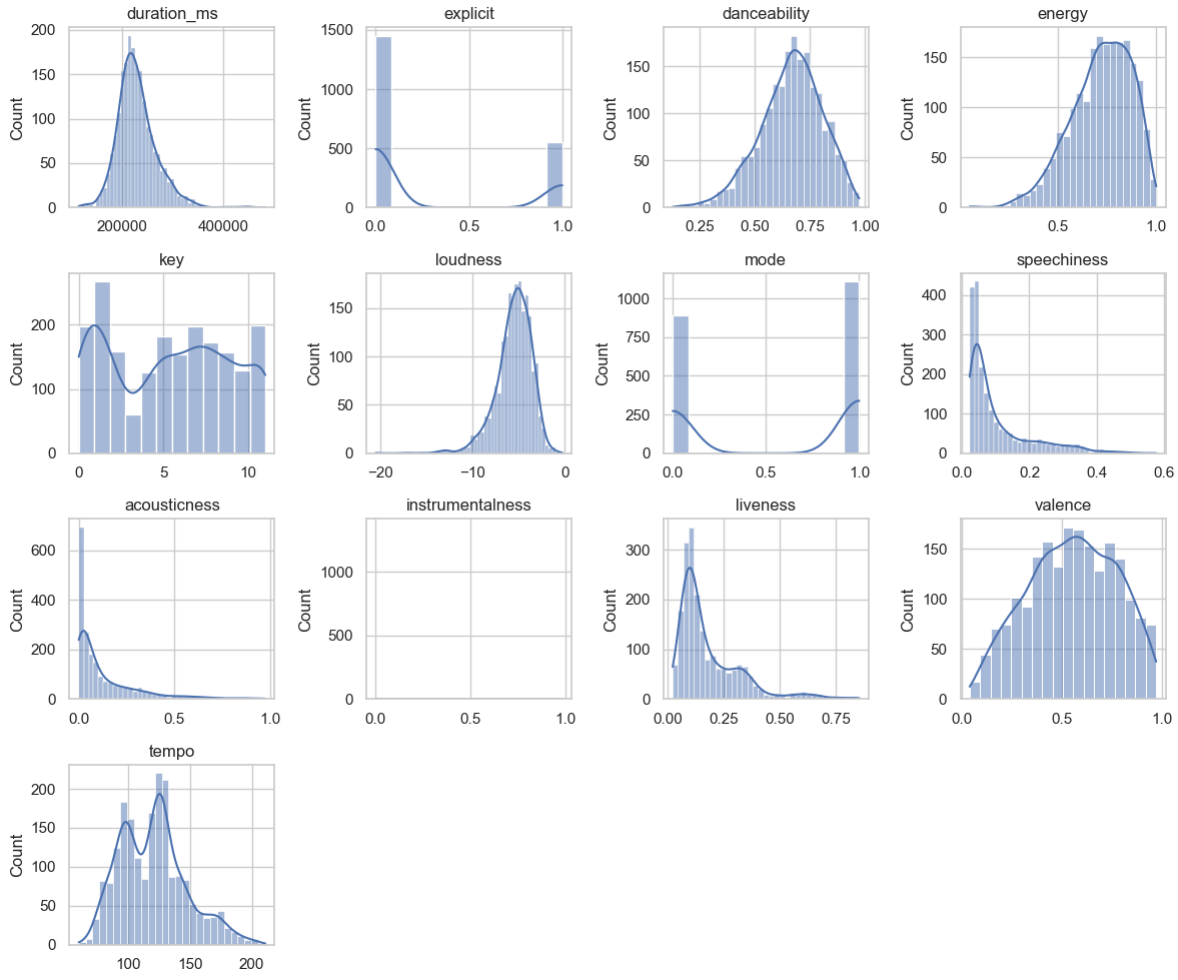

Fig 8. Features data distribution

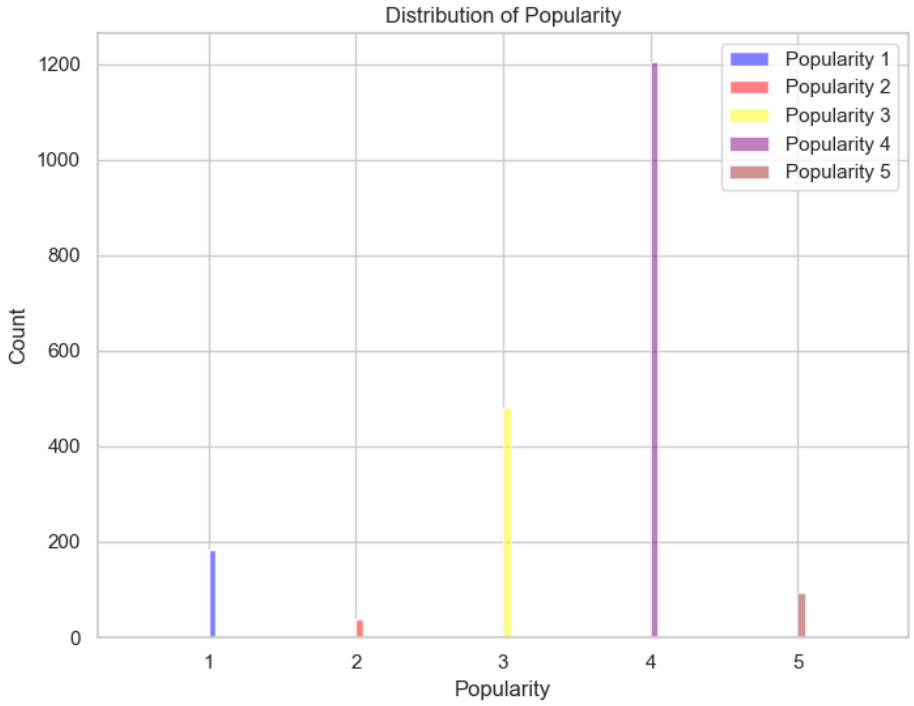

Fig 9. Distribution of popularity levels with different colors

Examining the dispersion of data in terms of different features helps in feature selection and engineering. Features that exhibit low variability or have minimal influence on the target variable may be candidates for removal or transformation. On the other hand, features with high dispersion or strong correlations with the target variable may be valuable for predictive modeling. The dispersion of data analysis aids in identifying outliers, which are observations that deviate significantly from the typical patterns or expected behavior. Outliers can have a substantial impact on the analysis and modeling results. Detecting and addressing outliers ensures the robustness and accuracy of subsequent analyses and models.

Showing the dispersion of data through visualizations, such as histograms, box plots, or scatter plots, allows for effective communication of the data's characteristics to stakeholders. Visual representations of data distribution provide an intuitive way to convey insights, patterns, and relationships, facilitating better understanding and decision-making. In Figure 10, we showed the dispersion of data in terms of different features.

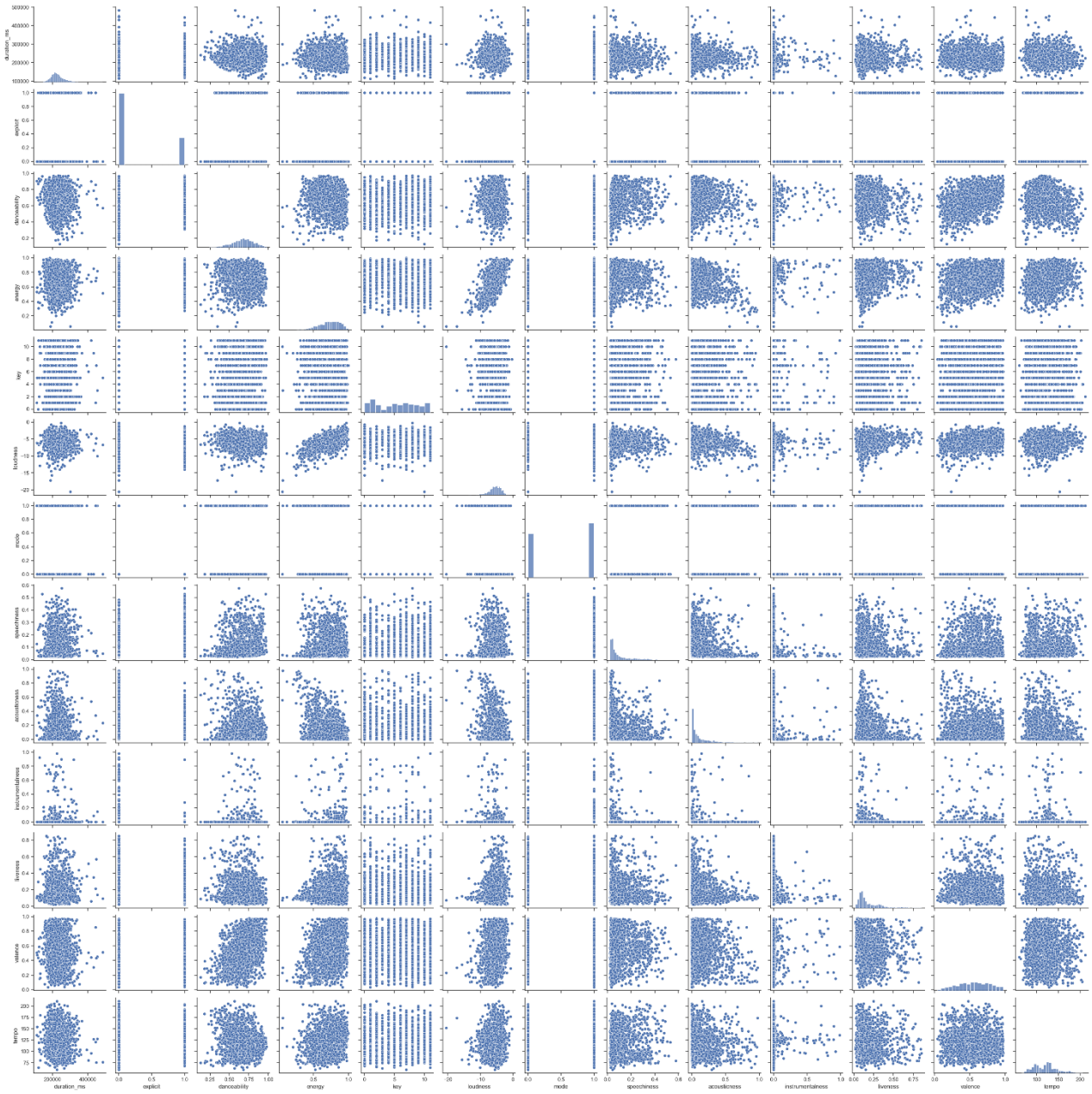

Fig 10. The dispersion of data in terms of different features

Understanding the nature of data through the dispersion of data points in a scatter plot can provide insights into the relationships and patterns among different features. We should look at the distribution and spread of the data points on the scatter plot and consider the following aspects. In the Figure 10, where the data points are clustered closely together, it suggests a high degree of similarity or correlation between the two features. It indicates that the two variables might have a strong positive or negative relationship. For example, in above plot, where we observe a tight cluster of points sloping upwards from left to right, it implies that as one feature increases, the other feature tends to increase as well.

We can identify any data points that lie far away from the main cluster or trend of the data. These outliers can indicate anomalies or unusual observations. They may also suggest interesting patterns or errors in the data.

We can look for any specific patterns or structures in the dispersion of the data points. It could be linear, curved, or irregular. These patterns can provide valuable information about the relationship between the two features.

We can observe the density of data points in different regions of the scatter plot. Areas with a higher concentration of data points suggest a greater frequency or occurrence of certain combinations of feature values.

Depending on the context, we can further analyze the dispersion of data by calculating statistical measures like correlation coefficients, regression lines, or fitting curves to the data. These techniques can quantify and describe the relationships between the features more precisely.

D. Normalization and correlation

i. Normalization

In this part, first, we select the all the columns for normalization except the genre and popularity columns. We can use the MinMaxScaler from the scikit-learn library. The MinMaxScaler scales the data to a specified range, typically between 0 and 1.

```
scaler = MinMaxScaler()
```

In the following code, we use the fit_transform() method to normalize the selected columns in the dataset, replacing the original values with the normalized values.

```
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])
```

Then, using the following code, we can see a few of normalized data as shown in Figure 11.

```
print(df.head())
```

	duration_ms	explicit	popularity	...	genre_pop	genre_rock	genre_set()
0	0.264478	0.0	4	...	1	0	0
1	0.145673	0.0	4	...	0	1	0
2	0.370598	0.0	4	...	1	0	0
3	0.300402	0.0	4	...	0	1	0
4	0.235918	0.0	4	...	1	0	0

Fig 11. Data normalization

ii. Correlation

To find the relationship and influence of each feature on the popularity data, we can use correlation analysis and visualize it using a pair plot from the seaborn library. By examining the correlation matrix and pair plot, we can gain insights into the relationship and influence of each feature on the popularity data. Positive or negative correlation values indicate the direction and strength of the relationship between features.

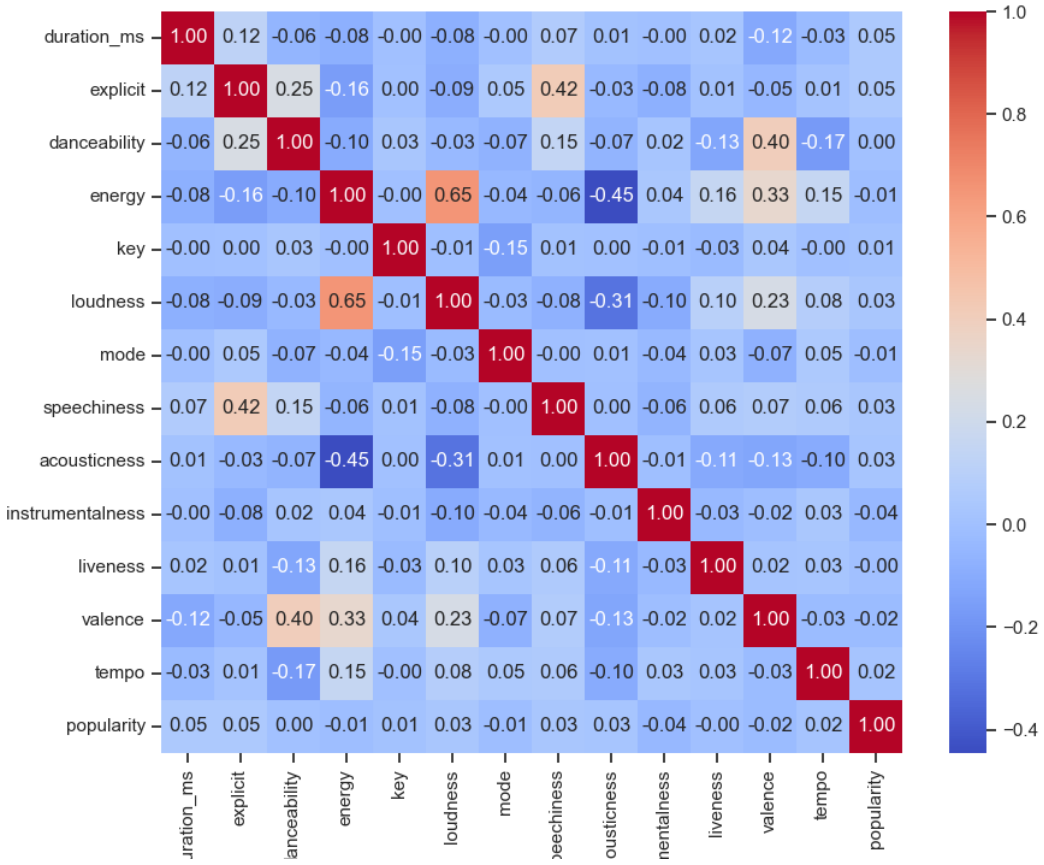

Fig 12. Relationship and influence of each feature on the popularity data using correlation matrix

In the above correlation matrix, the numbers represent the correlation coefficients between pairs of features. Correlation coefficients measure the strength and direction of the linear relationship between two variables. The correlation coefficient ranges from -1 to 1, with 0 indicating no linear relationship. Here's what different values of correlation coefficients represent:

A correlation coefficient between 0 and 1 indicates a positive correlation. This means that as one feature increases, the other feature tends to increase as well. The closer the coefficient is to 1, the stronger the positive correlation. A correlation coefficient between 0 and -1 indicates a negative correlation. This means that as one feature increases, the other feature tends to decrease. The closer the coefficient is to -1, the stronger the negative correlation. A correlation coefficient of 0 indicates no linear relationship between the two features. This means that there is no consistent pattern of change between the variables.

E. Splitting the data into training, testing, and validating sets

To split the data into training, testing, and validation sets with the specified ratios, we can use the scikit-learn library's train_test_split function. We start by splitting the data into a temporary set (train_temp) and a combined testing and validation set (test_val). We set the test_size parameter to 0.1, indicating that 10% of the data will be allocated to the test_val set. We also set the random_state parameter to a specific value (42 in this example) to ensure reproducibility of the split.

```
train_temp, test_val = train_test_split(df, test_size=0.2, random_state=42)
```

Next, we split the test_val set into separate testing (test) and validation (val) sets. We set the test_size parameter to 0.5, indicating that 50% of the temporary set will be allocated to each of the testing and validation sets.

```
test, val = train_test_split(test_val, test_size=0.5, random_state=42)
```

Finally, we print the sizes of the resulting training, testing, and validation sets using the shape attribute of the dataset using the following code and the result is given in Figure 13.

```
print(f"Training set size: {train_temp.shape[0]}")
print(f"Testing set size: {test.shape[0]}")
print(f"Validation set size: {val.shape[0]}")
```

```
[5 rows x 41 columns]
Training set size: 1600
Testing set size: 200
Validation set size: 200
```

Fig 13. Training, testing, and validation data sets

F. Decision tree, random forest, KNN, and SVM classifiers

i. Decision tree

We define the features (X) and the target variable (y) based on the column names in the dataset. Then, the train_test_split function is called twice. First, it splits the data into training data and temporary data with a test size of 20%. Then, the temporary data is further split into testing data and validation data with a test size of 50%. The decision tree is represented in Figure 14.

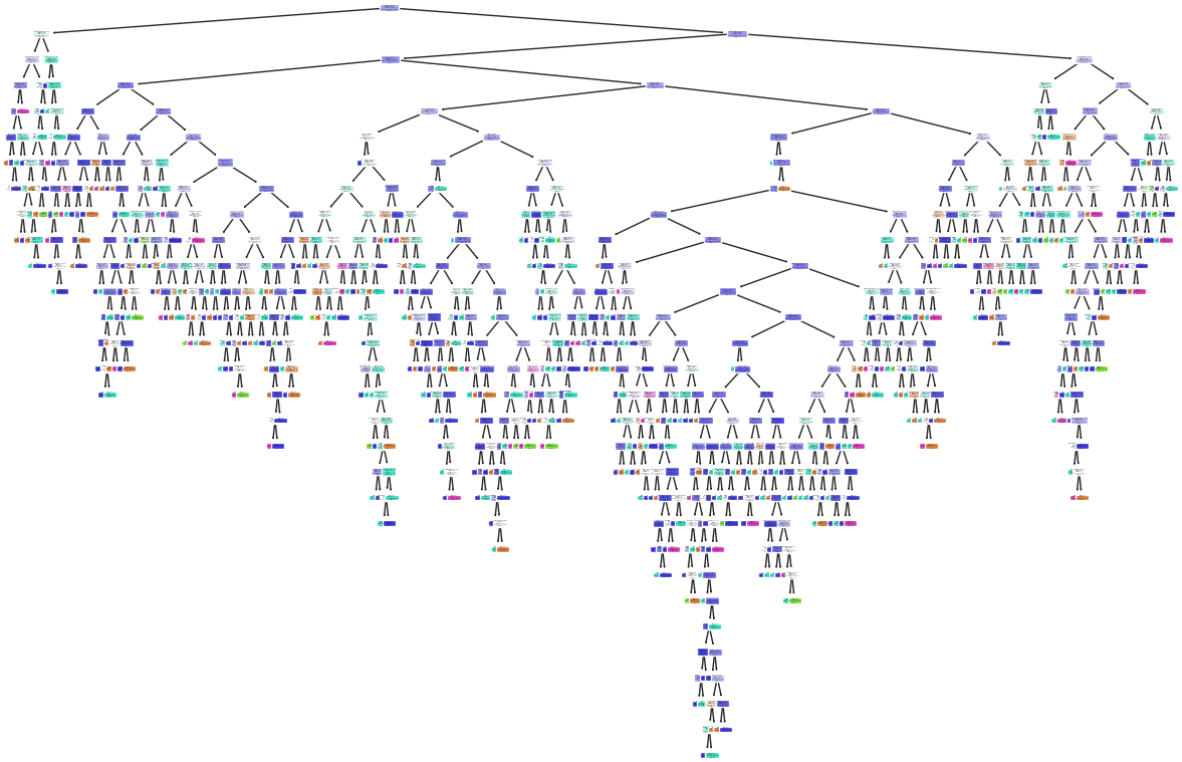

Fig 14. Decision tree

Then, we create a decision tree classifier using the DecisionTreeClassifier class. We train the classifier on the training data using the fit method. After that, we use the trained classifier to predict the popularity for the testing data using the predict method. Next, we create a confusion matrix using the confusion_matrix function, which compares the predicted values with the actual values from the testing set. Finally, we calculate and print the accuracy, precision, recall, F1, and error rate scores using the accuracy_score function, which measures the accuracy of the predictions compared to the actual values.

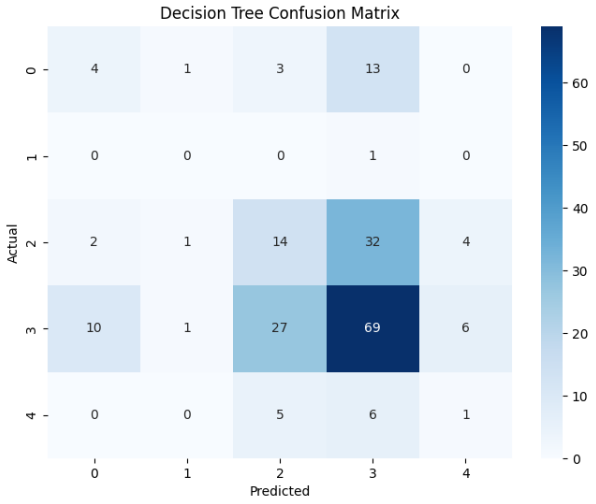

Fig 15. Decision tree confusion matrix

Table 11. Decision tree algorithm scores

Score	Value
Accuracy	**0.440**
Precision	0.429
Recall	0.440
F1-score	0.443
Error rate	0.560

ii. Random forest

We create a Random Forest classifier using the RandomForestClassifier class. After that, we train the classifier on the training data using the fit method. Next, we use the trained classifier to predict the popularity for the testing data using the predict method. We create a confusion matrix using the confusion_matrix function, which compares the actual popularity values (from test_target) with the predicted popularity values (stored in predictions). We calculate and print the accuracy score using the accuracy_score function, which measures the accuracy of the predictions. Finally, we print the confusion matrix to display the counts of true positive, true negative, false positive, and false negative predictions.

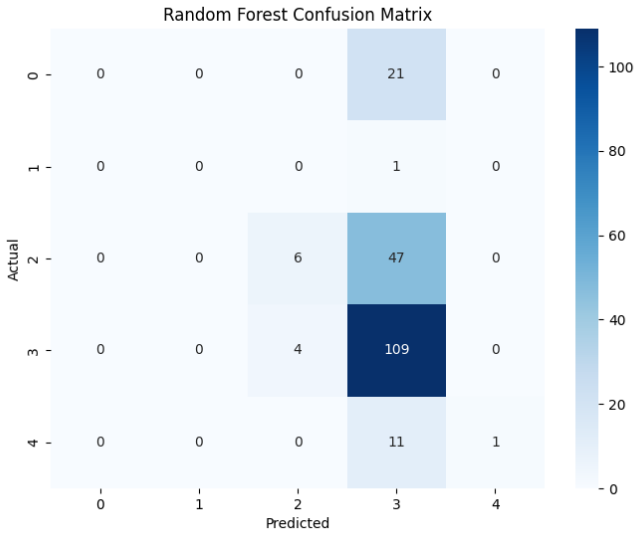

Fig 16. Random forest confusion matrix

Table 12. Random forest algorithm scores

Score	Value
Accuracy	**0.580**
Precision	0.544
Recall	0.580
F1-score	0.467
Error rate	0.420

iii. KNN

First of all, we create a KNN classifier using the KNeighborsClassifier class. After that, we train the classifier on the training data using the fit method. Next, we use the trained classifier to predict the popularity for the testing data using the predict method. We create a confusion matrix using the confusion_matrix function, which compares the actual popularity values (from test_target) with the predicted popularity values (stored in predictions). We calculate and print the accuracy score using the accuracy_score function, which measures the accuracy of the predictions. Finally, we print the confusion matrix to display the counts of true positive, true negative, false positive, and false negative predictions.

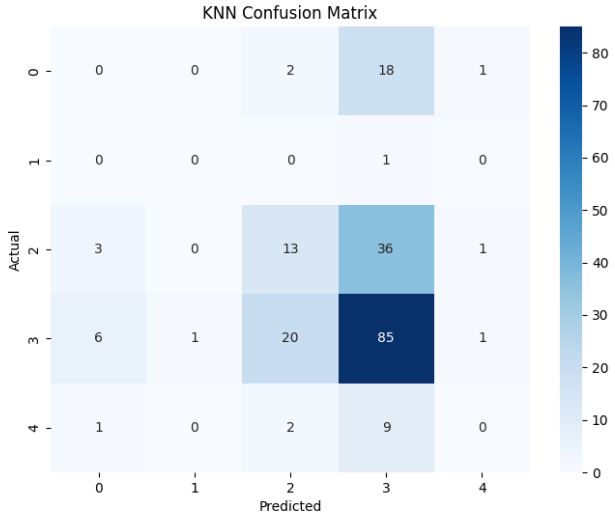

Fig 17. KNN confusion matrix

Table 13. KNN algorithm scores

Score	Value
Accuracy	**0.490**
Precision	0.410
Recall	0.490
F1-score	0.443
Error rate	0.510

iv. SVM

First, we create an SVM classifier using the SVC class. After that, we train the classifier on the training data using the fit method. Next, we use the trained classifier to predict the popularity for the testing data using the predict method. We create a confusion matrix using the confusion_matrix function, which compares the actual popularity values (from test_target) with the predicted popularity values (stored in predictions). We calculate and print the accuracy score using the accuracy_score function, which measures the accuracy of the predictions.

Finally, we print the confusion matrix to display the counts of true positive, true negative, false positive, and false negative predictions.

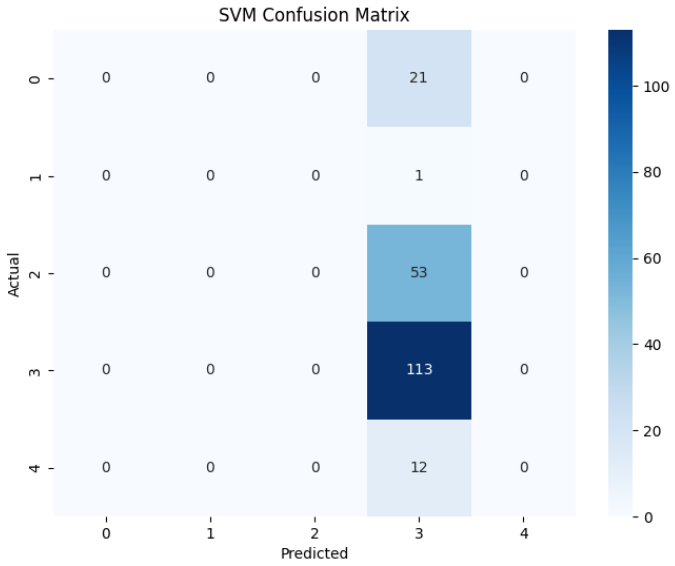

Fig 18. SVM confusion matrix

Table 14. SVM algorithm scores

Score	Value
Accuracy	**0.565**
Precision	0.319
Recall	0.565
F1-score	0.407
Error rate	0.435

To evaluate the best classifier algorithm among the decision tree, random forest, KNN, and SVM, we can compare their performance based on the calculated scores. Based on the obtained scores, the random forest classifier appears to have the highest accuracy score of 0.58. Comparing the other metrics, it also has the highest precision score (0.544), recall score (0.580), and F1-score (0.467) among the classifiers. The error rate score is relatively lower at 0.420 compared to the other classifiers. Therefore, the random forest classifier seems to be the best performing algorithm among the options.

The random forest algorithm outperforms the other classifiers due to its ensemble learning approach. It combines multiple decision trees and aggregates their predictions to make more accurate predictions. The random forest classifier can handle a large number of input variables effectively, handle missing data, and provide a robust solution against overfitting. It also

reduces variance and provides a better generalization ability compared to a single decision tree. These characteristics make the random forest classifier more robust and suitable for complex datasets like the one we used.

The accuracy score of the chosen best classifier algorithm, random forest, is 0.58. Whether this accuracy is appropriate or not depends on the specific context and the requirements of our project. It's important to consider the baseline accuracy and the desired level of performance. If the baseline accuracy is significantly lower than 0.58, then, achieving this accuracy can be considered appropriate. However, if the project requires higher accuracy, then further improvements may be needed.

Some suggestions for increasing the accuracy of the classifier algorithms:

a. Analyze and preprocess our dataset, identifying relevant features and removing any irrelevant or redundant ones. We can also consider creating new features based on domain knowledge to improve the performance of the classifiers.
b. Optimize the hyperparameters of the classifiers. Grid search or random search techniques can help us find the best combination of hyperparameters for each algorithm. This process involves adjusting parameters such as the maximum depth of decision trees, number of trees in random forests, the number of neighbors in KNN, or the kernel and regularization parameters in SVM.
c. Utilize cross-validation techniques to estimate the performance of the classifiers more accurately. This helps to detect and reduce overfitting and provides a more realistic evaluation of their generalization ability.
d. Consider using ensemble methods, such as boosting or stacking, to combine multiple classifiers and improve their overall performance. Ensemble methods can often produce better results by leveraging the strengths of different algorithms.
e. If our dataset is relatively small, we can explore techniques like oversampling or data augmentation to generate synthetic samples and balance the class distribution. This can help improve the performance of the classifiers, especially in scenarios with imbalanced classes.
f. We can also try other classification algorithms that are known to perform well on similar tasks, such as gradient boosting machines (GBM), support vector machines with different kernels, or deep learning approaches like neural networks.

G. False predictions analysis

i. Decision tree

There can be several reasons for false predictions by our decision tree model:

- Decision trees are prone to overfitting, especially when they are allowed to grow too deep and capture noise or outliers in the training data. This can lead to poor generalization on unseen data.
- If the decision tree model is trained on a small or unrepresentative dataset, it may not capture the underlying patterns and relationships accurately. This can result in poor predictive performance.
- The decision tree model may not have access to relevant features that are important for making accurate predictions. Inadequate feature selection or extraction can limit the model's ability to capture the true patterns in the data.
- If the dataset used for training the decision tree model is imbalanced, meaning one class dominates the others, the model may struggle to accurately predict the minority class. This can lead to biased predictions and lower accuracy.

To improve the accuracy of the decision tree model, we can apply pruning techniques to prevent overfitting. Pruning involves limiting the size or complexity of the decision tree by setting thresholds on parameters such as maximum depth, minimum samples per leaf, or maximum leaf nodes. This helps in generalizing the model and reducing overfitting. We can Evaluate and select relevant features that have a stronger impact on the target variable. Feature engineering techniques such as feature selection, dimensionality reduction, or creating new informative features can enhance the model's predictive power. Instead of using a standalone decision tree, we can consider ensemble methods such as random forest, which combines multiple decision trees and aggregates their predictions. Ensemble methods can help reduce the variance and bias of individual decision trees, leading to improved accuracy.

If the data is imbalanced, we can use techniques such as oversampling the minority class or undersampling the majority class to balance the dataset. This ensures that the decision tree model receives sufficient information about all classes, leading to better predictions for the minority class. We can experiment with different hyperparameter settings for the decision tree algorithm, such as the maximum depth, minimum samples per leaf, or splitting criteria. Fine-tuning these parameters can optimize the model's performance on the specific dataset.

ii. Random forest

False predictions in our random forest model can occur due to various reasons, including:

- The random forest may not have enough trees to capture the complexity of the underlying data patterns. Increasing the number of trees in the random forest can improve the accuracy by reducing the variance and improving the robustness of the predictions.
- If the random forest model includes weak or irrelevant features, it can negatively impact its performance. Feature selection or engineering techniques can be employed to identify and retain only the most informative and relevant features.
- Imbalanced datasets can lead to biased predictions, where the random forest model may struggle to accurately predict the minority class. Techniques such as oversampling, undersampling, or using class weights can help balance the data and improve the model's ability to correctly classify the minority class.
- Random forests are generally robust against overfitting, but it can still occur if the trees are allowed to grow too deep or if the model is excessively complex. Regularization techniques, such as limiting the maximum depth of the trees or increasing the minimum samples per leaf, can help prevent overfitting and improve generalization.
- The performance of a random forest model can be sensitive to its hyperparameters, such as the number of trees, maximum depth, minimum samples per leaf, or feature subsampling ratio. Conducting a systematic hyperparameter search or using optimization techniques, such as grid search or random search, can help find the optimal set of hyperparameters for improved accuracy.
- Presence of outliers or noisy data points in the training set can affect the random forest model's performance. Identifying and handling outliers or using robust techniques that are less sensitive to outliers can help improve the accuracy.
- We can ensure that the data is properly preprocessed, including handling missing values, scaling features if necessary, and encoding categorical variables appropriately. Proper preprocessing can eliminate potential sources of bias or inconsistencies in the data that can lead to false predictions.

By considering the above notations, we can address the potential issues and improve the accuracy of the random forest model. Additionally, it is essential to evaluate the model's

performance using cross-validation and assess its generalization ability on unseen data to ensure the reliability of the predictions.

iii. KNN

False predictions in our KNN (K-Nearest Neighbors) model has been happened due to various factors, including:

- The choice of the K value in KNN can significantly impact the model's accuracy. If the K value is too small, the model may overfit the training data and fail to generalize well to unseen data. Conversely, if the K value is too large, the model may introduce more noise from irrelevant neighbors. Conducting a hyperparameter search or cross-validation to find the optimal K value can help improve the accuracy.
- KNN algorithm considers the distance between data points to make predictions. If the feature space contains irrelevant or noisy features, it can introduce misleading distances and affect the model's accuracy. Feature selection or dimensionality reduction techniques can help identify and retain only the most informative features, improving the model's performance.
- KNN can be sensitive to imbalanced datasets, where the majority class may dominate the prediction process. This can lead to biased predictions and lower accuracy for the minority class. Applying techniques such as oversampling, undersampling, or using class weights can help balance the dataset and improve the model's ability to correctly classify the minority class.
- KNN calculates distances between data points based on the feature values. If the features have different scales or units, certain features may dominate the distance calculation, leading to inaccurate predictions. It is crucial to scale the features appropriately, such as using techniques like standardization or normalization, to ensure equal importance for all features.
- KNN can suffer from the curse of dimensionality when the feature space becomes high-dimensional. In high-dimensional spaces, the data becomes sparse, and the notion of distance becomes less reliable. Feature selection, dimensionality reduction (e.g., PCA), or using feature extraction techniques (e.g., LDA) can help mitigate the curse of dimensionality and improve the model's accuracy.

- KNN cannot handle missing data directly, so it requires preprocessing to handle missing values appropriately. Depending on the nature and extent of missing data, techniques such as imputation or deletion can be employed. Careful handling of missing data can help improve the accuracy of the KNN model.
- Outliers can disproportionately affect the KNN model, as they can significantly impact the distance calculation and the selection of nearest neighbors. Identifying and handling outliers appropriately, such as using outlier detection techniques or robust distance metrics, can help improve the model's accuracy.

iv. SVM

False predictions in our SVM (Support Vector Machine) model is occurred due to various factors, including:

- SVM models rely on choosing an appropriate kernel function that maps the data into a higher-dimensional space. Different kernel functions have different properties and are suitable for different types of data. If an incorrect kernel function is chosen, it may not capture the underlying patterns in the data accurately, leading to lower accuracy. Experimenting with different kernel functions, such as linear, polynomial, or radial basis function (RBF), and tuning their hyperparameters can improve the model's performance.
- The regularization parameter (C) in SVM controls the trade-off between maximizing the margin and minimizing the classification errors. If the C value is too small, the model may underfit the data, resulting in high bias and low accuracy. Conversely, if the C value is too large, the model may overfit the data, resulting in low bias but high variance. Conducting a hyperparameter search or using techniques like cross-validation can help find the optimal value of C for improved accuracy.
- SVM models can be sensitive to imbalanced datasets, where the majority class may dominate the model's training process. This can lead to biased predictions and lower accuracy for the minority class. Employing techniques such as oversampling, undersampling, or using class weights can help address the class imbalance and improve the model's ability to correctly classify both classes.
- SVM models are sensitive to the scaling of features. If the features have different scales or units, certain features may dominate the decision boundaries, leading to inaccurate

predictions. It is important to scale the features appropriately, such as using techniques like standardization or normalization, to ensure equal importance for all features and improve the model's accuracy.

- Noise or outliers in the data can adversely affect the SVM model's performance. Outliers can disrupt the construction of optimal decision boundaries, leading to misclassifications. It is important to handle outliers appropriately, either by removing them or using robust SVM variants that are less affected by outliers.
- SVM models heavily rely on the features provided. Insufficient feature engineering, such as not capturing relevant information or failing to extract discriminative features, can limit the model's accuracy. Conducting a thorough feature analysis, including feature selection, transformation, or incorporating domain knowledge, can help improve the model's performance.
- SVM models may require a sufficient amount of labeled data to accurately learn the decision boundaries and generalize well. If the training dataset is small or lacks diversity, the model may struggle to capture the underlying patterns, resulting in lower accuracy. Collecting more representative and diverse data or using techniques like data augmentation can help improve the model's accuracy.

H. Most and least frequent genres

Here, we ignore the popularity column and instead of that, consider the genre column. In the genre column, we find the most frequent and least frequent genres. First, we split the genres and create a list of all genres:

```
all_genres = df["genre"].str.split(",").explode().str.strip()
```

Then, we will find the most frequent genre:

```
most_frequent_genre = all_genres.mode().values[0]
```

And the least frequent genre:

```
least_frequent_genre = all_genres.value_counts().idxmin()
```

Through the following Figure, we can see the frequency of all the genres.

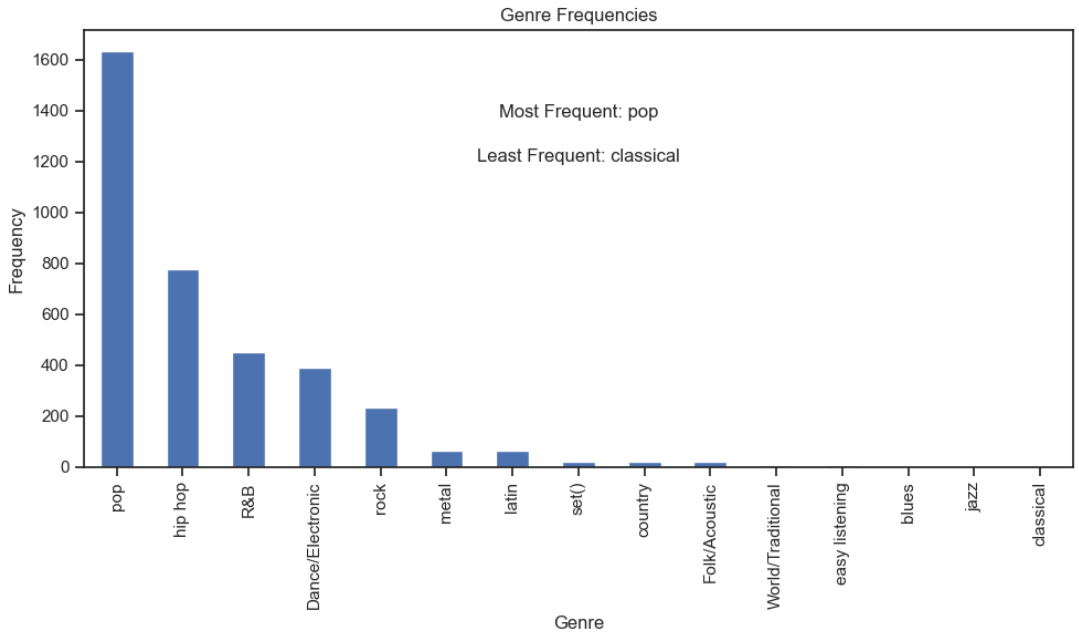

Fig 19. Genre frequencies

From the results, we can see that the pop is the most frequent genre and the classical is the least frequent genre.

Using the following code, we count the number of genres for each song:

```
genre_counts = df["genre"].str.split(",").apply(lambda x: len(x))
```

Then, we find the maximum number of genres:

```
max_num_genres = genre_counts.max()
```

Finally, we can get the songs with the most number of genres:

```
songs_with_max_genres = df[genre_counts == max_num_genres]
```

We will see from the results that the most number of genres for a song is 4. The songs with the most number of genres are given in Table 11.

Table 15. Songs with the most number of genres

Row	Artist	Genre
324	Baby Bash	hip hop, pop, R&B, latin
339	Simply Red	rock, R&B, Folk/Acoustic, pop
431	Baby Bash	hip hop, pop, R&B, latin
525	Baby Bash	hip hop, pop, R&B, latin
797	J. Holiday	hip hop, pop, R&B, Dance/Electronic
886	J. Holiday	hip hop, pop, R&B, Dance/Electronic
1135	M83	rock, pop, metal, Dance/Electronic
1419	Tinashe	hip hop, pop, R&B, Dance/Electronic

To calculate the probability of a song being placed in other genres using Bayes' Theorem, we use the following principle:

$$P(A|B) = (P(B|A) * P(A)) / P(B)$$

Where:

- P(A|B) is the posterior probability of A given B
- P(B|A) is the conditional probability of B given A
- P(A) is the prior probability of A
- P(B) is the prior probability of B

I wrote a code that will save the probabilities of each song being placed in each genre directly to a CSV file named 'song_probabilities.csv'. The resulting CSV file will have rows representing songs and columns representing genres, with the corresponding probability values. For example, the probability of a song being placed in R&B genre is 0.774.

I. Genre prediction using decision tree

In this part, we will split the data into an 80% training set, 10% testing set, and 10% validation set using the train_test_split function. We will then train a decision tree classifier on the training set and predict the genre of songs in the testing set. Finally, we will calculate and print the accuracy and some other scores of the model's predictions.

First, we will split the data into features (X) and the target variable (y). Then, we will perform one-hot encoding on the genre column. After that, we split the data into training, testing, and validation sets using the following code:

```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,
                test_size=0.2, random_state=42)
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test,
                test_size=0.5, random_state=42)
```

Now, we initialize the decision tree classifier:

```
clf = DecisionTreeClassifier()
```

And train the decision tree classifier on the training set:

```
clf.fit(X_train, y_train)
```

Finally, we can predict the genre of songs in the testing set:

```
y_pred = clf.predict(X_test)
```

To create the confusion matrix and print the accuracy, we can use the confusion_matrix and accuracy_score functions from scikit-learn. The confusion matrix of genre prediction using decision tree algorithm is given in Figure 20.

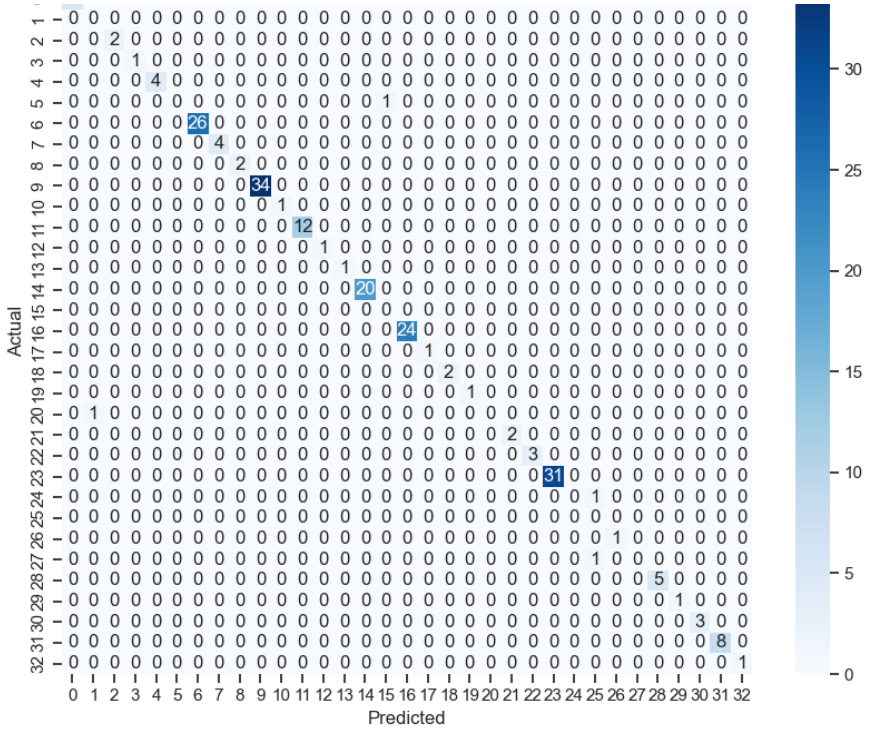

Fig 20. Confusion matrix of genre prediction using decision tree algorithm

Table 16. Decision tree algorithm scores for genre predictions

Score	Value
Accuracy	**0.980**
Precision	0.980
Recall	0.980
F1-score	0.980
Error rate	0.020

The decision tree classifier model we used to predict the genre of songs achieved high scores for accuracy, precision, recall, and F1 score, with a value of 0.98 for each metric. These scores indicate that the model performed exceptionally well in classifying songs into their respective genres. Accuracy measures the overall correctness of the predictions made by the model. With an accuracy score of 0.98, it means that the model accurately classified 98% of the songs correctly. This suggests that the model is highly reliable in determining the genre of songs. Precision is a measure of the model's ability to correctly identify positive predictions (correctly predicted genres) among all the predicted positive instances. A precision score of 0.98 indicates that 98% of the songs predicted as a particular genre were actually classified correctly. This

suggests that the model has a low rate of false positives and is precise in identifying the correct genre. Recall, also known as sensitivity or true positive rate, measures the model's ability to correctly identify positive instances (correct genre) among all the actual positive instances. A recall score of 0.98 implies that the model identified 98% of the songs with the correct genre out of all the songs that should have been classified as that genre. This indicates that the model has a low rate of false negatives and is capable of capturing the majority of the positive instances. F1-score is the harmonic mean of precision and recall, providing a balanced evaluation of the model's performance. An F1-score of 0.98 suggests that the model achieved a high balance between precision and recall, combining both metrics effectively.

Overall, the obtained scores of 0.98 for accuracy, precision, recall, and F1-score indicate that the decision tree classifier model is highly accurate, precise, and reliable in predicting the genre of songs. It has demonstrated excellent performance in correctly classifying songs into their respective genres.

Thanks for your Time