**Introduction:**

This comprehensive documentation encompasses a meticulous analysis conducted on an extensive dataset that includes features related to songs on the **Spotify** platform. The primary objective of this project is to predict the playlist genre using a set of features available in the dataset. The project embarked on a comprehensive journey through several stages, starting with a deep exploration of the dataset, progressing through the visualization phase, preprocessing, modeling, and culminating in the comprehensive evaluation of various classification techniques.

**Data Exploration:**

The initial stage encompassed a thorough exploration of the dataset's details. This comprehensive process included an in-depth examination of data types, structure, and overall integrity. Notably, the dataset exhibits a well-organized format, primarily composed of numerical columns alongside a subset of categorical attributes. The exploration phase revealed fundamental insights into variable distributions, intricate interrelationships between diverse features, and the early identification of prevailing patterns within the dataset. Crucial visualizations, such as track popularity levels and indications of the general classification type for playlists containing the track, played a significant role. These visualizations provided valuable context and aided in understanding the inherent characteristics of the dataset. Each column represents a different characteristic, with explanations provided for key features like track popularity, playlist genre, dance ability, energy, and more. This general explanation for each column serves as a foundation to identify trends and relationships between various variables during the data exploration phase.

**Data Preparation:**

**Cell 1: Importing Required Libraries:**

Purpose: Importing the necessary libraries for data analysis and visualization.

Description: This code cell imports the Style class from the colorama library. The Style class allows for the customization of text styles in the console output, such as brightness, colors, and background. The imported Style class is used later in the code.

**Cell2: Displaying DataFrame info():**

Propose: `df1.info()` is a method in Python, specifically for Pandas DataFrames. When you execute this line, it provides information about the DataFrame `df1`. Here's a proposed description for this line:

Description: The `df1.info()` method is used to obtain concise information about the DataFrame `df1`. It provides a summary of the DataFrame, including the data types of each column, the number of non-null values, and the memory usage. This information is valuable for understanding the structure of the DataFrame, identifying any missing or null values, and assessing the overall data quality. The output includes the total number of entries, the memory usage, and a summary of each column's data type and non-null counts, offering a quick overview of the dataset's composition. This method is a helpful tool in the initial stages of data exploration and preparation.

**Cell3: : Displaying DataFrame duplicated:**

Purpose: The code `df1.duplicated ().sum()` is used to count the number of duplicate rows in the DataFrame `df1`. Here's a brief description:

Description: The `df1.duplicated ()` function identifies duplicate rows in the DataFrame `df1`. It returns a boolean Series where each element is `True` if the corresponding row is a duplicate, and `False` otherwise. The `.sum ()` method is then applied to the boolean Series to count the total number of `True` values, representing the count of duplicate rows in the DataFrame. This line of code is useful for quickly assessing the presence of duplicate records in the dataset, providing insight into potential data quality issues or redundancies.

**Cell4: Displaying DataFrame missing values :**

Purpose: The code `df1.isna().sum()` is used to count the number of missing (NaN) values in each column of the DataFrame `df1`. Here's a brief description:

Description: The `df1.isna()` method returns a DataFrame of the same shape as `df1` where each element is `True` if the corresponding element in `df1` is a missing value (NaN), and `False` otherwise. The `.sum () ` method is then applied to this boolean DataFrame, resulting in a Series that contains the sum of missing values for each column. The output of this line is a Series where the index represents the column names of `df1`, and the values represent the count of missing values in each respective column. This information is valuable for understanding the data quality and identifying columns with missing data that may require handling during data preprocessing.

**Cell5: Displaying DataFrame dropping missing values:**

Purpose: The code `df1 = df1.dropna()` is used to remove any rows containing missing values (NaN) from the DataFrame `df1`. Following this, `df1.describe(include="all")` is applied to provide a summary statistics overview of the cleaned DataFrame. Here's a proposed description for these lines

Description: 1.Removing Rows with Missing Values:The line `df1 = df1.dropna()` is responsible for eliminating rows in the DataFrame `df1` that contain missing values (NaN). This operation helps enhance the data quality by ensuring that the subsequent analysis and modeling are performed on a dataset with complete information, contributing to more accurate results,2. Generating Summary Statistics: After the removal of rows with missing values, `df1.describe(include="all")` is utilized to generate a comprehensive summary statistics overview of the cleaned DataFrame. The `describe` method provides various statistics for each column, including count, mean, standard deviation, minimum, 25th percentile, median (50th percentile), 75th percentile, and maximum. The parameter `include="all"` ensures that both numerical and categorical columns are included in the summary, offering insights into the central tendency and spread of the data, as well as the distribution of categorical variables.These lines collectively serve the purpose of data cleaning by eliminating missing values and providing a statistical summary of the cleaned dataset for further analysis.

**Cell6: Displaying DataFrame dropping columns:**

Purpose: The provided code snippet is used to drop specific columns from the DataFrame `df1`. Here's a proposed description for these lines:

Description: 1. Specifying Columns to Drop:The variable `columns\_to\_drop` is a list containing the names of columns to be removed from the DataFrame `df1`. In this case, the columns specified for removal are 'track\_id', 'playlist\_id', 'track\_album\_id', and 'duration\_ms'. The decision to drop these columns might be based on the analysis requirements or the nature of the data.

2. \*\*Dropping Specified Columns: The line `df1 = df1.drop(columns=columns\_to\_drop, axis=1)` utilizes the `drop` method to eliminate the specified columns from the DataFrame. The `columns` parameter is set to the list of columns to be dropped, and `axis=1` indicates that the operation is applied along columns. The modified DataFrame is then assigned back to `df1`.

3. \*\*Resulting DataFrame:After executing these lines, `df1` now represents the original DataFrame with the specified columns removed. This operation is often employed during the data preprocessing stage to tailor the dataset to the specific needs of the analysis or modeling tasks by excluding irrelevant or redundant informationIn summary, these lines of code serve the purpose of refining the DataFrame by removing certain columns, contributing to a more focused and relevant dataset for subsequent analysis.

**Cell7: Displaying DataFrame converting a date column:**

This code segment is focused on converting a date column in the DataFrame `df1` to years and then calculating and displaying the range of track album release years. Here's a proposed description:

Description:1. \*\*Converting Date to Years:\*\*

Purpose: The line `df1['track\_album\_release\_date'] = pd.to\_datetime(df1['track\_album\_release\_date'], errors='coerce')` uses the `pd.to\_datetime` function to convert the 'track\_album\_release\_date' column to a datetime format. The `errors='coerce'` parameter is set to handle any errors during conversion by replacing them with NaT (Not a Time).

2. \*\*Extracting Year and Creating a New Column:\*\*

The line `df1['track\_album\_release\_year'] = df1['track\_album\_release\_date'].dt.year.astype('Int64')` extracts the year from the datetime column and creates a new column named 'track\_album\_release\_year'. The `.astype('Int64')` is used to handle nullable integer types.

3. \*\*Calculating and Displaying Release Year Range:\*\*

The code calculates the minimum and maximum values of the 'track\_album\_release\_year' column using `min()` and `max()`. The resulting range is then printed to the console with the line `print(f"The range of track album release years is from {min\_release\_year} to {max\_release\_year}.").`

In summary, this code prepares the 'track\_album\_release\_date' column for analysis by converting it to years and then calculates and displays the range of track album release years in the DataFrame `df1`.

**Cell8: Displaying DataFrame converting into scaling :**

Purpose: The provided code segment involves feature scaling using the Min-Max scaling technique. Here's a proposed description:

Description:1. \*\*Importing the Required Library:\*\*

The line `from sklearn.preprocessing import MinMaxScaler` imports the `MinMaxScaler` class from the scikit-learn library. This class is used for scaling numerical features to a specified range, commonly between 0 and 1.

2. \*\*Initializing the Scaler:\*\*

The line `scaler = MinMaxScaler()` initializes an instance of the `MinMaxScaler` class. This scaler will be used to transform the specified columns in the DataFrame.

3. \*\*Scaling Selected Columns:\*\*

The line `df1[['track\_popularity','loudness','key','tempo','instrumentalness']] = scaler.fit\_transform(df1[['track\_popularity','loudness','key','tempo','instrumentalness']])` applies the Min-Max scaling transformation to the selected columns in the DataFrame `df1`. The specified columns ('track\_popularity','loudness','key','tempo','instrumentalness') are scaled to the range [0, 1].

4. \*\*Note on 'instrumentalness':\*\*The comment `# values of instrumentalness must be between 0 and 1` emphasizes that the 'instrumentalness' column is specifically being scaled to ensure that its values fall within the [0, 1] range.In summary, this code segment performs Min-Max scaling on a subset of columns in the DataFrame `df1`, normalizing the values to a standardized range, which is a common preprocessing step in machine learning workflows.

**Cell8:Use Pycaret:**

Purpose:The line `# Use Pycaret` appears to be a comment indicating the intention to use the PyCaret library for some specific purpose. However, the provided code doesn't include the actual PyCaret function calls or methods. Below is a proposed description based on the comment:

Description:1. \*\*Comment Indicating PyCaret Usage:\*\*The line `# Use Pycaret` is a comment in the code, indicating the intention to utilize the PyCaret library for a specific data science or machine learning task.

2. \*\*PyCaret Overview:\* PyCaret is an open-source, low-code machine learning library in Python that simplifies the end-to-end machine learning process. It provides a high-level interface for automating common tasks such as data preprocessing, model selection, hyperparameter tuning, and model evaluation. PyCaret is designed to make machine learning accessible to non-experts and to streamline the workflow for experienced practitioners.

3. \*\*Potential Actions Using PyCaret:\*\*While the specific PyCaret functions or methods are not provided in the code, common tasks performed with PyCaret include setting up the environment, loading data, setting up the target variable, and using PyCaret's functions like `setup()`, `compare\_models()`, and `tune\_model()` to automate various aspects of the machine learning pipeline.For example, a typical PyCaret workflow might involve setting up the environment with `setup()`, comparing multiple models with `compare\_models()`, selecting the best-performing model, and then fine-tuning the model parameters with `tune\_model()`.In summary, the comment suggests an intention to leverage PyCaret for machine learning tasks, but the specific actions performed using PyCaret are not detailed in the provided code snippet.

**Classification Models and Evaluation:**"

I utilized several models for classification, including:

- \*\*Logistic Regression:\*\*

- Training Accuracy: 0.77

- Testing Accuracy: 0.78

- \*\*Decision Trees:\*\*

- Training Accuracy: 0.99

- Testing Accuracy: 0.94

- \*\*Random Forest Classifier:\*\*

- Training Accuracy: 0.99

- Testing Accuracy: 0.95

- \*\*Naive Bayes:\*\*

- Training Accuracy: 0.95

- Testing Accuracy: 0.95

- \*\*Gradient Boosting Classifier:\*\*

- Training Accuracy: 0.96

- Testing Accuracy: 0.95

- \*\*K-Neighbors Classifier:\*\*

- Training Accuracy: 0.94

- Testing Accuracy: 0.90

The best-performing models are:

- \*\*Support Vector Classifier (SVC):\*\*

- Training Accuracy: 0.95

- Testing Accuracy: 0.95

- \*\*Light Gradient Boosting Machine:\*\*

- Training Accuracy: 0.98

- Testing Accuracy: 0.95

This summary provides an overview of the training and testing accuracies achieved by each model, with the SVC and Light Gradient Boosting Machine standing out as the top performers."