**Project and code explanation  
  
Audio Data Analysis using deep learning / Audio Emotion Analysis and Emotion Detection Using Deep Learning.  
  
Script Data.py**

* This Python script, named **data.py**, captures video frames from a camera, extracts facial and hand landmarks using the MediaPipe library (holistic model), and saves the landmark data to a NumPy array in a .npy file. This data is often used for training machine learning models.   
    
  The code will continuously capture video frames from the camera and display them in a window.
* The window will show the detected facial, left hand, and right hand landmarks.
* The current data size (number of data points collected) will be displayed on the window.
* The code will save the collected landmark data to a .npy file with the specified name (e.g., "data.npy").
* The printed shape of the NumPy array will indicate the dimensions of the data collected, typically in the form of (number of data points, number of features).

The **.npy** file stores structured data, which consists of lists of relative coordinates for each frame, as extracted from the facial and hand landmarks. The structure of the data is determined by the code's logic for processing the MediaPipe landmarks, normalizing them, and then appending them to the **X** list.

Typically, the data stored in the **.npy** file will be in the form of a NumPy array with dimensions representing the number of frames collected (data points) and the features extracted from each frame (which include the relative x and y coordinates of facial and hand landmarks).

**Script trainingemotiondata.py**

The script **trainingemotiondata.py** is responsible for reading data from **.npy** files, preparing it for classification, building and training a neural network model to classify emotions, and then saving the trained model and labels associated with emotions.

1. Define the architecture of the neural network model:
   * Input layer with the shape of the input data.
   * Two hidden layers with 512 and 256 neurons, both using ReLU activation functions.
   * Output layer with the number of neurons equal to the number of emotion categories and using a softmax activation function.
2. Compile the model with the "rmsprop" optimizer, categorical cross-entropy loss, and accuracy as a metric.
3. Train the model using the prepared data and labels with 50 epochs.
4. Save the trained model as "detectormodel.h5."
5. Save the emotion labels in a **.npy** file as a NumPy array.

The ".h5" extension refers to Hierarchical Data Format version 5, commonly known as HDF5. HDF5 is a data file format designed to store and organize large amounts of data in a structured and efficient manner. It is widely used in scientific and engineering applications for handling complex data structures, including numerical data, metadata, and more.

In the context of deep learning and machine learning, ".h5" files are often used to store trained neural network models. Deep learning frameworks like TensorFlow and Keras, which are built on top of Python, frequently save and load models in the HDF5 format. These files can store the model's architecture, weights, and configuration, making it easy to save and reload models for later use or deployment.

The **.h5** file (in this case, "detectormodel.h5") will save the trained neural network model. The data saved in the **.h5** file typically includes the following:

1. **Model Architecture**: The structure of the neural network, including the type and configuration of layers, activation functions, and the flow of data through the network.
2. **Model Weights**: The learned parameters (weights and biases) of the model's layers. These are the values that were adjusted during training to minimize the loss function.
3. **Model Configuration**: The configuration of the model, such as optimizer settings and compilation information, to reproduce the model's state at the time of saving.

**Script face\_detector.py**Script is designed to perform real-time emotion detection and music recommendation based on computer vision and deep learning

1. **Video Stream**: The script captures video frames from the default camera (camera index 0) and displays them in a window.
2. **Emotion Detection**: It uses the MediaPipe holistic model to detect facial and hand landmarks in the video frames.
3. **Emotion Prediction**: The code loads a pre-trained neural network model for emotion detection from "detectormodel.h5" and a set of emotion labels from "emotionlabels.npy." It uses these to predict the emotion of the person in the video frame based on the extracted facial and hand landmarks.
4. **Music Recommendation**: Based on the detected emotion, the script recommends a song. The song recommendations are loaded from a CSV file named "songemotions.csv," which contains mappings between emotions and songs. The recommendations are chosen based on the detected emotion. The recommended song is printed to the console.
5. **Display**: The detected emotion is displayed on the video frame, so you can see in real-time what emotion the model predicts.
6. **Keyboard Interaction**: The script continues to run until you press the 'Esc' key (key code 27), at which point it closes the window, releases the camera, and terminates the script.
7. **Console Output**: The script prints the detected emotion and the recommended song to the console.

**Script Emotion-Recognition.py**

The script "emotion recognition.py" performs k-NN (k-Nearest Neighbors) classification on a dataset of music features to classify pieces of music into one of four emotions.

1. **Data Preparation**:
   * The code loads music feature data from a CSV file named "Emotion\_features.csv."
   * It extracts the features as the input data, which are stored in the **feature** variable. These features are used for music classification.
   * The features are scaled to be in the range [0, 1] to ensure uniformity.
2. **Data Splitting**:
   * The data is split into training and testing sets using **train\_test\_split** from scikit-learn (sklearn). The training set is used to train the k-NN classifier, and the testing set is used to evaluate its performance.
   * The size of the testing set is set to 20% of the entire dataset, and a random seed is used for reproducibility.
3. **k-NN Classification**:
   * The script performs k-NN classification for different numbers of neighbors (k values) ranging from 1 to 10.
   * For each value of k, a k-NN classifier is trained on the training data.
   * The classifier is then used to make predictions on the testing data.
   * The accuracy score of the predictions is calculated and stored in the **result** list.
4. **Plotting**:
   * The code generates a plot to visualize the results. The x-axis represents the number of neighbors (k values), ranging from 1 to 10. The y-axis represents the accuracy score.
   * The plot shows how the accuracy of the k-NN classifier changes as the number of neighbors (k) varies.
   * The resulting plot is saved as "1-fold 10NN Result.png" and displayed.

The output of the script "emotion recognition.py" is a plot that visually represents the accuracy of a k-NN (k-Nearest Neighbors) classifier for classifying music into different emotions. Here's an explanation of the plot's key components and what it conveys:

1. **X-Axis** (kNN Neighbors): This represents the number of neighbors (k values) used by the k-NN classifier. It starts from 1 and goes up to 10, indicating the range of k values tested in the script.
2. **Y-Axis** (Accuracy Score): This axis represents the accuracy score achieved by the k-NN classifier for each value of k. The accuracy score is a measure of how well the classifier is performing. It is expressed as a percentage, where higher values indicate better classification performance.
3. **Title** (kNN Classifier Results): The title of the plot indicates that it's showing the results of the k-NN classifier. It informs the viewer about the purpose of the plot.
4. **Plot Line**: The plot itself consists of a line graph. Each point on the graph represents the accuracy score achieved by the k-NN classifier for a specific value of k.
   * The plot line starts at k=1 and progresses through increasing values of k, up to k=10.
   * As k increases, the plot line indicates how the accuracy score changes. It might rise, fall, or reach a plateau.
5. **Limits**: The plot has limits set for both the x-axis and y-axis. The x-axis limit is from 0 to the maximum k value tested (in this case, 10). The y-axis limit is from 0 to 100, representing the percentage scale of accuracy scores.

**Script Emotion-Recognition-RandomSeed.py**The script "Emotion-Recognition-RandomSeed.py" performs k-NN (k-Nearest Neighbors) classification on a music dataset, classifying pieces of music into one of four emotions. However, unlike the previous script, this script performs k-NN classification for different numbers of neighbors (k values) and various random splits of the data. The key feature of this script is the visualization of the accuracy scores for different scenarios

1. **Data Preparation**:
   * The script loads music feature data from a CSV file named "Emotion\_features.csv."
   * It extracts the features as the input data and labels for emotions. These features are stored in the **feature** variable.
2. **Plot Initialization**:
   * The script sets up the plot for visualization. It uses the "ggplot" style.
   * An empty list **result** is initialized to store accuracy scores.
   * Lists **xlabel** and **color** are initialized to store x-axis values (k values) and colors for plotting, respectively.
   * The **colors** list defines the colors to be used in the plot.
3. **Random Seed Loop**:
   * The script enters a loop that iterates through different random seeds, ranging from 1 to 10. Each random seed represents a different random split of the data.
4. **k-NN Classification for Different k Values**:
   * Inside the loop, the data is randomly split into training and testing sets using the specified random seed.
   * Another loop iterates through different values of k (neighbors), ranging from 1 to 9.
   * For each combination of random seed and k value, a k-NN classifier is trained, predictions are made, and the accuracy score is calculated and stored in the **result** list.
   * The **xlabel** list is updated with the current k value, and the **color** list is updated with a color based on the **colors** list.
5. **Plotting**:
   * The script generates a scatter plot to visualize the accuracy scores for different k values. Each point on the plot represents an accuracy score.
   * The x-axis represents k values (neighbors), ranging from 1 to 9.
   * The y-axis represents the accuracy score, which is a value between 0 and 1.
   * Different colors are used to represent different random seeds, allowing you to see how the performance varies across different random splits of the data.
6. **Saving and Displaying the Plot**:
   * The resulting plot is saved as "10-folds kNN Result.png" in the working directory.
   * The plot is also displayed on the screen for visualization.

The output of the script "Emotion-Recognition-RandomSeed.py" is a scatter plot that visualizes the accuracy of a k-NN (k-Nearest Neighbors) classifier for classifying music into different emotions under different scenarios. Here's an explanation of the key components and what the plot conveys:

1. **X-Axis** (kNN Neighbors): This represents the number of neighbors (k values) used by the k-NN classifier. It starts from 1 and goes up to 9, indicating the range of k values tested in the script.
2. **Y-Axis** (Accuracy Score): This axis represents the accuracy score achieved by the k-NN classifier for each value of k. The accuracy score is a measure of how well the classifier is performing. It is expressed as a decimal between 0 and 1, where 1 indicates perfect classification.
3. **Title** (kNN Classifier Results): The title of the plot indicates that it's showing the results of the k-NN classifier. It informs the viewer about the purpose of the plot.
4. **Scatter Points**: The plot consists of multiple scatter points, each representing the accuracy score achieved by the k-NN classifier for a specific value of k. These points are spread along the x-axis (k values) and rise or fall on the y-axis based on the accuracy achieved.
5. **Colors**: The colors of the scatter points represent different random seeds. Each color corresponds to a different random split of the data, allowing you to see how the classifier's performance varies under different random data splits.
6. **Legend**: While there is no legend explicitly mentioned in the script, it is common practice to provide a legend to identify the colors with their respective random seed values.
7. **Limits**: The plot has limits set for both the x-axis and y-axis. The x-axis limit is from 0 to the maximum k value tested (9), and the y-axis limit is from 0 to 1, representing the range of accuracy scores.
8. **Emotion-Recognition.py**:
   * **Purpose**: The primary purpose of "Emotion-Recognition.py" is to classify music into one of four emotions using a k-NN classifier. The focus is on finding the optimal k value for classification by varying k from 1 to 10.
   * **Variation in Data Splits**: This script does not involve different random splits of the data. It uses a single split of the data, and the accuracy scores are calculated for different values of k.
   * **Output Visualization**: It visualizes the accuracy scores for different k values in a single plot, helping you identify the optimal k value for the given dataset.
9. **Emotion-Recognition-RandomSeed.py**:
   * **Purpose**: The main purpose of "Emotion-Recognition-RandomSeed.py" is also to classify music into emotions using a k-NN classifier, but it focuses on evaluating the classifier's performance across different scenarios with various random data splits.
   * **Variation in Data Splits**: This script involves looping through different random seeds (from 1 to 10), resulting in ten different random splits of the data. It aims to assess how the classifier performs under varying training and testing data combinations.
   * **Output Visualization**: The script visualizes the accuracy scores for different k values in a scatter plot, and each color in the plot represents a different random seed. This allows you to see how the classifier's performance varies under different data splits.

**Script genereprediction.py**

The script "genereprediction.py" is designed for audio genre classification using MFCC (Mel-Frequency Cepstral Coefficients) features and KNN (K-Nearest Neighbors) classification. It performs several key tasks, and here's an explanation of each part:

1. **Feature Extraction and Data Preparation**:
   * The script iterates through audio files in a specified directory that contains audio files organized by genres. It extracts MFCC features from the audio files using the Python library **python\_speech\_features**.
   * The extracted features include mean matrix, covariance matrix, and a genre label (numerical representation).
   * These features are serialized and saved to a file named "mydataset.dat" using the **pickle** module.
2. **Loading the Dataset**:
   * The script loads the dataset from "mydataset.dat" and splits it into a training set and a test set.
   * The split ratio is specified as 0.68 (68% training data and 32% test data).
3. **KNN Classification**:
   * The script defines several functions for KNN classification:
     + **getNeighbors**: Calculates the distance between feature vectors and finds the nearest neighbors.
     + **nearestclass**: Identifies the class label based on the nearest neighbors.
     + **getAccuracy**: Calculates the accuracy of the KNN classifier.
4. **KNN Model Evaluation**:
   * The script evaluates the KNN classifier on the test set and calculates the accuracy of the classifier.
   * The accuracy score is printed to the console.
5. **Genre Prediction**:
   * The script makes a genre prediction for a given audio feature (**feature**) using KNN classification.
   * It then prints the predicted genre based on the provided directory structure, associating numerical labels with genre names.
6. **Result Analysis**:
   * The script stores genre labels and their numerical representations in a dictionary (**results**).
   * It prints the predicted genre and its corresponding genre name.

The **mydataset.dat** file in the "genereprediction.py" script is storing serialized data using the **pickle** module in Python. The **.dat** extension is a generic extension used to denote a data file, and it doesn't specify a specific format like other file extensions might (e.g., **.txt** for text files or **.csv** for CSV files). The choice of the **.dat** extension is arbitrary and doesn't imply a standardized format.

1. **File Content**:
   * The **mydataset.dat** file contains serialized Python objects, specifically feature data extracted from audio files, such as mean matrices, covariance matrices, and genre labels.
2. **Serialization with Pickle**:
   * The **pickle** module is used for serialization. Serialization is the process of converting a Python object into a byte stream, which can be stored in a file or transmitted over a network.
   * The script uses **pickle.dump(feature, f)** to serialize each feature (mean matrix, covariance matrix, and genre label) and write it to the file.
3. **Format**:
   * The serialized data is in binary format. Pickle serializes Python objects into a binary format that preserves the object's structure, allowing it to be reconstructed later.
4. **Purpose**:
   * Serialization is used to store the extracted audio features in a compact and efficient format. It allows the script to save and load complex data structures, such as lists of features, in a way that can be easily reconstructed when needed.

**Script ScatterPlotDistribution.py**

The script "ScatterPlotDistribution.py" processes audio feature data, creates scatter plots for each feature, assigns colors to different class labels, saves the plots as image files, and displays them for visualization. The goal is to visualize the distribution of audio features across different classes

1. **Data Loading**:
   * The script loads audio feature data from a CSV file named 'Emotion\_features.csv' using the pandas library. The data includes features and class labels.
2. **Color Mapping**:
   * A color map is defined to associate colors with class labels. In the example, class labels 1, 2, 3, and 4 are mapped to specific colors ('red,' 'green,' 'blue,' and 'orange,' respectively).
3. **Scatter Plot Generation**:
   * The script iterates over each feature in the dataset, creating a scatter plot for each feature's distribution.
   * For each feature:
     + A new figure is created with specified dimensions (12x12 inches).
     + The x-axis represents the class labels, and the y-axis represents the feature values.
     + The plot's title includes the feature name followed by 'Distribution.'
     + A scatter plot is generated where each data point represents an instance, and its color is determined by its class label using the color map.
4. **Saving and Displaying Plots**:
   * Each scatter plot is saved as an image file in a subdirectory named 'Figure/ScatterPlot/' with a filename corresponding to the feature name.
   * After saving, the plot is displayed for visualization using **plt.show()**.
5. **Clearing Figures**:
   * After displaying a plot, the figure is cleared to prepare for the next plot using **plt.clf()**.

**Output Explanation:**

1. **Data Loading and Color Mapping**:
   * The script loads audio feature data and defines a color map for class labels (emotions).
2. **Scatter Plot Generation**:
   * The script iterates over each feature in the dataset and creates scatter plots.
   * Each scatter plot shows the distribution of feature values across different class labels.
3. **Saving and Displaying Plots**:
   * Scatter plots are saved as image files in the 'Figure/ScatterPlot/' subdirectory.
   * Each plot's filename corresponds to the feature name.

**Scatter Plots Explanation:**

For each audio feature, a scatter plot is generated. Let's break down what the scatter plots typically show:

* **X-axis (Horizontal Axis)**: Represents the class labels (emotions).
* **Y-axis (Vertical Axis)**: Represents the values of the audio feature.

Interpretation:

1. **Spread of Values**: The scatter plots show how the values of a specific audio feature are spread across different classes. Each point in the scatter plot represents an instance (sample) in the dataset.
2. **Patterns and Relationships**: Patterns or relationships between the feature and class labels may be visible. For example, certain emotions may show distinct patterns in the feature space, indicating separability.
3. **Color Coding**: Each point is color-coded based on its class label. The color map allows for easy identification of points belonging to different classes.
4. **Feature Distribution**: The plots help visualize the distribution of feature values within each class, providing insights into the characteristics of each emotion.
5. **Potential Outliers or Overlapping**: Examining the scatter plots can reveal outliers or instances where classes overlap. It helps in understanding the discriminative power of the feature.

**Example Interpretation:**

Suppose you have a scatter plot for the "tempo" feature:

* **X-axis**: Represents different emotions (e.g., 1 for happy, 2 for sad, etc.).
* **Y-axis**: Represents the tempo values for each instance.

The scatter plot might show:

* Clear separation between tempo values for different emotions.
* Instances of happy songs (class 1) having higher tempo values compared to sad songs (class 2), for example.

**Script SingleFeaturekNN.py**

The script "SingleFeaturekNN.py" performs kNN classification for audio features related to emotions. It normalizes the features, evaluates the model's performance for different k values, creates plots to visualize the results, and saves these plots as image files. This process is repeated for all the features in a CSV file, allowing you to assess the performance of kNN classification for each feature.

1. **Data Loading**:
   * The script loads audio feature data from a CSV file named 'Emotion\_features.csv' using pandas.
   * It extracts the features related to emotions and the corresponding class labels.
2. **Normalization**:
   * For each feature, the script performs normalization:
     + It subtracts the mean from each feature value.
     + It divides the result by the range (max - min) of the feature values.
3. **kNN Classification**:
   * The script performs kNN classification for each normalized feature.
   * It evaluates the model's performance for different values of k (number of neighbors).
   * For each feature:
     + Training and testing datasets are created using a 80-20% split.
     + A range of k values from 1 to 10 is used for kNN.
     + For each k value, the accuracy of the model is computed.
4. **Plot Generation**:
   * For each feature, a plot is created to visualize the performance of kNN for different k values.
   * The x-axis represents the number of neighbors (k values), and the y-axis represents the accuracy score.
5. **Saving Plots**:
   * The plots are saved as image files in the 'Figure/Individual/Normalized/' subdirectory.
   * The filenames of the saved plots are based on the feature names.
6. **Visualization**:
   * The script displays the plots one by one.

The output of the script "SingleFeaturekNN.py" is a set of scatter plots that visualize the performance of k-Nearest Neighbors (kNN) classification for different audio features related to emotions

1. **Data Loading**: The script loads audio feature data from a CSV file named 'Emotion\_features.csv'. This dataset includes various audio features, each related to different emotions.
2. **Normalization**: For each feature, the script performs the following steps:
   * Normalizes the feature values by subtracting the mean from each value.
   * Divides the result by the range (max - min) of the feature values. This ensures that all feature values are within the same range.
3. **kNN Classification and Evaluation**:
   * For each normalized feature, the script conducts kNN classification to predict the emotion (class) based on that feature.
   * The script evaluates the classification performance using different values of k (number of neighbors) from 1 to 10.
   * It calculates the accuracy of the kNN model for each k value.
4. **Plot Generation**:
   * A plot is created for each feature, illustrating the accuracy of kNN classification for different k values.
   * The x-axis represents the number of neighbors (k values) used in kNN.
   * The y-axis represents the accuracy score, indicating how well the model classifies emotions based on that particular feature

**Script ViolinAndStripSubplot.py**

The script "ViolinAndStripSubplot.py" iterates through each feature in the dataset and creates figures with two subplots: a strip plot and a violin plot. These plots show the distribution of data across different classes, helping to visualize how each feature varies with the class labels. The resulting figures are saved as image files for further analysis and interpretation

1. **Data Loading**: The script loads audio feature data from a CSV file named 'Emotion\_features.csv'. This dataset includes various audio features, each related to different emotions. It also contains class labels and class names.
2. **Iteration Over Features**:
   * The script iterates through each feature in the dataset. For each feature, it creates a figure with two subplots.
   * The two subplots are used to visualize the distribution of the feature data with respect to class labels.
3. **Subplot 1 - Strip Plot**:
   * The first subplot is a strip plot.
   * It uses seaborn's **stripplot** function to create a plot where individual data points are shown as strips along the y-axis.
   * The x-axis represents the class labels, and the y-axis represents the feature values.
   * Jitter is applied to the data points to spread them out and avoid overlap.
4. **Subplot 2 - Violin Plot**:
   * The second subplot is a violin plot.
   * It uses seaborn's **violinplot** function to create a plot that combines a box plot with a kernel density estimation.
   * The x-axis represents the class labels, and the y-axis represents the feature values.
   * The plot provides an overview of the data distribution and the probability density.
5. **Plot Titles**:
   * Each subplot is given a title indicating whether it's a strip plot or a violin plot for a specific feature.
6. **Tight Layout**: The **plt.tight\_layout()** function is used to ensure that the subplots and titles are well-organized within the figure.
7. **Saving Plots**:
   * After creating the subplots, the script saves the figure as an image file.
   * The filenames of the saved plots are based on the feature names, making it easy to identify each plot.
8. **Visualization**:
   * The script displays the figure, and this process is repeated for each feature.

The output of the script "ViolinAndStripSubplot.py" consists of a series of figures, each containing two subplots: a strip plot and a violin plot. These plots are used to visualize the distribution of data across different classes for each audio feature in the dataset

* **Multiple Figures**: The script generates multiple figures, one for each audio feature in the dataset. Each figure is identified by the feature name.
* **Strip Plot (Subplot 1)**:
  + In the first subplot (strip plot), individual data points are displayed along the y-axis.
  + The x-axis represents the class labels (emotions), and each strip represents a data point from a specific class.
  + The strips are jittered, meaning they are slightly displaced horizontally to avoid overlap.
  + This plot provides a clear view of how data points are distributed within different classes for the specific audio feature.
* **Violin Plot (Subplot 2)**:
  + In the second subplot (violin plot), the data distribution is visualized with a combination of a box plot and a kernel density estimation.
  + The x-axis still represents class labels (emotions), while the y-axis represents the feature values.
  + The plot consists of violin-shaped regions that illustrate the probability density of data points within each class.
  + The width of the violin at a particular value on the y-axis indicates the density of data points at that value.
  + The plot also shows summary statistics such as the median, quartiles, and possible outliers.

**Script ViolinStripAndMixPlot.py**

*the code iterates through each feature in the dataset and  
creates three types of plots (strip, violin, and combined)  
to visualize how each feature varies across different classes.  
The resulting plots are saved as image files for further analysis and interpretation.*

The output of the script "ViolinStripAndMixPlot.py" consists of a series of plots that visualize how each audio feature in the dataset varies across different classes (emotions). There are three types of plots for each feature: strip plots, violin plots, and a combination of both.

* **Strip Plots**:
  + The script creates strip plots to show the distribution of individual data points within each class for a specific audio feature.
  + The x-axis represents class labels (emotions), and the y-axis represents the values of the audio feature.
  + Each strip represents a data point from a specific class, and jitter is applied to avoid overlap of data points.
  + The title of each strip plot includes the feature name.
* **Violin Plots**:
  + The script generates violin plots to visualize the distribution of data within each class for the same audio feature.
  + Violin plots combine a box plot and a kernel density estimation to show data density and central tendencies.
  + The x-axis represents class labels, and the y-axis represents feature values.
  + Violin plots provide insights into the distribution of data and display summary statistics.
  + The title of each violin plot includes the feature name.
* **Combined Plots (Violin and Strip)**:
  + The script creates combined plots that overlay strip plots on top of violin plots for the same audio feature.
  + The violin plot provides a general view of data distribution, while the strip plot shows individual data points.
  + The inner area of the violin plot is set to be transparent (light gray) for better visibility.
  + The title of each combined plot includes the feature name.

**Strip Plot**:

* A strip plot, also known as a strip chart or dot plot, is a type of data visualization that displays individual data points along a single axis.
* In a strip plot, each data point is represented as a small dot or marker on the plot, aligned with the feature values on the y-axis.
* Strip plots are particularly useful for visualizing the distribution of data points within different categories or classes.
* They can also include jitter, a small amount of random horizontal displacement, to prevent data points from overlapping when multiple points share the same value.

**Violin Plot**:

* A violin plot is a data visualization that combines aspects of a box plot and a kernel density plot to represent the distribution of data.
* In a violin plot, the data distribution is displayed as a symmetrical, mirrored violin-shaped area. The width of the "violin" at each y-axis value represents the data density.
* A box plot is typically drawn inside the violin plot, showing summary statistics such as the median, quartiles, and potential outliers.
* Violin plots are useful for visualizing data distributions and comparing them across categories or classes.

**Subplot**:

* A subplot is a smaller plot or chart within a larger figure or canvas. Subplots are used to display multiple plots or charts in a single figure, allowing for side-by-side or stacked visualization.
* Subplots are beneficial when you want to compare different aspects of the data, display various features, or create multi-panel visualizations.
* In the context of the provided script, subplots are used to display multiple types of plots (strip plot, violin plot, combined plot) within a single figure for each feature.
* Subplots help organize and present different visualizations together, making it easier to understand the data relationships.

**Script Visualization.py**

This script aims to visualize the values of specific features within the "sad" class of the dataset using a heatmap

1. **Data Loading and Preparation**:
   * The script reads the dataset from the CSV file, which includes various features and their corresponding class labels.
   * It extracts the features, class labels, and feature names from the dataset.
   * A color map is created to assign colors to different class labels (e.g., "red" for class 1, "green" for class 2, etc.).
2. **Normalization**:
   * The script normalizes the features to ensure that they have a consistent scale and range.
   * Feature values are scaled to fall within the range [0, 1].
3. **Feature Selection**:
   * The code selects specific features for analysis. These selected features are stored in **feature\_mean**, **feature\_std**, and **feature\_var**.
4. **Heatmap Generation**:
   * A heatmap is created to visualize the values of the selected features for instances belonging to the "sad" class.
   * The heatmap is generated using the **seaborn** library, specifically **sns.heatmap()**.
   * The selected features (columns) are displayed on the x-axis, while individual instances within the "sad" class are shown on the y-axis.
   * The color intensity in the heatmap represents the values of the selected features, allowing you to identify patterns, variations, or correlations within the "sad" class.
5. **Visualization**:
   * The generated heatmap is displayed using **plt.show()**.
   * The heatmap helps visualize how the selected features vary for instances in the "sad" class, offering insights into the characteristics of this class.

The output of this script is a heatmap that visualizes the values of specific features within the "sad" class of the dataset

* **X-Axis**: The x-axis of the heatmap represents the selected features. Each feature corresponds to a column on the x-axis.
* **Y-Axis**: The y-axis represents individual instances (data points) within the "sad" class. Each row on the y-axis corresponds to a specific data point in the "sad" class.
* **Color Intensity**: The color intensity within the heatmap cells represents the values of the selected features for each instance. The color scale varies from lighter shades (e.g., light green) to darker shades (e.g., dark green).
* **Lighter vs. Darker Colors**: Lighter colors indicate lower feature values, while darker colors indicate higher feature values.
* **Patterns and Variations**: By examining the heatmap, you can identify patterns, trends, and variations in the selected features within the "sad" class. For example, you can look for clusters of dark or light cells, which may suggest that certain features have consistent values within this class or exhibit variations.
* **Correlations**: Heatmaps can also help you identify correlations or relationships between different features. If two features have similar patterns in the heatmap (e.g., both become dark or light at the same time), it may indicate a correlation between those features.
* **Insights**: The heatmap provides insights into how the selected features behave within the "sad" class. It allows you to visually assess the distribution and characteristics of these features for instances associated with this class.
* **Data Exploration**: This type of visualization is particularly useful for data exploration and gaining a deeper understanding of the dataset. It can help identify which features are relevant or distinctive for the "sad" class.

**In these codes, we are classifying music into one specific emotional category. For instance, a song is categorized as "happy," "sad," or "relaxed," but it's not classified into multiple emotional categories simultaneously. This is single-label classification, where each data point (in this case, a song) belongs to a single category or class.**

**Multi-label classification, on the other hand, involves assigning multiple labels or classes to a single data point. For example, a song might be tagged with both "happy" and "upbeat." This is commonly used in situations where a data point can belong to more than one category.**

**codes use both spectral and temporal features for audio data analysis. These features are essential for understanding and classifying audio content based on various characteristics. Here's a breakdown of the usage of these features:**

**1. \*Spectral Features\*: Spectral features capture information related to the frequency domain of audio signals. Some of the common spectral features used in audio analysis include:**

**- Chroma feature (chroma\_stft\_mean, chroma\_stft\_var, chroma\_stft\_std)**

**- Spectral contrast (spectral\_contrast\_mean, spectral\_contrast\_var, spectral\_contrast\_std)**

**- Spectral flatness (spectral\_flatness\_mean, spectral\_flatness\_var, spectral\_flatness\_std)**

**- Mel-frequency cepstral coefficients (MFCCs)**

**- Chroma\_cens**

**These features are extracted from the spectrogram of audio signals and provide valuable information about pitch, timbre, and tonal content.**

**2. \*Temporal Features\*: Temporal features, on the other hand, capture information related to the time domain of audio signals. In audio analysis, temporal features help understand characteristics like rhythm and temporal patterns. While the code snippet doesn't explicitly mention temporal features, they are often used implicitly when considering aspects like tempo and beat patterns.**

**It's important to note that the specific spectral and temporal features used may vary depending on the goals of the analysis. For instance, spectral features are crucial for understanding the tonal content and melody of music, while tempo-related features are temporal in nature and can be important for rhythm analysis.**

**\*MFCC (Mel-Frequency Cepstral Coefficients)\*:**

**- MFCCs are coefficients representing the short-term power spectrum of a sound signal.**

**- They are widely used in audio and speech processing for tasks like speech recognition and audio classification.**

**- The MFCC algorithm involves taking the Fourier transform of the log of the power spectrum of the audio signal, followed by a transformation into the mel-frequency scale.**

**2. \*Chroma Feature\*:**

**- Chroma features are used to represent the pitch content of an audio signal.**

**- They are often used in music information retrieval and genre classification.**

**- Chroma features summarize the energy distribution of pitch classes in an audio signal, typically organized into 12 distinct pitch classes (chroma).**

**3. \*Spectral Contrast\*:**

**- Spectral contrast measures the difference in amplitude between peaks and valleys in the power spectrum of an audio signal.**

**- It is useful for distinguishing between harmonic and percussive components in music signals.**

**4. \*Spectral Flatness\*:**

**- Spectral flatness, also known as tonality, quantifies how noise-like or tonal an audio signal is.**

**- It is calculated by comparing the geometric mean to the arithmetic mean of the power spectrum.**

1. **Softmax Activation Function:**
   * Softmax is an activation function used in the output layer of a neural network for multiclass classification problems.
   * It takes a vector of real numbers as input and transforms them into a probability distribution.
   * Softmax ensures that the sum of the output values is equal to 1, which makes it suitable for classification tasks.
   * The output of the softmax function for each class represents the probability of the input belonging to that class.
   * Mathematically, the softmax function takes a vector of scores (logits) and computes the exponential of each score and normalizes them by the sum of exponentials.
   * The formula for the softmax function for class **i** is: **softmax(x)\_i = exp(x\_i) / sum(exp(x))** for all classes.
2. **ReLU (Rectified Linear Unit) Activation Function:**
   * ReLU is a popular activation function used in hidden layers of neural networks.
   * It replaces all negative values in the input with zero and leaves positive values unchanged.
   * ReLU introduces non-linearity into the network, which helps it learn complex patterns and relationships in the data.
   * It is computationally efficient and prevents the vanishing gradient problem.
   * The formula for the ReLU function is: **ReLU(x) = max(0, x)**.