In this project, I first created a Song struct to include all the elements of each song from the CSV file. I chose f32 for numerical fields because it is suitable for storing data like danceability, acousticness, and energy. Additionally, I created a separate ClusterStats struct to store elements used for the PCA results and clustering. Separating these into two structs helped keep the implementation organized. Within the ClusterStats struct, I implemented a new() function to initialize each cluster with values starting at 0.0. These values are stored as usize to handle dynamic updates after analysis. Next, I wrote an update() function that takes elements from the Song struct as input and updates cluster values based on each song's features. This step is crucial for converting raw data from the file into meaningful numbers that can be stored and processed in Rust. The average() function was then implemented to calculate the averages of each feature across all songs in the dataset. Once the data was read from the CSV file and stored in the Song struct, this function calculated the average values for attributes like danceability, acousticness, and energy.

The main() function begins by reading the input CSV file, top1000_songs.csv. Each row is parsed into a Song struct containing features such as danceability, acousticness, energy, valence, and tempo. These parsed rows are stored in a vector to create a structured representation of the data. After parsing, the data is normalized using the normalize_features() function, which takes the vector of Song structs as input and outputs a 2D vector where each row represents the normalized feature values of a song. Normalization ensures that all features are in a consistent numerical format, making them suitable for PCA and clustering. The normalized data is then passed to the pca() function to perform Principal Component Analysis. This function reduces the data's dimensionality to two, retaining the most significant variance while simplifying subsequent processing. The output is a reduced feature matrix, where each row represents a song in a compressed two-dimensional space.

Next, the reduced feature matrix is passed to the k_means() function for clustering. This function takes the reduced matrix and the desired number of clusters (k) as input and outputs a vector assigning a cluster ID to each song. By grouping songs with similar characteristics, k-means clustering enables analysis of patterns within and across clusters. Finally, the cluster statistics are computed using the ClusterStats struct. For each cluster, the program calculates the averages of features such as danceability, acousticness, energy, valence, and tempo. These statistics are written to an output file, music_clusters.csv, summarizing the characteristics of each cluster. The output includes cluster IDs, average feature values, and the number of songs in each cluster. This final step provides a comprehensive view of the distinct groups in the dataset and offers insights into how musical features and group songs evolve over time.