Week 6 progress

This week, our team made progress in:

- **Data Drive Created**: We set up a drive containing both the CSV textual data and the audio files. This ensures all project data is organized and accessible in one location.
- **CSV Arabic Encoding Fixed**: We corrected encoding issues in the CSV file, allowing the Arabic text (e.g., lyrics and track titles) to display and process correctly.
- **Bonus Task - Audio Files Collected**: we gathered audio files for the songs in our dataset and uploaded them to the drive. These files are named identically to the entries in the CSV's "Track Title" column, ensuring consistency and easy cross-referencing between the textual and audio data.
- **Tokenized Lyrics Column Added**: We updated the dataset with , "Tokenized Lyrics," to the CSV. This column contains the lyrics after thorough preprocessing, which goes beyond basic tokenization. The steps included:
  - Removing punctuation, newline characters, and diacritics.
  - Standardizing Arabic letters for uniformity.
  - Filtering out stop words to retain only meaningful terms.
    This refined dataset is now primed for advanced analysis.
- **Initial Investigation for Keywords**: We began examining the preprocessed lyrics to identify prominent keywords, laying the groundwork for understanding recurring themes or significant terms in George Wassouf's songs.

Next step:

- **Lemmatization**: Explore applying stemming or lemmatization to the Arabic lyrics, reducing words to their root forms (e.g., "كتابة" to "كتب"). This can minimize redundancy and improve the performance of text analysis models.
- **Visualizing the Data**: Create visualizations such as word clouds or frequency plots of the preprocessed lyrics. These tools can provide early insights into recurring words or patterns, guiding our next analytical steps.

  □ Use **bar charts** to display the most frequent words or bigrams (two-word combinations).
  □ Try **heatmaps** to show word co-occurrences or sentiment trends across songs.
  □ Tools like **Plotly** (Python) or **Tableau** can produce interactive, visually appealing outputs to share with stakeholders.

Research Progress :

1. Using Large Language Models (LLMs) for Lyrics Analysis

Sentiment and Emotional Analysis

- **Why It’s Valuable**: LLMs can classify lyrics for sentiment or detect emotions (e.g., joy, sadness, love), adding depth to your analysis.
- **Model**:
  - **AraBERT**: Ideal for Arabic text, pretrained on large Arabic corpora.
  - **Multilingual BERT**: A solid fallback if you need flexibility across languages.
  - Use Python’s **Hugging Face Transformers** library to load these models.
  - For better results, **fine-tune** them on a small labeled subset of your lyrics (e.g., manually tag 50 songs for sentiment). If labeling is challenging, leverage existing Arabic sentiment datasets like **ASTD** to start.
  - For emotion detection, explore adapting the **NRC Emotion Lexicon** (if available in Arabic) or fine-tune AraBERT for multi-label emotion classification.

Embeddings for Deeper Insights

- **Why It’s Valuable**: LLM-generated embeddings capture semantic meaning, enabling tasks like clustering songs by theme or identifying stylistic similarities.
- **Approach**:
  - Extract embeddings using AraBERT or Multilingual BERT via the **Transformers** library.
  - Consider **sentence-level embeddings** (e.g., averaging word embeddings or using Sentence-BERT) to represent entire songs, capturing overarching moods or themes rather than just individual words.
- **Applications**:
  - **Clustering**: Group songs with similar lyrical content using scikit-learn’s clustering algorithms (e.g., K-means).
  - **Similarity Analysis**: Measure lyrical similarity between songs with cosine similarity.
  - **Visualization**: Plot embeddings with **t-SNE** or **UMAP** (via scikit-learn or umap-learn) to reveal thematic clusters visually.

Enhancing TF-IDF with LLMs

- **Why It’s Valuable**: TF-IDF identifies statistically significant words, while LLM embeddings add semantic context—a powerful combination.
- **How to Implement**:
    - Calculate TF-IDF scores using Python’s scikit-learn or R’s tidytext.
    - Pair these scores with embeddings to analyze the meaning of key terms in context.
    - For example, use TF-IDF to find frequent terms like "حب" (love), then use embeddings to explore how its usage varies across songs (e.g., romantic vs. melancholic).
- **Next Level**: Experiment with hybrid models that blend TF-IDF weights with embedding similarities for tasks like song recommendation.