1. Identify and Handle Missing Values

Question:

- Examine the dataset for any missing values. Which columns contain null values?
- How should missing values in the `Views` and `Likes` columns be handled? Should they be filled with a default value, removed, or handled in another way? Justify your approach.

Solution:

Step 1: Identifying Missing Values

- **Issue Identified:** Upon examining the dataset, several columns were found to contain missing values. Notably, the `Views`, `Likes`, `Comments`, and `Official_Video` columns contained a significant number of null values. Additionally, columns such as `Danceability`, `Energy`, and others had sporadic missing values (`NaN`).
- **Action Taken:**
    - The missing values in the `Views` and `Likes` columns were **left as `NaN`** (null) rather than filling them with default values like `0`. This was done because we could not be certain whether the missing values represented zero (no views or likes) or simply unavailable data. Keeping the missing values as `NaN` prevents misleading conclusions.
- **Justification:**
    - **Why not fill with `0`?** Filling missing values with zero could distort the analysis. For instance, if a song's views or likes are simply unavailable, filling the missing values with `0` would incorrectly suggest that no one has viewed or liked that song.
    - **Why keep `NaN`?** By leaving missing values as `NaN`, we ensure that the missing data is properly flagged. Analysts can handle these null values appropriately (e.g., excluding them from average calculations), preventing any distortion of the results.

Step 2: Handling Missing Values in Other Columns

- **Action Taken:** For columns like `Official_Video` and `Comments`, where data was either sparse or not essential to the core analysis, we chose to keep the `NaN` values as they are. This allows the dataset to remain accurate without making assumptions about the missing values.

Final Decision on Missing Values:

- The null values in columns like `Views`, `Likes`, and `Comments` were **preserved as `NaN`** to maintain the dataset's integrity. This ensures that no incorrect assumptions are made about missing data.

Why These Methods Were Applied:

1. **Leaving Missing Values as `NaN`:**
    - This approach ensures that the missing data is visible and flagged for further investigation. It prevents incorrect assumptions and avoids misleading analysis results by not assuming zero values for missing data.
2. **Consistency Across Columns:**
    - By applying the same approach to all relevant columns, we maintain consistency in handling missing values. This ensures that the dataset remains clean and accurate for future analysis.

---

2. Fix Irregularities in Merged Columns

Question:

- The `Spotify_Info` and `Youtube_Info` columns contain merged data separated by delimiters. Split these columns back into their original components.
- What are the original components, and how can you ensure that the split data is clean and accurate?
- After splitting, remove any unnecessary delimiters or prefixes/suffixes that do not belong.

Solution:

Step 1: Splitting the `Spotify_Info` Column

- **Issue Identified:** The `Spotify_Info` column contains Spotify artist URLs, which include both the album and track information.
- **Action Taken:**

    - We split the `Spotify_Info` column using the `/` delimiter to extract the **Spotify Artist ID** (the part of the URL after the last `/`). This allowed us to create a separate column for the artist ID.
    - After splitting, we created two new columns: `Spotify_Album_Link` and `Spotify_Artist_Id`.
- **Justification:**
    - This split ensures that the artist ID is stored separately from the full URL, making it easier to analyze or cross-reference with other datasets.

Step 2: Splitting the `Youtube_Info` Column

- **Issue Identified:** The `Youtube_Info` column contains YouTube video URLs, where each link includes the video ID after `?v=`.
- **Action Taken:**
    - We split the `Youtube_Info` column by using the `=` delimiter to extract the **YouTube Video ID** (the part of the URL after `?v=`). This provided a clean and accurate extraction of the video ID, creating a new column `Youtube_Video_Id`.
- **Justification:**
    - Using the `=` delimiter ensures that the video ID is correctly extracted, making it easier to analyze YouTube video data.

Step 3: Removing Unnecessary Suffixes

- **Issue Identified:** Some entries in the `Youtube_Info` column had unnecessary suffixes like `-GoSong_`, which do not contribute to the data.
- **Action Taken:**
    - After extracting the video ID, any unnecessary suffixes or prefixes were removed using string manipulation techniques to ensure that only the relevant part of the URL or data remains.
- **Justification:**
    - Removing suffixes ensures that the dataset remains clean and consistent, free from extra text that could confuse the analysis.

Final Structure After Splitting:

- After splitting the `Spotify_Info` and `Youtube_Info` columns, the dataset now contains:
    1. `Spotify_Album_Link` and `Spotify_Artist_Id` (from `Spotify_Info`)
    2. `Youtube_Video_Link` and `Youtube_Video_Id` (from `Youtube_Info`)

Why These Methods Were Applied:

1. **Splitting by Delimiters (`/` for Spotify, `=` for YouTube):**
   - These specific delimiters ensure accurate and consistent extraction of the relevant parts of the URLs, like artist and video IDs, without relying on character counts or other less reliable methods.
2. **Removing Unnecessary Text:**
   - By removing irrelevant suffixes, the dataset becomes cleaner and easier to work with, improving the accuracy and reliability of future analysis.

3. Correct Case Sensitivity and Naming Conventions:

Question:

- The column names have inconsistent case sensitivity (some are uppercase, others lowercase). Standardize all column names to follow a consistent format (e.g., all lowercase with underscores).
- Fix any data entries where case sensitivity might affect consistency (e.g., artist names or track titles).
- Ensure that the Artist and Track columns are formatted consistently.

Solution:

Step 1: Standardizing Column Names

- **Issue Identified:** The column names in the dataset have inconsistent case sensitivity, with some being in uppercase and others in lowercase.
- **Action Taken:**
  - To ensure consistency and ease of use, all column names were converted to **lowercase** and spaces were replaced with **underscores**. This standardized naming convention ensures that columns can be easily accessed and used in analysis or programming without any confusion.
- **Example:**
  - Columns like `Track`, `Album_Type`, and `Spotify_Track_Id` were transformed to `track`, `album_type`, and `spotify_track_id`.
- **Justification:**

    - Using **lowercase with underscores** is a widely accepted convention in data science and programming. It makes it easier to work with the dataset in tools like Python, SQL, or Excel, and reduces the risk of errors due to inconsistent column names.

Step 2: Fixing Case Sensitivity in Data Entries (Artist and Track Columns)

- **Issue Identified:** Some entries in the `artist` and `track` columns had inconsistent case sensitivity. For example, artist names and track titles were sometimes written in all lowercase (e.g., `gorillaz` instead of `Gorillaz`).
- **Action Taken:**
    - The data in the `artist` and `track` columns was transformed to **title case**, where the first letter of each word is capitalized. This ensures that artist names and track titles are presented consistently across the dataset.
- **Example:**
    - Artist names like `gorillaz` were converted to `Gorillaz`, and track titles like `plastic beach` were converted to `Plastic Beach`.
- **Justification:**
    - Using **title case** improves the readability and professionalism of the data, particularly in columns like `artist` and `track`, which are often used in reporting and visualizations. It ensures that all names follow a consistent and clean format.

Final Column Names After Cleaning:

- After applying the consistent naming conventions, the column names in the dataset were transformed as follows:
    - `track`, `album`, `album_type`, `spotify_album_link`, `spotify_track_id`, `duration_ms`, `stream`, `key`, `valence`, `liveliness`, `speechiness`, etc.

Why These Methods Were Applied:

1. **Lowercase Column Names with Underscores:**
    - This naming convention is widely used because it avoids issues with case sensitivity when using programming languages and tools. It also makes the dataset easier to understand and work with for anyone reading or analyzing the data.
2. **Title Case for Artist and Track Names:**
    - Capitalizing the first letter of each word in the `artist` and `track` columns ensures that the data is more readable and consistent. This format is especially

important for presentation purposes, where professionally formatted names are essential.

4. Remove or Handle Irrelevant Columns

Question:

- Identify and remove any irrelevant or randomly generated columns that do not provide useful information for analysis. Which columns should be removed, and why?
- If any random data exists in relevant columns, clean or remove those entries.

Solution:

Step 1: Identifying Irrelevant or Randomly Generated Columns

- **Issue Identified:** After reviewing the dataset, three columns—`random_column_1`, `RANDOM_COLUMN_2`, and `unnamed: 0`—were identified as irrelevant. These columns do not contain any meaningful information and appear to be either randomly generated or unnecessary.
- **Action Taken:**
  - The columns `random_column_1`, `RANDOM_COLUMN_2`, and `unnamed: 0` were removed from the dataset. Removing these columns helps streamline the dataset and focus only on the data that is relevant for analysis.
- **Justification:**
  - Removing these irrelevant columns ensures that the dataset is clean and free from unnecessary data. Keeping such columns would only clutter the dataset and could potentially confuse the analysis process. By removing them, we maintain a streamlined and efficient dataset.

Step 2: Ensuring No Random Data in Relevant Columns

- **Issue Identified:** After examining the remaining relevant columns (such as `ALBUM`, `TRACK`, `views`, etc.), no random or irrelevant data entries were found.
- **Action Taken:**
  - No further cleaning was required for random data within relevant columns, as all other entries appeared consistent with the expected data types and content.
- **Justification:**

    - Ensuring that the relevant columns contain only valid data is important for accurate analysis. Since no random data was identified, no further action was necessary.

Why These Methods Were Applied:

1. **Removing Irrelevant Columns:**
    - Columns such as `random_column_1`, `RANDOM_COLUMN_2`, and `unnamed: 0` do not contribute to the analysis and were removed to ensure that the dataset is clean and focused on the relevant data.
2. **Ensuring Valid Data in Relevant Columns:**
    - A thorough check of relevant columns ensured that there was no random data present, maintaining the integrity of the dataset for accurate analysis.

5. Handle Inconsistent Data Types

Question:

- Some columns that should be numeric (e.g., Danceability, Energy) are stored as text. Convert these columns back to numeric format. What steps would you take to identify and fix any issues that arise during this conversion?
- Ensure that all numeric columns are in the correct format and handle any non-numeric values or anomalies.

Solution:

Step 1: Identifying Inconsistent Data Types

- **Issue Identified:** Upon examining the dataset, we found that some columns, such as `Views`, were incorrectly stored as **text** (object type) even though they represent numeric data.
- **Action Taken:**

    - The `Views` column was converted from text to **numeric format** (`float64`). This ensures that the column can be used in numerical calculations without issues.
    - Columns like `Danceability` and `Energy` were already stored as **numeric** (`float64`), so no conversion was required for them.

Step 2: Converting Text Columns to Numeric Format

- **Action Taken:**
    - The conversion was handled using appropriate functions that safely convert text to numeric data types. For example, any non-numeric values or anomalies in the `Views` column (e.g., `NaN`) were handled using the `to_numeric` function, which automatically converts invalid entries to `NaN`.
    - After converting the `Views` column, a check was performed to ensure that the column was successfully transformed to the correct numeric format without any errors.
- **Justification:**
    - Converting text columns to numeric format allows for accurate calculations and avoids errors in analysis. For example, having `Views` as a numeric type allows for proper aggregation, averaging, or other calculations that rely on numeric data.

Step 3: Handling Non-Numeric Values and Anomalies

- **Issue Identified:** In some cases, columns may contain non-numeric values or anomalies (e.g., invalid text entries or empty values).
- **Action Taken:**
    - During the conversion process, any non-numeric values were automatically converted to `NaN` (null) using the `to_numeric` function with the `errors='coerce'` parameter. This ensures that the dataset remains clean and ready for analysis without incorrect entries.
- **Justification:**
    - Handling non-numeric values ensures that the dataset remains consistent and that invalid data does not interfere with the analysis.

Why These Methods Were Applied:

1. **Converting Text Columns to Numeric Format:**
    - Columns that represent numeric data (like `Views`) must be stored in the correct format (`float64`). This allows for accurate calculations and prevents issues in analysis tools or scripts.
2. **Handling Non-Numeric Values:**

  - Using `NaN` for invalid or non-numeric values ensures that the data remains clean and easy to work with. It also avoids errors during calculations, as most tools can handle `NaN` values appropriately.

6. Address and Fix Invalid Data Entries

Question:

- Check the `Views` column for any entries labeled as "invalid_data" or any other incorrect values. Replace these entries and justify your method.
- Ensure that all values in the `Album` column are correctly labeled and that there are no numeric entries or irrelevant data.

Solution:

Step 1: Identifying Invalid Data in the Album Column

- **Issue Identified:** After reviewing the `Album` column, we found that several entries were incorrectly represented by numbers, such as `123456`. These numeric entries are invalid album names and are not present on Spotify.
- **Action Taken:**
  - All invalid numeric entries in the `Album` column were replaced with `NaN` to indicate that the correct album information was either missing or unavailable. This ensures that these invalid entries do not affect any further analysis.
- **Justification:**
  - Replacing invalid numeric entries with `NaN` allows us to handle missing or incorrect data appropriately. By doing this, we avoid misinterpreting these numbers as valid album names, ensuring the integrity of the dataset.

Step 2: Ensuring Valid Data in the Views Column

- **Issue Identified:** After reviewing the `Views` column for any invalid entries like "invalid_data", we did not find any such entries.
- **Action Taken:**
  - No further cleaning was necessary for the `Views` column, as there were no anomalies or invalid values present.
- **Justification:**

    - Ensuring the validity of numeric columns like `Views` is crucial for accurate analysis. Since no invalid data was found, no changes were required for this column.

Final Dataset:

- The `Album` column no longer contains numeric or irrelevant values. Invalid entries were replaced with `NaN`, ensuring that the dataset remains clean and ready for analysis.

Why These Methods Were Applied:

1. **Replacing Invalid Entries with `NaN`:**
    - The numeric values in the `Album` column are not valid album names. By replacing them with `NaN`, we ensure that the dataset accurately reflects missing or incorrect data without making false assumptions.
2. **Ensuring Validity in the Views Column:**
    - A thorough check confirmed that there were no invalid data entries in the `Views` column. This ensures that the dataset is clean and ready for analysis without further changes.

7. Check for and Remove Duplicate Rows

Question:

- Identify and remove any duplicate rows in the dataset. How can you ensure that the remaining data is unique and accurate?

Solution:

Step 1: Identifying Duplicate Rows

- **Issue Identified:** Duplicate rows in the dataset can skew analysis results by artificially inflating data counts or averages. To ensure the accuracy and reliability of the dataset, we must identify and remove any duplicate rows.
- **Action Taken:**
    - Before removing duplicates, we used the **"Keep Duplicates"** feature in Power Query (or a similar data analysis tool). This feature allows us to identify and review all rows that are flagged as duplicates.
    - By selecting all columns in the dataset, we compared the rows to find any duplicates where all column values are the same.
- **Justification:**
    - Reviewing the duplicates before removing them ensures that we fully understand which records are being flagged and allows us to confirm whether they are true duplicates or if they require further review.

Step 2: Removing Duplicate Rows

- **Action Taken:**
    - Once the duplicates were identified and reviewed, we used the **"Remove Duplicates"** option provided by Power Query (or similar tools) to automatically remove all duplicate rows from the dataset.
    - This step ensured that only unique rows remain in the dataset, reducing the risk of double-counting or other errors in the analysis.
- **Justification:**
    - Removing duplicates ensures that the remaining data is both **unique** and **accurate**, which is essential for valid analysis. Duplicate rows can distort results, such as when calculating totals, averages, or other statistical measures.

Step 3: Ensuring Data Accuracy After Removal

- **Issue Identified:** After removing duplicates, it is important to ensure that the dataset remains accurate and that no important data was accidentally removed.
- **Action Taken:**
    - After removing the duplicates, a quick review was conducted to ensure that the remaining data was complete and that no essential data points were missing. This was done by cross-checking row counts and key columns.
- **Justification:**
    - This additional review step ensures that the data remains accurate after duplicates are removed and that no essential rows were unintentionally removed during the process.

Final Dataset:

- After removing duplicates, the dataset now contains only unique rows, ensuring that the analysis can proceed without any distortion caused by duplicate entries.

Why These Methods Were Applied:

1. **Using the "Keep Duplicates" Feature:**
   - This feature allows for a detailed review of potential duplicates before removal. By reviewing flagged rows, we can ensure that we're removing only true duplicates and not accidentally deleting important data.
2. **Removing Duplicates via Power Query:**
   - Using the "Remove Duplicates" option ensures that all duplicate rows are eliminated, leaving the dataset with unique and accurate entries.
3. **Post-Removal Review:**
   - A review after the duplicates are removed ensures that the data remains accurate and complete, preventing the accidental deletion of necessary data.

8. Reorder and Rename Columns for Clarity

Question:

- Reorder the columns in a logical sequence to improve the dataset's readability and usability. What order makes the most sense for this dataset?
- Rename columns where necessary to ensure that their names clearly reflect the data they contain.

Solution:

Step 1: Reordering Columns

- **Issue Identified:** The original column order in the dataset does not follow a logical flow, making it harder to interpret the data. Columns related to general information, platform-specific information, and audio analysis features are scattered across the dataset.
- **Action Taken:**
  - The columns were reordered to follow a more logical and intuitive flow:

        - **General Information:** Start with columns that provide general information about the track, artist, and album.
        - **Platform-Specific Information:** Follow up with columns related to Spotify and YouTube, which provide platform-related insights.
        - **Audio Features and Metrics:** Finish with columns that contain audio analysis data such as tempo, energy, loudness, etc.
- **Justification:**
    - This new ordering improves the readability of the dataset, ensuring that all related columns are grouped logically. It makes it easier for analysts to quickly find and understand the different types of information in the dataset.

New Logical Sequence:

1. **Track Information:**
    - `track`: The name of the track.
    - `artist`: The name of the artist.
    - `album`: The name of the album.
    - `album_type`: The type of the album (e.g., single, album).
2. **Platform-Specific Information:**
    - `spotify_info`: Information specific to Spotify.
    - `youtube_info`: Information specific to YouTube.
    - `description`: The description of the track (often used for YouTube).
    - `duration_ms`: Duration of the track in milliseconds.
3. **Engagement Metrics:**
    - `stream`: Number of streams.
    - `views`: Number of YouTube views.
    - `likes`: Number of YouTube likes.
    - `comments`: Number of YouTube comments.
    - `licensed`: Whether the track is licensed or not.
    - `channel`: The YouTube channel hosting the video.
4. **Key Audio Features and Metrics:**
    - `key`: The musical key of the track.
    - `tempo`: The tempo of the track in beats per minute (BPM).
    - `loudness`: The loudness of the track, measured in decibels.
    - `energy`: A measure of intensity and activity in the track.
    - `danceability`: How suitable the track is for dancing.
    - `acousticness`: A measure of the acoustic quality of the track.
    - `instrumentalness`: How much of the track is instrumental.
    - `liveness`: A measure of the "live" feel of the track.
    - `speechiness`: A measure of how much speech is present in the track.
    - `valence`: A measure of the musical positivity or happiness of the track.

Step 2: Renaming Columns

- **Issue Identified:** Some column names are not clear or consistent, making it difficult to interpret what the data represents. For example, the `DURATION_MS` column should be more descriptive.
- **Action Taken:**
    - Some columns were renamed to improve clarity and consistency:
        1. `DURATION_MS` → `duration_ms`: To make it lowercase and consistent with the naming convention used across other columns.
        2. `valence` → `mood_valence`: To more accurately describe that this column refers to the mood or positivity of the track.
        3. `ENERGY` → `energy`: To standardize the case.
- **Justification:**
    - Renaming the columns ensures that each name accurately reflects the data it contains, improving the dataset's readability and usability. Standardizing column names makes it easier to reference and work with the dataset in tools like Python, SQL, and Excel.

Step 3: Review of Reordering and Renaming

- **Action Taken:**
    - After reordering and renaming the columns, a final review was conducted to ensure that the new structure was logical and consistent with the data's flow.
- **Justification:**
    - Ensuring consistency and clarity in column names helps improve usability and readability, making the dataset more intuitive for analysis.

Why These Methods Were Applied:

1. **Reordering Columns for Logical Flow:**
    - The reordering starts with general track information, followed by platform-specific details (Spotify and YouTube), and ends with the detailed audio metrics. This flow allows for a more structured analysis, making it easier for analysts to work through the data.
2. **Renaming Columns for Clarity:**
    - Renaming ensures that column names clearly describe the data they contain. This reduces confusion and makes the dataset easier to use in various tools and analyses.

THANK YOU