Documentation: ETL pipeline using Apache Airflow and three different data sources (API, CSV file, database)

By Natalia López Gallego

During this workshop, the Spotify dataset was sourced from a CSV file, processed using Python and Airflow to perform necessary transformations, and subsequently loaded into a database. At the same time, the Grammys dataset was uploaded to a database, with Airflow employed to retrieve data from it. Furthermore, Airflow facilitated the extraction of audio features from an audio source via the Recocobeats API. Transformations were applied across all three datasets, which were then merged and loaded into a database as well as the Drive associated with a service account on Google Cloud Platform (GCP).

📂 Data Sources

1. **Grammy Awards Dataset**
    - Source: Kaggle (4,810 rows, 1958–2019).
    - Key Columns: `year`, `category`, `nominee`, `artist`, `winner` ,etc.
2. **Spotify Tracks Dataset**
    - Source: Kaggle (114,000 tracks, 2023).
    - Key Features: `danceability`, `energy`, `valence`, `track_genre` ,etc.
3. **Reccobeats API**
    - Info: https://reccobeats.com/docs/apis/extract-audio-features
    - Extracted Features: `acousticness`, `liveness`, `speechiness`, etc.
    - Scope: 126 tracks (90% coverage of Grammy nominees).

Notebooks

1. `1.0-nlg-load-grammys-to-db.ipynb`

Objective

Load historical Grammy Awards data from CSV into PostgreSQL.

Key Processes

- **Data Ingestion**:
    - Read `the_grammy_awards.csv` (4,810 rows × 10 columns).
    - Validate schema: `year`, `category`, `nominee`, `artist`, `winner`, etc.
- **Database Integration**:

- Use `SQLAlchemy` with connection parameters from `Params`.
- Write to PostgreSQL table `grammys_raw` via `df.to_sql()`.

In [28]:
```
table_name = 'grammys_raw'

try:
    # Write the DataFrame to the database table
    df.to_sql(table_name, con=db_client.engine, if_exists='replace', index=False)
    logging.info(f"CSV data has been successfully loaded into the table '{table_name}'.")
except Exception as e:
    logging.error(f"Failed to load CSV data into the table '{table_name}'.")
    logging.error(f"Error details: {e}")
```

```
2025-03-29 16:03:59,074 - INFO - root - CSV data has been successfully loaded into the table 'grammys_raw'.
```

Evidence of data writing from CSV to database through logging.

```
ws_001=# \d
             List of relations
 Schema |       Name        | Type  | Owner
--------+-------------------+-------+-------
 public | candidates_raw    | table | pg
 public | final_grammy_data | table | pg
 public | grammys_raw       | table | pg
 public | hirees_clean      | table | pg
(4 rows)

ws_001=#
select count(*) from final_grammy_data;
 count
-------
 80193
(1 row)
```

Evidence of data successfully loaded into a PostgreSQL database. Terminal output confirming the data load process. The command `\d` lists the relations in the database, icluding the table `grammys_raw`.

- **Validation**:
  - Compare CSV vs. PostgreSQL table dimensions (`4810 rows × 10 columns`).
  - Execute SQL queries to verify row/column counts.

In [30]:
```
queries = [
    "SELECT COUNT(*) FROM grammys_raw",
    "SELECT COUNT(*) FROM information_schema.columns WHERE table_name = 'grammys_raw'"
]


execute_queries(queries, db_client)
```

```
2025-03-29 16:03:59,106 - INFO - root - Executing query: SELECT COUNT(*) FROM grammys_raw
2025-03-29 16:03:59,114 - INFO - root - Query results:
2025-03-29 16:03:59,116 - INFO - root - (4810,)
2025-03-29 16:03:59,117 - INFO - root - Executing query: SELECT COUNT(*) FROM information_schema.columns WHERE table_name =
'grammys_raw'
2025-03-29 16:03:59,127 - INFO - root - Query results:
2025-03-29 16:03:59,128 - INFO - root - (10,)
```

The output indicates the table has 4810 rows and 10 columns. While the method '.shape' used for the grammy's dataset Dataframe returns the same dimension, so the insertion is consistent.

In [31]:
```
df.shape
```

Out[31]: (4810, 10)

Evidence of queries to verify `grammys_raw` dimensions vs `df` shape. They are the same, indicating a sucessfull migration.

---

2. `2.0-nlg-eda-spotify.ipynb`

Objective

Clean and enrich the Spotify dataset (114k tracks).

Key Processes

- **Data Cleaning**:
  - Drop redundant column `Unnamed: 0`.
  - Remove 450 duplicates and 1 row with null values.
- **Transformations**:
  - **Genre Consolidation**: 114 genres → 54 categories (e.g., `punk-rock` includes `alt-rock`, `garage`).
  - **Feature Engineering**:
    - `mode` (0/1) → `minor/major`.
    - `key` (0-11) → musical notes (e.g., `C`, `C♯/D♭`).
    - Derive `minutes` from `duration_ms`.

Challenges

- **Duplicate Handling**:
  - 450 tracks with identical `track_id` but conflicting `track_genre` (54%) and `popularity` (32%).
  - **Solution**: Resolve via mode (genre) and median (popularity).
- **Missing Data**:
  - Only 1 row with nulls in `track_name`, `artists`, `album_name`.

Insights

- **Distributions**:
  - `duration_ms`: 80% of tracks span 2.5–5.5 minutes.
  - `popularity`: Skewed toward lower values (mean = 33.3).
  - `explicit`: Only 8.7% of tracks marked explicit.

---

3. `3.0-nlg-eda-grammys.ipynb`

Objective

Prepare the Grammy dataset for analysis.

Key Processes

- **Data Cleaning**:
  - Drop `img` (broken URLs), `published_at`, and `updated_at` (irrelevant timestamps).

- **Data Imputation**:
  - **`nominee`** : 6 nulls fixed manually (e.g., “Hex Hector” for 1998).
  - **`artist`** : 132 nulls resolved using `workers` field (e.g., roles like “composer”).
- **Category Normalization**:
  - 638 raw categories → 62 standardized (e.g., “Song Of The Year”).
  - Fuzzy matching against official Grammy categories.

Challenges

- **Redundant `winner` Column**:
  - All entries were `True` , requiring redefinition of “winners” as unique `year + category` pairs.
- **Data Fragmentation**:
  - Categories like “Producer Of The Year” had inconsistent naming variants.

Insights

- **Temporal Trends**:
  - Post-2000 categories became more specialized (e.g., “Global Music Performance”).
  - Frequent winners: “(Various Artists)” won 66 times.

---

4. `4.0-nlg-extraction-api.ipynb`

Objective

Extract acoustic features for nominated songs using Reccobeats API.

Key Processes

- **Audio Download**:
  - YouTube searches via `yt-dlp` using `nominee` as query.
  - 90% success rate (126/140 tracks downloaded).
- **Feature Extraction**:
  - API endpoints (e.g., `/audio-features`) provided metrics like `danceability` , `valence` .

Challenges

- **API Limitations**:
  - Rate-limited to 1 call every 3 seconds (to avoid invalid responses in JSON due to high demand)
  - 14 tracks failed due to YouTube search errors.

Insights

- **Data Coverage**:
  - 72% of “Song Of The Year” nominees were found on Spotify.
  - Older tracks (1970s–1980s) faced higher download failure rates.

1. Dataset: “Record of the Year” and “Song of the Year” Nominees

The CSV file used for API extraction was derived from:

- **Primary Source**: Grammy Awards cleaned dataset obtained after the `3.0-nlg-eda-grammys` (`..data/intermediate/grammy.csv`)
- **Filtering**:

```
filtered_df = df[df['normalized_category'].isin(['Song Of The Year', 'Record Of The Year'])]
```

  - **Criteria**:
    - Normalized categories: “Song Of The Year” and “Record Of The Year”.
    - Key columns: `year` , `nominee` (song title), `artist` .

---

2. API Extraction Process

Key Steps:

1. **Nominee Filtering**:
   - Identified 124 unique songs in target categories.
   - Example filtered data:

year	category	nominee	artist
2020	Song Of The Year	“Bad Guy”	Billie Eilish
2019	Record Of The Year	“This Is America”	Childish Gambino

2. - **Drop duplicates**: Remove duplicate rows in the `filtered_df` DataFrame based on the values in the `nominee` (song) column.
3. **Audio Download**: in MP3 format
   - Used `yt-dlp` to search YouTube using `nominee + artist` (e.g., `"Bad Guy Billie Eilish"`).
   - Example command:

```
ydl.download([f"ytsearch1:{nominee} {artist}"])
```

6.**Trim audios**: to 30 seconds, because audiofile posted to the API must have a maximum file size of 5MB.
5. **Reccobeats API Call**:

- 30s trimmed audio sent to `/analysis/audio-features` endpoint.
- **API Response Example**:

```
{
  "danceability": 0.76,
  "energy": 0.82,
  "valence": 0.35,
  "tempo": 120.5,
  // ... (8 additional features)
}
```

---

3. Save results in JSON and convert to CSV to save both files

Extracted data (`results`) is saved in two formats, **JSON** and **CSV**, for easier access and further processing. The JSON file retains the raw structure of the data, while the CSV file organizes the features into separate columns.

Steps:

1. **Save as JSON**:
    - **File Path**: The JSON file is saved to `../data/raw/reccobeats_features.json` .
    - **Operation**:
        - Opens a file in write mode with UTF-8 encoding.
        - Uses `json.dump()` to serialize the `results` dictionary into JSON format.
        - Includes formatting with an indentation of 2 spaces for readability.
        - Ensures that non-ASCII characters are saved correctly (e.g., for special characters in text).
    - **Log Entry**: Logs a message confirming the successful save of the JSON file.
2. **Save as CSV**:
    - **Initial Conversion**:
        - Converts the `results` data into a pandas DataFrame (`df_results`), making it suitable for further manipulation.
    - **Expand Nested Features**:
        - Filters rows with a valid `"features"` column using `.dropna(subset=["features"])` to eliminate empty or missing values.
        - Converts the nested dictionaries in `"features"` into separate columns using `apply(pd.Series)` .
        - Combines the expanded `"features"` columns with the `"nominee"` column using `pd.concat()` to form a structured DataFrame (`features_combined`).
    - **File Path**: The CSV file is saved to `../data/external/reccobeats_features.csv` .
    - **Operation**:
        - Saves the cleaned DataFrame to a CSV file using `.to_csv()` , with no index included.
    - **Log Entry**: Logs a message confirming the successful save of the CSV file.

Output:

- **JSON File**: Contains raw data from `results` , formatted and encoded properly.
- **CSV File**: Organizes data into a tabular format, with expanded feature columns for easier analysis.

---

4. Common Issues & Solutions

- **Search Mismatches**:
    - **Cause**: Ambiguous names (e.g., “Hello” by Adele vs. Lionel Richie).
    - **Solution**: remove any characters that are considered unsafe or invalid for filenames and replaces spaces with underscores for easier readability and usability as a filename.(with `def safe_filename(title)` while downloading audio).

5. Results

```
Analizyng with ReccoBeats: 100%|██████████| 92/92 [09:03<00:00,  5.91s/it]
```

92 files analyzed with Reccobeats.

```
2025-04-10 16:44:30,062 - INFO - root - Results saved in JSON format: ../data/raw/reccobeats_features.json
2025-04-10 16:44:30,095 - INFO - root - Results saved in CSV format: ../data/external/reccobeats_features.csv
```

92 API results saved as CSV and JSON.

---

5. `5.0-nlg-eda-api.ipynb`

Objective

Validate and analyze Reccobeats API data.

Key Processes

- **Range Validation**:
  - All features adhered to expected ranges (e.g., `acousticness` ∈ [0.0, 1.0]).
- **Outlier Detection**:
  - **`loudness`** : 2 extreme values (-29.258 dB vs. mean = -16.485 dB).
  - **`tempo`** : 1 track at 54.99 BPM (outside IQR: 75–130 BPM).

Insights

- **Correlations**:
  - `danceability` and `energy` showed strong positive correlation (r = 0.68).
  - The data did not need any especial treatment.

---

6. `6.0-nlg-merge.ipynb`

Objective

Integrate Grammy, Spotify, and API data into a unified dataset.

Key Processes

- **Data Merging**:
  - Merge song data from API (`nominees`) , Grammys with (`nominees`) Spotify tracks, preserving artist information and prioritizing API data if `nominee` (song) matches `track_name` .
  - Add unmatched songs and optionally merges Grammy winner status.
  - 126 API-only tracks marked as “Unknown Artist”.

Key Observations

1. **Complete Columns (0 Nulls):**
   - `track_name` , audio features (`acousticness` , `danceability` , etc.)
   - **Why:** These exist in both API and Spotify data, with API values filling Spotify gaps

2. **Spotify-Specific Nulls (126 Nulls):**
    - `artists`, `album_name`, `popularity`, `duration_ms`, etc.
    - **Cause:** These represent 126 API nominees not found in Spotify's catalog
    - **Solution:** Manual artist/title disambiguation or accept missing Spotify metadata
3. **Massive Nulls (79,631):**
    - `year`, `title`, `artist`, `winner`
    - **Root Cause:** These columns likely come from `df_grammy` and indicate:
        - Grammy data only exists for `84,374 - 79,631 = 4,743` tracks
        - Mismatched merge keys (track_name alone isn't sufficient for Grammy matching)

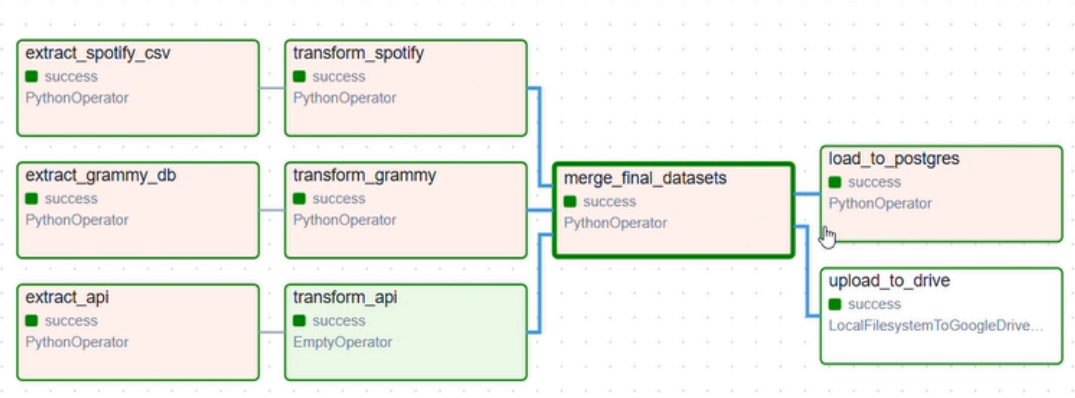
- **Null Handling**:
    - `winner` set to `False` for non-matching entries.
    - Spotify NaN columns (e.g., `popularity`) filled with medians or "Unknown".
    - Remove columns with >90% nulls
    - Merge again using both track and artist for higher accuracy.

Final Output

Unified dataset (`final_data.csv`) with:

- **84,374 rows**: 79,631 from Spotify + 4,743 from Grammy.
- **32 columns**: Acoustic features, metadata, and quality flags.
- **Key Variables**: `winner`, `danceability`, `valence`, `data_source` ("Spotify + API" or "API Only").

🛠 Pipeline Architecture (DAG)



Evidence of DAG in succesfull execution state. Data flow: Extract→ Transform → Merge → Load (upload to Drive and load to PostgreSQL database).

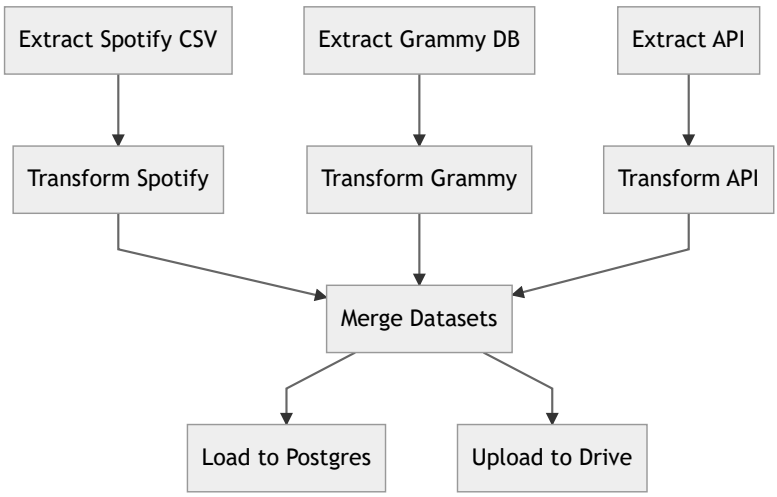
Overview

This Airflow DAG orchestrates an ETL pipeline for processing Grammy Awards data, Spotify track data, and audio features from an external API (ReccoBeats). The pipeline includes extraction, transformation, merging, and loading stages.

DAG Configuration

Property	Value
`dag_id`	`etl_full_pipeline_dag`
`description`	ETL pipeline for audio/grammy/spotify
`Schedule`	Daily (`@daily`)
`Retries`	3
`Retry Delay`	1 minute
`Max Active Runs`	1

Task Graph



Task Details

1. Extraction Tasks

Task ID	Operator	Description
`extract_spotify_csv`	PythonOperator	Loads Spotify data from CSV using `load_local_csv()`
`extract_grammy_db`	PythonOperator	Extracts Grammy data from PostgreSQL using `read_table_from_db()`
`extract_api`	PythonOperator	Processes audio features using `process_audio_dataset()`

Key Parameters:

```
params.SPOTIFY_DATASET_PATH = "data/external/spotify_dataset.csv"
table_name = 'grammys_raw'  # Source table for Grammy data
```

2. Transformation Tasks

Task ID	Operator	Description
`transform_spotify`	PythonOperator	Executes `transform_spotify_dataset()` for transformations mentioned in `2.0-nlg-eda-spotify.ipynb`
`transform_grammy`	PythonOperator	Runs `transform_grammy_dataset()` for transformations mentioned in `3.0-nlg-eda-grammys.ipynb`
`transform_api`	EmptyOperator	Placeholder task (API data requires no transformation)

3. Merge Task

Task ID	Operator	Description
`merge_final_datasets`	PythonOperator	Combines all datasets using `merge_and_enrich_datasets()` , which perfoms all transformations mentioned in `6.0-nlg-merge.ipynb`

4. Load Tasks

Task ID	Operator	Description
`load_to_postgres`	PythonOperator	Loads final data to PostgreSQL using `load_to_database()`
`upload_to_drive`	LocalFilesystemToGoogleDriveOperator	Uploads CSV to Google Drive (GCP connection required)

Key Features

1. **Error Handling**:
    - 3 retries with 1-minute delay
    - 10-minute timeout for all Python tasks
    - Email alerts on failure (configure `email_on_failure`)
2. **Dependencies**:

```
# Extraction → Transformation
extract_spotify >> transform_spotify
extract_grammy >> transform_grammy
extract_api >> transform_api

# Transformation → Merge → Load
[transform_spotify, transform_grammy, transform_api] >> merge_data
merge_data >> [load_db, upload_to_drive]
```

3. **Google Drive Integration**:

```
upload_to_drive = LocalFilesystemToGoogleDriveOperator(
    local_paths=[str(Path(params.processed_data) / 'final_data.csv'],
    drive_folder='root',
    gcp_conn_id='google_cloud_default'  # Requires configured GCP connection
)
```

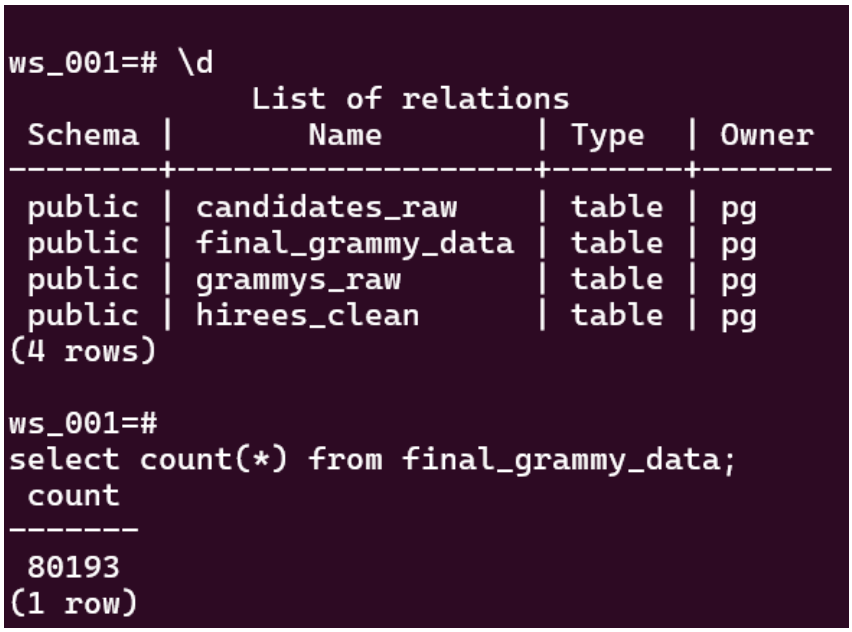
Custom Modules Used

Module	Purpose
`extraction_grammy_spotify`	CSV/DB loading utilities
`extraction_api`	Audio feature processing
`transform_spotify`	Genre consolidation and cleaning
`transform_grammys`	Category normalization
`merge`	Dataset integration logic
`load`	Database loading functions

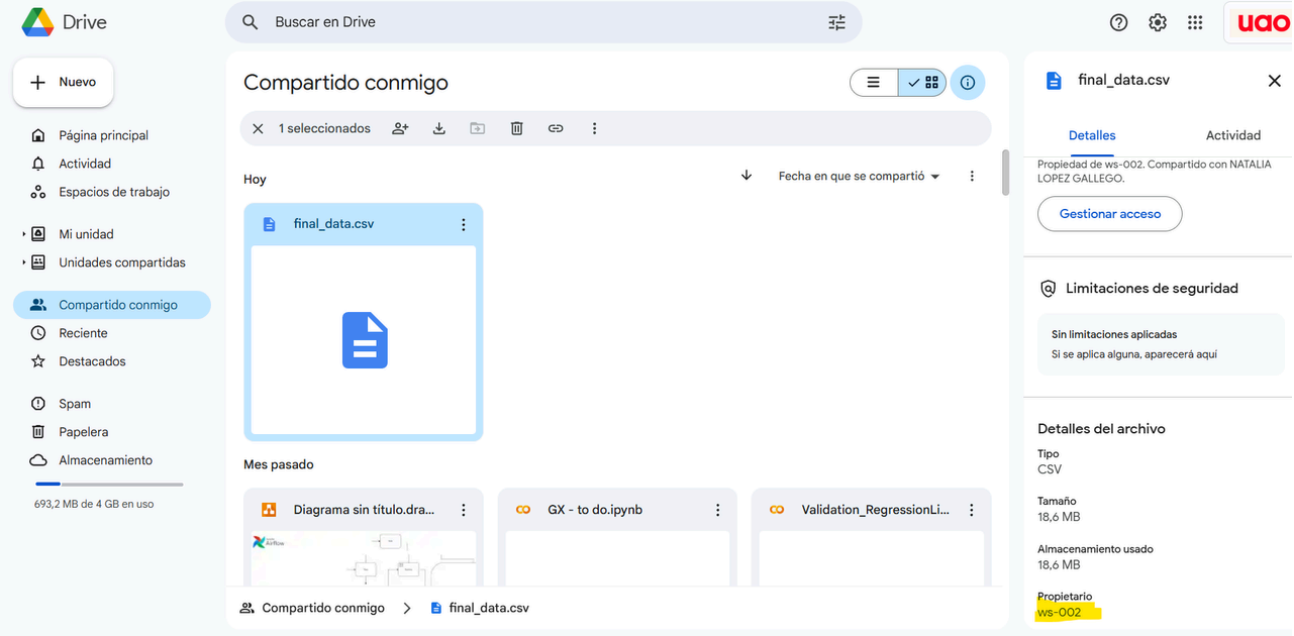
Execution Requirements

1. Airflow connection to PostgreSQL database
2. Google Cloud connection (`google_cloud_default`) in Airflow with Key
3. Python dependencies in `src/requirements.txt`
4. Proper file paths configured in `src/params.py`

Load evidence



Evidence of data successfully loaded into a PostgreSQL database. Terminal output confirming the data load process. The command `\d` lists the relations in the database, including 'final_grammy_data', the final processed data. Another command, `select count(*) from final_grammy_data;` , verifies the presence of 80,193 rows in the 'final_grammy_data' table, demonstrating the successful execution of the data load process for the workshop. This provides clear evidence of data integration into the PostgreSQL database.



*`final_data.csv` stored in the Drive of my institutional university account. The file's ownership is highlighted in yellow, clearly assigned to the project linked with the service account on Google Cloud Platform (GCP). This configuration was specifically established for the workshop and utilized in conjunction with the Airflow operator **LocalFilesystemToGCSOperator** to facilitate operation of data uploading from local filesystem to Google Cloud Services drive.*

Dashboard

Redash was used to make visualizations based the relationship between acoustic characteristics of the Grammy Award winning songs in **"Record of the Year"** and **"Song of the Year"** categories (obtained via API) VS those of the Non-Grammy Award winning songs from Spotify data. We answer:

- Do winners share distinct acoustic profiles?
- Can we predict winners using audio features?

*Dashboard with **Type of Mood** category distribution showing winners.*

*Dashboard with **Type of Mood** category distribution showing non- winners.*

Insights

1. **Tempo**:
    - Both Grammy winners and non-winners favor faster tempos; however, winners showed a slightly higher proportion of slower tempos. This might suggest that a balance between energy and subtlety contributes to a winning edge.
2. **Mood**:
    - Both winners and non-winners shared similar distributions of neutral and sad moods.
    - Grammy-winning songs clearly have a stronger presence of happy moods compared to non-winners. This suggests that cheerful and uplifting tracks are favored by voters for the **“Record of the Year”** and **“Song of the Year”** categories.
    - Non-winners (in magenta) show a lower representation of happy moods, indicating that while they might explore other emotional tones, happiness plays a less central role.
3. **Artistic Appeal of Happiness**:
    - Happy songs among winners might cater to broad emotional relatability and positive vibes, which resonate well with both audiences and award judges.
    - The contrast highlights how emotional positivity can be a hallmark of success in Grammy-winning tracks.
4. **Balance**:
    - While Grammy winners shine in the happy category, they might also blend happiness with thoughtful musical elements, such as catchy melodies, clever lyrics, and polished production, to stand out.
5. **Danceability**:
    - On average, Grammy-winning songs have slightly higher danceability scores compared to non-winners. This could hint at the appeal of rhythm and groove among award-winning tracks.
6. **Acousticness**:
    - While non-winning songs vastly outnumber winners in the dataset, the average level of acousticness is quite similar. Acoustic songs (with higher acousticness) are more likely to appear among winners, showing an appreciation for

natural, less synthetic sounds.

7. **Instrumentalness**:
    - Instrumental tracks are rare for both groups. However, Grammy winners have an even lower average instrumentalness, reinforcing the dominance of vocal-driven music in these prestigious categories.
8. **Loudness**:
    - The loudness distribution for both categories shows similarity, with no significant outliers. It reflects contemporary audio production standards rather than a distinguishing factor for Grammy success.
9. **Content Type**:
    - Winners are almost exclusively musical tracks, whereas non-winners include a slightly higher mix of speech or blended content. This highlights the expectation of polished and focused musical compositions among winners.

Conclusion

It seems that predicting Grammy winners purely based on audio features could be quite challenging. While the analysis shows subtle differences—like slightly higher danceability or acousticness among winners—these characteristics are not starkly distinct from non-winning songs. The overlap in loudness, mood, and other features suggests that Grammy success relies on more than just audio traits.

Factors such as cultural impact, lyrical content, artist popularity, emotional connection, and even industry politics likely play significant roles in determining winners. Audio features alone might capture part of the equation but not the full picture. Machine learning models could attempt to use these features for predictions; however, the accuracy would depend on how well additional context beyond just the audio characteristics is integrated.

> Written with StackEdit.