airflow\dags\matching_dag_OPTIONAL.py

```python
# 685.652, Spring 2025 - Group 6 Final Project
# matching_dag.py

# Matches up closely related tracks
# From different datasets
# Using fuzzy string matching
# And overrides group6_id based on findings



from airflow import DAG
from airflow.operators.empty import EmptyOperator
from airflow.providers.postgres.operators.postgres import PostgresOperator
from airflow.operators.python_operator import PythonOperator, BranchPythonOperator
from airflow.models import Variable
from airflow.providers.postgres.hooks.postgres import PostgresHook
import pandas as pd
from rapidfuzz import process, fuzz
import numpy as np
import time
from datetime import datetime, timedelta
from math import sqrt
import os
import uuid


default_args = {
    'owner': 'group6',
    'depends_on_past': False,
    'start_date': datetime(2025, 1, 1),
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=5),
}


# Bring data from all existing tables in
def load_all_from_db():
    pg_hook = PostgresHook(postgres_conn_id='pg_group6')

    with pg_hook.get_conn() as conn:
        with conn.cursor() as cur:
            cur.execute("SELECT * FROM spotify_tracks")
            spot_data = cur.fetchall()
            spot_columns = [desc[0] for desc in cur.description]

            cur.execute("SELECT * FROM billboard_chart_data")
```

```
            kag_data = cur.fetchall()
            kag_columns = [desc[0] for desc in cur.description]

            cur.execute("SELECT * FROM lastfm_tracks")
            lastfm_data = cur.fetchall()
            lastfm_columns = [desc[0] for desc in cur.description]

            cur.execute("SELECT * FROM acousticbrainz_features")
            ab_data = cur.fetchall()
            ab_columns = [desc[0] for desc in cur.description]

    df_spot = pd.DataFrame(spot_data, columns=spot_columns)
    df_kag = pd.DataFrame(kag_data, columns=kag_columns)
    df_lastfm = pd.DataFrame(lastfm_data, columns=lastfm_columns)
    df_ab = pd.DataFrame(ab_data, columns=ab_columns)

    return df_spot, df_kag, df_lastfm, df_ab

# Helper for match functions
def append_match_result(results, spot_artists, spot_song_name, other_artists,
other_song_name,
                        match_found, other_index, spot_index, match_score,
                        artist_match_pct, name_match_pct):
    results.append({
        'spot_artists': spot_artists,
        'spot_song_name': spot_song_name,
        'other_artists': other_artists,
        'other_song_name': other_song_name,
        'match_found': match_found,
        'other_index': other_index,
        'spot_index': spot_index,
        'best_match_score': match_score,
        'best_match_artist_match_pct': artist_match_pct,
        'best_match_name_match_pct': name_match_pct
    })



# Match spotify tracks to kaggle tracks
# Overwrite group6_id based on findings
def match_spot_to_kag(df_spot, df_kag):

    # Store the original dataframe before filtering
    df_kag_original = df_kag.copy()

    # To decrease matching time, filter out songs
    # That didn't reach a certain chart position
    # Because fuzzy string matching is slow
    KAG_FILTER = 15
    df_kag_filtered = df_kag[df_kag['peak_chart_pos'] <= KAG_FILTER]
```

```
    df_kag_filtered = df_kag_filtered.reset_index()  # Keep original index in 'index' column

    print(f"Filtered Kaggle dataset to {len(df_kag_filtered)} records with peak chart position <= {KAG_FILTER}")
    print(f"Will search for matches within the {len(df_spot)} Spotify records")
    print(f"This may take a while...")

    # Will be used to track matches
    df_kag_filtered['has_spot_match'] = False
    df_kag_filtered['spot_match_index'] = None
    df_kag_filtered['exact_match'] = False
    df_spot['has_kag_match'] = False
    df_spot['kag_match_index'] = None

    results = []

    start_time = time.time()

    # Pre-compute data for faster matching
    kag_performers = set(df_kag_filtered['top_artist'].unique())
    performer_indices = {performer: df_kag_filtered[df_kag_filtered['top_artist'] == performer].index[0]
                         for performer in kag_performers}

    performer_to_titles = {}
    for idx, row in df_kag_filtered.iterrows():
        performer = row['top_artist']
        title = row['song_name']
        if performer not in performer_to_titles:
            performer_to_titles[performer] = set()
        performer_to_titles[performer].add(title)

    kag_performer_title_to_index = {}
    for idx, row in df_kag_filtered.iterrows():
        performer = row['top_artist']
        title = row['song_name']
        kag_performer_title_to_index[(performer, title)] = idx

    # Pre-compute word sets
    kag_artist_words_dict = {idx: set(row['top_artist'].split()) for idx, row in df_kag_filtered.iterrows()}
    kag_title_words_dict = {idx: set(row['song_name'].split()) for idx, row in df_kag_filtered.iterrows()}

    # Pre-compute Spotify artist and title word sets
    spot_artist_words_dict = {idx: set(row['top_artist'].split()) for idx, row in df_spot.iterrows()}
    spot_title_words_dict = {idx: set(row['song_name'].split()) for idx, row in df_spot.iterrows()}

    # Create artist+first letter of title index
```

```
    artist_title_first = {}
    for idx, row in df_kag_filtered.iterrows():
        if row['song_name']:
            key = (row['top_artist'], row['song_name'][0])
            if key not in artist_title_first:
                artist_title_first[key] = []
            artist_title_first[key].append(idx)

    # Iterate through each spotify track
    for index, row in df_spot.iterrows():

        # If there is already a match, skip
        if df_spot.at[index, 'has_kag_match']:
            continue

        artist = row['top_artist']
        song_name = row['song_name']
        match_found = False

        # Get pre-computed word sets for this Spotify track
        spot_artist_words = spot_artist_words_dict[index]
        spot_title_words = spot_title_words_dict[index]

        best_match_performer = None
        best_match_title = None
        best_match_score = 0
        best_match_artist_match_pct = 0
        best_match_name_match_pct = 0
        best_match_kag_index = None

        # Print progress every 10 records
        if index % 10 == 0:
            print(f"Processing record {index + 1} of {len(df_spot)}...")

        if artist in kag_performers:
            curr_performer = artist

            # If the song name is in the performer's title set,
            # Then it is an exact match
            # These will already have the same group6_id
            if song_name in performer_to_titles.get(curr_performer, set()):
                filtered_kag_index = kag_performer_title_to_index.get((curr_performer,
song_name))
                # Get the original index from the filtered dataframe
                original_kag_index = df_kag_filtered.at[filtered_kag_index, 'index']
                best_match_kag_index = filtered_kag_index

                # Update both dataframes
                df_spot.at[index, 'has_kag_match'] = True
                df_spot.at[index, 'kag_match_index'] = original_kag_index
```

```
                df_kag_filtered.at[filtered_kag_index, 'has_spot_match'] = True
                df_kag_filtered.at[filtered_kag_index, 'spot_match_index'] = index
                df_kag_filtered.at[filtered_kag_index, 'exact_match'] = True

                match_found = True
                append_match_result(results, artist, song_name, curr_performer, song_name,
                                    match_found, original_kag_index, index, 100, 100, 100)
                continue

            # Start from that artist's first appearance in the dataset
            # For better likelihood of quick match
            start_index = performer_indices[curr_performer]
            for i in range(start_index, len(df_kag_filtered)):

                # If there is already a match, skip
                if df_kag_filtered.at[i, 'has_spot_match']:
                    continue

                # Get the current top artist and title
                curr_performer = df_kag_filtered.iloc[i]['top_artist']
                title = df_kag_filtered.iloc[i]['song_name']

                # If the first letter of the artist or title doesn't match, skip
                if curr_performer[0] != artist[0]:
                    continue
                if title[0] != song_name[0]:
                    continue

                # If the artist words don't intersect, skip
                curr_artist_words = kag_artist_words_dict[i]
                if not spot_artist_words.intersection(curr_artist_words):
                    continue

                # If the title words don't intersect, skip
                curr_title_words = kag_title_words_dict[i]
                if not spot_title_words.intersection(curr_title_words):
                    continue

                # Check QRatio fuzzy match for artist and title
                # Skip if either is very low
                artist_score = fuzz.QRatio(artist, curr_performer)
                if artist_score < 50:
                    continue
                title_score = fuzz.QRatio(song_name, title)
                if title_score < 50:
                    continue

                # Geometric mean is good - penalizes one very low score more
                geo_mean_score = sqrt(artist_score * title_score)

```

```
                # If the fuzzy match is better than the current best, update
                # This block was mostly for testing
                # To find the right threshold for a good match
                if geo_mean_score > best_match_score:
                    best_match_performer = curr_performer
                    best_match_title = title
                    best_match_score = geo_mean_score
                    best_match_artist_match_pct = artist_score
                    best_match_name_match_pct = title_score
                    best_match_kag_index = i

                # If the fuzzy match is close enough,
                # Then it is a match
                if geo_mean_score >= 70:
                    match_found = True

                    filtered_kag_index = i
                    original_kag_index = df_kag_filtered.at[filtered_kag_index, 'index']

                    df_spot.at[index, 'has_kag_match'] = True
                    df_spot.at[index, 'kag_match_index'] = original_kag_index
                    df_kag_filtered.at[filtered_kag_index, 'has_spot_match'] = True
                    df_kag_filtered.at[filtered_kag_index, 'spot_match_index'] = index

                    append_match_result(results, artist, song_name, curr_performer, title,
                                        match_found, original_kag_index, index,
best_match_score,
                                        best_match_artist_match_pct, best_match_name_matc-
h_pct)
                    break

            # Need to wrap back around and keep looking
            # Back to the start index
            # All else in this block is the same
            if not match_found:
                for i in range(0, start_index):
                    if df_kag_filtered.at[i, 'has_spot_match']:
                        continue

                    curr_performer = df_kag_filtered.iloc[i]['top_artist']
                    title = df_kag_filtered.iloc[i]['song_name']

                    if curr_performer[0] != artist[0]:
                        continue
                    if title[0] != song_name[0]:
                        continue

                    curr_artist_words = kag_artist_words_dict[i]
                    if not spot_artist_words.intersection(curr_artist_words):
                        continue
```

```
290
291                     curr_title_words = kag_title_words_dict[i]
292                     if not spot_title_words.intersection(curr_title_words):
293                         continue
294
295                     artist_score = fuzz.QRatio(artist, curr_performer)
296                     if artist_score < 50:
297                         continue  # Skip early
298
299                     # Only calculate title score if artist score is promising
300                     title_score = fuzz.QRatio(song_name, title)
301                     if title_score < 50:
302                         continue  # Skip early
303
304                     geo_mean_score = sqrt(artist_score * title_score)
305
306                     if geo_mean_score > best_match_score:
307                         best_match_performer = curr_performer
308                         best_match_title = title
309                         best_match_score = geo_mean_score
310                         best_match_artist_match_pct = artist_score
311                         best_match_name_match_pct = title_score
312                         best_match_kag_index = i
313
314                     if geo_mean_score >= 70:
315                         match_found = True
316
317                         filtered_kag_index = i
318                         original_kag_index = df_kag_filtered.at[filtered_kag_index, 'index']
319
320                         df_spot.at[index, 'has_kag_match'] = True
321                         df_spot.at[index, 'kag_match_index'] = original_kag_index
322                         df_kag_filtered.at[filtered_kag_index, 'has_spot_match'] = True
323                         df_kag_filtered.at[filtered_kag_index, 'spot_match_index'] = index
324
325                         append_match_result(results, artist, song_name, curr_performer,
    title,
326                                             match_found, original_kag_index, index,
    best_match_score,
327                                             best_match_artist_match_pct,
    best_match_name_match_pct)
328                         break
329
330         # This is all the same as above
331         # Just for artists that don't have an exact match between datasets
332         else:
333             for i in range(0, len(df_kag_filtered)):
334                 if df_kag_filtered.at[i, 'has_spot_match']:
335                     continue
336
337                 curr_performer = df_kag_filtered.iloc[i]['top_artist']
```

```
                title = df_kag_filtered.iloc[i]['song_name']

                if curr_performer[0] != artist[0]:
                    continue
                if title[0] != song_name[0]:
                    continue

                curr_artist_words = kag_artist_words_dict[i]
                if not spot_artist_words.intersection(curr_artist_words):
                    continue

                curr_title_words = kag_title_words_dict[i]
                if not spot_title_words.intersection(curr_title_words):
                    continue

                artist_score = fuzz.QRatio(artist, curr_performer)
                if artist_score < 50:
                    continue
                title_score = fuzz.QRatio(song_name, title)
                if title_score < 50:
                    continue
                geo_mean_score = sqrt(artist_score * title_score)

                if geo_mean_score > best_match_score:
                    best_match_performer = curr_performer
                    best_match_title = title
                    best_match_score = geo_mean_score
                    best_match_artist_match_pct = artist_score
                    best_match_name_match_pct = title_score
                    best_match_kag_index = i

                if geo_mean_score >= 70:
                    match_found = True

                    filtered_kag_index = i
                    original_kag_index = df_kag_filtered.at[filtered_kag_index, 'index']

                    df_spot.at[index, 'has_kag_match'] = True
                    df_spot.at[index, 'kag_match_index'] = original_kag_index
                    df_kag_filtered.at[filtered_kag_index, 'has_spot_match'] = True
                    df_kag_filtered.at[filtered_kag_index, 'spot_match_index'] = index

                    append_match_result(results, artist, song_name, curr_performer, title,
                                        match_found, original_kag_index, index,
best_match_score,
                                        best_match_artist_match_pct, best_match_name_matc-
h_pct)
                    break

        if not match_found and best_match_kag_index is not None:
```

```
            original_kag_index = df_kag_filtered.at[best_match_kag_index, 'index'] if best_match_kag_index is not None else None
            append_match_result(results, artist, song_name, best_match_performer, best_match_title,
                                match_found, original_kag_index, index, best_match_score,
                                best_match_artist_match_pct, best_match_name_match_pct)
        elif not match_found:
            append_match_result(results, artist, song_name, best_match_performer, best_match_title,
                                match_found, best_match_kag_index, index, best_match_score,
                                best_match_artist_match_pct, best_match_name_match_pct)

    # After all matches are found,
    # Update the original dataframe with the match information, but preserve existing matches
    for index, row in df_spot.iterrows():
        if row['has_kag_match'] and pd.notna(row['kag_match_index']):
            original_kag_index = int(row['kag_match_index'])
            if 0 <= original_kag_index < len(df_kag):  # Validate index
                # Only update if the group6_ids are different
                if df_kag.at[original_kag_index, 'group6_id'] != row['group6_id']:
                    df_kag.at[original_kag_index, 'group6_id'] = row['group6_id']

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"\nTotal time taken for Spotify and Kaggle matching: {elapsed_time:.2f} seconds")


    # Write to CSV for viewing results
    results_df = pd.DataFrame(results)
    results_df.sort_values(by=['best_match_score'], ascending=False, inplace=True)
    results_df.reset_index(drop=True, inplace=True)

    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    match_results_file_name = f"spottokaggle_match_results_{timestamp}.csv"
    data_dir = os.path.join(os.getcwd(), 'data')
    os.makedirs(data_dir, exist_ok=True)
    match_results_file_path = os.path.join(data_dir, match_results_file_name)
    results_df.to_csv(match_results_file_path, index=False)

    print(f"Successfully wrote to {match_results_file_name}\n")

    return df_spot, df_kag




# All the same as above
# Just between spotify and lastfm
def match_spot_to_lastfm(df_spot, df_lastfm):

    start_time = time.time()
```

```
434
435     # Initialize tracking columns
436     df_lastfm['has_spot_match'] = False
437     df_lastfm['spot_match_index'] = None
438     df_lastfm['exact_match'] = False
439     df_spot['has_lastfm_match'] = False
440     df_spot['lastfm_match_index'] = None
441
442     results = []
443
444     # Pre-compute word sets for faster matching
445     lastfm_artists = set(df_lastfm['artist'].unique())
446     artist_indices = {artist: df_lastfm[df_lastfm['artist'] == artist].index[0]
447                       for artist in lastfm_artists}
448
449     artist_to_titles = {}
450     for idx, row in df_lastfm.iterrows():
451         artist = row['artist']
452         title = row['song_name']
453         if artist not in artist_to_titles:
454             artist_to_titles[artist] = set()
455         artist_to_titles[artist].add(title)
456
457     lastfm_artist_title_to_index = {}
458     for idx, row in df_lastfm.iterrows():
459         artist = row['artist']
460         title = row['song_name']
461         lastfm_artist_title_to_index[(artist, title)] = idx
462
463     # Pre-compute word sets
464     lastfm_artist_words_dict = {idx: set(row['artist'].split()) for idx, row in
    df_lastfm.iterrows()}
465     lastfm_title_words_dict = {idx: set(row['song_name'].split()) for idx, row in
    df_lastfm.iterrows()}
466
467     # Pre-compute Spotify artist and title word sets
468     spot_artist_words_dict = {idx: set(row['top_artist'].split()) for idx, row in
    df_spot.iterrows()}
469     spot_title_words_dict = {idx: set(row['song_name'].split()) for idx, row in
    df_spot.iterrows()}
470
471     # Create artist+first letter of title index
472     artist_title_first = {}
473     for idx, row in df_lastfm.iterrows():
474         if row['song_name']:
475             key = (row['artist'], row['song_name'][0])
476             if key not in artist_title_first:
477                 artist_title_first[key] = []
478             artist_title_first[key].append(idx)
479
480     # Process each Spotify track
```

```
    for index, row in df_spot.iterrows():
        # Skip if already has a match
        if df_spot.at[index, 'has_lastfm_match']:
            continue

        artist = row['top_artist']
        song_name = row['song_name']
        match_found = False

        # Get pre-computed word sets for this Spotify track
        spot_artist_words = spot_artist_words_dict[index]
        spot_title_words = spot_title_words_dict[index]

        best_match_performer = None
        best_match_title = None
        best_match_score = 0
        best_match_artist_match_pct = 0
        best_match_name_match_pct = 0
        best_match_lastfm_index = None

        if index % 10 == 0:
            print(f"Processing record {index + 1} of {len(df_spot)}...")

        if artist in lastfm_artists:
            curr_artist = artist

            # Check for exact artist+title match first
            if song_name in artist_to_titles.get(curr_artist, set()):
                lastfm_index = lastfm_artist_title_to_index.get((curr_artist, song_name))
                best_match_lastfm_index = lastfm_index

                # Update both dataframes
                df_spot.at[index, 'has_lastfm_match'] = True
                df_spot.at[index, 'lastfm_match_index'] = lastfm_index
                df_lastfm.at[lastfm_index, 'has_spot_match'] = True
                df_lastfm.at[lastfm_index, 'spot_match_index'] = index
                df_lastfm.at[lastfm_index, 'exact_match'] = True

                match_found = True
                append_match_result(results, artist, song_name, curr_artist, song_name,
                                    match_found, best_match_lastfm_index, index, 100, 100,
100)
                continue

            start_index = artist_indices[curr_artist]

            for i in range(start_index, len(df_lastfm)):

                if df_lastfm.at[i, 'has_spot_match']:
                    continue
```

```
530
531                 curr_artist = df_lastfm.iloc[i]['artist']
532                 title = df_lastfm.iloc[i]['song_name']
533
534                 if curr_artist[0] != artist[0]:
535                     continue
536                 if title[0] != song_name[0]:
537                     continue
538
539                 curr_artist_words = lastfm_artist_words_dict[i]
540                 if not spot_artist_words.intersection(curr_artist_words):
541                     continue
542
543                 curr_title_words = lastfm_title_words_dict[i]
544                 if not spot_title_words.intersection(curr_title_words):
545                     continue
546
547                 artist_score = fuzz.QRatio(artist, curr_artist)
548                 if artist_score < 50:
549                     continue
550                 title_score = fuzz.QRatio(song_name, title)
551                 if title_score < 50:
552                     continue
553                 geo_mean_score = sqrt(artist_score * title_score)
554
555                 if geo_mean_score > best_match_score:
556                     best_match_performer = curr_artist
557                     best_match_title = title
558                     best_match_score = geo_mean_score
559                     best_match_artist_match_pct = artist_score
560                     best_match_name_match_pct = title_score
561                     best_match_lastfm_index = i
562
563                 # Need slightly higher threshold for lastfm
564                 # Has a few cover bands that foul things up
565                 if geo_mean_score >= 75:
566                     match_found = True
567
568                     lastfm_index = i
569                     df_spot.at[index, 'has_lastfm_match'] = True
570                     df_spot.at[index, 'lastfm_match_index'] = lastfm_index
571                     df_lastfm.at[lastfm_index, 'has_spot_match'] = True
572                     df_lastfm.at[lastfm_index, 'spot_match_index'] = index
573
574                     append_match_result(results, artist, song_name, curr_artist, title,
575                                         match_found, best_match_lastfm_index, index,
    best_match_score,
576                                         best_match_artist_match_pct, best_match_name_matc-
    h_pct)
577                     break
```

```
578
579             if not match_found:
580                 for i in range(0, start_index):
581                     if df_lastfm.at[i, 'has_spot_match']:
582                         continue
583
584                     curr_artist = df_lastfm.iloc[i]['artist']
585                     title = df_lastfm.iloc[i]['song_name']
586
587                     if curr_artist[0] != artist[0]:
588                         continue
589                     if title[0] != song_name[0]:
590                         continue
591
592                     curr_artist_words = lastfm_artist_words_dict[i]
593                     if not spot_artist_words.intersection(curr_artist_words):
594                         continue
595
596                     curr_title_words = lastfm_title_words_dict[i]
597                     if not spot_title_words.intersection(curr_title_words):
598                         continue
599
600                     artist_score = fuzz.QRatio(artist, curr_artist)
601                     if artist_score < 50:
602                         continue  # Skip early
603
604                     # Only calculate title score if artist score is promising
605                     title_score = fuzz.QRatio(song_name, title)
606                     if title_score < 50:
607                         continue  # Skip early
608
609                     geo_mean_score = sqrt(artist_score * title_score)
610
611                     if geo_mean_score > best_match_score:
612                         best_match_performer = curr_artist
613                         best_match_title = title
614                         best_match_score = geo_mean_score
615                         best_match_artist_match_pct = artist_score
616                         best_match_name_match_pct = title_score
617                         best_match_lastfm_index = i
618
619                     if geo_mean_score >= 75:
620                         match_found = True
621                         lastfm_index = i
622                         df_spot.at[index, 'has_lastfm_match'] = True
623                         df_spot.at[index, 'lastfm_match_index'] = lastfm_index
624                         df_lastfm.at[lastfm_index, 'has_spot_match'] = True
625                         df_lastfm.at[lastfm_index, 'spot_match_index'] = index
626
627                         append_match_result(results, artist, song_name, curr_artist, title,
```

```
                                        match_found, best_match_lastfm_index, index, best_match_score,
                                        best_match_artist_match_pct, best_match_name_match_pct)
                        break

        else:
            for i in range(0, len(df_lastfm)):
                if df_lastfm.at[i, 'has_spot_match']:
                    continue

                curr_artist = df_lastfm.iloc[i]['artist']
                title = df_lastfm.iloc[i]['song_name']

                if curr_artist[0] != artist[0]:
                    continue
                if title[0] != song_name[0]:
                    continue

                curr_artist_words = lastfm_artist_words_dict[i]
                if not spot_artist_words.intersection(curr_artist_words):
                    continue

                curr_title_words = lastfm_title_words_dict[i]
                if not spot_title_words.intersection(curr_title_words):
                    continue

                artist_score = fuzz.QRatio(artist, curr_artist)
                if artist_score < 50:
                    continue
                title_score = fuzz.QRatio(song_name, title)
                if title_score < 50:
                    continue
                geo_mean_score = sqrt(artist_score * title_score)

                if geo_mean_score > best_match_score:
                    best_match_performer = curr_artist
                    best_match_title = title
                    best_match_score = geo_mean_score
                    best_match_artist_match_pct = artist_score
                    best_match_name_match_pct = title_score
                    best_match_lastfm_index = i

                if geo_mean_score >= 75:
                    match_found = True
                    lastfm_index = i
                    df_spot.at[index, 'has_lastfm_match'] = True
                    df_spot.at[index, 'lastfm_match_index'] = lastfm_index
                    df_lastfm.at[lastfm_index, 'has_spot_match'] = True
                    df_lastfm.at[lastfm_index, 'spot_match_index'] = index
```

```
676
677                     append_match_result(results, artist, song_name, curr_artist, title,
678                                         match_found, best_match_lastfm_index, index,
best_match_score,
679                                         best_match_artist_match_pct, best_match_name_matc-
h_pct)
680                     break
681
682         if not match_found:
683             append_match_result(results, artist, song_name, best_match_performer,
best_match_title,
684                                 match_found, best_match_lastfm_index, index,
best_match_score,
685                                 best_match_artist_match_pct, best_match_name_match_pct)
686
687     # After all matches are found,
688     # Overwrite group6_id in lastfm set based on findings, but preserve existing matches
689     for index, row in df_spot.iterrows():
690         if df_spot.at[index, 'has_lastfm_match']:
691             lastfm_index = df_spot.at[index, 'lastfm_match_index']
692             # Only update if the group6_ids are different
693             if df_lastfm.at[lastfm_index, 'group6_id'] != row['group6_id']:
694                 df_lastfm.at[lastfm_index, 'group6_id'] = df_spot.at[index, 'group6_id']
695
696     end_time = time.time()
697     elapsed_time = end_time - start_time
698     print(f"\nTotal time taken for Spotify and LastFM matching: {elapsed_time:.2f} seconds")
699
700     results_df = pd.DataFrame(results)
701     results_df.sort_values(by=['best_match_score'], ascending=False, inplace=True)
702     results_df.reset_index(drop=True, inplace=True)
703
704     timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
705     match_results_file_name = f"spottolastfm_match_results_{timestamp}.csv"
706     data_dir = os.path.join(os.getcwd(), 'data')
707     os.makedirs(data_dir, exist_ok=True)
708     match_results_file_path = os.path.join(data_dir, match_results_file_name)
709     results_df.to_csv(match_results_file_path, index=False)
710
711     print(f"Successfully wrote to {match_results_file_name}\n")
712
713     return df_spot, df_lastfm
714
715 # LastFM is already tied to AcousticBrainz via mbid
716 # So this just updates group6_id in AcousticBrainz
717 def lastfm_to_ab(df_lastfm, df_ab):
718     start_time = time.time()
719
720     # Create a dictionary for faster lookup
721     lastfm_mbid_to_group6id = {
722         row['mbid']: row['group6_id']
```

```
        for idx, row in df_lastfm.iterrows()
        if pd.notna(row['mbid']) and pd.notna(row['group6_id'])
    }

    # Track how many were matched
    match_count = 0

    # Update df_ab rows
    for idx, row in df_ab.iterrows():
        if pd.notna(row['mbid']) and row['mbid'] in lastfm_mbid_to_group6id:
            df_ab.at[idx, 'group6_id'] = lastfm_mbid_to_group6id[row['mbid']]
            match_count += 1

    end_time = time.time()
    elapsed_time = end_time - start_time
    print(f"\nMatched {match_count} AcousticBrainz records with LastFM records by mbid")
    print(f"Total time taken: {elapsed_time:.2f} seconds")

    return df_ab

def update_db_after_matching(df_kag, df_lastfm, df_ab):
    pg_hook = PostgresHook(postgres_conn_id='pg_group6')
    conn = pg_hook.get_conn()

    with conn.cursor() as cur:
        # Update Billboard records based on matches found
        kag_updates = 0
        for idx, row in df_kag.iterrows():
            # Only update records that were explicitly matched in the matching process
            if 'has_spot_match' in df_kag.columns and row['has_spot_match']:
                cur.execute("""
                    UPDATE billboard_chart_data
                    SET group6_id = %s
                    WHERE song_name = %s AND top_artist = %s
                """, (
                    row['group6_id'],
                    row['song_name'],
                    row['top_artist']
                ))
                kag_updates += cur.rowcount

        print(f"Updated {kag_updates} rows in billboard_chart_data")

    # Update LastFM tracks
    lastfm_updates = 0
    with conn.cursor() as cur:
        for idx, row in df_lastfm.iterrows():
            # Only update records that were explicitly matched
            if 'has_spot_match' in df_lastfm.columns and row['has_spot_match']:
                # Use mbid as the primary key for updates - LastFM tracks always have an mbid
```

```
                cur.execute(
                    "UPDATE lastfm_tracks SET group6_id = %s WHERE mbid = %s",
                    (row['group6_id'], row['mbid'])
                )
                lastfm_updates += cur.rowcount

        print(f"Updated {lastfm_updates} rows in lastfm_tracks")

    # Update AcousticBrainz
    ab_updates = 0
    with conn.cursor() as cur:
        for idx, row in df_ab.iterrows():
            # Only update records with valid mbid
            if pd.notna(row['mbid']):
                cur.execute(
                    "UPDATE acousticbrainz_features SET group6_id = %s WHERE mbid = %s",
                    (row['group6_id'], row['mbid'])
                )
                ab_updates += cur.rowcount

        print(f"Updated {ab_updates} rows in acousticbrainz_features")

    conn.commit()
    print("All database updates committed")



def match_all_tracks():
    df_spot, df_kag, df_lastfm, df_ab = load_all_from_db()

    df_spot, df_kag = match_spot_to_kag(df_spot, df_kag)

    df_spot, df_lastfm = match_spot_to_lastfm(df_spot, df_lastfm)

    df_ab = lastfm_to_ab(df_lastfm, df_ab)

    update_db_after_matching(df_kag, df_lastfm, df_ab)




with DAG(
        'matching_dag',
        default_args=default_args,
        description='Dag for fuzzy matching of spotify, kaggle, lastfm, and acousticbrainz
data',
        schedule_interval=None,
        catchup=False
    ) as matching_dag:

```

```
822     start_task = EmptyOperator(task_id='start')
823 
824     match_all_tracks = PythonOperator(
825         task_id='match_all_tracks',
826         python_callable=match_all_tracks
827     )
828 
829     end_task = EmptyOperator(task_id='end')
830 
831     start_task >> match_all_tracks >> end_task
832 
833 
```