Giorgio Mendoza

CS539-F23-F02

Dr. J. Sethi

Lab 4-2: Databases, Custom Functions, and the Million-Song Dataset

The Million-Song Dataset

The Million-Song Dataset is a collection of metadata and audio features for a million pop songs from the past 50+ years. It's a whopping 280GB to download! But there are a variety of subsets -- and joins with other data sources like user-generated genre tags from Tagtraum and MusicBrainz -- available from their website. It's a great place to start when exploring musical metadata and what goes into an audience's conception of genre.

We'll start by looking at one of the smaller files from that dataset: `artist_term.db`. This is an SQLite database with multiple tables, containing information about user-generated genre tags applied to artists in the Million-Song Dataset (MSD, from now on). Here are the names of the 5 tables contained in that database.

Import the requisite libraries, and create a SQL database engine for `artist_term.db`.

- Import `pandas` as `pd` and `create_engine` from `sqlalchemy`.
- Create a database engine called `engine` from the SQLite database `artist_term.db`.
- Print the table names from that database.

Some Good References

Here is a helpful Pandas cheatsheet to remind you of some of the basic commands and workflows we'll be exploring here. Another great reference for pertinent skills and terms is introduced in Python Data Science Toolbox, Part 1 and and Importing Data in Python, Part 1. Please do feel free to review the slides and exercises there as you go through this project. You can also see some great student projects on this class blog.

```
# Import pandas as pd and create_engine from sqlalchemy
from google.colab import drive
drive.mount('/content/drive')
import random
import numpy as np
import pandas as pd
import csv
from sqlalchemy import create_engine, inspect

path = "/content/drive/MyDrive/Colab Notebooks/"
engine_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"

engine = create_engine(f"sqlite:///{engine_path}")
# Create an inspector
inspector = inspect(engine)

# Get the table names from the database
table_names = inspector.get_table_names()

# Print the table names
print(table_names)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
['artist_mbtag', 'artist_term', 'artists', 'mbtags', 'terms']
Ellipsis
```

Unpacking genre tags

Now let's dig in. The three tables `artists`, `mbtags`, and `terms` contain lists of elements contained in the first two tables, `artist_mbtag` and `artist_term`. We'll dig into the Musicbrainz genre tags linked to artists in the `artist_mbtag` table. Let's now load and display the first few rows of the `artist_mbtag` table.

- Use `read_sql_query()` to select *all* records from the `artist_mbtag` table and store the results in a dataframe of the same name.
- Print the first 5 rows of `artist_mbtag`.

Be sure to double-check the documentation for `read_sql_query()` (or your notes).

```
from google.colab import drive
import sqlite3
import pandas as pd

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Print the first 5 rows of artist_mbtag
print(artist_mbtag.head())

# Close the database connection
conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
             artist_id        mbtag
0  AR002UA1187B9A637D           uk
1  AR002UA1187B9A637D         rock
2  AR002UA1187B9A637D  garage rock
3  AR006821187FB5192B         bass
4  AR00A6H1187FB5402A      detroit
```

Finding artist names

There is some interesting data in here but those `artist_id` labels aren't very helpful. Thankfully, MSD has another file that matches the unique artist IDs to the artist's names.

Import `unique_artists.txt` into a dataframe `artist_meta` and display the first few rows.

- Use `read_csv()` to import `unique_artists.txt` into a dataframe named `artist_meta`.
- According to MSD, the fields in this file are separated by the string `<SEP>`. The file has no header, and we only need the first column (artist ID) and the fourth column (artist name) for our analysis so please set those parameters accordingly.
- Rename the columns to `artist_id` and `artist_name`, respectively.

Remember that `.columns` takes a list. And it will be clear later (if not already) why we need to rename `artist_id` in particular.

Also, note that because of the `<` and `>` in the delimiter, Python may produce a warning message. Don't worry about that, as long as it also produces the expected output. (Which is... -- remember: always evaluate your output before going on, making sure it matches what you predicted or expected, or if you didn't know what to expect, that it makes sense in context once you see it.)

```
from google.colab import drive
import sqlite3
import pandas as pd

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
```

```
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']

# Display the first 5 rows of 'artist_meta'
print(artist_meta.head())
conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-20-fed94b6d5277>:26: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
            artist_id                                        artist_name
0  AR002UA1187B9A637D                                       The Bristols
1  AR003FB1187B994355                                           The Feds
2  AR006821187FB5192B  Stephen Varcoe/Choir of King's College_ Cambri...
3  AR009211187B989185                                   Carroll Thompson
4  AR009SZ1187B9A73F4                                          Gorodisch
```

Merging artist names with their genre tags

Now that we've loaded the SQLite database with artist tags and the `<SEP>`-delimited text file with artist names, we need to join them together to make them useful. Here's what it looks like when we connect artists with the various genres users have attributed to their music.

Join the `artist_mbtag` and `artist_meta` dataframes together, creating a new dataframe `artists` that links each band name with each genre tag users associated with it.

- Use `pd.merge()` to join the two dataframes together. This is a new function but a really important one (see below). The first two arguments should be the names of the two dataframes to join. The `how=` argument specifies the kind of join -- in our case, `inner`.
- Specify the field to merge by with the `on=` argument: `artist_id`. (This is why we renamed the column when we imported it!)
- Remove the no-longer-needed column `artist_id` by subsetting `artists` to include only the `artist_name` and `mbtag` columns.
- Display the head of `artists` to confirm a successful merge.

Were the two tables we're interested in part of the same database, we could use `INNER JOIN` directly in SQL. However, because they are in different formats, and we've now imported them both into `pandas` dataframes, we can use the `pandas` function `merge()` to do the same thing.

We'll just stick with `inner` joins for now, but we'll go over a deeper explanation of different join types later in the course.

```
from google.colab import drive
import sqlite3
import pandas as pd

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']
# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]
```

```
# Display the first 5 rows of 'artist_meta'
print(artists.head())
conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
                                     artist_name        mbtag
0                                    The Bristols           uk
1                                    The Bristols         rock
2                                    The Bristols  garage rock
3  Stephen Varcoe/Choir of King's College_ Cambri...         bass
4                                     The Meatmen      detroit
<ipython-input-21-36f1e125a47e>:26: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
```

Genre tags by band

Let's look up some of our favorite artists, and see which genres they are associated with in MSD and Musicbrainz! Let's create a custom function `tags_for_band()` that returns a list of genre tags for a specified artist.

- This function takes two arguments: `dataframe` and `band`. Assuming a user provides the dataframe `artists`, define `tags` so that it returns a list of all the tags associated with the specified `band`.
- Test it out with a few different artist names, and see if the results make sense to you. One example in the dataset is provided.

```
from google.colab import drive
import sqlite3
import pandas as pd

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']
# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]

def tags_for_band(dataframe, band):
    """From a dataframe and a band name, return a list of tags for that band."""
    tags = dataframe[dataframe['artist_name'] == band]['mbtag'].tolist()
    return tags

# Test the function with a few different artist names
print("Tags for David Bowie:", tags_for_band(artists, 'David Bowie'))
print("Tags for The Beatles:", tags_for_band(artists, 'The Beatles'))
print("Tags for Led Zeppelin:", tags_for_band(artists, 'Led Zeppelin'))

conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-36-5528a9a200c2>:23: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
Tags for David Bowie: ['british', 'uk', 'rock', 'experimental', 'glam rock', 'english', 'classic pop and rock', 'post-disco', 'britannic
Tags for The Beatles: ['british', 'rock', 'pop', 'british invasion', '60s', 'heavy metal', 'classical pop', 'instrumental pop', 'folk-ro
Tags for Led Zeppelin: ['hard rock', 'rock', 'blues rock', 'british', 'english', 'classic pop and rock', 'uk', 'classic rock', 'folk',
```

Bands by genre tag

Now let's look up some of our favorite genres and see what artists are represented. Using the same pattern as in the previous task, create a new custom function that returns all the bands associated with a given genre tag.

```
from google.colab import drive
import sqlite3
import pandas as pd

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']
# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]

def bands_for_tag(dataframe, tag):
    """From a dataframe and a genre tag, return a list of bands associated with that tag."""
    bands = dataframe[dataframe['mbtag'] == tag]['artist_name'].tolist()
    return bands

print("Bands for glam rock:", bands_for_tag(artists, 'glam rock'))
print("Bands for punk rock:", bands_for_tag(artists, 'punk rock'))
print("Bands for hip hop:", bands_for_tag(artists, 'hip hop'))

conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-35-05ced5bd6413>:23: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
Bands for glam rock: ['Sparks', 'Alice Cooper', 'Iggy Pop', 'Goldfrapp', 'Whitesnake', 'Simple Kid', 'Sweet', 'David Bowie', 'Queen', 'E
Bands for punk rock: ['NOFX', 'Me First And The Gimme Gimmes', 'Deborah Harry', 'The Clash', 'Die Toten Hosen', 'AFI', 'The Lemonheads',
Bands for hip hop: ['D-Sisive', 'Eazy-E', 'The Weathermen', 'Naughty By Nature', 'Freundeskreis', 'Meanest Man Contest', 'MC Eiht', 'Brc
```

The most represented artists and genres

Now that we've explored some of our favorites, let's see which artists and which genre tags are most common in this dataset. Let's find, and then display, the number of occurrences of each artist name and each genre tag in the `artists` dataframe.

- Define `top_tags` and `top_artists` using the `.value_counts()` method learned in the Ramen lab.

```
from google.colab import drive
import sqlite3
import pandas as pd

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)
```

```
# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']
# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]

def bands_for_tag(dataframe, tag):
    """From a dataframe and a genre tag, return a list of bands associated with that tag."""
    bands = dataframe[dataframe['mbtag'] == tag]['artist_name'].tolist()
    return bands

# Find the most common tags and artists in the dataset
top_tags = artists['mbtag'].value_counts()
top_artists = artists['artist_name'].value_counts()

# Print the results
print("Top Genre Tags:")
print(top_tags)

print("\nTop Artists:")
print(top_artists)
conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-34-19db7b7f336f>:23: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
Top Genre Tags:
classic pop and rock    1073
american                1027
uk                      1013
british                  975
rock and indie           920
                        ...
rottun                     1
dubtronica                 1
serbian                    1
roger waters               1
slam                       1
Name: mbtag, Length: 2321, dtype: int64

Top Artists:
Moby                 22
Indidginus           20
Yodelice             20
Musetta              20
Prodigy              20
                     ..
Gayle San             1
Alek Szahala          1
I Haunt Wizards       1
Orchestra Baobab      1
Scouting for Girls    1
Name: artist_name, Length: 8797, dtype: int64
```

The tags of genre-transcending artists

Some artists hav more genre tags than others. In some cases, this is a reflection of popularity -- more popularity leads to more listeners adding their two cents about genre to a crowd-sourced database. In some cases, this is a reflection of an artist's ability to bridge or transcend genre boundaries.

Either way, let's see what genre tags are most common for the artists with the longest list of genre associations (20 or more). Maybe that will tell us something about their music -- or about music in general, and the relative rigidity of some genres. Let's find the genre tags most

associated with the 12 artists most heavily tagged in the corpus (those with 20 genre tags or more).

- Import the `Counter` function from the package `collections` (these are case-sensitive)
- Create an empty list `tags_of_popular_bands`.
- Loop through the 12 bands at the beginning of `top_artists`. (Note the code used here, as you'll need it later -- why do you think it might be necessary?)
- Find the tags associated with each band (there's a function for that!) and add them to `tags_for_popular_bands`.
- Use the `Counter()` function to count the number of occurrences of each tag in `tags_for_popular_bands`. Print the results.

```
from google.colab import drive
import sqlite3
import pandas as pd
from collections import Counter

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']

# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]

# Find the most common tags and artists in the dataset
top_tags = artists['mbtag'].value_counts()
top_artists = artists['artist_name'].value_counts()

# Create an empty list tags_of_popular_bands
tags_of_popular_bands = []

# Loop through the 12 most common artists
for band in top_artists.index.values[:12]:
    # Find the tags represented by those bands and add them to tags_of_popular_bands
    tags = artists[artists['artist_name'] == band]['mbtag'].tolist()
    tags_of_popular_bands.extend(tags)

# Use Counter to count the occurrences of each tag
tag_counts = Counter(tags_of_popular_bands)

# Print the results
print("\nGenre Tags Associated with the 12 Most Common Artists:")
for tag, count in tag_counts.items():
    print(f"{tag}: {count}")

conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-29-384e04be234c>:27: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex s
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])

Genre Tags Associated with the 12 Most Common Artists:
rap: 2
electronic: 3
american: 2
ambient: 1
house: 1
downtempo: 2
techno: 2
electronica: 4
```

```
harlem: 1
darien: 1
dance and electronica: 2
united states: 2
producteur: 2
producer: 3
compositeur: 3
composer: 3
parolier: 3
rock: 6
lyricist: 3
rave: 2
pop: 4
psychedelic trance: 1
world fusion: 1
didgeridoo: 1
idm: 1
organica: 1
indidginus: 1
electronic music: 1
organic soundscapes: 1
didg: 1
music: 1
chillout: 2
downbeat: 1
psybient: 1
psytrance: 1
progressive psytrance: 1
dubstep: 1
slide didgeridoo: 1
slide didg: 1
folk: 2
france: 1
french: 1
français: 1
guitarist: 1
guitariste: 1
bassiste: 1
chanteur: 2
francophone: 1
singer: 2
keyboardist: 1
bassist: 1
claviériste: 1
```

Genre pairings

That's not particularly instructive, so let's look at what genres are paired with each other throughout the entire dataset! To begin, here is every unique pair of genres associated with The Beatles.

Define a function `get_tag_pairs()` that returns a list of all unique pairs of genres associated with a given band. Print the result for The Beatles.

- This function takes two arguments: `dataframe` and `band`. Assuming a user provides the dataframe `artists`, define `tags` so that it returns a list of all the tags associated with the specified `band`. (This should sound familiar)
- Complete the `if` statement so that it is only true when the two tags are different, and in alphabetical order. (This will ensure that we don't count `rock-rock` as a "pair", and that `rock-pop` and `pop-rock` are not considered *different* pairs.)
- Inside the `if` statement, create a string that concatenates `tag` with `tag2`, with `&` in between, and add that string to the `tag_pairs` list.
- Return `tag_pairs`.

The looping syntax is provided because it's somewhat complex but please make sure you understand it before you go on to the next task.

```
from google.colab import drive
import sqlite3
import pandas as pd
from collections import Counter

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
```

```
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']

# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]

# Find the most common tags and artists in the dataset
top_tags = artists['mbtag'].value_counts()
top_artists = artists['artist_name'].value_counts()

# Create an empty list tags_of_popular_bands
tags_of_popular_bands = []

# Loop through the 12 most common artists
for band in top_artists.index.values[:12]:
    # Find the tags represented by those bands and add them to tags_of_popular_bands
    tags = artists[artists['artist_name'] == band]['mbtag'].tolist()
    tags_of_popular_bands.extend(tags)

# Use Counter to count the occurrences of each tag
tag_counts = Counter(tags_of_popular_bands)

def get_tag_pairs(dataframe, band):
    """Return list of pairs of tags associated with the same band."""

    tags = dataframe[dataframe['artist_name'] == band]['mbtag'].tolist()
    tag_pairs = []

    for tag in tags:
        for tag2 in tags:
            if tag != tag2 and tag < tag2:
                pair = f"{tag} & {tag2}"
                tag_pairs.append(pair)

    return tag_pairs

print(get_tag_pairs(artists, 'The Beatles'))

conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-30-1969b27f2f9b>:27: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
['british & rock', 'british & pop', 'british & british invasion', 'british & heavy metal', 'british & classical pop', 'british & instrum
```

Genre pairings - the whole kit and caboodle

That's a lot of genre pairings just for one band! Ironically, it's simpler to look at the genre pairings for the whole dataset, and get a handle on which ones tend to go together most frequently. Again, this is a result of both the relationship between the two genres *and* the relative popularity of both genre tags.

With that context in mind, let's take a look! Let's use the new `get_tag_pairs()` function on all artists in the dataset, and display the most common genre tag pairings in descending order.

- Use `set()` and `list()` together to reduce `artists['artist_name']` to a list of unique artist names, each occurring only once.
- Define an empty list `all_tag_pairs`.
- While looping through the unique band names in `artist_names`, use `get_tag_pairs()` to retrieve all genre tag pairings for each band and add them to `all_tag_pairs`.
- Use `Counter()` to count the number of occurrences of each genre pairing in `all_tag_pairs`, and assign it to a Pandas series `tag_pair_count`.
- Display `tag_pair_count`. Apply the method `.sort_values()` with the argument `ascending=False` to display them in descending order from the most to the least common.

There are a couple of new things here. First, we're taking the dictionary that `Counter()` produces and assigning it to a `pandas` series. This will allow us both to display it in a more user-friendly way, and to more easily sort it so our output is more meaningful to read.

Also, the `.sort_values()` function allows us to take that `pandas` series and sort it with a single, easy-to-use method. This is another one worth adding to your notes, even though we haven't met it in detail before.

```
from google.colab import drive
import sqlite3
import pandas as pd
from collections import Counter

drive.mount('/content/drive')
db_path = "/content/drive/MyDrive/Colab Notebooks/artist_term.db"
csv_path = "/content/drive/MyDrive/Colab Notebooks/unique_artists.txt"

# Connect to the SQLite database
conn = sqlite3.connect(db_path)

# Create a cursor object
cursor = conn.cursor()

# Execute a query to fetch all records from the artist_mbtag table
query = "SELECT * FROM artist_mbtag"
cursor.execute(query)

# Fetch all results into a DataFrame
artist_mbtag = pd.DataFrame(cursor.fetchall(), columns=[column[0] for column in cursor.description])

# Load a dataframe that connects artist IDs with artist names, and rename columns
artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
artist_meta.columns = ['artist_id', 'artist_name']

# Merge artist tags with artist names, joining on the common field 'artist_id'
artists = pd.merge(artist_mbtag, artist_meta, how='inner', on='artist_id')

# Clean artists, removing the no longer needed column 'artist_id'
artists = artists[['artist_name', 'mbtag']]

# Find the most common tags and artists in the dataset
top_tags = artists['mbtag'].value_counts()
top_artists = artists['artist_name'].value_counts()

# Create an empty list tags_of_popular_bands
tags_of_popular_bands = []

# Loop through the 12 most common artists
for band in top_artists.index.values[:12]:
    # Find the tags represented by those bands and add them to tags_of_popular_bands
    tags = artists[artists['artist_name'] == band]['mbtag'].tolist()
    tags_of_popular_bands.extend(tags)

# Use Counter to count the occurrences of each tag
tag_counts = Counter(tags_of_popular_bands)

def get_tag_pairs(dataframe, band):
    """Return list of pairs of tags associated with the same band."""

    tags = dataframe[dataframe['artist_name'] == band]['mbtag'].tolist()
    tag_pairs = []

    for tag in tags:
        for tag2 in tags:
            if tag != tag2 and tag < tag2:
                pair = f"{tag} & {tag2}"
                tag_pairs.append(pair)

    return tag_pairs

# Get a list of unique artist names
artist_names = list(set(artists['artist_name']))

# Define an empty list `all_tag_pairs`
all_tag_pairs = []

# Loop through the bands in artist_names
```

```
for band in artist_names:
    # Get all tag pairs for each artist and adding them to all_tag_pairs
    tag_pairs = get_tag_pairs(artists, band)
    all_tag_pairs.extend(tag_pairs)

# Count the occurrences of each genre tag pair in all_tag_pairs, and assign the results to a pandas series
tag_pair_count = pd.Series(Counter(all_tag_pairs))

# Sort and display the results in descending order
tag_pair_count = tag_pair_count.sort_values(ascending=False)

# Print the top genre tag pairings
print(tag_pair_count)

conn.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<ipython-input-32-1bc135a88caf>:27: ParserWarning: Falling back to the 'python' engine because the 'c' engine does not support regex sep
  artist_meta = pd.read_csv(csv_path, sep='<SEP>', header=None, usecols=[0, 3])
british & uk                  752
british & english             608
english & uk                  593
american & rock               312
classic pop and rock & uk     245
                             ...
accordion & male vocalists      1
accordion & mexico              1
accordion & cumbia              1
accordion & vallenato           1
80s & ska                       1
Length: 19829, dtype: int64
```

Conclusion

Looks like American and British/English music tops the charts in this dataset and is strongly associated with rock, pop, punk, and indie music. It would be interesting to tease out all of the UK- and US-specific terms and make a more direct comparison -- are some genres more "English-like" (punk or classic rock, perhaps?) and others more "American-like" (alternative or jazz, maybe?). We'll save that for a future project!