The Lyrics We Live In
Text Mining

Project Flow

[Data Acquisition and Extraction](#)

[Part1_Crawling.ipynb](#)

[Part3.1_Clustering&Evaluation.ipynb](#)

Part3.3.1_SoftGridSearch.ipynb
Part3.3.2_HardGridSearch.ipynb

Part3.3.3_LDAGridSearch.ipynb

Part4.1_SimilaritySearch.ipynb

Part4.2_TitlevsLyrics.ipynb

Each cell in all Jupyter notebooks is documented with a Markdown cell describing its functionality, and all code is commented accordingly.
In addition, in the end of each notebook includes a Markdown cell with conclusions and final thoughts.

The Dockerfile and Scraper.py are included in the PyCharm project. You can either build the image from theDockerfile, or use the pre-built compressed image myscraper.tar.gz included in the Docker Folder
To install the image on your local Docker, run the following comman-line:

docker load -i myscraper.tar.gz

Once the image is installed locally,in the docker you can run the scraper with command-line:

docker run --rm -v C:/LyricsTextMining/Songs:/songs myscraper python Scraper.py --start_page 1 --end_page 100 --genre grunge --output 50_100_grunge.csv

Where:

- C:/LyricsTextMining/Songs → The location where the CSV file will be saved.
- -- start_page / -- end_page → The range of songs pages you want to crawl from website .
- -- genre → Optional filter by music genre.
- -- output → The name of the output CSV file.

```
Scraper.py: error: argument --genre: expected one argument
PS C:\Users\iborg>  docker run --rm -v C:/LyricsTextMining/Songs:/songs myscraper python Scraper.py --start_page 1 --end
_page 5  --output test.csv
Logged in successfully - LyricsDB detected!
Scraping page 1: 20 songs collected.
Scraping page 2: 40 songs collected.
Scraping page 3: 60 songs collected.
Scraping page 4: 80 songs collected.
Scraping page 5: 100 songs collected.
Saved 100 songs to /songs/test.csv
```

This is an example of the command line used by crawling 5 pages , donot use genre filter and save as test.csv in the ./Songs directory. Since one page contains 20 songs only 100 songs were saved.

- Songs
  - 50_100_grunge.csv
  - crawled_songs.csv
  - scraped_songs.csv
  - test.csv

Part1_Crawling.ipynb | Part2.PreProcessing.ipynb | test.csv

	song_id	dist	artists	track_name	duration_ms	danceability
1	10298556365139914874	0	Bruno Mars	Moonshine	228573	
2	10018510778896022773	0	Bee Gees	How Can You Mend A Broken Heart	237826	
3	10413324139234907468	0	Lionel Richie	Angel	221853	
4	104481365335039022274	0	Bee Gees	I Started A Joke	188306	
5	10680489627552594443	0	Gloria Gaynor	I Will Survive	296106	
6	10751323331229108203	0	Chaka Khan	Any Old Sunday	217533	
7	1007611027555825226	0	Alice In Chains	Bleed The Freak	242600	
8	10133778154258328315	0	Nirvana	You Know You're Right	217000	
9	10223078488567371546	0	Nirvana	Pennyroyal Tea	218813	
10	10295951250721898025	0	Linkin Park	Hands Held High	233194	
11	10373816212121378580	0	Foo Fighters	Long Road To Ruin	224880	

Part1_Crawling.ipynb Connects to the website, crawls the songs, then into a crawled_songs.csv file.
That CSV can be generated either from the Docker container or directly from Jupyter.

```
    # execute function
    # ------------------------------
scrape_songs(max_pages=100)

✓ [2] 15s 219ms
  Scraping page 66: 1320 songs collected.
  Scraping page 67: 1340 songs collected.
  Scraping page 68: 1360 songs collected.
  Scraping page 69: 1380 songs collected.
  Scraping page 70: 1400 songs collected.
  Scraping page 71: 1420 songs collected.
  Scraping page 72: 1440 songs collected.
  Scraping page 73: 1460 songs collected.
  Scraping page 74: 1480 songs collected.
  Scraping page 75: 1500 songs collected.
  Scraping page 76: 1520 songs collected.
  Scraping page 77: 1540 songs collected.
  Scraping page 78: 1558 songs collected.
  All songs scraped successfully.
  Saved 1558 songs to Songs/crawled_songs.csv
```

The crawled_songs.csv is saved into .\Songs Folder

```
> SimilarityData
v Songs
    50_100_grunge.csv
    crawled_songs.csv
    my_freind.csv
    scraped_songs.csv
> SQL_DB
```

Sample of the csv file crawled with the jupyter

	song_id	dist	artists	track_name	duration_ms	danceability	energy	loudne
1	10018510778896022773	0	Bee Gees	How Can You Mend A Broken Heart	237826	0.335	0.262	
2	10062984395933961247	0	Foo Fighters	The Pretender	269373	0.433	0.9590000000000001	
3	10070033369023809680	0	Status Quo	In The Army Now	281746	0.686	0.722	
4	1007611027555825226	0	Alice In Chains	Bleed The Freak	242600	0.245	0.749	
5	10081762227510093749	0	Billy Joel	Piano Man	339000	0.331	0.55	
6	10081827871701357287	0	Steely Dan	Deacon Blues	454560	0.751	0.545	
7	10085378244052961164	0	Opeth	Hope Leaves	267146	0.501	0.48	
8	10094682406723622078	0	Marilyn Manson	mOBSCENE	205720	0.548	0.9540000000000001	
9	10133778154258328315	0	Nirvana	You Know You're Right	217000	0.289	0.716	
10	1014387586379463564	0	Nat King Cole	Nature Boy	174800	0.266	0.181	
11	10155933489062027826	0	Taylor Swift	Love Story	236266	0.617	0.741	-3.96
12	10166143888715534218	0	Michael W. Smith	Ancient Words	276066	0.23500000000000001	0.20800000000000002	

```python
with open("SQL_DB/db_config.txt", "r") as f:
    db_target = f.read().strip()

try:
    # Connect to SQL Server
    server, database = db_target.split("/")
    conn = pyodbc.connect(
    fr"DRIVER={{ODBC Driver 17 for SQL Server}};"
    fr"SERVER={server};"
    fr"DATABASE={database};"
    r"Trusted_Connection=yes;"
    )
```

Note: All functions in the Jupyter notebooks pull their data from this database. If you want the notebooks to run on your local machine, you will need to update the string stored into \\Sql_DB\db_config.txt accordingly

Just change the credential

SERVER=IVAN_PC\SQLEXPRESS

to match the SQL server on your local machine.

```
> Songs
v SQL_DB
    db_config.txt
    songs
```

The second cell of Part1_Crawling.ipynb , extracts the data from the CSV then imported into an SQL database called TextMiningHA, which contains a single table named songs.

```
[3] 6s 852ms

Connection successful and table ready.
Inserted 1558 new songs.
Connection closed.
```

161 %

Results | Messages

	song_id	name	artist	lyrics	genre	cleanGenre	tokenised	cleanTokens
1	10018510778896022773	How Can You Mend A Broken Heart	Bee Gees	I can think of younger days when living for my life ...	disco	NULL	NULL	NULL
2	10062984395933961247	The Pretender	Foo Fighters	Keep you in the dark You know they all pretend ...	alt-rock, alternative, grunge, metal	NULL	NULL	NULL
3	10070033369023809680	In The Army Now	Status Quo	A vacation in the foreign land Uncle Sam does th...	blues, british, hard-rock	NULL	NULL	NULL
4	1007611027555825226	Bleed The Freak	Alice In Chains	My cup runneth over Like blood from a stone T...	grunge	NULL	NULL	NULL
5	10081762227510093749	Piano Man	Billy Joel	It's nine o'clock on a Saturday The regular crowd ...	folk, piano, singer-songwriter, songwriter	NULL	NULL	NULL
6	10081827871701357287	Deacon Blues	Steely Dan	This is the day of the expanding man That shape i...	psych-rock	NULL	NULL	NULL
7	10085378244052961164	Hope Leaves	Opeth	In the corner beside my window Hangs a lonely p...	death-metal	NULL	NULL	NULL
8	10094682406723622078	mOBSCENE	Marilyn Manson	"Ladies and Gentlemen" We are the thing of sha...	industrial	NULL	NULL	NULL
9	10133778154258328315	You Know You're Right	Nirvana	I would never bother you I would never promise to...	grunge	NULL	NULL	NULL
10	1014387586379463564	Nature Boy	Nat King Cole	There was a boy A very strange enchanted boy ...	jazz	NULL	NULL	NULL
11	10155933489062027826	Love Story	Taylor Swift	We were both young when I first saw you I close ...	pop	NULL	NULL	NULL
12	10166143888715534218	Ancient Words	Michael W. Smith	Holy words long preserved For our walk in this wo...	world-music	NULL	NULL	NULL
13	10175118850676509515	Deep Six	Marilyn Manson	You wanna know what Zeus said to Narcissus Yo...	industrial	NULL	NULL	NULL
14	10189849469444300730	ROADS UNTRAVELED	Linkin Park	Weep not for roads untraveled Weep not for path...	alternative, grunge, metal	NULL	NULL	NULL
15	10198179906347417599	War Pigs	Faith No More	Generals gathered in their masses Just like witche...	alt-rock, alternative, funk, hard-rock, ...	NULL	NULL	NULL
16	10223078488567371546	Pennyroyal Tea	Nirvana	I'm on my time with everyone I have very bad post...	grunge	NULL	NULL	NULL
17	10229158451427325947	Red Red Wine	UB40	Red, red wine goes to my head, Makes me forget ...	reggae	NULL	NULL	NULL
18	1023597599474373863	Cocaine	Eric Clapton	If you want to hang out, you've gotta take her out, ...	blues	NULL	NULL	NULL
19	10236900408961779514	Best Guitar Picker	George Jones	I didn't think too much about books and schoolin' ...	honky-tonk	NULL	NULL	NULL
20	10241430274867347826	A Home In Heaven	Hank Williams	Around me many are building Homes of beauty an...	honky-tonk	NULL	NULL	NULL

Query executed successfully. | IVAN_PC\SQLEXPRESS (16.0 RTM) | IVAN_PC\iborg (53) | TextMiningHA | 00:00:00 | 1,558 rows

From the crawled_songs.csv only the song_ID, name, artist ,lyrics and genre are extracted with a total of 1558 songs.

`Part2_PreProcessing notebook` handles preprocessing tasks, including text cleaning and feature analysis.

```
#Tokenize all lyrics (raw tokens)
tokenize_all_lyrics()

# Clean tokenized lyrics for NLP (removes stopwords, noise, short words, etc.)
clean_lyrics_tokens(1, 1600, show_output=0)  # Set show_output=1 to view original

✓ [1] 12s 87ms

Basic tokenization process ready filled in tokenised column songsdb.
Polished tokens process ready filled in cleanTokens column songsdb.
```

Step 1: Clean the raw lyrics into basic tokens.
Step 2: Polish all tokens using regex and stopwords.
Note: In the call, an option is available to display the raw lyrics, basic tokens, and polished tokens.

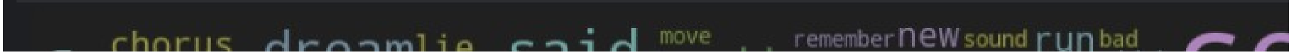
tokenised	cleanTokens
["i", "can", "think", "of", "younger", "days", "wh...	["i", "think", "younger", "days", "living", "my", "life...
["keep", "you", "in", "the", "dark", "you", "know...	["keep", "you", "dark", "you", "know", "they", "pr...
["a", "vacation", "in", "the", "foreign", "land", "u...	["vacation", "foreign", "land", "uncle", "sam", "be...
["my", "cup", "runneth", "over", "like", "blood", "...	["my", "cup", "runneth", "like", "blood", "stone", "...
["it", "is", "nine", "of", "the", "clock", "on", "a", ...	["nine", "clock", "saturday", "regular", "crowd", "s...
["this", "is", "the", "day", "of", "the", "expanding...	["day", "expanding", "man", "shape", "my", "shad...
["in", "the", "comer", "beside", "my", "window", ...	["comer", "beside", "my", "window", "hangs", "lon...
["ladies", "and", "gentlemen", "we", "are", "the",...	["ladies", "gentlemen", "thing", "shapes", "come", ...
["i", "would", "never", "bother", "you", "i", "woul...	["i", "would", "never", "bother", "you", "i", "would...
["there", "was", "a", "boy", "a", "very", "strange...	["boy", "strange", "enchanted", "boy", "they", "sa...
["we", "were", "both", "young", "when", "i", "firs...	["young", "i", "first", "saw", "you", "i", "close", "m...
["holy", "words", "long", "preserved", "for", "our"...	["holy", "words", "long", "preserved", "our", "walk...
["you", "want", "to", "know", "what", "zeus", "s...	["you", "want", "know", "zeus", "said", "narcissus...
["weep", "not", "for", "roads", "untraveled", "we...	["weep", "not", "roads", "untraveled", "weep", "n...
["generals", "gathered", "in", "their", "masses", "...	["generals", "gathered", "masses", "like", "witches...
["i", "am", "on", "my", "time", "with", "everyone"...	["i", "my", "time", "everyone", "i", "bad", "posture"...
["red", "red", "wine", "goes", "to", "my", "head",...	["red", "red", "wine", "goes", "my", "head", "make...
["if", "you", "want", "to", "hang", "out", "you", "...	["if", "you", "want", "hang", "you", "got", "take", "...
["i", "did", "not", "think", "too", "much", "about",...	["i", "not", "think", "much", "books", "schoolin", "i...

IVAN_PC\SQLEXPRESS (16.0 RTM) | IVAN_PC\iborg (53) | TextMiningHA | 00:00:00 | 1,558 rows

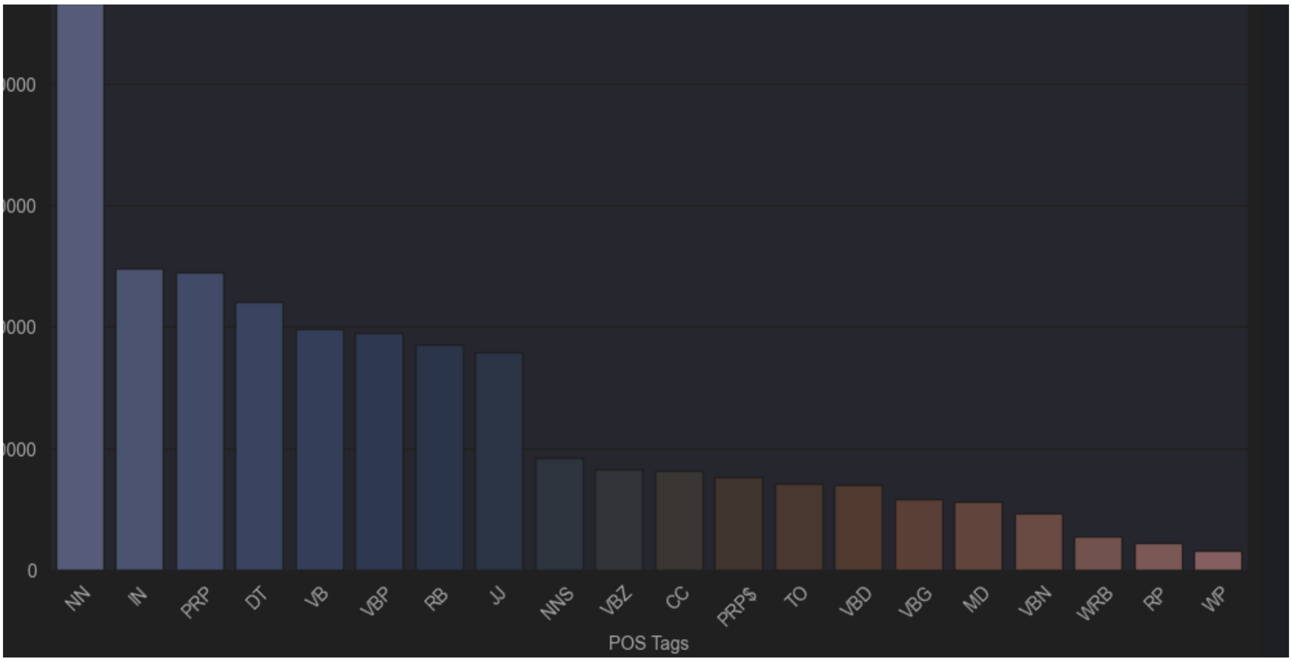
Search Error List

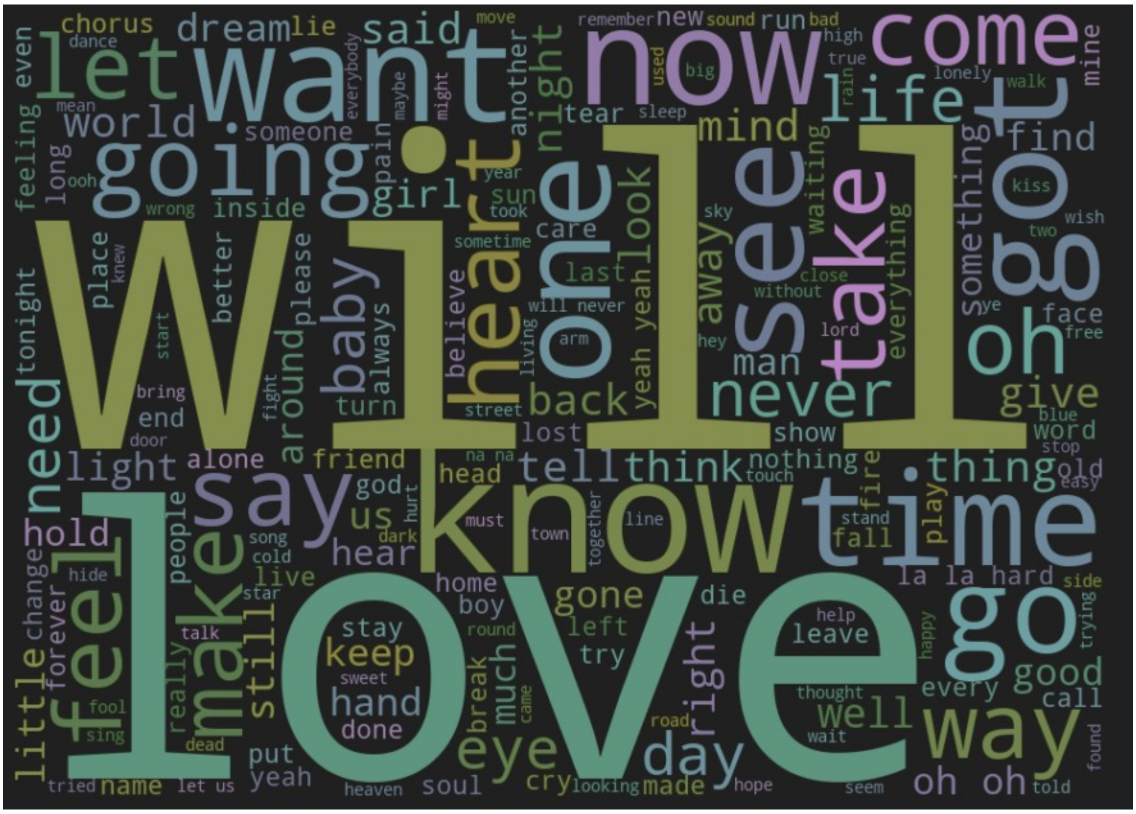
With these steps, the database is filled by inserting the tokenized data. The basic tokens are stored in the tokenised column, and the polished tokens are stored in the cleanTokens column of the songs SQL table.

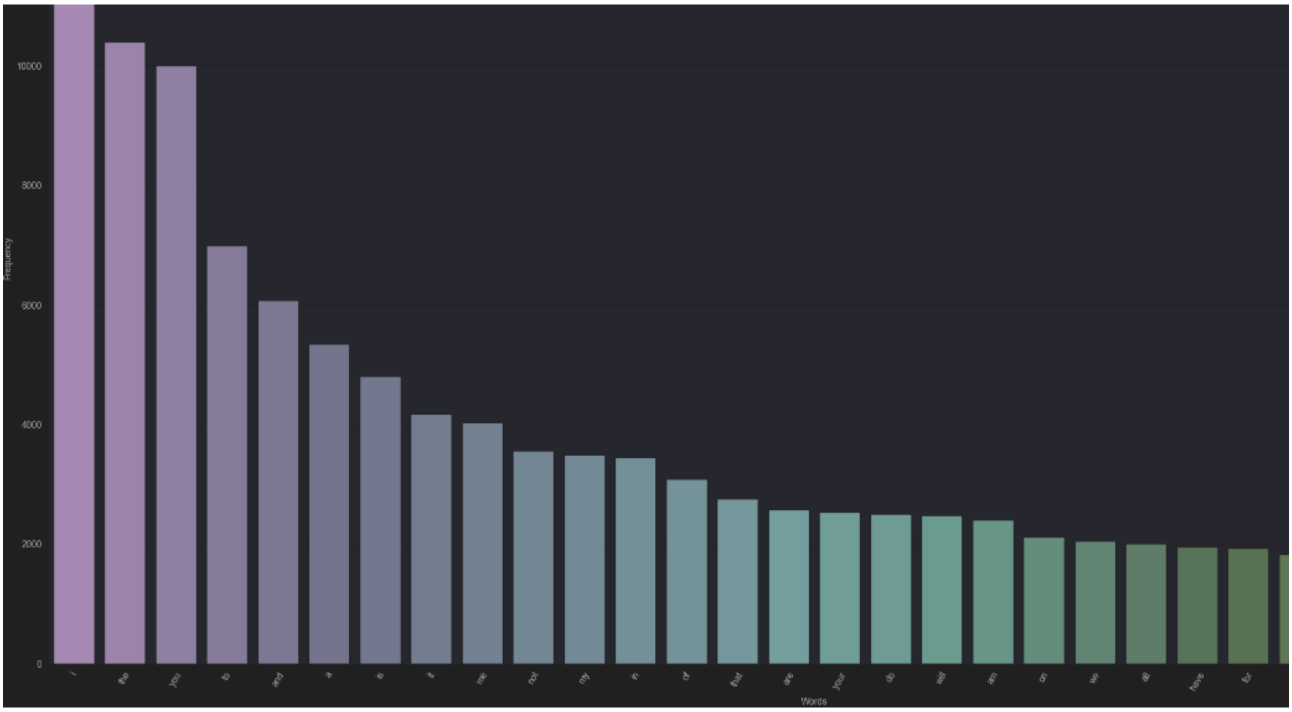
```
# ------------------------------
# Execute analysis functions
# ------------------------------
tokens = fetch_tokenized_lyrics()              # Fetch all raw tokenized lyrics
plot_wordcloud(tokens)                         # generate word cloud
plot_top_word_frequencies(tokens, top_n=30)   # plot top 30 words
print(vocabulary_diversity(tokens))            # print vocabulary metrics
plot_pos_distribution(tokens, top_n=20)       # POS tag distribution plot
fetch_cleaned_tokens()                         # Fetch and print unique cleaned tokens
```



In another cell, functions were added to fetch all raw tokenized lyrics, along with various tools to check and analyze the preprocessing and word features.





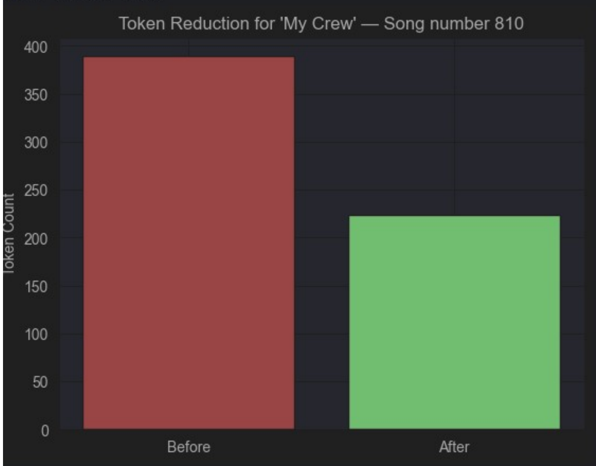



```
'dean' 'dear' 'dearly' 'death' 'deathbed' 'deaths' 'debating' 'debris' 'debt' 'debts' 'debut' 'decade' 'decaf' 'decapitated' 'decay' 'decays' 'deceit'
'deceive' 'deceived' 'deceivers' 'deceives' 'deceivin' 'deceiving' 'december' 'decembers' 'decendants' 'decent' 'deception' 'deceving' 'decide' 'decided'
'decides' 'decision' 'decisions' 'deck' 'decked' 'declare' 'declared' 'decline' 'declined' 'declines' 'deco' 'decorate' 'decorated' 'decorations' 'decree'
'dedicate' 'dedicated' 'dedication' 'deed' 'deeds' 'deejay' 'deep' 'deeper' 'deepest' 'deeply' 'deer' 'defeat' 'defeated' 'defecate' 'defection' 'defences'
'defend' 'defense' 'defenseless' 'define' 'defined' 'definite' 'deflower' 'deformation' 'defunkt' 'defying' 'degen' 'degrading' 'degree' 'degrees' 'deh'
'deity' 'deja' 'del' 'delayed' 'deliberate' 'delicate' 'delicious' 'delight' 'delights' 'delilah' 'delirium' 'deliver' 'deliverance' 'delivered' 'delusion'
'delusions' 'dem' 'demanded' 'demanding' 'demands' 'demented' 'demigods' 'demise' 'democracy' 'demolition' 'demon' 'demons' 'demonstrate' 'den' 'deng'
'denial' 'denied' 'denies' 'dennis' 'deny' 'denying' 'deo' 'depart' 'departed' 'departure' 'depend' 'dependable' 'dependent' 'depending' 'depends'
'depraving' 'depressed' 'depression' 'deptford' 'depth' 'depths' 'deputy' 'der' 'deranges' 'des' 'descend' 'descending' 'descends' 'descent' 'describe'
'desecrate' 'desecrated' 'desert' 'deserts' 'deserve' 'deserves' 'deserving' 'design' 'designed' 'desire' 'desired' 'desiree' 'desires' 'desirous' 'desk'
'desolate' 'desolation' 'despair' 'desperate' 'desperately' 'desperation' 'despise' 'despite' 'dessert' 'destination' 'destinations' 'destined' 'destiny'
'destroy' 'destroyer' 'destroying' 'destruct' 'destructing' 'destruction' 'destructive' 'detached' 'detain' 'detect' 'detective' 'determined' 'detonate'
'detroit' 'detuned' 'deuce' 'deus' 'deva' 'devalues' 'devastated' 'devastates' 'devastation' 'device' 'deviens' 'devil' 'devila' 'deville' 'devils' 'devious
```

```python
# --- Run analysis on 20 random songs ---
# Pick 20 random row offsets between 1 and 100 (change range
random_rows = random.sample(range(560, 900), 20)
for row in random_rows:
    analyze_row(row)
```

Another tool was implimneted, that can pick random songs based on the arguments passed to the function. It returns a visualization of the randomly selected songs, showing how much token reduction occurred and which tokens were removed.

```
Added tokens: set()
```




```
Before: 389 tokens
After: 224 tokens
Reduction: 165 tokens (42.4%)
Removed tokens: {'mi', 'your', 'all', 'eh', 'fi', 'have', 'd', 'her', 'because', 'over', 'out', 'go', 'when', 'again', 'ey', 'and', 'oh', 'what', 'that'
'up', 'hey', 'at', 'di', 'so', 'we', 'from', 'it', 'can', 'ya', 'do', 'the', 'to', 'yo', 'a'}
```

Part3.1_Clustering&Evaluation.– applies three clustering methods, then visualizes and evaluates the results for each clustering method.

```
✓ [1] 2s 336ms
    Genre cleaning completed: cleanGenre populated using rarest-genre rule.
```

I created a more meaningful genre and tag structure by cleaning the provided genre data, since most entries were duplicated and many songs had multiple genres assigned.

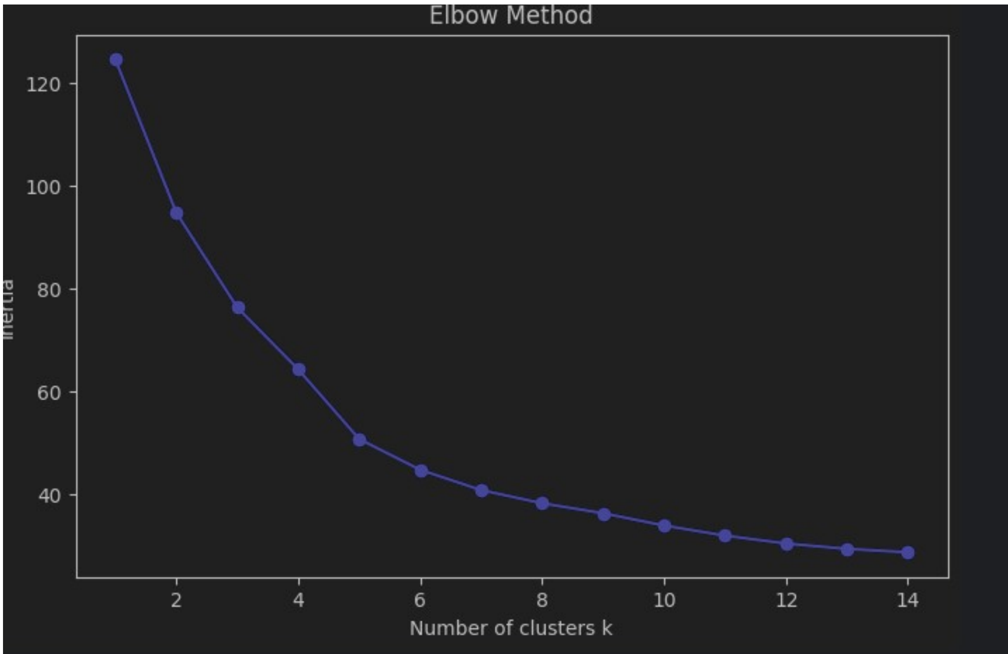
	genre	cleanGenre	tokenised
...	disco	disco	["i", "can", "think", "of
...	alt-rock, alternative, grunge, metal	alt-rock	["keep", "you", "in", "t
n...	blues, british, hard-rock	british	["a", "vacation", "in", "
T...	grunge	grunge	["my", "cup", "runneth
d ...	folk, piano, singer-songwriter, songwriter	folk	["it", "is", "nine", "of",
e i...	psych-rock	psych-rock	["this", "is", "the", "day
o...	death-metal	death-metal	["in", "the", "corner", "
a...	industrial	industrial	["ladies", "and", "gentl
to...	grunge	grunge	["i", "would", "never",
...	jazz	jazz	["there", "was", "a", "b
e ...	pop	pop	["we", "were", "both",
o...	world-music	world-music	["holy", "words", "long
'o...	industrial	industrial	["you", "want", "to", "k
h...	alternative, grunge, metal	alternative	["weep", "not", "for", "
ie...	alt-rock, alternative, funk, hard-rock, metal	funk	["generals", "gathered"
st...	grunge	grunge	["i", "am", "on", "my",
et ...	reggae	reggae	["red", "red", "wine", "
...	blues	blues	["if", "you", "want", "to
...	honky-tonk	honky-tonk	["i", "did", "not", "think

IVAN_PC\SQLEXPRESS (16.0 RTM) | IVAN_PC\

The cleaned genres are saved into the `cleanGenre` column of the `songs` table in SQL.

The `cleanTokens` from each song are transformed into numerical feature vectors using TF-IDF vectorization, enabling downstream tasks such as similarity analysis, clustering,

```
                                  LSA_1     LSA_2     LSA_3     LSA_4  \
name
How Can You Mend A Broken Heart  0.338857  0.031190  0.026390 -0.049272
The Pretender                    0.547757  0.056546 -0.102030 -0.003852
In The Army Now                  0.385917  0.376086  0.140731 -0.036963
Bleed The Freak                  0.450868  0.228416 -0.044728  0.019781
Piano Man                        0.387970  0.009719  0.342692 -0.294778

                                  LSA_5
name
How Can You Mend A Broken Heart -0.141624
The Pretender                    0.070484
In The Army Now                  0.049524
Bleed The Freak                  0.029892
Piano Man                        0.092591
Total unique words (features): 786
['across', 'afraid', 'air', 'alive', 'alone', 'alone i', 'along', 'alright', 'always', 'angel', 'another', 'anymore
'around', 'around i', 'around you', 'ask', 'away', 'away i', 'away you', 'babe', 'baby', 'baby i', 'baby not', 'bab
```

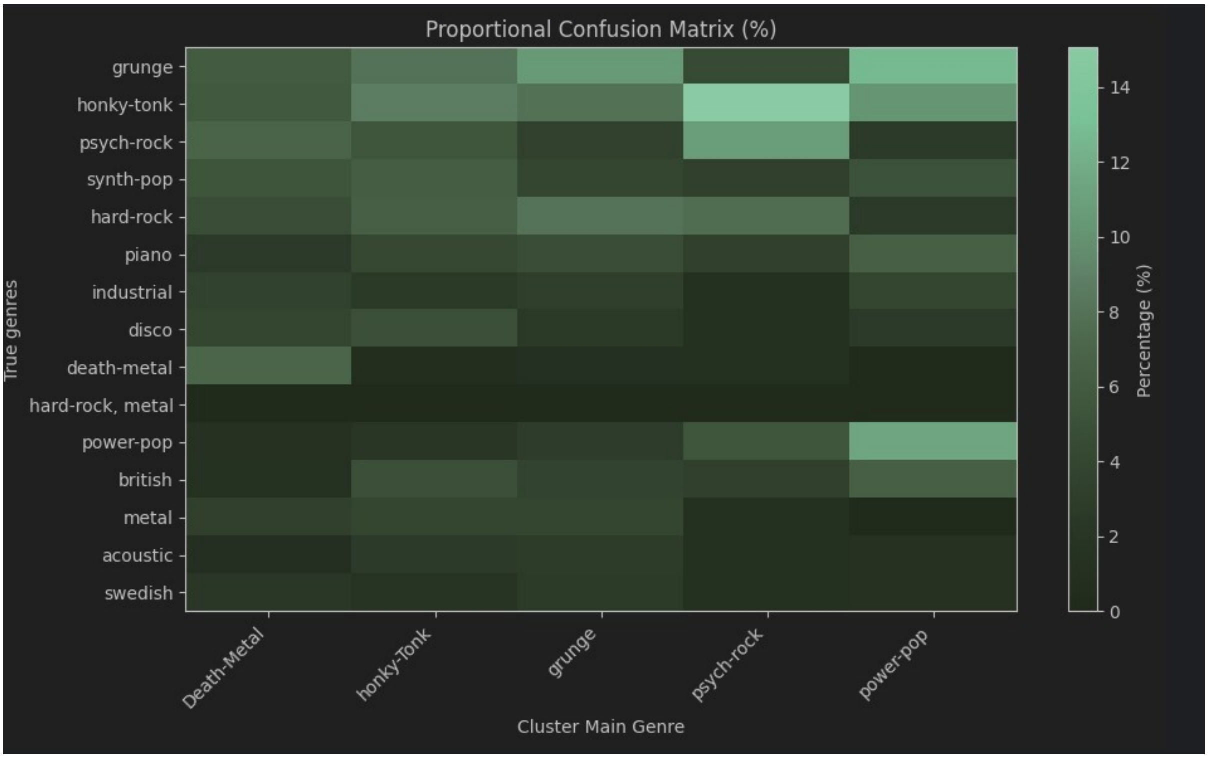



```
Elbow at k = 5
```

A scree plot was generated as an initial indication of the optimal number of clusters needed for soft clustering analysis.

Soft boundaries partitioning visualisation and evaluates

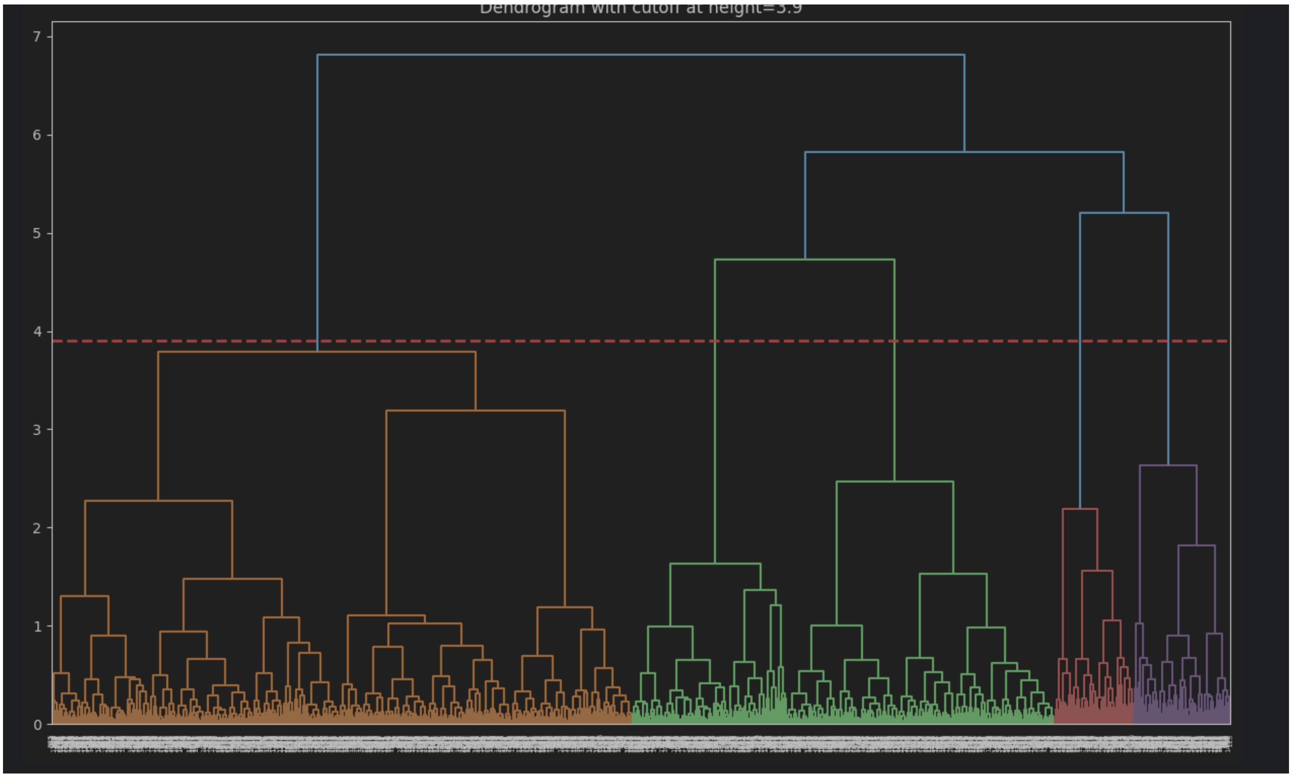

```
=== Cluster 0 (473 songs) | Top genres: death-metal (33/473), psych-rock (32/473), grunge (28/473), honky-tonk (27/473), synth-pop (25/473) ===

=== Cluster 1 (390 songs) | Top genres: honky-tonk (34/390), grunge (31/390), hard-rock (25/390), synth-pop (24/390), psych-rock (21/390) ===

=== Cluster 2 (523 songs) | Top genres: grunge (55/523), hard-rock (42/523), honky-tonk (41/523), piano (24/523), dance (22/523) ===

=== Cluster 3 (93 songs) | Top genres: honky-tonk (14/93), psych-rock (10/93), hard-rock (7/93), power-pop (5/93), alt-rock (5/93) ===

=== Cluster 4 (79 songs) | Top genres: grunge (10/79), power-pop (9/79), honky-tonk (8/79), blues (5/79), british (5/79) ===
```



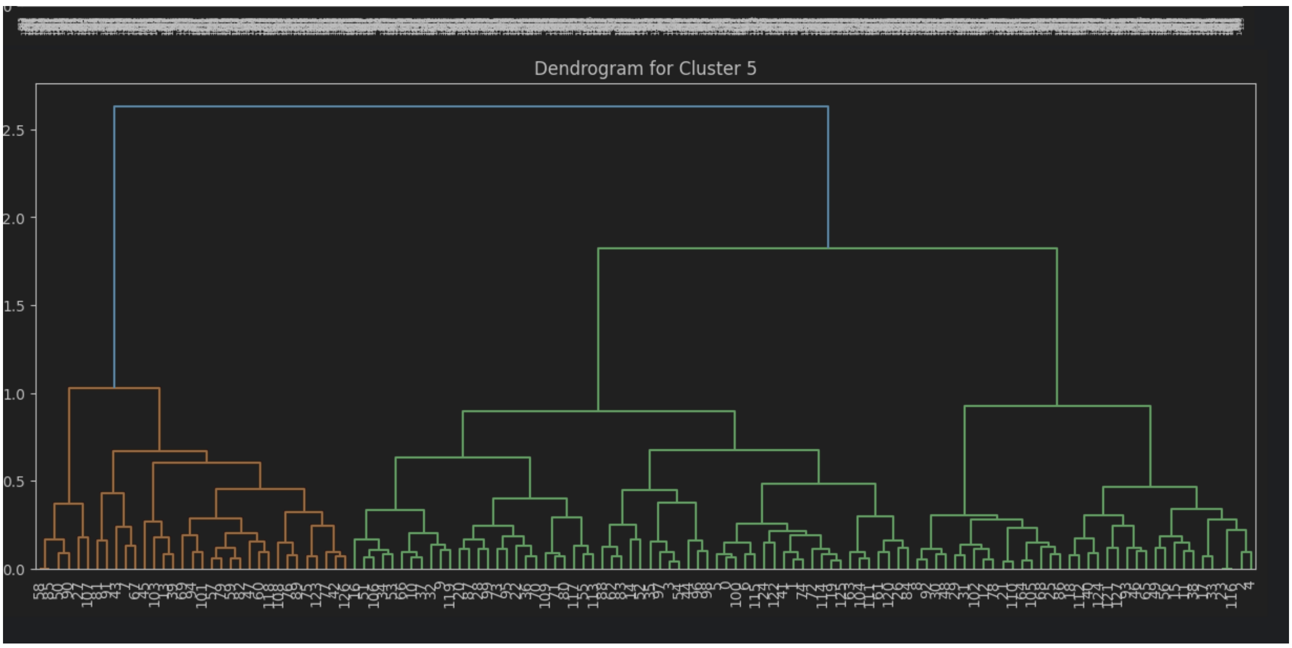
	Death-Metal	honky-Tonk	grunge	psych-rock	power-pop
grunge	5.92	7.95	10.52	4.30	12.66
honky-tonk	5.71	8.72	7.84	15.05	10.13
psych-rock	6.77	5.38	3.25	10.75	2.53
synth-pop	5.29	6.15	3.82	3.23	5.06
hard-rock	4.65	6.41	8.03	7.53	2.53
piano	2.54	4.10	4.59	3.23	6.33
industrial	3.38	2.31	3.06	1.08	3.80
disco	3.81	4.87	2.29	1.08	2.53
death-metal	6.98	0.51	0.96	1.08	0.00
hard-rock, metal	0.00	0.00	0.00	0.00	0.00
power-pop	1.27	2.05	2.87	5.38	11.39
british	1.06	4.87	3.63	3.23	6.33
metal	3.17	3.85	3.82	1.08	0.00
acoustic	0.85	2.56	2.68	1.08	1.27
swedish	2.11	1.54	2.49	1.08	1.27

```
   synth-pop       0.00      0.00      0.00        76

    accuracy                           0.21       693
   macro avg       0.07      0.19      0.10       693
weighted avg       0.09      0.21      0.13       693

Macro Precision: 0.06903055617395852
Macro Recall: 0.18886525810822624
Macro F1: 0.09799331535970432
Weighted F1: 0.12511895417668714
```

Hard boundaries in hierarchical methods
The height of the dendrogram determines the number of clusters to be included.



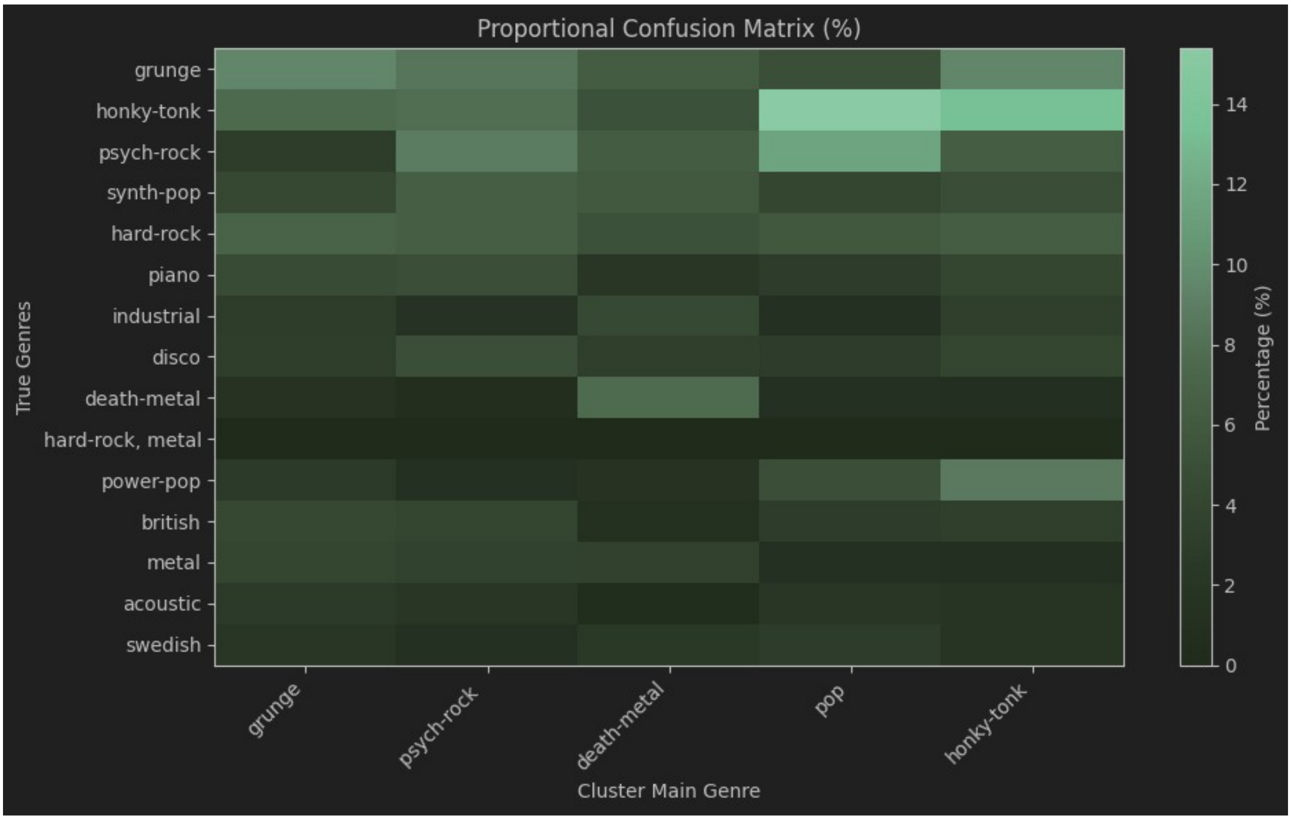
Zoom of cluster: The cluster to zoom into can be selected directly in the code. This



example shows a zoomed-in view of **Cluster 5**.

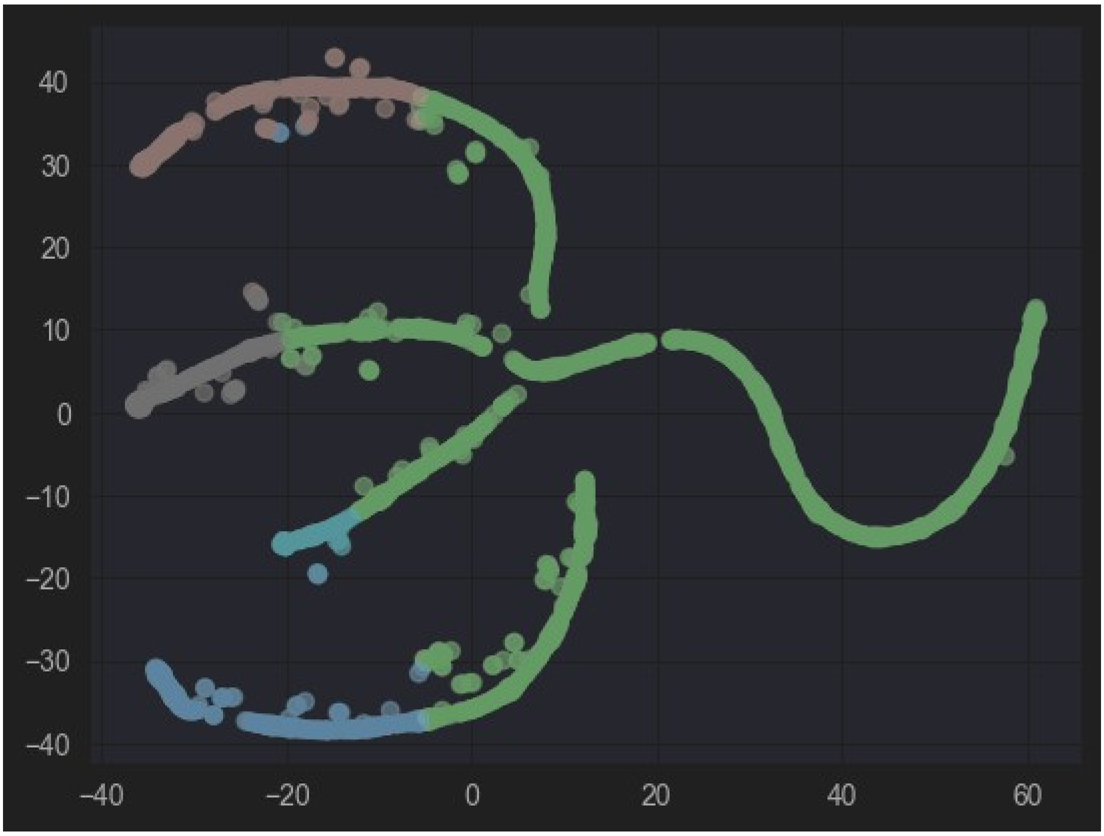
```
=== Cluster 1 (768 songs) | Top genres: grunge (72/768), honky-tonk (57/768), hard-rock (53/768), piano (35/768), british (32/768) ===

=== Cluster 2 (204 songs) | Top genres: psych-rock (18/204), grunge (17/204), honky-tonk (16/204), synth-pop (13/204), hard-rock (13/204) ===

=== Cluster 3 (354 songs) | Top genres: death-metal (27/354), psych-rock (22/354), grunge (22/354), synth-pop (21/354), hard-rock (18/354) ===

=== Cluster 4 (104 songs) | Top genres: honky-tonk (16/104), psych-rock (12/104), hard-rock (6/104), grunge (5/104), power-pop (5/104) ===

=== Cluster 5 (128 songs) | Top genres: honky-tonk (17/128), grunge (12/128), power-pop (11/128), hard-rock (8/128), psych-rock (8/128) ===
```

	Death-Metal	honky-Tonk	grunge	psych-rock	power-pop
grunge	5.92	7.95	10.52	4.30	12.66
honky-tonk	5.71	8.72	7.84	15.05	10.13
psych-rock	6.77	5.38	3.25	10.75	2.53
synth-pop	5.29	6.15	3.82	3.23	5.06
hard-rock	4.65	6.41	8.03	7.53	2.53
piano	2.54	4.10	4.59	3.23	6.33
industrial	3.38	2.31	3.06	1.08	3.80
disco	3.81	4.87	2.29	1.08	2.53
death-metal	6.98	0.51	0.96	1.08	0.00
hard-rock, metal	0.00	0.00	0.00	0.00	0.00
power-pop	1.27	2.05	2.87	5.38	11.39
british	1.06	4.87	3.63	3.23	6.33
metal	3.17	3.85	3.82	1.08	0.00
acoustic	0.85	2.56	2.68	1.08	1.27
swedish	2.11	1.54	2.49	1.08	1.27




```
    psych-rock       0.00      0.00      0.00        82
          rock       0.00      0.00      0.00        43
     synth-pop       0.00      0.00      0.00        76

      accuracy                           0.15       981
     macro avg       0.03      0.11      0.05       981
  weighted avg       0.05      0.15      0.07       981

Macro Precision: 0.029248530646515534
Macro Recall: 0.11331915486493574
Macro F1: 0.045348580430547646
Weighted F1: 0.06868316009540254
```

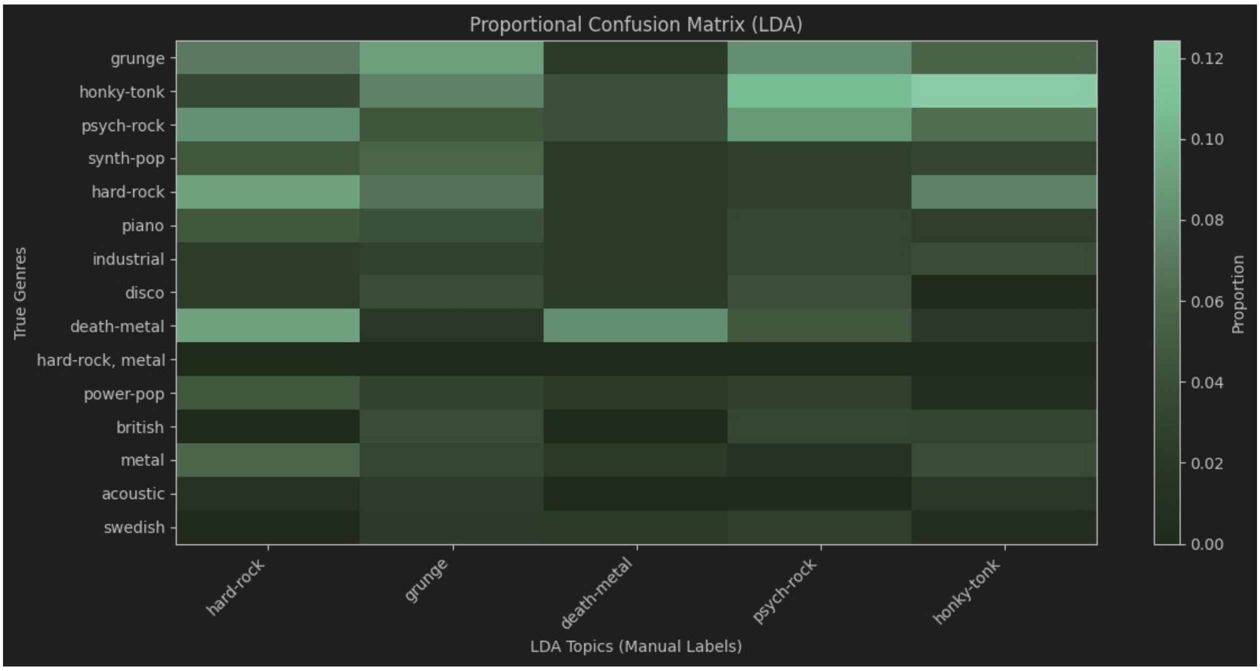
third method experimented is the LDA




```
=== Topic 0 (86 songs) | Top subgenres: hard-rock (8), death-metal (8), psych-rock (7), grunge (6), metal (5) ===

=== Topic 1 (1111 songs) | Top subgenres: grunge (100), honky-tonk (83), hard-rock (73), synth-pop (62), psych-rock (50) ===

=== Topic 2 (50 songs) | Top subgenres: jazz (5), new-age (5), death-metal (4), goth (3), world-music (3) ===

=== Topic 3 (150 songs) | Top subgenres: honky-tonk (16), psych-rock (13), grunge (12), jazz (8), alt-rock (8) ===

=== Topic 4 (161 songs) | Top subgenres: honky-tonk (20), hard-rock (12), psych-rock (10), grunge (9), alt-rock (8) ===
```

Proportional Confusion Matrix (Top genres, in %):

	hard-rock	grunge	death-metal	psych-rock	honky-tonk
grunge	6.98%	9.0%	2.0%	8.0%	5.59%
honky-tonk	3.49%	7.47%	4.0%	10.67%	12.42%
psych-rock	8.14%	4.5%	4.0%	8.67%	6.21%
synth-pop	4.65%	5.58%	2.0%	2.67%	3.11%
hard-rock	9.3%	6.57%	2.0%	2.67%	7.45%
piano	4.65%	4.14%	2.0%	3.33%	2.48%
industrial	2.33%	2.79%	2.0%	3.33%	3.73%
disco	2.33%	3.87%	2.0%	4.0%	0.0%
death-metal	9.3%	1.71%	8.0%	4.67%	1.86%
hard-rock, metal	0.0%	0.0%	0.0%	0.0%	0.0%
power-pop	4.65%	2.97%	2.0%	2.67%	0.62%
british	0.0%	3.69%	0.0%	3.33%	3.11%
metal	5.81%	3.33%	2.0%	1.33%	3.73%
acoustic	1.16%	2.34%	0.0%	0.0%	1.86%
swedish	0.0%	2.25%	2.0%	2.67%	0.62%



honky-tonk	0.26	0.29	0.28	124
piano	0.00	0.00	0.00	60
power-pop	0.00	0.00	0.00	43
psych-rock	0.00	0.00	0.00	82
synth-pop	0.00	0.00	0.00	76
accuracy			0.21	693
macro avg	0.11	0.14	0.10	693
weighted avg	0.13	0.21	0.13	693

Macro Precision: 0.10506514636211893

Macro Recall: 0.1389740232573491

Macro F1: 0.09501348091386405

Weighted F1: 0.1329435202232137

Part3.3.1_SoftGridSearch.ipynb-Performs grid search optimization for the soft clustering method.

```
Testing: max_features=8000, ngram_range=(1, 1), min_df=90, n_components=35, svd_random_state=40, n_clusters=5, kmeans_n_init=100, kmeans_random_state=100
→ Silhouette score: 0.0970

Testing: max_features=8000, ngram_range=(1, 1), min_df=90, n_components=35, svd_random_state=40, n_clusters=6, kmeans_n_init=50, kmeans_random_state=80
→ Silhouette score: 0.0717

Testing: max_features=8000, ngram_range=(1, 1), min_df=90, n_components=35, svd_random_state=40, n_clusters=6, kmeans_n_init=50, kmeans_random_state=90
→ Silhouette score: 0.0716

Testing: max_features=8000, ngram_range=(1, 1), min_df=90, n_components=35, svd_random_state=40, n_clusters=6, kmeans_n_init=50, kmeans_random_state=100
→ Silhouette score: 0.0717
```

Part3.3.2_HardGridSearch.ipynb Performs grid search optimization for the hard clustering method.

```
)

display(grid_results)

13s 588ms

Testing: max_features=8000, ngram_range=(1, 1), min_df=15, max_df=0.8, n_components=30, svd_random_state=42, n_clusters=5, linkage_method=average
→ Silhouette score: 0.2219

Testing: max_features=8000, ngram_range=(1, 1), min_df=15, max_df=0.8, n_components=30, svd_random_state=42, n_clusters=6, linkage_method=ward
→ Silhouette score: 0.0547

Testing: max_features=8000, ngram_range=(1, 1), min_df=15, max_df=0.8, n_components=30, svd_random_state=42, n_clusters=6, linkage_method=average
→ Silhouette score: 0.1977

Testing: max_features=8000, ngram_range=(1, 1), min_df=15, max_df=0.8, n_components=30, svd_random_state=42, n_clusters=7, linkage_method=ward
→ Silhouette score: 0.0601

Testing: max_features=8000, ngram_range=(1, 1), min_df=15, max_df=0.8, n_components=30, svd_random_state=42, n_clusters=7, linkage_method=average
```

Part3.3.3_LDAGridSearch.ipynb – Performs grid search optimization for the LDA method.

```
Testing: n_topics=5, max_features=10000, k_clusters=5, lda_random_state=42, kmeans_random_state=20
→ Silhouette score: 0.629

Testing: n_topics=5, max_features=10000, k_clusters=5, lda_random_state=42, kmeans_random_state=30
→ Silhouette score: 0.629

Testing: n_topics=5, max_features=10000, k_clusters=5, lda_random_state=42, kmeans_random_state=90
→ Silhouette score: 0.581

Testing: n_topics=5, max_features=10000, k_clusters=5, lda_random_state=100, kmeans_random_state=20
```

Part4.1_SimilaritySearch.ipynb – implements similarity search methods to compare and retrieve relevant songs.

TF-IDF + Cosine Similarity:
The function requires two inputs: the song ID and the number of similar songs to find.
It retrieves the most similar songs based on lyrics similarity, using the cosine similarity method.

```
# --- 5. Function Call ---
# Input song_id  to generate top 5 similar songs
top5_similar = top_similar('8036593539990052832', top_n=5 )
```

```
=== Top 5 songs similar to: Love of Money (ID=8036593539990052832) ===
113. [11190317389501666971] Mi Remember dancehall (similarity=0.400)
810. [2091181825240645407] My Crew j-dance (similarity=0.312)
52. [10528770166993144871] Voglio restare cosi opera (similarity=0.174)
782. [18377019154707697331] Wine Pon Me j-dance (similarity=0.140)
1253. [6923628035955983211] Hey Mama edm (similarity=0.116)
```

Word2Vec + Cosine Similarity:
This approach works the same way as TF-IDF + Cosine Similarity, but instead uses Word2Vec embeddings combined with cosine similarity to measure the closeness between song lyrics.

```
# Function call:
top_similar_word2vec('8036593539990052832', top_n=5)
✓ [2] 4s 773ms
```

```
=== Top 5 songs similar to: Love of Money (ID=8036593539990052832) ===
1257. [6959893070932530374] Clarks j-dance (similarity=0.998)
1195. [6291538727295026626] Why We Thugs funk (similarity=0.996)
95. [10929755647275482411] Maybellene rockabilly (similarity=0.995)
1428. [8830279411611115885] X hardcore (similarity=0.995)
1325. [7693101070296471068] Best of Things hardcore (similarity=0.995)
```

```
SimilarityData
    cosine_title_similarity.csv
    jaccard_title_similarity.csv
    tfidf_cosine_similarity.csv
    word2vec_cosine_similarity.csv
Songs
```

The returned songs are also saved into a CSV file for further analysis.

Part4.2_TitlevsLyrics.ipynb – analyzes the relationship between song titles and their corresponding lyrics.

Cosine Similarity over TF-IDF Vectors
The function requires two inputs: the song ID and the number of similar songs to find.
It returns the top 5 songs that show similarity between the title and the lyrics.

```
# --- call function ---
title_lyrics_cosine('13741824864810098075', top_n=5)

✓ [1] 3s 254ms

 === Top 5 lyrics most similar to title: Death Whispered a Lullaby (ID=13741824864810098075) ===
 1. [5506909473774015947] Lullaby (Genre: grunge, metal, similarity=0.271)
    Lyrics: I know the feeling Of finding yourself stuck out on the ledge And there ain't no heal
    you that it's never that bad And take it from someone who's been where your at You're laid ou
    So just give it one more try With a lullaby And turn this up on the radio If you can hear me
    And you can't tell, I'm scared as hell 'Cause I can't get you on the telephone So just close
    lullaby Please let me take you Out of the darkness and into the light 'Cause I have faith in
```

Jaccard Similarity based on Token Sets
This approach follows the same logic but measures similarity using the Jaccard index over token sets.

```
# --- call function ---
title_lyrics_jaccard('13860806082654952241', top_n=5)

✓ [2] 186ms

 === Top 5 lyrics most similar to title: Hallelujah I Love Her So (ID=13860806082654952241) ===
 1. [10018510778896022773] How Can You Mend A Broken Heart (Genre: disco, Jaccard=0.000)
    Tokens: "round", "but", "makes", "help", "gone", "tomorrow", "no", "ever", "one", "i", "stop", "
    "word", "still", "sorrow", "days", "world", "sun", "think", "want", "shining", "living", "please
    "everything", "never", "heart", "mend", "trees", "said", "told", "rain", "younger", "could", "mi

 2. [4597666000887757323] Everlong (Genre: alt-rock, alternative, grunge, metal, Jaccard=0.000)
    Tokens: "everlong", "got", "wonder", "real", "good", "ever", "say", "i", "stop", "hold", "not",
```

The returned songs are also saved into a CSV file for further analysis.