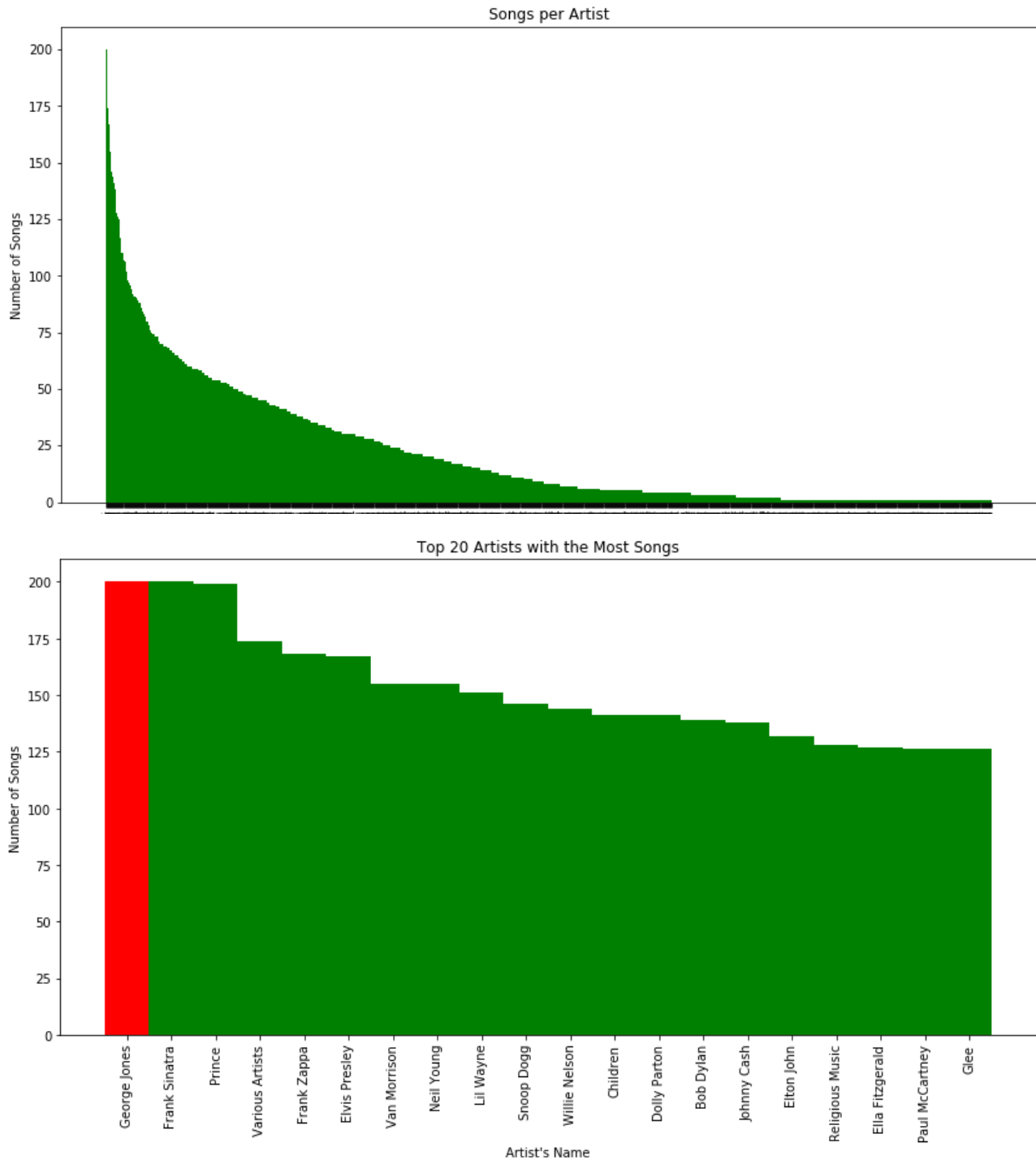


GROUP 7

Song statistics

To understand the songs we performed some basic statistics in the saregamapa_visualize.py file.

1. Creating a histogram of the number of songs per artist we can identify artists with most songs. See

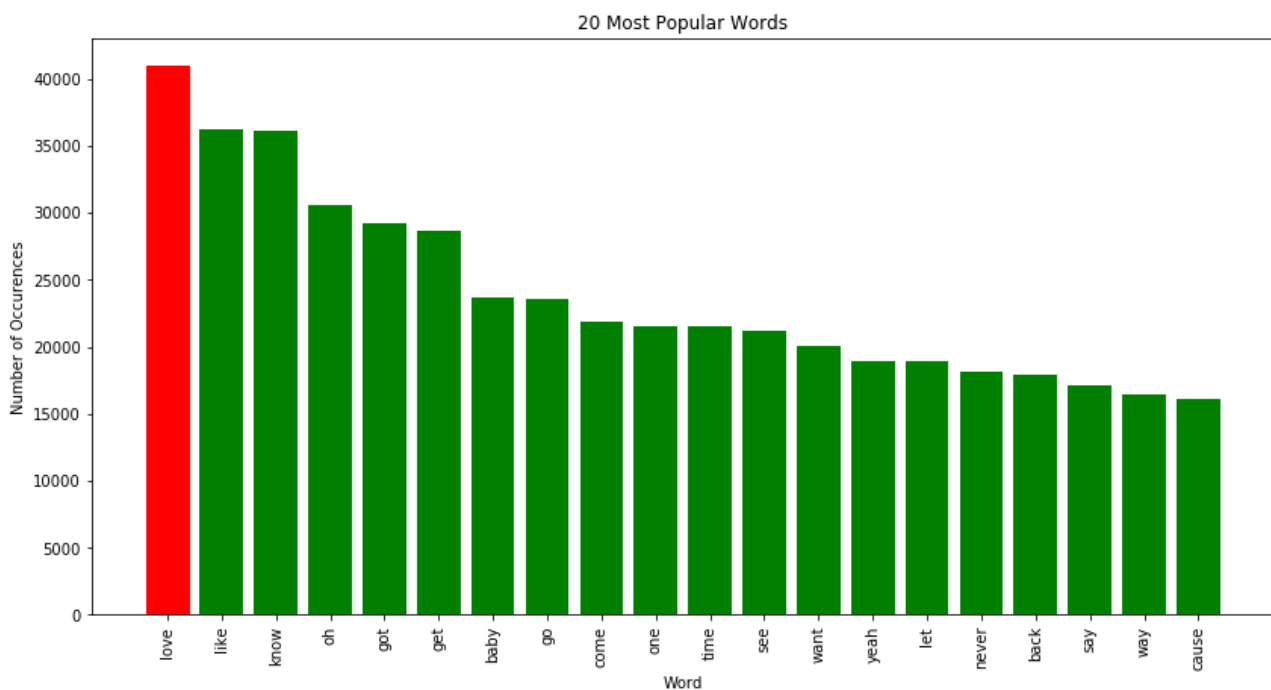


Conclusions:

ARTIST	CAREER PERIOD	YEARS OF CAREER
George Jones	1954 – 2013	59
Frank Sinatra	1932 – 1995	63
Prince	1975 – 2016	41
Frank Zappa	1955 – 1993	38
Elvis Presley	1953 – 1977	24
Van Morrison	1964 – till date	54
Neil Young	1960 – till date	58
Lil Wayne	1991 – till date	27
Snoop Dogg	1992 – till date	26
Willie Nelson	1956 – till date	62
Dolly Parton	1959 – till date	59
Bob Dylan	1959 – till date	59
Johnny Cash	1955 – 2003	48
Elton John	1964 – till date	54
Ella Fitzgerald	1934 – 1993	59
Paul McCartney	1957 – till date	61

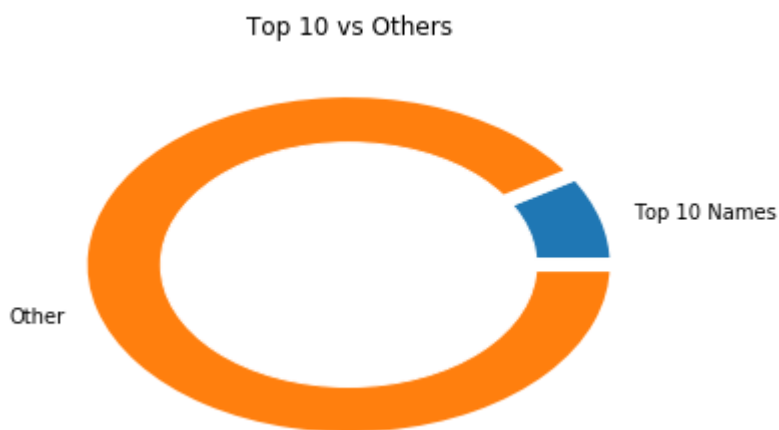
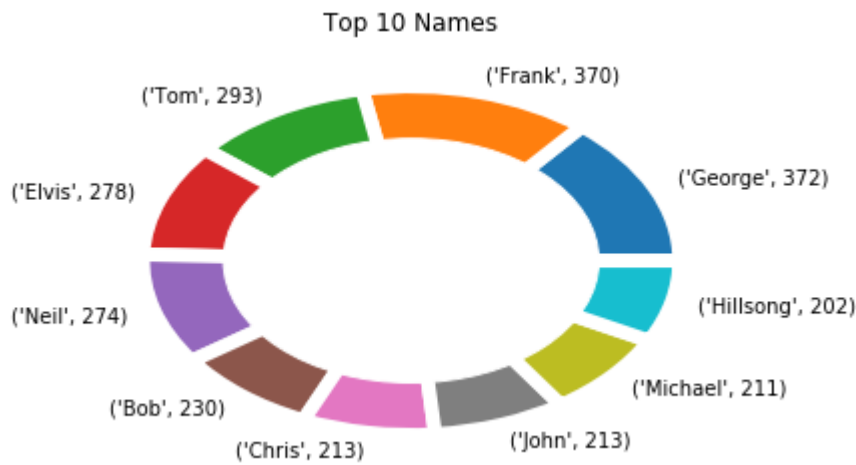
We can see that the most of the artists that have composed the most songs have a lot of years of career, except for 2 which they are rappers.

2. With a histogram we can identify the 20 most popular words (excluding stopwords). See `search_for_popular_words` function.



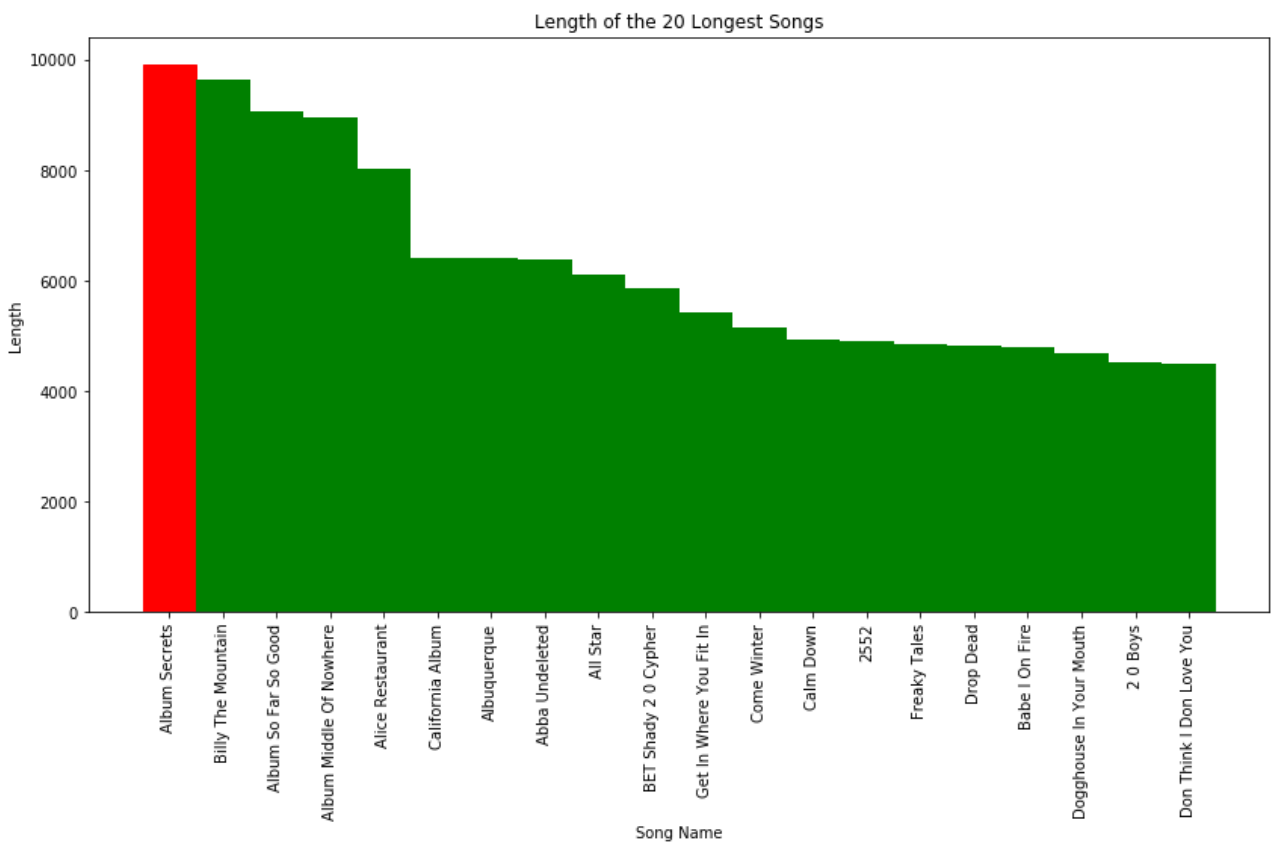
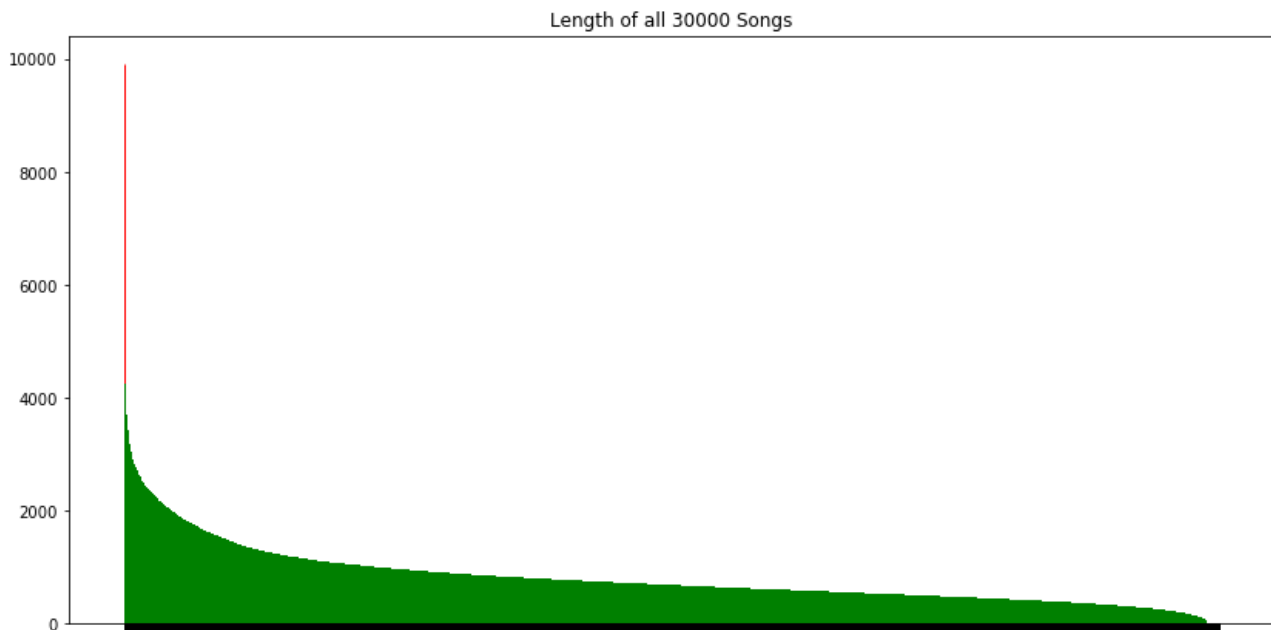
Conclusions: We can observe that most of the popular words are verbs with no particular importance, if we remove these we can see that the most common words are words referred to human feelings. So we can conclude that authors saw that songs that talk about feelings have more success.

3. With pie charts we identify the 10 most common singer names with it's relative number of songs composed from each name. Then we see whether singers whose name is the same tend to publish more songs than others. See `search_for_popular_artist_names` function.

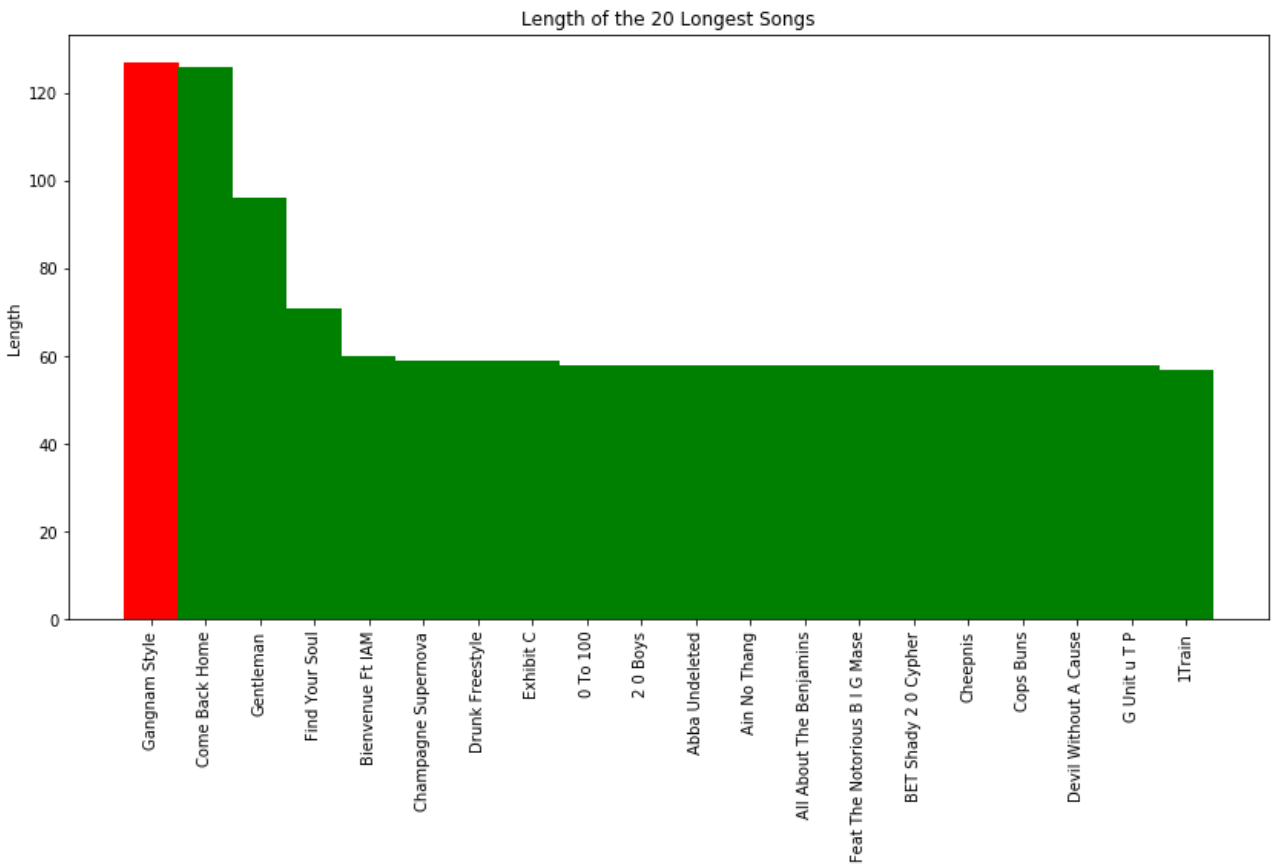
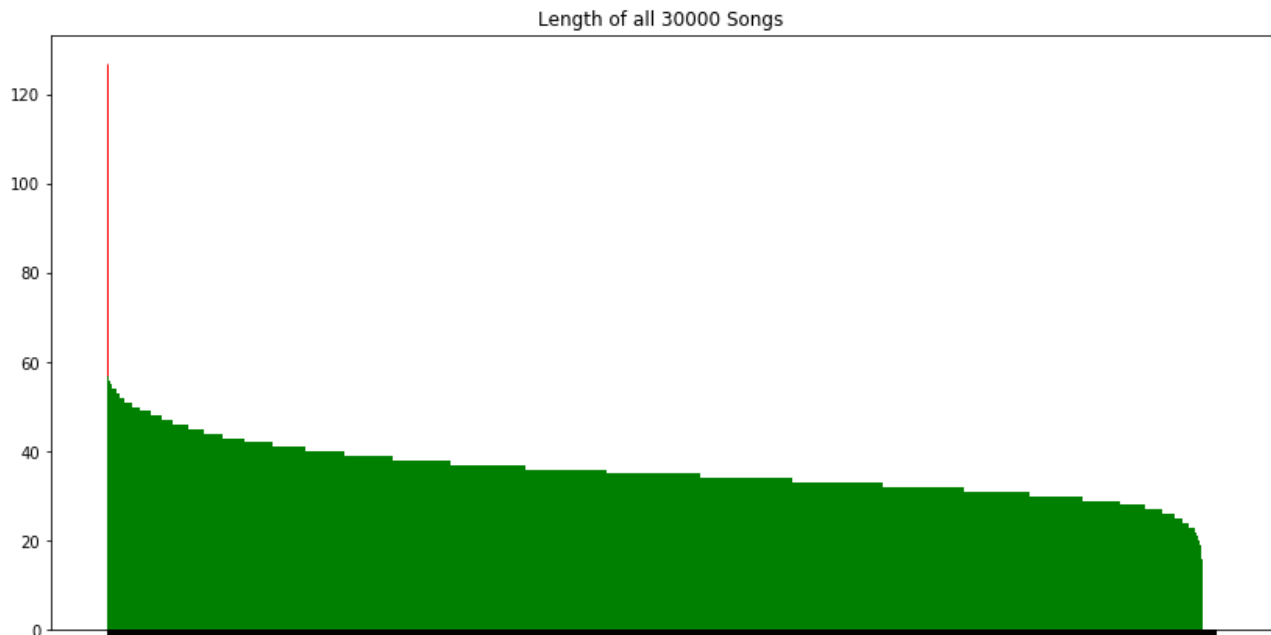


Conclusion: We can see that 9% of all the 30000 songs are written by the more inspired names, so having one of these names doesn't guarantee you a large repertory.

4. Histogram (using plot_songs_lengths_histogram function) of song lengths (total words per song with repetitions). See song_word_count_with_repetition function.



Histogram (using plot_songs_lengths_histogram function) of song lengths (total words per song with out repetitions, so it's vocabulary). See song_word_count_with_out_repitition function.



Search Engine

In the `saregamapa_search.py` file we created a Search Engine that answers Boolean queries and using as scoring function the (tf-idf cosine similarity). With the functions `format_text` and `remove_stopwords` in the `saregamapa_common.py` file, we pre-processed the documents by removing stopwords, punctuation, normalizing the vocabulary, and stemming.

The Search program receives as an argument a set of words (e.g. love sun). For the query of Type 1 we return the top-10 most relevant documents according to tf-idf-cosine scoring using the `apply_search` function. With a heap data structure in the `apply_heap_toresults` function for maintaining the top-10 documents.

```
(0.46909529675953027, 19691, 'Do You Love Me', 'https://www.azlyrics.com/k/kiss/do+you+love+me_20079767.html')
(0.41219300222171606, 4918, 'Another Sun', 'https://www.azlyrics.com/t/tracy+chapman/another+sun_20285097.html')
(0.405978074117865, 3142, 'All Love', 'https://www.azlyrics.com/i/ingrid+michaelson/all+love_20793749.html')
(0.39324786260327704, 11947, 'But You Know I Love You', 'https://www.azlyrics.com/d/dolly+parton/but+you+know+i+love+you_20041617.html')
(0.38663116187807445, 15352, 'Come Back The Sun', 'https://www.azlyrics.com/z/zucchero/come+back+the+sun_20647488.html')
(0.37566843895749136, 3612, 'All You Need Is Love', 'https://www.azlyrics.com/g/glee/all+you+need+is+love_21069131.html')
(0.37566843895749136, 3607, 'All You Need Is Love', 'https://www.azlyrics.com/b/beatles/all+you+need+is+love_10026698.html')
(0.3736317762529984, 11926, 'But Love', 'https://www.azlyrics.com/n/noa/but+love_20664949.html')
(0.36099384658730854, 21619, 'Drive', 'https://www.azlyrics.com/c/carly+rae+jepsen/drive_21040870.html')
(0.3565974223351557, 18923, 'Did I Ever Tell You', 'https://www.azlyrics.com/g/george+jones/did+i+ever+tell+you_20822808.html')
```

For the query of Type 2 (in the `saregamapa_cluster.py` file) we return all the songs that contain all the query terms with the function `search_complete`.

```
{2050, 8196, 22533, 8200, 10248, 16393, 16394, 20492, 2072, 20504, 18458, 27, 2
77, 78, 16461, 28753, 2132, 6228, 14420, 26709, 20569, 20570, 20571, 2142, 2262
24711, 138, 20618, 28811, 28813, 22671, 24721, 28821, 2198, 8344, 2202, 14494,
4279, 20661, 22707, 22716, 10429, 18623, 12482, 18627, 22722, 20677, 14534, 227
14573, 6385, 242, 18673, 22772, 12533, 12534, 16631, 20726, 20733, 254, 14593, ecc...
```

Then we ask the user for a value of `k` (the number of clusters)(for example `k = 3`). And we cluster with the K-means method (with the `cluster_documents` function), but each document can be represented as a normalized tf-idf vector (such that its length is 1) in the space of the terms of the vocabulary in the `normalize_results` function. Then, you can use as metric to define the distance among documents using the Euclidean distance, perfect for the k-means algorithm.

```
song 26284 is in cluster 0
song 11951 is in cluster 2
song 26287 is in cluster 2
song 28337 is in cluster 2
song 24242 is in cluster 2
song 9907 is in cluster 1
song 9909 is in cluster 1
song 9912 is in cluster 2
song 22203 is in cluster 2
song 16060 is in cluster 0
song 7870 is in cluster 2
song 7872 is in cluster 0
song 9920 is in cluster 2
song 18115 is in cluster 2
song 22212 is in cluster 2
ecc...
```

We could try it more times to see the best solution, since the clustering algorithm isn't optimal.

For each cluster we displayed a word cloud of the most common terms in the cluster, with the `cluster_documents` function.

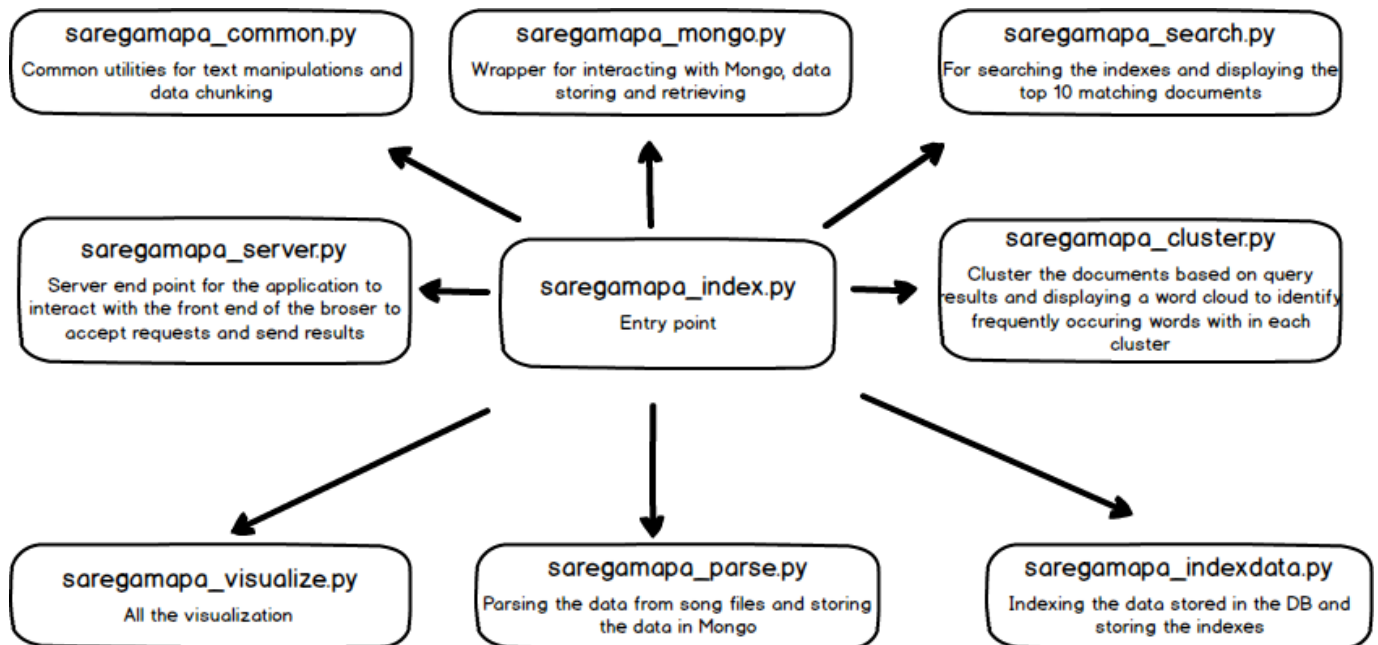
Cluster number: 0

Cluster number: 1

Cluster number: 2

Code Documentation:

As shown in the below picture following are the python modules we created to implement the search engine, clustering and word cloud functionality,



How inverted indexes are stored in the DB ?

For any search engine to be efficient indexes needs to be stored optimally. In our initial observation we observed that if we use a separate document of a collection for every word the number of documents are increased exponentially there by making the response time of a search query inefficient. So we designed a way in which the storing, retrieving and mapping of indexes is as efficient as possible. To achieve this we are grouping chunks of indexes as one document say 1500 indexes for one document and storing it in to DB. So once all the indexes are calculated for all the documents we are dividing the keys in to chunks of 1500 and storing them as separate documents, which means we need minimum documents to store all the indexes and thereby resulting in better response rates. After retrieval of all the indexes we are combining all the keys in all the documents in to a single dictionary.

Future enhancements: We want to make this process more efficient by sorting the index keys in alphabetical order or even split the indexes in to multiple collections so that given a word we can easily identify which collection to check from there by reducing number of comparisons in turn resulting in better search response rates.

Note:

Issue with Mongo DB document size limit:

While storing indexes in to the form of chunks of say 1000 keys we encountered a issue with Mongo DB document maximum size exceed. We identified that this is due to some of the keys occurring in lot of documents there by increasing the size required to store it, we eliminated this problem to some extent by sorting the document score (for that particular key) in descending order and putting a limit on number of document scores that can be stored for a particular key lets say 1000 document scores for a particular key.

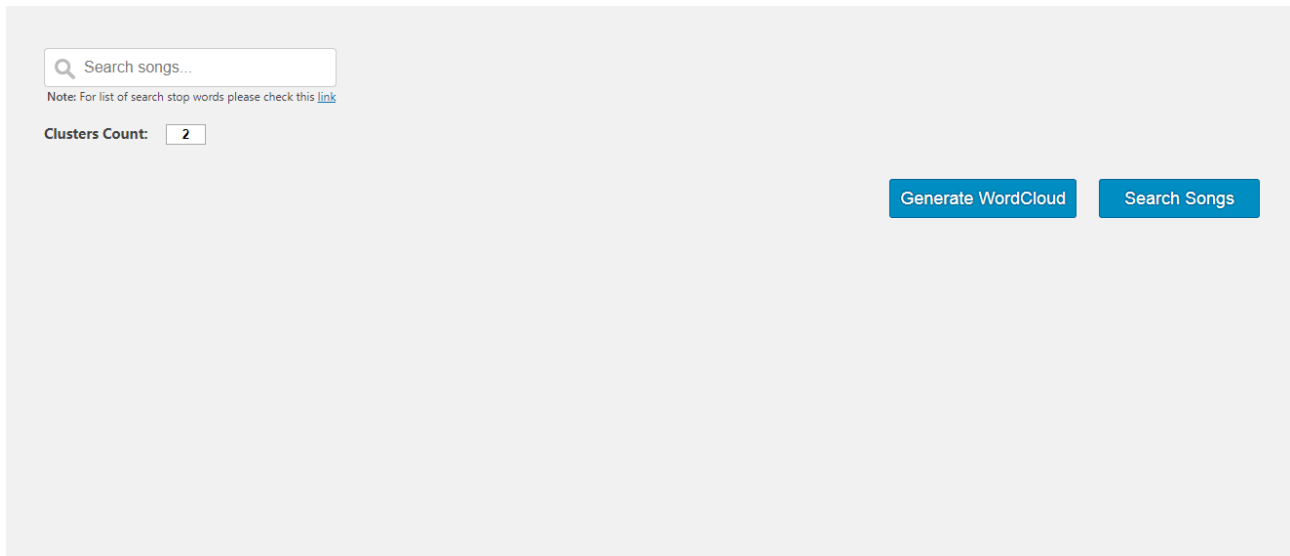
Web Layer:

Before starting, convert static>js>saregamapa_script.txt in .js, since it wasn't possible to send it by email.

We created a python module saregamapa_server.py which uses web.py library to receive AJAX calls from the browser. Code is present in saregamapa_server.py file. Up on executing this file (by running python saregamapa_server.py from the project root), we will see a message that server started on 0.0.0.0:8080. Now opening the following URL in chrome,

<http://localhost:8080>

in chrome will show the following page,

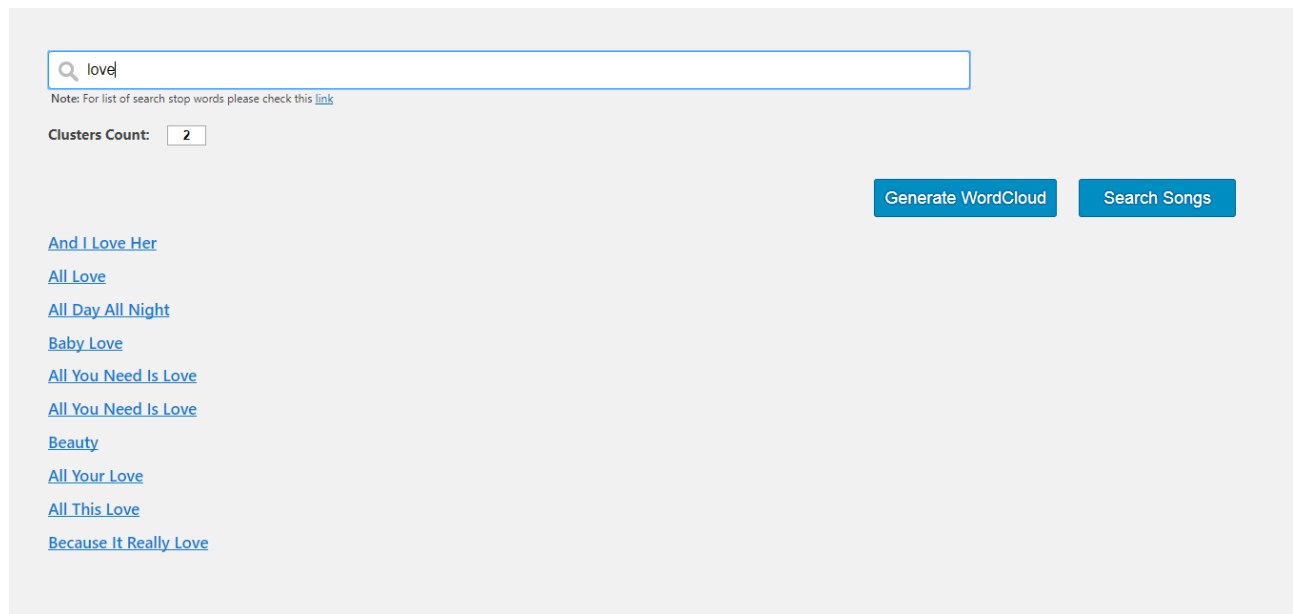


The screenshot displays a web application interface with a light gray background. At the top left, there is a search bar with a magnifying glass icon and the placeholder text "Search songs...". Below the search bar, a small note reads: "Note: For list of search stop words please check this [link](#)". To the left of the search bar, the text "Clusters Count:" is followed by a small input box containing the number "2". On the right side of the interface, there are two blue buttons with white text: "Generate WordCloud" and "Search Songs".

Here in this page user can search the songs by entering a keyword say "night" in the query or he can view the clusters of all the documents which contain word say "night" by entering query and specifying the how many clusters he wants to view.

Search Songs:

Up on entering query and clicking on enter or Search Songs button, an API will be invoked to saregamapa_server.py which in turn invokes saregamapa_search.py module (via saregamapa_index.py module which is like an entry point) and search will be returned which will be passed to browser and rendered as shown below,



The screenshot shows a web interface for searching songs. At the top, there is a search bar containing the text "love". Below the search bar, a note reads: "Note: For list of search stop words please check this [link](#)". To the left of the search results, there is a label "Clusters Count:" followed by a small input box containing the number "2". On the right side, there are two blue buttons: "Generate WordCloud" and "Search Songs". Below these elements, a list of ten song titles is displayed, each followed by a blue link. The song titles are: "And I Love Her", "All Love", "All Day All Night", "Baby Love", "All You Need Is Love", "All You Need Is Love", "Beauty", "All Your Love", "All This Love", and "Because It Really Love".

Top ten matching documents are rendered with song title and its link. Up on clicking on the link user will be redirected to that song link page in azlyrics.com site

When user enters a query and specify the cluster count and clicks on Generate WordCloud button, a separate api will be invoked to saregamapa_server.py package which in turn invokes the saregama_cluster.py package (via saregamapa_index.py package which is the entry point for search engine invocation). Once the clustering is done results are displayed as shown below

Album Secrets

Another Saturday Night

3AM

Bad For Good

Banga Banga

Abba Undeleted

How to configure the project:

Code for our project is available on Github on following link,

<http://github.com/vamsivarma/saregamapa>

we used git for easy collaboration and task division.

Download source code of above repository and install required softwares and python packages as mentioned below.

Note:

Software's that needs to be pre-installed: Python, MongoDB

Important python packages used: pymongo, bs4, heapq, web, matplotlib.pyplot, pylab, sklearn.cluster, wordcloud, itertools

Before executing the project you will have to start the mongo server in your local machine with mongod command. Then open command line in the project folder and execute,

python saregamapa_server.py

This will parse the songs in the folder with songs html files (we have created a folder songs_complete/lyrics_collection) and store those songs, artists and inverted indexes in the Mongo DB data base. To specify the number of songs to be parsed we used a meta data object inside saregamapa_index.py module which has a structure as follows,

```
parse_dict = {  
    "songs_collection": "songs_30000", # Mongo Collection For storing songs  
    "artist_collection": "artists_map_30000", # Mongo Collection For storing artists  
    "iindex_collection": "iindex_30000", # Mongo Collection For storing inverted indexes  
    "folder_name": "\\songs_complete\\lyrics_collection",  
    "max_records": 30000 #This number tells how many songs needs to be stored in the DB  
}
```

Once everything is done, server endpoint will start at port 8080. Now open Chrome and go to link localhost:8080 and do search and clustering as explained in the previous pages.

Valerio Guarrasi 1643257

Vamsi Krishna Varma Gunturi 1794653

Riccardo Gobbo 1820941

Collaboration with Group 17