

Steven universe lyrics

Big Data Project



Chauvet tIMOTHÉ – mathieu camille

Table of contents

[Problem definition: 2](#_Toc68961402)

[Methodology: 3](#_Toc68961403)

[Making the JSON format dictionary 4](#_Toc68961404)

[Using a database for big amount of data 5](#_Toc68961405)

[Searching the lyrics and who’s singing them 5](#_Toc68961406)

[About the data size 6](#_Toc68961407)

[Results and discussion: 6](#_Toc68961408)

[Guideline: 6](#_Toc68961409)

# Problem definition:

Steven universe is a show produced by Cartoon Network and written by Rebecca Sugar featuring a plethora of musical segment, especially songs accompanying character arc development, representing the growth of each of their respective journey. As each character battles with different issues, there are recurring thematic coming up through their songs, as such **we would like to be able to identify the character singing by the lyrics of the song**.

The main challenge is that even through the approach is different, any songs from the show is often revolved around very similar themes, such as “*love*”, “strength”, “*approval*” and much more. We might be able to simply count the occurrences of a certain type of vocabulary and attribute it to a specific character, and therefore understand a character’s psychology through the words they use. Moreover, we will be able to process these words and have clear interactions between those words.

The challenges of this specific thematic we want to solve are to first find correct lyrics and be able to automatically associate them to a character. As many songs from the Steven Universe original soundtracks do not feature the character singing them, and that some of these songs are not even sung by characters, **finding a way to differentiate characters and then make an analysis of their vocabulary is somewhat a difficult task to do**.

A second challenge we will have to face is **being able to store huge amount of data**, as Steven Universe features a lot of songs with many lyrics. What could be the best way to store and access data in this case? Perhaps saving the lyrics locally is not the most optimal way to do things in this case.

A third challenge is to be able to scrap lyrics in an automatic way, with a qualitative source of information. As many websites for lyrics exist online, **a website featuring good lyrics and an easy-to-use API is not easy to find.** Therefore, connecting our application with such a website is somewhat a challenge to overcome.

With these goals in mind, let us implement our application and help us solve this problem for Steven Universe fans.

# Methodology:

The first challenge we focused on was **to find a good website with an easy-to-use API for scraping lyrics**. Many websites need us to write by hand all lyrics, however one service stood out: [genius.com](https://genius.com/developers). The advantages of Genius are very important for our use case:

* Genius features community-driven lyrics for songs and trusted members are able to add lyrics (like Wikipedia)
* features the characters who are singing (like [this example](https://genius.com/Steven-universe-change-lyrics) from the Steven Universe movie)
* has [clear and repetitive guidelines for writing lyrics](https://genius.com/Genius-how-to-add-songs-to-genius-annotated), which will help us harmonize lyrics in our app
* also features lyrics that are not song (like [this example](https://genius.com/Steven-universe-appearance-related-teasing-and-bullying-annotated) from a Cartoon Network advertisement about bullying)
* has a great API for scraping json lyrics.

A great and up-to-date API for scraping Genius data is [*lyricsgenius*](https://github.com/johnwmillr/LyricsGenius) by John W. Miller. Many projects around lyrics scraping use this Python library that features great and easy interfaces. Thus, **we will use Python for our data scraping script with the lyricsgenius library**.

The library is featured in the [pip package manager](https://pypi.org/project/lyricsgenius/), thus a quick command for the installation does the trick. As required data, the API needs a token delivered by the Genius API. Therefore, we created a new application on Genius and generated a token in a few minutes. A few tweaks with the options, and the scraper is ready to be used.

genius = lyricsgenius.Genius(token)

genius.verbose = False # Turn off status messages

genius.remove\_section\_headers = True # Remove section headers (e.g. [Chorus]) from lyrics when searching

genius.skip\_non\_songs = True # Include hits thought to be non-songs (e.g. track lists)

## Making the JSON format dictionary

Then, we create a Python dictionary as the conversion from dictionary to json data is easier to make. We can append it with tuples containing the songs. Overall, the json skeleton looks like this:

{

    "songs": [

        {

            "title": "Title of the song",

            "lyrics": [

                {

                    "line": "all i want for christmas is you",

                    "singer": "MARIAH CAREY"

                },

                {

                    "line": "i want to break free",

                    "singer": "FREDDIE MERCURY"

                }

            ]

        }

    ]

}

Here, the dictionary features a tuple called “songs” which contains itself another dictionary. This second dictionary features the title of a song, with its lyrics. We decided to divide each line of a song for each character who is singing, as analyzing words from a specific singer will be easier to make.

We also decided to put singers in uppercase and lines in lowercase as this would be easier to analyze.

There is a json.dump(SUsongs, json\_file) interface that we will use for our tests. The lyricsgenius API is able to only take a specific number of songs, so we will not exceed 15 songs locally. However, the use of the json format is much more interesting when we use it in real-case scenario, using a database.

## Using a database for big amount of data

As this is a big data project, we would like to use a database for querying with ease these lyrics and be able to process them in bulk.

As we are using a json data format, one option that pops to us is **using MongoDB**. Indeed, MongoDB features

* a library for lots of languages including PyMongo for Python
* uses the JSON format for appending data to a specific dictionary
* serialize and deserialize easily data
* easily deals with dictionaries, a primary data type of Python

The songs will be appended automatically using this library thus making our program able to transfer data after an outage.

We didn’t choose the virtual machine Cloudera as we had issues using this method. PyMongo, in the contrary, is pretty straightforward

client = MongoClient('localhost', 27017)

mydb = client["mydatabase"]

mydb["steven\_universe"].insert\_one(SUsongs)

We start by specifying the path to connect to our client, here we’re using the defaut localhost on port 27017, then we create a database inside of our client, “mydatabase” along with a collection, “steven\_universe” that we fill we the content or our json output.

## Searching the lyrics and who’s singing them

Python, like many other languages, has a RegEx library that allows us fetching certain text patterns. This functionality will be helpful in our project as the lyrics from Genius are messy. Therefore, we did these patterns:

* **Finding who’s singing**, which is quite simple as the result data is written like a theater script
* **Finding the lyrics**, much more difficult as we had to find patterns between two colons and remove the last word (corresponding to the next person singing)
* **Removing annotations** that are between parenthesis in the lyrics

After that cleaning process, we were able to quickly remove unwanted songs that doesn’t have multiple characters singing, as these songs can’t be automatized since the character singing the song isn’t written on them.

## About the data size

We made the whole process using only a limited number of songs to save on time and see if the process was viable. On the genius website there are more than 20 albums corresponding to the singer “Steven Universe”, each with up to 40 titles, therefore we have to work with an initial dataset of up to 1000 elements. To reduce this number, we proceeded to a pre-selection: we scrapped only the pieces having lyrics where two characters sing. Then we eliminated duplicates and hand-searched if there were title who did not appear in the show or who were not sung by characters of the show.

# Results and discussion

Our local database is indeed filled with the wanted lyrics and serialized as a json document. Using a Python script for extracting necessary data is a great idea as Python features lots of visualization tools for data. It also features many libraries for data management, such as PyMongo for dumping our json lyrics in a local MongoDB, and perhaps one located in the cloud as this library works with all use-cases.