**The Symphonic Metal Rock Data — Analyzing Evanescence & Within Temptation (part 1 — web scraping)**

Photo by [Vishnu R Nair](https://www.pexels.com/@vishnu-r-nair-426592?utm_content=attributionCopyText&utm_medium=referral&utm_source=pexels) from [Pexels](https://www.pexels.com/photo/people-at-concert-1105666/?utm_content=attributionCopyText&utm_medium=referral&utm_source=pexels" \t "_blank)

With this article I want to start a series of articles showing how to use some of the important skills of a Data Scientist. To start, I will be talking about web scraping and how I’ve built a Python web scraper using Beautiful Soup, a Python library that make it easy scraping information from a website.

My motivation for these articles came some months ago, when I decided to build a personal project on a subjected that I was passionate about and where I could use some of the important skills for a Data Scientist. Just when I was thinking about it I heard about a tour where the American rock band [Evanescence](https://en.wikipedia.org/wiki/Evanescence) would join forces with the Dutch symphonic metal band [Within Temptation](https://en.wikipedia.org/wiki/Within_Temptation), April this year. Since I love music and I found amazing the idea of putting together these two bands, seemed to me a natural choice to explore it in my project.

The two bands lead by the most influential and creative women in the rock world, Amy Lee from Evanescence and Sharon den Adel from Within Temptation, named the tour **WORLD COLLIDE**. [According to Amy Lee](https://www.blabbermouth.net/news/evanescence-and-within-temptation-to-join-forces-for-worlds-collide-2020-european-tour/), although they live in very distant countries and have some differences in their music, they also share a lot of similarities, so the name reflects well the embracing of both.

Based on this some questions started popping up: How data can show us how similar and different the bands are? What are the expectations of the fans while tour approaches and what are their reactions during the tour? And this is nice because it shows other skills that are important for a Data Scientist: be curious, ask questions, look for answers in your data, and if there is no data enough, go chase it!

In a broader way, my goal developing this project is to answer these and other questions. I want to analyze and compare these bands using data retrieved through web scraping, using [Spotify](https://developer.spotify.com/documentation/web-api/) and [Twitter API](https://developer.twitter.com/en/docs)s, and by applying important data scientist’s skills as cleaning, wrangling, visualization, as well as performing some NLP.

The first thing I need is data, so here I’ll start retrieving the data I need. We start by collecting all lyrics from both bands.

I’ll show now how to build a Python web scraper using Beautiful soup, applying some simple [Python string methods](https://www.w3schools.com/python/python_strings.asp) as well as other tools to obtain the lyrics. You will also notice that you don’t need to be an HTML expert to do this (I’m certainly not one), just a bit of observation.

So, let’s start web scraping….

**Choosing an Information source**

First, I’ve checked the web around for some lyrics’ sites with lyrics of both bands. My goal them is to use the same code to extract lyrics for both bands. I scraped different sites to see which one was able to provide the best information, also in the best format.

Some of the websites I’ve explored were:

- <http://www.metrolyrics.com>  
- <https://www.azlyrics.com/>  
- <https://www.songteksten.nl/>  
- <https://www.songteksten.net/>

**Web scraping**

**Retrieving hyperlinks of lyrics**

All websites I’ve visited have hyperlinks for each lyric. Some of them have all lyrics in one page, some in more than one page.

Since we want to retrieve lyrics, the first step is to access the lyrics via their hyperlinks. There are usually many hyperlinks in a webpage and not all them give us the information we are looking for. Therefore, some filtering will be used and it will depend on the structure of the website considered.

First, I have built a function that retrieve all hyperlinks. This functions is applicable to all websites visited. After that we will need to keep only the links that lead us to lyrics and that’s the moment that you need your observation skills. Each website has its own structure and you will take this into account.

In the code below I start importing library **requests** and **BeautifulSoup**. Then my function starts using requests to send a request to the **main\_url** which is the page were the hyperlinks to the lyrics are. See the image below the code for an example of how one of our main\_url looks like.

By applying .text to the result of my request we have the content of the page in a more understandable way, and this is kept in html\_doc. We then create our BeautifulSoup object using an **lxml** parser which allow us extracting the desired information from the html source code. There are different parsers you can use lxml is considered very fast.

Next we apply to our soup the method .find\_all with argument ‘a’ to find all hyperlinks in main\_url. Using a list comprehension we create a list of the hyperlinks retrieved. Notice that before returning my list I filter any **None** that may be there. This because I noticed that for some of the websites the None retrieved as hyperlinks caused some trouble in the next step that is filtering out the hyperlinks that we don’t need.

import requests  
from bs4 import BeautifulSoup

def retrieve\_hyperlinks(main\_url):  
 """ Extract all hyperlinks in 'main\_url' and return a list   
 with these hyperlinks """  
   
 # Packages the request, send the request and catch the   
 response: r

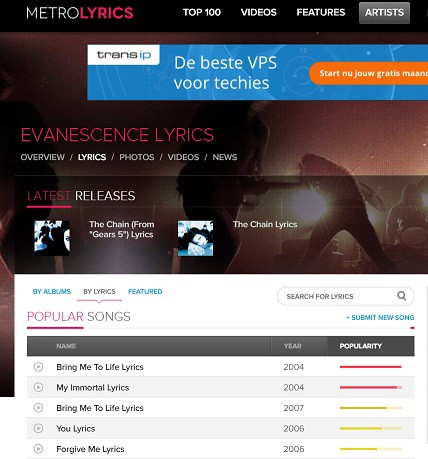
r = requests.get(main\_url)

# Extracts the response as html: html\_doc  
 html\_doc = r.text

# Create a BeautifulSoup object from the HTML: soup  
 soup = BeautifulSoup(html\_doc,"lxml")  
   
 # Find all 'a' tags (which define hyperlinks): a\_tags

a\_tags = soup.find\_all('a')  
   
 # Create a list with hyperlinks found

list\_links = [link.get('href') for link in a\_tags]  
   
 # Remove none values if there is some  
   
 list\_links = list(filter(None, list\_links))   
   
 return list\_links

Example of main\_url: [Metrolyric’s page with lyrics of Evanescence](https://www.metrolyrics.com/evanescence-lyrics.html).

In the case of metrolyrics there is only one page with all lyrics so we just need to apply our function once. Just to avoid repetition I apply set to my list.

url = '[http://www.metrolyrics.com/evanescence-lyrics.html'](http://www.metrolyrics.com/evanescence-lyrics.html%27)  
list\_links\_lyrics\_metrolyrics = retrieve\_hyperlinks(url)

# remove probable repetitions

list\_links\_lyrics\_metrolyrics = list(set(list\_links\_lyrics\_metrolyrics))

print('\n Number of links before filtering:', len(list\_links\_lyrics\_metrolyrics))  
list\_links\_lyrics\_metrolyrics[:20]

sample of the hyperlinks retrieved at metrolyrics for Evanescence.

A quick look in the list above reveals that links containing lyrics are of the form ***http://www.metrolyrics.com/TITLE-lyrics-evanescence.html***. So, we can select the elements of the list that contains ‘***-lyrics-evanescence.html***’.

The list comprehension bellow does the job and return us 80 hyperlinks instead of the initial 111 hyperlinks.

list\_links\_lyrics\_metrolyrics = [link for link in list\_links\_lyrics\_metrolyrics if '-lyrics-evanescence.html' in link]

In the case of metrolyrics an inspection on the result of the list comprehension revealed some links finishing with ‘/correction’, so we update the list comprehension above to eliminate the links with ‘/correction’.

# updated version

list\_links\_lyrics\_metrolyrics = [link for link in list\_links\_lyrics\_metrolyrics if ('-lyrics-evanescence.html' in link and '/correction' not in link)]

I’ve done the same procedure to the other 3 websites, i.e., apply our function to find hyperlinks, observe for patterns and use it to keep only the links of lyrics. [AZLyrics](https://www.azlyrics.com/) has also only one webpage with all lyrics, however [Songteksten.net](https://songteksten.net/) has 3 webpages with lyrics and [Songteksten.nl](https://www.songteksten.nl/) has 2 pages. In these last two cases we define main\_url as a list of webpages addresses and use a for-loop to apply our function and keep the results in list using the extend method.

# retrieving all hyperlinks  
urls = ['[https://songteksten.net/artist/lyrics/1938/evanescence.html'](https://songteksten.net/artist/lyrics/1938/evanescence.html%27),  
 '[https://songteksten.net/artist/lyrics/1938/evanescence/page/2.html'](https://songteksten.net/artist/lyrics/1938/evanescence/page/2.html%27),  
 '[https://songteksten.net/artist/lyrics/1938/evanescence/page/3.html'](https://songteksten.net/artist/lyrics/1938/evanescence/page/3.html%27)]

list\_links\_lyrics\_songteksten\_net = []

for url in urls:  
 list\_links\_lyrics\_songteksten\_net.extend(retrieve\_hyperlinks(url))  
   
# remove probable repetitions

list\_links\_lyrics\_songteksten\_net = list(set(list\_links\_lyrics\_songteksten\_net))

The interesting thing when observing the links I retrieved was that links with lyrics contain ***/songteksten.net/lyric/1938*** then I used here also some Python string methods to help with the filtering part.

# using url address to filter lyrics

spliting = urls[0].split('/')  
filter\_lyrics = spliting[2]+'/lyric/'+spliting[-2]

list\_links\_lyrics\_songteksten\_net = [link for link in list\_links\_lyrics\_songteksten\_net if (filter\_lyrics   
 in link) ]

After, working with all 4 websites I’ve decided to eliminate songteksten.nl because of the misspelling issues, and to work further with songteksten.net because it seemed to be the less complicate to extract lyrics.

**Extracting lyrics**

Now that we have hyperlinks for the lyrics is time to retrieve those lyrics and build a .csv file containing song\_title and lyrics.

First, it is important to check out some of the hyperlinks and inspect the *html* code of the webpage to identify which elements we want to extract from it, in our case lyrics.