Introduction to NLP 2022/2023
University of Warsaw
MA Cognitive Science

NLP analysis: Polish Rap Lyrics

By Sofija Krivokapić & Anna Sazonov

INTRODUCTION

Dear reader,

Welcome to our report for a research project under the name “NLP analysis: Polish Rap Music”! We are very excited to show you our project, methods we used, and all the exciting results we obtained.

Motivation

Apart from this research being part of our course requirement for *Introduction to Natural Language processing* class at University of Warsaw (incorporated in Master programme Cognitive Science), with this research we wanted to express our appreciation for Polish rap culture, and add our own contribution to it.

Summary

In our project we decided to look into the characteristics of Polish rap lyrics by analyzing unique words and checking which artists use more multisyllable words. We also performed topic classification (LDA) to learn about the most salient themes present in Polish rap music.

1. Data Collection

   Due to the lack of availability of corpora of Polish rap lyrics we created our own corpus through using a genius lyrics API and LyricsGenius library. This step was surprisingly challenging but we now know how to use available tools to scrape data off of web pages.

2. Data Analysis

   In the data analysis part of our project we employed many basic NLP tools we learned about in class after adapting them to suit the Polish language better.

1. Data collection & Analysis

Gathering the data

Given the lack of already available resources and corpora to work with, we resorted to making our own database. This consisted of several steps.

Obtaining Genius API

Genius (https://genius.com/) is a well-known website which hosts the world's biggest collection of song lyrics and in general, musical knowledge. So, we figured, If we wanted to obtain vast amounts of lyrics data and texts, this is where we should start.

First step consisted of obtaining Genius API (see how: https://docs.genius.com/#/getting-started-h1).

Installing **lyricsgenius** library

LyricsGenius library provides a simple interface to the song, artis, and lyrics data stored on Genius.com. All this is easily accessible through using their public API (see the documentation for this library: https://lyricsgenius.readthedocs.io/en/master/). For reference, see how this library works in the following chunk of code.

Python

```
!pip install lyricsgenius
```

Python

```
import lyricsgenius
from lyricsgenius import Genius

#my_token is a variable with genius API as a string
my_token  = "BbI104BwqEgGemxyZCxkV4EAgdjmJG7JV6snrcSMfG43JN4DkRe4RI_yRToc8_kw"
genius = lyricsgenius.Genius(my_token)

#.search_songs() looks for all of the songs of a particular artist
songs_schafter = genius.search_songs('schafter')

#accessing a particular song
songs_schafter["hits"][0]["result"]["id"]

#or a foor loop
```

```
for i in range(6):
    print(songs_schafter["hits"][i]["result"]["title"])
```

Gathering the names

This was a particularly tricky moment for us, because we weren't completely sure how to decide which artists to include and which ones not. So, we asked our good friend Chat GPT-3 to give us a list of Polish rappers and rap groups by decades. Since the list wasn't exhaustive, we decided to use our own knowledge and in fact, add all rappers we knew. So basically, in the end, we ended up with a list of 82 rappers, not sorted in any way particular. More or less popular, everyone was added.

Python

```
import pandas as pd

rappers_list = pd.read_csv("rapper list - Sheet1 (1).csv")
rappers_list = rappers_list["name"]

#rappers_list = list(rappers_list)
rappers_list = ['Tede', 'Paktofonika', 'Kaliber 44', 'Fisz Emade', 'Molesta
Ewenement', 'Wzgórze Ya-Pa 3', 'Kasta', 'Płomień 81', 'Hemp Gru', 'Eldo',
'Quebonafide', 'Paluch','JWP', 'Miuosh', 'Vienio', 'Bonus RPK', 'Łona i Webber',
'DonGURALesko', 'Sokół', 'Liroy', 'Kękę', 'Taco Hemingway', 'Otsochodzi',
'Bedoes', 'Tymek', 'Żabson', 'Young Multi', 'Białas', 'Solar', 'Kali',
'Kubańczyk', 'Kabe', 'Borixon', 'Malik Montana', 'PlanBe', 'Włodi',
'Rasmentalism', 'Wac Toja', 'Jan-rapowanie', 'Eldorado FM', 'B.R.O', 'Kuba
Knap', 'Kabe', 'Buka', 'Vixen', 'Szpaku', 'Biały', 'Kafar Dixon 37',
'Shellerini', 'ZBUKU', 'Bonson', 'Gutek', 'Pikers', 'Szad', 'Młody Dzban',
'Leh', 'Mata', 'Syny', 'UNDADASEA', 'Zdechły Osa', 'Stereofonia', 'Jacuś',
'Kuki', 'ĆPAJ STAJL', 'Belmondawg', 'Holak', 'asthma', 'schafter', 'Lanek',
'Pelson', 'Young Leosia', 'ten typ mes', 'Avi', 'Stare Miasto', 'Kukon',
'Warszafski Deszcz', 'Peja', 'Łona i Webber', 'O.S.T.R.', 'Problem', 'Pezet']
```

Then we had to check if the name spellings were correct and whether the artist actually existed on Genius.com, so we made a function to check that. Out of two rappers which were not found, one was removed from the list, first of all - well, because he didn't exist either on Genius.com or IRL, and the second one was misspelled, so we corrected it.

```python
def name_correct(my_list):
    not_found=[]
    for name in my_list:
        artist = genius.search_artist(name, max_songs=1, sort='title')
        if artist is None:
            not_found.append(name)
    for name in not_found:
        print(f"Artist not found: {name}.")
```

Later, we also realized that some artists may exist on Genius but have none of their songs there, so we ran our list through the function again, this time throwing out five artists (Avi, JWP, Leh, Kasta, Kali) whose songs we weren't able to find.

Making a dataset of rap songs

In order to perform the analyses we wished, we needed to construct a large .csv file with relevant information for each rapper on our list and for each of their songs. We wanted to have a dataframe with:

- Artist's name
- Song title
- Year of release
- Lyrics
- Tokenized lyrics

We decided to clean the lyrics of some unnecessary words already while downloading them into our database. We made a **regex** pattern which would remove headings and ads which were downloaded with the lyrics.

Then we also performed a simple tokenization with **nltk** tokenizer and saved the tokens into a separate column.

Due to missing information for some artists' songs, we needed to come up with two different functions for downloading their data. For some less popular artists, the functions in the **lyricsgenius** library did not have access to the year that the song was published in and for others this information was accessible. Hence, we ended up with one function that gets the correct year, and a second function that fills the year column with 0 to be manually filled later. (The function below is a combination of the two functions).

```python
import re
from nltk.tokenize import RegexpTokenizer
import pandas as pd

tokenizer = RegexpTokenizer(r'\w+')
pattern = r'(Intro|\d+\s+Contributors[\w+\s]*\w+|1 Contributor|You might also
likeEmbed|Refren|Hook|Lyrics|\d+Embed|Zwrotka(?: \d+)?)'

def get_everything(artist_name):
    result = genius.search_artist(artist_name, max_songs = 30, sort =
'popularity')
    list_of_dictionaries = list()
    for song in result.songs:
        cleaned_lyrics = re.sub(pattern, "", song.lyrics)
        tokenized_lyrics = tokenizer.tokenize(cleaned_lyrics)

        if genius.search_song(title = song.title) is not None:

            list_of_dictionaries.append({
                "artist name" : artist_name,
                "title": song.title,
```

```
                "year" : genius.search_song(title =
song.title).__dict__['_body']['release_date_components']['year'],
                "clean lyrics": cleaned_lyrics,
                "tokens" : tokenized_lyrics})
        else:
            list_of_dictionaries.append({
                "artist name" : artist_name,
                "title": song.title,
                "year" : 0,
                "clean lyrics": cleaned_lyrics,
                "tokens" : tokenized_lyrics})

    return pd.DataFrame.from_dict(list_of_dictionaries)

def make_csv(artist_list):
    result = pd.DataFrame()
    for artist in artist_list:
        artist_df = get_everything(artist)
        result = pd.concat([result, artist_df])
        result.to_csv(f"{artist}.csv", index=False)
    return result
```

One may wonder why we did not create just one function to do both options like the one above. That was due to timeout issues. That is also why we downloaded the data artist by artist with a limit of 30 songs per artist. Otherwise the operation kept timing out and progress was lost.

So, after having gone through our list of rap artists and having a separate .csv for each artist, we combined those into one big dataframe.

Python

```
import pandas as pd
dataset = pd.DataFrame()
```

```
for artist in rappers_list:
    single_df = pd.read_csv(f"{artist}.csv")
    dataset = pd.concat([dataset, single_df])

dataset.to_csv("Rappers_collected_10_May.csv", index=False, encoding = "UTF-8")
```

After a while we discovered that our dataset's token column was a string so we converted it into a list.

```
Python
for number in range(len(loaded_dataset)):
    loaded_dataset['tokens'][number] = eval(loaded_dataset['tokens'][number])
```

Removing stopwords

Since NLTK stopwords are not supported for Polish, we found a list of Polish stopwords on github.

```
Python
with open("polish.stopwords.txt", "r", encoding='utf-8') as file:
    data = file.read()
    data_into_list = data.split("\n")
```

As we didn't remove the stopwords before making the dataset, we just went through the column of tokens in our dataset removing those words which were stopwords and did not carry much useful information for our analysis.

```python
for number in range(len(loaded_dataset)):
    for word in loaded_dataset['tokens'][number]:
        if word.lower() in data_into_list:
            loaded_dataset['tokens'][number].remove(word)
```

Our analyses of unique words and multisyllable words were done on a dataset with removed stopwords.

Stemming and lemmatization

For the purpose of LDA (Latent Dirichlet Allocation) which we had planned to perform on our dataset, we had to lemmatize our tokens. LDA is a way of classifying themes present in a text. Before performing this analysis the tokens should be lemmatized, stemmed and short words should be removed (in an example of performing LDA on a base of English newspaper articles, len(token) < 3 were removed before analysis).

We used SpaCy for lemmatization.

```python
!pip install spacy
!python -m spacy download pl_core_news_lg
import spacy

# Load the Polish language model
nlp = spacy.load('pl_core_news_lg')
```

In the chunk of code below you can see how we lemmatized the tokens column of our dataset.

```python
for number in range(len(dataset_without_stopwords)):
    lemmatized_words = []
    # Lemmatize each word in the 'tokens' column of the current row
    for word in dataset_without_stopwords.loc[number, 'tokens']:
        doc = nlp(word)
        lemmatized_words.append(doc[0].lemma_)
    # Replace the 'tokens' column with the lemmatized words
    dataset_without_stopwords.at[number, 'tokens'] = lemmatized_words

# Print the first 10 lemmatized words in each row's 'tokens' column
for row in dataset_without_stopwords['tokens']:
    print(row[:10])
```

Extra cleaning

After performing a preliminary analysis of our data we discovered that it still needs some cleaning due to the presence of short, not very meaningful words, kind of like stopwords, which we didn't catch during our data cleaning stage.

Turns out that we did not remove stopwords carefully enough and performed our preliminary analysis with them in the dataset. We also noticed that many Polish past forms of verbs were lemmatized into two words. For example, 'chciałbym' was lemmatized into 'chcieć być', or 'wiedziałabym' into 'wiedzieć być'.

```python
import re
# regex pattern
pattern = r'(\b\w+)\sbyć\b'

for number in range(len(dataset_stemmed)):
    cleaned_stems = []
    for string in dataset_stemmed['tokens'][number]:
        clean_string = re.sub(pattern, r'\1', string)
```

```
        cleaned_stems.append(clean_string)

    dataset_stemmed.at[number, 'tokens'] = cleaned_stems
```

As you can see in the chunk of code above, this issue was solved with the help of regex.

2. Analyzing the data

After we created our dataset we could begin its analysis. Rap lyrics are often praised for being original if they contain distinctive or long words which are more difficult to create rhymes with. In order to learn more about Polish rap artists' lyric writing habits we analyzed how many unique words an artist uses, the average number of syllables in the words they use and how many of the words in their songs are multisyllabic.

Unique words per artist

In order to ascertain which Polish rapper uses the most unique words we counted how many unique words an artist has in their lyrics.

We encountered a problem of having varying numbers of songs per artist - with some having 30 (maximum number of songs per artist in our database) and others having just one or two. Which is why we decided to exclude artists who had less than 10 songs (Biały, Ziomcy, Stare Miasto, Gutek, Stereofonia) from this analysis.

```python
import pandas as pd

per_artist_unique = pd.DataFrame(columns=['Rapper', 'Count'])  # Create an empty DataFrame with specified column names

for artist in songs_per_artist.index:
    exclude = list()
```

```
    num_songs = songs_per_artist[artist]
    if num_songs < 10:
        exclude.append(artist)
    for number in range(len(lyrics_per_artist['artist name'])):

        name_rapper = lyrics_per_artist['artist name'][number]
        if name_rapper == artist:
            total_words = len(lyrics_per_artist['tokens'][number])/num_songs
            per_artist = len(set(lyrics_per_artist['tokens'][number]))/num_songs
            percentage = (per_artist/total_words)*100
            data = {'Rapper': name_rapper, 'Count': per_artist, 'Total_words' :
total_words, 'unique by total' : percentage}  # Create a dictionary with
'Rapper' and 'Count' keys
            per_artist_unique = per_artist_unique.append(data,
ignore_index=True)
        else:
            pass
    print(exclude)

per_artist_unique.sort_values(by = ['Count'], axis=0, ascending=False)
```

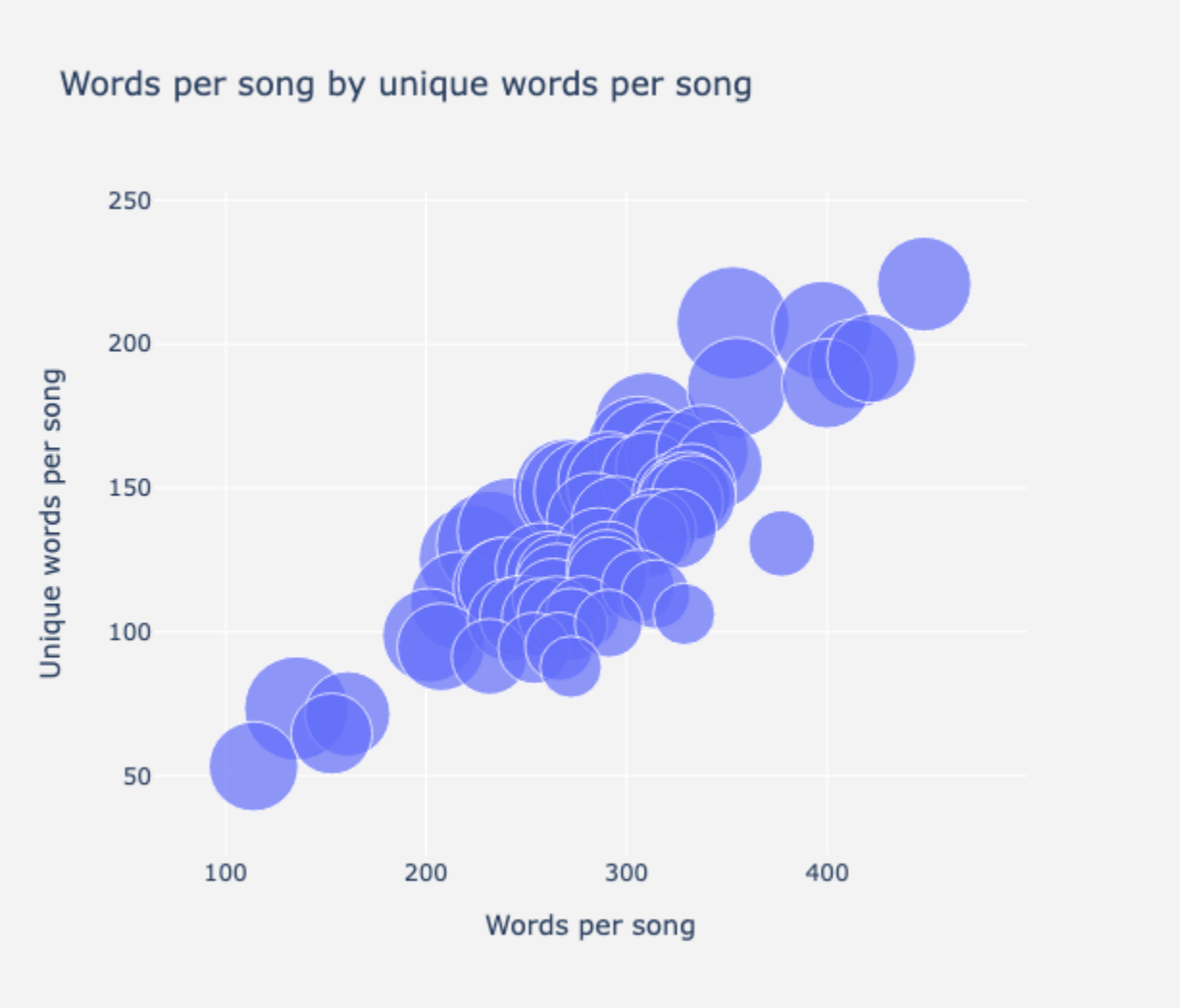
The artists who scored highest in terms of percentage of unique words per song were **Szad**, **Pelson**, **Młody Dzban**, **Włodi** and **ten typ mes**.

	Rapper	Count	Total_words	unique by total
10	Szad	207.433333	353.033333	58.757436
17	Pelson	125.433333	223.233333	56.189339
55	Młody Dzban	130.266667	232.133333	56.117174
21	Włodi	135.033333	242.366667	55.714482
27	ten typ mes	171.800000	310.333333	55.359828

On the other hand, those with the lowest percentages of unique words were: **KaBe, asthma, Wzgórze Ya-Pa 3, Young Multi** and **Tymek**.

47	Kabe	94.866667	266.466667	35.601701
65	asthma	102.882353	291.117647	35.340473
20	Wzgórze Ya-Pa 3	130.700000	377.300000	34.640869
22	Young Multi	106.200000	328.766667	32.302545
14	Tymek	87.766667	272.300000	32.231607

In the plot below you can see the relationship between unique words per song and words per song for all artists. In our jupyter notebook there is an interactive version of this plot where one can see which bubble belongs to which artist. The sizes of the bubbles represent the percentages of unique words an artist uses.



Top 10 words per artist

In order to see which rappers used which words most we counted word occurrences among their lyrics on lemmatized words with stopwords removed. We believe that this allows us to glimpse into the themes each artist sings about most often.

```python
from collections import Counter
```

```
def most_used_words(df, artist_list):
    common_words_by_artist = list()
    for artist in artist_list:
        artist_df = df[df['artist name'] == artist]
        all_tokens = [token for tokens_list in artist_df['tokens'] for token in
tokens_list]
        for token in all_tokens:
            if len(token) < 4:
# we chose to exclude words shorter than 4 because they were not very
informative
                all_tokens.remove(token)
            else:
                pass
        word_counts = Counter(all_tokens)
        most_common_words = word_counts.most_common(10)

        for word, count in most_common_words:
            one_artist = {
                'artist' : artist,
                'word' : word,
                'count' : count
            }

            common_words_by_artist.append(one_artist)
    return common_words_by_artist
```

Top 40 words among our artists' top 10s

Since it would be difficult to illustrate each of our artist's top 10 words, we made a word cloud with the **top 40 words among our artists' top 10s**.

Mean number of syllables per artist

In order to check the average length of words used by an artist in their lyrics we created a function which counts the number of syllables for each word in a list and then extracts the mean.

```python
import pyphen
# Load the Polish hyphenation dictionary
dic = pyphen.Pyphen(lang='pl_PL')

def calculate_mean_syllables(list_1):
    syllable_counts = [dic.inserted(word).count('-') + 1 for word in list_1]
    return np.mean(syllable_counts)

# Create an empty DataFrame with specified column names
per_artist_syllables = pd.DataFrame()

for number in range(len(lyrics_per_artist['tokens'])):
    syllables = calculate_mean_syllables(lyrics_per_artist['tokens'][number])
    rapper_name = lyrics_per_artist['artist name'][number]
    data = pd.DataFrame({'Rapper': rapper_name, 'Mean': syllables}, index=[0])
    per_artist_syllables = pd.concat([per_artist_syllables, data])
# List the rappers by mean syllable number descending
per_artist_syllables.sort_values(by = ['Mean'], ascending = False)
```

The artists who scored the highest mean syllables per word were Kuki, Bonus RPK, Molesta Ewenement, Paktofonika, and Peja.

Rapper	Mean
Kuki	2.177540
Bonus RPK	2.079259
Paktofonika	2.076969
Molesta Ewenement	2.062953
Peja	2.041112
Fisz Emade	2.028264
Stare Miasto	2.025798
Szad	2.025493
donGURALesko	2.022443
Młody Dzban	2.012493

Percentage of multisyllabic words per artist.

We also looked at how many percent of an artist's text consists of words longer than one syllable.

```python
import pyphen
def calculate_syllables(list_2):
    syllable_counts = [dic.inserted(word).count('-') + 1 for word in list_2]
    return syllable_counts

# Create an empty DataFrame with specified column names
per_artist_syllables_percentage = pd.DataFrame()
for number in range(len(lyrics_per_artist['tokens'])):
    syllables_2 = calculate_syllables(lyrics_per_artist['tokens'][number])
    rapper_name = lyrics_per_artist['artist name'][number]
```

```
    percentage = sum(1 for syllable in syllables_2 if syllable > 1) /
len(syllables_2) * 100
    data1 = pd.DataFrame({'Rapper': [rapper_name], 'Syll': [syllables_2], '% or
multisyllabic' : [percentage]})

    per_artist_syllables_percentage =
pd.concat([per_artist_syllables_percentage, data1])
per_artist_syllables_percentage
```

The mean percentage of multisyllabic words among our dataset was 68.3%.

The artists with the highest percentage of multisyllabic words were Kuki, Bonus RPK, Pelson, Stereofonia and Taco Hemingway. The artists with the lowest percentages of multisyllabic words were asthma, Young Multi, Kabe, Schafter and Bonson.

3. Latent Dirichlet Analysis (LDA)

LDA is a way of performing unsupervised theme discovery in a dataset made up of words. It allows us to learn about themes present in a dataset (what we are most looking forward to doing) and additionally it can later serve as a way of classifying a song into one of the topics found in the data.

The gensim library is needed in order to perform LDA. We also imported the simple_preprocess module.

Python

```
!pip install gensim
import gensim
from gensim.utils import simple_preprocess
```

Then we created a dictionary from our dataset_stemmed tokens column containing the number of times a word appears in the training set.

```
Python

dictionary = gensim.corpora.Dictionary(dataset_stemmed['tokens'])
```

Next we filtered out tokens which appeared in less than 5 songs or more than half of the songs. After the two steps, we kept 100 000 of the most frequent tokens.

```
Python

dictionary.filter_extremes(no_below=5, no_above=0.5, keep_n=100000)
```

For each document - in our case for each song - we created a dictionary reporting how many words and how many times those words appear - a bag of words. Then we inspected how this operation worked out.

```
Python

bow_corpus = [dictionary.doc2bow(doc) for doc in dataset_stemmed['tokens']]

#Preview how this worked

bow_doc_4310 = bow_corpus[10]

for i in range(len(bow_doc_4310)):

    print("Word {} (\"{}\") appears {} time.".format(bow_doc_4310[i][0],
```

```
                                    dictionary[bow_doc_4310[i][0]],
                                    bow_doc_4310[i][1]))
```

The code above resulted in the output pictured below. We were happy with the result.

```
Word 1 ("Bedoes") appears 1 time.
Word 2 ("Borek") appears 2 time.
Word 13 ("Polska") appears 1 time.
Word 14 ("SB") appears 3 time.
Word 34 ("brzydko") appears 1 time.
Word 46 ("czarny") appears 1 time.
Word 48 ("człowiek") appears 1 time.
Word 50 ("dać") appears 1 time.
Word 54 ("drugi") appears 4 time.
Word 55 ("dupa") appears 3 time.
Word 77 ("jakbym") appears 1 time.
Word 90 ("koncert") appears 1 time.
Word 108 ("musieć") appears 1 time.
Word 112 ("młody") appears 4 time.
```

We decided to run LDA in two different ways, using Bag of Words and TF-IDF. In both approaches we trained our LDA model using gensim.models.LdaMulticore.

Python

```
lda_model = gensim.models.LdaMulticore(bow_corpus, num_topics=3,
id2word=dictionary, passes=2, workers=2)
```

At first we tried to run it by setting the number of topics to 5 but due to the fact that the topics seemed to all contain very similar words, we decided to cut down the number of topics we were looking for to three.

```python
for idx, topic in lda_model.print_topics(-1):
    print('Topic: {} \nWords: {}'.format(idx, topic))
```

```
Topic: 0
Words: 0.007*"móc" + 0.006*"życie" + 0.005*"człowiek" + 0.005*"dobry" + 0.005*"czas" + 0.005
*"kurwa" + 0.005*"żyć" + 0.005*"robić" + 0.004*"widzieć" + 0.004*"dać"
Topic: 1
Words: 0.006*"życie" + 0.005*"człowiek" + 0.005*"widzieć" + 0.005*"świat" + 0.005*"miasto" +
0.005*"robić" + 0.004*"czas" + 0.004*"swój" + 0.004*"dzień" + 0.004*"raz"
Topic: 2
Words: 0.007*"robić" + 0.006*"życie" + 0.006*"kurwa" + 0.005*"dać" + 0.004*"czas" + 0.004*"mó
c" + 0.004*"nowy" + 0.004*"człowiek" + 0.004*"to" + 0.003*"świat"
```

In order to get a grasp of which of the topics found in our data could refer to what we generated simple word clouds based on the order of the words assigned to each of the topics:

It seems that **topic 1** has the unique word ‘dobry’ - good, and focuses on possibility and life.

Topic 2 can be characterized by the presence of the word ‘miasto’ which means city and again focuses on life but also has words such as ‘swój’ (own, as in one’s own), ‘człowiek’ (human, person) and ‘widzieć’ (to see).

Topic 3 focuses on time, life, doing and giving. All of the topics have ‘kurwa’ in them, the most popular of Polish vulgarities.

TF-IDF model

We also ran LDA using TF-IDF. For that, we created a TF-IDF model object using models and transformed our corpus to have tf-idf scores.

```python
from gensim import corpora, models
tfidf = models.TfidfModel(bow_corpus)
corpus_tfidf = tfidf[bow_corpus]
# Previewing tf-idf scores for our first song
from pprint import pprint
for doc in corpus_tfidf:
    pprint(doc)
    break
```

We trained the TF-IDF LDA on the tf-idf corpus, and again had to revise the number of topics down from 5 to three.

```python
lda_model_tfidf = gensim.models.LdaMulticore(corpus_tfidf, num_topics=3,
id2word=dictionary, passes=2, workers=4)
```

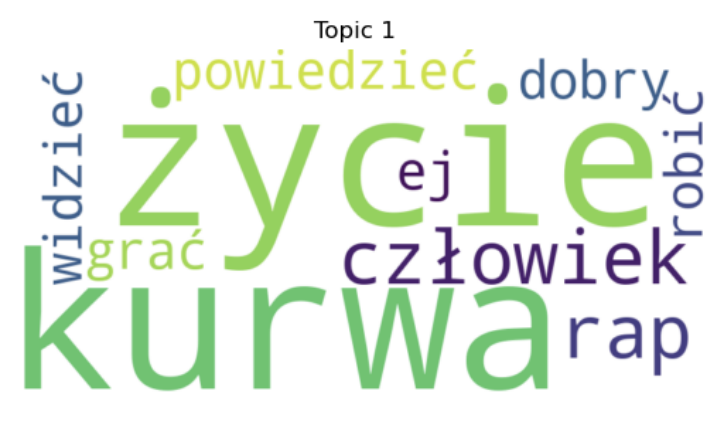
```python
for idx, topic in lda_model_tfidf.print_topics(-1):

    print('Topic: {} Word: {}'.format(idx, topic))
```

```
Topic: 0 Word: 0.001*"świat" + 0.001*"życie" + 0.001*"dzień" + 0.001*"mó
c" + 0.001*"dać" + 0.001*"czas" + 0.001*"człowiek" + 0.001*"miasto" + 0.0
01*"robić" + 0.001*"kurwa"
Topic: 1 Word: 0.001*"kurwa" + 0.001*"życie" + 0.001*"człowiek" + 0.001
*"rap" + 0.001*"powiedzieć" + 0.001*"dobry" + 0.001*"grać" + 0.001*"widzi
eć" + 0.001*"robić" + 0.001*"ej"
Topic: 2 Word: 0.001*"the" + 0.001*"ej" + 0.001*"życie" + 0.001*"kurwa" +
0.001*"dom" + 0.001*"robić" + 0.001*"czuć" + 0.001*"móc" + 0.001*"młody"
+ 0.001*"widzieć"
```

In the TF-IDF model we can see some more variety in the topics making it a bit easier to come up with an interpretation.

Topic 1 seems to center around life, the world, the city and the day.



Topic 2 seems to center around life and people, rap and expressing oneself (‘powiedzieć’ - to say and ‘ej’).

Topic 3 has the word ‘dom’ in it which means home or house, as well as the word ‘młody’ meaning young and ‘czuć’ - to feel. This could be interpreted as a topic of home and youth and feelings.

Interestingly enough, before handing our project in, we also discovered that running the tf-idf model training with a larger number of passes produced the most out of the ordinary topics.

```
lda_model_tfidf = gensim.models.LdaMulticore(corpus_tfidf, num_topics=3, id2word=dictionary, passes=4, workers=4)
```

```
for idx, topic in lda_model_tfidf.print_topics(-1):
    print('Topic: {} Word: {}'.format(idx, topic))
```

```
Topic: 0 Word: 0.001*"życie" + 0.001*"człowiek" + 0.001*"kurwa" + 0.001*"świat" + 0.001*"dzień" + 0.001*"czas" + 0.00
1*"dobry" + 0.001*"móc" + 0.001*"rap" + 0.001*"robić"
Topic: 1 Word: 0.002*"the" + 0.001*"that" + 0.001*"you" + 0.001*"yeah" + 0.001*"s" + 0.001*"t" + 0.001*"oh" + 0.001
*"ej" + 0.001*"Montana" + 0.001*"in"
Topic: 2 Word: 0.001*"Knap" + 0.001*"tańcz" + 0.001*"Kuba" + 0.001*"kurwa" + 0.001*"kot" + 0.001*"se" + 0.001*"młody"
+ 0.001*"dupa" + 0.001*"robić" + 0.001*"puk"
```

Evaluating the models

With LDA, we could not only learn about hidden themes in our data but also see how a document from outside of our corpus would be classified by our models.

For that we ran a quick lemmatization on a song by Młody Leh (Miasto nocą) whom we did not include in our original analysis. The song lyrics, turned into a bow, figure in the code below under 'dictionary_words'.

Python

```
for index, score in sorted(lda_model[dictionary_words], key=lambda tup:
-1*tup[1]):

    print("\nScore: {}\t \nTopic: {}".format(score, lda_model.print_topic(index,
10)))
```

```
Score: 0.9220494627952576
Topic: 0.007*"życie" + 0.006*"człowiek" + 0.006*"móc" + 0.006*"czas" + 0.005*"świat" + 0.005*"swój" + 0.005*"widzieć"
+ 0.005*"dzień" + 0.004*"żyć" + 0.004*"siebie"

Score: 0.07639899849891663
Topic: 0.008*"robić" + 0.007*"kurwa" + 0.005*"ej" + 0.005*"życie" + 0.005*"dać" + 0.005*"dobry" + 0.004*"człowiek" +
0.004*"widzieć" + 0.004*"raz" + 0.004*"zrobić"
```

We expected the song to be classified in the topic connected to miasto - city due to its title but it seems that our model decided that it is more about life, people and possibility (topic number one in the bag of words based model).

```python
for index, score in sorted(lda_model_tfidf[dictionary_words], key=lambda tup:
-1*tup[1]):

    print("\nScore: {}\t \nTopic: {}".format(score,
lda_model_tfidf.print_topic(index, 10)))
```

```
Score: 0.9374219179153442
Topic: 0.001*"życie" + 0.001*"dzień" + 0.001*"ej" + 0.001*"człowiek" + 0.001*"świat" + 0.001*"kurwa" + 0.001*"dobry"
+ 0.001*"dać" + 0.001*"widzieć" + 0.001*"móc"

Score: 0.06086229532957077
Topic: 0.001*"życie" + 0.001*"czas" + 0.001*"robić" + 0.001*"dom" + 0.001*"dzień" + 0.001*"świat" + 0.001*"nowy" + 0.
001*"żyć" + 0.001*"serce" + 0.001*"swój"
```

The tf-idf model classified it into its second topic which we defined as the theme of life, people, rap and self-expression.

CONCLUSION

In conclusion, our research project on "NLP analysis: Polish Rap Music" has been an introductory journey into the world of natural language processing and its application to the realm of Polish rap lyrics.

Throughout the project, we encountered various challenges, such as the lack of available corpora of Polish rap lyrics which we had to overcome by making our own dataset. This allowed us to explore the characteristics of Polish rap lyrics by analyzing unique words, multisyllable words, and performing topic classification using Latent Dirichlet Analysis (LDA).

Our analysis provided valuable insights into the lyrical styles of different Polish rap artists. We examined the usage of unique words per artist, identifying those who exhibited the most distinctive vocabularies. Additionally, we delved into the average number of syllables used by each artist, shedding light on the complexity of their lyrics. Moreover, the topic classification revealed prevalent themes in Polish rap music, ranging from life and city experiences to self-expression and emotion.

Overall, this project has not only enhanced our understanding of NLP techniques but also deepened our appreciation for Polish rap as a form of artistic expression. The combination of language and culture presented an exciting and challenging landscape to explore. We hope that our research contributes to the existing knowledge about Polish rap music and inspires further investigations into the rich and diverse world of NLP applications in the domain of music and culture.