

Projekt, Inteligencja Obliczeniowa

# Operacje na tekście na przykładzie analizy piosenek

Marta Leszczyńska,  
nr indeksu 234325  
Informatyka II, niestacjonarne

# Wstęp

Niniejszy projekt został wykonany na potrzeby przedmiotu Inteligencja Obliczeniowa, wykładanego na Uniwersytecie Gdańskim. Projekt ma na celu ukazanie użycia wybranych metod analizy tekstu, takich jak:

- tokenizacja
- lematyzacja
- analiza tematu/tematów
- analiza słów kluczowych
- analiza podobieństwa tekstów
- spełnienie prawa Zipf'a

Projekt wykonano w oparciu o wiedzę wyniesioną z zajęć laboratoryjnych oraz z artykułów dostępnych w Internecie. Projekt wykonano w języku programowania Python.

W niniejszym projekcie wykorzystano następujące pakiety:

- NLTK
- gensim
- WordCloud
- networkx
- matplotlib
- csv
- pickle
- string
- itertools

## 1. Wybór i przygotowanie danych

Baza danych, na której przeprowadzona została niniejsza analiza tekstu, pochodzi kaggle.com: [55000+ Song Lyrics](#) . Dane zostały pozyskane za pomocą metody web - scraping'u. Pierwotna baza zawierała ponad 55000 rekordów (piosenek) pogrupowanych alfabetycznie według zespołów. Baza została przeze mnie podzielona na pliki .csv o długościach po 1 000 rekordów, z czego do niniejszego projektu wykorzystano i przetworzono 14 plików po 1000 rekordów. Wyniki zaprezentowane w dalszych częściach projektu to przykłady analizy poszczególnych autorów i ich piosenek.

Pierwszym krokiem było pogrupowanie piosenek względem zespołów, co zostało zrealizowane automatycznie w języku Python, podobnie jak reszta operacji. Następnie,

otrzymane wyniki zostały poddane tokenizacji oraz lematyzacji, przy pomocy pakietu NLTK. Na potrzeby innych operacji, pliki zostały również poddane łączeniu piosenek dla zespołów.

Na każdym etapie obróbki danych należało zapisać wyniki do pliku .pkl, z uwagi na długi czas wykonywania poszczególnych operacji. Długi czas przetwarzania wynikał z iterowania słowo po słowie każdej piosenki każdego zespołu.

```
def get_data(filepath):
    data = []
    file = open(filepath)
    csv_file = csv.reader(file)

    for row in csv_file:
        data.append(row)

    return data

def get_artists_dict(data):
    dict = {}
    for row in data:
        dict[row[index_artist]] = []
    for row in data:
        dict[row[index_artist]].append(row[index_text])
    return dict

def save_object_to_file(object, name):
    with open(objects_repo_location + name, 'wb') as file:
        pickle.dump(object, file, pickle.HIGHEST_PROTOCOL)

def load_object(name):
    with open(objects_repo_location + name, 'rb') as file:
        return pickle.load(file)
```

*Metody przetwarzania, zapisu do pliku .pkl i jego odczytu*

```
def tokenize_dict(dict_to_tokenize):
    tokenized_dict = {}
    for key, value in dict_to_tokenize.items():
        song_texts_list_for_one_artist = value
        artist_name = key
        tokenized_texts_list_for_one_artist = []
        for song_text in song_texts_list_for_one_artist:
            song_text_without_line_breaks = song_text.replace("\n", " ")
            tokenized_text = word_tokenize(song_text_without_line_breaks)
            tokenized_texts_list_for_one_artist.append(tokenized_text)
        tokenized_dict[artist_name] = tokenized_texts_list_for_one_artist
    return tokenized_dict

def lemmatize_dict(tokenized_dict):
    lem = WordNetLemmatizer()
    lemmatized_dict = {}
    for key, value in tokenized_dict.items():
        tokenized_songs = value
        artist_name = key
        lemmatized_song_texts = []
        for tokenized_song in tokenized_songs:
            lemmatized_song = [lem.lemmatize(word, get_wordnet_pos(word)) for word in tokenized_song]
            lemmatized_song_texts.append(lemmatized_song)
        lemmatized_dict[artist_name] = lemmatized_song_texts
    return lemmatized_dict
```

*Metody tokenizacji i lematyzacji*

## 2. Chmury najczęstszych słów

Dla każdego artysty dokonano analizy najczęściej używanych słów i zwizualizowano wyniki jako chmury tych słów. Użyto do tego pakietu WordCloud.



*Chmura najczęstszych słów zespołu The Beatles*



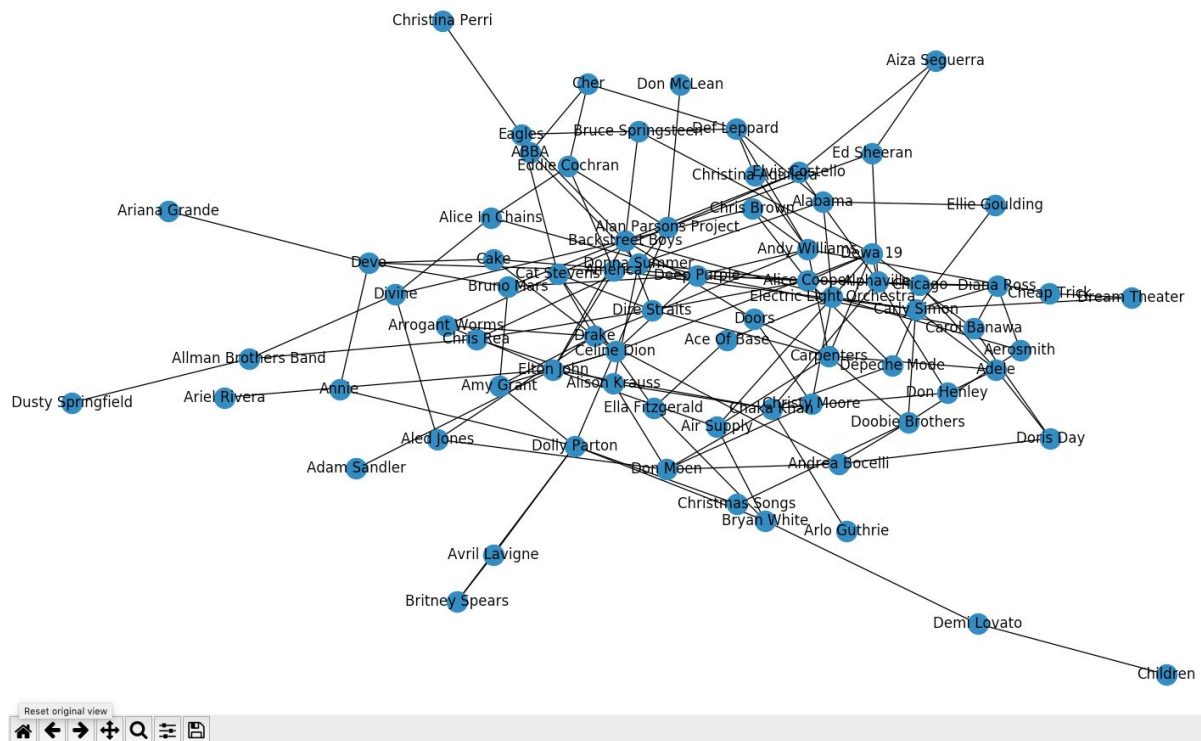
*Chmura najczęstszych słów Michael'a Jackson'a*

```
def create_wordcloud(dict_to_create_wordcloud):  
    for key, value in dict_to_create_wordcloud.items():  
        artist = key  
        text = value  
        text_list = []  
        for data in text:  
            word = data[0]  
            freq = data[1]  
            for i in range(freq):  
                text_list.append(word)  
        new_text = ' '.join(text_list)  
  
        wordcloud = WordCloud(collocations=False, background_color="white", max_font_size=50, max_words=100).generate(new_text)  
        wordcloud.to_file("/Users/reckony/Desktop/UGSTUDIA/I0project/Wordclouds/" + f'Wordcloud_{artist}.png')  
    return True
```

*Metoda tworząca chmury słów i zapisująca je do pliku*

### 3. Podobieństwo zespołów

Do określenia podobieństw między zespołami wykorzystano podobieństwo cosinusowe. Przykładowy graf został stworzony przy pomocy paczki networkx. Jako węzły podano nazwy zespołów, natomiast miara podobieństwa cosinus'owego podano jako wagę połączenia.



Graf podobieństwa dla wybranych zespołów

```
def get_cosine_similarity(tokenized_filtered_text1, tokenized_filtered_text2):
    text1 = []
    text2 = []
    text1_tokenized = word_tokenize(tokenized_filtered_text1)
    text2_tokenized = word_tokenize(tokenized_filtered_text2)
    sw = stopwords.words('english')

    text1_set = {w for w in text1_tokenized if not w in sw}
    text2_set = {w for w in text2_tokenized if not w in sw}

    rvector = text1_set.union(text2_set)
    for w in rvector:
        if w in text1_set:
            text1.append(1) # create a vector
        else:
            text1.append(0)
        if w in text2_set:
            text2.append(1)
        else:
            text2.append(0)

    c = 0
    # cosine formula
    for i in range(len(rvector)):
        c += text1[i] * text2[i]
    cosine = c / float((sum(text1) * sum(text2)) ** 0.5)

    return cosine
```

Metoda obliczająca podobieństwo cosinusowe



## 4. Wydobywanie tematu piosenek

Do wydobywania tematów piosenek użyto pakietu gensim. Za jego pomocą zbudowany został model LDAModel, który szukał 5 głównych tematów w zdaniach. Dzięki wynikom możemy domyślić się, o czym jest dana piosenka, nie czytając / nie słuchając jej.

```
Foreigner, Mountain of love:
('0.161*"honey" + 0.057*"touch" + 0.057*"baby" + 0.055*"feel" + 0.031*"pretendin"')
('0.139*"high" + 0.095*"little" + 0.072*"build" + 0.050*"edge" + 0.050*"\'cause"')
('0.268*"love" + 0.255*"mountain" + 0.056*"climb" + 0.042*"risin" + 0.068*"good")

Gloria Estefan, Along came you:
('0.241*"love" + 0.156*"baby" + 0.139*"come" + 0.088*"yes" + 0.037*"certain"')
('0.168*"along" + 0.127*"oh" + 0.127*"thula" + 0.066*"show" + 0.045*"angel"')
('0.240*"know" + 0.124*"knew" + 0.047*"apart" + 0.047*"live" + 0.047*"look"')

Glee, Away in a manger
('0.175*"directions" + 0.092*"love" + 0.092*"kitty" + 0.050*"baby" + 0.050*"bed"')
('0.208*"lord" + 0.088*"sky" + 0.088*"stay" + 0.048*"morning" + 0.048*"cattle"')
('0.140*"new" + 0.107*"little" + 0.073*"look" + 0.040*"head" + 0.040*"hay"')
```

Porównanie tematów wybranych piosenek

Możemy zauważyć, że zarówno piosenka zespołu Foreigner, Glorii Estefan oraz Glee, nawiązują do miłości.

```
Gloria Estefan, Bad Boy:
('0.100*"drive" + 0.100*"right" + 0.100*"heart" + 0.100*"oh" + 0.100*"phone"')
('0.641*"boy" + 0.107*"nothin" + 0.072*"good" + 0.038*"go" + 0.021*"night"'),
('0.632*"bad" + 0.052*"breathless" + 0.052*"gettin" + 0.052*"always" + 0.052*"restless"')

Glee, Cool:
('0.356*"cool" + 0.096*"easy" + 0.096*"loose" + 0.096*"breeze" + 0.052*"take"'),
('0.169*"got" + 0.092*"school" + 0.092*"coolie" + 0.092*"like" + 0.092*"high"')
('0.423*"boy" + 0.121*"crazy" + 0.051*"buzz" + 0.051*"get" + 0.051*"turn"')
```

Porównanie tematów wybranych piosenek

Natomiast powyższe dwie piosenki dotyczą młodości, spontaniczności.

```
def get_main_topics(dict_to_check):
    topics_dict = {}
    for key, value in dict_to_check.items():
        artist = key
        songs = value
        songs_topic_list = []
        for song in songs:
            song1 = [d.split() for d in song1]
            dictionary = corpora.Dictionary(song1)
            corpus = [dictionary.doc2bow(text) for text in song1]
            ldamodel = gensim.models.ldamodel.LdaModel(corpus, num_topics=NUM_TOPICS, id2word=dictionary, passes=15)
            topics = ldamodel.print_topics(num_words=5)
            songs_topic_list.append(songs.index(song))
            songs_topic_list.append(topics)
        topics_dict[artist] = songs_topic_list
    return topics_dict
```

Metoda szukania tematów w tekście

## 5. Prawo Zipfa

Ostatnią analizą przeprowadzoną na piosenkach jest sprawdzenie, czy zachodzi w nich prawo Zipfa. Prawo to głosi, że ranga słowa i częstotliwość jego występowania pomnożone przez siebie, powinny stanowić constans. Patrząc na poniższe wyniki można zauważyć, że odchylenia między poszczególnymi słowami jest dosyć spore.

| Rank | Word       | Actual Freq | Zipf Frac | Zipf Freq | Actual Diff | Pct Diff |
|------|------------|-------------|-----------|-----------|-------------|----------|
| 1    | know       | 104         | 1/1       | 104.00    | 0.00        | 100.00%  |
| 2    | come       | 87          | 1/2       | 52.00     | 35.00       | 167.31%  |
| 3    | yeah       | 85          | 1/3       | 34.67     | 50.33       | 245.19%  |
| 4    | love       | 75          | 1/4       | 26.00     | 49.00       | 288.46%  |
| 5    | never      | 67          | 1/5       | 20.80     | 46.20       | 322.12%  |
| 6    | want       | 62          | 1/6       | 17.33     | 44.67       | 357.69%  |
| 7    | could      | 62          | 1/7       | 14.86     | 47.14       | 417.31%  |
| 8    | like       | 52          | 1/8       | 13.00     | 39.00       | 400.00%  |
| 9    | time       | 48          | 1/9       | 11.56     | 36.44       | 415.38%  |
| 10   | when       | 42          | 1/10      | 10.40     | 31.60       | 403.85%  |
| 11   | life       | 42          | 1/11      | 9.45      | 32.55       | 444.23%  |
| 12   | ever       | 41          | 1/12      | 8.67      | 32.33       | 473.08%  |
| 13   | lost       | 41          | 1/13      | 8.00      | 33.00       | 512.50%  |
| 14   | world      | 38          | 1/14      | 7.43      | 30.57       | 511.54%  |
| 15   | feel       | 38          | 1/15      | 6.93      | 31.07       | 548.08%  |
| 16   | always     | 38          | 1/16      | 6.50      | 31.50       | 584.62%  |
| 17   | light      | 34          | 1/17      | 6.12      | 27.88       | 555.77%  |
| 18   | every      | 34          | 1/18      | 5.78      | 28.22       | 588.46%  |
| 19   | what       | 34          | 1/19      | 5.47      | 28.53       | 621.15%  |
| 20   | cause      | 33          | 1/20      | 5.20      | 27.80       | 634.62%  |
| 21   | rain       | 33          | 1/21      | 4.95      | 28.05       | 666.35%  |
| 22   | nothing    | 32          | 1/22      | 4.73      | 27.27       | 676.92%  |
| 23   | find       | 30          | 1/23      | 4.52      | 25.48       | 663.46%  |
| 24   | that       | 30          | 1/24      | 4.33      | 25.67       | 692.31%  |
| 25   | would      | 29          | 1/25      | 4.16      | 24.84       | 697.12%  |
| 26   | eyes       | 29          | 1/26      | 4.00      | 25.00       | 725.00%  |
| 27   | right      | 28          | 1/27      | 3.85      | 24.15       | 726.92%  |
| 28   | going      | 28          | 1/28      | 3.71      | 24.29       | 753.85%  |
| 29   | everything | 28          | 1/29      | 3.59      | 24.41       | 780.77%  |
| 30   | just       | 28          | 1/30      | 3.47      | 24.53       | 807.69%  |
| 31   | arms       | 27          | 1/31      | 3.35      | 23.65       | 804.81%  |
| 32   | maybe      | 26          | 1/32      | 3.25      | 22.75       | 800.00%  |
| 33   | live       | 25          | 1/33      | 3.15      | 21.85       | 793.27%  |
| 34   | feeling    | 25          | 1/34      | 3.06      | 21.94       | 817.31%  |
| 35   | away       | 24          | 1/35      | 2.97      | 21.03       | 807.69%  |
| 36   | back       | 24          | 1/36      | 2.89      | 21.11       | 830.77%  |
| 37   | hear       | 23          | 1/37      | 2.81      | 20.19       | 818.27%  |
| 38   | sing       | 22          | 1/38      | 2.74      | 19.26       | 803.85%  |
| 39   | good       | 22          | 1/39      | 2.67      | 19.33       | 825.00%  |
| 40   | came       | 22          | 1/40      | 2.60      | 19.40       | 846.15%  |
| 41   | around     | 21          | 1/41      | 2.54      | 18.46       | 827.88%  |
| 42   | things     | 21          | 1/42      | 2.48      | 18.52       | 848.08%  |
| 43   | water      | 21          | 1/43      | 2.42      | 18.58       | 868.27%  |
| 44   | look       | 21          | 1/44      | 2.36      | 18.64       | 888.46%  |

Prawo Zipfa dla zespołu Coldplay

| Rank | Word    | Actual Freq | Zipf Frac | Zipf Freq | Actual Diff | Pct Diff |
|------|---------|-------------|-----------|-----------|-------------|----------|
| 1    | never   | 57          | 1/1       | 57.00     | 0.00        | 100.00%  |
| 2    | hell    | 43          | 1/2       | 28.50     | 14.50       | 150.88%  |
| 3    | take    | 42          | 1/3       | 19.00     | 23.00       | 221.05%  |
| 4    | united  | 40          | 1/4       | 14.25     | 25.75       | 280.70%  |
| 5    | love    | 39          | 1/5       | 11.40     | 27.60       | 342.11%  |
| 6    | know    | 35          | 1/6       | 9.50      | 25.50       | 368.42%  |
| 7    | night   | 32          | 1/7       | 8.14      | 23.86       | 392.98%  |
| 8    | last    | 32          | 1/8       | 7.12      | 24.88       | 449.12%  |
| 9    | like    | 31          | 1/9       | 6.33      | 24.67       | 489.47%  |
| 10   | blood   | 29          | 1/10      | 5.70      | 23.30       | 508.77%  |
| 11   | come    | 28          | 1/11      | 5.18      | 22.82       | 540.35%  |
| 12   | want    | 26          | 1/12      | 4.75      | 21.25       | 547.37%  |
| 13   | time    | 25          | 1/13      | 4.38      | 20.62       | 570.18%  |
| 14   | around  | 24          | 1/14      | 4.07      | 19.93       | 589.47%  |
| 15   | sinner  | 24          | 1/15      | 3.80      | 20.20       | 631.58%  |
| 16   | rose    | 24          | 1/16      | 3.56      | 20.44       | 673.68%  |
| 17   | stand   | 23          | 1/17      | 3.35      | 19.65       | 685.96%  |
| 18   | summer  | 21          | 1/18      | 3.17      | 17.83       | 663.16%  |
| 19   | give    | 20          | 1/19      | 3.00      | 17.00       | 666.67%  |
| 20   | inside  | 20          | 1/20      | 2.85      | 17.15       | 701.75%  |
| 21   | that    | 20          | 1/21      | 2.71      | 17.29       | 736.84%  |
| 22   | away    | 19          | 1/22      | 2.59      | 16.41       | 733.33%  |
| 23   | life    | 19          | 1/23      | 2.48      | 16.52       | 766.67%  |
| 24   | fall    | 19          | 1/24      | 2.38      | 16.62       | 800.00%  |
| 25   | when    | 18          | 1/25      | 2.28      | 15.72       | 789.47%  |
| 26   | stained | 17          | 1/26      | 2.19      | 14.81       | 775.44%  |
| 27   | hand    | 17          | 1/27      | 2.11      | 14.89       | 805.26%  |
| 28   | eyes    | 16          | 1/28      | 2.04      | 13.96       | 785.96%  |
| 29   | fire    | 16          | 1/29      | 1.97      | 14.03       | 814.04%  |
| 30   | hear    | 16          | 1/30      | 1.90      | 14.10       | 842.11%  |
| 31   | found   | 16          | 1/31      | 1.84      | 14.16       | 870.18%  |
| 32   | bent    | 16          | 1/32      | 1.78      | 14.22       | 898.25%  |
| 33   | cause   | 16          | 1/33      | 1.73      | 14.27       | 926.32%  |
| 34   | world   | 15          | 1/34      | 1.68      | 13.32       | 894.74%  |
| 35   | every   | 15          | 1/35      | 1.63      | 13.37       | 921.05%  |
| 36   | saints  | 15          | 1/36      | 1.58      | 13.42       | 947.37%  |
| 37   | feel    | 15          | 1/37      | 1.54      | 13.46       | 973.68%  |
| 38   | fear    | 14          | 1/38      | 1.50      | 12.50       | 933.33%  |
| 39   | light   | 14          | 1/39      | 1.46      | 12.54       | 957.89%  |
| 40   | with    | 14          | 1/40      | 1.43      | 12.57       | 982.46%  |
| 41   | death   | 14          | 1/41      | 1.39      | 12.61       | 1007.02% |
| 42   | savage  | 14          | 1/42      | 1.36      | 12.64       | 1031.58% |
| 43   | what    | 13          | 1/43      | 1.33      | 11.67       | 980.70%  |
| 44   | better  | 13          | 1/44      | 1.30      | 11.70       | 1003.51% |
| 45   | near    | 13          | 1/45      | 1.27      | 11.73       | 1026.32% |

*Prawo Zipfa dla zespołu Judas Priest*

```
def _create_zipf_table(frequencies):
    zipf_table = []
    top_frequency = frequencies[0][1]

    for index, item in enumerate(frequencies, start=1):
        relative_frequency = "1/{}".format(index)
        zipf_frequency = top_frequency * (1 / index)
        difference_actual = item[1] - zipf_frequency
        difference_percent = (item[1] / zipf_frequency) * 100

        zipf_table.append({"word": item[0],
                           "actual_frequency": item[1],
                           "relative_frequency": relative_frequency,
                           "zipf_frequency": zipf_frequency,
                           "difference_actual": difference_actual,
                           "difference_percent": difference_percent})

    return zipf_table
```

*Jedna z metod obliczających zgodność z prawem Zipfa*

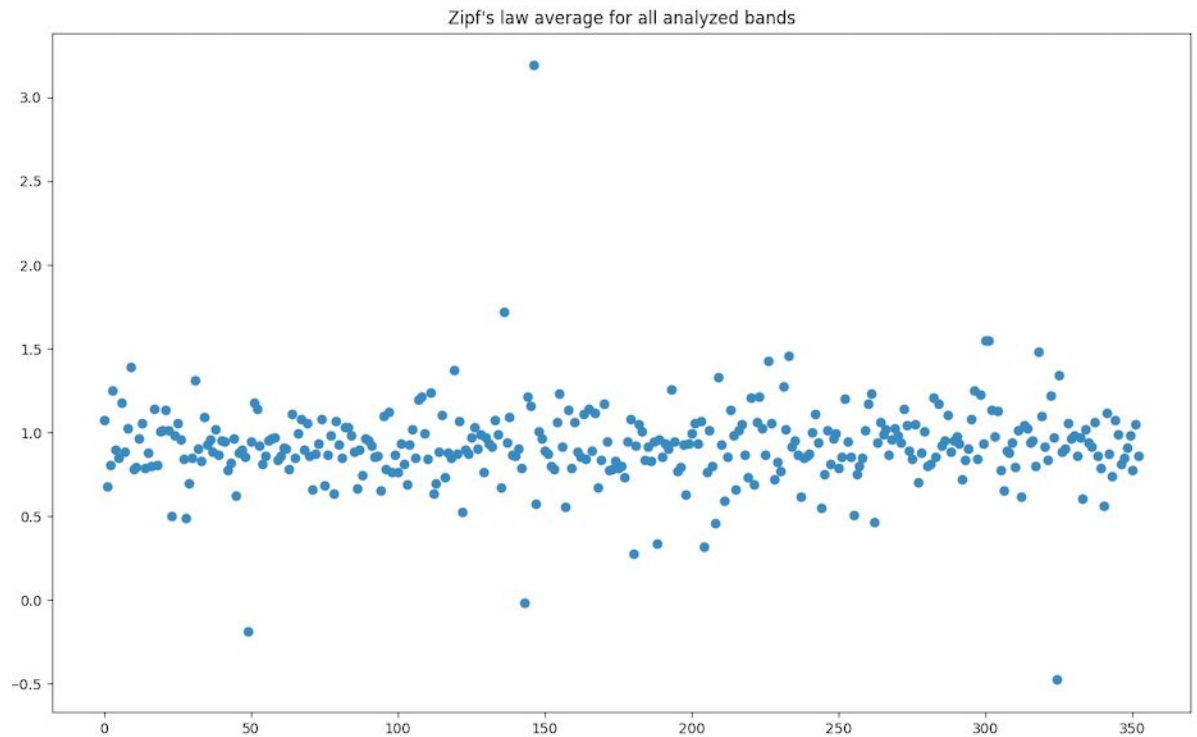
Jednakże, niektóre zespoły były bliskie ideału.



| Rank | Word        | Actual Freq | Zipf Frac | Zipf Freq | Actual Diff | Pct Diff |
|------|-------------|-------------|-----------|-----------|-------------|----------|
| 1    | ahha        | 42          | 1/1       | 42.00     | 0.00        | 100.00%  |
| 2    | voulezvous  | 21          | 1/2       | 21.00     | 0.00        | 100.00%  |
| 3    | know        | 15          | 1/3       | 14.00     | 1.00        | 107.14%  |
| 4    | leave       | 6           | 1/4       | 10.50     | -4.50       | 57.14%   |
| 5    | take        | 5           | 1/5       | 8.40      | -3.40       | 59.52%   |
| 6    | nothing     | 5           | 1/6       | 7.00      | -2.00       | 71.43%   |
| 7    | promise     | 5           | 1/7       | 6.00      | -1.00       | 83.33%   |
| 8    | regret      | 5           | 1/8       | 5.25      | -0.25       | 95.24%   |
| 9    | decision    | 5           | 1/9       | 4.67      | 0.33        | 107.14%  |
| 10   | mean        | 4           | 1/10      | 4.20      | -0.20       | 95.24%   |
| 11   | scene       | 3           | 1/11      | 3.82      | -0.82       | 78.57%   |
| 12   | back        | 3           | 1/12      | 3.50      | -0.50       | 85.71%   |
| 13   | question    | 3           | 1/13      | 3.23      | -0.23       | 92.86%   |
| 14   | cest        | 3           | 1/14      | 3.00      | 0.00        | 100.00%  |
| 15   | start       | 2           | 1/15      | 2.80      | -0.80       | 71.43%   |
| 16   | masters     | 2           | 1/16      | 2.62      | -0.62       | 76.19%   |
| 17   | still       | 2           | 1/17      | 2.47      | -0.47       | 80.95%   |
| 18   | people      | 1           | 1/18      | 2.33      | -1.33       | 42.86%   |
| 19   | everywhere  | 1           | 1/19      | 2.21      | -1.21       | 45.24%   |
| 20   | sense       | 1           | 1/20      | 2.10      | -1.10       | 47.62%   |
| 21   | expectation | 1           | 1/21      | 2.00      | -1.00       | 50.00%   |
| 22   | hang        | 1           | 1/22      | 1.91      | -0.91       | 52.38%   |
| 23   | giving      | 1           | 1/23      | 1.83      | -0.83       | 54.76%   |
| 24   | spark       | 1           | 1/24      | 1.75      | -0.75       | 57.14%   |
| 25   | across      | 1           | 1/25      | 1.68      | -0.68       | 59.52%   |
| 26   | room        | 1           | 1/26      | 1.62      | -0.62       | 61.90%   |
| 27   | glow        | 1           | 1/27      | 1.56      | -0.56       | 64.29%   |
| 28   | dark        | 1           | 1/28      | 1.50      | -0.50       | 66.67%   |
| 29   | think       | 1           | 1/29      | 1.45      | -0.45       | 69.05%   |
| 30   | girl        | 1           | 1/30      | 1.40      | -0.40       | 71.43%   |
| 31   | business    | 1           | 1/31      | 1.35      | -0.35       | 73.81%   |
| 32   | offer       | 1           | 1/32      | 1.31      | -0.31       | 76.19%   |
| 33   | drink       | 1           | 1/33      | 1.27      | -0.27       | 78.57%   |
| 34   | looking     | 1           | 1/34      | 1.24      | -0.24       | 80.95%   |
| 35   | mighty      | 1           | 1/35      | 1.20      | -0.20       | 83.33%   |
| 36   | proud       | 1           | 1/36      | 1.17      | -0.17       | 85.71%   |
| 37   | table       | 1           | 1/37      | 1.14      | -0.14       | 88.10%   |
| 38   | push        | 1           | 1/38      | 1.11      | -0.11       | 90.48%   |
| 39   | crowd       | 1           | 1/39      | 1.08      | -0.08       | 92.86%   |
| 40   | really      | 1           | 1/40      | 1.05      | -0.05       | 95.24%   |
| 41   | glad        | 1           | 1/41      | 1.02      | -0.02       | 97.62%   |
| 42   | come        | 1           | 1/42      | 1.00      | 0.00        | 100.00%  |
| 43   | rule        | 1           | 1/43      | 0.98      | 0.02        | 102.38%  |
| 44   | game        | 1           | 1/44      | 0.95      | 0.05        | 104.76%  |
| 45   | master      | 1           | 1/45      | 0.93      | 0.07        | 107.14%  |

### *Prawo Zipfa dla zespołu ABBA*

Aby zwizualizować średnie wyniki analizy zgodności z prawem Zipf'a, napisano metody przetwarzające otrzymane wyniki i uśredniające je. Dla każdego zespołu analiza dokonana była osobno na każdej piosence, której uśrednione wyniki zostały zaprezentowane na poniższym wykresie. Wykres wykonano przy pomocy pakietu matplotlib.



*Uśrednione wyniki analizy zgodności z prawem Zipf'a dla wszystkich zespołów*

```
def get_average_zipf_per_artist(zipfs_dict):
    diff_list = []
    for key, value in zipfs_dict.items():
        artist = key
        zipfs_table = value
        zipfs_average_values = []
        av = []
        for row_list in zipfs_table:
            diff_row_list = []
            for dict_row in row_list:
                diff = dict_row["difference_actual"]
                diff_row_list.append(diff)
            diff_row = sum(diff_row_list) / len(diff_row_list)
            av.append(diff_row)
        average_difference = sum(av) / len(av)
        zipfs_average_values.append(average_difference)
        overall_average = sum(zipfs_average_values) / len(zipfs_average_values)
        diff_info = (artist, overall_average)
        diff_list.append(diff_info)
    return diff_list

def create_zip_table_for_dict(dict):
    zip_dict = {}
    for key, value in dict.items():
        zip_table_list = []
        artist = key
        songs = value
        for song in songs:
            song = [w.lower() for w in song]
            song = [w for w in song if len(w) > 3]
            song = " ".join(song)
            zipfs_table = generate_zipf_table(song, 50)
            zip_table_list.append(zipfs_table)
        zip_dict[artist] = zip_table_list
    return zip_dict
```

*Metody uśredniające wyniki analizy prawa Zipf'a*

## Podsumowanie

Analiza przebiegła dobrze, jednakże wystąpiła przeszkoda w postaci konieczności zwiększenia dostępności do heap (sterty) dla programu PyCharm. Można się domyślać, że było to spowodowane znaczną liczbą iteracji: po zespole -> po każdej piosence danego zespołu -> po każdym słowie każdej piosenki danego zespołu. Z tego samego powodu, aby nie "męczyć" komputera, po każdej ważniejszej operacji, jej wyniki były zapisywane do pliku pickle .pkl, czyli serializowane do pliku. Kiedy zaistniała potrzeba ich ponownego wykorzystania, obiekty były deserializowane do zmiennych lokalnych. Wymienione we wstępie pakiety bardzo usprawniły pracę.

Jeśli chodzi o sam temat, uważam, że lepszym materiałem do analizy testu są długie i przemyślane teksty, na przykład książki, wiersze, artykuły. Piosenki często bywają chaotyczne, słowa dobierane są pod rym. Podczas analizy tematów często nie można było znaleźć tematu przewodniego piosenki, ponieważ poszczególne słowa bardzo od siebie odbiegały lub były mało znaczące. Był to jednak ciekawy projekt, dzięki któremu zapoznałam się z nowymi dla mnie narzędziami i metodami analitycznymi.