Project: Song recognition

Anna Derevianko, Ivan Yuzvyshyn, Maryam Zahra

June 14, 2018

1 Why this data ?

```
<li> Can't remember a familiar song in the club or the restaurant. But the sentimentality of the
<li>You have a phone with music recognition software installed so the software tell you the name
<li>Wanted to add something similar to software recognition in our application so we changed dat
```

2 Description of Data Set

Our data set includes

```
<li> Song Tittle</li>
<li> Song Author</li>
<li> Song Genre </li>
<li> Song Fingerprints </li>
```

3 Gathering all data set

We started our way with datasets, so we put songs in folder and started converting each to byte array

From songs name we have author, tittle, genre and fingerprint

Converting each song into bytes array by using code below

```
In [ ]: fs, data = wavfile.read(filename) # load the data
```

Plotting the data of one of out songs

```
In [ ]: # this is a two channel soundtrack, I get the first track
        a = data.T[0]
        plt.plot(a,'r')
        plt.show()
```

representation of one song in byte format

4 The Discrete Fourier Transform

So we need to find a way to convert our signal from the time domain to the frequency domain. Here we call on the Discrete Fourier Transform (DFT) for help. The DFT is a mathematical methodology for performing Fourier analysis on a discrete (sampled) signal. It converts a finite list of equally spaced samples of a function into the list of coefficients of a finite combination of complex sinusoids, ordered by their frequencies, by considering if those sinusoids had been sampled at the same rate.

One of the most popular numerical algorithms for the calculation of DFT is the Fast Fourier transform (FFT). By far the most commonly used variation of FFT is the Cooley–Tukey algorithm. This is a divide-and-conquer algorithm that recursively divides a DFT into many smaller DFTs. Whereas evaluating a DFT directly requires O(n2) operations, with a Cooley-Tukey FFT the same result is computed in O(n log n) operations.

So the song after the FFT Analysis

It's not hard to find an appropriate library for FFT. Here are few of them: Python – NumPy

```
In [ ]: # this is 8-bit track, b is now normalized on [-1,1)
        b=[(ele/2**8.)*2-1 for ele in a]
        # calculate fourier transform (complex numbers list)
        c = fft(b)
        # you only need half of the fft list (real signal symmetry)
        d = int(len(c)/2)
        plt.plot(abs(c[:(d-1)]),'r')
        plt.show()
```

in frequency domain our song looks like this

Analyzing a signal in the frequency domain simplifies many things immensely. It is more convenient in the world of digital signal processing because the engineer can study the spectrum (the representation of the signal in the frequency domain) and determine which frequencies are present, and which are missing. After that, one can do filtering, increase or decrease some frequencies, or just recognize the exact tone from the given frequencies.

One unfortunate side effect of FFT is that we lose a great deal of information about timing. (Although theoretically this can be avoided, the performance overheads are enormous.) For a three-minute song, we see all the frequencies and their magnitudes, but we don't have a clue when in the song they appeared. But this is the key information that makes the song what it is! Somehow we need to at know what point of time each frequency appeared.

So instead of analyzing the entire frequency range at once, we can choose several smaller intervals, chosen based on the common frequencies of important musical components, and analyze each separately. For example, we might use the intervalslike this 30 Hz - 40 Hz, 40 Hz - 80 Hz and 80 Hz - 120 Hz for the low tones (covering bass guitar, for example), and 120 Hz - 180 Hz and 180 Hz - 300 Hz for the middle and higher tones (covering vocals and most other instruments).

```
In [ ]: def get_index(freq):

            RANGE = [40, 80, 120, 180, 300]

            i = 0
            while ( RANGE[i] < freq ):
```

```
            i = i + 1

        return i
```

Below is function which gose through all song bytes spitting it for small invervals and on each runs Fourier Transform

```
In [ ]: def fourier_transform(data):

            a = data.T[0]

            total_size = len(a)
            chunk_size = 4096;

            sampled_chunk_size = int(total_size/chunk_size);
            result = [];
            for j in range(0, sampled_chunk_size):
                   complex_array = [];

                   for i in range(0, chunk_size):
                        complex_array.append(complex(a[(j*chunk_size)+i], 0))
                   result.append(fft(complex_array))

            return result
```

After getting result from prev function we go through all intervals and finding max magetude and frequescy for each range i.e [40-80] than [80-120] and so on....

```
In [ ]: def get_magnetude(result):
            high_scores = []
            freq_score = []
            for t in range(0, len(result)):
                max = [0,0,0,0,0]
                freq_max = [0,0,0,0,0]
                for freq in range(40,300):
                    mag = math.log(abs(result[t][freq]) + 1)

                    index = get_index(freq)

                    if (mag > max[index]):
                        max[index] = mag
                        freq_max[index] = freq

                high_scores.append(max)
                freq_score.append(hash(freq_max))

            return high_scores, freq_score
```

This function converts our chunk (array of 5 elements to an hashnumber) we are not using last element w.r.t. faster calculations

```
In [ ]: def hash(freq):
            FUZ_FACTOR = 2;
            p0 = freq[0]
            p1 = freq[1]
            p2 = freq[2]
            p3 = freq[3]
            return  (p3-(p3%FUZ_FACTOR)) * 100000000 + (p2-(p2%FUZ_FACTOR)) * 100000  + (p1-(p1%
```

That is our main functoin, which goes through all songs in folder and doing algorithm which was described above

```
In [ ]: def dm_run():

                path = os.path.dirname(os.path.abspath(__file__)) + '\\music\\' + '*.wav'

                #in_file = open("Come A Little Bit Closer  -  Jay  The Americans.wav.txt", "rb")
                #data = in_file.read() # if you only wanted to read 512 bytes, do .read(512)
                #in_file.close()

                end_data = [];
                end_data_author = []
                end_data_title = []
                end_data_style = []

                counter = 0;
                for filename in glob.glob(path):

                    try:
                         print("Uplodaing song number {0}".format(counter))

                         name = os.path.basename(filename).split('.')[0]
                         print('Magic with file {0} started'.format(name))

                         fs, data = wavfile.read(filename) # load the data

                         author, tittle, style = nm_run(name)

                         result = fourier_transform(data)

                         high_scores, freq_score = get_magnetude(result)

                         #insert(tittle, author)
                         print(name)
                         print(len(freq_score))
                         print(freq_score)

                         #plt.plot( high_scores, freq_score ,'ro')
                         #plt.show()
```

```
                #plt.plot(freq_score, 'ro')
                #plt.show()


                end_data.append(freq_score)
                end_data_author.append(author)
                end_data_title.append(tittle)
                end_data_style.append(style)
                counter = counter + 1
        except IOError as e:
                print ("I/O error({0}): {1}".format(e.errno, e.strerror))
        except ValueError:
            print ("Could not convert data to an integer.")
        except:
            print ("Unexpected error:", sys.exc_info()[0])

    print('uploading started')


    my_df_author = pd.DataFrame(end_data_author)
    my_df_author.to_csv('data_authors.csv', index=False, header=False)

    my_df_tittle = pd.DataFrame(end_data_title)
    my_df_tittle.to_csv('data_tittles.csv', index=False, header=False)

    my_df_style = pd.DataFrame(end_data_style)
    my_df_style.to_csv('data_styles.csv', index=False, header=False)

    my_df = pd.DataFrame(end_data)
    my_df.to_csv('data.csv', index=False, header=False)

    print('uploading ended')
```

as output we got 4 CSV files with hashes, tittles, authors, and styles of our song

5 Now lets start doning ANALISYS

```
In [4]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import matplotlib.cm as cm
        import csv
        import random
        import math
        import operator
        from sklearn.preprocessing import StandardScaler
```

```
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_samples, silhouette_score
```

In [5]:
```
dfname = pd.read_csv('data_tittles.csv', sep=',')
dfname_set = pd.read_csv('data_tittles.csv', sep=',', header=None)
dfname_set.columns = ["Title"]
```

In [6]:
```
dfstyle = pd.read_csv('data_styles.csv', sep=',')
dfgenre_set = pd.read_csv('data_styles.csv', sep=',', header=None)
dfgenre_set.columns = ['Genre']
dfgenre_set['Genre'] = dfgenre_set['Genre'].str.strip()
```

In [7]:
```
dfauthor = pd.read_csv('data_authors.csv', sep=',')
dfauthor_set = pd.read_csv('data_authors.csv', sep=',', header=None)
dfauthor_set.columns = ["Author"]
```

In [8]:
```
dfhashes = pd.read_csv('data.csv', sep=',')
dfhashes_set = pd.read_csv('data.csv', sep=',', header=None)
```

In [9]:
```
df = pd.concat([dfhashes, dfstyle], axis=1, join='inner')
df_full = pd.concat([dfhashes_set, dfauthor_set, dfname_set, dfgenre_set] , axis=1)
```

6 Data cleaning

Each song have different length and frequencies, so cleaning data is important.

If length of one song is shorter than the other we are adding the zeros frequency in the end so that the length of songs are same and adding zeros frequency means we are adding the silence.

In [160]:
```
df_full = df_full.fillna(0)
df_full
```

Out[160]:

	0	1	2	3	4 \
0	0	0	0	0	0
1	0	0	14810205840	0	14209405040
2	0	12008004040	13211004040	13208007440	12408204040
3	0	0	12609404840	17208007240	12608005440
4	0	0	0	0	0
5	12208207640	12410404240	17809005640	18010404440	15410406040
6	12009805640	16410805440	13010805440	13010804240	13010804240
7	0	0	0	0	0
8	16810604040	13411404240	0	0	0
9	0	0	0	0	12808405440
10	15809604440	14410004640	14608204440	14808805840	16409806440
11	0	0	17008204440	12809205040	16211804640
12	0	0	15009807840	13009004640	15610606240
13	0	0	12008404040	14410406640	13608607440
14	0	0	0	0	0

15	0	0	0	13008404040	14009204840
16	0	0	0	0	0
17	0	0	0	12810604040	12209806040
18	0	13409605040	12210205040	14408204640	15009605840
19	0	0	0	0	0
20	12008005240	15010805640	15809204640	13010004640	16610804640
21	0	0	0	0	12210807240
22	12608204440	12608804640	12808206440	13408407240	12409406440
23	0	15011805040	15009205040	16408005440	13608205440
24	0	0	14210004440	12210205040	12210205040
25	0	0	0	0	12809805040
26	0	0	0	0	0
27	13809204640	13809204640	13811404640	13809204640	13609204640
28	14011208040	12411604840	12010004640	12208804240	15809404240
29	13809604640	12008604640	13008204440	14409204640	13411604040
..	...	...	...	...	...
41	0	0	0	12410005040	14608207240
42	0	0	0	0	13608204240
43	0	0	0	0	16011004240
44	0	0	0	18012005440	13609806040
45	0	0	0	12212007040	15009204240
46	0	0	0	0	14011804240
47	0	0	0	14408606840	12208405240
48	0	0	0	0	16811007240
49	0	0	0	13809004640	13809204440
50	0	0	0	0	13208205840
51	0	0	0	0	13208204640
52	0	0	0	12008004040	14209406240
53	0	0	0	0	15811404240
54	0	0	0	18008004240	15408407040
55	0	0	0	12008004040	14609604040
56	13609205440	15409804640	14812004640	14810404240	12211005440
57	12408004040	16609004440	17008604640	17409204440	17209604640
58	14411207240	12008007240	12609606040	14411806640	12008806840
59	14009805040	14210607240	16409204040	14411604040	14408404640
60	13009404840	12410204640	12009404640	12409204640	14809204240
61	0	14209605840	15411605840	15409007640	14211605240
62	0	16210405840	17009406240	13809206240	17211404640
63	0	0	0	17608007240	13608204240
64	13008605040	13408605440	12208404640	13608406240	12208205040
65	0	0	0	0	0
66	15811404440	12809404640	15211404640	13011004640	13208004240
67	0	0	0	0	0
68	0	0	0	0	0
69	0	12209204640	17609405240	16809005040	17209804640
70	0	0	0	0	0

5 6 7 8 9 ... \

0	14409608040	14611607240	15011607440	14411607640	12211607840	...
1	12009404240	12409404240	16209204840	15409404240	17612004240	...
2	17608204040	17608004040	16208004040	12209604040	12208004040	...
3	12009005440	15209005440	12210806040	12210805440	12209005440	...
4	0	0	0	12008004040	13008004040	...
5	14409004640	13408007440	13609006040	13608204640	13408806040	...
6	13008604240	13011004240	13010804240	13010804240	13010804240	...
7	0	0	0	0	0	...
8	12408004240	12408204240	12408204040	16608204840	12208204840	...
9	13009005440	13008605840	17408605840	13008605840	17408605840	...
10	15408204440	15409805240	14408204440	15411404440	15408205840	...
11	17611806240	12211008040	17811404440	13610406840	13810806840	...
12	15410807640	15410604640	15410604640	12209206040	12209206040	...
13	17808206040	13809406040	12408406040	13608806040	13409406040	...
14	0	0	0	0	13609004040	...
15	13810804840	13810804840	17609204840	17409604240	17409604040	...
16	0	14410207240	13210806440	13210806440	13210606440	...
17	12409204640	12808406240	12810606240	12408004240	14608004840	...
18	15611605240	15209004640	12810604240	17411805040	14409806440	...
19	0	0	0	0	0	...
20	12610604640	17011604640	16410204640	13408204640	12208404640	...
21	14808206440	13208204240	12408204040	13008204040	16608204040	...
22	17408604840	13808805040	16411005440	16408205440	16408205440	...
23	13608205440	13608205440	13608205440	13608205440	13608205440	...
24	12210404040	12211804040	12212004040	12210204040	12210404040	...
25	12809004240	12810404040	12811005640	17411204440	12210204040	...
26	0	0	16008606440	16409406040	12008204040	...
27	13810204640	13810205440	13610205440	13809205440	13809204640	...
28	13410404240	12209206040	12209005840	15811605840	14612005240	...
29	15610407440	15409204240	15608004240	12408006440	17608404240	...
..	...	...	...	...	...	...
41	15808007240	14611405440	15211406040	14411406240	12211405440	...
42	15008204040	15008204040	15008204040	16408204040	12208204040	...
43	12211005240	13409005040	15208604640	17611005240	12608604240	...
44	17609807240	15409206840	12411206840	13609607240	17808805440	...
45	14209005840	13409004640	13409004440	13409005240	12810604440	...
46	12611404040	13008206640	14411004240	14810606040	12408007040	...
47	12008606840	15412006840	12208206040	14208006040	12208006040	...
48	13811006240	13811006240	12011008040	13410608040	16611005440	...
49	13212004040	12210804440	12410804440	13810604240	16809204640	...
50	12208204640	13808206240	15210806240	12208207640	14408606040	...
51	12411005840	12010807240	12209804040	14611404840	13011404840	...
52	15409206840	14409206840	14609206240	13809206840	16009206840	...
53	15808604240	13008604240	15808604440	17208404240	17208604240	...
54	15608407040	15608407040	15608407040	13408804240	13408804240	...
55	14608206240	12008207240	12208207240	12208004040	12208204840	...
56	12011005440	12212006240	12209806240	14609804840	12409804840	...
57	17009004640	16608205240	12211804440	12809606440	15211406440	...

58	12008407840	12208206640	12409207240	14209206840	14208006640	...
59	15009204040	13809205640	14609604840	12411804040	12008204240	...
60	14609007240	14608207240	14608207240	14609007240	14609007240	...
61	12811607640	18011605040	15411605240	15411605040	15411605040	...
62	15809007040	15809805440	15410605840	12610805640	12810407840	...
63	17211605040	16610606040	16212004440	12411806040	13808804040	...
64	14008805240	12208205840	13208004840	12408404240	12609804840	...
65	0	0	0	0	0	...
66	13608006040	13209004840	16010407440	15008607240	12811405640	...
67	0	0	0	14411404640	15010804240	...
68	0	0	0	0	0	...
69	16808404240	16408204240	16208204040	14009404640	12209004440	...
70	0	0	0	14209405640	13008406440	...

	5208	5209	5210	5211	5212	5213	5214	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
11	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
12	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
14	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
15	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
18	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
21	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
22	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
23	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
24	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
26	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
27	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
28	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
29	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
..	...	...	...	...	...	...	...	
41	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
42	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

43	0.0	0.0	0.0	0.0	0.0	0.0	0.0
44	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45	0.0	0.0	0.0	0.0	0.0	0.0	0.0
46	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47	0.0	0.0	0.0	0.0	0.0	0.0	0.0
48	0.0	0.0	0.0	0.0	0.0	0.0	0.0
49	0.0	0.0	0.0	0.0	0.0	0.0	0.0
50	0.0	0.0	0.0	0.0	0.0	0.0	0.0
51	0.0	0.0	0.0	0.0	0.0	0.0	0.0
52	0.0	0.0	0.0	0.0	0.0	0.0	0.0
53	0.0	0.0	0.0	0.0	0.0	0.0	0.0
54	0.0	0.0	0.0	0.0	0.0	0.0	0.0
55	0.0	0.0	0.0	0.0	0.0	0.0	0.0
56	0.0	0.0	0.0	0.0	0.0	0.0	0.0
57	0.0	0.0	0.0	0.0	0.0	0.0	0.0
58	0.0	0.0	0.0	0.0	0.0	0.0	0.0
59	0.0	0.0	0.0	0.0	0.0	0.0	0.0
60	0.0	0.0	0.0	0.0	0.0	0.0	0.0
61	0.0	0.0	0.0	0.0	0.0	0.0	0.0
62	0.0	0.0	0.0	0.0	0.0	0.0	0.0
63	0.0	0.0	0.0	0.0	0.0	0.0	0.0
64	0.0	0.0	0.0	0.0	0.0	0.0	0.0
65	0.0	0.0	0.0	0.0	0.0	0.0	0.0
66	0.0	0.0	0.0	0.0	0.0	0.0	0.0
67	0.0	0.0	0.0	0.0	0.0	0.0	0.0
68	0.0	0.0	0.0	0.0	0.0	0.0	0.0
69	0.0	0.0	0.0	0.0	0.0	0.0	0.0
70	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	Author	Title \
0	Anthony Gonzalez Gael García Bernal	Un Poco Loco
1	Ben E King	Stand By Me
2	BOB DYLAN	Mr Tambourine Man
3	Calvin Harris	Feels
4	Camila Cabello	Havana
5	Channa Mereya	Arjit singh
6	Christina Perri	A Thousand Years
7	Coldplay	Fix You
8	Coldplay	The Scientist
9	Daniel Powter	Bad Day
10	Deicide	Homage for Satan
11	Eagles	Hotel California
12	Ed Sheeran	Photograph
13	Ed Sheeran	Shape of You
14	Elvis Presley	Cant Help Falling In Love
15	Frank Sinatra	Killing me softly
16	Frank Sinatra	Strangers In the Night
17	Frank Sinatra	The Way You Look Tonight

```
18                                       Grieg  In the Hall of the Mountain King
19                          Jay  The Americans        Come A Little Bit Closer 
20                              Joseph LoDuca             Ashs Dream piano cover
21                             Kelly Clarkson                       Silent Night 
22                              Kim Jang Woo                             Destiny
23                                      Laura                       Say Something
24                              Laura pausini                    its not goodbye
25                               Led Zeppelin        Stairway To Heaven Lyrics
26                              Lionel Richie                        Endless Love
27                                 Luis Fonsi                           Despacito
28                                Nathan Lane                       Hakuna Matata
29                                      Oasis                          Wonderwall
..                                        ...                                ...
41                                  Scorpions        He's a Woman, She's a Man
42                                  Scorpions                             Holiday
43                                  Scorpions                       I'm Goin' Mad
44                                  Scorpions                           In Trance
45                                  Scorpions           Is There Anybody There
46                                  Scorpions                       Love Is Blind
47                                  Scorpions        Loving You Sunday Morning
48                                  Scorpions                        Make It Real
49                                  Scorpions                     No one like you
50                                  Scorpions           Passion Rules the Game
51                                  Scorpions                      Rhythm Of Love
52                                  Scorpions        Rock You Like a Hurricane
53                                  Scorpions                   Send Me an Angel
54                                  Scorpions                   Still Loving You
55                                  Scorpions                     Wind of Change
56                               SCOTT JOPLIN                   The Entertainer
57                                    Shakira                      Try Everything
58                                    Shakira                           Waka Waka
59                                     SLAYER                          Repentless
60                                 statkowski                                 idk
61                          System of a Down                               BYOB 
62                                  Tom Odell                               Healv
63                              Tracy Chapman                           Fast car 
64                          twenty one pilots                           Heathens 
65                          twenty one pilots                                Ride
66                                   unnnamed                    low_town_groove
67                                   Westlife        I Wanna Grow Old With You
68                                  Zara Zara                          Rahul Jain
69
70

      Genre
0       POP
1       POP
2   CLASSIC
```

```
3         POP
4         POP
5     UNKNOWN
6         POP
7         POP
8        ROCK
9         POP
10      Metal
11    CLASSIC
12        POP
13        POP
14        POP
15    CLASSIC
16    CLASSIC
17    CLASSIC
18    CLASSIC
19        POP
20    CLASSIC
21        POP
22    CLASSIC
23        POP
24        POP
25       ROCK
26        POP
27        POP
28        POP
29        POP
..        ...
41       ROCK
42       ROCK
43       ROCK
44       ROCK
45       ROCK
46       ROCK
47       ROCK
48       ROCK
49       ROCK
50       ROCK
51       ROCK
52       ROCK
53       ROCK
54       ROCK
55       ROCK
56    CLASSIC
57        POP
58        POP
59      Metal
60    CLASSIC
```

```
61     Metal
62       POP
63       POP
64       POP
65       POP
66   UNKNOWN
67       POP
68       POP
69       POP
70       POP

[71 rows x 5218 columns]
```

Now let's check if dataset has been correctly cleaned looking for NaN values

In [161]:
```
df_full.isnull().any().any()
```

Out[161]:
```
False
```

answer is False, so no NaN values, import is correct

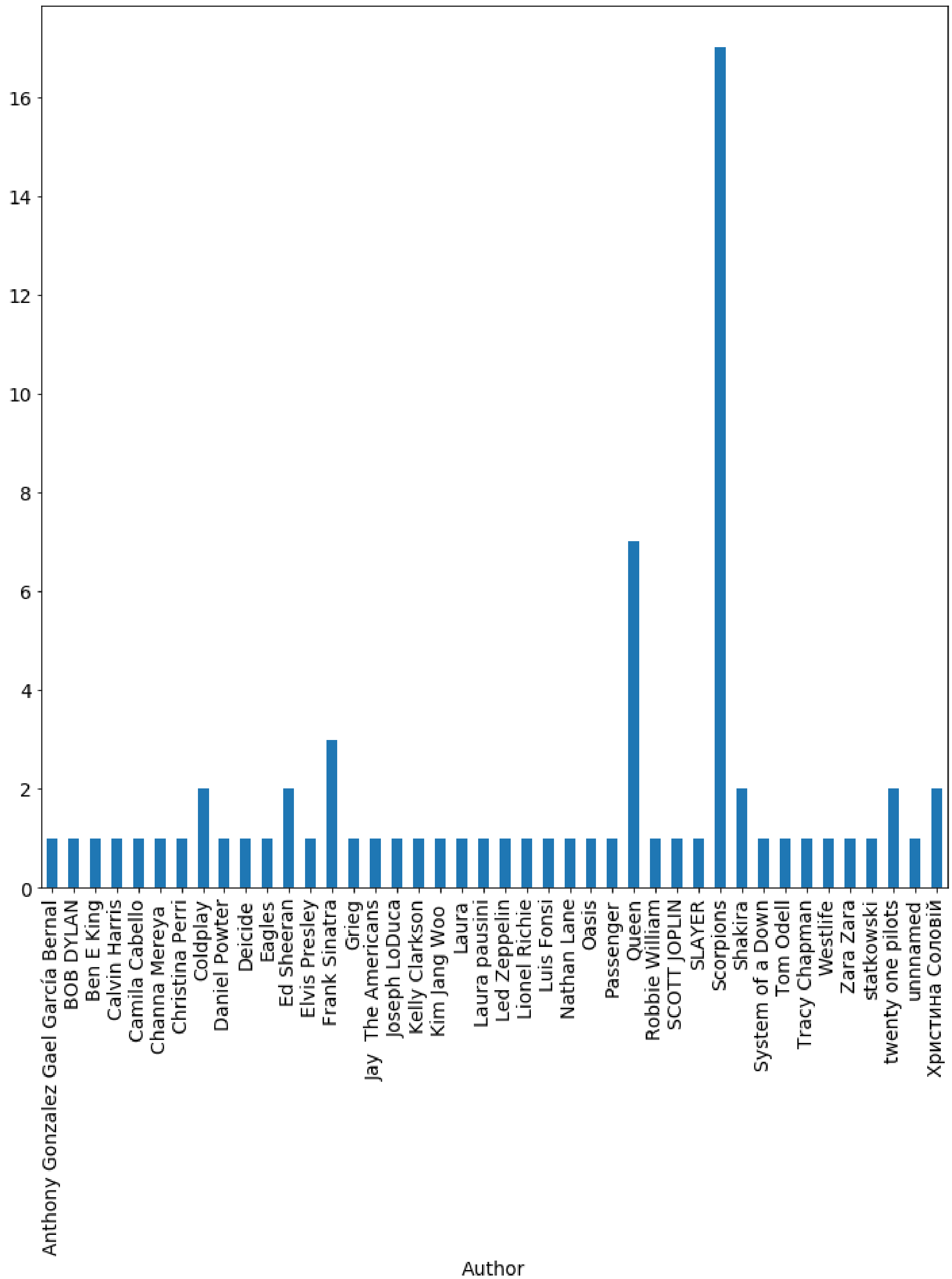
7 Exploratory analysis

In [162]:
```
#descriptive statistics using pandas method
df_full.describe()
```

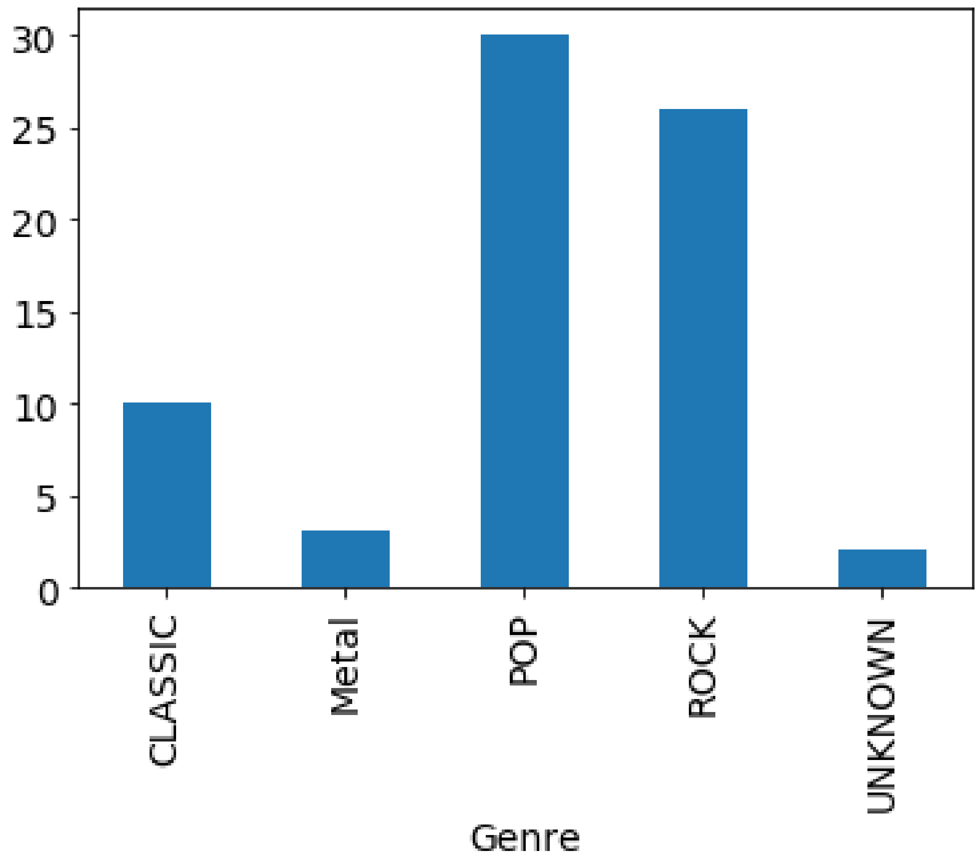
Out[162]:
```
                  0             1             2             3             4  \
count  7.100000e+01  7.100000e+01  7.100000e+01  7.100000e+01  7.100000e+01
mean   3.884340e+09  5.152875e+09  6.559326e+09  8.414028e+09  1.121040e+10
std    6.288815e+09  6.900000e+09  7.399656e+09  7.184579e+09  6.002823e+09
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  1.220831e+10
50%    0.000000e+00  0.000000e+00  0.000000e+00  1.240920e+10  1.360820e+10
75%    1.210901e+10  1.270910e+10  1.440910e+10  1.410871e+10  1.490920e+10
max    1.681060e+10  1.741121e+10  1.780901e+10  1.801201e+10  1.721140e+10

                  5             6             7             8             9  \
count  7.100000e+01  7.100000e+01  7.100000e+01  7.100000e+01  7.100000e+01
mean   1.178522e+10  1.174313e+10  1.197729e+10  1.259163e+10  1.264222e+10
std    5.849863e+09  5.536814e+09  5.393917e+09  4.760110e+09  4.558894e+09
min    0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
25%    1.210961e+10  1.221091e+10  1.221031e+10  1.241010e+10  1.220900e+10
50%    1.360821e+10  1.340901e+10  1.340900e+10  1.361041e+10  1.321061e+10
75%    1.550961e+10  1.480991e+10  1.531101e+10  1.470991e+10  1.481140e+10
max    1.780821e+10  1.801161e+10  1.781140e+10  1.761101e+10  1.780881e+10

       ...  5205  5206  5207  5208  5209  5210  5211  5212  5213  5214
count  ...  71.0  71.0  71.0  71.0  71.0  71.0  71.0  71.0  71.0  71.0
```

```
mean   ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
std    ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
min    ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
25%    ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
50%    ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
75%    ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0
max    ...   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

[8 rows x 5215 columns]
```

In [68]:
```
ganreGrouped = df_full.groupby(['Author'])['Author'].count()
ganreGrouped.plot(kind='bar',figsize=(12, 12))
plt.rcParams.update({'font.size': 14})
plt.show()
```




```
In [69]: ganreGrouped = df_full.groupby(['Genre'])['Genre'].count()
         ganreGrouped.plot(kind='bar');
         plt.show()
```



In [167]:
```
#Correlation matrix
correlations = df_full.corr().fillna(0)
correlations
```

Out[167]:

	0	1	2	3	4	5	6 \
0	1.000000	0.814685	0.605492	0.422165	0.249182	0.289033	0.276502
1	0.814685	1.000000	0.789197	0.570691	0.368388	0.380233	0.406699
2	0.605492	0.789197	1.000000	0.679394	0.511097	0.463069	0.458144
3	0.422165	0.570691	0.679394	1.000000	0.623773	0.619895	0.601659
4	0.249182	0.368388	0.511097	0.623773	1.000000	0.893085	0.846795
5	0.289033	0.380233	0.463069	0.619895	0.893085	1.000000	0.915614
6	0.276502	0.406699	0.458144	0.601659	0.846795	0.915614	1.000000
7	0.235219	0.314813	0.399739	0.484940	0.788260	0.850674	0.889062
8	0.245978	0.281842	0.280225	0.335167	0.635438	0.688354	0.716057
9	0.233581	0.252694	0.285000	0.388315	0.643369	0.676804	0.685940
10	0.161002	0.232076	0.282356	0.365704	0.646381	0.674183	0.703547
11	0.276984	0.312191	0.334137	0.369239	0.610426	0.645478	0.673143
12	0.201499	0.273350	0.308183	0.409374	0.666986	0.667904	0.710017
13	0.218844	0.297922	0.290151	0.396508	0.616778	0.681595	0.714548
14	0.157001	0.248655	0.281423	0.436113	0.681041	0.695152	0.736229

15	0.170072	0.204997	0.254663	0.386102	0.637517	0.647325	0.680885
16	0.171980	0.237659	0.295854	0.378897	0.582091	0.583088	0.628954
17	0.177781	0.201624	0.259843	0.345770	0.600241	0.628506	0.644008
18	0.224312	0.246936	0.317615	0.366361	0.600248	0.627558	0.630209
19	0.122869	0.202970	0.297034	0.374068	0.600063	0.649373	0.667165
20	0.129066	0.172624	0.210247	0.298252	0.543781	0.596133	0.592156
21	0.171497	0.181881	0.190521	0.335335	0.543412	0.584770	0.607834
22	0.146033	0.177525	0.172556	0.367912	0.535843	0.596940	0.613246
23	0.151521	0.197974	0.231918	0.361995	0.534504	0.530027	0.558638
24	0.149052	0.200069	0.218370	0.349936	0.577072	0.595238	0.618420
25	0.153037	0.221471	0.238663	0.389771	0.634734	0.650107	0.668444
26	0.185239	0.262645	0.253097	0.420284	0.634533	0.643515	0.669229
27	0.226972	0.213039	0.205215	0.371838	0.546266	0.579223	0.591918
28	0.198142	0.265183	0.285003	0.402062	0.643354	0.637112	0.677674
29	0.185403	0.229757	0.245749	0.321445	0.562786	0.589131	0.638649
...	...	...	...	...	...	...	...
5185	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5186	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5187	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5188	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5189	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5190	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5191	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5192	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5193	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5194	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5195	-0.074350	-0.089894	-0.106704	-0.140972	0.032073	0.021066	0.023203
5196	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5197	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5198	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5199	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5200	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5201	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5202	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5203	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5204	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5205	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5206	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5207	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5208	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5209	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5210	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5211	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5212	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5213	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
5214	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000

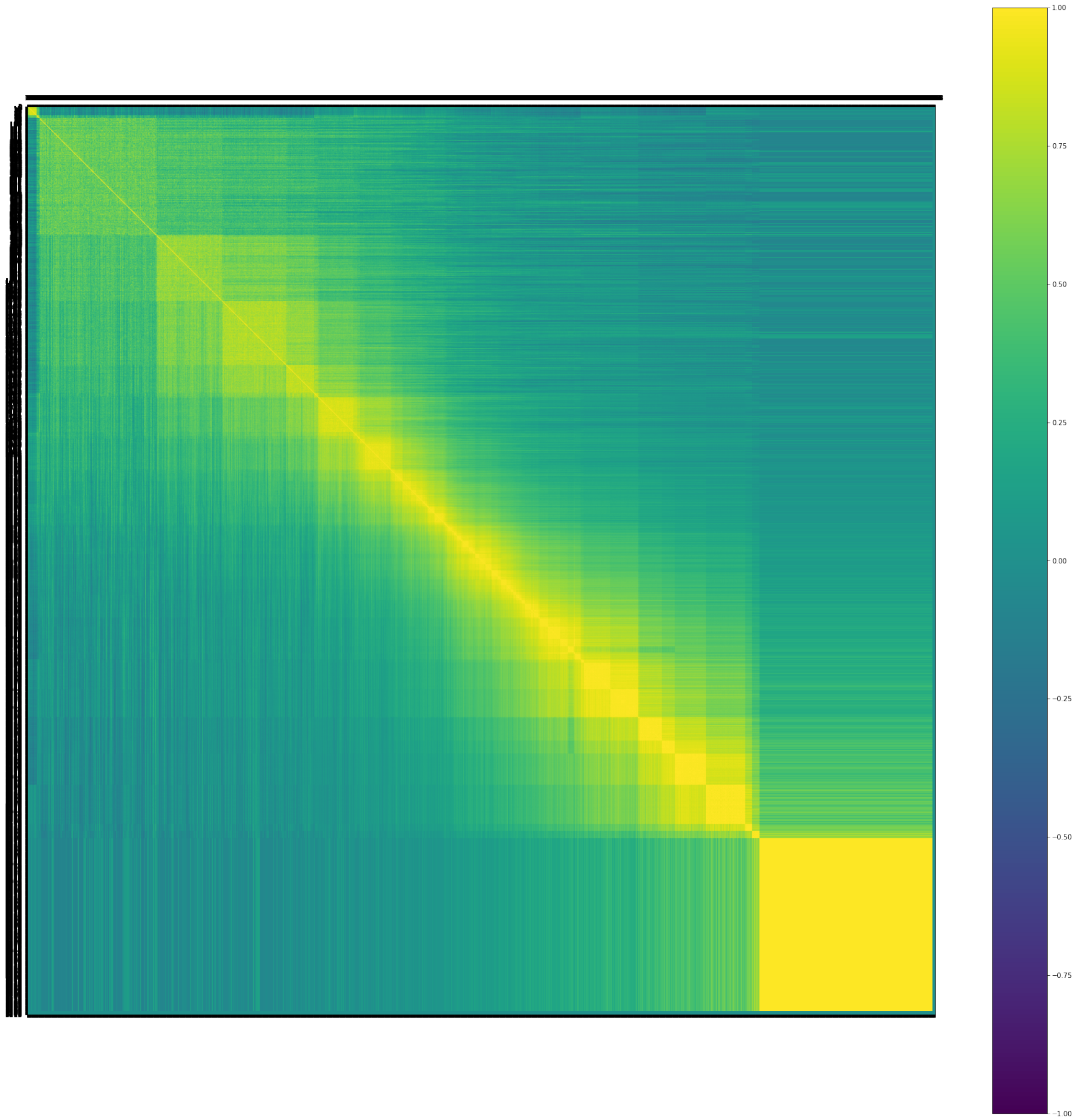
```
          7         8         9    ...  5205  5206  5207  5208  5209  5210  \
```

0	0.235219	0.245978	0.233581	...	0.0	0.0	0.0	0.0	0.0	0.0
1	0.314813	0.281842	0.252694	...	0.0	0.0	0.0	0.0	0.0	0.0
2	0.399739	0.280225	0.285000	...	0.0	0.0	0.0	0.0	0.0	0.0
3	0.484940	0.335167	0.388315	...	0.0	0.0	0.0	0.0	0.0	0.0
4	0.788260	0.635438	0.643369	...	0.0	0.0	0.0	0.0	0.0	0.0
5	0.850674	0.688354	0.676804	...	0.0	0.0	0.0	0.0	0.0	0.0
6	0.889062	0.716057	0.685940	...	0.0	0.0	0.0	0.0	0.0	0.0
7	1.000000	0.789485	0.717285	...	0.0	0.0	0.0	0.0	0.0	0.0
8	0.789485	1.000000	0.842740	...	0.0	0.0	0.0	0.0	0.0	0.0
9	0.717285	0.842740	1.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
10	0.747250	0.847340	0.918329	...	0.0	0.0	0.0	0.0	0.0	0.0
11	0.710788	0.827737	0.915509	...	0.0	0.0	0.0	0.0	0.0	0.0
12	0.742834	0.848544	0.916246	...	0.0	0.0	0.0	0.0	0.0	0.0
13	0.736904	0.847419	0.882909	...	0.0	0.0	0.0	0.0	0.0	0.0
14	0.733066	0.823568	0.907915	...	0.0	0.0	0.0	0.0	0.0	0.0
15	0.742380	0.826697	0.897684	...	0.0	0.0	0.0	0.0	0.0	0.0
16	0.665228	0.764517	0.829507	...	0.0	0.0	0.0	0.0	0.0	0.0
17	0.674648	0.784511	0.826378	...	0.0	0.0	0.0	0.0	0.0	0.0
18	0.667469	0.770631	0.838296	...	0.0	0.0	0.0	0.0	0.0	0.0
19	0.696685	0.757196	0.824098	...	0.0	0.0	0.0	0.0	0.0	0.0
20	0.639006	0.716703	0.788020	...	0.0	0.0	0.0	0.0	0.0	0.0
21	0.632875	0.736658	0.828758	...	0.0	0.0	0.0	0.0	0.0	0.0
22	0.657551	0.736770	0.826949	...	0.0	0.0	0.0	0.0	0.0	0.0
23	0.604091	0.722143	0.810515	...	0.0	0.0	0.0	0.0	0.0	0.0
24	0.652179	0.772988	0.811663	...	0.0	0.0	0.0	0.0	0.0	0.0
25	0.687799	0.776678	0.834503	...	0.0	0.0	0.0	0.0	0.0	0.0
26	0.673517	0.791789	0.846492	...	0.0	0.0	0.0	0.0	0.0	0.0
27	0.608890	0.740555	0.816373	...	0.0	0.0	0.0	0.0	0.0	0.0
28	0.686189	0.764869	0.816603	...	0.0	0.0	0.0	0.0	0.0	0.0
29	0.667396	0.781508	0.798225	...	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...
5185	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5186	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5187	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5188	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5189	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5190	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5191	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5192	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5193	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5194	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5195	0.018606	0.121877	-0.011407	...	0.0	0.0	0.0	0.0	0.0	0.0
5196	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
5197	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
5198	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
5199	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
5200	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0
5201	0.000000	0.000000	0.000000	...	0.0	0.0	0.0	0.0	0.0	0.0

```
5202  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5203  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5204  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5205  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5206  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5207  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5208  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5209  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5210  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5211  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5212  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5213  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0
5214  0.000000  0.000000  0.000000  ...    0.0    0.0    0.0    0.0    0.0    0.0

      5211  5212  5213  5214
0      0.0   0.0   0.0   0.0
1      0.0   0.0   0.0   0.0
2      0.0   0.0   0.0   0.0
3      0.0   0.0   0.0   0.0
4      0.0   0.0   0.0   0.0
5      0.0   0.0   0.0   0.0
6      0.0   0.0   0.0   0.0
7      0.0   0.0   0.0   0.0
8      0.0   0.0   0.0   0.0
9      0.0   0.0   0.0   0.0
10     0.0   0.0   0.0   0.0
11     0.0   0.0   0.0   0.0
12     0.0   0.0   0.0   0.0
13     0.0   0.0   0.0   0.0
14     0.0   0.0   0.0   0.0
15     0.0   0.0   0.0   0.0
16     0.0   0.0   0.0   0.0
17     0.0   0.0   0.0   0.0
18     0.0   0.0   0.0   0.0
19     0.0   0.0   0.0   0.0
20     0.0   0.0   0.0   0.0
21     0.0   0.0   0.0   0.0
22     0.0   0.0   0.0   0.0
23     0.0   0.0   0.0   0.0
24     0.0   0.0   0.0   0.0
25     0.0   0.0   0.0   0.0
26     0.0   0.0   0.0   0.0
27     0.0   0.0   0.0   0.0
28     0.0   0.0   0.0   0.0
29     0.0   0.0   0.0   0.0
...    ...   ...   ...   ...
5185   0.0   0.0   0.0   0.0
5186   0.0   0.0   0.0   0.0
```

```
5187   0.0   0.0   0.0   0.0
5188   0.0   0.0   0.0   0.0
5189   0.0   0.0   0.0   0.0
5190   0.0   0.0   0.0   0.0
5191   0.0   0.0   0.0   0.0
5192   0.0   0.0   0.0   0.0
5193   0.0   0.0   0.0   0.0
5194   0.0   0.0   0.0   0.0
5195   0.0   0.0   0.0   0.0
5196   0.0   0.0   0.0   0.0
5197   0.0   0.0   0.0   0.0
5198   0.0   0.0   0.0   0.0
5199   0.0   0.0   0.0   0.0
5200   0.0   0.0   0.0   0.0
5201   0.0   0.0   0.0   0.0
5202   0.0   0.0   0.0   0.0
5203   0.0   0.0   0.0   0.0
5204   0.0   0.0   0.0   0.0
5205   0.0   0.0   0.0   0.0
5206   0.0   0.0   0.0   0.0
5207   0.0   0.0   0.0   0.0
5208   0.0   0.0   0.0   0.0
5209   0.0   0.0   0.0   0.0
5210   0.0   0.0   0.0   0.0
5211   0.0   0.0   0.0   0.0
5212   0.0   0.0   0.0   0.0
5213   0.0   0.0   0.0   0.0
5214   0.0   0.0   0.0   0.0

[5215 rows x 5215 columns]
```

In [127]:
```
import matplotlib.pyplot as plt
import numpy as np

# plot correlation matrix

names = list(correlations.columns)
fig = plt.figure(figsize=[30,30])
ax = fig.add_subplot(111)
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = np.arange(0,5215,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
plt.show()
```



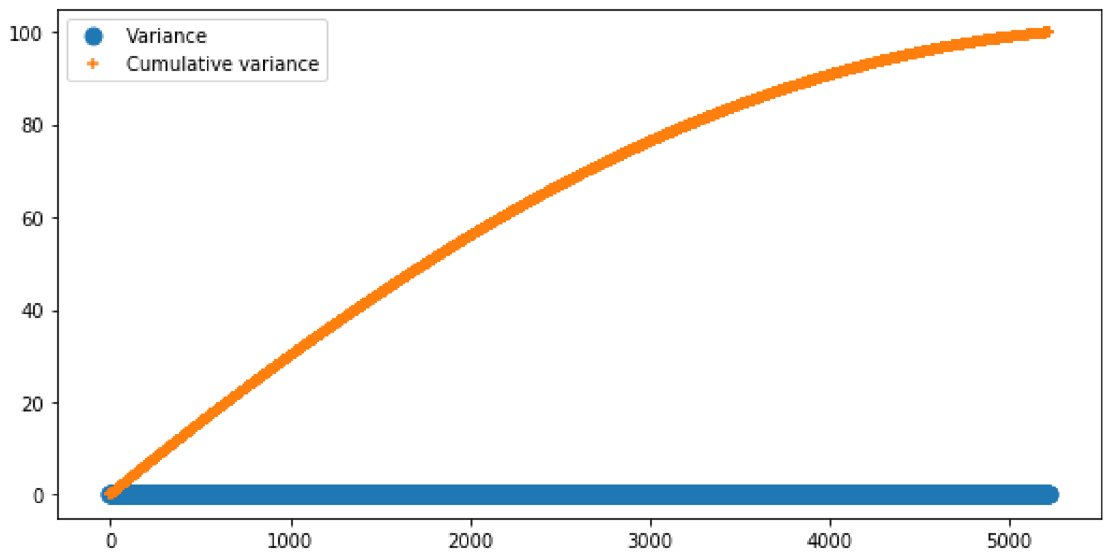
8 Unsupervised learning: clustering

In [172]:
```
dfhashes=dfhashes.fillna(0)
dataframe_std = pd.DataFrame(StandardScaler().fit_transform(dfhashes))
cov_std = dataframe_std.corr()
cov_std=cov_std.fillna(0)
```

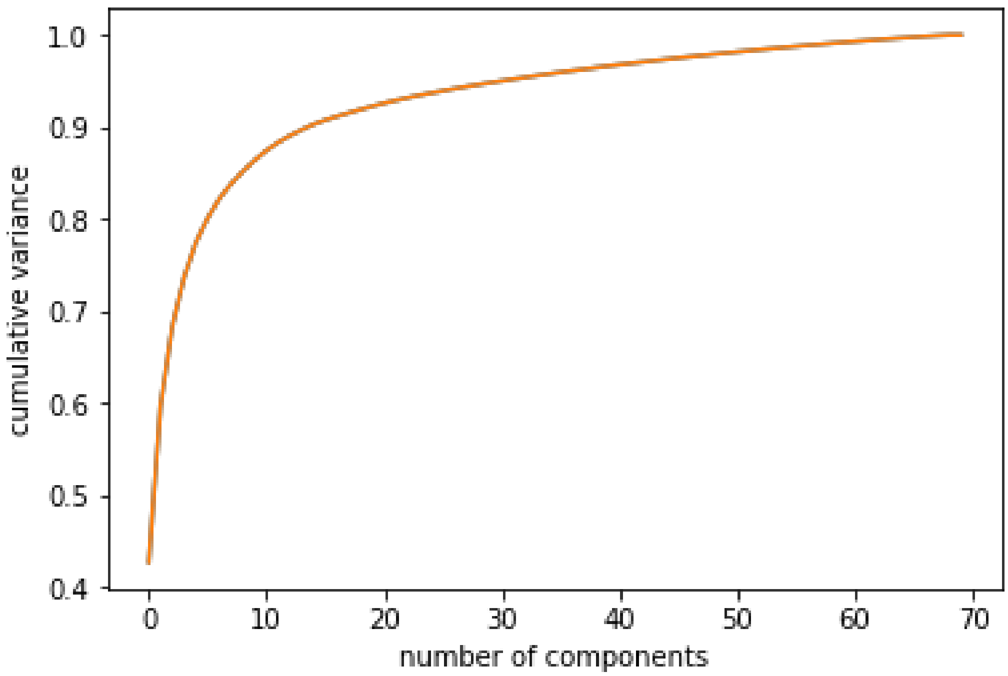
In [174]:
```
#We need to take components with the highest value to keep the information on the proj
#Here we're sure that we need the first and the second. For the rest we run the comput

eig_vals, eig_vect = np.linalg.eig(cov_std)
```

```
eig_pairs = [(np.abs(eig_vals[i]), eig_vect[:,i]) for i in range(len(eig_vals))]

sum_ev = sum(eig_vals)
pve = [(i / sum_ev)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_pve = np.cumsum(pve)

fig = plt.figure(figsize=[10,5])
plt.scatter([i for i in range(len(dataset_std.columns))], pve, s=80)
plt.scatter([i for i in range(len(dataset_std.columns))], cum_var_pve, marker='+')
plt.legend(['Variance', 'Cumulative variance'])
plt.show()
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\numpy\core\
  return array(a, dtype, copy=False, order=order, subok=True)
```




```
In [176]: pca = PCA().fit(dfhashes)
          plt.plot(np.cumsum(pca.explained_variance_ratio_))
          plt.xlabel('number of components')
          plt.ylabel('cumulative variance');
          plt.show()
```



In [18]:
```
dataframe_pca = PCA(n_components=2).fit_transform(dataframe_std)

dataframe_pca

#the coordinates of the points projected into the space
```

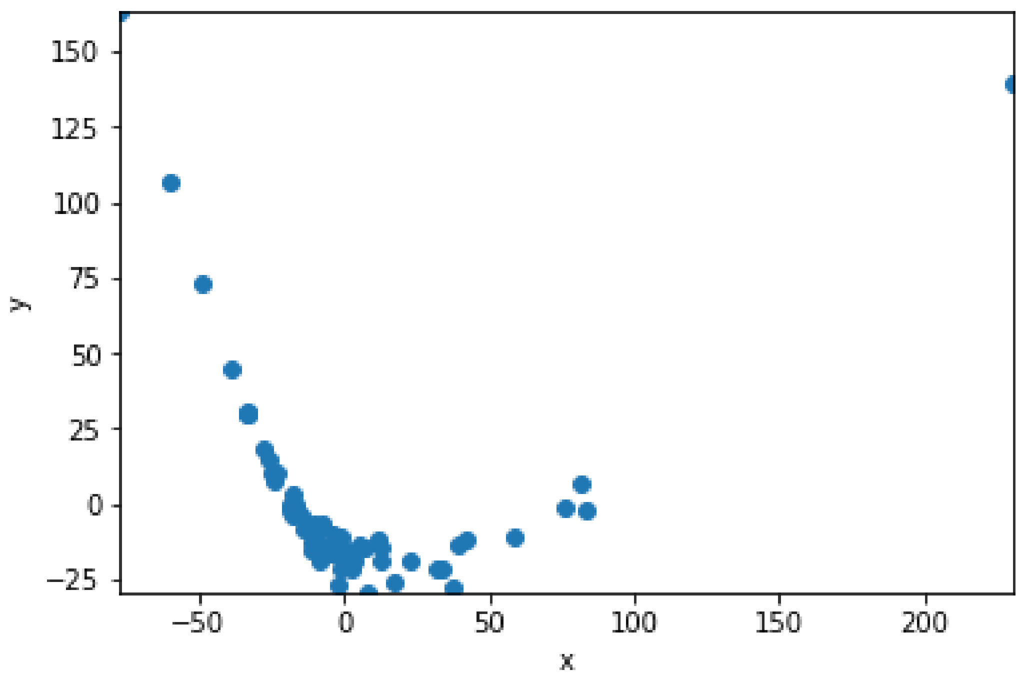
Out[18]:
```
array([[ -4.89505350e+01,   7.31403248e+01],
       [ -2.40300887e+01,   7.92593429e+00],
       [  3.19386155e+01,  -2.10378859e+01],
       [ -1.23756063e+01,  -8.52920377e+00],
       [ -1.50742218e+01,  -3.67588035e+00],
       [  4.23055466e+01,  -1.18122994e+01],
       [  7.05189270e+00,  -1.38940425e+01],
       [  1.69114038e+01,  -2.57993964e+01],
       [ -1.03502505e+00,  -1.07756891e+01],
       [ -1.07328595e+01,  -1.05041501e+01],
       [ -2.28636838e+00,  -2.63212729e+01],
       [  8.12439583e+01,   7.15134968e+00],
       [ -2.52756214e+00,  -1.26739222e+01],
       [ -4.94745585e+00,  -1.31003172e+01],
       [ -2.58084446e+01,   1.43935890e+01],
       [  3.49655231e+00,  -1.89707273e+01],
       [ -3.31266470e+01,   2.96480650e+01],
       [ -1.86180495e+01,  -4.13652533e-02],
       [ -3.33280048e+01,   3.07814623e+01],
```

```
[ -2.77487777e+01,   1.80630637e+01],
[ -5.98955352e+01,   1.06422118e+02],
[ -2.49082069e+01,   1.05147072e+01],
[  3.35273594e+01,  -2.11395403e+01],
[ -1.26642358e+01,  -6.48268697e+00],
[  3.04981981e+00,  -2.17661008e+01],
[  2.30629399e+02,   1.39519293e+02],
[ -3.50152028e+00,  -9.71560592e+00],
[  3.82480595e+00,  -1.56996571e+01],
[ -1.01869818e+01,  -1.06915287e+01],
[  5.23053382e+00,  -1.31675144e+01],
[ -1.29083156e+00,  -2.10520957e+01],
[  5.88547328e+01,  -1.08978455e+01],
[ -1.61804686e+01,  -2.50387155e+00],
[ -8.57871373e+00,  -1.85507171e+01],
[ -5.19060415e+00,  -1.64769445e+01],
[ -1.25033812e+00,  -1.33815487e+01],
[ -1.10678263e+01,  -1.23046728e+01],
[ -7.46038212e+00,  -6.33417354e+00],
[ -1.78845081e+01,  -1.21527026e+00],
[  1.20610016e+01,  -1.15520981e+01],
[ -9.88159421e+00,  -1.32076082e+01],
[ -1.87084163e+01,  -1.79411504e+00],
[  7.56650716e+01,  -7.39741251e-01],
[  1.28310719e+01,  -1.86565142e+01],
[  3.73297450e+01,  -2.80218075e+01],
[ -2.19077357e-01,  -2.01251761e+01],
[ -1.10086343e+01,  -7.13847877e+00],
[  3.92934153e+01,  -1.38619861e+01],
[ -8.90751818e+00,  -1.30324818e+01],
[ -1.02840946e+01,  -6.20405082e+00],
[ -9.48931552e+00,  -8.84129602e+00],
[ -1.15399795e+01,  -8.37408688e+00],
[ -1.77686361e+00,  -1.72763059e+01],
[  8.50958333e+00,  -2.91703050e+01],
[  8.37004618e+01,  -1.73952760e+00],
[  2.30944382e+01,  -1.89496708e+01],
[ -1.78716210e+01,   3.22853365e+00],
[ -1.79685677e+01,  -3.45841119e+00],
[ -1.42850407e+01,  -7.78284933e+00],
[ -1.07764100e+01,  -1.55830834e+01],
[ -3.89793469e+01,   4.49474668e+01],
[ -1.40937363e+00,  -1.79813433e+01],
[ -2.27889769e+01,   1.01908514e+01],
[  1.28554894e+01,  -1.47324997e+01],
[ -1.67646944e+01,  -3.18651793e-01],
[ -1.01338872e+01,  -1.32943949e+01],
[ -7.70647178e+01,   1.62907188e+02],
```

```
[ -4.78824418e+00,  -1.46714171e+01],
[ -3.33390077e+01,   2.95773498e+01],
[ -1.71791652e+01,  -2.23367373e+00],
[ -1.75905526e+01,  -1.15379516e+00]])
```

This show how our songs representation with respect to their special hashes, since we have big amout close to each outhers it mean that alot of song have same maximal frequensy

```
In [19]: plt.scatter(dataframe_pca[:,0],dataframe_pca[:,1])

         plt.xlabel('x')
         plt.ylabel('y')
         plt.xlim(min(dataframe_pca[:,0]),max(dataframe_pca[:,0]))
         plt.ylim(min(dataframe_pca[:,1]),max(dataframe_pca[:,1]))

         plt.show()
```

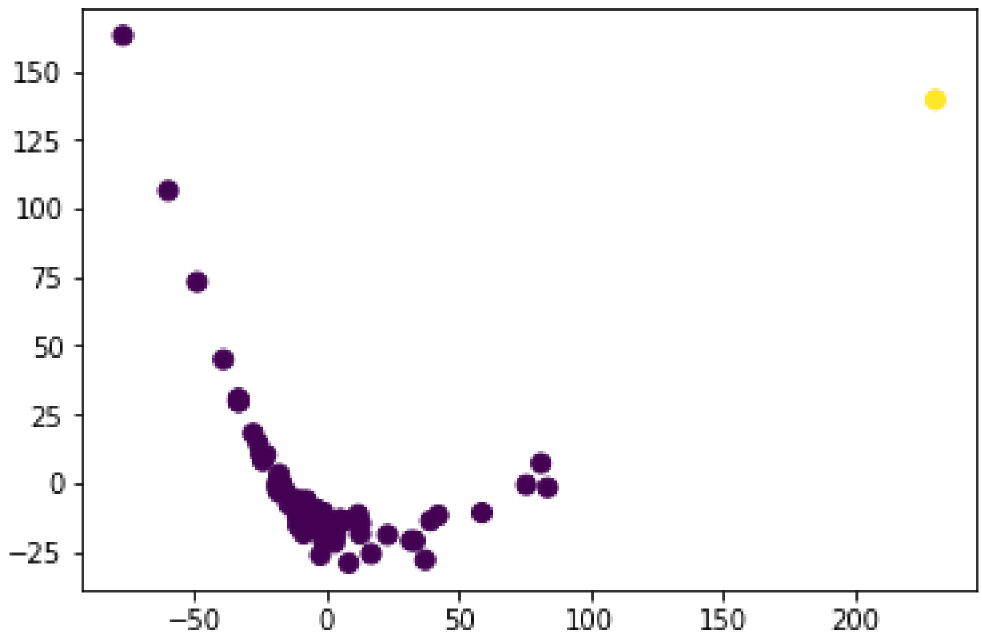



```
In [182]: k = range(2,20)
          silhouette = [0.0]*20
          for n_clusters in k:
              clusterer = KMeans(n_clusters=n_clusters, random_state=10)
              cluster_labels = clusterer.fit_predict(dataframe_pca)
              silhouette_avg = silhouette_score(dataframe_pca, cluster_labels)
              silhouette[n_clusters] = silhouette_avg
```

```
# We compute the score for each cluster and take the closest to 1
best_nb_clust = silhouette.index(max(silhouette))
print("The best number of cluster is : " + str(best_nb_clust))
```

```
The best number of cluster is : 2
```

In [179]:
```
kmeans_label = KMeans(n_clusters=2, random_state=10).fit_predict(dataframe_pca)
plt.scatter(dataframe_pca[:, 0], dataframe_pca[:, 1], c=kmeans_label,s=50,cmap='viridi
plt.show()
```

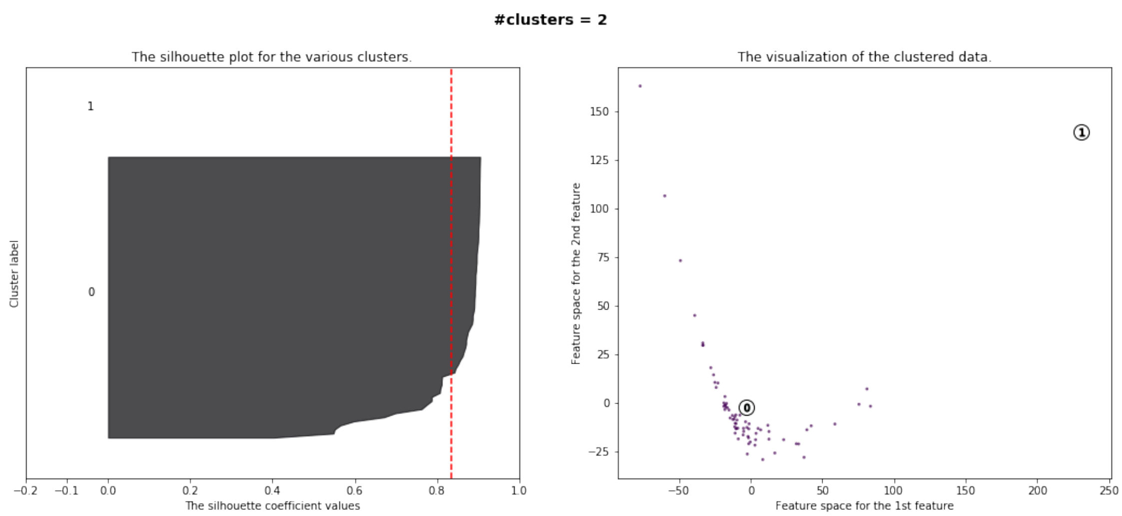


In [51]:
```
X = dataframe_pca
range_n_clusters = range(2,8)

for n_clusters in range_n_clusters:
    fig, (ax1, ax2) = plt.subplots(1, 2)
    fig.set_size_inches(18, 7)

    # Limit of the figure for the silhouette -1, 1
    ax1.set_xlim([-0.2, 1])
    ax1.set_ylim([0, len(X) + (n_clusters + 1) * 10])

    # Initialize the clusterer with n_clusters value and a random generator with speed
    clusterer = KMeans(n_clusters=n_clusters, random_state=10)
    cluster_labels = clusterer.fit_predict(X)
```

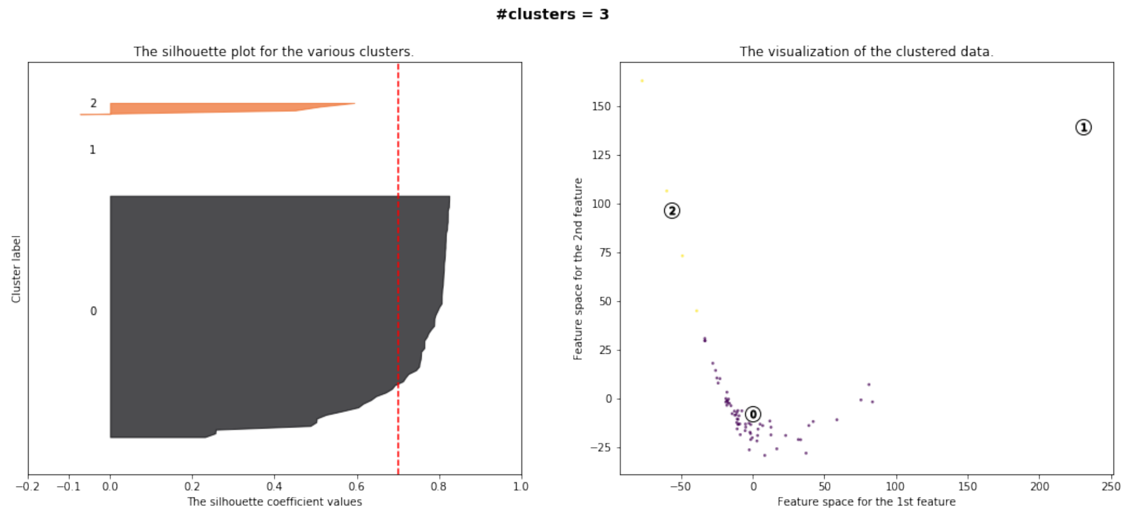
```python
# Silhouette score between -1 (worse) and 1 (better)
silhouette_avg = silhouette_score(X, cluster_labels)
print("For n_clusters =", n_clusters,
      "The average silhouette_score is :", silhouette_avg)

# Compute the silhouette scores for each sample
sample_silhouette_values = silhouette_samples(X, cluster_labels)

y_lower = 10
for i in range(n_clusters):
    # Aggregate the silhouette scores for samples belonging to
    # cluster i, and sort them
    ith_cluster_silhouette_values = \
        sample_silhouette_values[cluster_labels == i]

    ith_cluster_silhouette_values.sort()

    size_cluster_i = ith_cluster_silhouette_values.shape[0]
    y_upper = y_lower + size_cluster_i

    color = plt.cm.inferno(float(i) / n_clusters)
    ax1.fill_betweenx(np.arange(y_lower, y_upper),
                      0, ith_cluster_silhouette_values,
                      facecolor=color, edgecolor=color, alpha=0.7)

    # Label the silhouette plots with their cluster numbers at the middle
    ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))

    # Compute the new y_lower for next plot
    y_lower = y_upper + 10  # 10 for the 0 samples

ax1.set_title("The silhouette plot for the various clusters.")
ax1.set_xlabel("The silhouette coefficient values")
ax1.set_ylabel("Cluster label")

# The vertical line for average silhouette score of all the values
ax1.axvline(x=silhouette_avg, color="red", linestyle="--")

ax1.set_yticks([])  # Clear the yaxis labels / ticks
ax1.set_xticks([-0.2, -0.1, 0, 0.2, 0.4, 0.6, 0.8, 1])

# 2nd Plot showing the actual clusters formed
colors = plt.cm.inferno(cluster_labels.astype(float) / n_clusters)
ax2.scatter(X[:, 0], X[:, 1], marker='.', s=30, lw=0, alpha=0.7,
            c= cluster_labels , edgecolor='k')

# Labeling the clusters
centers = clusterer.cluster_centers_
```

```python
            # Draw white circles at cluster centers
            ax2.scatter(centers[:, 0], centers[:, 1], marker='o',
                        c="white", alpha=1, s=200, edgecolor='k')

            for i, c in enumerate(centers):
                ax2.scatter(c[0], c[1], marker='$%d$' % i, alpha=1,
                            s=50, edgecolor='k')

            ax2.set_title("The visualization of the clustered data.")
            ax2.set_xlabel("Feature space for the 1st feature")
            ax2.set_ylabel("Feature space for the 2nd feature")

            plt.suptitle(("#clusters = %d" % n_clusters),
                         fontsize=14, fontweight='bold')

            plt.show()
```

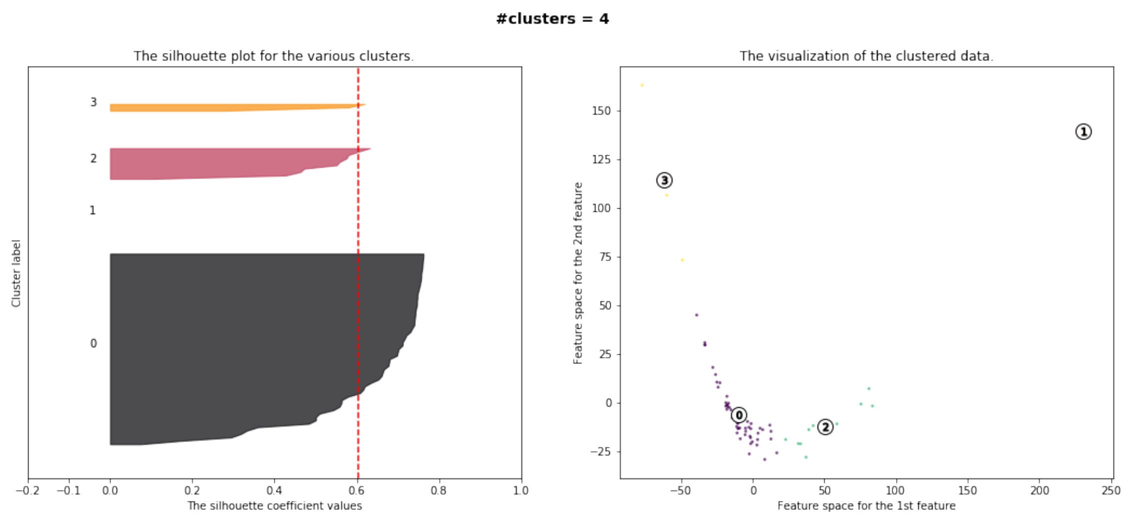
```
For n_clusters = 2 The average silhouette_score is : 0.834987361982
```



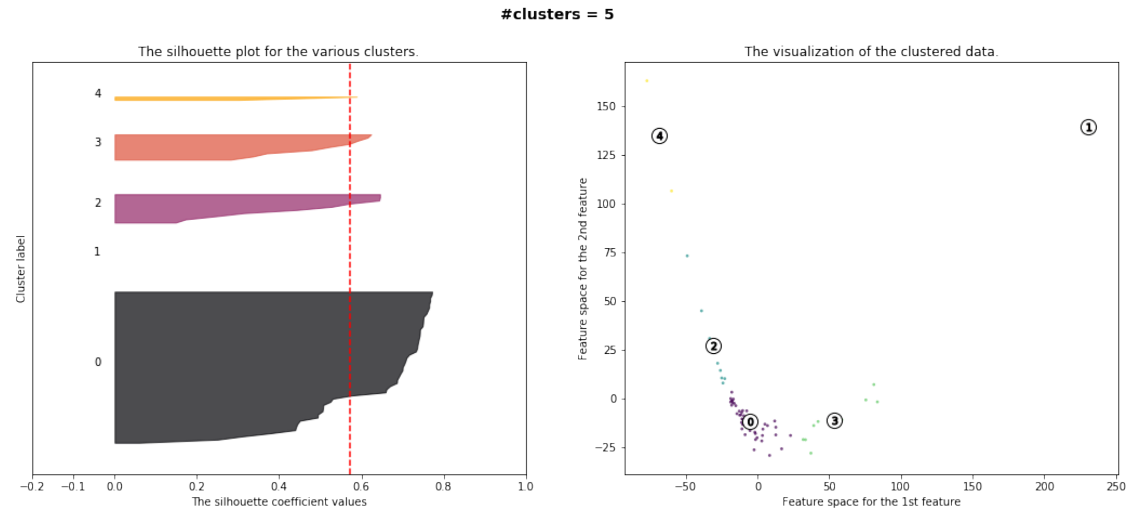

```
For n_clusters = 3 The average silhouette_score is : 0.702288559342
```



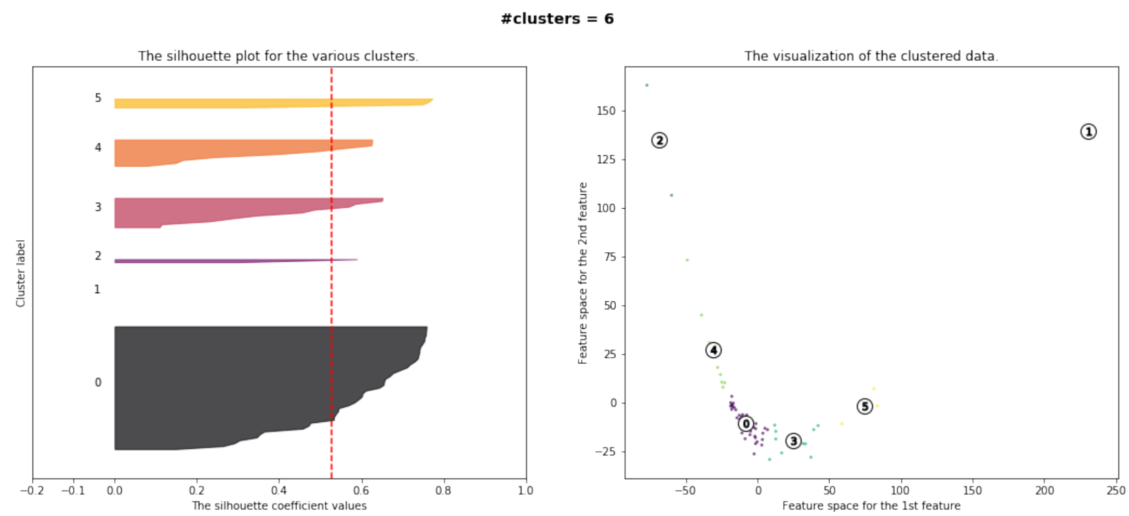
For n_clusters = 4 The average silhouette_score is : 0.604247818379



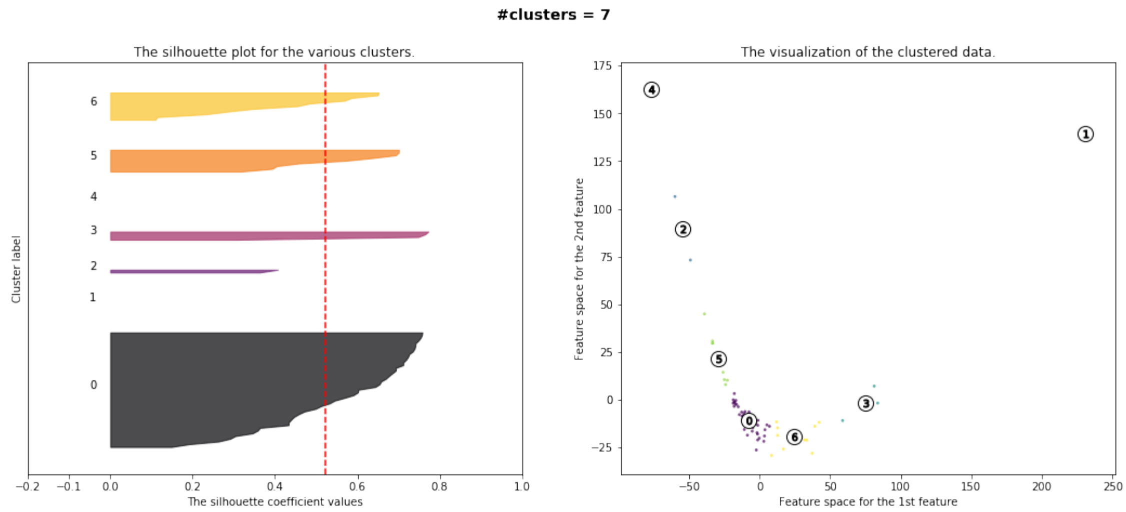
For n_clusters = 5 The average silhouette_score is : 0.57124965693



For n_clusters = 6 The average silhouette_score is : 0.527046673433



For n_clusters = 7 The average silhouette_score is : 0.521728418291



9 Supervised learning:KNN

Now let's talk about supervised learning, first we will show K-nearest neighbors

We split our info for two sets, training and testing, here we will show features of our KNN

9.0.1 - genre of the song recognition

In [94]:
```
trainingSet=[]
testSet=[]
split =0.9
```

In [95]:
```
numbers=[]
for x in range(len(df)-1):
    if random.random() < split:
        trainingSet.append(df.loc[x,:])
    else:
        testSet.append(df.loc[x,:])
        numbers.append(x)
```

In [79]:
```
def euclideanDistance(instance1, instance2, length):
        distance = 0
        for x in range(length):
                distance += pow((instance1[x] - instance2[x]), 2)
        return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
        distances = []
        length = len(testInstance)-1
        for x in range(len(trainingSet)):
                dist = euclideanDistance(testInstance, trainingSet[x], length)
```

```
                distances.append((trainingSet[x], dist))
        distances.sort(key=operator.itemgetter(1))
        neighbors = []
        for x in range(k):
                neighbors.append(distances[x][0])
        return neighbors
```

```
In [97]: def getResponse(neighbors):
                classVotes = {}
                for x in range(len(neighbors)):
                        response = neighbors[x][-1]
                        if response in classVotes:
                                classVotes[response] += 1
                        else:
                                classVotes[response] = 1
                sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=Tr
                return sortedVotes[0][0]
```

```
In [98]: print ('Train set: ' + repr(len(trainingSet)))
         print ('Test set: ' + repr(len(testSet)))
```

```
Train set: 66
Test set: 3
```

```
In [100]: predictions=[]
          k = 5
          for x in range(len(testSet)):
                  neighbors = getNeighbors(trainingSet, testSet[x], k)
                  result = getResponse(neighbors)
                  predictions.append(result)
                  print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1])+' son
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\ipykernel_l
  after removing the cwd from sys.path.
```

```
> predicted='ROCK', actual='ROCK' song: Passion Rules the Game  author: Scorpions
> predicted='ROCK', actual='ROCK' song: Still Loving You  author: Scorpions
> predicted='ROCK', actual='Metal' song: BYOB   author: System of a Down
```

9.0.2 - finding closest songs

```
In [74]: df = pd.concat([dfhashes, dfstyle], axis=1, join='inner')
         df_hashes_names = pd.concat([dfhashes_set, dfname_set] , axis=1)
```

```
In [75]: df_hashes_names=df_hashes_names.fillna(0)
```

```
In [76]: trainingSet=[]
         testSet=[df_hashes_names.loc[10,:],df_hashes_names.loc[6,:]]
         for x in range(len(df_hashes_names)-1):
             trainingSet.append(df_hashes_names.loc[x,:])
```

```
In [82]: predictions=[]
         k = 5
         for x in range(len(testSet)):
             neighbors = getNeighbors(trainingSet, testSet[x], k)
             print('> search for ' + repr(testSet[x][-1]))
             for i in range(len(neighbors)):
                 print('closest=' + str(neighbors[i][-1]))
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\ipykernel_l
  after removing the cwd from sys.path.


> search for 'Homage for Satan '
closest=Homage for Satan
closest=Passion Rules the Game
closest=Sweet Lady
closest=No one like you
closest=I Wanna Grow Old With You
> search for 'A Thousand Years '
closest=A Thousand Years
closest=Wonderwall
closest=I'm Goin' Mad
closest=Despacito
closest=Always Somewhere
```

9.0.3 - neural network for prediction genre

```
In [84]: dfname = pd.read_csv('data_tittles.csv', sep=',', header=None)
         dfname.columns = ["Title"]
         dfgenre = pd.read_csv('data_styles.csv', sep=',', header=None)
         dfgenre.columns = ['Genre']
         dfgenre['Genre'] = dfgenre['Genre'].str.strip()
         dfauthor = pd.read_csv('data_authors.csv', sep=',', header=None)
         dfauthor.columns = ["Author"]
         dfhashes = pd.read_csv('data.csv', sep=',', header=None)
         df_full = pd.concat([dfhashes,dfauthor,dfname,dfgenre] , axis=1)
         df_full=df_full.fillna(0)
         dfhashes=dfhashes.fillna(0)
```

```
In [85]: import glob
         import os
         import numpy as np
         import keras
```

```
from keras.layers import Input, Activation, Dense, BatchNormalization,Dropout
from keras.models import Model, Sequential
from keras.callbacks import ModelCheckpoint,Callback
import keras.backend as K
from keras.optimizers import SGD
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-packages\h5py\__init
  from ._conv import register_converters as _register_converters
Using TensorFlow backend.
```

In [86]:
```
model = Sequential()
model.add(Dense(units=5215*2, activation='sigmoid', input_dim=5215))
model.add(Dense(units=1000, activation='sigmoid'))

#model.add(Dropout(0.1))
model.add(Dense(units=5, activation='softmax'))

sgd = SGD(lr=0.01, momentum=0.9, decay=0, nesterov=True)
adam = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.

model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 10430)             54402880
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              10431000
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 5005
=================================================================
Total params: 64,838,885
Trainable params: 64,838,885
Non-trainable params: 0
_________________________________________________________________
```

In [87]:
```
def genreToVector(genre):
    genres = ['POP', 'CLASSIC', 'UNKNOWN', 'ROCK', 'Metal']
    vector = [0] * len(genres)
    vector[genres.index(genre)] = 1
    return vector
```

In [88]:
```
def vectorToGenre(vector):
    genres = ['POP', 'CLASSIC', 'UNKNOWN', 'ROCK', 'Metal']
    genre = genres[np.where(vector==1)[0][0]]
    return genre
```

```
In [89]: genre_train_str = np.array(df_full['Genre'])
         genre_train = np.array(list(map(genreToVector, genre_train_str)))

         data_train_nonorm = np.array(dfhashes)
         data_train = [0]*len(data_train_nonorm)

         for i in range(len(data_train_nonorm)):
             data_train[i] = data_train_nonorm[i]/float(max(data_train_nonorm[i]))

         data_train = np.array(data_train)
```

```
In [90]: #earlystop = keras.callbacks.EarlyStopping(monitor='loss', min_delta=1e-5, patience=5,
         model.fit(data_train, genre_train, epochs=200, batch_size=5)
```

```
Epoch 1/200
71/71 [==============================] - 17s 232ms/step - loss: 2.8564 - acc: 0.3099
Epoch 2/200
71/71 [==============================] - 14s 197ms/step - loss: 1.7149 - acc: 0.4648
Epoch 3/200
71/71 [==============================] - 14s 197ms/step - loss: 1.2901 - acc: 0.4366
Epoch 4/200
71/71 [==============================] - 14s 191ms/step - loss: 1.3377 - acc: 0.4507
Epoch 5/200
71/71 [==============================] - 14s 192ms/step - loss: 1.2845 - acc: 0.4507
Epoch 6/200
71/71 [==============================] - 14s 194ms/step - loss: 1.2614 - acc: 0.3944
Epoch 7/200
71/71 [==============================] - 14s 196ms/step - loss: 1.2709 - acc: 0.3803
Epoch 8/200
71/71 [==============================] - 14s 197ms/step - loss: 1.1919 - acc: 0.5493
Epoch 9/200
71/71 [==============================] - 14s 198ms/step - loss: 1.2928 - acc: 0.3803
Epoch 10/200
71/71 [==============================] - 14s 196ms/step - loss: 1.2262 - acc: 0.5070
Epoch 11/200
71/71 [==============================] - 13s 189ms/step - loss: 1.2290 - acc: 0.4648
Epoch 12/200
71/71 [==============================] - 13s 189ms/step - loss: 1.2163 - acc: 0.5211
Epoch 13/200
71/71 [==============================] - 13s 189ms/step - loss: 1.2064 - acc: 0.4648
Epoch 14/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1835 - acc: 0.5070
Epoch 15/200
71/71 [==============================] - 13s 190ms/step - loss: 1.1691 - acc: 0.5070
Epoch 16/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1466 - acc: 0.4930
Epoch 17/200
71/71 [==============================] - 14s 192ms/step - loss: 1.1401 - acc: 0.5493
```

```
Epoch 18/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1849 - acc: 0.4930
Epoch 19/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1015 - acc: 0.5070
Epoch 20/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1449 - acc: 0.4789
Epoch 21/200
71/71 [==============================] - 13s 190ms/step - loss: 1.1145 - acc: 0.5352
Epoch 22/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1309 - acc: 0.4366
Epoch 23/200
71/71 [==============================] - 13s 188ms/step - loss: 1.1704 - acc: 0.5352
Epoch 24/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1493 - acc: 0.5211
Epoch 25/200
71/71 [==============================] - 14s 192ms/step - loss: 1.1449 - acc: 0.4648
Epoch 26/200
71/71 [==============================] - 14s 194ms/step - loss: 1.0467 - acc: 0.4930
Epoch 27/200
71/71 [==============================] - 13s 188ms/step - loss: 1.0603 - acc: 0.4507
Epoch 28/200
71/71 [==============================] - 13s 189ms/step - loss: 1.0750 - acc: 0.5352
Epoch 29/200
71/71 [==============================] - 13s 190ms/step - loss: 1.1252 - acc: 0.4930
Epoch 30/200
71/71 [==============================] - 14s 190ms/step - loss: 1.0809 - acc: 0.4789
Epoch 31/200
71/71 [==============================] - 13s 189ms/step - loss: 1.0724 - acc: 0.4930
Epoch 32/200
71/71 [==============================] - 13s 189ms/step - loss: 1.0169 - acc: 0.5352
Epoch 33/200
71/71 [==============================] - 13s 189ms/step - loss: 1.1601 - acc: 0.5070
Epoch 34/200
71/71 [==============================] - 13s 190ms/step - loss: 1.0876 - acc: 0.4648
Epoch 35/200
71/71 [==============================] - 14s 193ms/step - loss: 1.0379 - acc: 0.4789
Epoch 36/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9943 - acc: 0.5070
Epoch 37/200
71/71 [==============================] - 14s 192ms/step - loss: 1.0924 - acc: 0.5352
Epoch 38/200
71/71 [==============================] - 13s 190ms/step - loss: 1.0284 - acc: 0.5211
Epoch 39/200
71/71 [==============================] - 14s 192ms/step - loss: 0.9739 - acc: 0.5352
Epoch 40/200
71/71 [==============================] - 13s 188ms/step - loss: 1.0125 - acc: 0.5352
Epoch 41/200
71/71 [==============================] - 13s 188ms/step - loss: 1.0160 - acc: 0.5352
```

```
Epoch 42/200
71/71 [==============================] - 13s 188ms/step - loss: 1.0263 - acc: 0.5634
Epoch 43/200
71/71 [==============================] - 14s 190ms/step - loss: 1.0020 - acc: 0.5634
Epoch 44/200
71/71 [==============================] - 14s 191ms/step - loss: 0.9759 - acc: 0.5634
Epoch 45/200
71/71 [==============================] - 13s 189ms/step - loss: 1.0504 - acc: 0.4366
Epoch 46/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9975 - acc: 0.5634
Epoch 47/200
71/71 [==============================] - 13s 190ms/step - loss: 1.0161 - acc: 0.5352
Epoch 48/200
71/71 [==============================] - 14s 191ms/step - loss: 0.9425 - acc: 0.5493
Epoch 49/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9520 - acc: 0.5775
Epoch 50/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9351 - acc: 0.5915
Epoch 51/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9214 - acc: 0.5211
Epoch 52/200
71/71 [==============================] - 14s 192ms/step - loss: 1.0036 - acc: 0.5211
Epoch 53/200
71/71 [==============================] - 15s 208ms/step - loss: 0.9543 - acc: 0.6056
Epoch 54/200
71/71 [==============================] - 16s 226ms/step - loss: 0.9682 - acc: 0.5211
Epoch 55/200
71/71 [==============================] - 15s 211ms/step - loss: 0.9333 - acc: 0.5211
Epoch 56/200
71/71 [==============================] - 15s 207ms/step - loss: 0.8984 - acc: 0.6338
Epoch 57/200
71/71 [==============================] - 13s 188ms/step - loss: 0.9241 - acc: 0.5775
Epoch 58/200
71/71 [==============================] - 13s 188ms/step - loss: 0.9354 - acc: 0.6338
Epoch 59/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9458 - acc: 0.5775
Epoch 60/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9040 - acc: 0.5915
Epoch 61/200
71/71 [==============================] - 14s 192ms/step - loss: 0.9176 - acc: 0.6056
Epoch 62/200
71/71 [==============================] - 13s 189ms/step - loss: 0.8517 - acc: 0.6197
Epoch 63/200
71/71 [==============================] - 14s 193ms/step - loss: 0.9173 - acc: 0.5775
Epoch 64/200
71/71 [==============================] - 15s 209ms/step - loss: 0.8601 - acc: 0.6479
Epoch 65/200
71/71 [==============================] - 13s 190ms/step - loss: 0.9111 - acc: 0.5634
```

```
Epoch 66/200
71/71 [==============================] - 14s 191ms/step - loss: 0.8487 - acc: 0.6479
Epoch 67/200
71/71 [==============================] - 15s 205ms/step - loss: 0.8950 - acc: 0.6620
Epoch 68/200
71/71 [==============================] - 14s 191ms/step - loss: 0.8830 - acc: 0.6197
Epoch 69/200
71/71 [==============================] - 14s 193ms/step - loss: 0.8111 - acc: 0.6338
Epoch 70/200
71/71 [==============================] - 14s 197ms/step - loss: 1.0874 - acc: 0.4930
Epoch 71/200
71/71 [==============================] - 13s 189ms/step - loss: 0.8981 - acc: 0.5634
Epoch 72/200
71/71 [==============================] - 13s 190ms/step - loss: 0.8798 - acc: 0.5775
Epoch 73/200
71/71 [==============================] - 13s 189ms/step - loss: 0.8296 - acc: 0.7042
Epoch 74/200
71/71 [==============================] - 13s 190ms/step - loss: 0.8601 - acc: 0.6056
Epoch 75/200
71/71 [==============================] - 13s 189ms/step - loss: 0.9654 - acc: 0.5775
Epoch 76/200
71/71 [==============================] - 14s 193ms/step - loss: 0.8764 - acc: 0.6197
Epoch 77/200
71/71 [==============================] - 13s 190ms/step - loss: 0.8857 - acc: 0.6901
Epoch 78/200
71/71 [==============================] - 15s 217ms/step - loss: 0.9511 - acc: 0.5634
Epoch 79/200
71/71 [==============================] - 17s 245ms/step - loss: 0.8113 - acc: 0.5915
Epoch 80/200
71/71 [==============================] - 18s 260ms/step - loss: 0.8219 - acc: 0.6620
Epoch 81/200
71/71 [==============================] - 19s 267ms/step - loss: 0.8050 - acc: 0.6338
Epoch 82/200
71/71 [==============================] - 19s 272ms/step - loss: 0.8322 - acc: 0.6338
Epoch 83/200
71/71 [==============================] - 19s 272ms/step - loss: 0.7710 - acc: 0.6901
Epoch 84/200
71/71 [==============================] - 18s 258ms/step - loss: 0.7765 - acc: 0.6761
Epoch 85/200
71/71 [==============================] - 19s 271ms/step - loss: 0.7777 - acc: 0.7042
Epoch 86/200
71/71 [==============================] - 19s 264ms/step - loss: 0.8030 - acc: 0.6338
Epoch 87/200
71/71 [==============================] - 19s 267ms/step - loss: 0.7215 - acc: 0.7042
Epoch 88/200
71/71 [==============================] - 19s 266ms/step - loss: 0.6926 - acc: 0.6901
Epoch 89/200
71/71 [==============================] - 19s 263ms/step - loss: 0.7125 - acc: 0.6761
```

```
Epoch 90/200
71/71 [==============================] - 19s 264ms/step - loss: 0.7955 - acc: 0.6479
Epoch 91/200
71/71 [==============================] - 19s 272ms/step - loss: 0.6990 - acc: 0.7042
Epoch 92/200
71/71 [==============================] - 19s 268ms/step - loss: 0.7437 - acc: 0.6761
Epoch 93/200
71/71 [==============================] - 19s 265ms/step - loss: 0.6686 - acc: 0.7324
Epoch 94/200
71/71 [==============================] - 19s 262ms/step - loss: 0.6385 - acc: 0.7606
Epoch 95/200
71/71 [==============================] - 19s 266ms/step - loss: 0.7434 - acc: 0.6338
Epoch 96/200
71/71 [==============================] - 19s 270ms/step - loss: 0.7021 - acc: 0.7042
Epoch 97/200
71/71 [==============================] - 21s 289ms/step - loss: 0.5949 - acc: 0.7465
Epoch 98/200
71/71 [==============================] - 19s 271ms/step - loss: 0.7038 - acc: 0.6901
Epoch 99/200
71/71 [==============================] - 19s 264ms/step - loss: 0.5815 - acc: 0.8028
Epoch 100/200
71/71 [==============================] - 19s 269ms/step - loss: 0.7438 - acc: 0.6479
Epoch 101/200
71/71 [==============================] - 19s 265ms/step - loss: 0.6480 - acc: 0.6901
Epoch 102/200
71/71 [==============================] - 19s 265ms/step - loss: 0.6947 - acc: 0.7183
Epoch 103/200
71/71 [==============================] - 19s 269ms/step - loss: 0.5939 - acc: 0.7606
Epoch 104/200
71/71 [==============================] - 20s 275ms/step - loss: 0.5438 - acc: 0.7465
Epoch 105/200
71/71 [==============================] - 19s 267ms/step - loss: 0.4991 - acc: 0.8451
Epoch 106/200
71/71 [==============================] - 19s 266ms/step - loss: 0.4673 - acc: 0.8028
Epoch 107/200
71/71 [==============================] - 19s 268ms/step - loss: 0.5184 - acc: 0.7746
Epoch 108/200
71/71 [==============================] - 19s 266ms/step - loss: 0.6038 - acc: 0.7324
Epoch 109/200
71/71 [==============================] - 19s 269ms/step - loss: 0.5121 - acc: 0.8592
Epoch 110/200
71/71 [==============================] - 19s 273ms/step - loss: 0.5159 - acc: 0.8310
Epoch 111/200
71/71 [==============================] - 19s 265ms/step - loss: 0.3997 - acc: 0.8169
Epoch 112/200
71/71 [==============================] - 19s 268ms/step - loss: 0.3832 - acc: 0.8873
Epoch 113/200
71/71 [==============================] - 19s 269ms/step - loss: 0.3406 - acc: 0.8451
```

```
Epoch 114/200
71/71 [==============================] - 19s 266ms/step - loss: 0.3446 - acc: 0.9014
Epoch 115/200
71/71 [==============================] - 19s 264ms/step - loss: 0.3496 - acc: 0.9155
Epoch 116/200
71/71 [==============================] - 19s 272ms/step - loss: 0.4317 - acc: 0.8592
Epoch 117/200
71/71 [==============================] - 19s 266ms/step - loss: 0.4872 - acc: 0.8310
Epoch 118/200
71/71 [==============================] - 19s 271ms/step - loss: 0.7486 - acc: 0.7042
Epoch 119/200
71/71 [==============================] - 19s 266ms/step - loss: 0.3040 - acc: 0.9437
Epoch 120/200
71/71 [==============================] - 19s 265ms/step - loss: 0.3062 - acc: 0.8873
Epoch 121/200
71/71 [==============================] - 19s 265ms/step - loss: 0.1965 - acc: 0.9577
Epoch 122/200
71/71 [==============================] - 19s 264ms/step - loss: 0.3304 - acc: 0.8732
Epoch 123/200
71/71 [==============================] - 19s 270ms/step - loss: 0.1780 - acc: 0.9437
Epoch 124/200
71/71 [==============================] - 19s 262ms/step - loss: 0.4113 - acc: 0.8451
Epoch 125/200
71/71 [==============================] - 19s 269ms/step - loss: 0.4134 - acc: 0.8592
Epoch 126/200
71/71 [==============================] - 19s 264ms/step - loss: 0.7960 - acc: 0.7324
Epoch 127/200
71/71 [==============================] - 19s 267ms/step - loss: 0.3653 - acc: 0.8732
Epoch 128/200
71/71 [==============================] - 19s 264ms/step - loss: 0.3147 - acc: 0.9014
Epoch 129/200
71/71 [==============================] - 19s 271ms/step - loss: 0.3339 - acc: 0.8732
Epoch 130/200
71/71 [==============================] - 19s 269ms/step - loss: 0.1462 - acc: 0.9437
Epoch 131/200
71/71 [==============================] - 19s 265ms/step - loss: 0.1427 - acc: 0.9718
Epoch 132/200
71/71 [==============================] - 19s 263ms/step - loss: 0.1000 - acc: 0.9718
Epoch 133/200
71/71 [==============================] - 19s 264ms/step - loss: 0.0978 - acc: 0.9718
Epoch 134/200
71/71 [==============================] - 19s 271ms/step - loss: 0.1076 - acc: 0.9859
Epoch 135/200
71/71 [==============================] - 20s 279ms/step - loss: 0.0958 - acc: 0.9859
Epoch 136/200
71/71 [==============================] - 19s 274ms/step - loss: 0.0925 - acc: 0.9718
Epoch 137/200
71/71 [==============================] - 19s 274ms/step - loss: 0.1540 - acc: 0.9437
```

```
Epoch 138/200
71/71 [==============================] - 19s 269ms/step - loss: 0.4208 - acc: 0.9577
Epoch 139/200
71/71 [==============================] - 20s 276ms/step - loss: 1.4502 - acc: 0.7183
Epoch 140/200
71/71 [==============================] - 20s 281ms/step - loss: 0.2704 - acc: 0.9296
Epoch 141/200
71/71 [==============================] - 20s 282ms/step - loss: 0.2422 - acc: 0.9296
Epoch 142/200
71/71 [==============================] - 20s 280ms/step - loss: 0.1647 - acc: 0.9437
Epoch 143/200
71/71 [==============================] - 19s 273ms/step - loss: 0.1268 - acc: 0.9718
Epoch 144/200
71/71 [==============================] - 20s 279ms/step - loss: 0.1254 - acc: 0.9718
Epoch 145/200
71/71 [==============================] - 20s 275ms/step - loss: 0.0998 - acc: 0.9718
Epoch 146/200
71/71 [==============================] - 20s 275ms/step - loss: 0.1606 - acc: 0.9296
Epoch 147/200
71/71 [==============================] - 19s 272ms/step - loss: 0.2967 - acc: 0.8873
Epoch 148/200
71/71 [==============================] - 20s 279ms/step - loss: 0.0565 - acc: 1.0000
Epoch 149/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0484 - acc: 1.0000
Epoch 150/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0500 - acc: 1.0000
Epoch 151/200
71/71 [==============================] - 19s 273ms/step - loss: 0.0376 - acc: 1.0000
Epoch 152/200
71/71 [==============================] - 19s 274ms/step - loss: 0.0455 - acc: 1.0000
Epoch 153/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0351 - acc: 1.0000
Epoch 154/200
71/71 [==============================] - 20s 277ms/step - loss: 0.0362 - acc: 1.0000
Epoch 155/200
71/71 [==============================] - 20s 281ms/step - loss: 0.0272 - acc: 1.0000
Epoch 156/200
71/71 [==============================] - 21s 295ms/step - loss: 0.0285 - acc: 1.0000
Epoch 157/200
71/71 [==============================] - 20s 280ms/step - loss: 0.0244 - acc: 1.0000
Epoch 158/200
71/71 [==============================] - 19s 272ms/step - loss: 0.0233 - acc: 1.0000
Epoch 159/200
71/71 [==============================] - 19s 268ms/step - loss: 0.0213 - acc: 1.0000
Epoch 160/200
71/71 [==============================] - 20s 280ms/step - loss: 0.0201 - acc: 1.0000
Epoch 161/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0236 - acc: 1.0000
```

```
Epoch 162/200
71/71 [==============================] - 19s 264ms/step - loss: 0.0202 - acc: 1.0000
Epoch 163/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0215 - acc: 1.0000
Epoch 164/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0179 - acc: 1.0000
Epoch 165/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0148 - acc: 1.0000
Epoch 166/200
71/71 [==============================] - 20s 279ms/step - loss: 0.0160 - acc: 1.0000
Epoch 167/200
71/71 [==============================] - 19s 272ms/step - loss: 0.0132 - acc: 1.0000
Epoch 168/200
71/71 [==============================] - 19s 268ms/step - loss: 0.0138 - acc: 1.0000
Epoch 169/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0121 - acc: 1.0000
Epoch 170/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0132 - acc: 1.0000
Epoch 171/200
71/71 [==============================] - 19s 268ms/step - loss: 0.0129 - acc: 1.0000
Epoch 172/200
71/71 [==============================] - 19s 274ms/step - loss: 0.0121 - acc: 1.0000
Epoch 173/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0106 - acc: 1.0000
Epoch 174/200
71/71 [==============================] - 19s 272ms/step - loss: 0.0102 - acc: 1.0000
Epoch 175/200
71/71 [==============================] - 19s 269ms/step - loss: 0.0103 - acc: 1.0000
Epoch 176/200
71/71 [==============================] - 20s 287ms/step - loss: 0.0095 - acc: 1.0000
Epoch 177/200
71/71 [==============================] - 20s 281ms/step - loss: 0.0097 - acc: 1.0000
Epoch 178/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0084 - acc: 1.0000
Epoch 179/200
71/71 [==============================] - 20s 276ms/step - loss: 0.0088 - acc: 1.0000
Epoch 180/200
71/71 [==============================] - 19s 267ms/step - loss: 0.0083 - acc: 1.0000
Epoch 181/200
71/71 [==============================] - 19s 269ms/step - loss: 0.0080 - acc: 1.0000
Epoch 182/200
71/71 [==============================] - 20s 276ms/step - loss: 0.0088 - acc: 1.0000
Epoch 183/200
71/71 [==============================] - 20s 279ms/step - loss: 0.0078 - acc: 1.0000
Epoch 184/200
71/71 [==============================] - 20s 275ms/step - loss: 0.0085 - acc: 1.0000
Epoch 185/200
71/71 [==============================] - 20s 281ms/step - loss: 0.0073 - acc: 1.0000
```

```
Epoch 186/200
71/71 [==============================] - 19s 272ms/step - loss: 0.0071 - acc: 1.0000
Epoch 187/200
71/71 [==============================] - 19s 268ms/step - loss: 0.0074 - acc: 1.0000
Epoch 188/200
71/71 [==============================] - 19s 271ms/step - loss: 0.0068 - acc: 1.0000
Epoch 189/200
71/71 [==============================] - 19s 266ms/step - loss: 0.0069 - acc: 1.0000
Epoch 190/200
71/71 [==============================] - 20s 280ms/step - loss: 0.0066 - acc: 1.0000
Epoch 191/200
71/71 [==============================] - 20s 283ms/step - loss: 0.0065 - acc: 1.0000
Epoch 192/200
71/71 [==============================] - 19s 273ms/step - loss: 0.0061 - acc: 1.0000
Epoch 193/200
71/71 [==============================] - 20s 275ms/step - loss: 0.0063 - acc: 1.0000
Epoch 194/200
71/71 [==============================] - 19s 268ms/step - loss: 0.0057 - acc: 1.0000
Epoch 195/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0065 - acc: 1.0000
Epoch 196/200
71/71 [==============================] - 19s 270ms/step - loss: 0.0062 - acc: 1.0000
Epoch 197/200
71/71 [==============================] - 17s 246ms/step - loss: 0.0070 - acc: 1.0000
Epoch 198/200
71/71 [==============================] - 17s 236ms/step - loss: 0.0058 - acc: 1.0000
Epoch 199/200
71/71 [==============================] - 17s 236ms/step - loss: 0.0051 - acc: 1.0000
Epoch 200/200
71/71 [==============================] - 17s 235ms/step - loss: 0.0053 - acc: 1.0000
```

```
Out[90]: <keras.callbacks.History at 0x172a4d8b278>
```

```
In [93]: end_result = model.evaluate(x_train, y_train, batch_size=1)
         print(loss_and_metrics)
```

```
71/71 [==============================] - 3s 40ms/step
[0.004926996493120553, 1.0]
```

```
In [96]: count_t = 0
         count_f = 0

         for line in range(0,71):
             print(dfgenre.iloc[[line]].values[0][0])
             classes = model.predict(np.array(dfhashes.iloc[[line]]))
             vector = np.zeros(5)
             vector[np.where(classes == max(max(classes)))[1][0]] = 1
```

```
            print(vectorToGenre(vector))

            if vectorToGenre(vector) == dfgenre.iloc[[line]].values[0][0]:
                print('True')
                count_t += 1
            else :
                print('False')
                count_f += 1
            print()
```

```
POP
POP
True

POP
POP
True

CLASSIC
CLASSIC
True

POP
POP
True

POP
POP
True

UNKNOWN
ROCK
False

POP
POP
True

POP
POP
True

ROCK
ROCK
True

POP
POP
```

```
True

Metal
Metal
True

CLASSIC
CLASSIC
True

POP
POP
True

POP
POP
True

POP
POP
True

CLASSIC
CLASSIC
True

CLASSIC
CLASSIC
True

CLASSIC
CLASSIC
True

CLASSIC
CLASSIC
True

POP
POP
True

CLASSIC
CLASSIC
True

POP
POP
```

```
True

CLASSIC
CLASSIC
True

POP
POP
True

POP
POP
True

ROCK
ROCK
True

POP
POP
True

POP
POP
True

POP
POP
True

POP
POP
True

POP
POP
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
```

```
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

POP
POP
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
```

```
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

ROCK
ROCK
True

CLASSIC
CLASSIC
True

POP
POP
```

```
True

POP
POP
True

Metal
Metal
True

CLASSIC
CLASSIC
True

Metal
Metal
True

POP
POP
True

POP
POP
True

POP
POP
True

POP
POP
True

UNKNOWN
CLASSIC
False

POP
POP
True

POP
POP
True

POP
POP
```

```
True

POP
POP
True
```

```
In [95]: print('# of true : ' + str(count_t))
         print('# of false : ' + str(count_f))
```

```
# of true : 69
# of false : 2
```

9.0.4 - Shazam alogithm

```
In [97]: i_love_this_song = df_full.iloc[6][1000:2000]
         print(str(df_full.iloc[6]['Author'])
               + ' - '+ str(df_full.iloc[6]['Title'])
               + 'and genre of song is '+  str(df_full.iloc[6]['Genre']))
```

```
Christina Perri - A Thousand Years and genre of song is POP
```

```
In [98]: def subfinder(mylist, pattern):
             result = []
             ansv = False
             for i in range(0,len(mylist)):
                 print('Checking the {0} song for similar interval'.format(i))
                 for j in range(len(mylist.iloc[i]) - len(pattern)):
                     if list(pattern) == list(mylist.iloc[i][j:j + len(pattern)]):
                          ansv = True
                 if ansv == True:
                     result.append(mylist.iloc[i])
                     ansv = False
                     break
             return result
```

```
In [99]: ans = subfinder(df_full, df_full.iloc[6][1000:2000])
```

```
Checking the 0 song for similar interval
Checking the 1 song for similar interval
Checking the 2 song for similar interval
Checking the 3 song for similar interval
Checking the 4 song for similar interval
Checking the 5 song for similar interval
Checking the 6 song for similar interval
```

```
In [ ]: ans = pd.DataFrame(ans)
        print(str(ans['Author'].values[0]) +' - '+ str(ans['Title'].values[0]) +'and genre of so
```

9.0.5 Conclusion

So in our project we were working with song recognition, for that we create our new dataset, by algorithm parse it into specific unique representation, then we did some data analisys which are visualized on graphics in report, unsupervised learning is presented by k-means clasteing, and for supervised learning we used few algorithms as KNN and neural network, also we used the shazam algorithm for detecting song, as result we got genre recognition with accuracy 100% for neural network, and 77% for KNN, also by KNN we found the most similar songs for chosen one, and shazam alogrithm gave us result of song recognition with small interval of song as input.