WORD2VEC - Exploracion con SongLyrics.csv (canciones en ingles) y comparacion con modelo pre-entrenado de Google

**En este notebook aun no vamos a eliminar las stopwords por lo que nos saldran conjugciones y palabras con la misma raiz. (quitaremos stopwords en los siguientes notebooks para comparar)

Depdende de las palabras que usemos el modelo entrenado con songlyrics.csv funciona bastante bien! Hasta casi tan bien como el de google. Esto sucede si usamos palabras muy recurrentes en canciones en ingles, ya que el corpus de canciones es batante grande.

In [25]:
```
import pandas as pd
import numpy as np
import gensim.models.word2vec as w2v
import multiprocessing
import os
import re
import pprint
import sklearn.manifold
import matplotlib.pyplot as plt
from gensim.test.utils import datapath

from keras.utils import get_file
import gensim
import subprocess
from IPython.core.pylabtools import figsize
```

Though non english artists were removed, the dataset contained Hindi lyrics of Lata Mangeshkar written in English. Therefore, I decided to remove all songs sung by her.

In [26]:
```
songs = pd.read_csv("data/songdata.csv", header=0)
#songs.head()
songs = songs[songs.artist != 'Lata Mangeshkar']
songs.head()
```

Out[26]:

	artist	song	link	text
0	ABBA	Ahe's My Kind Of Girl	/a/abba/ahes+my+kind+of+girl_20598417.html	Look at her face, it's a wonderful face \nAnd...
1	ABBA	Andante, Andante	/a/abba/andante+andante_20002708.html	Take it easy with me, please \nTouch me gentl...
2	ABBA	As Good As New	/a/abba/as+good+as+new_20003033.html	I'll never know why I had to go \nWhy I had t...
3	ABBA	Bang	/a/abba/bang_20598415.html	Making somebody happy is a question of give an...
4	ABBA	Bang-A-Boomerang	/a/abba/bang+a+boomerang_20002668.html	Making somebody happy is a question of give an...

To train the word2vec model, we first need to build its vocabulary. To do that, I iterated over each song and added it to an array that can later be fed to the model.

In [27]:

```
import nltk
text_corpus = []
for song in songs['text']:
    #words = song.lower().split()
    #text_corpus.append(words)

    tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+') #para dividir por words y quitar puntuacion
    lower_case = song.lower()
    tokens_sin_puntuacion = tokenizer.tokenize(lower_case)

    text_corpus.append(tokens_sin_puntuacion)


# Dimensionality of the resulting word vectors.
#more dimensions, more computationally expensive to train
#but also more accurate
#more dimensions = more generalized
num_features = 50
# Minimum word count threshold.
min_word_count = 1

# Number of threads to run in parallel.
#more workers, faster we train
num_workers = multiprocessing.cpu_count()

# Context window length.
context_size = 7


downsampling = 1e-1

# Seed for the RNG, to make the results reproducible.
#random number generator
#deterministic, good for debugging
seed = 1

songs2vec = w2v.Word2Vec(
    sg=1,
    seed=seed,
    workers=num_workers,
    size=num_features,
```

```
    min_count=min_word_count,
    window=context_size,
    sample=downsampling
)

songs2vec.build_vocab(text_corpus)
```

In [28]:
```
print (len(text_corpus))
```

```
57618
```

In [29]:
```
import time
start_time = time.time()



songs2vec.train(text_corpus, total_examples=songs2vec.corpus_count, epochs=2)

if not os.path.exists("trained"):
    os.makedirs("trained")

songs2vec.save(os.path.join("trained", "songs2vectors.w2v"))

print("--- %s seconds ---" % (time.time() - start_time))
```

```
--- 67.86585187911987 seconds ---
```

In [30]:
```
songs2vec = w2v.Word2Vec.load(os.path.join("trained", "songs2vectors.w2v"))
```

Let's explore our model

Find similar words

In [31]:
```
songs2vec.wv.most_similar("kiss")
```

Out[31]:
```
[('hug', 0.879738450050354),
 ('caress', 0.8356099128723145),
 ('touch', 0.8292093276977539),
 ('sweetness', 0.8195240497589111),
 ('savor', 0.818192720413208),
 ('flutter', 0.7946850061416626),
 ('clumsy', 0.7910999059677124),
 ('kisses', 0.7899938225746155),
 ('tenderness', 0.7872726917266846),
 ('lips', 0.7870891094207764)]
```

In [32]:
```
songs2vec.wv.most_similar("forever")
```

Out[32]:
```
[('eternally', 0.8463695049285889),
 ('forevermore', 0.8092575073242188),
 ('tarry', 0.7957216501235962),
 ('happily', 0.7942827939987183),
 ('cherish', 0.7940607666969299),
 ('endures', 0.7932143211364746),
 ('surely', 0.7909528017044067),
 ('ever', 0.7885740399360657),
 ('outlive', 0.787304699420929),
 ('lorelei', 0.7854410409927368)]
```

In [33]:
```
songs2vec.wv.most_similar("time")
```

Out[33]:
```
[('risk', 0.8297646045684814),
 ('chance', 0.8290538787841797),
 ('moment', 0.827455997467041),
 ('discussion', 0.8233293294906616),
 ('phase', 0.8231333494186401),
 ('consideration', 0.8189870715141296),
 ('everday', 0.81794273853302),
 ('celebate', 0.8118423223495483),
 ('understandin', 0.8114659190177917),
 ('argument', 0.8095056414604187)]
```

In [34]:
```
songs2vec.wv.most_similar("love")
```

Out[34]:
```
[('survives', 0.8622327446937561),
 ('soften', 0.8612666130065918),
 ('unconditionally', 0.8534113168716431),
 ('ecstasy', 0.8522416949272156),
 ('surrendering', 0.8446434736251831),
 ('tonite', 0.8432831764221191),
 ('devotion', 0.8411402702331543),
 ('bothers', 0.8404875993728638),
 ('coz', 0.8395731449127197),
 ('affection', 0.8363974690437317)]
```

In [35]:
```
songs2vec.wv.most_similar("fuck")
```

Out[35]:
```
[('yall', 0.8672010898590088),
 ('fuckin', 0.862307608127594),
 ('nigga', 0.8555626273155212),
 ('shit', 0.8489099740982056),
 ('bitch', 0.8438244462013245),
 ('beef', 0.8434240818023682),
 ('mutha', 0.8390918970108032),
 ('motherfucker', 0.8297766447067261),
 ('motherfuckin', 0.8297736644744873),
 ('hype', 0.8288718461990356)]
```

Words out of context

In [36]:
```
songs2vec.wv.doesnt_match("happiness love joy hate".split())
```

```
C:\Users\jhern\Anaconda3\lib\site-packages\gensim\models\keyedvectors.py:877: FutureWarning:

arrays to stack must be passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such
as generators is deprecated as of NumPy 1.16 and will raise an error in the future.
```

Out[36]:
```
'hate'
```

In [37]:
```
songs2vec.wv.doesnt_match("fun funny enjoy hate".split())
```

Out[37]:
```
'hate'
```

In [38]:
```
songs2vec.wv.doesnt_match("people man woman song".split())
```

Out[38]:
```
'song'
```

In [39]:
```
songs2vec.most_similar(positive=['woman', 'king'], negative=['man'])
#queen
```

```
C:\Users\jhern\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: DeprecationWarning:

Call to deprecated `most_similar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
```

Out[39]:
```
[('gift', 0.7659988403320312),
 ('jesus', 0.7612189054489136),
 ('behold', 0.7578075528144836),
 ('lowly', 0.7544093132019043),
 ('christ', 0.7520182728767395),
 ('alleluia', 0.7495805621147156),
 ('newborn', 0.7477871775627136),
 ('joy', 0.7450593113899231),
 ('hark', 0.7428459525108337),
 ('allelujah', 0.7406814098358154)]
```

Semantic distance between words

In [40]:
```
def nearest_similarity_cosmul(start1, end1, end2):
    similarities = songs2vec.wv.most_similar_cosmul(
        positive=[end2, start1],
        negative=[end1]
    )
    start2 = similarities[0][0]
    print("{0} es a {1}, lo que {2} es a {3}".format(start1, end1, start2, end2))
```

In [41]:
```
nearest_similarity_cosmul("paris", "france", "alabama")
```

```
paris es a france, lo que georgia es a alabama
```

COMPARACION CON MODELO PREENTRENADO DE GOOGLE

Voy a probar distintas palabras y similitudes con ambos modelos

In [ ]:
```
MODEL = 'GoogleNews-vectors-negative300.bin'
path= get_file(MODEL + '.gz','https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz')
#path = get_file(MODEL + '.gz', 'https://deeplearning4jblob.blob.core.windows.net/resources/wordvectors/%s.gz' % MODE
if not os.path.isdir('dara'):
    os.mkdir('data')


if not os.path.isfile(MODEL):
    with open(MODEL, 'wb') as fout:
        zcat = subprocess.Popen(['zcat'],
                                stdin=open(path),
                                stdout=fout
                               )
        zcat.wait()
```

si ya esta descargado:

In [44]:
```
path = "C:/Users/jhern/Jupyter Notebooks/Analisis Datos no Estructurados/NLP/data/GoogleNews-vectors-negative300.bin
```

In [45]:
```
model = gensim.models.KeyedVectors.load_word2vec_format(datapath(path), binary=True) #UNZIPPED
```

In [46]:
```
#PRE-TRAINED GOOGLE-MODEL
model.most_similar(positive=['beer'])
```

Out[46]:
```
[('beers', 0.8409688472747803),
 ('lager', 0.7733745574951172),
 ('Beer', 0.71753990650177),
 ('drinks', 0.668931245803833),
 ('lagers', 0.6570085883140564),
 ('Yuengling_Lager', 0.6554553508758545),
 ('microbrew', 0.6534324884414673),
 ('Brooklyn_Lager', 0.6501551270484924),
 ('suds', 0.6497017741203308),
 ('brewed_beer', 0.6490240097045898)]
```

In [47]:
```
songs2vec.wv.most_similar("beer")
```

Out[47]:
```
[('gin', 0.8669494390487671),
 ('booze', 0.8502517342567444),
 ('coffee', 0.8398589491844177),
 ('liquor', 0.838886022567749),
 ('drink', 0.8196074962615967),
 ('eggs', 0.814251184463501),
 ('whiskey', 0.8079968690872192),
 ('cigarettes', 0.8002623319625854),
 ('drinking', 0.7971779704093933),
 ('java', 0.7939865589141846)]
```

In [48]:
```
#PRE-TRAINED GOOGLE-MODEL
model.most_similar(positive=['rich'])
```

Out[48]:
```
[('Melamine_nitrogen', 0.6186584234237671),
 ('richer', 0.6159480810165405),
 ('wealthy', 0.5974444150924683),
 ('Scicasts_Resource_Library', 0.5795599222183228),
 ('friend_Francie_Vos', 0.5740315318107605),
 ('Autonomy_Virage_visionary', 0.561326265335083),
 ('richest', 0.5493366122245789),
 ('Hunton_liquids', 0.5271087288856506),
 ('wealthiest', 0.507983922958374),
 ('fabulously_rich', 0.4980970025062561)]
```

In [49]:
```
songs2vec.wv.most_similar("rich")
```

Out[49]:
```
[('fat', 0.814489483833313),
 ('large', 0.7935049533843994),
 ('robbin', 0.788662314414978),
 ('bred', 0.7866466045379639),
 ('lame', 0.7865903377532959),
 ('bastard', 0.7837294340133667),
 ('richer', 0.7777025699615479),
 ('wealthy', 0.7763299345970154),
 ('pig', 0.7671093344688416),
 ('blackjack', 0.7666871547698975)]
```

In [50]:
```
#PRE-TRAINED GOOGLE-MODEL
model.most_similar(positive=['america'])
```

Out[50]:
```
[('american', 0.7169356346130371),
 ('americans', 0.7042055130004883),
 ('europe', 0.6617692708969116),
 ('usa', 0.6611838340759277),
 ('texas', 0.6593319177627563),
 ('india', 0.6589399576187134),
 ('africa', 0.6377725601196289),
 ('mexico', 0.6325021982192993),
 ('england', 0.6323367357254028),
 ('obama', 0.6311532855033875)]
```

In [51]:
```
songs2vec.wv.most_similar("america")
```

Out[51]:
```
[('farmers', 0.7663319110870361),
 ('africa', 0.7178546786308289),
 ('god', 0.716428279876709),
 ('canada', 0.7081302404403687),
 ('england', 0.7072291374206543),
 ('land', 0.7061785459518433),
 ('caesar', 0.7021406888961792),
 ('throwaway', 0.6998007297515869),
 ('aging', 0.6987722516059875),
 ('native', 0.6923732757568359)]
```

In [52]:
```
#PRE-TRAINED GOOGLE-MODEL
def A_is_to_B_as_C_is_to(a, b, c, topn=1):
    a, b, c = map(lambda x:x if type(x) == list else [x], (a, b, c))
    res = model.most_similar(positive=b + c, negative=a, topn=topn)
    if len(res):
        if topn == 1:
            return res[0][0]
        return [x[0] for x in res]
    return None

A_is_to_B_as_C_is_to('man', 'woman', 'king')
```

Out[52]: 'queen'

In [53]:
```
nearest_similarity_cosmul("man", "king", "woman")
```

```
man es a king, lo que fella es a woman
```

In [54]:
```
#PRE-TRAINED GOOGLE-MODEL
A_is_to_B_as_C_is_to('hi', 'bye', 'hello')
```

Out[54]: 'byes'

In [55]:
```
nearest_similarity_cosmul('hi', 'bye', 'hello')
```

```
hi es a bye, lo que mrs es a hello
```

CLUSTERING 2D Y 3D DEL SONG2VEC

CALCULO DEL NORMALISED SUM VECTOR

With the word vector embeddings in place, it is now time to calculate the normalised vector sum of each song. This process can take some time since it has to be done for each of 57,000 songs.

PRIMERO CREAMOS UNA COLUMNA EN EL DATAFRAME QUE CONTENGA LAS LETRAS LIMPIAS SIN PUNTUACION

In [ ]:
```
lyrics_clean=[]
for row in text_corpus:
    lyrics_clean.append(' '.join(row))

songs['lyrics_clean']=lyrics_clean
```

In [ ]:
```
def songVector(row):
    vector_sum = 0
    words = row.lower().split()
    for word in words:
        vector_sum = vector_sum + songs2vec[word]
    vector_sum = vector_sum.reshape(1,-1)
    normalised_vector_sum = sklearn.preprocessing.normalize(vector_sum)
    return normalised_vector_sum


import time
start_time = time.time()

songs['song_vector'] = songs['lyrics_clean'].apply(songVector)
```

t-sne and random song selection

The songs have 50 dimensions each. Application of t-sne is memory intensive and hence it is slightly easier on the computer to use a random sample of the 57,000 songs.

In [58]:
```
song_vectors = []
from sklearn.model_selection import train_test_split

train, test = train_test_split(songs, test_size = 0.9)


for song_vector in train['song_vector']:
    song_vectors.append(song_vector)

train.head(10)
```

Out[58]:

	artist	song	link	text	lyrics_clean	song_vector
1327	Bette Midler	Song Of Bernadette	/b/bette+midler/song+of+bernadette_20016919.html	There was a child named Bernadette. \nI heard...	there was a child named bernadette i heard the...	[[0.22459385, 0.09673958, -0.10339555, -0.1177...
5886	Faith Hill	Fireflies	/f/faith+hill/fireflies_20538286.html	Before you met me I was a fairy princess \nI ...	before you met me i was a fairy princess i cau...	[[0.19841048, 0.10609983, -0.07111432, -0.1403...
26498	Britney Spears	Now Until Forever	/b/britney+spears/now+until+forever_20642430.html	Oh yeah... \nThey say in this world, \nLove ...	oh yeah they say in this world love can t last...	[[0.24957141, 0.09553351, -0.03974519, -0.0885...
40727	Korn	A.D.I.D.A.S.	/k/korn/adidas_20495152.html	Honestly somehow it always seems that I'm drea...	honestly somehow it always seems that i m drea...	[[0.174156, 0.11663693, -0.05611469, -0.111130...
2194	Carly Simon	Make Me Feel Something	/c/carly+simon/make+me+feel+something_20027367...	Make me feel something \nI used to feel \nI ...	make me feel something i used to feel i used t...	[[0.17942503, 0.08656214, -0.11021735, -0.1627...
22481	Zero 7	Spinning	/z/zero+7/spinning_10198561.html	Was it loneliness that brought you here \nBro...	was it loneliness that brought you here broken...	[[0.19964877, 0.047254905, -0.08050153, -0.096...

	artist	song	link	text	lyrics_clean	song_vector
27600	Chris Rea	I Don't Care Anymore	/c/chris+rea/i+dont+care+anymore_20030556.html	I don't care anymore \nAll the things i hear ...	i don t care anymore all the things i hear you...	[[0.20957652, 0.07502813, -0.07251, -0.1348645...
42032	Loretta Lynn	Satin Sheets	/l/loretta+lynn/satin+sheets_20703064.html	Satin sheets to lie on satin pillows to cry on...	satin sheets to lie on satin pillows to cry on...	[[0.24822222, 0.087165944, -0.087976806, -0.13...
44068	Miley Cyrus	Something About Space Dude	/m/miley+cyrus/something+about+space+dude_2110...	[Verse 1] \nSomething in the way you love me ...	verse 1 something in the way you love me somet...	[[0.19431762, 0.09959254, -0.050940055, -0.147...
44748	Nat King Cole	For Sentimental Reasons	/n/nat+king+cole/for+sentimental+reasons_10189...	I love you for sentimental reasons \nI hope y...	i love you for sentimental reasons i hope you ...	[[0.24730659, 0.07641012, -0.09324811, -0.1375...

I had a fairly measly 4gb machine and wasn't able to generate a more accurate model. However, one can play around with the number of iterations, learning rate and other factors to fit the model better. If you have too many dimensions (~300+), it might make sense to use PCA first and then t-sne.

In [59]:

```
X = np.array(song_vectors).reshape((5761, 50))

start_time = time.time()
tsne = sklearn.manifold.TSNE(n_components=2, n_iter=250, random_state=0, verbose=2)

all_word_vectors_matrix_2d = tsne.fit_transform(X)

print("--- %s seconds ---" % (time.time() - start_time))
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 5761 samples in 0.023s...
[t-SNE] Computed neighbors for 5761 samples in 3.128s...
[t-SNE] Computed conditional probabilities for sample 1000 / 5761
[t-SNE] Computed conditional probabilities for sample 2000 / 5761
[t-SNE] Computed conditional probabilities for sample 3000 / 5761
[t-SNE] Computed conditional probabilities for sample 4000 / 5761
[t-SNE] Computed conditional probabilities for sample 5000 / 5761
[t-SNE] Computed conditional probabilities for sample 5761 / 5761
[t-SNE] Mean sigma: 0.044481
[t-SNE] Computed conditional probabilities in 0.320s
[t-SNE] Iteration 50: error = 87.7672958, gradient norm = 0.0149886 (50 iterations in 5.841s)
[t-SNE] Iteration 100: error = 87.6206589, gradient norm = 0.0151304 (50 iterations in 4.690s)
[t-SNE] Iteration 150: error = 87.1604462, gradient norm = 0.0062469 (50 iterations in 4.052s)
[t-SNE] Iteration 200: error = 86.9971542, gradient norm = 0.0207214 (50 iterations in 3.741s)
[t-SNE] Iteration 250: error = 87.1513290, gradient norm = 0.0179854 (50 iterations in 5.162s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 87.151329
[t-SNE] KL divergence after 251 iterations: 17976931348623157081452742373170435679807056752584499659891747680315726
07800285387605895586327668781715404589535143824642343213268894641827684675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658085593321233482747978262041447231687381771809192998812504040261841
24858368.000000
--- 26.981619119644165 seconds ---
```

In [60]:
```
df=pd.DataFrame(all_word_vectors_matrix_2d,columns=['X','Y'])

df.head(10)

train.head()

df.reset_index(drop=True, inplace=True)
train.reset_index(drop=True, inplace=True)
```

Joining two dataframes to obtain each song's corresponding X,Y co-ordinate.

In [61]:
```
two_dimensional_songs = pd.concat([train, df], axis=1)

two_dimensional_songs.head()
```

Out[61]:

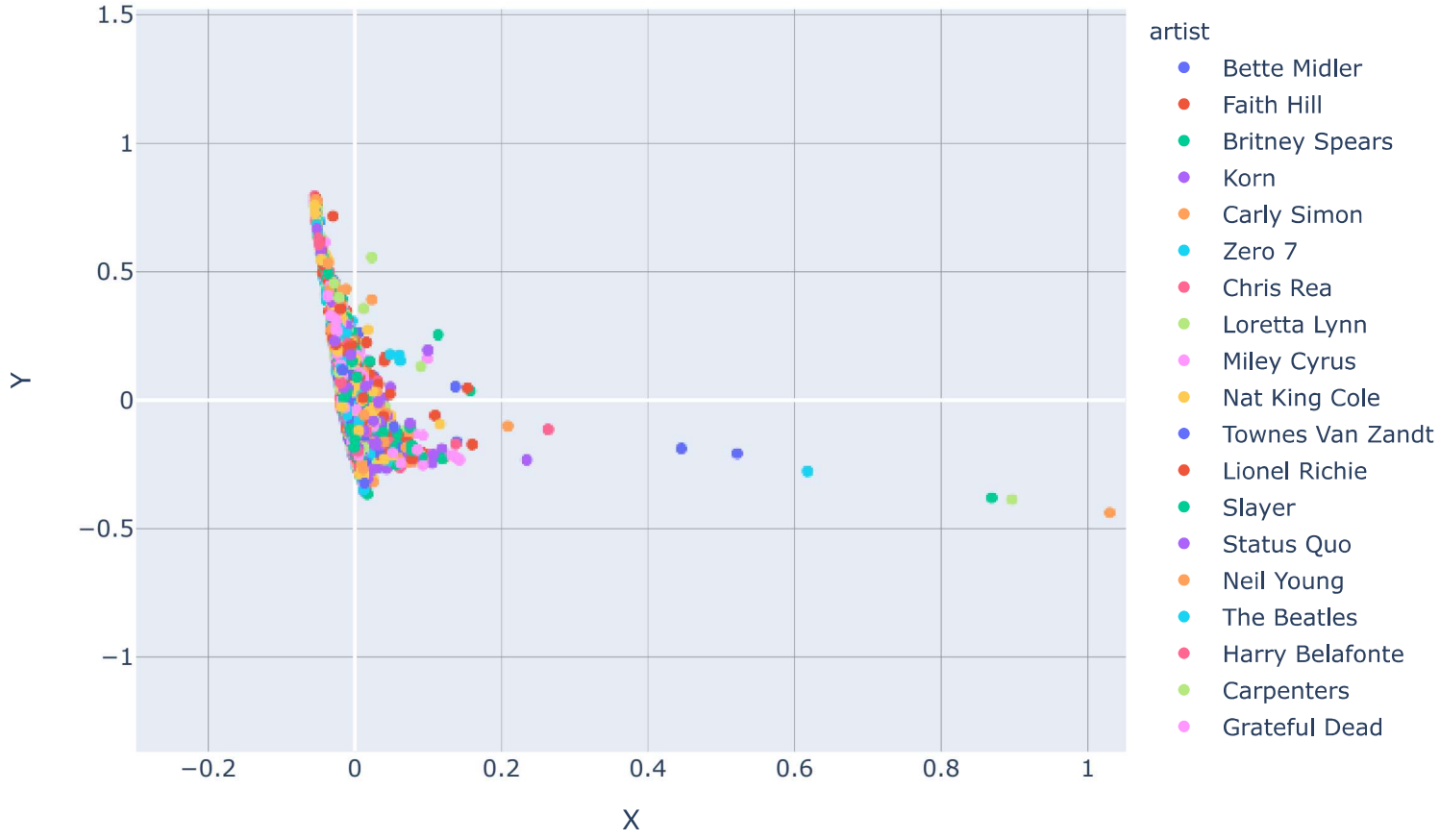
	artist	song	link	text	lyrics_clean	song_vector	X	Y
0	Bette Midler	Song Of Bernadette	/b/bette+midler/song+of+bernadette_20016919.html	There was a child named Bernadette. \nI heard...	there was a child named bernadette i heard the...	[[0.22459385, 0.09673958, -0.10339555, -0.1177...	-0.011570	-0.036562
1	Faith Hill	Fireflies	/f/faith+hill/fireflies_20538286.html	Before you met me I was a fairy princess \nI ...	before you met me i was a fairy princess i cau...	[[0.19841048, 0.10609983, -0.07111432, -0.1403...	-0.012404	0.181835
2	Britney Spears	Now Until Forever	/b/britney+spears/now+until+forever_20642430.html	Oh yeah... \nThey say in this world, \nLove ...	oh yeah they say in this world love can t last...	[[0.24957141, 0.09553351, -0.03974519, -0.0885...	-0.012467	-0.082400
3	Korn	A.D.I.D.A.S.	/k/korn/adidas_20495152.html	Honestly somehow it always seems that I'm drea...	honestly somehow it always seems that i m drea...	[[0.174156, 0.11663693, -0.05611469, -0.111130...	0.000185	-0.143068
4	Carly Simon	Make Me Feel Something	/c/carly+simon/make+me+feel+something_20027367...	Make me feel something \nI used to feel \nI ...	make me feel something i used to feel i used t...	[[0.17942503, 0.08656214, -0.11021735, -0.1627...	-0.008977	-0.062605

Plotting the results

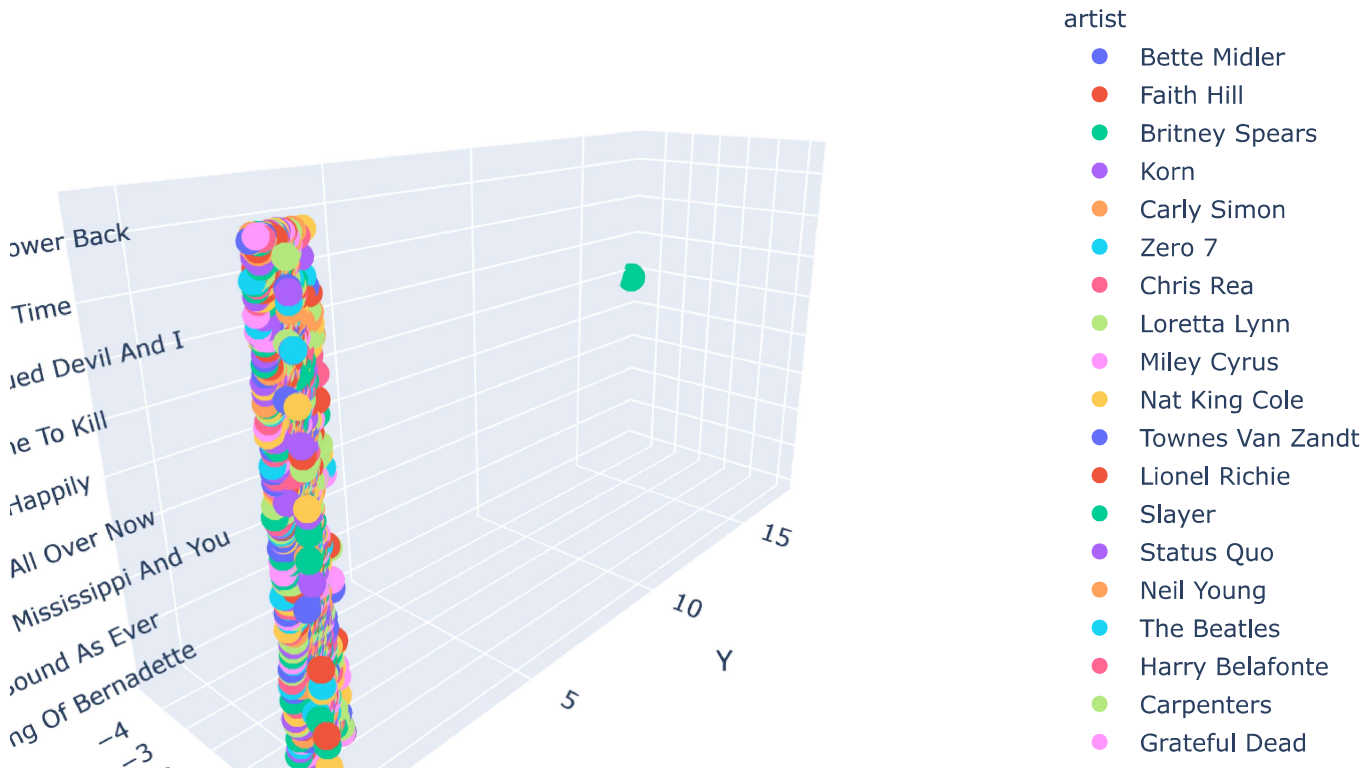
Using plotly, I plotted the results so that it becomes easier to explore similar songs based on their colors and clusters.

In [73]:
```
import plotly
plotly.offline.init_notebook_mode(connected=True)
```

In [74]:
```
import plotly.express as px
fig=px.scatter(two_dimensional_songs, x='X', y='Y', color='artist')
fig.show()
```



In [68]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='song',
                    color='artist')
fig.show()
```

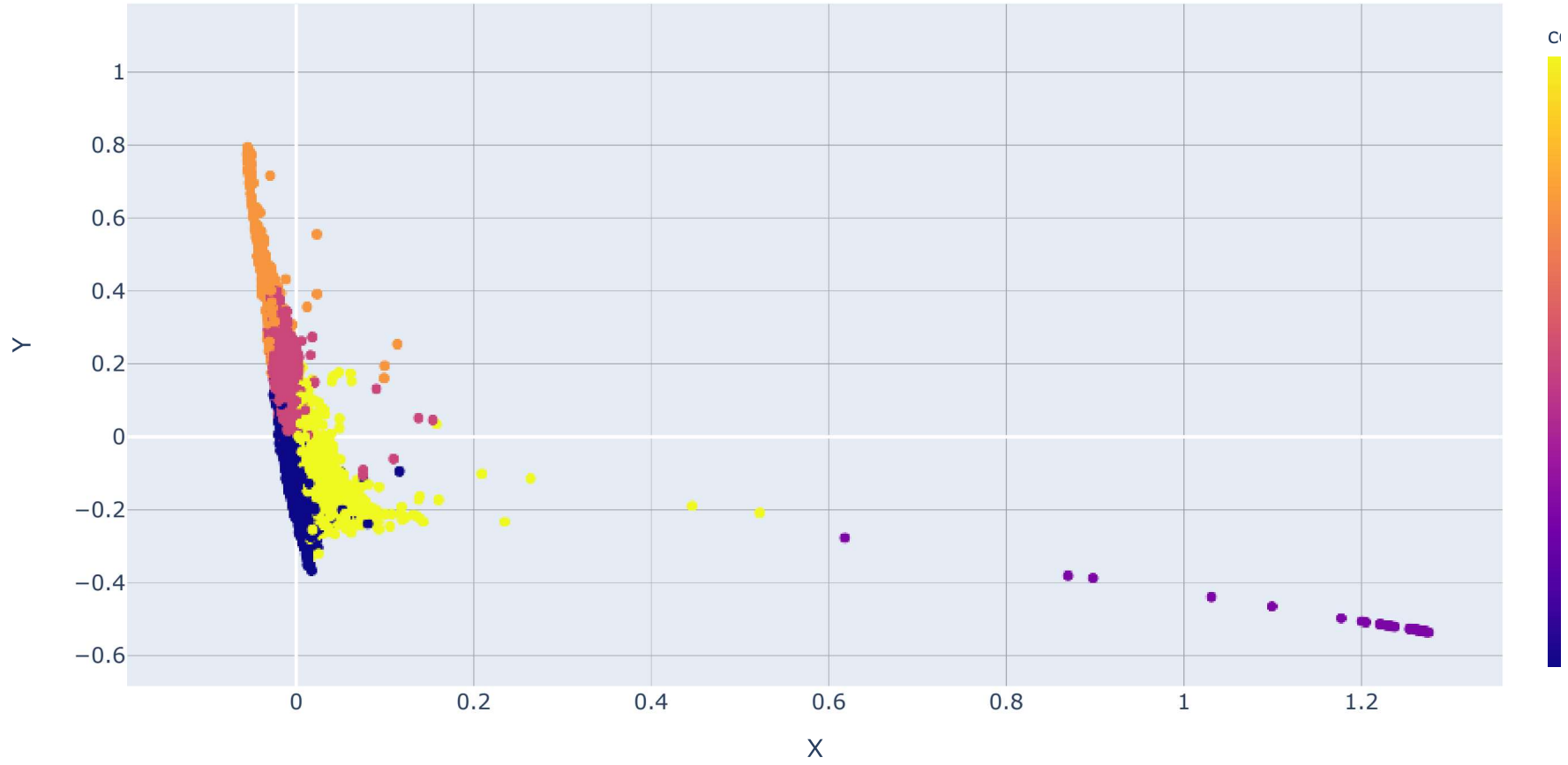


KMEANS

In [75]:

```
from sklearn import cluster
kmeans = cluster.KMeans(n_clusters=5,
                        random_state=42).fit(X)
```

In [70]:
```
import plotly.express as px
fig = px.scatter(two_dimensional_songs, x="X", y="Y",
                 hover_data=['artist', 'song'],
                color=kmeans.labels_)
fig.show()
```



In [71]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='artist',
                    color=kmeans.labels_)
fig.show()
```