WORDS EMBEDDING - SONGS IN ENGLISH CORPUS

In [1]:
```
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
```

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 [3]:
```
songs = pd.read_csv("data/songdata.csv", header=0)
#songs.head()
songs = songs[songs.artist != 'Lata Mangeshkar']
songs.head()
```

Out[3]:

	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.

VOY A EXTRAER MAS DIMENSIONES (100) PARA QUE SEA MÁS PRECISO Y BAJAR EL CONTEXT_SIZE A 5 PARA EVITAR SOBRE-ENTRENAMIENTO

ADEMAS VOY A USAR EL TOKINAZER PARA QUITAR LA PUNTUACION Y VER MEJOR LAS COMPARACINES

In [9]:

```
import nltk
text_corpus = []
for song in songs['text']:
    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 = 100
# 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 = 5


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)
print (len(text_corpus))
```

```
57618
```

AÑADO MAS EPOCHS PARA QUE ENTRENE MEJOR

In [10]:
```
import time
start_time = time.time()



songs2vec.train(text_corpus, total_examples=songs2vec.corpus_count, epochs=5)

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

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

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

```
--- 146.71829867362976 seconds ---
```

In [11]:
```
songs2vec = w2v.Word2Vec.load(os.path.join("trained", "songs2vectors.w2v"))
```

Let's explore our model

Find similar words

In [12]:
```
songs2vec.wv.most_similar("love")
```

Out[12]:
```
[('passionately', 0.72443687915802),
 ('unconditional', 0.7058649063110352),
 ('ohoo', 0.7044984102249146),
 ('peacefulness', 0.701347827911377),
 ('disarm', 0.7007878422737122),
 ('belive', 0.7006666660308838),
 ('forgivin', 0.6966491341590881),
 ('ensure', 0.6950032711029053),
 ('anointing', 0.6941039562225342),
 ('sacrifices', 0.6907321214675903)]
```

In [60]:
```
songs2vec.wv.most_similar("fuck")
```

Out[60]:
```
[('motherfuck', 0.7544417381286621),
 ('nigga', 0.751230001449585),
 ('fuckin', 0.739067018032074),
 ('yall', 0.7256109118461609),
 ('mayne', 0.7208091616630554),
 ('muthafucka', 0.7040141820907593),
 ('bitch', 0.6964313387870789),
 ('sissy', 0.6931982636451721),
 ('niggaare', 0.6921432614326477),
 ('kokane', 0.6913703083992004)]
```

In [61]:
```
songs2vec.wv.most_similar("song")
```

Out[61]:
```
[('melody', 0.8022059202194214),
 ('tune', 0.7796891331672668),
 ('sing', 0.7613018751144409),
 ('songs', 0.7612196207046509),
 ('catchy', 0.7224592566490173),
 ('poem', 0.7222405076026917),
 ('chord', 0.7161662578582764),
 ('elijah', 0.7111837863922119),
 ('singing', 0.7108408808708191),
 ('damo', 0.7071022391319275)]
```

In [62]:
```
songs2vec.wv.most_similar("sweet")
```

Out[62]:
```
[('bittersweet', 0.7285575866699219),
 ('augusta', 0.7050266265869141),
 ('embraceable', 0.695698082447052),
 ('chincherinchee', 0.6954518556594849),
 ('ellie', 0.6925228834152222),
 ('tender', 0.6829348802566528),
 ('sugar', 0.6756668090820312),
 ('tenderly', 0.6692420244216919),
 ('ohm', 0.665321946144104),
 ('cordelia', 0.6604176759719849)]
```

In [63]:
```
songs2vec.wv.most_similar("angel")
```

Out[63]:
```
[('guardian', 0.7370727062225342),
 ('misspent', 0.6530040502548218),
 ('orphan', 0.6419200897216797),
 ('angels', 0.6313167810440063),
 ('aeroplane', 0.6237426996231079),
 ('starlight', 0.6178510785102844),
 ('skyway', 0.6141771078109741),
 ('heaven', 0.6140141487121582),
 ('lady', 0.6109330654144287),
 ('contacting', 0.6030869483947754)]
```

TODAS LAS ANTERIORES LAS HA HECHO MUY BIEN PORQUE ESTÁN RELACIONADAS CON CANCIONES! LAS SIGUIENTES LE VA A COSTAR UN POCO MÁS

In [15]:
```
songs2vec.wv.most_similar("espresso")
```

Out[15]:
```
[('whoring', 0.8959778547286987),
 ('beseen', 0.8920033574104309),
 ('shareing', 0.8912284970283508),
 ('cryyy', 0.8895580172538757),
 ('nicea', 0.8880053758621216),
 ('vindaloo', 0.886662483215332),
 ('detriment', 0.8855420351028442),
 ('seewhat', 0.8851668834686279),
 ('blankest', 0.8851426839828491),
 ('sweetened', 0.8850816488265991)]
```

In [64]:
```
songs2vec.wv.most_similar("computer")
```

Out[64]:
```
[('monopoly', 0.6512895822525024),
 ('technology', 0.6435378789901733),
 ('assassination', 0.6344221234321594),
 ('britannia', 0.6183570623397827),
 ('lending', 0.6160614490509033),
 ('outlet', 0.6156333684921265),
 ('routine', 0.6133609414100647),
 ('program', 0.6128994226455688),
 ('tradition', 0.6104429960250854),
 ('random', 0.6093229055404663)]
```

In [65]:
```
songs2vec.wv.most_similar("data")
```

Out[65]:
```
[('edition', 0.6682997941970825),
 ('processed', 0.6628574132919312),
 ('navigation', 0.6585507392883301),
 ('competion', 0.6580613255500793),
 ('kel', 0.6554206609725952),
 ('suplex', 0.6551501154899597),
 ('ahhhumm', 0.6546577215194702),
 ('bp', 0.6528483629226685),
 ('specs', 0.6528357267379761),
 ('knahmsayin', 0.6522589921951294)]
```

LO MISMO VA A PASAR CON LAS WORDS OUT OF CONTEXT

Words out of context

In [16]:
```
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 b
e passed as a "sequence" type such as list or tuple. Support for non-sequence iterables such as generators is depre
cated as of NumPy 1.16 and will raise an error in the future.
  vectors = vstack(self.word_vec(word, use_norm=True) for word in used_words).astype(REAL)
```

Out[16]:
```
'hate'
```

In [17]:
```
songs2vec.wv.doesnt_match("breakfast milk lunch dinner".split())
```

Out[17]:
```
'milk'
```

In [18]:
```
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_si
milar` (Method will be removed in 4.0.0, use self.wv.most_similar() instead).
  """Entry point for launching an IPython kernel.
```

Out[18]:
```
[('newborn', 0.6602928638458252),
 ('queen', 0.6283557415008545),
 ('alleluia', 0.6270447969436646),
 ('princess', 0.6126868724822998),
 ('shepards', 0.602617084980011),
 ('kings', 0.5959986448287964),
 ('myrrh', 0.5797290802001953),
 ('hosts', 0.5780145525932312),
 ('sages', 0.5758150219917297),
 ('homecoming', 0.5740165710449219)]
```

Semantic distance between words

In [19]:
```
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 [20]:
```
nearest_similarity_cosmul("paris", "france", "alabama")
```

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

In [21]:
```
nearest_similarity_cosmul("paris", "france", "london")
```

```
paris es a france, lo que autumn es a london
```

Con estas diferentes palabras que hemos probado podemos ver como para palabras similares a lo que hay en una canción, lo hace muy bien, pero para palabras extrañas musicalmente hablando (como paises y capitales) le cuesta BASTANTE

CALCULO DE NORMALIZED SUM VECTOR

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

In [22]:
```
lyrics_clean=[]
for row in text_corpus:
    lyrics_clean.append(' '.join(row))

songs['lyrics_clean']=lyrics_clean
```

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.

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)
```

CLUSTERING

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 [25]:
```
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[25]:

	artist	song	link	text	lyrics_clean	song_vector
52699	Talking Heads	Once In A Lifetime	/t/talking+heads/once+in+a+lifetime_20135070.html	And you may find yourself living in a shotgun ...	and you may find yourself living in a shotgun ...	[[0.08573852, 0.011250232, 0.032575868, 0.0172...
16067	Pink Floyd	Cymbaline	/p/pink+floyd/cymbaline_20108653.html	The path you tread is narrow \nAnd the drop i...	the path you tread is narrow and the drop is s...	[[0.097509705, -0.008937475, 0.018912565, -0.0...
8815	Israel	Turn It Around	/i/israel/turn+it+around_20496715.html	Verse One: All things are possible for you all...	verse one all things are possible for you all ...	[[0.045757134, -0.04534775, 0.008854553, -0.01...
33214	Garth Brooks	Maybe	/g/garth+brooks/maybe_20266762.html	Yesterday the odds were stacked in favor of my...	yesterday the odds were stacked in favor of my...	[[0.08446377, 0.03587889, 0.027160928, -0.0156...
35487	Harry Belafonte	Go Down Old Hannah	/h/harry+belafonte/go+down+old+hannah_20204491...	Lyrics: \nOh, we call the sun ol' Hannah \nB...	lyrics oh we call the sun ol hannah blazing on...	[[0.05132534, 0.05194061, 0.011062955, 0.01107...
43282	Matt Redman	Amazing	/m/matt+redman/amazing_20612950.html	A love so undeserved, a gift that's free \nYo...	a love so undeserved a gift that s free you la...	[[0.055255663, -0.064083286, 0.06893965, -0.03...
50561	Red Hot Chili Peppers	Special Secret Song Inside	/r/red+hot+chili+peppers/special+secret+song+i...	Well, my young lady, she lives \nThree houses...	well my young lady she lives three houses away...	[[0.035303008, 0.06256649, 0.025440158, 0.0315...

	artist	song	link	text	lyrics_clean	song_vector
20894	Violent Femmes	Hey Nonny Nonny	/v/violent+femmes/hey+nonny+nonny_20144661.html	Beauty sat bathing by a spring where fairest s...	beauty sat bathing by a spring where fairest s...	[[0.045159437, 0.05390758, 0.02340851, 0.03776...
9461	John Legend	Who Do We Think We Are	/j/john+legend/who+do+we+think+we+are_21059366...	Who do we think we are? \nBaby, tell me, who ...	who do we think we are baby tell me who do we ...	[[0.014882138, 0.015325717, 0.037290893, -0.00...
23913	America	Survival	/a/america/survival_20007090.html	I'll survive you, I will survive you \nWell, ...	i ll survive you i will survive you well it s ...	[[0.062760495, 0.0070115677, 0.037029132, -0.0...

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 [27]:
```
X = np.array(song_vectors).reshape((5761, 100))

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 0.030s...
[t-SNE] Computed neighbors for 5761 samples in 6.100s...
[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: 0.055591
[t-SNE] Computed conditional probabilities in 0.326s
[t-SNE] Iteration 50: error = 87.6070480, gradient norm = 0.0707111 (50 iterations in 5.267s)
[t-SNE] Iteration 100: error = 87.0127182, gradient norm = 0.1085714 (50 iterations in 6.653s)
[t-SNE] Iteration 150: error = 87.5839539, gradient norm = 0.0744651 (50 iterations in 3.543s)
[t-SNE] Iteration 200: error = 87.8476562, gradient norm = 0.0705422 (50 iterations in 3.473s)
[t-SNE] Iteration 250: error = 87.5748138, gradient norm = 0.0620312 (50 iterations in 3.502s)
[t-SNE] KL divergence after 250 iterations with early exaggeration: 87.574814
[t-SNE] KL divergence after 251 iterations: 17976931348623157081452742373170435679807056752584499659891747680315726
07800285387605895586327668781715404589535143824642343213268894641827684675467035375169860499105765512820762454900903893289440758685084551339423045832369032229481658085593321233482747978262041447231687381771809192998812504040261841
24858368.000000
--- 28.89676022529602 seconds ---
```

In [28]:
```
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 [29]:
```
two_dimensional_songs = pd.concat([train, df], axis=1)

two_dimensional_songs.head()
```

Out[29]:

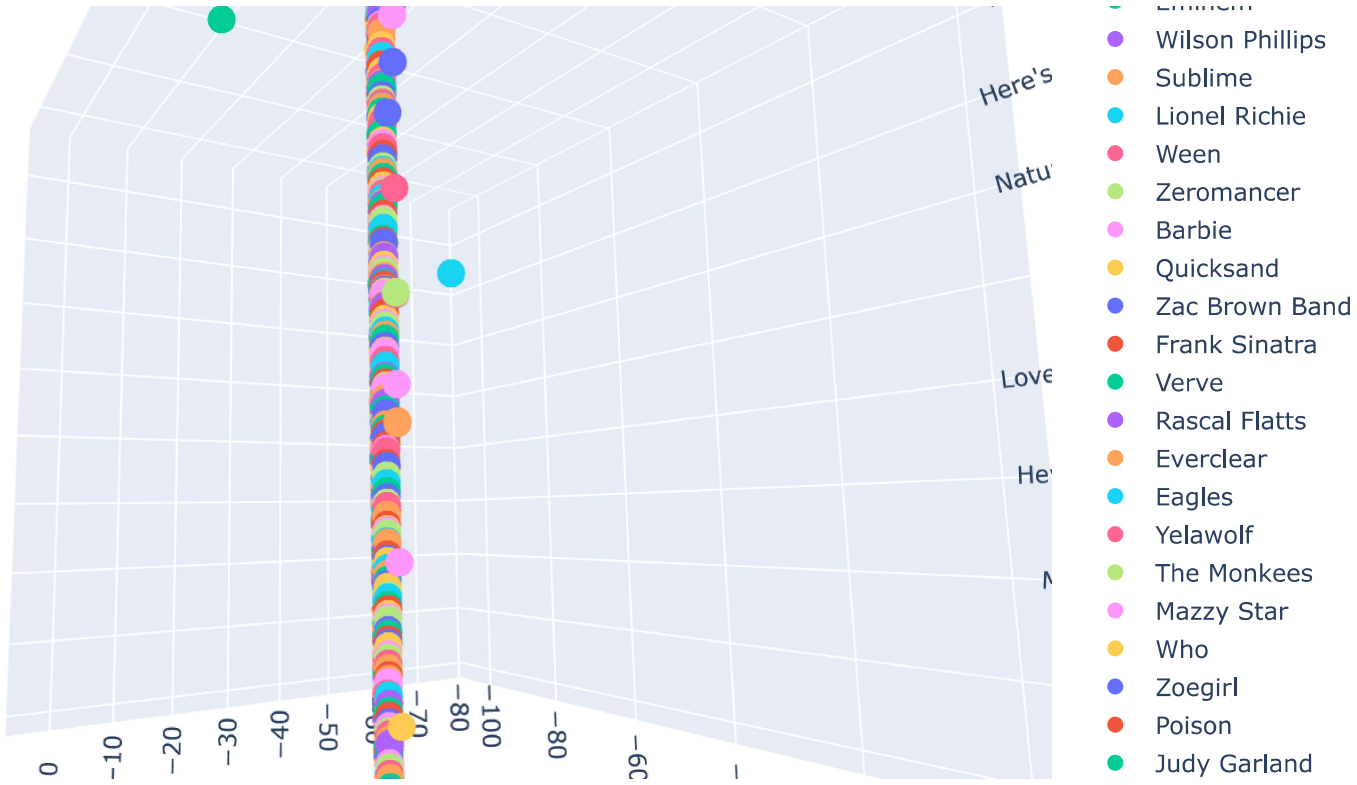
	artist	song	link	text	lyrics_clean	song_vector	X	Y
0	Talking Heads	Once In A Lifetime	/t/talking+heads/once+in+a+lifetime_20135070.html	And you may find yourself living in a shotgun ...	and you may find yourself living in a shotgun ...	[[0.08573852, 0.011250232, 0.032575868, 0.0172...	-0.022574	-0.124840
1	Pink Floyd	Cymbaline	/p/pink+floyd/cymbaline_20108653.html	The path you tread is narrow \nAnd the drop i...	the path you tread is narrow and the drop is s...	[[0.097509705, -0.008937475, 0.018912565, -0.0...	-0.028421	-0.151639
2	Israel	Turn It Around	/i/israel/turn+it+around_20496715.html	Verse One: All things are possible for you all...	verse one all things are possible for you all ...	[[0.045757134, -0.04534775, 0.008854553, -0.01...	0.003565	-0.026752
3	Garth Brooks	Maybe	/g/garth+brooks/maybe_20266762.html	Yesterday the odds were stacked in favor of my...	yesterday the odds were stacked in favor of my...	[[0.08446377, 0.03587889, 0.027160928, -0.0156...	0.009596	-0.036501
4	Harry Belafonte	Go Down Old Hannah	/h/harry+belafonte/go+down+old+hannah_20204491...	Lyrics: \nOh, we call the sun ol' Hannah \nB...	lyrics oh we call the sun ol hannah blazing on...	[[0.05132534, 0.05194061, 0.011062955, 0.01107...	0.007779	0.023304

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 [54]:
```
import plotly.express as px
fig=px.scatter(two_dimensional_songs, x='X', y='Y', color='artist')
fig.show()
```

In [32]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='song',
                    color='artist')
fig.show()
```



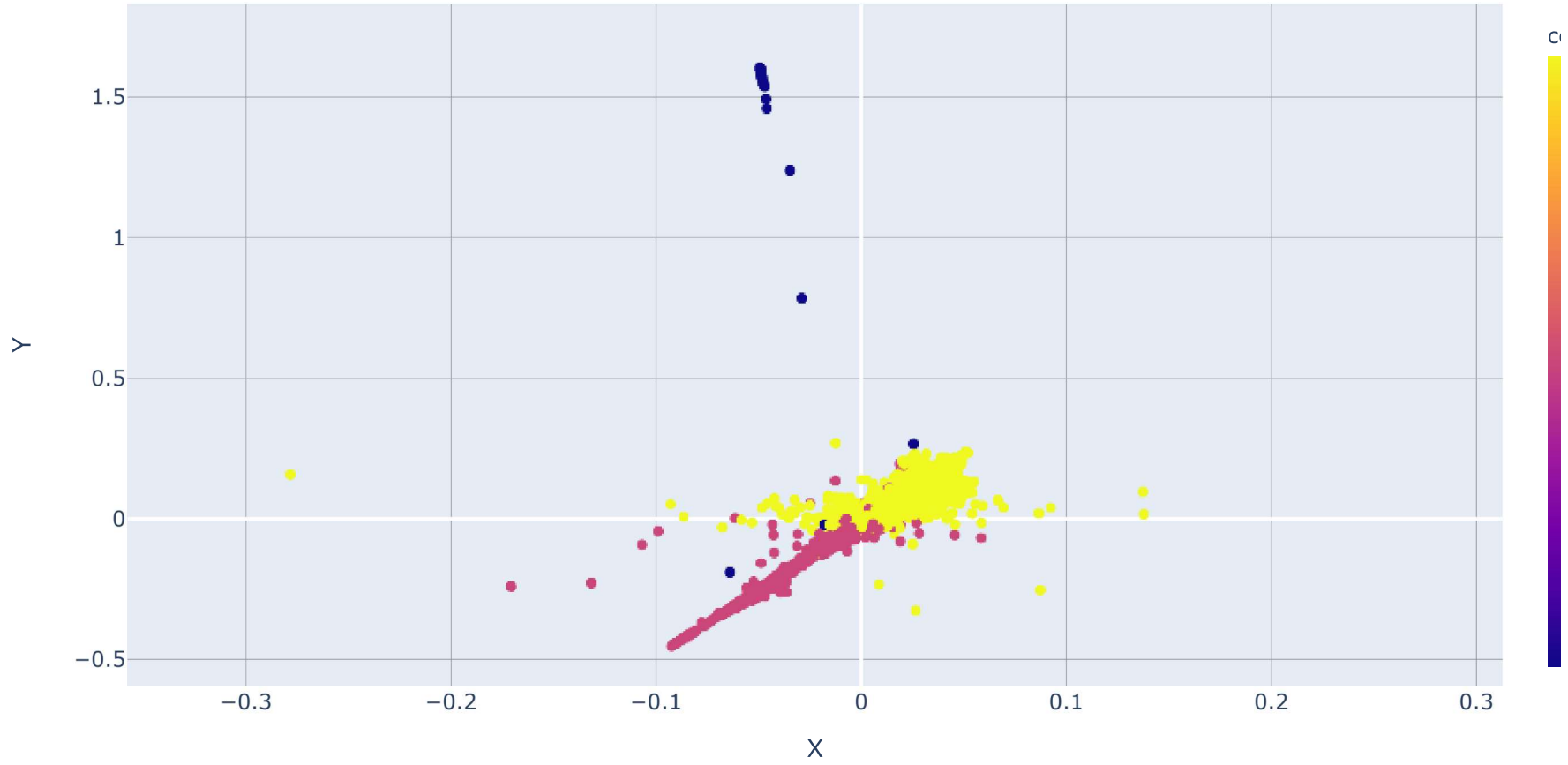
CLUSTERING CON KMEANS

In [36]:

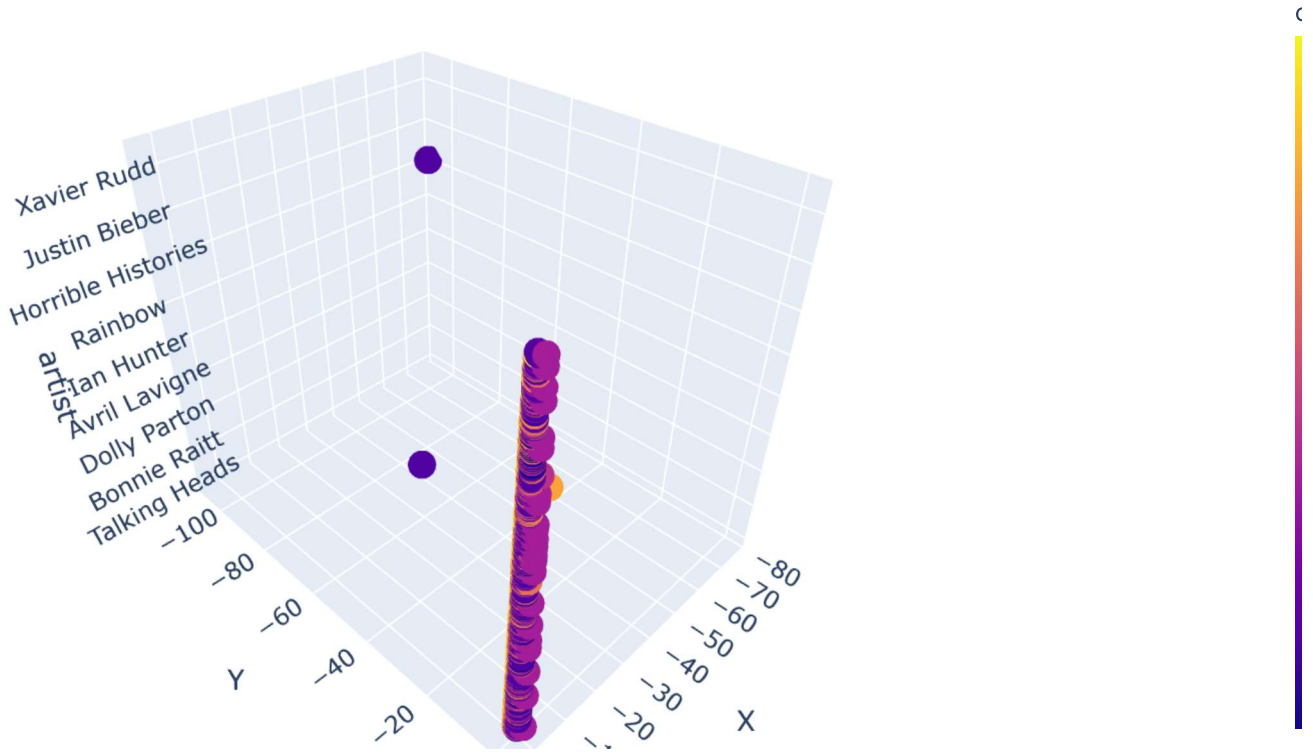
```
from sklearn import cluster
X = np.array(song_vectors).reshape((5761, 100))

kmeans = cluster.KMeans(n_clusters=3,
                        random_state=42).fit(X)
```

In [66]:
```
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()
```



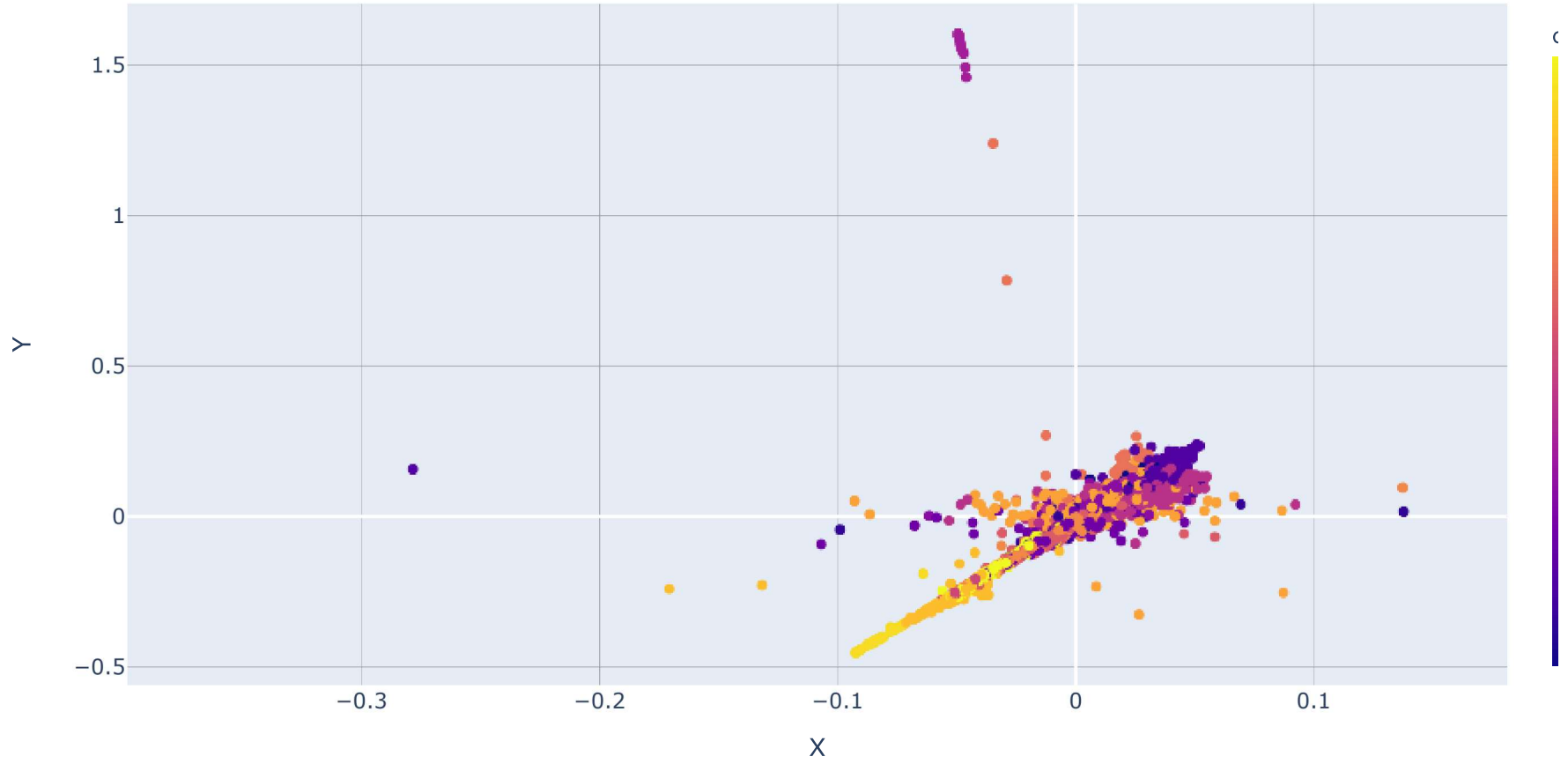
PINTANDO LOS DISTINTOS CLUSTERS DE KMEANS PODEMOS ENCONTRAR QUE CANCIONES SIMILARES EN TEMATICA SE AGRUPAN JUNTAS

- Por ejemplo las canciones de amor suelen estar en el cluster amarilo (Adam Sandler - Best Friend, whiteny Houston - For the love of you)
- En el azul hay música más"independiente" que quizá creando más clusters podriamos categorizar mejor (probamos a continuacion)
- En el rosa parece haber música más energética como raps y rocks

KMEANS CON 15 CLUSTERS

In [67]:
```
kmeans = cluster.KMeans(n_clusters=15,
                        random_state=42).fit(X)
```

In [68]:
```
import plotly.express as px
fig = px.scatter(two_dimensional_songs, x="X", y="Y",
                 hover_data=['artist', 'song'],
                color=kmeans.labels_)
fig.show()
```



AQUI PODEMOS VER QUE LA AGRUPACION TIENE SENTIDO CON RESPECTO A LAS CARACTERISTICAS X E Y. EL COLOR PASA DE CLARO A OSCURO FORMANDO DISTINTOS CLUSTERS

ANALIZANDO LAS LETRAS DE LAS CANCIONES DE CADA CLUSTER SE PODRIA VER QUE MUCHAS PALABRAS SE REPITEN PARA LAS CANCIONES DE UN MISMO CLUSTER