The aim of this notebook is to make use of the word2vec model to find similar songs

WORD2VEC - Exploracion con un Corpus de canciones en Español usando tokenizer y stopwords para crear el corpus

In [ ]:
```
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
```

Cargamos las canciones en ingles y ponemos el index a la columna 0 (id en el csv)

In [ ]:
```
songs = pd.read_csv("data/corpus_esp.csv", header=0, index_col = 0)
songs.head()
```

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.

Usamos tokenizer y stopwords para tener un mejor vocabulario

In [ ]:

```
import nltk
from nltk.corpus import stopwords

text_corpus = []
for song in songs['letra']:
    #words = song.lower()
    #words = re.findall(r'\w+', words) #LIMPIAMOS PARA COGER SOLO LAS PALABRAS

    tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+') #para dividir por words y quitar puntuacion
    lower_case = song.lower()
    tokens_sin_puntuacion = tokenizer.tokenize(lower_case)
    tokens = [i for i in tokens_sin_puntuacion if (len(i)>1) and i not in stopwords.words('spanish')] #quitamos stop

    text_corpus.append(tokens)



# 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)
print (len(text_corpus))
```

Entrenamiento:

In [ ]:
```
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))
```

In [33]:
```
songs2vec = w2v.Word2Vec.load(os.path.join("trained", "songs2vectors.w2v"))
```

Let's explore our model

Find similar words

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

Out[34]:
```
[('platónico', 0.8963768482208252),
 ('odio', 0.8941953778266907),
 ('rencor', 0.8910723328590393),
 ('dolor', 0.8837886452674866),
 ('honor', 0.8661893606185913),
 ('cariño', 0.8650456666946411),
 ('amistad', 0.8645437359809875),
 ('relación', 0.8637412786483765),
 ('desamor', 0.8608136177062988),
 ('pasión', 0.8603465557098389)]
```

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

Out[35]:
```
[('pavo', 0.8948357105255127),
 ('cerda', 0.8900874853134155),
 ('mierdas', 0.8885449171066284),
 ('industria', 0.8838553428649902),
 ('hostia', 0.8828763961791992),
 ('apesta', 0.8812952041625977),
 ('feria', 0.8807495832443237),
 ('hablando', 0.8782275915145874),
 ('menuda', 0.8773846626281738),
 ('tele', 0.876879870891571)]
```

In [36]:
```
songs2vec.wv.most_similar("duerme")
```

Out[36]:
```
[('despierta', 0.9428236484527588),
 ('sueña', 0.9177895188331604),
 ('cruza', 0.9149852395057678),
 ('nana', 0.9128752946853638),
 ('enciende', 0.9128324389457703),
 ('apagada', 0.911525309085846),
 ('apaga', 0.9110377430915833),
 ('sueñas', 0.9108179807662964),
 ('abraza', 0.910004734992981),
 ('calienta', 0.9098016619682312)]
```

In [37]:
```
songs2vec.wv.most_similar("cancion")
```

Out[37]:
```
[('relato', 0.956216037273407),
 ('tropiezo', 0.9552488327026367),
 ('manifiesto', 0.9548647403717041),
 ('libero', 0.9539966583251953),
 ('narro', 0.9535489082336426),
 ('resucito', 0.9528268575668335),
 ('inmortal', 0.9515603184700012),
 ('latido', 0.9514897465705872),
 ('argumento', 0.9514478445053101),
 ('narra', 0.9513541460037231)]
```

In [38]:
```
songs2vec.wv.most_similar("ángel")
```

Out[38]:
```
[('demonio', 0.9419563412666321),
 ('lloré', 0.9247255921363831),
 ('árbol', 0.921562135219574),
 ('pájaro', 0.9153767228126526),
 ('maldito', 0.9118456840515137),
 ('eva', 0.9115332365036011),
 ('fruto', 0.9070749878883362),
 ('llevó', 0.904128909111023),
 ('amada', 0.9040050506591797),
 ('rompí', 0.9031476974487305)]
```

Words out of context

In [39]:
```
songs2vec.wv.doesnt_match("amor corazon television".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[39]:
```
'television'
```

In [40]:
```
songs2vec.wv.doesnt_match("hola adios hora".split())
```

Out[40]:
```
'adios'
```

In [41]:
```
songs2vec.wv.doesnt_match("cerveza cafe árbol".split())
```

Out[41]:
```
'cerveza'
```

In [42]:
```
songs2vec.most_similar(positive=['mujer', 'reina'], negative=['hombre'])
#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[42]:
```
[('cama', 0.8692028522491455),
 ('niña', 0.8367359042167664),
 ('dama', 0.835940957069397),
 ('jugada', 0.8302148580551147),
 ('bebé', 0.8288430571556091),
 ('morena', 0.825384795665741),
 ('piernas', 0.8248263001441956),
 ('nevera', 0.8195244073867798),
 ('llamas', 0.8192764520645142),
 ('llama', 0.8186324834823608)]
```

Semantic distance between words

In [43]:
```
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 [44]:
```
nearest_similarity_cosmul("paris", "francia", "españa") #jeje mas o menos
```

```
paris es a francia, lo que barna es a españa
```

In [45]:
```
nearest_similarity_cosmul("beso", "amor", "odio") #se ha liado
```

```
beso es a amor, lo que abrazo es a odio
```

In [46]:
```
nearest_similarity_cosmul("ojo", "cabeza", "pie") #porque no son palabras "muy" de canciones
```

```
ojo es a cabeza, lo que caballo es a pie
```

In [47]:
```
nearest_similarity_cosmul("verso", "poema", "cancion") #mas o menos
```

```
verso es a poema, lo que sentimiento es a cancion
```

Funciona bastante bien para palabras recurrentes en canciones!! Si nos salimos de estas palabras comunes le cuesta un poco mas

CLUSTERING

PRIMERO CREAMOS UNA COLUMNA EN EL DATAFRAME QUE CONTENGA LAS LETRAS LIMPIAS SIN PUNTUACION PARA PODER COMPARAR CON EL VOCABULARIO

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

songs['letras_limpias']=letras_limpias
```

In [ ]:
```
#print(songs2vec['amor'])
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['letras_limpias'].apply(songVector)
```

t-sne and random song selection

In [ ]:
```
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)
```

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 [ ]:
```
X = np.array(song_vectors).reshape((932, 50)) #5761

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))
```

In [50]:
```
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 [51]:
```
two_dimensional_songs = pd.concat([train, df], axis=1)

two_dimensional_songs.head()
```

Out[51]:

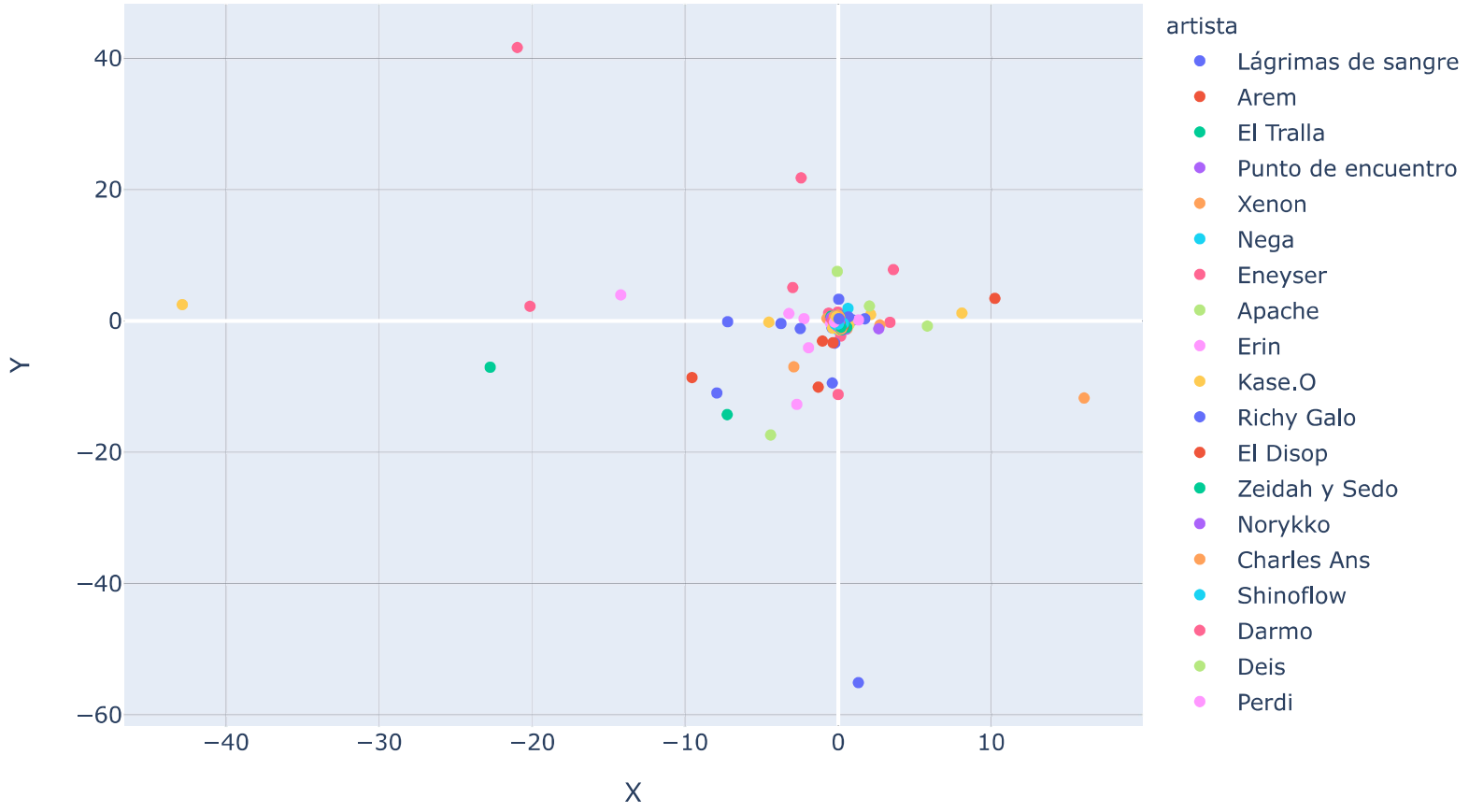
	artista	cancion	album	letra	anyo	visitas	letras_limpias	song_vector	X	Y
0	Lágrimas de sangre	A ver si lo pilláis ya	Vértigo	[Microbio]\nLágrimas de sangre, inclasificable...	2019	1303	microbio lágrimas sangre inclasificable estilo...	[[-0.20203725, 0.045902506, -0.0776025, -0.284...	-0.067109	-0.732819
1	Arem	Apuntan alto	Tan solo a solas	Capta en este track todo aquello que te digo,\...	2013	665	capta track aquello digo rimas entrelazadas ri...	[[-0.1956612, 0.036341548, -0.09813297, -0.297...	0.197000	-0.296885
2	El Tralla	Esta ciudad	Esta ciudad	Esta ciudad, el templo de los fariseos\nQuiere...	2013	731	ciudad templo fariseos quieren cabeza cuernos ...	[[-0.19008844, 0.02465769, -0.07188727, -0.247...	0.165761	-0.248713
3	Punto de encuentro	Agua envenenada	Ariete	[Tcap Leviatan]\nCuando entras yo me salgo\nSo...	2016	347	tcap leviatan entras salgo ave talgo llega ins...	[[-0.1988243, 0.032173447, -0.07440752, -0.282...	0.131691	-0.339394
4	Xenon	Alcatraz	Sin álbum	Yo sé que el tiempo pasa y me echas de menos,\...	2018	1306	sé tiempo pasa echas menos verlo tiempo pone l...	[[-0.17864558, 0.02122039, -0.107486874, -0.23...	0.081246	0.592924

Plotting the results

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

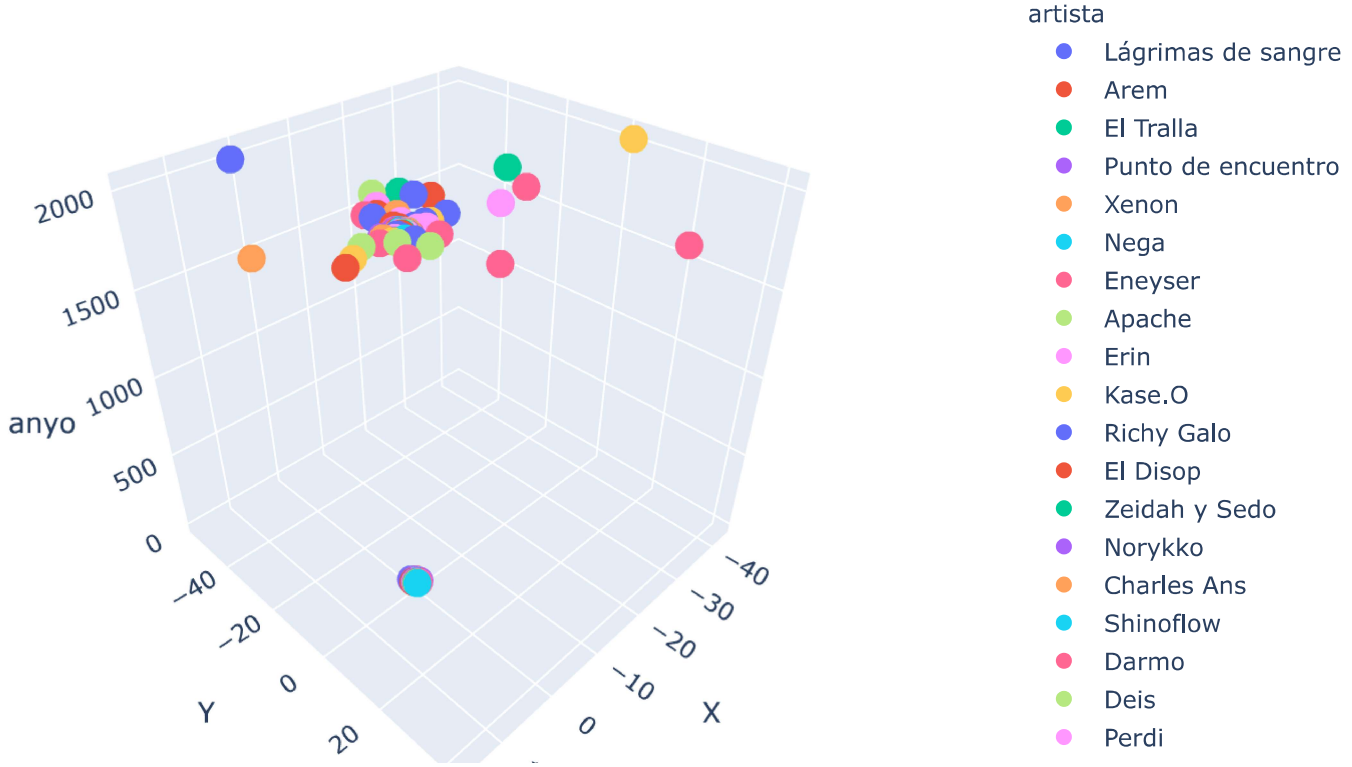
In [52]:
```
import plotly
plotly.offline.init_notebook_mode(connected=True)
```

In [53]:
```
import plotly.express as px
fig=px.scatter(two_dimensional_songs, x='X', y='Y', color='artista')
fig.show()
```

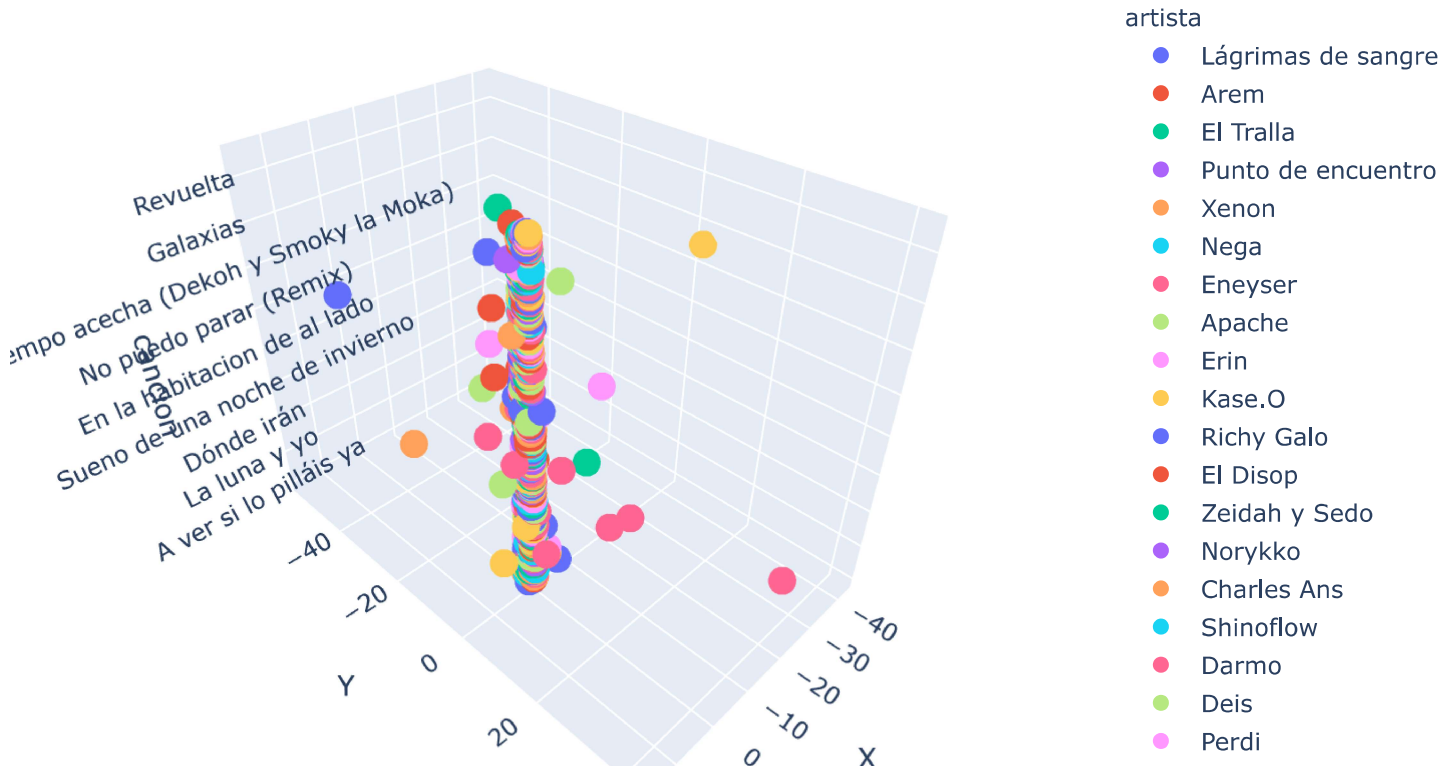


3D

In [54]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='anyo',
                    color='artista')
fig.show()
```



In [55]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='cancion',
                    color='artista')
fig.show()
```

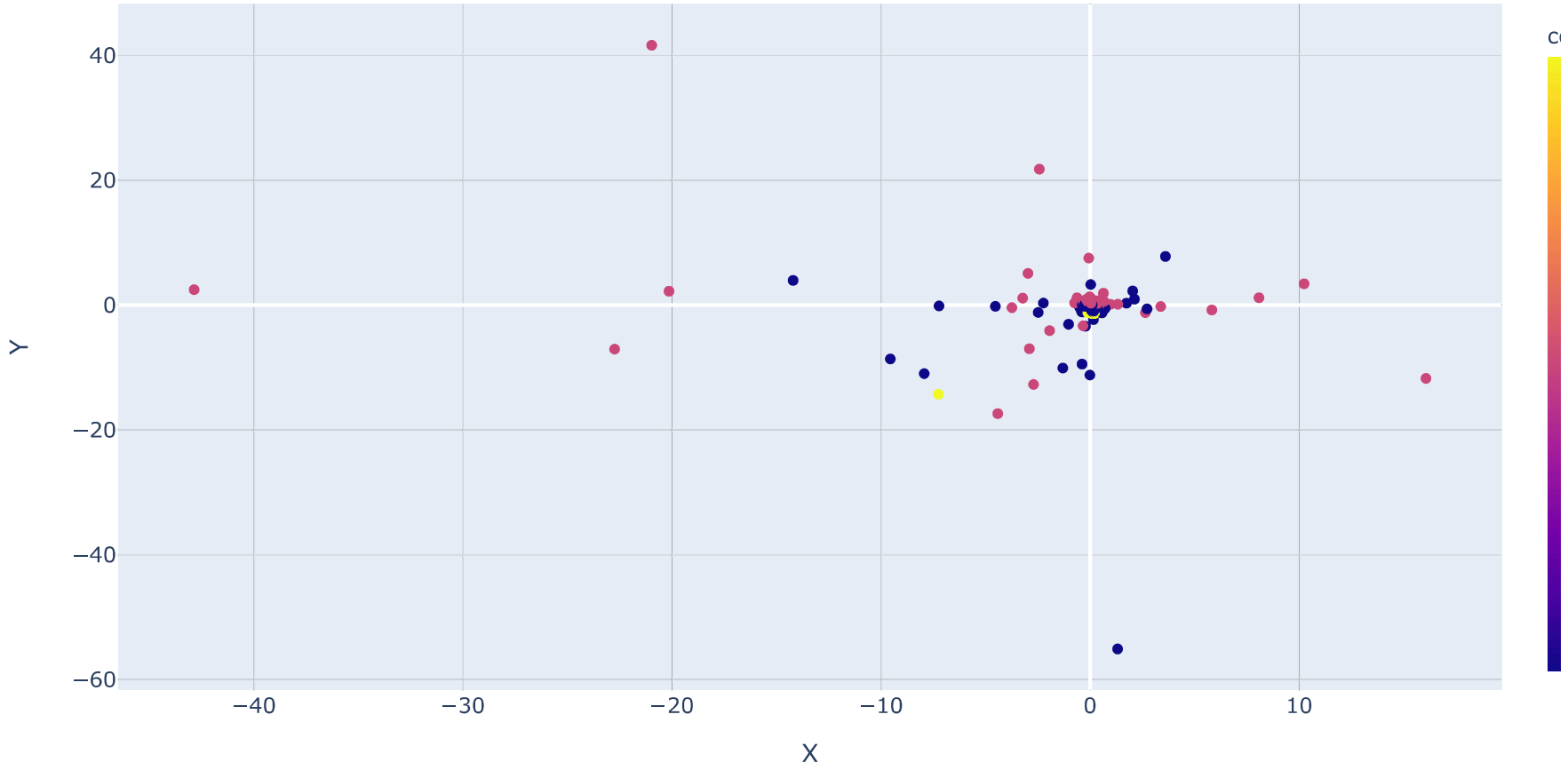


KMEANS

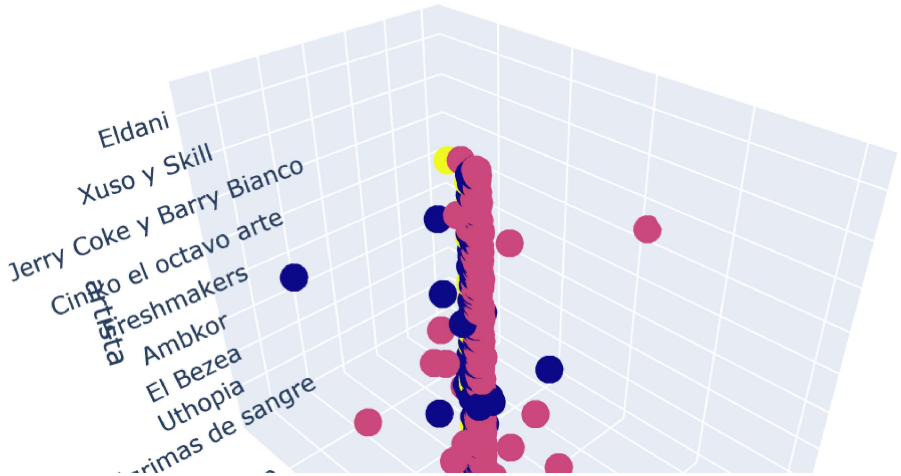
In [60]:

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

In [61]:
```
import plotly.express as px
fig = px.scatter(two_dimensional_songs, x="X", y="Y",
                 hover_data=['artista', 'cancion'],
                color=kmeans.labels_)
fig.show()
```



In [62]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='artista',
                    color=kmeans.labels_)
fig.show()
```



In [63]:
```
import plotly.express as px
fig = px.scatter_3d(two_dimensional_songs, x='X', y='Y', z='anyo',
                    color=kmeans.labels_)
fig.show()
```