In [209]:

```
import pandas as pd
import numpy as np
import os
import re
import matplotlib.pyplot as plt
import nltk
from nltk import RegexpTokenizer
from nltk.stem.snowball import SnowballStemmer
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from wordcloud import WordCloud
%matplotlib inline
```

In [2]:

```
import warnings
warnings.filterwarnings('ignore')
```

Prepare lyrics data

In [137]:

```
# Get lyrics data
song_lyrics = pd.read_csv('merge_w_lyrics_full.csv')[['title','artist_x','lyrics

# Clean lyrics
song_lyrics['lyrics'] = song_lyrics['lyrics'].str.replace('\n', ' ').replace('.'

song_lyrics
```

Out[137]:

	title	artist_x	lyrics	hit
0	10,000 Reasons (Bless the Lord)	Matt Redman	Bless the Lord, O my soul, O my soul Worship H...	0.0
1	100 Proof	Kellie Pickler	Ain't no rain as cold as the look she just gav...	0.0
2	101	Alicia Keys	You used to the sound of a heart that's breaki...	0.0
3	110%	Jessie Ware	Carving my initials on your forehead Now if y...	0.0
4	1313	The Big Pink	Convey your thoughts, translate them well Say ...	0.0
...	...	...	...	...
20977	Danny Phantom	Trippie Redd	Yeah, first I wanna show you some of this dama...	1.0
20978	What's Wrong	Rod Wave	You know what they gon' say When the world tur...	1.0
20979	Demon Time	Trippie Redd	Ha (Ha-ha) Demon time, nigga (Yeah) Gang, uh (...	1.0
20980	Tick Tock	Young Thug	Yeah, Spider, yeah (Slime) Okay, shit, I just...	1.0
20981	Time Heals	Rod Wave	Ayo Keyz, chop this up, for the one time MarsG...	1.0

20982 rows × 4 columns

In [200]:

```python
# tokenize and stem lyrics
tokenizer = RegexpTokenizer(r'\w+')

# Instantiate stemmer
stemmer = SnowballStemmer('english')

# List to append stemmed words
stemmed = []
# List to append tokenized words
tokenized = []

# Create a for loop to iterate through all the rows in specific column
for i in song_lyrics['lyrics']:

    # Converting lyrics text to tokens
    tokens = tokenizer.tokenize(i.lower())
    tokenized.append(tokens)

    # Stemming all tokens
    stems = [stemmer.stem(token) for token in tokens]
    # Appending stems to stemmed list
    stemmed.append(stems)

# Creating new dataframe columns
song_lyrics['tokenized_lyrics'] = [' '.join(i) for i in tokenized]
song_lyrics['stemmed_lyrics'] = [' '.join(i) for i in stemmed]

song_lyrics
```

Out[200]:

	title	artist_x	lyrics	hit	label_3	tokenized_lyrics	stemmed_lyrics
0	10,000 Reasons (Bless the Lord)	Matt Redman	Bless the Lord, O my soul, O my soul Worship H...	0.0	0	bless the lord o my soul o my soul worship his...	bless the lord o my soul o my soul worship his...
1	100 Proof	Kellie Pickler	Ain't no rain as cold as the look she just gav...	0.0	0	ain t no rain as cold as the look she just gav...	ain t no rain as cold as the look she just gav...
2	101	Alicia Keys	You used to the sound of a heart that's breaki...	0.0	0	you used to the sound of a heart that s breaki...	you use to the sound of a heart that s break i...
3	110%	Jessie Ware	Carving my initials on your forehead Now if y...	0.0	0	carving my initials on your forehead now if yo...	carv my initi on your forehead now if you re n...
4	1313	The Big Pink	Convey your thoughts, translate them well Say ...	0.0	0	convey your thoughts translate them well say t...	convey your thought translat them well say tho...
...	...	...	...	...	...	...	...
20977	Danny Phantom	Trippie Redd	Yeah, first I wanna show you some of this dama...	1.0	1	yeah first i wanna show you some of this damag...	yeah first i wanna show you some of this damag...

	title	artist_x	lyrics	hit	label_3	tokenized_lyrics	stemmed_lyrics
20978	What's Wrong	Rod Wave	You know what they gon' say When the world tur...	1.0	1	you know what they gon say when the world turn...	you know what they gon say when the world turn...
20979	Demon Time	Trippie Redd	Ha (Ha-ha) Demon time, nigga (Yeah) Gang, uh (...	1.0	1	ha ha ha demon time nigga yeah gang uh gang ga...	ha ha ha demon time nigga yeah gang uh gang ga...
20980	Tick Tock	Young Thug	Yeah, Spider, yeah (Slime) Okay, shit, I just...	1.0	1	yeah spider yeah slime okay shit i just woke u...	yeah spider yeah slime okay shit i just woke u...
20981	Time Heals	Rod Wave	Ayo Keyz, chop this up, for the one time MarsG	1.0	2	ayo keyz chop this up for the one time	ayo keyz chop this up for the one time

In [16]:

```python
# CountVectorizer
cv = CountVectorizer(input='content',
                     stop_words='english',
                     lowercase=True,
                     strip_accents='ascii',
                     max_features=800)

dtm_cv = cv.fit_transform(song_lyrics['stemmed_lyrics'])

df_cv = pd.DataFrame(dtm_cv.toarray(),
                      columns=cv.get_feature_names(),
                      index=song_lyrics['title'])

# Filter non-alphabetic features
df_cv = df_cv[[c for c in df_cv.columns if c.isalpha()]]

df_cv
```

Out[16]:

	abl	abov	act	action	afraid	age	ago	ah	ain	air	...	wrong	ya	ye	yeah	year
title																
10,000 Reasons (Bless the Lord)	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1
100 Proof	0	0	0	0	0	0	0	0	2	0	...	0	0	0	1	0
101	0	0	0	0	0	0	0	0	0	0	...	0	0	0	1	0
110%	0	0	0	0	0	0	0	0	0	0	...	1	0	0	0	0
1313	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
Danny Phantom	0	0	0	0	0	0	0	0	0	0	...	0	0	0	32	0
What's Wrong	0	0	1	0	0	0	0	0	3	0	...	2	0	0	23	0
Demon Time	0	0	0	0	0	0	0	0	5	0	...	0	0	0	18	0
Tick Tock	0	0	0	0	0	0	0	0	0	0	...	0	0	0	18	1
Time Heals	0	0	0	0	0	0	0	0	2	0	...	1	0	0	6	0

20982 rows × 793 columns

In [17]:

```python
# TfidfVectorizer
tfidf = TfidfVectorizer(input='content',
                        stop_words='english',
                        lowercase=True,
                        strip_accents='ascii',
                        max_features=800)

dtm_tfidf = tfidf.fit_transform(song_lyrics['stemmed_lyrics'])

df_tfidf = pd.DataFrame(dtm_tfidf.toarray(),
                         columns=tfidf.get_feature_names(),
                         index=song_lyrics['title'])

df_tfidf = df_tfidf[[c for c in df_tfidf.columns if c.isalpha()]]

df_tfidf
```

Out[17]:

	abl	abov	act	action	afraid	age	ago	ah	ain	air	...	wrong	ya
title													
10,000 Reasons (Bless the Lord)	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.000000	0.0
100 Proof	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.081779	0.0	...	0.000000	0.0
101	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.000000	0.0
110%	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.025611	0.0
1313	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.000000	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
Danny Phantom	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.000000	0.0
What's Wrong	0.0	0.0	0.048145	0.0	0.0	0.0	0.0	0.0	0.085281	0.0	...	0.079602	0.0
Demon Time	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.087408	0.0	...	0.000000	0.0
Tick Tock	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	...	0.000000	0.0
Time Heals	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.060140	0.0	...	0.042102	0.0

20982 rows × 793 columns

In [19]:

```python
sum(df_tfidf.columns == df_cv.columns)
```

Out[19]:

793

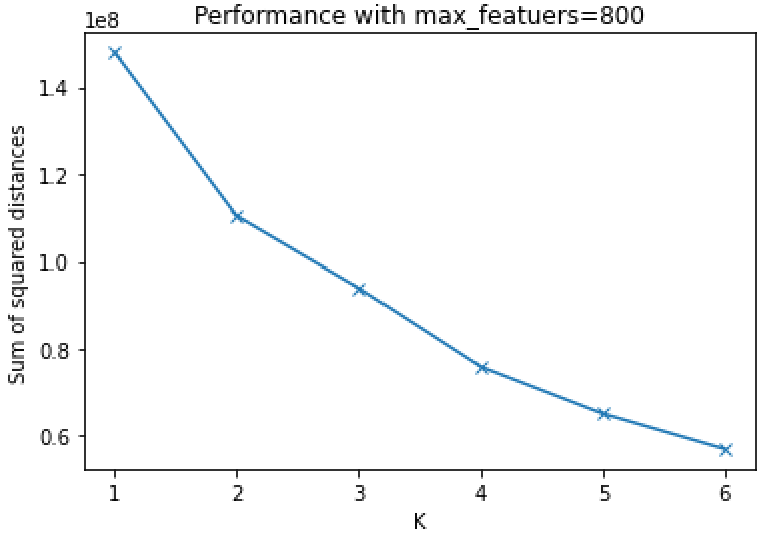
K-means

In [31]:

```
sse = []
K = range(1,7)

for k in K:
    kmeans = KMeans(n_clusters = k).fit(df_cv)
    sse.append(kmeans.inertia_)

plt.plot(K, sse, 'x-')
plt.xlabel('K')
plt.ylabel('Sum of squared distances')
plt.title('Performance with max_featuers=800')
```

Out[31]:

```
Text(0.5, 1.0, 'Performance with max_featuers=800')
```



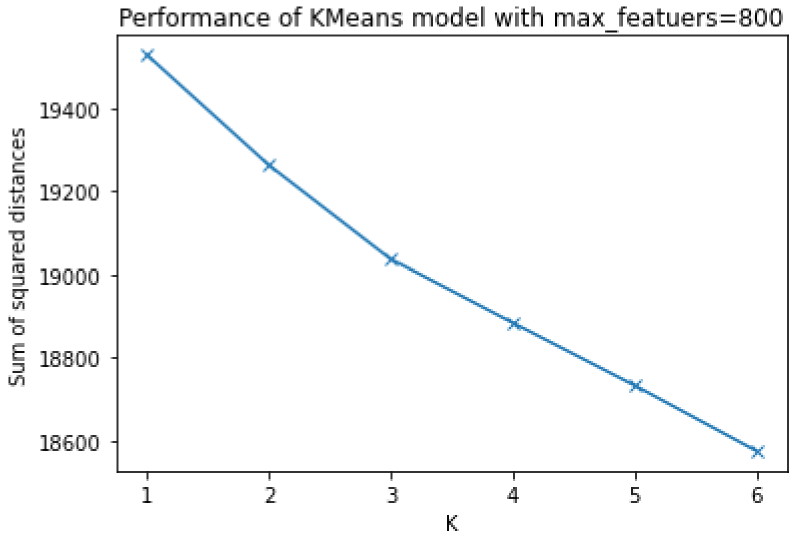
The plot tells a lot. With some pre-knowledge, the lyrics belong two major categories - hit song and non-hit song. Therefore, there is an elbow point at $k = 2$. But hit and non-hit seems not sufficient. $k = 4$ is also somehow reasonable because it is possible that different genre may produce different styles of lyrics.

In [28]:

```python
# Use TfidfVectorizer
sse = []
K = range(1,7)

for k in K:
    kmeans = KMeans(n_clusters = k).fit(df_tfidf)
    sse.append(kmeans.inertia_)

plt.plot(K, sse, 'x-')
plt.xlabel('K')
plt.ylabel('Sum of squared distances')
plt.title('Performance of KMeans model with max_featuers=800')
```

Out[28]:

```
Text(0.5, 1.0, 'Performance of KMeans model with max_featuers=800')
```



Using TfidfVectorizer, $k = 3$ gives a clear elbow. This makes sense because there might be songs pretty poplular but do not feature on Billboard Hot 100 charts.

k=2 with CounterVectorizer

In [59]:

```python
model_2 = KMeans(n_clusters = 2)
model_2.fit(df_cv)

# accuracy
sum(model_2.labels_ == song_lyrics['hit'])/len(song_lyrics)
```

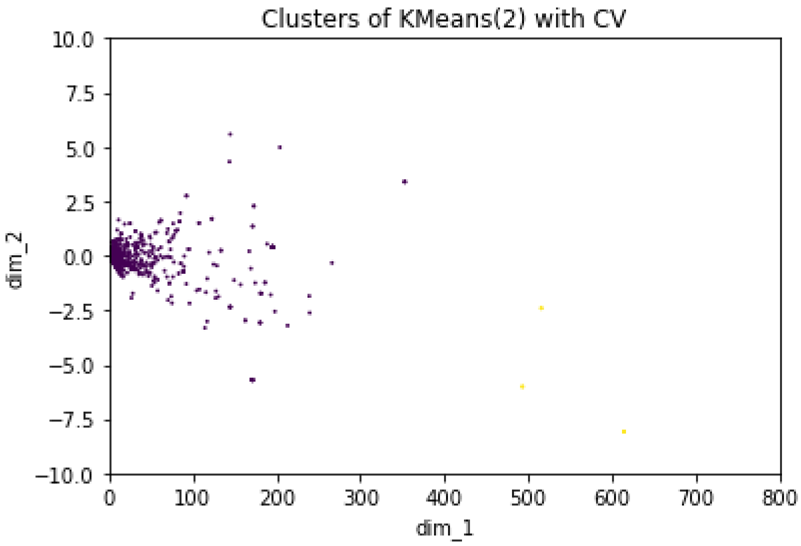
Out[59]:

```
0.8279001048517777
```

Cluster visualization using TruncatedSVD

In [218]:

```
from sklearn.decomposition import TruncatedSVD

svd = TruncatedSVD(n_components=3, random_state=42)
dtm_cv_3d = svd.fit_transform(dtm_cv)

plt.figure(figsize=(6,4))
plt.scatter(dtm_cv_3d[:, 1], dtm_cv_3d[:, 2], s=0.8, c=model_2.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.xlim(0,800)
plt.ylim(-10,10)
plt.title('Clusters of KMeans(2) with CV')
plt.show()
```



In [183]:

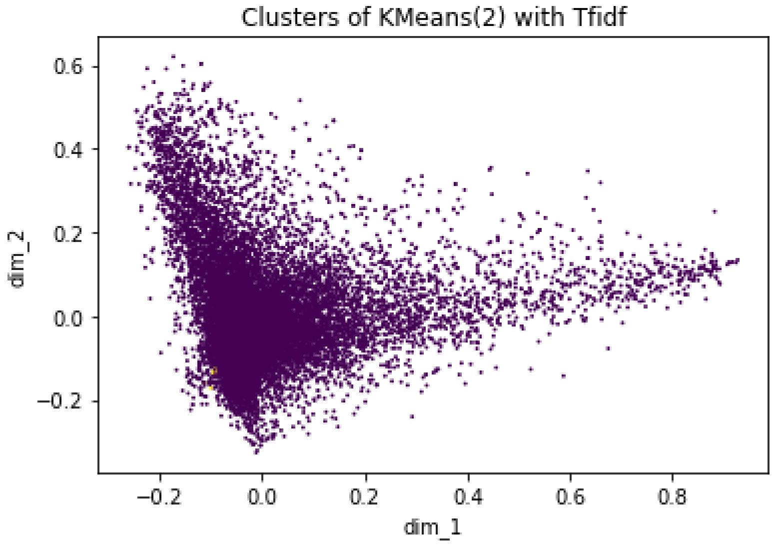
```
model_2_t = KMeans(n_clusters = 2)
model_2_t.fit(df_cv)

# accuracy
sum(model_2_t.labels_ == song_lyrics['hit'])/len(song_lyrics)
```

Out[183]:

```
0.8279001048517777
```

In [213]:

```
dtm_tfidf_3d = svd.fit_transform(dtm_tfidf)
plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.8, c=model_2_t.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('Clusters of KMeans(2) with Tfidf')
plt.show()
```



The accuracy of KMeans clustering with $k = 2$ is around 83%, which is not bad (because we cannot tell hit songs merely by the lyrics). The visualization of clsuters does not reveal too much information when $k = 2$, as most of the songs are non-hit songs and only 3300 are hit songs; therefore, the majorities are purple points (non-hit songs). Specifically, when comparing CountVectorizer and TfidfVectorizer, the clustering results given by TfidfVectorizer is much more staisfying than CountVectorizer, as it is more centered.

Let's try more number of clusters.

k=3

In [155]:

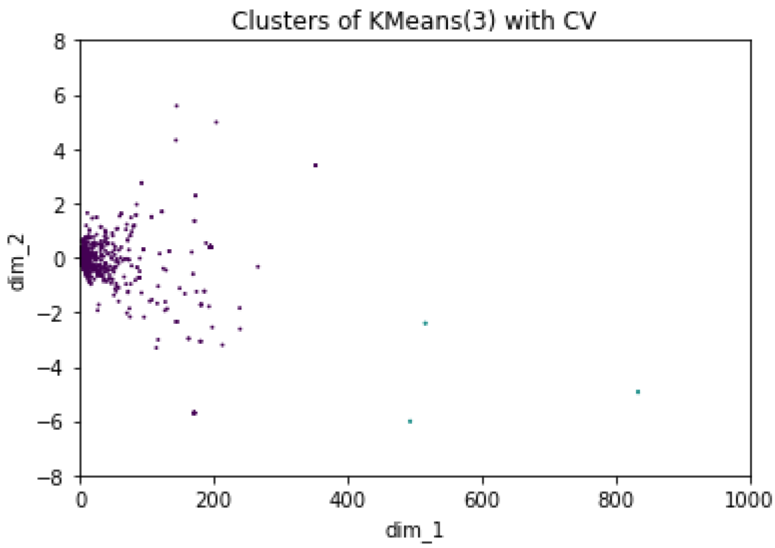
```
# CountVectorizer
model_3 = KMeans(n_clusters = 3)
model_3.fit(df_cv)

# TfidfVectorizer
model_3_t = KMeans(n_clusters = 3)
model_3_t.fit(df_tfidf)
```

Out[155]:

```
KMeans(n_clusters=3)
```

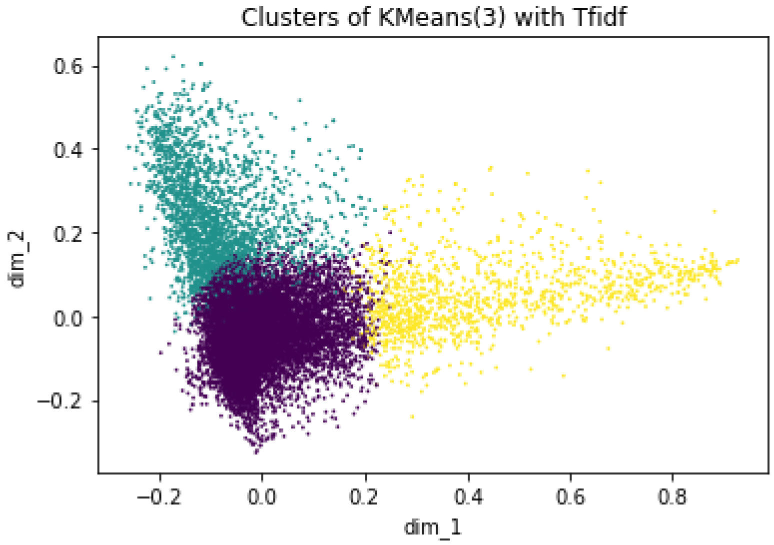
In [176]:

```
plt.figure(figsize=(6,4))
plt.scatter(dtm_cv_3d[:, 1], dtm_cv_3d[:, 2], s=0.8, c=model_3.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.xlim(-1,1000)
plt.ylim(-8,8)
plt.title('Clusters of KMeans(3) with CV')
plt.show()
```



In [215]:

```
plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=model_3_t.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('Clusters of KMeans(3) with Tfidf')
plt.show()
```



Aligned with what is observed using $k = 2$, the clusters obtained using TfidfVectorizer are much more reliable than CountVectorizer when $k = 3$. It can be easily seen that $k = 3$ is a very good choice of number of clusters because cases in each group are tighly clustered without much overlapping.

k=5

In [164]:

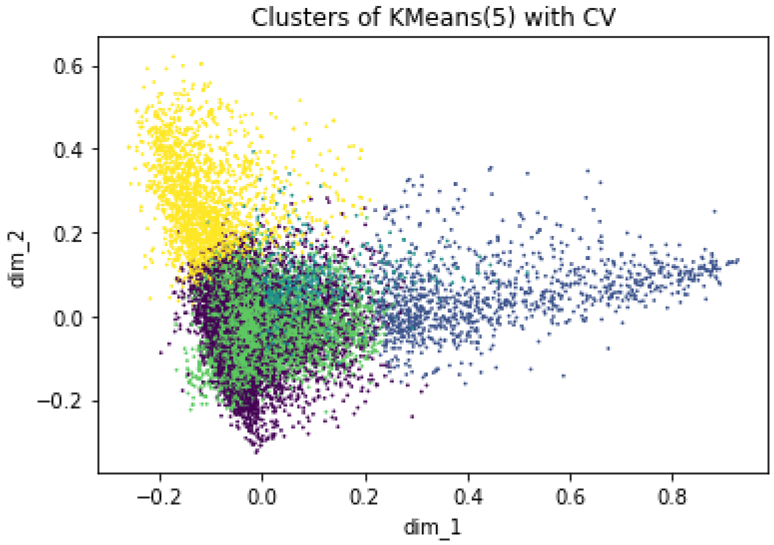
```
# CountVectorizer
model_5 = KMeans(n_clusters = 5)
model_5.fit(df_cv)

# TfidfVectorizer
model_5_t = KMeans(n_clusters = 5)
model_5_t.fit(df_tfidf)
```

Out[164]:

```
KMeans(n_clusters=5)
```

In [216]:

```
plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=model_5_t.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('Clusters of KMeans(5) with CV')
plt.show()
```



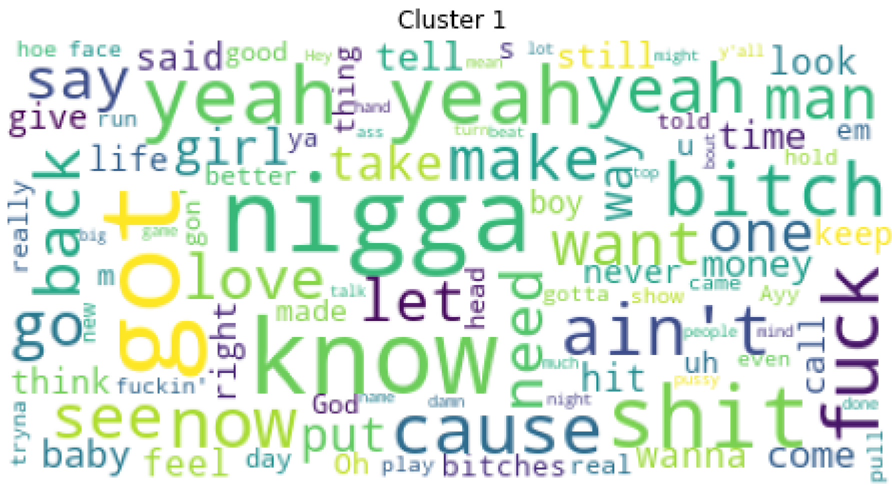
$k = 5$ might not be a good idea, since the seperation does not seem maximized as $k = 3$.

Wordcloud of each cluster

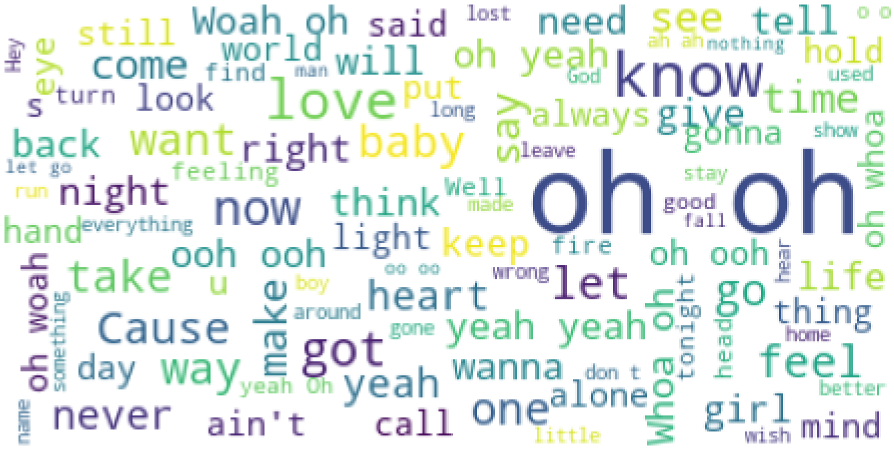
$k = 3$

In [206]:

```python
song_lyrics['label_3'] = model_3_t.labels_
for i in range(3):
    text = ' '.join(song_lyrics.loc[song_lyrics['label_3']==i, 'lyrics'])
    plt.figure(figsize=(8,6))
    wordcloud = WordCloud(max_font_size=50, max_words=100, background_color='whi
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.title('Cluster %s' %i)
    plt.show()
```



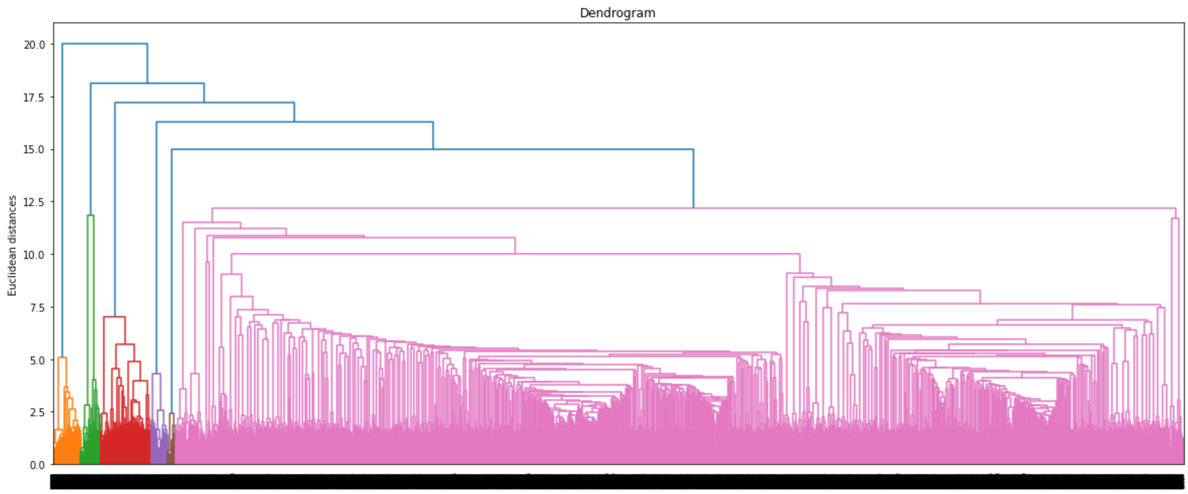
Cluster 2



Hierarchical clustering

In [272]:

```python
import scipy.cluster.hierarchy as sch

plt.figure(figsize=(20,8))
dendrogram = sch.dendrogram(sch.linkage(df_tfidf, method  = 'ward'))
plt.title('Dendrogram')
plt.ylabel('Euclidean distances')
plt.show()
```



In [229]:

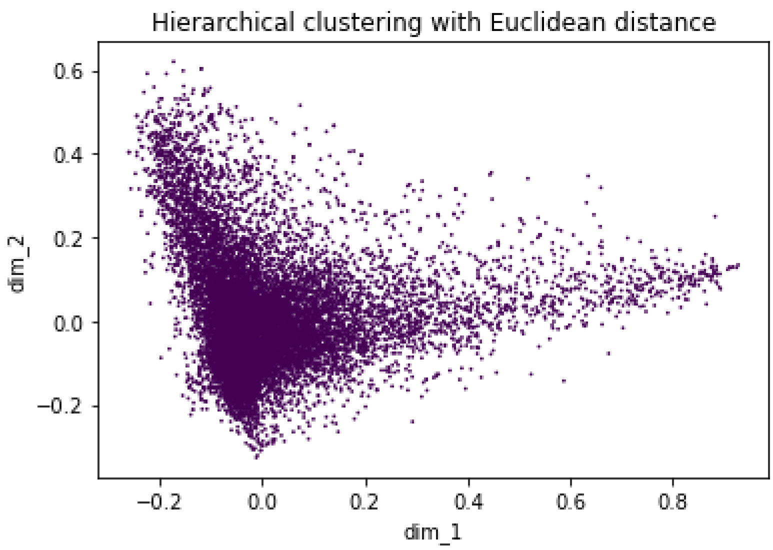
```python
hc_euc = AgglomerativeClustering(n_clusters=3,
                                 affinity = 'euclidean',
                                 linkage = 'single')

hc_euc.fit(df_tfidf)
```

Out[229]:

```
AgglomerativeClustering(linkage='single', n_clusters=3)
```

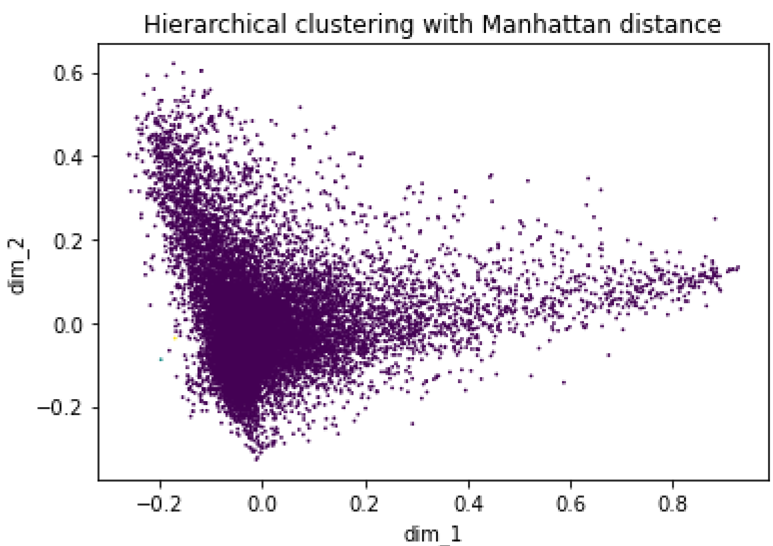
In [241]:

```
plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=hc_euc.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('Hierarchical clustering with Euclidean distance')
plt.show()
```



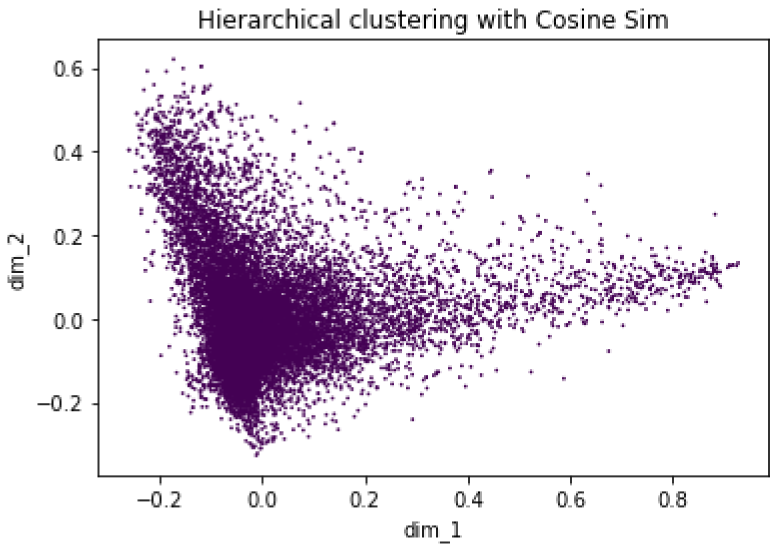
In [239]:

```
hc_man = AgglomerativeClustering(n_clusters=3,
                                 affinity = 'manhattan',
                                 linkage = 'single')

hc_man.fit(df_tfidf)

plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=hc_man.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('Hierarchical clustering with Manhattan distance')
plt.show()
```



In [240]:

```
hc_cos = AgglomerativeClustering(n_clusters=3,
                                  affinity = 'cosine',
                                  linkage = 'single')

hc_cos.fit(df_tfidf)

plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=hc_cos.labels_)
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('Hierarchical clustering with Cosine Sim')
plt.show()
```



DBSCAN

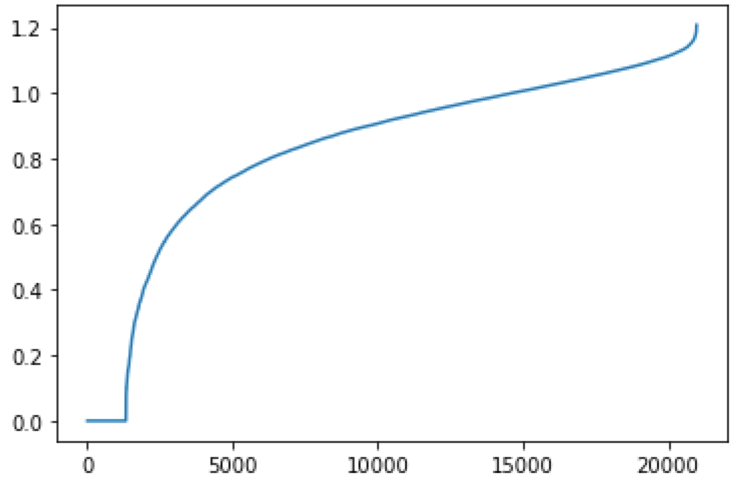
Determine epsilon

In [256]:

```
from sklearn.neighbors import NearestNeighbors

nbrs = NearestNeighbors(n_neighbors=3).fit(dtm_tfidf)
distances, indices = nbrs.kneighbors(dtm_tfidf)
distances = np.sort(distances, axis=0)
distances = distances[:,1]
plt.plot(distances)
```

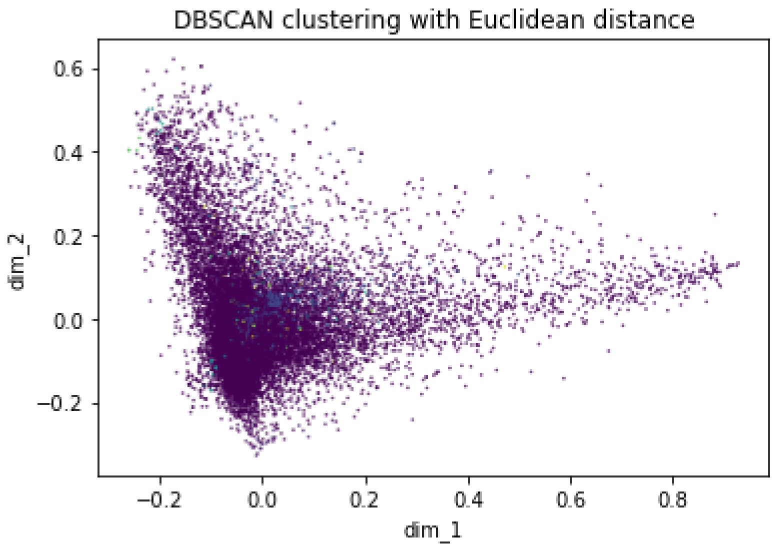
Out[256]:

```
[<matplotlib.lines.Line2D at 0x7fb5cc747820>]
```



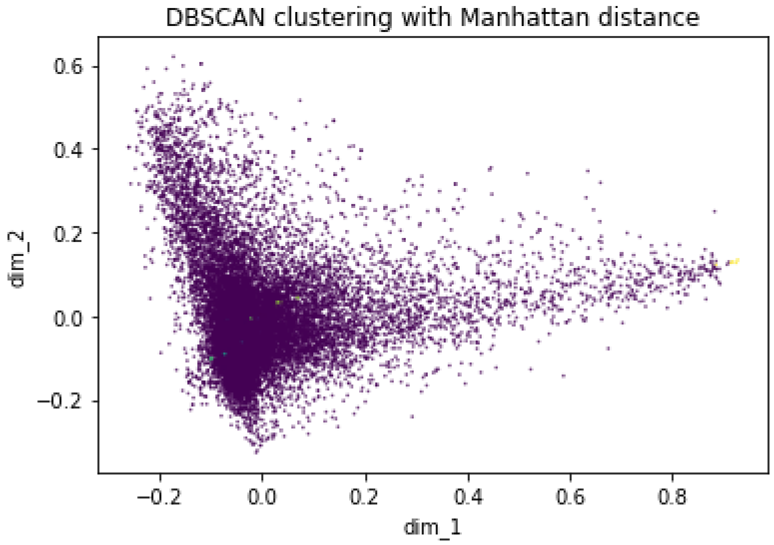
In [269]:

```
db_euc = DBSCAN(eps=0.62, metric='euclidean', n_jobs=-1)
db_euc.fit(dtm_tfidf)

plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=db_euc.labels_, alp
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('DBSCAN clustering with Euclidean distance')
plt.show()
```



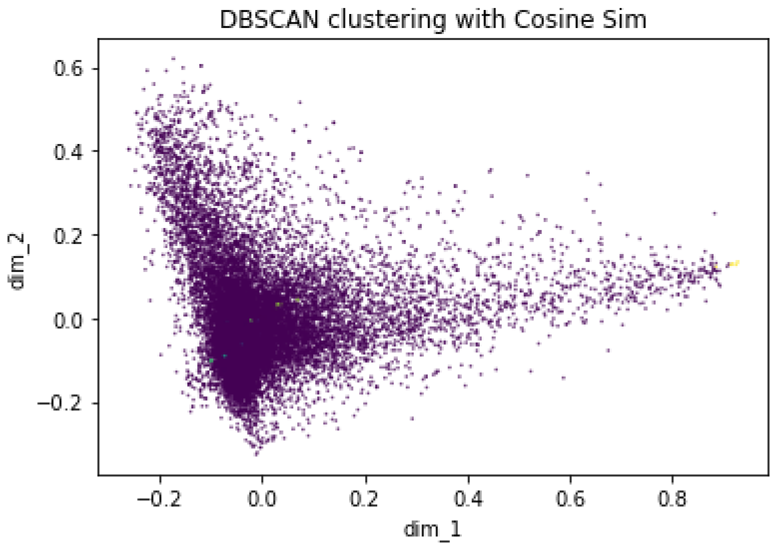
In [270]:

```
db_man = DBSCAN(eps=0.62, metric='manhattan', n_jobs=-1)
db_man.fit(dtm_tfidf)

plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=db_man.labels_, alp
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('DBSCAN clustering with Manhattan distance')
plt.show()
```



In [271]:

```
db_cos = DBSCAN(eps=0.62, metric='manhattan', n_jobs=-1)
db_cos.fit(dtm_tfidf)

plt.figure(figsize=(6,4))
plt.scatter(dtm_tfidf_3d[:, 1], dtm_tfidf_3d[:, 2], s=0.6, c=db_cos.labels_, alp
plt.xlabel('dim_1')
plt.ylabel('dim_2')
plt.title('DBSCAN clustering with Cosine Sim')
plt.show()
```



In [ ]:

```

```