CMSC320 Final Project: Top Music Charts and Song Lyric Analysis

Authors: **Noah Burkhardt**, **Courtland Climer**, **Ares Shackleford**, and **Ryan Sutton**

Introduction

In this project we wished to analyze song lyrics of popular songs now and over time. We decided to use the Billboard Top 100 chart and genre specific charts as our main metric of popularity and supplemented that with Spotify popularities for all of the current songs. We wanted to create an interesting dataset from scratch and we thought that combining Billboard charts and Spotify popularity scores would be a new and interesting way to analyze popular music based on lyrics. This is a relevant topic to explore because, at the end of the day, everyone listens to music and lyrics play an important role in our culture. This is an important inquiry in data science because if certain trends in popular songs are identified, it could help produce more new, popular songs. We wanted to look at the following questions:

1. How similar are songs on the Billboard Top 100 chart and genre specific charts to each other in terms of lyrics?
2. How has the level of profanity present in songs on the Billboard Top 100 chart changed over the last 14 years, and does this correlate with Billboard ranking?
3. What are the most common words found in the Billboard Top 100 chart and genre specific charts? Are they the same or does it differ based on genre?
4. Can we train a model to guess what the rank of a current song on the Billboard Top 100 chart is based solely on the most frequent lyrics of its chart?
5. Can we guess which song in the Billboard Top 100 chart somebody is describing by only having them input a few words?
6. Does the popularity of a song on Spotify correlate to the ranking of the song on the Billboard Top 100?

In [49]:
```
%%capture
!pip install spotipy
!pip install billboard.py;
!pip install lyricsgenius;
!pip install nltk;
!pip install matplotlib;
!pip install seaborn;
!pip install profanity-check
```

Required libraries: spotipy, billboard, lyricsgenius, nltk, matplotlib, seaborn, profanity-check

In [63]:
```
%%capture
import requests
import pandas as pd
import numpy as np
import billboard as bb
import lyricsgenius
import seaborn as sns
import sqlite3
import matplotlib.pyplot as plt
import nltk
from sklearn.feature_extraction.text import CountVectorizer
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from sklearn.feature_extraction.text import TfidfVectorizer
import nltk
nltk.download("stopwords")
from nltk.corpus import stopwords
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from profanity_check import predict, predict_prob
```

```
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Data Gathering

The hardest part of this section of the project was the data collection. We wanted to combine various different data sources in order to do more meaningful analysis. We decided we wanted to analyze the Billboard Top 100 chart and the charts for the top songs in the following genres: rap, pop, country, and rock as they are some of the most common and popular music genres currently. Scraping the Billboard website gave us the rank of a song in a particular chart, the song's title, and the song's artist(s). We then used the song title and artist to query the genius.com API for lyrics to every song. It took a long time to find an API that could be easily used to fetch all of the lyrics to each song with the information we had from the Billboard website. We tried numerous different APIs and websites such as azlyrics and Musixmatch before settling on genius.com. The only drawback of this API is that it gets rate limited and is fairly slow. In addition, we wanted to gather information about each song's Spotify popularity in order to see whether rankings of songs correlated with Spotify popularity or not. After these steps we had all of the songs, lyrics, ranks, and popularities for each current chart, but we also wanted to analyze trends over time for the Billboard Top 100 songs. This particular dataset that we constructed has almost two thousand entries, and so querying the lyrics for each song took an immense amount of time. For this reason, we chose to store the historical data in a SQLite database.

In [3]:
```
genius = lyricsgenius.Genius("") #insert API key here
genius.verbose = False # Turn off status messages
genius.remove_section_headers = True # Remove section headers (e.g. [Chorus]) fro
m lyrics when searching
genius.skip_non_songs = False # Include hits thought to be non-songs (e.g. track
 lists)
genius.excluded_terms = ["(Remix)", "(Live)"] # Exclude songs with these words in
their title

client_credentials_manager = SpotifyClientCredentials(client_id = "", client_secr
et = "") # insert API keys here
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

The above cell sets up using the genius API to get lyrics for specific songs. One would need to replace "Api Key" with their own genius API key in order to use the API. If you want to know more about the genius api you can find information about it here: https://docs.genius.com/ (https://docs.genius.com/). If you want to know more about the lyricsgenius python library that wraps the genius API you can find more about it here: https://github.com/johnwmillr/LyricsGenius (https://github.com/johnwmillr/LyricsGenius). We tried using many different lyrics sites/APIs to get the lyrics such as Musixmatch or azlyrics, but we ran into the problem that you needed a paid API key to get access to all of the lyrics or that the search function for songs did not work with the way Billboard formulated the Song artist and title.

We use a python wrapper for the Spotify API called `spotipy`. The documentation for `spotipy` can be found here: https://spotipy.readthedocs.io/en/latest/#non-authorized-requests (https://spotipy.readthedocs.io/en/latest/#non-authorized-requests). The documentation for the Spotify API as a whole can be found here: https://developer.spotify.com/documentation/web-api/ (https://developer.spotify.com/documentation/web-api/)

In [4]:
```
def create_songs_df(chart_name):
    top100 = bb.ChartData(chart_name)
    songs = top100.entries
    dataframe = {"Rank": [], "Title": [], "Artist": []}
    for song in songs:
        dataframe["Title"].append(song.title)
        dataframe["Artist"].append(song.artist)
        dataframe["Rank"].append(song.rank)
    df = pd.DataFrame(dataframe)
    return df
```

The above cell allows for creation of dataframes to store different charts from the billboard website. We used a `billboard` python package that you can find more information about here https://github.com/guoguo12/billboard-charts (https://github.com/guoguo12/billboard-charts). This method creates a dataframe for a specific chart with columns for song title, song artist, and the rank of the song on the chart and then returns the dataframe.

In [5]:
```
top100 = create_songs_df("hot-100")
country = create_songs_df("country-songs")
rock = create_songs_df("rock-songs")
pop = create_songs_df("pop-songs")
rap = create_songs_df('rap-song')
```

Here we load in five different billboard top100 charts that we will use later to perform our analysis. If you want to explore Billboard's website you can view it here: https://www.billboard.com/ (https://www.billboard.com/) and you can find more information about how the charts are actually compiled here: https://www.billboard.com/p/billboard-charts-legend (https://www.billboard.com/p/billboard-charts-legend).

In [6]:
```
def try_load_lyrics(title,artist):
    try:
        return genius.search_song(title, artist).lyrics
    except:
        return "Failed to Find Lyrics"
```

In [7]:
```
def populate_chart_with_lyrics(chart):
    chart["lyrics"] = chart.apply(lambda x: try_load_lyrics(x["Title"],x["Artist"
]), axis = 1)
    return chart
```

The above methods are used to add a column to a dataframe for the lyrics of each song using the genius API.

In [8]:
```
top100 = populate_chart_with_lyrics(top100)
rap = populate_chart_with_lyrics(rap)
pop = populate_chart_with_lyrics(pop)
country = populate_chart_with_lyrics(country)
rock = populate_chart_with_lyrics(rock)

display(top100.head(10))
```

	Rank	Title	Artist	lyrics
0	1	Heartless	The Weeknd	Young Metro, young Metro, young Metro (Sheesh)...
1	2	Circles	Post Malone	Oh, oh, oh\nOh, oh, oh\nOh, oh, oh, oh, oh\n...
2	3	All I Want For Christmas Is You	Mariah Carey	I don't want a lot for Christmas\nThere is jus...
3	4	Someone You Loved	Lewis Capaldi	I'm going under, and this time, I fear there's...
4	5	Memories	Maroon 5	Here's to the ones that we got\nCheers to the ...
5	6	Good As Hell	Lizzo	I do my hair toss, check my nails\nBaby, how y...
6	7	Roxanne	Arizona Zervas	All for the 'Gram\nBitches love the 'Gram\nOh ...
7	8	Rockin' Around The Christmas Tree	Brenda Lee	Rockin' around the Christmas tree\nAt the Chri...
8	9	Lose You To Love Me	Selena Gomez	You promised the world and I fell for it\nI pu...
9	10	10,000 Hours	Dan + Shay & Justin Bieber	Do you love the rain, does it make you dance\n...

```
def getSongRanks(year):
    r = requests.get(f"https://www.billboard.com/charts/year-end/{year}/hot-100-songs")
    if r.status_code != 200:
        print(f"Failed on year {year}")
        return
    time.sleep(2) # If I remove this I get a "Too Many Requests" error from billboard.com
    soup = BeautifulSoup(r.text, 'html.parser')
    ranks = soup.find_all("div", {"class": "ye-chart-item__rank"})
    titles = soup.find_all("div", {"class": "ye-chart-item__title"})
    artists = soup.find_all("div", {"class": "ye-chart-item__artist"})
    ranks = [rank.text.strip() for rank in ranks]
    titles = [title.text.strip() for title in titles]
    artists = [artist.text.strip() for artist in artists]
    songs = list(zip(ranks, titles, artists))
    print(f"Done with year {year}")
    return songs

ranks = pd.DataFrame(columns=["Title", "Artist", "Year", "Rank", "lyrics"])

year_range = range(2006, 2020)
for year in year_range:
    top = getSongRanks(year)
    for (rank, title, artist) in top:
        ranks = ranks.append({"Title": title, "Artist": artist, "Year": year, "Rank": rank, "lyrics": np.nan}, ignore_index=True)
```

The code in the code block above is the key for building that database. Some boilerplate code was left out for brevity. We use this database to examine two relationships: the use of profanity in song lyrics for the Billboard Top 100 over the past 14 years, and the relationship between profanity in song lyrics and the Billboard Top 100 ranking.

In [9]:
```
def get_spotify_popularity(title):
    title = title.replace("$","s")
    answer = sp.search(title, type="track", limit = 1)
    track = sp.track(answer["tracks"]["items"][0]["id"])
    return track["popularity"]
```

The above method gets the Spotify popularity associated with a specific song. The popularity metric on Spotify is a number between 0 and 100 based on how many plays the track has and how recent those plays are. The algorithm is not publically available, but if you want to look at the API documentation it is here: [https://developer.spotify.com/documentation/web-api/reference/tracks/get-track/ (https://developer.spotify.com/documentation/web-api/reference/tracks/get-track/)](https://developer.spotify.com/documentation/web-api/reference/tracks/get-track/).

In [10]:
```
def populate_chart_with_spotify_popularity(chart):
    chart["spotify_popularity"] = chart.apply(lambda x: get_spotify_popularity(x["Title"]), axis = 1)
    return chart
```

This method adds a column to an existing dataframe of a music chart with the Spotify popularity of each song.

In [11]:
```
top100 = populate_chart_with_spotify_popularity(top100)
rap = populate_chart_with_spotify_popularity(rap)
pop = populate_chart_with_spotify_popularity(pop)
country = populate_chart_with_spotify_popularity(country)
rock = populate_chart_with_spotify_popularity(rock)

display(top100.head(10))
```

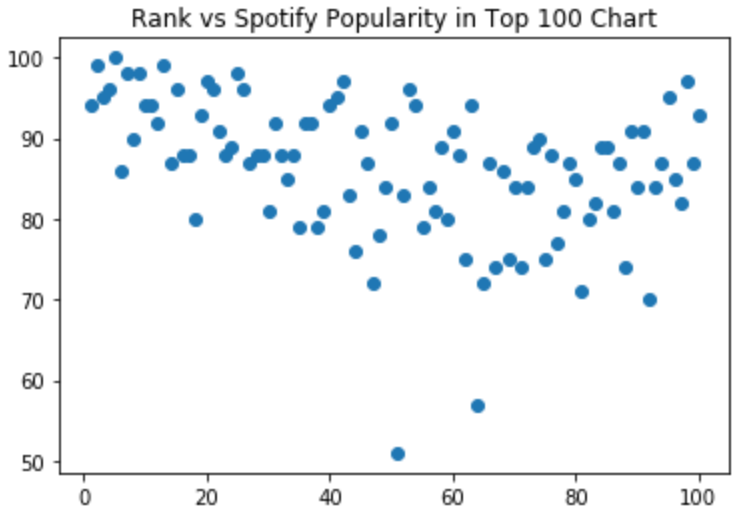
	Rank	Title	Artist	lyrics	spotify_popularity
0	1	Heartless	The Weeknd	Young Metro, young Metro, young Metro (Sheesh)...	94
1	2	Circles	Post Malone	Oh, oh, oh\nOh, oh, oh\nOh, oh, oh, oh, oh\n...	99
2	3	All I Want For Christmas Is You	Mariah Carey	I don't want a lot for Christmas\nThere is jus...	95
3	4	Someone You Loved	Lewis Capaldi	I'm going under, and this time, I fear there's...	96
4	5	Memories	Maroon 5	Here's to the ones that we got\nCheers to the ...	100
5	6	Good As Hell	Lizzo	I do my hair toss, check my nails\nBaby, how y...	86
6	7	Roxanne	Arizona Zervas	All for the 'Gram\nBitches love the 'Gram\nOh ...	98
7	8	Rockin' Around The Christmas Tree	Brenda Lee	Rockin' around the Christmas tree\nAt the Chri...	90
8	9	Lose You To Love Me	Selena Gomez	You promised the world and I fell for it\nI pu...	98
9	10	10,000 Hours	Dan + Shay & Justin Bieber	Do you love the rain, does it make you dance\n...	94

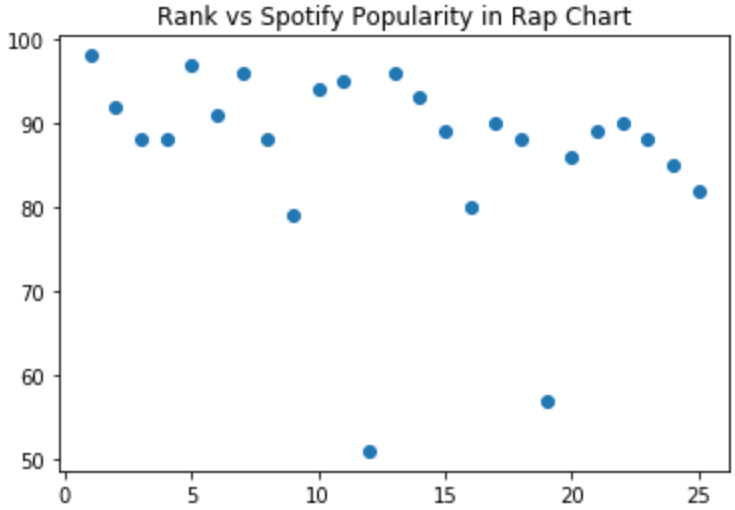
Top Music Chart NLP and Analysis

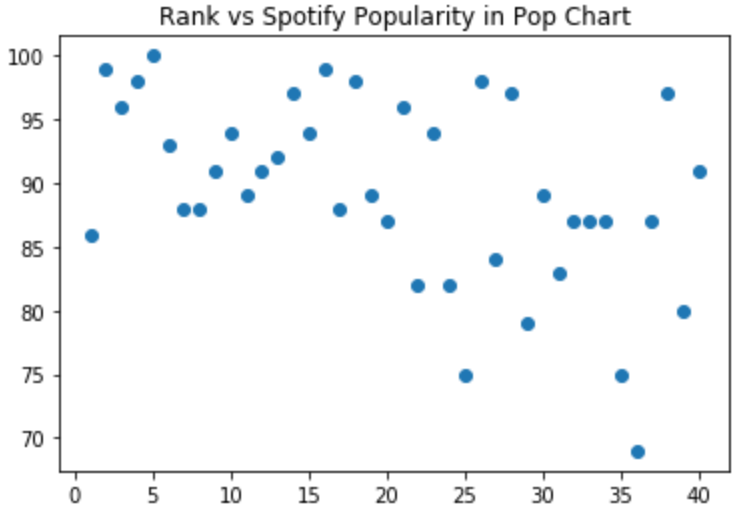
We wanted to look at the relationship between Billboard chart rankings and Spotify popularity ratings, so we explored the data by creating a scatter plot of ranking versus popularity for each chart we were looking at. This gives us a quick visual view of whether or not the chart rankings and popularity ratings are related as we would expect them to be. We then set up two methods to create a term document frequency matrix (tdf) and a term document inverse document frequency matrix (tf-idf), which are two common natural language process methods for looking at and analyzing data. Both of these concepts are explained further below. We then used cosine similarity to determine how similar songs in the same chart were to each other and whether certain charts were much more similar than each other. We thought that we would see that many songs in the same genre would contain very similar words in their lyrics. We used barplots to quickly visualize this data and draw conclusions from it. In addition, we looked at what two songs were the most similar to each other in each chart to see if two very similar songs, in terms of lyrics, would both be popular at the same time. We thought it would also be interesting to get a quick view of what percentage of the songs on the Billboard Top 100 chart came from the 4 other genre specific charts we were looking at for our analysis and to visualize this in a bar chart.

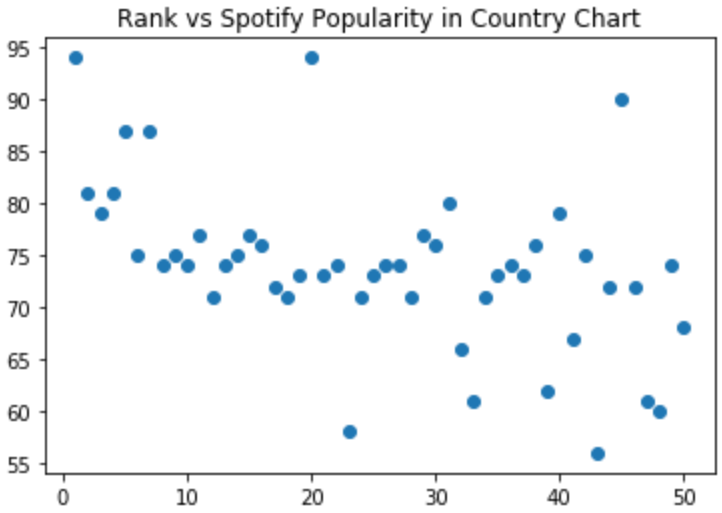
In [281]:

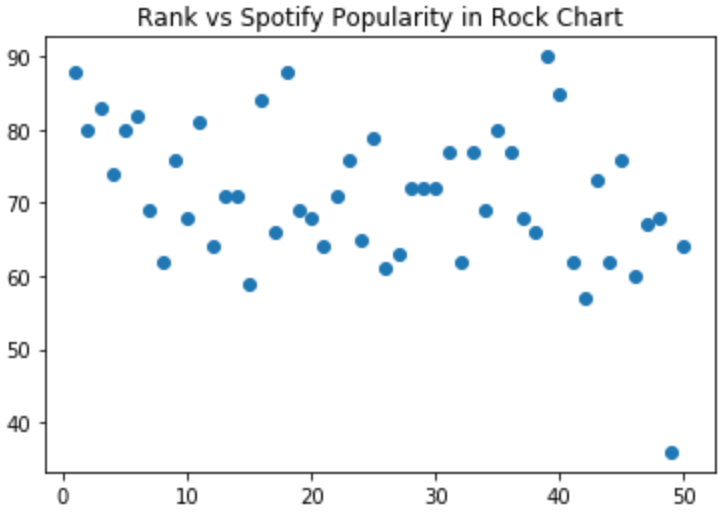
```
plt.scatter('Rank', 'spotify_popularity', data=top100)
plt.title('Rank vs Spotify Popularity in Top 100 Chart')
plt.show()

plt.scatter('Rank', 'spotify_popularity', data=rap)
plt.title("Rank vs Spotify Popularity in Rap Chart")
plt.show()

plt.scatter('Rank', 'spotify_popularity', data=pop)
plt.title("Rank vs Spotify Popularity in Pop Chart")
plt.show()

plt.scatter('Rank', 'spotify_popularity', data=country)
plt.title("Rank vs Spotify Popularity in Country Chart")
plt.show()

plt.scatter('Rank', 'spotify_popularity', data=rock)
plt.title("Rank vs Spotify Popularity in Rock Chart")
plt.show()
```


Rank vs Spotify Popularity in Top 100 Chart
100
90
80
70
60
50
0
20
40
60
80
100


Rank vs Spotify Popularity in Rap Chart
100
90
80
70
60
50
0
5
10
15
20
25


Rank vs Spotify Popularity in Pop Chart
100
95
90
85
80
75
70
0
5
10
15
20
25
30
35
40





In the charts above, we explore the relationship between the chart ranking and the level of popularity on Spotify for multiple genres and the Top 100 chart. We expected that the well-ranked songs would also be heavily listened to recently on Spotify. This absolutely turned out to be the case, especially with the very well-ranked songs in the first through tenth ranking spots for all of the genre charts and the Top 100 chart. Surprisingly, this was more so the case with specific genres, namely rap and pop, than it was in country and rock. This could likely be explained by the fact that more people that use Spotify listen to rap and pop than country and rock, so these genres likely suffer an inherent Spotify popularity point loss (we guess this since the Spotify documentation states that they use number of listens, in part, to determine a song's Spotify popularity).

In [13]:
```
def create_term_doc_matrix(chart):
    lyrics = chart["lyrics"].to_list()
    vec = CountVectorizer()
    X = vec.fit_transform(lyrics)
    termdocmatrix = pd.DataFrame(X.toarray(), columns=vec.get_feature_names(), in
dex=chart.Title)
    return termdocmatrix
```

In [14]:
```
top100 = top100[top100.lyrics != "Failed to Find Lyrics"]
create_term_doc_matrix(top100).head()
```

Out[14]:

Title	000	02	10	100	101	10k	11	12	12am	14	...	yvncc	zaar	zella	zervas	zhu	zombie	zo
Heartless	0	0	0	0	0	0	0	0	0	0	...	0	1	0	0	0	0	
Circles	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
All I Want For Christmas Is You	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
Someone You Loved	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
Memories	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

5 rows × 4161 columns

This method creates a term-document matrix, which is a matrix that has every word in every song in the dataframe as a column and then the count of how many times each word appeared in a specific songas the value for each row, which in our case is a song. If you want to read further about term-document matrices here is a short blog post that explains the idea https://www.darrinbishop.com/blog/2017/10/text-analytics-document-term-matrix/ (https://www.darrinbishop.com/blog/2017/10/text-analytics-document-term-matrix/).

In [15]:
```
def create_tfidf(chart):
    lyrics = chart["lyrics"].to_list()
    vectorizer = TfidfVectorizer()
    doc_vec = vectorizer.fit_transform(lyrics)
    tfidf = pd.DataFrame(doc_vec.toarray().transpose(), index=vectorizer.get_feat
ure_names())
    tfidf.columns = chart["Title"]
    return tfidf
```

In [16]:
```
tfidf = create_tfidf(top100)
tfidf.head()
```

Out[16]:

Title	Heartless	Circles	All I Want For Christmas Is You	Someone You Loved	Memories	Good As Hell	Roxanne	Rockin' Around The Christmas Tree	Lose You To Love Me	10,000 Hours	...	Foll G
000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
02	0.0	0.0	0.0	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	0.0	0.0	0.0	...	
100	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	
101	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	

5 rows × 100 columns

This method creates a term frequency-inverse document frequency (tf-idf) matrix. This is done by multiplying the term frequency score for each value by the inverse document frequency score of each value. The term frquency is calculated for each word in each song by counting the number of times the word appears in the song and dividing it by the total number of words in the song. The idf score is the inverse document frequency of a word, which is defined as the log(# of documents or in this case songs/ # of documents with the word in them). These matrices are used to determine the importance of a word to a song based off how often it is used in the song itself and in the other songs in the dataframe. A word is more important if it is only used by a single song in the dataframe. If you want to read more about tf-idf matrices here is a good resource: http://www.tfidf.com/ (http://www.tfidf.com/).

In [17]:
```
def average_cosine_similarity(tfidf):
    similarity = 0
    count = 0
    for index in range(tfidf.shape[1]):
        col = tfidf.iloc[: , index]
        for ind2 in range(index + 1, tfidf.shape[1]):
            col2 = tfidf.iloc[: , ind2]
            similarity = similarity + np.dot(col.values,col2.values)/(np.linalg.n
orm(col.values)*np.linalg.norm(col2.values))
            count = count + 1
    return similarity/count
```

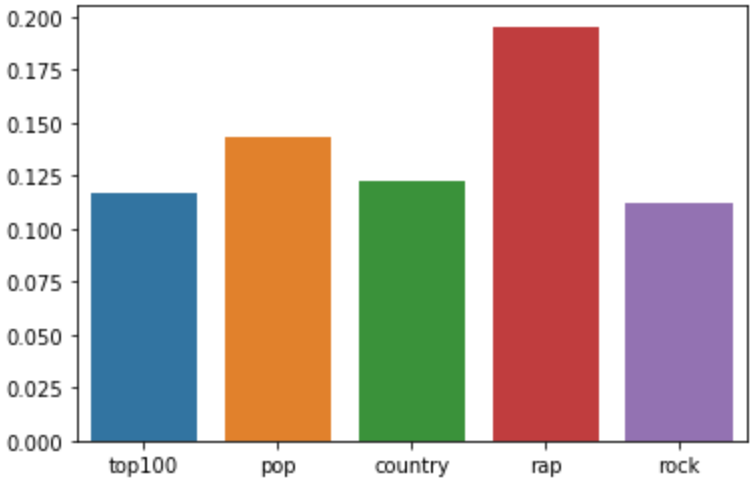
This method calculates the average pairwise cosine similarity for each song in the chart. Cosine similarity is a measure of the similarity between two different songs x and y. It is calculated by similarity(x,y) = $\frac{x^T y}{|x|*|y|}$. This gives a value between 0 and 1, with 0 meaning the vectors are not related at all and 1 meaning the vectors are the same. If you want to read more about cosine similarity you can look here: https://www.machinelearningplus.com/nlp/cosine-similarity/ (https://www.machinelearningplus.com/nlp/cosine-similarity/).

In [18]:
```
def top_pairwise_cosine_similarity(tfidf):
    similarity = 0
    songs = []
    for index in range(tfidf.shape[1]):
        col = tfidf.iloc[: , index]
        for ind2 in range(index + 1, tfidf.shape[1]):
            col2 = tfidf.iloc[: , ind2]
            temp = np.dot(col.values,col2.values)/(np.linalg.norm(col.values)*np.
linalg.norm(col2.values))
            if temp > similarity:
                similarity = temp
                names = [col.name,col2.name]
    return similarity, names
```

This method calculates and returns the cosine similarity for the most similar pair of songs in the chart and the names of the two songs that were the most similar.

In [19]:
```
x = np.array(["top100", "pop", "country", "rap", "rock"])
y = []
y.append(average_cosine_similarity(create_tfidf(top100)))
y.append(average_cosine_similarity(create_tfidf(pop)))
y.append(average_cosine_similarity(create_tfidf(country)))
y.append(average_cosine_similarity(create_tfidf(rap)))
y.append(average_cosine_similarity(create_tfidf(rock)))
y = np.array(y)
sns.barplot(x=x, y=y)
```

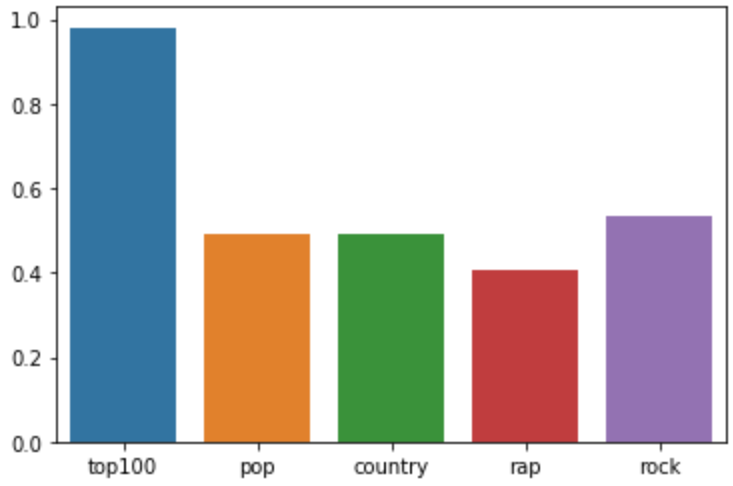
Out[19]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f7477e4a6d8>
```



This chart shows the average pairwise cosine similarity over the five different charts we are looking at. This means that the cosine similarity was calculated bewteen every pair of songs in each chart and then the average was taken for all of those values. It is interesting to see that rap has the most similarity between all of the songs in its top chart and then the second highest similarity is pop. The top100, rock and country all have similar average simalirty scores. The fact that all of these scors are relatively low as the largest one is less than .2 means that songs in the same genre or on the top100 chart are often very different in their lyrics.

In [20]:
```
x = np.array(["top100", "pop", "country", "rap", "rock"])
y = []
z = {}
score,names = top_pairwise_cosine_similarity(create_tfidf(top100))
y.append(score)
z["top100"] = names
score,names = top_pairwise_cosine_similarity(create_tfidf(pop))
y.append(score)
z["pop"] = names
score,names = top_pairwise_cosine_similarity(create_tfidf(country))
y.append(score)
z["country"] = names
score,names = top_pairwise_cosine_similarity(create_tfidf(rap))
y.append(score)
z["rap"] = names
score,names = top_pairwise_cosine_similarity(create_tfidf(rock))
y.append(score)
z["rock"] = names
y = np.array(y)
sns.barplot(x=x, y=y).set_title("Cosine Similarity")
for chart in z:
    print(chart + "'s most similar pair of songs is " + z[chart][0] + " and " + z
[chart][1])
```

```
top100's most similar pair of songs is Into The Unknown and Into The Unknown
pop's most similar pair of songs is Panini and Take What You Want
country's most similar pair of songs is I Hope and I Hope You're Happy Now
rap's most similar pair of songs is Death and 223's
rock's most similar pair of songs is Under The Graveyard and Legendary
```

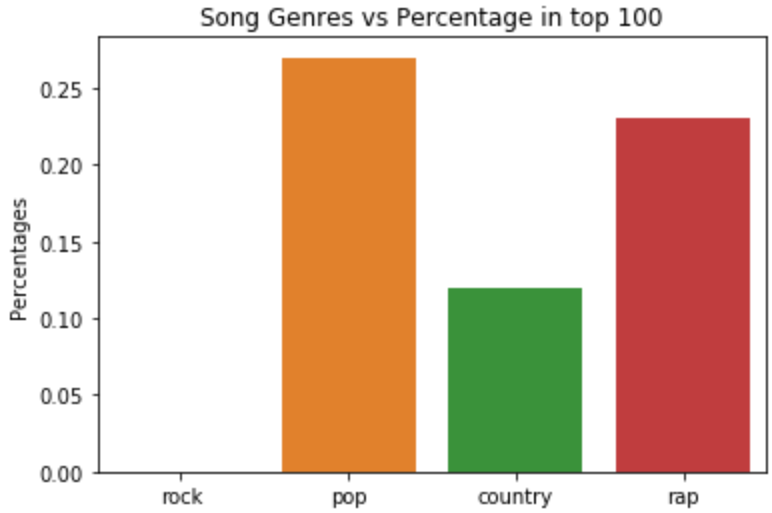


The graph shows that the top100 has the most similar pair of songs by far than any other chart. This is due to the fact that two different versions of the same song are both on the Billboard top100 chart, so this is a bit of an outlier and does not normally occur. It is interesting to see that rap has the lowest max similarity between any pair of songs as it was the chart that had the highest average pairwise similarity out of all of the charts. Also, the fact that the max similarity value is usually around .5 shows that there is significant overlap between certain songs in the same genre.

In [21]:
```
def percent_of_top_100(songs):
    count = 0
    len = songs.shape[0] #num of rows

    for index, row in songs.iterrows():
        if ((top100['Title'] == row['Title']) & (top100['Artist'] == row['Artist'
])).any():
            count += 1

    return count / 100
```

In [315]:
```
bins = (rock, pop, country, rap)
percents = []

for song in bins:
    percents.append(percent_of_top_100(song))

percents = np.array(percents)
x = np.array(["rock", "pop", "country", "rap"])

ax = sns.barplot(x=x, y=percents)
ax.set_title('Song Genres vs Percentage in top 100')
ax.set_ylabel('Percentages')
ax
```

Out[315]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff8d210d240>
```



Word Frequency Analysis

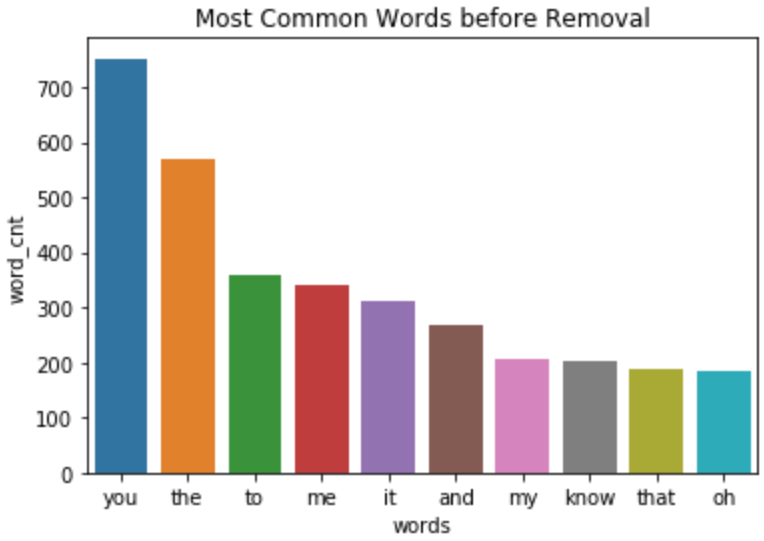
In this section, we try to answer the research questions provided in the introduction. We start by getting the ten most frequent words for the Top 100 and each genre chart. We then "clean" these most frequent words, removing the common and generally "uninteresting" words. Further analysis is performed on frequent words. We then do analysis on profanity present in these charts, models for predicting rank and Spotify popularity, and a model for guessing the Billboard Top 100 song given certain lyrics.

In [23]:
```
def get_lyric_len(chart):
    return create_term_doc_matrix(chart).sum(axis = 1, skipna = True)
```

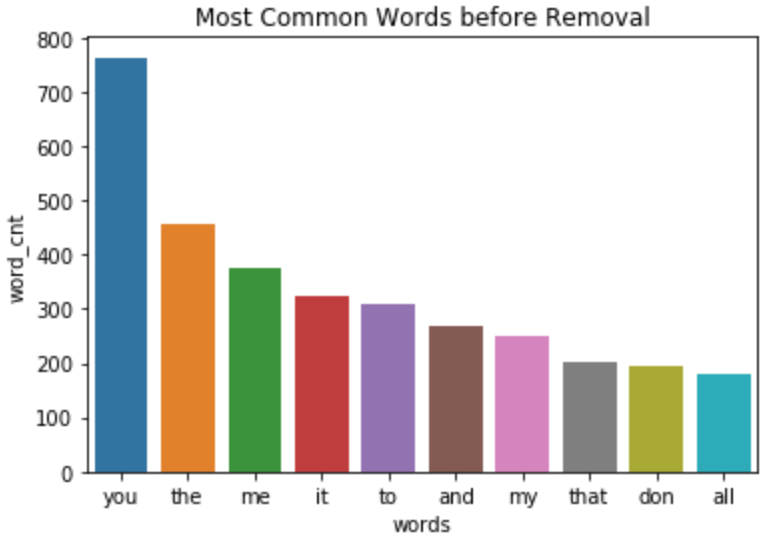
In [24]:
```
def get_chart_word_count(chart):
    return create_term_doc_matrix(chart).sum(axis = 0, skipna = True)

def find_most_freq_words(chart):
    counts = get_chart_word_count(chart)
    return counts.nlargest(10)
```

In [85]:
```
def create_word_freq_chart(chart):
    freq = find_most_freq_words(chart)
    freq = pd.DataFrame({'words':freq.index, 'word_cnt':freq.values})
    sns.barplot(x="words", y="word_cnt", data=freq).set_title("Most Common Words
 before Removal")
```

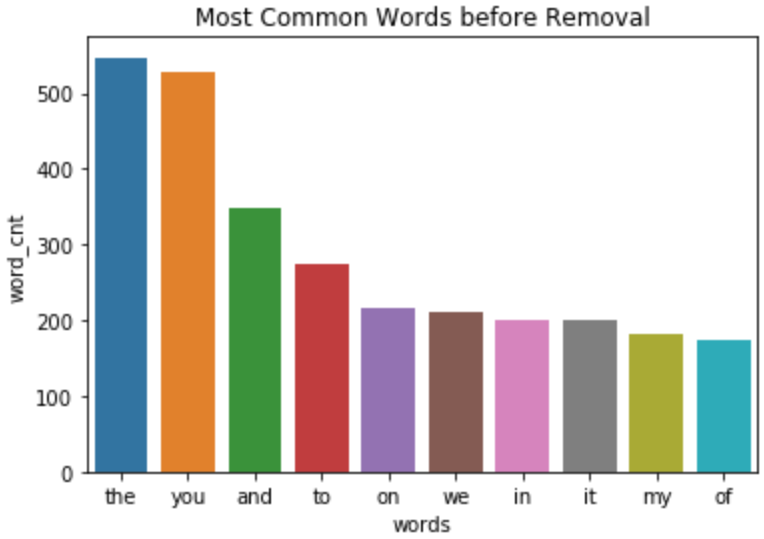
In [86]:
```
create_word_freq_chart(rock)
```



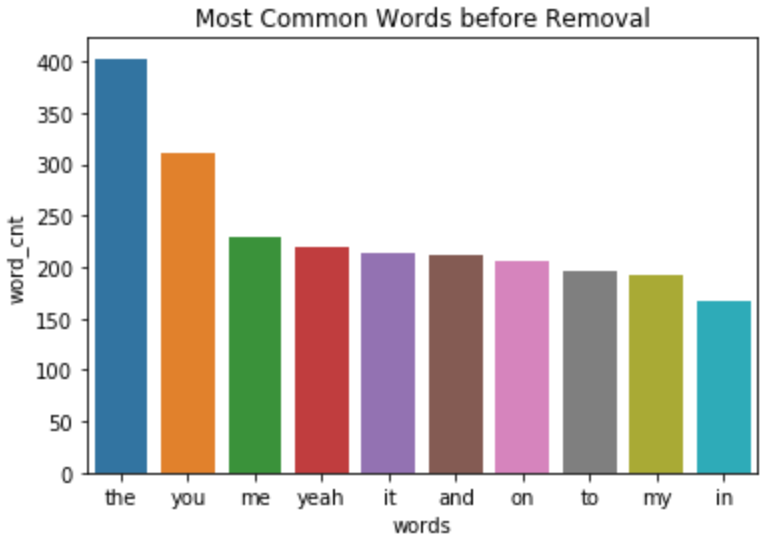
In [87]:
```
create_word_freq_chart(pop)
```



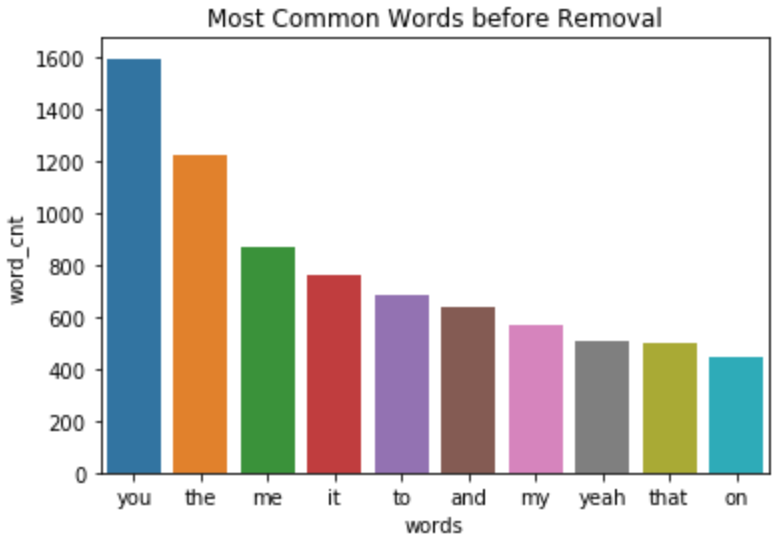
In [88]:
```
create_word_freq_chart(country)
```



In [89]:
```
create_word_freq_chart(rap)
```



In [90]:
```
create_word_freq_chart(top100)
```



All of the words seem to be very similar to one another. The words 'the' and 'you' are consistently the top two words in all of the barplots. This is expected considering those words are commonly used in English in general. Since these results were all very similar to one another we decided to ignore common words in all songs in order to find the distinct words in song lyrics.

In [39]:
```
def find_most_freq_key_words(chart): #This removes unimportant words such as a, a
n, to, the, etc...
    words = ['a', 'an', 'the','that','you','of', 'it', 'my', 'yeah', 'all', 'tha
t', 'to', 'on', 'and', 'oh', 'be', 'in', 'when', 're', 'but', 'can', 'for', 'so']
    stop_words = set(stopwords.words('english'))
    other_words_to_remove = ['oh', 'yeah', 'like', 'go', 'ah', 'ft', 'ooh', 'caus
e', 'got', 'gonna', 'me', 'we', 'don', 'with', 'get', 'doo', 'let', 'wanna', 'us'
, 'la', 'uh', 'want', 'ayy']
    for word in other_words_to_remove:
        stop_words.add(word)
    counts = get_chart_word_count(chart)
    counts = counts[~counts.index.isin(stop_words)]
    return counts.nlargest(10)
```

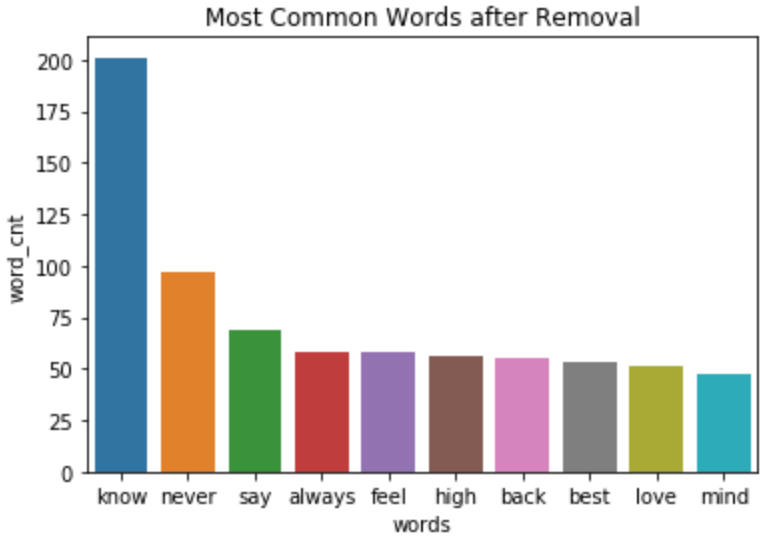
In this method we use stopwords in order to remove some filler or 'fluff' words in english. This is a used commonly for natural language processing. More information about stopwords can be found at [https://pypi.org/project/stop-words/#overview (https://pypi.org/project/stop-words/#overview)](https://pypi.org/project/stop-words/#overview).

This method runs the same way as the most_freq_words method expect that this method makes sure to ignore certain words. The first few words we decided to ignore were articles (a, an, the); as stated earlier these words are stated very commonly in all english text and seeing that these words are commonly used will not give us any new information. The next few words we decided to ignore were prepositions (to, on, of, in, etc...). We want to ignore these words because these words also, do not give much information about the actual topic, and meanings of song lyrics. The third set of words that have been eliminated were interjections (oh, yeah, ah, doo, uh, etc...). The final sets of words were ones we just saw in all of the song genres. Since we wanted to find songs that were unique to each genre, we decided to take out popular common words such as 'you' or 'me'. These words are slightly more descriptive than the other words but again not descriptive enough to be labeled a keyword.

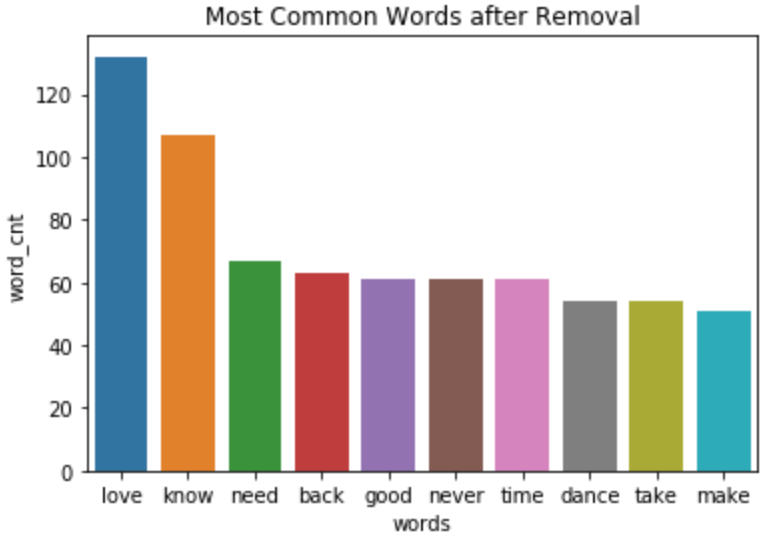
In [79]:
```
def create_word_key_chart(chart):
    freq = find_most_freq_key_words(chart)
    freq = pd.DataFrame({'words':freq.index, 'word_cnt':freq.values})
    sns.barplot(x="words", y="word_cnt", data=freq).set_title("Most Common Words
 after Removal")
```

This method runs exactly the same way as `create_word_chart` except that this method ignores common English words. After running this method on the different song genres, we hypothesize that the song lyrics will be much more distinct from one another.

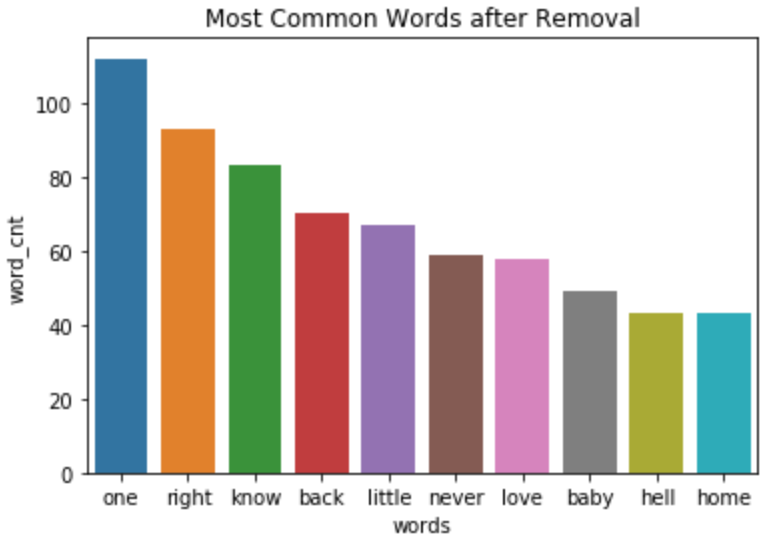
In [80]:
```
create_word_key_chart(rock)
```



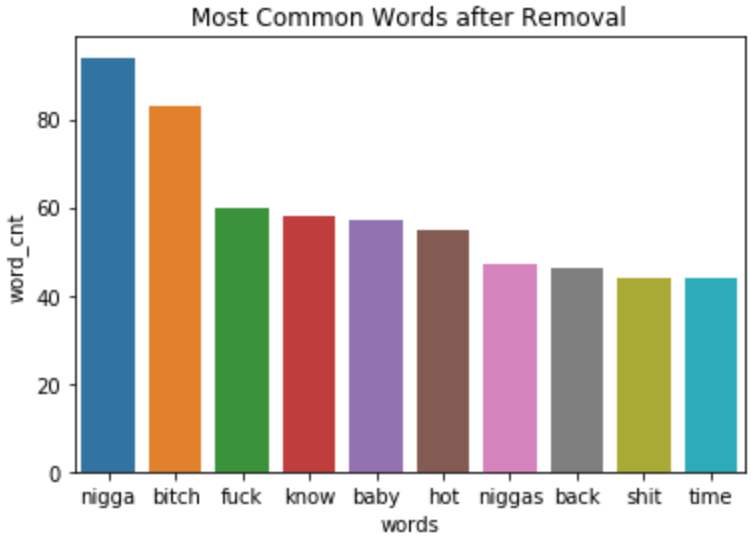
In [81]:
```
create_word_key_chart(pop)
```



In [82]:
```
create_word_key_chart(country)
```

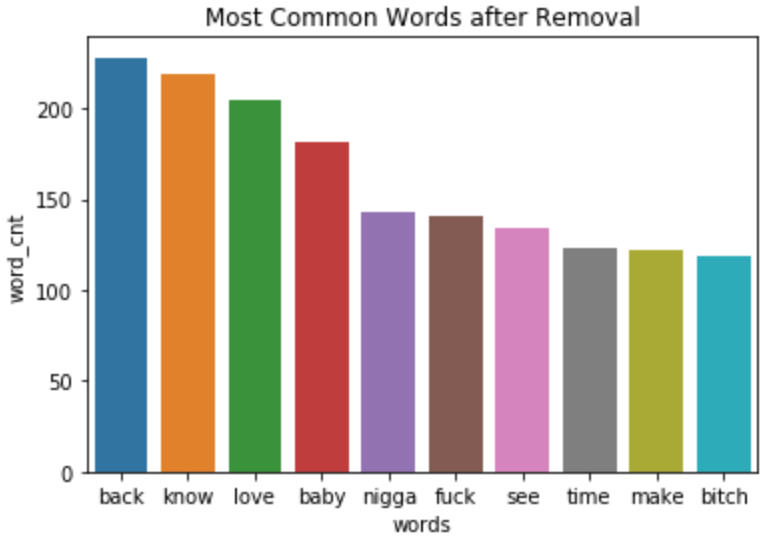


In [83]:
```
create_word_key_chart(rap)
```



This shows that rap songs have a lot of profanity throughout. Half of the words in the top ten list are just profanity.

In [84]:
```
create_word_key_chart(top100)
```



The Top 100 shows some profanity throughout, however, not nearly as much as the rap genre. This plot seems to be more diverse in its word choice. The words 'know' and 'back' seem to be the most popular words used.

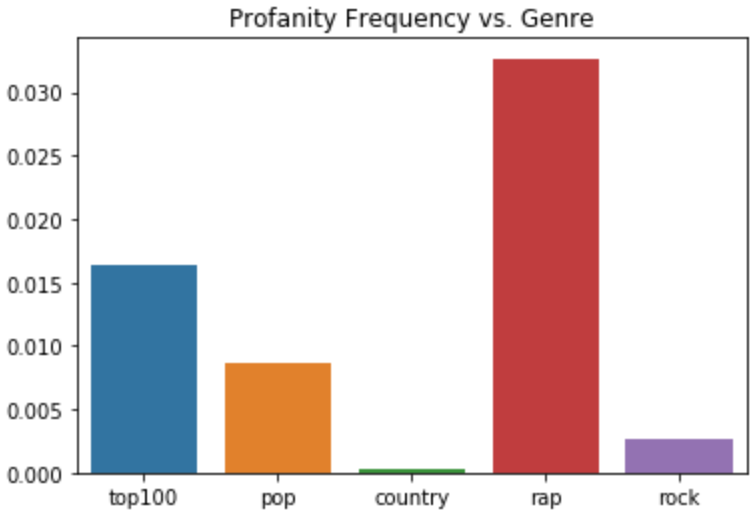
After looking at the most popular words throughout, we saw that many of the words would be considered profanity so we wanted to see how much profanity was in each genre. At this point the bar plot will probably show that rap and Top 100 have the most profanity, however, looking at the specific percentages could be interesting.

In [51]:
```
def get_profan_freq(chart):
    freq = get_chart_word_count(chart)
    profan = {}
    percent = {}

    for row in freq.index:
        if predict([row]):
            profan[row] = freq[row]
            percent[row] = profan[row] / get_lyric_len(chart).sum(axis = 0, skipn
a = True)

    return profan, percent
```

This method checks for profanity using the `profanity-check` python library. Profanity-check uses a linear SMV model, trained with clean and profan text strings. Many profanity checkers just a hard coded list of profane words and thus are less accurate than this one. The profanity-check has a 95 percent accuracy. You can learn about this library here at https://pypi.org/project/profanity-check/ (https://pypi.org/project/profanity-check/).

In [72]:
```
x = np.array(["top100", "pop", "country", "rap", "rock"])
y = []
y.append(sum(get_profan_freq(top100)[1].values()))
y.append(sum(get_profan_freq(pop)[1].values()))
y.append(sum(get_profan_freq(country)[1].values()))
y.append(sum(get_profan_freq(rap)[1].values()))
y.append(sum(get_profan_freq(rock)[1].values()))
y = np.array(y)
sns.barplot(x=x, y=y).set_title("Profanity Frequency vs. Genre")
```

Out[72]:
```
Text(0.5, 1.0, 'Profanity Frequency vs. Genre')
```

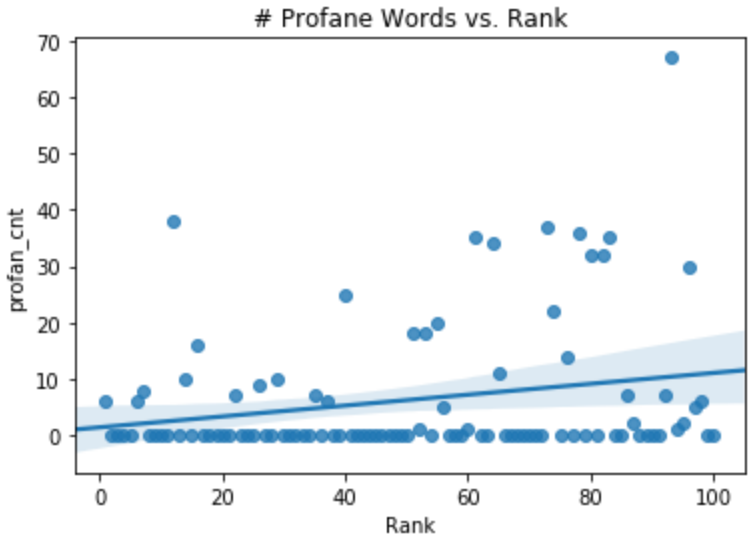


This graph shows the total percentage of lyrics in each chart that are classified as profane. As one would expect the rap chart has a much higher percentage of profane words then any other chart and top100 is the next highest because it has rap songs on it. Country has an extremely low percentage of profane words and rock also has a low percentage of profane words.

In [69]:
```
def profan_cnt_graph(chart):
    doc = create_term_doc_matrix(chart)
    cols = doc.columns

    for i in cols:
        if not predict([i]):
            cols = cols.drop([i])

    doc = doc[cols]
    sum = doc.sum(axis = 1, skipna = True)
    sum = pd.DataFrame({'Title':sum.index, 'profan_cnt':sum.values})

    sum['Rank'] = chart['Rank']
    sns.regplot(x="Rank", y="profan_cnt", data=sum).set_title("# Profane Words vs. Rank")
```

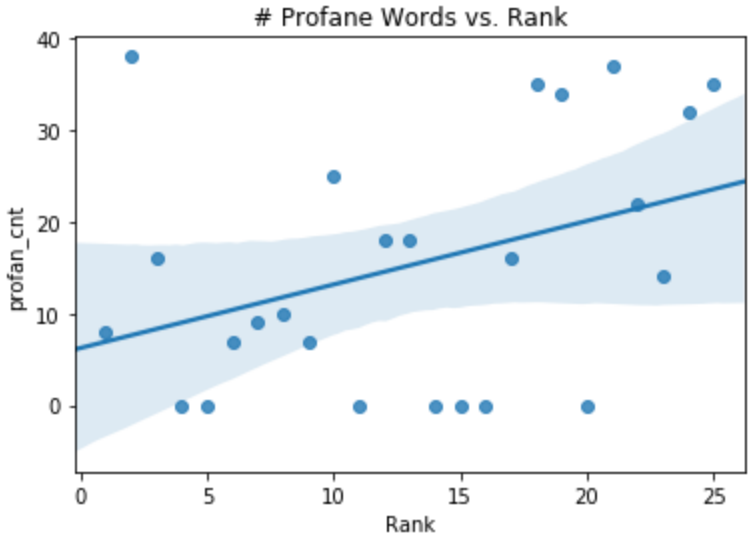
This method allows us to create a graph comparing the rank of the song to the number of profanities throughout. We wanted to see if there is any relationship between the two variables.

In [70]:
```
profan_cnt_graph(top100)
```



For the Top 100 songs, the trend seems to be that higher ranking songs tend to have less profanity. This makes sense, less profanity makes the song easier to play in more places.

In [71]:
```
profan_cnt_graph(rap)
```



For the rap songs, the trend again seems to be that the higher ranking songs have less profanity. This makes sense because of the reasons stated previously however, the slope is way steeper than the Top 100 plot. Also, the number of profanities per song is still higher in this bar plot than the Top 100. Rank 1 rap has around 7 counts of profanity where as rank 1 for the Top 100 has near to 0.

In [297]:
```
#Matching songs based off of lyrics
#A person enters n lyrics and we try to match which song in top100 is the best ma
tch
def find_best_match(words):
    print(tfidf[tfidf.index.isin(words)].sum(axis = 0, skipna = True).nlargest(1
))

num = int(input("How many key words do you want to enter? "))
words = []

for i in range(num - 1):
    words.append(input("Please enter a key word: "))
words.append(input("Please enter your final key word: "))

find_best_match(words)
```

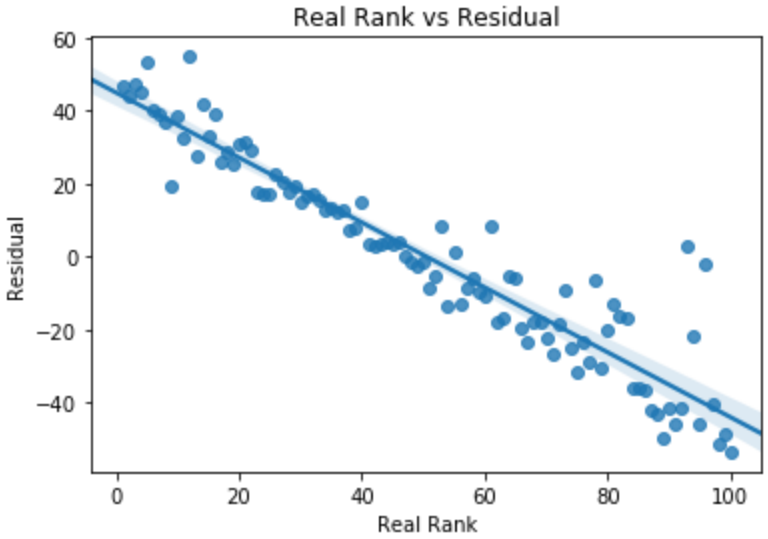
```
How many key words do you want to enter? 3
Please enter a key word: want
Please enter a key word: Christmas
Please enter your final key word: all
Title
All I Want For Christmas Is You    0.506153
dtype: float64
```

With all of this word frequency information, we thought that it would be interesting to try to match songs based off of the lyrics. A person enters *n* words in the song and we try to match which song in the Top 100 is the best match. We do this matching by looking at the word freq chart created previously and finding the song that has the highest percentage of lyrics that contain the entered words.

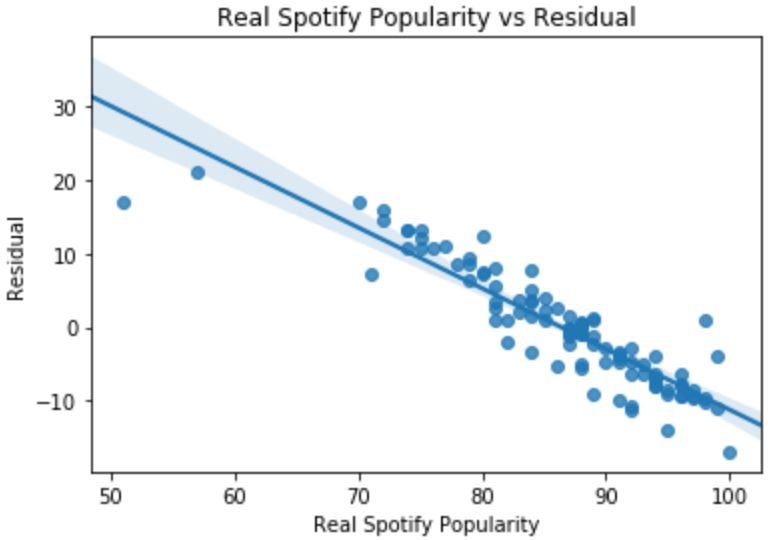
In [316]:
```
def create_linear_model_frequent_words_ranks(chart):
    m = create_term_doc_matrix(chart)
    m['rank'] = chart['Rank'].values
    m['spotify_popularity'] = chart['spotify_popularity'].values
    stop_words = set(stopwords.words('english'))
    other_words_to_remove = ['oh', 'yeah', 'like', 'go', 'ah', 'ft', 'ooh', 'caus
e', 'got', 'gonna', 'me', 'we', 'don', 'with', 'get', 'doo', 'let', 'wanna', 'us'
, 'la', 'uh', 'want', 'ayy']
    for word in other_words_to_remove:
        stop_words.add(word)

    stop_word_columns = [x for x in m.columns if x in stop_words]
    better_m = m.drop(columns=stop_word_columns)
    most_freq = find_most_freq_key_words(chart).index.tolist()

    X = better_m[most_freq]
    y_rank = m['rank']
    reg_rank = LinearRegression().fit(X, y_rank)

    y_spotify_pop = m['spotify_popularity']
    reg_spot_pop = LinearRegression().fit(X, y_spotify_pop)

    reg_rank_residual = reg_rank.predict(X) - y_rank
    real_rank_predict_rank = sns.regplot(x=y_rank, y=reg_rank_residual)
    plt.title("Real Rank vs Residual")
    plt.xlabel("Real Rank")
    plt.ylabel("Residual")
    plt.show(real_rank_predict_rank)
    print("The 10-fold cross validation score for chart ranking is: {}".format(cr
oss_val_score(reg_rank, X, y_rank, cv=10).mean()))

    reg_spot_residual = reg_spot_pop.predict(X) - y_spotify_pop
    real_spot_pop_predict_spot_pop = sns.regplot(x=y_spotify_pop, y=reg_spot_resi
dual)
    plt.title("Real Spotify Popularity vs Residual")
    plt.xlabel("Real Spotify Popularity")
    plt.ylabel("Residual")
    plt.show(real_spot_pop_predict_spot_pop)
    print("The 10-fold cross validation score for Spotify popularity is: {}".form
at(cross_val_score(reg_spot_pop, X, y_spotify_pop, cv=10).mean()))
```

In this model, we tried to see if the makeup of the top 10 most frequent words in a song could be used to predict the chart ranking or the Spotify popularity of that song. This was a focus question for the project. We found that for chart ranking, the makeup of the top 10 most frequent words in a song has little-to-no impact. This was overly the case for most of the genres, but in the case of the pop genre the model was closer to accurate for estimating rank. The story is different for Spotify popularity of that song. For this ranking measurement, it was actually pretty close for most of the genres. This is an interesting difference between the two ranking metrics as we displayed earlier, it seemed the two metrics were very closely related.

In [317]:
```
top100_model_rank = create_linear_model_frequent_words_ranks(top100)
```

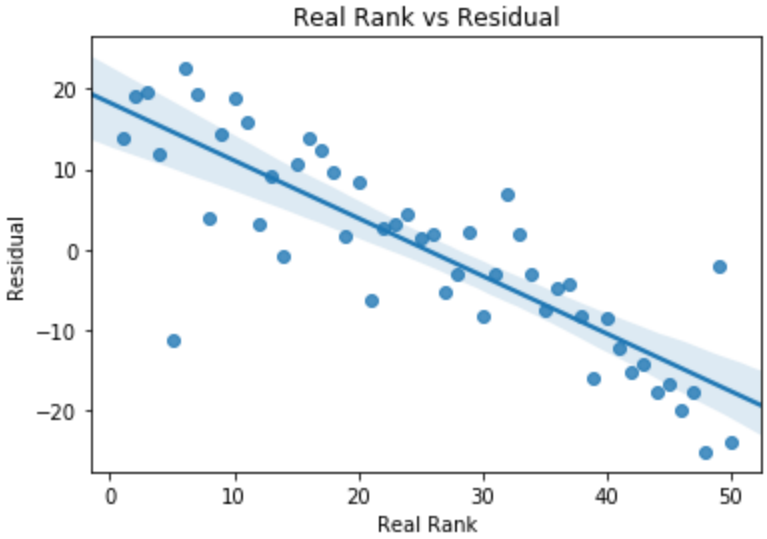



```
The 10-fold cross validation score for chart ranking is: -144.08303485840526
```

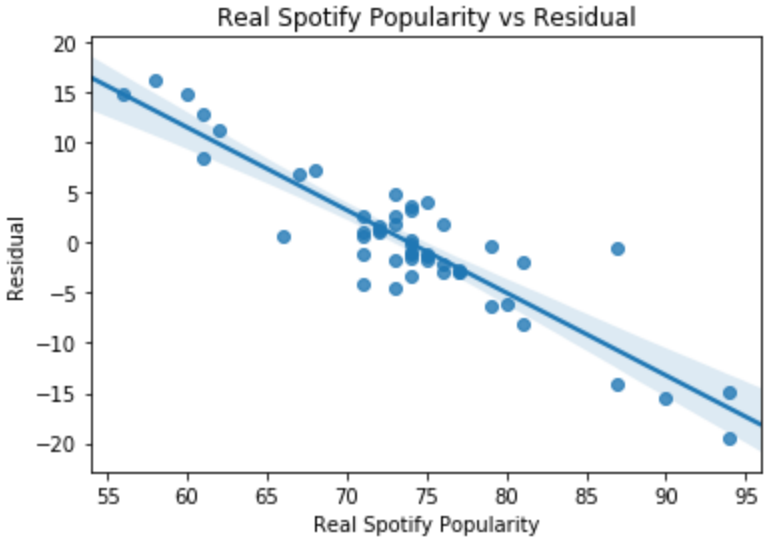



```
The 10-fold cross validation score for Spotify popularity is: -1.3234046545077365
```

In [318]:
```
country_model_rank = create_linear_model_frequent_words_ranks(country)
```

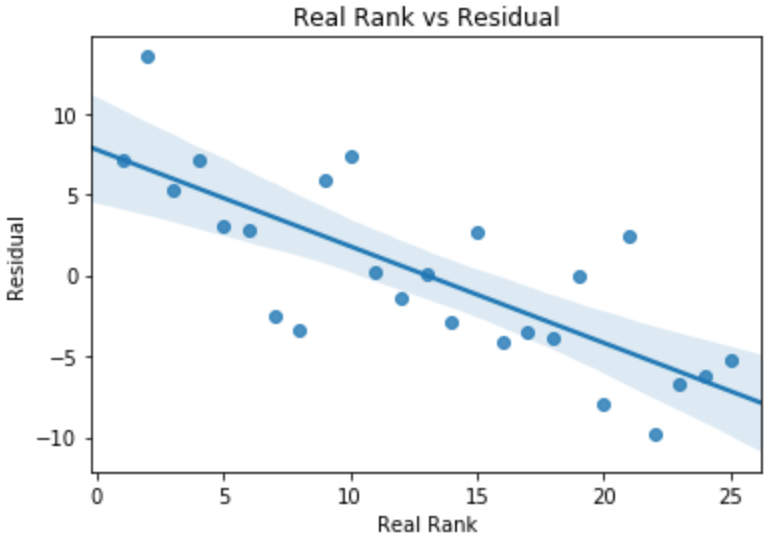


The 10-fold cross validation score for chart ranking is: -1510.9494804072147

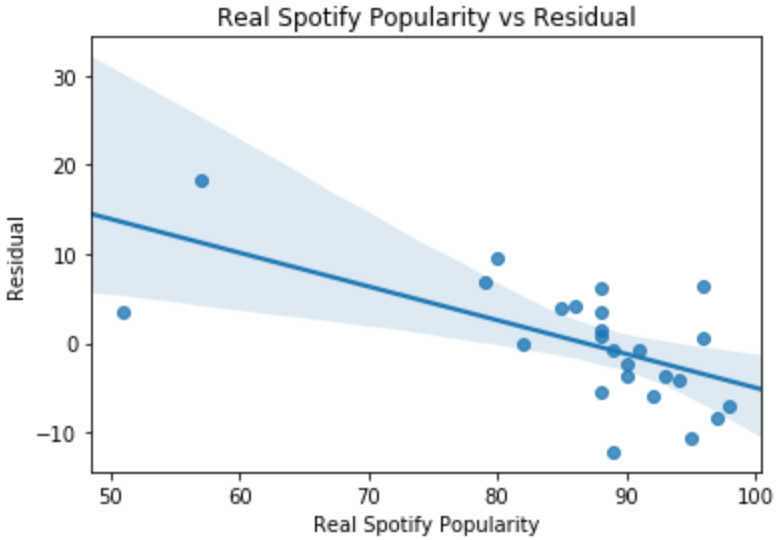


The 10-fold cross validation score for Spotify popularity is: -27.77381194700957

In [319]:
```
rap_model_rank = create_linear_model_frequent_words_ranks(rap)
```

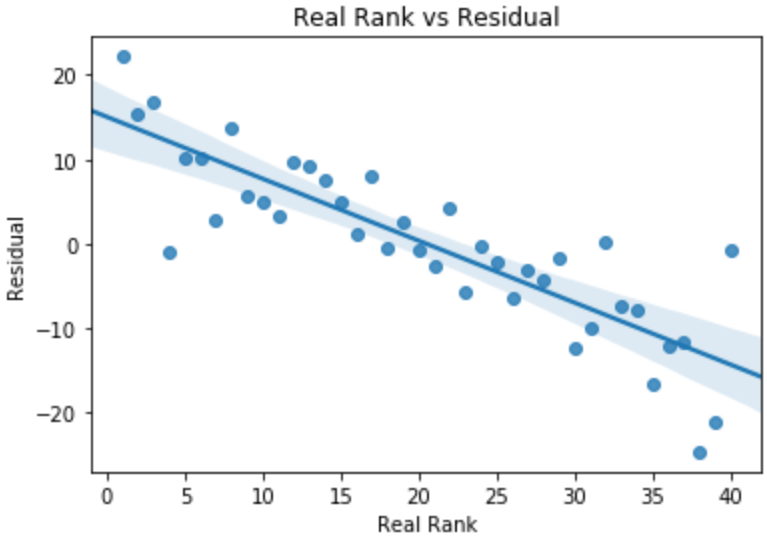



```
The 10-fold cross validation score for chart ranking is: -1095.7605426360697
```

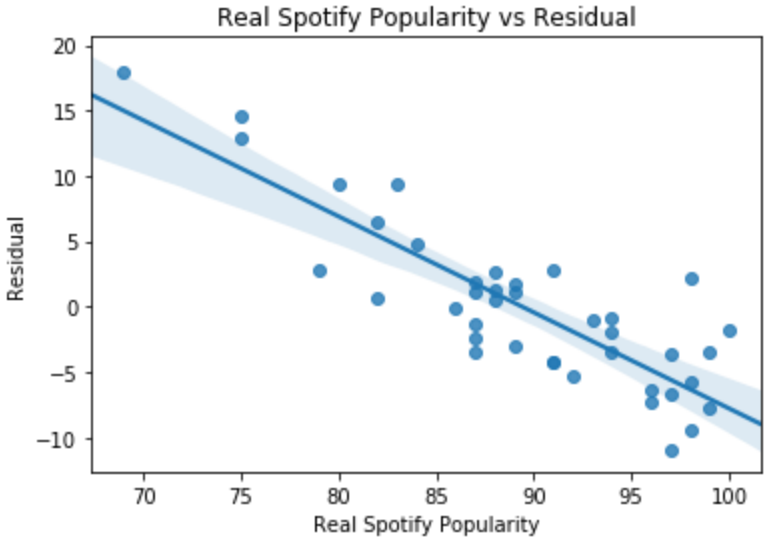



```
The 10-fold cross validation score for Spotify popularity is: -18.275751114451246
```

In [320]:
```
pop_model_rank = create_linear_model_frequent_words_ranks(pop)
```

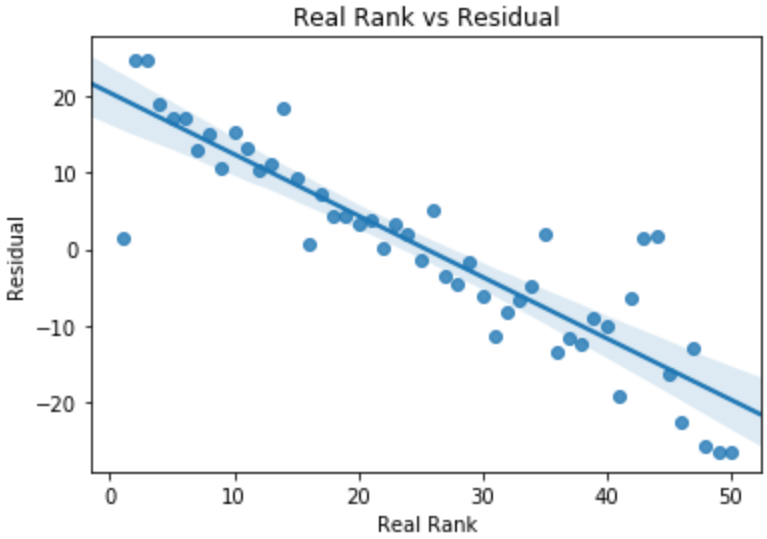



```
The 10-fold cross validation score for chart ranking is: -192.9379512128733
```

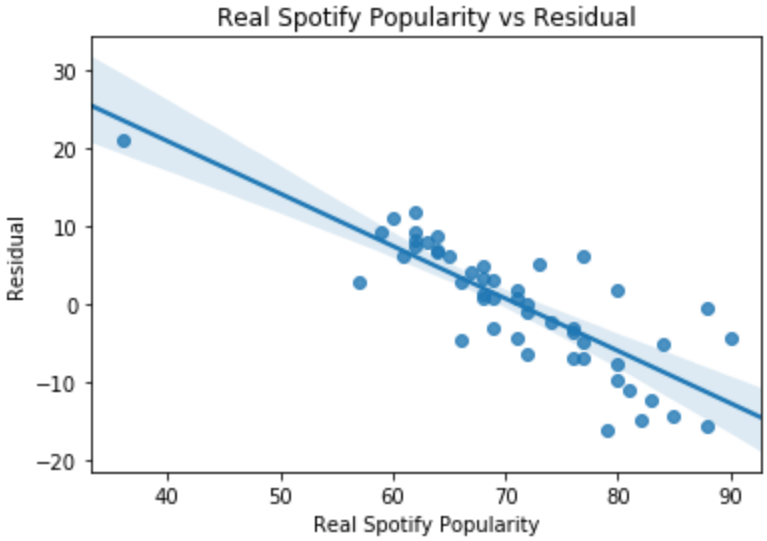



```
The 10-fold cross validation score for Spotify popularity is: -6.093570953447446
```

In [321]:
```
rock_model_rank = create_linear_model_frequent_words_ranks(rock)
```




```
The 10-fold cross validation score for chart ranking is: -249.39997254214558
```

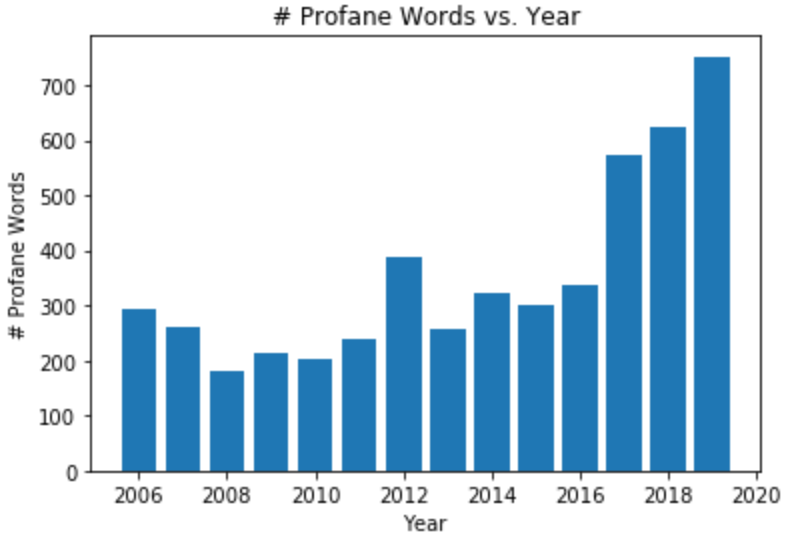



```
The 10-fold cross validation score for Spotify popularity is: -1.564121970972092
6
```

In [248]:
```
conn = sqlite3.connect('billboard100.db')

df = pd.read_sql_query("SELECT * FROM billboard100", conn)

display(df.head(10))

profanity_per_year = {}

for year in range(2006,2020):
    top100inYear = df[df["Year"] == year]
    profane, _ = get_profan_freq(top100inYear)
    profanity_per_year[year] = profane
```

	index	Rank	Title	Artist	Year	lyrics
0	0	1	Bad Day	Daniel Powter	2006.0	Where is the moment we needed the most?\nYou k...
1	1	2	Temperature	Sean Paul	2006.0	The gyal dem Schillaci, Sean da Paul\nSome gi...
2	2	3	Promiscuous	Nelly Furtado Featuring Timbaland	2006.0	Am I throwin' you off?\nNope\nDidn't think so\...
3	3	4	You're Beautiful	James Blunt	2006.0	My life is brilliant\n\nMy life is brilliant, ...
4	4	5	Hips Don't Lie	Shakira Featuring Wyclef Jean	2006.0	Ladies up in here tonight\nNo fighting (We got...
5	5	6	Unwritten	Natasha Bedingfield	2006.0	I am unwritten, can't read my mind\nI'm undefi...
6	6	7	Crazy	Gnarls Barkley	2006.0	I remember when\nI remember, I remember when I...
7	7	8	Ridin'	Chamillionaire Featuring Krayzie Bone	2006.0	Hi. I'm the Rap Critic. Let's talk about Chami...
8	8	9	SexyBack	Justin Timberlake	2006.0	I'm bringin' sexy back (Yeah)\nThem other boys...
9	9	10	Check On It	Beyonce Featuring Slim Thug	2006.0	Swizz Beatz\nDC, Destiny Child (Slim Thug)\n\n...

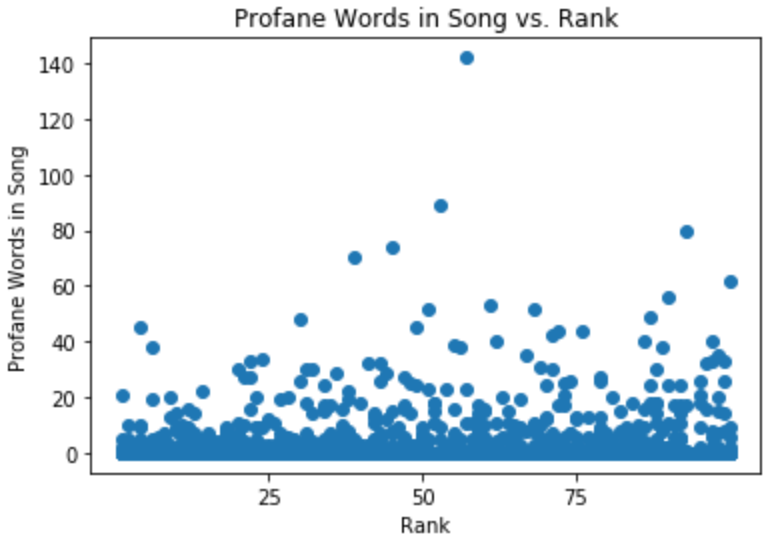
In [66]:
```
profanity_in_top_charts = {}
for year in range(2006,2020):
    p = profanity_per_year[year]
    profanity_in_top_charts[year] = sum(p.values())

plt.bar(profanity_in_top_charts.keys(), profanity_in_top_charts.values())
plt.xlabel("Year")
plt.ylabel("# Profane Words")
plt.title("# Profane Words vs. Year")
plt.show()
```



To see the variation in the use of profanity over the years, we simply count the number of what are considered “bad words” using a library called `profanity-check`. This count is calculated for 2006-2019, and plotted using `matplotlib`. We can see a general upward trend in the use of profanity across the years, with a particularly interesting spike in 2017.

In [60]:
```
doc = create_term_doc_matrix(df)
cols = doc.columns

for i in cols:
    if not predict([i]):
        cols = cols.drop([i])

doc = doc[cols]
s = doc.sum(axis = 1, skipna = True)
s = pd.DataFrame({'Title':s.index, 'profan_cnt':s.values})

s['Rank'] = df['Rank']
```

In [65]:
```
plt.scatter(s["Rank"], s["profan_cnt"])
plt.xlabel("Rank")
plt.ylabel("Profane Words in Song")
plt.xticks([24, 49, 74])
plt.title("Profane Words in Song vs. Rank")
plt.show()
```



To examine the second relationship we compare the profanity count per song to that song's ranking for all of the songs as a whole, with the intention of revealing a relationship between profanity usage and song ranking. As can be seen in the plot above, there does not appear to be an immediate relationship between profanity level and song ranking.

Insights and Conclusions

There are several interesting insights and answers we gathered from the work conducted above. First, let's aggregate our answers to our original research questions:

How similar are songs on the Billboard Top 100 chart and genre specific charts to each other in terms of lyrics? Answer: Somewhat similar! Once removing common words, words like 'know' and 'love' appear in almost all the top word lists for each genre. As one would expect, the common or 'stop' words were very similar in frequency between the charts. There were expected overlaps between closely related genres like rock and country, and the Top 100 most frequent words list was, for the most part, an amalgamation of words from the other genres most frequent words lists.

How has the level of profanity present in songs on the Billboard Top 100 chart changed over the last 14 years, and does this correlate with Billboard ranking? Answer: Over the past 14 years, profanity has increased in songs on the Billboard Top 100 chart. The increase has been particularly high in the past couple of years. We found that profanity does not correlate with the Billboard ranking, with a high skew towards 'no profanity'. This could be indicative to the fact that they are in the Top 100, but we can not statistically affirm that without more data.

What are the most common words found in the Billboard Top 100 chart and genre specific charts? Are they the same or does it differ based on genre? Answer: The most common words found in the current Billboard Top 100 chart and genre specific charts can be found in the histograms in the analysis above. We found that it does differ based on genre, but there is certainly overlap with common words.

Can we train a model to guess what the rank of a current song on the Billboard Top 100 chart is based solely on the most frequent lyrics of its chart? Answer: No, not really. The linear regression model we built had a terrible cross validation score for ranking. We plotted the residuals for each of the charts and, while there certainly was a range of residuals depending on genre, they were all very high and clearly, readily observable that there was no strong relation. Interestingly, there seemed to be a stronger correlation and ability to predict for Spotify popularity using the most frequent lyrics of that chart.

Can we guess which song in the Billboard Top 100 chart somebody is describing by only having them input a few words? Answer: Surprisingly, kind-of! We used the "interactive" (in the Jupyter notebook, not on the website) model to see if it could guess the right song with just a few words and it most did if the lyrics given were genre-specific enough. An example of this can be seen below the relevant code in the analysis above. This was mostly just an interesting question we had that we wanted a quick answer to and did not bother to explore the data science here in great depth.

Does the popularity of a song on Spotify correlate to the ranking of the song on the Billboard Top 100? Answer: Certainly. The well-ranked songs were also heavily listened to recently on Spotify. This was especially true with the very well-ranked songs in the first through tenth ranking spots for all of the genre charts and the Top 100 chart. This was more so the case with specific genres. There were, of course, outliers present, but the general trend was a high popularity rating between 70 and 100 for most of the Top 100 and a similar range for the genre specific charts.

From these answers, we can provide the following "policy" recommendation to music artists trying to make it big: words don't really matter, but your profanity is more accepted now. There is also a correlation between ranking and Spotify popularity, so try to get more listens on Spotify and climb the Top 100 ladder (makes sense considering Billboard takes into account streaming views). It's also important to note that it is better to be unique, as evidenced by the low similarities in lyrics between songs on charts. If you are desperate to use the high frequency words in order to make it on the charts, try writing a song called "Know Love" that includes an abundance of those words. We hope that this was an interesting and useful read!

Extra Resources

- Linear Regression Guide: https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-scikit-learn-83a8f7ae2b4f (https://towardsdatascience.com/a-beginners-guide-to-linear-regression-in-python-with-

scikit-learn-83a8f7ae2b4f)

- Inspiration and Similar Work: https://pudding.cool/2017/05/song-repetition/ (https://pudding.cool/2017/05/song-repetition/)
- Scikit Tutorial: https://scikit-learn.org/stable/tutorial/basic/tutorial.html (https://scikit-learn.org/stable/tutorial/basic/tutorial.html)
- Those Interested in NLTK: https://www.nltk.org/book/ (https://www.nltk.org/book/)