General guidelines when writing chart hits for the 21st Century

By: Darrin Cates, Kimberly Erin, and Rachel Pontow

- Is there a formula for creating a hit song?
- Are there common components that chart-topping songs have?
- Has this changed over time? If so how?

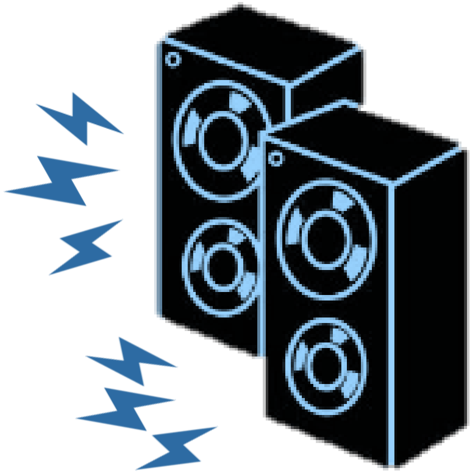
- How does the positivity/negativity of lyrics affect popularity?
- How does word repetition affect popularity?
- How does the beats per minute affect popularity?
- How does acousticness, duration, energy, instrumentalness, loudness, and valence affect popularity?
- How have these changed over time?

Summary

- What did we find?
  - Acoustic songs don’t tend to be as popular
  - Negative lyrics, while around 25% in popularity, are becoming more popular
  - Top songs are increasingly using repetition
  - Song length is at about 2.5 minutes to 5 minutes with few exception after 2010
  - Songs that have high energy and you can dance to have will do better than songs that you can’t dance to
  - Instrumentals don’t do very well on the chart toppers-best to stay away
  - Keep you music LOUD
  - Chart toppers can be happy or sad sounding. The sound seems to have no impact on chart toppers these days
  - Average 120 BPM
- Has this changed over time? If so how?

Data

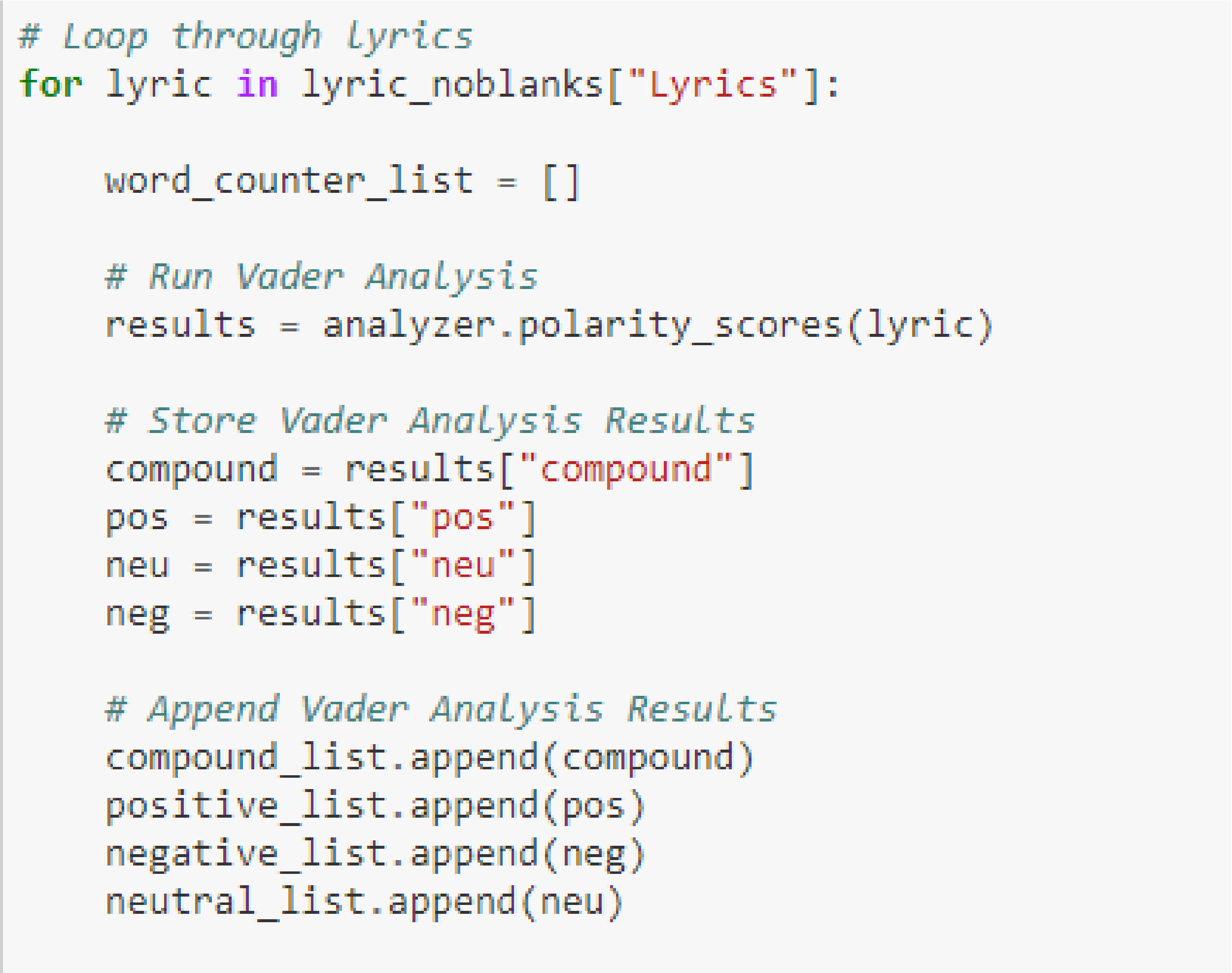
- Song Source:
  - Billboard 1964-2015 Songs + Lyrics csv from Kaggle.com
- Sentiment Analysis:
  - VADER Sentiment Analysis
- Beats Per Minute:
  - Get Song Bpm API Beats Per Minute:
- Spotify Attributes (Spotify API):
  - Acousticness, Danceability, Duration, Energy, Instrumentalness, Loudness, Valence



Data

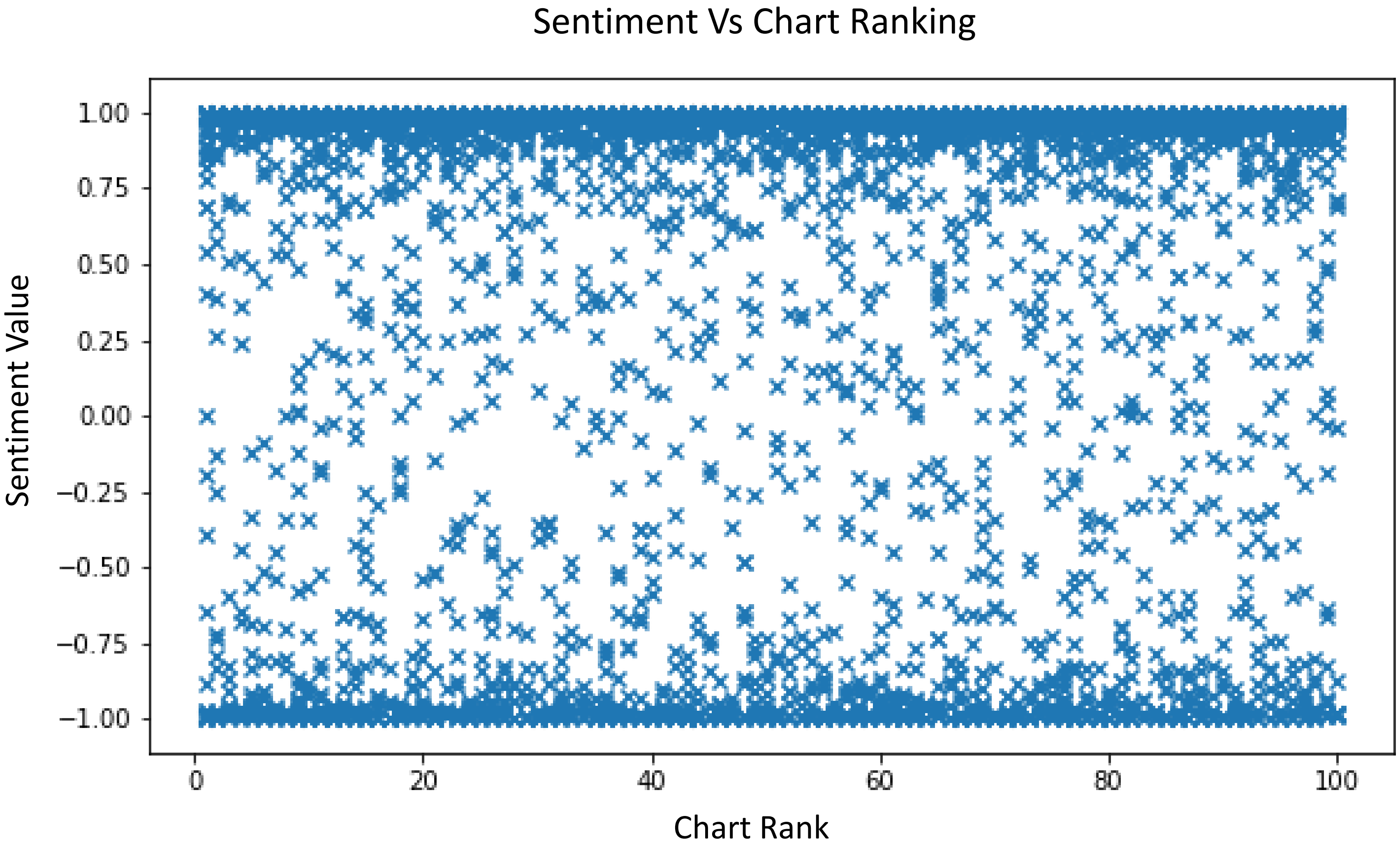
- The Process
  - Asked ourselves what we wanted to learn
  - Identified the data sources
  - Worked through a retrieval plan
  - Cleaned the data
  - Analyzed trends
- The Struggles
  - Complicated API’s (Spotify)
  - Unclear Documentation (Get Song BPM)
  - Slowness of API calls
- Insights
  - Use time delays to make API calls. 5 seconds in a delay, makes a world of difference

Sentiment



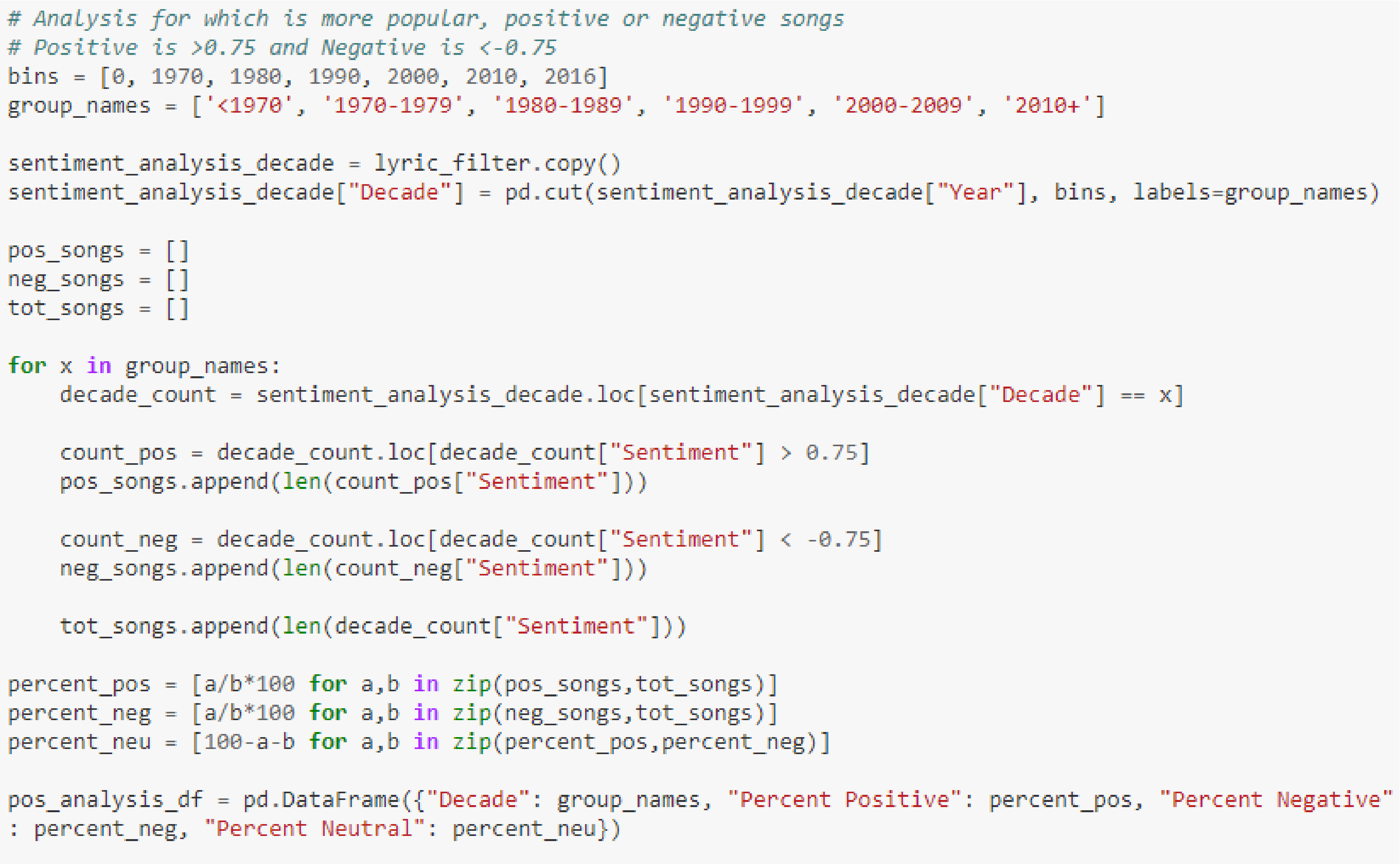
```
# Loop through lyrics
for lyric in lyric_noblanks["Lyrics"]:

    word_counter_list = []

    # Run Vader Analysis
    results = analyzer.polarity_scores(lyric)

    # Store Vader Analysis Results
    compound = results["compound"]
    pos = results["pos"]
    neu = results["neu"]
    neg = results["neg"]

    # Append Vader Analysis Results
    compound_list.append(compound)
    positive_list.append(pos)
    negative_list.append(neg)
    neutral_list.append(neu)
```

Sentiment

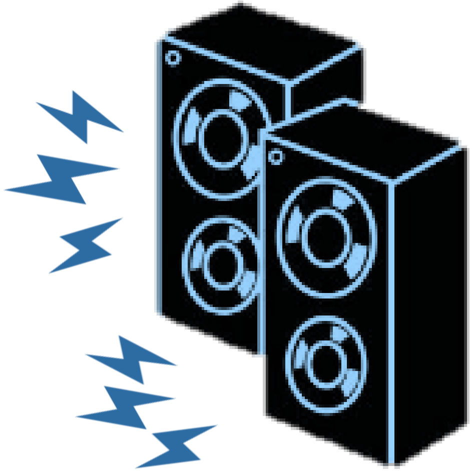


- Positive: > 0.75
- Negative: <-0.75

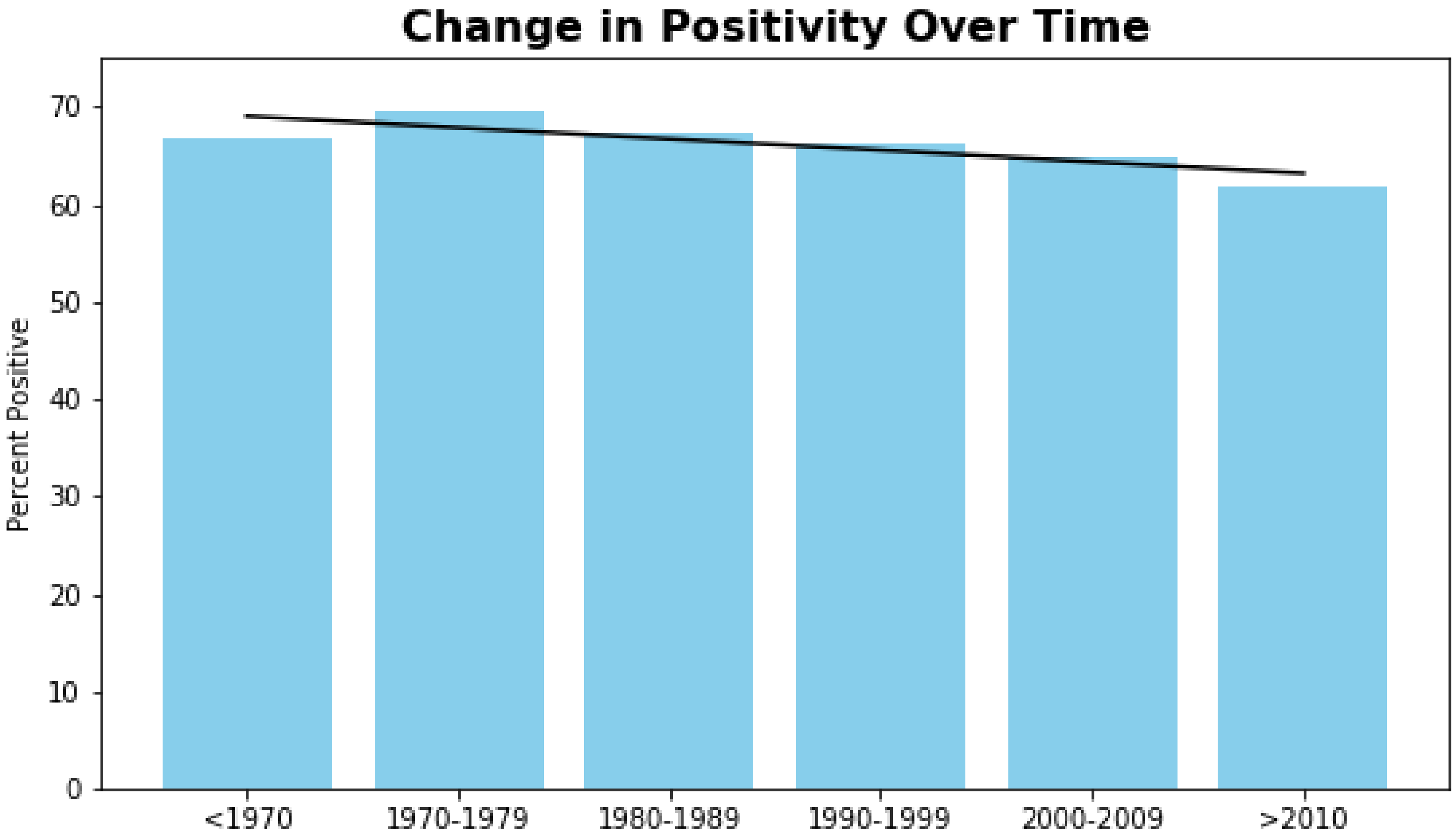
Sentiment



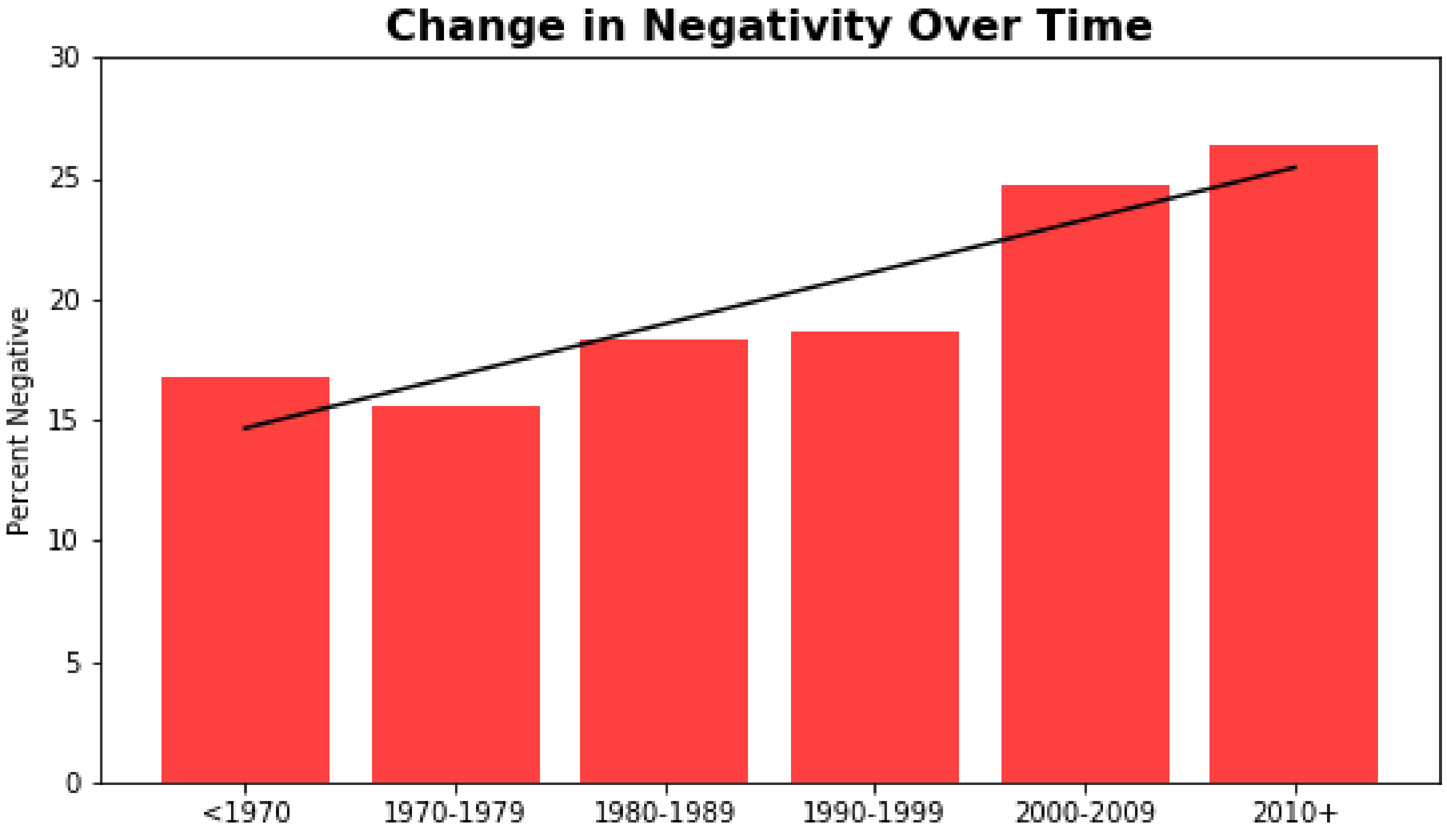
```
# Analysis for which is more popular, positive or negative songs
# Positive is >0.75 and Negative is <-0.75
bins = [0, 1970, 1980, 1990, 2000, 2010, 2016]
group_names = ['<1970', '1970-1979', '1980-1989', '1990-1999', '2000-2009', '2010+']

sentiment_analysis_decade = lyric_filter.copy()
sentiment_analysis_decade["Decade"] = pd.cut(sentiment_analysis_decade["Year"], bins, labels=group_names)

pos_songs = []
neg_songs = []
tot_songs = []

for x in group_names:
    decade_count = sentiment_analysis_decade.loc[sentiment_analysis_decade["Decade"] == x]

    count_pos = decade_count.loc[decade_count["Sentiment"] > 0.75]
    pos_songs.append(len(count_pos["Sentiment"]))

    count_neg = decade_count.loc[decade_count["Sentiment"] < -0.75]
    neg_songs.append(len(count_neg["Sentiment"]))

    tot_songs.append(len(decade_count["Sentiment"]))

percent_pos = [a/b*100 for a,b in zip(pos_songs,tot_songs)]
percent_neg = [a/b*100 for a,b in zip(neg_songs,tot_songs)]
percent_neu = [100-a-b for a,b in zip(percent_pos,percent_neg)]

pos_analysis_df = pd.DataFrame({"Decade": group_names, "Percent Positive": percent_pos, "Percent Negative"
: percent_neg, "Percent Neutral": percent_neu})
```



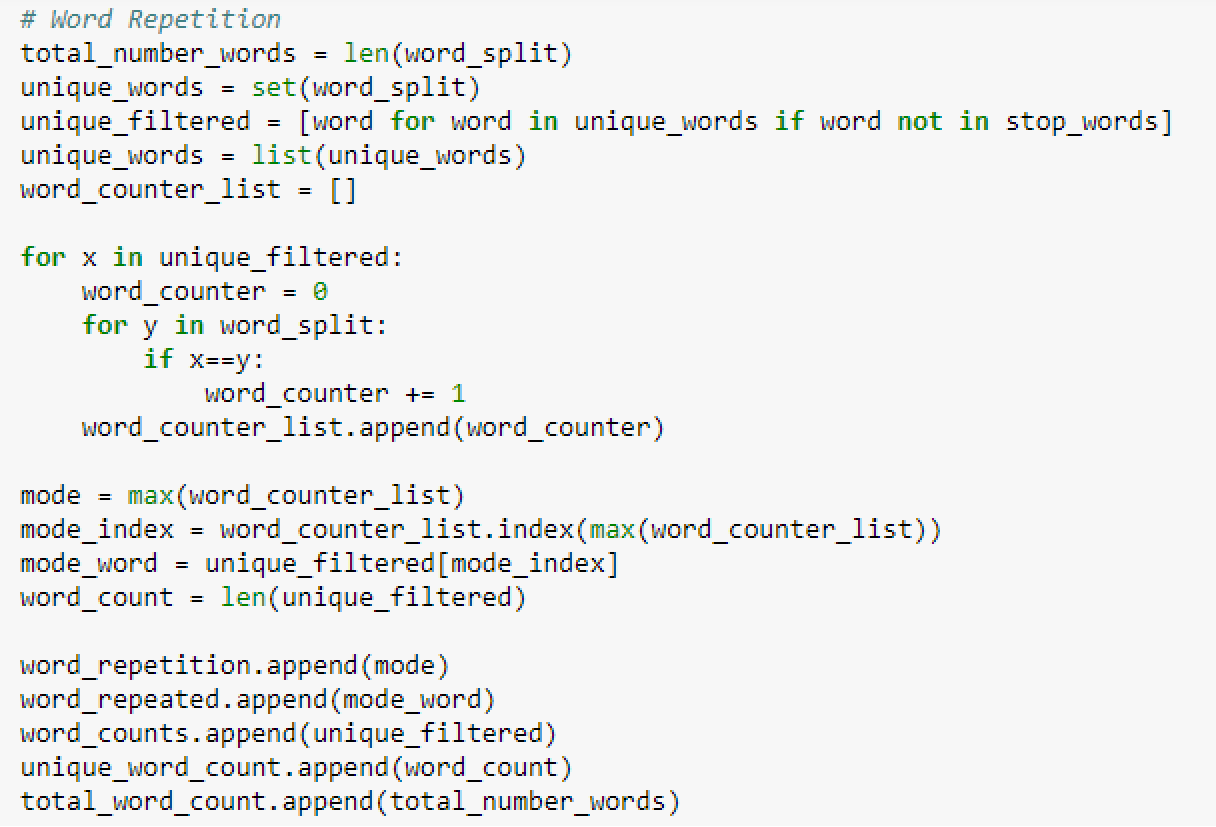
Positivity


Change in Positivity Over Time
Percent Positive
0
10
20
30
40
50
60
70
<1970
1970-1979
1980-1989
1990-1999
2000-2009
>2010

Negativity

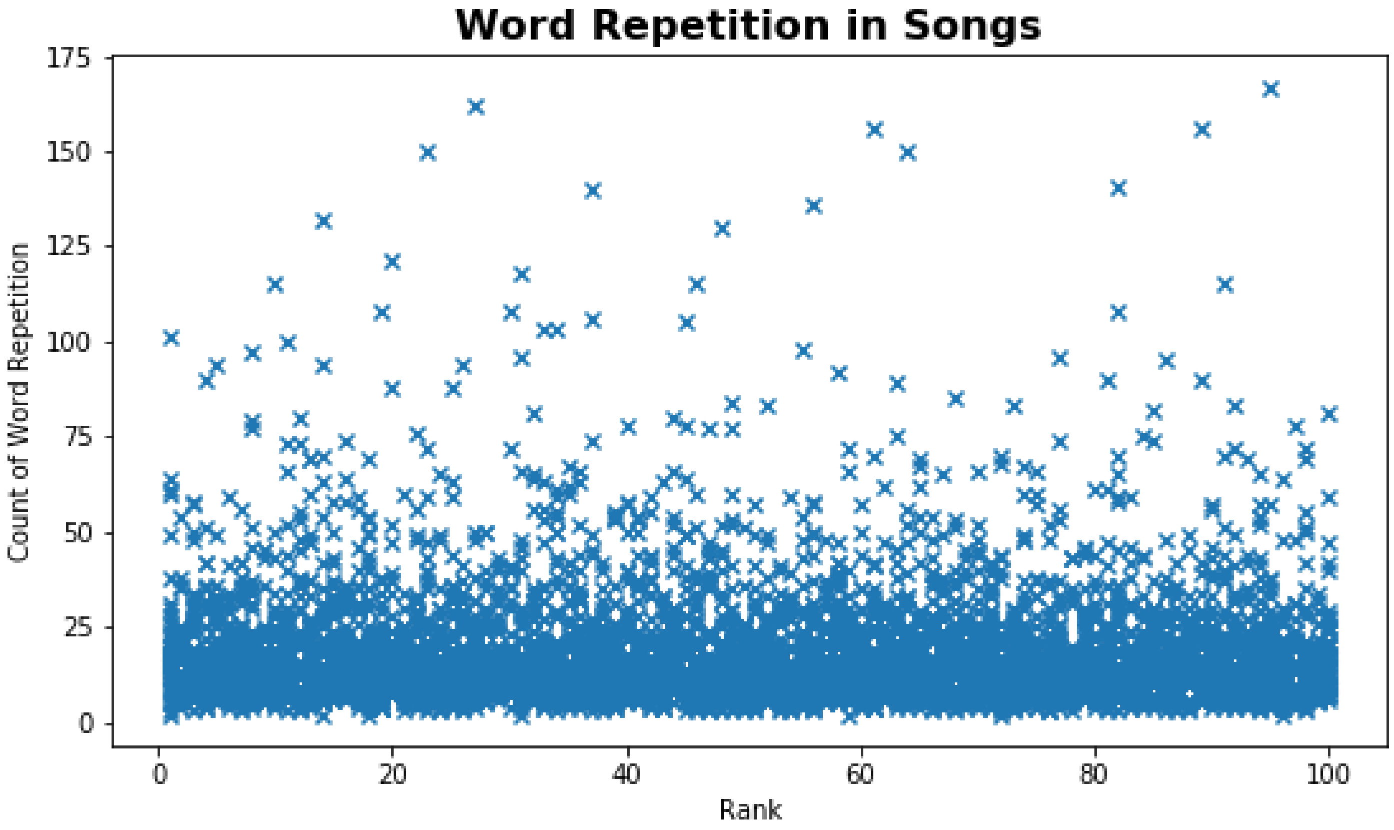

Change in Negativity Over Time
Percent Negative
30
25
20
15
10
5
0
<1970
1970-1979
1980-1989
1990-1999
2000-2009
2010+

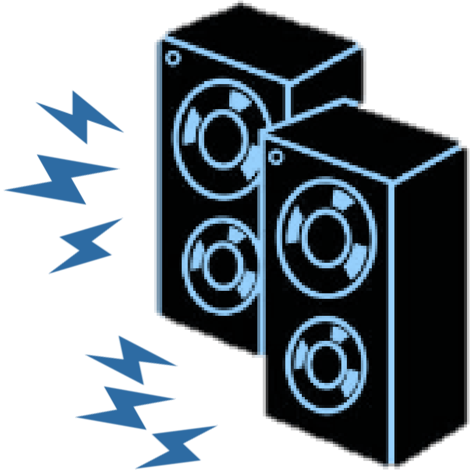
Repetition



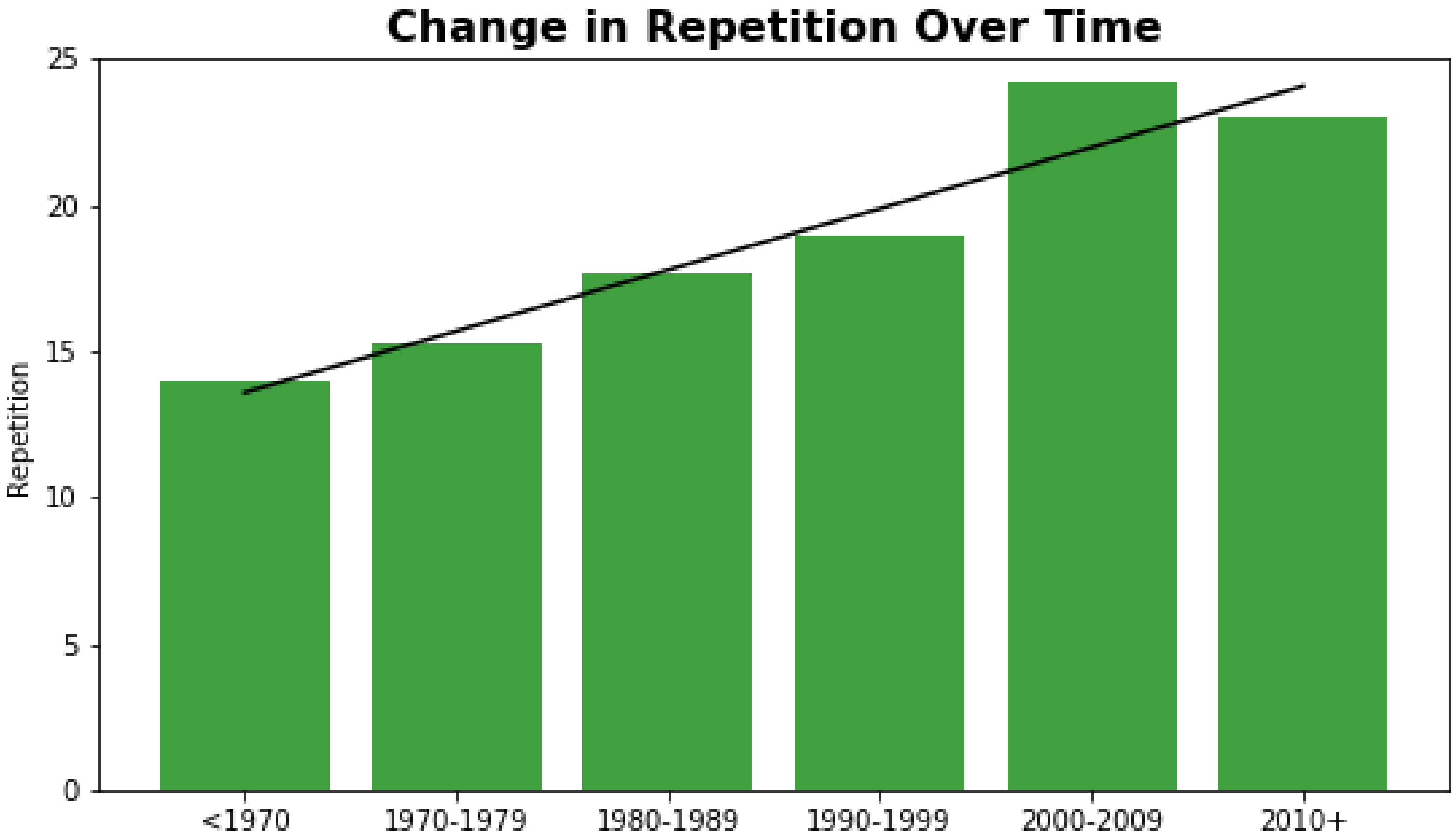
```python
# Word Repetition
total_number_words = len(word_split)
unique_words = set(word_split)
unique_filtered = [word for word in unique_words if word not in stop_words]
unique_words = list(unique_words)
word_counter_list = []

for x in unique_filtered:
    word_counter = 0
    for y in word_split:
        if x==y:
            word_counter += 1
    word_counter_list.append(word_counter)

mode = max(word_counter_list)
mode_index = word_counter_list.index(max(word_counter_list))
mode_word = unique_filtered[mode_index]
word_count = len(unique_filtered)

word_repetition.append(mode)
word_repeated.append(mode_word)
word_counts.append(unique_filtered)
unique_word_count.append(word_count)
total_word_count.append(total_number_words)
```

Repetition

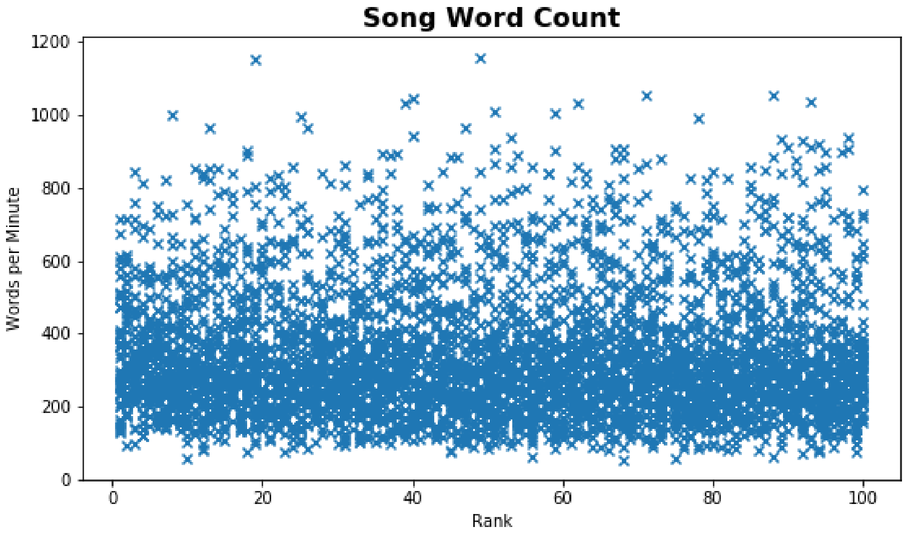

Word Repetition in Songs
Count of Word Repetition
0
25
50
75
100
125
150
175
Rank
0
20
40
60
80
100

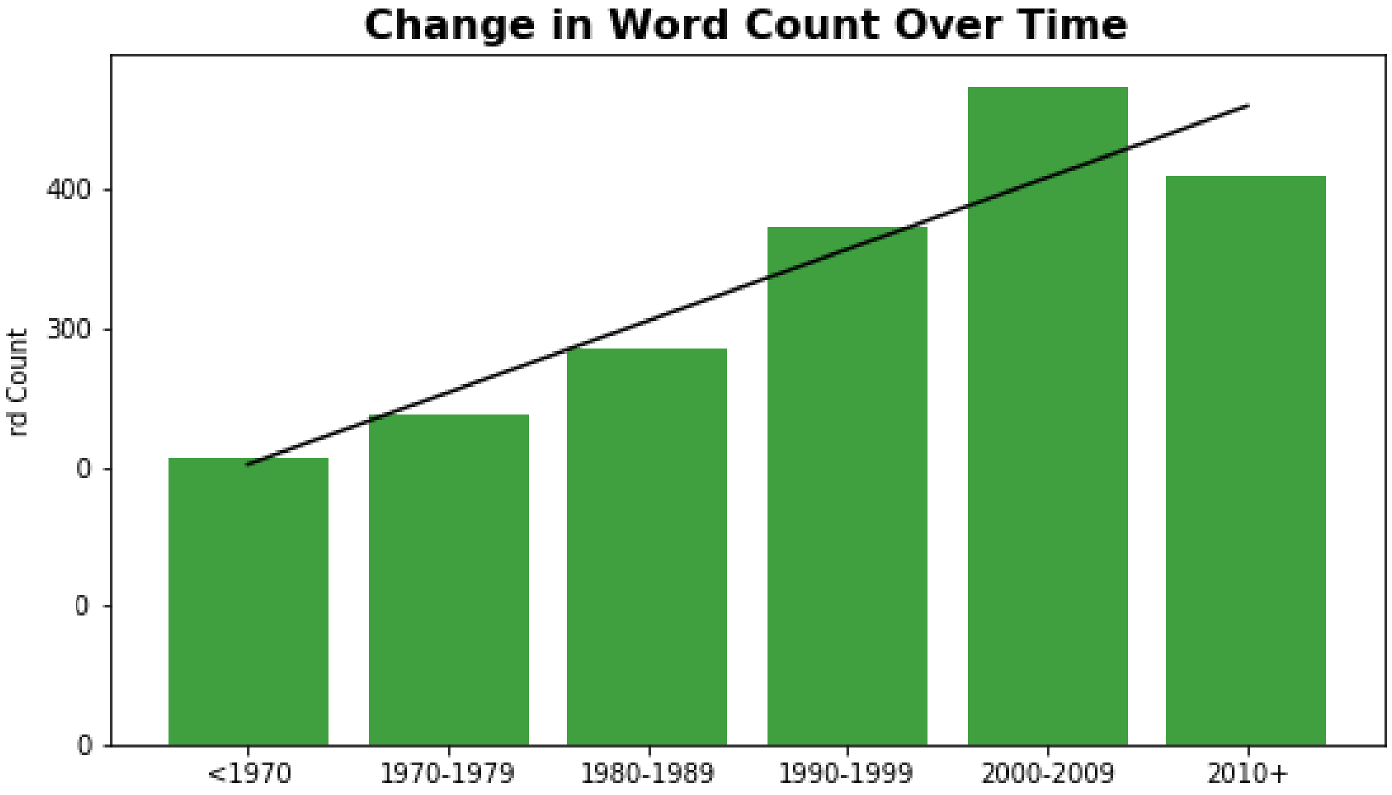


Repetition

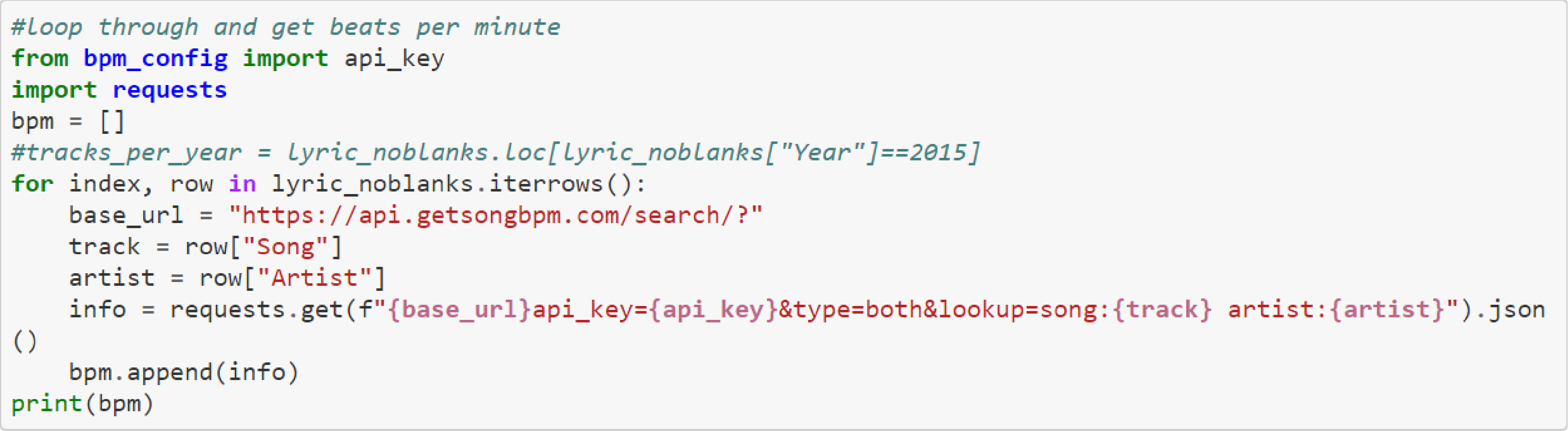

Change in Repetition Over Time
Repetition
0
5
10
15
20
25
<1970
1970-1979
1980-1989
1990-1999
2000-2009
2010+

Word

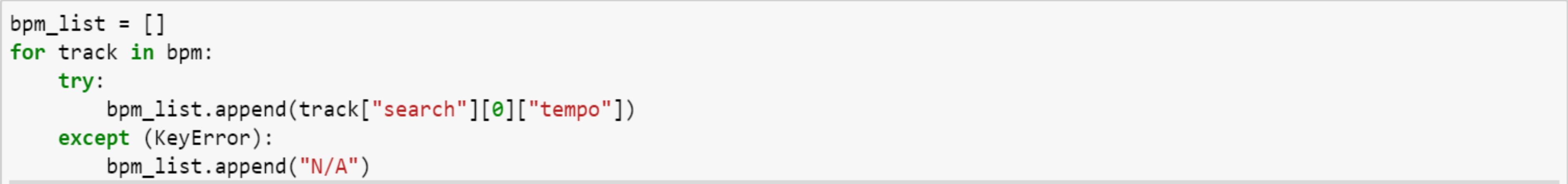

Song Word Count
Words per Minute
0
200
400
600
800
1000
1200
Rank
0
20
40
60
80
100


Change in Word Count Over Time
rd Count
400
300
0
0
0
<1970
1970-1979
1980-1989
1990-1999
2000-2009
2010+

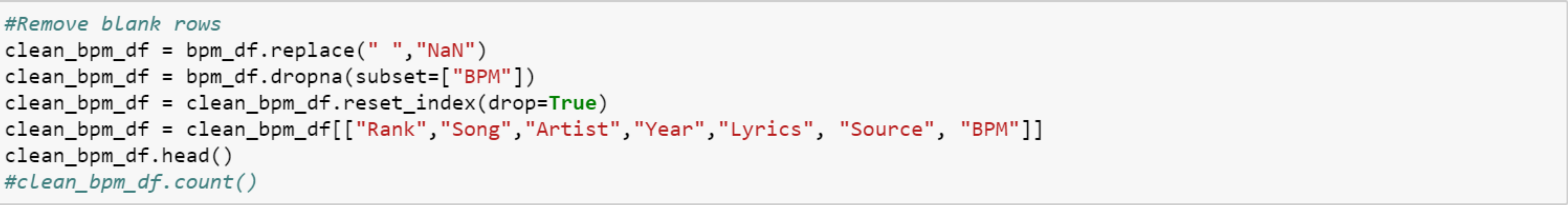
Beats



```
#loop through and get beats per minute
from bpm_config import api_key
import requests
bpm = []
#tracks_per_year = lyric_noblanks.loc[lyric_noblanks["Year"]==2015]
for index, row in lyric_noblanks.iterrows():
    base_url = "https://api.getsongbpm.com/search/?"
    track = row["Song"]
    artist = row["Artist"]
    info = requests.get(f"{base_url}api_key={api_key}&type=both&lookup=song:{track} artist:{artist}").json
()
    bpm.append(info)
print(bpm)
```

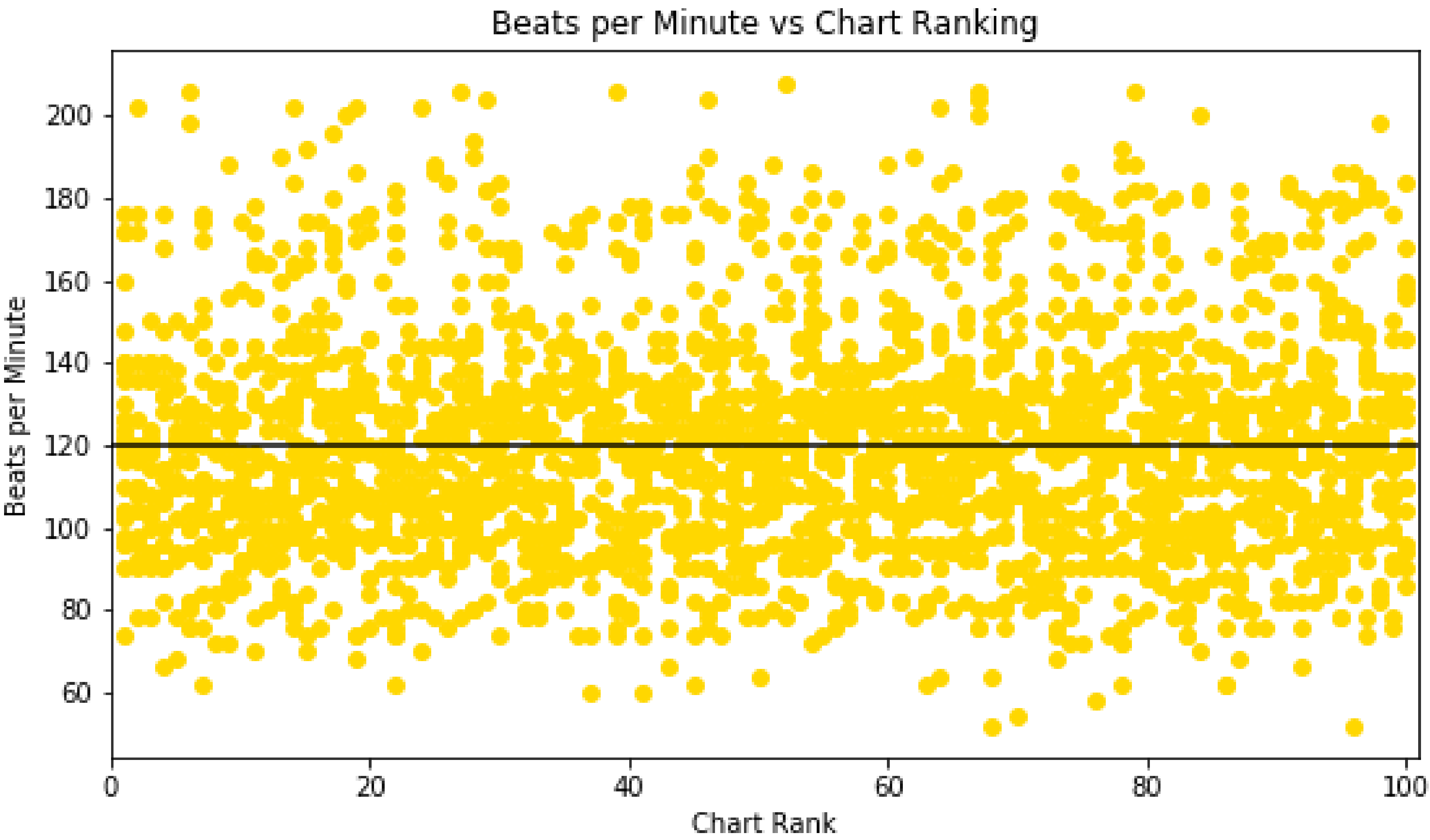


```
bpm_list = []
for track in bpm:
    try:
        bpm_list.append(track["search"][0]["tempo"])
    except (KeyError):
        bpm_list.append("N/A")
```

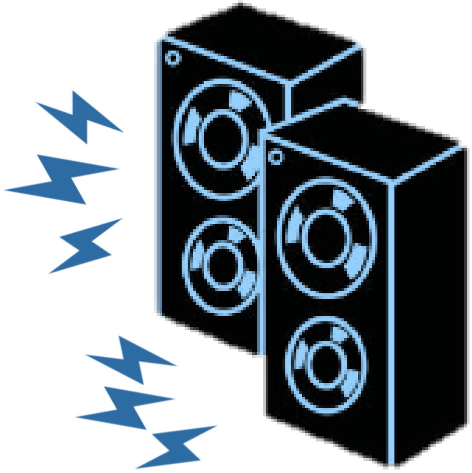


```
#Remove blank rows
clean_bpm_df = bpm_df.replace(" ","NaN")
clean_bpm_df = bpm_df.dropna(subset=["BPM"])
clean_bpm_df = clean_bpm_df.reset_index(drop=True)
clean_bpm_df = clean_bpm_df[["Rank","Song","Artist","Year","Lyrics", "Source", "BPM"]]
clean_bpm_df.head()
#clean_bpm_df.count()
```

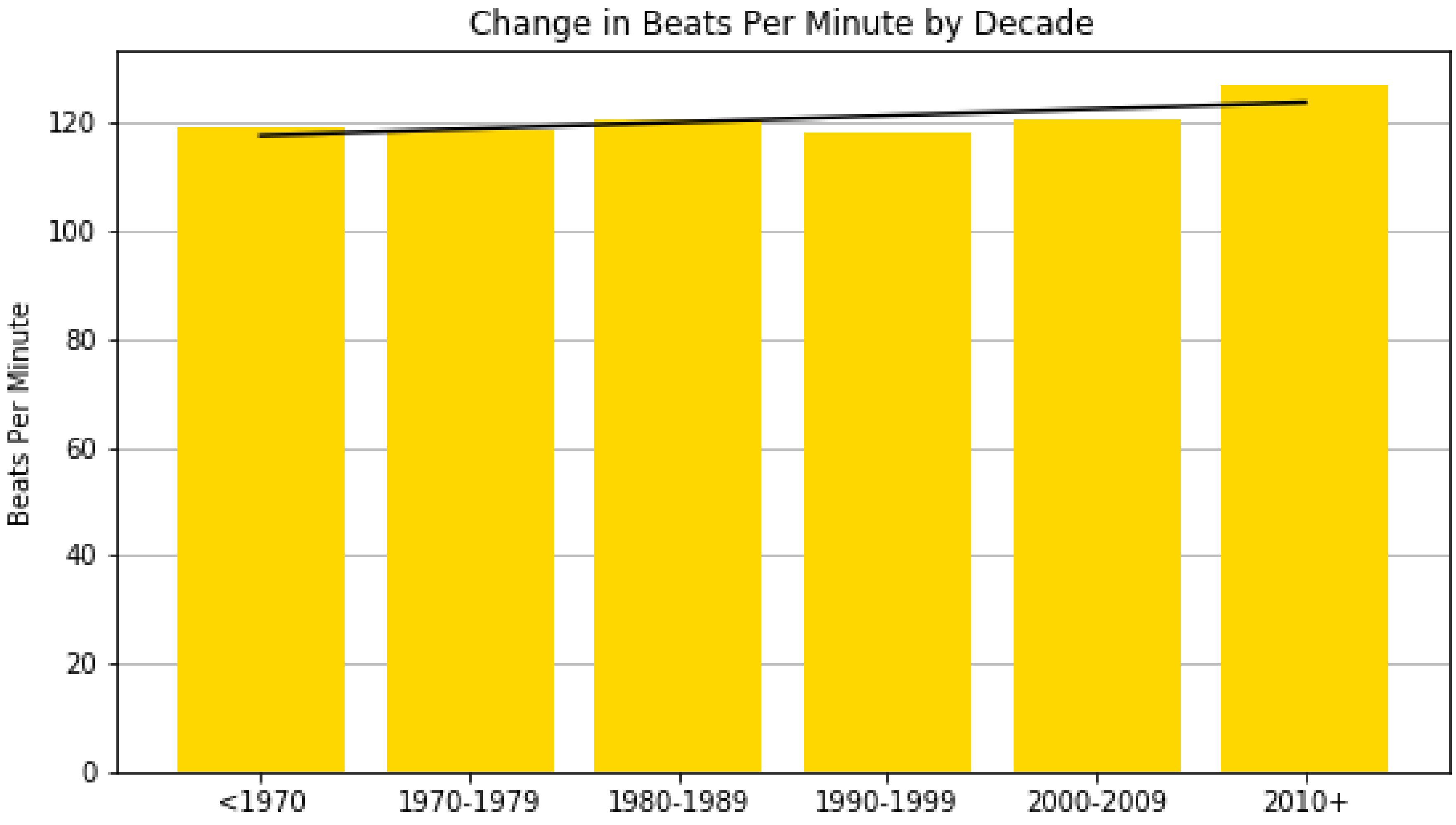
BPM



- Average Beats Per Minute: 120
- Max Beats Per Minute: 208
- Min Beats Per Minute: 52



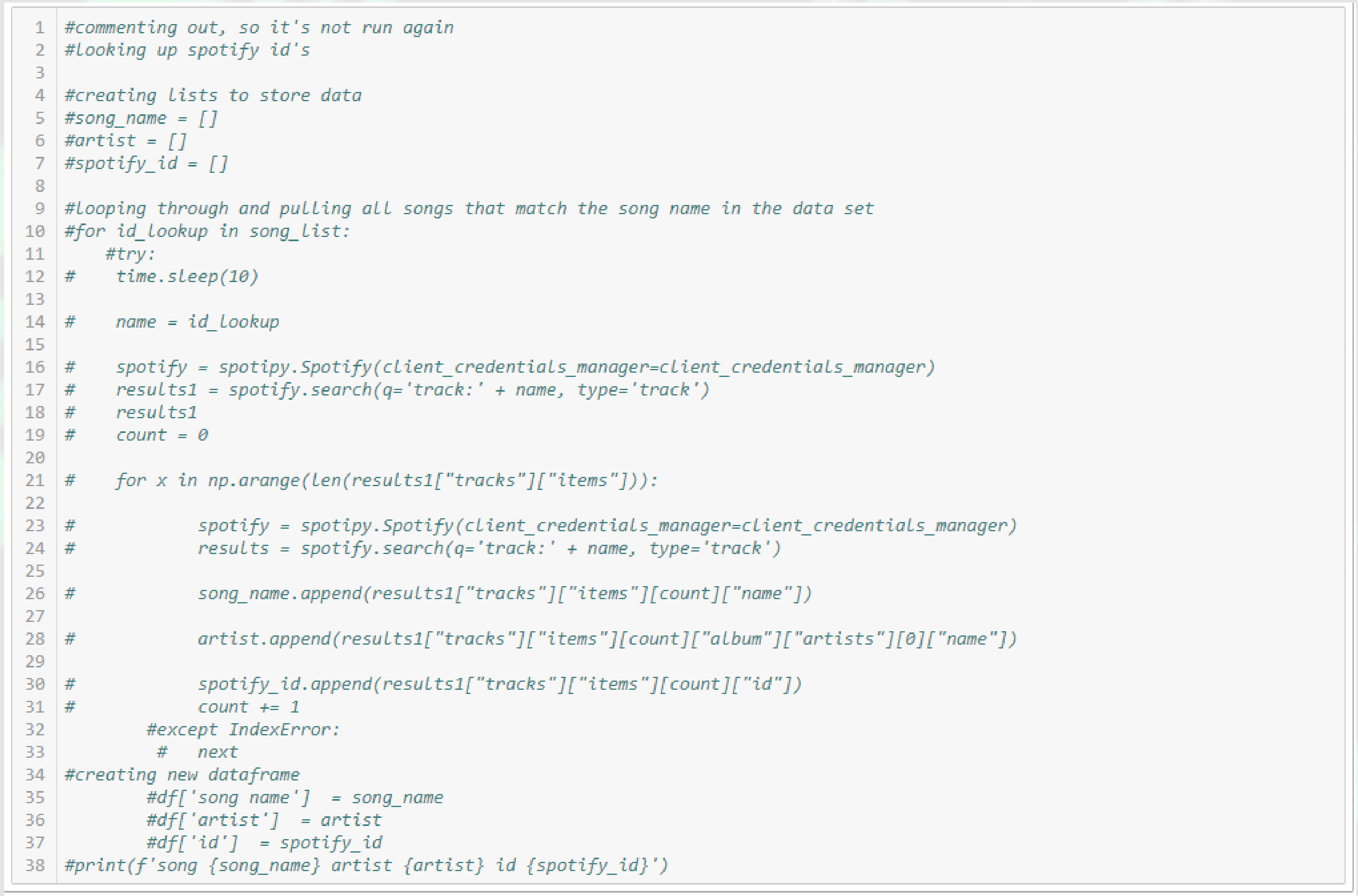
BPM



Decade	Average BPM
<1970	119.042373
1970-1979	118.607375
1980-1989	120.415771
1990-1999	118.209446
2000-2009	120.700219
2010+	126.850000

Spotify

The quest for song ID’s

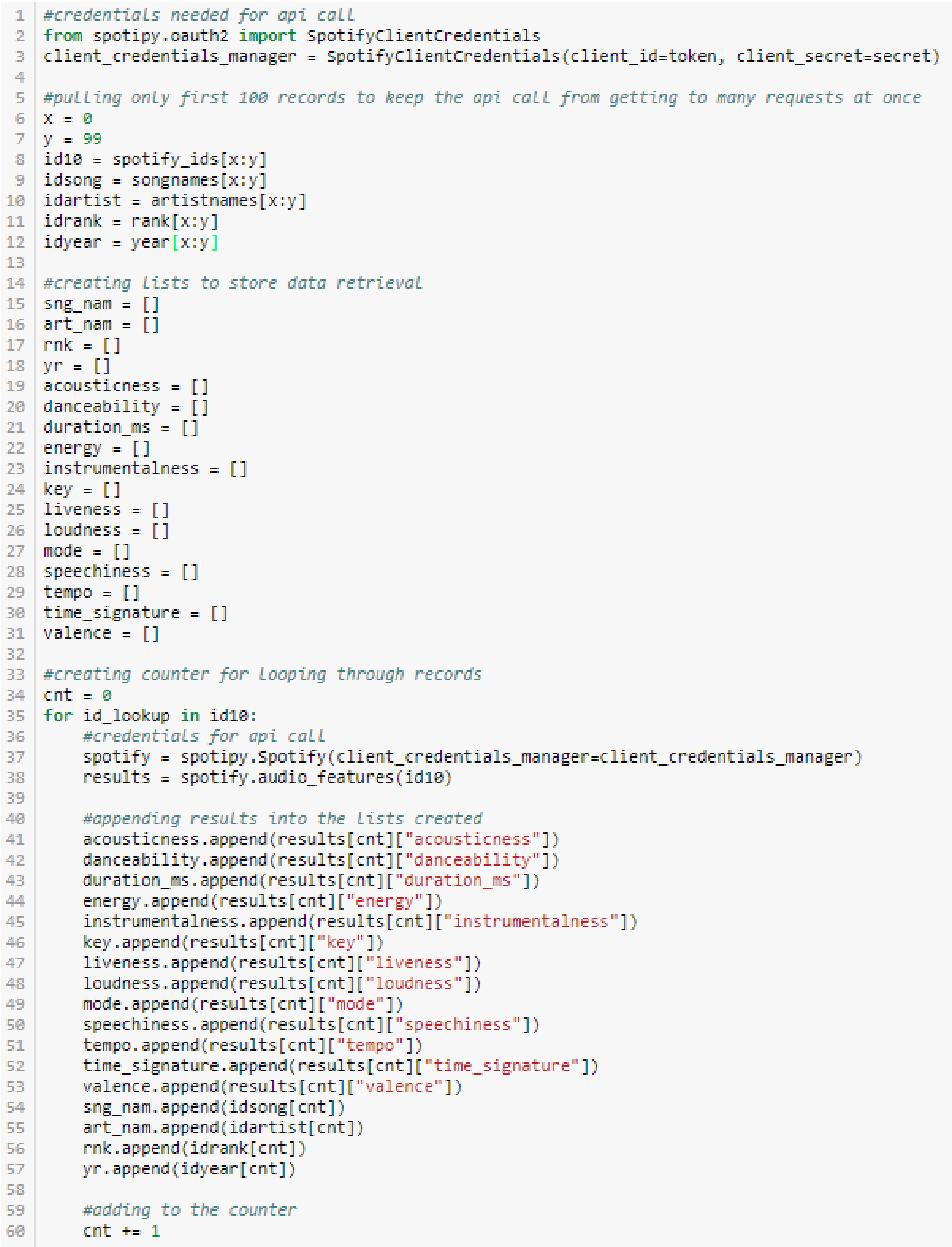


```
#commenting out, so it's not run again
#looking up spotify id's

#creating lists to store data
#song_name = []
#artist = []
#spotify_id = []

#looping through and pulling all songs that match the song name in the data set
#for id_lookup in song_list:
    #try:
#    time.sleep(10)

#    name = id_lookup

#    spotify = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
#    results1 = spotify.search(q='track:' + name, type='track')
#    results1
#    count = 0

#    for x in np.arange(len(results1["tracks"]["items"])):

#            spotify = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
#            results = spotify.search(q='track:' + name, type='track')

#            song_name.append(results1["tracks"]["items"][count]["name"])

#            artist.append(results1["tracks"]["items"][count]["album"]["artists"][0]["name"])

#            spotify_id.append(results1["tracks"]["items"][count]["id"])
#            count += 1
        #except IndexError:
         #   next
#creating new dataframe
        #df['song name']  = song_name
        #df['artist']  = artist
        #df['id']  = spotify_id
#print(f'song {song_name} artist {artist} id {spotify_id}')
```

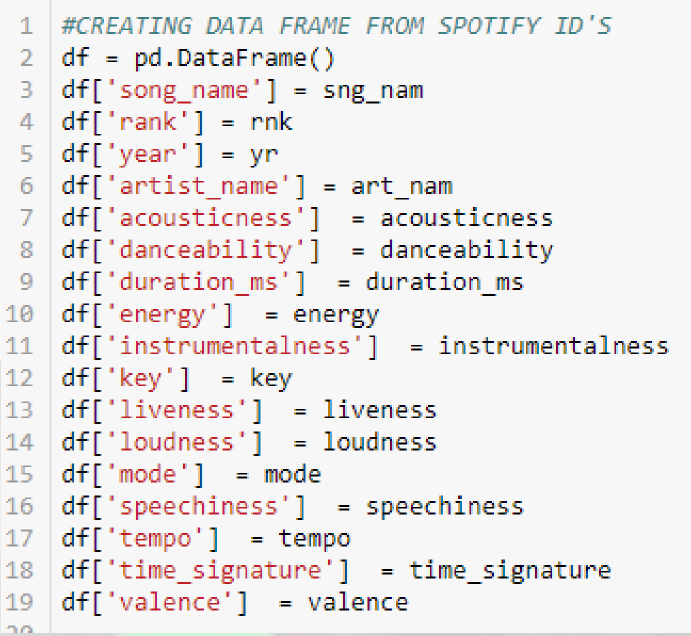
- Was unable to call Spotify to look up only 1 specific ID with Artist and song
- Made API call with the song name and pulled all track ID’s for future lookups
- Took all songs and pulled out only the songs that matched both the song name and artist name from original dataset and appended the Spotify ID

Spotify

Pulling the Data Points

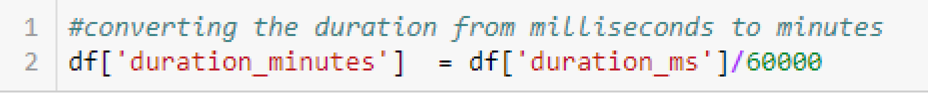


```
#credentials needed for api call
from spotipy.oauth2 import SpotifyClientCredentials
client_credentials_manager = SpotifyClientCredentials(client_id=token, client_secret=secret)

#pulling only first 100 records to keep the api call from getting to many requests at once
x = 0
y = 99
id10 = spotify_ids[x:y]
idsong = songnames[x:y]
idartist = artistnames[x:y]
idrank = rank[x:y]
idyear = year[x:y]

#creating lists to store data retrieval
sng_nam = []
art_nam = []
rnk = []
yr = []
acousticness = []
danceability = []
duration_ms = []
energy = []
instrumentalness = []
key = []
liveness = []
loudness = []
mode = []
speechiness = []
tempo = []
time_signature = []
valence = []

#creating counter for looping through records
cnt = 0
for id_lookup in id10:
    #credentials for api call
    spotify = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
    results = spotify.audio_features(id10)

    #appending results into the lists created
    acousticness.append(results[cnt]["acousticness"])
    danceability.append(results[cnt]["danceability"])
    duration_ms.append(results[cnt]["duration_ms"])
    energy.append(results[cnt]["energy"])
    instrumentalness.append(results[cnt]["instrumentalness"])
    key.append(results[cnt]["key"])
    liveness.append(results[cnt]["liveness"])
    loudness.append(results[cnt]["loudness"])
    mode.append(results[cnt]["mode"])
    speechiness.append(results[cnt]["speechiness"])
    tempo.append(results[cnt]["tempo"])
    time_signature.append(results[cnt]["time_signature"])
    valence.append(results[cnt]["valence"])
    sng_nam.append(idsong[cnt])
    art_nam.append(idartist[cnt])
    rnk.append(idrank[cnt])
    yr.append(idyear[cnt])

    #adding to the counter
    cnt += 1
```



```
#CREATING DATA FRAME FROM SPOTIFY ID'S
df = pd.DataFrame()
df['song_name'] = sng_nam
df['rank'] = rnk
df['year'] = yr
df['artist_name'] = art_nam
df['acousticness']  = acousticness
df['danceability']  = danceability
df['duration_ms']  = duration_ms
df['energy']  = energy
df['instrumentalness']  = instrumentalness
df['key']  = key
df['liveness']  = liveness
df['loudness']  = loudness
df['mode']  = mode
df['speechiness']  = speechiness
df['tempo']  = tempo
df['time_signature']  = time_signature
df['valence']  = valence
```

- Made another Spotify API Call to collect the attributes for each of the songs in the list
- This was ran multiple times to make all of the API calls as only 100 songs could be looked up at once
- Created a new dataframe with the new data
- Ran a conversion on milliseconds to convert the song duration to minutes

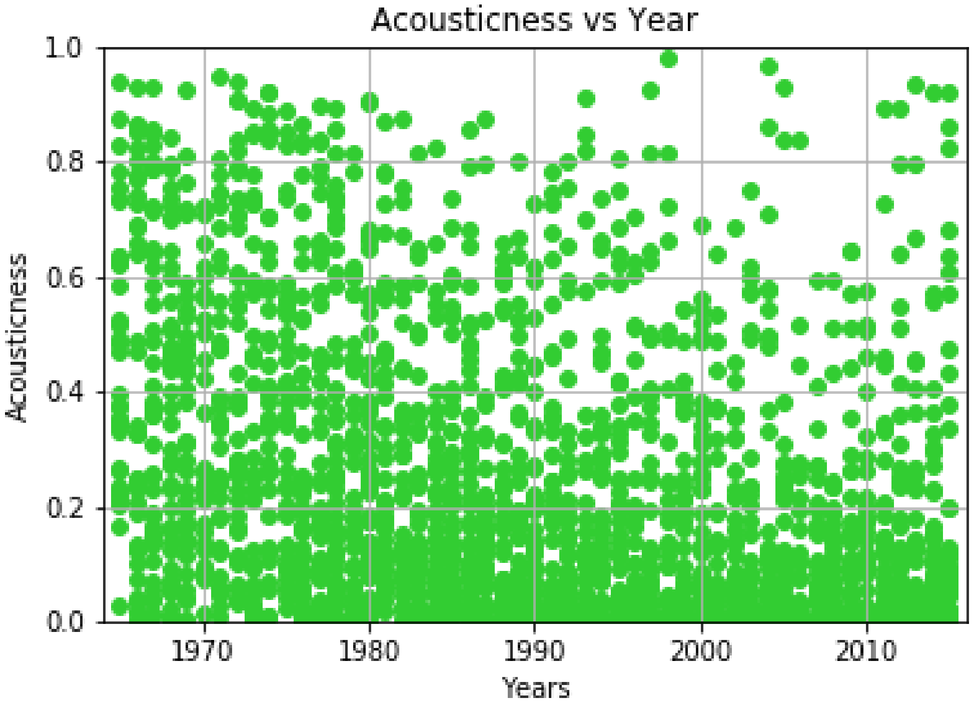


```
#converting the duration from milliseconds to minutes
df['duration_minutes']  = df['duration_ms']/60000
```

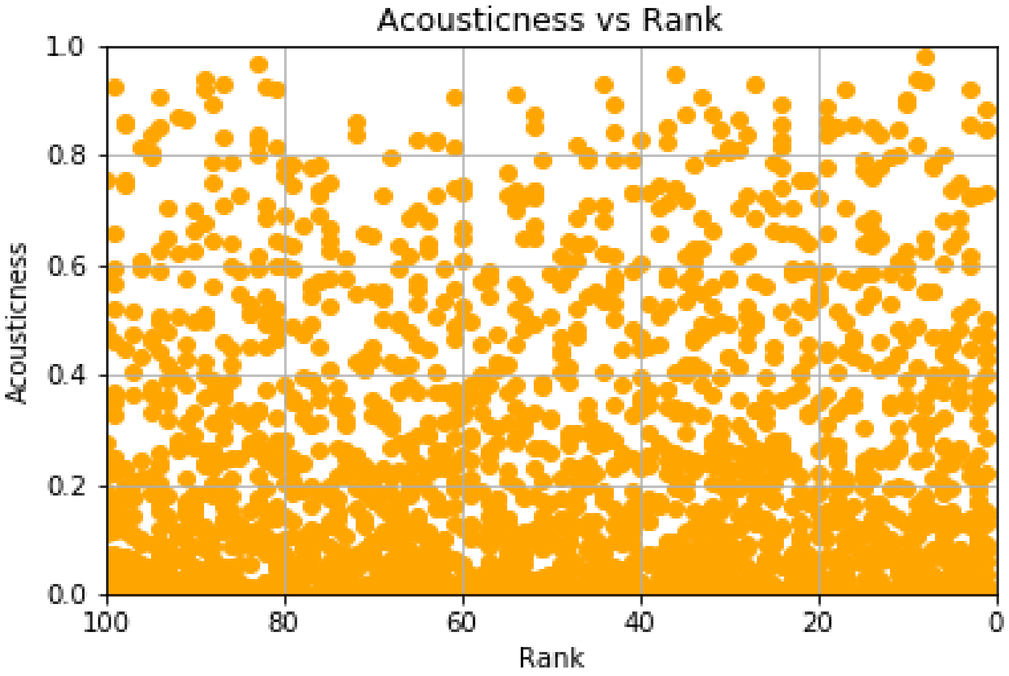
Acousticnes

Definition: A confidence measure from 0.0 to 1.0 of whether the track is acoustic. 1.0 represents high confidence the track is acoustic

Year



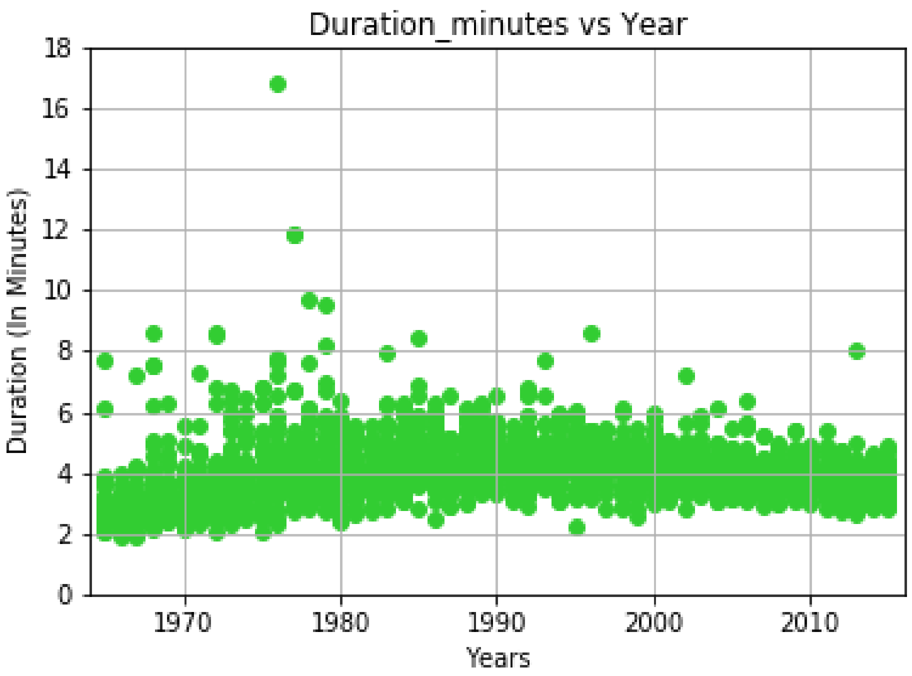
Rank



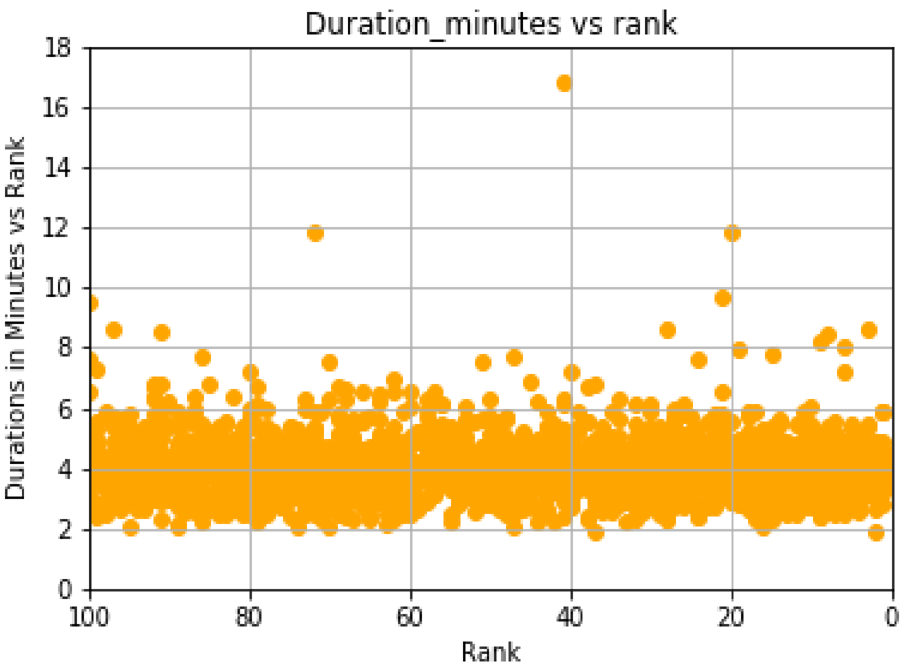
Duration

Definition: Length of Song

Year



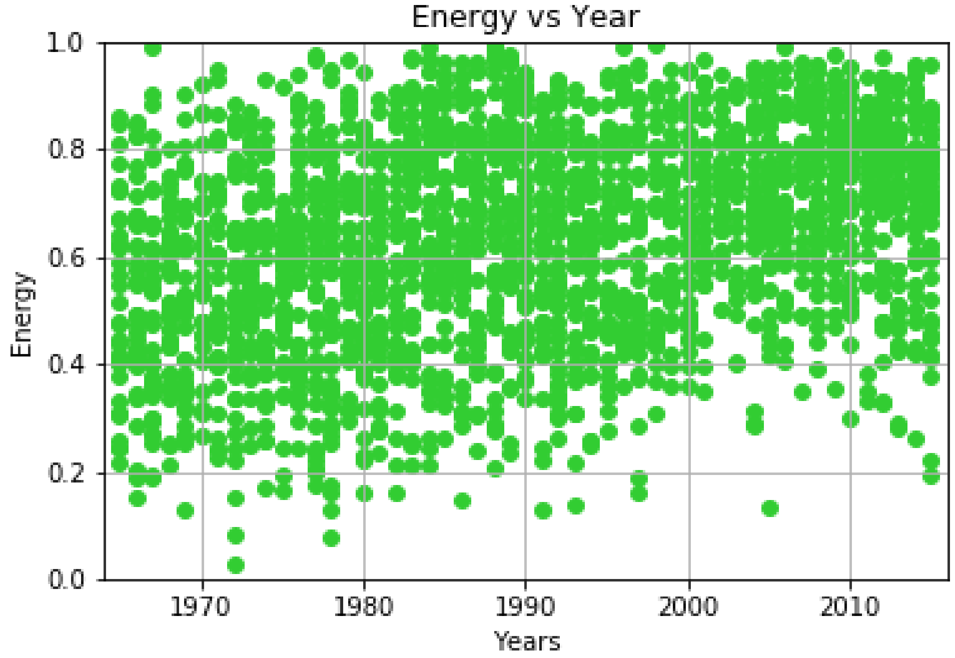
Rank



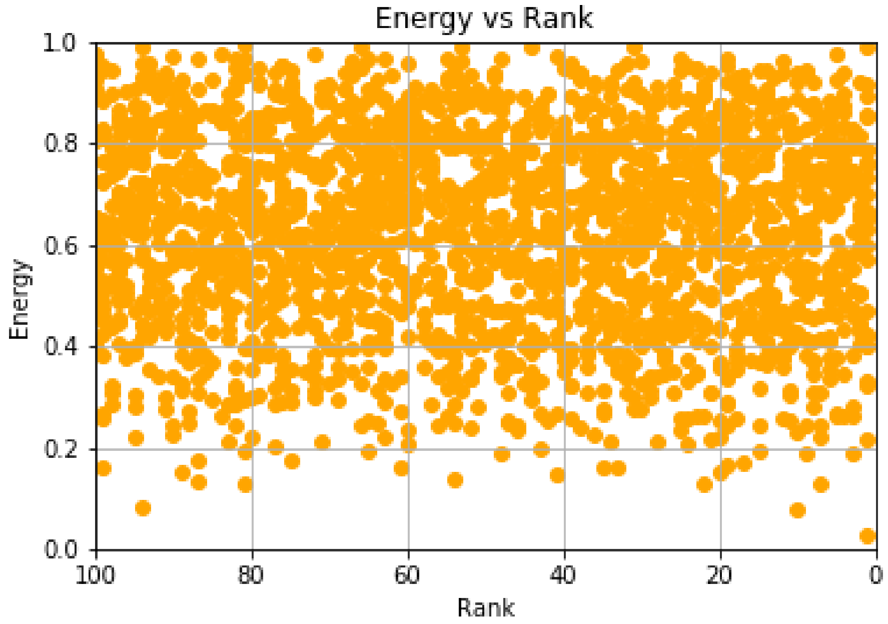
Energy

Definition: Energy is a measure from 0.0 to 1.0 and represents a perceptual measure of intensity and activity. Typically, energetic tracks feel fast, loud, and noisy. For example, death metal has high energy, while a Bach prelude scores low on the scale. Perceptual features contributing to this attribute include dynamic range, perceived loudness, timbre, onset rate, and general entropy

Year



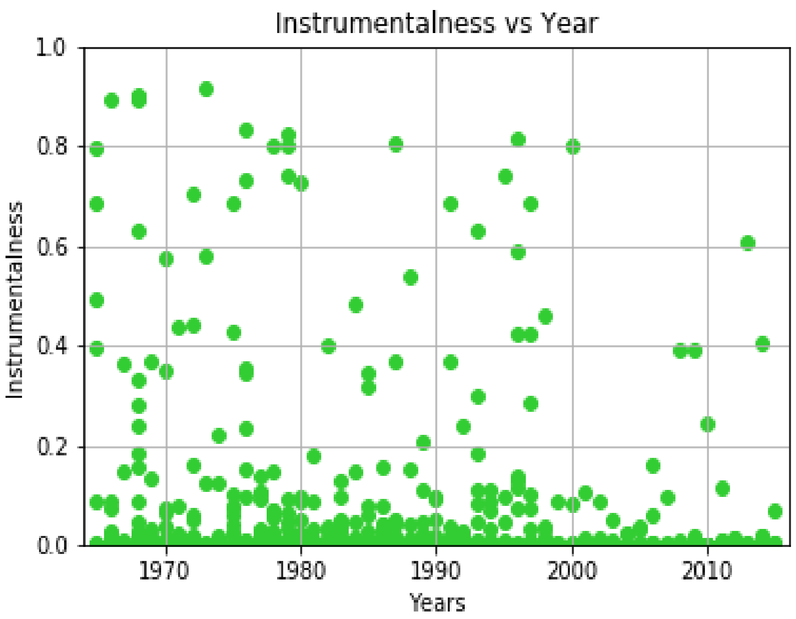
Rank

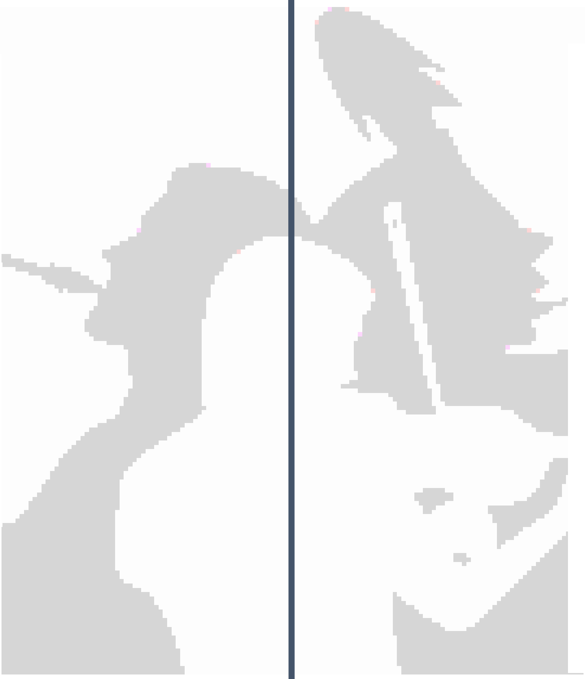


Instrumenta

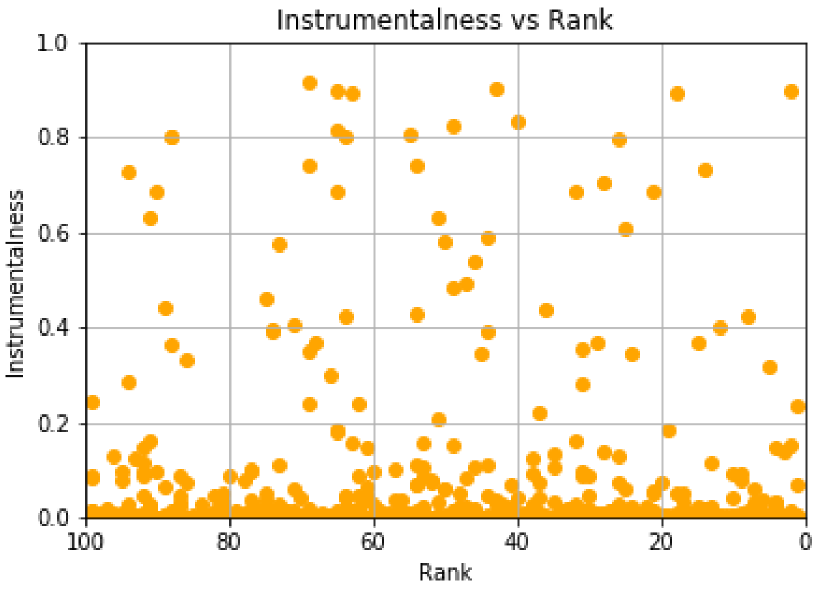
Definition: Predicts whether a track contains no vocals. “Ooh” and “aah” sounds are treated as instrumental in this context. Rap or spoken word tracks are clearly “vocal”. The closer the instrumentalness value is to 1.0, the greater likelihood the track contains no vocal content. Values above 0.5 are intended to represent instrumental tracks, but confidence is higher as the value approaches 1.0.

Year





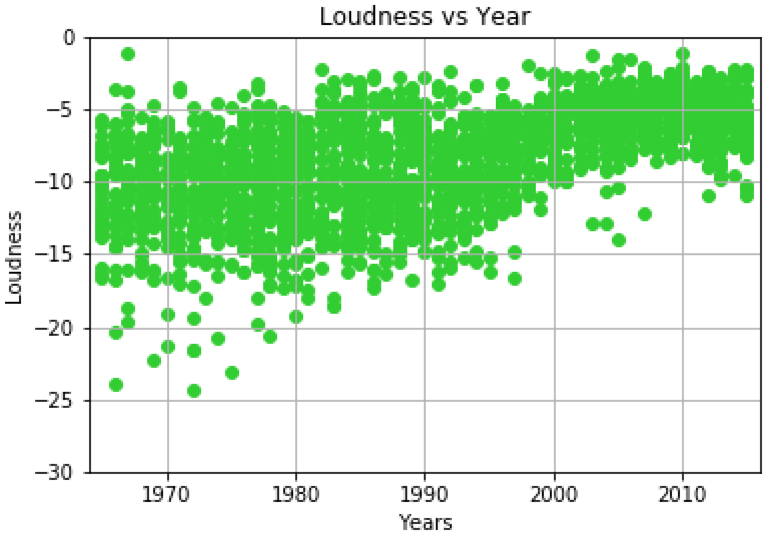
Rank



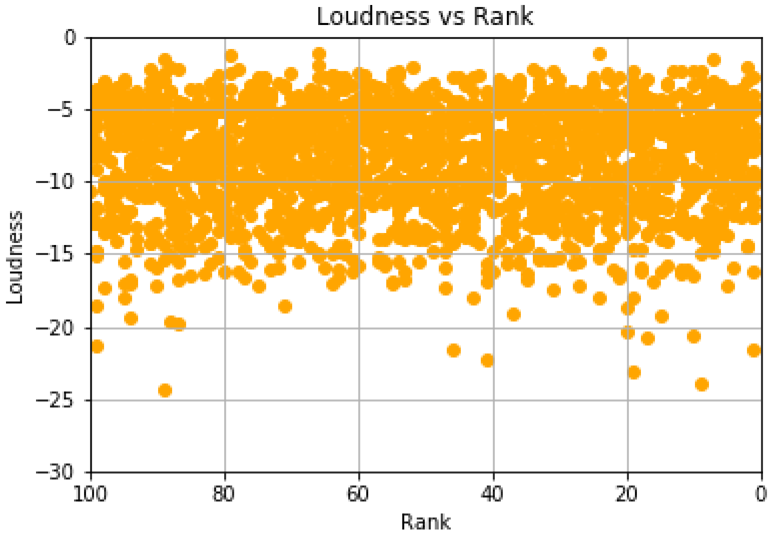
loudness

Definition: The overall loudness of a track in decibels (dB). Loudness values are averaged across the entire track and are useful for comparing relative loudness of tracks. Loudness is the quality of a sound that is the primary psychological correlate of physical strength (amplitude). Values typical range between -60 and 0 db.

Year



Rank



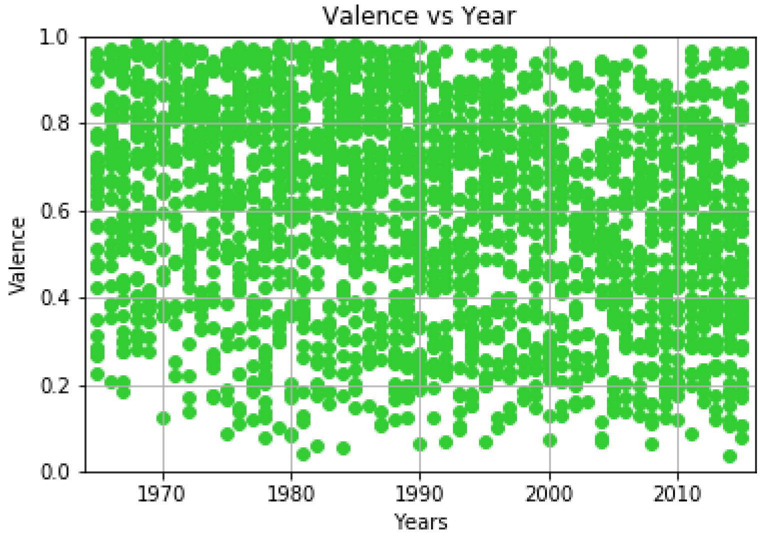
LOUDNESS war

The loudness war (or loudness race) refers to the trend of increasing audio levels in recorded music which many critics believe reduces sound quality and listener enjoyment. Increasing loudness was first reported as early as the 1940s, with respect to mastering practices for 7" singles. https://en.wikipedia.org/wiki/Loudness_war

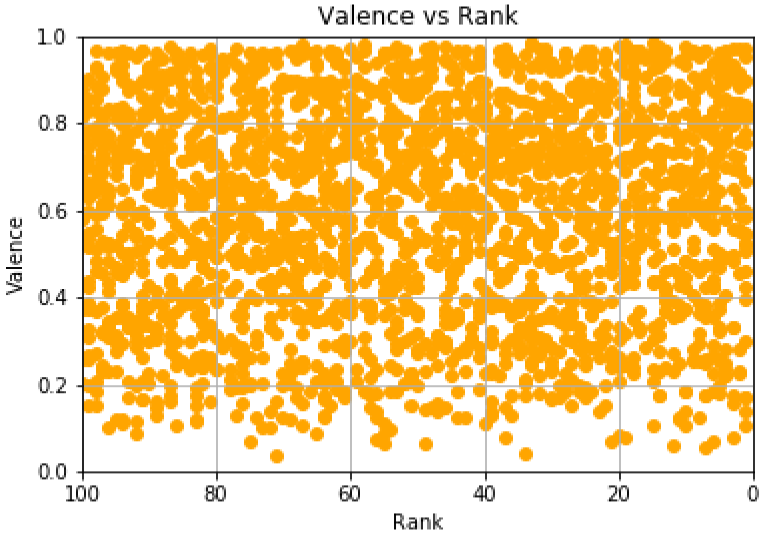
Valence

Definition: A measure from 0.0 to 1.0 describing the musical positiveness conveyed by a track. Tracks with high valence sound more positive (e.g. happy, cheerful, euphoric), while tracks with low valence sound more negative (e.g. sad, depressed, angry).

Year



Rank



Conclusion

- Keep your song length between 2.5 – 5 minutes
- Don’t write instrumentals
- More likely to have a pop hit with positive words in your song
- Keep the beats per minute around 120 bpm

2

- Break out songs based on Genre and review for trends
- Trend what the guidelines would be for the year 2050
- Complete regression analysis for the Spotify data
- Add in a review of Time Signatures and Key Signatures
- Analyze the song structures
- Address the record label, producer, artist

Questions?