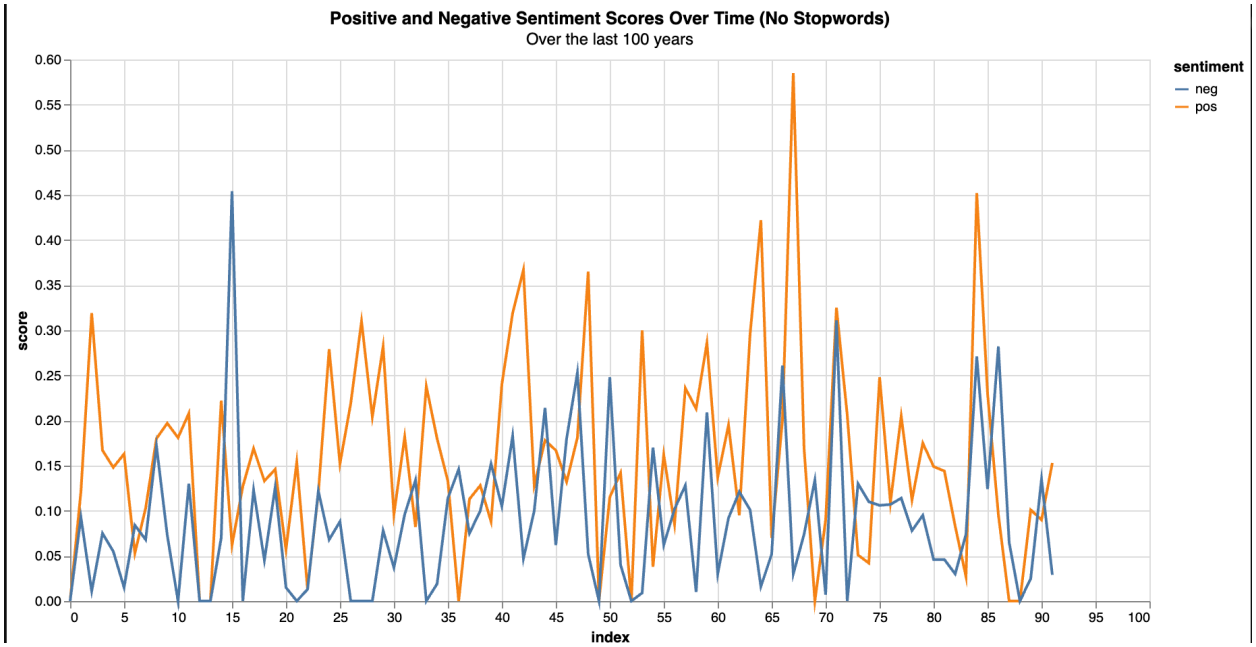
Lyrical Analysis through the Ages

INFO 3510—Music as Information—Most recognizable songs over the past 100 years.—A journey of the data life cycle.



This is the story of the intimate interaction between me and the data.

When brainstorming possible data-centered projects that I could attempt - that have to do with music, this YouTube video surfaced to my mind early on. On Spotify, I found an album of these top songs. I wanted to generalize my findings to apply them to human culture over the years.

Skills for this project:

- Data Cleaning
- Data Collection
- Natural Language Processing (NLP) | VADER
- Data Visualization
- Machine Learning (TF-IDF / SVM*)
- and more!

Inspiration Video:

To kick off the data-gathering process, I leveraged the Spotify API, targeting an album of top hits. The same album from the inspiration video. I retrieved song titles and artist names for further lyric scraping, which I achieved through the Lyrics Genius API, a handy alternative to traditional scraping tools like BeautifulSoup or Selenium.

Data source: Spotify API, getting the list of all of our songs!

```
playlistSearch = requests.get(rootURL +'playlists/59VV1FP9jNO8TwbHCkna5t',headers=headers).json()
playlistSearch.keys()
```

```
dict_keys(['collaborative', 'description', 'external_urls', 'followers', 'href', 'id', 'images', 'name',
'snapshot_id', 'tracks', 'type', 'uri'])
```

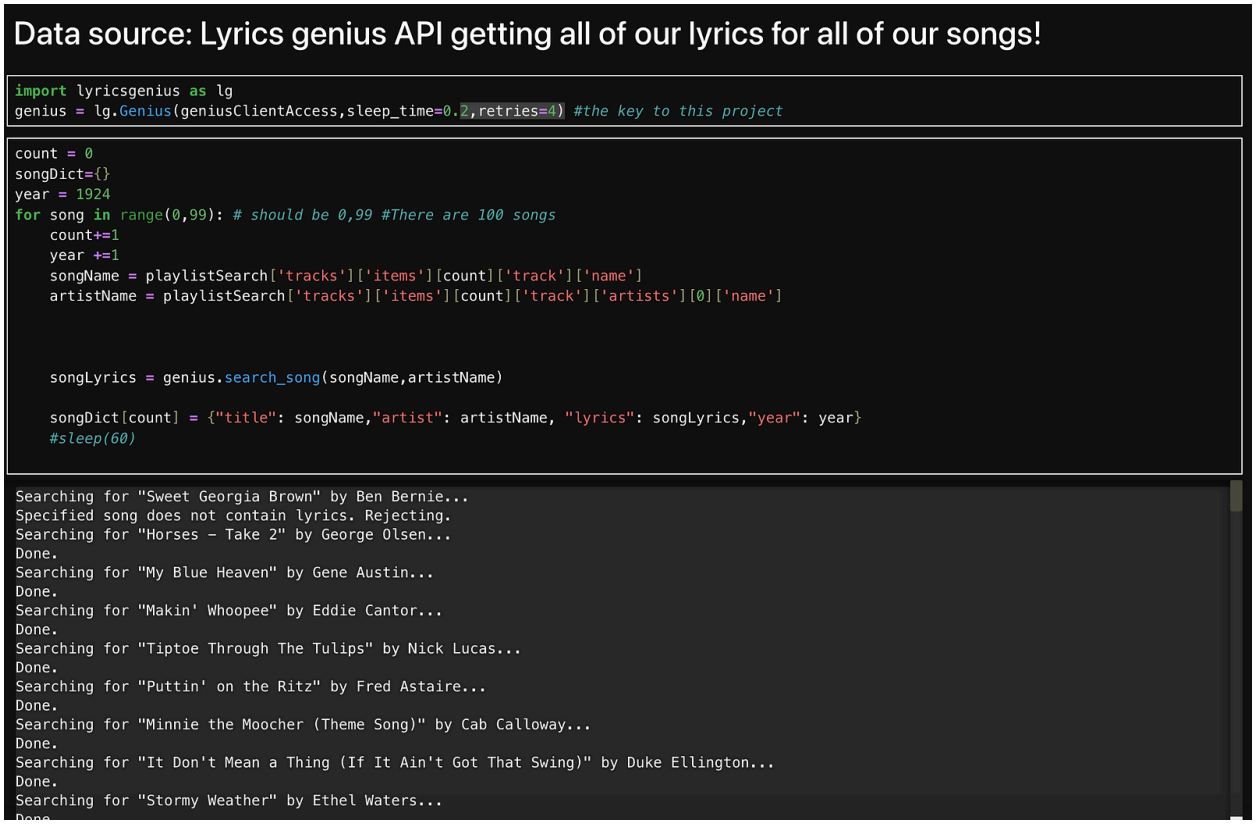
```
count = 0
for song in range(0,99): #There are 100 songs
    count+=1
    songName = playlistSearch['tracks']['items'][count]['track']['name']
    artistName = playlistSearch['tracks']['items'][count]['track']['artists'][0]['name']
    print(songName,", ",artistName)
```

```
Sweet Georgia Brown ,  Ben Bernie
Horses - Take 2 ,  George Olsen
My Blue Heaven ,  Gene Austin
Makin' Whoopee ,  Eddie Cantor
Tiptoe Through The Tulips ,  Nick Lucas
Puttin' on the Ritz ,  Fred Astaire
Minnie the Moocher (Theme Song) ,  Cab Calloway
It Don't Mean a Thing (If It Ain't Got That Swing) ,  Duke Ellington
Stormy Weather ,  Ethel Waters
The Very Thought of You ,  Ray Noble
Top Hat, White Tie And Tails ,  Fred Astaire
Pennies From Heaven - Single Version ,  Bing Crosby
Sing, Sing, Sing ,  Benny Goodman
Jeepers Creepers ,  Paul Whiteman & His Orchestra
Over The Rainbow ,  Judy Garland
I'll Never Smile Again ,  The Ink Spots
Boogie Woogie Bugle Boy ,  The Andrews Sisters
Paper Doll ,  The Mills Brothers
```

Code to get song name and artist name for each song using SpotifyAPI/requests module.

This portion of the project went by really fast, when I was printing out the song title and artist name I was prepping the output so that when I was ready to use lyricsgenius, it would programmatically search for all my songs. A great idea that could not possibly generate any problems throughout the process– I know I thought that too!

Using lyricsgenius is better than using a manual and general web scraper, like BeautifulSoup and Selenium.



SpotifyAPI and lyrics genius being used in conjunction to programmatically get all lyrical data.

The code above shows me importing the lyrics genius module. The *key* to this project was to set “retries to four”. For the longest time, my lyric genius loop would stop working around the 50th song. Although it felt rather random, sometimes it would crap out around seven, sometimes we’d get to the 60’s. I tried splitting up the loop so half would go in one cell and the other requests would be made in another cell. This wasn’t working because my code continued to time out, this enigma seemed to be happening at random times. The key was reading the lyrics' genius documentation and a tiny bit of stack-overflow to find-out that:

- I wasn’t the only one.
- Set retries to be equal to 4 in the lyricsgenius object

viewed 145k times, not so niche i guess.

retries = 2 didn’t work, retries = 4 always worked. I could even help other people with this ‘discovery’

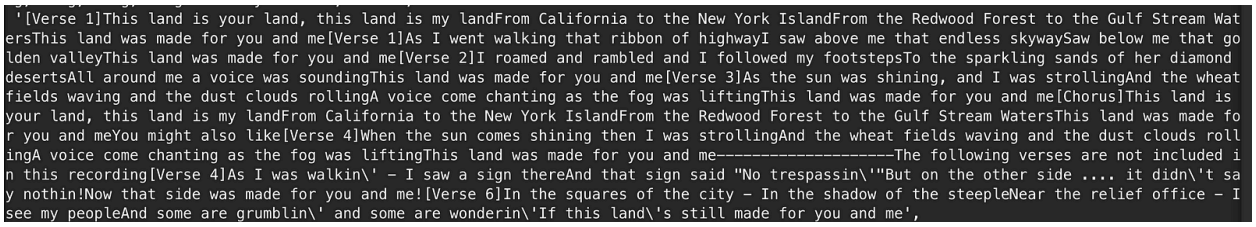
There were actually a few problems when I was working with lyricsgenius, it actually wasn't my first choice either. Originally, I wanted to work with AZLyrics for getting all of my lyrics. The problems with AZlyrics were ambiguous and the documentation seemed less so I ended up pivoting to using Lyrics genius completely. Later on during my process I wanted to use AZlyrics for a second time so that I could do a sort of 'data validation' because my trust in lyricsgenius started to drop.

In general I think lyricsgenius is good for getting a song or two, but my problem is I was asking for songs from 100 years ago and 100 of them. The search engine probably needs an overhaul.

Data Cleaning:

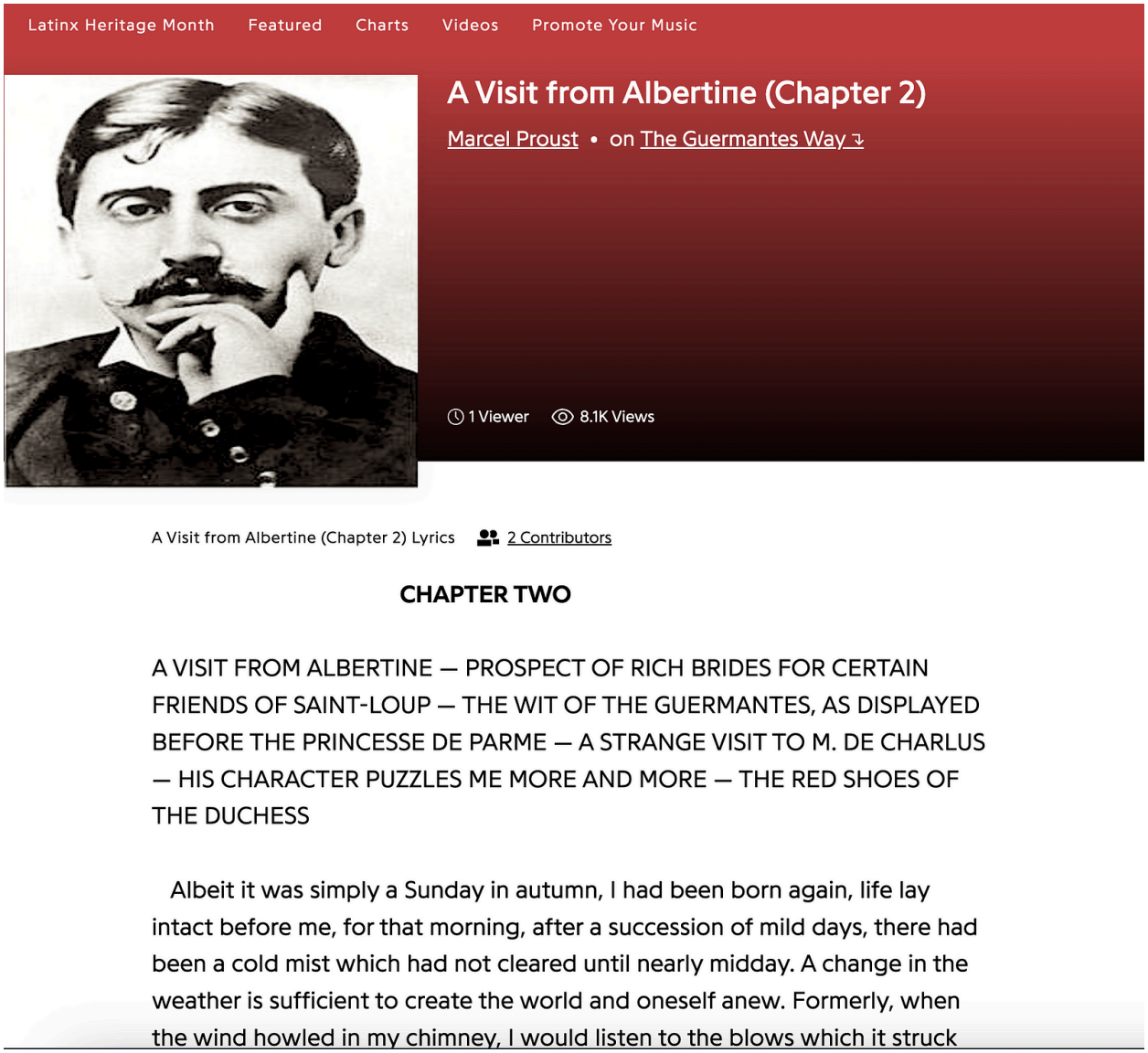
90% of a Data Scientist's time is spent cleaning data, data wrangling and getting data ready for analysis. This trend stands true in my experience working here as well. It is easy to blame all of the data problems I have on the greatest source, lyricsgenius. I tried using regex to do some cleaning and standardization of lyrical entries made by lyrics genius. Occasionally lyrics genius would consistently return bad lyrics and in those cases I ended up just dropping them from the analysis. (.pop(n))

For example:



an example of lyrics that are cleaned the way I want them to be.

This is what I wanted for all 93 songs that had lyrics of my top 100 songs.



An example of probably the worst entry in the entire project.

For my 34th song, an entire book chapter came out! Silly Lyrics genius- thats not what I asked for!

{'title': 'The Sound of Silence - Electric Version', 'artist': 'Simon & Garfunkel', 'lyrics': Song(id, artist, ...), 'year': 1964}

That isn't what I asked for!

```
df_cleaned_lyric_list.drop(35)
```

	0
0	illyrical
1	[Intro]Shades night creepingWillow trees weepi...
2	seen well-to-do, Park AvenueOn famous thorough...
3	[Verse 1]Folks, here's story 'bout Minnie Mooc...
4	[Trumpet solo: Louis Armstrong][Verse]What goo...
...	...
85	[Intro]Yeah[Verse 1]I've tryna callI've long e...
86	[Intro: Jung Kook]'Cause I, I, I'm stars tonig...
87	BTS - Permission DancePost Malone - Motley Cre...
88	[Intro]Come on, Harry, wanna say goodnight you...
89	[Verse 1]We good, goldKinda dream can't soldWe...

89 rows × 1 columns

```
# clean out bad data::
cleanedLyricList.pop(1)
cleanedLyricList.pop(38)
cleanedLyricList.pop(10)
print("Cleaned out bad songs, with bad data!")
```

```
Cleaned out bad songs, with bad data!
```

```
cleanedLyricList[10]
```

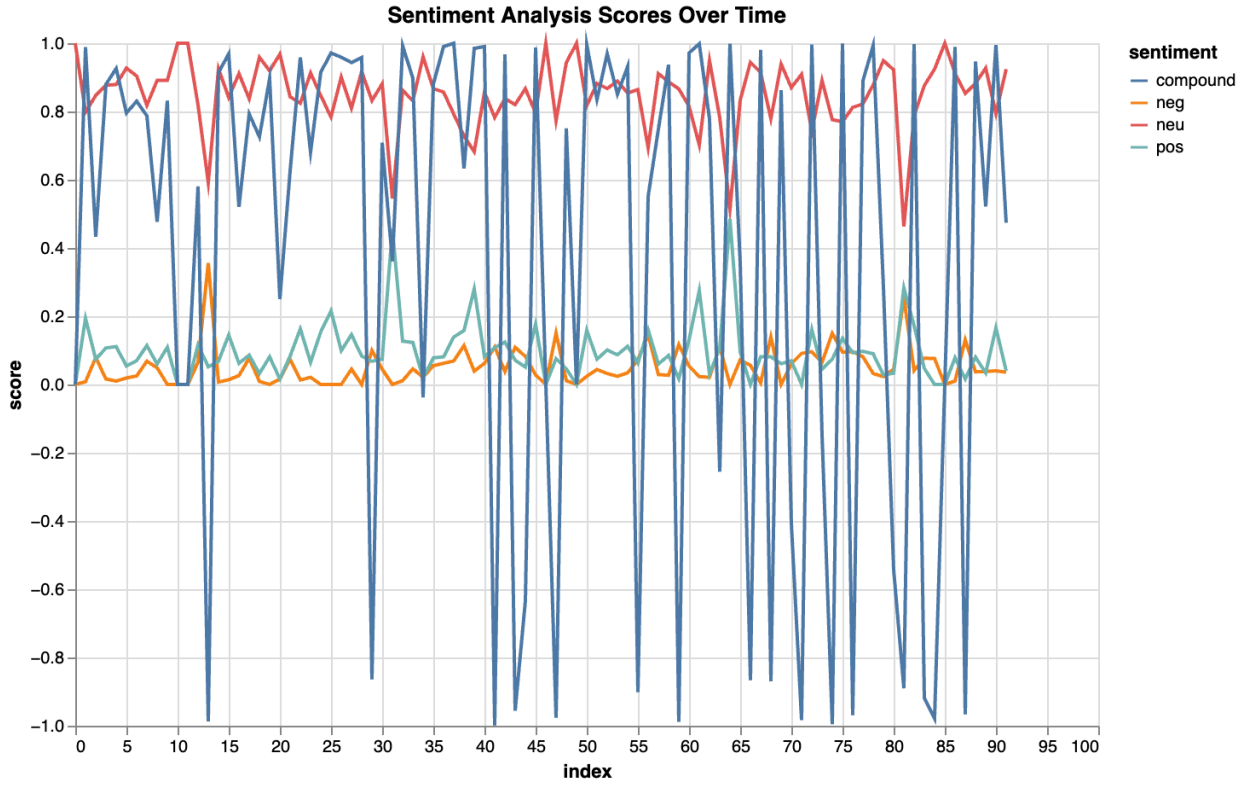
```
"Baby, baby, matter you?Baby, baby, matter you?You got world jug got nothin' doYou know always told ya, you'll death meAnd I'm always get th
ird degreeThat ain't right, ain't right allAnd takin' money goin' havin' ballI took night club bought big champagneYou rolled home taxi caug
ht subway trainThat ain't right, ain't right allAnd takin' money goin' havin' ballI went fortune teller fortune toldShe said, didn't love m
e, wanted goldThat ain't right, ain't right allAnd takin' money goin' havin' ballYou might also like"
```

Structuring everything better would have allowed my data cleansing journey to be done so much better!

VADER Sentiment Analysis:

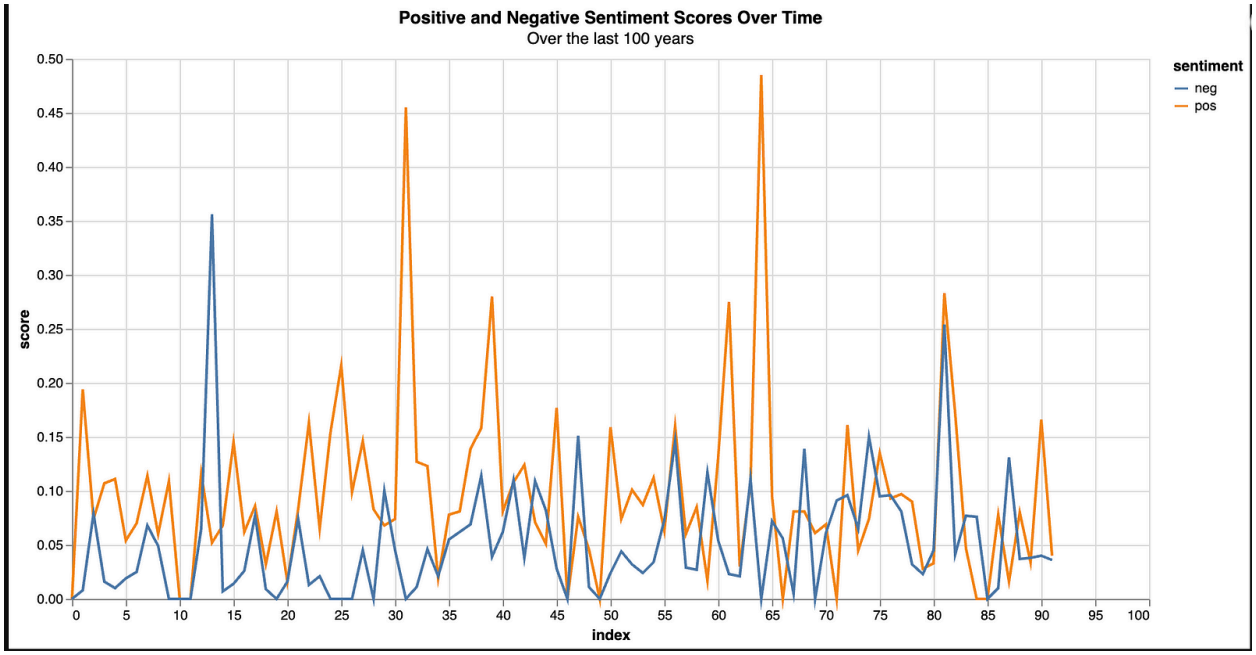
Valence Aware Dictionary for sEntiment Reasoning or (VADER), some say that VADER is better at reading sentiment in text better than humans.

Compound Scores with VADER



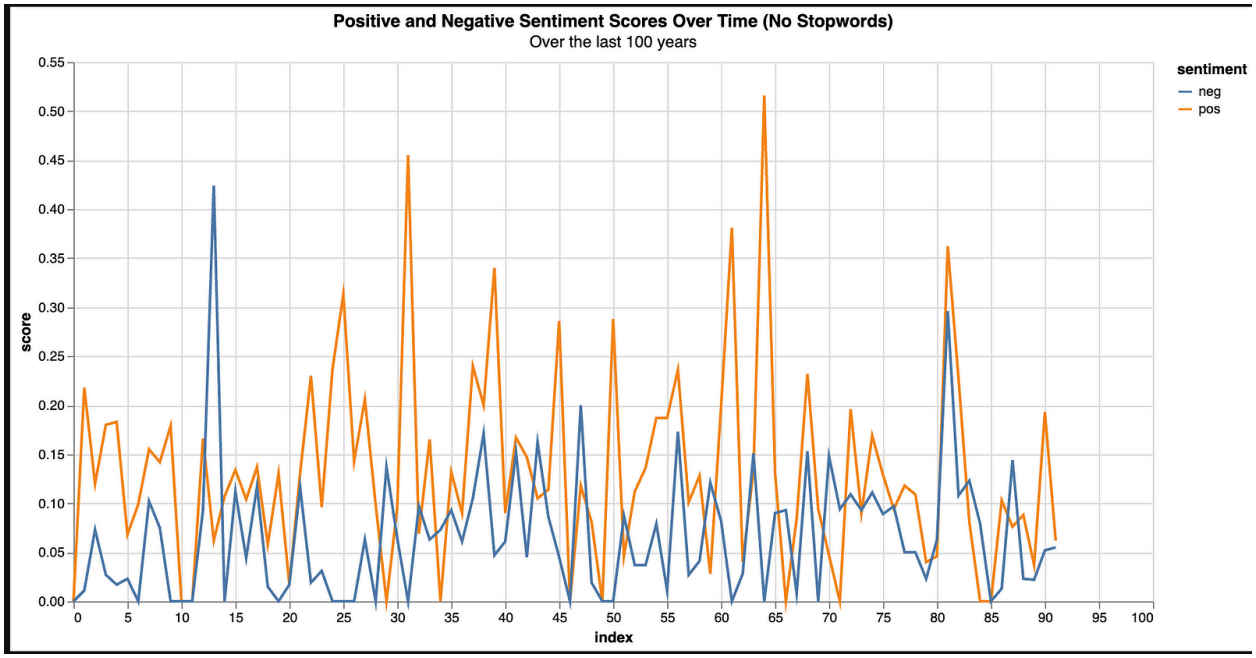
This can be overwhelming, so let's break it up and throw stopwords into the mix~!

Positive & Negative Scores:



with stopwords included

Positive & Negative Scores (without stopwords):



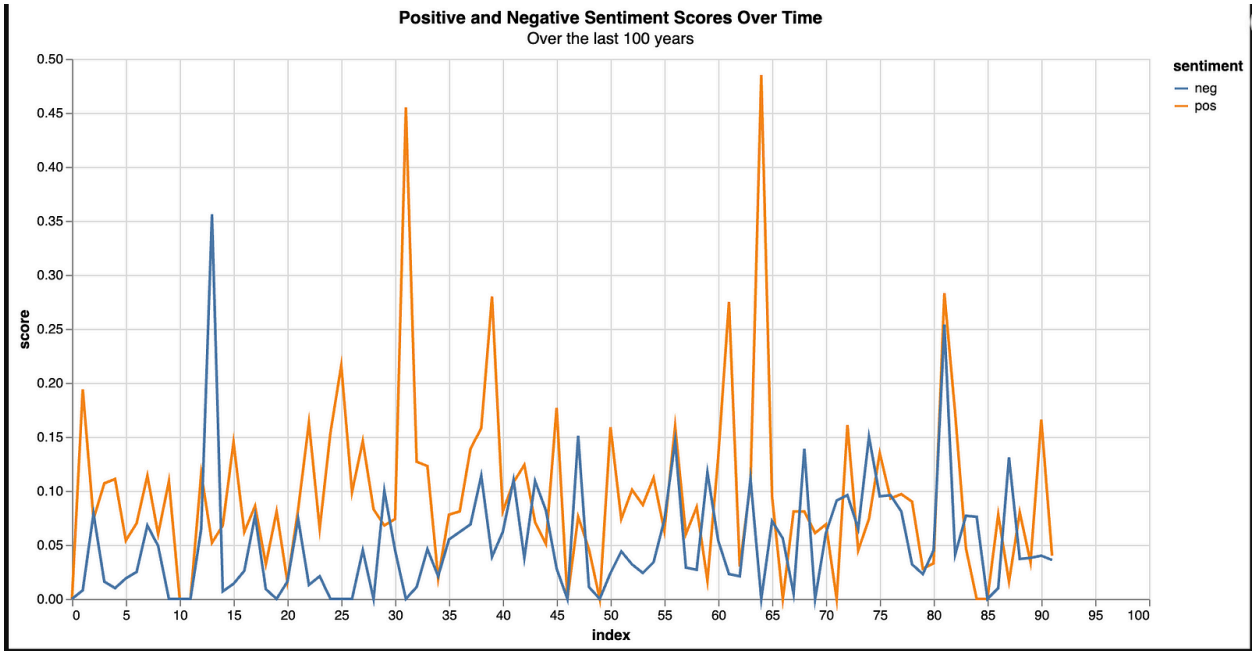
There is an argument for and against having stopwords in NLP and sentiment analysis. I think that stop words can skew data from the original goal, I also think that the meaning, inherent and to give sentences direction, are important with stopwords. For this analysis I wanted to compare with stop words and without that way I could really see the difference in a large corpus. When the total amount of text data is so big there is a not a super noticeable difference in the charts weather or not stopwords are included.

```
df_vader_scores
```

	neg	neu	pos	compound
0	0.000	1.000	0.000	0.0000
1	0.027	0.793	0.180	0.8796
2	0.017	0.800	0.183	0.9260
3	0.023	0.908	0.068	0.7955
4	0.000	0.901	0.099	0.9034
5	0.102	0.744	0.155	0.7231
6	0.075	0.783	0.142	0.7579
7	0.000	0.821	0.179	0.8316
8	0.000	1.000	0.000	0.0000
9	0.093	0.741	0.166	0.5806

This was the data frame I made to store all the vader sentiment scores. Each row is a song. I found visualizing negative and positive columns were the most useful to look at.

To create my visualizations with Altair (I liked its look and felt like mixing it up compared to the usual Seaborn..) I used the .melt function to turn the vader_df to long form, basically unpivoting it to get it ready for Altair.



The index axis is just each song, to reference, the highest point for positive, is whitney houston's I will always love you. Vader just counts 'love' and this positive score shoots up. These songs with a really positive or negative word count become the outliers, especially when they are repeated to make really positive or negative sentences.

This idea of counting the words shooting up the VADER inspired me to learn a new skill, to do TF-IDF or term frequency * inverse document frequency. I had previous experience with scikit learn in my Applied Machine Learning class so I thought how hard could it be to get it to work.

```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer(
    max_df=0.95,
    min_df=2,
    stop_words='english',
    ngram_range=(1, 2)
)
tfidf_matrix = tfidf.fit_transform(cleanedLyricList)

tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf.get_feature_names_out())

tfidf_df.head()

# save each row in the df and print the lyrics
```

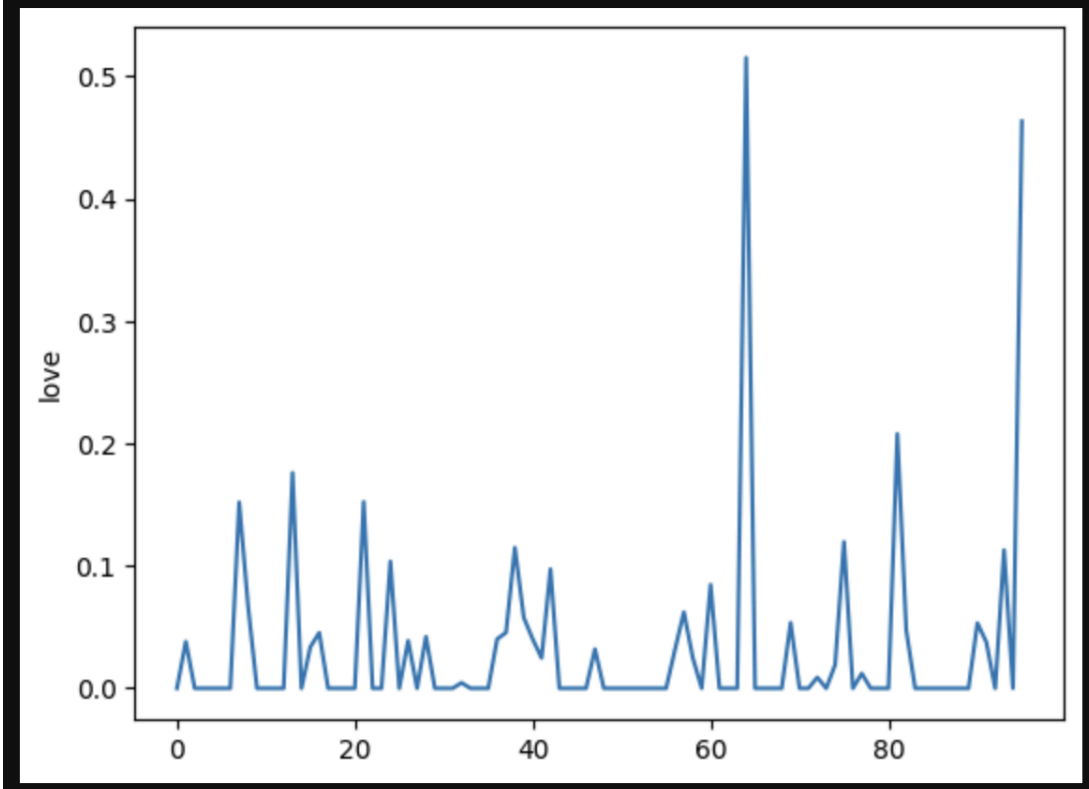
	000	10	10 mravinsky	10 story	11	12	12 single	13	17	1800	...	youyou	ysnirvana	ysnirvana nevermindin	zeit	zeppelin	zeppelin ivleftfield	zeppelin led	zombies	zoo	zur
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 4717 columns

It creates an extremely wide dataframe, each column is a word that is being counted across the 'document' its found in and then again in the entire corpus. The number that is produced is just a percentage of the 'importance' in the corpus. Some interesting words were as follows.

```
import seaborn as sns
sns.lineplot(tfidf_df['love'])
```

```
<Axes: ylabel='love'>
```

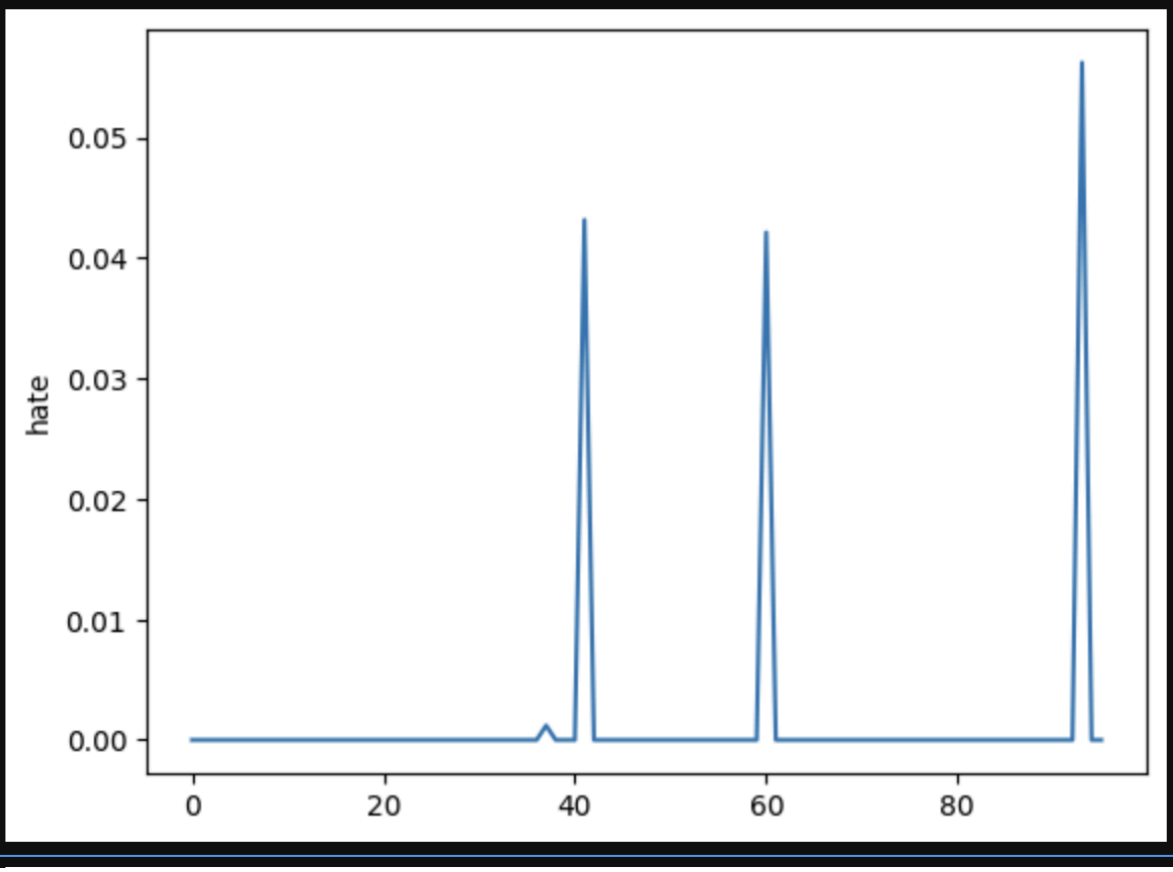


(seaborn cause its easier)

This shows the word love over all the songs from the 1924 (0) to 2024(93) Love is strongest in “I will always love you”, which by the way, I performed in the 8th grade talent show. But overall it is safe to generalize that the most popular songs across the human experience are about love. The final song, Flowers by Miley Cyrus was the final song and ‘love’ seems to be pretty ubiquitous throughout that song as well. People loved to sing about love in the last century. Love is the one thing we're capable of perceiving that transcends dimensions of time and space.” — Dr. Brand, Interstellar.

.

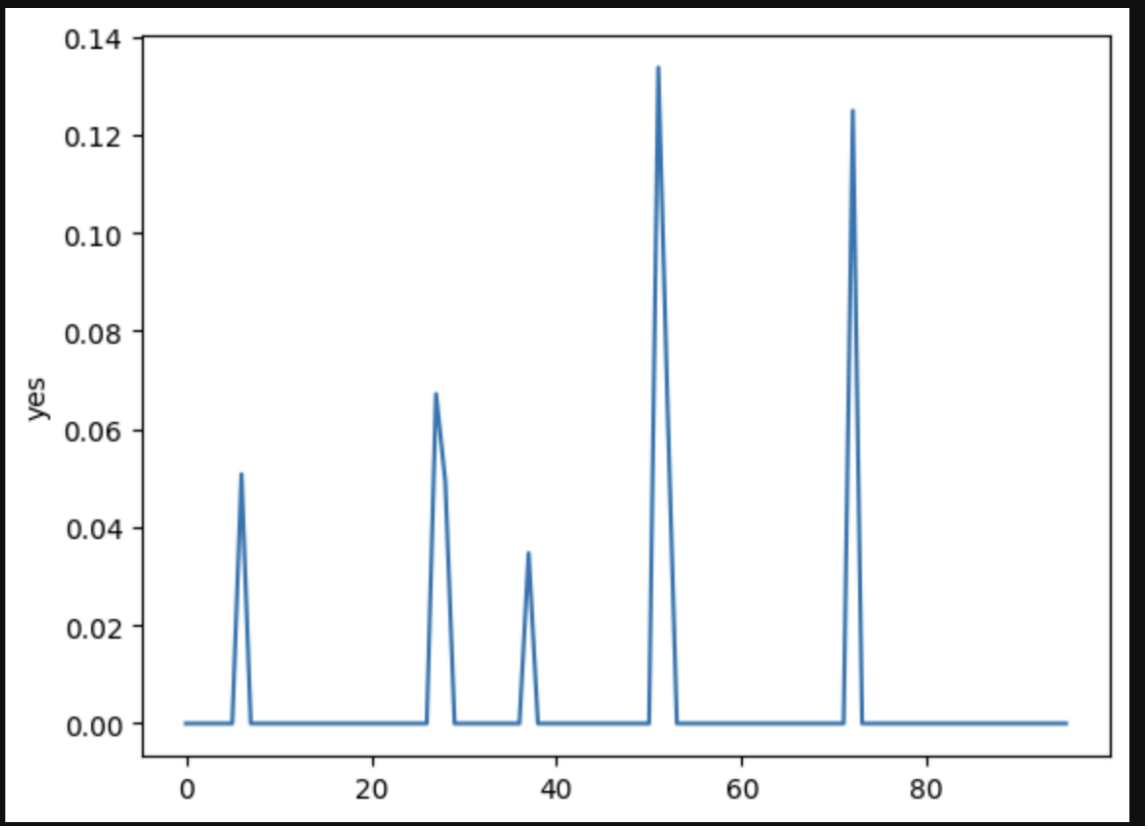
```
]: sns.lineplot(tfidf_df['hate'])

]: <Axes: ylabel='hate'>
```



This one was hate, I think it is safe to say that people more often or not like to be positive. The song that comes out on top, typically is more positive and uses more positive words.

```
[47]: sns.lineplot(tfidf_df['yes'])
```

```
[47]: <Axes: ylabel='yes'>
```



I thought this one was interesting as well – I wanted to share!

```
Document 1:
  heaven: 0.37992067619228
  blue: 0.3762106442494116
  nest: 0.34309111379940116
  little: 0.29651647706808715
  makes: 0.2679109571091535
  molly: 0.2573183353495509
  smiling: 0.2573183353495509
  roses: 0.1881053221247058
  face: 0.1826262077198315
  birds: 0.17154555689970058
```

This shows the top ten most important words in the first song! ("Horses - Take 2" by George Olsen) I don't know the song personally but I thought this was pretty cool to see~!

```
from sklearn.svm import SVC
```

I wanted to take it a step further and use a Support Vector Machine learning algorithm to do some basic classification, although I quickly realized that I had to turn my compound vader scores to black and white classes to actually train the model, this defeated the purpose as all the songs ended up being classified as positive songs.

```
X_train, X_test, y_train, y_test = train_test_split(X, sentiment, test_size=0.2, random_state=42)
```

(standard data splitting up)

```
import numpy as np
y_train = np.array(y_train).astype(int)
y_test = np.array(y_test).astype(int) #
```

Basically turning the y_train into an integer.

```
Confusion Matrix:
 [[19]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19

    accuracy                           1.00        19
   macro avg       1.00      1.00      1.00        19
weighted avg       1.00      1.00      1.00        19


Accuracy Score: 1.0
```

This was my confusion matrix which as you can see only has one element. The root problem was an unbalanced dataset which I decided to pin for later research. There were 19 songs that made it into test, which makes sense since 20% of the data was for testing purposes and 80% was used for training purposes. Which means around 79 songs were used for the X and y training dataset.

Things I would have differently:

- I would have organized my lyrics as a data frame or a csv instead of using a list. I didn't think of this till late in the project, and changing the entire structure would have been a lot of work.
    - This also would have allowed me to include a date for my visualizations making them more effective overall.
- I wanted to take better time to classify songs positive or negative in preparation for machine learning. (the problem was songs aren't just positive or negative based on their scores)

Areas for more future research:

- Using the spotify API and measuring the change in track features over time.
- Dive deeper into using TF-IDF to unearth more findings.
- Create an effective model for predicting popularity of songs based off of lyrical choice in a designated time frame.