1. Data Collection

I collected data through the Spotify API. I had to register my application on Spotify which gave me a unique key and private key. I needed these credentials in order to access the API through the Jupyter notebook. I made methods to extract specific kinds of data. To access Spotify's API for Python users I had to install "spotipy". However, I noticed that the "get_categories" and "get_category_playlists" did not actually work the way it is supposed to work. I went on Spotify's website to use their API and those methods worked differently on the official Spotify website than the Spotipy library for Python users. The Python version only returns the featured playlists in the browse feature and not all of the playlists like it does on the offical Spotify interactive API website. To fix this problem, I converted cURL requests to the official Spotify website into python requests. Requests to the API returned JSON files so I made methods depending on what I wanted from the JSON file. For example, sometimes I only needed the playlist_id or track_id of all the playlists and tracks but the JSON included a lot of other information too like the artists, album names, country, etc. I made methods that would only extract the necesary components for my project. Th

2 Data Format Description

The format of my datasets were multiple JSON files. I did not save these JSON files because I am making requests to the Spotify API so I am getting the most updated playlists and tracks. I am calling the API to get categories, playlists in categories, tracks in categories, and also audio features of each track. I focused on five categories: chill, hiphop, mood, indie_alt, and latin. The most relevant attributes within these files are the category ids, playlist ids, track ids, and the individual audio features of each track. These features include things like danceability, energy, mode, speechiness, acousticness, instrumentalness, liveness, and valence. I am using these features to analyze the trends between different categories on Spotify so that I can predict what category a song belongs to.

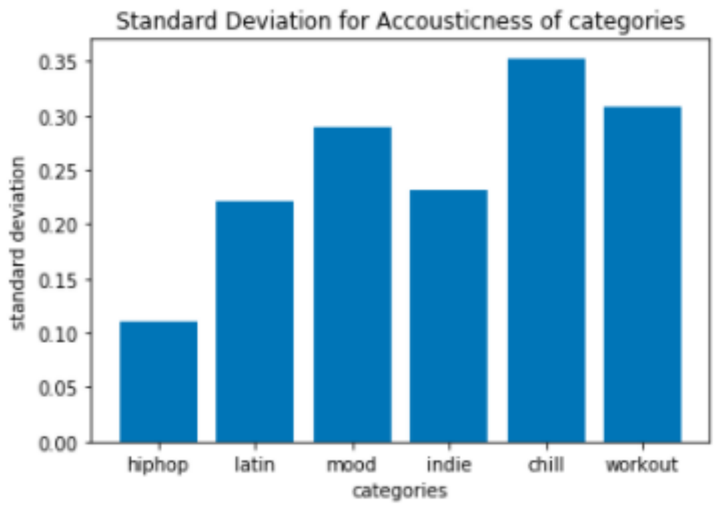
3 Descriptive Statistics

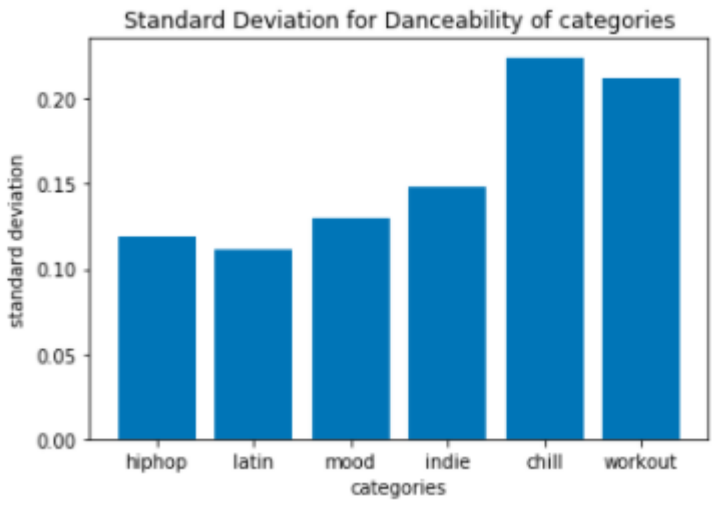
These are averages for all the audio features for 5 categories

Out[93]:

	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo
workout	0.5891	0.8259	5.3331	-5.0008	0.6139	0.1049	0.0781	0.0937	0.2063	0.4892	134.2434
chill	0.5112	0.3863	5.2815	-12.9346	0.7092	0.0593	0.5635	0.3670	0.1456	0.3180	111.9995
indie_alt	0.5180	0.7050	4.9919	-8.1119	0.7497	0.0529	0.1565	0.1463	0.1834	0.5612	125.9208
mood	0.6735	0.8354	5.6209	-5.3983	0.5349	0.0875	0.0641	0.2038	0.2172	0.5177	125.9416
latin	0.6735	0.8354	5.6209	-5.3983	0.5349	0.0875	0.0641	0.2038	0.2172	0.5177	125.9416
hiphop	0.6735	0.8354	5.6209	-5.3983	0.5349	0.0875	0.0641	0.2038	0.2172	0.5177	125.9416

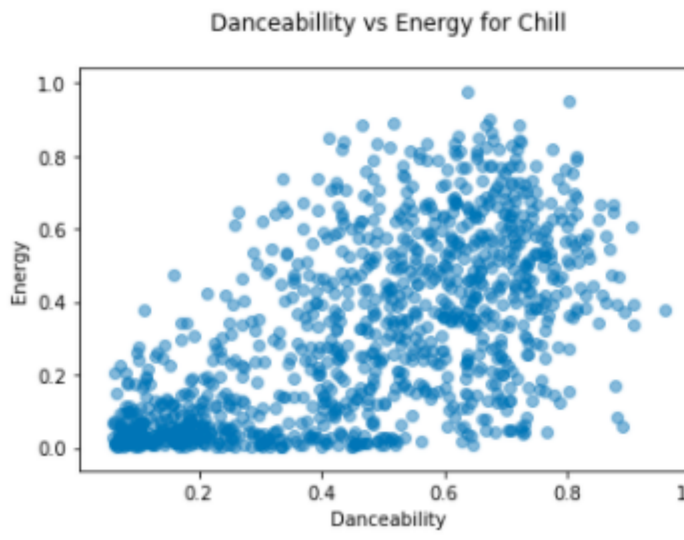
In these graphs, hiphop does not show a high standard deviation for both accousticness and danceability. This probably means out of all categories hiphop tends to cluster around the same number which can make it easier to detect if a song is hiphop or not since there will be a larger probability that certain numbers correlate with hiphop.

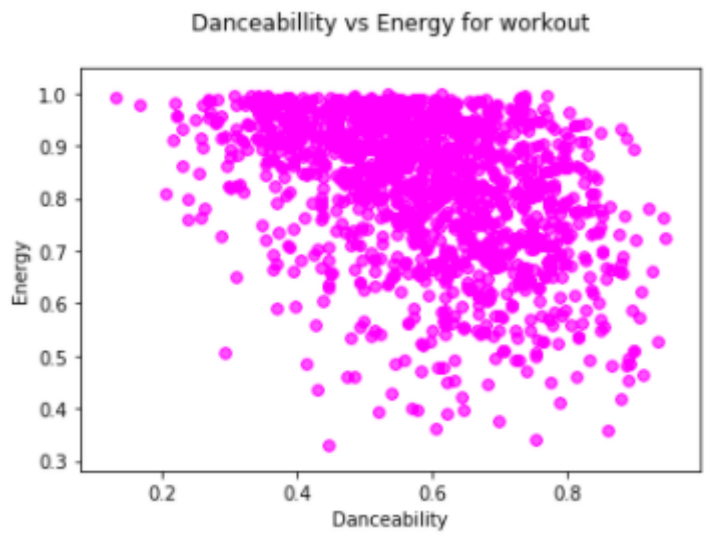


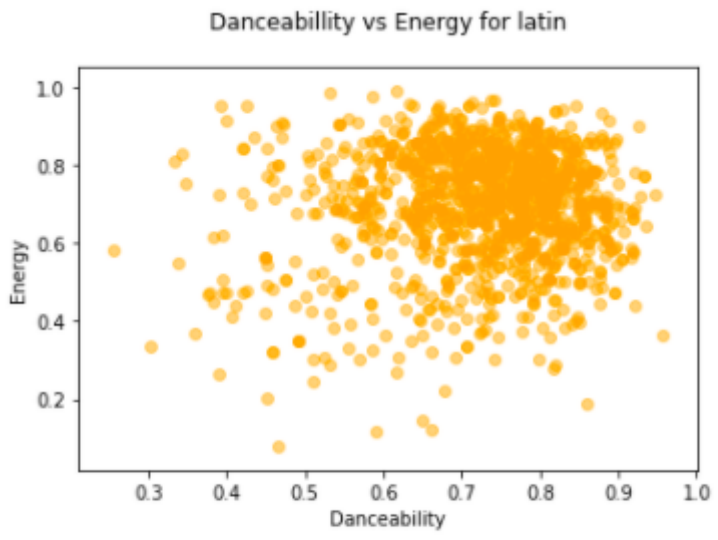


4 Data Analysis, Visualization, and Insights

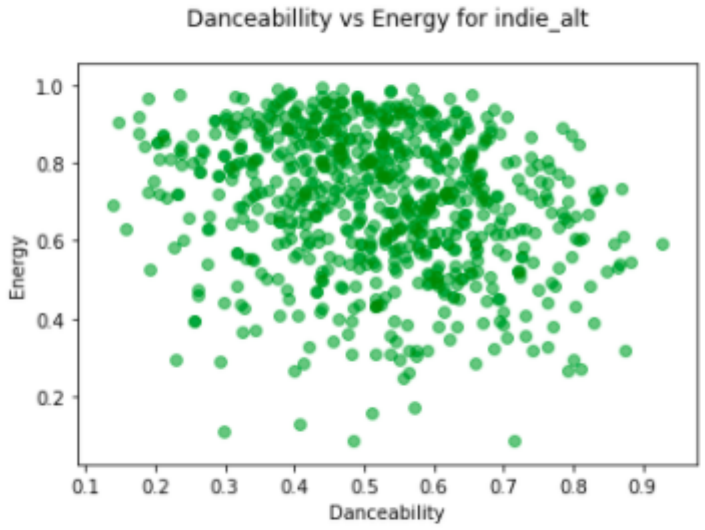
Danceability vs Energy scatter plots From the statistics from the dataframe above, I knew that danceability had around the same standard deviation except for chill. Chill had one of the greater standard deviations for danceability which is why I chose this feature. I thought it would be easier to find clusters between the other categories. However, most of them looked pretty similar except for chill and hiphop. Most of the data clustered around the same parts.

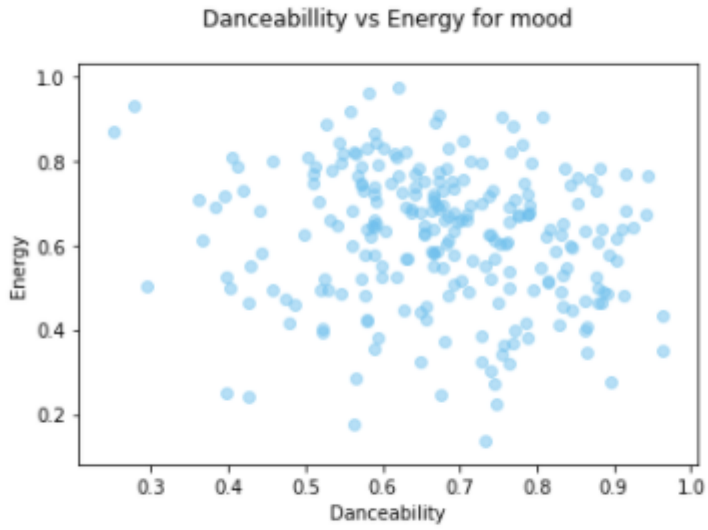

Danceabillity vs Energy for Chill
Energy
Danceability
1.0
0.8
0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8
1.0


Danceabillity vs Energy for workout
Energy
Danceability
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.4
0.6
0.8

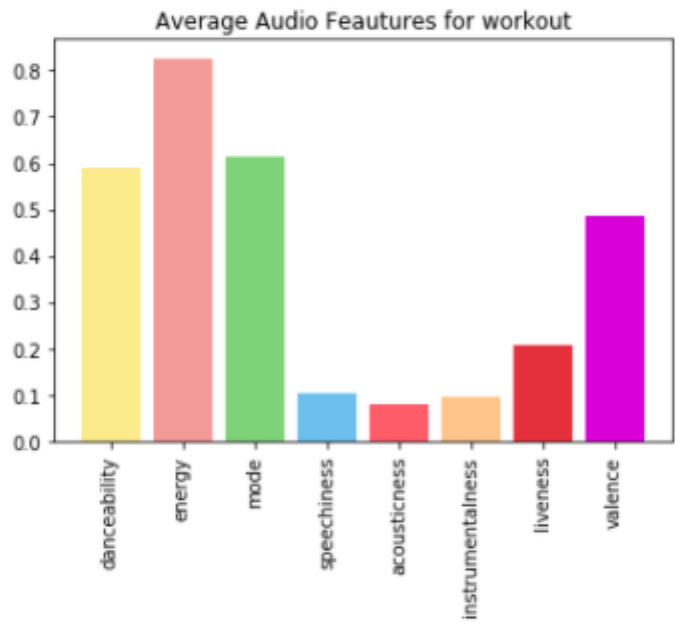

Danceabillity vs Energy for latin
Energy
Danceability
1.0
0.8
0.6
0.4
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0

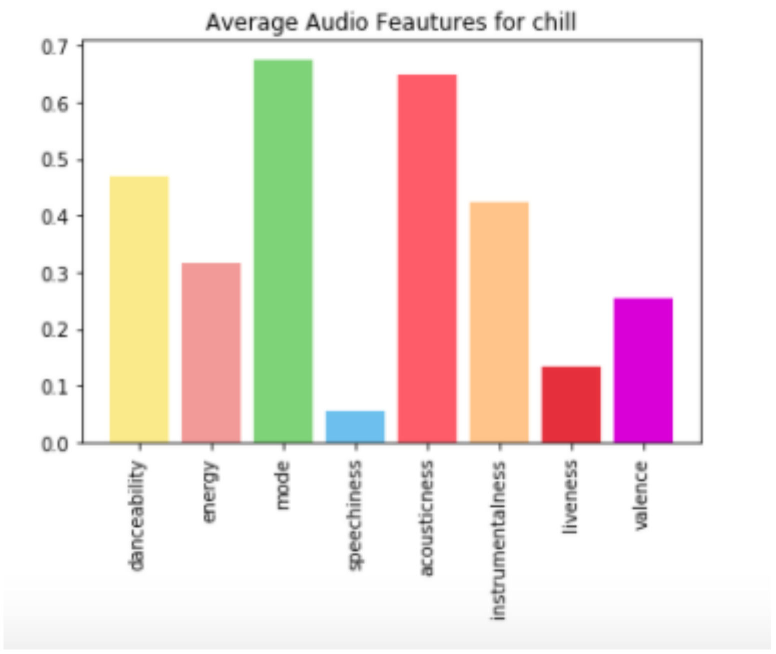
```
Out[49]: <matplotlib.collections.PathCollection at 0x11f4088d0>
```

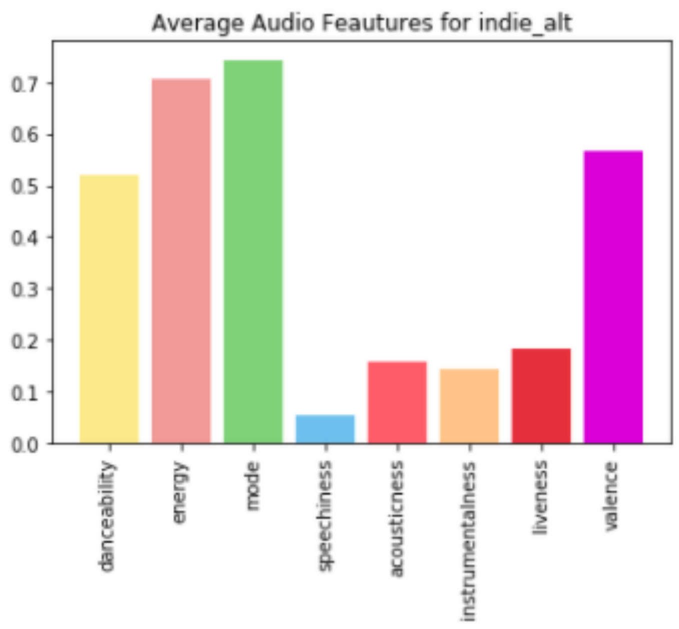


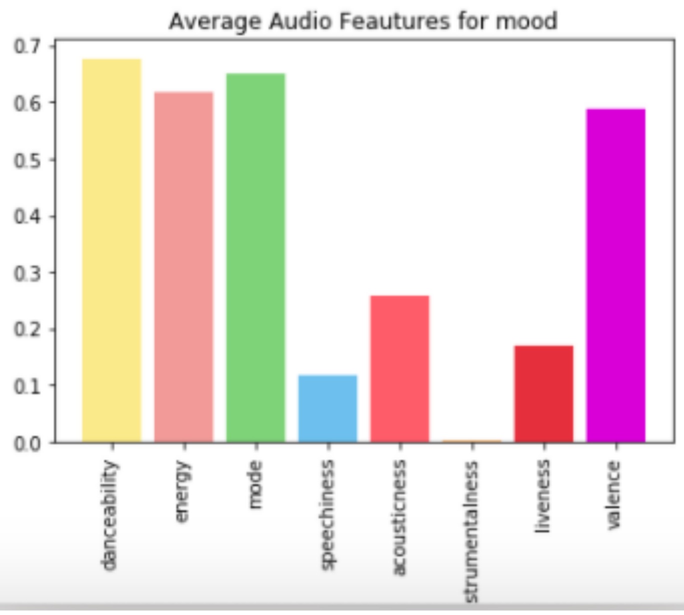


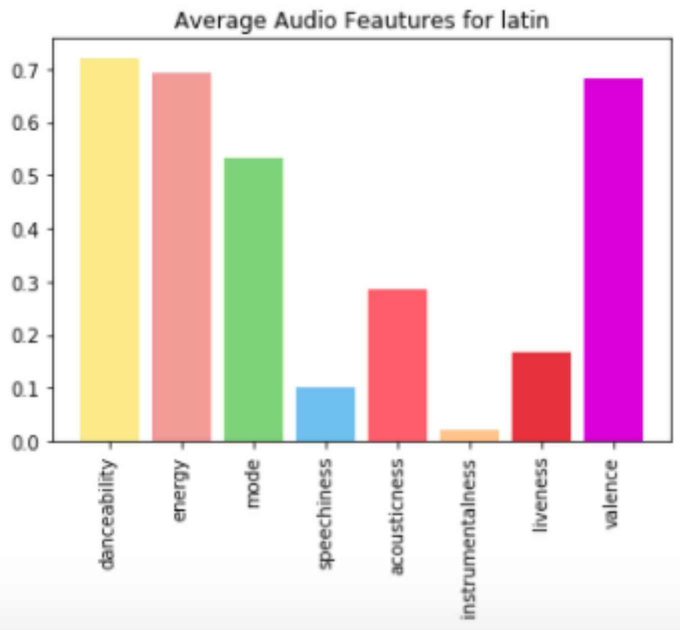
I also made a bar plot for all the averages of ever feature for every category since its hard to visualize with just the dataframe itself. With these graphs, we can tell that energy is high for almost every category with the exception for the "chill" category which definitely makes sense. There was a lot of variance between the averages for "acousticness" across the categories. Speechiness for all of them were relatively low and danceability for most of them were also pretty high and not too much of a difference between all of them. Mode was high for chill, indie and mood categories.

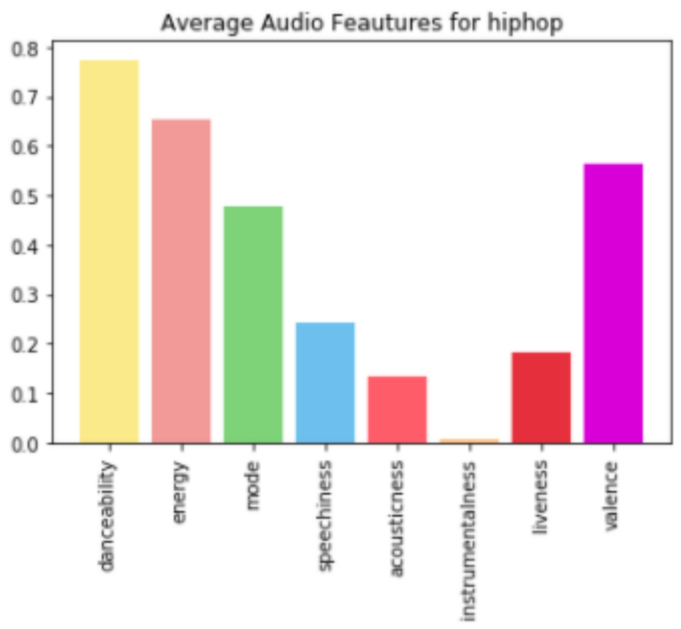

Average Audio Feautures for workout
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
danceability
energy
mode
speechiness
acousticness
instrumentalness
liveness
valence


Average Audio Feautures for chill
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
danceability
energy
mode
speechiness
acousticness
instrumentalness
liveness
valence

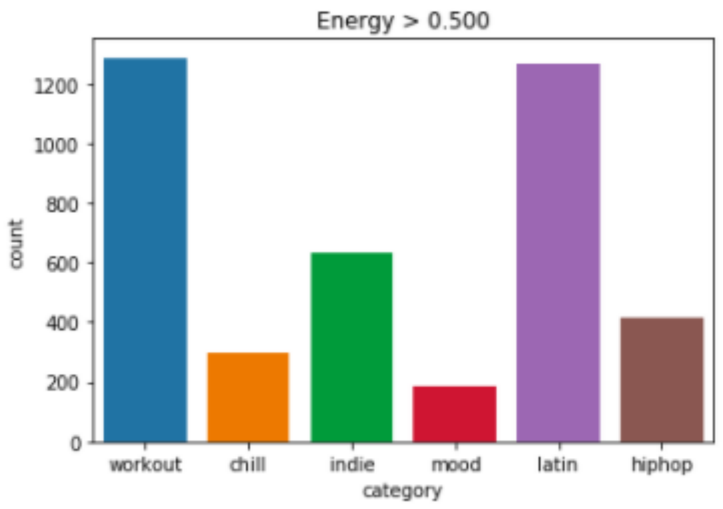

Average Audio Feautures for indie_alt
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
danceability
energy
mode
speechiness
acousticness
instrumentalness
liveness
valence


Average Audio Feautures for mood
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
danceability
energy
mode
speechiness
acousticness
strumentalness
liveness
valence


Average Audio Feautures for latin
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
danceability
energy
mode
speechiness
acousticness
instrumentalness
liveness
valence

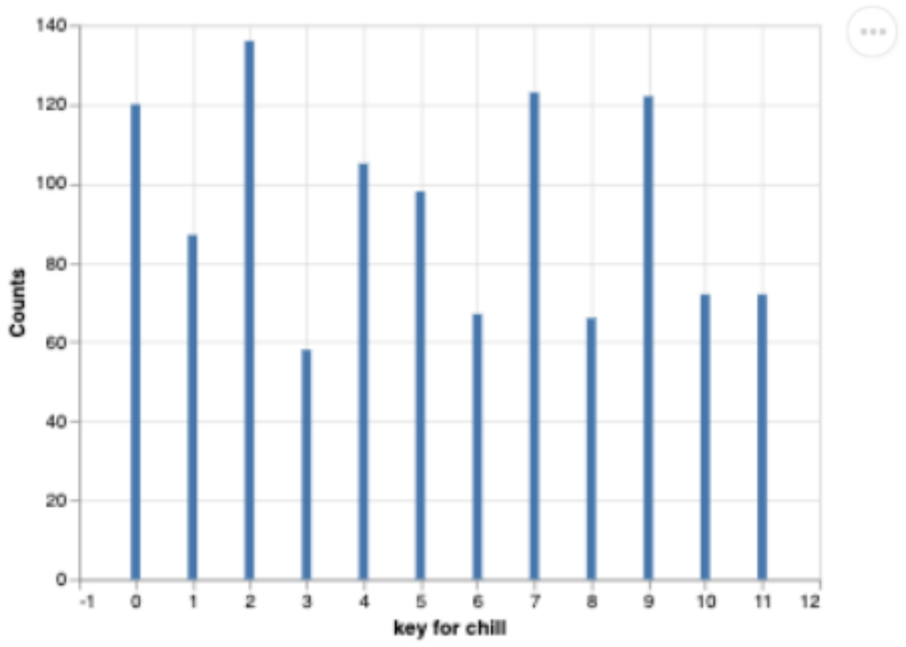


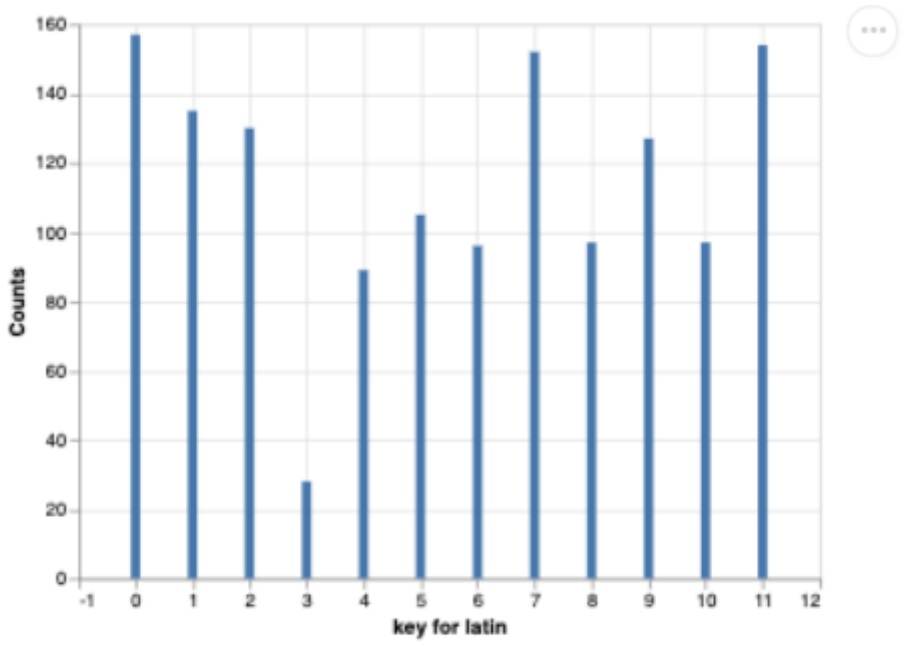
From the bar graph and scatter plots, I knew that energy would be an important feature that distinguishes the different categories so I graphed the energy levels. I know that I currently do not have equal amount of data across all categories but I will fix that as soon as I can figure out how to pull all the playlists and not just the featured ones through the spotipy API.



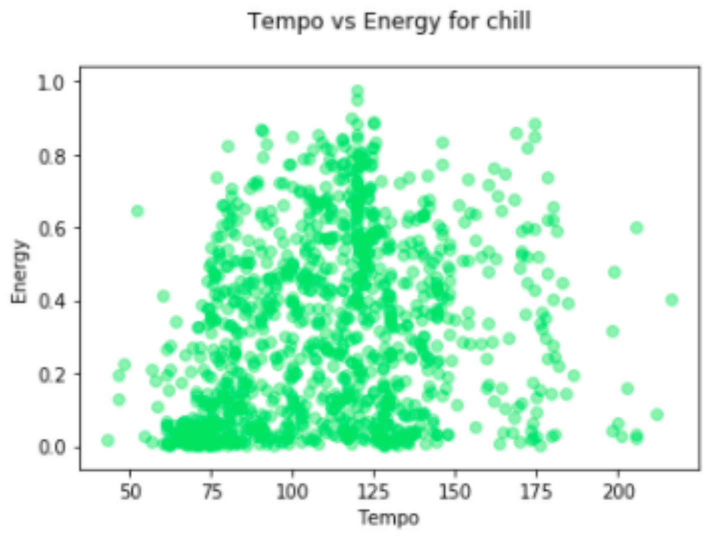
I also noticed that the keys that are most common also differ a lot within categories. Latin and Chill differ a lot with what keys are used.

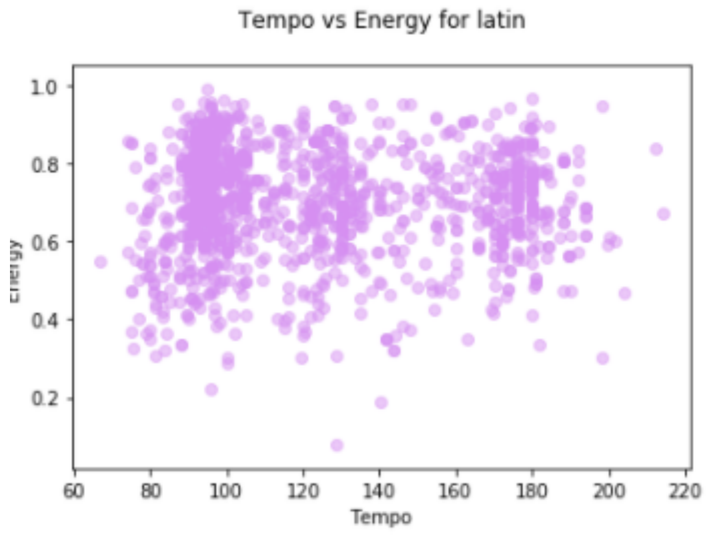
Out[79]:





Scatter plots showing two different features show the dependence of them and if there is a high or low correlation between them depending on the category. For these two, the clustering at different location for both categories can clearly be seen. Chill tempo lays towards the bottom while latin categories have higher tempos. There is also a wider distribution for chill while latin there are 3 distinct clusters.


Tempo vs Energy for chill
Energy
Tempo
0.0
0.2
0.4
0.6
0.8
1.0
50
75
100
125
150
175
200


Tempo vs Energy for latin
Energy
Tempo
0.2
0.4
0.6
0.8
1.0
60
80
100
120
140
160
180
200
220

In [ ]:
```python
import altair as alt

plt.figure(figsize=(10,10))
df_temp = alldfs[(alldfs['category']=='chill')]
df_temp = df_temp["key"].value_counts()
df_temp = df_temp.reset_index()
df_temp.rename(columns={"index": "key for 'chill'", "key": "Counts"})

newdf = df_temp.rename(columns={"index": "key for chill", "key": "Counts"})
alt.Chart(newdf).mark_bar().encode(
    x='key for chill',
    y='Counts:Q'
) #for chill category


plt.figure(figsize=(10,10))
df_temp = alldfs[(alldfs['category']=='latin')]
df_temp = df_temp["key"].value_counts()
df_temp = df_temp.reset_index()
df_temp.rename(columns={"index": "key for 'latin'", "key": "Counts"})

newdf = df_temp.rename(columns={"index": "key for latin", "key": "Counts"})
alt.Chart(newdf).mark_bar().encode(
    x='key for latin',
    y='Counts:Q'
) #for latin category
```

5 Future Plans

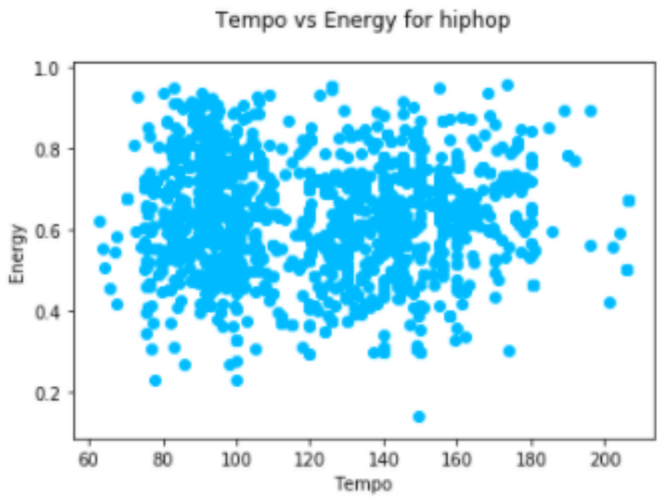
Next I plan to do more analysis and add more data to the categories that do not have enough. I want to use clustering techniques for certain features and see if clustering can detect any obvious differences between the categories. I also need to come up with what numbers are the most important for each category and create bounds of some sort. I plan to construct bar graphs with certain bounds to see what categories fall under a certain bound more. So, if I want to compare speechiness, I might make bounds that (0.0.2, 0.2-0.4..etc) and see what categories fall under which bound more frequently. Using these techniques I hope to come up with features that I can use for machine learning.

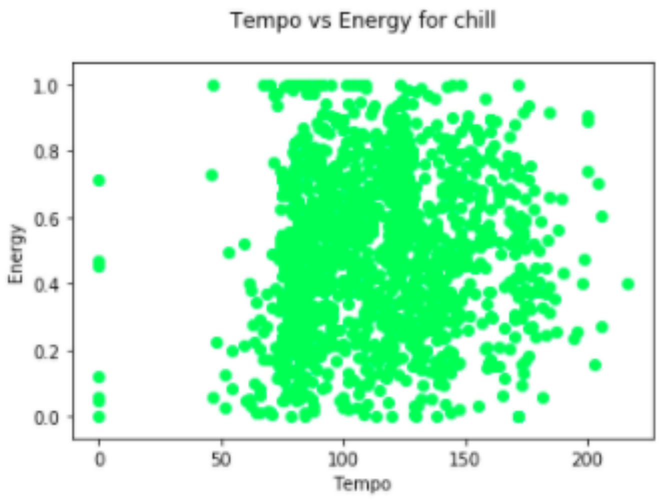
**Also, what machine learning methods do you recommend for predicting what category a song belongs to? I know I didn't explain exactly what categories are but they're similar to genres. On Spotify, there is a browse feature that leads you to a number of categories for different occasions like dinner, sleep, wellness..etc. Within these categories there are playlists that suit that specific category.*

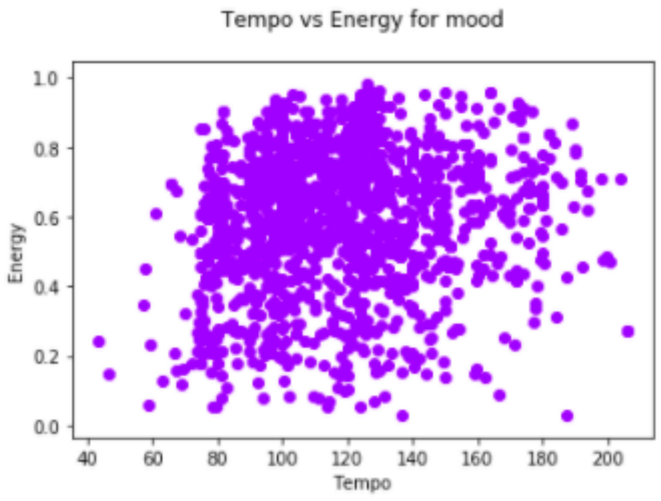
FINAL - PART 2

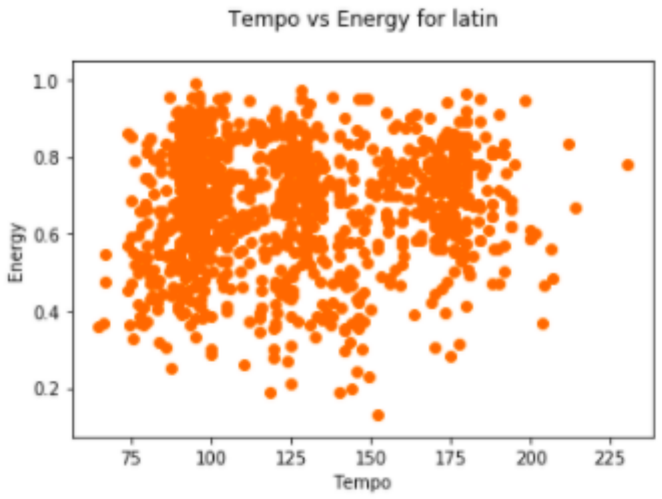
More Visualizations

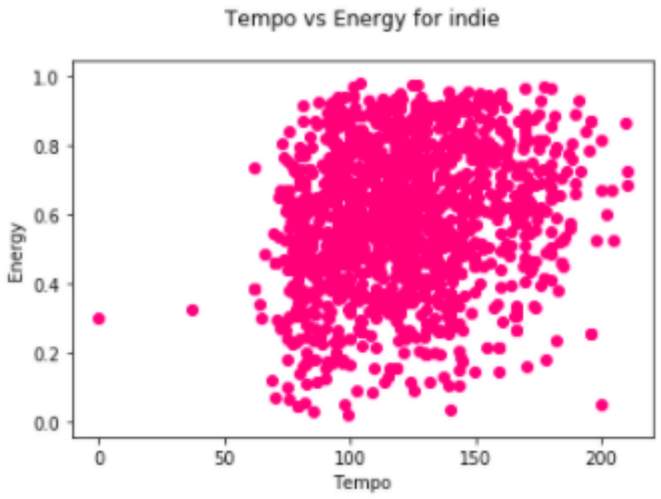
Tempo vs Energy I used these features for a comparision because I hypothesized that by knowing the category names, the tempo and energy would be features that would vary a lot between the different categories. However, the points do not accumulate in completely different areas for all categories. It is really only distinguishable in hiphop, chill, and indie. For the others, it is hard to tell because they all seem to accumulate in around the same areas.



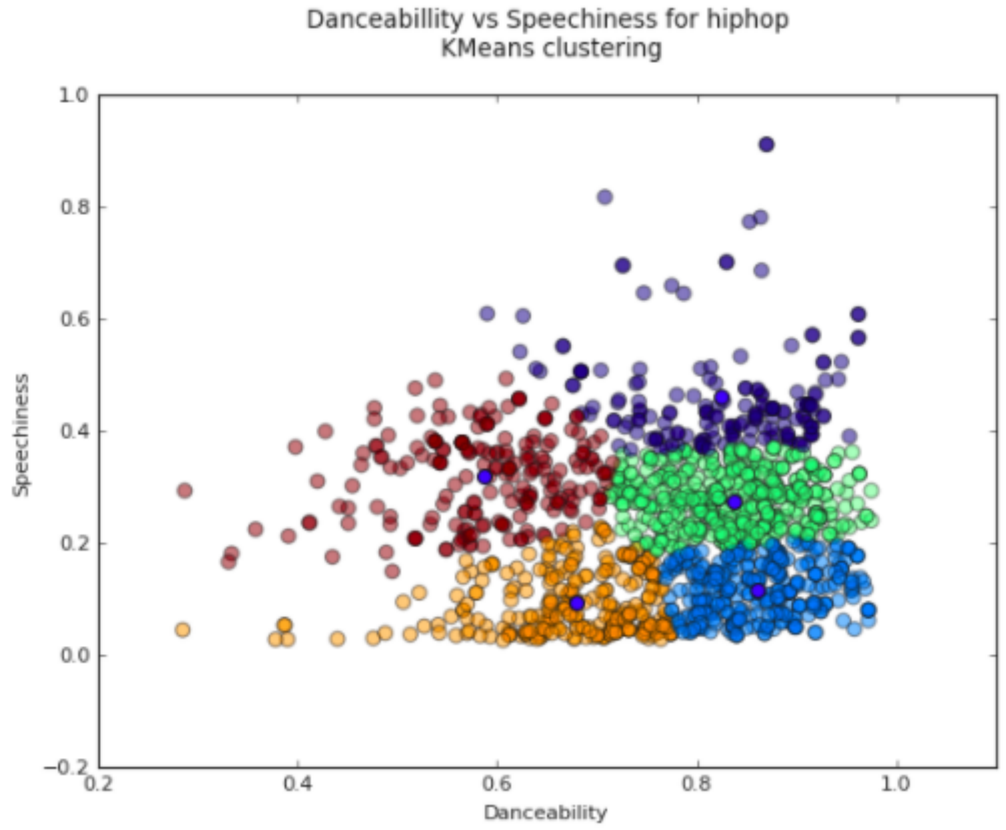


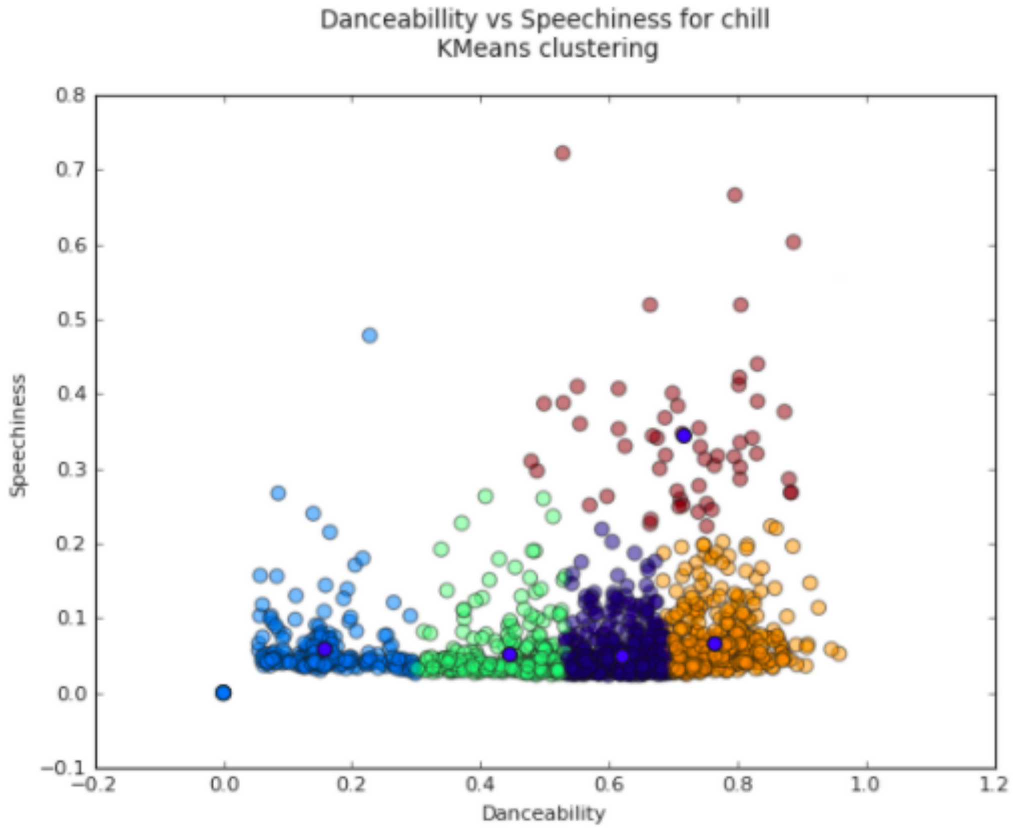


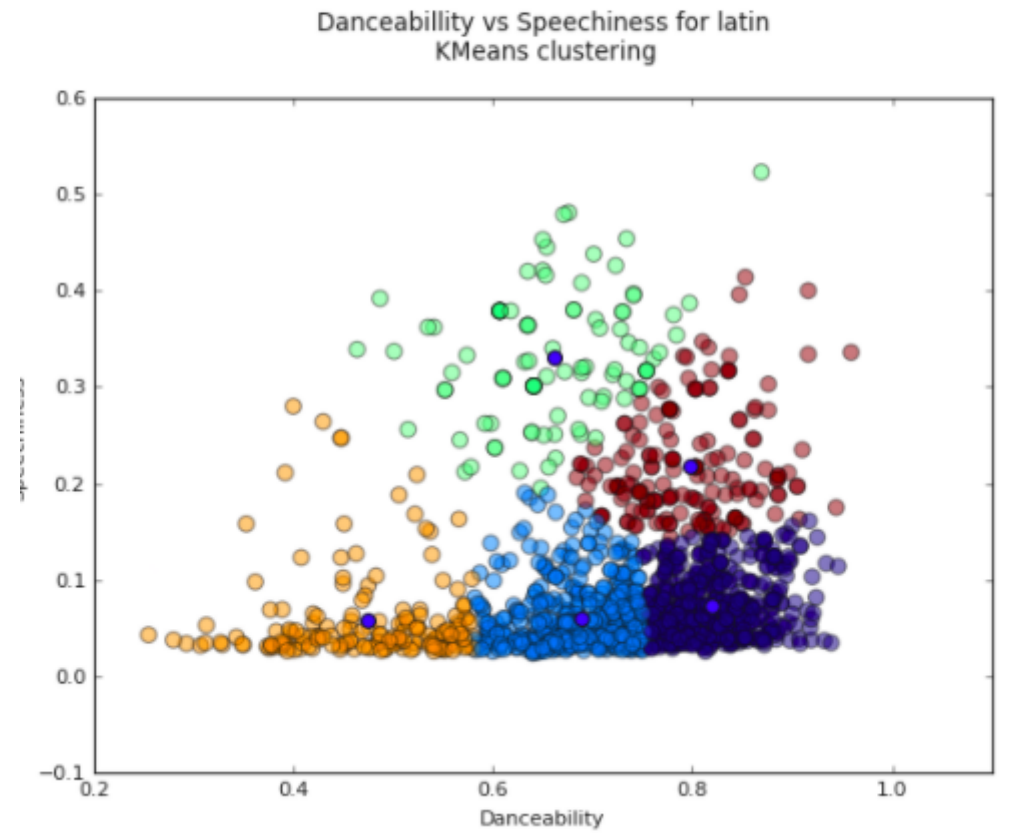


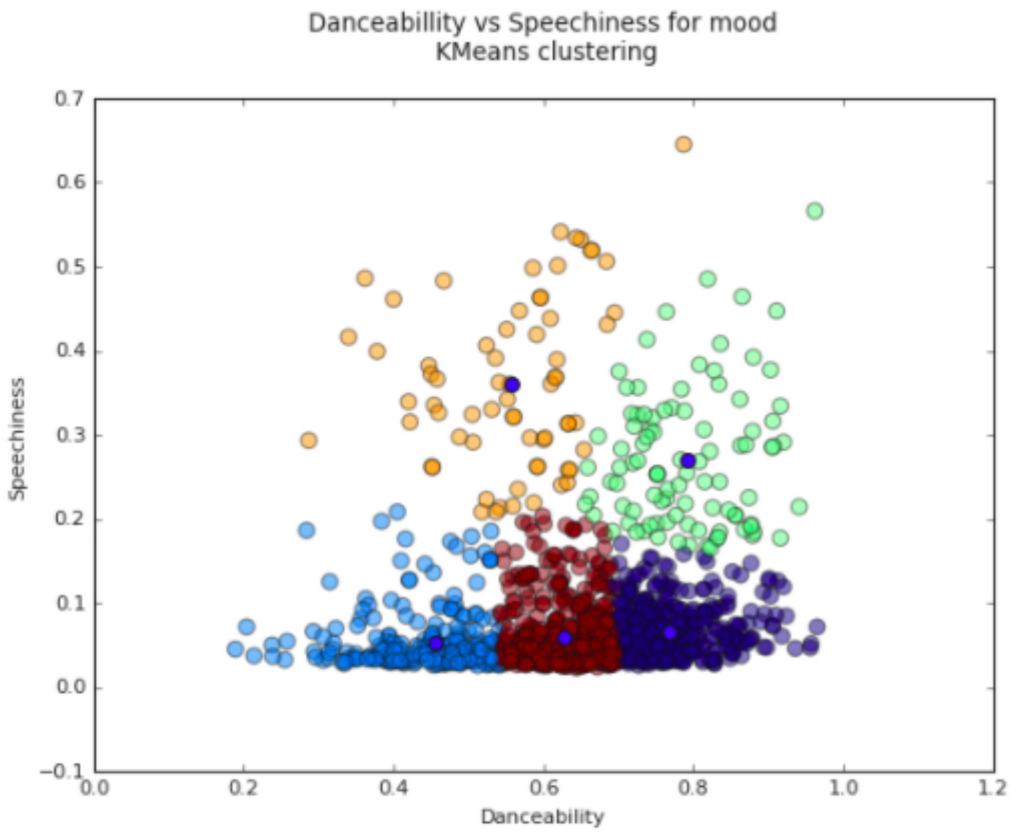


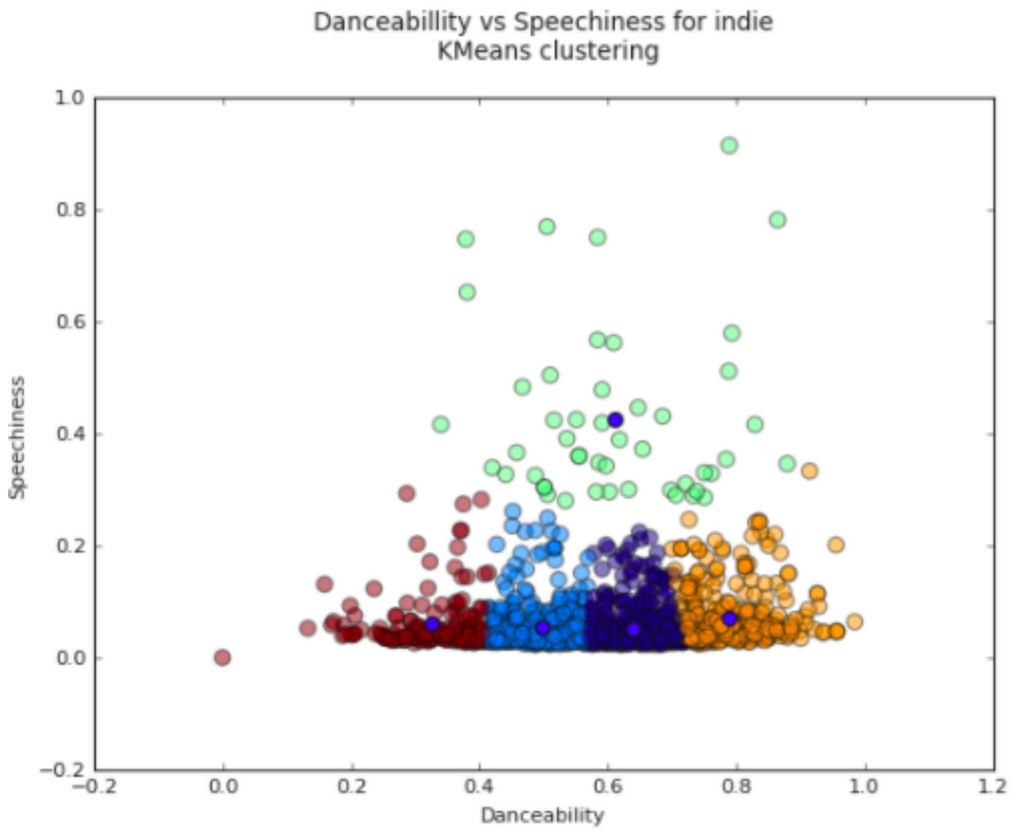
K-Means Clustering I also knew that speechiness would vary between the genres so I decided to use K-Means clustering so I can see where the different clusters are forming for danceability vs speechiness. The last pictures shows the clusters for all the categories. We can see the in the blue cluster, indie and chill most likely do not belong there. Hiphop, latin, and mood have higher probabilities of being in the blue cluster. Hiphop also most likely would not belong in the green or orange clusters. (mostly towards the bottom). Also as predicted, since hiphop had a smaller standard deviation from in the features, the clusters are also clustered together the most compared to the other categories.

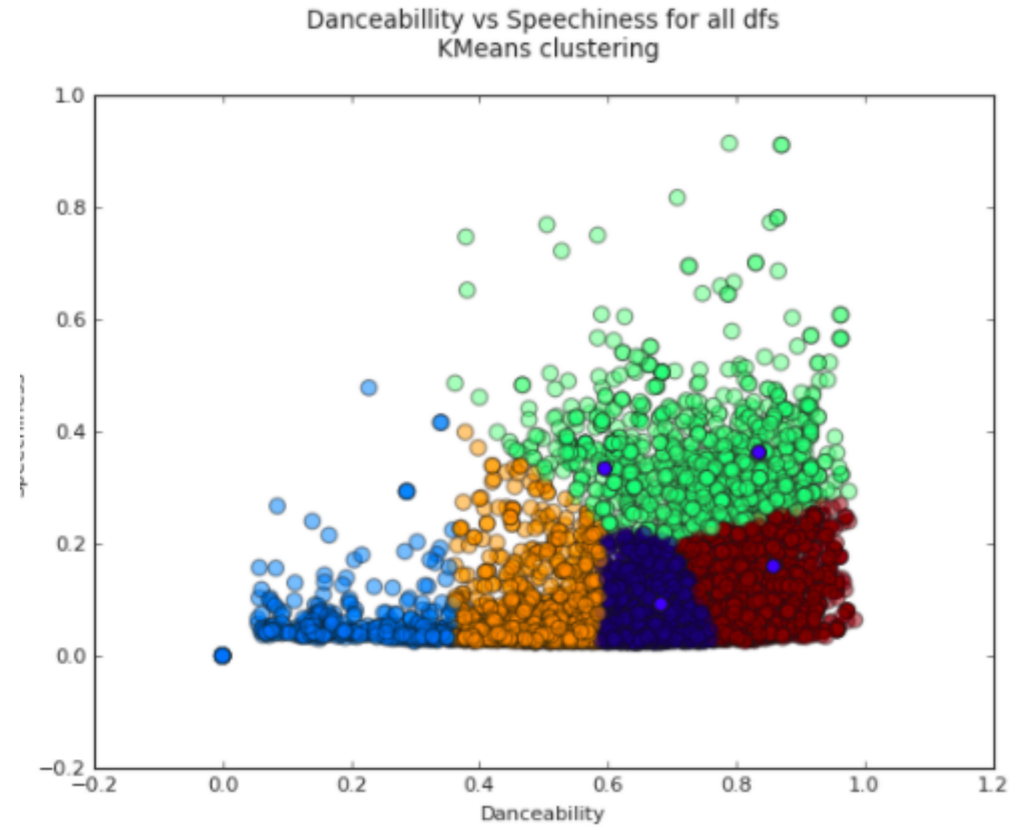

Danceabillity vs Speechiness for hiphop
KMeans clustering
Speechiness
Danceability
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.2
0.4
0.6
0.8
1.0


Danceabillity vs Speechiness for chill
KMeans clustering
Speechiness
Danceability
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

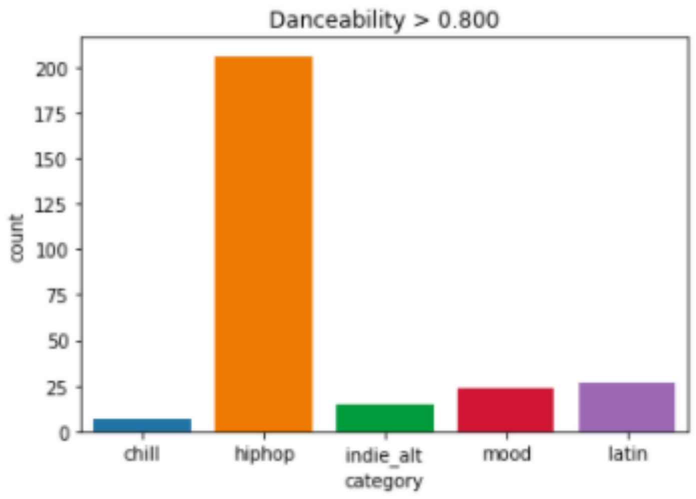

Danceabillity vs Speechiness for latin
KMeans clustering
Speechiness
Danceability
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0.2
0.4
0.6
0.8
1.0

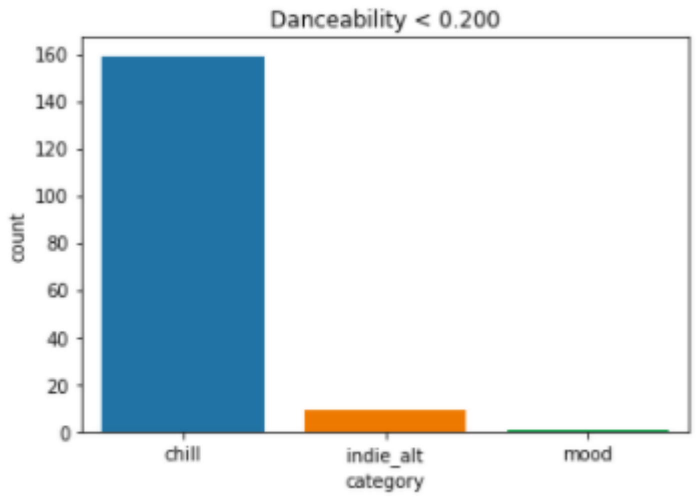

Danceabillity vs Speechiness for mood
KMeans clustering
Speechiness
Danceability
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−0.1
0.0
0.2
0.4
0.6
0.8
1.0
1.2

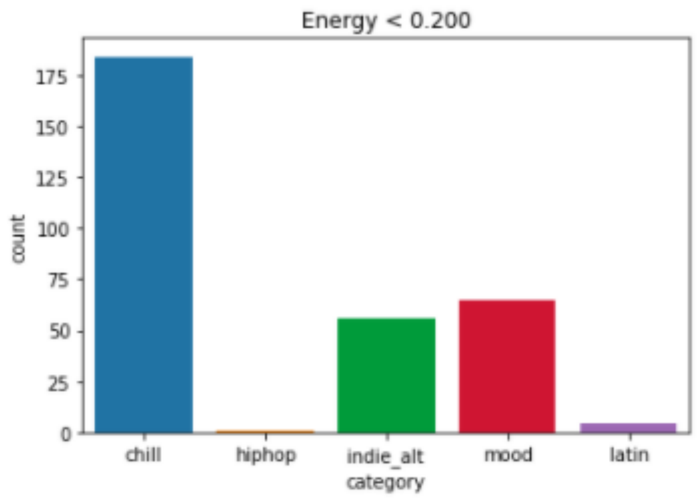


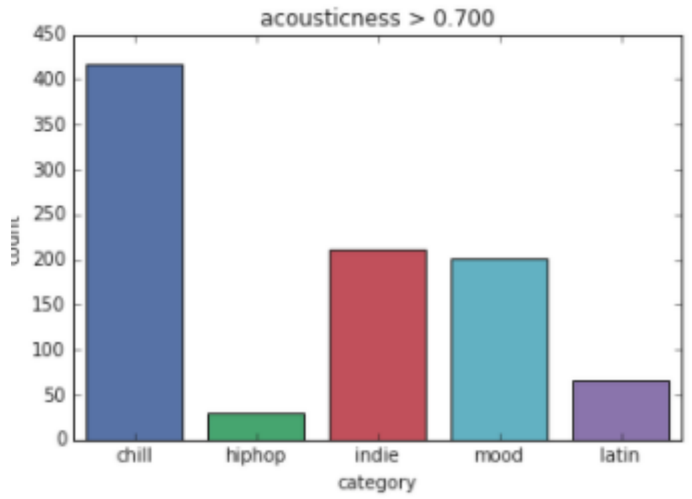


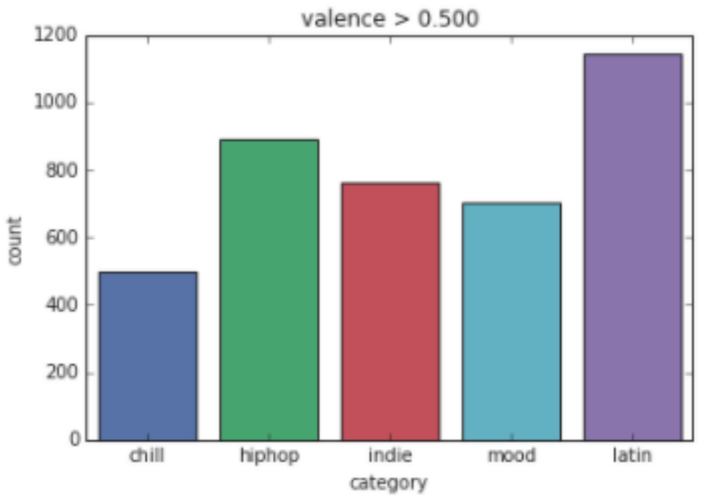
I used the seaborn library to get the counts of tracks that fell under a specific bound so I can get a clearler picture of what is different between the categories. These features also helped me add columns to my dataframe so that I can make my machine learning more accurate. I added columns like "danceability > 0.9" and put 0 or 1 depending on the tracks's danceability.


Danceability > 0.800
count
category
chill
hiphop
indie_alt
mood
latin
0
25
50
75
100
125
150
175
200


Danceability < 0.200
count
category
chill
indie_alt
mood
0
20
40
60
80
100
120
140
160


Energy < 0.200
count
category
chill
hiphop
indie_alt
mood
latin
0
25
50
75
100
125
150
175





Machine Learning

After my analysis I tried to see what features with what specific numbers had the most variability. As mentioned above, I used seaborn for these insights. I made my own features here and added them to the dataframe. Without my additional features, the accuracy of the test set was 48.82%. With my added features it was 52.32%

```
In [537]: #not normalized values
          import matplotlib.pyplot as plt
          import numpy as np
          import pandas as pd
          import seaborn as sns
          import tensorflow as tf

          from sklearn.model_selection import train_test_split
          from sklearn.linear_model import LogisticRegression

          shuffled = alldfs.sample(frac=1)
          shuf = copy
          train_dataset = shuffled.sample(frac=0.8,random_state=0)
          test_dataset = shuffled.drop(train_dataset.index)
          X_train, X_test, label_train, label_test = train_test_split(shuffled.drop('category', axis=1), shuffled['category'],
                                                                      test_size=0.2, random_state=112)

          logit = LogisticRegression(C = 0.95)
          lrfit = logit.fit(X_train, label_train)
          print("The score for logistic regression is")
          print("Test set: {:6.2f}%".format(100*logit.score(X_test, label_test)))
```

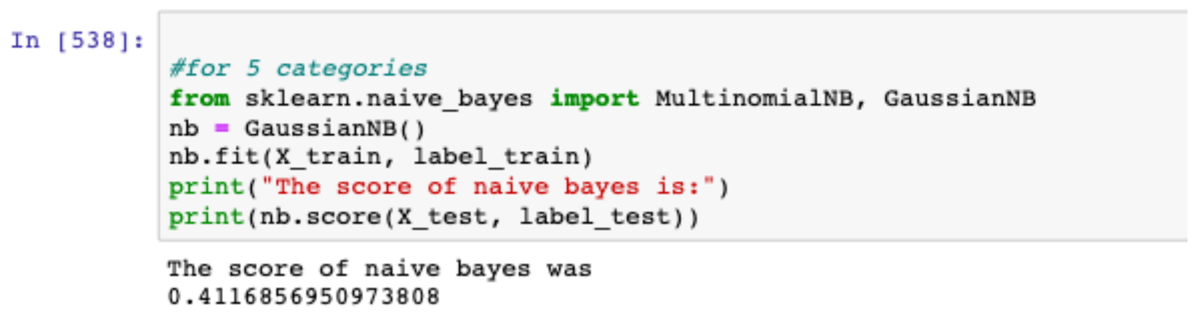
```
/Users/radhakatkamwar/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:432: FutureWarning:
Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/Users/radhakatkamwar/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:469: FutureWarning:
Default multi_class will be changed to 'auto' in 0.22. Specify the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

```
The score for logistic regression is
Test set:  52.32%
```

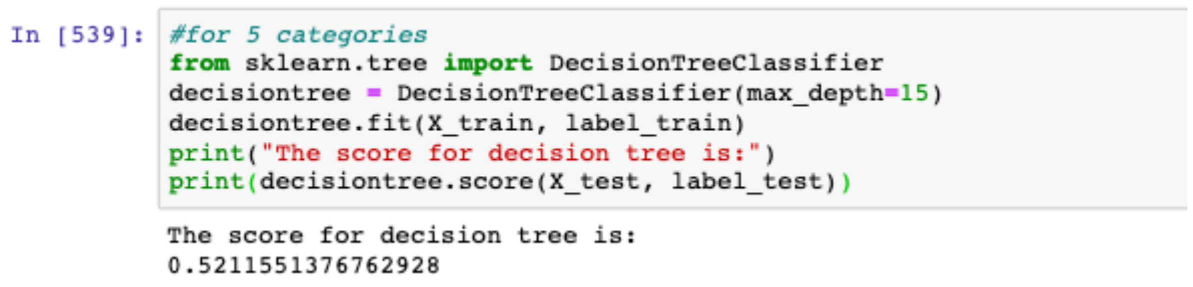
Logistic Regression Code for five cateogories(chill, hiphop, indie, mood, latin)

```
import numpy as np
#more features

alldfs['danceability > 0.9'] = np.where(alldfs['danceability']>=0.9, '1', '0')
alldfs['energy < 0.200']  = np.where(alldfs['energy'] <0.200, '1', '0')
alldfs['danceability < 0.2'] = np.where(alldfs['danceability']<0.2, '1', '0')
#alldfs['acousticness < 0.100'] = np.where(alldfs['acousticness']<0.100, '1', '0')
#alldfs['valence > 0.500'] = np.where(alldfs['valence'] > 0.500, alldfs['valence']*0.7 , '0')

alldfs
```

Naive Bayes and Decision Tree for five categories(chill, hiphop, indie, mood, latin) --> 1,2,3,4,5 respectively



```
In [538]: #for 5 categories
          from sklearn.naive_bayes import MultinomialNB, GaussianNB
          nb = GaussianNB()
          nb.fit(X_train, label_train)
          print("The score of naive bayes is:")
          print(nb.score(X_test, label_test))

          The score of naive bayes was
          0.4116856950973808
```



```
In [539]: #for 5 categories
          from sklearn.tree import DecisionTreeClassifier
          decisiontree = DecisionTreeClassifier(max_depth=15)
          decisiontree.fit(X_train, label_train)
          print("The score for decision tree is:")
          print(decisiontree.score(X_test, label_test))

          The score for decision tree is:
          0.5211551376762928
```

Logistic Regression for three categories(chill, hiphop, indie) --> 1,2,3 respectively

```
#for 3 categories
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import tensorflow_docs.modeling
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression


train_dataset3 = threedfs.sample(frac=0.8,random_state=0)
test_dataset3 = threedfs.drop(train_dataset3.index)

X_train3, X_test3, label_train3, label_test3 = train_test_split(threedfs.drop('category', axis=1), threedfs['category'
                                                                test_size=0.2, random_state=112)


logit = LogisticRegression(C = 0.95)
lrfit = logit.fit(X_train3, label_train3)
print("The accuracy for logistic regression is")
print("Test set: {:6.2f}%".format(100*logit.score(X_test3, label_test3)))
```

```
The accuracy for logistic regression is
Test set:  66.44%
```

Naive Bayes and Decision Tree for three categories

```
In [569]: #for 3 categories hiphop, chill, indie
          from sklearn.naive_bayes import MultinomialNB, GaussianNB
          nb = GaussianNB()
          nb.fit(X_train3, label_train3)
          print("The score for naive bayes is:")
          nb.score(X_test3, label_test3)

          The score for naive bayes is:

Out[569]: 0.6277777777777778
```

```
In [568]: #for 3 categories hiphop, chill, indie
          from sklearn.tree import DecisionTreeClassifier
          decisiontree = DecisionTreeClassifier(max_depth=15)
          print("The score for decision tree is:")
          decisiontree.fit(X_train3, label_train3)
          decisiontree.score(X_test3, label_test3)

          The score for decision tree is:

Out[568]: 0.6411111111111111
```

I also put the accuracies for including/not including certain features in the source code(labeled as a header as Logistic Regression values for 5 categories). I included the features that resulted in the best accuracy('danceability > 0.9', 'energy < 0.200','danceability < 0.2'). I assigned the value 0 or 1 depending on if the track met the condition. My added features improved accuracy because the features put more weight or emphasis on certain values. It still was not enough to give a good accuracy because the features were not a good indicator of track categories.

```
# Numerical Summary
All five categories:

Logistic Regression: 52.32%

Naive Bayes: 42.16%
```

```
Decision Tree: 52.11%



Three categories

Logistic Regression: 66.44%

Naive Bayes: 62.77%

Decision Tree: 64.11%
```

Conclusion Section

Originally I thought that I can predict what song a category belongs to just by the Spotify audio features. However, I did not realize how similar categories can be just based on those features alone. After doing a lot of analysis I realized I cannot predict something like this just based on the audio features(that spotify provided) alone. I would need advanced audio features to predict the category of a song. Even after adding some of my own features to the multivariable logistic regression, I was only able to improve the accuracy by 4%..48%-52%. So, the features did help but not too much. The features provided also feel subjective. Before extracting data and using different APIs I should pay more attention to what kind of data it is and try to see whether or not that data is helpful for my analysis/question. I decided to perform logistic regression on the three categories(hiphop, chill, indie) which showed the most differences. This resulted in the highest accuracy.

I performed naive bayes and the decision tree and realized logistic regression was still the best. Decision tree gave me the second best results and naive bayes gave me the worst. I tried to normalize the values in the dataframe, but that made my accuracy even worse so I decided to not normalize. It is most likely because my values were already meaningful and had a comparable scale.

In addition, I also learned a lot about APIs in general because I never played around with them before. It took a long time to correctly extract the information from the JSON files and because it was live, it took a long time for it to load. I made mistakes along the way with the JSON files which would crash my code. I did not extract information correctly so it would crash at random times and I would have to go back to fixing some of the first few methods.

What else would you do if you had more time (or could start over)?

I would have started over and tried to use a different API to focus on lyrics and the music itself. If I used something that allowed me to analyze tracks at every second and then graph the trends I would have gotten a better result. I would want to use something that would help me analyze the sound itself so I can do an analysis on what instruments are being played or what pitch the vocals are.

Acknowledgments section

Libraries used:

-pandas

-matplotlib

-altair

-numpy

-sklearn

-seaborn

-spotipy

Third Party Material

Used to set up environment

https://developer.spotify.com/documentation/web-api/quick-start/ (https://developer.spotify.com/documentation/web-api/quick-start/)

Used for Spotipy documentation: https://spotipy.readthedocs.io/en/2.12.0/ (https://spotipy.readthedocs.io/en/2.12.0/)

I used this to help me write code for logistic regression and understanding terminologies. https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8 (https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8)

https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1 (https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1)

I used this to help me write the code for the k-means clustering https://datatofish.com/k-means-clustering-python/ (https://datatofish.com/k-means-clustering-python/)

I used this to familiarize myself more with the library and to learn how to use logistic regression https://www.youtube.com/playlist?list=PL5-da3qGB5ICeMbQuqbbCOQWcS6OYBr5A (https://www.youtube.com/playlist?list=PL5-da3qGB5ICeMbQuqbbCOQWcS6OYBr5A) (playlist for sklearn)

I copied the curl requests from here https://developer.spotify.com/discover/ (https://developer.spotify.com/discover/) (how I accessed spotify interactive API)

and then copied it here so I can put in Python format https://curl.trillworks.com/#python (https://curl.trillworks.com/#python)

```
In [ ]:
```