In [ ]:

```python
import pandas as pd
import statistics
from numpy import mean
from matplotlib import pyplot as plt
#df=pd.read_csv("/Users/baileylarea/Downloads/top50.csv", encoding="ISO-8859-1")
df = pd.read_csv(r"C:\Users\innes\Downloads\Datasets\top50.csv", encoding = "ISO-8859-1
print(df.head())
```

```
   Unnamed: 0                       Track.Name    Artist.Name           Genre  \
0           1                         Señorita   Shawn Mendes    canadian pop
1           2                            China       Anuel AA  reggaeton flow
2           3   boyfriend (with Social House)  Ariana Grande       dance pop
3           4  Beautiful People (feat. Khalid)     Ed Sheeran             pop
4           5      Goodbyes (Feat. Young Thug)    Post Malone         dfw rap

   Beats.Per.Minute  Energy  Danceability  Loudness..dB..  Liveness  Valence.  \
0               117      55            76              -6         8        75
1               105      81            79              -4         8        61
2               190      80            40              -4        16        70
3                93      65            64              -8         8        55
4               150      65            58              -4        11        18

   Length.  Acousticness..  Speechiness.  Popularity
0      191               4             3          79
1      302               8             9          92
2      186              12            46          85
3      198              12            19          86
4      175              45             7          94
```

In [ ]:

```python
import os
import warnings

warnings.filterwarnings('ignore') #filtering out any warning messages
```

In [ ]:

```python
df = df.rename(columns={'Track.Name': 'TrackName', 'Artist.Name': 'ArtistName', 'Beats.
print((df).head(5))
```

```
   Unnamed: 0                        TrackName     ArtistName           Genre  \
0           1                         Señorita   Shawn Mendes    canadian pop
1           2                            China       Anuel AA  reggaeton flow
2           3   boyfriend (with Social House)  Ariana Grande       dance pop
3           4  Beautiful People (feat. Khalid)     Ed Sheeran             pop
4           5      Goodbyes (Feat. Young Thug)    Post Malone         dfw rap

   BeatsPerMinute  Energy  Danceability  LoudnessdB  Liveness  Valence  \
0             117      55            76          -6         8       75
1             105      81            79          -4         8       61
2             190      80            40          -4        16       70
3              93      65            64          -8         8       55
4             150      65            58          -4        11       18

   Length  Acousticness  Speechiness  Popularity
0     191             4            3          79
1     302             8            9          92
2     186            12           46          85
```

```
3     198             12             19           86
4     175             45              7           94
```

In [ ]:
```python
from collections import Counter
genrevalues=(Counter(df['Genre'].values))
print(genrevalues)
```

```
Counter({'dance pop': 8, 'pop': 7, 'latin': 5, 'canadian hip hop': 3, 'edm': 3, 'canadia
n pop': 2, 'reggaeton flow': 2, 'dfw rap': 2, 'country rap': 2, 'electropop': 2, 'reggae
ton': 2, 'panamanian pop': 2, 'brostep': 2, 'trap music': 1, 'escape room': 1, 'pop hous
e': 1, 'australian pop': 1, 'atl hip hop': 1, 'big room': 1, 'boy band': 1, 'r&b en espa
nol': 1})
```

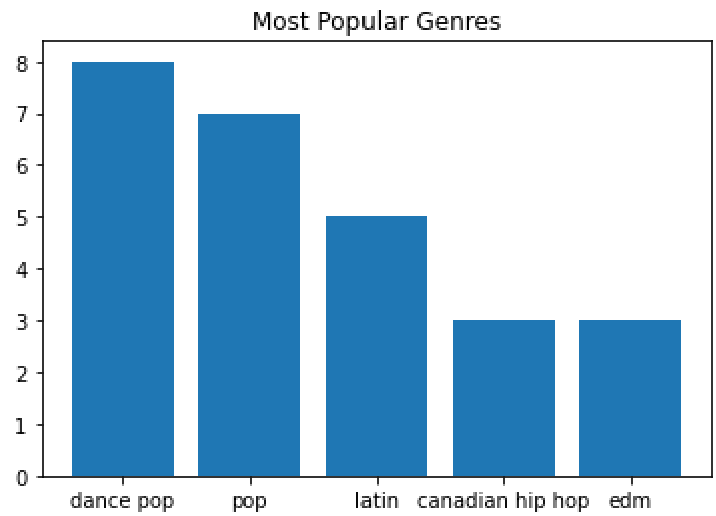
In [ ]:
```python
popsort=df.sort_values('Popularity', ascending=False, inplace=True)
print(df.head())
```

```
    Unnamed: 0                               TrackName     ArtistName  \
9           10                                 bad guy  Billie Eilish
4            5              Goodbyes (Feat. Young Thug)    Post Malone
10          11                                Callaita      Bad Bunny
14          15  Money In The Grave (Drake ft. Rick Ross)          Drake
1            2                                   China       Anuel AA

               Genre  BeatsPerMinute  Energy  Danceability  LoudnessdB  \
9         electropop             135      43            70         -11
4            dfw rap             150      65            58          -4
10         reggaeton             176      62            61          -5
14  canadian hip hop             101      50            83          -4
1     reggaeton flow             105      81            79          -4

    Liveness  Valence  Length  Acousticness  Speechiness  Popularity
9         10       56     194            33           38          95
4         11       18     175            45            7          94
10        24       24     251            60           31          93
14        12       10     205            10            5          92
1          8       61     302             8            9          92
```

In [ ]:
```python
mostpopgenres=((genrevalues).most_common(5))
print(mostpopgenres)
```

```
[('dance pop', 8), ('pop', 7), ('latin', 5), ('canadian hip hop', 3), ('edm', 3)]
```

In [ ]:
```python
import matplotlib.pyplot as plt
bar_plot=dict(mostpopgenres)
plt.bar(*zip(*bar_plot.items()))
plt.title('Most Popular Genres')
plt.show()
```



In [ ]:

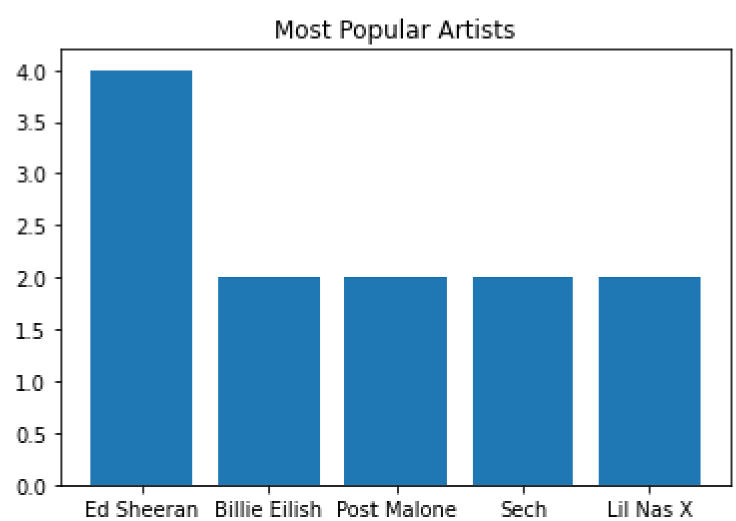
```
mostpopartists=dict(Counter(df['ArtistName'].values).most_common(5))
mostpopartists
```

Out[ ]:

```
{'Ed Sheeran': 4,
 'Billie Eilish': 2,
 'Post Malone': 2,
 'Sech': 2,
 'Lil Nas X': 2}
```

In [ ]:

```
bar_plot=dict(mostpopartists)
plt.bar(*zip(*bar_plot.items()))
plt.title("Most Popular Artists")
plt.show()
```



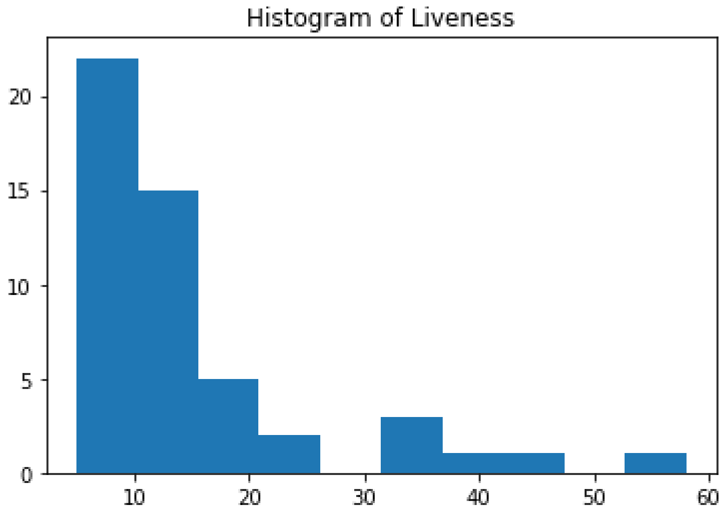
In [ ]:

```
### Finding the average amount of "liveness"
```

In [ ]:

```
avglive=(mean(df['Liveness'].values))
print(avglive)

plt.hist(df['Liveness'].values)
plt.title("Histogram of Liveness")
plt.show()
```

```
14.66
```



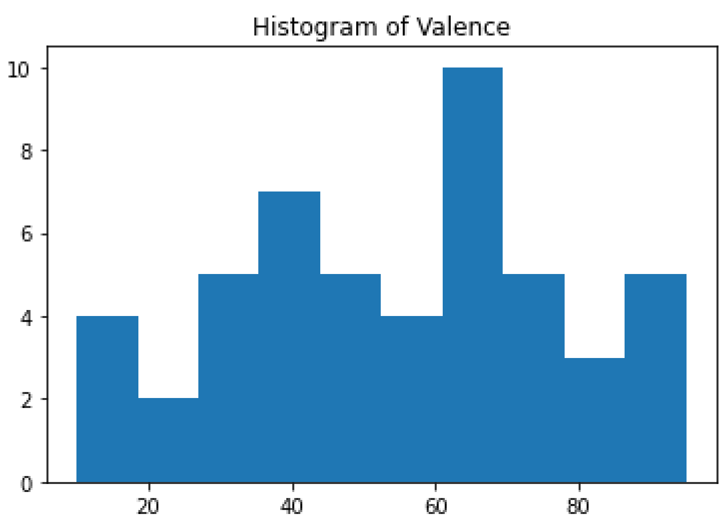
In [ ]:

```
### Finding the average amount of "valence"
```

In [ ]:

```
avgval=(mean(df['Valence'].values))
print(avgval)

plt.hist(df['Valence'].values)
plt.title("Histogram of Valence")
plt.show()
```

```
54.6
```



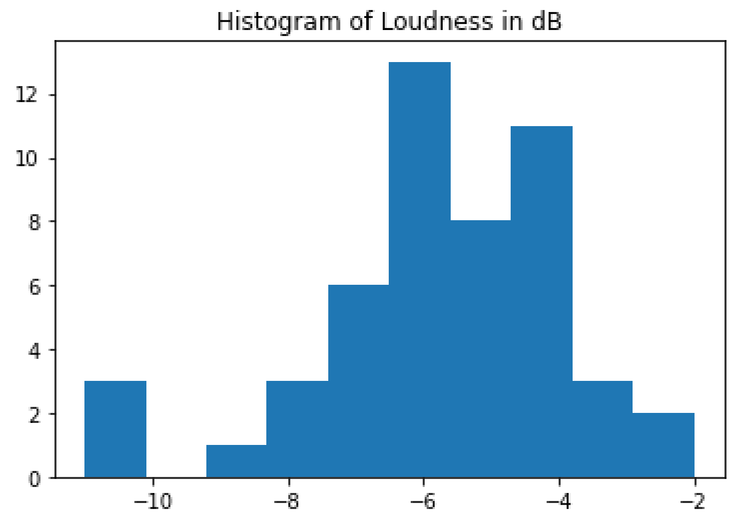
In [ ]:

```
### Finding the average amount of "loudness" in decibels (dB)
```

In [ ]:

```
avgloud=(mean(df['LoudnessdB'].values))
print(avgloud)
plt.hist(df['LoudnessdB'].values)
plt.title("Histogram of Loudness in dB")
plt.show()
```

```
-5.66
```



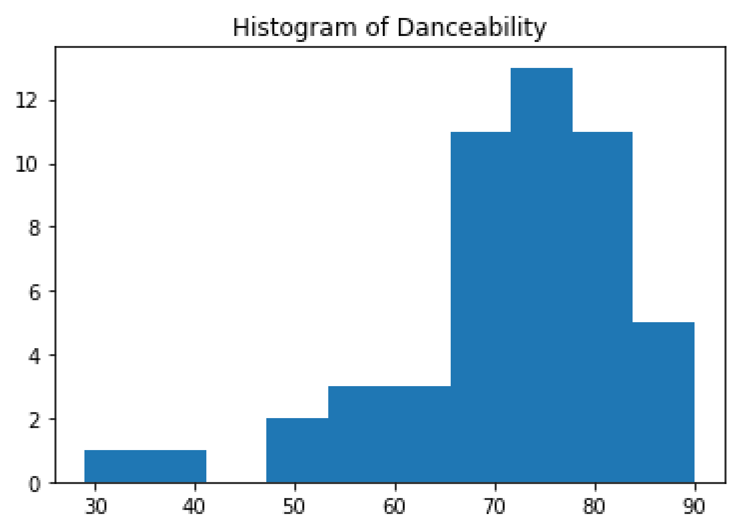
In [ ]:

```
### Finding the average amount of "danceability"
```

In [ ]:

```
avgdance=(mean(df['Danceability'].values))
print(avgdance)

plt.hist(df['Danceability'].values)
plt.title("Histogram of Danceability")
plt.show()
```

```
71.38
```



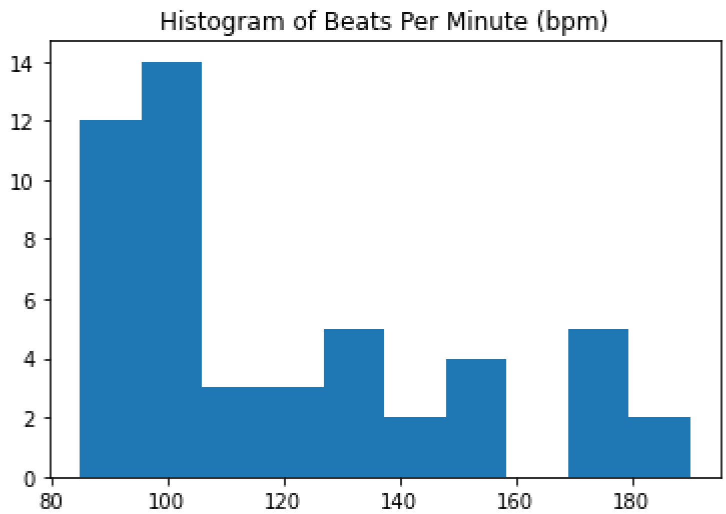
In [ ]:

```
### Finding the average amount of "beats per minute" (bpm)
```

In [ ]:

```
avgbpm=(mean(df['BeatsPerMinute'].values))
print(avgbpm)

plt.hist(df['BeatsPerMinute'].values)
plt.title("Histogram of Beats Per Minute (bpm)")
plt.show()
```

```
120.06
```



In [ ]:

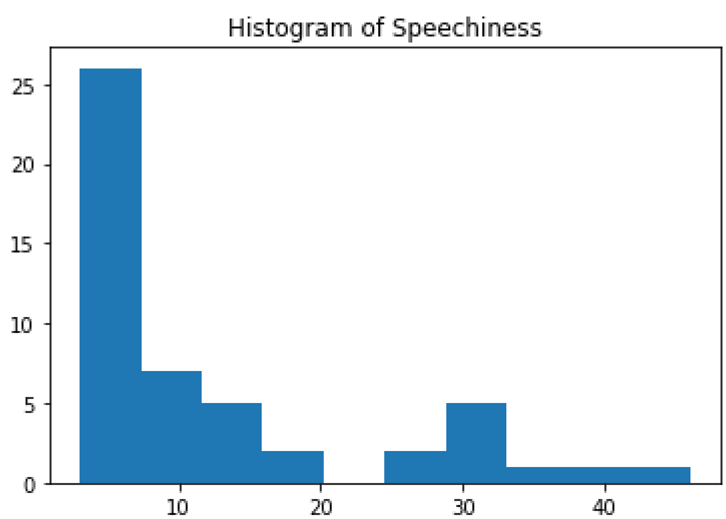
```
### Finding the average score of "popularity"
avgpop=(mean(df['Popularity'].values))
print(avgpop)
```

87.5

In [ ]:

```
avgspeech=(mean(df['Speechiness'].values))
print(avgspeech)

plt.hist(df['Speechiness'].values)
plt.title("Histogram of Speechiness")
plt.show()
```

12.48



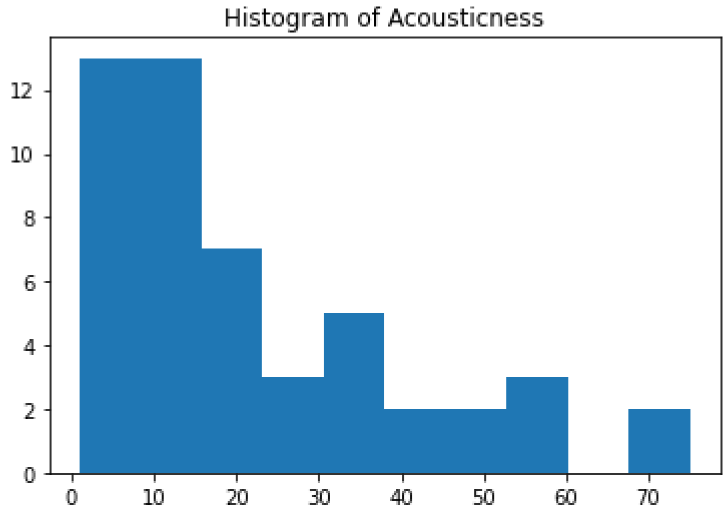
In [ ]:

```
### Finding the average amount of "acousticness"
```

In [ ]:

```
avgacc=(mean(df['Acousticness'].values))
print(avgacc)

plt.hist(df['Acousticness'].values)
plt.title("Histogram of Acousticness")
plt.show()
```

22.16



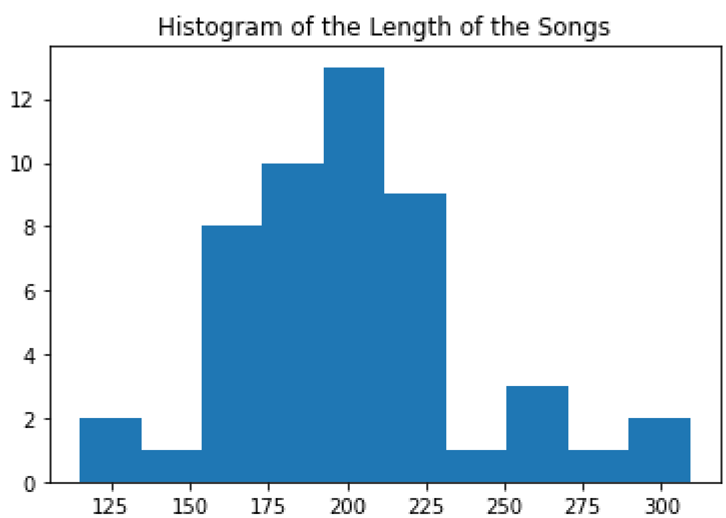
In [ ]:

```
### Finding the average "Length" of the song
```

In [ ]:

```
avglength=(mean(df['Length'].values))
print(avglength)

plt.hist(df['Length'].values)
plt.title("Histogram of the Length of the Songs")
plt.show()
```

200.96



Baseline: Linear Regression

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC, LinearSVC

from sklearn.metrics import make_scorer, accuracy_score, roc_auc_score
from sklearn.model_selection import GridSearchCV
```

```
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import seaborn as sns

#data=pd.read_csv("/Users/baileylarea/Downloads/2019songs.csv")
#data = pd.read_csv(r'C:\Users\innes\Downloads\Datasets\2019songs.csv')
data = pd.read_csv(r'C:\Users\innes\Downloads\Datasets\2019_songs_API.csv', encoding =
```

In [ ]:

```
data.dropna(inplace=True) # dropping NAs
```

In [ ]:

```
data.head(5) #print first 5 rows
```

Out[ ]:

	id	name	popularity	duration_ms	explicit	artists	id_
0	4zP7ADsgJgHGY6VzxbNp1z	Year 3000	67	201960	0	['Jonas Brothers']	['7gOdHgIoIKoe4i9Tta
1	5Q8lhXskGhfVIMbRMGi9nk	Logical Brain - Year 3000 Mix	8	216705	0	['Electro Mann']	['715ETHjAlf1sXM4vF
2	6GEOjP12NG6wnuQIg6NC5A	3000 Years of Lies - Original Version	9	308824	0	['UVB']	['1LiE3TKOyCds5Ggla
3	0soQcQgZGxvHa3MyQMVfes	Year 3000 (Live)	0	188145	0	['The Friars']	['6Fb8ldZIVQEaszpTmz
4	5g5fyKQcR8H5U5c7znvrgs	The Year 3000	0	393676	0	['Reza Golroo']	['5PgfADiJty3IuidZvC

In [ ]:

```
data = pd.get_dummies(data, columns=['time_signature', 'key', 'mode'], drop_first=True)

pd.options.display.max_columns = None
```

In [ ]:

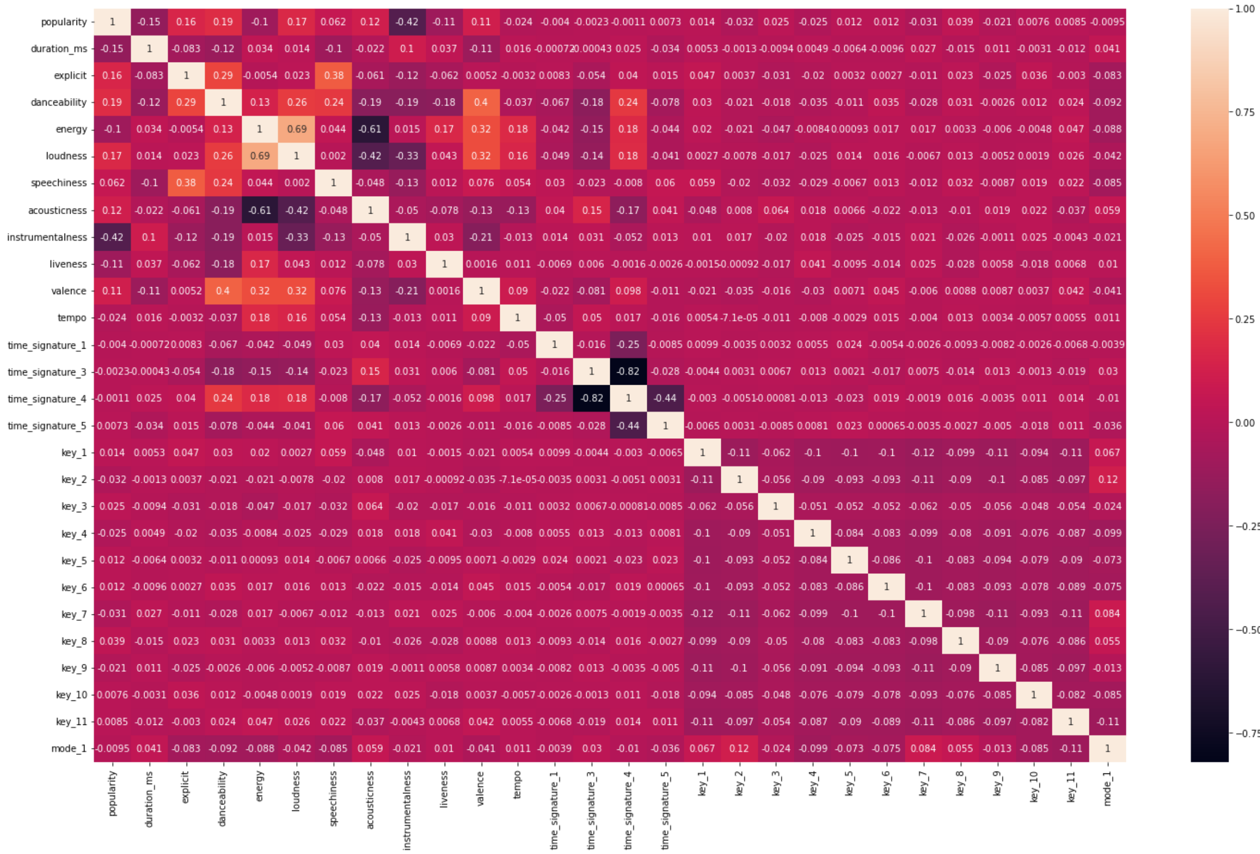
```
# defining our features
features = ["acousticness", "danceability", "duration_ms", "energy", "instrumentalness"
            "liveness", "loudness", "tempo",  "valence"]


X = data[features]
Y = data['popularity']
```

In [ ]:

```
features_2 = ['acousticness', 'instrumentalness', 'loudness','energy']
#X_train, X_test, Y_train, Y_test = train_test_split(X[features_2], Y, test_size=0.3, r
```

In [ ]:

```python
# creating a heatmap for the correlations
corrMatrix = data.corr()
plt.figure(figsize=(25,15))
sns.heatmap(corrMatrix, annot=True)
plt.show()
```



In [ ]:

```python
data.corr()
```

Out[ ]:

	popularity	duration_ms	explicit	danceability	energy	loudness	speechiness	ac
popularity	1.000000	-0.147147	0.155392	0.193814	-0.102099	0.170810	0.061784	
duration_ms	-0.147147	1.000000	-0.082648	-0.121470	0.033802	0.013959	-0.104894	
explicit	0.155392	-0.082648	1.000000	0.290529	-0.005351	0.022977	0.379301	
danceability	0.193814	-0.121470	0.290529	1.000000	0.134523	0.257602	0.244818	
energy	-0.102099	0.033802	-0.005351	0.134523	1.000000	0.688572	0.044134	
loudness	0.170810	0.013959	0.022977	0.257602	0.688572	1.000000	0.001977	
speechiness	0.061784	-0.104894	0.379301	0.244818	0.044134	0.001977	1.000000	
acousticness	0.117704	-0.021703	-0.061392	-0.192691	-0.613762	-0.421470	-0.048135	
instrumentalness	-0.417007	0.102895	-0.122278	-0.194143	0.014575	-0.330859	-0.134671	
liveness	-0.107463	0.036597	-0.061529	-0.180732	0.166020	0.043427	0.012082	
valence	0.113404	-0.109888	0.005151	0.398079	0.322794	0.322223	0.076226	

	popularity	duration_ms	explicit	danceability	energy	loudness	speechiness	ac
tempo	-0.023634	0.016473	-0.003159	-0.036500	0.180079	0.160547	0.053573	
time_signature_1	-0.004033	-0.000718	0.008255	-0.066895	-0.042177	-0.049175	0.030314	
time_signature_3	-0.002260	-0.000432	-0.053603	-0.180319	-0.153828	-0.141275	-0.022922	
time_signature_4	-0.001143	0.025068	0.039658	0.243833	0.180346	0.183509	-0.008002	
time_signature_5	0.007348	-0.033851	0.015348	-0.078271	-0.043518	-0.041434	0.060417	
key_1	0.013852	0.005349	0.047144	0.030226	0.019886	0.002667	0.058946	
key_2	-0.031502	-0.001336	0.003683	-0.021147	-0.021495	-0.007752	-0.019727	
key_3	0.025115	-0.009369	-0.031289	-0.017705	-0.046901	-0.016813	-0.032032	
key_4	-0.025322	0.004886	-0.019780	-0.034569	-0.008371	-0.024841	-0.029103	
key_5	0.011992	-0.006388	0.003208	-0.011441	0.000927	0.014449	-0.006737	
key_6	0.012256	-0.009624	0.002750	0.035256	0.016766	0.015631	0.012810	
key_7	-0.031497	0.026924	-0.011442	-0.028429	0.016864	-0.006703	-0.012215	
key_8	0.039321	-0.014791	0.022509	0.030615	0.003263	0.013043	0.031864	
key_9	-0.020950	0.010972	-0.024641	-0.002629	-0.005985	-0.005173	-0.008661	
key_10	0.007592	-0.003051	0.036118	0.011992	-0.004840	0.001860	0.018920	
key_11	0.008489	-0.011725	-0.003006	0.024000	0.046571	0.026409	0.021988	
mode_1	-0.009488	0.040919	-0.083026	-0.092353	-0.088441	-0.041964	-0.084880	

Setting Up the Models

In [ ]:

```
X = data[features]
Y = data['popularity']
Y_temp = Y
```

In [ ]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=8


model = LinearRegression()
model.fit(X_train,Y_train)
model.score(X_test, Y_test)
```

Out[ ]:

```
0.2265451890655975
```

In [ ]:

```
Y_pred = model.predict(X_test)
```

In [ ]:

```
from sklearn import metrics
from sklearn.metrics import r2_score
```

In [ ]:

```
print("R2 Score: ",metrics.r2_score(Y_test, Y_pred))
print("MAE: ", metrics.mean_absolute_error(Y_test, Y_pred))
print("MSE: ", metrics.mean_squared_error(Y_test, Y_pred))
print("RMSE:", np.sqrt(metrics.mean_squared_error(Y_test, Y_pred)))
```

```
R2 Score:  0.2265451890655975
MAE:  13.774146702743273
MSE:  333.337245311379
RMSE: 18.257525717122213
```

In [ ]:

```
from sklearn.model_selection import cross_val_score
from sklearn import datasets, linear_model

scores = cross_val_score(model,X,Y, cv=5)
print("Print all scores: ", scores)
print("Mean Accuracy: ", scores.mean())
```

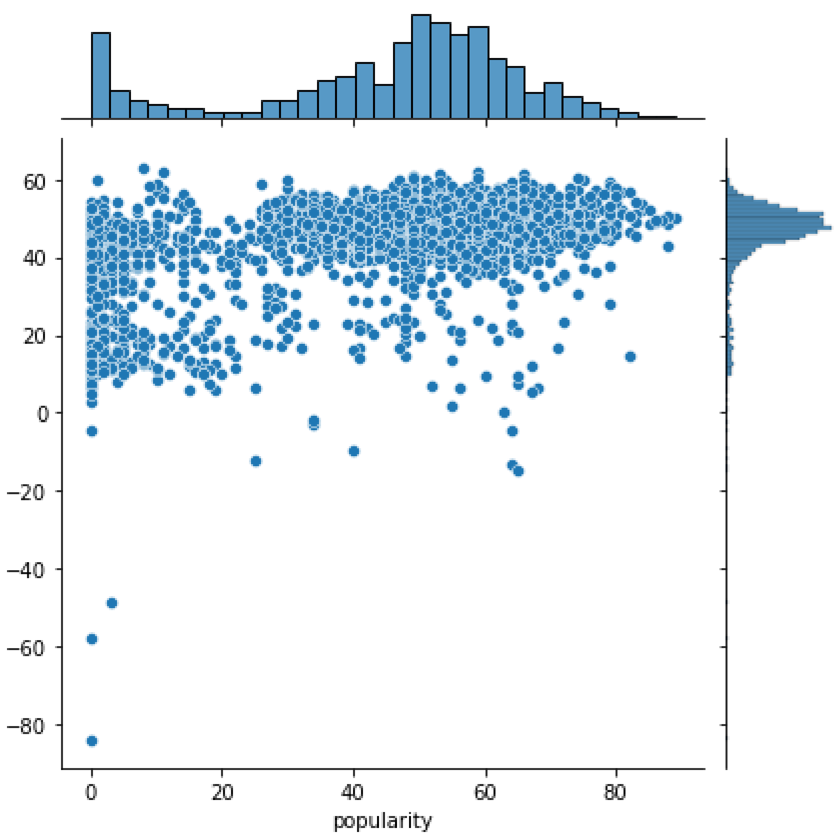
```
Print all scores:  [0.2105213  0.17116564 0.09005013 0.15640372 0.08064842]
Mean Accuracy:  0.1417578413143356
```

In [ ]:

```
sns.jointplot(Y_test,Y_pred)
```

Out[ ]:

```
<seaborn.axisgrid.JointGrid at 0x21ed07857c0>
```



Decision Tree Regressor

In [ ]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC, LinearSVC

from sklearn.metrics import make_scorer, accuracy_score, roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
```

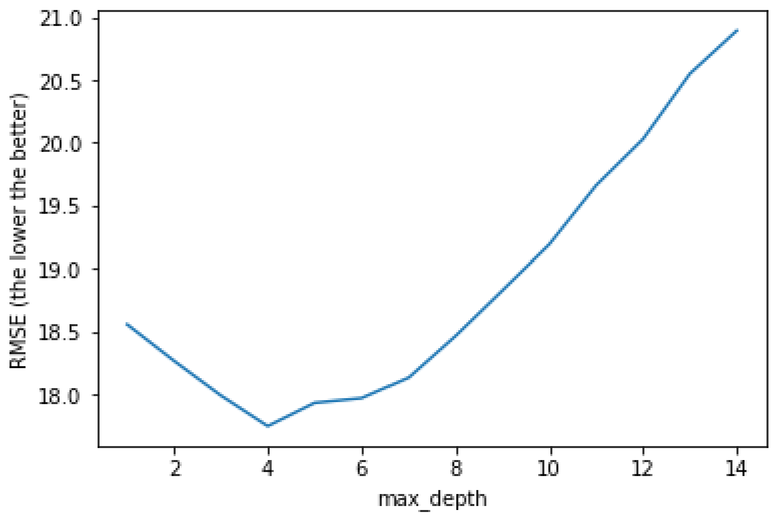
In [ ]:

```python
from sklearn.tree import DecisionTreeRegressor

maxdr = range(1, 15)

RMSE_scores = []

from sklearn.model_selection import cross_val_score
for depth in maxdr:
    treeregular = DecisionTreeRegressor(max_depth=depth, random_state=1)
    MSE_scores = cross_val_score(treeregular, X, Y, cv=5, scoring='neg_mean_squared_err
    RMSE_scores.append(np.mean(np.sqrt(-MSE_scores)))
    print(mean(RMSE_scores))
```

```
18.5560053914302
18.410969716788248
18.27054365907453
18.139235994015117
18.097540649719626
18.076014471400196
18.084031186852126
18.131574985661896
18.208805820761583
18.307130097467457
18.430280845277522
18.563633289248468
18.716521191965725
18.87206091149817
```

In [ ]:

```python
plt.plot(maxdr, RMSE_scores);
plt.xlabel('max_depth');
plt.ylabel('RMSE (the lower the better)');
```



In [ ]:
```
sorted(zip(RMSE_scores, maxdr))[0]
```

Out[ ]:
```
(17.745312998836887, 4)
```

In [ ]:
```
treeregular = DecisionTreeRegressor(max_depth=10, random_state=1)
treeregular.fit(X, Y)
```

Out[ ]:
```
DecisionTreeRegressor(max_depth=10, random_state=1)
```

In [ ]:
```
pd.DataFrame({'feature':features, 'importance':treeregular.feature_importances_}).sort_
```

Out[ ]:

	feature	importance
4	instrumentalness	0.401280
2	duration_ms	0.140301
0	acousticness	0.115545
6	loudness	0.071760
1	danceability	0.063537
8	valence	0.058355
3	energy	0.056643
7	tempo	0.054504
5	liveness	0.038076

Random Forest Regression

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RepeatedStratifiedKFold
```

In [ ]:

```
rf = RandomForestRegressor(n_estimators=150,
                            max_features=5,
                            oob_score=True,
                            random_state=1)
rf.fit(X, Y)
```

Out[ ]:
```
RandomForestRegressor(max_features=5, n_estimators=150, oob_score=True,
                      random_state=1)
```

In [ ]:
```
pd.DataFrame({'feature':features,
              'importance':rf.feature_importances_}).sort_values(by='importance', ascen
```

Out[ ]:

	feature	importance
4	instrumentalness	0.219625
0	acousticness	0.131723
2	duration_ms	0.122471
6	loudness	0.099546
3	energy	0.093128
1	danceability	0.088225
8	valence	0.083528
7	tempo	0.081956
5	liveness	0.079798

In [ ]:
```
print("OOB Score: ", (rf.oob_score_))
```

```
OOB Score:  0.3644964249932898
```

In [ ]:
```
#evaluate the model
#cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
#rf_scores = cross_val_score(rf, X, Y, scoring='accuracy', cv=cv, n_jobs=-1, error_scor
#print('Accuracy: %.3f (%.3f)' % (mean(rf_scores), std(rf_scores)))
```

In [ ]:
```
scores = cross_val_score(rf, X, Y, cv=5, scoring='neg_mean_squared_error')
print("RMSE:")
np.mean(np.sqrt(-scores))
```

```
RMSE:
```

Out[ ]:
```
16.893829732016442
```

Classification Learning Models

KNN

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [ ]:
```
pd.cut(data['popularity'], bins=3)
```
Out[ ]:
```
0          (62.667, 94.0]
1         (-0.094, 31.333]
2         (-0.094, 31.333]
3         (-0.094, 31.333]
4         (-0.094, 31.333]
                ...
10944     (31.333, 62.667]
10945     (31.333, 62.667]
10946     (31.333, 62.667]
10947     (31.333, 62.667]
10948       (62.667, 94.0]
Name: popularity, Length: 10949, dtype: category
Categories (3, interval[float64, right]): [(-0.094, 31.333] < (31.333, 62.667] < (62.66
7, 94.0]]
```
In [ ]:
```
from sklearn import preprocessing
```
In [ ]:
```
pd.cut(data['popularity'], bins=3, labels = ["Low", "Medium", "High"]).value_counts() #
```
Out[ ]:
```
Medium    6936
Low       2145
High      1868
Name: popularity, dtype: int64
```
In [ ]:
```
#data.head(5)
```
In [ ]:
```
data['popularity'] = pd.cut(data.popularity, bins=3, labels = ["low", "medium", "high"]
```
In [ ]:
```
data.loc[data['popularity']== 'high']
```
Out[ ]:

	id	name	popularity	duration_ms	explicit	artists	
0	4zP7ADsgJgHGY6VzxbNp1z	Year 3000	high	201960	0	['Jonas Brothers']	['7gOdH
336	60ynsPSSKe6O3sfwRnIBRf	Streets	high	226987	1	['Doja Cat']	['5cj0ILj
337	6UelLqGlWMcVH1E5c4H7lY	Watermelon Sugar	high	174000	0	['Harry Styles']	['6KImCVD
339	7qEHsqek33rTcFNT9PFqLf	Someone You Loved	high	182161	0	['Lewis Capaldi']	['4GNC7GD6
340	21jGcNKet2qwijlDFuPiPb	Circles	high	215280	0	['Post Malone']	['246d
...	...	...	...	...	...	...	
10841	4P1EGoXLWQ1YF6Nsmr1pfy	You And I	high	224064	0	['LÃ ON']	['4SqTiw(

	id	name	popularity	duration_ms	explicit	artists	
10871	0HZgYFimoJG9Ijy8InUWcV	Bad Reputation (feat. Joe Janiak)	high	205417	0	['Avicii', 'Joe Janiak']	['1vCWHaC '142T
10884	2K8elWg8ihrZRwZJ7Gy6L3	Come Home To Me	high	225868	0	['LÃ ON']	['4SqTiwC
10913	6EBlOYNcZ8MrdEov9lEdV6	Hold The Line (feat. A R I Z O N A)	high	171786	0	['Avicii', 'A R I Z O N A']	['1vCWHaC '7hOGh
10948	44r4zta6P9flkhKaVnbsvG	Freaks	high	174800	0	['Jordan Clarke']	['14Y3

1868 rows × 33 columns

In [ ]:

```
y = data.popularity
y.value_counts()/y.count()
```

Out[ ]:

```
medium    0.633483
low       0.195908
high      0.170609
Name: popularity, dtype: float64
```

In [ ]:

```
pop_count = data.popularity.value_counts()
```

START HERE- Notes: Split, Oversample, 4 bins

In [ ]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=.3, random_state=7)
```

In [ ]:

```
#pip install -U imbalanced-learn

from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import RandomOverSampler

ro = RandomOverSampler()
X_ro, y_ro = ro.fit_resample(X,Y)

print('Removed indexes:')

print(X_ro.shape, y_ro.shape)
```

```
Removed indexes:
(44268, 9) (44268,)
```

In [ ]:

```
y_ro.value_counts()
```

```
67      476
```

Out[ ]:
```
73    476
46    476
59    476
44    476
     ...
17    476
29    476
51    476
20    476
26    476
Name: popularity, Length: 93, dtype: int64
```

In [ ]:
```
X_ro.value_counts()
```

Out[ ]:
```
acousticness  danceability  duration_ms  energy  instrumentalness  liveness  loudness  t
empo    valence
0.2080        0.749         226987       0.463   0.037100          0.3370    -8.433    9
0.028   0.190       476
0.1220        0.548         174000       0.816   0.000000          0.3350    -4.209    9
5.390   0.557       476
0.7510        0.501         182161       0.405   0.000000          0.1050    -5.679    1
09.891  0.446       476
0.8180        0.450         183624       0.329   0.001090          0.1350    -12.603   7
1.884   0.266       254
0.1920        0.695         215280       0.762   0.002440          0.0863    -3.497    1
20.042  0.553       222

...
0.2200        0.531         203520       0.616   0.000001          0.3660    -5.507    1
24.953  0.255         1
0.4890        0.759         243693       0.501   0.880000          0.1520    -12.510   1
18.009  0.446         1
0.0859        0.430         169193       0.960   0.000000          0.0617    -2.721    1
51.920  0.454         1
0.0224        0.612         158041       0.539   0.000004          0.0790    -8.411    1
24.660  0.787         1
0.0334        0.498         192424       0.720   0.000000          0.1080    -6.304    8
8.891   0.148         1
Length: 10859, dtype: int64
```

In [ ]:
```
knn = KNeighborsClassifier(n_neighbors=1)

knn.fit(X_ro, y_ro)
```

Out[ ]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [ ]:
```
y_pred_knn = knn.predict(X_ro)

print("Mean Accuracy Score: ", metrics.accuracy_score(y_ro, y_pred_knn))
```

```
Mean Accuracy Score:  0.99268094334508
```

RANDOM FOREST CLASSIFIER

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier

# METHOD 1 - RF
```

```python
X = data[features]
Y = data['popularity']
# before = 0.7165905631659056 - est = 1000
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
clf = RandomForestClassifier(max_depth=5, random_state=None, n_estimators=1000)
clf.fit(X_train, Y_train)
y_pred=clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
Accuracy: 0.693455098934551
```

In [ ]:

```python
# METHOD 2 - RF

# define the model
model = RandomForestClassifier()

# fit the model
model.fit(X_train,Y_train)

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, Y, scoring='accuracy', cv=cv, n_jobs=-1, error_sco
# report performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), np.std(n_scores)))
```

```
Mean Accuracy: 0.717 (0.008)
```

In [ ]:

```python
from sklearn.model_selection import GridSearchCV
C_list = [0.001, 0.01, 0.1, 1, 10, 100, 1000]
Gamma_list = [0.001, 0.01, 0.1, 1, 10, 100, 1000]



gscv = GridSearchCV(model, scoring='accuracy', cv =10,
    param_grid={
        'max_depth' : [4, 5, 6, 7, 8, 9, 10],
        'max_features': ['sqrt', 'auto'],
        'min_samples_split': [2, 0.3, 0.5],
        'min_samples_leaf': [1, 0.3, 0.5]
    })
```

In [ ]:

```python
gscv.fit(X_train, Y_train)
```

Out[ ]:

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(),
             param_grid={'max_depth': [4, 5, 6, 7, 8, 9, 10],
                         'max_features': ['sqrt', 'auto'],
                         'min_samples_leaf': [1, 0.3, 0.5],
                         'min_samples_split': [2, 0.3, 0.5]},
             scoring='accuracy')
```

In [ ]:

```python
result = pd.DataFrame(gscv.cv_results_)
result.head()
```

Out[ ]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_features

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	param_max_features
0	1.849487	0.917693	0.049675	0.041452	4	sqrt
1	1.003463	0.053198	0.025807	0.003304	4	sqrt
2	1.005499	0.075230	0.028431	0.004130	4	sqrt
3	0.570722	0.029344	0.027196	0.004258	4	sqrt
4	0.637451	0.074299	0.029521	0.004219	4	sqrt

In [ ]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

pred = model.predict(X_test)
pred_prob = model.predict_proba(X_test)


# roc curve for the classes

fpr = {}
tpr = {}
thresh = {}

n_class = 3

for i in range(n_class):
    fpr[i], tpr[i], thresh[i] = roc_curve(Y_test, pred_prob[:,i], pos_label=i)
```

In [ ]:

```
pred_prob
```

Out[ ]:

```
array([[0.26, 0.06, 0.68],
       [0.05, 0.43, 0.52],
       [0.26, 0.11, 0.63],
       ...,
       [0.05, 0.29, 0.66],
       [0.17, 0.09, 0.74],
       [0.48, 0.08, 0.44]])
```

In [ ]:

```
J = fpr[i] - tpr[i]
```

```python
ix = np.argmin(J)
best_thresh = thresh[i][ix]
avg_thresh = mean(thresh[i])

print('Average Threshold = %f' % mean(thresh[i]))

print('Best Threshold = %f' % (best_thresh))
```

```
Average Threshold = 0.563217
Best Threshold = 1.980000
```

PREDICTIONS WITH AVERAGE

In [ ]:

```python
# Defining our variables
X = data[features]
Y = data['popularity']
trackNames = data['name']
```

In [ ]:

```python
def PopAssign (popularity, bestThreshold):
    values = pd.DataFrame(columns=['popularity'])
    for i in range(len(popularity)):
        if (max(popularity[i])) >= bestThreshold:
            #print((popularity[i]))
            values = values.append({'popularity':1, 'name': trackNames[i]} , ignore_ind
        else:
            values = values.append({'popularity':0, 'name': trackNames[i]}, ignore_inde
    return values
```

In [ ]:

```python
testing_thresh = 0.7
```

In [ ]:

```python
np.count_nonzero(pred_prob)
```

Out[ ]:

```
9763
```

In [ ]:

```python
# shape 3285 and 3 col

p = PopAssign(pred_prob, testing_thresh )
```

In [ ]:

```python
p['popularity'].value_counts()
```

Out[ ]:

```
1    1744
0    1541
Name: popularity, dtype: int64
```

In [ ]:

```python
# sending dataframe to CSV

p.to_csv("Popularity_Assignments4.csv")
```

DECISION TREE CLASSIFIER

In [ ]:

```
from sklearn.tree import DecisionTreeClassifier

# Defining our variables
X = data[features]
Y = data['popularity']

# Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
```

In [ ]:

```
# Define model
dt = DecisionTreeClassifier()

# evaluate the model
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
n_scores = cross_val_score(model, X, Y, scoring='accuracy', cv=5, n_jobs=-1, error_scor
# report performance
print('Mean Accuracy: %.3f (%.3f)' % (mean(n_scores), np.std(n_scores)))
```

```
Mean Accuracy: 0.697 (0.029)
```

In [ ]:

```
# TP - song listed on our prediction AND our top50 dataset
# FP - song listed on our prediction AND NOT on top50
# TN - song not listed on our prediction AND NOT on top50
# FN - song not listed on our prediction AND appears on top50
```

In [ ]:

```
```