ja59ydxvk

February 8, 2024

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
df = pd.read_csv("data/african_tracks.csv")
```

[3]:
```
df.shape
```

[3]:
```
(20959, 20)
```

This study focuses on the top 7 African music industries with the primary aim of conducting a nuanced and detailed analysis of regions that wield significant influence within the continent's musical landscape. The selected countries, consistently identified as key players by reputable sources, have been chosen strategically to align with the objectives of our machine learning analysis. The goal of the machine learning model is to predict the popularity of songs and understand the factors contributing to their popularity.

In the realm of machine learning, the inclusion of too many diverse and potentially noisy datasets could hinder the precision and interpretability of the model. By narrowing our focus to the top 7 African music industries, we seek to streamline the dataset to include only the most influential and impactful regions. This deliberate selection enhances the model's ability to discern patterns, trends, and features relevant to predicting the popularity of songs.

The criteria employed to identify the top music industries encompassed key factors like market size, cultural influence, export/import dynamics of musical content, and the overall impact on the global music landscape. Two independent articles (https://www.boomplay.com/buzz/3520053 and https://www.schooldrillers.com/biggest-music-industry-in-africa/) were reviewed, ensuring reliability and consistency. This dual-source analysis produced a harmonious list, confirming the top 7 African music industries. Notably, the sources shared a uniform methodology, further enhancing the credibility of the selected regions. The countries selected includes **Nigeria, South Africa, Ghana, Kenya, Tanzania, DR Congo**, and **Benin Republic**.

We exclusively examine songs by *top* and *popular* artists hailing from the selected countries (using google search), and intriguingly, the artists listed in Forbes list of the 20 biggest African artists in 2022 are from the countries selected. Note that the term *top* and *popular* maybe subjective.

[4]:
```
# Top African artist according to forbes:
# https://www.forbesafrica.com/cover-story/2022/08/19/
↪the-playlist-africas-top-20-musicians/
forbes = ['Angelique Kidjo', 'Burna Boy', 'Tiwa Savage', 'Davido',
```

```
          'Wizkid', 'Master KG', 'Major League Djz', 'Diamond Platnumz',
          'Nasty C', 'Mr Eazi', 'Lebo M.', 'Black Coffee', '2Baba',
          'Cassper Nyovest', 'Yvonne Chaka Chaka', 'KDDO', 'Rayvanny',
          'Fally Ipupa', 'DJ Maphorisa', 'Lira'
         ]

# Biggest Music Industries In Africa:
# https://www.boomplay.com/buzz/3520053
# https://www.schooldrillers.com/biggest-music-industry-in-africa/

NGA = ["Burna Boy", "Davido", "Wizkid", "Olamide", "Tiwa Savage", "Fireboy DML",
       "Joeboy", "Rema", "Patoranking", "Tekno", "Mr Eazi", "Falz", "Blaqbonez",
       "Adekunle Gold", "Mayorkun", "Oxlade", "Peruzzi", "Tems", "Naira Marley",
       "Simi", "Ajebo Hustlers", "Bella Shmurda", "Ruger", "Bnxn", "Terri",␣
 ↪"Fela Kuti",
       "Mohbad", "Asake", "CKay", "Victony", "Omah Lay", "Zinoleesky", "Lyta",
      ]


GHA = ['Sarkodie', 'Shatta Wale', 'Stonebwoy', 'KiDi', 'Black Sherif',
       'Gyakie', 'Amerado', 'Kwesi Arthur', 'Kofi Kinaata', 'Efya',
       'Adina Thembi', 'Medikal', 'Wendy Shay', 'King Promise', 'Becca',
       'MzVee', 'Kelvyn Boy', 'Cina Soul', 'DarkoVibes', 'Joey B',
       'Kuami Eugene', 'Camidoh', 'Fameye', 'Akwaboah', 'Mzbel',
       'R2Bees', 'Guru', 'A.B. Crentsil', 'Daddy Lumba', 'Castro',
      ]

ZAF = ["Nasty C", "DJ Maphorisa", "Kabza De Small", "Sho Madjozi", "Blxckie",
       "Busiswa", "Shekhinah", "YoungstaCPT", "Kwesta", "Black Motion","Mi␣
 ↪Casa",
       "Moonchild Sanelly", "Msaki", "Locnville", "Die Antwoord", "TRESOR",
       "Berita", "The Soil", "Mafikizolo", "Brenda Fassie", "Johnny Clegg",
       "Thandiswa", "Hugh Masekela", "Miriam Makeba", "Lucky Dube", "Lady␣
 ↪Zamar",
       "Black Coffee", 'Cassper Nyovest', 'AKA', 'Sho Madjozi', 'Prince␣
 ↪Kaybee', "ANATII"
      ]

KEN = ["Sauti Sol", "Nyashinski", "Khaligraph Jones", "ETHIC",
       "Nikita Kering'", "Rekles", "Mr Seed", "Masauti", "Ethic Entertainment",
       "Willy Paul", "Akothee", "Avril", "Kagwe Mungai", "Sanaipei Tande",
       "Fena Gitu", "Mejja", "Eko Dydda", "Teddy Afro", "MOG",
       'Nameless', 'Victoria Kimani', "Kristoff",
      ]

TZA = ["Diamond Platnumz", "Nandy", "Harmonize", "Rayvanny", "Zuchu",
       "Alikiba", "Marioo", "Baba Levo", "B-Boy", " Mr Nice",
```

```
       "Mzee Bwax", "Queen Darleen", "Dulla Makabila", "Chege Chege",
       "Ben Pol", "Alikiba", "Linah Sanga",
       "Nikki Mbishi", "Afande Sele", "Rosa Ree",
      ]

DRC = ["Papa Wemba", "Fally Ipupa", "Yxng Bane", "Koffi Olomide", "Werrason",
       "JB Mpiana", "Dadju", "Luciana de Paula", "Gims", "Atele", "Koffi␣
 ↪Olomide",
       "Mbilia Bel", "Celeo Scram", "Ferre Gola", "Deplick Pomba", "Werrason",␣
 ↪'Awilo Logomba',
       "Cindy Le Coeur", "Robinio Mundibu", "Fabregas le Métis Noir", "Barbara␣
 ↪Kanam"
      ]

BEN = ["Gangbé Brass Band", "T.P. Orchestre Poly-Rythmo", "Gnonnas Pedro",
       "Gabo Brown", "Lokonon Andre", "Les Volcans", "Tcheba",
       "Angelique Kidjo", "Sessimè", "Adje", "Virgul",
      ]

all_artists = list(set(forbes + NGA + GHA + ZAF  + KEN + TZA + DRC + BEN))
len(all_artists)
```

[4]: 172

[5]:
```
df = df[df['artist_name'].isin(all_artists)]
df.reset_index(drop=True, inplace=True)
```

[6]:
```
len(df)
```

[6]: 9130

[7]:
```
#looking at the stats of different columns
df.describe()
```

[7]:
```
        duration_ms   popularity  danceability          key  acousticness  \
count  9.130000e+03  9130.000000   9130.000000  9130.000000   9130.000000
mean   2.838526e+05    17.403286      0.659500     5.303724      0.338132
std    1.686987e+05    15.286461      0.142996     3.682027      0.276288
min    4.937000e+03     0.000000      0.000000     0.000000      0.000012
25%    1.940000e+05     4.000000      0.555000     2.000000      0.092300
50%    2.478065e+05    14.000000      0.676000     6.000000      0.276000
75%    3.423890e+05    27.000000      0.770000     9.000000      0.547000
max    4.851037e+06    81.000000      0.985000    11.000000      0.994000

              mode       energy  instrumentalness     liveness     loudness  \
count  9130.000000  9130.000000       9130.000000  9130.000000  9130.000000
mean      0.615991     0.670846          0.064456     0.203249    -7.850288
```

```
std        0.486387    0.187392         0.191063    0.183564     3.484919
min        0.000000    0.000101         0.000000    0.000000   -34.996000
25%        0.000000    0.562000         0.000000    0.090425    -9.615750
50%        1.000000    0.699000         0.000012    0.126000    -7.262500
75%        1.000000    0.812000         0.003120    0.262000    -5.429500
max        1.000000    0.999000         0.998000    0.989000     1.231000

       speechiness        tempo  time_signature      valence
count  9130.000000  9130.000000     9130.000000  9130.000000
mean      0.129323   117.854094        3.953450     0.659412
std       0.126092    25.365729        0.401718     0.223417
min       0.000000     0.000000        0.000000     0.000000
25%       0.046900   100.988250        4.000000     0.510000
50%       0.074950   115.979000        4.000000     0.704000
75%       0.170000   129.160000        4.000000     0.842000
max       0.962000   230.186000        5.000000     0.997000
```

[8]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9130 entries, 0 to 9129
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   track_name        9130 non-null   object
 1   track_id          9130 non-null   object
 2   genre             8826 non-null   object
 3   album_name        9130 non-null   object
 4   artist_name       9130 non-null   object
 5   release_date      9130 non-null   object
 6   duration_ms       9130 non-null   float64
 7   popularity        9130 non-null   float64
 8   danceability      9130 non-null   float64
 9   key               9130 non-null   float64
 10  acousticness      9130 non-null   float64
 11  mode              9130 non-null   float64
 12  energy            9130 non-null   float64
 13  instrumentalness  9130 non-null   float64
 14  liveness          9130 non-null   float64
 15  loudness          9130 non-null   float64
 16  speechiness       9130 non-null   float64
 17  tempo             9130 non-null   float64
 18  time_signature    9130 non-null   float64
 19  valence           9130 non-null   float64
dtypes: float64(14), object(6)
memory usage: 1.4+ MB
```

We once again see that we have 9130 tracks in the dataset with both categorical and numerical columns. In order to use the information from the categorical columns (`genre`, `artist_name`, `track_name`, `album_name`, `track_id`, `key`, `mode`, `time_signature`) we will either need to represent them numerically by feature engineering or drop them to be able to train the models.

[9]:
```
#looking at different values contained within columns
for col in df.columns:
    print(f"Column: {col}")
    print(df[col].value_counts())
    print("--------------------")
```

```
Column: track_name
Bandana                  1
Ba Gerants Ya Mabala     1
Bakwiti                  1
Vina                     1
Boya Yé                  1
                        ..
Yaka toluka mwana        1
Mudinda                  1
Kayile Inga              1
Donner et recevoir       1
Sugarcane - Remix        1
Name: track_name, Length: 9130, dtype: int64
--------------------
Column: track_id
2qWwuCVeMjF9mUT0S5Iqvl    1
2EneceW18tuSTNNWIYdERo    1
7H7fPHJCV3aURGBH640OCf    1
51Yl46yxbqTt4igAnJ438v    1
3z9a3Ml656ZaZfY8mIUkCj    1
                          ..
0UlmM1nJR2ZQyBTvVgZHuX    1
0WhZ1mNzS4d6YiWzDRmakD    1
0diwYljrfhJ0h1KVYtmplk    1
7kyNmVrWKtrpiTG7mXnuWr    1
6NuG2JgERZZXvvjmtjOFix    1
Name: track_id, Length: 9130, dtype: int64
--------------------
Column: genre
afropop,south african jazz,world,xhosa                          609
azontobeats,ndombolo,rumba congolaise,soukous,zilizopendwa     488
afropop,rumba congolaise,soukous,zilizopendwa                  419
afropop,jazz trumpet,kwaito,south african jazz                 303
azonto,hiplife                                                 272
                                                              …
south african pop                                                1
motown                                                           1
house argentino,organic electronic                               1
```

```
funk carioca,funk rj                                                      1
melodic house                                                             1
Name: genre, Length: 133, dtype: int64
--------------------
Column: album_name
Miriam Makeba (Five Original Albums)        66
The Healers: The Last Chapter               41
Highlife: Jazz and Afro- Soul (1963-1969)   39
13ième apôtre                               38
Answers (The Hybrid)                        33
                                            ..
Goodbye to Africa                            1
Pieces Of Me (Platinum Mixed Edition)        1
Playing at Work (Re-Worked)                  1
Emotion (25th Anniversary Edition)           1
The Many Voices of Miriam Makeba             1
Name: album_name, Length: 866, dtype: int64
--------------------
Column: artist_name
Miriam Makeba      622
Koffi Olomide      497
Papa Wemba         432
Hugh Masekela      311
Lucky Dube         268
                  …
Victony              6
Nikita Kering'       6
Tems                 6
Robinio Mundibu      5
Lyta                 5
Name: artist_name, Length: 130, dtype: int64
--------------------
Column: release_date
2014-01-01    173
2009-01-01     87
2011-01-01     84
2013-01-01     78
2008-01-01     68
             …
2012-06-01      1
2014-09-08      1
1963-11-26      1
2011-11-08      1
2014-11-16      1
Name: release_date, Length: 639, dtype: int64
--------------------
Column: duration_ms
240000.0    10
```

```
180000.0     9
216000.0     8
160000.0     8
190000.0     7
            ..
175254.0     1
206118.0     1
165201.0     1
188681.0     1
251147.0     1
Name: duration_ms, Length: 8264, dtype: int64
--------------------
Column: popularity
0.0     729
1.0     560
2.0     419
3.0     404
4.0     354
       …
81.0      2
73.0      2
75.0      2
69.0      2
74.0      1
Name: popularity, Length: 78, dtype: int64
--------------------
Column: danceability
0.759    38
0.809    35
0.712    35
0.728    35
0.707    34
         ..
0.311     1
0.945     1
0.341     1
0.282     1
0.960     1
Name: danceability, Length: 706, dtype: int64
--------------------
Column: key
0.0     1216
7.0     1005
1.0      942
9.0      867
2.0      816
11.0     789
5.0      746
```

```
10.0     728
6.0      638
8.0      558
4.0      555
3.0      270
Name: key, dtype: int64
--------------------
Column: acousticness
0.118000    29
0.159000    21
0.252000    21
0.106000    21
0.117000    20
            ..
0.092500     1
0.002480     1
0.014400     1
0.000962     1
0.088400     1
Name: acousticness, Length: 2127, dtype: int64
--------------------
Column: mode
1.0    5624
0.0    3506
Name: mode, dtype: int64
--------------------
Column: energy
0.7390    31
0.8330    30
0.8310    29
0.7960    29
0.6690    29
          ..
0.0865     1
0.1750     1
0.0477     1
0.0419     1
0.0801     1
Name: energy, Length: 922, dtype: int64
--------------------
Column: instrumentalness
0.000000    3461
0.000014       9
0.000013       9
0.104000       9
0.000107       9
            …
0.000007       1
```

```
0.000006       1
0.000003       1
0.000005       1
0.000004       1
Name: instrumentalness, Length: 3050, dtype: int64
--------------------
Column: liveness
0.1110    85
0.1030    80
0.1040    80
0.1080    75
0.1090    75
          ..
0.8830     1
0.8310     1
0.0404     1
0.0249     1
0.8350     1
Name: liveness, Length: 1447, dtype: int64
--------------------
Column: loudness
-5.556     7
-6.044     6
-8.069     6
-6.901     6
-8.984     6
          ..
-13.095    1
-6.656     1
-8.219     1
-7.027     1
-5.533     1
Name: loudness, Length: 6261, dtype: int64
--------------------
Column: speechiness
0.111    37
0.103    30
0.109    30
0.104    30
0.123    29
         ..
0.546     1
0.455     1
0.843     1
0.818     1
0.881     1
Name: speechiness, Length: 1245, dtype: int64
--------------------
```

```
Column: tempo
112.999    12
112.998    10
113.001    10
112.995     9
113.013     9
           ..
97.055      1
120.044     1
108.062     1
127.658     1
202.034     1
Name: tempo, Length: 7562, dtype: int64
--------------------
Column: time_signature
4.0    8164
3.0     575
5.0     333
1.0      49
0.0       9
Name: time_signature, dtype: int64
--------------------
Column: valence
0.9610    65
0.9620    46
0.9640    37
0.9650    35
0.9600    31
          ..
0.0673     1
0.0830     1
0.2430     1
0.0367     1
0.0948     1
Name: valence, Length: 946, dtype: int64
--------------------
```

0.0.1 Missing Values

[10]:
```
#checking for missing values
df.isna().sum()
```

[10]:
```
track_name           0
track_id             0
genre              304
album_name           0
artist_name          0
```

```
release_date         0
duration_ms          0
popularity           0
danceability         0
key                  0
acousticness         0
mode                 0
energy               0
instrumentalness     0
liveness             0
loudness             0
speechiness          0
tempo                0
time_signature       0
valence              0
dtype: int64
```

We have 304 missing values in the `'genre'` column

[11]:
```
df[df['genre'].isna()]
```

[11]:
```
                                              track_name  \
126                                                 Dada
588                                            Par amour
766                                      Afro Beat Blues
769                                                Joala
772                                          Za Labalaba
…                                                      …
7794  Présentation des fioti-fioti par Rouf Mbuta Ng…
7867                                               Allah
7869                                    Kuiti ya bolingo
8578        Inyakanyaka (feat. S.C Gorna & Khandu Cash)
8582                                              Umgido

                    track_id genre  \
126   7gOiZ1yDVv3teExIKt6O5c   NaN
588   0XG1u0KC21G7qFlY0LAFt4   NaN
766   4xclRUqjOM5HMzDZQyRaPo   NaN
769   2ZFywHbfQDiTLJLzk5wj9U   NaN
772   4iw3PchnWTNJFaqeEFVsf1   NaN
…                        …   …
7794  594Q8xWo0spLm5wEdIhdOF   NaN
7867  5PwcufFyTYhOhVLDFMPSzG   NaN
7869  5VKCl6fSxBxkxEQvshQI70   NaN
8578  3WuQZRMIWXH6yY2A5d4xfs   NaN
8582  0aJ1Ql3XCBoGiQot9GZTFw   NaN
```

	album_name	artist_name
126	Karibu	Barbara Kanam
588	Techno malewa sans cesse, Vol. 1	Werrason
766	The Chisa Years 1965-1975 (Rare and Unreleased)	Hugh Masekela
769	The Chisa Years 1965-1975 (Rare and Unreleased)	Hugh Masekela
772	The Chisa Years 1965-1975 (Rare and Unreleased)	Hugh Masekela
…	…	…
7794	Le zénith de papa wemba, vol. 1 (Esprit de fêtes)	Papa Wemba
7867	Merveilles du passé (1977-1985)	Papa Wemba
7869	Merveilles du passé (1977-1985)	Papa Wemba
8578	Blaqboy Music Presents Gqom Wave	DJ Maphorisa
8582	Blaqboy Music Presents Gqom Wave	DJ Maphorisa

	release_date	duration_ms	popularity	danceability	key	acousticness
126	2009	168253.0	5.0	0.572	0.0	0.870000
588	2009-01-01	531773.0	21.0	0.570	7.0	0.275000
766	2006-03-13	408106.0	44.0	0.776	10.0	0.434000
769	2006-03-13	122946.0	17.0	0.511	5.0	0.696000
772	2006-03-13	187160.0	13.0	0.649	11.0	0.560000
…	…	…	…	…	…	…
7794	1999-12-17	6025.0	0.0	0.000	2.0	0.787000
7867	1997-04-21	408986.0	1.0	0.379	0.0	0.243000
7869	1997-04-21	477746.0	0.0	0.621	11.0	0.762000
8578	2017-11-17	325320.0	10.0	0.809	9.0	0.000165
8582	2017-11-17	284560.0	6.0	0.798	11.0	0.002260

	mode	energy	instrumentalness	liveness	loudness	speechiness
126	0.0	0.691	0.000000	0.1110	-4.967	0.0291
588	1.0	0.837	0.000000	0.1070	-4.625	0.3090
766	0.0	0.828	0.289000	0.1430	-7.076	0.0692
769	1.0	0.759	0.000000	0.1580	-6.865	0.0562
772	0.0	0.937	0.000000	0.4700	-6.949	0.0974
…	…	…	…	…	…	…
7794	0.0	0.620	0.000000	0.0000	-10.055	0.0000
7867	1.0	0.641	0.000364	0.1480	-9.953	0.1570
7869	1.0	0.849	0.106000	0.3320	-6.765	0.1160
8578	1.0	0.782	0.285000	0.0911	-9.443	0.0684
8582	1.0	0.707	0.876000	0.1100	-7.376	0.0505

	tempo	time_signature	valence
126	79.035	4.0	0.701
588	138.119	4.0	0.706
766	96.501	4.0	0.900
769	78.142	4.0	0.776
772	110.571	3.0	0.545
…	…	…	…
7794	0.000	0.0	0.000

```
7867   87.195             3.0    0.652
7869  121.112             4.0    0.844
8578  126.022             4.0    0.157
8582  125.012             4.0    0.158

[304 rows x 20 columns]
```

We shall drop all rows with missing values from the dataset

[12]:
```
df = df.dropna()
df.shape
```

[12]:
```
(8826, 20)
```

[13]:
```
df.isna().sum()
```

[13]:
```
track_name          0
track_id            0
genre               0
album_name          0
artist_name         0
release_date        0
duration_ms         0
popularity          0
danceability        0
key                 0
acousticness        0
mode                0
energy              0
instrumentalness    0
liveness            0
loudness            0
speechiness         0
tempo               0
time_signature      0
valence             0
dtype: int64
```

[14]:
```
# Check for duplicated tracks by using their unique id numbers.
df[df['track_id'].duplicated()]
```

[14]:
```
Empty DataFrame
Columns: [track_name, track_id, genre, album_name, artist_name, release_date,
duration_ms, popularity, danceability, key, acousticness, mode, energy,
instrumentalness, liveness, loudness, speechiness, tempo, time_signature,
valence]
Index: []
```

We do not have any duplicated track.

Multiple genres are associated with each track because the genres of the track is based on the genre which the artist belong. What makes the most sense in this case would be to create different columns with the genre names and display with binary values whether a song belongs to that genre or not. Before we do this, we need to address some few key issues.

First, we have both 'afrobeat' and 'afrobeats' listed as genres. Also 'azonto' and 'azontobeats' should be listed as same genre. To ensure consistency and accurate categorization, these terms should be treated as synonymous:

[15]:
```
import re
```

[16]:
```
# Check genres that contains afrobeat
pattern = fr'\bafrobeat\b'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[16]: 130

[17]:
```
# Check genres that contains afrobeats
pattern = fr'\bafrobeats\b'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[17]: 2020

We have 130 genres with 'afrobeat' (without 's') and 2020 genres with 'afrobeats' (with 's')

[18]:
```
# Replace all 'afrobeat' with 'afrobeats'
pattern = r'\bafrobeat\b'
df['genre'] = df['genre'].apply(lambda x: re.sub(pattern, 'afrobeats', x))
```

Recheck to see if the issue has been resolved

[19]:
```
pattern = r'\bafrobeat\b'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[19]: 0

[20]:
```
pattern = r'\bafrobeats\b'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[20]: 2150

After replacing all 'afrobeat' with 'afrobeats' we now have a total of 2150 afrobeats (which is the sum total of afrobeat with 's' and without 's'). We will do the same for 'azonto', 'azontobeat', and 'azontobeats' (with s)

[21]:
```python
pattern = r'(\bazonto\b)|(\bazontobeat\b)'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[21]: 859

[22]:
```python
pattern = r'\bazontobeats\b'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[22]: 1677

[23]:
```python
# Replace 'azonto' and 'azontobeat' with 'azontobeats'
pattern = r'(\bazonto\b)|(\bazontobeat\b)'
df['genre'] = df['genre'].apply(lambda x: re.sub(pattern, 'azontobeats', x))
```

[24]:
```python
pattern = r'(\bazonto\b)|(\bazontobeat\b)'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[24]: 0

[25]:
```python
pattern = r'\bazontobeats\b'
pattern = re.compile(pattern, flags=re.IGNORECASE)
df[df['genre'].apply(lambda x: bool(pattern.search(x)))].shape[0]
```

[25]: 2368

Secondly, in the `genre` column, we observe various subgenres, including 'south african pop', ghanian pop', nigerian pop' which all fall under the broader category of pop music. Similarly, 'south african hip hop', 'nigerian hip hop,' and 'christian hip hop' are subgenres falling within the hip hop music category. To streamline our machine learning process, we will group these subgenres together under their respective main genres for effective model training and classification.

[26]:
```python
def genres_from_string(series):
    all_genres = set()    # Remove duplicates
    genres = series.str.split(',')
    for item in genres:
        all_genres.update(item)
    return list(all_genres)
```

[27]:
```python
# generating a list with the genre names
genre_list = genres_from_string(df['genre'])
```

[28]:
```python
len(genre_list)
```

[28]: 93

[29]:
```
genre_list
```

[29]:
```
['kenyan pop',
 'nigerian pop',
 'brass band',
 'dancehall',
 'house argentino',
 'bongo flava',
 'south african trap',
 'minimal tech house',
 'funk carioca',
 'african rock',
 'old school highlife',
 'afro r&b',
 'afrikaans hip hop',
 'microhouse',
 'rumba congolaise',
 'tanzanian pop',
 'kwaito',
 'alte',
 'afrobeats',
 'r&b francais',
 'german house',
 'nigerian hip hop',
 'deep deep house',
 'south african soulful deep house',
 'afro soul',
 'bolobedu house',
 'barcadi',
 'french hip hop',
 'uk dancehall',
 'ghanaian alternative',
 'south african deep house',
 'sda a cappella',
 'melodic house',
 'afro house',
 'azontobeats',
 'xhosa hip hop',
 'funk rj',
 'beninese pop',
 'movie tunes',
 'cape town indie',
 'eritrean pop',
 'xhosa',
 'tanzanian hip hop',
 'ndombolo',
 'grime',
```

```
'minimal techno',
'south african choral',
'swiss house',
'dutch hip hop',
'jazz trumpet',
'ghanaian hip hop',
'pop urbaine',
'amharic pop',
'asakaa',
'nubian traditional',
'funky house',
'south african hip hop',
'african reggae',
'kenyan hip hop',
'motown',
'arab alternative',
'melodic techno',
'musique urbaine kinshasa',
'south african pop',
'christian afrobeats',
'afroswing',
'organic electronic',
'ethiopian pop',
'sudanese pop',
'south african alternative',
'uk hip hop',
'south african jazz',
'deep house',
'gengetone',
'ghanaian pop',
'south african house',
'swedish dancehall',
'kenyan r&b',
'israeli techno',
'world',
'south african pop dance',
'belgian techno',
'soukous',
'portuguese pop',
'hiplife',
'kasi rap',
'afropop',
'organic house',
'gqom',
'amapiano',
'zilizopendwa',
'downtempo',
```

```
 'xitsonga pop']
```

[30]:
```
main_genres = ['hip hop', 'pop', 'rock', 'rap', 'r&b', 'jazz', 'trap',
               'afrobeat', 'alternative', 'soul', 'blues', 'techno', 'amapiano',
               'reggae', 'highlife', 'house', 'dancehall', 'funk']
```

[31]:
```
new_genres = genre_list.copy()
```

[32]:
```
for genre in main_genres:
    pattern = fr'\b{genre}\b'
    pattern = re.compile(pattern, flags=re.IGNORECASE)
    for i, sub_genre in enumerate(new_genres):
        if pattern.search(sub_genre):
            new_genres[i] = genre
```

The code above turns every subgenres in new_genres into its main genres

[33]:
```
genre_list[:8]
```

[33]:
```
['kenyan pop',
 'nigerian pop',
 'brass band',
 'dancehall',
 'house argentino',
 'bongo flava',
 'south african trap',
 'minimal tech house']
```

[34]:
```
new_genres[:8]
```

[34]:
```
['pop',
 'pop',
 'brass band',
 'dancehall',
 'house',
 'bongo flava',
 'trap',
 'house']
```

[35]:
```
# remove duplicates genres
new_genres = list(set(new_genres))
```

[36]:
```
len(new_genres)
```

[36]:
```
47
```

[37]:
```
new_genres
```

[37]:
```
['techno',
 'brass band',
 'dancehall',
 'sda a cappella',
 'bongo flava',
 'asakaa',
 'nubian traditional',
 'reggae',
 'trap',
 'hip hop',
 'world',
 'microhouse',
 'soukous',
 'rumba congolaise',
 'azontobeats',
 'motown',
 'hiplife',
 'kwaito',
 'jazz',
 'highlife',
 'rock',
 'alte',
 'musique urbaine kinshasa',
 'afrobeats',
 'christian afrobeats',
 'pop',
 'afroswing',
 'house',
 'rap',
 'afropop',
 'organic electronic',
 'movie tunes',
 'cape town indie',
 'xhosa',
 'r&b',
 'gqom',
 'funk',
 'ndombolo',
 'amapiano',
 'grime',
 'soul',
 'barcadi',
 'alternative',
 'gengetone',
 'south african choral',
 'zilizopendwa',
 'downtempo']
```

As we focus on popular music, we'll exclude genres that are either unpopular or infrequent (with a low count or appearance) in our dataset. This involves counting each genre and eliminating those that constitute less than 5 percent of the total dataset.

[38]:
```
genre_counts = {}
for genre in new_genres:
    pattern = re.compile(fr'\b{genre}\b')
    count = df['genre'].apply(lambda x: bool(pattern.search(x))).sum()
    genre_counts[genre] = count

genre_counts
```

[38]:
```
{'techno': 5,
 'brass band': 41,
 'dancehall': 285,
 'sda a cappella': 45,
 'bongo flava': 318,
 'asakaa': 97,
 'nubian traditional': 1,
 'reggae': 268,
 'trap': 307,
 'hip hop': 1997,
 'world': 950,
 'microhouse': 1,
 'soukous': 1303,
 'rumba congolaise': 1453,
 'azontobeats': 2368,
 'motown': 1,
 'hiplife': 475,
 'kwaito': 960,
 'jazz': 1239,
 'highlife': 55,
 'rock': 245,
 'alte': 114,
 'musique urbaine kinshasa': 294,
 'afrobeats': 2150,
 'christian afrobeats': 22,
 'pop': 4212,
 'afroswing': 38,
 'house': 482,
 'rap': 194,
 'afropop': 3123,
 'organic electronic': 3,
 'movie tunes': 8,
 'cape town indie': 148,
 'xhosa': 676,
 'r&b': 433,
```

```
 'gqom': 22,
 'funk': 1,
 'ndombolo': 876,
 'amapiano': 297,
 'grime': 24,
 'soul': 1051,
 'barcadi': 59,
 'alternative': 252,
 'gengetone': 182,
 'south african choral': 45,
 'zilizopendwa': 1032,
 'downtempo': 1}
```

[39]:
```
0.05*len(df)
```

[39]:
```
441.3
```

[40]:
```
new_genres = [genre for genre in genre_counts if genre_counts[genre] >= 0.
 ↪05*len(df)]
new_genres
```

[40]:
```
['hip hop',
 'world',
 'soukous',
 'rumba congolaise',
 'azontobeats',
 'hiplife',
 'kwaito',
 'jazz',
 'afrobeats',
 'pop',
 'house',
 'afropop',
 'xhosa',
 'ndombolo',
 'soul',
 'zilizopendwa']
```

To refine our dataset for analysis, genres were binarized, transforming them into distinct binary columns. This process involved assigning a '1' to indicate the presence of a genre and '0' for absence. Notably, only genres above 5%, determined based on their prevalence within the dataset, were retained for further investigation. This selective approach ensures that our analysis focuses on the most influential genres, allowing for a more concentrated examination of the predominant musical styles in our dataset.

[41]:
```
df
```

[41]:

```
                   track_name                track_id  \
0                     Bandana  2qWwuCVeMjF9mUTOS5Iqvl
1        All Of Us (Ashawo)  6459gZKddpOoPIH8PAcCwS
2                     Playboy  2gGAyatRqjjx3DOmLGI12W
3        Adore (feat. euro)  3ouP8HFixJmafK7hd1wJ0q
4                       Sofri  6S5XNauc7v8FLJWEIk0z2c
…                           …                       …
9125                 Odo Dede  5JB0EcpkbUsyaU9EvzK3bw
9126             Save My Soul  0dXCiV6LK9YkpBP5lbFiD4
9127                Decisions  2U5vPEm0m58dY8DCmKx1hr
9128                Sugarcane  2HfK1KumDffDWPZga46Hmw
9129       Sugarcane - Remix  6NuG2JgERZZXvvjmtjOFix

                                  genre                 album_name  artist_name  \
0               afrobeats,nigerian pop                     Playboy  Fireboy DML
1               afrobeats,nigerian pop                     Playboy  Fireboy DML
2                  azontobeats,hiplife                    Play Boy  Daddy Lumba
3               afrobeats,nigerian pop                     Playboy  Fireboy DML
4               afrobeats,nigerian pop                     Playboy  Fireboy DML
…                                     …                           …            …
9125  afro r&b,afrobeats,ghanaian pop  L.I.T.A (Deluxe Edition)      Camidoh
9126  afro r&b,afrobeats,ghanaian pop  L.I.T.A (Deluxe Edition)      Camidoh
9127  afro r&b,afrobeats,ghanaian pop                     L.I.T.A      Camidoh
9128  afro r&b,afrobeats,ghanaian pop                     L.I.T.A      Camidoh
9129  afro r&b,afrobeats,ghanaian pop                     L.I.T.A      Camidoh

     release_date  duration_ms  popularity  danceability   key  acousticness  \
0      2022-08-04     178225.0        73.0         0.818   1.0         0.293
1      2022-08-04     183349.0        62.0         0.605  11.0         0.304
2      1992-10-05     316440.0        16.0         0.732  11.0         0.225
3      2022-08-04     201826.0        42.0         0.709   0.0         0.108
4      2022-08-04     179246.0        47.0         0.745   6.0         0.341
…               …            …          …             …     …             …
9125   2023-06-23     236202.0        29.0         0.651   6.0         0.112
9126   2023-06-23     139080.0        13.0         0.529   7.0         0.672
9127   2023-06-02     197041.0        25.0         0.835   4.0         0.466
9128   2023-06-02     156781.0        56.0         0.519   8.0         0.415
9129   2023-06-02     251147.0        64.0         0.838   8.0         0.347

      mode  energy  instrumentalness  liveness  loudness  speechiness  \
0      1.0   0.605          0.011600    0.0696    -7.121       0.0380
1      1.0   0.813          0.003300    0.1320    -6.416       0.0903
2      1.0   0.797          0.138000    0.2650   -10.205       0.0671
3      1.0   0.511          0.000019    0.1410    -6.972       0.1490
4      1.0   0.580          0.002610    0.1270    -5.596       0.0780
…        …       …                 …         …         …            …
9125   1.0   0.707          0.000000    0.0894    -4.835       0.1230
```

```
9126   0.0   0.526          0.000000    0.4190    -7.153        0.1640
9127   0.0   0.590          0.001660    0.1690    -8.347        0.0942
9128   1.0   0.713          0.000507    0.1230    -5.497        0.2320
9129   1.0   0.707          0.000029    0.1130    -5.533        0.0449

        tempo  time_signature  valence
0     104.931             4.0    0.366
1     199.837             4.0    0.748
2     115.015             4.0    0.972
3     199.775             4.0    0.785
4     196.078             4.0    0.927
…         …             …      …
9125  101.011             4.0    0.314
9126  102.196             4.0    0.568
9127  106.004             4.0    0.683
9128  202.034             4.0    0.518
9129  100.980             4.0    0.630

[8826 rows x 20 columns]
```

[42]:
```python
#creating columns for each genre in the new_genres list
for genre in new_genres:
    pattern = re.compile(fr'\b{genre}\b')
    df[genre] = (df['genre'].apply(lambda x: bool(pattern.search(x))).
 ↪astype('int')
```

[43]:
```python
# View all rows where 'pop' is included as a genre
df[df['pop']==1]
```

[43]:
```
                      track_name                track_id  \
0                        Bandana  2qWwuCVeMjF9mUTOS5Iqvl
1             All Of Us (Ashawo)  6459gZKddpOoPIH8PAcCwS
3             Adore (feat. euro)  3ouP8HFixJmafK7hd1wJ0q
4                          Sofri  6S5XNauc7v8FLJWEIk0z2c
6      Compromise (feat. Rema)  2dG1cXdbEPKEOyUq96R9xz
…                          …                       …
9125                    Odo Dede  5JBOEcpkbUsyaU9EvzK3bw
9126                Save My Soul  0dXCiV6LK9YkpBP5lbFiD4
9127                   Decisions  2U5vPEm0m58dY8DCmKx1hr
9128                   Sugarcane  2HfK1KumDffDWPZga46Hmw
9129            Sugarcane - Remix  6NuG2JgERZZXvvjmtjOFix

                            genre                album_name  artist_name  \
0            afrobeats,nigerian pop                   Playboy  Fireboy DML
1            afrobeats,nigerian pop                   Playboy  Fireboy DML
3            afrobeats,nigerian pop                   Playboy  Fireboy DML
4            afrobeats,nigerian pop                   Playboy  Fireboy DML
```

```
6             afrobeats,nigerian pop                  Playboy  Fireboy DML
…                                  …                      …          …
9125  afro r&b,afrobeats,ghanaian pop  L.I.T.A (Deluxe Edition)      Camidoh
9126  afro r&b,afrobeats,ghanaian pop  L.I.T.A (Deluxe Edition)      Camidoh
9127  afro r&b,afrobeats,ghanaian pop                   L.I.T.A      Camidoh
9128  afro r&b,afrobeats,ghanaian pop                   L.I.T.A      Camidoh
9129  afro r&b,afrobeats,ghanaian pop                   L.I.T.A      Camidoh

     release_date  duration_ms  popularity  danceability   key  …  kwaito  \
0      2022-08-04     178225.0        73.0         0.818   1.0  …       0
1      2022-08-04     183349.0        62.0         0.605  11.0  …       0
3      2022-08-04     201826.0        42.0         0.709   0.0  …       0
4      2022-08-04     179246.0        47.0         0.745   6.0  …       0
6      2022-08-04     195939.0        53.0         0.686   7.0  …       0
…             …            …           …            …    …  …       …
9125   2023-06-23     236202.0        29.0         0.651   6.0  …       0
9126   2023-06-23     139080.0        13.0         0.529   7.0  …       0
9127   2023-06-02     197041.0        25.0         0.835   4.0  …       0
9128   2023-06-02     156781.0        56.0         0.519   8.0  …       0
9129   2023-06-02     251147.0        64.0         0.838   8.0  …       0

      jazz  afrobeats  pop  house  afropop  xhosa  ndombolo  soul  \
0        0          1    1      0        0      0         0     0
1        0          1    1      0        0      0         0     0
3        0          1    1      0        0      0         0     0
4        0          1    1      0        0      0         0     0
6        0          1    1      0        0      0         0     0
…      …          …  …      …        …      …         …     …
9125     0          1    1      0        0      0         0     0
9126     0          1    1      0        0      0         0     0
9127     0          1    1      0        0      0         0     0
9128     0          1    1      0        0      0         0     0
9129     0          1    1      0        0      0         0     0

      zilizopendwa
0                0
1                0
3                0
4                0
6                0
…              …
9125             0
9126             0
9127             0
9128             0
9129             0
```

```
[4212 rows x 36 columns]
```

[44]:
```python
# View all rows where 'azontobeats' is included as a genre
df[df['azontobeats']==1]
```

[44]:
```
        track_name                track_id  \
2          Playboy  2gGAyatRqjjx3DOmLGI12W
11           Glory  5KLFqxmGAZKj3HpGzExiZR
27       Vibration  1G9vMHSCONlfAJpr43dXLp
73      Superwoman  2NOCQeerTwRs3qHicCma4J
76         Beat It  3rL8A5P8pMH6E3KdK1xG3n
…              …                       …
9112     Neighbour  0nmNi1EhdLOSwTntGieWzs
9113    Armageddon  7zvjLlVmJ6r3g2EiSWpJ4W
9114          Dana  5D3MhUkeFoOHmdGG8uOVTX
9115   Ugly Parade  4H8dMbq5ffZHI5oNjuq1S5
9116      Mistakes  0kSLSvGkVJKSeBvUdiBVPC

                                                 genre   album_name  \
2                                  azontobeats,hiplife     Play Boy
11       afrobeats,afropop,azontobeats,ghanaian hip hop      Highest
27            afrobeats,azontobeats,azontobeats,hiplife  inVeencible
73                  azontobeats,bongo flava,tanzanian pop     Flamingo
76      afrobeats,afropop,alte,azontobeats,nigerian pop       Oga Ju
…                                                    …            …
9112  afrobeats,afropop,azontobeats,nigerian hip hop…  Old Romance
9113  afrobeats,afropop,azontobeats,nigerian hip hop…  Old Romance
9114  afrobeats,afropop,azontobeats,nigerian hip hop…  Old Romance
9115  afrobeats,afropop,azontobeats,nigerian hip hop…  Old Romance
9116  afrobeats,afropop,azontobeats,nigerian hip hop…  Old Romance

      artist_name release_date  duration_ms  popularity  danceability   key  \
2     Daddy Lumba   1992-10-05     316440.0        16.0         0.732  11.0
11       Sarkodie   2017-09-08     201750.0        37.0         0.450   5.0
27          MzVee   2020-12-11     189500.0         3.0         0.825   9.0
73        Ben Pol   2023-12-15     173165.0        20.0         0.888   1.0
76           Simi   2011-03-27     195880.0         6.0         0.850   9.0
…              …          …          …          …            …    …
9112        Tekno   2020-12-10     152142.0        28.0         0.894   1.0
9113        Tekno   2020-12-10     184800.0        19.0         0.720   5.0
9114        Tekno   2020-12-10     216000.0        24.0         0.661   8.0
9115        Tekno   2020-12-10     124878.0        18.0         0.807   2.0
9116        Tekno   2020-12-10     177541.0        14.0         0.914   0.0

      …  kwaito  jazz  afrobeats  pop  house  afropop  xhosa  ndombolo  \
2     …       0     0          0    0      0        0      0         0
11    …       0     0          1    0      0        1      0         0
```

```
27    …    0    0    1    0    0    0    0    0
73    …    0    0    0    1    0    0    0    0
76    …    0    0    1    1    0    1    0    0
…   …   …   …   …  …   …   …   …   …
9112  …    0    0    1    1    0    1    0    0
9113  …    0    0    1    1    0    1    0    0
9114  …    0    0    1    1    0    1    0    0
9115  …    0    0    1    1    0    1    0    0
9116  …    0    0    1    1    0    1    0    0

      soul  zilizopendwa
2        0             0
11       0             0
27       0             0
73       0             0
76       0             0
…      …             …
9112     0             0
9113     0             0
9114     0             0
9115     0             0
9116     0             0

[2368 rows x 36 columns]
```

[45]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8826 entries, 0 to 9129
Data columns (total 36 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   track_name        8826 non-null   object
 1   track_id          8826 non-null   object
 2   genre             8826 non-null   object
 3   album_name        8826 non-null   object
 4   artist_name       8826 non-null   object
 5   release_date      8826 non-null   object
 6   duration_ms       8826 non-null   float64
 7   popularity        8826 non-null   float64
 8   danceability      8826 non-null   float64
 9   key               8826 non-null   float64
 10  acousticness      8826 non-null   float64
 11  mode              8826 non-null   float64
 12  energy            8826 non-null   float64
 13  instrumentalness  8826 non-null   float64
 14  liveness          8826 non-null   float64
```

```
 15  loudness          8826 non-null   float64
 16  speechiness       8826 non-null   float64
 17  tempo             8826 non-null   float64
 18  time_signature    8826 non-null   float64
 19  valence           8826 non-null   float64
 20  hip hop           8826 non-null   int32
 21  world             8826 non-null   int32
 22  soukous           8826 non-null   int32
 23  rumba congolaise  8826 non-null   int32
 24  azontobeats       8826 non-null   int32
 25  hiplife           8826 non-null   int32
 26  kwaito            8826 non-null   int32
 27  jazz              8826 non-null   int32
 28  afrobeats         8826 non-null   int32
 29  pop               8826 non-null   int32
 30  house             8826 non-null   int32
 31  afropop           8826 non-null   int32
 32  xhosa             8826 non-null   int32
 33  ndombolo          8826 non-null   int32
 34  soul              8826 non-null   int32
 35  zilizopendwa      8826 non-null   int32
dtypes: float64(14), int32(16), object(6)
memory usage: 2.0+ MB
```

[46]:

```
#removing the redundant genre column
df.drop('genre', axis=1, inplace=True)
df.head()
```

[46]:

```
            track_name                track_id album_name  artist_name  \
0              Bandana  2qWwuCVeMjF9mUTOS5Iqvl    Playboy  Fireboy DML
1  All Of Us (Ashawo)  6459gZKddpOoPIH8PAcCwS    Playboy  Fireboy DML
2              Playboy  2gGAyatRqjjx3DOmLGI12W   Play Boy  Daddy Lumba
3  Adore (feat. euro)  3ouP8HFixJmafK7hd1wJ0q    Playboy  Fireboy DML
4                Sofri  6S5XNauc7v8FLJWEIk0z2c    Playboy  Fireboy DML

  release_date  duration_ms  popularity  danceability   key  acousticness  \
0   2022-08-04     178225.0        73.0         0.818   1.0         0.293
1   2022-08-04     183349.0        62.0         0.605  11.0         0.304
2   1992-10-05     316440.0        16.0         0.732  11.0         0.225
3   2022-08-04     201826.0        42.0         0.709   0.0         0.108
4   2022-08-04     179246.0        47.0         0.745   6.0         0.341

   …  kwaito  jazz  afrobeats  pop  house  afropop  xhosa  ndombolo  soul  \
0  …       0     0          1    1      0        0      0         0     0
1  …       0     0          1    1      0        0      0         0     0
2  …       0     0          0    0      0        0      0         0     0
3  …       0     0          1    1      0        0      0         0     0
```

```
4  …        0      0           1     1       0         0       0          0      0

   zilizopendwa
0             0
1             0
2             0
3             0
4             0

[5 rows x 35 columns]
```

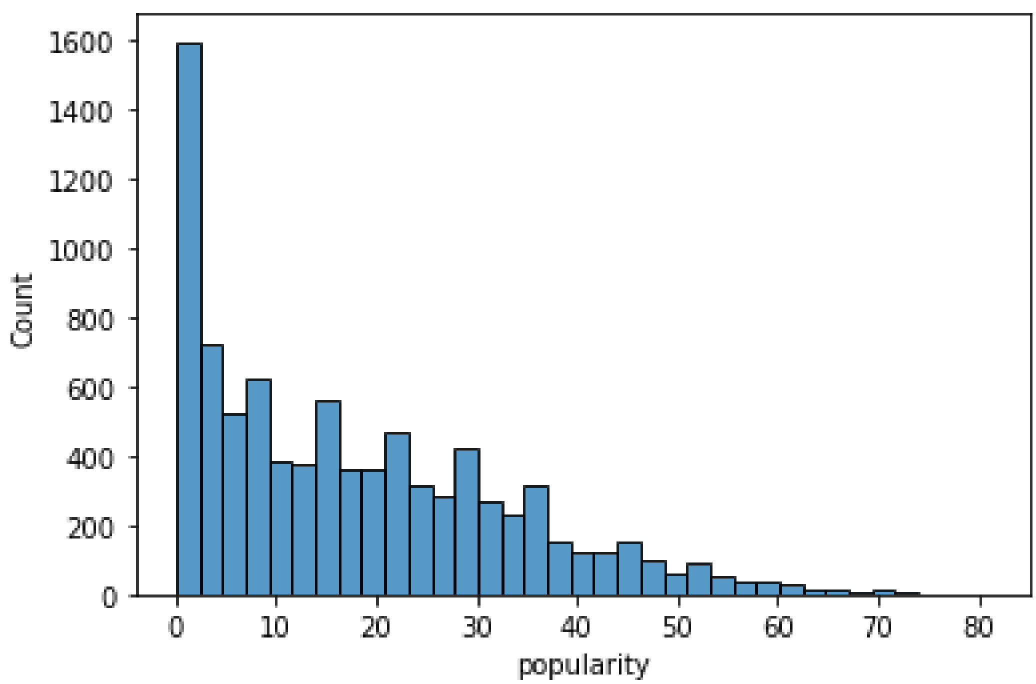
0.0.2 Feature Engineering - is_popular

Since our goal is to be able to identify which tracks will be popular, we need to feature engineer a new column by binarizing the popularity column. To be able to do this, we need to decide on a cut-off point of popularity score which if a song stays above this cut-off point it will be considered "popular" and if it stays below it will be considered "not popular". We can start off by taking a look at the distribution of the popularity score distribution.

[47]:
```python
import matplotlib.pyplot as plt
import seaborn as sns
```

[48]:
```python
#creating a histogram to see distribution of popularity scores in the dataset.
sns.histplot(df['popularity'], bins='auto')
```

[48]: <AxesSubplot:xlabel='popularity', ylabel='Count'>



Top 100 Songs In order to better decide what's popular, we can take a look at the Top 100 songs' popularity scores from a playlist that contains top 100 popular songs by african artist created by a spotify user.

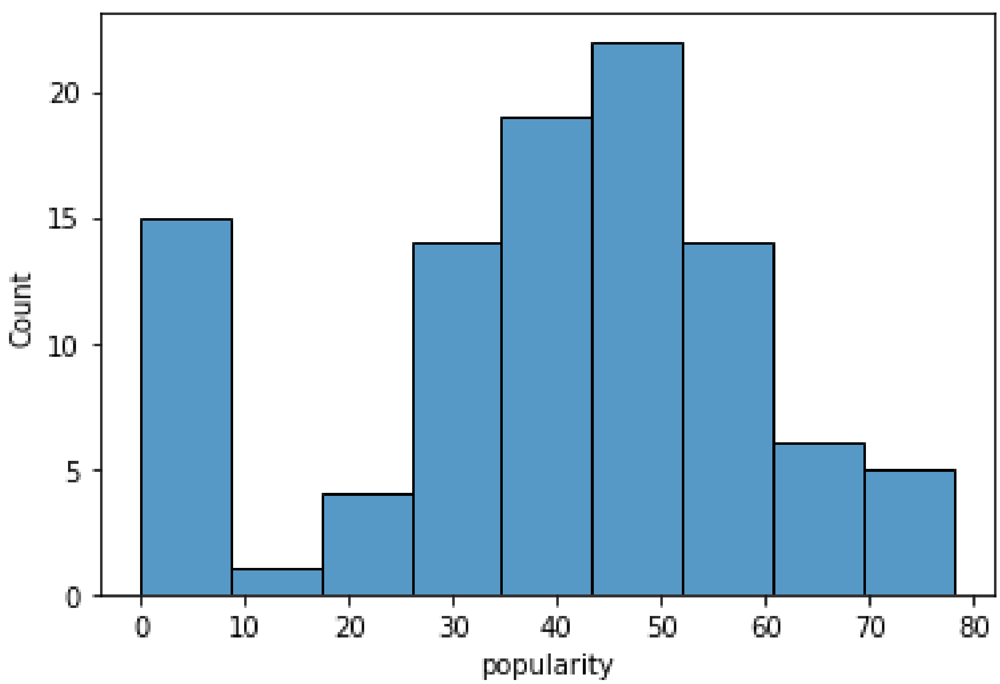
[49]:
```
# https://open.spotify.com/playlist/1C9vnCyBuQykXAe2U1EcHW?
 ↪si=kEjb6Rj2R7ahxdbeoFY94A&pi=e-FwF0r-nyTmSg
df_100 = pd.read_csv('data/top_100_african_hits.csv')
```

[50]:
```
df_100['popularity'].describe()
```

[50]:
```
count    100.000000
mean      38.860000
std       20.584892
min        0.000000
25%       30.500000
50%       42.500000
75%       51.250000
max       78.000000
Name: popularity, dtype: float64
```

[51]:
```
fig, ax = plt.subplots()
sns.histplot(df_100['popularity'], bins='auto', ax=ax)
```

[51]: <AxesSubplot:xlabel='popularity', ylabel='Count'>



From the above histogram we see that we have a bimodal distribution. One of the peaks is around 5, and the other one seems to be around 45.

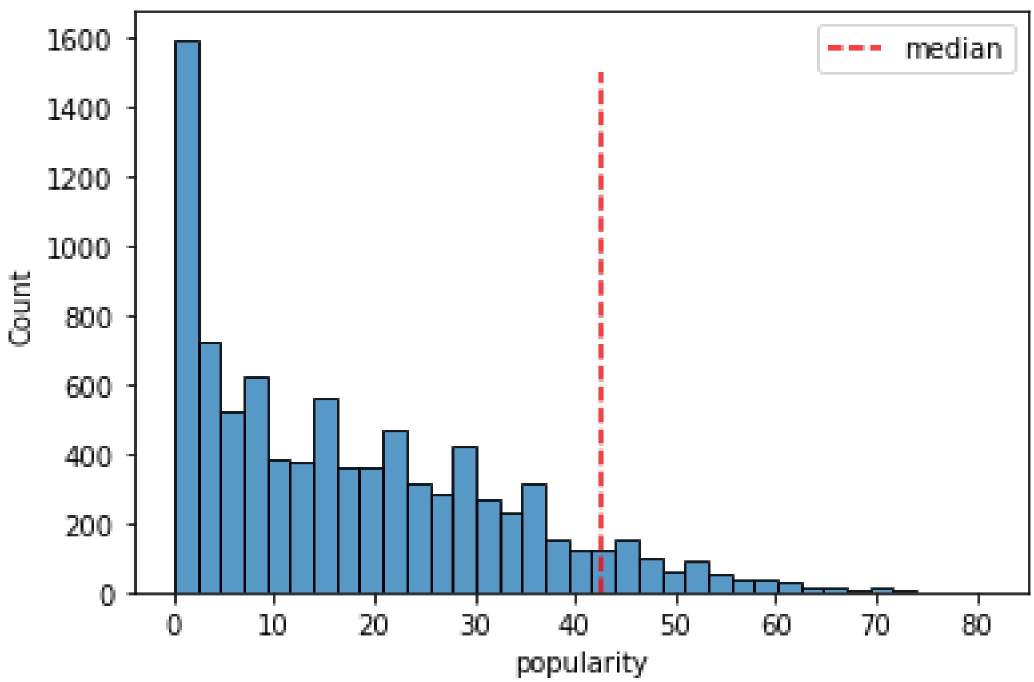
[52]:
```
df_100['popularity'].describe()['50%']        # Median value
```

[52]:
```
42.5
```

We will be defining a song being popular as being African Top 100 worthy and therefore we will establish our cutoff point at the median value (42.5)

[53]:
```
# Visualizing the meadian popularity scores on the overall dataset histogram
fig, ax = plt.subplots()
sns.histplot(df['popularity'], bins='auto', ax=ax)
ax.vlines(x=df_100['popularity'].describe()['50%'], ymin=0, ymax=1500,␣
 ↪linestyles='dashed', colors='red', label='median')
plt.legend()
```

[53]:
```
<matplotlib.legend.Legend at 0x19e493a3820>
```



[54]:
```
#creating is_popular column with our cutoff point
df['is_popular']=(df['popularity']>=42.5).astype('int')
df.head()
```

[54]:
```
            track_name                track_id album_name  artist_name  \
0              Bandana  2qWwuCVeMjF9mUTOS5Iqvl    Playboy  Fireboy DML
1  All Of Us (Ashawo)  6459gZKddpOoPIH8PAcCwS    Playboy  Fireboy DML
2              Playboy  2gGAyatRqjjx3DOmLGI12W   Play Boy  Daddy Lumba
3  Adore (feat. euro)  3ouP8HFixJmafK7hd1wJ0q    Playboy  Fireboy DML
4                Sofri  6S5XNauc7v8FLJWEIk0z2c    Playboy  Fireboy DML

  release_date  duration_ms  popularity  danceability   key  acousticness  \
0   2022-08-04     178225.0        73.0         0.818   1.0         0.293
1   2022-08-04     183349.0        62.0         0.605  11.0         0.304
2   1992-10-05     316440.0        16.0         0.732  11.0         0.225
3   2022-08-04     201826.0        42.0         0.709   0.0         0.108
4   2022-08-04     179246.0        47.0         0.745   6.0         0.341

   …  jazz  afrobeats  pop  house  afropop  xhosa  ndombolo  soul  \
0  …     0          1    1      0        0      0         0     0
1  …     0          1    1      0        0      0         0     0
2  …     0          0    0      0        0      0         0     0
3  …     0          1    1      0        0      0         0     0
4  …     0          1    1      0        0      0         0     0

   zilizopendwa  is_popular
0             0           1
1             0           1
2             0           0
3             0           0
4             0           1

[5 rows x 36 columns]
```

[55]:
```
#dropping popularity score column since we will not be using it
df.drop(['popularity', 'artist_name', 'track_name', 'album_name',␣
 ↪'release_date'], axis=1, inplace=True)
df.set_index('track_id', inplace=True)    # Set the 'track_id' column as the␣
 ↪index
df.head()
```

[55]:
```
                        duration_ms  danceability   key  acousticness  mode  \
track_id
2qWwuCVeMjF9mUTOS5Iqvl     178225.0         0.818   1.0         0.293   1.0
6459gZKddpOoPIH8PAcCwS     183349.0         0.605  11.0         0.304   1.0
2gGAyatRqjjx3DOmLGI12W     316440.0         0.732  11.0         0.225   1.0
3ouP8HFixJmafK7hd1wJ0q     201826.0         0.709   0.0         0.108   1.0
6S5XNauc7v8FLJWEIk0z2c     179246.0         0.745   6.0         0.341   1.0

                        energy  instrumentalness  liveness  loudness  \
track_id
```

```
2qWwuCVeMjF9mUT0S5Iqvl   0.605         0.011600   0.0696    -7.121
6459gZKddpOoPIH8PAcCwS   0.813         0.003300   0.1320    -6.416
2gGAyatRqjjx3DOmLGI12W   0.797         0.138000   0.2650   -10.205
3ouP8HFixJmafK7hd1wJ0q   0.511         0.000019   0.1410    -6.972
6S5XNauc7v8FLJWEIk0z2c   0.580         0.002610   0.1270    -5.596

                        speechiness  …  jazz  afrobeats  pop  house  \
track_id                             …
2qWwuCVeMjF9mUT0S5Iqvl       0.0380  …     0          1    1      0
6459gZKddpOoPIH8PAcCwS       0.0903  …     0          1    1      0
2gGAyatRqjjx3DOmLGI12W       0.0671  …     0          0    0      0
3ouP8HFixJmafK7hd1wJ0q       0.1490  …     0          1    1      0
6S5XNauc7v8FLJWEIk0z2c       0.0780  …     0          1    1      0

                        afropop  xhosa  ndombolo  soul  zilizopendwa  \
track_id
2qWwuCVeMjF9mUT0S5Iqvl        0      0         0     0             0
6459gZKddpOoPIH8PAcCwS        0      0         0     0             0
2gGAyatRqjjx3DOmLGI12W        0      0         0     0             0
3ouP8HFixJmafK7hd1wJ0q        0      0         0     0             0
6S5XNauc7v8FLJWEIk0z2c        0      0         0     0             0

                        is_popular
track_id
2qWwuCVeMjF9mUT0S5Iqvl           1
6459gZKddpOoPIH8PAcCwS           1
2gGAyatRqjjx3DOmLGI12W           0
3ouP8HFixJmafK7hd1wJ0q           0
6S5XNauc7v8FLJWEIk0z2c           1

[5 rows x 30 columns]
```

We dropped popularity scores since we already binarized that column, but additionally we are dropping `'artist_name'`, `'track_name'`, `'album_name'`, and `'release_date'` since we are looking at the anatomy of a song and not who sings it, what it's called or when it was released. The goal is to identify songs that will become popular without being affected by the artist's name since we would also like to find songs from up-and-coming artists.

0.0.3 train_test_split

[56]:
```
df.shape
```

[56]:
```
(8826, 30)
```

[57]:
```
#splitting the data to training and test sets in order to be able to measure␣
↪performance
from sklearn.model_selection import train_test_split
```

```
y=df['is_popular']
X=df.drop('is_popular',axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30,
 ↪random_state=42)
```

0.0.4 One Hot Encoding the Categorical Columns

We still have categorical columns that need one hot encoding. Namely, these columns are `key`, `mode` and `time_signature`.

[58]:
```
#Check to see how many more columns we will be creating by OHE the cat_cols.
df.nunique()
```

[58]:
```
duration_ms         8002
danceability         706
key                   12
acousticness        2117
mode                   2
energy               919
instrumentalness    3023
liveness            1439
loudness            6104
speechiness         1238
tempo               7326
time_signature         5
valence              945
hip hop                2
world                  2
soukous                2
rumba congolaise       2
azontobeats            2
hiplife                2
kwaito                 2
jazz                   2
afrobeats              2
pop                    2
house                  2
afropop                2
xhosa                  2
ndombolo               2
soul                   2
zilizopendwa           2
is_popular             2
dtype: int64
```

[59]:
```
df.nunique()['mode']
```

[59]: 2

[60]:
```
df.nunique()['time_signature']
```

[60]: 5

[61]:
```
df.nunique()['key']
```

[61]: 12

We will be creating 2 (mode) + 5 (time_signature) + 12 (key) - 3 (we will drop the three original columns) = 16 columns

[62]:
```
#define categorical columns
cat_cols = ['key', 'mode', 'time_signature']
```

[63]:
```
#One hot encoding the dataframes
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse_output=False, drop='first')
#Training set
data_ohe_train = encoder.fit_transform(X_train[cat_cols])
df_ohe_train = pd.DataFrame(data_ohe_train, columns=encoder.
 ↪get_feature_names_out(cat_cols), index=X_train.index)

#Testing set
data_ohe_test = encoder.transform(X_test[cat_cols])
df_ohe_test = pd.DataFrame(data_ohe_test, columns=encoder.
 ↪get_feature_names_out(cat_cols), index=X_test.index)
```

[64]:
```
pd.set_option("display.max_columns", None)
df_ohe_train
```

[64]:
```
                        key_1.0  key_2.0  key_3.0  key_4.0  key_5.0  key_6.0  \
track_id
4bq7abLmcXYeXAqJNIRJQZ      1.0      0.0      0.0      0.0      0.0      0.0
5BFEc76XGgq7jvfUPZcgtr      0.0      0.0      0.0      0.0      0.0      0.0
21VTQO5QzUZJzCNnCDcr2e      0.0      0.0      0.0      0.0      0.0      0.0
4586JTjH3ZQsahmhxFOOvX      0.0      0.0      0.0      0.0      0.0      0.0
1tiUbKSS0iIG636taFaday      0.0      0.0      1.0      0.0      0.0      0.0
…                             …       …       …       …       …       …
6jXp6dIALVTtfVctb4ukNi      0.0      1.0      0.0      0.0      0.0      0.0
4QobRESI1KqQyNpWtxjUqm      0.0      0.0      0.0      1.0      0.0      0.0
7L3sQ9DSqZTmxkxZy7HMxe      0.0      0.0      0.0      0.0      0.0      0.0
4MccnsxZ9Dog74vkSrzInx      0.0      0.0      0.0      0.0      1.0      0.0
3sk2cdMqfkuoThtBt1G9Ls      0.0      0.0      0.0      0.0      0.0      0.0
```

```
                          key_7.0  key_8.0  key_9.0  key_10.0  key_11.0  \
track_id
4bq7abLmcXYeXAqJNIRJQZ      0.0      0.0      0.0       0.0       0.0
5BFEc76XGgq7jvfUPZcgtr      1.0      0.0      0.0       0.0       0.0
21VTQO5QzUZJzCNnCDcr2e      0.0      0.0      0.0       0.0       1.0
4586JTjH3ZQsahmhxF0OvX      0.0      0.0      0.0       1.0       0.0
1tiUbKSS0iIG636taFaday      0.0      0.0      0.0       0.0       0.0
…                           …        …        …         …         …
6jXp6dIALVTtfVctb4ukNi      0.0      0.0      0.0       0.0       0.0
4QobRESIlKqQyNpWtxjUqm      0.0      0.0      0.0       0.0       0.0
7L3sQ9DSqZTmxkxZy7HMxe      0.0      0.0      0.0       0.0       0.0
4MccnsxZ9Dog74vkSrzInx      0.0      0.0      0.0       0.0       0.0
3sk2cdMqfkuoThtBt1G9Ls      0.0      0.0      0.0       0.0       0.0

                          mode_1.0  time_signature_1.0  time_signature_3.0  \
track_id
4bq7abLmcXYeXAqJNIRJQZ       1.0                 0.0                 0.0
5BFEc76XGgq7jvfUPZcgtr       1.0                 0.0                 0.0
21VTQO5QzUZJzCNnCDcr2e       0.0                 0.0                 0.0
4586JTjH3ZQsahmhxF0OvX       1.0                 0.0                 0.0
1tiUbKSS0iIG636taFaday       0.0                 0.0                 0.0
…                            …                   …                   …
6jXp6dIALVTtfVctb4ukNi       1.0                 0.0                 0.0
4QobRESIlKqQyNpWtxjUqm       0.0                 0.0                 0.0
7L3sQ9DSqZTmxkxZy7HMxe       1.0                 0.0                 0.0
4MccnsxZ9Dog74vkSrzInx       1.0                 0.0                 0.0
3sk2cdMqfkuoThtBt1G9Ls       0.0                 0.0                 0.0

                          time_signature_4.0  time_signature_5.0
track_id
4bq7abLmcXYeXAqJNIRJQZ                   1.0                 0.0
5BFEc76XGgq7jvfUPZcgtr                   1.0                 0.0
21VTQO5QzUZJzCNnCDcr2e                   1.0                 0.0
4586JTjH3ZQsahmhxF0OvX                   1.0                 0.0
1tiUbKSS0iIG636taFaday                   1.0                 0.0
…                                        …                   …
6jXp6dIALVTtfVctb4ukNi                   1.0                 0.0
4QobRESIlKqQyNpWtxjUqm                   1.0                 0.0
7L3sQ9DSqZTmxkxZy7HMxe                   1.0                 0.0
4MccnsxZ9Dog74vkSrzInx                   1.0                 0.0
3sk2cdMqfkuoThtBt1G9Ls                   1.0                 0.0

[6178 rows x 16 columns]
```

[65]:

```
#merging OHE columns with numerical columns
X_train = pd.concat([X_train.drop(cat_cols, axis=1), df_ohe_train], axis=1)
X_test = pd.concat([X_test.drop(cat_cols, axis=1), df_ohe_test], axis=1)
```

```python
X_train.tail()
```

[65]:

```
                        duration_ms  danceability  acousticness  energy  \
track_id
6jXp6dIALVTtfVctb4ukNi     295367.0         0.697         0.174   0.739
4QobRESIlKqQyNpWtxjUqm     167151.0         0.809         0.383   0.727
7L3sQ9DSqZTmxkxZy7HMxe     265440.0         0.615         0.220   0.723
4MccnsxZ9Dog74vkSrzInx     322226.0         0.630         0.486   0.574
3sk2cdMqfkuoThtBt1G9Ls     193873.0         0.776         0.440   0.672

                        instrumentalness  liveness  loudness  speechiness  \
track_id
6jXp6dIALVTtfVctb4ukNi          0.000000    0.0612    -7.456       0.1230
4QobRESIlKqQyNpWtxjUqm          0.000000    0.1280    -5.572       0.0967
7L3sQ9DSqZTmxkxZy7HMxe          0.000000    0.0820    -4.808       0.4040
4MccnsxZ9Dog74vkSrzInx          0.903000    0.1390   -12.067       0.0336
3sk2cdMqfkuoThtBt1G9Ls          0.000002    0.0513    -4.992       0.0600

                          tempo  valence  hip hop  world  soukous  \
track_id
6jXp6dIALVTtfVctb4ukNi  108.797    0.866        0      0        1
4QobRESIlKqQyNpWtxjUqm  140.062    0.811        1      0        0
7L3sQ9DSqZTmxkxZy7HMxe   97.998    0.941        0      0        0
4MccnsxZ9Dog74vkSrzInx  123.562    0.583        0      0        0
3sk2cdMqfkuoThtBt1G9Ls   99.969    0.961        0      0        0

                        rumba congolaise  azontobeats  hiplife  kwaito  jazz  \
track_id
6jXp6dIALVTtfVctb4ukNi                 0            0        0       0     0
4QobRESIlKqQyNpWtxjUqm                 0            0        0       0     0
7L3sQ9DSqZTmxkxZy7HMxe                 0            0        0       0     0
4MccnsxZ9Dog74vkSrzInx                 0            0        0       1     1
3sk2cdMqfkuoThtBt1G9Ls                 0            1        1       0     0

                        afrobeats  pop  house  afropop  xhosa  ndombolo  soul  \
track_id
6jXp6dIALVTtfVctb4ukNi          0    0      0        0      0         0     0
4QobRESIlKqQyNpWtxjUqm          0    0      0        0      0         0     0
7L3sQ9DSqZTmxkxZy7HMxe          1    1      0        1      0         0     0
4MccnsxZ9Dog74vkSrzInx          0    0      0        1      0         0     0
3sk2cdMqfkuoThtBt1G9Ls          1    0      0        0      0         0     0

                        zilizopendwa  key_1.0  key_2.0  key_3.0  key_4.0  \
track_id
6jXp6dIALVTtfVctb4ukNi             1      0.0      1.0      0.0      0.0
4QobRESIlKqQyNpWtxjUqm             0      0.0      0.0      0.0      1.0
7L3sQ9DSqZTmxkxZy7HMxe             0      0.0      0.0      0.0      0.0
```

```
4MccnsxZ9Dog74vkSrzInx              0      0.0      0.0      0.0      0.0
3sk2cdMqfkuoThtBt1G9Ls              0      0.0      0.0      0.0      0.0

                        key_5.0  key_6.0  key_7.0  key_8.0  key_9.0  key_10.0  \
track_id
6jXp6dIALVTtfVctb4ukNi      0.0      0.0      0.0      0.0      0.0       0.0
4QobRESIlKqQyNpWtxjUqm      0.0      0.0      0.0      0.0      0.0       0.0
7L3sQ9DSqZTmxkxZy7HMxe      0.0      0.0      0.0      0.0      0.0       0.0
4MccnsxZ9Dog74vkSrzInx      1.0      0.0      0.0      0.0      0.0       0.0
3sk2cdMqfkuoThtBt1G9Ls      0.0      0.0      0.0      0.0      0.0       0.0

                        key_11.0  mode_1.0  time_signature_1.0  \
track_id
6jXp6dIALVTtfVctb4ukNi       0.0       1.0                 0.0
4QobRESIlKqQyNpWtxjUqm       0.0       0.0                 0.0
7L3sQ9DSqZTmxkxZy7HMxe       0.0       1.0                 0.0
4MccnsxZ9Dog74vkSrzInx       0.0       1.0                 0.0
3sk2cdMqfkuoThtBt1G9Ls       0.0       0.0                 0.0

                        time_signature_3.0  time_signature_4.0  \
track_id
6jXp6dIALVTtfVctb4ukNi                 0.0                 1.0
4QobRESIlKqQyNpWtxjUqm                 0.0                 1.0
7L3sQ9DSqZTmxkxZy7HMxe                 0.0                 1.0
4MccnsxZ9Dog74vkSrzInx                 0.0                 1.0
3sk2cdMqfkuoThtBt1G9Ls                 0.0                 1.0

                        time_signature_5.0
track_id
6jXp6dIALVTtfVctb4ukNi                 0.0
4QobRESIlKqQyNpWtxjUqm                 0.0
7L3sQ9DSqZTmxkxZy7HMxe                 0.0
4MccnsxZ9Dog74vkSrzInx                 0.0
3sk2cdMqfkuoThtBt1G9Ls                 0.0
```

[66]:
```
#concatenating all parts of our data for future reference (see Data␣
↪Visualizations section)
df_ohe_x = pd.concat([X_train, X_test])
df_ohe_y = pd.concat([y_train, y_test])
df_ohe = pd.concat([df_ohe_x, df_ohe_y], axis=1)
```

With both the X_train and X_test dataframes scrubbed and one hot encoded we can move onto the modelling process.

0.1 MODEL

The first model we will be generating is a dummy classifier. We will be comparing our models' success to each other but also to this baseline model.

0.1.1 Model #0 - Baseline - Dummy Classifier

[67]:
```
from sklearn.dummy import DummyClassifier

clf_dummy = DummyClassifier(random_state=42)
clf_dummy.fit(X_train, y_train)
y_pred = clf_dummy.predict(X_test)
```

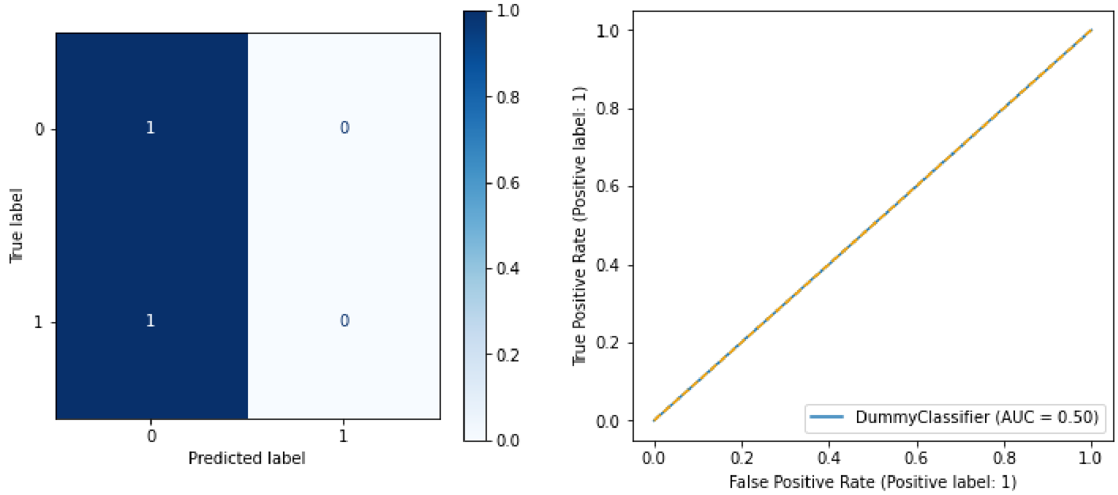
We need a function that will show us the classification report, the confusion matrix as well as the ROC curve to be able to evaluate our models.

[68]:
```
from sklearn.metrics import classification_report, ConfusionMatrixDisplay,␣
↪RocCurveDisplay

def classification(y_true, y_pred, X, clf):
    """This function shows the classification report,
    the confusion matrix as well as the ROC curve for evaluation of model␣
↪quality.

    y_true: Correct y values, typically y_test that comes from the␣
↪train_test_split performed at the beginning of model development.
    y_pred: Predicted y values by the model.
    clf: classifier model that was fit to training data.
    X: X_test values"""

    #Classification report
    print("CLASSIFICATION REPORT")
    print("------------------------------------------")
    print(classification_report(y_true=y_true, y_pred=y_pred, zero_division=0))

    #Creating a figure/axes for confusion matrix and ROC curve
    fig, ax = plt.subplots(ncols=2, figsize=(12, 5))

    #Plotting the normalized confusion matrix
    ConfusionMatrixDisplay.from_estimator(estimator=clf, X=X, y=y_true,␣
↪cmap='Blues', normalize='true', ax=ax[0])

    #Plotting the ROC curve
    RocCurveDisplay.from_estimator(estimator=clf, X=X, y=y_true, ax=ax[1])

    #Plotting the 50-50 guessing plot for reference
    ax[1].plot([0,1], [0,1], ls='--', color='orange')
```

[69]:
```
classification(y_test, y_pred, X_test, clf_dummy)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support
```

```
              0       0.92      1.00      0.96      2449
              1       0.00      0.00      0.00       199

       accuracy                           0.92      2648
      macro avg       0.46      0.50      0.48      2648
   weighted avg       0.86      0.92      0.89      2648
```



0.1.2 Addressing Class Imbalance with SMOTENC

[70]:
```
#class imbalance percentages
y_train.value_counts(normalize=True)
```

[70]:
```
0    0.922791
1    0.077209
Name: is_popular, dtype: float64
```

Our dummy classifier correctly predicted 93% of the unpopular songs as unpopular; however, it correctly predicted only 7% of the popular songs and classified the remaining 93% as unpopular. We clearly have a class imbalance problem where approximately 93% of our data is not popular and only about 7% of it is. To address this we can SMOTE the training data and see if training a model with this method would improve our results.

[71]:
```
#looking at column names to extract categorical column indices for SMOTENC
list(X_train.columns)
```

[71]:
```
['duration_ms',
 'danceability',
 'acousticness',
```

```
 'energy',
 'instrumentalness',
 'liveness',
 'loudness',
 'speechiness',
 'tempo',
 'valence',
 'hip hop',
 'world',
 'soukous',
 'rumba congolaise',
 'azontobeats',
 'hiplife',
 'kwaito',
 'jazz',
 'afrobeats',
 'pop',
 'house',
 'afropop',
 'xhosa',
 'ndombolo',
 'soul',
 'zilizopendwa',
 'key_1.0',
 'key_2.0',
 'key_3.0',
 'key_4.0',
 'key_5.0',
 'key_6.0',
 'key_7.0',
 'key_8.0',
 'key_9.0',
 'key_10.0',
 'key_11.0',
 'mode_1.0',
 'time_signature_1.0',
 'time_signature_3.0',
 'time_signature_4.0',
 'time_signature_5.0']
```

[72]:
```
#creating a list of categorical column indices
cat_cols = list(range(10, len(X_train.columns)))
X_train.columns[cat_cols]
```

[72]:
```
Index(['hip hop', 'world', 'soukous', 'rumba congolaise', 'azontobeats',
       'hiplife', 'kwaito', 'jazz', 'afrobeats', 'pop', 'house', 'afropop',
       'xhosa', 'ndombolo', 'soul', 'zilizopendwa', 'key_1.0', 'key_2.0',
```

```
       'key_3.0', 'key_4.0', 'key_5.0', 'key_6.0', 'key_7.0', 'key_8.0',
       'key_9.0', 'key_10.0', 'key_11.0', 'mode_1.0', 'time_signature_1.0',
       'time_signature_3.0', 'time_signature_4.0', 'time_signature_5.0'],
      dtype='object')
```

[73]:
```
# pip install imblearn --user
```

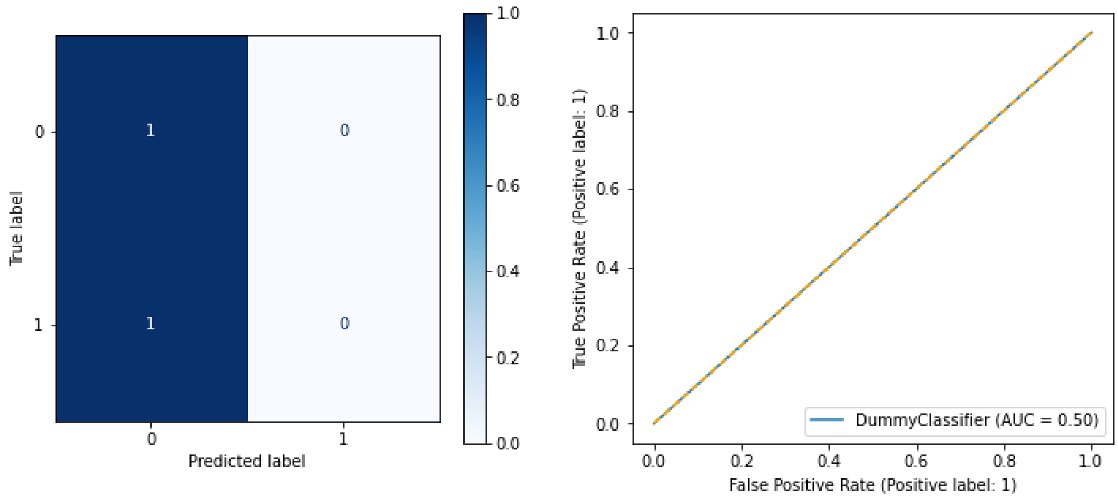
[74]:
```
import imblearn
```

[75]:
```
#Using SMOTENC to address class imbalance. We are not using SMOTE since we have␣
↪categorical columns.
from imblearn.over_sampling import SMOTE, SMOTENC

sm = SMOTENC(categorical_features=cat_cols, random_state=42)

X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
y_train_sm.value_counts(normalize=True)
```

[75]:
```
0    0.5
1    0.5
Name: is_popular, dtype: float64
```

[76]:
```
#fitting Dummy Classifier to data without the class imbalance problem to serve␣
↪as a true baseline
clf_dummy_sm = DummyClassifier(random_state=42)
clf_dummy_sm.fit(X_train_sm, y_train_sm)
y_pred = clf_dummy_sm.predict(X_test)
classification(y_test, y_pred, X_test, clf_dummy_sm)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.92      1.00      0.96      2449
           1       0.00      0.00      0.00       199

    accuracy                           0.92      2648
   macro avg       0.46      0.50      0.48      2648
weighted avg       0.86      0.92      0.89      2648
```



We see here that the dummy classifier is essentially flipping a coin and guessing whether a song is popular or not which is not very useful. However, this serves as a great baseline for our other models to be evaluated against. We can now initialize a results dataframe and keep track of the recall scores of our models for comparison later.

[77]:
```
from sklearn.metrics import recall_score
df_results = pd.DataFrame(columns=['Model Name', 'Recall Score'])

def add_results(model_name, df):
    df = df.append({'Model Name': model_name,
                    'Recall Score': round(recall_score(y_test, y_pred),2)},
                    ignore_index=True)
    return df
```

[78]:
```
df_results = add_results('Dummy Classifier', df_results)
df_results.head()
```

[78]:
```
         Model Name  Recall Score
0  Dummy Classifier           0.0
```

0.1.3 Model #1 - Random Forest Classifier

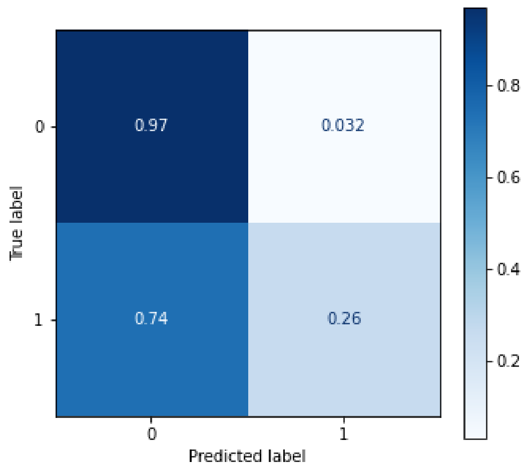
The first model we will be developing is the Random Forest classifier.

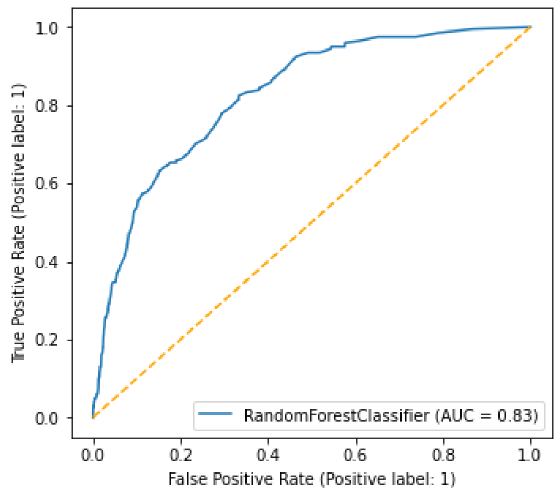
Initial Model

[79]:
```
#Fitting RF Classifier to SMOTE'd data
from sklearn.ensemble import RandomForestClassifier

clf_rf = RandomForestClassifier(random_state=42)
clf_rf.fit(X_train_sm, y_train_sm)
```

```
#Making predictions and evaluation.
y_pred = clf_rf.predict(X_test)
classification(y_test, y_pred, X_test, clf_rf)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      2449
           1       0.40      0.26      0.32       199

    accuracy                           0.92      2648
   macro avg       0.67      0.61      0.64      2648
weighted avg       0.90      0.92      0.91      2648
```



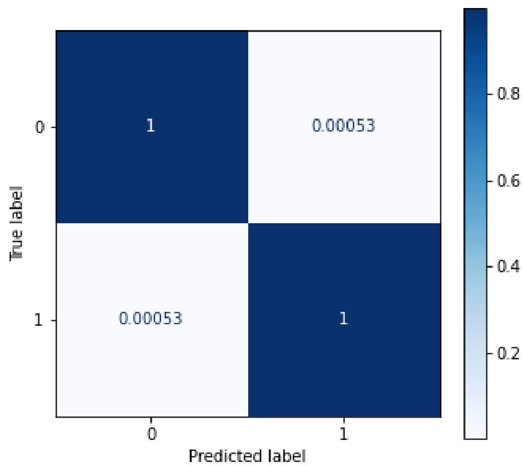


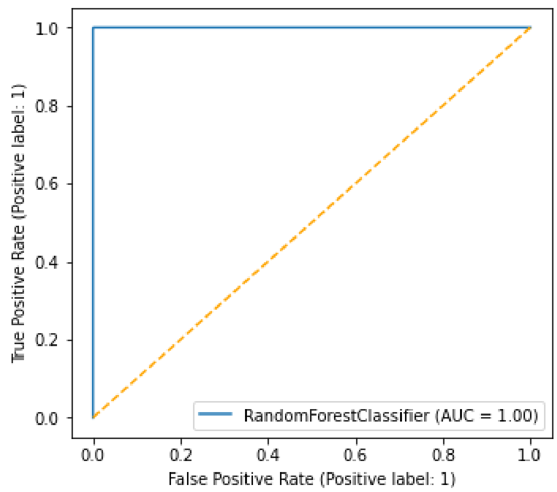
The model may be underfitting, so to confirm we will look at the performance of the model with the training data.

[80]:
```
#Evaluating the model performance for the training data
y_pred = clf_rf.predict(X_train_sm)
classification(y_train_sm, y_pred, X_train_sm, clf_rf)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5701
           1       1.00      1.00      1.00      5701
```

```
    accuracy                           1.00     11402
   macro avg       1.00      1.00      1.00     11402
weighted avg       1.00      1.00      1.00     11402
```





Our model is performing perfectly on the training data but not so much on the test data since it is overfitting to the training set. We need to tune our model to get more accurate results on unseen data. We will be using a grid search to optimize for the recall score. We are optimizing recall instead of other scores since we primarily care about correctly identifying a song that will be popular and we don't mind it if we pick a few songs that don't end up becoming popular.

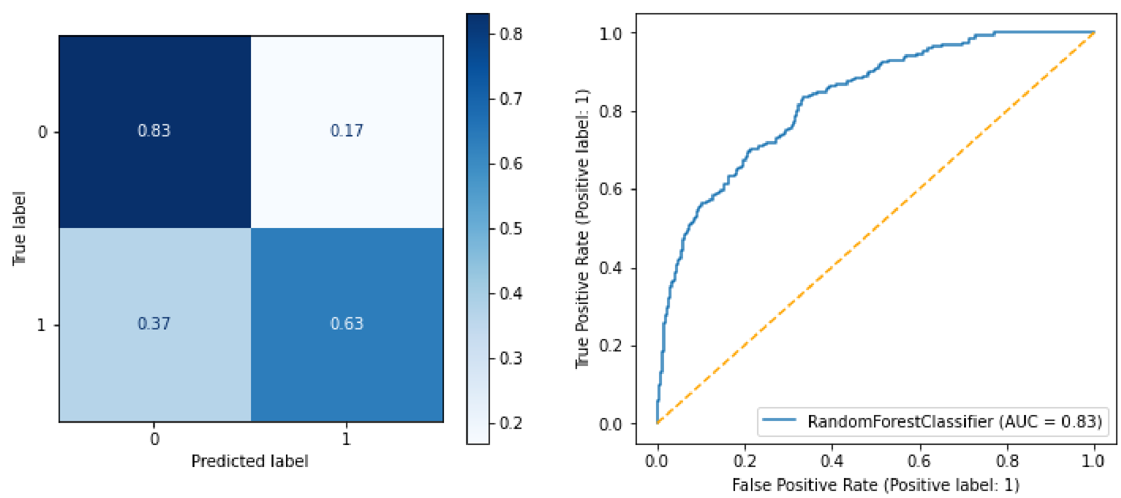
Hyperparameter Tuning

[81]:
```
# from sklearn.model_selection import GridSearchCV

# clf = RandomForestClassifier()
# grid = {'criterion': ['gini', 'entropy'],
#         'max_depth': [4, 5, 6],
#         'min_samples_leaf': [3, 4, 5, 6, 7]
#        }

# gridsearch = GridSearchCV(estimator=clf, param_grid=grid, scoring='recall',␣
↪n_jobs=-1, verbose=2)

# gridsearch.fit(X_train_sm,  y_train_sm)
# gridsearch.best_params_
# #Results: {'criterion': 'gini', 'max_depth': 6, 'min_samples_leaf': 3}
```

[82]:
```
clf_rf_tuned = RandomForestClassifier(criterion='gini', max_depth=6,
                                      min_samples_leaf=3,␣
↪class_weight='balanced',
```

```
                                                     random_state=42)
clf_rf_tuned.fit(X_train_sm, y_train_sm)

y_pred = clf_rf_tuned.predict(X_test)
classification(y_test, y_pred, X_test, clf_rf_tuned)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.97      0.83      0.89      2449
           1       0.23      0.63      0.34       199

    accuracy                           0.82      2648
   macro avg       0.60      0.73      0.62      2648
weighted avg       0.91      0.82      0.85      2648
```



[83]:
```
#appending the recall score to the results dataframe
df_results = add_results('Random Forest', df_results)
df_results.head()
```

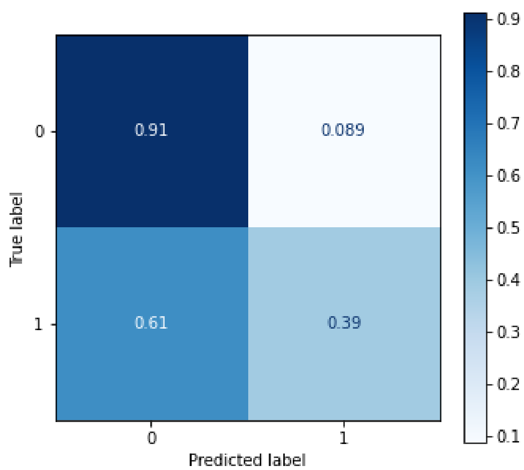
[83]:

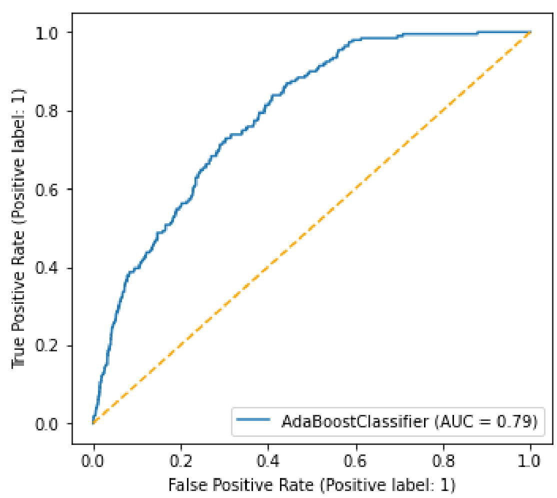
	Model Name	Recall Score
0	Dummy Classifier	0.00
1	Random Forest	0.63

0.1.4 Model #2- AdaBoost

[84]:
```
from sklearn.ensemble import AdaBoostClassifier

# Create the AdaBoost classifier
ada = AdaBoostClassifier(random_state=42)

ada.fit(X_train_sm,  y_train_sm)
y_pred = ada.predict(X_test)
classification(y_test, y_pred, X_test, ada)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.91      0.93      2449
           1       0.26      0.39      0.31       199

    accuracy                           0.87      2648
   macro avg       0.61      0.65      0.62      2648
weighted avg       0.90      0.87      0.88      2648
```

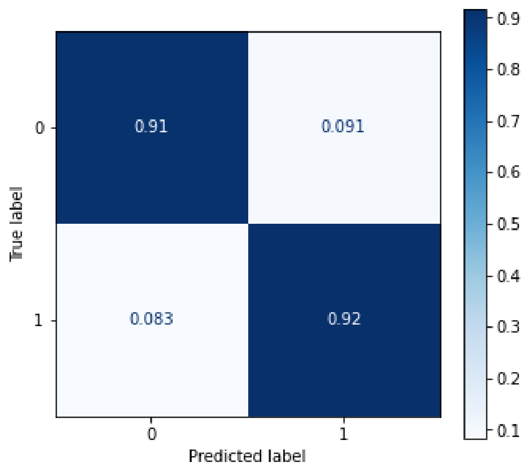


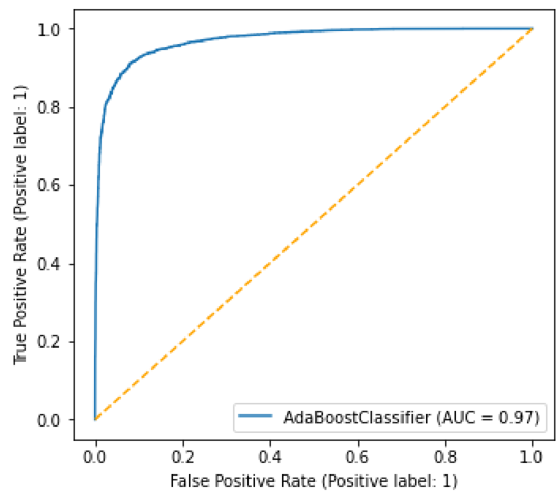


[85]:
```
#Evaluating the model performance for the training data
y_pred = ada.predict(X_train_sm)
classification(y_train_sm, y_pred, X_train_sm, ada)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support
```

```
           0       0.92      0.91      0.91      5701
           1       0.91      0.92      0.91      5701

    accuracy                           0.91     11402
   macro avg       0.91      0.91      0.91     11402
weighted avg       0.91      0.91      0.91     11402
```





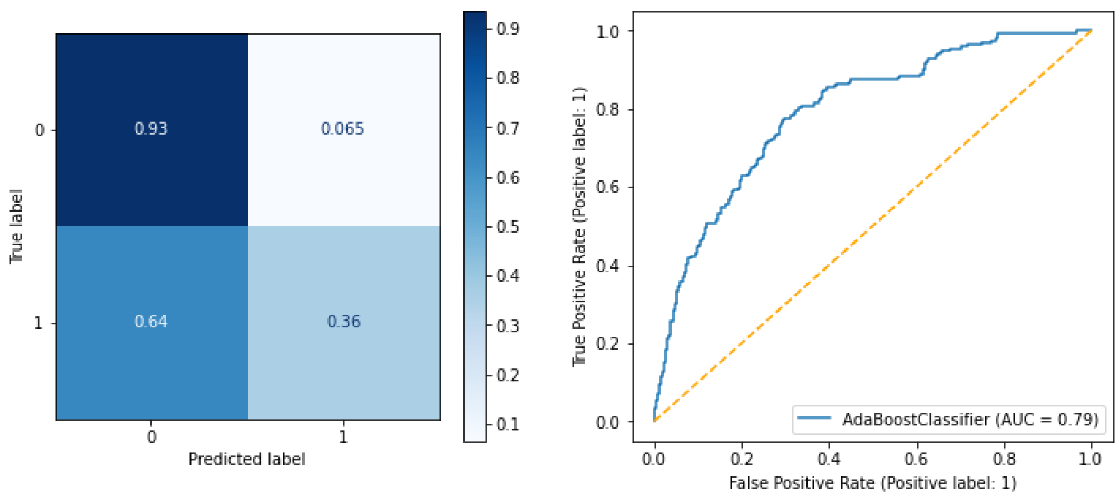
The model also might be overfitting, so we'll try grid search once again to get the best hyperparameters.

[86]:
```
# from sklearn.model_selection import GridSearchCV
# from sklearn.tree import DecisionTreeClassifier

# grid = {'n_estimators': [100, 200],
#         'estimator': [DecisionTreeClassifier(max_depth=1),␣
↪DecisionTreeClassifier(max_depth=2)],
#         'learning_rate': [0.01, 0.1, 1],
#        }

# gridsearch = GridSearchCV(estimator=ada, param_grid = grid, scoring='recall',␣
↪n_jobs=-1, verbose=2)

# gridsearch.fit(X_train_sm,  y_train_sm)
# gridsearch.best_params_
# # Results: {'estimator': DecisionTreeClassifier(max_depth=2), 'learning_rate':
↪1, 'n_estimators': 200}
```

[87]:
```
from sklearn.tree import DecisionTreeClassifier
ada_tuned = AdaBoostClassifier(estimator=DecisionTreeClassifier(max_depth=2),
                               learning_rate=1, n_estimators=200,
```

```
                                random_state=42
                               )

ada_tuned.fit(X_train_sm, y_train_sm)
y_pred = ada_tuned.predict(X_test)
classification(y_test, y_pred, X_test, ada_tuned)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.95      0.93      0.94      2449
           1       0.31      0.36      0.33       199

    accuracy                           0.89      2648
   macro avg       0.63      0.65      0.64      2648
weighted avg       0.90      0.89      0.89      2648
```



The recall score dropped rather than improving

[88]:
```
#appending the recall score to the results dataframe
df_results = add_results('AdaBoost', df_results)
df_results.head()
```

[88]:

	Model Name	Recall Score
0	Dummy Classifier	0.00
1	Random Forest	0.63
2	AdaBoost	0.36

0.1.5 Model #3 - LogisticRegressionCV

Since the Logistic Regression models are potentially sensitive to outliers and need scaled data we will need to process our data one more time to remove outliers and scale it. #### Removing Outliers

[89]:
```
#separating out the numerical columns for outlier removal
num_cols = ['acousticness', 'danceability', 'duration_ms', 'energy',␣
↪'instrumentalness',
            'liveness', 'loudness', 'speechiness', 'tempo', 'valence']
num_cols
```

[89]:
```
['acousticness',
 'danceability',
 'duration_ms',
 'energy',
 'instrumentalness',
 'liveness',
 'loudness',
 'speechiness',
 'tempo',
 'valence']
```

[90]:
```
#Concatenating the training and testing sets together for outlier removal
df_train = pd.concat([X_train, y_train], axis=1)
df_test = pd.concat([X_test, y_test], axis=1)
```

[91]:
```
#Outlier Removal with the IQR method

def find_outliers_IQR(data, return_limits = False):
    """Use Tukey's Method of outlier removal AKA InterQuartile-Range Rule
    and return boolean series where True indicates it is an outlier.
    - Calculates the range between the 75% and 25% quartiles
    - Outliers fall outside upper and lower limits, using a treshold of  1.
↪5*IQR the 75% and 25% quartiles.

    IQR Range Calculation:
        res = df.describe()
        IQR = res['75%'] -  res['25%']
        lower_limit = res['25%'] - 1.5*IQR
        upper_limit = res['75%'] + 1.5*IQR

    Args:
        data (Series, or ndarray): data to test for outliers.

    Returns:
        [boolean Series]: A True/False for each row use to slice outliers.

```

```
    Adapted from Flatiron School Phase #2 Py Files.
    URL = https://github.com/flatiron-school/Online-DS-FT-022221-Cohort-Notes/
↪blob/master/py_files/functions_SG.py

    """
    df_b=data
    res= df_b.describe()

    IQR = res['75%'] -  res['25%']
    lower_limit = res['25%'] - 1.5*IQR
    upper_limit = res['75%'] + 1.5*IQR

    if return_limits:
        return lower_limit, upper_limit

    else:
        idx_outs = (df_b>upper_limit) | (df_b<lower_limit)
        return idx_outs
```

[92]:

```
#finding and removing outliers based on X_train (df_train) to avoid data leakage

original_length_train = len(df_train)
original_length_test = len(df_test)

for col in num_cols:

    lower_limit, upper_limit = find_outliers_IQR(df_train[col],␣
↪return_limits=True)

    df_train = df_train[(df_train[col]>lower_limit) &␣
↪(df_train[col]<upper_limit)]
    df_test = df_test[(df_test[col]>lower_limit) & (df_test[col]<upper_limit)]

print(f'{original_length_train - len(df_train)} outliers removed from training␣
↪set')
print(f'{original_length_test - len(df_test)} outliers removed from test set')
```

```
2312 outliers removed from training set
1005 outliers removed from test set
```

[93]:

```
#Separating out the X and y values for training and test sets
y_train = df_train['is_popular']
X_train = df_train.drop('is_popular', axis=1)

y_test = df_test['is_popular']
X_test = df_test.drop('is_popular', axis=1)
```

Addressing Class Imbalance with SMOTENC

[94]:
```
y_train.value_counts(normalize=True)
```

[94]:
```
0    0.914382
1    0.085618
Name: is_popular, dtype: float64
```

Once again our data has a class imbalance issue so we will be using SMOTENC to address this.

[95]:
```
X_train.columns
```

[95]:
```
Index(['duration_ms', 'danceability', 'acousticness', 'energy',
       'instrumentalness', 'liveness', 'loudness', 'speechiness', 'tempo',
       'valence', 'hip hop', 'world', 'soukous', 'rumba congolaise',
       'azontobeats', 'hiplife', 'kwaito', 'jazz', 'afrobeats', 'pop', 'house',
       'afropop', 'xhosa', 'ndombolo', 'soul', 'zilizopendwa', 'key_1.0',
       'key_2.0', 'key_3.0', 'key_4.0', 'key_5.0', 'key_6.0', 'key_7.0',
       'key_8.0', 'key_9.0', 'key_10.0', 'key_11.0', 'mode_1.0',
       'time_signature_1.0', 'time_signature_3.0', 'time_signature_4.0',
       'time_signature_5.0'],
      dtype='object')
```

[96]:
```
cat_cols = list(range(10, len(X_train.columns)))
cat_cols
```

[96]:
```
[10,
 11,
 12,
 13,
 14,
 15,
 16,
 17,
 18,
 19,
 20,
 21,
 22,
 23,
 24,
 25,
 26,
 27,
 28,
 29,
 30,
 31,
```

```
32,
 33,
 34,
 35,
 36,
 37,
 38,
 39,
 40,
 41]
```

[97]:
```
from imblearn.over_sampling import SMOTENC
```

[98]:
```
sm = SMOTENC(categorical_features=cat_cols, random_state=42)

X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
y_train_sm.value_counts(normalize=True)
```

[98]:
```
0    0.5
1    0.5
Name: is_popular, dtype: float64
```

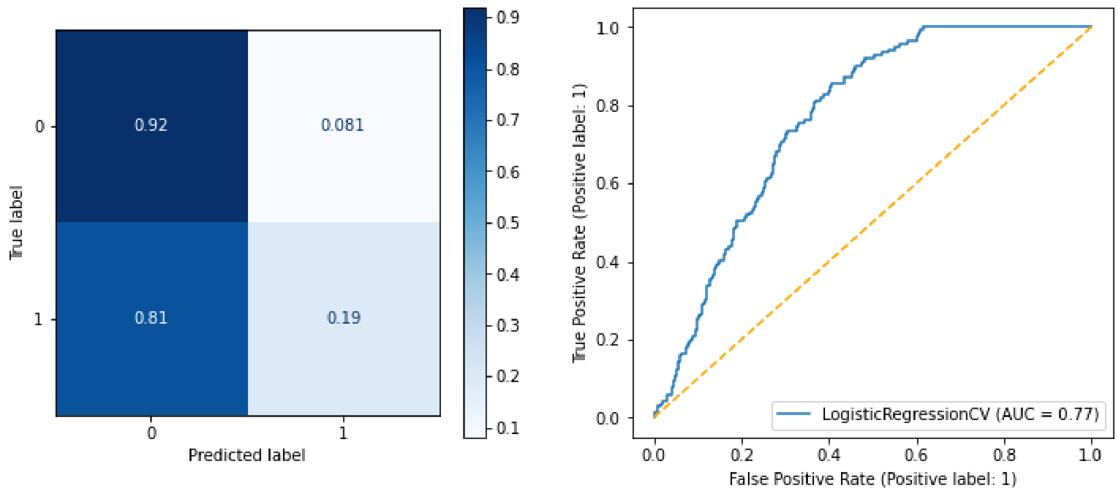
0.1.6 Scaling the Data

[99]:
```
#Using Standard Scaler to scale the smote'd data
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train_sm_sc = scaler.fit_transform(X_train_sm)
X_test_sc = scaler.transform(X_test)
```

[100]:
```
from sklearn.linear_model import LogisticRegressionCV
clf_logregcv = LogisticRegressionCV(cv=5, max_iter=500, random_state=42)
clf_logregcv.fit(X_train_sm_sc,  y_train_sm)
y_pred = clf_logregcv.predict(X_test_sc)
classification(y_test, y_pred, X_test_sc, clf_logregcv)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.93      0.92      0.92      1504
           1       0.18      0.19      0.19       139

    accuracy                           0.86      1643
   macro avg       0.55      0.56      0.55      1643
```

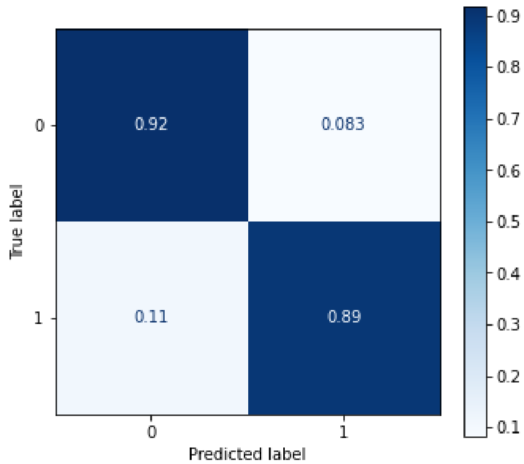
```
weighted avg       0.86      0.86      0.86      1643
```

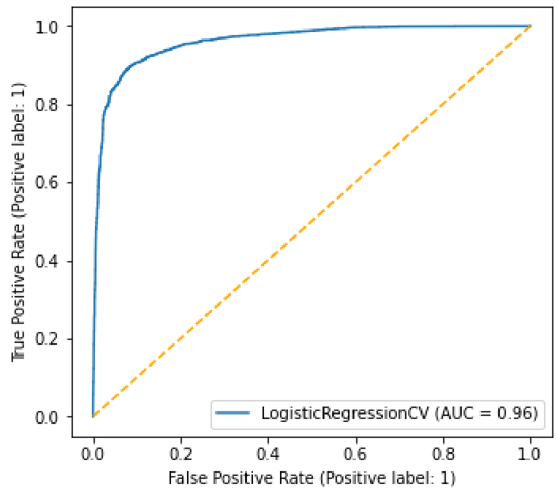


[101]:

```
#Evaluating the model performance for the training data
y_pred = clf_logregcv.predict(X_train_sm_sc)
classification(y_train_sm, y_pred, X_train_sm_sc, clf_logregcv)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      3535
           1       0.92      0.89      0.90      3535

    accuracy                           0.91      7070
   macro avg       0.91      0.91      0.91      7070
weighted avg       0.91      0.91      0.91      7070
```





Our model is once again overfitting to the training data and performing very well on it but the model's performance drops significantly when we test it with the test data. In order to address this, we can once again perform a grid search and try to tune the model.

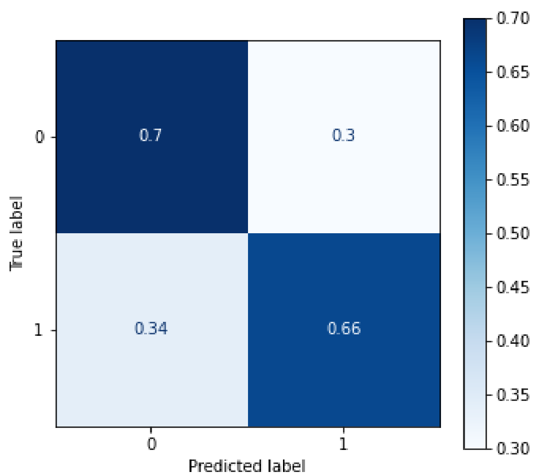
0.1.7 Hyperparameter Tuning

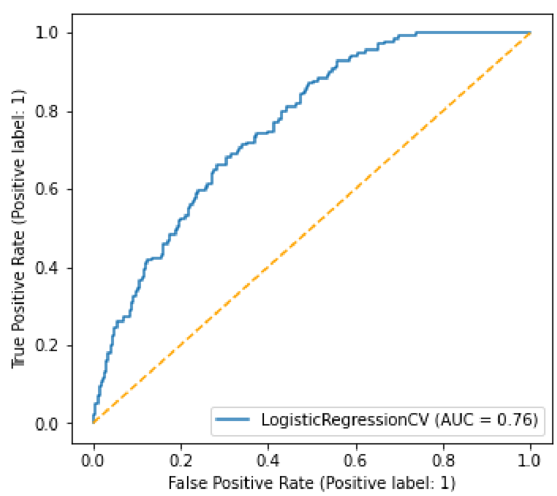
[102]:
```
# clf = LogisticRegressionCV(cv=5)
# grid = {'class_weight': ['balanced', None],
#         'penalty':['l1', 'l2'],
#         'solver':['liblinear'],
#         'Cs': [10, 1]
#        }

# gridsearch = GridSearchCV(estimator=clf, param_grid = grid, scoring='recall', n_jobs=-1, verbose=2)

# gridsearch.fit(X_train_sm_sc,  y_train_sm)
# gridsearch.best_params_
# # {'Cs': 1, 'class_weight': 'balanced', 'penalty': 'l2', 'solver': 'liblinear'}
```

The grid search returned 'l2' as the regularization method which is the Ridge regularization as well as a C value of 1. We will use these parameters on a new model to see if the recall score improves.

[103]:
```
clf_logregcv_tuned = LogisticRegressionCV(cv=5, class_weight='balanced', Cs=1,
                                          penalty='l2', solver='liblinear',
                                          max_iter=500, random_state=42)
clf_logregcv_tuned.fit(X_train_sm_sc,  y_train_sm)
y_pred = clf_logregcv_tuned.predict(X_test_sc)
classification(y_test, y_pred, X_test_sc, clf_logregcv_tuned)
```

```
CLASSIFICATION REPORT
------------------------------------------
              precision    recall  f1-score   support

           0       0.96      0.70      0.81      1504
           1       0.17      0.66      0.27       139

    accuracy                           0.70      1643
   macro avg       0.56      0.68      0.54      1643
weighted avg       0.89      0.70      0.76      1643
```





[104]:
```python
#appending the recall score to the results dataframe
df_results = add_results('Logistic Regression', df_results)
df_results.head()
```

[104]:

	Model Name	Recall Score
0	Dummy Classifier	0.00
1	Random Forest	0.63
2	AdaBoost	0.36
3	Logistic Regression	0.66

[ ]:

0.2 INTERPRETATION

Now that we have 3 tuned models, we can analyze which attributes they used in predicting whether a song was going to be popular or not and interpret these values. For this we will be looking at feature importances of each model and comparing them against each other to see if we can see any common threads between the models.

0.2.1 Parsing Feature Importances to Dataframes

Random Forest

[105]:
```
#accessing feature importance values of the tuned random forest model and␣
↪sorting them
rf_importances_df = pd.Series(clf_rf_tuned.feature_importances_, index=X_train.
↪columns).sort_values(ascending=False)
#parsing the series to a dataframe
rf_importances_df = rf_importances_df.reset_index()
rf_importances_df.columns = ['RF-Attribute', 'RF-Importance']
rf_importances_df
```

[105]:

	RF-Attribute	RF-Importance
0	afrobeats	0.205459
1	pop	0.106933
2	duration_ms	0.085142
3	soukous	0.046265
4	afropop	0.045079
5	danceability	0.044942
6	rumba congolaise	0.039885
7	jazz	0.039506
8	instrumentalness	0.038272
9	speechiness	0.036514
10	energy	0.030468
11	acousticness	0.028936
12	world	0.020007
13	valence	0.019567
14	hip hop	0.019411
15	hiplife	0.018923
16	kwaito	0.018167
17	azontobeats	0.017165
18	soul	0.016123
19	zilizopendwa	0.014761
20	key_1.0	0.013894
21	ndombolo	0.009331
22	xhosa	0.008818
23	loudness	0.008790
24	time_signature_4.0	0.008678
25	key_10.0	0.008485
26	key_7.0	0.007009
27	key_9.0	0.006802
28	house	0.005517
29	key_11.0	0.004749
30	key_5.0	0.004211
31	liveness	0.004178
32	time_signature_3.0	0.004056
33	tempo	0.003408

```
34            key_8.0        0.003345
35   time_signature_5.0        0.002796
36            key_6.0        0.001970
37           mode_1.0        0.001772
38            key_2.0        0.000534
39            key_4.0        0.000075
40            key_3.0        0.000059
41   time_signature_1.0        0.000000
```

AdaBoost

[106]:
```
#parsing feature importances to a series and sorting
ada_importances_df = pd.Series(ada_tuned.feature_importances_, index=X_train.
 ↪columns).sort_values(ascending=False)
#parsing the series to a dataframe
ada_importances_df = ada_importances_df.reset_index()
ada_importances_df.columns=['Ada-Attribute', 'Ada-Importance']
ada_importances_df
```

[106]:

	Ada-Attribute	Ada-Importance
0	duration_ms	0.147835
1	valence	0.084881
2	liveness	0.081620
3	tempo	0.081185
4	loudness	0.072786
5	speechiness	0.071792
6	acousticness	0.070484
7	energy	0.069103
8	danceability	0.056117
9	instrumentalness	0.049725
10	afrobeats	0.023186
11	key_7.0	0.015003
12	afropop	0.012635
13	key_1.0	0.011317
14	key_5.0	0.010584
15	key_9.0	0.010561
16	key_8.0	0.010452
17	key_11.0	0.010087
18	hip hop	0.009670
19	soul	0.009046
20	key_10.0	0.008163
21	key_6.0	0.007992
22	key_2.0	0.007047
23	hiplife	0.006151
24	key_4.0	0.006130
25	mode_1.0	0.005632
26	ndombolo	0.005503

```
27                 pop        0.005248
28         azontobeats        0.004655
29              kwaito        0.004610
30               world        0.004484
31             key_3.0        0.004157
32  time_signature_4.0        0.004043
33                jazz        0.004014
34               xhosa        0.003195
35             soukous        0.002842
36    rumba congolaise        0.002570
37         zilizopendwa       0.002297
38               house        0.002277
39  time_signature_5.0        0.000921
40  time_signature_1.0        0.000000
41  time_signature_3.0        0.000000
```

[107]:
```python
#accessing feature importance values of the tuned logistic regression model and␣
↪sorting them
logregcv_importances_df = pd.Series(clf_logregcv_tuned.coef_[0], index=X_train.
↪columns).sort_values(ascending=False)
#parsing the series to a dataframe
logregcv_importances_df = logregcv_importances_df.reset_index()
logregcv_importances_df.columns = ['LogReg-Attribute', 'LogReg-Importance']
logregcv_importances_df
```

[107]:
```
      LogReg-Attribute  LogReg-Importance
0            afrobeats           0.094527
1                  pop           0.082875
2              hip hop           0.068160
3         danceability           0.058378
4   time_signature_4.0           0.038778
5     instrumentalness           0.009620
6          speechiness           0.005456
7             loudness           0.002386
8          azontobeats          -0.010649
9   time_signature_1.0          -0.010891
10            liveness          -0.014016
11               tempo          -0.018184
12             valence          -0.021372
13               house          -0.023658
14              energy          -0.023889
15        acousticness          -0.024297
16  time_signature_5.0          -0.029243
17             key_3.0          -0.030541
18            mode_1.0          -0.030810
19               xhosa          -0.033512
20  time_signature_3.0          -0.034166
```

```
21                key_4.0           -0.034281
22               ndombolo           -0.036121
23                key_2.0           -0.037580
24                 kwaito           -0.038962
25               key_10.0           -0.042670
26                key_6.0           -0.043374
27                key_8.0           -0.044082
28                  world           -0.045208
29                   jazz           -0.046798
30           zilizopendwa           -0.047544
31                key_7.0           -0.047808
32                key_5.0           -0.048620
33            duration_ms           -0.050616
34                soukous           -0.051305
35                hiplife           -0.051495
36                key_9.0           -0.052331
37                key_1.0           -0.052754
38               key_11.0           -0.054207
39                afropop           -0.055665
40                   soul           -0.055759
41       rumba congolaise           -0.055819
```

[108]:

```python
#Concatenating feature importances into a single dataframe
importances_df = pd.concat([rf_importances_df, ada_importances_df,␣
 ↪logregcv_importances_df], axis=1)
importances_df
```

[108]:

```
          RF-Attribute  RF-Importance       Ada-Attribute  Ada-Importance  \
0            afrobeats       0.205459         duration_ms        0.147835
1                  pop       0.106933             valence        0.084881
2          duration_ms       0.085142            liveness        0.081620
3              soukous       0.046265               tempo        0.081185
4              afropop       0.045079            loudness        0.072786
5         danceability       0.044942         speechiness        0.071792
6     rumba congolaise       0.039885        acousticness        0.070484
7                 jazz       0.039506              energy        0.069103
8     instrumentalness       0.038272        danceability        0.056117
9          speechiness       0.036514    instrumentalness        0.049725
10              energy       0.030468           afrobeats        0.023186
11        acousticness       0.028936             key_7.0        0.015003
12               world       0.020007             afropop        0.012635
13             valence       0.019567             key_1.0        0.011317
14             hip hop       0.019411             key_5.0        0.010584
15             hiplife       0.018923             key_9.0        0.010561
16              kwaito       0.018167             key_8.0        0.010452
17         azontobeats       0.017165            key_11.0        0.010087
18                soul       0.016123             hip hop        0.009670
```

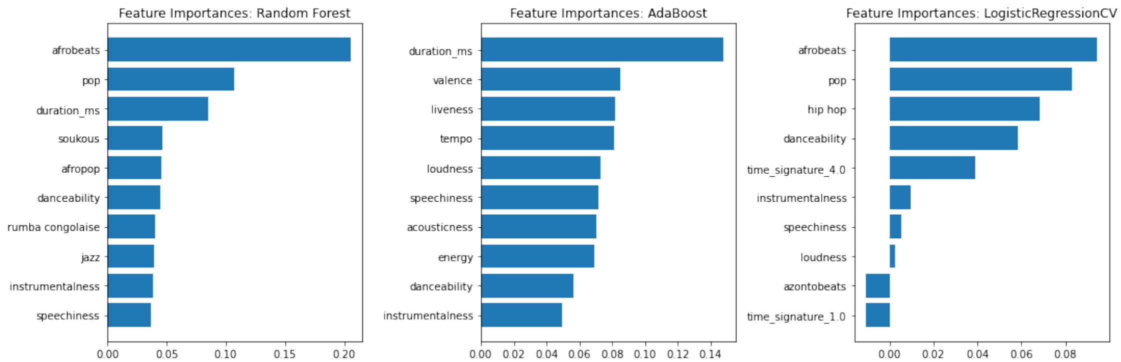
19	zilizopendwa	0.014761	soul	0.009046
20	key_1.0	0.013894	key_10.0	0.008163
21	ndombolo	0.009331	key_6.0	0.007992
22	xhosa	0.008818	key_2.0	0.007047
23	loudness	0.008790	hiplife	0.006151
24	time_signature_4.0	0.008678	key_4.0	0.006130
25	key_10.0	0.008485	mode_1.0	0.005632
26	key_7.0	0.007009	ndombolo	0.005503
27	key_9.0	0.006802	pop	0.005248
28	house	0.005517	azontobeats	0.004655
29	key_11.0	0.004749	kwaito	0.004610
30	key_5.0	0.004211	world	0.004484
31	liveness	0.004178	key_3.0	0.004157
32	time_signature_3.0	0.004056	time_signature_4.0	0.004043
33	tempo	0.003408	jazz	0.004014
34	key_8.0	0.003345	xhosa	0.003195
35	time_signature_5.0	0.002796	soukous	0.002842
36	key_6.0	0.001970	rumba congolaise	0.002570
37	mode_1.0	0.001772	zilizopendwa	0.002297
38	key_2.0	0.000534	house	0.002277
39	key_4.0	0.000075	time_signature_5.0	0.000921
40	key_3.0	0.000059	time_signature_1.0	0.000000
41	time_signature_1.0	0.000000	time_signature_3.0	0.000000

	LogReg-Attribute	LogReg-Importance
0	afrobeats	0.094527
1	pop	0.082875
2	hip hop	0.068160
3	danceability	0.058378
4	time_signature_4.0	0.038778
5	instrumentalness	0.009620
6	speechiness	0.005456
7	loudness	0.002386
8	azontobeats	-0.010649
9	time_signature_1.0	-0.010891
10	liveness	-0.014016
11	tempo	-0.018184
12	valence	-0.021372
13	house	-0.023658
14	energy	-0.023889
15	acousticness	-0.024297
16	time_signature_5.0	-0.029243
17	key_3.0	-0.030541
18	mode_1.0	-0.030810
19	xhosa	-0.033512
20	time_signature_3.0	-0.034166
21	key_4.0	-0.034281

```
22           ndombolo          -0.036121
23            key_2.0          -0.037580
24             kwaito          -0.038962
25           key_10.0          -0.042670
26            key_6.0          -0.043374
27            key_8.0          -0.044082
28              world          -0.045208
29               jazz          -0.046798
30       zilizopendwa          -0.047544
31            key_7.0          -0.047808
32            key_5.0          -0.048620
33        duration_ms          -0.050616
34            soukous          -0.051305
35            hiplife          -0.051495
36            key_9.0          -0.052331
37            key_1.0          -0.052754
38           key_11.0          -0.054207
39            afropop          -0.055665
40               soul          -0.055759
41    rumba congolaise          -0.055819
```

0.2.2 Feature Importance Comparison

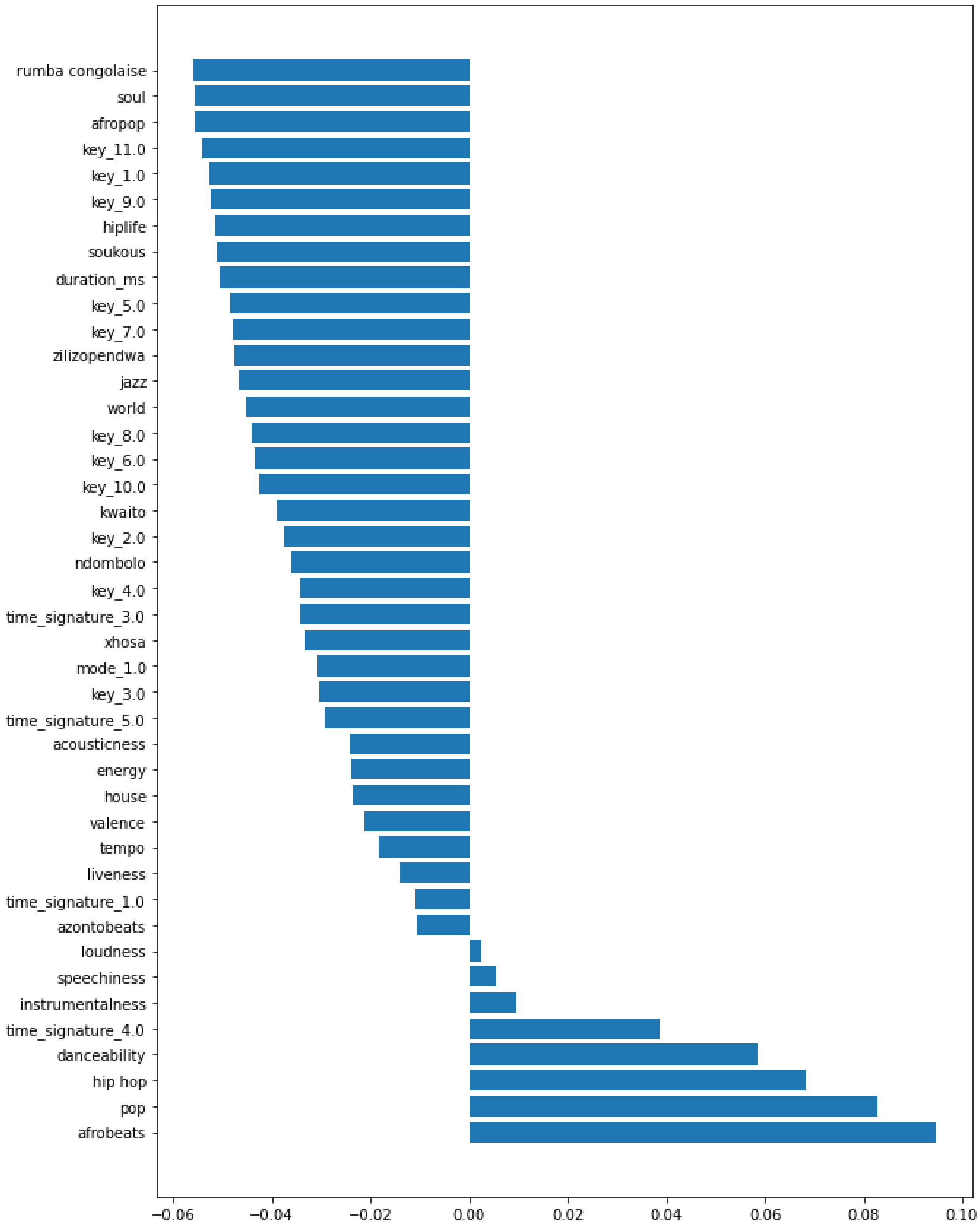
[109]:
```
#plotting feature importances for all models for comparison

fig, ax = plt.subplots(ncols=3, figsize=(15,5))

rf_importances_df = rf_importances_df.sort_values(by='RF-Importance',␣
 ↪ascending=True).tail(10)
ax[0].barh(rf_importances_df['RF-Attribute'],␣
 ↪rf_importances_df['RF-Importance'])
ax[0].set_title('Feature Importances: Random Forest')

ada_importances_df = ada_importances_df.sort_values(by='Ada-Importance',␣
 ↪ascending=True).tail(10)
ax[1].barh(ada_importances_df['Ada-Attribute'],␣
 ↪ada_importances_df['Ada-Importance'])
ax[1].set_title('Feature Importances: AdaBoost')

logregcv_importances_df = logregcv_importances_df.
 ↪sort_values(by='LogReg-Importance', ascending=True).tail(10)
ax[2].barh(logregcv_importances_df['LogReg-Attribute'],␣
 ↪logregcv_importances_df['LogReg-Importance'])
ax[2].set_title('Feature Importances: LogisticRegressionCV')
plt.tight_layout()
```



Among the 3 models we built we can see that Genre of a song has the highest effect on the popularity of a song for Random Forest and Logistic Regression models, while the track features like duration, valence, liveness, etc has the highest effect for the AdaBoost model. On the first and last models, a song having Afrobeats as its genre had the most impact on its popularity. This makes sense since Afrobeat songs by nature are considered popular especially in sub saharan Africa. Among the rest of the features shown above, danceabilty, speechiness and instrumentalness tends to have quiet a significant effects on all 3 models. Next, we can inspect the full gamut of the feature importances for Logistic Regression for reference.

[110]:
```
logregcv_importances_df = pd.Series(clf_logregcv_tuned.coef_[0], index=X_train.
 ↪columns).sort_values(ascending=False)
#parsing the series to a dataframe
logregcv_importances_df = logregcv_importances_df.reset_index()
logregcv_importances_df.columns = ['Attribute', 'Importance']

fig, ax = plt.subplots(figsize=(10,15))
ax.barh(logregcv_importances_df['Attribute'],␣
 ↪logregcv_importances_df['Importance'])
```

[110]:
```
<BarContainer object of 42 artists>
```



We can see here that while certain features like ‘afrobeats’, ‘pop’, and ‘danceability’ positively affected the prediction, other features such as ‘rumba congolaise’, ‘soul’ and ‘key_11 (or Key_B)’ negatively affected it. Next we can dive into our processed dataframe and explore some of these attributes for popular and unpopular songs to come to conclusions.

0.3 Exploring Track Features and Popularity

In this section, we examine how track features such as `'danceability'`, `'speechiness'`, and `'instrumentalness'` influence popularity, independent of genre classifications with reference to the definitions provided in the Spotify documentation.

0.3.1 Danceability

Danceability describes how suitable a track is for dancing based on a combination of musical elements including tempo, rhythm stability, beat strength, and overall regularity. A value of 0.0 is least danceable and 1.0 is most danceable.

[111]:
```
#separating popular and unpopular songs to two dfs
popular_songs_df = df_ohe[df_ohe['is_popular'] == 1]
unpopular_songs_df = df_ohe[df_ohe['is_popular']==0]
```

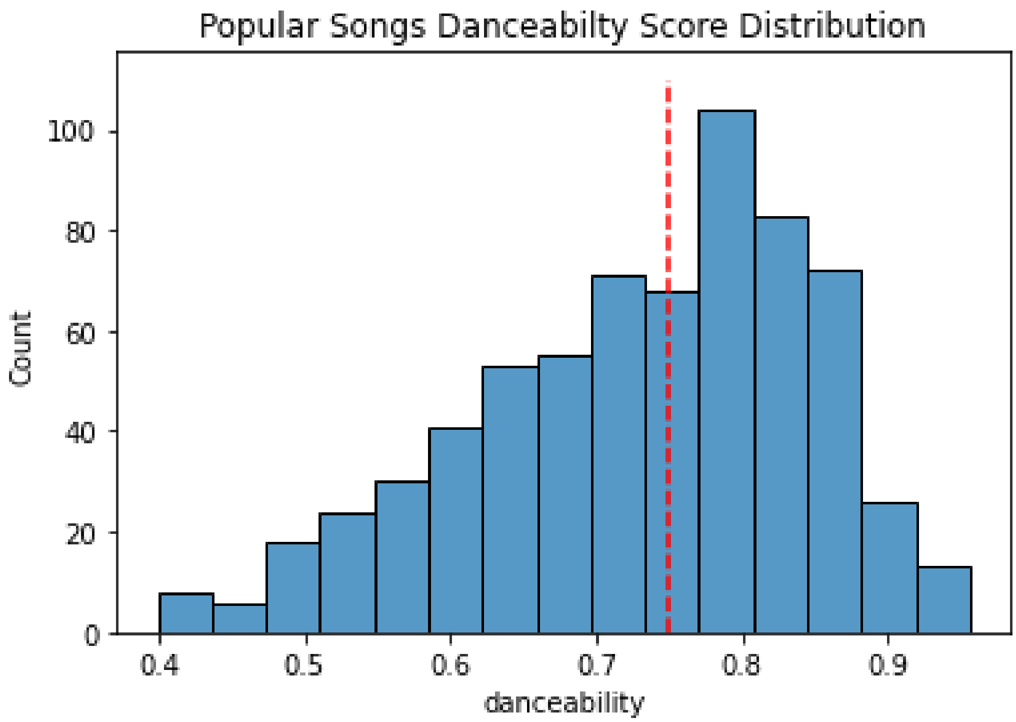
[112]:
```
#removing outliers from danceability scores and separating them to Series for␣
↪popular and unpopular songs
popular_dance_clean =␣
↪popular_songs_df[find_outliers_IQR(popular_songs_df['danceability'])==False]
print(popular_dance_clean['danceability'].describe())

unpopular_dance_clean =␣
↪unpopular_songs_df[find_outliers_IQR(unpopular_songs_df['danceability'])==False]
print(unpopular_dance_clean['danceability'].describe())
```

```
count    672.000000
mean       0.729509
std        0.115845
min        0.399000
25%        0.651750
50%        0.750000
75%        0.819250
max        0.956000
Name: danceability, dtype: float64
count    8134.000000
mean        0.656625
std         0.141746
min         0.231000
25%         0.550000
50%         0.672000
75%         0.767000
max         0.985000
Name: danceability, dtype: float64
```

[113]:
```
sns.histplot(data=popular_dance_clean, x='danceability', bins='auto')
plt.title('Popular Songs Danceabilty Score Distribution')
```

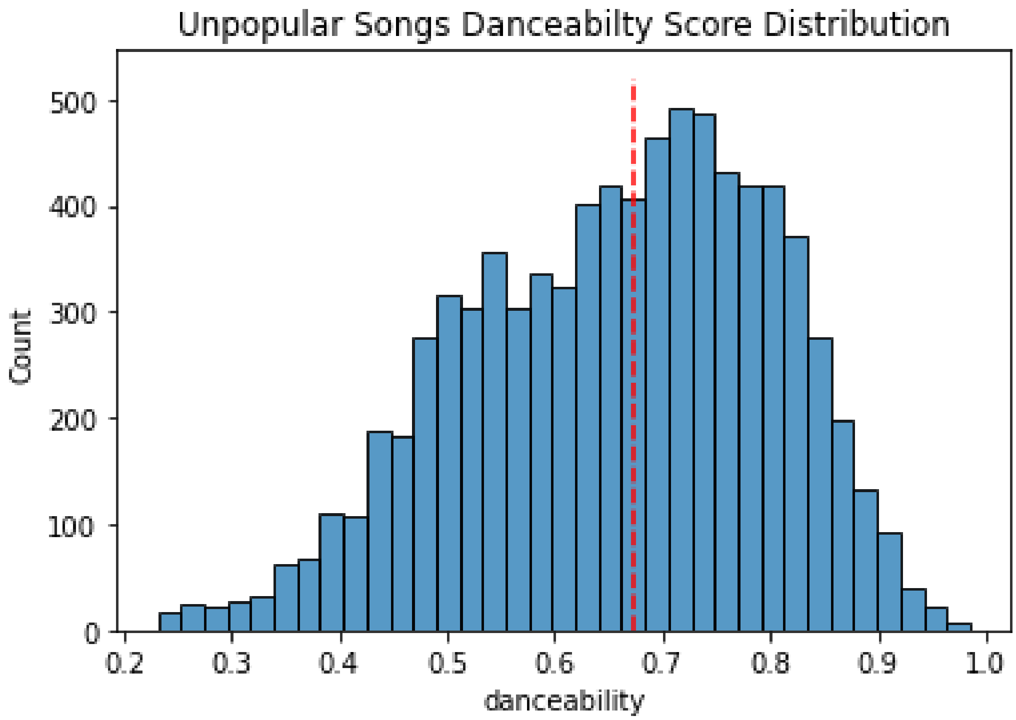
```
plt.vlines(x=popular_dance_clean['danceability'].median(), ymin=0, ymax=110,␣
  ↪color='red', ls='--')
```

[113]: <matplotlib.collections.LineCollection at 0x19e4aafa610>

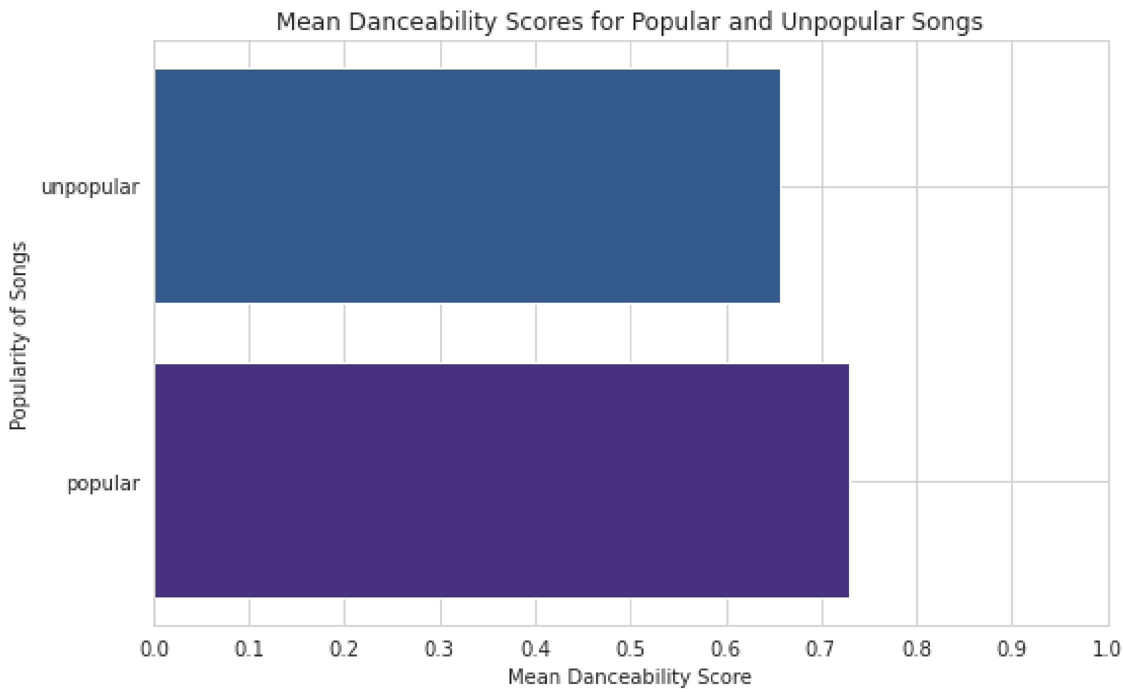


[114]:
```
sns.histplot(data=unpopular_dance_clean, x='danceability', bins='auto')
plt.title('Unpopular Songs Danceabilty Score Distribution')
plt.vlines(x=unpopular_dance_clean['danceability'].median(), ymin=0, ymax=520,␣
  ↪color='red', ls='--')
```

[114]: <matplotlib.collections.LineCollection at 0x19e4ab9be20>



[115]:
```
#storing mean danceability scores in dict
mean_danceability = {'popular': popular_dance_clean['danceability'].mean(),
                     'unpopular': unpopular_dance_clean['danceability'].mean()}

#visualizing mean scores
with sns.axes_style("whitegrid"):
    fig, ax = plt.subplots(figsize=(8,5))
    ax.barh(y=list(mean_danceability.keys()),
            width=list(mean_danceability.values()),
            color=[sns.color_palette('viridis')[0],sns.
 ↪color_palette('viridis')[1]])
    ax.set_xlim(0, 1)
    ax.set_xticks(np.arange(0,1.1,0.1))
    ax.set_ylabel('Popularity of Songs')
    ax.set_xlabel('Mean Danceability Score')
    ax.set_title('Mean Danceability Scores for Popular and Unpopular Songs')
    plt.tight_layout()
```



Above, it is clear that the popular songs tended to have a higher danceability score compared to unpopular songs.

0.3.2 Speechiness

Speechiness detects the presence of spoken words in a track. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), the closer to 1.0 the attribute value. Values above 0.66 describe tracks that are probably made entirely of spoken words. Values between 0.33 and 0.66 describe tracks that may contain both music and speech, either in sections or layered, including such cases as rap music. Values below 0.33 most likely represent music and other non-speech-like tracks.

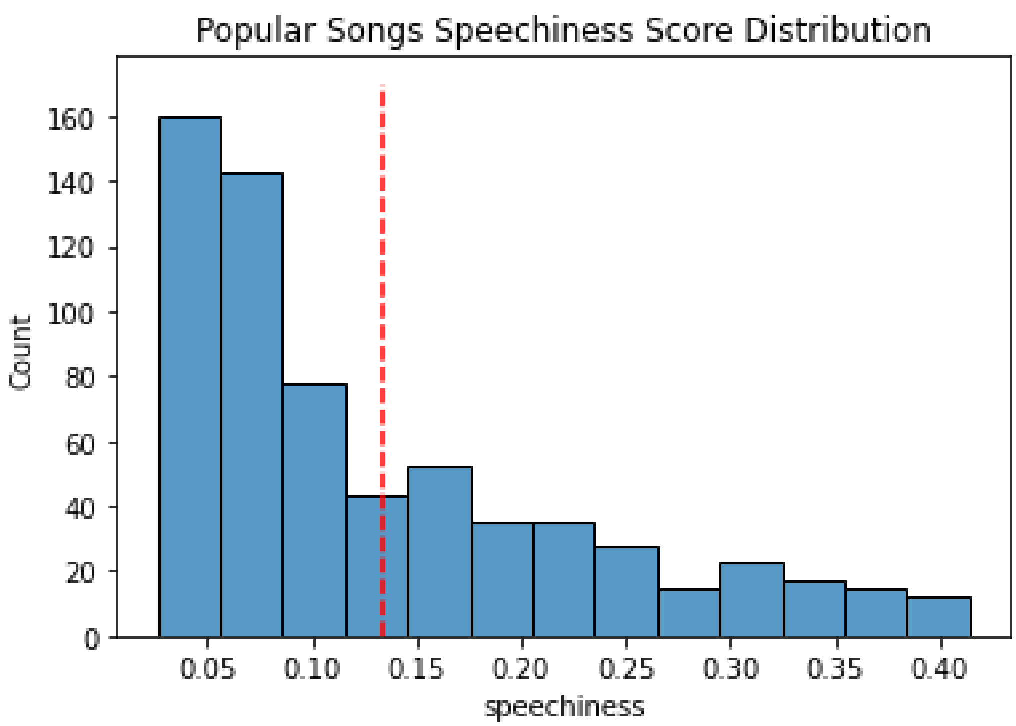
[116]:
```
#removing outliers from danceability scores and separating them to Series for␣
↪popular and unpopular songs
popular_speechiness_clean =␣
↪popular_songs_df[find_outliers_IQR(popular_songs_df['speechiness'])==False]
print(popular_speechiness_clean['speechiness'].describe())

unpopular_speechiness_clean =␣
↪unpopular_songs_df[find_outliers_IQR(unpopular_songs_df['speechiness'])==False]
print(unpopular_speechiness_clean['speechiness'].describe())
```

```
count    656.000000
mean       0.133307
std        0.097773
min        0.026100
```

```
25%        0.056700
50%        0.092850
75%        0.188500
max        0.414000
Name: speechiness, dtype: float64
count    7573.000000
mean        0.102046
std         0.079439
min         0.000000
25%         0.044700
50%         0.067100
75%         0.135000
max         0.348000
Name: speechiness, dtype: float64
```

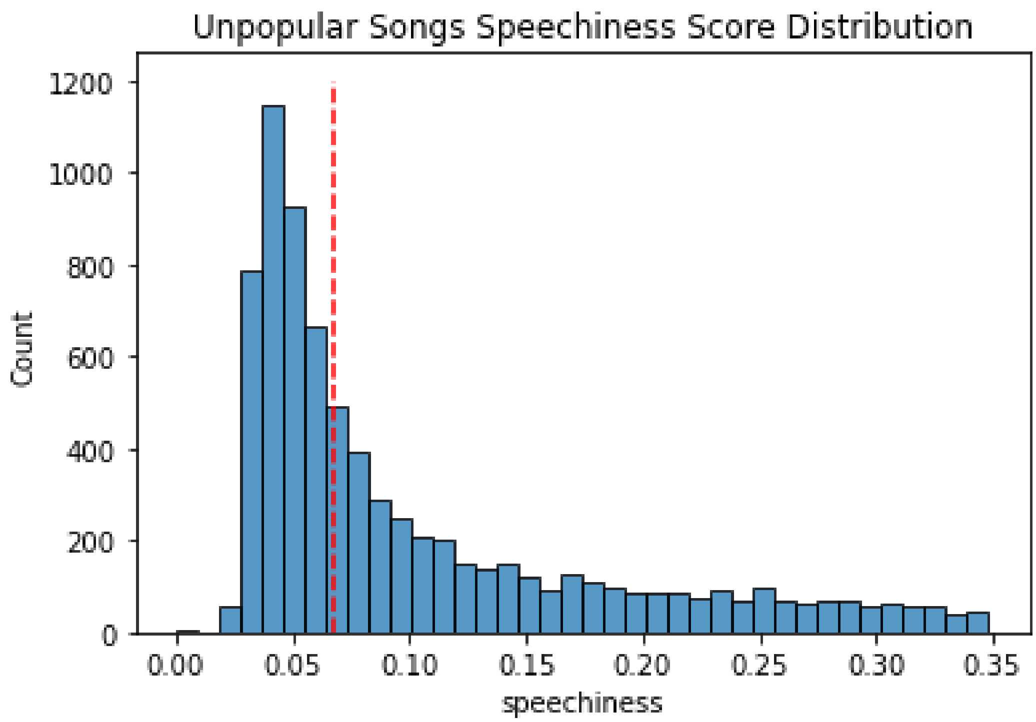
[117]:
```
sns.histplot(data = popular_speechiness_clean, x='speechiness', bins='auto')
plt.title('Popular Songs Speechiness Score Distribution')
plt.vlines(x=popular_speechiness_clean['speechiness'].mean(), ymin=0, ymax=170,␣
 ↪color='red', ls='--')
```

[117]: <matplotlib.collections.LineCollection at 0x19e4ccf01c0>



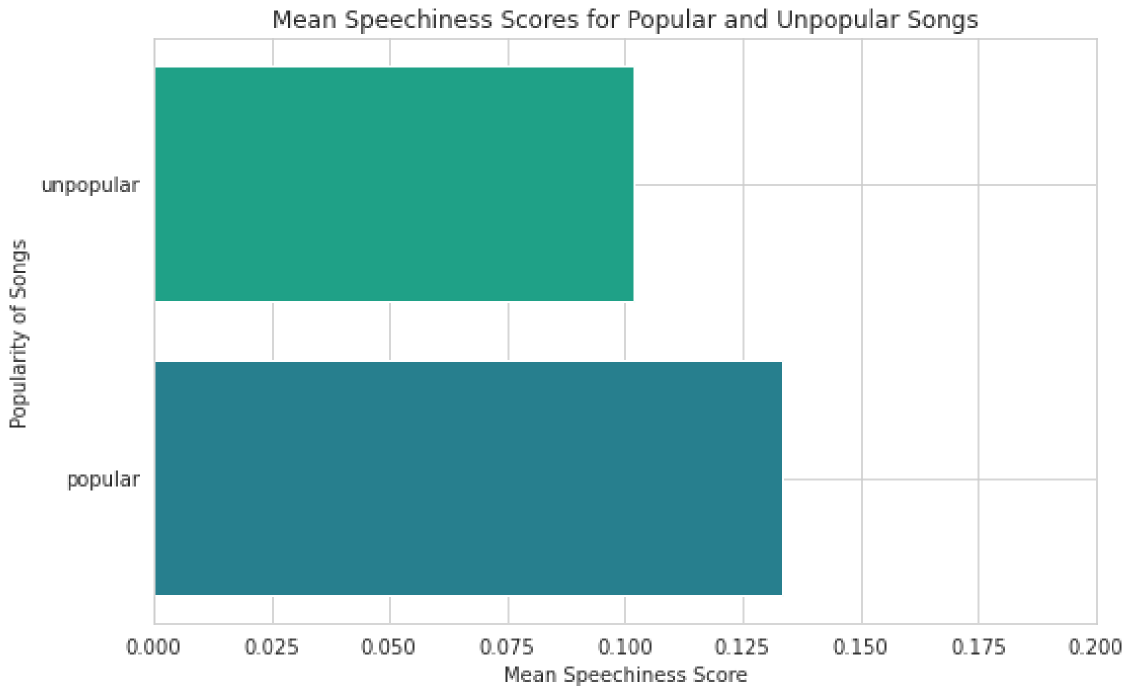
[118]:
```
sns.histplot(data=unpopular_speechiness_clean, x='speechiness', bins='auto')
plt.title('Unpopular Songs Speechiness Score Distribution')
plt.vlines(x=unpopular_speechiness_clean['speechiness'].median(), ymin=0,␣
  ↪ymax=1200, color='red', ls='--', label='median')
```

[118]:
```
<matplotlib.collections.LineCollection at 0x19e4d8a2f40>
```



[119]:
```
#storing mean acousticness scores in dict
mean_speechiness = {'popular': popular_speechiness_clean['speechiness'].mean(),
                    'unpopular': unpopular_speechiness_clean['speechiness'].
  ↪mean()
                   }

#visualizing mean scores
with sns.axes_style("whitegrid"):
    fig, ax = plt.subplots(figsize=(8,5))
    ax.barh(y=list(mean_speechiness.keys()),
            width=list(mean_speechiness.values()),
            color=[sns.color_palette('viridis')[2],sns.
  ↪color_palette('viridis')[3]])
    ax.set_xlim(0, 0.2)
    ax.set_ylabel('Popularity of Songs')
    ax.set_xlabel('Mean Speechiness Score')
```

```
    ax.set_title('Mean Speechiness Scores for Popular and Unpopular Songs')
    plt.tight_layout()
```



Similar to danceability scores we see that the popular songs tends to have a higher speechiness score.

0.3.3 Instrumentalness

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.

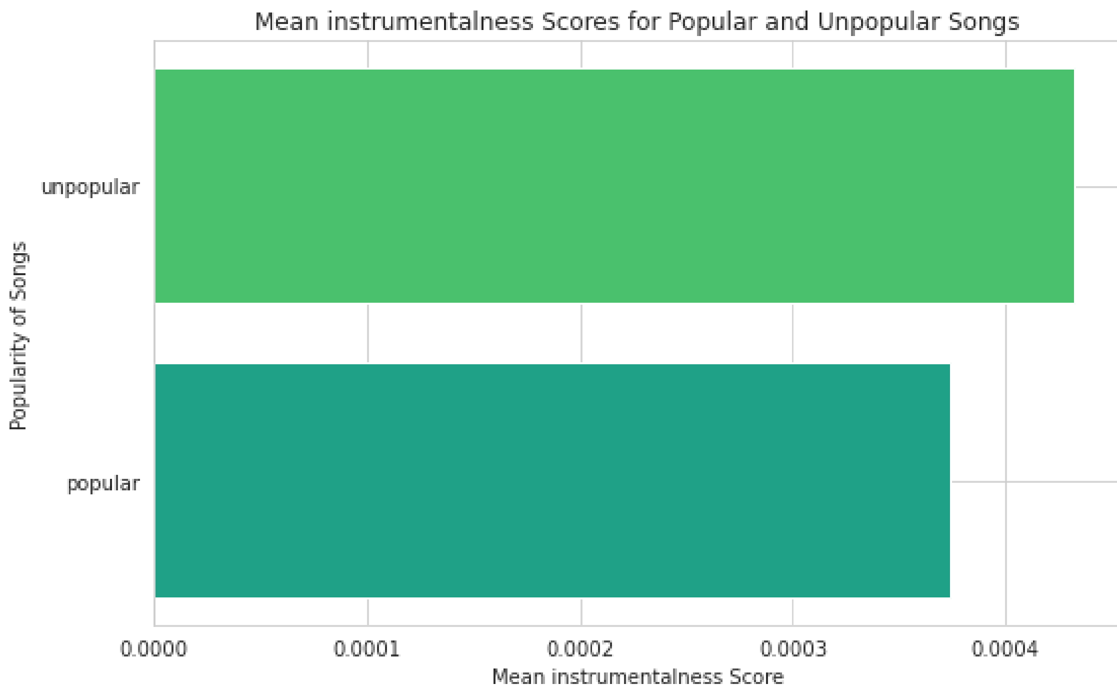
[120]:
```
#removing outliers from instrumentalness scores and separating them to Series␣
 ↪for popular and unpopular songs
popular_instrumentalness_clean =␣
 ↪popular_songs_df[find_outliers_IQR(popular_songs_df['instrumentalness'])==False]
print(popular_instrumentalness_clean['instrumentalness'].describe())

unpopular_instrumentalness_clean =␣
 ↪unpopular_songs_df[find_outliers_IQR(unpopular_songs_df['instrumentalness'])==False]
print(unpopular_instrumentalness_clean['instrumentalness'].describe())
```

```
count    530.000000
mean       0.000373
```

```
std        0.001009
min        0.000000
25%        0.000000
50%        0.000003
75%        0.000112
max        0.006770
Name: instrumentalness, dtype: float64
count    6410.000000
mean        0.000432
std         0.001290
min         0.000000
25%         0.000000
50%         0.000001
75%         0.000077
max         0.008560
Name: instrumentalness, dtype: float64
```

[121]:

```
#storing mean instrumentalness scores in dict
mean_instrumentalness = {'popular':
 ↪popular_instrumentalness_clean['instrumentalness'].mean(),
                         'unpopular':
 ↪unpopular_instrumentalness_clean['instrumentalness'].mean()
                        }
#visualizing mean scores
with sns.axes_style("whitegrid"):
    fig, ax = plt.subplots(figsize=(8,5))
    ax.barh(y=list(mean_instrumentalness.keys()),
            width=list(mean_instrumentalness.values()),
            color=[sns.color_palette('viridis')[3],sns.
 ↪color_palette('viridis')[4]])
    ax.set_ylabel('Popularity of Songs')
    ax.set_xlabel('Mean instrumentalness Score')
    ax.set_title('Mean instrumentalness Scores for Popular and Unpopular Songs')
    plt.tight_layout()
```



As can be seen above, the popular songs tends to be more vocal (low instrumentalness score) compare to unpopular songs.

0.4 Prediction and Evaluation

In this section, I employ each model to predict the popularity of songs and assess their performance on a new dataset. This dataset comprises track features obtained from Spotify for songs by artists not included in either the training or test data. By utilizing this unseen dataset, I can evaluate how well the models perform on entirely new data that was not previously encountered during the training or testing phases.

[122]:
```
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
import pandas as pd
from credentials import SPOTIPY_CLIENT_ID, SPOTIPY_CLIENT_SECRET

client_credentials_manager = SpotifyClientCredentials(SPOTIPY_CLIENT_ID,␣
  ↪SPOTIPY_CLIENT_SECRET)
spotify = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
```

Get New Set of Data

[123]:
```
# The code below (modified), used to get track features and properties, was␣
  ↪adapted from
# https://www.kaggle.com/code/worlaalex/
  ↪top-50-afrobeats-data-extraction-from-spotify
```

```python
def TrackFeatures(track_id):
    meta = spotify.track(track_id)
    artist = spotify.artist(meta["artists"][0]["external_urls"]["spotify"])

    features = spotify.audio_features(track_id)
    genres = artist["genres"]
    # metadata
    track_name = meta['name']
    album_name = meta['album']['name']
    artist_name = meta['album']['artists'][0]['name']
    release_date = meta['album']['release_date']
    duration_ms = meta['duration_ms']
    popularity = meta['popularity']

    # specific feartures
    if features[0]:
        acousticness = features[0]['acousticness']
        danceability = features[0]['danceability']
        energy = features[0]['energy']
        instrumentalness = features[0]['instrumentalness']
        liveness = features[0]['liveness']
        loudness = features[0]['loudness']
        speechiness = features[0]['speechiness']
        tempo = features[0]['tempo']
        time_signature = features[0]['time_signature']
        key = features[0]['key']
        mode = features[0]['mode']
        valence = features[0]['valence']

        track = [track_name, track_id, ",".join(genres), album_name,␣
 ↪artist_name, release_date, duration_ms,
                 popularity, danceability, key, acousticness, mode, energy,␣
 ↪instrumentalness, liveness,
                 loudness, speechiness, tempo, time_signature, valence,
                ]
    else:
        track = [np.nan] * 20
    return track
```

[124]:
```python
def get_features(track_ids):
    if isinstance(track_ids, list):
        tracks = [TrackFeatures(track_id) for track_id in track_ids]
        columns = ['track_name', 'track_id', 'genre', 'album_name',␣
 ↪'artist_name', 'release_date', 'duration_ms',
                   'popularity', 'danceability', 'key', 'acousticness', 'mode',␣
 ↪'energy', 'instrumentalness',
```

```
                   'liveness', 'loudness', 'speechiness', 'tempo',␣
↪'time_signature', 'valence',
                  ]
        df = pd.DataFrame(tracks, columns=columns)
        return df

    else:
        print("Track id must be surplied as a list")
```

[125]:
```
def predict(df, model='logreg'):
    import re
    df_new = df.dropna()
    df_new['key'] = df_new['key'].astype('float')
    df_new['mode'] = df_new['mode'].astype('float')
    df_new['time_signature'] = df_new['time_signature'].astype('float')

    # Replace all 'afrobeat' with 'afrobeats'
    pattern = r'\bafrobeat\b'
    df['genre'] = df['genre'].apply(lambda x: re.sub(pattern, 'afrobeats', x))

    # Replace 'azonto' and 'azotobeat' with 'azontobeats'
    pattern = r'(\bazonto\b)|(\bazontobeat\b)'
    df_new['genre'] = df_new['genre'].apply(lambda x: re.sub(pattern,␣
↪'azontobeats', x))

    #creating columns for each genre in the new_genres list
    for genre in new_genres:
        pattern = re.compile(fr'\b{genre}\b')
        df_new[genre] = (df_new['genre'].apply(lambda x: bool(pattern.
↪search(x)))).astype('int')

    #removing the redundant genre column
    df_new.drop('genre', axis=1, inplace=True)

    #dropping 'artist_name', 'track_name', 'album_name', and 'release_date'␣
↪columns.
    df_new.drop(['artist_name', 'track_name', 'album_name', 'release_date'],
            axis=1, inplace=True, errors='ignore',
           )
    df_new.set_index('track_id', inplace=True)    # Set the 'track_id' column␣
↪as the index

    #define categorical columns
    cat_cols = ['key', 'mode', 'time_signature']

    #One hot encoding the dataframes
    from sklearn.preprocessing import OneHotEncoder
```

```
    encoder = OneHotEncoder(sparse_output=False, drop='first')
    data_ohe = encoder.fit_transform(df_new[cat_cols])
    df_ohe = pd.DataFrame(data_ohe, columns=encoder.
 ↪get_feature_names_out(cat_cols), index=df_new.index)

    #merging OHE columns with numerical columns
    df_new = pd.concat([df_new.drop(cat_cols, axis=1), df_ohe], axis=1)

    # The test set must have the same columns as the training set, therefore
    # we'll create the missing columns in the test set and fill with zeros
    missing_cols = X_train.columns.difference(df_new.columns)
    if any(missing_cols):
        for cols in missing_cols:
            df_new[cols] = 0
    df_new = df_new[X_train.columns]

    # PREDICT
    if model == 'rf':
        y_pred = clf_rf_tuned.predict(df_new)
    elif model == 'adaboost':
        y_pred = ada_tuned.predict(df_new)
    elif model == 'logreg':
        if len(df_new) > 1:
            from sklearn.preprocessing import StandardScaler
            scaler = StandardScaler()
            df_new_sc = scaler.fit_transform(df_new)
            y_pred = clf_logregcv_tuned.predict(df_new_sc)
        elif len(df_new) == 1:
            y_pred = clf_logregcv_tuned.predict(df_new.values)

    return y_pred
```

[127]:
```
# This ids are track ids from artist not in the original data set
ids = ['2khv04F26pnJr4989Maowi', '1rrqJ9QkOBYJlsZgqqwxgB',␣
 ↪'1IMRi5UVOV77PsAgdWDvzh', '5FHwYRqxv08eyWWw7ARzJj',
       '7f3xivnGz4HUOUigVxvlEe', '3cRYXW7xZ6GJttdlPhBb1k',␣
 ↪'54KmblozuEemR23n9a4Grt', '4vb777iaycnlFxVkJMmtfd',
       '5aIVCx5tnkOntmdiinnYvw', "7lu6f7znGvbUpjFKvdqC8B",␣
 ↪'3eWpfsYgd5OL2QdwcVcF6Q', '4YAd7QqSKHz6dS2MCnq4mO',
       '7xzMrUmlooPa1Fmp88hlYc', '6gfdkLXXBzNUkCsf31PVYm',␣
 ↪'24qQClclS8CCjiCZKM8d9m', '5aNRjr4RchxYx1tT8z6CWa',
      ]
df_new = get_features(ids)
df_new
```

[127]:

```
                               track_name                track_id  \
0                                 Twe Twe  2khv04F26pnJr4989Maowi
1                                    Rush  1rrqJ9QkOBYJlsZgqqwxgB
2                                    Egwu  1IMRi5UVOV77PsAgdWDvzh
3                                  Liquor  5FHwYRqxv08eyWWw7ARzJj
4                                  Mukulu  7f3xivnGz4HUOUigVxvlEe
5                                   Abena  3cRYXW7xZ6GJttdlPhBb1k
6   Rainbow in the Sky (feat. Ijahman Levi)  54KmblozuEemR23n9a4Grt
7                                    Peru  4vb777iaycnlFxVkJMmtfd
8                                   Water  5aIVCx5tnk0ntmdiinnYvw
9                                       7lu6f7znGvbUpjFKvdqC8B
10                              Bust Down  3eWpfsYgd5OL2QdwcVcF6Q
11                               Alkassam  4YAd7QqSKHz6dS2MCnq4mO
12                                 Sodade  7xzMrUmlooPa1Fmp88hlYc
13                      Craving You Heavy  6gfdkLXXBzNUkCsf31PVYm
14                       Njila ia Dikanga  24qQClclS8CCjiCZKM8d9m
15                         Had El Maktoub  5aNRjr4RchxYx1tT8z6CWa

                                             genre  \
0         afrobeats,afropop,azontobeats,nigerian pop
1                                        afrobeats
2                     afrobeats,afropop,nigerian pop
3                             afrobeats,nigerian pop
4                             afrobeats,nigerian pop
5                             afrobeats,nigerian pop
6                 african reggae,reggae,roots reggae
7                             afrobeats,nigerian pop
8
9             arab pop,classic arab pop,egyptian pop
10                            afrobeats,nigerian pop
11        classic moroccan pop,gnawa,moroccan pop,rai
12  afropop,cape verdean folk,morna,musica cabo-ve…
13                                      ugandan pop
14       kizomba,kizomba antigas,musica angolana,semba
15

                                        album_name      artist_name  \
0                                          Twe Twe      Kizz Daniel
1                                             Rush       Ayra Starr
2                                             Egwu            Chike
3                                     High Tension    Bella Shmurda
4                                 Sincerely, Benson             Bnxn
5                                         Stranger             Lyta
6                                  Positive Energy     Alpha Blondy
7                                          Playboy      Fireboy DML
8                                            Water             Tyla
9                                                       Angham
```

10	Bust Down	Zlatan
11	Best of Nass El Ghiwane (Double album remaster…	Nass El Ghiwane
12	Miss Perfumado	Cesária Evora
13	AFRICAN MUSIC	Azawi
14	Independência	Paulo Flores
15	Had El Maktoub	Olfa Ben Romdhane

	release_date	duration_ms	popularity	danceability	key	acousticness	\
0	2024-01-25	143111	71	0.530	0	0.46200	
1	2022-09-16	185093	77	0.792	1	0.03690	
2	2023-12-15	136132	75	0.878	9	0.36600	
3	2020-01-27	193846	29	0.703	7	0.77300	
4	2023-10-04	163223	51	0.623	7	0.59800	
5	2023-02-10	141087	0	0.729	6	0.61700	
6	2015-05-18	267106	1	0.853	9	0.00489	
7	2022-08-04	187111	70	0.956	7	0.57200	
8	2023-07-28	200255	95	0.673	3	0.08560	
9	1988-01-01	284969	5	0.724	5	0.71500	
10	2024-02-01	196363	66	0.827	8	0.36000	
11	2011-03-01	341546	27	0.533	5	0.93500	
12	1992-10-21	293640	61	0.575	8	0.82200	
13	2021-10-09	201926	32	0.740	3	0.46000	
14	2021-04-30	341946	39	0.759	10	0.63900	
15	2020-11-14	237221	11	0.635	11	0.26100	

	mode	energy	instrumentalness	liveness	loudness	speechiness	tempo	\
0	0	0.844	0.000000	0.1230	-8.214	0.2110	134.977	
1	1	0.503	0.000570	0.0959	-8.044	0.0626	99.970	
2	1	0.722	0.003110	0.1410	-6.917	0.0473	117.967	
3	0	0.737	0.000135	0.1320	-6.344	0.1650	103.935	
4	1	0.130	0.000015	0.1160	-18.676	0.2970	97.710	
5	0	0.425	0.000047	0.0984	-14.477	0.1100	96.738	
6	0	0.451	0.000000	0.0593	-5.680	0.0679	124.898	
7	0	0.417	0.000710	0.0782	-7.892	0.0926	108.015	
8	0	0.722	0.000000	0.1370	-3.495	0.0755	117.187	
9	0	0.465	0.000000	0.4680	-8.730	0.0378	102.314	
10	1	0.819	0.000003	0.1550	-6.324	0.0787	110.008	
11	1	0.626	0.866000	0.1650	-12.575	0.0728	116.549	
12	1	0.430	0.000661	0.1150	-13.168	0.0363	82.691	
13	0	0.601	0.004310	0.0744	-6.596	0.0753	170.022	
14	0	0.640	0.000058	0.1120	-8.534	0.0649	96.986	
15	1	0.746	0.000000	0.3190	-5.236	0.1240	153.272	

	time_signature	valence
0	4	0.834
1	4	0.381
2	4	0.570

```
3                  4    0.812
4                  4    0.624
5                  4    0.152
6                  4    0.580
7                  4    0.714
8                  4    0.519
9                  4    0.651
10                 4    0.882
11                 3    0.926
12                 4    0.427
13                 4    0.942
14                 4    0.726
15                 3    0.488
```

Sixteen new songs were gathered, comprising eight popular and eight unpopular songs. A song is considered popular if its popularity score exceeds 42.5, as previously utilized in the model training process.

Random Forest

[128]:
```
# Drop the popularity column of the df before passing it to the predict function
prediction_rf = predict(df_new.drop('popularity', axis=1, errors='ignore'),␣
    ↪model='rf')
```

[129]:
```
df_pred_rf = df_new.loc[:, ['track_name', 'artist_name', 'popularity']]
df_pred_rf['true_value'] = df_pred_rf['popularity'].apply(lambda x: 'popular'␣
    ↪if x>=42.5 else 'unpopular')
df_pred_rf['prediction'] = prediction_rf
df_pred_rf['prediction'] = df_pred_rf['prediction'].apply(lambda x: 'popular'␣
    ↪if x==1 else 'unpopular')

correct = (df_pred_rf['true_value'] == df_pred_rf['prediction']).sum()
misclassified = (df_pred_rf['true_value'] != df_pred_rf['prediction']).sum()
print(f'Correctly classified: {correct}')
print(f'Misclassified: {misclassified}')
df_pred_rf
```

```
Correctly classified: 11
Misclassified: 5
```

[129]:

	track_name	artist_name	popularity	\
0	Twe Twe	Kizz Daniel	71	
1	Rush	Ayra Starr	77	
2	Egwu	Chike	75	
3	Liquor	Bella Shmurda	29	
4	Mukulu	Bnxn	51	
5	Abena	Lyta	0	
6	Rainbow in the Sky (feat. Ijahman Levi)	Alpha Blondy	1	

```
7                                        Peru        Fireboy DML          70
8                                       Water               Tyla          95
9                                                        Angham          5
10                                  Bust Down             Zlatan          66
11                                   Alkassam     Nass El Ghiwane          27
12                                     Sodade       Cesária Evora          61
13                          Craving You Heavy               Azawi          32
14                           Njila ia Dikanga        Paulo Flores          39
15                             Had El Maktoub   Olfa Ben Romdhane          11
```

```
   true_value prediction
0     popular  unpopular
1     popular    popular
2     popular    popular
3   unpopular    popular
4     popular    popular
5   unpopular    popular
6   unpopular  unpopular
7     popular    popular
8     popular  unpopular
9   unpopular  unpopular
10    popular    popular
11  unpopular  unpopular
12    popular  unpopular
13  unpopular  unpopular
14  unpopular  unpopular
15  unpopular  unpopular
```

The Random Forest model performed well on unseen data, correctly predicting 11 out of 16 instances. Notably, it tended to classify unpopular songs as popular, aligning with our goal of prioritizing high recall compare to precision. This suggests that the model may occasionally misclassify unpopular songs as popular, which is still acceptable given the context.

AdaBoost

[130]:
```python
prediction_ada = predict(df_new.drop('popularity', axis=1, errors='ignore'),
 ↪model='adaboost')
```

[131]:
```python
df_pred_ada = df_new.loc[:, ['track_name', 'artist_name', 'popularity']]
df_pred_ada['true_value'] = df_pred_ada['popularity'].apply(lambda x: 'popular'
 ↪if x>=42.5 else 'unpopular')
df_pred_ada['prediction'] = prediction_ada
df_pred_ada['prediction'] = df_pred_ada['prediction'].apply(lambda x: 'popular'
 ↪if x==1 else 'unpopular')

correct = (df_pred_ada['true_value'] == df_pred_ada['prediction']).sum()
misclassified = (df_pred_ada['true_value'] != df_pred_ada['prediction']).sum()
print(f'Correctly classified: {correct}')
```

```
print(f'Misclassified: {misclassified}')
df_pred_ada
```

```
Correctly classified: 12
Misclassified: 4
```

[131]:

```
                                track_name       artist_name  popularity  \
0                                  Twe Twe       Kizz Daniel          71
1                                     Rush        Ayra Starr          77
2                                     Egwu             Chike          75
3                                   Liquor     Bella Shmurda          29
4                                   Mukulu              Bnxn          51
5                                    Abena              Lyta           0
6   Rainbow in the Sky (feat. Ijahman Levi)      Alpha Blondy           1
7                                     Peru       Fireboy DML          70
8                                    Water              Tyla          95
9                                                     Angham           5
10                               Bust Down            Zlatan          66
11                                Alkassam    Nass El Ghiwane          27
12                                  Sodade     Cesária Evora          61
13                       Craving You Heavy             Azawi          32
14                        Njila ia Dikanga      Paulo Flores          39
15                          Had El Maktoub  Olfa Ben Romdhane         11

   true_value prediction
0     popular    popular
1     popular  unpopular
2     popular    popular
3   unpopular  unpopular
4     popular    popular
5   unpopular  unpopular
6   unpopular  unpopular
7     popular  unpopular
8     popular    popular
9   unpopular  unpopular
10    popular  unpopular
11  unpopular  unpopular
12    popular  unpopular
13  unpopular  unpopular
14  unpopular  unpopular
15  unpopular  unpopular
```

In contrast to the Random Forest model, the Adaboost model may occasionally classify popular songs as unpopular. However, this tendency can be advantageous when high precision is crucial. If the Adaboost model predicts a song as popular, it likely has a high probability of being so. This characteristic enhances confidence in the model's predictions and ensures a more precise identification of popular songs. In addition, Adaboost focuses on track features such as danceability, duration, loudness, speechiness, and instrumentalness, rather than relying solely on genre classification. This approach allows Adaboost to predict tracks that may not conform to typical genre patterns but exhibit characteristics associated with popularity. For instance, in the unseen dataset, the song “Water” by Tyla with a missing genre, and a popularity score of 95, was correctly classified as popular by Adaboost, while Random Forest and Logistic Regression failed to do so. This highlights Adaboost’s advantage in leveraging specific track features to make accurate predictions, compared to models that rely primarily on genre classification.

Logistic Regression

[132]:
```
prediction_logreg = predict(df_new.drop('popularity', axis=1, errors='ignore'),
    model='logreg')
```

[133]:
```
df_pred_logreg = df_new.loc[:, ['track_name', 'artist_name', 'popularity']]
df_pred_logreg['true_value'] = df_pred_logreg['popularity'].apply(lambda x:
    'popular' if x>=42.5 else 'unpopular')
df_pred_logreg['prediction'] = prediction_logreg
df_pred_logreg['prediction'] = df_pred_logreg['prediction'].apply(lambda x:
    'popular' if x==1 else 'unpopular')

correct = (df_pred_logreg['true_value'] == df_pred_logreg['prediction']).sum()
misclassified = (df_pred_logreg['true_value'] != df_pred_logreg['prediction']).
    sum()
print(f'Correctly classified: {correct}')
print(f'Misclassified: {misclassified}')
df_pred_logreg
```

```
Correctly classified: 11
Misclassified: 5
```

[133]:
```
                                track_name        artist_name  popularity  \
0                                  Twe Twe        Kizz Daniel          71
1                                     Rush         Ayra Starr          77
2                                     Egwu              Chike          75
3                                   Liquor      Bella Shmurda          29
4                                   Mukulu               Bnxn          51
5                                    Abena               Lyta           0
6   Rainbow in the Sky (feat. Ijahman Levi)      Alpha Blondy           1
7                                     Peru        Fireboy DML          70
8                                    Water               Tyla          95
9                                                   Angham           5
10                               Bust Down             Zlatan          66
11                                Alkassam    Nass El Ghiwane          27
12                                  Sodade      Cesária Evora          61
13                       Craving You Heavy              Azawi          32
14                        Njila ia Dikanga       Paulo Flores          39
15                          Had El Maktoub  Olfa Ben Romdhane          11

   true_value prediction
```

```
0     popular    popular
1     popular    popular
2     popular    popular
3   unpopular    popular
4     popular    popular
5   unpopular    popular
6   unpopular  unpopular
7     popular    popular
8     popular  unpopular
9   unpopular  unpopular
10    popular    popular
11  unpopular  unpopular
12    popular  unpopular
13  unpopular    popular
14  unpopular  unpopular
15  unpopular  unpopular
```

For logistic regression, the prediction approach is similar to that of Random Forest, primarily relying on the genre of the music. However, logistic regression tends to perform slightly better than the Random Forest model in some cases. Lastly let's concatenate all models' prediction into a single dataframe.

[136]:
```
df_pred_all_model = df_new.loc[:, ['track_name', 'artist_name', 'popularity']]
df_pred_all_model['true_value'] = df_pred_all_model['popularity'].apply(lambda␣
 ↪x: 'popular' if x>=42.5 else 'unpopular')
df_pred_all_model['Random Forest Prediction'] = df_pred_rf['prediction']
df_pred_all_model['AdaBoost Prediction'] = df_pred_ada['prediction']
df_pred_all_model['Logistic Regrssion Prediction'] =␣
 ↪df_pred_logreg['prediction']
df_pred_all_model
```

[136]:
```
                                track_name         artist_name  popularity  \
0                                  Twe Twe         Kizz Daniel          71
1                                     Rush          Ayra Starr          77
2                                     Egwu               Chike          75
3                                   Liquor      Bella Shmurda          29
4                                   Mukulu                Bnxn          51
5                                    Abena                Lyta           0
6   Rainbow in the Sky (feat. Ijahman Levi)       Alpha Blondy           1
7                                     Peru         Fireboy DML          70
8                                    Water                Tyla          95
9                                                    Angham          5
10                               Bust Down              Zlatan          66
11                                Alkassam     Nass El Ghiwane          27
12                                  Sodade      Cesária Evora          61
13                        Craving You Heavy               Azawi          32
14                          Njila ia Dikanga        Paulo Flores          39
```

```
15                          Had El Maktoub  Olfa Ben Romdhane         11

   true_value Random Forest Prediction AdaBoost Prediction  \
0     popular                unpopular             popular
1     popular                  popular           unpopular
2     popular                  popular             popular
3   unpopular                  popular           unpopular
4     popular                  popular             popular
5   unpopular                  popular           unpopular
6   unpopular                unpopular           unpopular
7     popular                  popular           unpopular
8     popular                unpopular             popular
9   unpopular                unpopular           unpopular
10    popular                  popular           unpopular
11  unpopular                unpopular           unpopular
12    popular                unpopular           unpopular
13  unpopular                unpopular           unpopular
14  unpopular                unpopular           unpopular
15  unpopular                unpopular           unpopular

   Logistic Regrssion Prediction
0                        popular
1                        popular
2                        popular
3                        popular
4                        popular
5                        popular
6                      unpopular
7                        popular
8                      unpopular
9                      unpopular
10                       popular
11                     unpopular
12                     unpopular
13                       popular
14                     unpopular
15                     unpopular
```

[138]:
```
df_pred_all_model.to_csv("Prediction_result.csv", index=False)
```