



LKH

SPOTIFY ML

LABEL: TOP 100?

Fun Fact: Spotify launched in 2011

<https://github.com/lolasery/khdatasci>

<https://www.kaggle.com/ektanegi/spotifydata-19212020>

A BRIEF OVERVIEW OF MUSIC!

1900S -> JAZZ, CLASSICAL, BLUEGRASS

Google

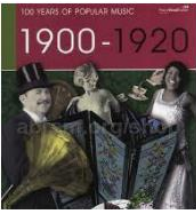
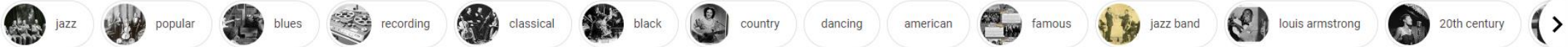
music in the 1900s



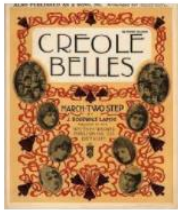
All Images Maps Videos News More

Settings Tools

Collections SafeSearch



Popular Music: 1900-1920 (...)
shop.abrm.org



1900 in music - Wikipedia
en.wikipedia.org



timeline | Timetoast timelines
timetoast.com



9 Musika - Malaya's Early Music Scene ...
pinterest.com



History of the School of Music - School ...
sc.edu



The story of jazz: 100 years on and ...
express.co.uk



National Jukebox now online, serving ...
arstechnica.com



Early 1900s Musicians High Res...
alamy.com



Kyiv Philharmonic (early 1900s) - Imgur
imgur.com



Early 1900's music S...
playlists.net



22 Free 1900s music playlis...
8tracks.com



List of pre-1920 jazz standards - Wikipedia
en.wikipedia.org



1900s Turn of the Cent...
southernmusic.net



Buddy Bolden 'invents' jazz | Jazz ...
theguardian.com



United States - Popular music | Britan...
britannica.com



Louis Armstrong
PD-NYTWIS PD-US-no notice



A BRIEF OVERVIEW OF MUSIC! 2020S -> POP, R&B, SOUL, INDIE

Google

mtv 2020



All

Images

Videos

News

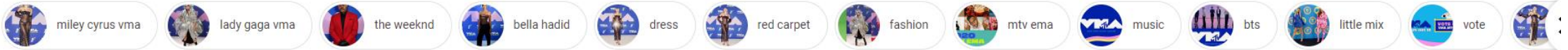
Shopping

More

Settings

Tools

Collections SafeSearch



2020 MTV Video Music Awards - Wikipedia
en.wikipedia.org



MTV VMAs 2020 Red Carpet: The Must-S...
vanityfair.com



MTV Video Music Awar...
harpersbazaar.com



The Best Looks From The 2020 MTV VMAs
forbes.com



sheer Mugler gown at the 2020 MTV VMAs
harpersbazaar.com



MTV VMAs 2020: The Best-Dressed ...
glamour.com



MTV's 2020 VMAs: Recapping Lady Gaga's ...
npr.org



MTV VMAs 2020 Winners: See the Full ...
pitchfork.com



MTV VMAs 2020: Lady Gaga's outfits ...
usatoday.com



How To Watch The 2020 MTV VMAs And ...
deadline.com



MTV VMAs 2020 Red Carpet Celeb...
elle.com



MTV Video Music Awards ...
goldderby.com



The Best Looks From T...
forbes.com

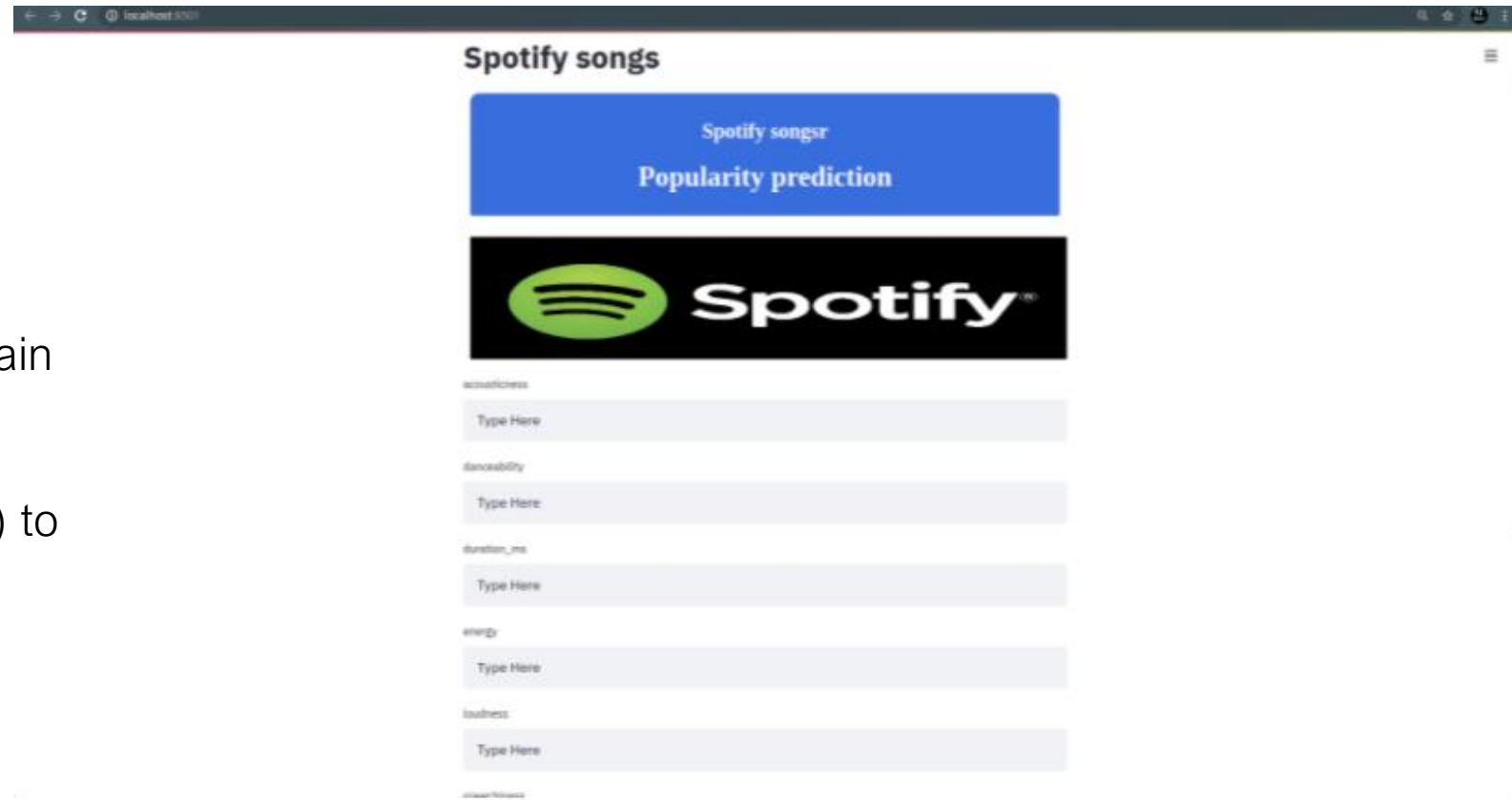


MTV (VMAs ~2020) Watch "MTV Video Music ...
mtv-vmas-2020-watch-mtv-video-music-awards-full-show.p...



• LABEL SIGNIFICANCE

- Very humble and casual
- Can help artistes aim for the top 100s to gain more fame, investors, etc.
- help to shape future trends more easily
- perhaps can create a app (pic on the right) to let spotify users predict a song's popularity based on the features I trained my ML with



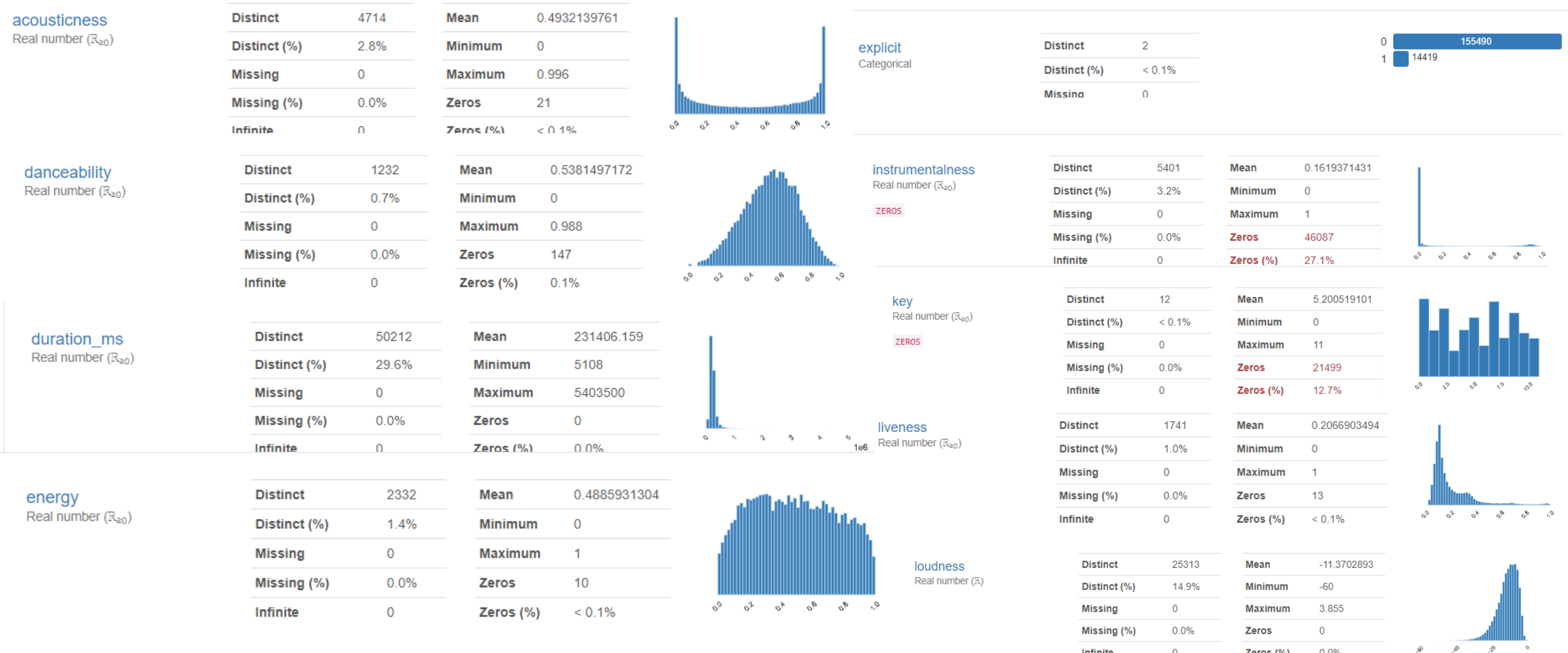
The screenshot shows a web browser window with the address bar set to localhost:5000. The page is titled "Spotify songs" and features a blue button labeled "Spotify songs" and "Popularity prediction". Below the button is a black banner with the Spotify logo. The form contains five input fields, each with a label and a "Type Here" placeholder:

- acousticness
- danceability
- duration_ms
- energy
- loudness

At the bottom of the form, there is a small "Submit" button.

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

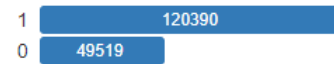
Using Pandas profile analyze if data is imbalanced



DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

mode
Categorical

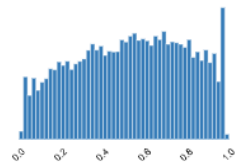
Distinct	2
Distinct (%)	< 0.1%
Missing	0



valence
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	1739
Distinct (%)	1.0%
Missing	0
Missing (%)	0.0%
Infinite	0

Mean	0.5320951423
Minimum	0
Maximum	1
Zeros	185
Zeros (%)	0.1%

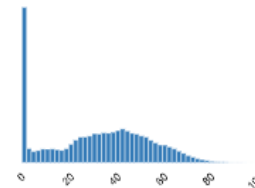


popularity
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	100
Distinct (%)	0.1%
Missing	0
Missing (%)	0.0%
Infinite	0

ZEROS

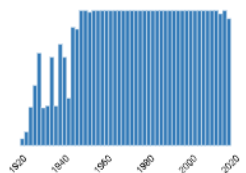
Mean	31.55660971
Minimum	0
Maximum	100
Zeros	27357
Zeros (%)	16.1%



year
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	100
Distinct (%)	0.1%
Missing	0
Missing (%)	0.0%
Infinite	0

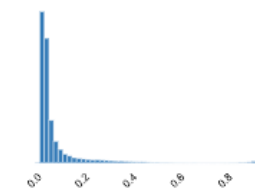
Mean	1977.223231
Minimum	1921
Maximum	2020
Zeros	0
Zeros (%)	0.0%



speechiness
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	1628
Distinct (%)	1.0%
Missing	0
Missing (%)	0.0%
Infinite	0

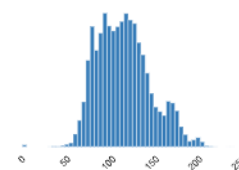
Mean	0.09405769441
Minimum	0
Maximum	0.969
Zeros	148
Zeros (%)	0.1%



tempo
Real number ($\mathbb{R}_{\geq 0}$)

Distinct	84548
Distinct (%)	49.8%
Missing	0
Missing (%)	0.0%
Infinite	0

Mean	116.9480174
Minimum	0
Maximum	244.091
Zeros	147
Zeros (%)	0.1%



DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

- 1. Acousticness (0-1 float, not having electrical amplification eg. guitar over electric guitar, piano over keyboard)
- 2. danceability (0-1 float, How likely one can dance to the song)
- 3. energy (0-1 float, whatever keeps the listener engaged and listening -> abit too subjective, might drop)
- 4. explicit (0 or 1 int, contains explicit language)
- 5. instrumentalness (0-1 float, Predicts whether a track contains no vocals 0 = vocals, 1 = no vocals)
- 6. key (0-11 int, All keys on octave encoded as values ranging from 0 to 11, starting on C as 0, C# as 1 and so on -> needs musical background to determine what it really means eg. C major gives a happy yet melancholic undertone etc.)
- 7. liveness (0-1 float, Detects the presence of an audience in the recording, 0= not live , 1= live)
- 8. loudness (-60 to 3 float, -60 = soft, 3 = loud)
- 9. mode (0-1 float, 0= minor (sad), 1=major(happy))
- 10. release_date (datetime)

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

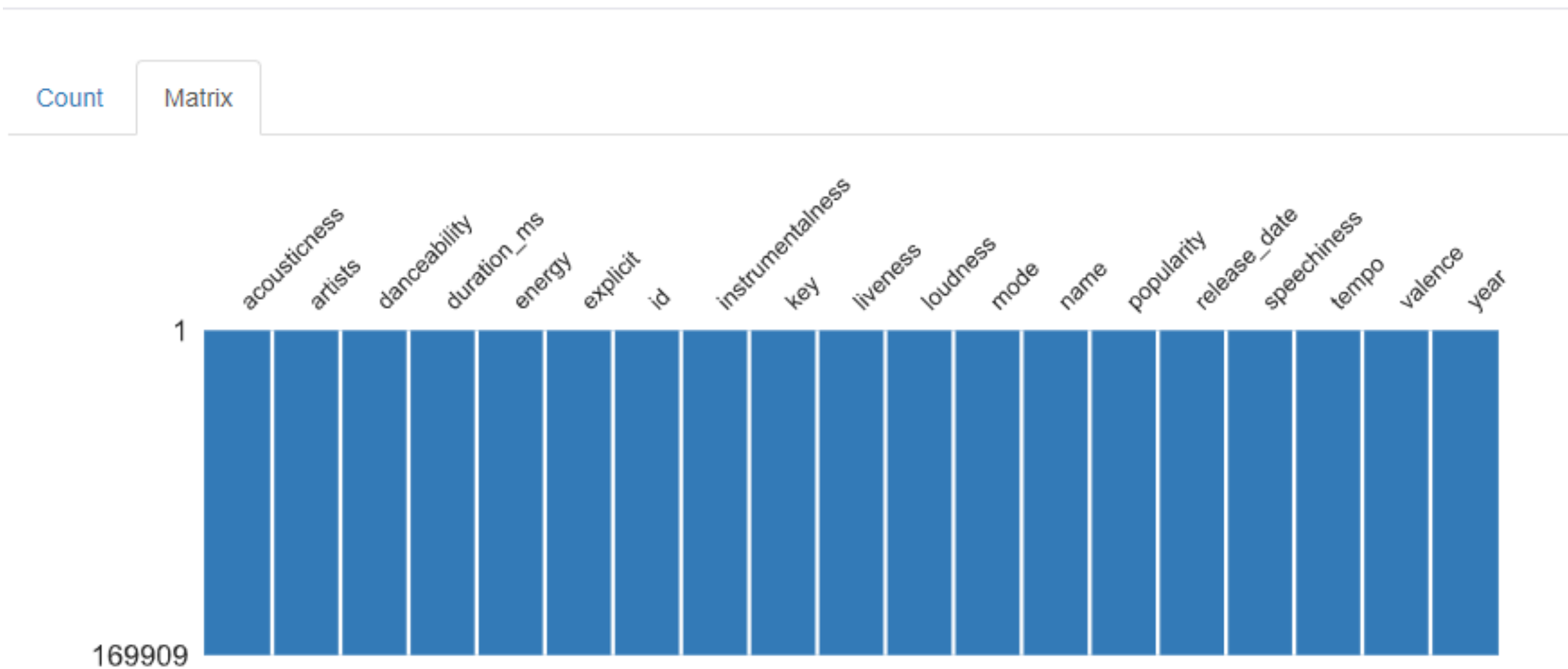
- 11. speechiness (0-1 float, >0.66 = made of spoken words (audio book), $0.33-0.66$ = mix of music and speech, <0.33 = no speech)
 - - <http://open.spotify.com/track/1u9vcc9PQvAv3Nh4qRp3lf> <-- definitely singing but somehow is more speechy
 - - <http://open.spotify.com/track/1UEVy1gNCTTCTjp0Tk01rm> <-- there is background talking but also 100% singing
 - - There are definitely non-songs (podcasts, readings) in the mix but are too difficult to immediately identify
- 12. tempo (0-244 float, speed of music, higher = faster)
- 13. valence (0-1 float, 0= sad/negative, 1=happy/positive)
- 14. Artist (str, name of artist -> able to feature engineer into length of name)
- 15. duration_ms (float, Duration in milliseconds)
- 16. id (dropping this as it is useless)
- 17. name (str, of song)
- 18. popularity (0-100 int, 0 = no ranking)
- 19. release_date [Song was released] (datetime) -> Considering to remove since not all release dates have same format
- 20. year [Song was released] (1921-2020)

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

- This dataset is special in the sense that Year is not the year in which the popularity was obtained: It is the year song was released.
- Difficulties:
 - Speechiness can be very high despite the person obviously singing.
 - Loudness -> Not too sure which measure they used for this -> Can only assume the higher the value the louder it is

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

Missing values



Yes!!

Nullity matrix is a data-dense display which lets you quickly visually pick out patterns in data completion.

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

Correlations

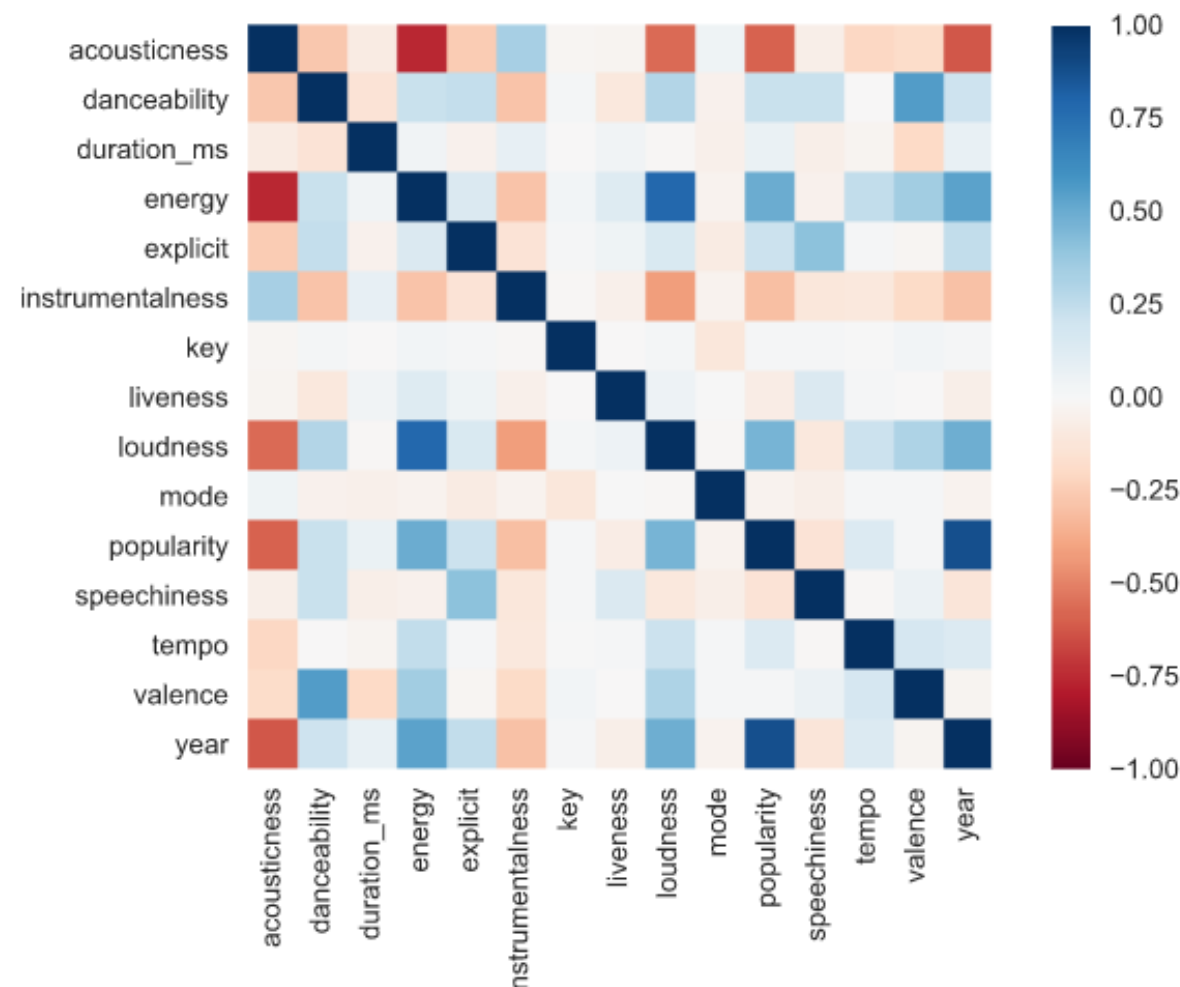
Pearson's r

Spearman's ρ

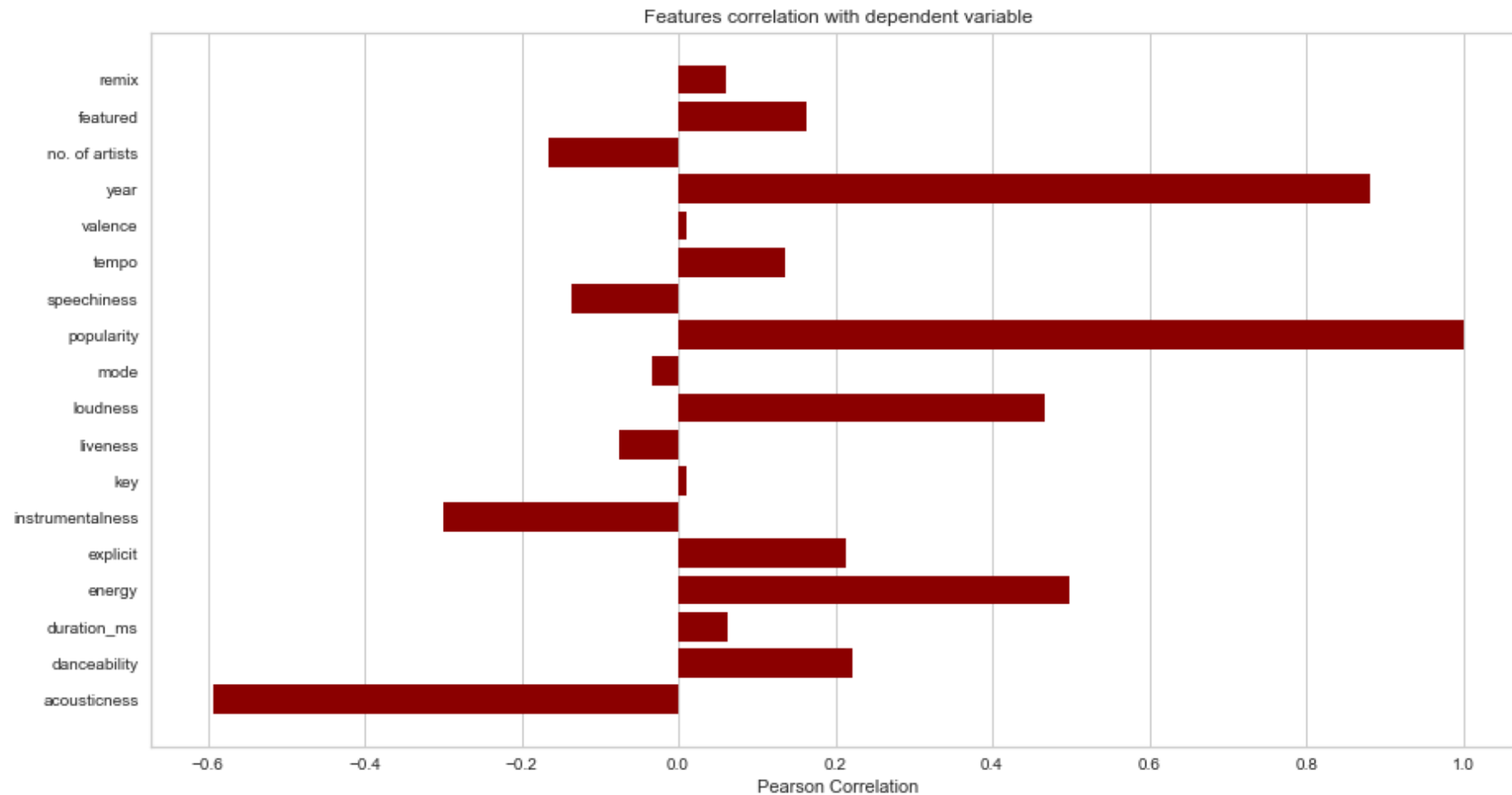
Kendall's τ

Phik (ϕ_k)

Cramér's V (ϕ_c)



DATA UNDERSTANDING, EXPLORATION + PREPROCESSING



from yellowbrick library:
Year, loudness, energy

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

- Exploration!

CASE STUDY

POTENTIALLY DUPLICATED SONG (EITHER LATEST OR MOST POPULAR?)

SONG 1: POLONAISE-FANTAISIE IN A-FLAT MAJOR, OP. 61

	artists	name	id	popularity	release_date
4	['Frédéric Chopin', 'Vladimir Horowitz']	Polonaise-Fantaisie in A-Flat Major, Op. 61	6N6tiFZ9vLTSOlxkj8qKrd	1	1928
83	['Frédéric Chopin', 'Vladimir Horowitz']	Polonaise-Fantaisie in A-Flat Major, Op. 61	71FaVeFy9ZOiQRY4yOijey	0	1928
8185	['Frédéric Chopin', 'Vladimir Horowitz']	Polonaise-Fantaisie in A-Flat Major, Op. 61	7aH7AMePMza5bZX53oHfgr	0	1928
117019	['Frédéric Chopin', 'Vladimir Horowitz']	Polonaise-Fantaisie in A-Flat Major, Op. 61	2RM4VG4Rkwmp1EbM95Uo7E	0	1928
126435	['Frédéric Chopin', 'Vladimir Horowitz']	Polonaise-Fantaisie in A-Flat Major, Op. 61	2gxdizo1tudU7R72Wpd2pe	0	1928
152990	['Frédéric Chopin', 'Vlado Perlemuter']	Polonaise-Fantaisie in A-Flat Major, Op. 61	4c1VrYOOoFaa3v1Zcw9LBO	0	1926

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	valence	year	no. of artists	featured	remix
4	0.99	0.21	687733	0.20	0	0.91	11	0.10	-16.83	1	1	0.04	62.15	0.07	1928	2	0	0
83	0.99	0.30	785427	0.08	0	0.85	1	0.09	-23.28	1	0	0.04	137.30	0.05	1928	2	0	0
8185	0.99	0.30	785427	0.08	0	0.85	1	0.09	-23.28	1	0	0.04	137.30	0.05	1928	2	0	0
117019	0.99	0.31	785133	0.10	0	0.86	1	0.09	-22.30	1	0	0.04	136.08	0.07	1928	2	0	0
126435	0.99	0.30	797547	0.14	0	0.86	11	0.88	-21.54	1	0	0.04	133.28	0.06	1928	2	0	0
152990	0.99	0.24	707813	0.09	0	0.89	11	0.08	-20.95	1	0	0.04	71.31	0.04	1926	2	0	0

CASE STUDY

POTENTIALLY DUPLICATED SONG (EITHER LATEST OR MOST POPULAR?)

SONG 1: POLONAISE-FANTAISIE IN A-FLAT MAJOR, OP. 61

- <https://open.spotify.com/album/1n81HsE0rnviDNlIfX3fp0?highlight=spotify:track:6N6tiFZ9vLTSOIxkj8qKrd> #disc3
- <https://open.spotify.com/album/1n81HsE0rnviDNlIfX3fp0?highlight=spotify:track:2RM4VG4Rkwmp1EbM95Uo7E> #disc4
- <https://open.spotify.com/album/1n81HsE0rnviDNlIfX3fp0?highlight=spotify:track:2gxdizo1tudU7R72Wpd2pe> #disc6
- <https://open.spotify.com/album/6P9bPQ1LDtgAB5V8Bt50ne?highlight=spotify:track:71FaVeFy9ZOiQRY4yOijey>
- <https://open.spotify.com/album/27NrfgJFNdDIKJnSxHXcJt?highlight=spotify:track:7aH7AMePMza5bZX53oHfgr>
- <https://open.spotify.com/album/2tBnuQsf1e2rXt6oW4Vy2N?highlight=spotify:track:4c1VrYOOoFaa3v1Zcw9LBO>
- #individual songs probably played by different people
- Despite sounding almost exactly the same (Yes I actually listened to all 6 songs)
 - They have very different popularity!

CASE STUDY

POTENTIALLY DUPLICATED SONG (EITHER LATEST OR MOST POPULAR?)

SONG 2: MORE HEARTS THAN MINE

artists		name	id	popularity	release_date
116612	['Ingrid Andress']	More Hearts Than Mine	0LcspVKJxhEQQSvVMiTPWz	70	2019-04-05
169908	['Ingrid Andress']	More Hearts Than Mine	60RFIt48hm0l4Fu0JoccOl	65	2020-03-27

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	popularity	speechiness	tempo	valence	year	no. of artists	featured	remix
116612	0.11	0.41	214160	0.43	0	0.00	0	0.11	-7.41	1	70	0.03	79.98	0.39	2019	1	0	0
169908	0.11	0.51	214787	0.43	0	0.00	0	0.10	-7.39	1	65	0.03	80.59	0.37	2020	1	0	0

CASE STUDY

POTENTIALLY DUPLICATED SONG (EITHER LATEST OR MOST POPULAR?)

SONG 2: MORE HEARTS THAN MINE

- # <https://open.spotify.com/album/4VMYwWFqX9vUv9otWLRRF5?highlight=spotify:track:0LcspVKJxhEQQSvVMiTPWz> released as single in 2019
- # <https://open.spotify.com/album/6qon3hv0lhwK8o57PvVWZI?highlight=spotify:track:60RFIt48hm0l4Fu0JoccOl> re-released as an album collection in 2020
- Despite being the same song, the timing released allowed a better performance for the later re-release

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

FEATURE ENGINEERING AKA CREATING NEW COLUMNS

- ## 'no. of artists' - no. of artists in a song

```
spotify_global_top100['no. of artists'] #for checking
```

```
0      1
1      2
2      1
3      1
4      2
..
169904  2
169905  2
169906  2
169907  2
169908  1
```

```
Name: no. of artists, Length: 169909, dtype: int64
```

artists

['Carl Woitschach']

['Robert Schumann', 'Vladimir Horowitz']

['Seweryn Goszczyński']

['Francisco Canaro']

['Frédéric Chopin', 'Vladimir Horowitz']

['Felix Mendelssohn', 'Vladimir Horowitz']

['Franz Liszt', 'Vladimir Horowitz']

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

FEATURE ENGINEERING AKA CREATING NEW COLUMNS

- ## 'featured' - whether or not the song has a feature

```
spotify_global_top100['featured']
```

```
0      0
1      0
2      0
3      0
4      0
..
169904  1
169905  1
169906  0
169907  0
169908  0
```

```
Name: featured, Length: 169909, dtype: int64
```

name

Rough Ryder

I Dare You

Letter To Nipsey (feat. Roddy Ricch)

Back Home (feat. Summer Walker)

Ojos De Maniaco

Skechers (feat. Tyga) - Remix

Sweeter (feat. Terrace Martin)

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

FEATURE ENGINEERING AKA CREATING NEW COLUMNS

- ## 'no. of artists' - no. of artists in a song

```
spotify_global_top100['no. of artists'] #for checking
```

```
0      1
1      2
2      1
3      1
4      2
..
169904  2
169905  2
169906  2
169907  2
169908  1
```

```
Name: no. of artists, Length: 169909, dtype: int64
```

artists

['Carl Woitschach']

['Robert Schumann', 'Vladimir Horowitz']

['Seweryn Goszczyński']

['Francisco Canaro']

['Frédéric Chopin', 'Vladimir Horowitz']

['Felix Mendelssohn', 'Vladimir Horowitz']

['Franz Liszt', 'Vladimir Horowitz']

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

FEATURE ENGINEERING AKA CREATING NEW COLUMNS

- ## 'no. of artists' - no. of artists in a song

```
spotify_global_top100['remix']
```

```
0      0
1      0
2      0
3      0
4      0
..
169904  1
169905  0
169906  0
169907  0
169908  0
```

```
Name: remix, Length: 169909, dtype: int64
```

```
spotify_global_top100["name"][spotify_global_top100['name']].
```

```
2868      Merry Go Round - Take 2 Master Version with St...
3045      Beginnings - 50th Anniversary Remix
3157      Getting In Tune - New York Record Plant Sessio...
3202      Pure And Easy - New York Record Plant Session ...
3305      You Curl Your Toes in Fun / Childhood Heroes /...
```

```
...
```

```
169776      Que Mas Pues - Remix
169779      Dream Girl - Remix
169792      Triggered - Remix
169887      My Truck (feat. Sam Hunt) - Remix
169904      Skechers (feat. Tyga) - Remix
```

```
Name: name, Length: 951, dtype: object
```

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

REMOVE "DUPES"

```
#boolean list that removes
spotify_global_top100['artists + name'] = spotify_global_top100["artists"] + spotify_global_top100['name']
boool = spotify_global_top100['artists + name'].duplicated(keep='last')
boool
```

```
0      False
1       True
2      False
3      False
4       True
...
169904  False
169905  False
169906  False
169907  False
169908  False
Name: artists + name, Length: 169909, dtype: bool
```

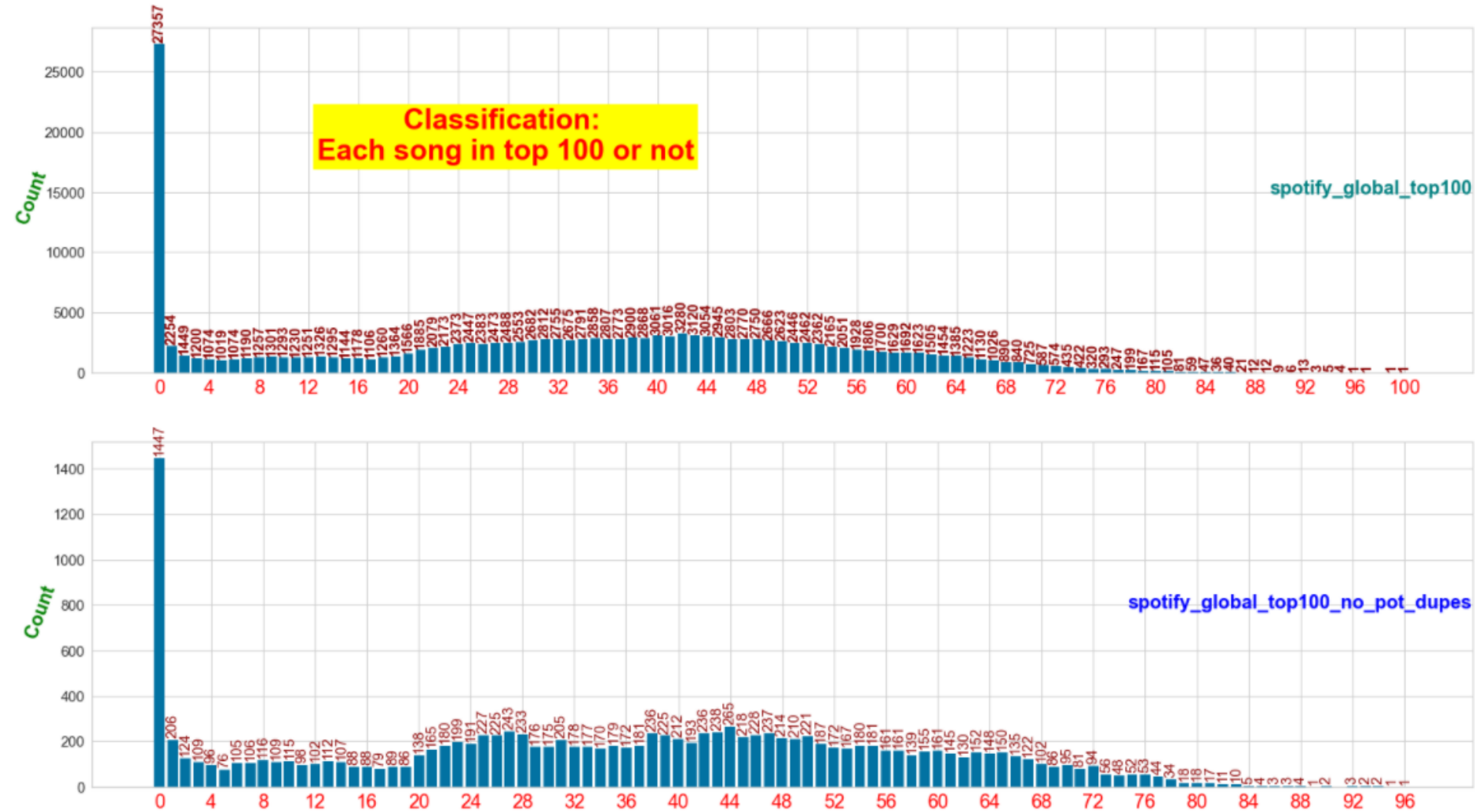
```
boool.value_counts()
```

```
False    156608
True      13301
Name: artists + name, dtype: int64
```

Data has truncated from
169909 to 13301

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

LABEL



A lot of 0s -> Not top 100

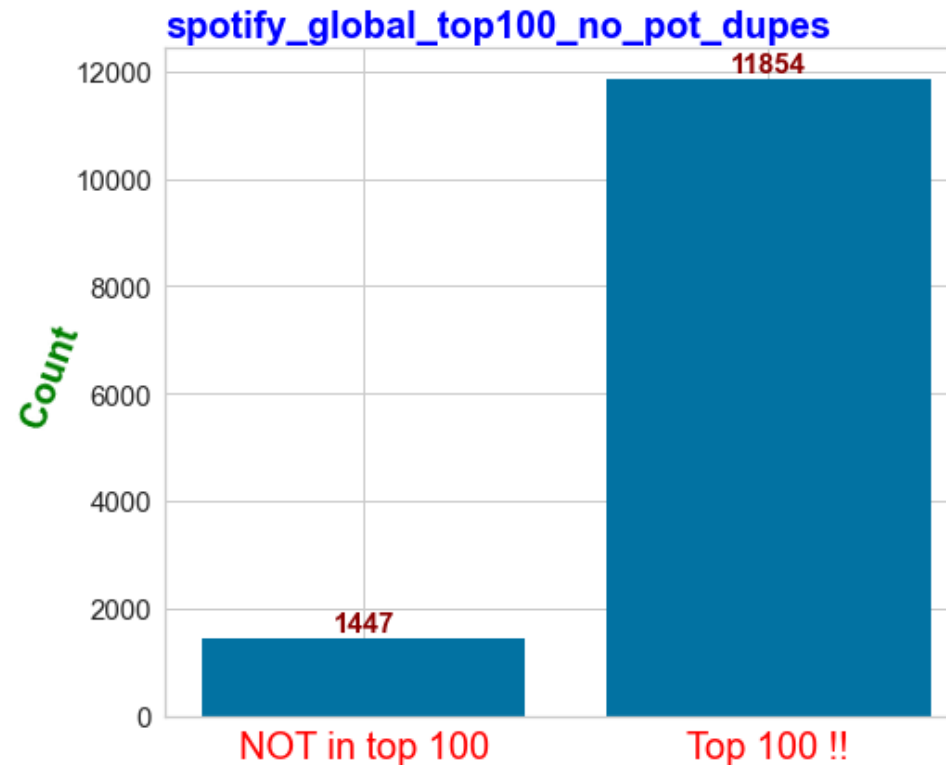
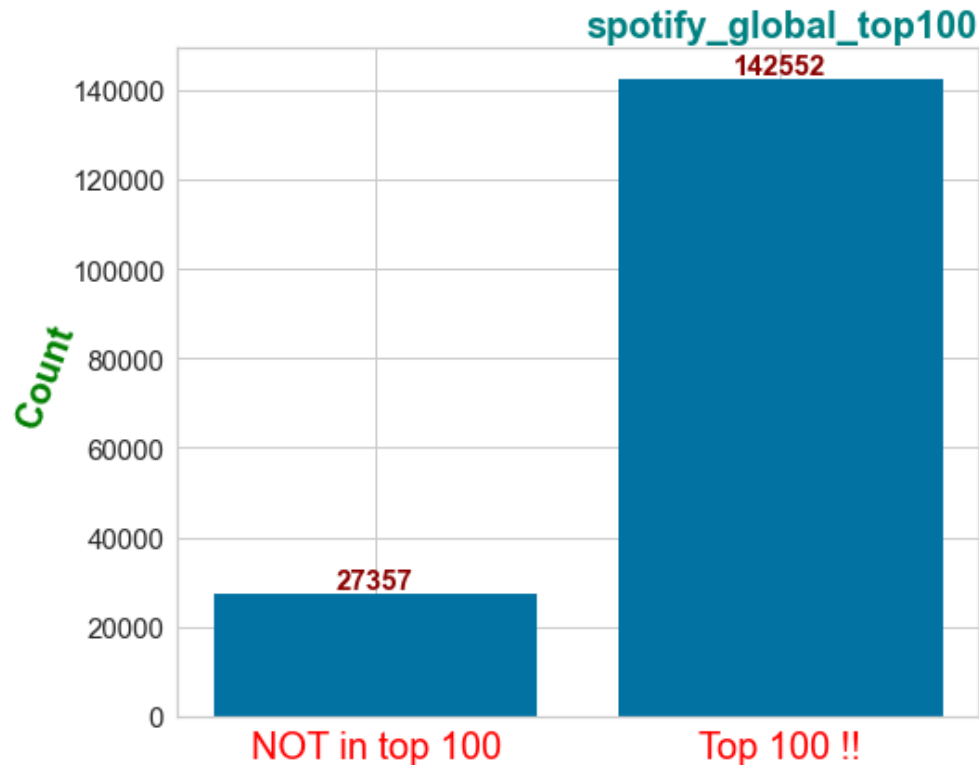
DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

Transform multiclass into Binary for simplicity sake

still A lot of 0s -> Not top 100

LABEL

**Classification (NOT BINNED):
No. of songs in top 100s VS top > 100**

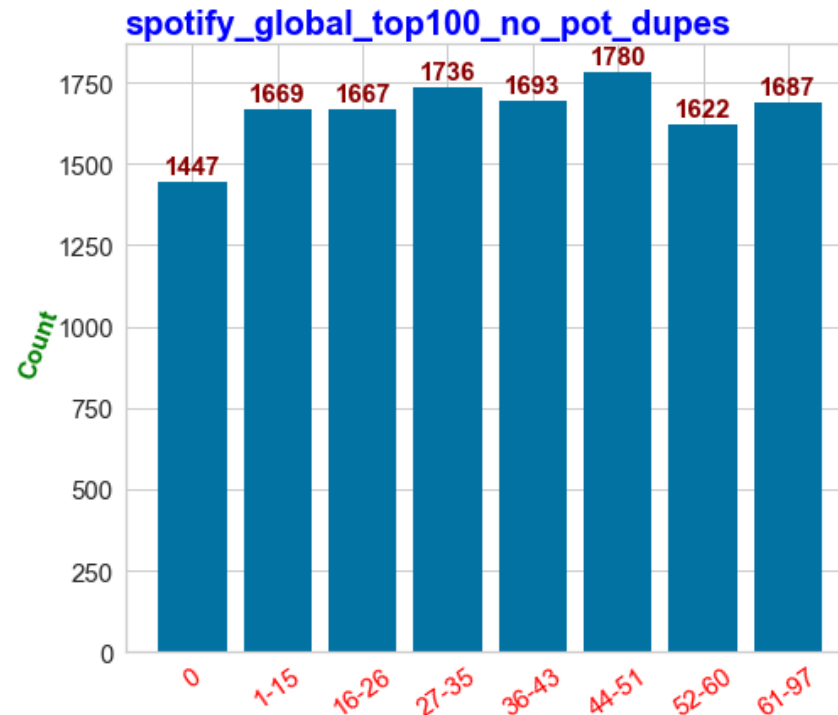
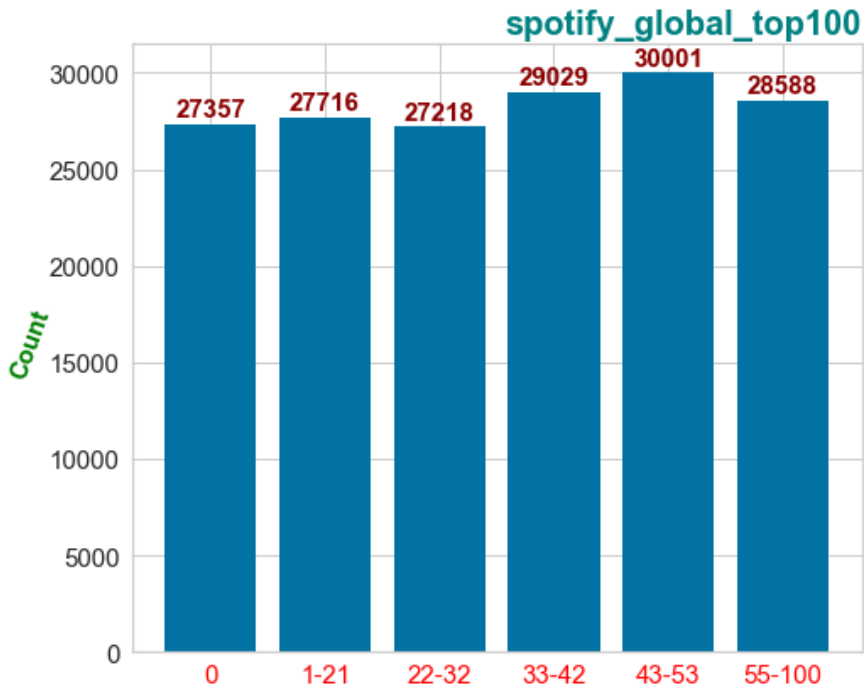


DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

LABEL

- Should I bin it? I could let it be and use other methods to solve it... but lets try binning to make it more balanced

Classification, BINNED: Each song in top 100 or not



- spotify_global_top100

- top 1:21 -> 1
- top 22:32 -> 2
- top 33:42 -> 3
- top 43:53 -> 4
- top 54:100 -> 5

-
spotify_global_top100_no
_pot_dupes

- top 1:15 -> 1
- top 16:26 -> 2
- top 27:35 -> 3
- top 36:43 -> 4
- top 44:51 -> 5
- top 52:61 -> 6
- top 62:96 -> 7

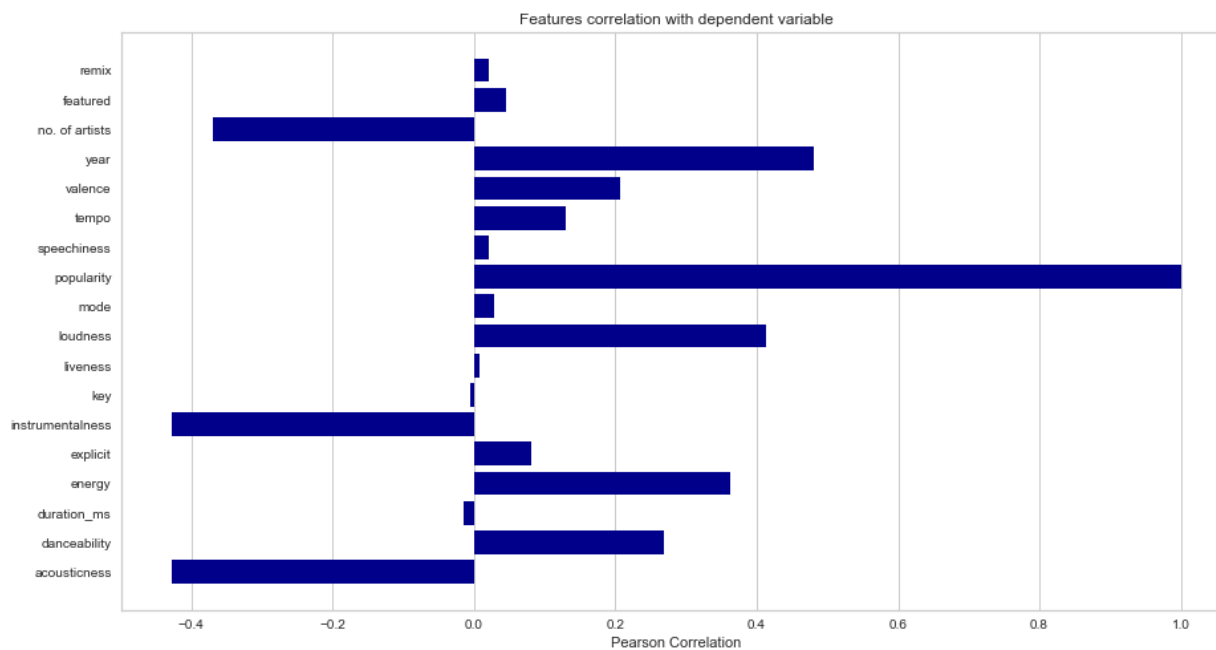
DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

RE-UNDERSTAND

spotify_global_top100_no_pot_dupes

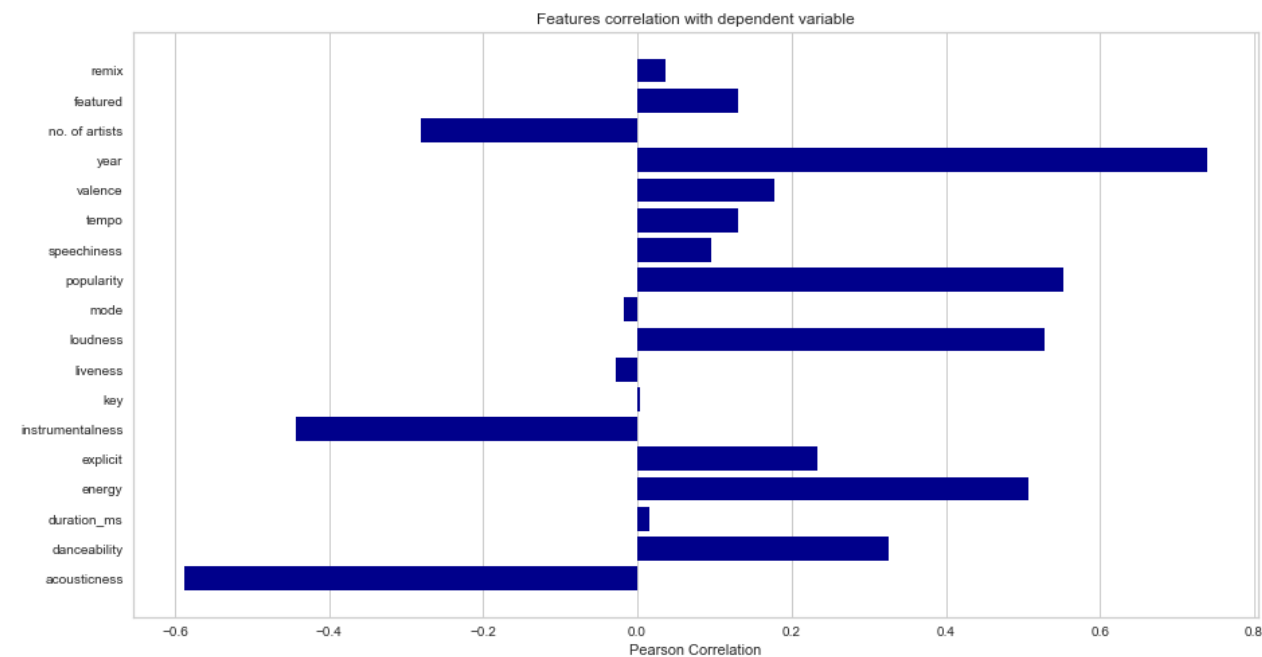
Feat. corre for **popularity NOT BINNED**

...



Feat. corre for **Popularity in top 100 separately BINNED**

...



Binning made the other positive features more positive

DATA UNDERSTANDING, EXPLORATION + PREPROCESSING

RE-UNDERSTAND

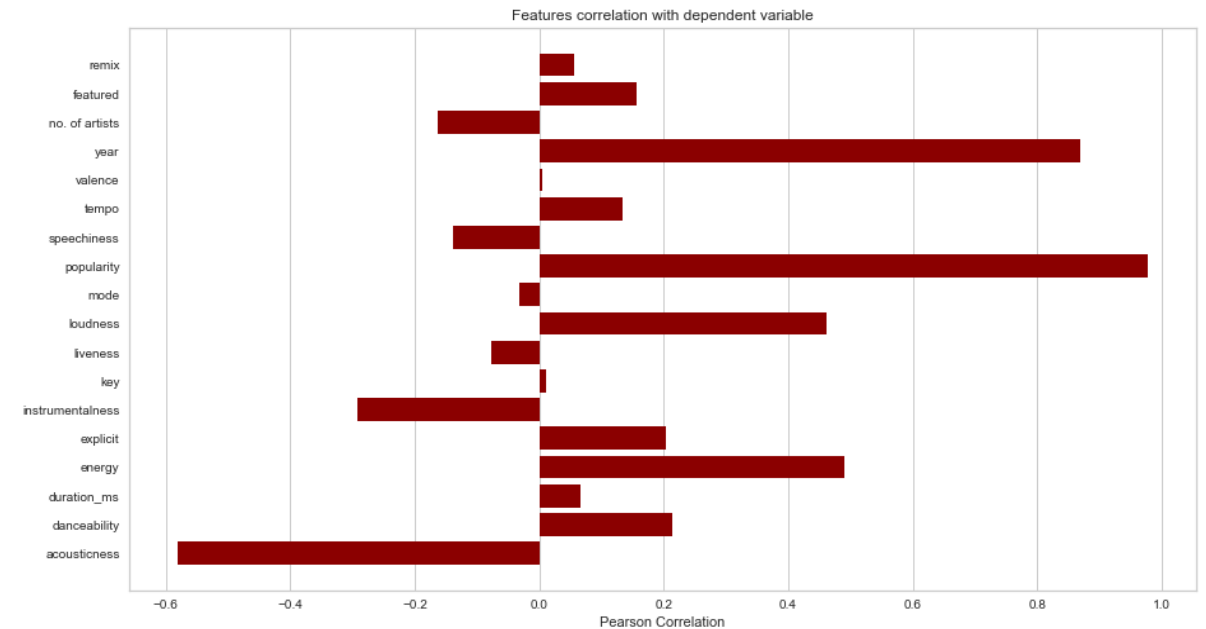
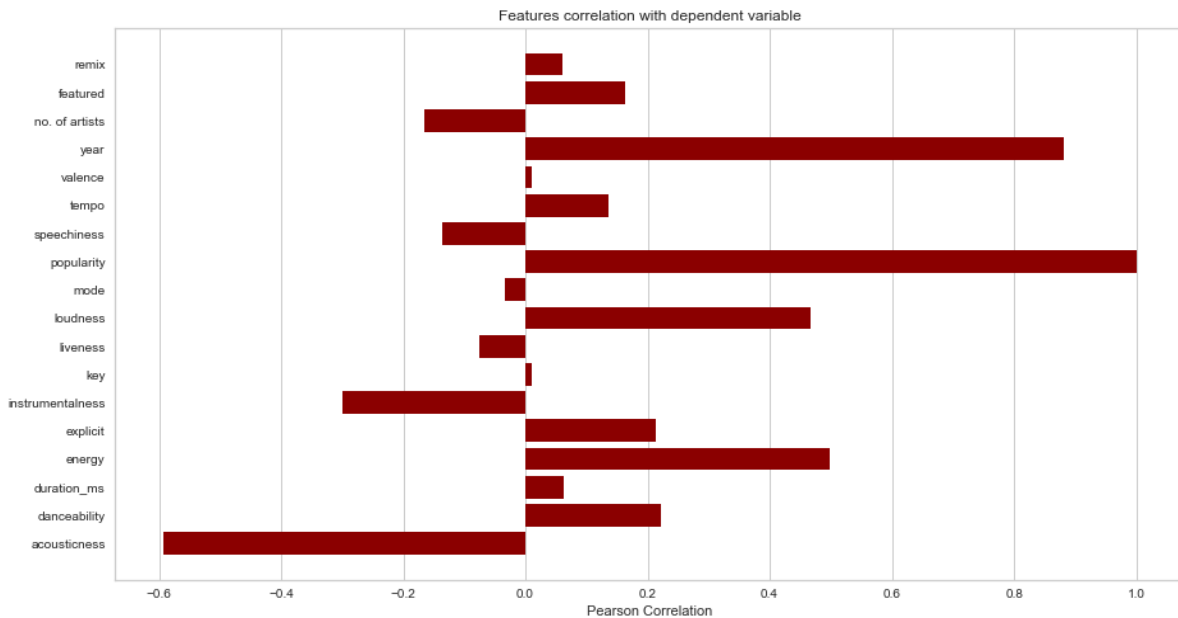
spotify_global_top100

Feat. corre for popularity *NOT BINNED*

Feat. corre for Popularity in top 100 separately *BINNED*

...

...



Binning made the other
positive features more
positive

ML PORTION

GOAL // STRATEGY

TRAIN WITH THE NO_DUPES DATASET AND TEST IT AGAINST THE FULL DATASET
TO SEE HOW WELL IT DOES



ML: MODELS

TRAIN_TEST_SPLIT

```
[456]: a = spotify_global_top100_no_pot_dupes.drop(["popularity", "Popularity in top 100 separately"], axis = 1)
      b = spotify_global_top100_no_pot_dupes["popularity"]

      a_train, a_test, b_train, b_test = train_test_split(a, b ,test_size=0.2, random_state=42)
```

```
[457]: c = spotify_global_top100_no_pot_dupes.drop(["popularity", "Popularity in top 100 separately"], axis = 1)
      d = spotify_global_top100_no_pot_dupes["Popularity in top 100 separately"]

      c_train, c_test, d_train, d_test = train_test_split(c, d ,test_size=0.2, random_state=42)
```

ML: MODELS

STANDARDISE & THEN NORMALIZE

Standardize: mean 0 & SD 0
-> AKA center and 0
Normalize: range from 0-1
-> Force it into a range

Train set -> Fit & Transform
Test set -> Transform

```
[459]: #https://towardsdatascience.com/what-and-why-behind-fit-transform-vs-transform-in-scikit-learn-78f915cf96fe
sc = preprocessing.StandardScaler().fit(a_train[columns_to_std_and_nor]) # -> Standardize -> rescaling the distribution of values so that the mean of observed values is 0 and the standard deviation is 1

a_train[columns_to_std_and_nor] = sc.transform(a_train[columns_to_std_and_nor])
a_test[columns_to_std_and_nor] = sc.transform(a_test[columns_to_std_and_nor])

nc = preprocessing.MinMaxScaler().fit(a_train[columns_to_std_and_nor]) # -> Normalize -> rescaling of the data from the original range so that all values are in the range [0, 1]

a_train[columns_to_std_and_nor] = nc.transform(a_train[columns_to_std_and_nor])
a_test[columns_to_std_and_nor] = nc.transform(a_test[columns_to_std_and_nor])

# nc.transform(a_train[columns_to_std_and_nor])
# nc.transform(a_test[columns_to_std_and_nor])
# Standardization can give values that are both positive and negative centered around zero.
# It may be desirable to normalize data after it has been standardized.
```

...

[461]: a_train		acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	valence	year	no. of artists	featured	remix	no_of_times
	2791	0.76	0.62	0.07	0.57	0	0.00	0.09	0.20	0.82	1	0.37	0.49	0.44	0.28	0.18	0	0	
	7078	0.00	0.43	0.11	0.94	0	0.00	0.64	0.10	0.95	1	0.09	0.72	0.34	0.95	0.00	0	0	
	9674	0.81	0.48	0.06	0.54	0	0.00	0.82	0.27	0.82	0	0.04	0.40	0.73	0.21	0.00	0	0	
	10721	0.91	0.57	0.08	0.34	0	0.00	0.64	0.15	0.85	1	0.05	0.82	0.78	0.15	0.00	0	0	
	8287	0.05	0.73	0.12	0.76	0	0.00	0.27	0.18	0.80	1	0.04	0.51	0.95	0.54	0.09	0	0	

	11964	0.72	0.75	0.06	0.54	0	0.00	0.18	0.07	0.84	1	0.04	0.44	0.81	0.86	0.00	0	0	
	5191	0.02	0.62	0.10	0.79	0	0.00	0.18	0.05	0.82	1	0.03	0.45	0.78	0.62	0.00	0	0	
	5390	0.00	0.63	0.11	0.75	0	0.00	0.64	0.38	0.90	1	0.07	0.43	0.44	0.86	0.00	0	0	
	860	0.04	0.47	0.12	0.75	0	0.00	0.09	0.13	0.90	1	0.03	0.81	0.12	0.90	0.00	0	0	
	7270	1.00	0.45	0.09	0.14	0	0.36	0.18	0.09	0.79	1	0.03	0.34	0.18	0.11	0.00	0	0	

10640 rows × 18 columns



ML: MODELS

STANDARDISE
& THEN
NORMALIZE

- Fit -> Calc. mean & var of each features present
- transform -> transforming all the features using the respective mean and variance.
- If we fit test data to test -> new mean and variance that is a new scale for each feature towards test and will let our model learn about our test data too

ML: MODELS

This is the column by column fit transform

STANDARDISE
& THEN
NORMALIZE

```
[438]: ### No need to run this 1 by 1.... it is literally the same result as below.  
# for i in columns_to_std_and_nor: #to normalize and standardize individually without touching the binaries.  
#     a_train[i] = StandardScaler().fit_transform(np.array(a_train[i]).reshape(-1, 1))  
#     a_train[i] = MinMaxScaler().fit_transform(np.array(a_train[i]).reshape(-1, 1))  
#     a_train[i] = sc.fit_transform(a_train)  
#     a_train[i] = nc.fit_transform(a_train)
```

```
[439]: a_train
```

```
[439]:
```

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	speechiness	tempo	valence	year	no. of artists	featured	remix	no_c
2791	0.76	0.62	0.07	0.57	0	0.00	0.09	0.20	0.82	1	0.37	0.49	0.44	0.28	0.18	0	0	
7078	0.00	0.43	0.11	0.94	0	0.00	0.64	0.10	0.95	1	0.09	0.72	0.34	0.95	0.00	0	0	
9674	0.81	0.48	0.06	0.54	0	0.00	0.82	0.27	0.82	0	0.04	0.40	0.73	0.21	0.00	0	0	
10721	0.91	0.57	0.08	0.34	0	0.00	0.64	0.15	0.85	1	0.05	0.82	0.78	0.15	0.00	0	0	
8287	0.05	0.73	0.12	0.76	0	0.00	0.27	0.18	0.80	1	0.04	0.51	0.95	0.54	0.09	0	0	
...	
11964	0.72	0.75	0.06	0.54	0	0.00	0.18	0.07	0.84	1	0.04	0.44	0.81	0.86	0.00	0	0	
5191	0.02	0.62	0.10	0.79	0	0.00	0.18	0.05	0.82	1	0.03	0.45	0.78	0.62	0.00	0	0	
5390	0.00	0.63	0.11	0.75	0	0.00	0.64	0.38	0.90	1	0.07	0.43	0.44	0.86	0.00	0	0	
860	0.04	0.47	0.12	0.75	0	0.00	0.09	0.13	0.90	1	0.03	0.81	0.12	0.90	0.00	0	0	
7270	1.00	0.45	0.09	0.14	0	0.36	0.18	0.09	0.79	1	0.03	0.34	0.18	0.11	0.00	0	0	

10640 rows × 18 columns

ML: MODELS

LAZY

Okay not very
sure how to
deal with
multiclass
label ..

So I will stick
with the
Binary one!

for 'popularity' Label

...

100% ██████████ 29/29 [00:27<00:00, 1.06it/s]

for 'popularity' Label

[468]:

	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
Model					
XGBClassifier	0.96	0.93	0.93	0.96	0.74
BaggingClassifier	0.95	0.92	0.92	0.96	0.38
RandomForestClassifier	0.96	0.92	0.92	0.96	1.81
LGBMClassifier	0.96	0.92	0.92	0.96	0.17
AdaBoostClassifier	0.95	0.89	0.89	0.95	0.65
DecisionTreeClassifier	0.95	0.88	0.88	0.95	0.08
ExtraTreesClassifier	0.95	0.88	0.88	0.95	0.71
NearestCentroid	0.85	0.86	0.86	0.88	0.04
SVC	0.95	0.85	0.85	0.95	1.89
ExtraTreeClassifier	0.93	0.83	0.83	0.93	0.03
BernoulliNB	0.85	0.82	0.82	0.87	0.04
LabelSpreading	0.93	0.81	0.81	0.93	9.46
LabelPropagation	0.93	0.81	0.81	0.93	6.78
LinearDiscriminantAnalysis	0.91	0.79	0.79	0.92	0.14
PassiveAggressiveClassifier	0.91	0.79	0.79	0.91	0.05
KNeighborsClassifier	0.92	0.79	0.79	0.92	1.36
LogisticRegression	0.93	0.78	0.78	0.93	0.11
CalibratedClassifierCV	0.93	0.77	0.77	0.92	1.97
LinearSVC	0.93	0.76	0.76	0.92	0.53
SGDClassifier	0.93	0.75	0.75	0.92	0.09

for 'Popularity in top 100 separately' Label

...

100% ██████████ 29/29 [01:47<00:00, 3.70s/it]

for 'Popularity in top 100 separately' Label

]:

	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
Model					
RandomForestClassifier	0.42	0.44	None	0.40	3.33
LGBMClassifier	0.42	0.43	None	0.39	2.52
XGBClassifier	0.41	0.42	None	0.39	8.30
ExtraTreesClassifier	0.41	0.42	None	0.39	1.54
BaggingClassifier	0.40	0.41	None	0.38	0.84
SVC	0.39	0.40	None	0.36	10.56
NuSVC	0.36	0.38	None	0.34	22.86
DecisionTreeClassifier	0.36	0.37	None	0.36	0.20
LogisticRegression	0.34	0.36	None	0.32	0.66
LinearSVC	0.33	0.34	None	0.29	8.21
CalibratedClassifierCV	0.33	0.34	None	0.30	29.16
KNeighborsClassifier	0.32	0.33	None	0.31	1.28
LinearDiscriminantAnalysis	0.32	0.33	None	0.30	0.06
RidgeClassifier	0.31	0.32	None	0.27	0.05
RidgeClassifierCV	0.31	0.32	None	0.27	0.06
LabelPropagation	0.30	0.31	None	0.30	6.58
ExtraTreeClassifier	0.30	0.31	None	0.30	0.04
LabelSpreading	0.30	0.31	None	0.30	9.36
NearestCentroid	0.28	0.29	None	0.27	0.02
BernoulliNB	0.28	0.29	None	0.26	0.04
SGDClassifier	0.27	0.29	None	0.24	0.57



ML: MODEL SELECTION

RANDOM FOREST

Reasons:

- Lazy predict has very good accuracy, AUC & F1
- able to deal with imbalanced dataset with `class_weight = "balanced"`
- No sensitive to outliers (but I alrdy standardised and normalised)
- It is an ensemble of DTs -> Typically more depth -> Less overfitting
 - > AKA bagging -> reduce the complexity of models which will overfit.

ML: TRAINING

RANDOM FOREST

spotify_global_top100_no_pot_dupes

Important features

number of train sample in train set: (10640, 18)

Number of samples in validation set: (2661,)

TRAINing with RF.score: 99.92

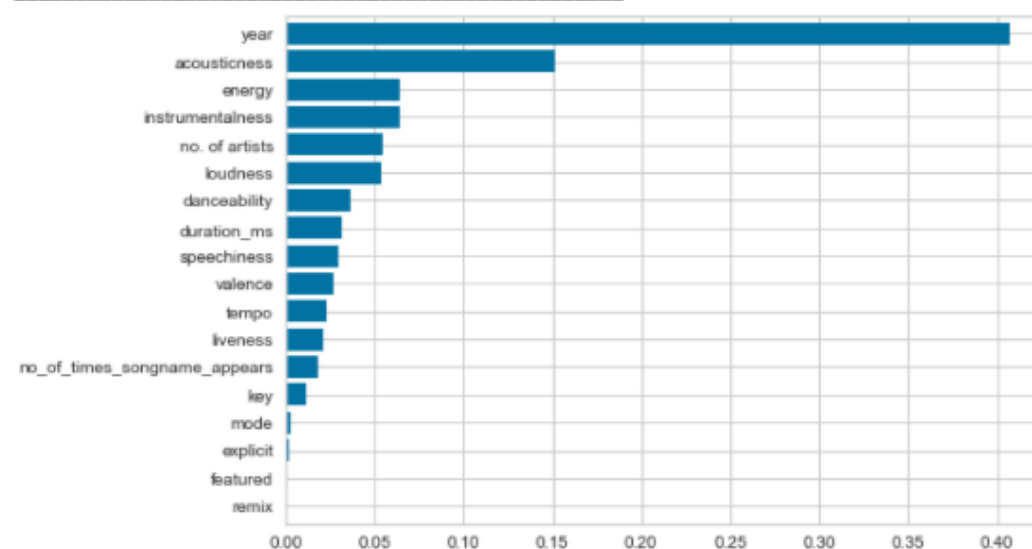
TESTing with RF.score: 95.83

Accuracy: 95.83 ^^ same as above since they call the same function

Precision: 98.15

Recall: 97.21

F1 score: 97.68



looks really good tbh

Can proceed to tune

ML: TRAINING

RANDOM FOREST

Important features

number of train sample in train set: (10640, 18)

Number of samples in validation set: (2661,)

TRAINing with RF.score: 99.19

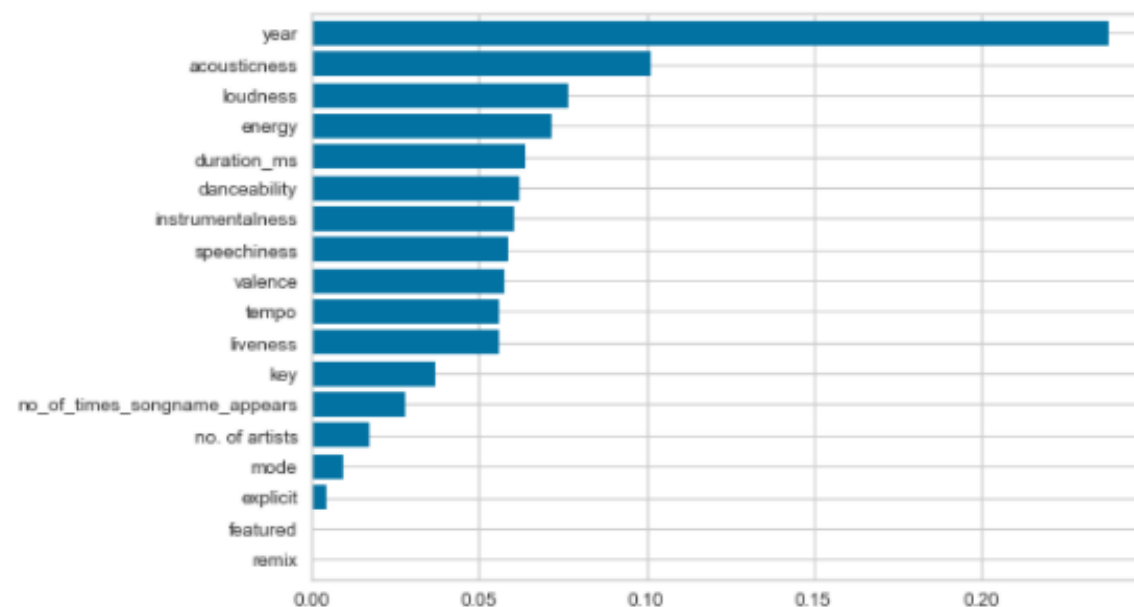
TESTing with RF.score: 41.41

Accuracy: 41.41 ^^ same as above since they call the same function

Precision: 39.92

Recall: 41.41

F1 score: 39.43



Looks like overfitting.... lets abandon it as it takes too much time

ML: TRAINING

OVER AND
UNDER SAMPLING

#SMOTE
#TOMEK

```
[700]: b_train
```

2791	0
7078	1
9674	1
10721	1
8287	1
..	..
11964	1
5191	1
5390	1
860	1
7270	1

Name: popularity, Length: 10

```
[595]: from imblearn.combine import  
from numpy import where  
counter = Counter(b_train)
```

Tomek links are the opposite class paired samples that are the closest neighbors to each other.

Before, the Counter({1: 9453, 0: 1187})
after, the Counter({0: 9449, 1: 9449})

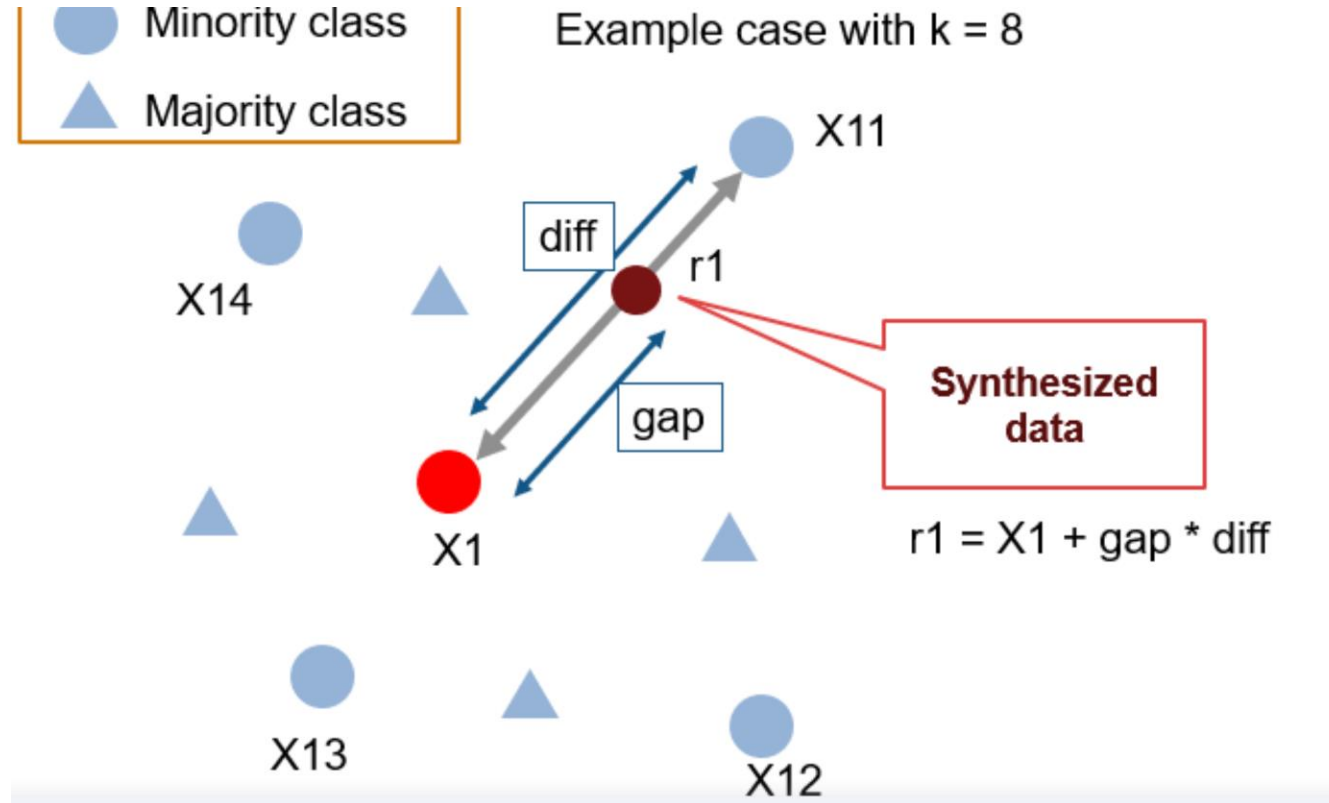


Smoting minority 1187 to
9453 and
Removing Tomek links on
both classes to 9449 to
prevent overfitting

ML: TRAINING

OVER AND
UNDER SAMPLING

#SMOTE
#TOMEK



Pictorial on smoting
specifically

ML: TRAINING

OVER AND UNDER
SAMPLING

#SMOTE
#TOMEK

Important features

number of train sample in train set: (18898, 18)

Number of samples in validation set: (2661,)

TRAINing with RF.score: 99.95

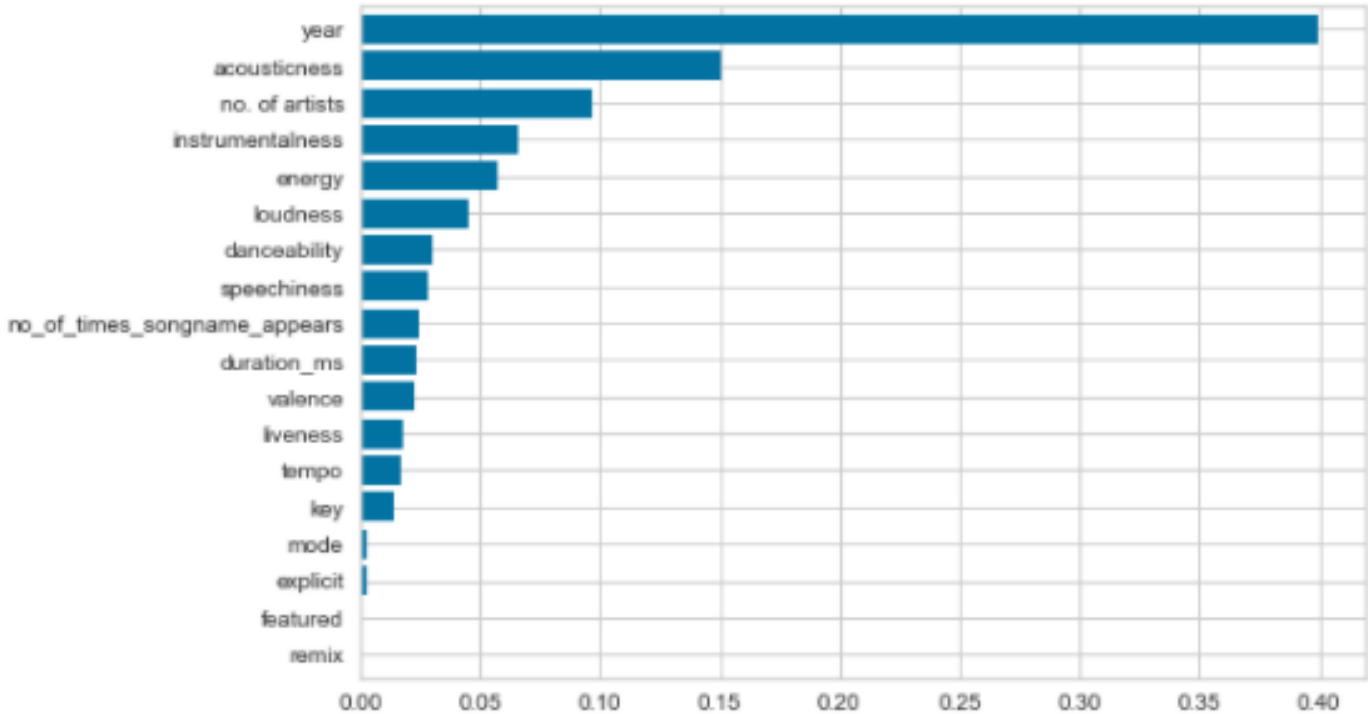
TESTing with RF.score: 95.15

Accuracy: 95.15 ^^ same as above since they call the same function

Precision: 99.22

Recall: 95.38

F1 score: 97.26



ML: TRAINING

OVER AND UNDER
SAMPLING

#SMOTE
#TOMEK

Lets use the RF class weight balanced set

Our classifier is pickier and has **more precision** but misses a slightly bit more on the actual songs that do hit top 100.

IE **sacrificed 1.83 recall** for 0.93 precision

From an more objective point of view, the `class_weight` balanced RF is better as it has a better ratio between Precision and recall.

- aka better F1 score

but it must be noted in certain industries where money is involved the higher precision will be preferred over the recall trade off.

But since we're in Spotify and we want a more balanced approach to seeing which songs can hit the top 100.

ML: TRAINING

HYPER-
PARAMETER
TUNING

#GRIDSEARCHCV

```
[640]: grid_model_v8.fit(a_train, b_train)
```

Fitting 12 folds for each of 22 candidates, totalling 264 fits

[Parallel(n_jobs=16)]: Using backend LokyBackend with 16 concurrent workers.

[Parallel(n_jobs=16)]: Done 18 tasks | elapsed: 1.1min

[Parallel(n_jobs=16)]: Done 168 tasks | elapsed: 9.8min

[Parallel(n_jobs=16)]: Done 264 out of 264 | elapsed: 16.2min finished

```
[640]: GridSearchCV(cv=RepeatedStratifiedKFold(n_repeats=3, n_splits=4, random_state=42),
                  estimator=RandomForestClassifier(class_weight='balanced',
                                                    random_state=42),
                  n_jobs=16,
                  param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [14],
                              'n_estimators': [800, 810, 820, 830, 840, 850, 860,
                                                870, 880, 890, 900]},
                  verbose=1)
```

Looks more or less finalized..... lets use V8

```
[642]: grid_model_v8.best_estimator_
```

```
[642]: RandomForestClassifier(class_weight='balanced', criterion='entropy',
                             max_depth=14, n_estimators=840, random_state=42)
```

ML: TRAINING

RUN AGAIN

Important features

number of train sample in train set: (10640, 18)

Number of samples in validation set: (2661,)

Training with RF.score: 98.83

Testing with RF.score: 95.79

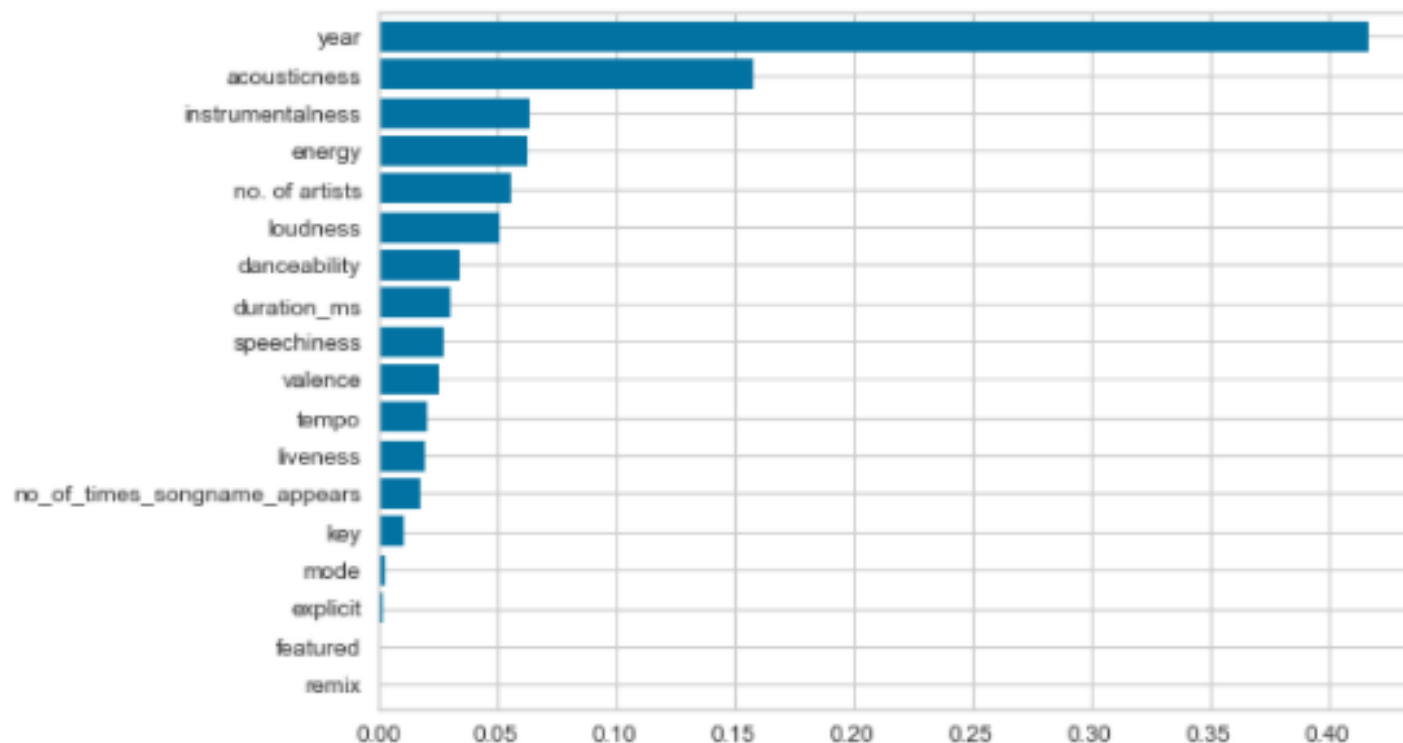
Testing with Accuracy: 95.79

Precision: 98.68

Recall: 96.63

F1 score: 97.64

^ Same as RF.score since they call the same function



ML: MODEL EVALUATION

ACCURACY
PRECISION,
RECALL,
F1

Accuracy -> Correctness

- $(tp + tn) / (p + n)$

Precision -> Exactness

- $tp / (tp + fp)$

Recall -> Completeness

- $tp / (tp + fn)$

F1 -> how complete & exact IE balance of precision and recall

- $2 tp / (2 tp + fp + fn)$

Dropped from 99.92 to 98.83 for accuracy

- -1.09

precision upped from 98.15 to 98.68

- +0.53

recall dropped from 97.21 to 96.63

- -0.58

F1 dropped 0.04



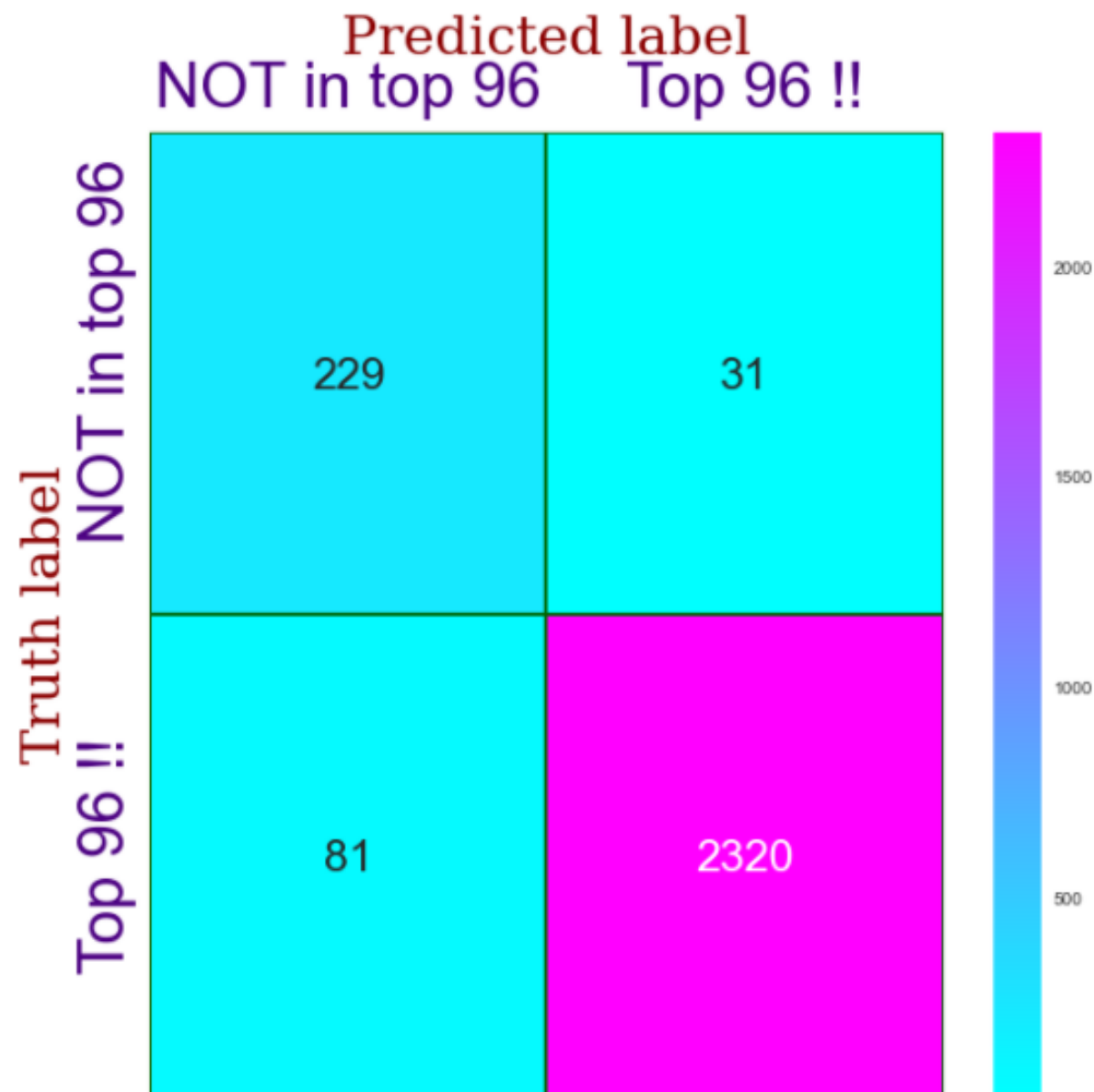
This should have mitigated any overfitting that might have occurred

ML: MODEL EVALUATION

ACCURACY
PRECISION,
RECALL,
F1

Heat map of test data
against predictions with
test data

Very decent TNs & TPs



ML: TEST ON BIG SET

Important features

number of train sample in train set: (127431, 18)

Number of samples in validation set: (42478,)

TRAINing with RF.score: 95.65

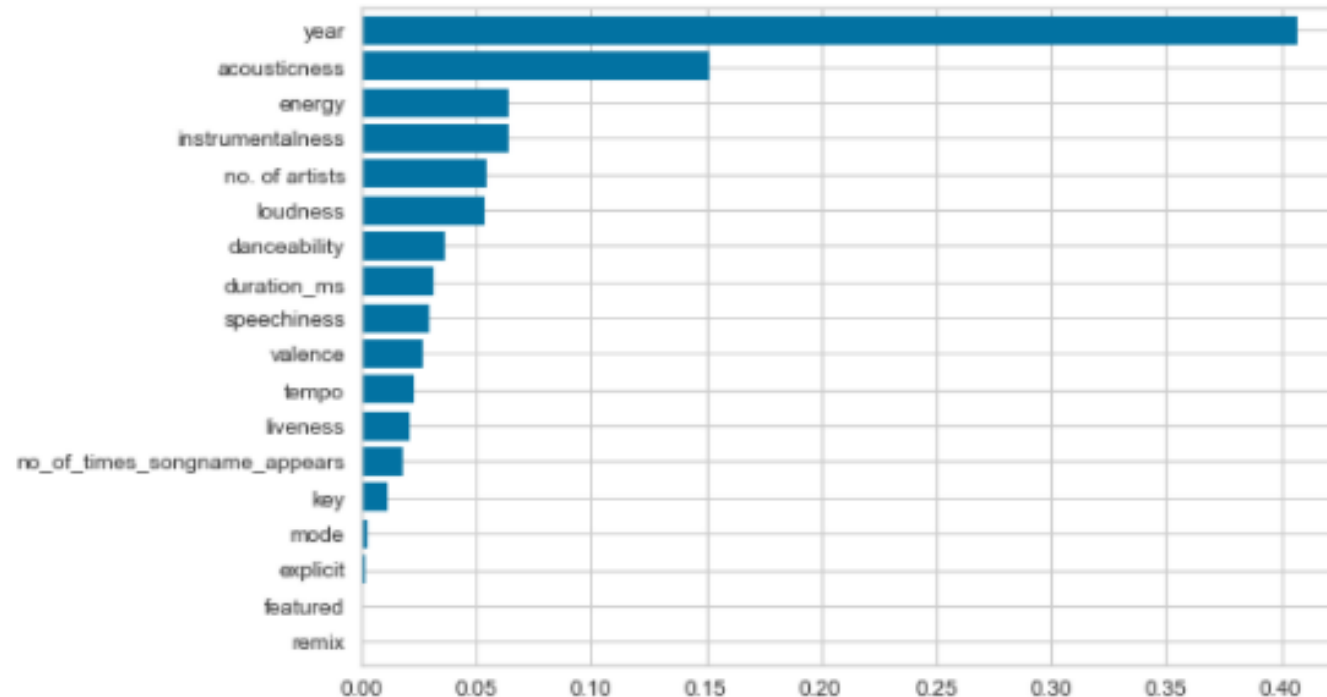
TESTing with RF.score: 94.59

Accuracy: 94.59 ^^ same as above since they call the same function

Precision: 99.56

Recall: 93.97

F1 score: 96.68



Generalizes well! Very good!

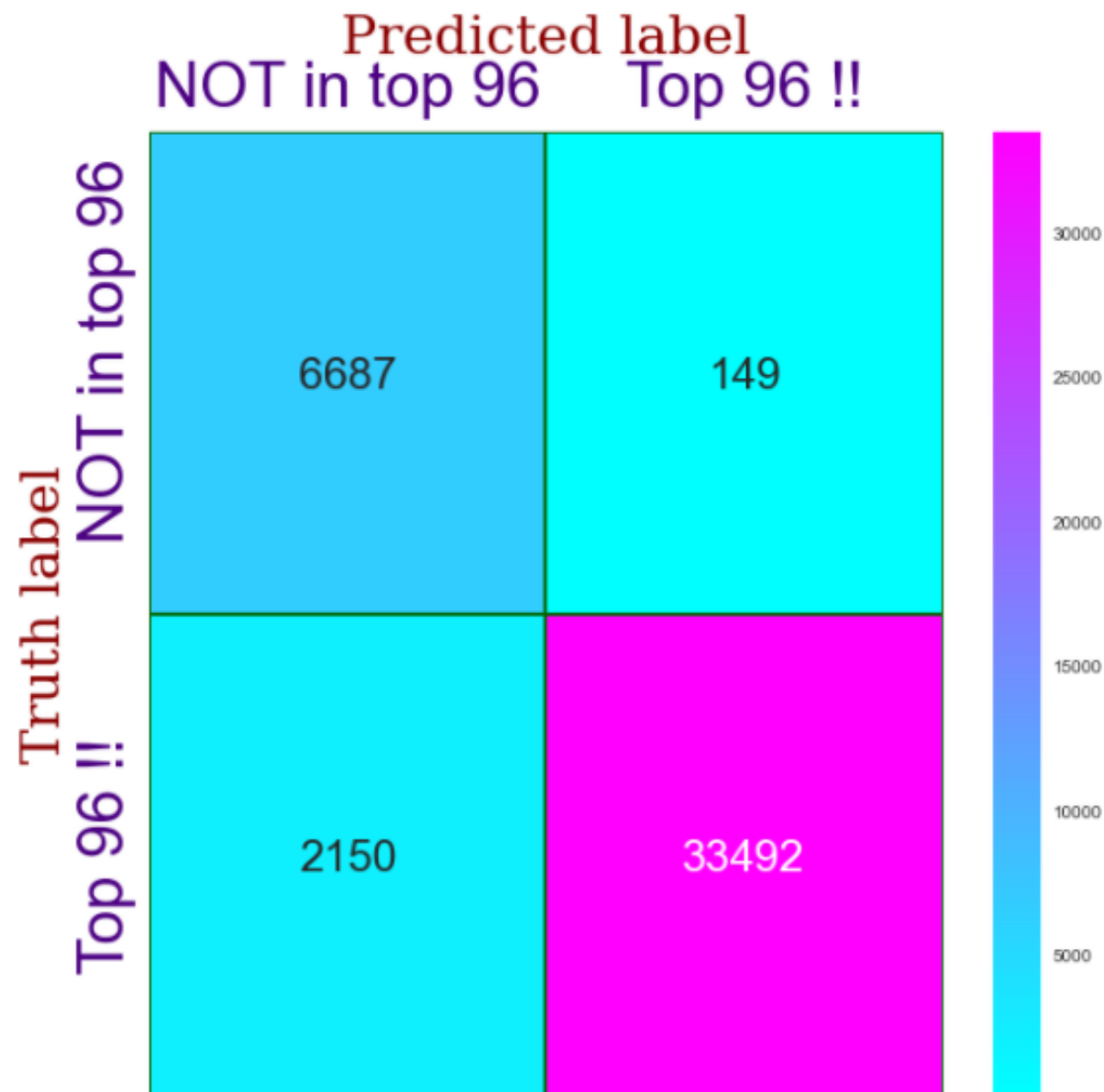
In hindsight I could have easily parsed out the no dupes set from the original 169k data set to make the results even more convincing.

ML: MODEL EVALUATION

ACCURACY
PRECISION,
RECALL,
F1

Heat map of test data
against predictions with
test data

Still Very decent TNs &
TPs





END OF PRESENTATION

<https://github.com/lolasery/khdatasci>

<https://www.kaggle.com/ektanegi/spotifydata-19212020>