Popularity Prediction

Guy Kahana & Anat Peled

NAYA College

10/2020

Project III: Regression Models in ML

Why do we care?

- There are several reasons to predict a track's popularity in Spotify:
  - **Commercial:**
    - Advertisements
    - Design promoting algorithms
  - **Financial:**
    - Increase revenues from advertisers
    - increase commissions
  - **Musical:**
    - Engineering of a Spotify hit

The Dataset

```
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   acousticness      169909 non-null  float64
 1   artists           169909 non-null  object
 2   danceability      169909 non-null  float64
 3   duration_ms       169909 non-null  int64
 4   energy            169909 non-null  float64
 5   explicit          169909 non-null  int64
 6   id                169909 non-null  object
 7   instrumentalness  169909 non-null  float64
 8   key               169909 non-null  int64
 9   liveness          169909 non-null  float64
 10  loudness          169909 non-null  float64
 11  mode              169909 non-null  int64
 12  name              169909 non-null  object
 13  popularity        169909 non-null  int64
 14  release_date      169909 non-null  object
 15  speechiness       169909 non-null  float64
 16  tempo             169909 non-null  float64
 17  valence           169909 non-null  float64
 18  year              169909 non-null  int64
dtypes: float64(9), int64(6), object(4)
memory usage: 24.6+ MB
```

- Spotify Dataset 1921-2020, 160k+ Tracks: Kaggle, 06/2020, generated by Yamaç Eren Ay with Spotify API
- The goal: predict the popularity (0-100) of a track according to the track's features.
- > 160,000 tracks
- 19 features

Preliminary Cleaning

- From 19 to 16 features

```
id            : 169909 unique values
name          : 132940 unique values
artists       : 33375 unique values
release_date  : 10882 unique values
year          : 100 unique values
```

- Remove 669 duplicates

```
df = df[~df.duplicated()==1]
df.shape
```

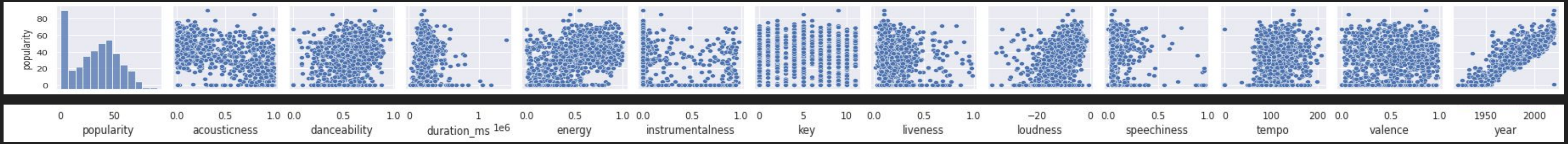
```
(169240, 16)
```

Overview

Correlation between numerical features (abs.)

Overview

Pairplot of Numerical Features exc. dummies

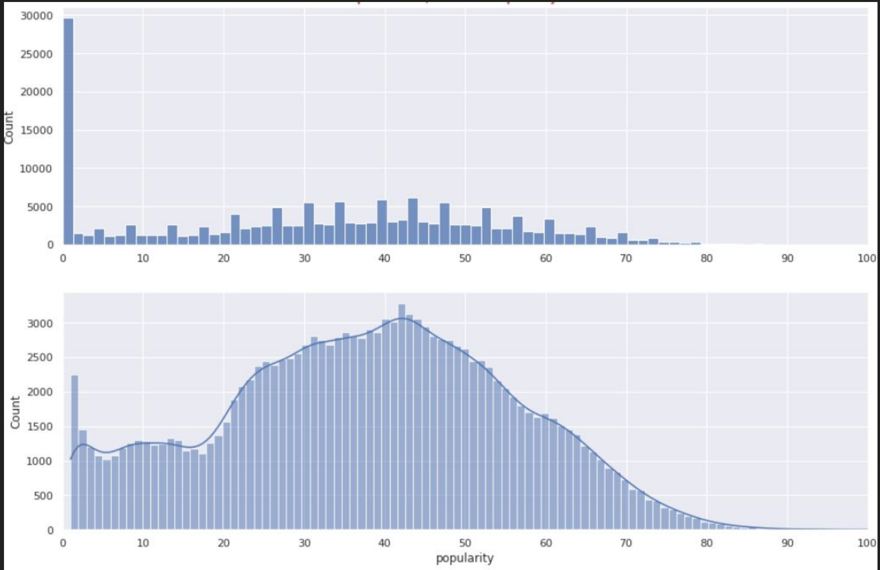


Popularity: The target

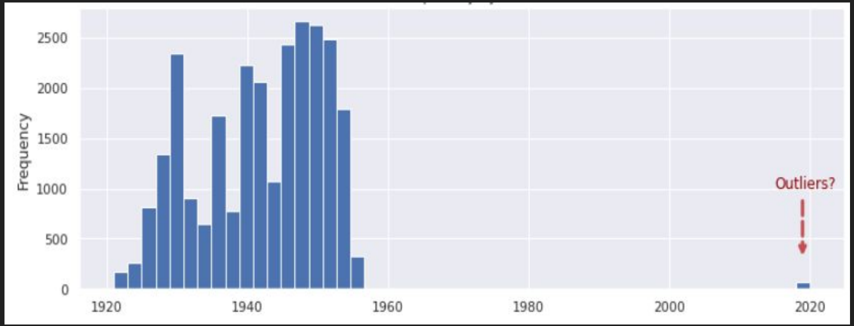
Mean: 31.66 Median: 34 STD: 21.53

All Data

Popularity > 0



◀ **Who gets 0 popularity?**



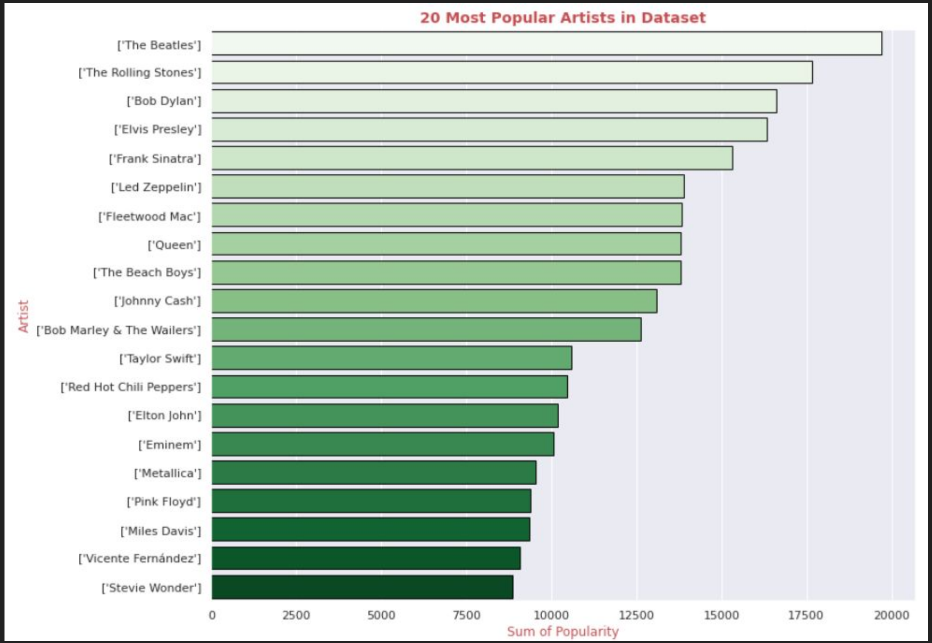

```
[24] 1 df.loc[(df['popularity']==0)&(df['year']==2020),'year'].count()
```

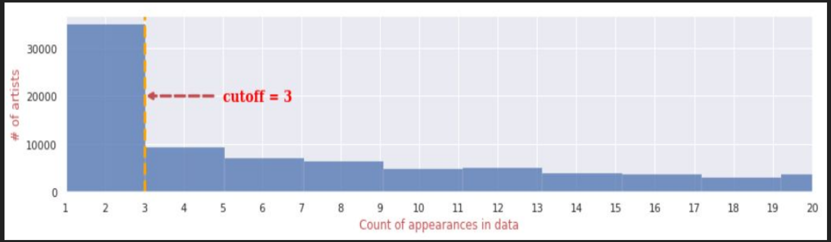
78

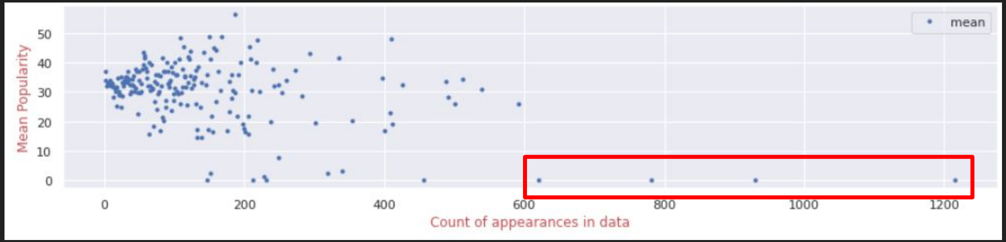
Categorical Features

Artists

- 33375 unique values
- Target Encoding: average the target value by category link







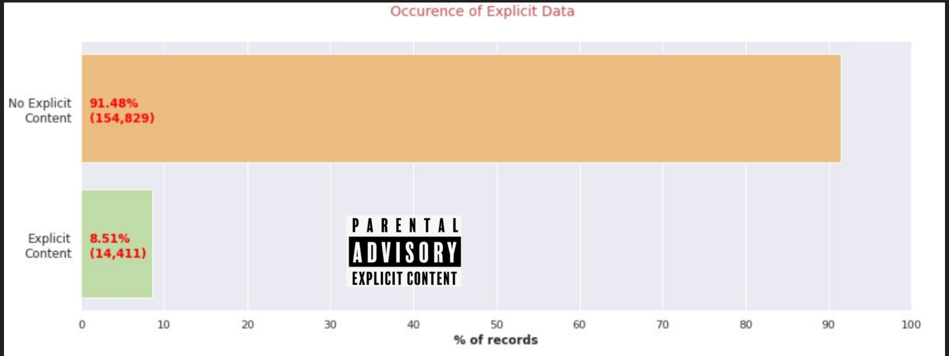
Artists

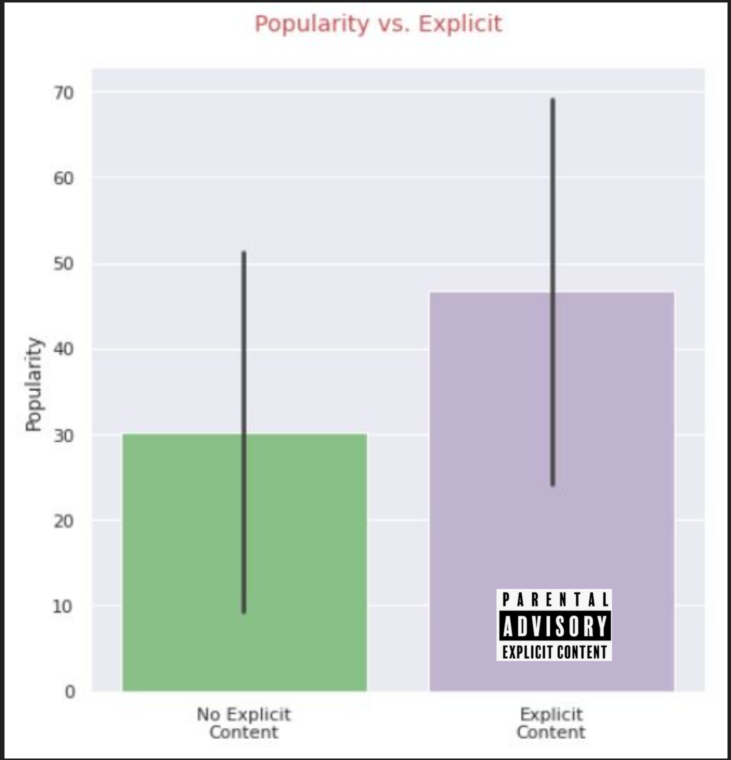
- Target Encoding:
  - <= 3 times = general mean
  - > 600 times = 0
  - else: unique mean
- MinCnt & MaxCnt set us parameters

```python
class ArtistsTransformer():
  """ This transformer recives a DF with a feature 'artists' of dtype object
      and convert the feature to a float value as follows:
      1. Replace the data with the artists mean popularity
      2. Replace values where artists appear less than MinCnt with y.mean()
      3. Replace values where artists appear more than MaxCnt with 0

      PARAMETERS:
      ----------
      MinCnt (int): Minimal treshold of artisits apear in dataset, default = 3
      MaxCnt (int): Maximal treshold of artisits apear in dataset, default = 600

      RERTURN:
      ----------
      A DataFrame with converted artists str feature to ordinal floats
  """

  def __init__(self, MinCnt = 3.0, MaxCnt = 600.0):
      self.MinCnt = MinCnt
      self.MaxCnt = MaxCnt
      self.artists_df = None

  def fit (self, X, y):
      self.artists_df =  y.groupby(X.artists).agg(['mean', 'count'])
      self.artists_df.loc['unknown'] = [y.mean(), 1]
      self.artists_df.loc[self.artists_df['count'] <= self.MinCnt, 'mean'] = y.mean()
      self.artists_df.loc[self.artists_df['count'] >= self.MaxCnt, 'mean'] = 0
      return self

  def transform(self, X, y=None):
      X['artists'] = np.where(X['artists'].isin(self.artists_df.index), X['artists'], 'unknown')
      X['artists'] = X['artists'].map(self.artists_df['mean'])
      return X
```

Explicit (binary)

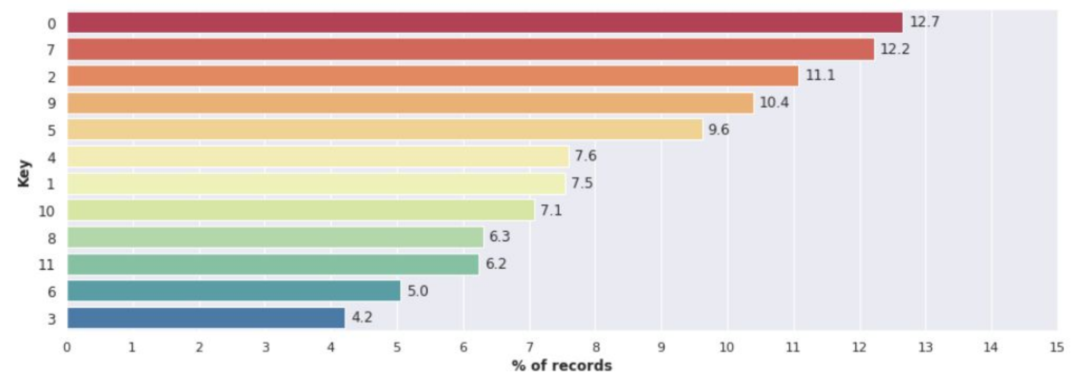
- Warning label (abb. PAL)
- No explicit = 0, Explicit content = 1
- Linear corr. to popularity = 0.2134
- Explicit = higher mean popularity?

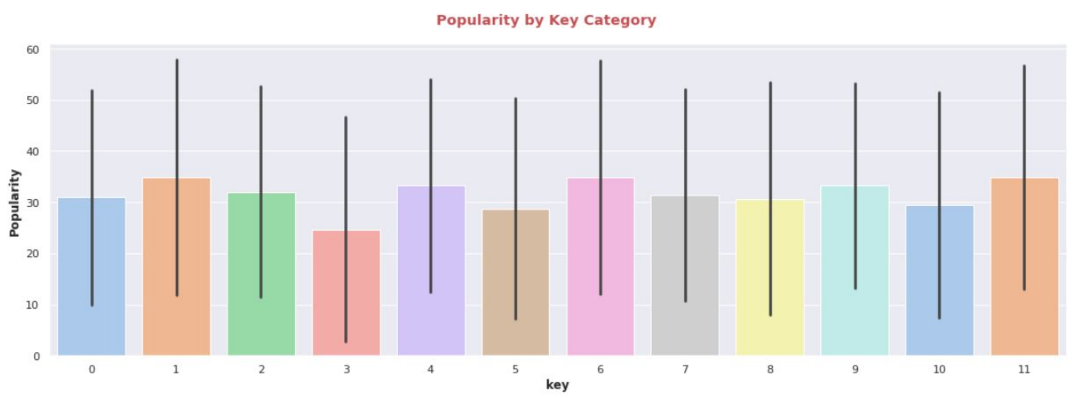




No Action Required

Key (0-11)

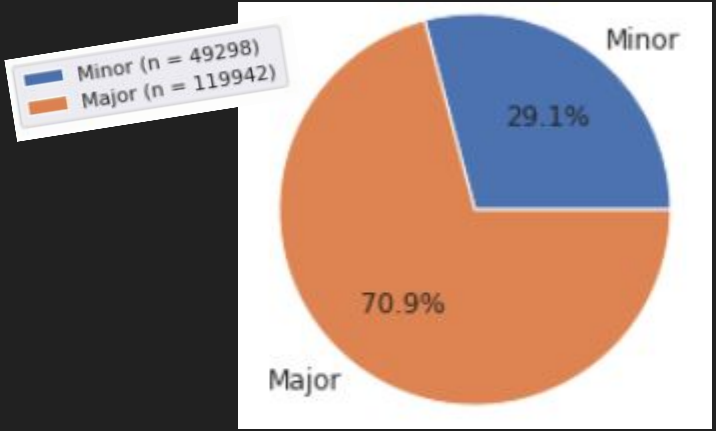




- 4-13% disposition
- Key's popularity STD is ~ 20,
- Key 3 has the lowest mean (24.67)

Mode (binary)

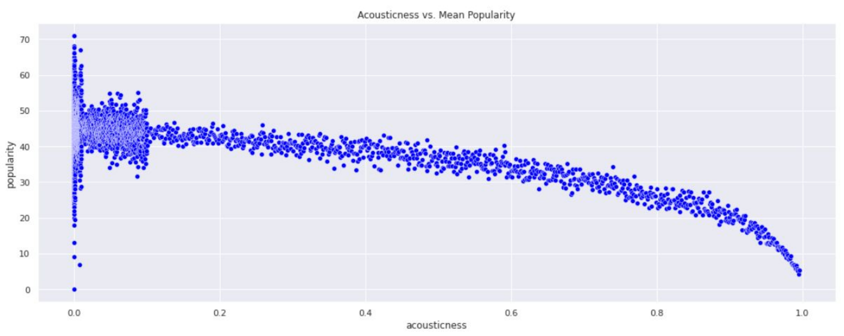
- Warning label (abb. PAL)
- Minor = 0, Major = 1
- Linear corr. to popularity = -0.033

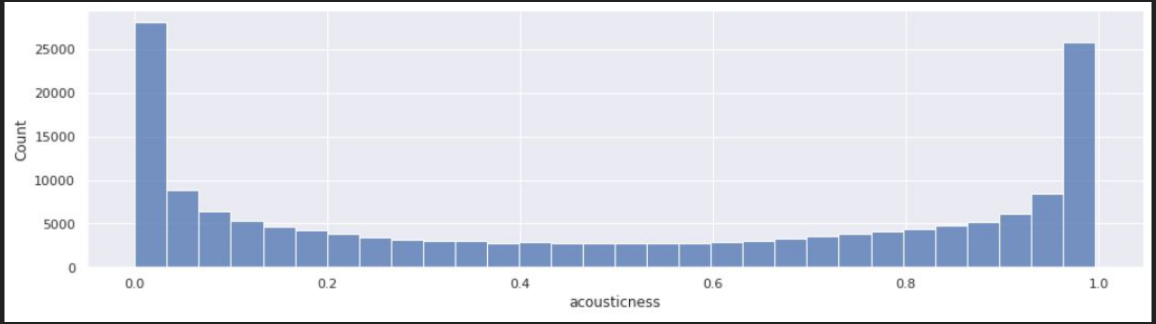


Numerical Features

Acousticness

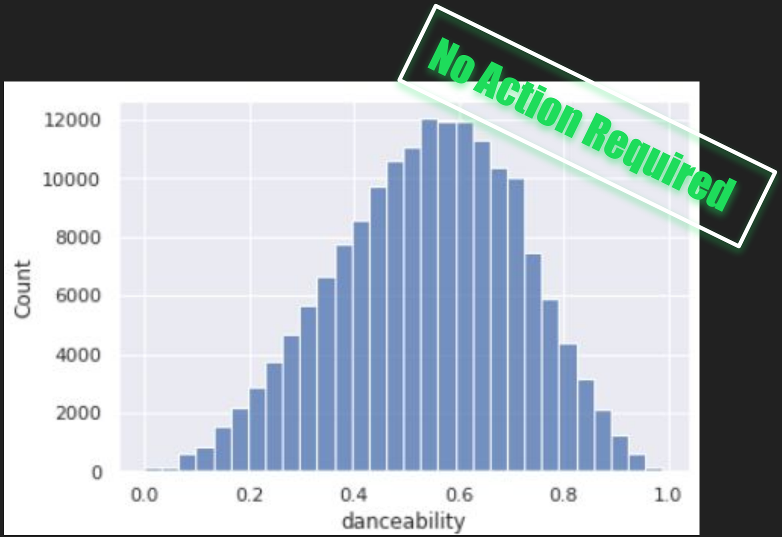
- Higher acousticness means lower mean popularity (apply for acousitcness > 0.1)
- Linear corr. to popularity = -0.5
- Majority of tracks is either close to 0 or 1



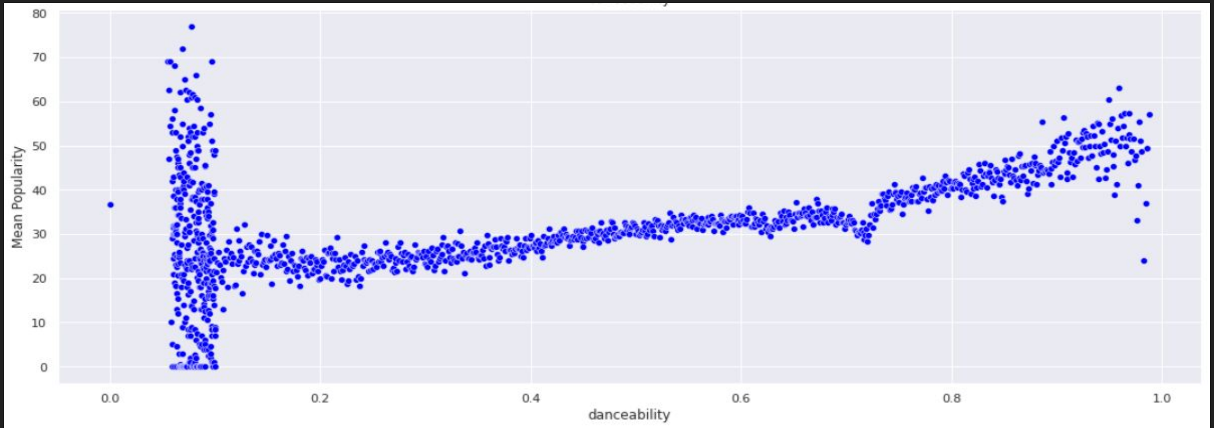


Dancebility

- Is the song danceable based on the combination of tempo, rhythm and beat strength
- Linear corr. to popularity = 0.22



count	169240.000000
mean	0.538717
std	0.175194
min	0.000000
25%	0.418000
50%	0.549000
75%	0.668000
max	0.988000



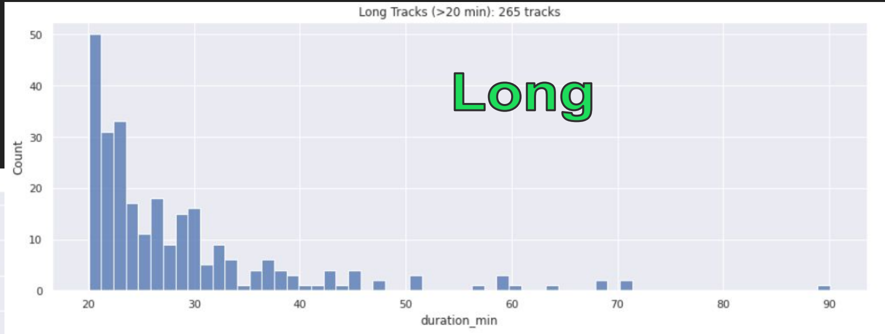
Duration_ms

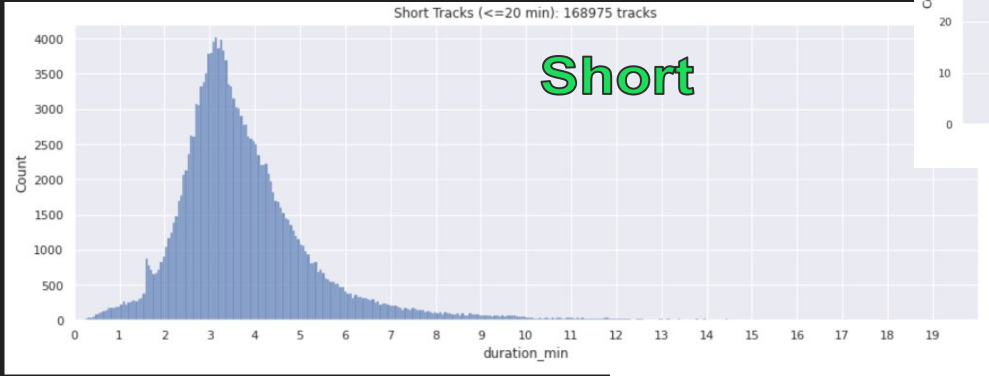
- duration_min = duration_ms / 60,000
- Some tracks are unexpectedly long…
- to understand the data it was divided to long tracks and short tracks:

```
long_tracks = df.loc[df['duration_min']>20]
short_tracks = df.loc[df['duration_min']<=20]
```

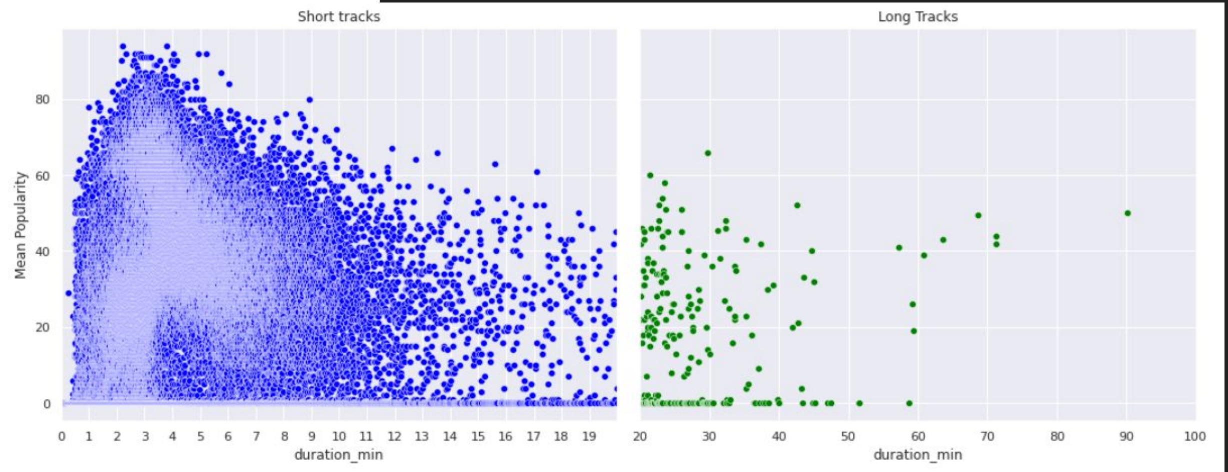
count	169240.000000
mean	3.857319
std	2.018018
min	0.085133
25%	2.852946
50%	3.477333
75%	4.382450
max	90.058333

Duration_ms





- Long tracks are mainly classic music and atmosphere music
- There are not enough samples of tracks longer than 45 minutes

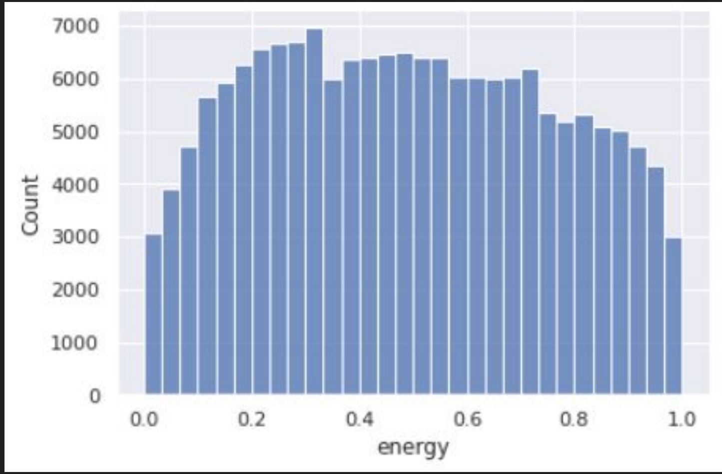


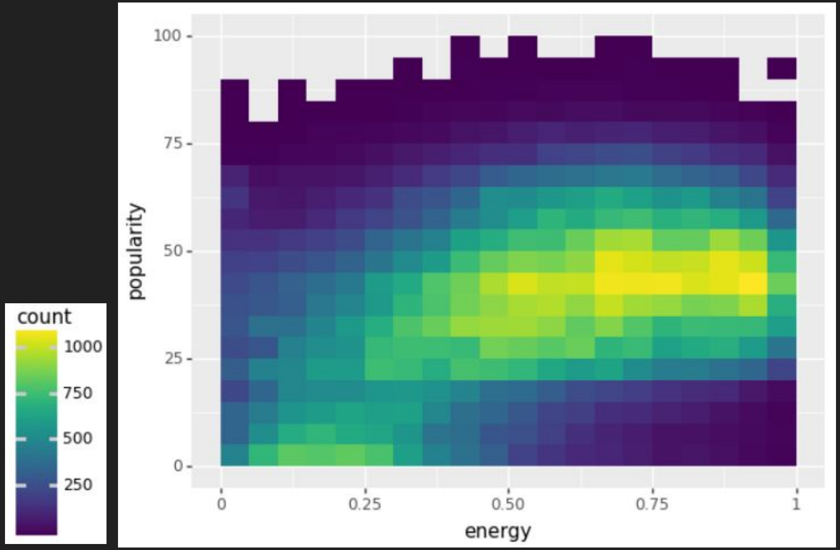
Energy

- Measure the intensity and activity, energetic track feels more fast loud and noisy
- Linear corr. to Popularity = 0.495

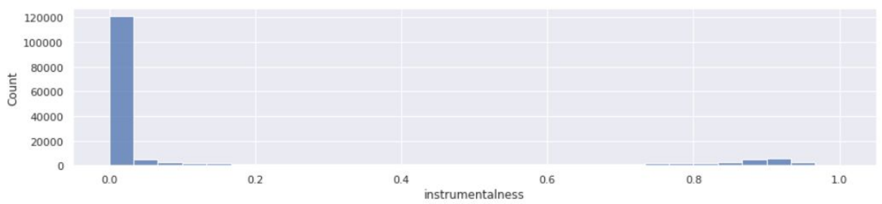
count	169240.000000
mean	0.489632
std	0.267099
min	0.000000
25%	0.264000
50%	0.482000
75%	0.711000
max	1.000000

No Action Required



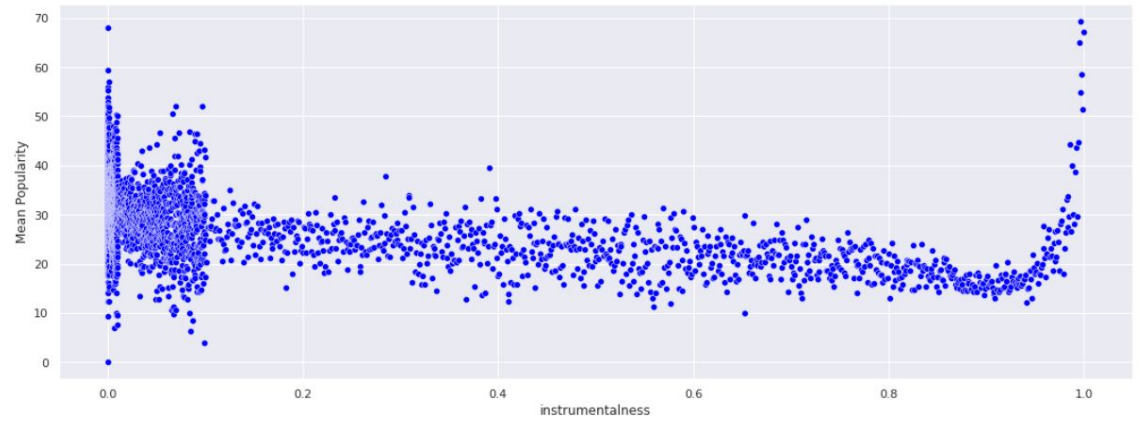


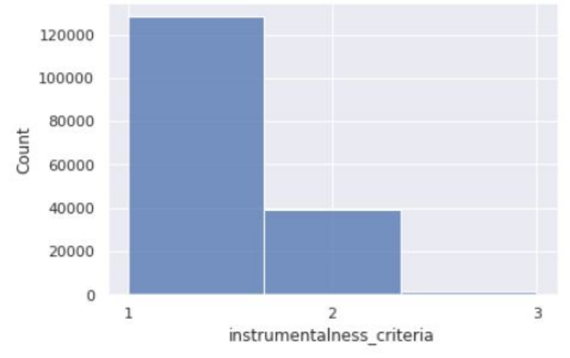
Instrumentalness



- The closer to 1 the greater likelihood the track contain NO vocals
- Linear corr. to popularity = -0.29

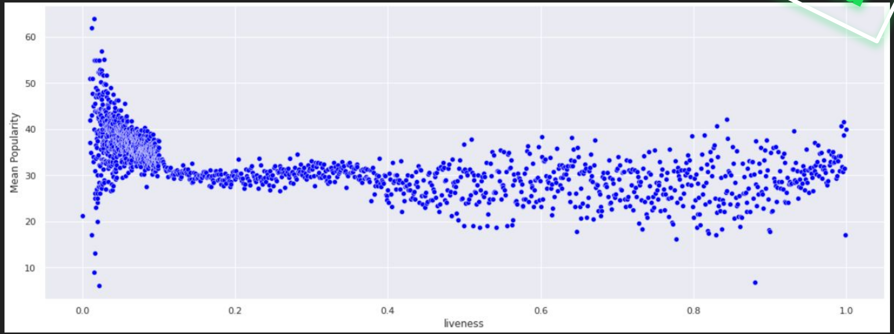
```
criteria= [df['instrumentalness'].between(0, 0.1),
           df['instrumentalness'].between(0.1000001, 0.95),
           df['instrumentalness'].between(0.950001, 1)]
values = [1, 2, 3]
df['instrumentalness_criteria'] = np.select(criteria, values, 0)
```

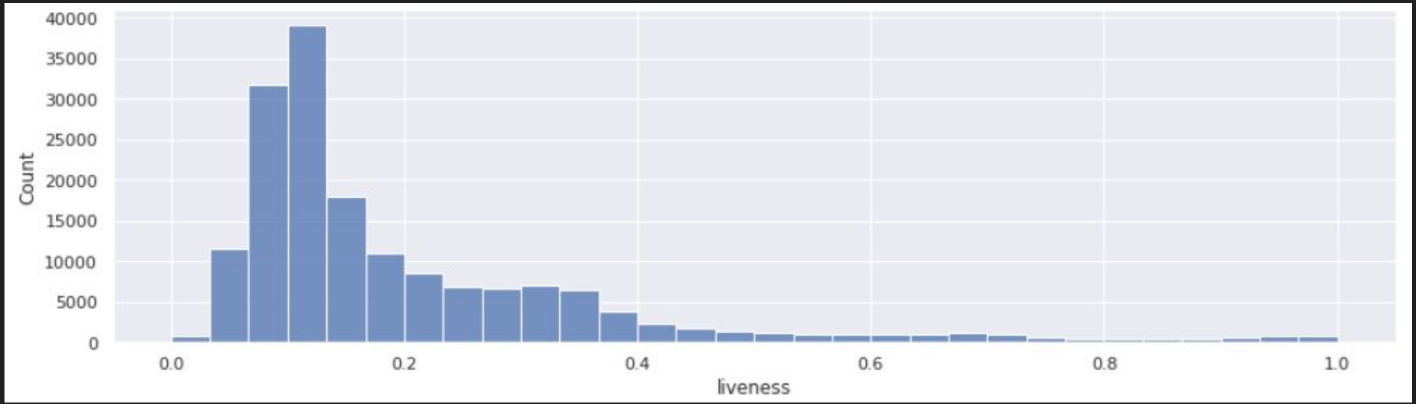




Liveness

- Detect the presence of audience, high livness represent that the track was live
- Linear corr. to popularity = -0.076

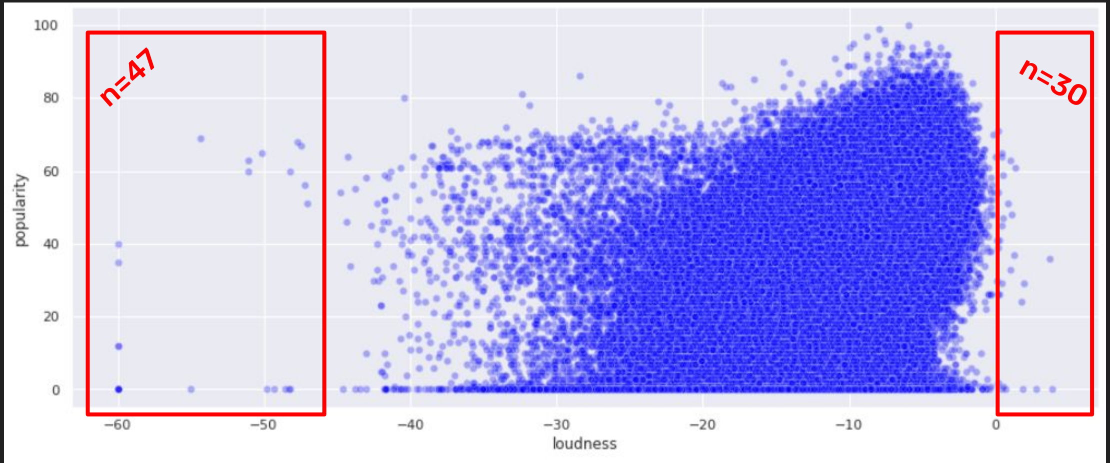




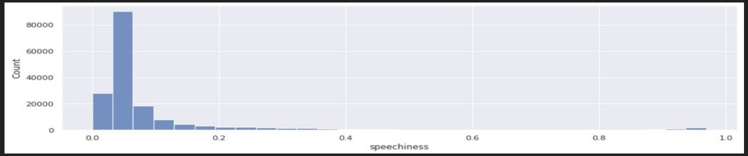
Loudness

- Loudness is the quality of a sound that is the primary psychological correlate of amplitude.
- Linear corr. to popularity = 0.463

count	169240.000000
mean	-11.342179
std	5.642759
min	-60.000000
25%	-14.430000
50%	-10.453000
75%	-7.109000
max	3.855000



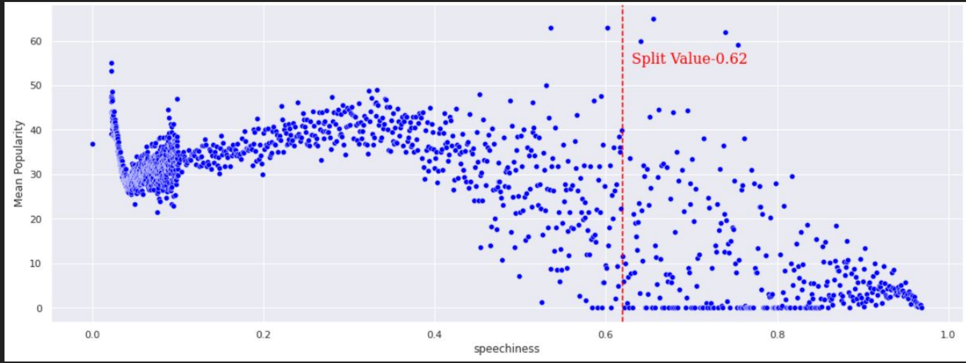
Speechiness

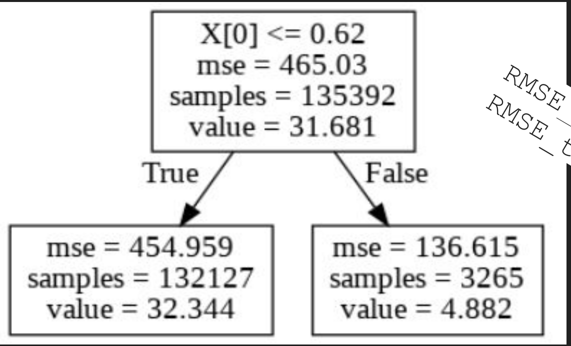


- Due to an obvious change in in speechiness trend vs. popularity, we decided to divide the records into two sub-groups.

- The cut-off point is based on a decision tree model.

- Linear corr. to popularity = -0.14



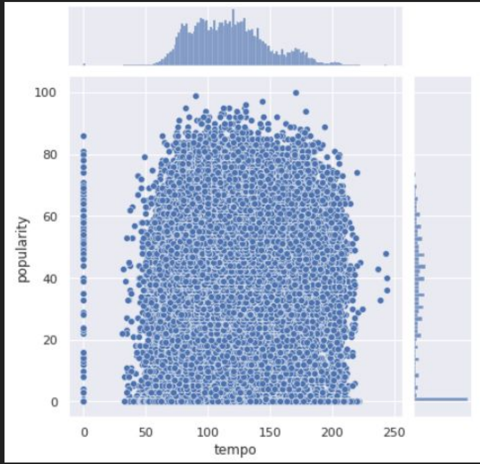


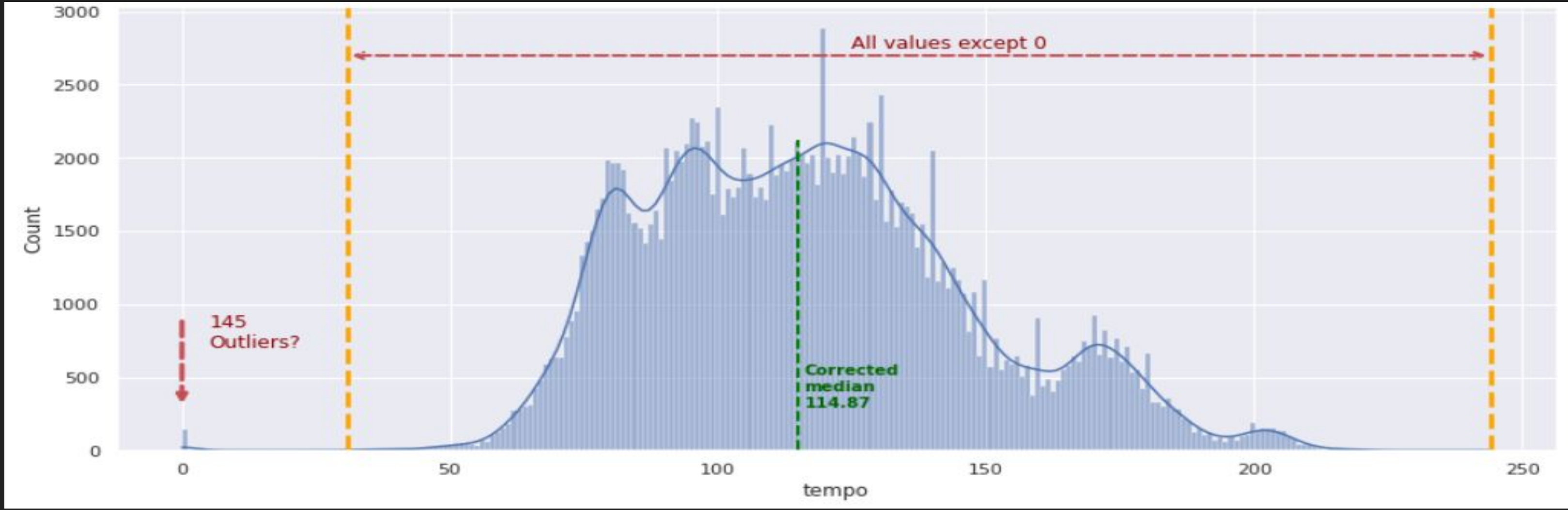
RMSE Train = 21.15
RMSE_test = 21.00

Tempo

- The speed or pace of a given piece. Derives directly from the average beat duration.

- Linear corr. to popularity = 0.13





Tempo

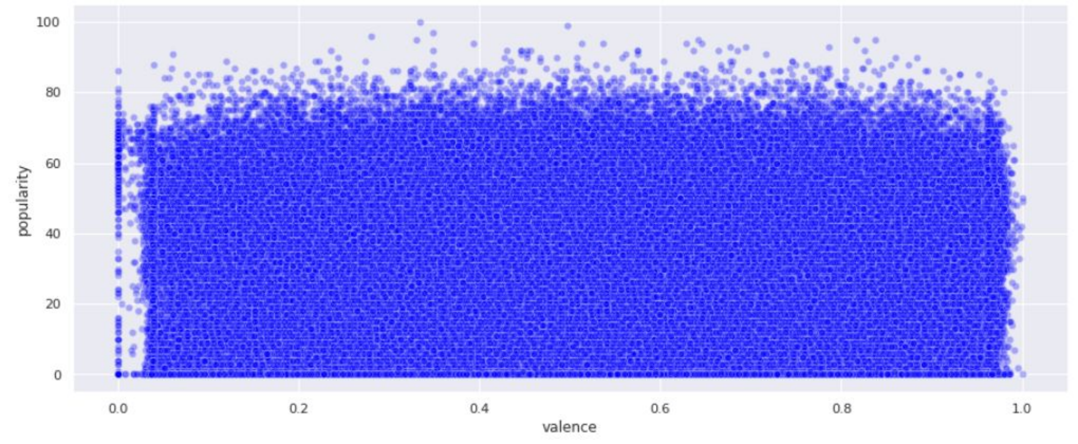
- Self made imputer with median or mean

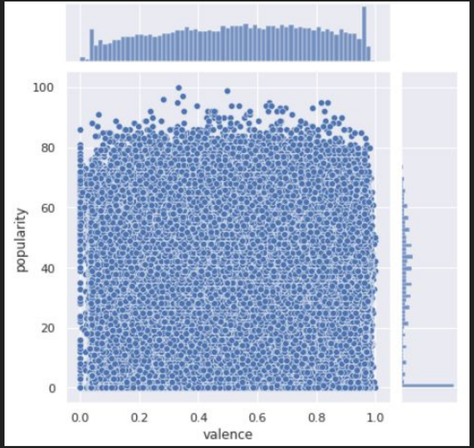
```python
class ReplaceZeroTransformer():
    """Eliminates Zero values from tempo columns and replace it
       with the median or mean of non-zero values as specified.
       defaut is set to 'median'.
    """

    def __init__(self, method='median'):
        self.method = method

    def transform(self, X):
        if self.method == 'median':
            X.loc[X['tempo']==0, 'tempo'] = X.loc[X['tempo']>0, 'tempo'].median()
        elif self.method == 'mean':
            X.loc[X['tempo']==0, 'tempo'] = X.loc[X['tempo']>0, 'tempo'].mean()
        else:
            raise Exception("Method can be 'median' or 'mean' only!")
        return X
```

Valence

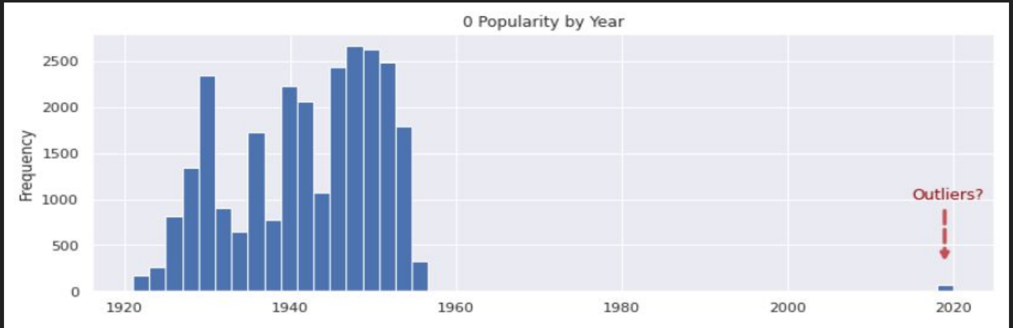
- A measure describing the musical positiveness conveyed by a track
- Linear corr. = 0.01

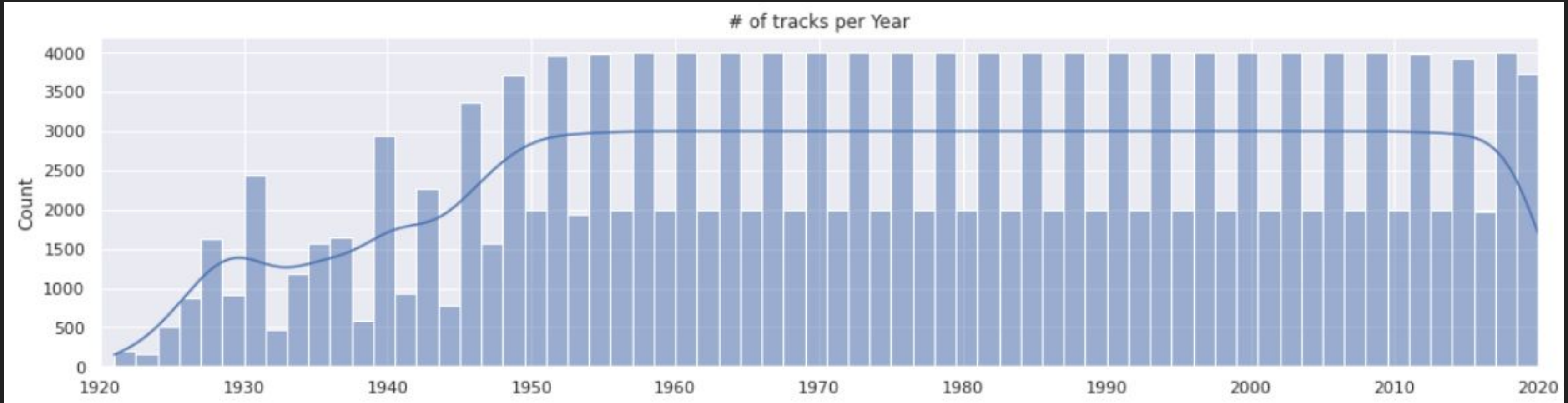
No Action Required





Year





- Linear corr. to popularity = 0.88
- Most of the years are 2000 or 4000, due to the 2000 maximal batch limit in the Spotify's API.

Pre-Processing

Pre-Processing

- Import data with relevant columns only
- Remove duplicates
- Split data to train & test
- For KNN only: sample data (frac = 0.3)

```python
# Read column names from file
cols = list(pd.read_csv('data.csv', nrows =1))
df = pd.read_csv('data.csv', usecols=[i for i in cols if i not in ['id','name','release_date','year']])

# Remove duplicated
df = df[~df.duplicated()==1]
df = df.sample(frac=0.3)
#Split the data to train and test
X_train, X_test, y_train, y_test = split(df.drop('popularity', axis=1), df['popularity'], test_size = 0.33, random_state = 12345)
```

Transformers

- Unique feature transformers
- OneHotEncoder
- MinMaxScaling
- Target Scaling

Unique Feature Transformers

```
# Apply AritistsTransformer on train and test seperatly
artists_transformer = ArtistsTransformer(MinCnt=2)
X_train = artists_transformer.fit(X_train, y_train).transform(X_train, y_train)
X_test = artists_transformer.transform(X_test, y_test)
```

```
def instrumentalness_criteria(X):
    X['instrumentalness'] = list(map((lambda x: 1 if x < 0.1 else (3 if x > 0.95 else 2)), X.instrumentalness))

instrumentalness_tranformer = FunctionTransformer(instrumentalness_criteria)
instrumentalness_tranformer.transform(X_train)
instrumentalness_tranformer.transform(X_test)
```

Unique Feature Transformers

```
def speech_criteria(X):
    X['speechiness'] = list(map((lambda x: 1 if x > 0.62 else 0), X.speechiness))

speech_tranformer = FunctionTransformer(speech_criteria)
speech_tranformer.transform(X_train)
speech_tranformer.transform(X_test)
```

```
tempo_transformer = ReplaceZeroTransformer()
X_train = tempo_transformer.transform(X_train, 'median')
X_test = tempo_transformer.transform(X_test, 'median')
```

OneHotEncoder

```
ohe = OneHotEncoder(categories='auto', drop='first')

# Train
feature_arr = ohe.fit_transform(X_train[['instrumentalness','key']]).toarray()
columns_key = ['key_'+str(i) for i in list(set(X_train['key'].values))[1:]]
instrumentalness_key = ['ins_'+str(i) for i in list(set(X_train['instrumentalness'].values))[1:]]
feature_labels = columns_key + instrumentalness_key
feature_labels = np.concatenate((feature_labels), axis=None)
features = pd.DataFrame(feature_arr, columns = feature_labels, index = X_train.index)
X_train = pd.concat([X_train, features], axis=1).drop(['key','instrumentalness'], axis=1)

# Test
feature_arr = ohe.fit_transform(X_test[['instrumentalness','key']]).toarray()
columns_key = ['key_'+str(i) for i in list(set(X_test['key'].values))[1:]]
instrumentalness_key = ['ins_'+str(i) for i in list(set(X_test['instrumentalness'].values))[1:]]
feature_labels = columns_key + instrumentalness_key
feature_labels = np.concatenate((feature_labels), axis=None)
features = pd.DataFrame(feature_arr, columns = feature_labels, index = X_test.index)
X_test = pd.concat([X_test, features], axis=1).drop(['key','instrumentalness'], axis=1)
```

MinMaxScaler

```python
scaler = MinMaxScaler()
cols = ['artists','duration_ms','loudness','tempo']
X_train[cols] = scaler.fit_transform(X_train[cols])
X_test[cols] = scaler.fit_transform(X_test[cols])
```

Target Scaling

```python
# Divide the popularity by 100
y_train = y_train / 100
y_test = y_test / 100
```

Prepared Data

```
X_train shape is: (113216, 25)
y_train shape is: (113216,)
X_test shape is: (55764, 25)
y_test shape is: (55764,)
```

```
1  y_train.describe().drop(['count','25%', '50%', '75%'])
```

```
mean    0.317545
std     0.215262
min     0.000000
max     0.990000
Name: popularity, dtype: float64
```

	acousticness	artists	danceability	duration_ms	energy	explicit	liveness	loudness	mode	speechiness	tempo	valence
mean	0.490510	0.376646	0.538910	0.041969	0.488863	0.086066	0.206873	0.774618	0.708813	0.024361	0.404491	0.532166
std	0.375787	0.202948	0.175158	0.022925	0.267015	0.280462	0.177242	0.089876	0.454311	0.154166	0.143558	0.261987
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	0.996000	1.000000	0.986000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

	key_1	key_2	key_3	key_4	key_5	key_6	key_7	key_8	key_9	key_10	key_11	ins_2	ins_3
mean	0.232591	0.009318	0.075908	0.111398	0.042167	0.075537	0.095066	0.050187	0.123233	0.063286	0.103934	0.070600	0.062509
std	0.422485	0.096082	0.264852	0.314625	0.200971	0.264257	0.293307	0.218332	0.328706	0.243478	0.305176	0.256156	0.242078
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

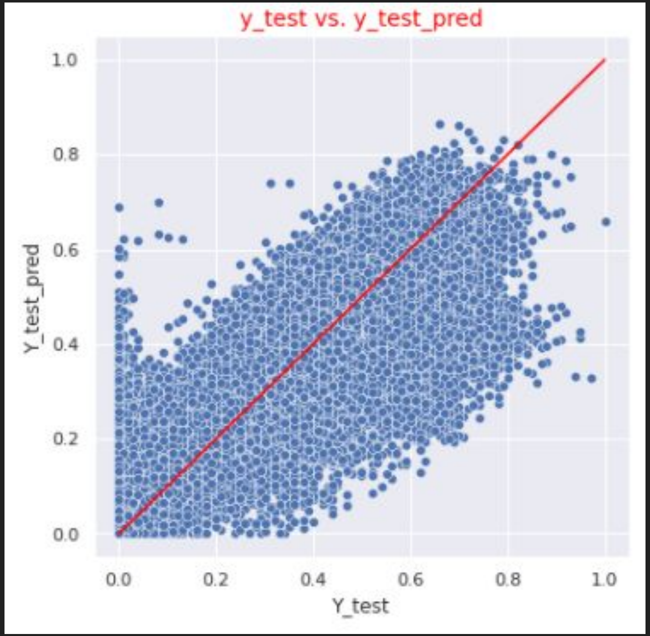
Regression Models

Linear Regression

Score Train	Score Test	Validation	Parameters
0.129	0.133	96.99%	All data mincnt = 3, test size = 0.33
0.127	0.132	96.21%	All data mincnt = 3, test size = 0.2
0.206	0.206	100%	corr > 0.2, mincnt = 3
0.124	0.131	94.65%	All data, mincnt = 2, test size = 0.33
0.123	0.129	95.34%	All data, mincnt = 2, test size = 0.2

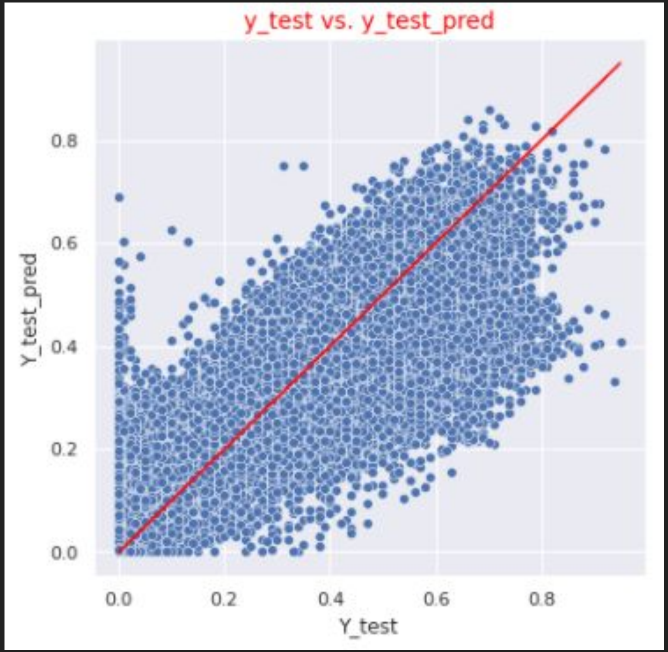
Linear Regression (test size = 0.33, MinCnt = 2)

$$y = 0.04 + -0.10acousticness + \boxed{0.62artists} + 0.14danceability + 0.04duration_m s + 0.03energy + 0.05explicit + -0.05liveness + 0.08loudness + -0.00mode + -0.11speechiness + 0.03tempo + -0.10valence + -0.02key_1 + 0.04key_2 + 0.01key_3 + -0.00key_4 + -0.01key_5 + 0.00key_6 + 0.00key_7 + 0.01key_8 + -0.00key_9 + 0.00key_10 + -0.00key_11 + -0.00ins_2 + -0.00ins_3$$



Linear Regression (test size = 0.2, MinCnt = 2)

$$y = 0.03 + -0.10 acousticness + \boxed{0.63 artists + 0.14 danceability} + 0.05 duration_m s + 0.03 energy + 0.05 explicit + -0.05 liveness + 0.08 loudness + -0.00 mode + -0.10 speechiness + 0.03 tempo + -0.10 valence + -0.02 key_1 + 0.03 key_2 + 0.01 key_3 + -0.00 key_4 + -0.01 key_5 + 0.00 key_6 + 0.00 key_7 + 0.01 key_8 + -0.00 key_9 + 0.00 key_1 0 + -0.00 key_1 1 + -0.00 ins_2 + -0.00 ins_3$$



Linear Regression

```
import statsmodels.api as sm
model = sm.OLS(y_train, X_train).fit()
model.summary()
```

OLS Regression Results

Dep. Variable:	popularity	**R-squared (uncentered):**	0.895
Model:	OLS	**Adj. R-squared (uncentered):**	0.895
Method:	Least Squares	**F-statistic:**	3.847e+04
Date:	Thu, 15 Oct 2020	**Prob (F-statistic):**	0.00
Time:	07:58:03	**Log-Likelihood:**	75239.
No. Observations:	113216	**AIC:**	-1.504e+05
Df Residuals:	113191	**BIC:**	-1.502e+05
Df Model:	25		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
acousticness	-0.0996	0.002	-60.610	0.000	-0.103	-0.096
artists	0.6236	0.002	263.162	0.000	0.619	0.628
danceability	0.1417	0.003	49.197	0.000	0.136	0.147
duration_ms	0.0633	0.016	3.839	0.000	0.031	0.096
energy	0.0210	0.003	7.311	0.000	0.015	0.027
explicit	0.0479	0.001	32.408	0.000	0.045	0.051
liveness	-0.0513	0.002	-23.676	0.000	-0.056	-0.047
loudness	0.1246	0.004	29.688	0.000	0.116	0.133
mode	-0.0032	0.001	-3.816	0.000	-0.005	-0.002
speechiness	-0.1058	0.003	-40.035	0.000	-0.111	-0.101
tempo	0.0317	0.003	11.897	0.000	0.027	0.037
valence	-0.0999	0.002	-50.679	0.000	-0.104	-0.096
key_1	-0.0198	0.001	-20.962	0.000	-0.022	-0.018
key_2	0.0423	0.004	10.869	0.000	0.035	0.050
key_3	0.0076	0.002	4.487	0.000	0.004	0.011
key_4	-0.0011	0.002	-0.746	0.456	-0.004	0.002
key_5	-0.0062	0.002	-2.954	0.003	-0.010	-0.002
key_6	0.0036	0.002	2.106	0.035	0.000	0.007
key_7	0.0019	0.002	1.216	0.224	-0.001	0.005
key_8	0.0077	0.002	3.932	0.000	0.004	0.012
key_9	0.0014	0.001	0.984	0.325	-0.001	0.004
key_10	0.0042	0.002	2.358	0.018	0.001	0.008
key_11	-0.0007	0.002	-0.444	0.657	-0.004	0.002
ins_2	-0.0004	0.002	-0.216	0.829	-0.004	0.003
ins_3	0.0003	0.002	0.165	0.869	-0.003	0.004

Omnibus:	4560.460	**Durbin-Watson:**	1.993
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6698.732
Skew:	0.390	**Prob(JB):**	0.00
Kurtosis:	3.901	**Cond. No.**	73.6

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

KNN

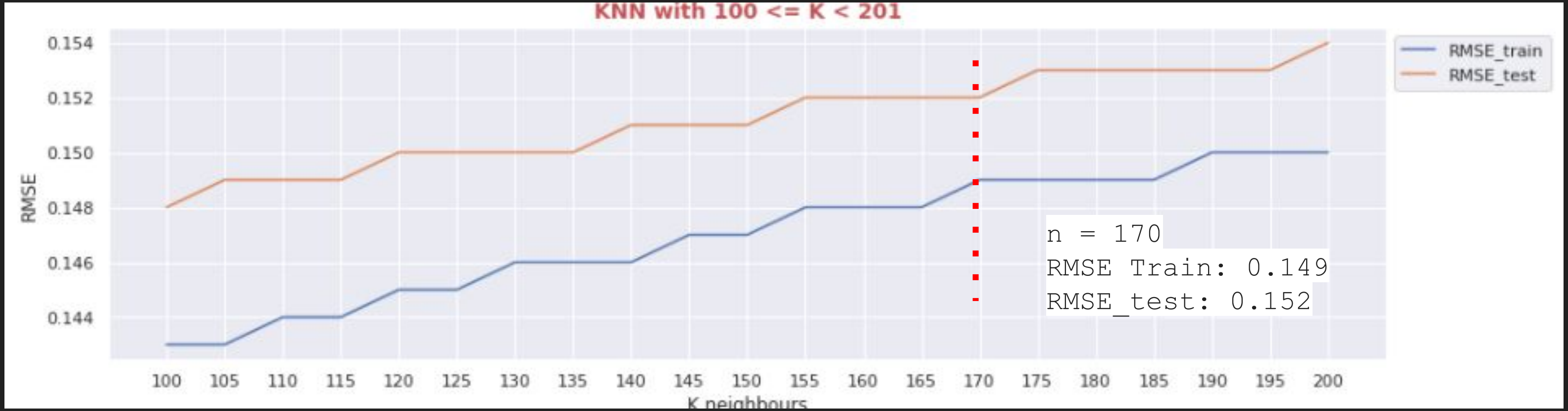
Due to long running time, this model used a sampled data (frac = 0.3)

- MinCnt = 2
- 2 loops with different n_neighbors
  - 0-101, 100-201 with 5's step
- Results stored in lists
- The smallest train-test gap is n = 170

```python
RMSE_train, RMSE_test = [], []

for i in range(100,201,5):
  knn = KNeighborsRegressor(n_neighbors=i)
  knn.fit(X_train,y_train)
  y_train_pred = knn.predict(X_train)
  knn_train_rmse = np.sqrt(mse(y_train, y_train_pred))
  RMSE_train.append(knn_train_rmse.round(3))
  y_test_pred = knn.predict(X_test)
  knn_test_rmse = np.sqrt(mse(y_test, y_test_pred))
  RMSE_test.append(knn_test_rmse.round(3))
```

KNN


KNN with 10 <= n-neighbors < 100
RMSE
0.145
0.140
0.135
0.130
0.125
0.120
0.115
5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
Neighbors
RMSE_train
RMSE_test
n=100
RMSE Train: 0.143
RMSE_test: 0.148


KNN with 100 <= K < 201
RMSE
0.154
0.152
0.150
0.148
0.146
0.144
100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200
K neighbours
RMSE_train
RMSE_test
n = 170
RMSE Train: 0.149
RMSE_test: 0.152

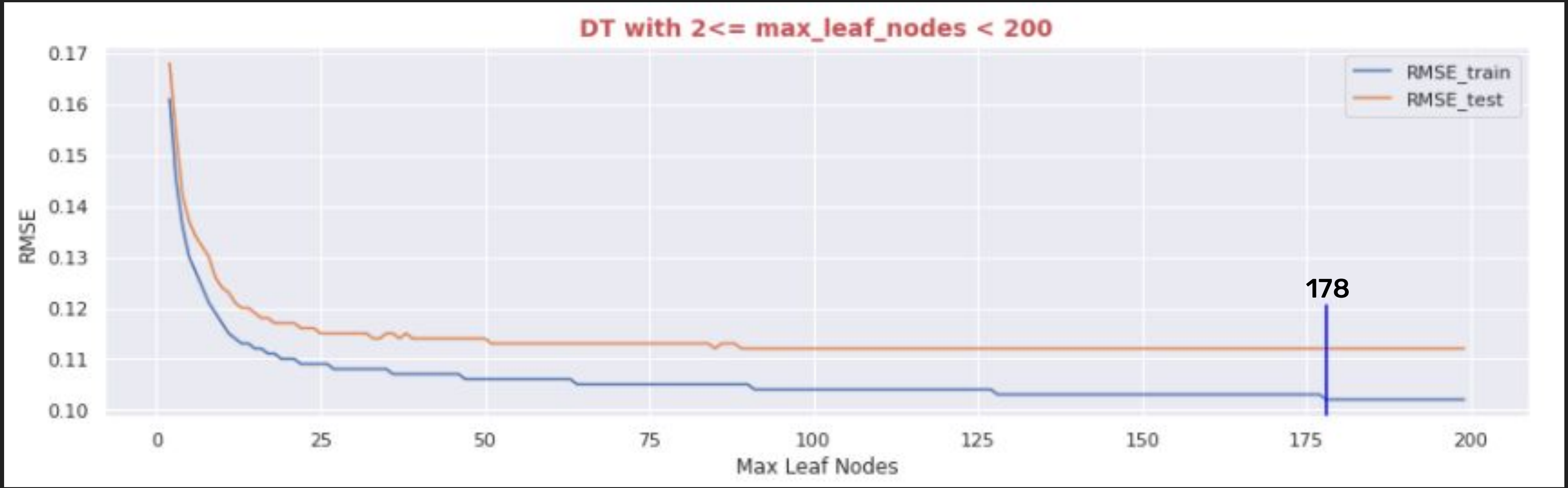
Decision Tree (test size = 0.2)

Score Train	Score Test	Validation	Parameters
0.121	0.127	95.27%	max_leaf_nodes=10, mincnt = 3, test_size = 0.2
0.116	0.121	95.86%	max_leaf_nodes=15, mincnt = 3
0.114	0.119	95.80%	max_leaf_nodes=20, mincnt = 3
0.106	0.115	92.17%	max_leaf_nodes=139, mincnt = 3
0.106	0.114	92.98%	max_leaf_nodes=183, mincnt = 3
0.107	0.114	93.86%	mincnt = 2, max_leaf_nodes = 41, min_samples_split=2000
0.102	0.112	91.07%	max_leaf_nodes = 178, mincnt = 2
0.136	0.142	95.77%	max_leaf_nodes = 4, mincnt = 2

Decision Tree

(test size = 0.2)

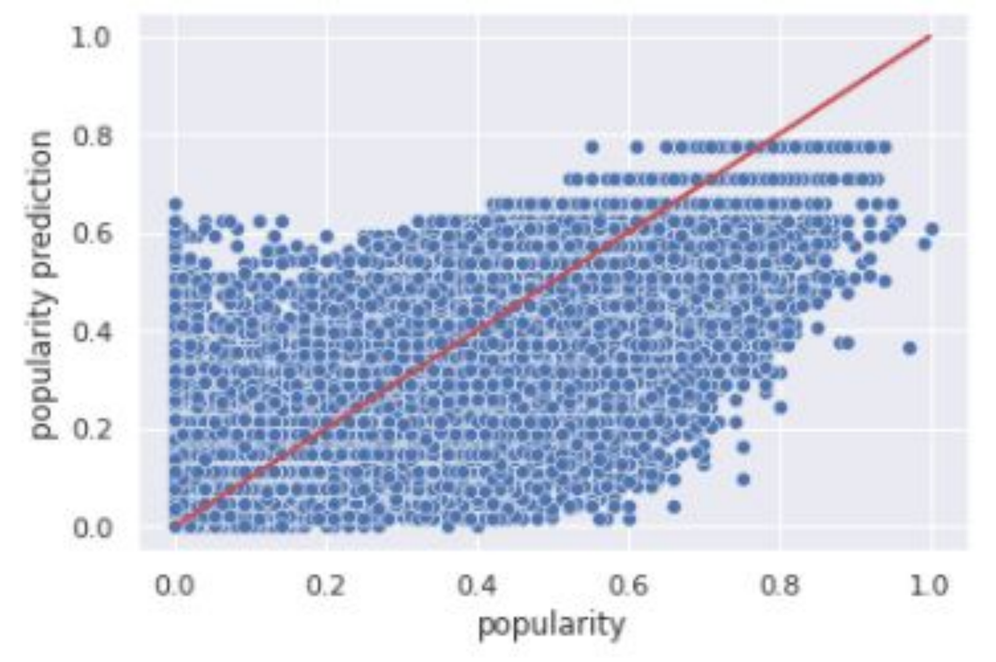
```python
RMSE3_train, RMSE3_test = [], []

for i in range(2,200):
  tree = DecisionTreeRegressor(random_state = 15, max_leaf_nodes=i)
  tree.fit(X_train, y_train)
  y_train_pred = tree.predict(X_train).clip(0, 1)
  train_rmse = np.sqrt(mse(y_train, y_train_pred))
  RMSE3_train.append(train_rmse.round(3))
  y_test_pred = tree.predict(X_test).clip(0, 1)
  test_rmse = np.sqrt(mse(y_test, y_test_pred))
  RMSE3_test.append(test_rmse.round(3))
```



Decision Tree

(test size = 0.2)

artists <= 0.288
mse = 0.046
samples = 135184
value = 0.318




```
Feature importances:
acousticness: 0.087
artists     : 0.856
danceability: 0.003
duration_ms : 0.006
energy      : 0.003
explicit    : 0.006
liveness    : 0.004
loudness    : 0.025
mode        : 0.000
speechiness : 0.000
tempo       : 0.001
valence     : 0.007
key_1       : 0.001
key_2       : 0.000
key_3       : 0.000
key_4       : 0.000
key_5       : 0.000
key_6       : 0.000
key_7       : 0.000
key_8       : 0.000
key_9       : 0.000
key_10      : 0.000
key_11      : 0.000
ins_2       : 0.000
ins_3       : 0.000
```

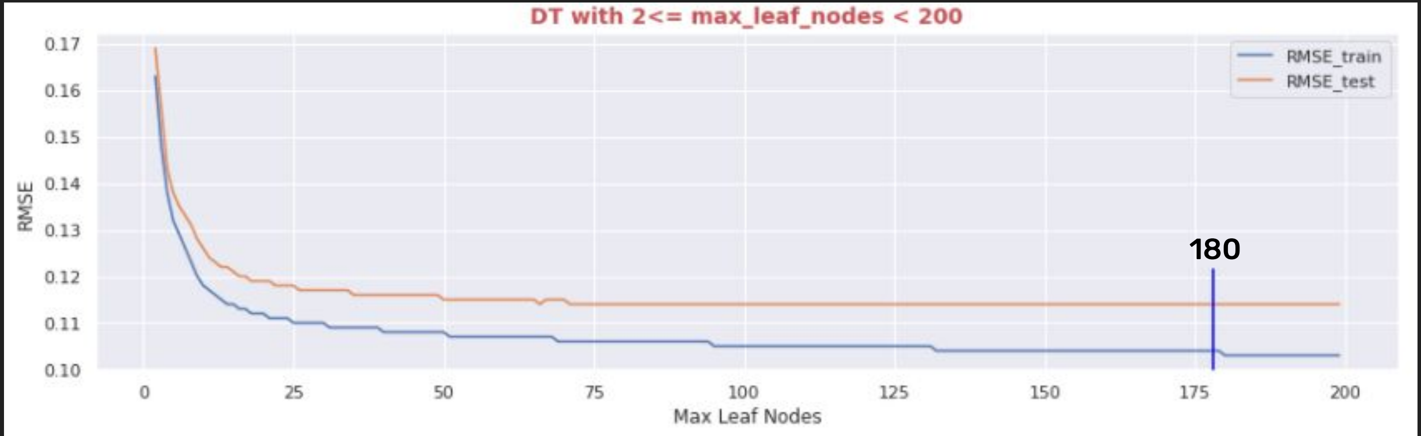
Decision Tree (test size = 0.33)

Score Train	Score Test	Validation	Parameters
0.124	0.129	96.12%	max_leaf_nodes=10, mincnt = 3, test_size = 0.33
0.119	0.125	95.20%	max_leaf_nodes=15, mincnt = 3
0.117	0.122	95.90%	max_leaf_nodes=20, mincnt = 3
0.108	0.116	93.10%	max_leaf_nodes=139, mincnt = 3
0.107	0.117	91.45%	max_leaf_nodes=183, mincnt = 3
0.108	0.116	93.10%	mincnt = 2, max_leaf_nodes = 41, min_samples_split=2000
0.103	0.114	90.35%	max_leaf_nodes = 180, mincnt = 2
0.138	0.143	96.50%	max_leaf_nodes = 4, mincnt = 2

Decision Tree

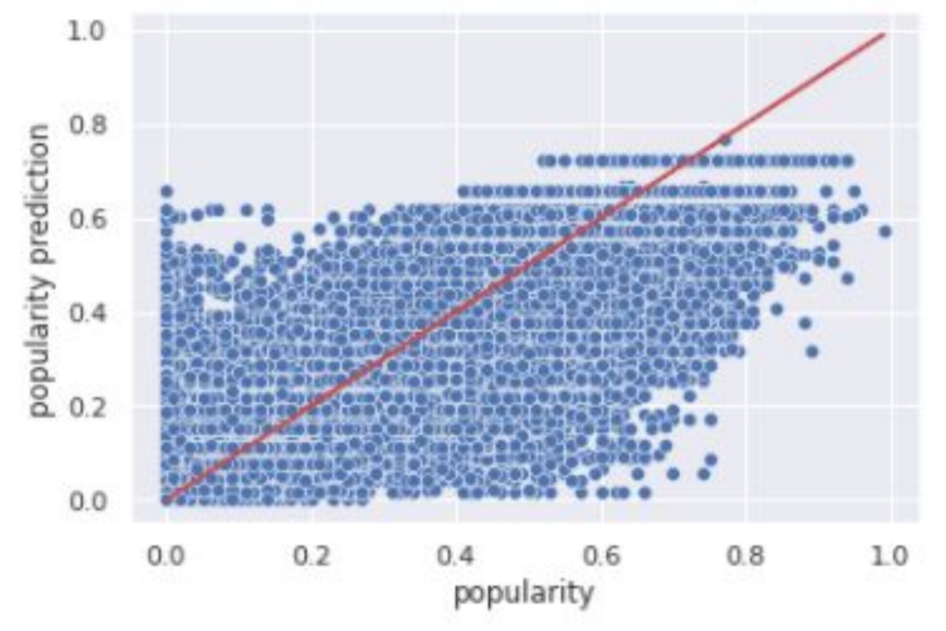
(test size = 0.33)

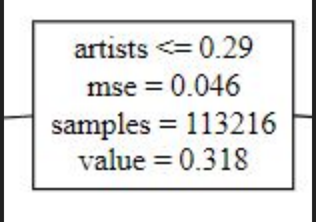
```python
RMSE3_train, RMSE3_test = [], []

for i in range(2,200):
  tree = DecisionTreeRegressor(random_state = 15, max_leaf_nodes=i)
  tree.fit(X_train, y_train)
  y_train_pred = tree.predict(X_train).clip(0, 1)
  train_rmse = np.sqrt(mse(y_train, y_train_pred))
  RMSE3_train.append(train_rmse.round(3))
  y_test_pred = tree.predict(X_test).clip(0, 1)
  test_rmse = np.sqrt(mse(y_test, y_test_pred))
  RMSE3_test.append(test_rmse.round(3))
```



Decision Tree

(test size = 0.33)






```
Feature importances:
acousticness: 0.096
artists     : 0.841
danceability: 0.004
duration_ms : 0.007
energy      : 0.004
explicit    : 0.007
liveness    : 0.004
loudness    : 0.026
mode        : 0.000
speechiness : 0.000
tempo       : 0.001
valence     : 0.008
key_1       : 0.001
key_2       : 0.001
key_3       : 0.000
key_4       : 0.000
key_5       : 0.000
key_6       : 0.000
key_7       : 0.000
key_8       : 0.000
key_9       : 0.000
key_10      : 0.000
key_11      : 0.000
ins_2       : 0.000
ins_3       : 0.000
```

Thanks