MUSIC GENRE CLASSIFICATION

By: Ashraf Abdulkhlaiq | Hamza Al-Habash | Ammar Abushukur

DATA DESCRIPTION

artist: Name of the Artist.

song: Name of the Track.

popularity: The **higher** the value the **more popular** the song is.

danceability: Describes how **suitable a track** is for **dancing** based on a combination of musical elements **including tempo, rhythm**.

energy: A measure from **0.0 to 1.0** and represents a perceptual measure of **intensity and activity**.

key: The key the track is in. Integers map to **pitches** using standard **Pitch Class notation**.

loudness: The **overall loudness** of a track in **decibels (dB).** Loudness values are averaged across the entire track and are useful for comparing relative.

mode: Mode indicates the **modality (major or minor)** of a track, the type of scale from which its melodic content is derived. **Major is represented by 1 and minor is 0.**

DATA DESCRIPTION

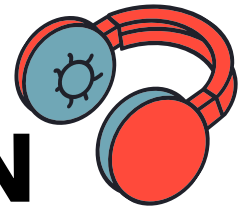
speechiness: Speechiness detects the presence of **spoken words in a track**. The more exclusively speech-like the recording (e.g. talk show, audio book, poetry), **the closer to 1.0 the attribute value.**

acousticness: A confidence measure from **0.0 to 1.0** of whether the track is acoustic. **1.0 represents high confidence the track is acoustic**.

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.

liveness: Detects the presence of an audience in the recording. **Higher liveness values represent an increased probability** that the track was performed live. **A value above 0.8 provides strong likelihood that the track is live.**

DATA DESCRIPTION



valence: A measure from **0.0 to 1.0** describing the musical positiveness conveyed by a track. **Tracks with high valence sound more positive** (e.g. **happy, cheerful, euphoric**), while **tracks with low valence sound more negative** (e.g. **sad, depressed, angry**).

tempo: The overall estimated **tempo of a track in beats per minute** (BPM). In musical terminology, tempo is the **speed or pace of a given piece** and derives directly from the average beat duration.

duration in milliseconds: Time of the song.

time_signature: a **notational convention** used in **Western musical notation** to specify how **many beats (pulses)** are contained in each measure (bar), and which note value is equivalent to a beat.

Class: Genre of the track.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC

import xgboost as xgb
from lazypredict.Supervised import LazyClassifier
```

IMPORT LIBRARIES

```
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")
```

```
train.head()
```

}]:

	Id	Artist Name	Track Name	Popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instru
0	1	Marina Maximilian	Not Afraid	37.00	0.33	0.54	9.00	-6.65	0	0.04	0.38	
1	2	The Black Keys	Howlin' for You	67.00	0.72	0.75	11.00	-5.54	1	0.09	0.03	
2	3	Royal & the Serpent	phuck u	NaN	0.58	0.80	7.00	-6.09	1	0.06	0.00	
3	4	Detroit Blues Band	Missing You	12.00	0.52	0.31	NaN	-14.71	1	0.03	0.91	
4	5	Coast Contra	My Lady	48.00	0.56	0.78	6.00	-5.10	0	0.25	0.18	

LOADING DATA

EXPLORATORY DATA ANALYSIS (EDA)

First: Data Cleaning

Sorting out the unit

The duration in ms/min column had values in either ms or min units, we changed all the values to be in min using 5000 as threshold

duration_in min/ms
204947.00
191956.00
161037.00
298093.00
254145.00

duration_in min
3.42
3.20
2.68
4.97
4.24

EXPLORATORY DATA ANALYSIS (EDA)

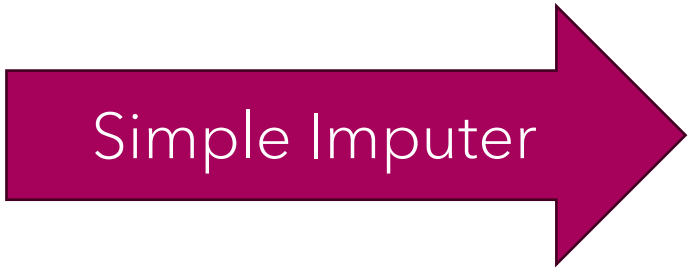
First: Data Cleaning

Null Values

Imputing null values using simple imputer

```
train.isnull().sum()
```

```
Id                      0
Artist Name             0
Track Name              0
Popularity            333
danceability            0
energy                  0
key                  1609
loudness                0
mode                    0
speechiness             0
acousticness            0
instrumentalness     3541
liveness                0
valence                 0
tempo                   0
duration_in min/ms      0
time_signature          0
Class                   0
dtype: int64
```




```
df.isnull().sum()
```

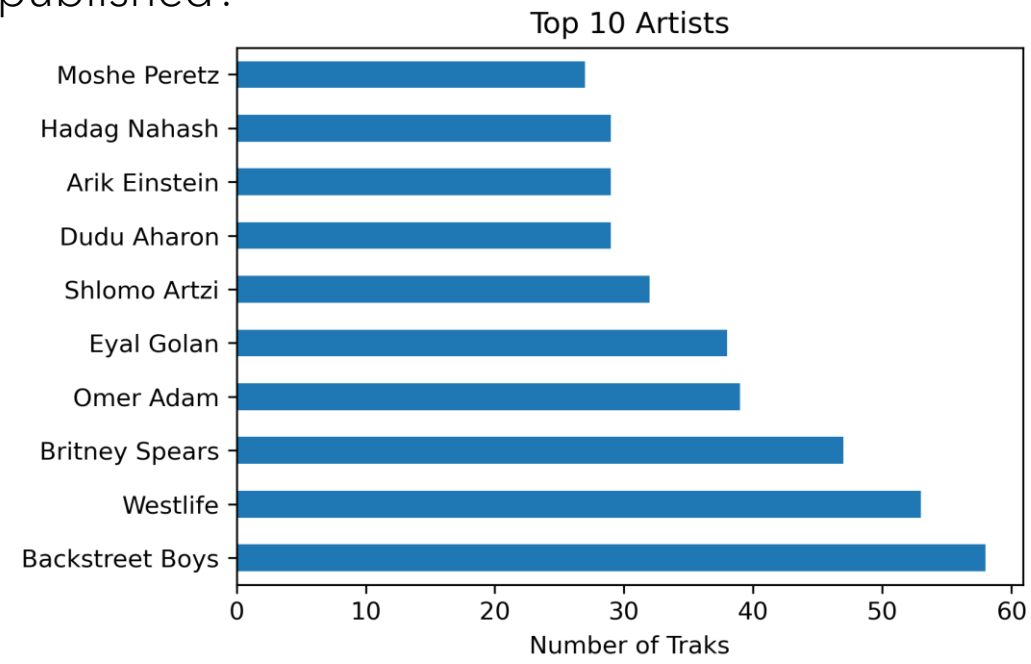
```
id                  0
artist_name         0
track_name          0
popularity          0
danceability        0
energy              0
key                 0
loudness            0
mode                0
speechiness         0
acousticness        0
instrumentalness    0
liveness            0
valence             0
tempo               0
duration_in_ms      0
time_signature      0
class               0
dtype: int64
```

EXPLORATORY DATA ANALYSIS (EDA)

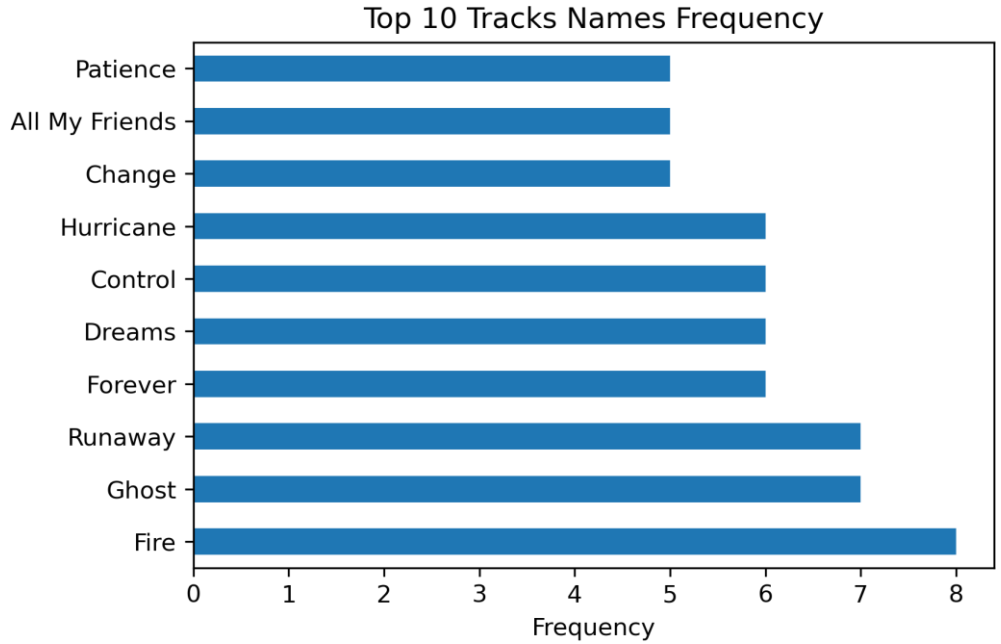
Second: Data Exploring & Visualization

Bar Charts

Which artist has the highest number of tracks published?



What are the top 10 most frequent track names in the dataset?

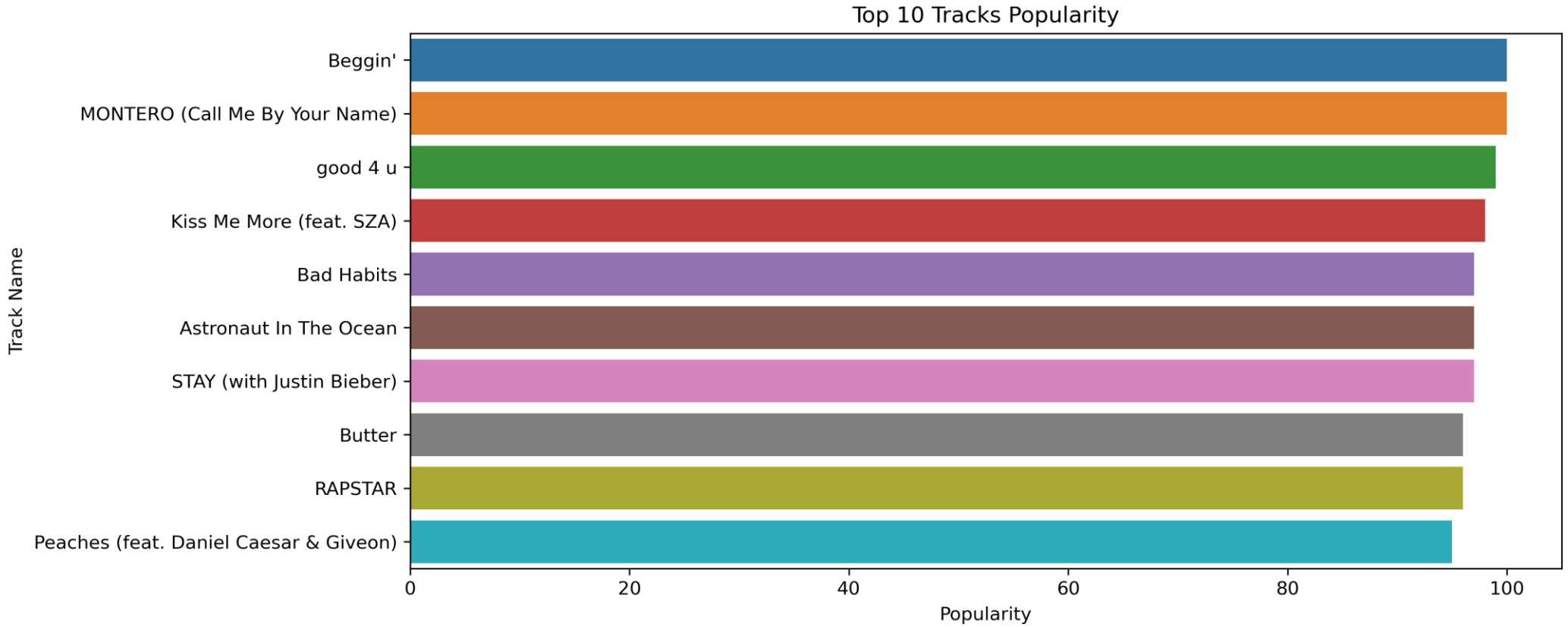


EXPLORATORY DATA ANALYSIS (EDA)

Second: Data Exploring & Visualization

Bar Charts

Which track holds the highest position as the most popular among the top 10?

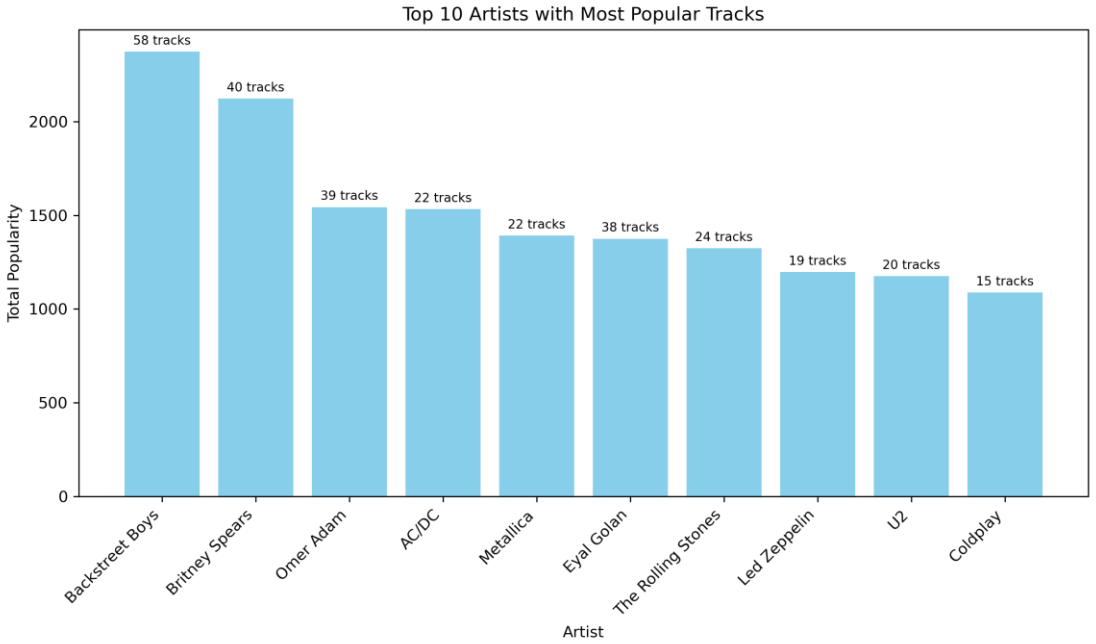


EXPLORATORY DATA ANALYSIS (EDA)

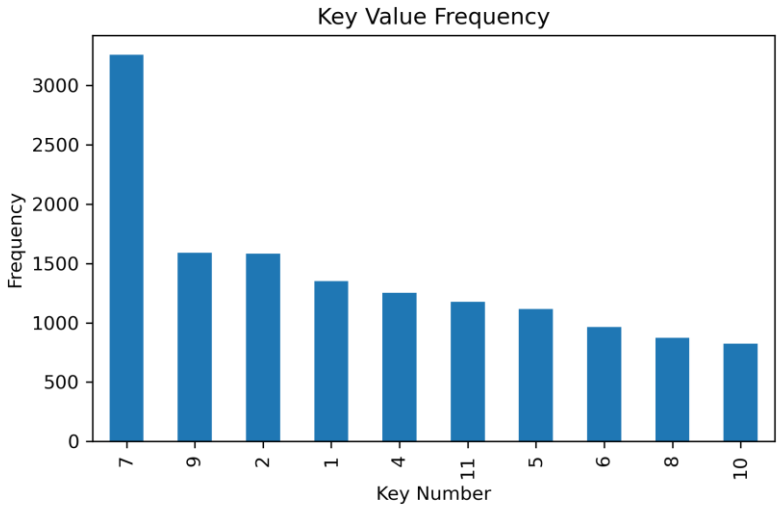
Second: Data Exploring & Visualization

Bar Charts

How is the popularity of tracks determined for each artist in the dataset?



Key frequency



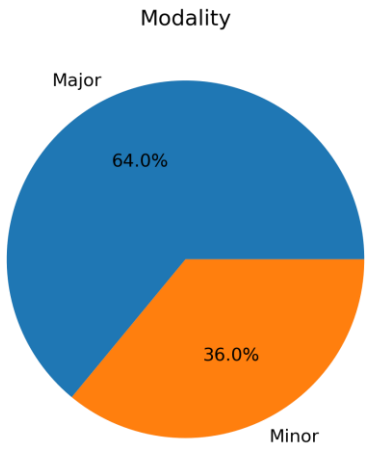
EXPLORATORY DATA ANALYSIS (EDA)

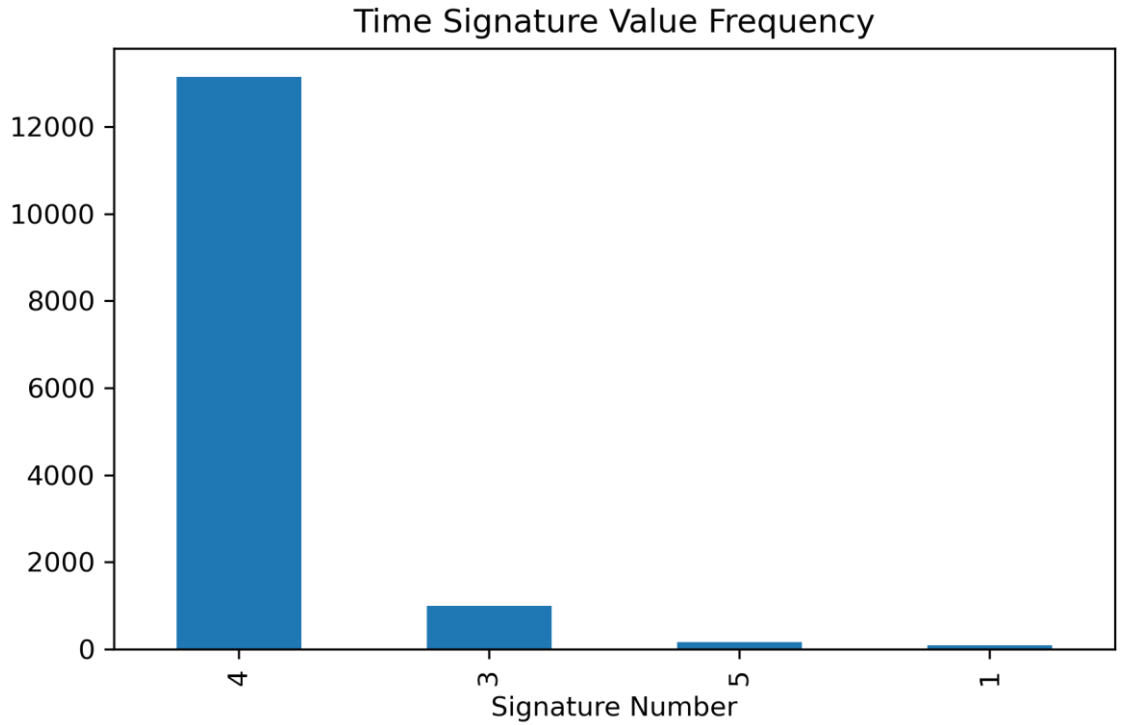
Second: Data Exploring & Visualization

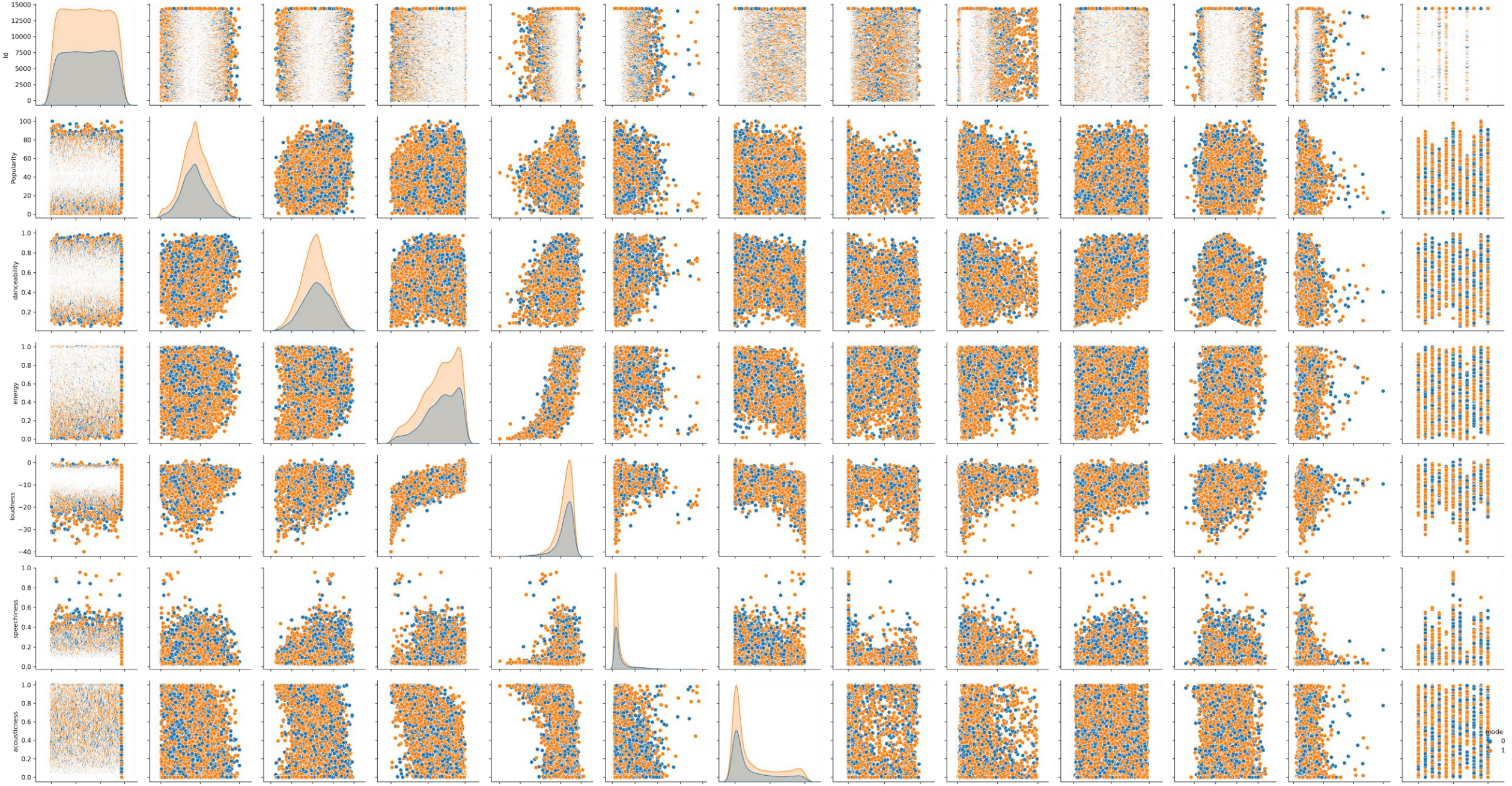
Bar Charts & Pie

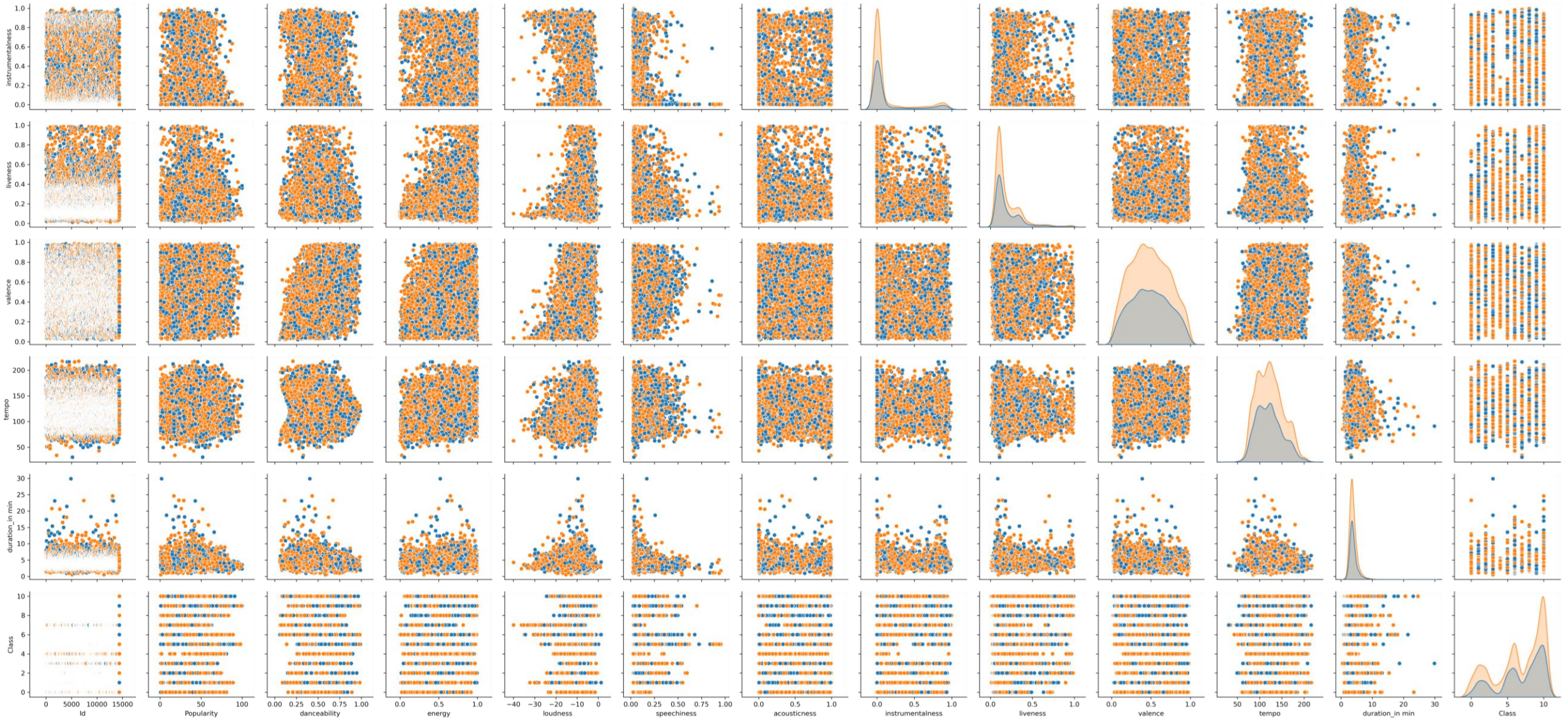
Major represented by 1

Minor represented by 0






Id
Popularity
danceability
energy
loudness
speechiness
acousticness
mode
0
1



- Major represented by 1 [orange]

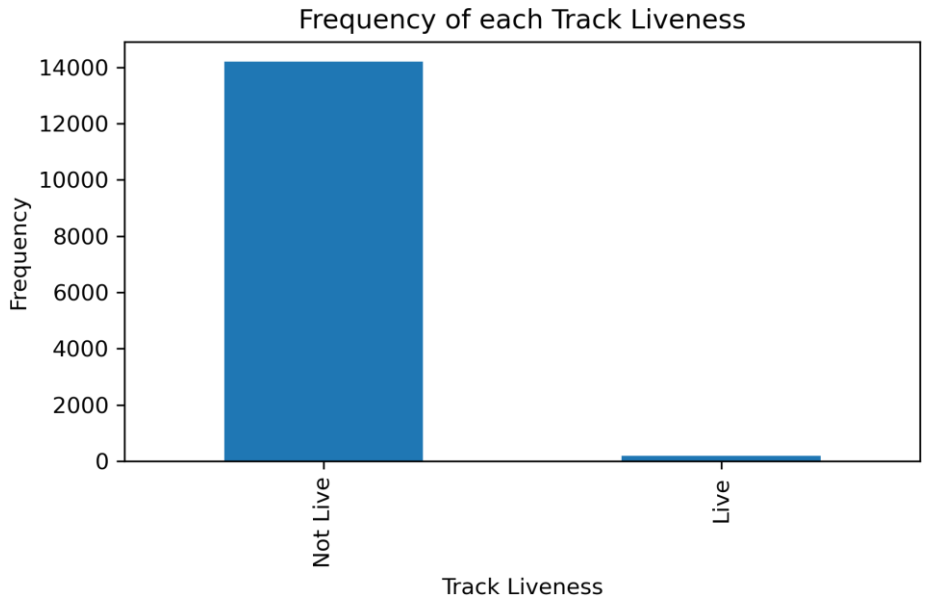
- Minor represented by 0 [blue]

EXPLORATORY DATA ANALYSIS (EDA)

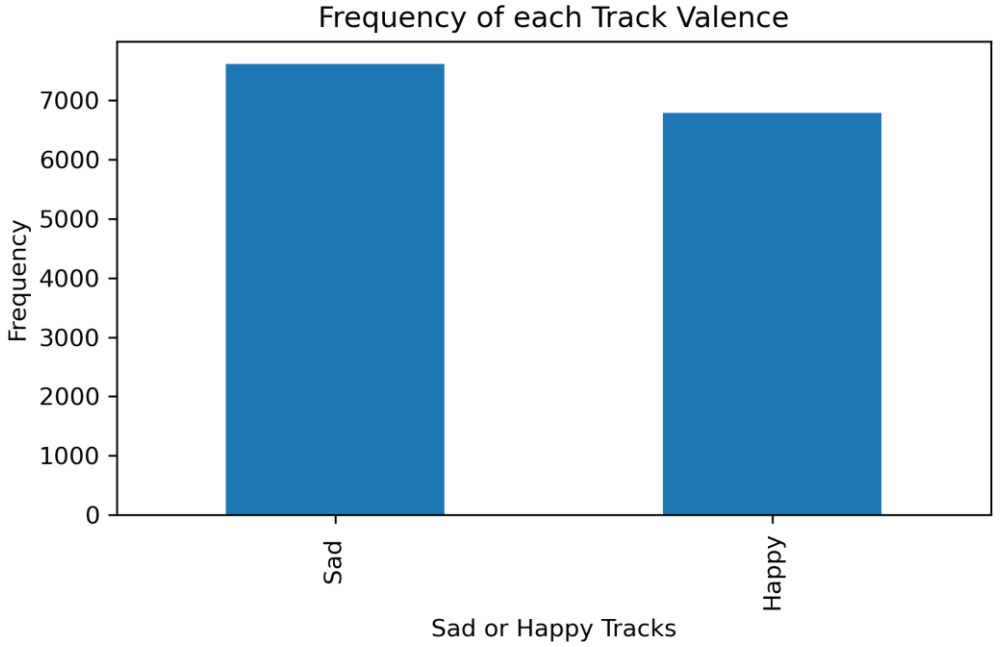
Second: Data Exploring & Visualization

Bar Charts

Our data have only 195 tracks was live from total 14396 tracks.



Our data have 7614 tracks that have sad/depression tone, but in the other hand we have 6782 tracks that have happy/cheerful tone.

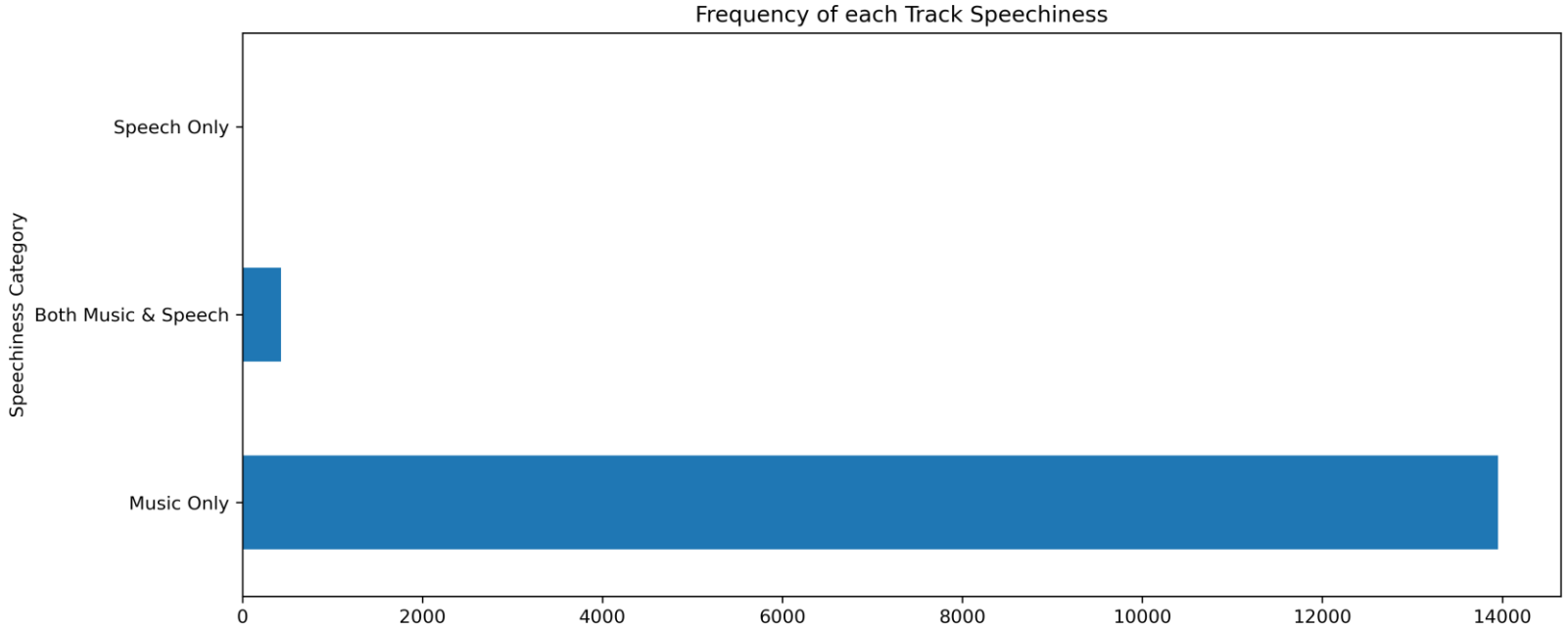


EXPLORATORY DATA ANALYSIS (EDA)

Second: Data Exploring & Visualization

Bar Charts

Our data have 13953 tracks that only music, 430 tracks that music & speech and only 13 track speech.

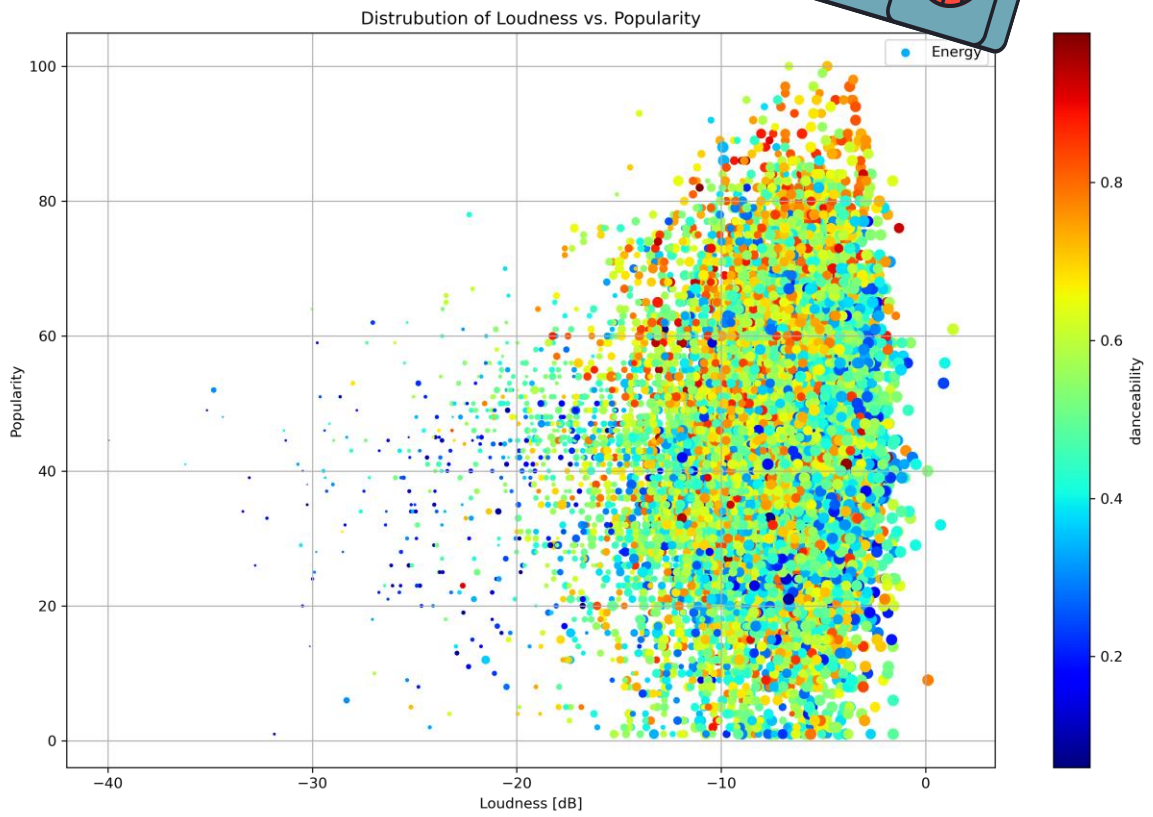


EXPLORATORY DATA ANALYSIS (EDA)

Second: Data Exploring & Visualization

Scatter Plot

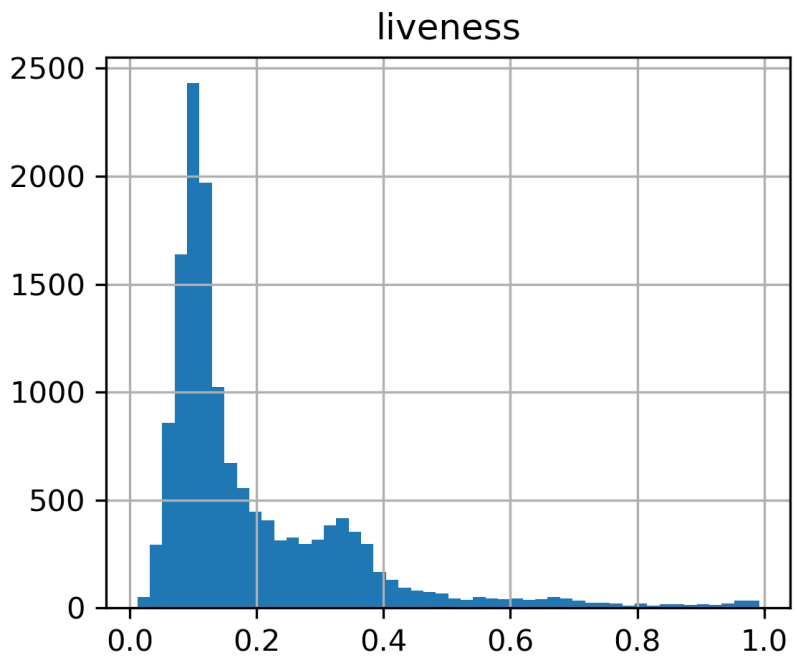
Is there a relationship between the loudness of a song and its popularity, and does this relationship differ based on the intensity of danceability and energy levels of the songs?



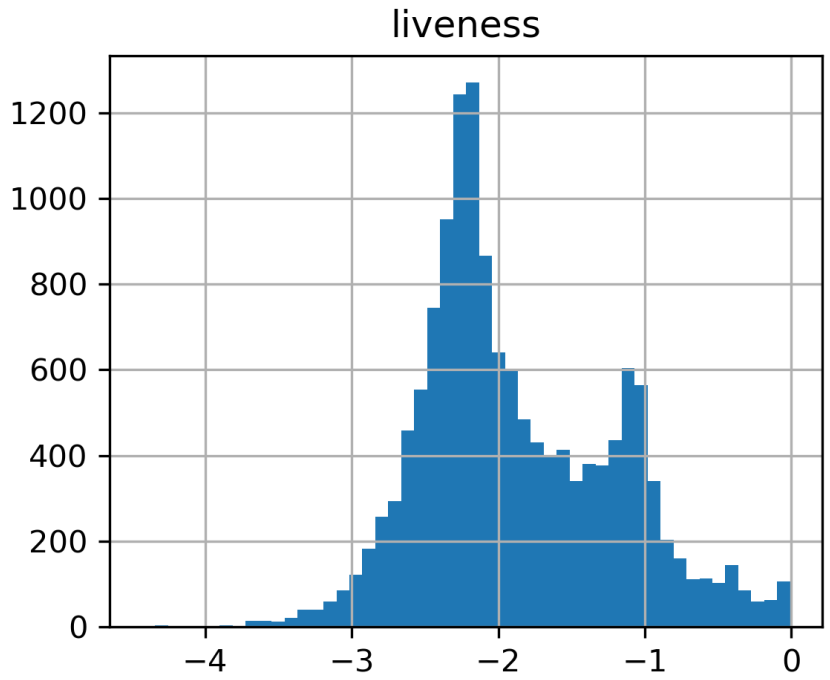
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Making data normally distributed(Liveliness)



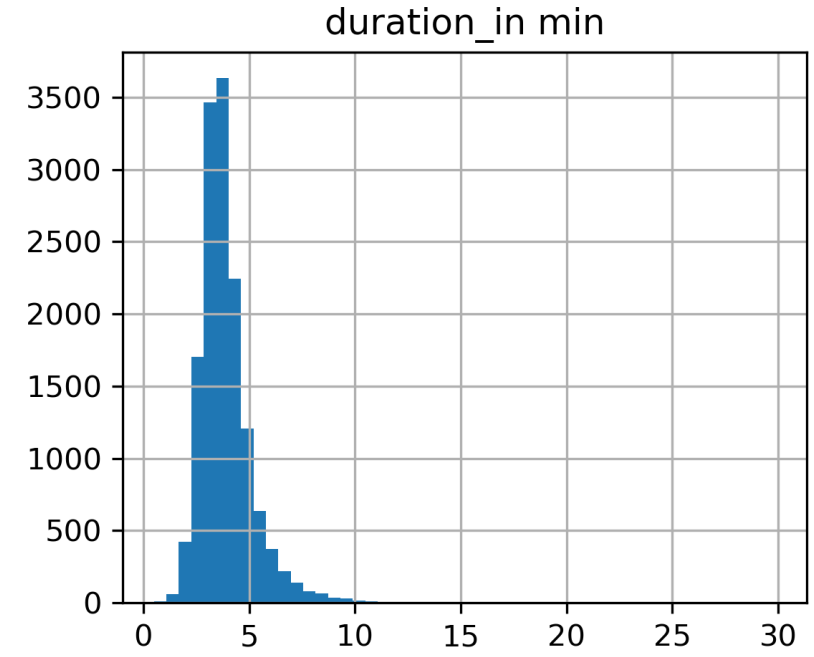
LOG



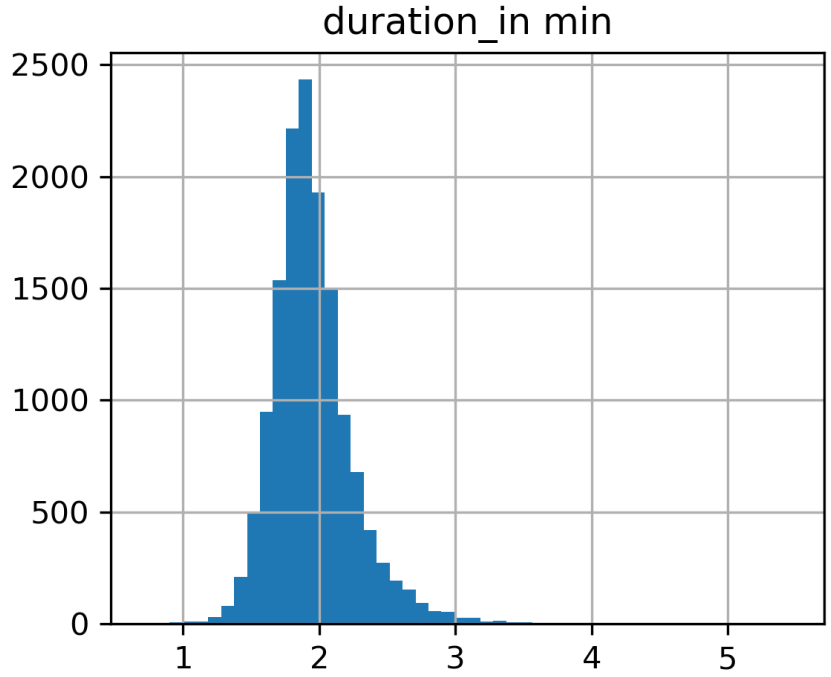
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Making data normally distributed (Duration in min)



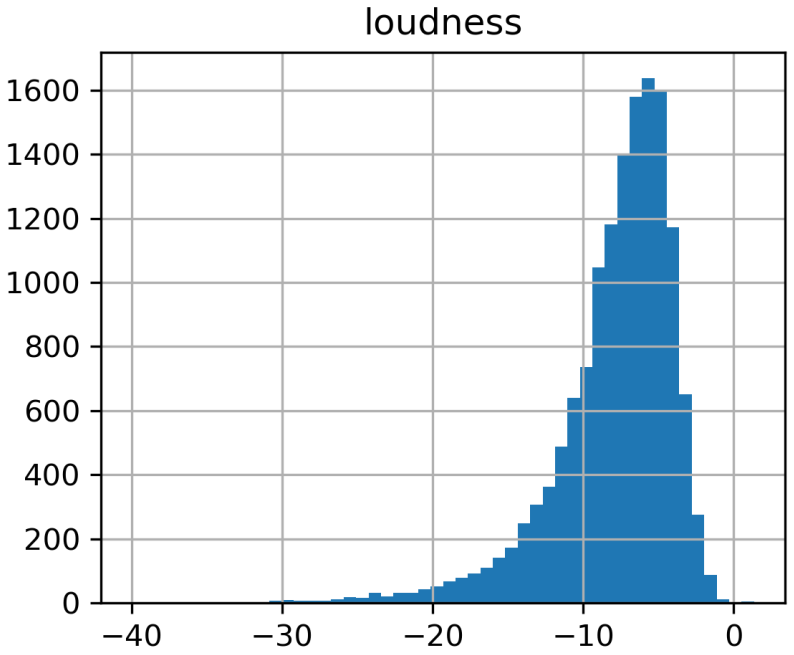
SQRT



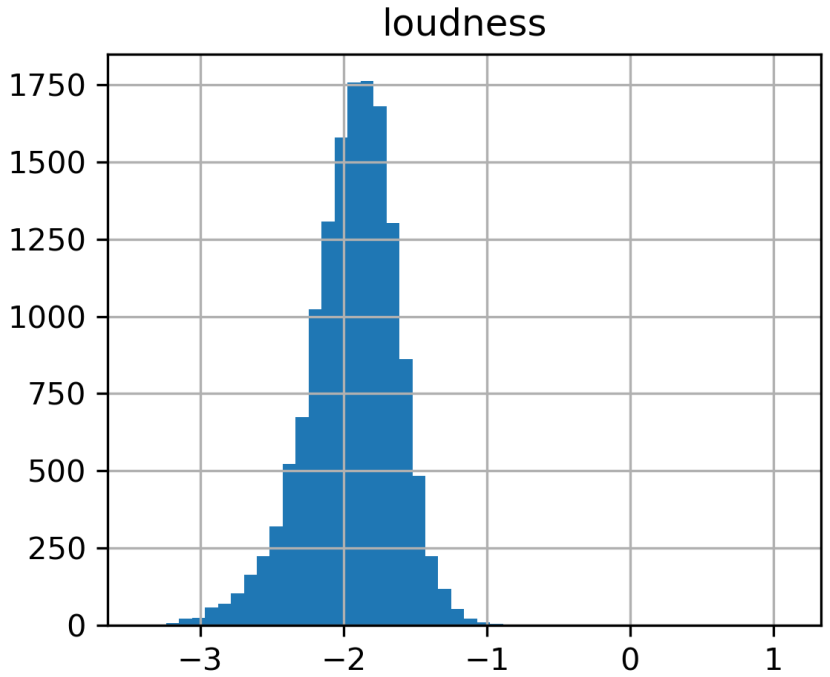
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Making data normally distributed (Loudness) with scaling



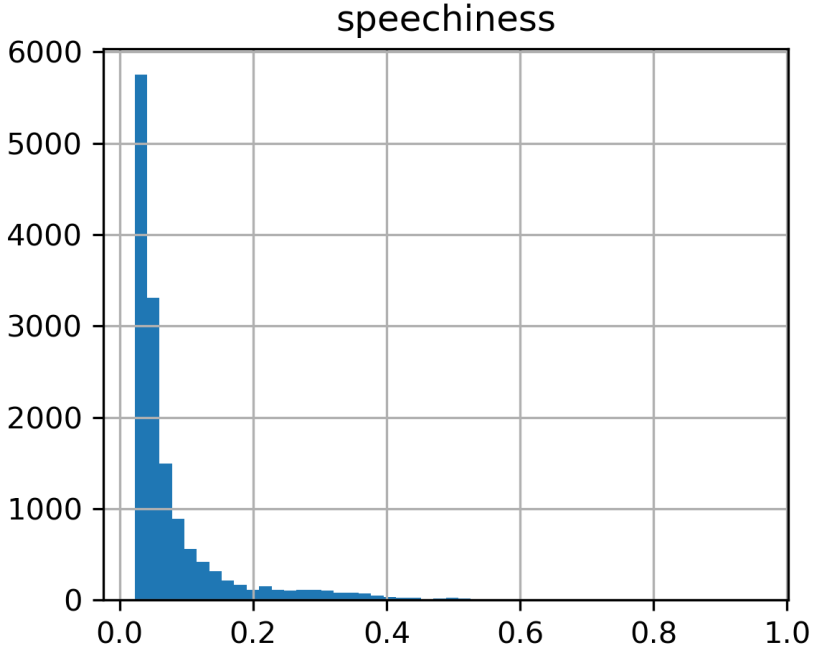
CBRT



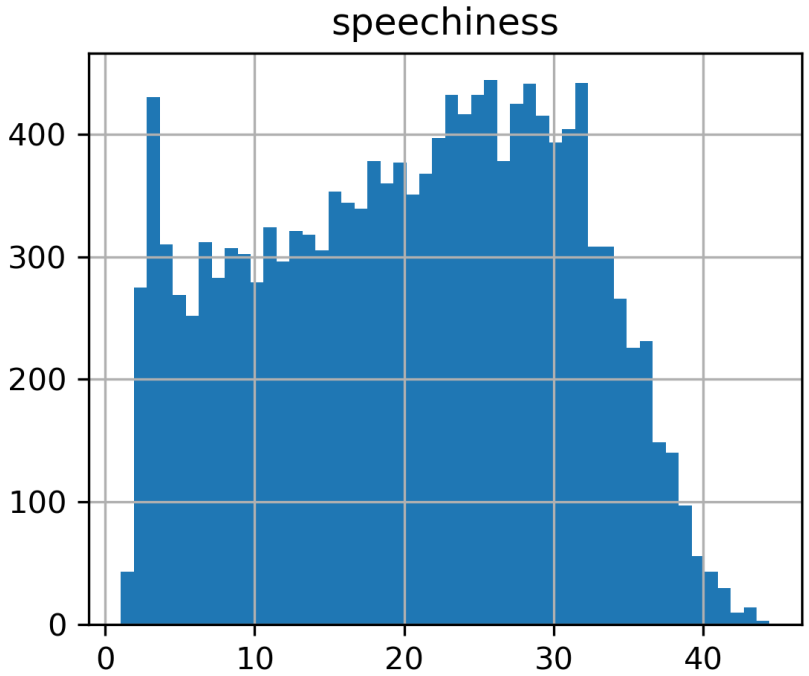
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Making data normally distributed(Speechiness)



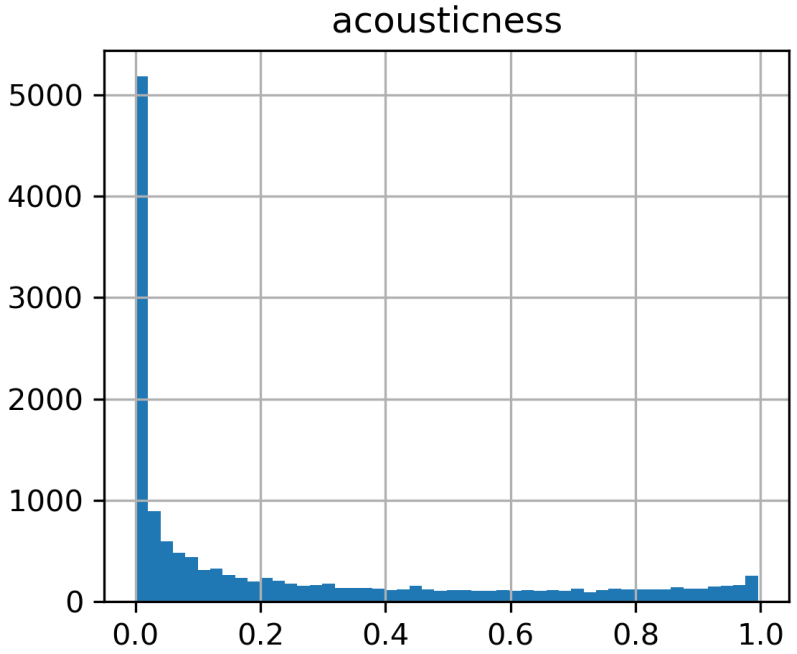
Reciprocal



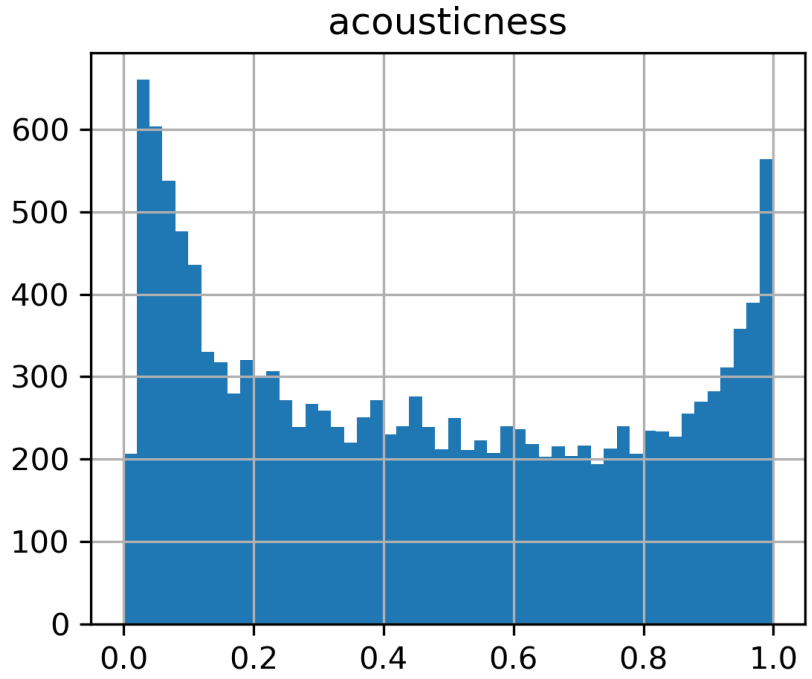
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Making data normally distributed(Acousticness)



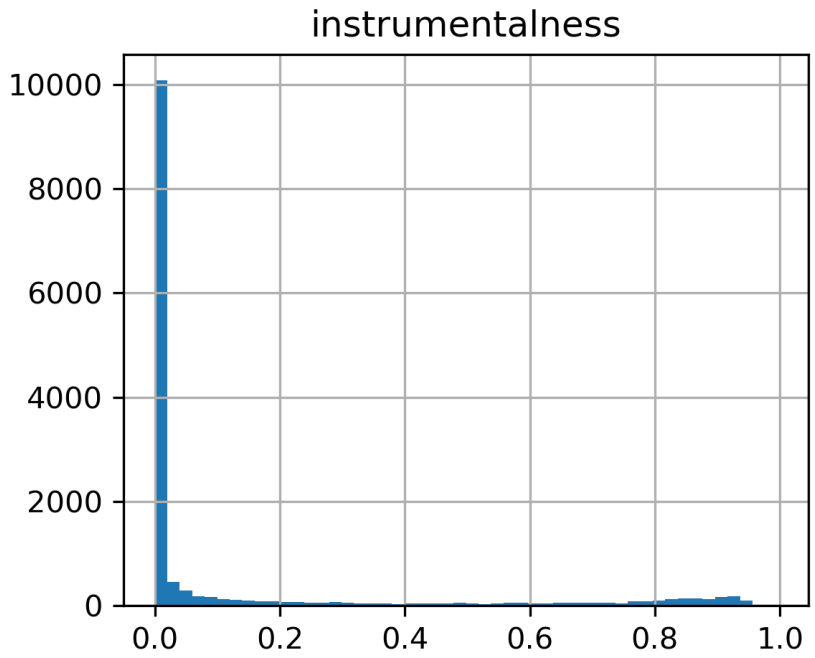
CBRT



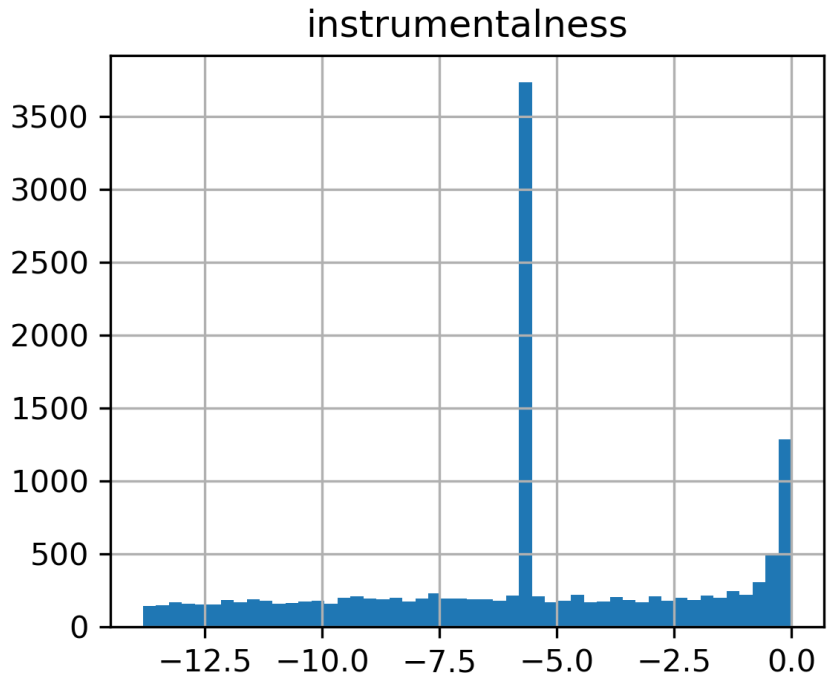
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Making data normally distributed(Instrumentalness)



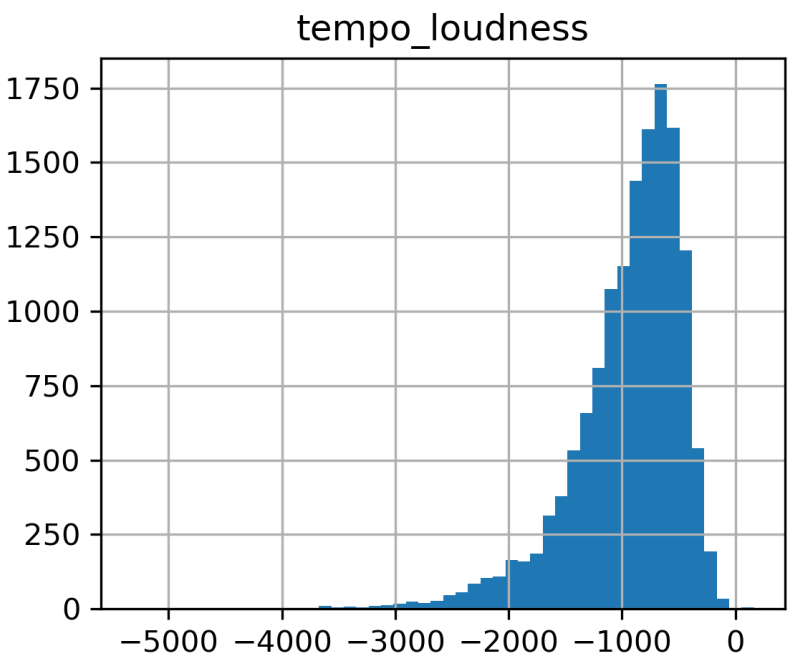
LOG



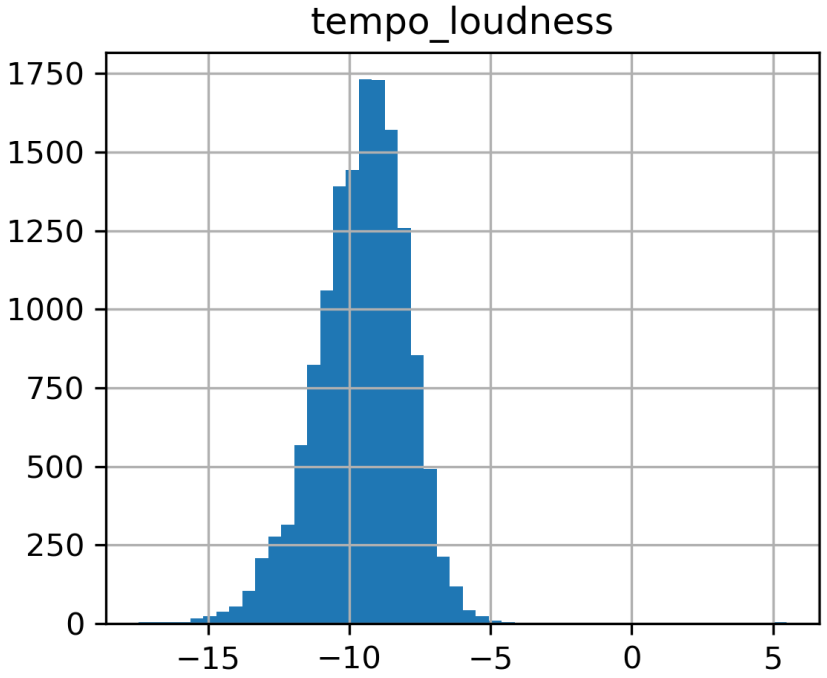
EXPLORATORY DATA ANALYSIS (EDA)

Third: Data Transformations & Engineering

Our model underfit at first, so we decided to make a new polynomial feature consisting of tempo times loudness which gave better results



CBRT

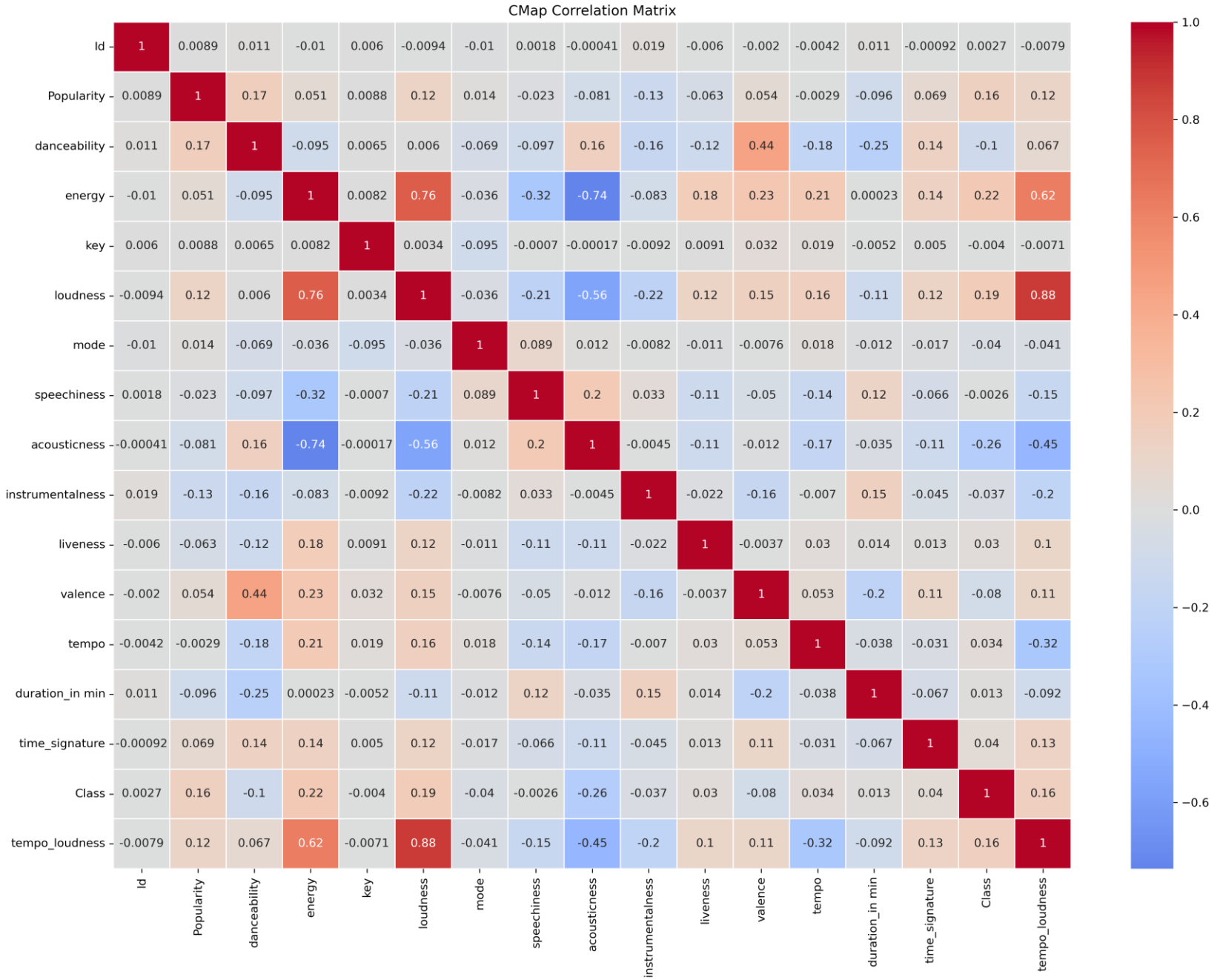


EDA

Correlation heatmap

The correlation between features ranges from -1 to 1, where negative values indicate negative correlation, positive values imply positive correlation, and 0 represents no correlation. Higher values indicate stronger correlations, regardless of their direction (positive or negative). The redness of the box is correlated with the values.

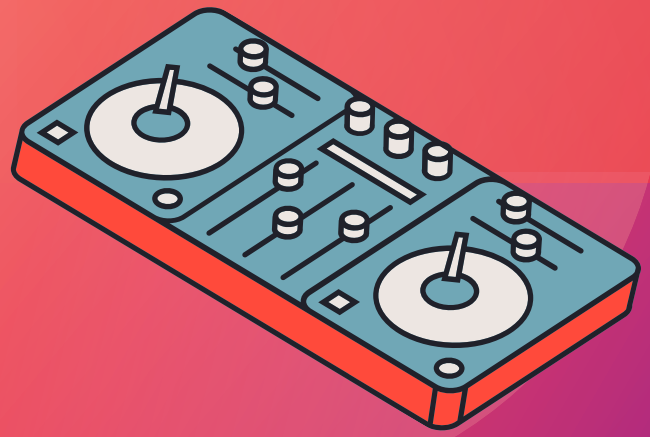
What correlations can we conclude from this? Which features are strongly each others?



EDA

Wrangle Function

To clean and prepare our data in one step!

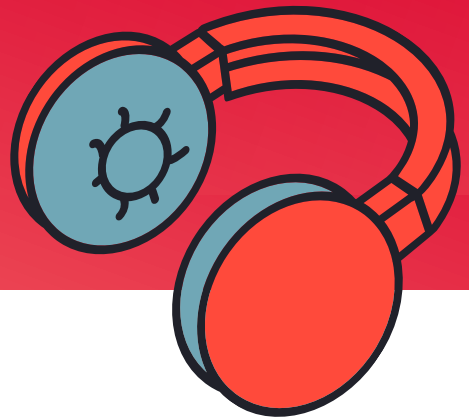


```python
def wrangle(df):
    # List of columns that will be dropped
    drop_cols = []

    # Transforming all ms rows to min
    df.loc[df['duration_in min/ms'] > 5000 ,'duration_in min/ms'] = df.loc[df['duration_in min/ms'] > 5000 ,
                                                                           'duration_in min/ms']/60000
    df.rename(columns={"duration_in min/ms": "duration_in min"}, inplace=True)

    # Handle null values
    mean_cols = ["Popularity"]
    median_cols = ["instrumentalness"]
    mode_cols = ["key"]

    mean_imputer = SimpleImputer(strategy="mean")
    median_imputer = SimpleImputer(strategy="median")
    mode_imputer = SimpleImputer(strategy="most_frequent")

    df[mean_cols] = mean_imputer.fit_transform(df[mean_cols])
    df[median_cols] = median_imputer.fit_transform(df[median_cols])
    df[mode_cols] = mode_imputer.fit_transform(df[mode_cols])

    # Add high cardinality columns to the list
    drop_cols.append("Artist Name")
    drop_cols.append("Track Name")

    # Change key column data type from float to int
    df["key"] = df["key"].astype(int)

    # Transform feature from others
    df["tempo_loudness"] = df["tempo"] * df["loudness"]

    # Applying transformations on the features
    df['liveness']=np.log(df['liveness'])
    df['duration_in min']=np.sqrt(df['duration_in min'])
    df['loudness']=np.cbrt(df['loudness'])
    df['tempo_loudness']=np.cbrt(df['tempo_loudness'])
    df['speechiness']=np.reciprocal(df['speechiness'])
    df['acousticness']=np.cbrt(df['acousticness'])
    df['instrumentalness']=np.log(df['instrumentalness'])

    # Scale data
    columns = df.select_dtypes("float").columns
    scaler = StandardScaler()
    df[columns] = scaler.fit_transform(df[columns])

    # Add ID column to drop list
    drop_cols.append("Id")

    # Add non-corr features that will be dropped
    drop_cols.append("loudness")
    drop_cols.append('liveness')
    drop_cols.append("key")


    # Drop columns
    df.drop(columns=drop_cols, inplace=True)

    return df
```

MODEL SELECTION & TRAINING

Features Selection

Dropping high cardinality columns, non/high correlation columns



```
high_cardinality_cols = ["Artist Name", "Track Name"]
train.drop(columns=high_cardinality_cols, inplace=True)
```

```
non_corr_cols = ["Id", "liveness", "loudness", "key"]
train.drop(columns=non_corr_cols, inplace=True)
```

```
train.columns
```

```
Index(['Popularity', 'danceability', 'energy', 'mode', 'speechiness',
       'acousticness', 'instrumentalness', 'valence', 'tempo',
       'duration_in min', 'time_signature', 'Class', 'tempo_loudness'],
      dtype='object')
```

MODEL SELECTION & TRAINING

Features Scaling

Scale continues columns using standard scaler

```
columns = train.select_dtypes("float").columns

scaler = StandardScaler()
train[columns] = scaler.fit_transform(train[columns])
```

MODEL SELECTION & TRAINING

Train/Test split

Using stratifying technique, we have split our data

An unbiased evaluation is crucial to avoid introducing temporal leakage and to ensure the model's generalization to future data. The traditional random train-test split can lead to biased results, undermining the model's performance assessment. To address this, a "stratified split" based on the time signature column.

time_signature	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	0.60	0.59	0.56	-2.33	-8.07
3	6.90	6.91	6.53	0.07	-5.46
4	91.34	91.35	91.60	0.02	0.28
5	1.15	1.15	1.32	-0.63	14.43

MODEL SELECTION & TRAINING

Lazy predict

Lazy Predict help build a lot of basic models without much code and helps understand which models works better without any parameter tuning

Model				
NearestCentroid	0.35	0.46	None	0.33
QuadraticDiscriminantAnalysis	0.46	0.45	None	0.43
GaussianNB	0.41	0.44	None	0.38
XGBClassifier	0.47	0.43	None	0.46
LGBMClassifier	0.48	0.43	None	0.47
SVC	0.50	0.42	None	0.46
ExtraTreesClassifier	0.46	0.41	None	0.44
RandomForestClassifier	0.46	0.41	None	0.45
BaggingClassifier	0.42	0.40	None	0.42
LogisticRegression	0.46	0.39	None	0.42
LinearDiscriminantAnalysis	0.44	0.39	None	0.41
KNeighborsClassifier	0.38	0.39	None	0.38
SGDClassifier	0.40	0.38	None	0.37
CalibratedClassifierCV	0.45	0.37	None	0.41
AdaBoostClassifier	0.38	0.37	None	0.37
LinearSVC	0.44	0.37	None	0.38
LabelPropagation	0.34	0.33	None	0.34
LabelSpreading	0.34	0.33	None	0.34
Perceptron	0.37	0.33	None	0.32
BernoulliNB	0.35	0.33	None	0.31
DecisionTreeClassifier	0.31	0.31	None	0.32
ExtraTreeClassifier	0.30	0.30	None	0.30
RidgeClassifier	0.42	0.30	None	0.35
RidgeClassifierCV	0.42	0.30	None	0.35
PassiveAggressiveClassifier	0.31	0.20	None	0.29
DummyClassifier	0.27	0.09	None	0.12

MODEL SELECTION & TRAINING

Hyperparameter Tunning

Using GridSearchCV [old run]

```
param_grid = {
    'C': [0.1, 1, 10],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto']
}

grid_search = GridSearchCV(SVC(), param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
best_score = grid_search.best_score_
best_model.fit(X_train, y_train)
predictions = best_model.predict(X_test)

accuracy = accuracy_score(y_test, predictions)
print(accuracy)
```

```
0.4913013221990257
```

```
best_params
```

```
3]: {'C': 10, 'gamma': 'scale', 'kernel': 'rbf'}
```

```
best_model
```

```
4]: SVC(C=10)
```

MODEL SELECTION & TRAINING

Support Vector Machine

```
In [ ]: model = SVC(C=10)

In [44]: model.fit(X_train, y_train)

Out[44]: SVC(C=10)

In [45]: y_pred = model.predict(X_test)

In [46]: accuracy_score(y_test, y_pred)

Out[46]: 0.5038194444444445
```

MODEL SELECTION & TRAINING

Ensemble Methods

XGBOOST

```
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test)

params = {
    'objective': 'multi:softmax',
    'num_class': 12,
    'eval_metric': 'mlogloss',
    'max_depth': 3,
    'eta': 0.15
}

num_rounds = 50
xgboost_model = xgb.train(params, dtrain, num_rounds)

y_predXGB = xgboost_model.predict(dtest)
y_pred_binary = [round(pred) for pred in y_predXGB]

accuracy = accuracy_score(y_test, y_pred_binary)
print(accuracy)
```

```
0.5114583333333333
```

THE WAY TO GET STARTED IS TO QUIT TALKING AND BEGIN DOING.

Walt Disney

THANK YOU

Ashraf Abdulkhaliq

Hamza Al-Habash

Ammar Abushukur