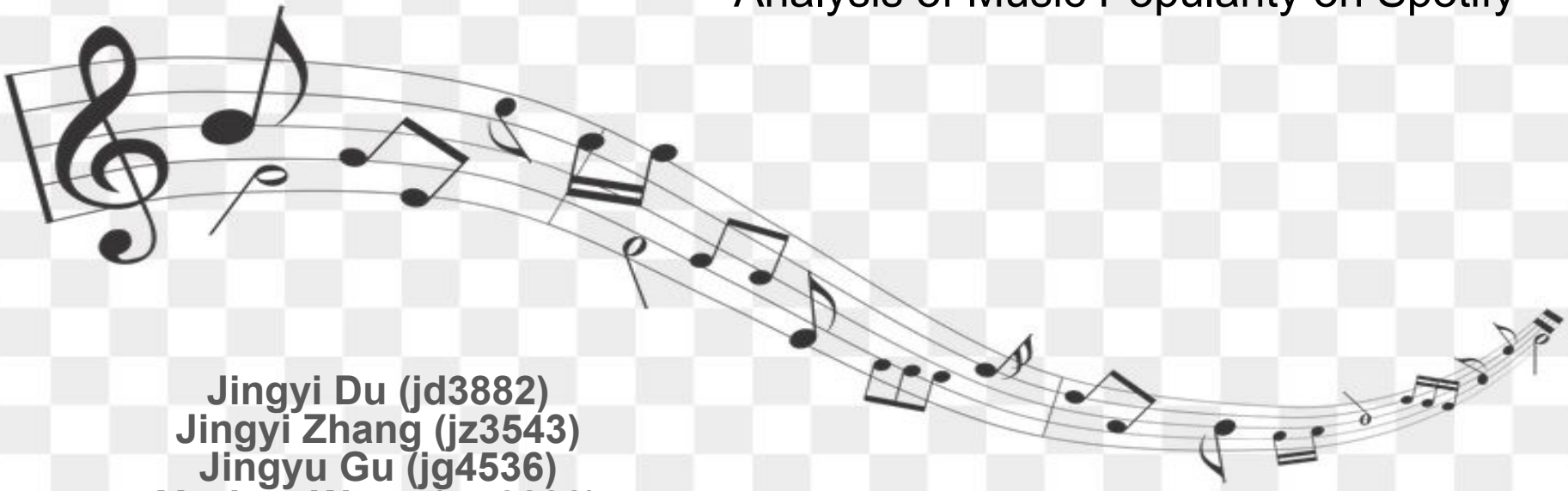
What Makes a Song Popular?

-Analysis of Music Popularity on Spotify



Jingyi Du (jd3882)
Jingyi Zhang (jz3543)
Jingyu Gu (jg4536)
Yuchen Wang (yw3890)

05.04.2023

Motivation:
Why is this song popular?

- What are the most popular genres of music on Spotify, and how have they changed over time?
- Are there any trends in the characteristics of popular songs, such as tempo or duration?
- How to recommend songs based on user listening records?

Dataset

- the data source we are using for this project is “the Spotify data 1921 - 2020” dataset in kaggle database.

  Url: https://www.kaggle.com/datasets/ektanegi/spotifydata-19212020

- The dataset is from Spotify and it contains hit songs from each year from the year 1921 to year 2020.

Dataset

- For each row, it contains the popularity of each song and its 13 characteristics.
- The closer the score is to 1, the stronger this feature is.

Factors	Explanation
Popularity	Popularity of the song. From 0 to 100. The more popular the music is, the closer the attribute value is to 100.
Acousticness	Acoustic music refers to the pure band performance version that is not processed by electronic sound effects. The closer the music is to the original sound, the closer the attribute value is to 1.0.
Danceability	Danceability describes how danceable a track is in terms of a combination of musical elements.
Duration	Track duration in milliseconds.
Energy	Energy refers to a perceived measure of intensity and activity.
Explicit	Many artists use profanity in their music. This is a situation where music should be labeled as explicit.
Instrumentalness	Instrumentalness refers to whether a track contains no vocals. The fewer vocals the music contains, the closer the attribute value is to 1.0.
key	The estimated pitch of the song.
Liveness	A higher liveness value indicates an increased likelihood that the piece will be played live.
Loudness	The overall loudness of the composition in decibels (dB).
Mode	“Any of several ways of ordering the notes of a scale according to the intervals they form with the tonic, thus providing a theoretical framework for the melody.”
Speechiness	Speechiness detects the presence of spoken words in a track. The more the recording resembles speech (such as an audiobook), the closer the attribute value is to 1.0.
Tempo	The overall estimated tempo of the track in beats per minute (BPM).
Valence	Valence is used to describe the musical positivity conveyed by a track.

Dataset

- Categorical Features:

  Mode, Explicit, Key

- Numerical Features:

  Acousticness, Danceability, Duration_ms, Energy, Instrumentalness, Liveness, Loudness, Popularity, Speechiness, Tempo, Valence, Year

Factors	Explanation
Popularity	Popularity of the song. From 0 to 100. The more popular the music is, the closer the attribute value is to 100.
Acousticness	Acoustic music refers to the pure band performance version that is not processed by electronic sound effects. The closer the music is to the original sound, the closer the attribute value is to 1.0.
Danceability	Danceability describes how danceable a track is in terms of a combination of musical elements.
Duration	Track duration in milliseconds.
Energy	Energy refers to a perceived measure of intensity and activity.
Explicit	Many artists use profanity in their music. This is a situation where music should be labeled as explicit.
Instrumentalness	Instrumentalness refers to whether a track contains no vocals. The fewer vocals the music contains, the closer the attribute value is to 1.0.
key	The estimated pitch of the song.
Liveness	A higher liveness value indicates an increased likelihood that the piece will be played live.
Loudness	The overall loudness of the composition in decibels (dB).
Mode	“Any of several ways of ordering the notes of a scale according to the intervals they form with the tonic, thus providing a theoretical framework for the melody.”
Speechiness	Speechiness detects the presence of spoken words in a track. The more the recording resembles speech (such as an audiobook), the closer the attribute value is to 1.0.
Tempo	The overall estimated tempo of the track in beats per minute (BPM).
Valence	Valence is used to describe the musical positivity conveyed by a track.

Dataset Information

Statistics Summary

	count	mean	std	min	25%	50%	75%	max
acousticness	169909.0	0.493214	0.376627	0.0	0.0945	0.492000	0.8880	0.996
danceability	169909.0	0.538150	0.175346	0.0	0.4170	0.548000	0.6670	0.988
duration_ms	169909.0	231406.158973	121321.923219	5108.0	171040.0000	208600.000000	262960.0000	5403500.000
energy	169909.0	0.488593	0.267390	0.0	0.2630	0.481000	0.7100	1.000
explicit	169909.0	0.084863	0.278679	0.0	0.0000	0.000000	0.0000	1.000
instrumentalness	169909.0	0.161937	0.309329	0.0	0.0000	0.000204	0.0868	1.000
key	169909.0	5.200519	3.515257	0.0	2.0000	5.000000	8.0000	11.000
liveness	169909.0	0.206690	0.176796	0.0	0.0984	0.135000	0.2630	1.000
loudness	169909.0	-11.370289	5.666765	-60.0	-14.4700	-10.474000	-7.1180	3.855
mode	169909.0	0.708556	0.454429	0.0	0.0000	1.000000	1.0000	1.000
popularity	169909.0	31.556610	21.582614	0.0	12.0000	33.000000	48.0000	100.000
speechiness	169909.0	0.094058	0.149937	0.0	0.0349	0.045000	0.0754	0.969
tempo	169909.0	116.948017	30.726937	0.0	93.5160	114.778000	135.7120	244.091
valence	169909.0	0.532095	0.262408	0.0	0.3220	0.544000	0.7490	1.000
year	169909.0	1977.223231	25.593168	1921.0	1957.0000	1978.000000	1999.0000	2020.000

Dataset Size & Types

```
 #   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
```

Data Cleaning

1.2. Missing Values

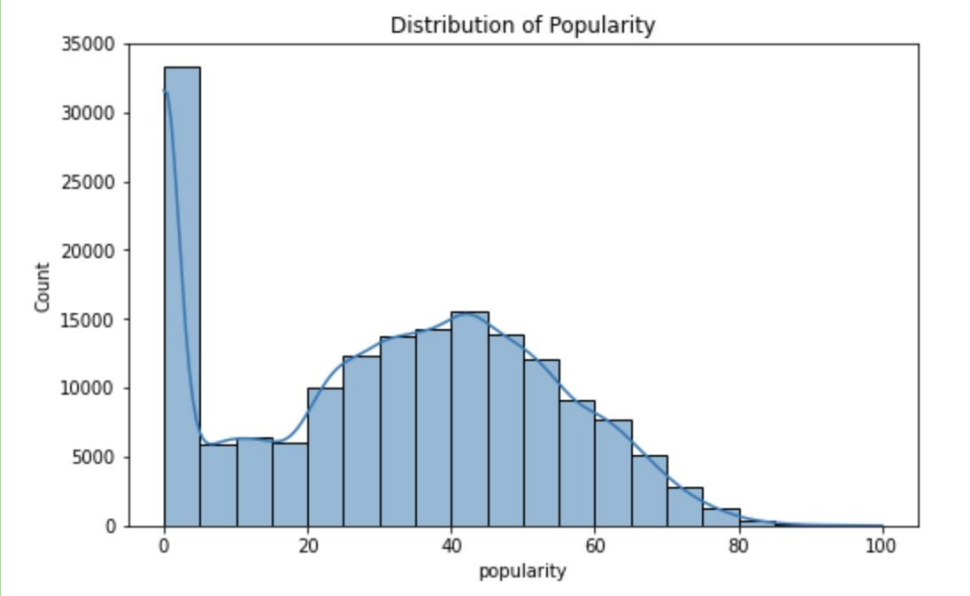
```
# Detection
# Columns that contain NA values
df.columns[df.isnull().any()]
df = df.dropna()

# Column names and the total number of NA values that each column contains
df[df.columns[df.isnull().any()]].isna().sum()

# Calculate Missing rate and dropping columns
missing_rate = np.sum(df.isna(), axis = 0).values/df.shape[0]
print(missing_rate)

# There is no NA Values in each column
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

- We dropped all NA values

Data Visualization: Dependent Variable

The Distribution of Popularity



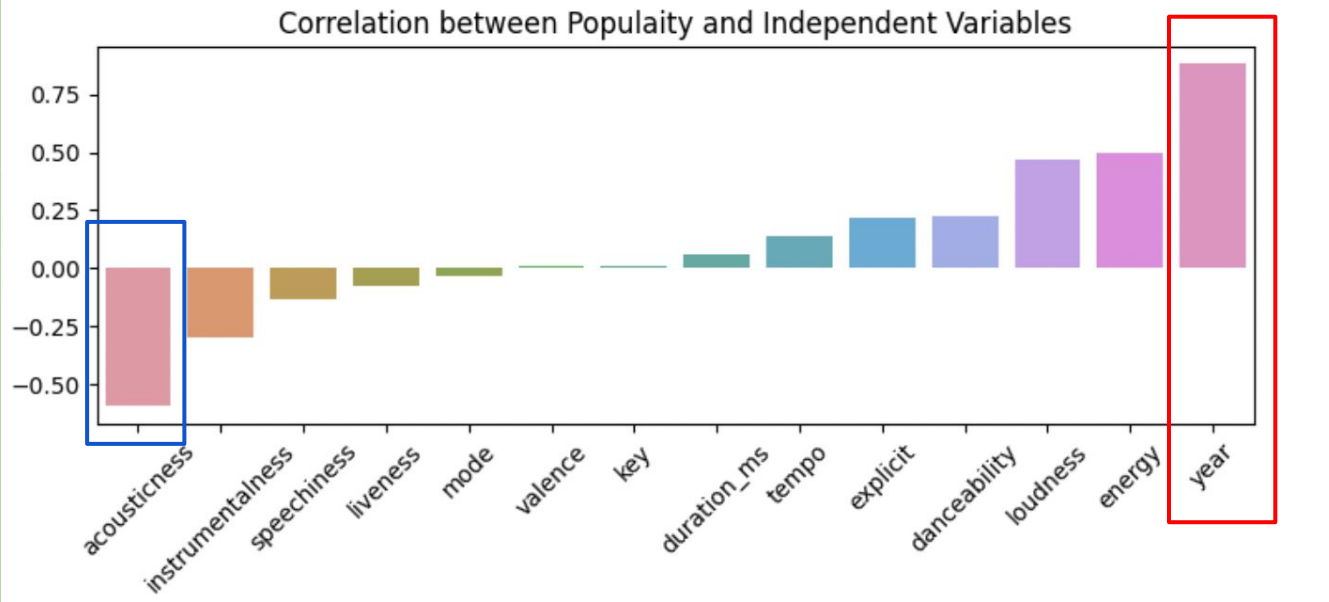
Mean = 31.56, Median=33.00, Standard Deviation = 21.58

Insights:

1. A majority songs have the popularity equal to 0 due to their early release year
2. Without considering about songs with a popularity 0, the distribution of popularity looks like normal distribution
3. There are no so much extremely popular songs or extremely not popular songs

Data Visualization: Dependent Variable

The Correlation between Popularity and Independent Variables

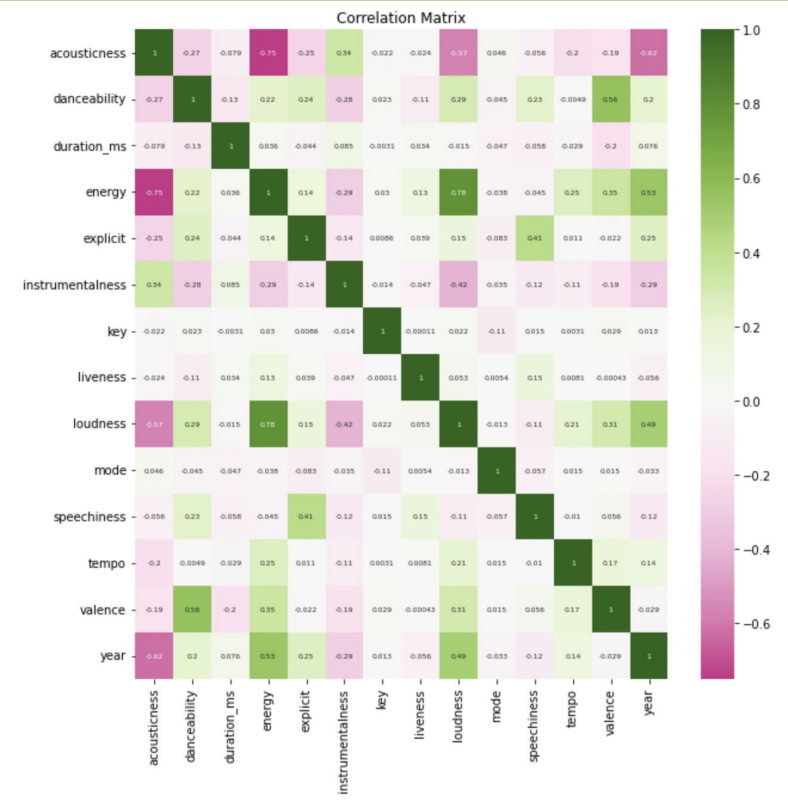


- Year influences the popularity of songs most positively
- Acousticness influences the popularity of songs most negatively

Data Visualization: Independent Variables

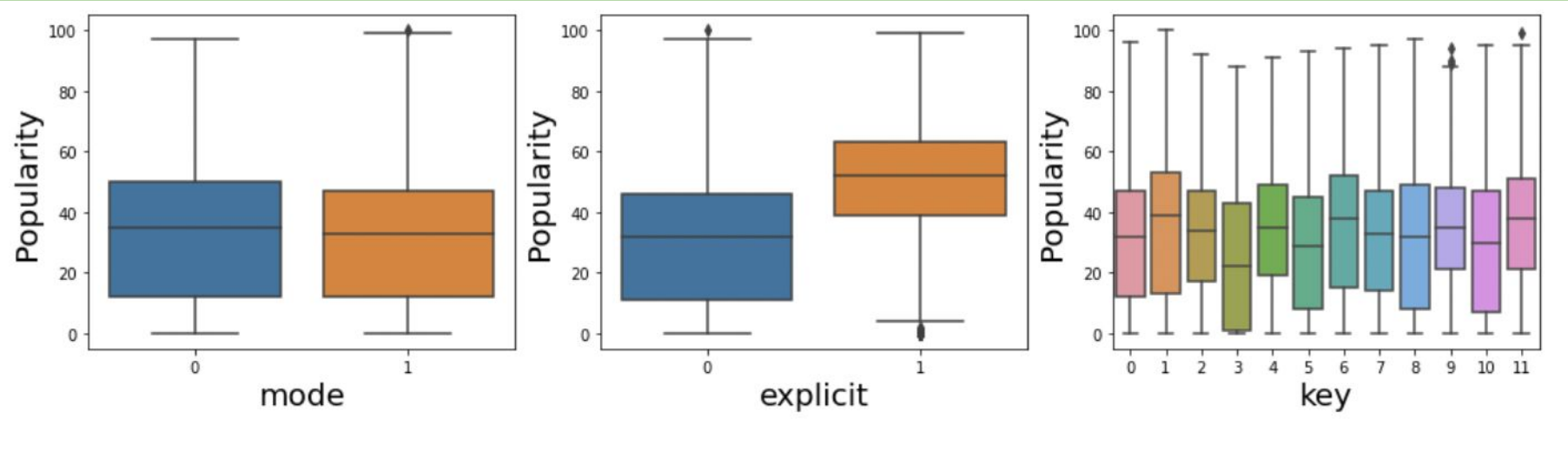
The Correlation between Independent Variables

- Test the collinearity
- No high-correlated features



Data Visualization: Independent Variables

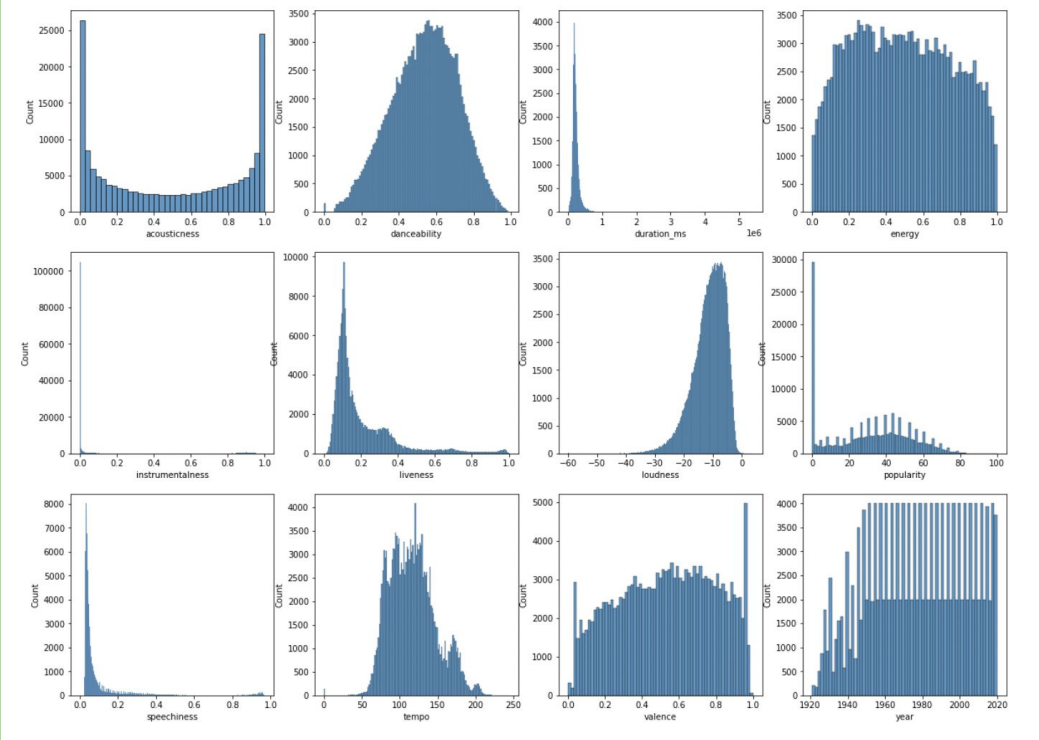
The Boxplots of Categorical Features



- The influence of categorical features on songs' popularity

Data Visualization: Independent Variables

The Histograms of Numerical Features



- Distributions of each numerical features

Machine Learning Preprocessing

- Categorical Features Encoding
  - Ordinal Encoding for “key”
  - One-Hot Encoding for mode and explicit which are dummy variables
- Scaling:
  - StandardScaler()
  - Train test 0.8/0.2 split.
  - Scaling for X_train and X_test

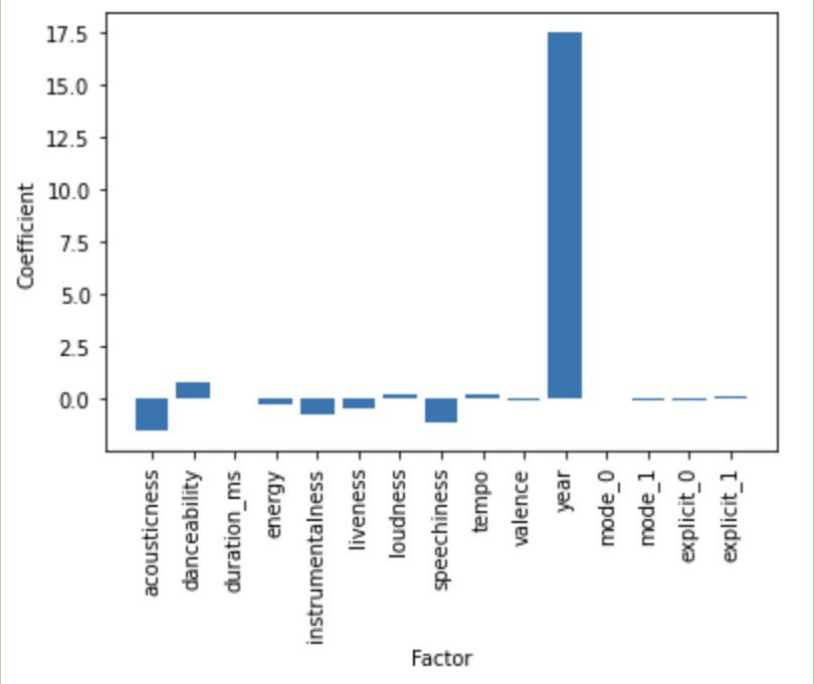
Machine Learning: Linear Regression

Linear Regression:

```
[37] lr.coef_

array([-1.53591828,  0.80976957,  0.05081846, -0.27031072, -0.77669475,
       -0.46334896,  0.16726247, -1.15209312,  0.15258958, -0.05629723,
       17.46652945,  0.05171866, -0.05171866, -0.06758168,  0.06758168])
```

- Score of Linear Regression on the test set is score = 0.7855
- MSE: 100.69
- The newer the song is, the higher popularity it will have

Feature Importance of Linear Regression



Machine Learning: Decision Tree

Decision Tree Classifier:

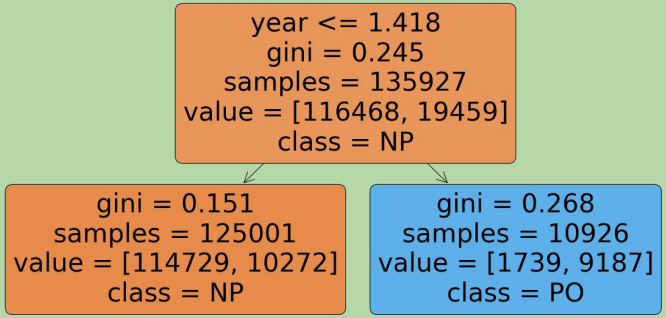
- Set popularity score of 55 as the criterion for judging whether the song is popular or not popular
- E.g., the song “Let It Ride” with the popularity score of 46 is an NP (not popular) and the song “With a Smile” with popularity scope of 63 is a PO (popular).

Let It Ride	46
Burn The Witch	46
With a Smile (feat. Kelly Badon)	63
Good Luck	53

Machine Learning: Decision Tree

Hyperparameter Tuning and Best Model Selection

```
[16] dt = DecisionTreeClassifier()

     # Use grid search to find the optimal max depth value
     max_depth_range = range(1, 11)
     param_grid = {'max_depth': max_depth_range}
     grid = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy')
     grid.fit(tX_train, ty_train)

     # Print the best max depth value and corresponding accuracy score
     print('Best max depth value: {}'.format(grid.best_params_['max_depth']))
     print('Best accuracy score: {:.2f}'.format(grid.best_score_))

     Best max depth value: 1
     Best accuracy score: 0.91
```



Insights:

- Max_depth =1, tree only consider “year” as the criterion for classification
- The attribute “year” has too strong influence.
- To get deeper tree, we need to drop it.

Machine Learning: Decision Tree

Hyperparameter Tuning and Best Model Selection

```
dt = DecisionTreeClassifier()

# Use grid search to find the optimal max depth value
max_depth_range = range(1, 11)
param_grid = {'max_depth': max_depth_range}
grid = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy')
grid.fit(tX_train, ty_train)

# Print the best max depth value and corresponding accuracy score
print('Best max depth value: {}'.format(grid.best_params_['max_depth']))
print('Best accuracy score: {:.2f}'.format(grid.best_score_))
```

```
Best max depth value: 9
Best accuracy score: 0.87
```

Model Performance:

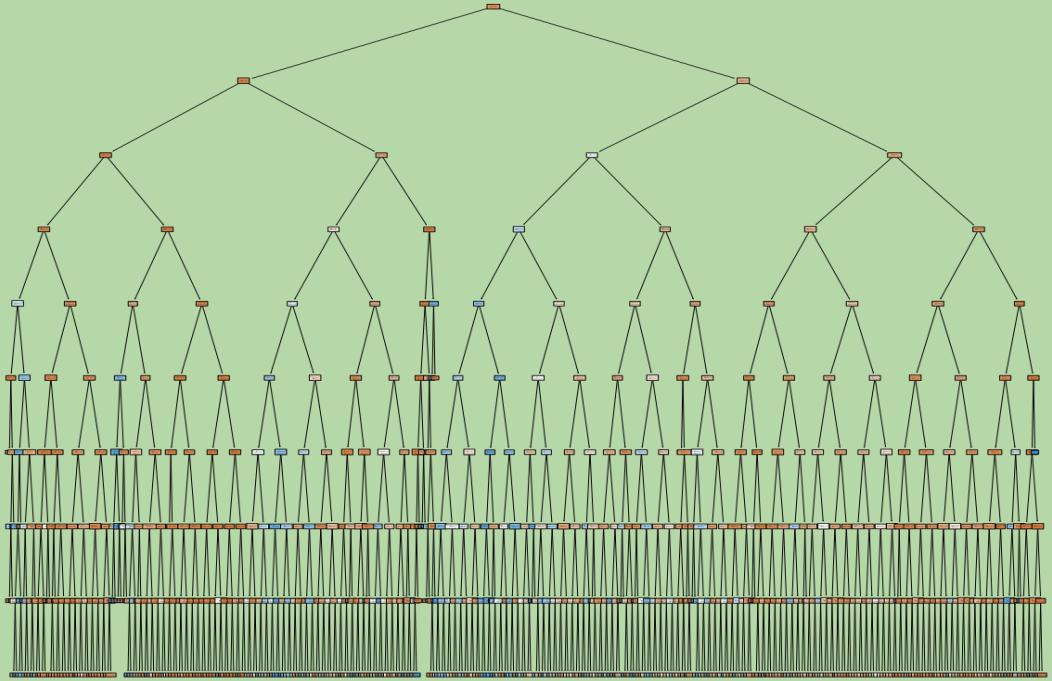
- The best max depth=9
- Model accuracy=0.87

Insights:

- After dropping “year”, we achieve a deeper tree which take more features into consideration
- Avoid Over-fitting on a specific dataset

Machine Learning: Decision Tree

Feature Importance Evaluation - Top 3



```
acousticness: 0.05059972555416042
danceability: 0.05629077781948119
duration_ms: 0.09289909415022382
energy: 0.06922837361335663
instrumentalness: 0.05399879488653735
liveness: 0.013131385549398699
loudness: 0.3908178773313258
speechiness: 0.056998214281144634
tempo: 0.013305869449348834
valence: 0.08034646172886548
0: 0.0027227714438752453
mode_0: 0.0005633731633972945
mode_1: 0.0
explicit_0: 0.08496351489228168
explicit_1: 0.0341337661366029
```

- Year;
- Loudness;
- Duration of the song.

Machine Learning: Decision Tree

Model Performance on test set

R^2 - Decision Tree Classifier

```
[86]
     #decision tree
     tree_class.score(tX_test,ty_test)

     0.8697251486080866
```

Machine Learning: Decision Tree

Decision Tree Regressor:

```
for max_d in range(1,10):
  model = DecisionTreeClassifier(max_depth=max_d, random_state=42)
  model.fit(tX_train, ty_train)
  print('The Training Accuracy for max_depth {} is:'.format(max_d), model.score(tX
  print('The Validation Accuracy for max_depth {} is:'.format(max_d), model.score(
  print('')
```

```
The Training Accuracy for max_depth 1 is: 0.8568422756332443
The Validation Accuracy for max_depth 1 is: 0.8554234594785475

The Training Accuracy for max_depth 2 is: 0.8568422756332443
The Validation Accuracy for max_depth 2 is: 0.8554234594785475

The Training Accuracy for max_depth 3 is: 0.8636106145210297
The Validation Accuracy for max_depth 3 is: 0.8612795009122477

The Training Accuracy for max_depth 4 is: 0.8668623599432048
The Validation Accuracy for max_depth 4 is: 0.8671943970337237

The Training Accuracy for max_depth 5 is: 0.8683043104019069
The Validation Accuracy for max_depth 5 is: 0.8672532517214996

The Training Accuracy for max_depth 6 is: 0.8701141053653799
The Validation Accuracy for max_depth 6 is: 0.8681949267259137

The Training Accuracy for max_depth 7 is: 0.8720710381307614
The Validation Accuracy for max_depth 7 is: 0.8686069095403449

The Training Accuracy for max_depth 8 is: 0.8760658294525738
The Validation Accuracy for max_depth 8 is: 0.8716379259608028

The Training Accuracy for max_depth 9 is: 0.8789423734798826
The Validation Accuracy for max_depth 9 is: 0.8698428579836384
```

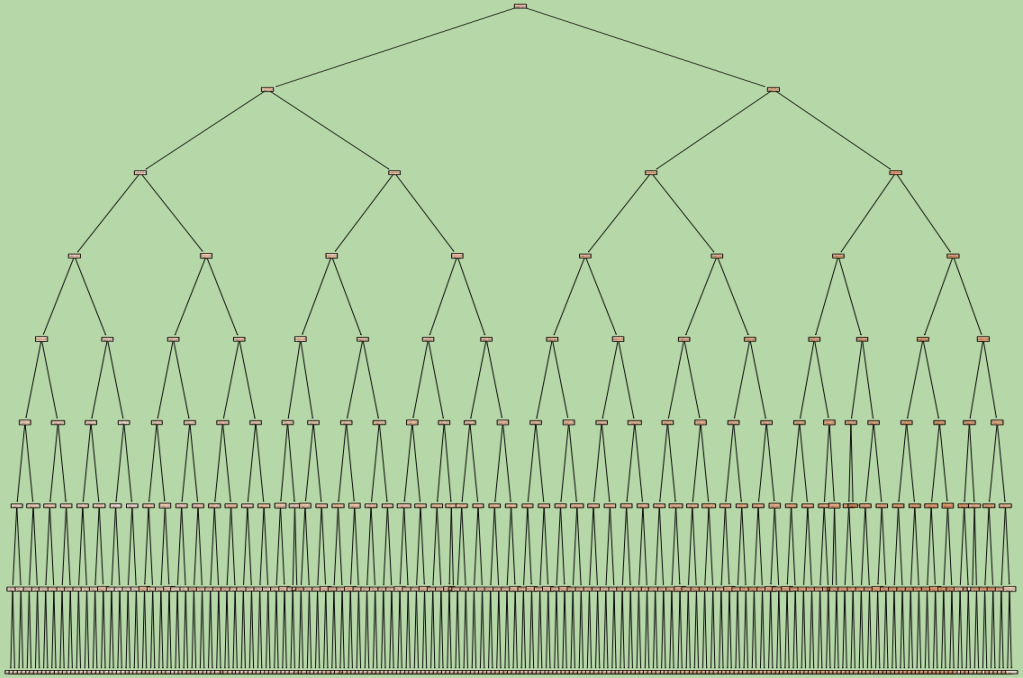
Model Performance:

- The best max depth=8
- Model R^2 =0.87

We select the best model with the maximum validation accuracy

Machine Learning: Decision Tree

Feature Importance Evaluation - Top 3



```
acousticness: 0.05057903330802523
danceability: 0.05605616736506855
duration_ms: 0.09359325773316672
energy: 0.06915244095902404
instrumentalness: 0.05386830742001122
liveness: 0.01389964238408265
loudness: 0.39100970498225107
speechiness: 0.05777818735349089
tempo: 0.013699078095883034
valence: 0.08074767487505269
mode_0: 0.0
mode_1: 0.0004520463138336798
explicit_0: 0.0
explicit_1: 0.11916445921011025
```

- Year;
- Loudness;
- Explicit.

Machine Learning: Decision Tree

Model Performance on test set

R^2 - Decision Tree Regressor

```
[141] tree_reg.score(X_test,y_test)

      0.8238528809138743
```

MSE - Decision Tree Regressor

```
[ ]  #Evaluate the model using DecisionTreeRegressor
     tree_reg_pred = tree_reg.predict(X_test)
     tree_mse = mean_squared_error(y_test,tree_reg_pred)

     tree_mse

     82.70101384707216
```

Machine Learning: Random Forest

Random Forest Classifier

```
parameters = {'n_estimators':np.arange(50,200,15),
              'max_features':np.arange(0.1, 1, 0.1),
              'max_depth': [3, 5, 7, 9],
              'max_samples': [0.3, 0.5, 0.8]}

rf_c = RSCV(RandomForestClassifier(), parameters, n_iter = 15).fit(tX_train, ty_train)
rf_c = rf_c.best_estimator_

#y_rfc_pred = model.predict(tX_test)
rf_c.fit(tX_train, ty_train)
```

- Find the best estimator using cross-validation

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=9, max_features=0.6, max_samples=0.3,
                       n_estimators=170, random_state=42)
```

- Max_depth = 9, max_feature = 0.6, max_samples=0.3, n_estimators = 170

Machine Learning: Random Forest

Model Performance - Random Forest Classifier

```
# Random forest accuracy
rf_c.score(tX_test,ty_test)
```

```
0.9714717234435308
```

Accuracy: 0.97

```
acousticness: 0.02169670374000964
danceability: 0.019846472893318024
duration_ms: 0.020331497987021333
energy: 0.021914366703246563
instrumentalness: 0.012673409260715016
liveness: 0.021512494018155623
loudness: 0.037111889189310424
speechiness: 0.024175365197544776
tempo: 0.021410884699261584
valence: 0.019098727595594035
year: 0.7540497244894294
mode_0: 0.0017645644433329738
mode_1: 0.0021388087489801607
explicit_0: 0.009589552055499056
explicit_1: 0.012685538978581476
```

Feature Importance - Top 5

1. Year
2. Loudness
3. Speechiness
4. Energy
5. Acousticness

Machine Learning: Random Forest

Random Forest Regressor

```
parameters = {'n_estimators':np.arange(50,200,15),
              'max_features':np.arange(0.1, 1, 0.1),
              'max_depth': [3, 5, 7, 9],
              'max_samples': [0.3, 0.5, 0.8]}

rf_reg = RSCV(RandomForestRegressor(), parameters, n_iter = 15).fit(X_train, y_train)
rf_reg = rf_reg.best_estimator_
```

- Find the best estimator using cross-validation

```
RandomForestRegressor
RandomForestRegressor(max_depth=7, max_features=0.8, max_samples=0.3,
                      n_estimators=50)
```

- Max_depth = 7, max_feature = 0.8, max_samples=0.3, n_estimators = 50

Machine Learning: Random Forest

Model Performance - Random Forest Regressor

```
#random forest regressor mse
y_rfr_pred = rf_reg.predict(X_test)
mean_squared_error(y_test,y_rfr_pred)
```

```
82.00924521381229
```

MSE: 82.00

```
acousticness: 0.12091817707865765
danceability: 0.00209052109107218
duration_ms: 0.0052768898122967905
energy: 0.03258039608997183
instrumentalness: 0.0025278695091314357
liveness: 0.002330469128196349
loudness: 0.020561822392536632
speechiness: 0.007403181672026032
tempo: 0.0012406467236190106
valence: 0.0018539596258975478
year: 0.7990324586769896
mode_0: 5.9447446217266006e-05
mode_1: 4.6305867863405704e-05
explicit_0: 0.0020658144977599648
explicit_1: 0.002012040387764534
```

Feature Importance - Top 5

1. Year
2. Acousticness
3. Energy
4. Loudness
5. Speechiness

Machine Learning: SVM

Support Vector Machine Classifier

```
from sklearn.svm import LinearSVC
svc = LinearSVC(max_iter=1500, dual=False)
param_grid = {'C': [0.1, 1, 10, 100]}

# Create a GridSearchCV object
grid = GridSearchCV(svc, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Train the model using the training sets
grid.fit(tX_train, ty_train)
print(grid.best_params_)
```

Hyperparameter Tuning

- Find the best regularization parameter C using cross-validation
- Best parameter is 0.1.

Machine Learning: SVM

Model Performance - SVC

Accuracy: 0.86
lower than decision tree(0.8697)
and random forest(0.9715)
So it is not a good method.

```
# Predict the target values of test data
svc_accuracy = grid.score(tX_test, ty_test)
svc_accuracy
```

```
{'C': 0.1}
0.8628685774821965
```

Feature Importance - Top 3

1. Year
2. Danceability
3. Acousticness

```
acousticness: 0.028185351445601754
danceability: 0.10321954371755754
duration_ms: -0.0813231804902278
energy: -0.034221853696447875
instrumentalness: -0.027393659105001977
liveness: -0.010833224143771244
loudness: -0.06231311459362136
speechiness: -0.008712130245731536
tempo: 0.024543982336081075
valence: -0.022200455274147487
year: 2.2341077223751227
mode_0: 0.016419824399775254
mode_1: -0.016419824399775254
explicit_0: 0.0033757757909015085
explicit_1: -0.0033757757909122946
```

Machine Learning: SVM

Support Vector Machine Regression

```
from sklearn.svm import LinearSVR
svr = LinearSVR()

# Define the parameter grid for hyperparameter tuning
param_grid = {'C': [0.1, 1, 10, 100]}

# Create a GridSearchCV object
grid = GridSearchCV(svr, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs

# Train the model using the training sets
grid.fit(X_train, y_train)
print(grid.best_params_)
```

Hyperparameter Tuning

- Find the best regularization parameter C using CV
- Best parameter is 10

Machine Learning: SVM

Model Performance - SVR

MSE: 103.8

```
# Predict the target values of test data
svr_mse = -grid.score(X_test,y_test)
svr_mse
```

```
{'C': 10}
/usr/local/lib/python3.9/dist-packages/sklearn/svm/_base.py:1244: ConvergenceWarning: Libl
  warnings.warn(
103.80110881996053
```

Feature Importance - Top 3

1. Year
2. Acousticness
3. Speechiness

```
acousticness: -0.6718307394314222
danceability: 0.035393039983286204
duration_ms: 0.01891485967838854
energy: -0.3679998076250845
instrumentalness: -0.13243262529268307
liveness: 0.0015581948685782555
loudness: -0.09355194320702825
speechiness: -0.525964688502589
tempo: 0.038665259582373095
valence: 0.27314472921817023
year: 17.61521087445993
mode_0: 0.12392010784131179
mode_1: -0.12392010784131179
explicit_0: -0.07226880880491808
explicit_1: 0.07226880880492172
```

Machine Learning: Adaboost

Adaboost Classifier

Hyperparameter Tuning and Best Model Selection

```python
# Grid Search
params = {'n_estimators':[10,50,100],'learning_rate':[0.01,0.1,1]}
ada_gscv = GridSearchCV(estimator=AdaBoostClassifier(random_state=123),
                        param_grid=params, cv=3, refit=True, n_jobs=-1)
ada_gscv.fit(bX_train, by_train)
print(f'ada best hyperparams      : {ada_gscv.best_params_}')
print(f'ada best mean cv accuracy : {ada_gscv.best_score_:.2f}')
```

✓ 54.5s

```
ada best hyperparams      : {'learning_rate': 0.01, 'n_estimators': 10}
ada best mean cv accuracy : 0.91
```

Model Performance:

- The best learning rate = 0.01
- The optimal n estimators = 10

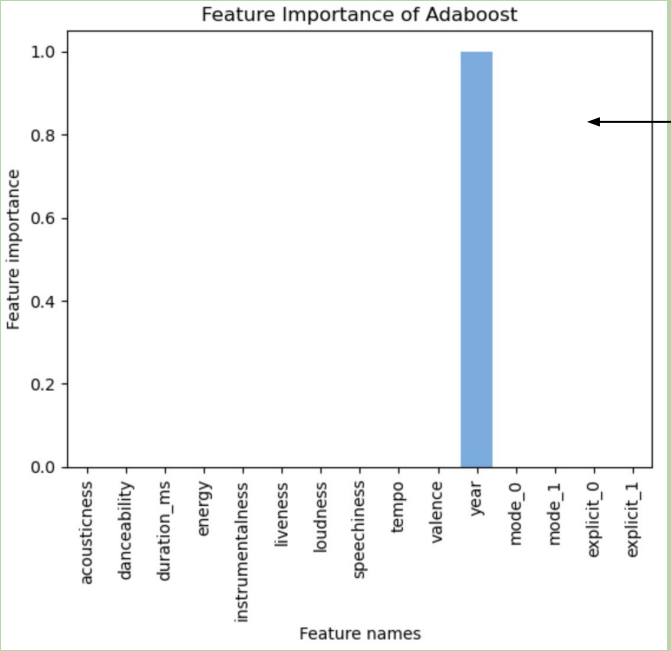
Best Model Performance:

- Training score: 0.9116
- Testing score: 0.9101

- Here we use the training set containing “year” feature to train the Adaboost Classifier
- We know that feature “year” will influence the model too much, so when considering about feature importance, we train two Adaboost models with/without feature “year” later

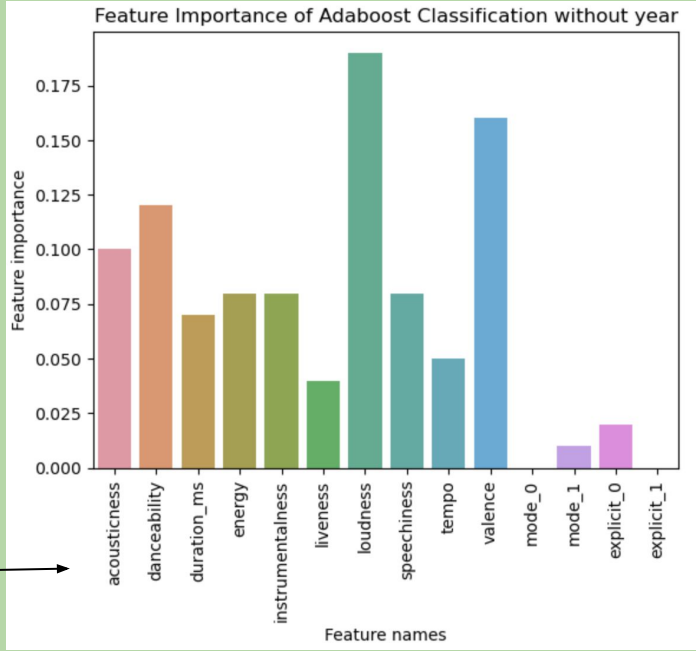
Machine Learning: Adaboost

Adaboost Classifier - Feature Importance



Top 5 Features:
('year', 1.0),
('acousticness', 0.0),
('danceability', 0.0),
('duration_ms', 0.0),
('energy', 0.0)

Top 5 Features:
('loudness', 0.19),
('valence', 0.16),
('danceability', 0.12),
('acousticness', 0.1),
('energy', 0.08)



Machine Learning: Adaboost

Adaboost Regressor

Hyperparameter Tuning and Best Model Selection

```python
# Grid Search
params = {'n_estimators':[10,50,100],'learning_rate':[0.01,0.1,1]}
ada_gscv = GridSearchCV(estimator=AdaBoostRegressor(random_state=123),
                        param_grid=params, cv=3, refit=True, n_jobs=-1)
ada_gscv.fit(X_train, y_train)
print(f'ada best hyperparams      : {ada_gscv.best_params_}')
print(f'ada best mean cv accuracy : {ada_gscv.best_score_:.2f}')
```

✓ 1m 20.6s

```
/Users/jingyi/opt/anaconda3/envs/MyCodingSpace/lib/python3.9/site-packages/joblib
  warnings.warn(
ada best hyperparams      : {'learning_rate': 0.01, 'n_estimators': 100}
ada best mean cv accuracy : 0.81
```

Model Performance:

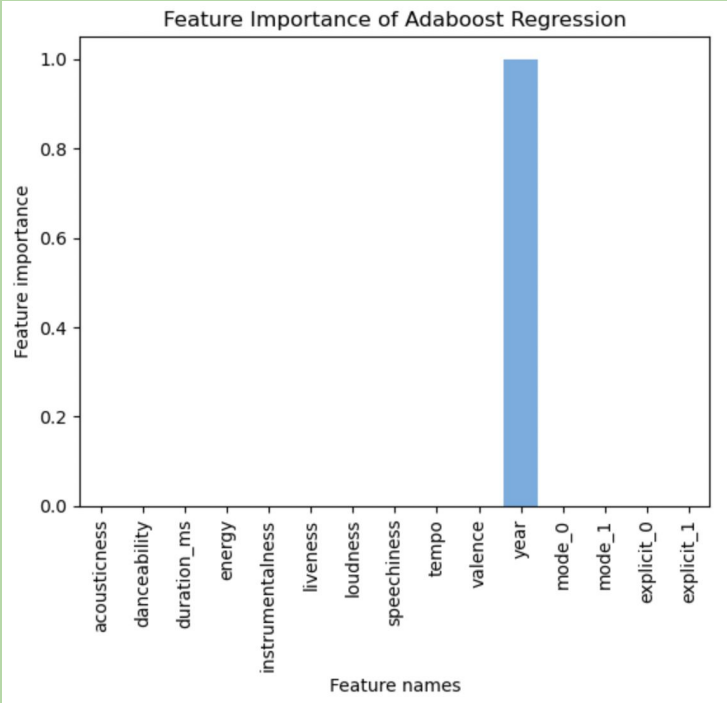
- The best learning rate = 0.01
- The optimal n estimators = 100

Best Model Performance:

- Training score: 0.8092
- MSE: 88.5075

Machine Learning: Adaboost

Adaboost Regressor - Feature Importance



Feature Importances:

('year', 0.999911475733514),
('instrumentalness',8.8524e-05),
('acousticness', 0.0),
('danceability', 0.0),
('duration_ms', 0.0),
('energy', 0.0),
('liveness', 0.0)

('loudness', 0.0),
('speechiness', 0.0),
('tempo', 0.0),
('valence', 0.0),
('mode_0', 0.0),
('mode_1', 0.0),
('explicit_0', 0.0),
('explicit_1', 0.0)

Insights:

- The releasing day of songs influence the popularity of songs significantly comparing with others attributes

Machine Learning: Gradient Boost

Gradient Boost Classifier

Hyperparameter Tuning and Best Model Selection

```python
# Grid Search - year
params = {'n_estimators':[10,50,100],'learning_rate':[0.01,0.1,1]}
gr_gscv = GridSearchCV(estimator=GradientBoostingClassifier(random_state=123),
                       param_grid=params, cv=3, refit=True, n_jobs=-1)
gr_gscv.fit(bX_train, by_train)
print(f'gr best hyperparams      : {gr_gscv.best_params_}')
print(f'gr best mean cv accuracy : {gr_gscv.best_score_:.2f}')
```

✓ 1m 19.7s

```
gr best hyperparams      : {'learning_rate': 0.1, 'n_estimators': 50}
gr best mean cv accuracy : 0.91
```

Model Performance:

- The best learning rate = 0.1
- The optimal n estimators = 50

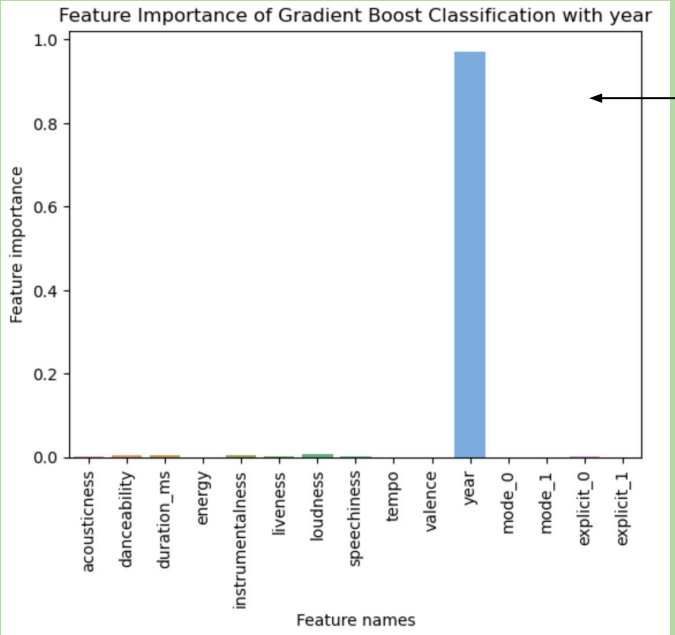
Best Model Performance:

- Training score: 0.9121
- Testing score: 0.9109

- Here we use the training set containing “year” feature to train the Gradient Boost Classifier
- We know that feature “year” will influence the model too much, so when considering about feature importance, we train two Gradient Boost models with/without feature “year” later

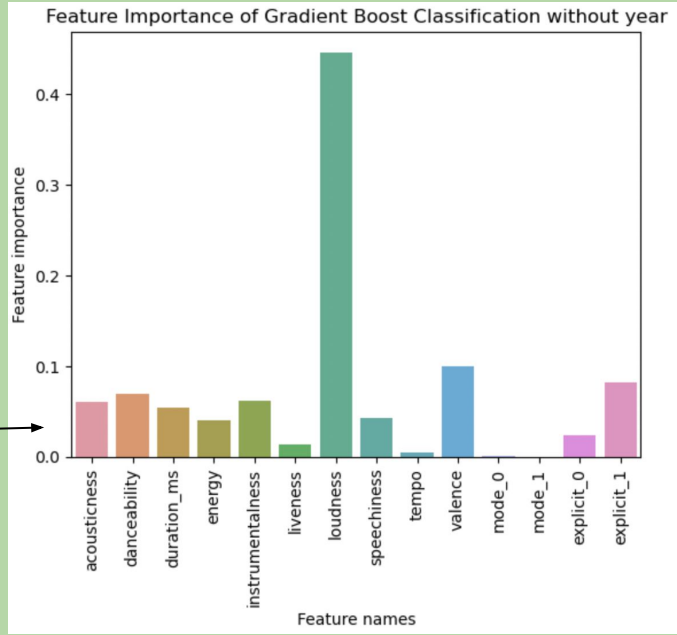
Machine Learning: Gradient Boost

Gradient Boost Classifier - Feature Importance



Top 5 Features:
('year', 0.975),
('loudness', 0.0080),
('duration_ms', 0.0055),
('instrumentalness', 0.0039),
('danceability', 0.0036)

Top 5 Features:
('loudness', 0.4465),
('valence', 0.0993),
('explicit_1', 0.0825),
('danceability', 0.0688),
('instrumentalness', 0.0612)



Machine Learning: Gradient Boost

Gradient Boost Regressor

Hyperparameter Tuning and Best Model Selection

```python
# Grid Search
params = {'n_estimators':[10,50,100],'learning_rate':[0.01,0.1,1]}
gr_gscv = GridSearchCV(estimator=GradientBoostingRegressor(random_state=123),
                       param_grid=params, cv=3, refit=True, n_jobs=-1)
gr_gscv.fit(X_train, y_train)
print(f'gr best hyperparams      : {gr_gscv.best_params_}')
print(f'gr best mean cv accuracy : {gr_gscv.best_score_:.2f}')
```

✓ 1m 31.4s

```
gr best hyperparams      : {'learning_rate': 0.1, 'n_estimators': 100}
gr best mean cv accuracy : 0.82
```

Model Performance:

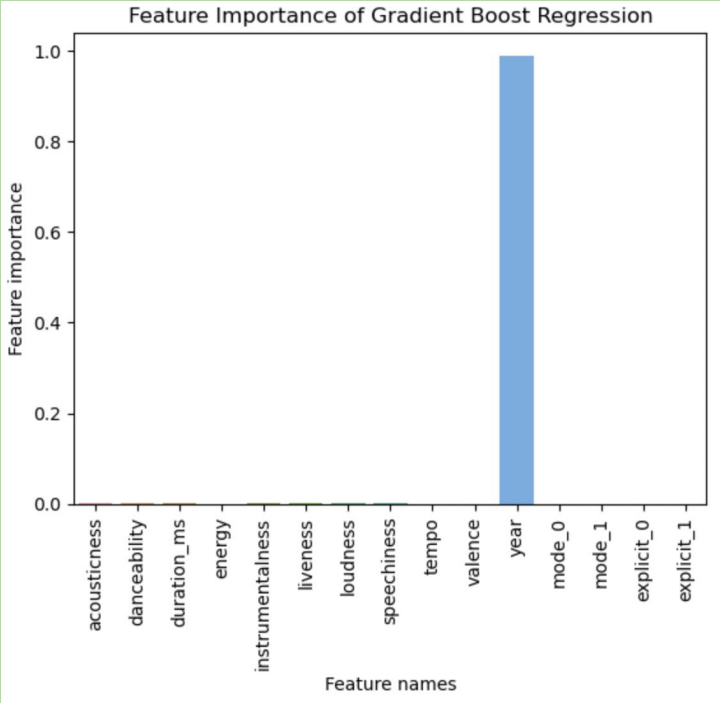
- The best learning rate = 0.1
- The optimal n estimators = 100

Best Model Performance:

- Training score: 0.8254
- MSE: 81.6608

Machine Learning: Gradient Boost

Gradient Boost Regressor - Feature Importance



Feature Importances:

('year', 0.9891290250186039),
('duration_ms', 0.001983907069081938),
('loudness', 0.0018745940843747676),
('acousticness', 0.0013931947691998496),
('instrumentalness', 0.0013718304785855248),

Top 5 Features

('speechiness', 0.0012971578025173917),
('liveness', 0.0011978459273449314),
('danceability', 0.0010141338359043433),
('energy', 0.0004281651086814007),
('explicit_1', 0.00010188195441835634),
('valence', 7.239236318631966e-05),
('explicit_0', 4.9889088651247526e-05),
('tempo', 4.977244604196847e-05),
('mode_0', 2.2183807538233905e-05),
('mode_1', 1.4026245869601496e-05)

Machine Learning: Xgboost

Gradient Boost Regressor

Hyperparameter Tuning and Best Model Selection

```
# Impute missing values
imputer = SimpleImputer(strategy='mean')
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Fit the model
xgb = XGBRegressor(max_depth=3, learning_rate=0.1, n_estimators=100)
xgb.fit(X_train_imputed, y_train)

# Make predictions on the test set
y_pred = xgb.predict(X_test_imputed)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)

print(mse)
```

```
81.72502065764989
```

Model Performance:

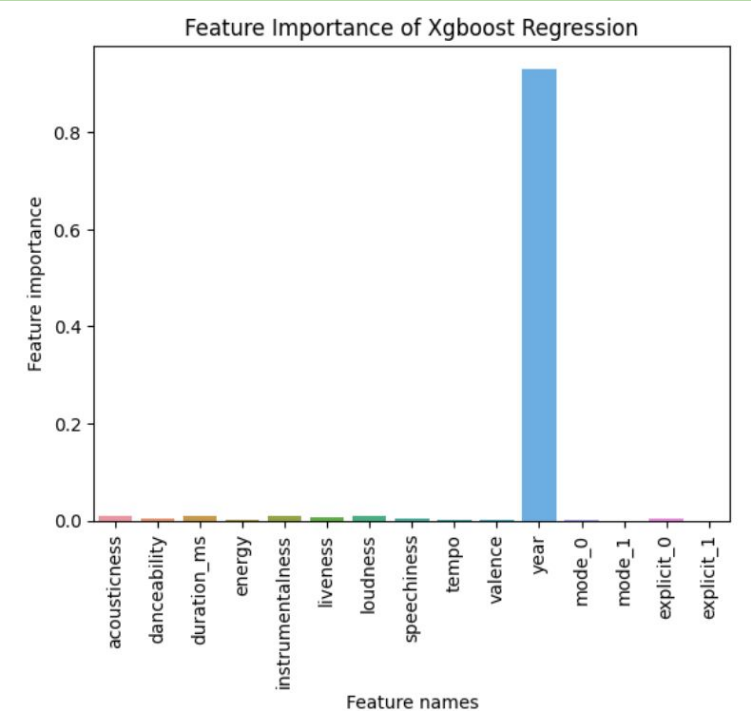
- The best learning rate = 0.1
- Max_depth = 3
- The optimal n estimators = 100

Best Model Performance:

- Training score: 0.83
- MSE: 81.7250

Machine Learning: XgBoost

XgBoost Regressor - Feature Importance



Feature Importances:

('year', 0.93086195),
('loudness', 0.010514024),
('acousticness', 0.009820845),
('instrumentalness', 0.009517046),
('duration_ms', 0.009120449),
('liveness', 0.0070600067),
('danceability', 0.005774328),
('explicit_0', 0.0051405346),
('speechiness', 0.0047789575),
('energy', 0.0031560701),
('valence', 0.0016335753),
('mode_0', 0.0016307181),
('tempo', 0.000991506),
('mode_1', 0.0),
('explicit_1', 0.0)

Conclusion: Model Comparison

	Linear Regression	Decision Tree	Random Forest	SVM	AdaBoost	Gradient Boost	XgBoost
Classification (Accuracy Score)	-	0.87	0.97	0.86	0.91	0.91	-
Regression (MSE)	100.69	82.70	82.00	103.80	88.51	81.66	81.73

- Random Forest has the best performance in Classification
- Gradient Boost has the best performance in Regression
- Random Forest is an ensemble model using bagging method, Gradient is also an ensemble model using boosting method, which both help to lower the variance and lower the probability of overfitting of the model

Feature Selection

- **Feature Importance from Model**

	Linear Regression	Decision Tree	Random Forest	SVM	AdaBoost	Gradient Boost	XgBoost
Classification	-	Year Loudness Duration	Year Loudness Speechiness	Year Danceability Acousticness	Year Acousticness Danceability	Year Loudness Duration	-
Regression	Year Acousticness Speechiness	Year Loudness Explicit	Year Acousticness Energy	Year Acousticness Speechiness	Year Instrumentalness Acousticness	Year Duration Loudness	Year Loudness Acousticness

- **Univariate Tests**
    - SelectKBest: ['acousticness', 'energy', 'year']

- We find that year is the most influential feature in all the machine learning models we applied
- Acousticness, Loudness, and Danceability are important feature we should pay attention to when predicting the popularity of a new song

NLP: Text Features: ‘artists’, ‘name’

- Preprocessing

```
df['artists']=df['artists'].apply(
    lambda x: x.replace("[", "").replace("]", "").replace("'", "").replace(",", "").lower()
    )
```

```
def lm_pre(name):
  a=re.sub('[^a-zA-Z]', ' ', name).lower()
  a1=a.split()
  lm = WordNetLemmatizer()

  a2=[lm.lemmatize(word) for word in a1 if not word in stopwords.words('english')]
  a2=[word for word in a2 if word not in ['remaster','remastered','remasterizado',
                                          'feat','remix','mix','version']]
  a3 = ' '.join(a2)
  return a3
```

```
df['name']=df['name'].apply(lm_pre)
```

NLP: Word Cloud

Word Cloud of song name

blue
love
live
song
girl
minor
op
major
time
single
instrumental
man
life
night
world
heart
take
el
flat major
stereo
baby
day
one
la
mono
dream
come
home
little
back
way
right
need
rain
want
flat
big
good
woman
black
never
cry
tonight
mind
know
dance
rock
gone
boy
tu
pt
long
b flat
minor bwv
symphony c
alternate take
piano sonata
non troppo
new
eye
let
got
se
make
go
lady
wan
na
fire
mi
de
gon na
c minor
ii andante
kiss
ride
change
soul
bad
river
heaven
young
away
hold
around
teddy wilson
violin concerto
op allegro
ist bei
sharp minor
forever
train
god
radio edit
goodbye
two
town
sky
da ist
another
without
got ta
summer
g major
glich kapitel
c major
g minor
street
water
head
lover
dead
better
angel
star
feel
hey
light
bei un
pr lude
op iii
op ii
new york
glich
nicht
soundtrack
radio
nothing
beautiful
party act scene
old
child
run
walk
stop
theme
people
tell
wilson orchestra
act ii
lonely
mr
piano concerto
city
da
edit
road
last
un nicht
voice
alone
morning
choru
high
see
king
rose
f major
extended
everything
remember van gelder
feeling
still
que
u
n
mama
believe
house
ki
hand
happy
fool
money
give
ke
digital
lo
say
i
mr
time
run
walk
act ii
wilson orchestra
lo
lonely

Content-based filtering recommendation system

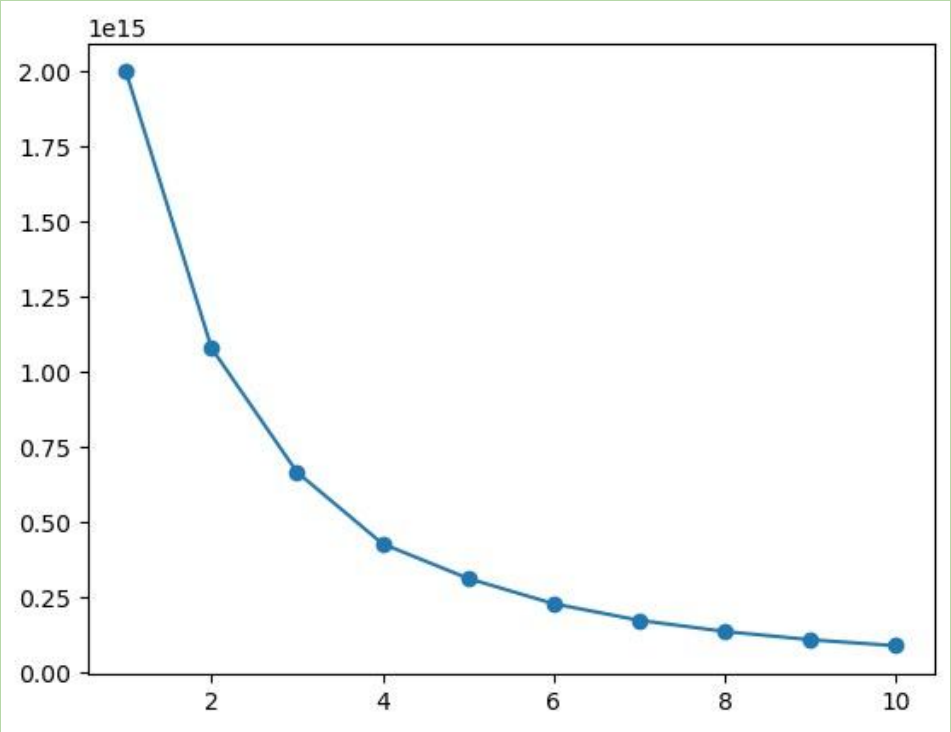
- TF-IDF transformation (column ‘artists’)
- PCA dimension reduction
- Similarity Matrix (cosine_similarity(df_scaled))
- **Recommendation Function**: input an index of a song -> get top ten recommendation songs

```
recommend_songs(1)
```

	acousticness	danceability	duration_ms	energy	explicit	instrumentalness	key	liveness	loudness	mode	...	856
1	0.18800	0.745	152453	0.656	0	0.003370	10	0.0941	-6.393	1	...	0.0
517	0.03550	0.648	132867	0.540	0	0.000004	9	0.0386	-12.563	1	...	0.0
578	0.36700	0.657	171773	0.371	0	0.045900	5	0.1570	-9.186	1	...	0.0
146	0.16400	0.727	194120	0.832	0	0.000000	2	0.1050	-5.851	1	...	0.0
292	0.17800	0.564	219760	0.523	0	0.030300	7	0.0624	-12.908	1	...	0.0
58	0.13900	0.731	171067	0.491	0	0.000002	5	0.0786	-13.951	1	...	0.0
513	0.96900	0.678	194320	0.175	0	0.001050	3	0.1770	-13.021	1	...	0.0
416	0.09870	0.516	200573	0.486	0	0.000001	3	0.1430	-11.810	0	...	0.0
903	0.04550	0.642	395573	0.806	0	0.314000	10	0.0498	-12.973	1	...	0.0

K-means Clustering

- Used elbow method;
- n_cluster=4;
- silhouette_score=0.553



K-means Clustering

Check on the mean of the X variables of each clusters:

cluster	counts	acousticness	danceability	duration_ms	energy	explicit	Instrumental-ness
0	60607	0.361428	0.542781	2.792041e+05	0.555768	0.099444	0.129972
1	8734	0.556893	0.427892	5.198750e+05	0.417558	0.023929	0.345756
2	100004	0.566147	0.546030	1.708483e+05	0.455100	0.081777	0.163641
3	564	0.736788	0.350738	1.365547e+06	0.308810	0.008865	0.448182

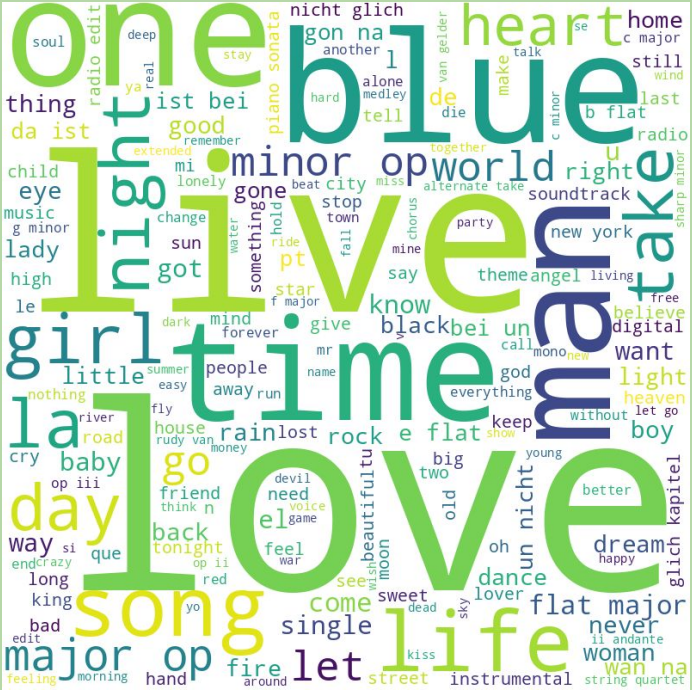
K-means Clustering

Check on the mean of the X variables of each clusters (cont.):

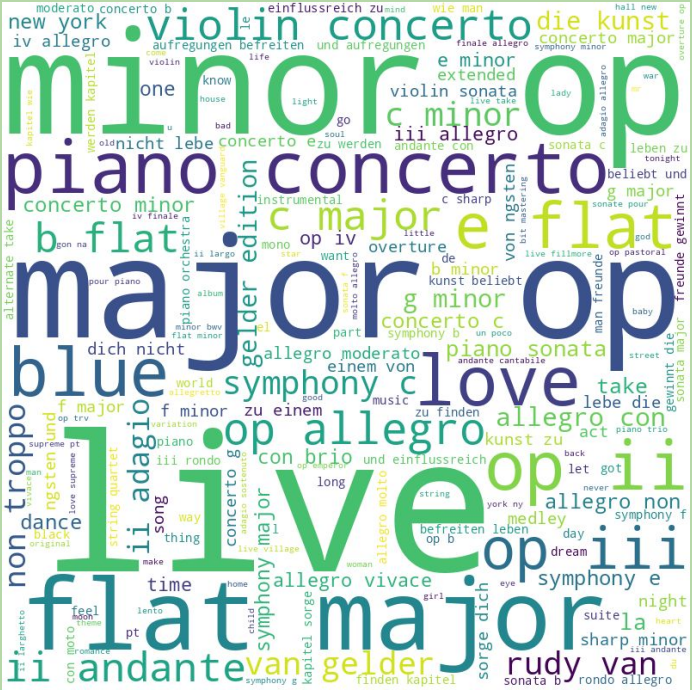
cluster	key	liveness	loudness	mode	speechiness	tempo	valence	year
0	5.204168	0.203747	-10.385591	0.677183	0.081153	117.845023	0.491251	1986.56102
1	5.148157	0.240370	-14.025472	0.652507	0.087403	112.191172	0.371329	1970.03298
2	5.203572	0.205133	-11.705472	0.732551	0.101996	116.896174	0.572541	1972.29161
3	5.078014	0.277517	-16.635665	0.693262	0.176325	103.412475	0.239232	1959.57447

K-means Clustering: Word Cloud

Word Cloud of Cluster 0



Word Cloud of Cluster 1



First the word cloud of each cluster are different in some aspect.

K-means Clustering: Word Cloud

Word Cloud of Cluster 2

love
instrumental
blue
girl
man
live
elde
take
heart
mono
time
single
baby
song
little
life
world
want
right
go
know
stereo
tu
boy
let
rain
gon na
wan na
dance
tonight
mind
rock
home
back
make
town
de la
teddy wilson
around
wilson orchestra
way
day
good
one
eye
le dream
hai
come
night
never
got
que
fire
lo
oh
woman
flat major
minor op
op variation
mil
u
n
cry
sweet
big
sun
say
long
pr lude
se
still
act ii
ke
ya
thing
ki
walk
radio edit
tell
need
friend
major op
old
angel
pt
na
see
happy
black
hand
alone
amor
light
goodbye
road
mr
look
people
summer
everything
place
lonely
give
act scene
call
le sacre
ta
il
mine
head
heaven
crazy
country
party
soul
believe
train
today
always
dead
gone
mama
lady
stay
voice
better
soundtrack
prelude op
beautiful
rite spring
sharp minor
feel
hey
star
high
young new
h
alternate take
mazurka op
moon
river
kiss
city
remember
new york
minor bwv
talk
un
king
nothing
choru
two
stop
ti
sleep
forever
run
bwv variation
nobody
orchestra take
house
something
goldberg variation
reprise
radio
think
last
fool
bad
variation bwv
corea ii
let go
du printemps
music
main
call
river
lover
rose
morning
e flat
child
si
pa
ka
mama
stay
lady
dead
always
voice
better
beautiful
rite spring
sacre du
lost
mein
lonely
place
everything
act scene
give
fire
lo
ki
ta
radio edit
walk
thing
never
night
come
hai
eye
long
say
sweet
big
heart
time
mono
u
cry
way
day
good
one
eye
le dream
woman
flat major
minor op
op variation
mil
n
sun
city
kiss
house
something
goldberg variation
reprise
radio
think
last
fool
bad
remember
new york
minor bwv
talk
un
king
nothing
choru
two
stop
ti
sleep
forever
run

K-means Clustering - Cluster 0&2

- Common: Music from these two clusters both have **high valence, explicit, tempo and danceability** values, meanwhile have **low instrumentalness** values;
- Difference: The year of produce of music from cluster 0 are latter than the ones from cluster 2;
- According to the word cloud from previous slide, we can conclude that cluster 0 and cluster 2 share the common features below:
  - Having the attribute of “live”, “feat”, and “blue”, music in these two clusters have a strong and instrumental rhythm, which are obvious in melody;
  - The scene of music are concentrate on “dream”, “holiday”, “romance”, giving a feeling of relax, happiness, cozy etc.

K-means Clustering - Cluster 1

- Music in cluster 1 are more likely released in 1970s;

  <u>Background information about 1970s:</u>

  "The Seventies was the decade of the super long song. While the 50s and 60s was spent honing the short, tight, radio friendly pop song, the 1970s saw bands exploring ***concept albums, concept tracks,*** *more free form styles and longer* ***storytelling*** *and in the process creating some very long songs indeed."*

- Music in cluster 1 tend to have a longer duration and a larger amount of words and lyrics;
- The total number of songs in cluster 1 is less than 1000.
- We conclude that the music of cluster 1 are mostly long storytelling music, or mix of music collections.

K-means Clustering - Cluster 3

- There are not so much music in cluster 3;
- The music of cluster 3 are mostly order music with high instrumentalness that are gentle and light music;
- We conclude that most of the music in cluster 3 is pure music, such as Bach's piano pieces.

Discussion

- The genre and artists' popularity of the songs are missing in the data set.

- For factor Mode, there should be 8 modes in total according to our research, but it's included as a dummy variable in the data set.

- And sometimes, whether the song is popular also has a certain luck factor. Sometimes it does happen that some songs just becomes a big hit by chance.

- The dataset we used focus on the attributes of music.
  User-based dataset may perform better in music categorization and recommendation.

References

- Negi, E. (2020) Spotify-data 1921-2020, Kaggle. Available at: https://www.kaggle.com/datasets/ektanegi/spotifydata-19212020

- James, Gareth et al. An Introduction To Statistical Learning. 2nd ed., Springer, 2021.

- "Mode | Music", Encyclopedia Britannica, 2022, https://www.britannica.com/art/mode-music

- "Random Forests - Classification Description". Stat. Berkeley.Edu, 2022, https:// www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm

- Venetia. “The Long and the Short of It.” Living in the 70s, 9 July 2021, joy.org.au/livinginthe70s/2021/07/10/very-long-and-very-short-songs/