SC1015: Intro to DSAI

Mini Project: Song Popularity Prediction

FDAA Group 7:
Rahul Shanker Ramkrishna , Fatima Nowshad , Goh Jie Rong,Sean

PRACTICAL MOTIVATION

01

“Music is a growing industry”

Music companies struggle with having their songs consistently chart the lists

02

Optimal features = popular song ?

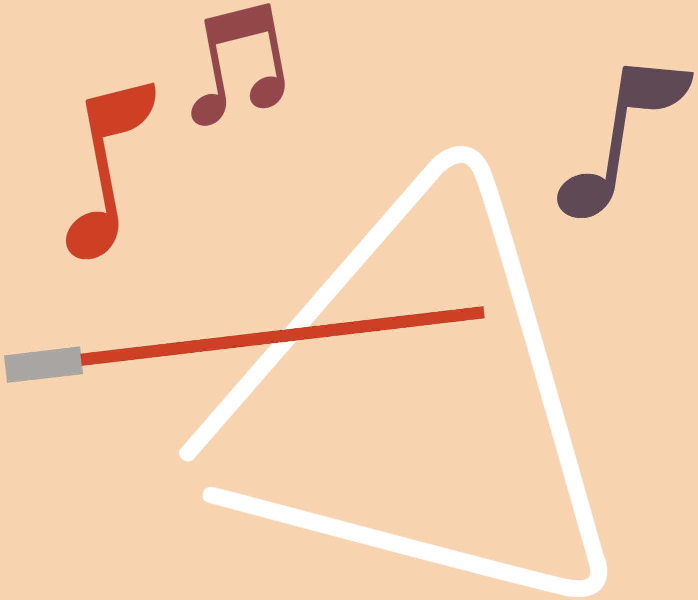
Song popularity influenced by current trends but song features may play a part as well

Hence...

Do different features of a song affect the song's popularity and how well it is received by the public and if so, can music producers use this to have their songs gain recognition?

<u>DATA SET USED :</u> "Spotify Tracks DB" by Zaheen Hamidani from Kaggle

https://www.kaggle.com/datasets/zaheenhamidani/ultimate-spotify-tracks-db#SpotifyFeatures.csv



EXPLORATORY DATA ANALYSIS

Exploring the song features :

Recognized 12 main features
that could be used to predict popularity

Out of the 12:

- 10 -> numerical
- 2 -> categorical

Exploring the 10 numerical figures ¶

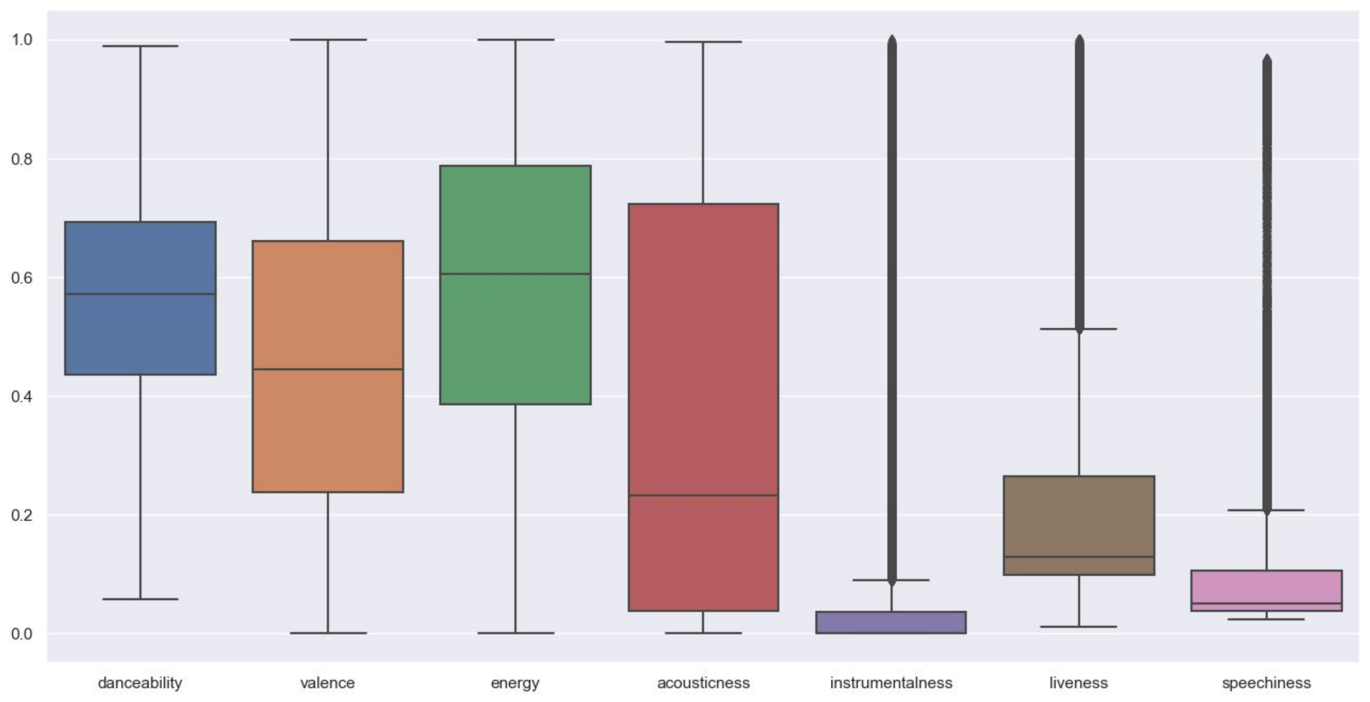
In [15]:
```
#tempo, loudness and duration_ms have much larger values in comparison to the other 7 and will be shown seperately
# Extract only the numeric data variables
numFeature = pd.DataFrame(spotdata[["danceability", "valence", "energy", "acousticness", "instrumentalness","liveness","speechine

# Summary Statistics for all Variables
numFeature.describe()
```

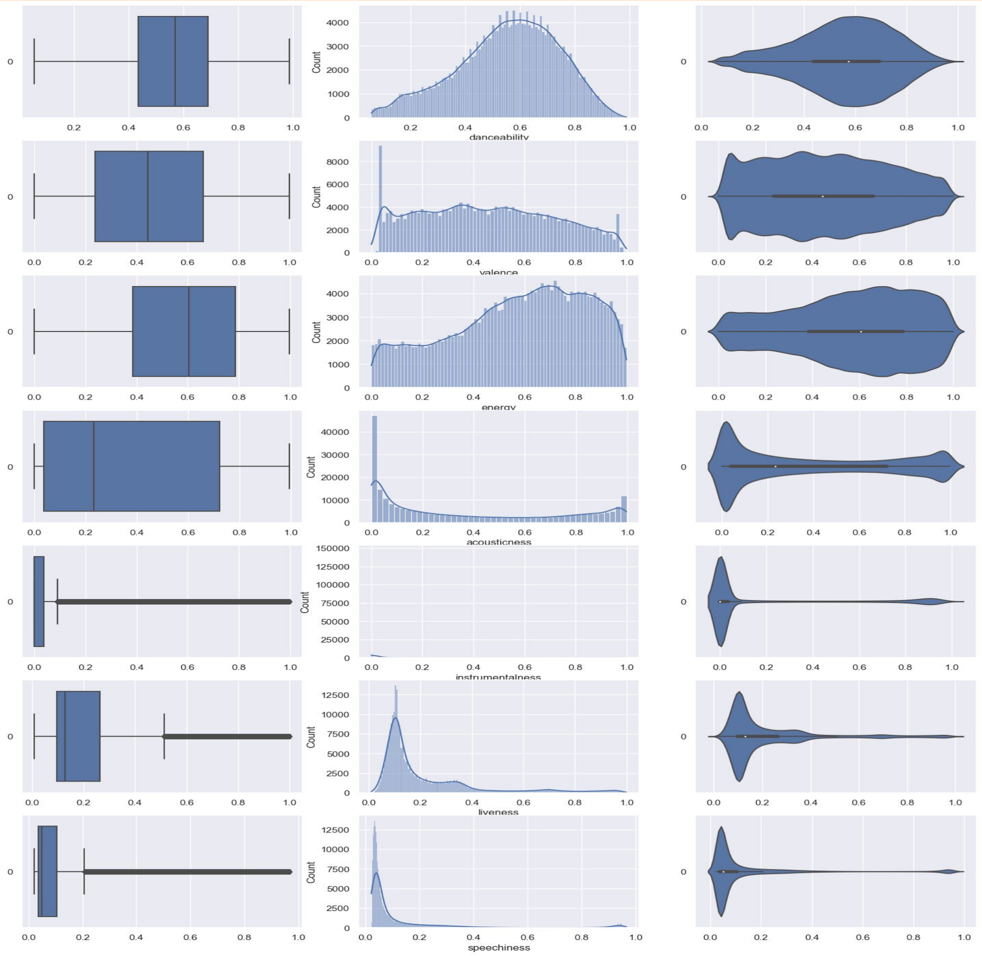
Out[15]:

	danceability	valence	energy	acousticness	instrumentalness	liveness	speechiness
count	232725.000000	232725.000000	232725.000000	232725.000000	232725.000000	232725.000000	232725.000000
mean	0.554364	0.454917	0.570958	0.368560	0.148301	0.215009	0.120765
std	0.185608	0.260065	0.263456	0.354768	0.302768	0.198273	0.185518
min	0.056900	0.000000	0.000020	0.000000	0.000000	0.009670	0.022200
25%	0.435000	0.237000	0.385000	0.037600	0.000000	0.097400	0.036700
50%	0.571000	0.444000	0.605000	0.232000	0.000044	0.128000	0.050100
75%	0.692000	0.660000	0.787000	0.722000	0.035800	0.264000	0.105000
max	0.989000	1.000000	0.999000	0.996000	0.999000	1.000000	0.967000

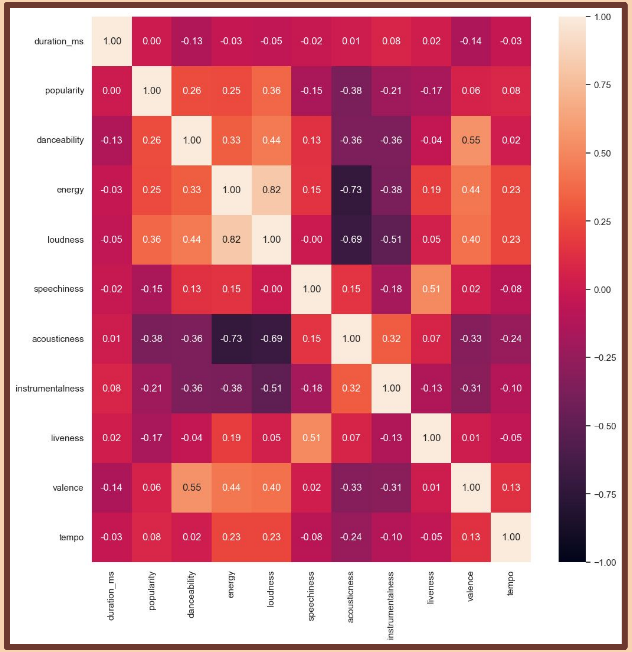
Exploring the numerical features

BOX PLOTS OF SEVEN NUMERICAL FEATURES :




Count
danceability
valence
energy
acousticness
instrumentalness
liveness
speechiness

Identify correlation between features and popularity :



Linear correlation of features with popularity not very high

Higher correlations with popularity: acousticness, loudness, danceability, energy

High correlations found between features themselves, ex: loudness & energy -> 0.82

DATA CLEANING :

```
pd.isnull(spotdata).sum()
```

```
genre               0
artist_name         0
track_name          1
track_id            0
popularity          0
acousticness        0
danceability        0
duration_ms         0
energy              0
instrumentalness    0
key                 0
liveness            0
loudness            0
mode                0
speechiness         0
tempo               0
time_signature      0
valence             0
dtype: int64
```

Only one null value identified under track_name which was then changed to NAN

Duplicate songs found in the dataset but belonging to different genres
-> hence not excluded

MACHINE LEARNING : Application

01

LINEAR REGRESSION

To check if the features with higher correlations among others would be able to predict popularity

Because popularity along with the recognised features are numerical, hence, linear regression

First...

```
Goodness of Fit of Model        Train Dataset
Explained Variance (R^2)        : 0.14602258191415918
Mean Squared Error (MSE)        : 282.0760878273076

Goodness of Fit of Model        Test Dataset
Explained Variance (R^2)        : 0.14348192434778317
Mean Squared Error (MSE)        : 284.846342398853
```

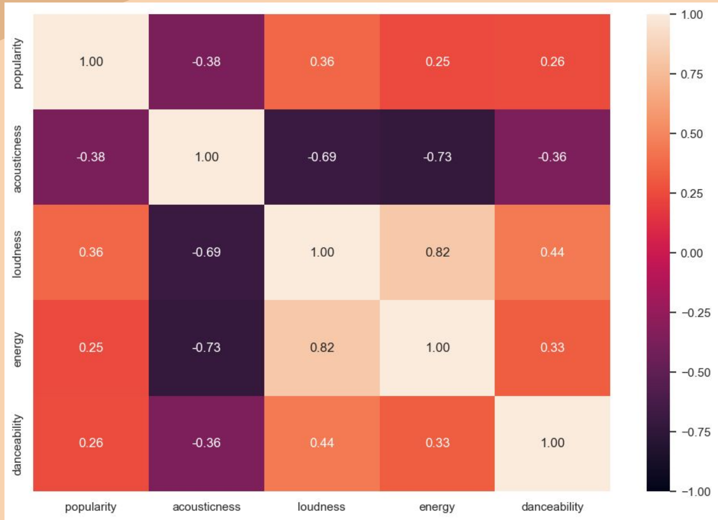
Acousticness & Popularity (negative correlation)

```
Goodness of Fit of Model        Train Dataset
Explained Variance (R^2)        : 0.1320445888829027
Mean Squared Error (MSE)        : 287.31871282292326

Goodness of Fit of Model        Test Dataset
Explained Variance (R^2)        : 0.13096993009309033
Mean Squared Error (MSE)        : 287.12973311031215
```

Loudness & Popularity (positive correlation)

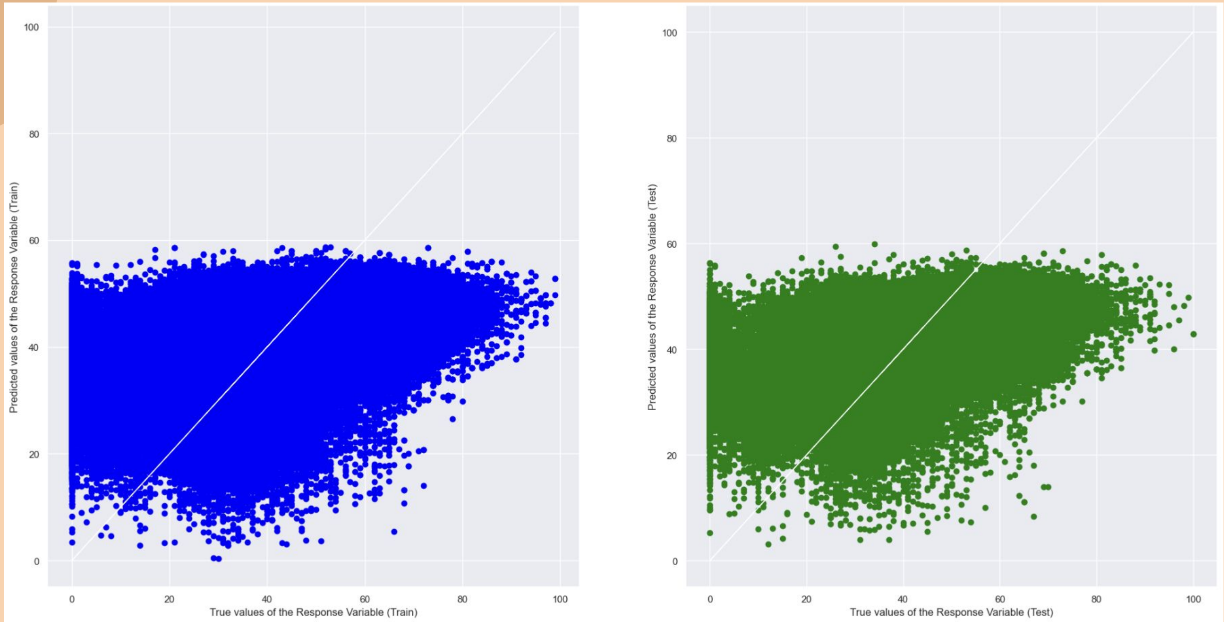
- **Tested uni-variate linear regression first with features that showed higher correlation**
- **First between acousticness & popularity -> Explained Variance not high**
- **Next, between loudness & popularity -> Again, Explained Variance not high**

Since …



- **Univariate linear regression with popularity didn't give what we expected**
- **Tried out multivariate regression**
- **Chose all features that seemed to have "higher" correlation**

```
# Extract Response and Predictors
y = pd.DataFrame(spotdata["popularity"])
X = pd.DataFrame(spotdata[["acousticness", "loudness", "energy", "danceability"]])
```




```
Goodness of Fit of Model        Train Dataset
Explained Variance (R^2)        : 0.19708861633691876
Mean Squared Error (MSE)        : 264.86717216830306

Goodness of Fit of Model        Test Dataset
Explained Variance (R^2)        : 0.1972626793553749
Mean Squared Error (MSE)        : 267.9839878859325
```

02

CLASSIFICATION TREE

Categorise into “popular” or not “popular” based on the features, rather than predicting numerical value of popularity

For this, we had to first convert popularity into a categorical variable

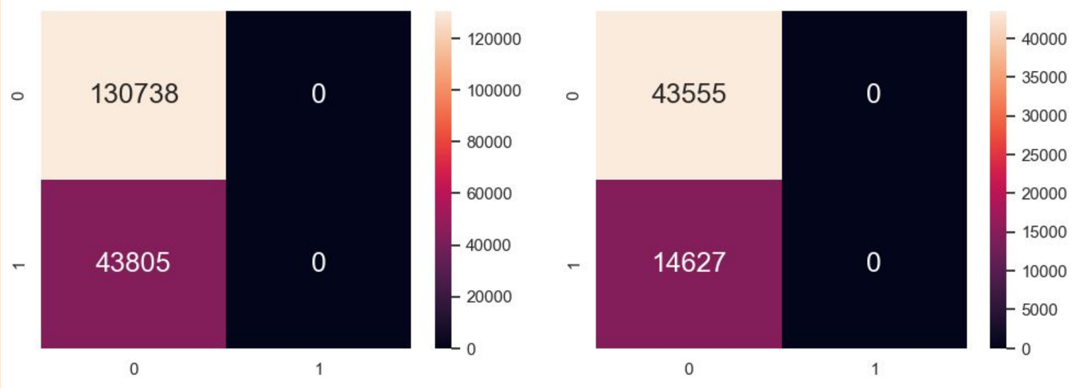
```
popDF = pd.DataFrame(spotdata["popularity"])
popDF.describe()
```

	popularity
count	232725.000000
mean	41.127502
std	18.189948
min	0.000000
25%	29.000000
50%	43.000000
75%	55.000000
max	100.000000

- **Splitting “popular” and “not popular” based on Third Quartile (75%)**
- **Above 55.00 -> “popular”**
  - **Indicated as “1”**
- **Below 55.00 -> “not popular”**
  - **Indicated as “0”**

```
#Splitting data set to above and below 55
featStreamDF.loc[featStreamDF['popularity'] < 55, 'popularity'] = 0
featStreamDF.loc[featStreamDF['popularity'] >= 55, 'popularity'] = 1
featStreamDF.loc[featStreamDF['popularity'] == 1]
```

- Similar, to regression, we first tried classifying popularity based on singular features
- First, loudness & acousticness



Loudness & Popularity

- Our TP and FP were very extremely low
- Hence, we needed to fix this

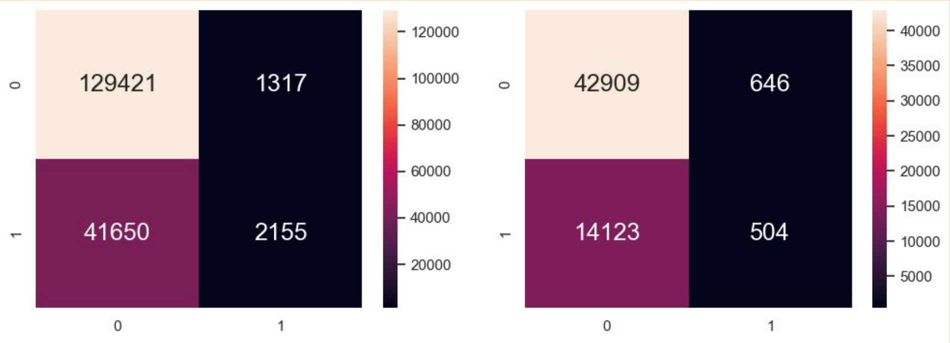
AdaBoostClassifier

- **Fit along with decision tree to improve misclassification**

- **Does this by weighing the incorrectly classified instances more heavily so that the subsequent weak learners focus more on the difficult cases**

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier

base_estimator=DecisionTreeClassifier(max_depth=5,criterion='gini', splitter='best', min_samples_split=2)
model = AdaBoostClassifier(base_estimator=base_estimator,n_estimators=100)
model.fit(X_train, y_train)
```

Better classification than univariate classification and higher accuracy

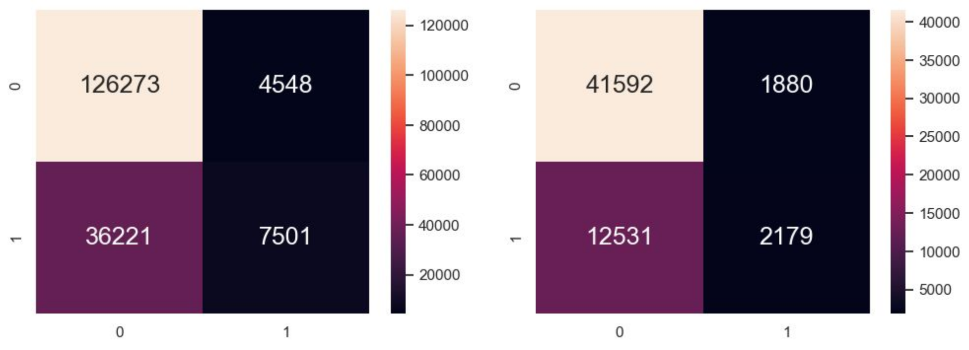



```
Goodness of Fit of Model        Train Dataset
Classification Accuracy         : 0.7538314340878752
True Positive rate              : 0.04919529734048625
False Positive rate             : 0.010073582279061941
False Negative rate             : 0.9508047026595138
F1 Score                        : 0.09116483702434588

Goodness of Fit of Model        Test Dataset
Classification Accuracy         : 0.746158605754357
True Positive rate              : 0.03445682641690025
False Positive rate             : 0.014831821834462175
False Negative rate             : 0.9655431735830997
F1 Score                        : 0.0638904734740445
```

Loudness & Popularity

With more features :

- **Multivariate classification with AdaBoost Classifier to see if better outcome ->**




```
Goodness of Fit of Model        Train Dataset
Classification Accuracy         : 0.7664243195086597
True Positive rate              : 0.17156122775719318
False Positive rate             : 0.03476506065539936
False Negative rate             : 0.8284387722428068
F1 Score                        : 0.268992845744204

Goodness of Fit of Model        Test Dataset
Classification Accuracy         : 0.7523117115259015
True Positive rate              : 0.14813052345343303
False Positive rate             : 0.04324622745675377
False Negative rate             : 0.851869476546567
F1 Score                        : 0.23219137940220577
```

- **Higher true positive rate than earlier**
- **Slightly better classification accuracy**

Secondary Question :

What genre is the most popular?

03

K Means clustering

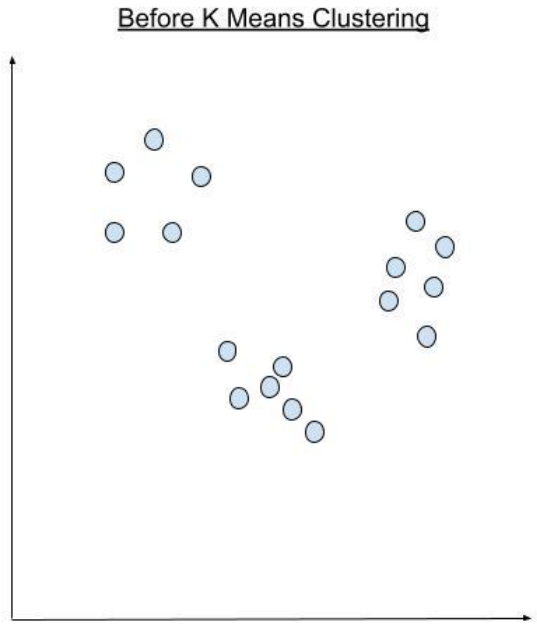
Identify clusters among the dataset

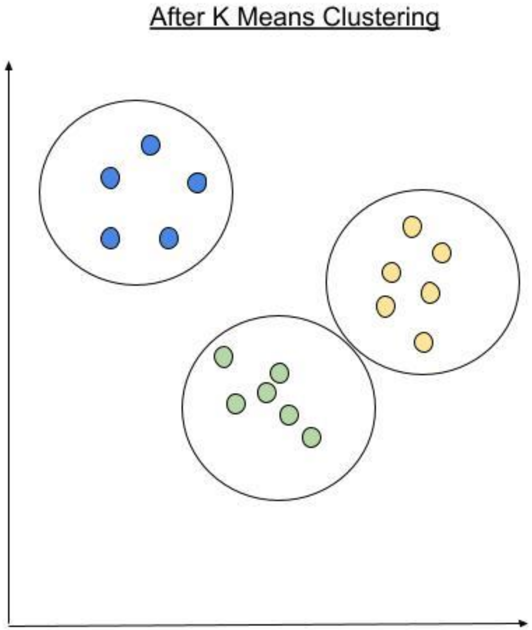
Genres have their own distinct features which differentiates themselves from each other

THEORY

How does it work?

- Predetermining a k-number of clusters
- Identify the center of the cluster
- Distance of points to each center
- Group points based on nearest distance to a cluster center
- Create new center by taking mean distance of data points from centre in a cluster


Before K Means Clustering


After K Means Clustering

Applying K Means

How accurate is the model?

```
features = ["acousticness", "danceability", "duration_ms", "energy", "instrumentalness", "liveness",
          "speechiness", "tempo", "valence"]
```

```
kmeans = KMeans(n_clusters = 27, random_state = 0, n_init='auto')
kmeans.fit(X_norm)
```

```
Accuracy: 0.04
```

```
array([[-7.22755189e-14],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00],
       [ 1.00000000e+00]])
```

Further information

Visualising cluster centers used in the model

04

Neural Network

- Library TensorFlow and Keras
- Building more robust ML model for prediction
- 5x Hidden Layer , Deep Neural Network
- Hyper Parameter Tuning(Optimizer,Activation,Loss)

```
[77]: #Step 1 create model
model = Sequential()

model.add(Dense(216,activation = 'relu',kernel_initializer = 'he_normal',input_shape = [10,]))
model.add(Dropout(0.20))

model.add(Dense(128, kernel_initializer='he_normal', activation = 'relu'))
model.add(Dropout(0.20))

model.add(Dense(64, kernel_initializer = 'he_normal', activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(32, kernel_initializer = 'he_normal', activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(16,kernel_initializer = 'he_normal', activation = 'relu'))
model.add(Dropout(0.2))

model.add(Dense(1,activation = "sigmoid"))
```

```
: # step 2. compile model

opt =   keras.optimizers.RMSprop(learning_rate = 0.0002)
model.compile(loss = 'binary_crossentropy'   , optimizer = opt, metrics = ["accuracy"])
```

- Sequential model used, total 7 layers
- Output is 1 node with “sigmoid” activation
- Regularization Techniques(Dropout,initialization)

Normalizing Inputs

```
# scaling both Xvalid and Xtrain and x-test

#scaler object
scaler = MinMaxScaler()

#scale the 3 X-sets --- will get a numpy array of scaled values
X_train = scaler.fit_transform(X_train)
X_valid = scaler.fit_transform(X_valid)
X_test = scaler.fit_transform(X_test)
```

```
0]: # Step 3 fit model

history = model.fit(X_train,y_train,epochs = 30 , validation_data = (X_valid,y_valid), batch_size = 128 )
```

```
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7730 - loss: 0.4633 - val_accuracy: 0.7730 - val_loss: 0.4611
Epoch 22/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7740 - loss: 0.4625 - val_accuracy: 0.7737 - val_loss: 0.4623
Epoch 23/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7703 - loss: 0.4658 - val_accuracy: 0.7719 - val_loss: 0.4602
Epoch 24/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7722 - loss: 0.4650 - val_accuracy: 0.7749 - val_loss: 0.4580
Epoch 25/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7727 - loss: 0.4651 - val_accuracy: 0.7733 - val_loss: 0.4608
Epoch 26/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7712 - loss: 0.4660 - val_accuracy: 0.7756 - val_loss: 0.4579
Epoch 27/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7735 - loss: 0.4612 - val_accuracy: 0.7757 - val_loss: 0.4581
Epoch 28/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7732 - loss: 0.4632 - val_accuracy: 0.7748 - val_loss: 0.4584
Epoch 29/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7745 - loss: 0.4617 - val_accuracy: 0.7746 - val_loss: 0.4599
Epoch 30/30
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.7723 - loss: 0.4640 - val_accuracy: 0.7754 - val_loss: 0.4576
```

Results

```
# how well it did
model.evaluate(X_test,y_test)
```

```
1455/1455 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.7703 - loss: 0.4728

[0.4725198447704315, 0.7706305980682373]
```

- True Accuracy always increased more than Validation Accuracy -> Indicating no Overfitting
- Actual model gave 77% accuracy

Conclusion

OUTCOME & EVALUATION

- In terms of predicting song popularity,

  -> decision tree classifier + AdaBoost Classifier = more positive results compared to linear regression

  -> also found that predicting popularity based on multiple features seemed to be better than uni-variate prediction.

- Data needs to be prepared and selected properly to serve our purpose effectively

- Each model has its own purpose and is more optimal in different scenarios

INSIGHTS

- Overall, we decided :
    - Popularity of a song may not necessarily be predictable by just features alone.
    - Certain features do seem to have higher affinity with popularity (loudness, energy, etc.)
    - All humans have different interests-> might be why we could not pinpoint any one genre or feature as being “more popular”

THANK YOU!!