# Predicting Song Popularity

## Using Machine Learning

Dixit Ayushman U2121836F
Summit Bajaj U2120089A
Lab Z136 Team 5

# The Problem: Independent artists struggle to predict Spotify song popularity, leading to various drawbacks

As an independent artist, it is **difficult to gauge the potential popularity** of a new song before releasing it on Spotify

Without data on how previous songs have performed, it's hard to make informed decisions about what to release next

Even with data, it is challenging to interpret it, especially for artists without a background in data science or analytics.

# The Problem: Independent artists struggle to predict Spotify song popularity, leading to various drawbacks

About 70% of indie artists generate less than $10,000 from their music annually, despite accounting for 41.4% of the music industry

# The Problem: Can a predictive model help these independent artists?

## Informed Decision Making

Artists can make more informed decisions about which songs to release and how to allocate their time and resources.

## Improve negotiation power

Better Negotiate deals with record labels, potentially leading to more opportunities and revenue.

## Targeted Promotion

Help artists tailor their marketing strategies and promotional efforts to maximize the song's impact upon release.

## Improve Artistic Growth

Enables artists to identify strengths and weaknesses, facilitating artistic growth and improvement in future releases.

# DS Problem: Develop a predictive model to predict the popularity of a song on Spotify before its release.

## The Dataset

- Spotify Dataset 1921-2020, 600k+ Tracks
- Features such as artist name, track name, release date, popularity, danceability, energy, and other audio features.

## Approach

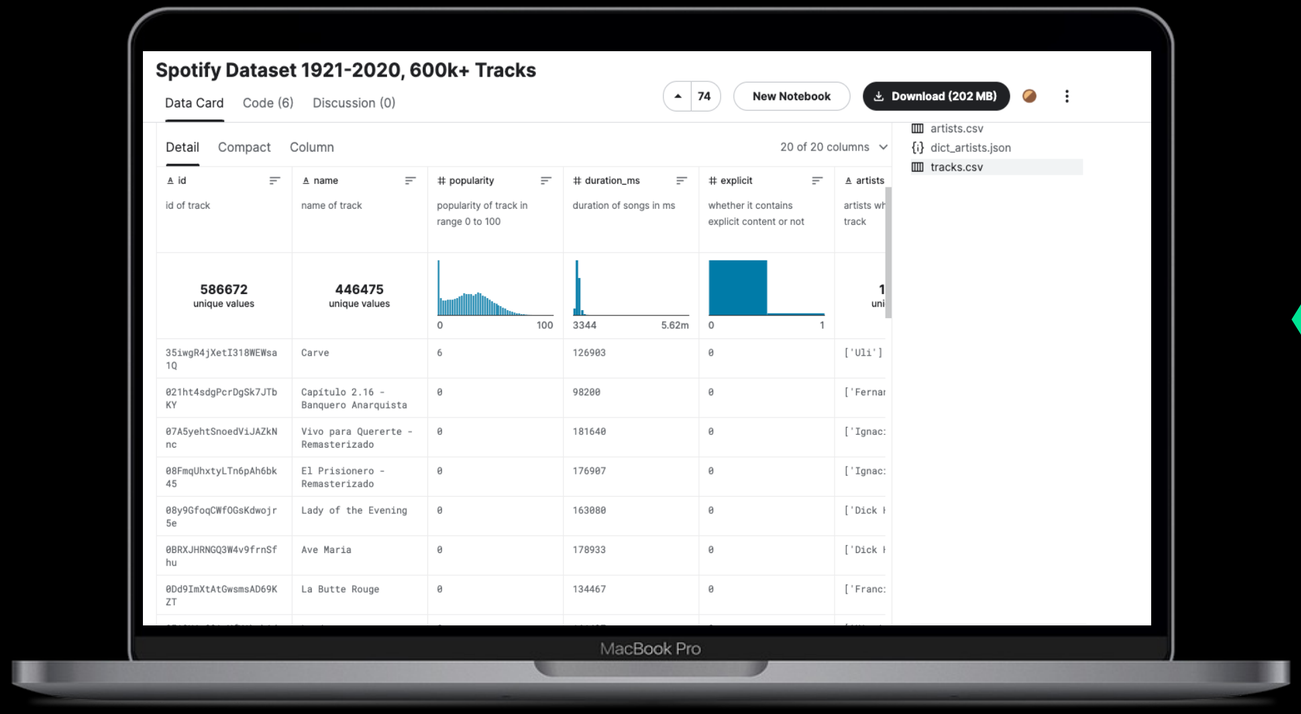
- Finding the best model in predicting popularity of songs on Spotify
- Suggestion of certain features to take note of when releasing songs

## Methodology

- Evaluating performance of 4 models in predicting popularity
- Exploring ways to improve performance of the models

# Data Collection: Used Kaggle to import the preliminary data to form our data set.

kaggle




The obtained dataset is **well-structured**, **comprehensive**, and **meaningful.** This makes it valuable for machine learning and data analysis

The dataset contains **20 columns** of statistics of **600K+ music tracks**. The data was obtained through Spotify.

# Data Cleaning: Before exploring the dataset, there was a need to ensure data was consistent and clean.

```
# Identifying missing data

missing_data = musicData.isnull().sum()
print("Number of missing values per column:\n", missing_data)

# Removing rows with null values
musicData = musicData.dropna()
musicData.info()
```

Removing Missing Data

```
# Removing duplicates if they exist
musicData = musicData.drop_duplicates()
```

Removing Duplicates

**Removing duplicates** and **missing values** is vital as they can influence model bias and negatively influence accuracy and reliability.

# Before exploring the dataset, there we understood the data collected.

| | |
|---|---|
| **id** | id of track |
| **name** | name of track |
| **popularity** | popularity of track in range 0 to 100 |
| **duration_ms** | duration of songs in ms |
| **explicit** | whether it contains explicit content or not |
| **artists** | artists who created the track |
| **id_artists** | id of artists who created the track |
| **release_date** | date of release |
| **danceability** | how danceable a song is in range 0 to 1 |
| **energy** | how energized a song is in range 0 to 1 |
| **key** | major note of track [0: C, 1: C#/Db, 2: D, ...] |
| **loudness** | How loud a song is in db |
| **mode** | The modality of track, 0 if minor and 1 if major |
| **speechiness** | The presence of spoken words in track in range 0 to 1 |
| **acousticness** | How acoustic a track is in range 0 to 1 |
| **instrumentalness** | The absence of vocal sounds in track in range 0 to 1 |
| **liveness** | The presence of audience in track in range 0 to 1 |
| **valence** | The positiveness of the track in range 0 to 1 |
| **tempo** | The overall tempo of track in BPM |
| **time_signature** | The time signature (4 in almost every track) |

**Not all the data is meaningful. However, we can extract and manipulate to get meaningful features.**

**Feature Engineering**

## Transforming artists into num_artists

Allow us to explore the relation between num_artists and popularity.

# Cleaning and Curation: Upon initial exploration, we removed irrelevant data.

**Removing irrelevant or unnecessary data is important to ensure that we do not overfit the model.**

## Dimension Reduction

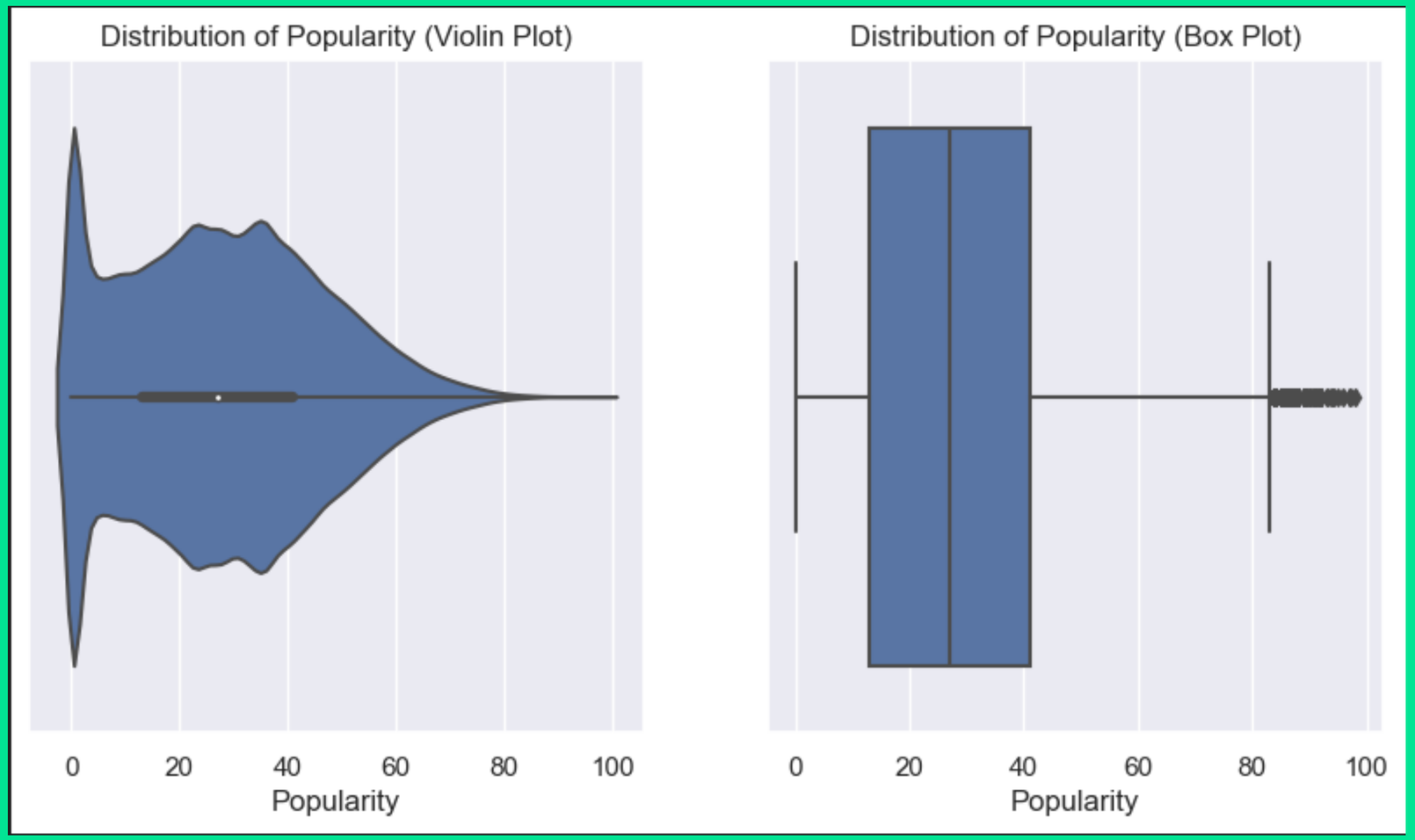
### Feature Selection

Performing exploratory data analysis, to determine relevant features.

### Feature Extraction

Utilizing domain knowledge to do feature engineering.

# Data Analysis: Popularity as a whole



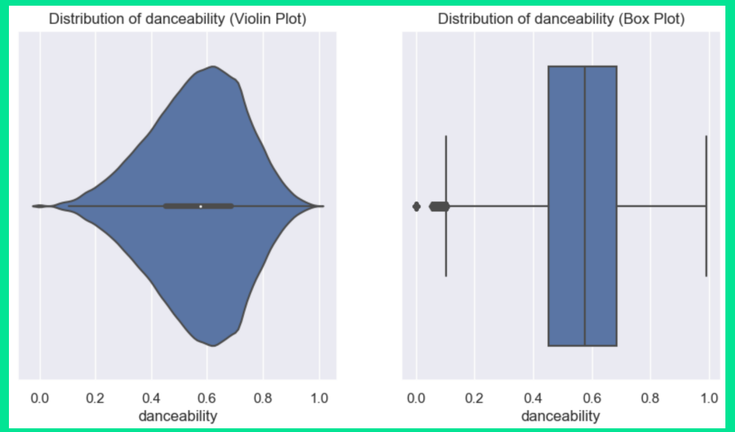

| | | | | |
|---|---|---|---|---|
| Mean | 27.6 | Q1 | 13.0 | SD: 18.4 |
| Median | 27.0 | Q3 | 41.0 | |

Distribution of popularity in the dataset is varied between 0-100 with a high concentration at the lower levels.

In the boxplot, we see multitude of outliers with high popularity. Moving forward, we will try to account for factors that contribute to such high popularity.

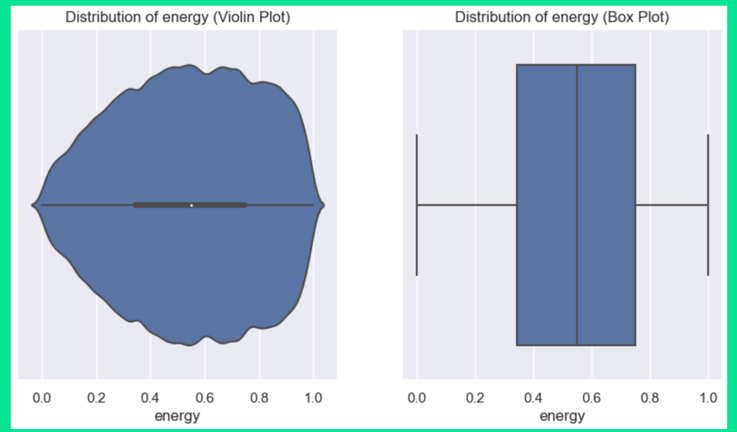
# Data Analysis: Relationship of features with popularity
# Non-Categorical Features

## Danceability




Correlation Coeff: 0.188

## Energy



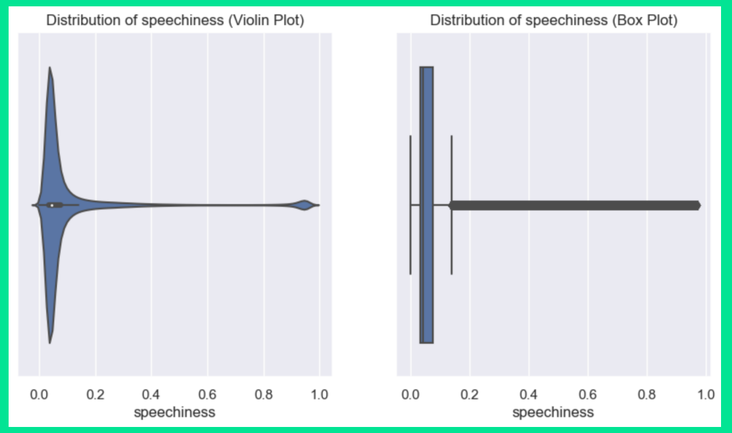

Correlation Coeff: 0.302

## Loudness


Correlation Coeff: 0.327

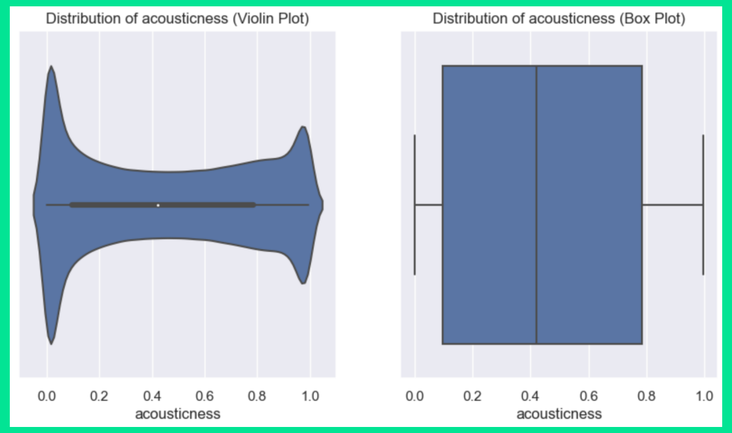
# Data Analysis: Relationship of features with popularity
# Non-Categorical Features

## Speechiness



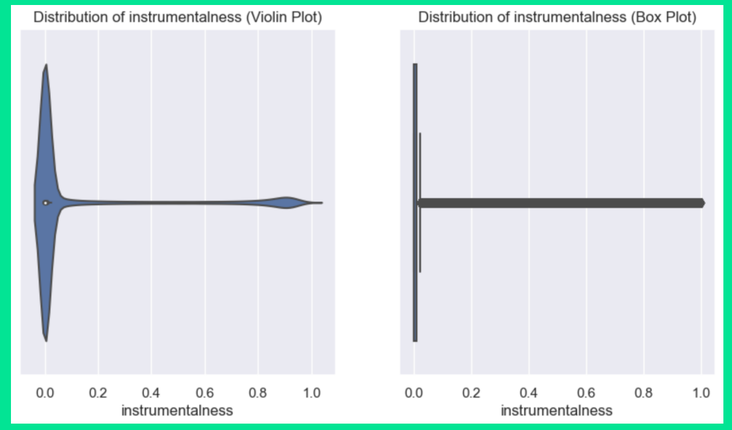

**Correlation Coeff: -0.0477**

## Acousticness




**Correlation Coeff: -0.371**

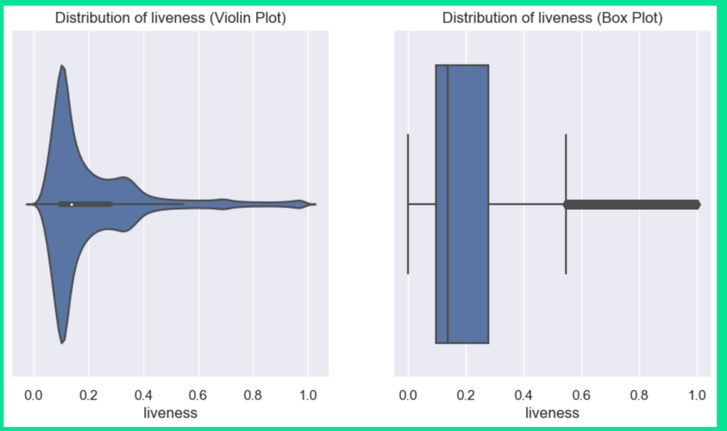
## Instrumentalness




**Correlation Coeff: -0.236**

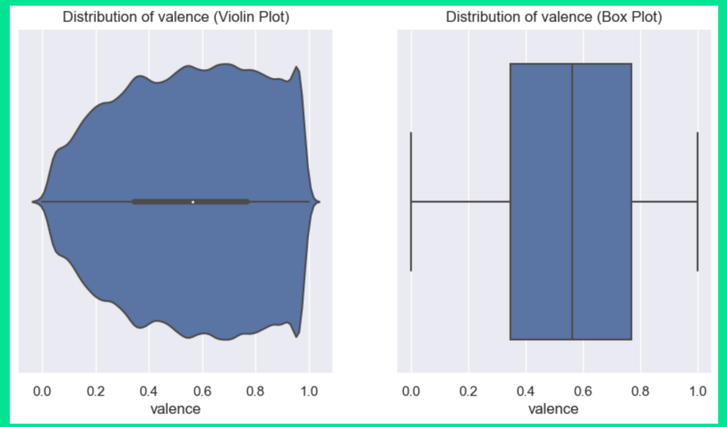
# Data Analysis: Relationship of features with popularity
# Non-Categorical Features

## Liveness



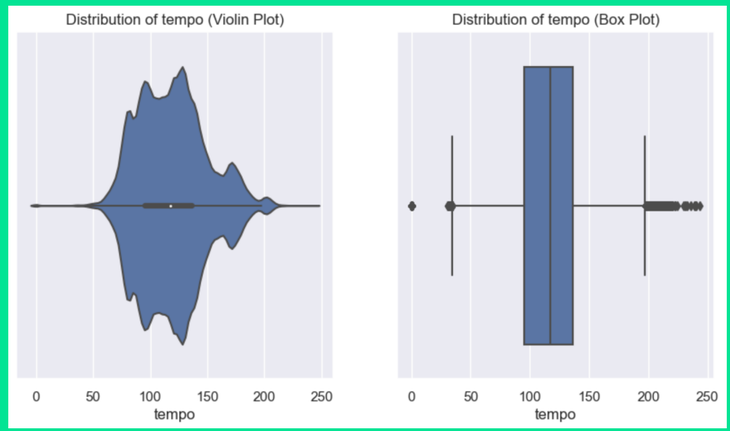

Correlation Coeff: -0.0493

## Valence




Correlation Coeff: 0.00535

## Tempo

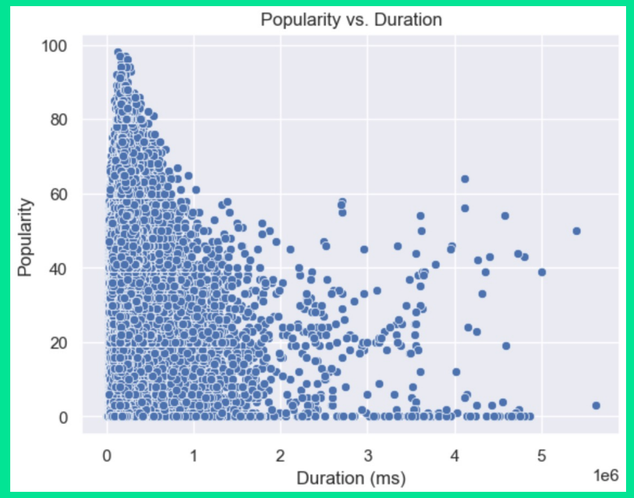



Correlation Coeff: 0.0712

Time Signature CC: 0.0871

# Data Analysis: Relationship of features with popularity
# Numeric ⟶ Cluster

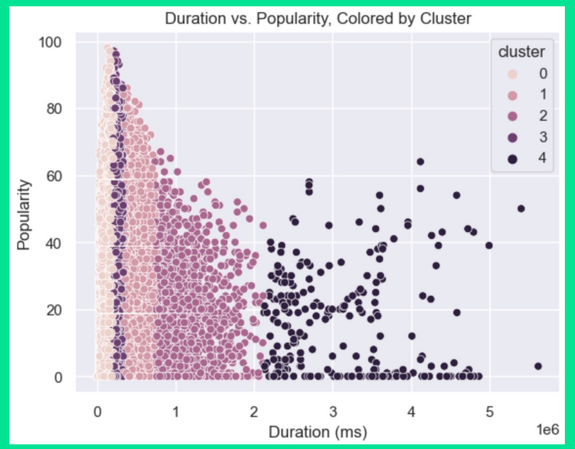
## Duration



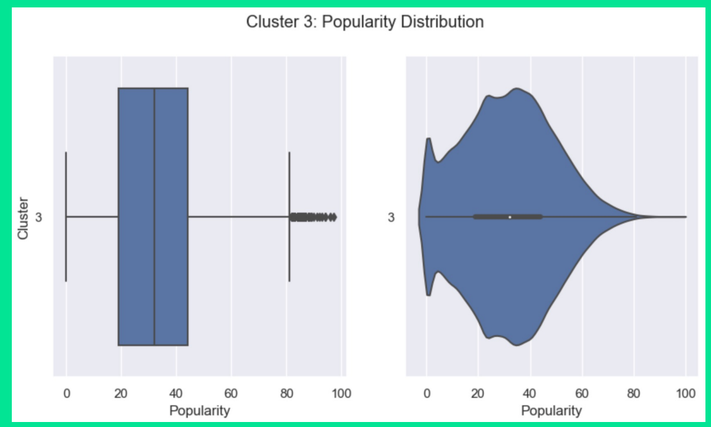

**Correlation Coeff: 0.0275**

We notice that songs with popularity over 60 always have a relatively low duration. Let us now account for this using a new analysis method - Clustering

Use K-means clustering to see popularity distribution in each cluster, observe this performance and then use Kruskal-Wallis test to prove that a cluster is better than the others



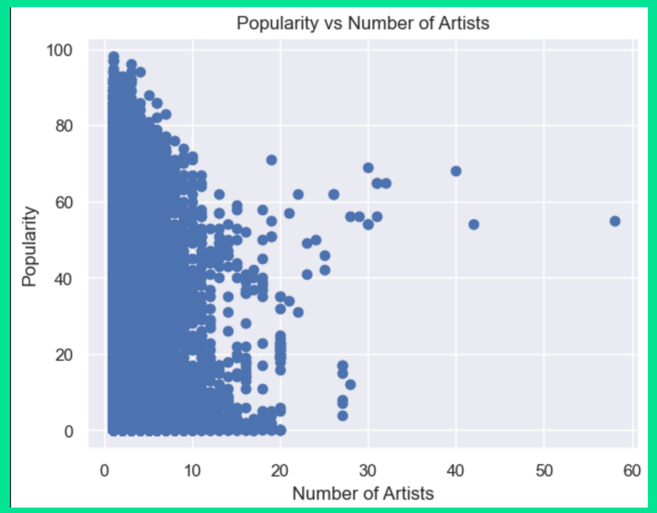

## Conclusion




Cluster 3 had the highest median with a significant difference. Additionally, its box plot was most to the right. As such, we can safely suggest aspiring top artists to have songs with duration in cluster 3. Cluster 3 = 205.7 seconds - 338.1 seconds

# Data Analysis: Relationship of features with popularity Numeric -> Cluster

## Num_Artists



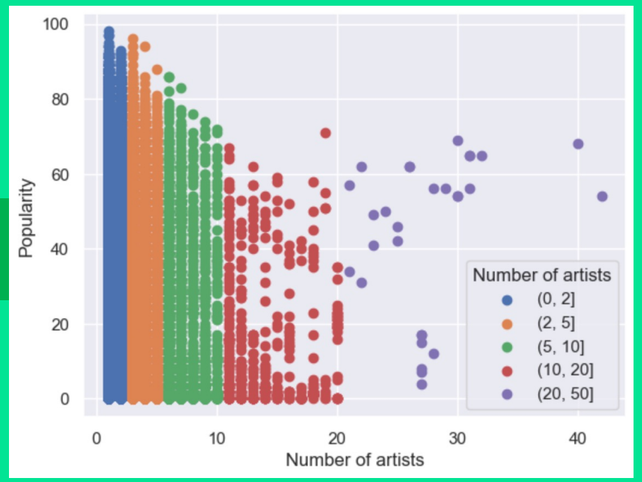

**Correlation Coeff: -0.072**

Majority of the songs in our dataset had only one artist. The range of the number of artists is limited, and the majority of the data falls within a narrow range. As a result, the relationship between the number of artists and popularity may not be fully captured.

**Key: Weak Relationship**

We group the songs into categories based on the number of artists and analyze the relationship between popularity and the number of artists within each category. This would provide a more detailed view of how the number of artists affects popularity, and help to avoid bias towards songs with only one artist.

## Conclusion

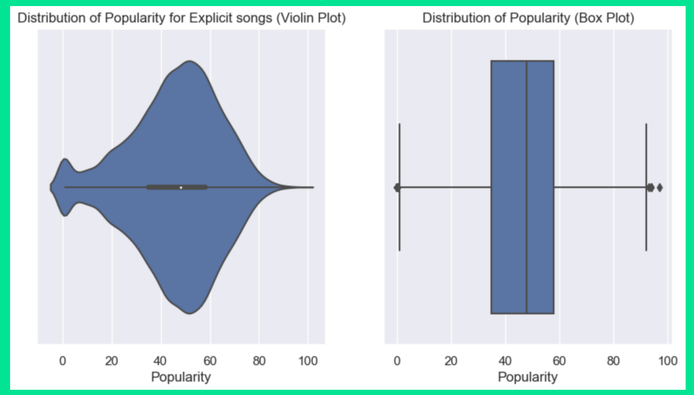



None of the above groups had a high correlation coefficient with popularity and we were safely able to conclude that the relationship between popularity and num_artists was a weak one

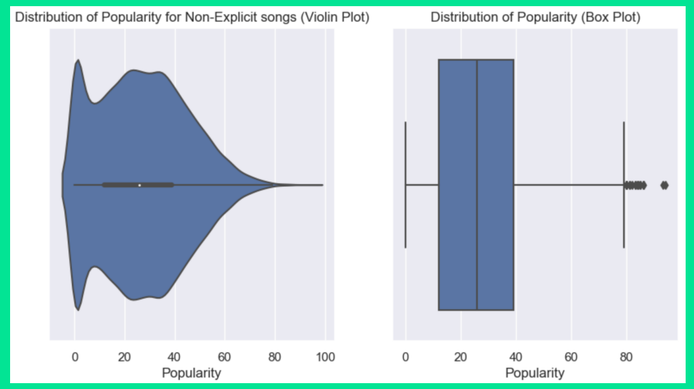
# Data Analysis: Relationship of features with popularity
# Categorical Features

## Explicit

### Explicit

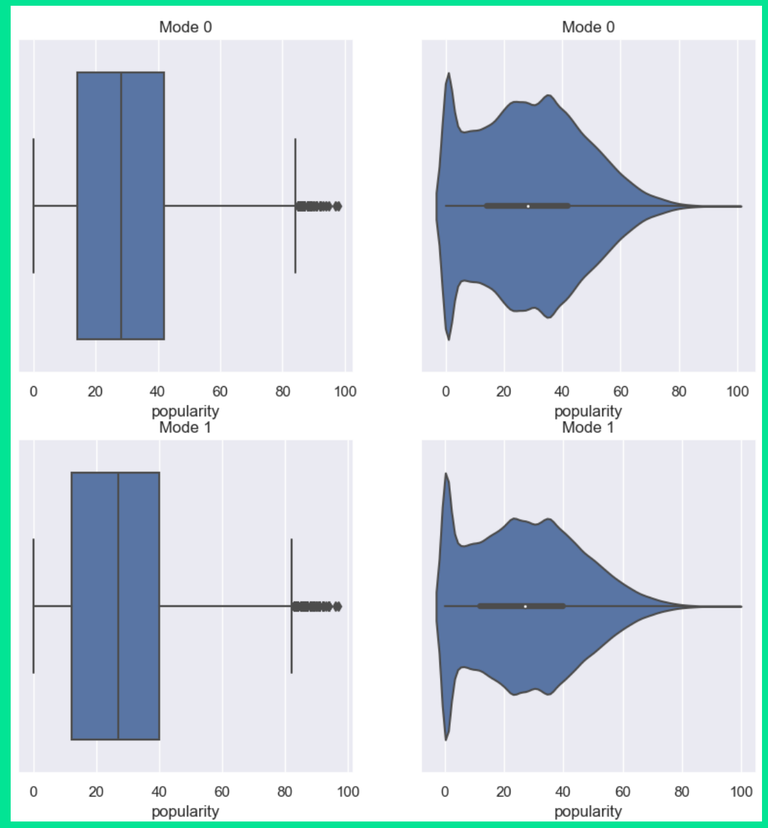



### Non- Explicit




- Hypothesized from the two plots that on average, explicit songs do better
- Proved using two-sample test that popularity does indeed differ when it comes to explicit vs non-explicit songs
- As such, explicit songs have higher popularity

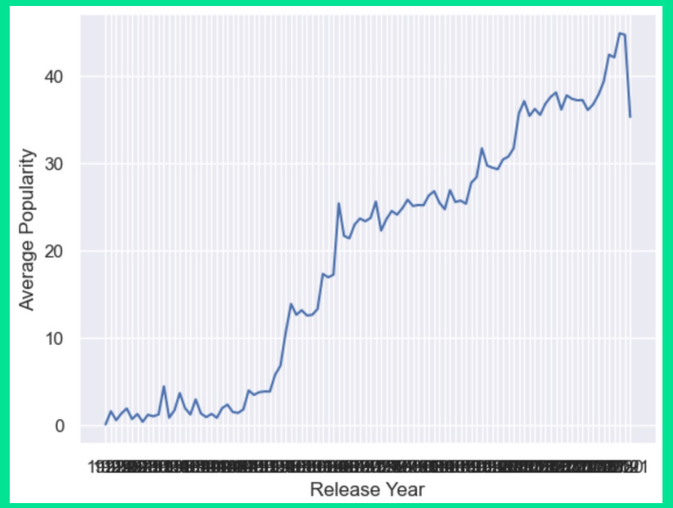
## Mode




- Plots look identical for both mode = 0 and mode = 1
- Concluded that popularity is not related to mode at all

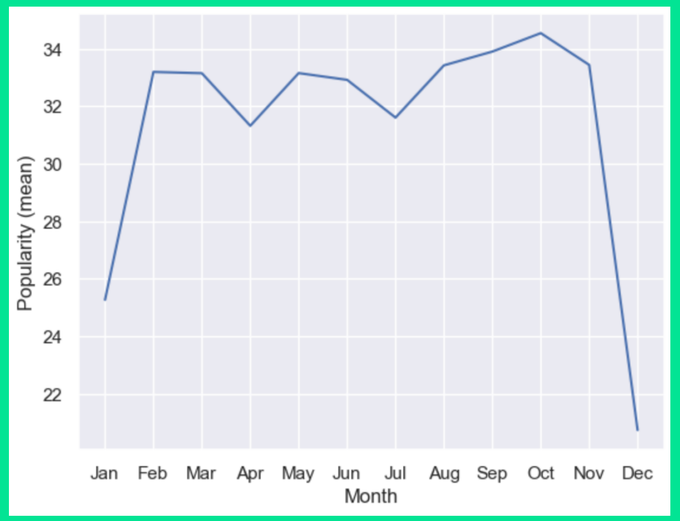
# Data Analysis: Relationship of features with popularity
# Release Date

## Year




- An increasing line chart suggests that there is a positive trend. However, it's important to note that correlation does not necessarily imply causation
- Songs released later are expected to do better when it comes to popularity on Spotify. This is due to factors such as artists advertising their newly-released songs, music events covering mainly new songs etc.

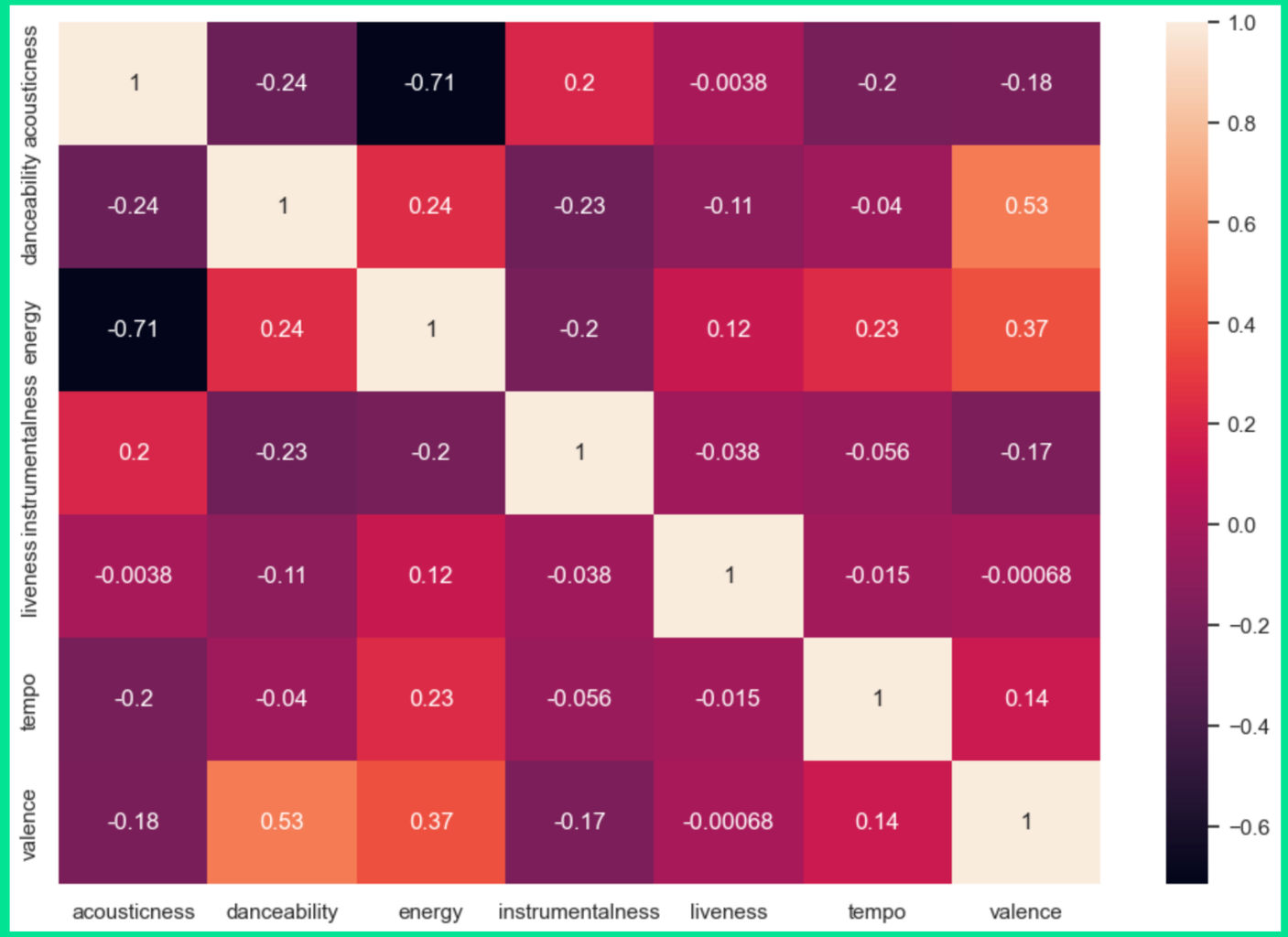
## Month




Possible reasons for low popularity for songs released in Dec and Jan:

1. Competition: December and January are often considered the holiday season, where people may be more focused on festivities rather than discovering new music
2. Release strategy: Some artists may strategically avoid releasing new music during December and January.

# Further Analysis: Relationship of features with one-another




A strong correlation coefficient of -0.71 between energy and acousticness suggests that if one is closely related to popularity, the other will be too in the same range. This is supported by our analysis earlier which shows that both have an absolute correlation coefficient of above 0.3

# Data Analysis: Best Indicators


## Non-Negligible Factors

CC >0.15

- Duration
- Explicit
- Release-Year
- Release-Month
- Danceability
- Energy
- Loudness
- Acousticness
- Instrumentalness

## Negligible Factors

- Num_Artists
- Key
- Mode
- Speechiness
- Liveness
- Valence
- Tempo
- Time-Signature

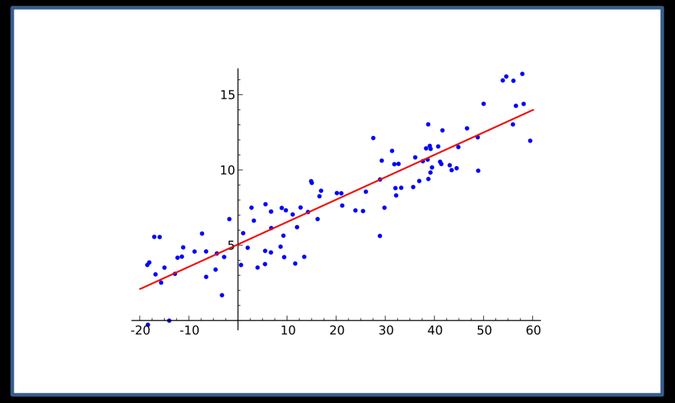
## Insights

- **Learnings from this EDA will be used in evaluating our models' performance following this**
- **While some features have been labelled as negligible, they may still play an important role when grouped with other features. This will be explored as well**

# Machine Learning: After, feature engineering, we can apply the ML models to our dataset.

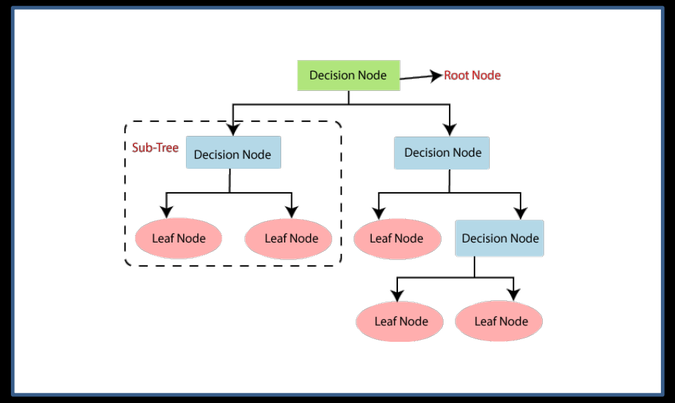
**Initially, we employed ML models we learnt.**

## Linear Regression



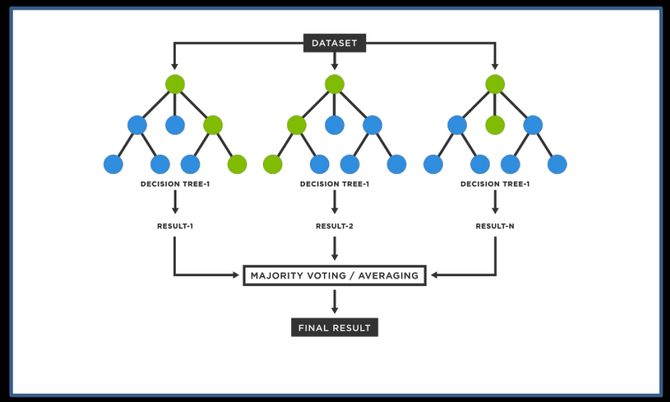

Predict a continuous dependent variable based on one or more independent variables.

## Decision Trees




The goal is to split the data into subsets that are as homogeneous as possible in terms of the target variable. This process continues recursively until the subsets are no longer split.

## Random Forests




Builds many decision trees and combining their predictions to make a final prediction. Each tree is trained on a subset of the data and a random subset of features, which helps to reduce overfitting.

# Machine Learning: Applying the data and analyzing the results.

## Linear Regression

- Performed analysis using all **features**, **essential features** and **top 6 features**.
- Observed that **correlation reduces** slightly as features are reduced.

## Decision Trees

- Performed analysis using all **features**, **essential features** and **top 6 features**.
- Observed that **correlation reduces** slightly as features are reduced.
- Model tends to overfit.

## Random Forests

- Performed analysis using all **features.**
- Used **top 6 features** for **hyperparameter tuning using cross validation.**
- **Up sampled data to reduce class imbalance**
- **Improved performance** after fine tuning.

## Performance

| Features | Train Set | Test Set |
|---|---|---|
| All | 0.38 | 0.38 |
| Essential | 0.37 | 0.37 |
| Top 6 | 0.36 | 0.36 |

| Features | Train Set | Test Set |
|---|---|---|
| All | 0.99 | 0.03 |
| Essential | 0.99 | 0.01 |
| Top 6 | 0.99 | -0.02 |

| Features | Train Set | Test Set |
|---|---|---|
| All | 0.49 | 0.48 |
| After Tuning | 0.59 | 0.51 |

# Machine Learning: Applying the data and analyzing the results.

## Linear Regression

- Performed analysis using all **features**, **essential features** and **top 6 features**.

## Decision Trees

- Performed analysis using all **features**, **essential features** and **top 6 features**.

## Random Forests

- Performed analysis using all **features**.
- Used **top 6 features** for

**Random Forest performs the best amongst the 3 models . However, the correlation is still relatively low. Hence, they are not good at predicting popularity.**

imbalance

- **Improved performance** after fine tuning.

## Performance

| Features | Train Set | Test Set |
|---|---|---|
| All | 0.38 | 0.38 |
| Essential | 0.37 | 0.37 |
| Top 6 | 0.36 | 0.36 |

| Features | Train Set | Test Set |
|---|---|---|
| All | 0.99 | 0.03 |
| Essential | 0.99 | 0.01 |
| Top 6 | 0.99 | -0.02 |

| Features | Train Set | Test Set |
|---|---|---|
| All | 0.49 | 0.48 |
| After Tuning | 0.59 | 0.51 |

## RMSE: $R^2$ values might not be the best predictor. We need to look at a more holistic predictor.

$R^2$ doesn't account for magnitude of the errors model . Hence $R^2 \neq$ Accuracy.

RMSE measures the average magnitude of the errors made by the model. It considers the magnitude of the residuals and is therefore a better indicator of the absolute accuracy

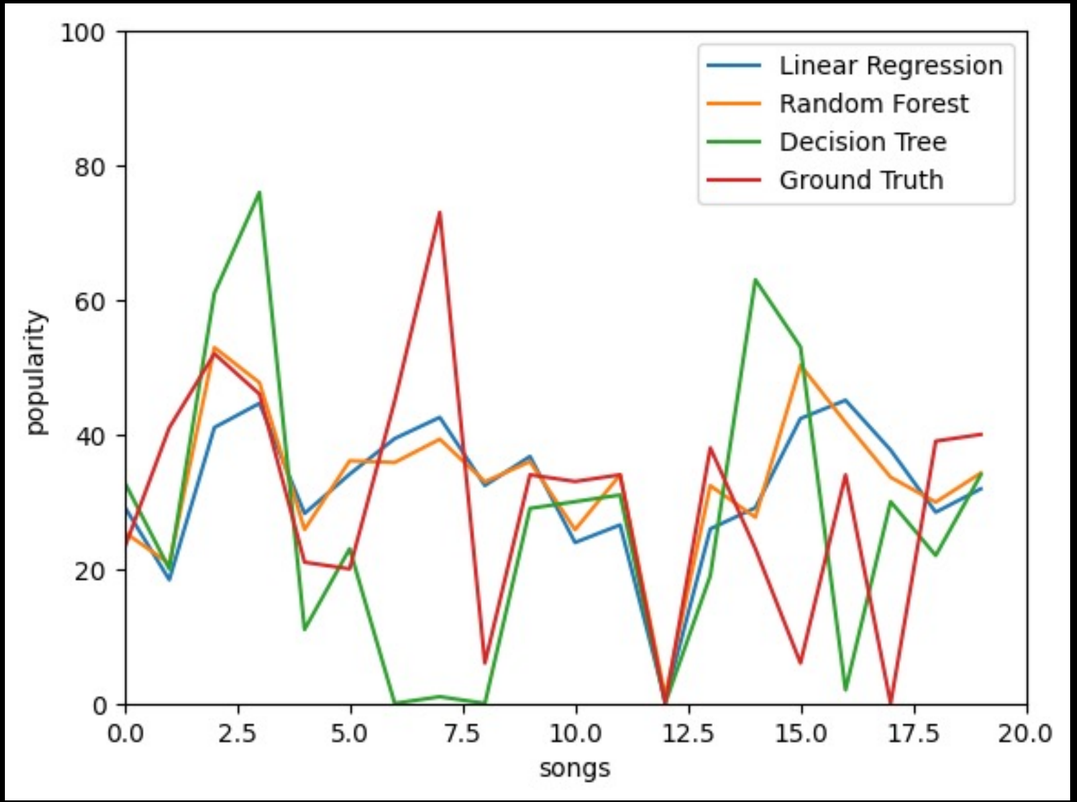
# Comparison: Re-evaluating and comparing the 3 models using RMSE.

**We use the best models of each of the 3 models and compare the results.**

| Models | Train Set | Test Set |
|---|---|---|
| Linear Regression | 14.5 | 14.5 |
| Decision Tree | 1.69 | 18.0 |
| Random Forest | 11.9 | 11.9 |

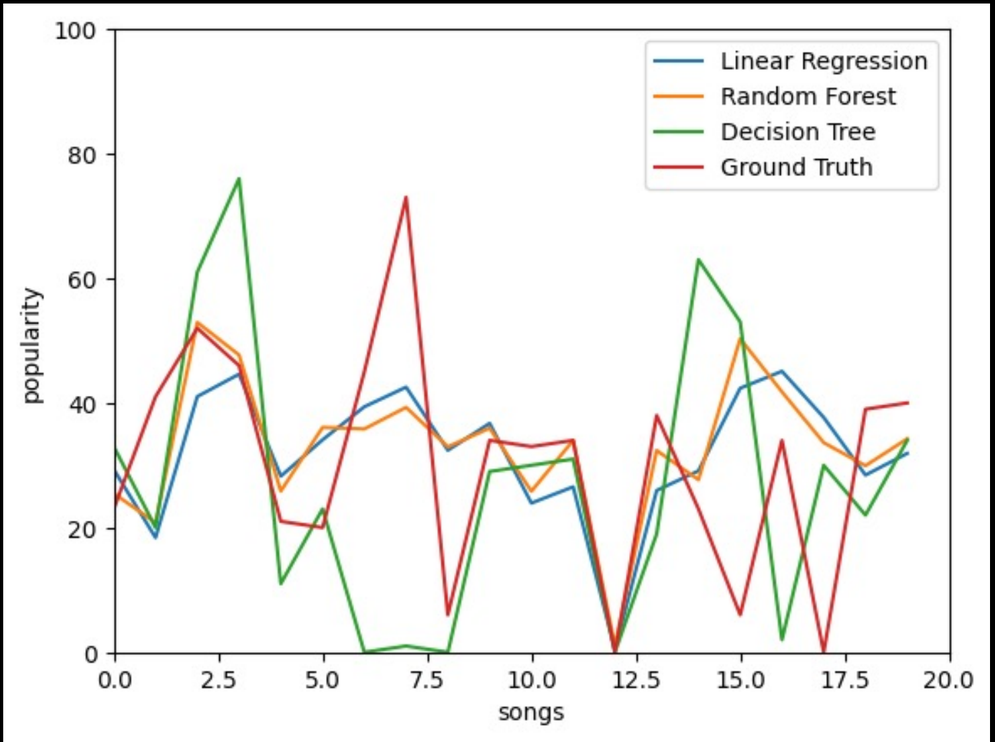
```python
test_samples = 20 # amount of songs which would be evaluated
# initialized empty lists for the models predictions
regression = []
random_forest = []
decision_tree = []
ground_truth = []
# collecting the models' predictions
for i in range(test_samples):
    regression.append(model_regression.predict([X_test[i]]))
    random_forest.append(model_random_forest_improved.predict([X_test[i]]))
    decision_tree.append(model_decision_tree.predict([X_test[i]]))
    ground_truth.append(y_test[i])
# Plotting the models' predictions in comparison to the ground truth
plt.plot(range(len(regression)), regression, label='Linear Regression')
plt.plot(range(len(random_forest)), random_forest, label='Random Forest')
plt.plot(range(len(decision_tree)), decision_tree, label='Decision Tree')
plt.plot(range(len(ground_truth)), ground_truth, label='Ground Truth')
plt.xlim([0, test_samples])
plt.ylim([0, 100])
plt.xlabel('songs')
plt.ylabel('popularity')
plt.legend()
plt.show()
```

Plotting the models against ground truth



# Comparison: The models are a good attempt at predicting popularity.

| Models | Train Set | Test Set |
|---|---|---|
| Linear Regression | 14.5 | 14.5 |
| Decision Tree | 1.69 | 18.0 |
| Random Forest | 11.9 | 11.9 |




## Accurate Models

The 3 models have a relatively low RMSE values. This means that the prediction are accurate.

## Consistent

Looking at the graph, the models do not stray too far from the ground truth.

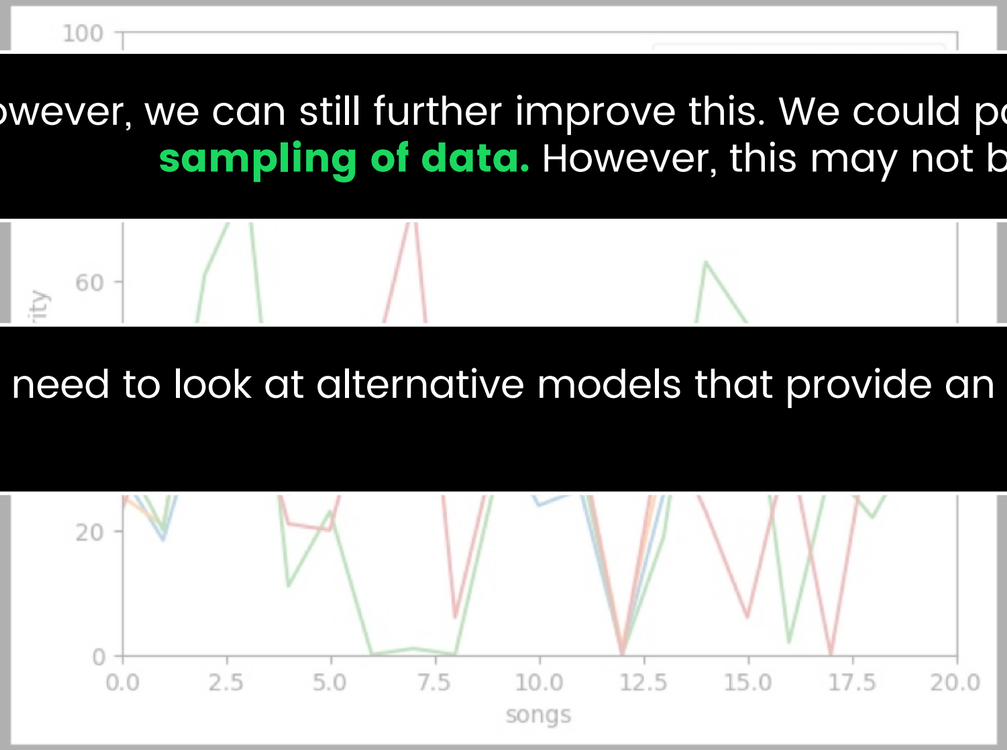
## Random Forest is best

The graph and the RMSE values concur that Random Forest is the best model amongst the 3 to predict popularity.

# Comparison: The models are a good attempt at predicting popularity.

| Models | Train Set | Test Set |
|---|---|---|

Random Forest gives us a good baseline to begin with. The variations in RMSE can be supported by non-quantitative factors such as marketing strategies and an artist's inherent popularity.




However, we can still further improve this. We could potentially further fine tune the model using **cross-validation and up sampling of data.** However, this may not be feasible due to **high computational cost of** fine tuning.

We need to look at alternative models that provide an **extensive depth as a random forests and is less computationally expensive**.

## Accurate Models

Looking at the graph, the models do not stray too far from the ground truth.

## Random Forest is best

The graph and the RMSE values concur that Random Forest is the best model amongst the 3 to predict popularity.

# Machine Learning: Reducing errors further by using a new algorithm - K-nearest neighbors (KNN) algorithm

This technique starts by determining the k points that are closest to the data point of interest. The popularity of the k closest points will then be averaged out to determine the expected popularity of this point of interest.

## Improving Effectiveness

### Year Constraint

Only considering songs released from 2016 onwards as songs released earlier have a significantly lower popularity. This significant effect might have been affecting the actual weights of the other attributes in our models.

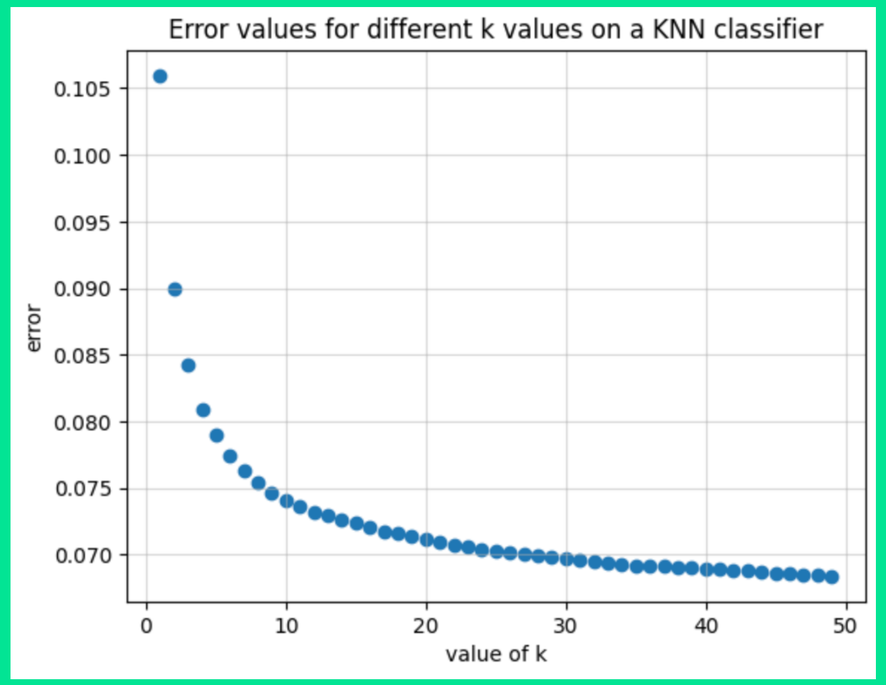
### Normalizing data

The dataset will be normalized so that each variable has a value between 0 and 1. As a result, our predictor will give equal weight to each of our features. Normalization can help improve the performance by reducing the effects of outliers and improving the convergence.

### Error Function

Mean Squared Error will be used as our data is continuous and thus best suited for this

# Machine Learning: K-nearest neighbors (KNN) algorithm



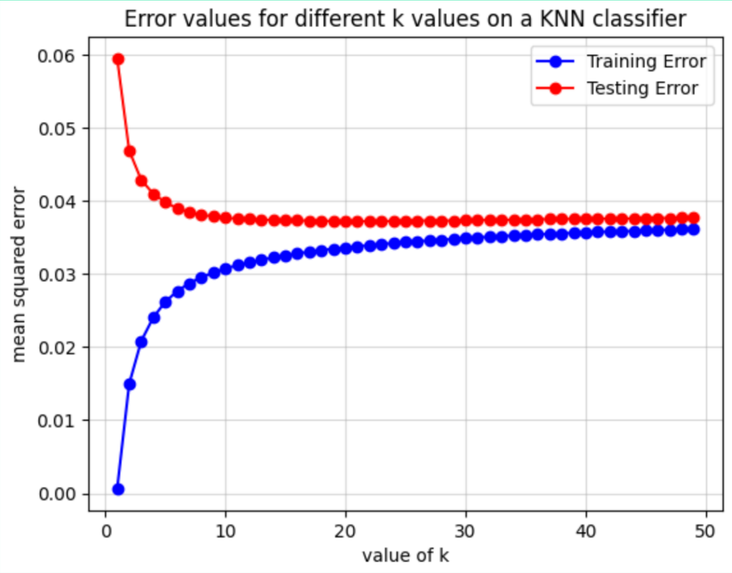

```
k=9
model = KNeighborsRegressor(n_neighbors=k)
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
print(f"Our testing error is {calculate_error(Y_pred, Y_test)}\n\n")
```

The error constantly lowers as the value of k increases, with a k value of 49 corresponding to the lowest error. However, if we select this lowest error value, our model would be overfit to our dataset.

You can use the elbow of the plot as a guide to choose the best k value for the KNN. The error stops rapidly dropping at this elbow. As such, the value of 9 was chosen.

With the k value set, the model was tested on the test set and a testing error(MSE) of 0.0740 was produced.
However, we could reduce this error further!

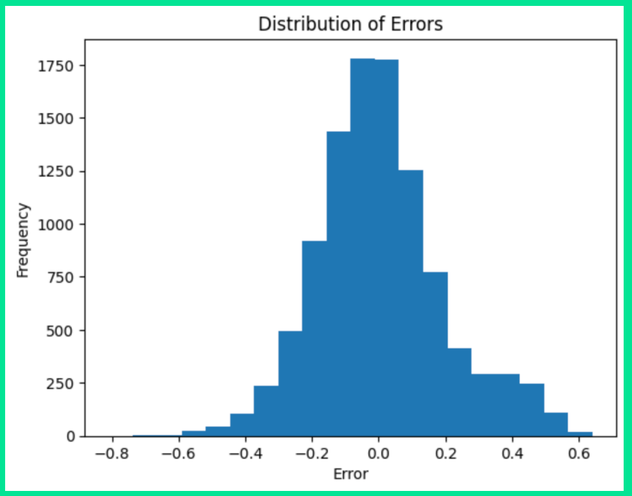
# Machine Learning: Reducing error using cross-validation




```
# set up cross-validation with k=5 folds
kf = KFold(n_splits=5, shuffle=True, random_state=1)
```

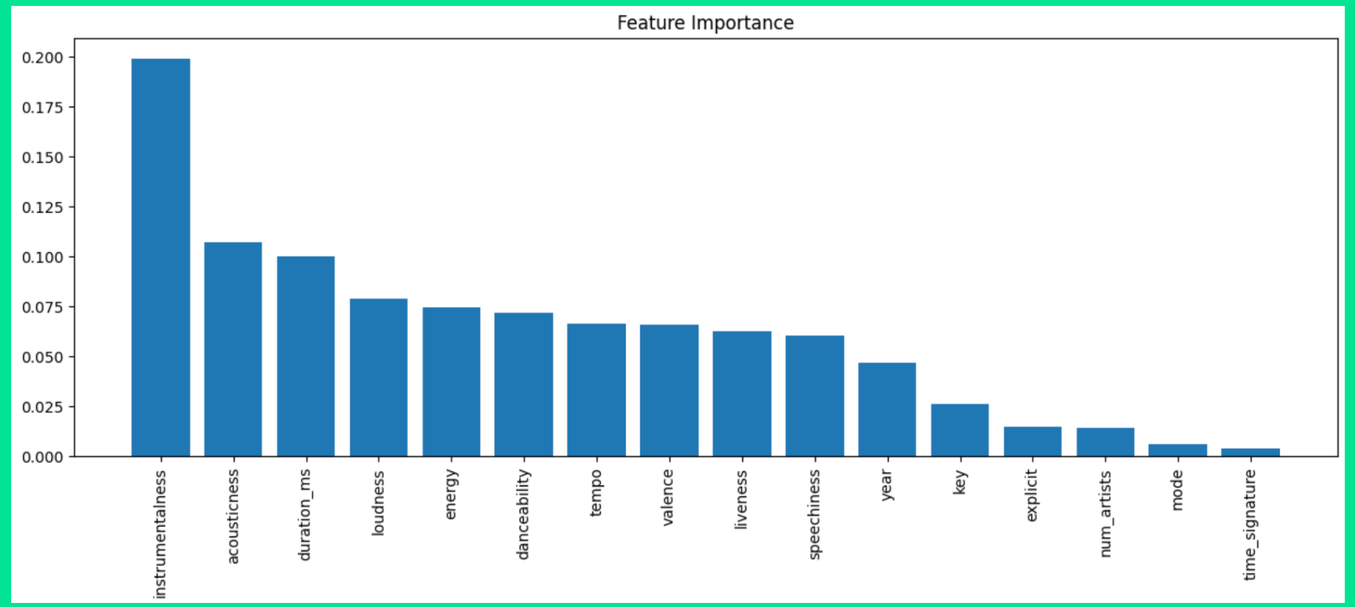
By using Cross-validation to reduce overfitting, we were able to reduce the testing error to 0.037. A mean squared error of 0.037 suggests that this model is performing reasonably well at predicting the popularity of songs in the test data.

## Visualizing Performance




The errors are almost normally distributed around 0, this is a good indication that the model is making accurate predictions

# Machine Learning: KNN- Feature Analysis

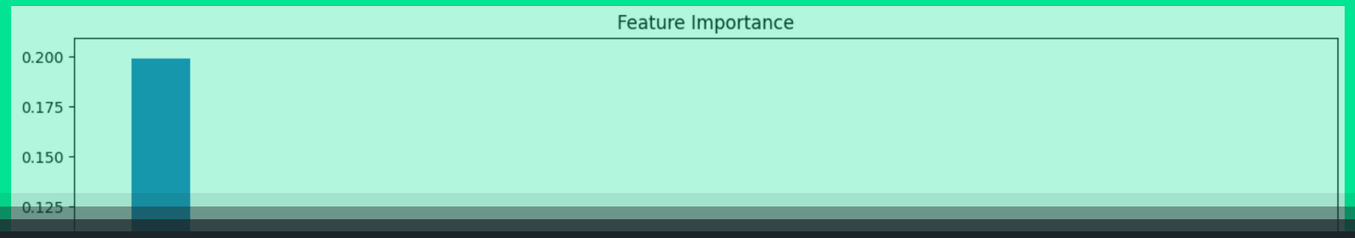



```python
from sklearn.ensemble import RandomForestRegressor
# Create a Random Forest Regressor model with 100 trees
model = RandomForestRegressor(n_estimators=100)
# Fit the model on the training data
model.fit(X_train, Y_train)
# Get feature importances from the model
importances = model.feature_importances_
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]
# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]
```

The features and their respective importance matches that of our conclusion in our EDA. This proves that the KNN algorithm has been assigning more weights to the important features we predicted earlier to ensure its accuracy

Note that year shows a lower importance here since we shaved off data before 2016 in this algorithm

# Machine Learning: KNN- Feature Analysis




As such, we choose KNN as our winning model for  independent artists to use as:

1. It has a low MSE of 0.037
2. It gives higher importance to features we concluded are important in our EDA

```
from sklearn.ensemble import RandomForestRegressor
# Create a Random Forest Regressor model with 100 trees
model = RandomForestRegressor(n_estimators=100)
# Fit the model on the training data
model.fit(X_train, Y_train)
# Get feature importances from the model
importances = model.feature_importances_
# Sort feature importances in descending order
indices = np.argsort(importances)[::-1]
# Rearrange feature names so they match the sorted feature importances
names = [X_train.columns[i] for i in indices]
```

importance here since we shaved off data before 2016 in this algorithm

# New techniques learnt

Using clustering to analyze data

Modeling Using KNN and Random Forest for prediction

Cross Validation to reduce overfitting and error

Feature Selection on Models to determine importance

# Key learning points and future exploration

## Key Learning Points

Working with and cleaning a raw dataset to better address our problem statement

Using techniques such as cross-validation to improve models

Linking back model's results to EDA to understand the workings of the model

## Future Exploration

Web scraping and removing top artists from dataset to further target independent artists

Going a step further and classifying songs by genre first and then evaluating each

# Thank You

Dixit Ayushman U2121836F
Summit Bajaj U2120089A