



TAYLOR'S UWE DUAL AWARDS PROGRAMMES

April 2024 Semester

Machine Learning and Parallel Computing

(ITS66604)

Assignment 2 – Group (30%)

Submission Date: 7th July 2024

Project Title: Prediction of song's popularity

by




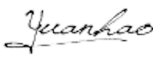
Group Name: MLPC grp 12

STUDENT DECLARATION

- I confirm that I am aware of the University's Regulation Governing Cheating in a University Test and Assignment and of the guidance issued by the School of Computing*

and IT concerning plagiarism and proper academic practice, and that the assessed work now submitted is in accordance with this regulation and guidance.

- 2. I understand that, unless already agreed with the School of Computing and IT, assessed work may not be submitted that has previously been submitted, either in whole or in part, at this or any other institution.*
- 3. I recognise that should evidence emerge that my work fails to comply with either of the above declarations, then I may be liable to proceedings under Regulation.*

Student Name	Student ID	Date	Signature	Score
Kao Cheng Yuan, Isaac	0363253	18/05/2024		
Tan Wei En	0363125			
Jocelyn Neoh Sze Ming	0363483			
Kao Cheng Buo, Caleb	0363254	17/07/2024		
Ngan Mei Suen	0350289	19/07/2024		
Li Yuanhao	0348637	19/07/2024		

Evidence of Originality

Similarity Score:

URL link to Dataset:

	https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube/data
--	---

Marking Rubrics (Lecturer's Use Only)		
Criteria	Weight	Score
Abstract	10	
Implementation & Results	30	
Analysis & Recommendations	25	
Conclusion	5	
Submission Requirements	5	
Project Presentation	10	
Grading <i>Excellent 90 – 100 marks</i> <i>Good 75 – 89 marks</i> <i>Fair 40 – 74 marks</i> <i>Poor 0 – 39 marks</i>	Total Marks (100%)	
	Total Marks (20%)	
<i>Remarks:</i>		

Table of Contents

Table of Contents	4
Abstract	5
Acknowledgments	5
1. Introduction	6
1.1 Background of the Problem	6
1.2 Importance and Relevance	6
1.3 Objectives of the Project	6
2. Literature Review	7
2.1 Related works and summary	7
2.2 Critical Analysis of Relevant Literature	8
2.3 Identification of Gaps in Current Research	11
2.4 Scope of Research	13
2.5 Summary of Literature review	14
3. Methodology	14
3.1 Dataset Description	14
3.1.1 Data Collection and Sources	14
3.1.2 Dataset Overview	14
3.2 Proposed Solution	16
3.2.1 Solution Overview	16
3.2.2 Justification	17
3.3 Potential Models	17
Random Forest	17
Ensemble methods	18
XGBoost	18
4. Implementation & Results	19
4.1 Code Implementation	19
JustSpotify	23
SpotiTube	33
JustSpotify vs SpotiTube	41
4.2 Summary of result	43
5. Analysis & Recommendations	45
Limitations	54
Recommendations	55
6. Conclusions	57
7. References	58

Abstract

The idea of predicting a song's popularity is thought to be a difficult task due to the huge amount of internal and external factors that may affect it. With the advent of machine learning all these factors can be calculated using various regression algorithms. Many others have also noticed this, and a recurring gap was the lack of variety of sources, most using just Spotify API as their dataset. This assignment rectifies this by using a dataset with variables outside of just Spotify API. Since the dataset can be split into "data that comes from Spotify" and "data that comes from YouTube", the plan is to train a model with only data from Spotify, then train similar models with the added data from YouTube, to see if there are any significant differences that can be attributed to bias from Spotify.

Acknowledgments

We would like to express our sincere gratitude to Ms. Nicole Teah Yi Fan for her exceptional teaching, accommodating multiple extensions, and promptly addressing any questions we had. We are also thankful to Kaggle for providing a wealth of datasets for exploration, and we specifically acknowledge Marco Guarisco, Marco Sallustio, and Salvatore Rastelli for their contributions to the dataset we used. Additionally, we extend our thanks to Google Scholar for offering a diverse range of sources for our literature review. Lastly, we deeply appreciate the support and collaboration of all our group members throughout this project.

1. Introduction

1.1 Background of the Problem

Every day, on music streaming platforms such as Spotify, there is an ever increasing variety and amount of songs that are created and uploaded. These platforms come with a variety of metadata that can be analysed, which can provide great potential insights. Using machine learning, the data can be used to learn factors that contribute to a song's popularity, as well as predict if any future songs will be popular.

1.2 Importance and Relevance

The information gained from analysis can help artists see what kind of factors are popular in the market, allowing for proper preparation and more tailored marketing. Additionally, artists can also focus their songs based on what seems popular from the data.

1.3 Objectives of the Project

The goal of this project is to use machine learning to predict the popularity of a song based on a variety of features as well as look for the biggest factors that go into a song's popularity.

2. Literature Review

2.1 Related works and summary

There are 4 articles we'll be going over in this literature review. The first one being *Predicting song popularity based on Spotify's audio features: insights from the Indonesian streaming users* by Harriman Samuel Saragih (2023). The second being *Song Popularity Predictor* by Mohamed Nasreldin (2018). The third article is *Predicting Song Popularity* by James Pham, Edric Kyauk and Edwin Park (2015). The last article is *Predicting Song Popularity Using Machine Learning Algorithms*, a report by Bihag Dave, Koushiki Bhattacharyya, Prayag Savsani, and Yashraj Kakkad (2020).

The following table details the Article, Authors, Year, Focus and Results of each paper:

Article	Authors	Year	Focus	Results
Predicting song popularity based on Spotify's audio features: insights from the Indonesian streaming users	Harriman Samuel Saragih	2023	Predicting song popularity in the Indonesian market	Identified specific audio characteristics that correlate with song popularity in the Indonesian market

Song Popularity Predictor	Mohamed Nasreldin, Stephen Ma, Eric Dailey, Phuc Dang	2018	Factors influencing song success using Million Song Dataset and Spotify's API	Achieved over 68% accuracy in predicting whether a song would hit Billboard's Top 100
Predicting Song Popularity	James Pham, Edric Kyauk, Edwin Park	2015	Predicting song popularity using various machine learning algorithms	Identified most relevant predictors of song popularity; SVM with Gaussian kernel achieved highest F1 score
Predicting Song Popularity Using Machine Learning Algorithms	Bihag Dave, Koushiki Bhattacharyya, Prayag Savsani, Yashraj Kakkad	2020	Predicting song popularity using Spotify's Web API data	Found that Random Forests and Boosting showed improved recall and F1 scores. Had challenges with high number of false negatives due to imbalanced dataset

2.2 Critical Analysis of Relevant Literature

The follow table for critical analysis will go over each paper's Features Used, Machine Learning Techniques used, Train Test Split, Dataset Size (before and after cleaning, if possible), Scope of the paper, Strengths, Weaknesses, Methodology

Article	Features Used	Machine Learning Techniques	Train Test Split	Dataset Size	Scope	Strengths	Weaknesses	Methodology
Predicting song popularity based on Spotify's audio features: insights from the Indonesian streaming users	Popularity, Popularity Class, Acousticness, Danceability, Energy, Duration, Instrumentalness, Tempo, Liveness, Loudness, Speechiness, Mode, Explicit . Key, Time Signature , Release Date	linear regression, neural network, support vector regression, and random forest	70% Train, 30% Test	600K songs (Spotify API)	Localized to Indonesian market, uses Spotify's Web API data	Focuses on regional differences in music preferences, used a very large dataset for analysis	Limited to Indonesian market, may not be applicable to larger audience	Machine learning models, audio features

Song Popularity Predictor	tempo, duration, mode, loudness, key, time signature, section start, artist familiarity, artist popularity, artist name, artist location, releases, title, year, song hotness	XGBoost, grid search tuning	Cross validation with 10 folds (Split into 10 parts, 1 is used for testing, other 9 for training, each part used as test set once)	10,000 songs. (Subset of Million Songs Dataset), Cleaning was done, but the amount of rows dropped was not mentioned	Broad, using Million Song Dataset, Spotify's Web API data and Billboard top 100	Comprehensive dataset and preprocessing, used more datasets than other papers	Challenges with missing data, data cleaning, unknown how much data was cut	Machine learning models, Billboard Top 100 benchmark, XGBoost, grid search tuning
Predicting Song Popularity	Key, loudness, mode, mode confidence, release	SVMs, neural networks, logistic regression, linear	90% train, 10% test	10,000 tracks (Subset of Million Songs Dataset), 2,717 tracks (After cleaning)	Broad, using Million Song Dataset and Spotify's Web API data	Detailed use of feature selection and machine learning evaluation	Potential overfitting due to high-dimensional data	Various ML algorithms, feature selection, SVMs, neural networks, logistic regression, linear

								regression
Predicting Song Popularity Using Machine Learning Algorithms	Acousticness, Danceability, Energy, Instrumentalness, Liveness, Loudness, Speechiness, Tempo, Valence, Popularity, Count, Mode,	Regression, classification, Random Forests, Boosting	Stratified K-fold Cross-Validation, 3-fold Cross-Validation	175,000 songs (From Spotify API)	Broad, using Spotify's Web API data	Thorough use and evaluation of ensemble learning techniques	Imbalanced dataset challenges, computational expense of ensemble methods	Regression, classification, ensemble learning, Randomized Grid Search

2.3 Identification of Gaps in Current Research

Despite the amount covered by the papers, various gaps in research can be observed, some gaps being specific to only 1 paper, with others over multiple papers

Gap	Description	Papers with the Gap
Market Generalizability	Problem: Saragih’s (2023) study is region-specific to Indonesia and may not apply to other markets. Solution: Further research in different cultural and	Saragih (2023)

	geographic contexts is needed.	
Social Influence	<p>Problem:</p> <p>Pham et al. (2015) mentions social influence but does not explore it deeply.</p> <p>Solution:</p> <p>Integrating social media metrics and user engagement data could enhance the predictive power of datasets.</p>	Pham et al. (2015)
Imbalanced Datasets	<p>Problem:</p> <p>Dave et al. (2020) discuss challenges with imbalanced datasets. An important one they were unable to overcome was predicting popular vs. unpopular songs.</p> <p>Solution:</p> <p>More advanced techniques for handling class imbalance should be explored.</p>	Dave et al. (2020)
Real-time Data Integration	<p>Problem:</p> <p>All the studies rely on historical data, which will not keep up with any growing trends that may occur. This issue is especially pertinent as the taste people music has the potential to rapidly change</p> <p>Solution:</p> <p>Incorporating real-time streaming data could improve prediction accuracy by reflecting current trends and user behavior more effectively.</p>	Saragih (2023), Nasreldin et al. (2018), Pham et al. (2015), Dave et al. (2020)

Lack of Variety in Datasets	<p>Problem:</p> <p>Most studies use the Million Song Dataset and Spotify API, which might introduce bias, either by Spotify's algorithms or by the method Million Song Dataset gathers data.</p> <p>Solution:</p> <p>A broader variety of datasets would enhance the robustness of findings.</p>	Saragih (2023), Nasreldin et al. (2018), Pham et al. (2015), Dave et al. (2020)
-----------------------------	--	---

Overall, the most relevant gaps are 'Real-time Data Integration' and 'Lack of Variety in Datasets', with them affecting all papers. In this assignment, the main focus will be 'Lack of Variety in Datasets', as the implementation of real time data is outside the capabilities and scope of this assignment.

2.4 Scope of Research

The main scope of research for this assignment will be to find a dataset and perform machine learning to find out the factors that go into popularity of a song as well as training a model that will be able to predict if a song is popular or unpopular in testing.

This is similar to what the other papers in the literature review have done, however, the differential in this assignment is a focus on a dataset that does not take solely from Spotify or The Million Song Dataset. As one of the reviewed papers has also used a dataset of the Top 100 Billboard songs, care will be taken to avoid similar datasets to that as well.

To achieve this, a dataset that contains both Youtube and Spotify was found. Having both Spotify and Youtube will help for bias, and the final model and result may even be compared to the research papers that only used Spotify API to check for bias.

2.5 Summary of Literature review

4 papers, *Predicting song popularity based on Spotify's audio features: insights from the Indonesian streaming users* Saragih (2023), *Song Popularity Predictor* Nasreldin (2018), *Predicting Song Popularity* by Pham et al (2015), and *Predicting Song Popularity Using Machine Learning Algorithms* Dave et al (2020) were explored. Multiple gaps in the reviewed papers were identified, with the most achievable and relevant being potential bias from a lack of dataset variety.

In comparison, this research plans to use a more diverse dataset that includes data from both Spotify and Youtube. This approach will allow for a mitigation of any biases from Spotify, and can be compared to previous literature to truly check for any biases once finished.

3. Methodology

3.1 Dataset Description

3.1.1 Data Collection and Sources

‘Spotify and Youtube’ by Salvatore Rastelli, Marco Guarisco, and Marco Sallustio fits the criteria that was established. This dataset was last updated in 2023 and taken from Kaggle. The dataset contains nearly 21k rows, and has 26 variables.

3.1.2 Dataset Overview

Variable name	Description	Source
Artist	The name of the artist.	Spotify
Url_spotify	The URL of the artist on Spotify.	Spotify
Track	The name of the song that is shown on the Spotify platform.	Spotify
Album	Which album the song is included in on Spotify.	Spotify
Album_type	Indicates if the song is released on Spotify as a single or part of an	Spotify

	album	
Uri	The Spotify URI used to find the song through the API.	Spotify
Danceability	A measure of how suitable a track is for dancing, based on elements like tempo and rhythm.	Spotify
Energy	A perceptual measure of intensity and activity, ranging from 0.0 to 1.0.	Spotify
Key	The key in which the track is composed, mapped to pitches using standard pitch notation.	Spotify
Loudness	The average loudness of the track in decibels (dB).	Spotify
Speechiness	Detects the presence of spoken words in a track, ranging from 0.0 to 1.0.	Spotify
Acousticness	A confidence measure from 0.0 to 1.0 of whether the track is acoustic.	Spotify
Instrumentalness	Predicts whether a track is instrumental, with higher values indicating instrumental tracks.	Spotify
Liveness	Detects the presence of an audience in the recording, with values above 0.8 indicating live performance.	Spotify
Valence	A measure of the musical positiveness conveyed by a track, from 0.0 to 1.0.	Spotify
Tempo	The overall estimated tempo of a track in beats per minute (BPM).	Spotify
Duration_ms	The duration of the track in milliseconds.	Spotify
Stream	The number of streams on Spotify.	Spotify
Url_youtube	The URL of the YouTube video linked to the song.	YouTub

		e
Title	The title of the YouTube video.	YouTub e
Channel	The name of the YouTube channel that published the video.	YouTub e
Views	The number of views on YouTube.	YouTub e
Likes	The number of likes on YouTube.	YouTub e
Comments	The number of comments on YouTube.	YouTub e
Description	The description of the YouTube video.	YouTub e
Licensed	Boolean that indicates whether the video represents licensed content (TRUE/FALSE).	YouTub e
official_video	Boolean that indicates whether the video is the official music video (TRUE/FALSE).	YouTub e

As we can see, a majority of variables that make up a song comes from Spotify's API, with YouTube providing useful statistics such as 'Likes' and 'Views'.

3.2 Proposed Solution

3.2.1 Solution Overview

The solution to find bias with this dataset is twofold. First, a model is made using only data from Spotify sources. After, a model that includes data from YouTube will be trained, and the two models will be compared.

The general plan for the machine learning models is to use linear regression at first to predict a song's popularity. If the linear regression model underperforms, considerations will be made towards using a polynomial regression model. A logistic regression model will also be used to train a model that can predict whether a song will be popular or unpopular. The first model will be trained using variables "Artist" to "Stream", with the second model including variables "Url_youtube" to "official_video"

More models, such as random forest or XGBoost similar to the researched papers may be considered upon further research later in this assignment.

3.2.2 Justification

The proposed solution seeks to look for any biases in Spotify's API by comparing a model that uses just data sources from just Spotify, and a similar model that uses the same data sources from Spotify, but also adds YouTube sources as well. Comparing the two models and their results would be able to show if there are any large biases that comes from Spotify's API, depending on how different the two models are in the end.

Linear regression is a basic, straightforward model that can be used as a baseline for predicting the popularity of songs.

Polynomial regression can be used if the data turns out to be too complex for linear regression to help identify more complex trends.

Since songs can be classified as popular or not popular, logistic regression can be used for this binary classification.

3.3 Potential Models

It was decided that the best way to look for more advanced methods of machine learning was to look at previous examples. *Predicting Song Popularity Using Machine Learning Algorithms*, for example, found great success with Random Forest and Ensemble methods. Similarly, *Song Popularity Predictor* also found results with XGBoost.

Random Forest

Random Forest, according to Breiman (2001), is an ensemble learning method that constructs multiple decision trees during training and outputs either the mode of the classes, or the mean

prediction of the individual trees. Random Forest has each tree is built from a random part of the training data and features, to attempt to reduce overfitting and to improving generalization. Random forests may be effective to study a song's popularity as it can handle the large amount of features the dataset can bring, as well as the various interactions between each variable. Additionally, random forest has the ability to capture non-linear relationships and provide feature importance metrics, which will be important to find the factors that contribute to a song's popularity.

Ensemble methods

Zhou (2012) discusses multiple ensemble methods, but in this case only the basics of Bagging, Boosting, and Stacking will be focused on.

The basic premise of ensemble methods is to combine the predictions of multiple models to average out errors to create a more accurate model.

Bagging is the training of multiple independent models on random parts of the data and averaging their predictions. Boosting builds models in order, with each model correcting the errors of the previous. Stacking involves training multiple models and using their predictions as inputs to a final model.

By using Bagging, Boosting, and Stacking to combine different models, a larger range of patterns and interactions can be captured from the data, which will lead to a greater understanding of the complexities of a song's popularity.

XGBoost

Chen (2012), describes XGBoost as “a scalable end-to-end tree boosting system”. It is an advanced version of gradient boosting, and similarly builds models in order, with each model correcting the errors of the previous. Unlike ensemble boosting, XGBoost has various optimization techniques, making it more efficient and able to scale better. XGBoost also possesses regularization parameters, allowing control over the model's complexity and the prevention of overfitting.

XGBoost will be useful to the analysis of a song's popularity due to its high accuracy and efficiency in handling large datasets.

4. Implementation & Results

4.1 Code Implementation

```
[ ] df.info()
    print("\nTotal rows is: " + str(df.shape[0]))

[ ] df.isnull().any()

[ ] df.isnull().sum()

[ ] df = df.dropna()
```

Figure 1: EDA performed

EDA consists of checking the dataset for which fields have empty records, counting how many empty records per field, and removing the entries/rows that contain empty records.

```
[ ] df.duplicated().any()

False
```

Figure 2: duplicate row check

The code then checks the dataset for any duplicate rows.

```
[ ] df = df.sample(n=2000, random_state=42)
    #Reducing Sample size to 2000 to limit RAM usage and allow for quicker runtimes
```

Figure 3: random sampling

They randomly choose 2000 rows of the dataset that will be used in order to reduce computational requirements

```

  ▾ JustSpotify

  Create of df with just sources from spotify

  [ ] df_JustSpotify = df.iloc[:, [0, 3, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 27]]
      df_JustSpotify.head()

  ⇌ Show hidden output

  ▾ SpotiTube

  Similar to JustSpotify, but with added values Views Likes and Comments that are Youtube sources

  [ ] df_SpotiTube = df.iloc[:, [0,3, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 27, 21, 22, 23]]
      df_SpotiTube.head()

  ⇌ Show hidden output

```

Figure 4: Creating the data frames

As the diagram shown above, there are two data frames which are ‘df_JustSpotify’ and ‘df_SpotiTube’. For each of the dataframe, there are selected columns from the original dataset and this is accomplished by the “.loc” feature. This is beneficial for removing irrelevant columns from the dataset.

After the selection, the ‘df_JustSpotify’ data frame contains the information that is only on spotify, which including column of ‘Index’, ‘Track’, ‘Danceability’, ‘Energy’, ‘Key’, ‘Loudness’, ‘Speechiness’, ‘Acousticness’, ‘Instrumentalness’, ‘Liveness’, ‘Valence’, ‘Tempo’, ‘Duration’ in milliseconds, and ‘Stream’. On the other hand, the ‘SpotiTube’ data frame contains the information that is on spotify and youtube. They are mostly the same as the spotify only, but with ‘Likes’, ‘Comments’, and ‘Views’ added.

```

[ ] correlation_matrix = df_numeric.corr() #Calculating correlation matrix

plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', vmin=-1, vmax=1)
plt.title('Correlation Matrix of Numeric Features')
plt.show()

```

Figure 5: code for calculating correlation matrix

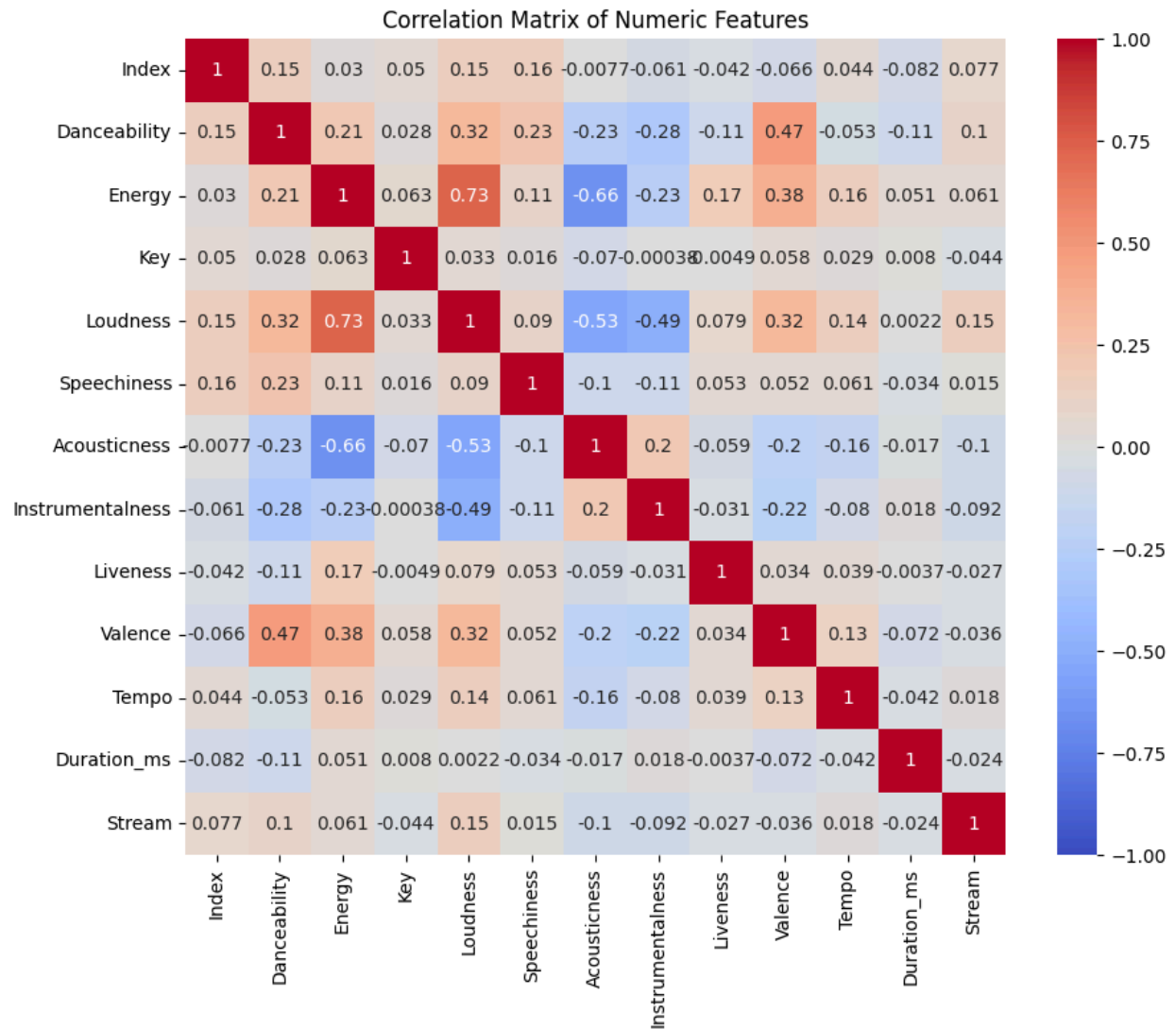


Figure 6: Heatmap for correlation analysis

Before proceeding to the modeling, understanding the correlations between variables is crucial.

The diagrams above illustrate the use of heatmaps in identifying values with the highest correlation to ‘Stream’.

```
correlation_matrix.loc['Stream', :]
```

Index	0.077361
Danceability	0.101546
Energy	0.061348
Key	-0.044129
Loudness	0.145692
Speechiness	0.014827
Acousticness	-0.100368
Instrumentalness	-0.091670
Liveness	-0.027207
Valence	-0.036142
Tempo	0.018258
Duration_ms	-0.024045
Stream	1.000000
Name: Stream, dtype: float64	

Figure 7: summary of correlation coefficients

The code above summarizes the correlation coefficients of each variable in the dataset with the variable 'Stream'. Based on the output, variable 'Loudness' appears to be most positively correlated with 'Stream'. In this case, when the variable 'Stream' increases and the variable 'Loudness' tends to increase too. On the other hand, the variable 'Acousticness' is the most negatively correlated with 'Stream'. This indicates that as 'Stream' increases, the variable 'Acousticness' tends to decrease.

```
[ ] correlation_matrix.loc['Stream', :].abs().sort_values(ascending=False)
```

```
⇒ Stream      1.000000  
   Loudness    0.145692  
   Danceability 0.101546  
   Acousticness 0.100368  
   Instrumentalness 0.091670  
   Index        0.077361  
   Energy       0.061348  
   Key          0.044129  
   Valence      0.036142  
   Liveness     0.027207  
   Duration_ms  0.024045  
   Tempo        0.018258  
   Speechiness  0.014827  
   Name: Stream, dtype: float64
```

The diagram above rank variables by their absolute correlation strength with ‘Stream’. It sorts them from highest to lowest absolute correlation coefficient, providing insights into which variables have the strongest overall relationship with ‘Stream’, regardless of direction.

JustSpotify

```
▶ from sklearn.model_selection import train_test_split  
   y_Spotify = df_JustSpotify['Stream']  
   X_Spotify = df_JustSpotify.iloc[:,2:-1]  
   y_Spotify.head()
```

```
⇒ 19563      231064701.0  
   9743       186724705.0  
   355        524847088.0  
   12607       21672529.0  
   17678       198920216.0  
   Name: Stream, dtype: float64
```

To create a model, the initial step involves splitting the data into X and y variables, where y represents the target variable ‘Stream’, and X represents the feature variables. In this case, the ‘X_Spotify’ extracts all columns from the third to the second-to-last in ‘df_JustSpotify’. This is

because the first and second columns are the index, also the last column is the target variable 'Stream'.

```
[ ] y_Spotify.shape
⇌ (2000,)

[ ] x_Spotify.shape
⇌ (2000, 11)
```

The code above provides the size and structure of data, this helps to ensure the data is properly prepared for machine learning and compatible with the algorithms that plan to apply.

```
[ ] X_train_Spotify, X_test_Spotify, y_train_Spotify, y_test_Spotify = train_test_split(X_Spotify, y_Spotify, test_size=0.2, random_state=42)
```

The code above split the data into training and testing sets. This is to ensure the machine learning model can be trained on one subset of the data and evaluate its performance on another subset. As shown in the diagram, the data was split into a ratio of 8:2, with 20% designated for testing and the remaining 80% allocated for training.


```

from sklearn.linear_model import LinearRegression

# Create an instance of the class
lin_reg_Spotify = LinearRegression()

# Check if 'Track' column exists before dropping
if 'Track' in X_train_Spotify.columns:
    X_train_Spotify = X_train_Spotify.drop('Track', axis=1)
if 'Track' in X_test_Spotify.columns:
    X_test_Spotify = X_test_Spotify.drop('Track', axis=1)


# Fit the instance on the data and then predict the expected value
lin_reg_Spotify = lin_reg_Spotify.fit(X_train_Spotify, y_train_Spotify) # lin_reg stores model parameters
lin_pred_Spotify = lin_reg_Spotify.predict(X_test_Spotify)

# Ensure it is 1-dimensional
coefficients = lin_reg_Spotify.coef_.flatten()

# Create a DataFrame to store feature names and their corresponding coefficients
# Exclude 'Track' from the feature names
coef_df_lr = pd.DataFrame({'Feature': X_train_Spotify.columns, 'Coefficient': coefficients})

print("Spotify Coefficients:\n", coef_df_lr.to_string(index=False))
print('\nSpotify Intercept: ', lin_reg_Spotify.intercept_)

```

 Spotify Coefficients:

Feature	Coefficient
Danceability	1.896632e+08
Energy	-7.123225e+07
Key	-3.276722e+06
Loudness	8.882143e+06
Speechiness	-8.256267e+07
Acousticness	-5.357891e+07
Instrumentalness	-4.101182e+07
Liveness	-1.531096e+07
Valence	-1.522195e+08
Tempo	3.027299e+05
Duration_ms	-1.822324e+01

Spotify Intercept: 228671258.35550418

The diagram above illustrates the creation and fitting of a linear regression model, and drops unnecessary columns. By building this model, the intercept and coefficient are discovered.

In line 7, the code checks if the ‘Track’ column exists in the training and testing datasets and drops it if it does. This ensures that the ‘Track’ column is not used as a feature in the model.

As the result displayed, there are both negative and positive values showing as coefficients. This explains the feature’s positive and negative relationship to the target feature. Further, the ending part of the coefficient value (Eg: +07 and +08) is scientific notation. This is a more condensed method of representing extremely big or very small numbers. For the intercept, it provides a baseline value for the target variable when all features are zero.

According to the result, the feature 'Loudness' with coefficient value 8.882143×10^6 . This means that keeping other parameters equal, a one-unit increase in 'Loudness' corresponds to an increase of roughly 8882143 (8.882143×10^6) in the target variable 'Streams'.

```
[ ] lin_reg_importance_Spotify = np.abs(lin_reg_Spotify.coef_) / np.sum(np.abs(lin_reg_Spotify.coef_))

print("Linear Regression Normalized Feature Importance:")
for feature, importance in zip(X_Spotify, lin_reg_importance_Spotify):
    print(f"{feature}: {importance:.4f}")
```



```
Linear Regression Normalized Feature Importance:
Danceability: 0.3069
Energy: 0.1153
Key: 0.0053
Loudness: 0.0144
Speechiness: 0.1336
Acousticness: 0.0867
Instrumentalness: 0.0664
Liveness: 0.0248
Valence: 0.2463
Tempo: 0.0005
Duration_ms: 0.0000
```

By illustrating the relative importance of each feature, the code computes and prints the normalized feature importance for a linear regression model, aiding in model interpretation. This is helpful in feature selection, model enhancement, and deriving conclusions from the data.

By observing the output, 'Danceability' is the most significant feature with a value of 0.3069, which is a 30.69% contribution to the model. This indicates that changes in the danceability level are the most influential, since it has the greatest effect on the target variable.

```
[ ] from sklearn.model_selection import GridSearchCV
    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import PolynomialFeatures

    # setting up a pipeline
    estimator = Pipeline([
        ("polynomial_features", PolynomialFeatures()),
        ("linear_regression", LinearRegression())])

    # parameters for gridsearch
    params = {
        'polynomial_features__degree': [1, 2, 3],
    }

[ ] # search for the best parameters
    grid_search_poly_Spotify = GridSearchCV(estimator, params)

    # read the best parameters
    grid_search_poly_Spotify.fit(X_train_Spotify, y_train_Spotify)
    grid_search_poly_Spotify.best_score_, grid_search_poly_Spotify.best_params_

⇒ (0.04304663680392364, {'polynomial_features__degree': 1})
```

The diagram above shows the setup of a pipeline with polynomial features. Also, a parameter for ‘GridSearchCV’ is defined to tune the degree of polynomial features. In the second cell of the diagram, it uses ‘GridSearchCV’ to search for the best parameters on the dataset ‘X_train_Spotify’ and ‘y_train_Spotify’.

According to the output, 0.043 indicates the best cross-validation score obtained throughout the grid search procedure. On the other hand, ‘{‘polynomial_features__degree’:1}’ result implies that a polynomial degree of 1 is the optimal degree of polynomial features as found by the grid search. In other words, a basic linear relationship (degree 1 polynomial) was found to be optimal for the dataset after assessing degrees 1, 2, and 3.

```
[ ] # train a model with best parameter
pf_Spotify = PolynomialFeatures(degree=1)

poly_reg_Spotify = Pipeline([
    ("make_higher_degree", pf_Spotify),
    ("linear_regression", LinearRegression())])

poly_reg_Spotify = poly_reg_Spotify.fit(X_train_Spotify, y_train_Spotify)
poly_pred_Spotify = poly_reg_Spotify.predict(X_test_Spotify)
```

The code above is to train a new pipeline with the best parameters found ('degree=1' for polynomial features) and evaluate its performance on the test set 'X_test_Spotify'.

```
# read the model coefficients
features = poly_reg_Spotify.named_steps["make_higher_degree"].get_feature_names_out()
coefficients = poly_reg_Spotify.named_steps["linear_regression"].coef_.flatten()

# Create a DataFrame with the coefficients
coef_df_pr_Spotify = pd.DataFrame({
    'Feature': features,
    'Coefficient': coefficients
})

# Display the coefficients

print("Spotify Polynomial Coefficients:\n", coef_df_pr_Spotify.to_string(index=False))
print ('\nSpotify Polynomial Intercept: ', poly_reg_Spotify.named_steps["linear_regression"].intercept_)
```

```
Spotify Polynomial Coefficients:
  Feature  Coefficient
1      0.000000e+00
Danceability  1.896632e+08
Energy -7.123225e+07
Key -3.276722e+06
Loudness  8.882143e+06
Speechiness -8.256267e+07
Acousticness -5.357891e+07
Instrumentalness -4.101182e+07
Liveness -1.531096e+07
Valence -1.522195e+08
Tempo  3.027299e+05
Duration_ms -1.822324e+01

Spotify Polynomial Intercept:  228671258.3562097
```

After training and making predictions with 'poly_reg_Spotify', the code extracts and display the model coefficients. According to the result, the variable 'Instrumentalness' with coefficient value -4.101182e+07. This imply that if 'Instrumentalness' increases by 1 unit, the predicted number of 'Streams' would decrease by approximately 41011820 ($4.101182e * (10^7)$) units, in the scenario of all other factors remaining unchanged.

```


# Extract the linear regression model from the pipeline
poly_reg_model_Spotify = poly_reg_Spotify.named_steps['linear_regression']

# Get the feature names including the polynomial features
# Use the original features used to fit the pipeline
poly_features_Spotify = poly_reg_Spotify.named_steps['make_higher_degree'].get_feature_names_out(X_train_Spotify.columns)

# Normalize coefficients for Polynomial Regression
poly_reg_importance_Spotify = np.abs(poly_reg_model_Spotify.coef_) / np.sum(np.abs(poly_reg_model_Spotify.coef_))

# Display Polynomial Regression normalized feature importances
print("Polynomial Regression Normalized Feature Importance:")
for feature, importance in zip(poly_features_Spotify, poly_reg_importance_Spotify):
    print(f"{feature}: {importance:.4f}")

```

 Polynomial Regression Normalized Feature Importance:
 1: 0.0000
 Danceability: 0.3069
 Energy: 0.1153
 Key: 0.0053
 Loudness: 0.0144
 Speechiness: 0.1336
 Acousticness: 0.0867
 Instrumentalness: 0.0664
 Liveness: 0.0248
 Valence: 0.2463
 Tempo: 0.0005
 Duration_ms: 0.0000

The code above retrieves the coefficients from the polynomial regression model and normalizes them to determine the relative importance of each feature. The output displayed that ‘Danceability’ is the most significant feature with a value of 0.3069, which is a 30.69% contribution to the model. This implies that changes in ‘Danceability’ have the greatest impact on predicting the target variable ‘Streams’ compared to the other features included in the model.

```

[ ] from sklearn.ensemble import RandomForestRegressor

rf_reg_Spotify = RandomForestRegressor(random_state=42)
rf_reg_Spotify.fit(X_train_Spotify, y_train_Spotify)
rf_pred_Spotify = rf_reg_Spotify.predict(X_test_Spotify)

```

The code above is creating and using a Random Forest regressor model using ‘sklearn’. In line 4, it fits the Random Forest model using the training data (‘X_train_Spotify’) and their corresponding target values (‘y_train_Spotify’). Subsequently, it employs the trained model (‘rf_reg_Spotify’) to make predictions on the test data (‘X_test_Spotify’). Also, the predicted values are stored in ‘rd_pred_Spotify’.

```

# Feature importance for Random Forest
rf_feature_importance_Spotify = rf_reg_Spotify.feature_importances_

# Use the same feature names used to train the Random Forest
# Assuming X_train_Spotify contains the correct features
rf_feature_importance_df_Spotify = pd.DataFrame({
    'Feature': X_train_Spotify.columns, # Change to the features used for training
    'Importance': rf_feature_importance_Spotify
}).sort_values(by='Importance', ascending=False)

print("Random Forest Feature Importance:")
print(rf_feature_importance_df_Spotify)

```

```

Random Forest Feature Importance:
   Feature  Importance
3  Loudness    0.142406
8   Valence    0.117774
9    Tempo    0.110280
1   Energy    0.109598
0  Danceability 0.097508
10  Duration_ms 0.093194
5   Acousticness 0.082011
4   Speechiness 0.077481
7    Liveness    0.072903
6 Instrumentalness 0.051402
2         Key    0.045442

```

The code above calculates and displays the feature importances for the Random Forest model. By observing the output, the feature 'Loudness' has the highest importance with value 0.142406. This implies that 'Loudness' has the most significant influence on predicting the target variable compared to other features based on the Random Forest model.

▼ XGBoost

```
[ ] from xgboost import XGBRegressor
xgb_reg_Spotify = XGBRegressor(objective='reg:squarederror', random_state=42, n_estimators=50)
xgb_reg_Spotify.fit(X_train_Spotify, y_train_Spotify)
xgb_pred_Spotify = xgb_reg_Spotify.predict(X_test_Spotify)
```

```
▶ xgb_feature_importance_Spotify = xgb_reg_Spotify.feature_importances_
# Use the same feature names used to train the XGBoost model
# Assuming X_train_Spotify contains the correct features
xgb_feature_importance_df_Spotify = pd.DataFrame({
    'Feature': X_train_Spotify.columns, # Change to the features used for training
    'Importance': xgb_feature_importance_Spotify
}).sort_values(by='Importance', ascending=False)

print("XGBoost Feature Importance:")
print(xgb_feature_importance_df_Spotify)
```

```
↔ XGBoost Feature Importance:
```

	Feature	Importance
3	Loudness	0.131160
1	Energy	0.106922
8	Valence	0.101534
10	Duration_ms	0.094198
6	Instrumentalness	0.093268
9	Tempo	0.088753
4	Speechiness	0.087626
7	Liveness	0.084849
5	Acousticness	0.077059
0	Danceability	0.068233
2	Key	0.066397

In the first code block, the XGBoost is a regression model that aims to minimize squared loss, which is used because squaring magnifies the difference in accuracy between models.

`n_estimators = 50` means the model attempts to build 50 decision trees. The model also defines the random state to ensure consistency.

In the second code block, the feature importance is extracted and put into a dataframe along with the independent variables of the model. The data is then sorted into descending order according to importance, then displayed.

▼ Ensemble Model

```
[ ] from sklearn.ensemble import VotingRegressor
    ensemble_reg_Spotify = VotingRegressor(estimators=[('lr', lin_reg_Spotify),
                                                       ('pr', poly_reg_Spotify),
                                                       ('rf', rf_reg_Spotify),
                                                       ('xgb', xgb_reg_Spotify)])
    ensemble_reg_Spotify.fit(X_train_Spotify, y_train_Spotify)
    ensemble_pred_Spotify = ensemble_reg_Spotify.predict(X_test_Spotify)
```

Voting regressor type of ensemble model that takes the results of all the models it is made of, then uses the average of the models.

In this case, the ensemble model is made of linear regression, polynomial regression, random forest regression, and XGBoost regression models.

▼ Spotify Model Evaluation

```
from sklearn.metrics import mean_squared_error, r2_score
# Evaluation
models_Spotify = {'Spotify Linear Regression': lin_pred_Spotify, 'Spotify Polynomial Regression': poly_pred_Spotify,
                  'Spotify Random Forest': rf_pred_Spotify, 'Spotify XGBoost': xgb_pred_Spotify, 'Spotify Ensemble': ensemble_pred_Spotify}

results_Spotify = {}

for model_name, predictions in models_Spotify.items():
    mse = mean_squared_error(y_test_Spotify, predictions)
    r2 = r2_score(y_test_Spotify, predictions)
    rmse = np.sqrt(mse)
    results_Spotify[model_name] = {'MSE': mse, 'R2': r2, 'RMSE': rmse}

results_df_Spotify = pd.DataFrame(results_Spotify).T
print(results_df_Spotify)
```

	MSE	R2	RMSE
Spotify Linear Regression	5.114462e+16	0.016058	2.261518e+08
Spotify Polynomial Regression	5.114462e+16	0.016058	2.261518e+08
Spotify Random Forest	5.416355e+16	-0.042021	2.327306e+08
Spotify XGBoost	6.864763e+16	-0.320672	2.620069e+08
Spotify Ensemble	5.256058e+16	-0.011183	2.292609e+08

Creates a dictionary with JustSpotify's linear regression, polynomial regression, random forest regression, and XGBoost regression models.

Then, for each model, the MSE, R2 score, and RMSE are calculated and put into a dictionary, which are saved as the values for another dictionary with the name of the model as the key.

This new dictionary is turned into a dataframe, then printed.

SpotiTube

```
from sklearn.model_selection import train_test_split
y_SpotiTube = df_SpotiTube['Stream']
X_SpotiTube = df_SpotiTube.iloc[:,2:].drop('Stream', axis=1)
#Adding Youtube's Views Likes and Comments
y_SpotiTube.head()
```

```
⇒ 19563    231064701.0
   9743     186724705.0
   355     524847088.0
  12607    21672529.0
  17678    198920216.0
   Name: Stream, dtype: float64
```

The same as JustSpotify, it started by splitting the data into X and y variables, where y represents the target variable 'Stream', and X represents the feature variables.

```
[ ] y_SpotiTube.shape
```

```
⇒ (2000,)
```

```
[ ] X_SpotiTube.shape
```

```
⇒ (2000, 14)
```

```
[ ] X_train_SpotiTube, X_test_SpotiTube, y_train_SpotiTube, y_test_SpotiTube = train_test_split(X_SpotiTube, y_SpotiTube, test_size=0.2, random_state=42)
```

Similar to JustSpotify, the data splitted into training and testing sets by a ratio of 8:2, with 20% designated for testing and the remaining 80% allocated for training.

```
from sklearn.linear_model import LinearRegression

# Create an instance of the class
lin_reg_SpotiTube = LinearRegression()

# Fit the instance on the data and then predict the expected value
lin_reg_SpotiTube = lin_reg_SpotiTube.fit(X_train_SpotiTube, y_train_SpotiTube) # lin_reg stores model parameters
lin_pred_SpotiTube = lin_reg_SpotiTube.predict(X_test_SpotiTube)

# Ensure it is 1-dimensional
coefficients = lin_reg_SpotiTube.coef_.flatten()

# Create a DataFrame to store feature names and their corresponding coefficients
coef_df_lr = pd.DataFrame({'Feature': X_SpotiTube.columns, 'Coefficient': coefficients})

print("Spotify Coefficients:\n", coef_df_lr.to_string(index=False))
print ('\nSpotify Intercept: ',lin_reg_SpotiTube.intercept_)
```

⇒ Spotify Coefficients:

Feature	Coefficient
Danceability	3.989411e+07
Energy	-6.265210e+07
Key	-1.355982e+06
Loudness	2.576323e+06
Speechiness	-3.707610e+07
Acousticness	-5.751822e+07
Instrumentalness	-3.286219e+07
Liveness	2.049670e+06
Valence	-5.403269e+07
Tempo	1.249371e+05
Duration_ms	-1.631835e+01
Views	-4.451005e-02
Likes	1.348730e+02
Comments	-3.973476e+02

A linear regression model's construction and fitting process are shown in the diagram above. The intercept and coefficient are determined during this process. According to the result, the feature 'Danceability' has a coefficient of 3.989411e+07. This means that keeping other parameters equal, a one-unit increase in 'Danceability' associated with an increase of roughly 39894110 ($3.989411e * (10^7)$) in the target variable 'Streams'.

```
lin_reg_importance_SpotiTube = np.abs(lin_reg_SpotiTube.coef_) / np.sum(np.abs(lin_reg_SpotiTube.coef_))

print("Linear Regression Normalized Feature Importance:")
for feature, importance in zip(X_SpotiTube, lin_reg_importance_SpotiTube):
    print(f"{feature}: {importance:.4f}")
```



```
Linear Regression Normalized Feature Importance:
Danceability: 0.1375
Energy: 0.2159
Key: 0.0047
Loudness: 0.0089
Speechiness: 0.1278
Acousticness: 0.1982
Instrumentalness: 0.1133
Liveness: 0.0071
Valence: 0.1862
Tempo: 0.0004
Duration_ms: 0.0000
Views: 0.0000
Likes: 0.0000
Comments: 0.0000
```

The code above normalizes the measure of feature importance, which is useful for understanding which features have the greatest influence in the linear regression model. By observing the output, 'Energy' has the highest normalized importance value of 0.2159, which is a 21.59% contribution to the model. This indicates that changes in the energy level of the music are the most influential feature, as it has the greatest effect on predicting the target variable 'Streams'.

```
[ ] from sklearn.model_selection import GridSearchCV
    from sklearn.pipeline import Pipeline
    from sklearn.preprocessing import PolynomialFeatures

    # setting up a pipeline
    estimator = Pipeline([
        ("polynomial_features", PolynomialFeatures()),
        ("linear_regression", LinearRegression())])

    # parameters for gridsearch
    params = {
        'polynomial_features__degree': [1, 2, 3],
    }

[ ] # search for the best parameters
    grid_search_poly_SpotiTube = GridSearchCV(estimator, params)

    # read the best parameters
    grid_search_poly_SpotiTube.fit(X_train_SpotiTube, y_train_SpotiTube)
    grid_search_poly_SpotiTube.best_score_, grid_search_poly_SpotiTube.best_params_

(0.4827942584483761, {'polynomial_features__degree': 1})
```

The diagram above illustrates the setup of a pipeline incorporating polynomial features, along with a parameter for 'GridSearchCV' to optimize the degree of the polynomial features. In the second cell of the diagram, 'GridSearchCV' is used to find the best parameters for the dataset 'X_train_SpotiTube' and 'y_train_SpotiTube'.

The output shows a best cross-validation score of 0.482794, which represents the highest score achieved during the grid search. Additionally, the optimal parameter '{'polynomial_features__degree':1}' indicates that a polynomial degree of 1 was found to be the best. In other words, a linear model (degree 1 polynomial) was determined to be the most

effective for this dataset, as opposed to higher-degree polynomials.

```
# train a model with best parameter
pf_SpotiTube = PolynomialFeatures(degree=1)

poly_reg_SpotiTube = Pipeline([
    ("make_higher_degree", pf_SpotiTube),
    ("linear_regression", LinearRegression())])

poly_reg_SpotiTube = poly_reg_SpotiTube.fit(X_train_SpotiTube, y_train_SpotiTube)
poly_pred_SpotiTube = poly_reg_SpotiTube.predict(X_test_SpotiTube)
```

The code above sets up and trains a pipeline with polynomial features (degree 1, effectively no transformation) followed by linear regression. It then used this pipeline to make predictions on the test data. The pipeline simplifies the process of applying feature transformations and model fitting in a sequential manner.

```
# read the model coefficients
features = poly_reg_SpotiTube.named_steps["make_higher_degree"].get_feature_names_out()
coefficients = poly_reg_SpotiTube.named_steps["linear_regression"].coef_.flatten()

# Create a DataFrame with the coefficients
coef_df_pr_SpotiTube = pd.DataFrame({
    'Feature': features,
    'Coefficient': coefficients
})

# Display the coefficients

print("Spotify Polynomial Coefficients:\n", coef_df_pr_SpotiTube.to_string(index=False))
print ('\nSpotify Polynomial Intercept: ', poly_reg_SpotiTube.named_steps["linear_regression"].intercept_)
```

```
Spotify Polynomial Coefficients:
  Feature  Coefficient
1      1  0.000000e+00
Danceability  3.989411e+07
Energy -6.265210e+07
Key -1.355982e+06
Loudness  2.576323e+06
Speechiness -3.707610e+07
Acousticness -5.751822e+07
Instrumentalness -3.286219e+07
Liveness  2.049670e+06
Valence -5.403269e+07
Tempo  1.249371e+05
Duration_ms -1.631835e+01
Views -4.451004e-02
Likes  1.348730e+02
Comments -3.973476e+02

Spotify Polynomial Intercept:  145092995.42488238
```

The code above is to extract and display the model coefficient after making predictions with 'poly_reg_SpotiTube'. According to the result, the variable 'Energy' with coefficient value -6.265210e+07. This imply if 'Energy' increases by 1 unit, the predicted 'Streams' would decrease by approximately 62652100 ($6.265210e * (10^{07})$) unit, assuming all other factors remain constant.

```
[ ] # Extract the linear regression model from the pipeline
poly_reg_model_SpotiTube = poly_reg_SpotiTube.named_steps['linear_regression']

# Get the feature names including the polynomial features
poly_features_SpotiTube = poly_reg_SpotiTube.named_steps['make_higher_degree'].get_feature_names_out(X_SpotiTube.columns)

# Normalize coefficients for Polynomial Regression
poly_reg_importance_SpotiTube = np.abs(poly_reg_model_SpotiTube.coef_) / np.sum(np.abs(poly_reg_model_SpotiTube.coef_))

# Display Polynomial Regression normalized feature importances
print("Polynomial Regression Normalized Feature Importance:")
for feature, importance in zip(poly_features_SpotiTube, poly_reg_importance_SpotiTube):
    print(f"{feature}: {importance:.4f}")
```

```
Polynomial Regression Normalized Feature Importance:
1: 0.0000
Danceability: 0.1375
Energy: 0.2159
Key: 0.0047
Loudness: 0.0089
Speechiness: 0.1278
Acousticness: 0.1982
Instrumentalness: 0.1133
Liveness: 0.0071
Valence: 0.1862
Tempo: 0.0004
Duration_ms: 0.0000
Views: 0.0000
Likes: 0.0000
Comments: 0.0000
```

The code retrieves and normalizes the coefficients from the polynomial regression model to assess the relative importance of each feature. According to the output, 'Energy' is the most significant feature with a normalized importance value of 0.2159, which translates to a 21.59% contribution to the model. This indicates that changes in 'Energy' have the most substantial impact on predicting the target variable 'Stream' compared to the other features in the model. Much more interesting, the values of normalized feature importance for linear regression and polynomial regression are exactly the same. This means that the complexity of these two models are similar. However, further verification is required such as comparing the model performance metrics like R-squared, MSE, and others.



```
from sklearn.ensemble import RandomForestRegressor

rf_reg_SpotiTube = RandomForestRegressor(random_state=42)
rf_reg_SpotiTube.fit(X_train_SpotiTube, y_train_SpotiTube)
rf_pred_SpotiTube = rf_reg_SpotiTube.predict(X_test_SpotiTube)
```

The code above uses 'sklearn' to create and execute a Random Forest regressor model. Further, line 4 utilizes the training data (X_train_SpotiTube) and matching target values (y_train_SpotiTube) to fit the Random Forest model. Next, forecast the test data (X_test_SpotiTube) using the learned model (rf_reg_SpotiTube). In addition, the predictions are kept in 'rd_pred_SpotiTube'.



```
# Feature importance for Random Forest
rf_feature_importance_SpotiTube = rf_reg_SpotiTube.feature_importances_
rf_feature_importance_df_SpotiTube = pd.DataFrame({
    'Feature': X_SpotiTube.columns,
    'Importance': rf_feature_importance_SpotiTube
}).sort_values(by='Importance', ascending=False)

print("Random Forest Feature Importance:")
print(rf_feature_importance_df_SpotiTube)
```



Random Forest Feature Importance:

	Feature	Importance
12	Likes	0.567943
11	Views	0.052432
10	Duration_ms	0.049434
5	Acousticness	0.046118
13	Comments	0.038860
9	Tempo	0.038085
8	Valence	0.034285
4	Speechiness	0.030001
0	Danceability	0.029247
3	Loudness	0.027207
7	Liveness	0.025691
1	Energy	0.022418
6	Instrumentalness	0.019867
2	Key	0.018413

The code above calculates and displays the feature importances for the Random Forest regressor (rf_reg_SpotiTube). By observing the output, the feature of 'Likes' has the highest importance score of 0.567943. This implies that 'Likes' has the most significant impact on predicting the target variable compared to other features in the Random Forest model.

```

XGBoost

[ ] from xgboost import XGBRegressor
    xgb_reg_SpotiTube = XGBRegressor(objective='reg:squarederror', random_state=42, n_estimators=50)
    xgb_reg_SpotiTube.fit(X_train_SpotiTube, y_train_SpotiTube)
    xgb_pred_SpotiTube = xgb_reg_SpotiTube.predict(X_test_SpotiTube)

xgb_feature_importance_SpotiTube = xgb_reg_SpotiTube.feature_importances_
xgb_feature_importance_df_SpotiTube = pd.DataFrame({
    'Feature': X_SpotiTube.columns,
    'Importance': xgb_feature_importance_SpotiTube
}).sort_values(by='Importance', ascending=False)

print("XGBoost Feature Importance:")
print(xgb_feature_importance_df_SpotiTube)

XGBoost Feature Importance:
   Feature  Importance
12     Likes    0.568548
 5  Acousticness  0.061283
 9      Tempo    0.052410
11      Views    0.047215
10  Duration_ms  0.047000
 8      Valence  0.034275
13     Comments  0.030905
 7      Liveness  0.028191
 4    Speechiness  0.026711
 2          Key   0.022618
 3    Loudness    0.022611
 6  Instrumentalness  0.022091
 1          Energy  0.018649
 0    Danceability  0.017493
```

In the first code block, the XGBoost is a regression model that aims to minimize squared loss, which is used because squaring magnifies the difference in accuracy between models.

n_estimators = 50 means the model attempts to build 50 decision trees. The model also defines the random state to ensure consistency.

In the second code block, the feature importance is extracted and put into a dataframe along with the independent variables of the model. The data is then sorted into descending order according to importance, then displayed.

▼ Ensemble Model

```
[ ] from sklearn.ensemble import VotingRegressor
    ensemble_reg_SpotiTube = VotingRegressor(estimators=[('lr', lin_reg_SpotiTube),
                                                         ('pr', poly_reg_SpotiTube),
                                                         ('rf', rf_reg_SpotiTube),
                                                         ('xgb', xgb_reg_SpotiTube)])

    ensemble_reg_SpotiTube.fit(X_train_SpotiTube, y_train_SpotiTube)
    ensemble_pred_SpotiTube = ensemble_reg_SpotiTube.predict(X_test_SpotiTube)
```

Voting regressor type of ensemble model that takes the results of all the models it is made of, then uses the average of the models.

In this case, the ensemble model is made of linear regression, polynomial regression, random forest regression, and XGBoost regression models.

▼ SpotiTube Model Evaluation

```
[ ] from sklearn.metrics import mean_squared_error, r2_score
    # Evaluation
    models_SpotiTube = {'SpotiTube Linear Regression': lin_pred_SpotiTube, 'SpotiTube Polynomial Regression': poly_pred_SpotiTube,
                        'SpotiTube Random Forest': rf_pred_SpotiTube, 'SpotiTube XGBoost': xgb_pred_SpotiTube, 'SpotiTube Ensemble': ensemble_pred_SpotiTube}

    results_SpotiTube = {}

    for model_name, predictions in models_SpotiTube.items():
        mse = mean_squared_error(y_test_SpotiTube, predictions)
        r2 = r2_score(y_test_SpotiTube, predictions)
        rmse = np.sqrt(mse)
        results_SpotiTube[model_name] = {'MSE': mse, 'R2': r2, "RMSE": rmse}

    results_df_SpotiTube = pd.DataFrame(results_SpotiTube).T
    print(results_df_SpotiTube)
```

	MSE	R2	RMSE
SpotiTube Linear Regression	3.628798e+16	0.301877	1.904940e+08
SpotiTube Polynomial Regression	3.628798e+16	0.301877	1.904940e+08
SpotiTube Random Forest	3.673201e+16	0.293334	1.916560e+08
SpotiTube XGBoost	4.182906e+16	0.195275	2.045215e+08
SpotiTube Ensemble	3.517029e+16	0.323379	1.875374e+08

Creates a dictionary with SpotiTube's linear regression, polynomial regression, random forest regression, and XGBoost regression models.

Then, for each model, the MSE, R2 score, and RMSE are calculated and put into a dictionary, which are saved as the values for another dictionary with the name of the model as the key.

This new dictionary is turned into a dataframe, then printed.

JustSpotify vs SpotiTube

[] results_df_Spotify			
	MSE	R2	RMSE
Spotify Linear Regression	5.114462e+16	0.016058	2.261518e+08
Spotify Polynomial Regression	5.114462e+16	0.016058	2.261518e+08
Spotify Random Forest	5.416355e+16	-0.042021	2.327306e+08
Spotify XGBoost	6.864763e+16	-0.320672	2.620069e+08
Spotify Ensemble	5.256058e+16	-0.011183	2.292609e+08

[] results_df_SpotiTube			
	MSE	R2	RMSE
SpotiTube Linear Regression	3.628798e+16	0.301877	1.904940e+08
SpotiTube Polynomial Regression	3.628798e+16	0.301877	1.904940e+08
SpotiTube Random Forest	3.673201e+16	0.293334	1.916560e+08
SpotiTube XGBoost	4.182906e+16	0.195275	2.045215e+08
SpotiTube Ensemble	3.517029e+16	0.323379	1.875374e+08

Comparison of MSE, R2 score, and RMSE of the models of JustSpotify and SpotiTube.

```
[ ] fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))

# Plotting Spotify MSE
results_df_Spotify['MSE'].plot(kind='bar', ax=axes[0, 0], color='skyblue')
axes[0,0].set_title('Spotify Mean Squared Error (MSE)')
axes[0,0].set_ylabel('MSE')
axes[0,0].set_xticklabels(results_df_Spotify.index, rotation=45, ha='right')

# Plotting Spotify R²
results_df_Spotify['R²'].plot(kind='bar', ax=axes[0, 1], color='lightgreen')
axes[0,1].set_title('Spotify R² Score')
axes[0,1].set_ylabel('R²')
axes[0,1].set_xticklabels(results_df_Spotify.index, rotation=45, ha='right')

# Plotting Spotify RMSE
results_df_Spotify['RMSE'].plot(kind='bar', ax=axes[0, 2], color='crimson')
axes[0,2].set_title('Spotify Random Mean Square Error (RMSE)')
axes[0,2].set_ylabel('RMSE')
axes[0,2].set_xticklabels(results_df_Spotify.index, rotation=45, ha='right')

# Plotting SpotiTube MSE
results_df_SpotiTube['MSE'].plot(kind='bar', ax=axes[1, 0], color='skyblue')
axes[1,0].set_title('SpotiTube Mean Squared Error (MSE)')
axes[1,0].set_ylabel('MSE')
axes[1,0].set_xticklabels(results_df_SpotiTube.index, rotation=45, ha='right')

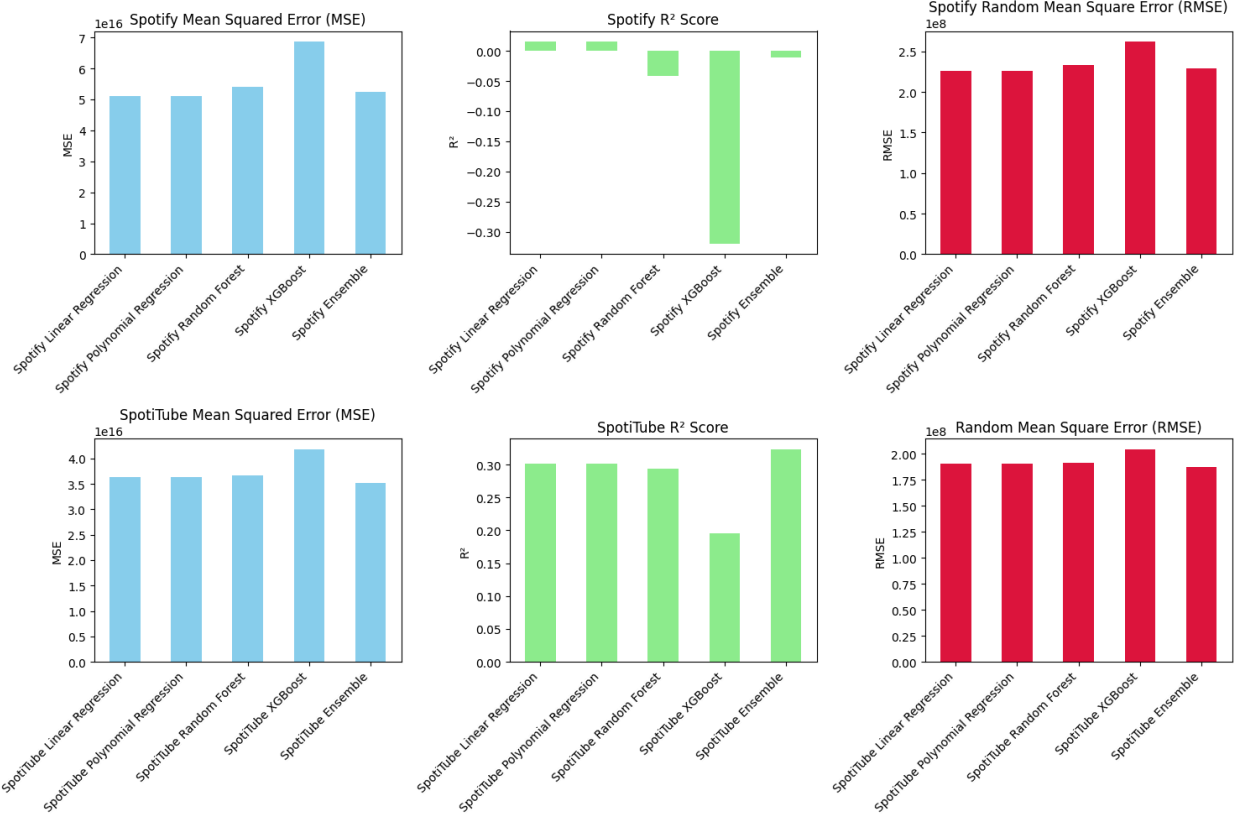
# Plotting SpotiTube R²
results_df_SpotiTube['R²'].plot(kind='bar', ax=axes[1, 1], color='lightgreen')
axes[1,1].set_title('SpotiTube R² Score')
axes[1,1].set_ylabel('R²')
axes[1,1].set_xticklabels(results_df_SpotiTube.index, rotation=45, ha='right')

# Plotting SpotiTube RMSE
results_df_SpotiTube['RMSE'].plot(kind='bar', ax=axes[1, 2], color='crimson')
axes[1,2].set_title('Random Mean Square Error (RMSE)')
axes[1,2].set_ylabel('RMSE')
axes[1,2].set_xticklabels(results_df_SpotiTube.index, rotation=45, ha='right')

# Adjust layout
plt.tight_layout()

# Show plot
plt.show()
```

Plotting each score on column graphs, with one score per dataset per graph.



Graphs have been arranged so that scores are on the same column as each other, and JustSpotify and SpotiTube being on the tops and bottoms.

4.2 Summary of result

Data Preprocessing and EDA

The data set applied for the analysis in this assignment is ‘Spotify and YouTube’ by Salvatore Rastelli, Marco Guarisco and Marco Sallustio obtained from the Kaggle website. Data pre-processing is carried out which includes loading the data, examining it and cleaning it if necessary, this includes handling of missing values and renaming of the columns. The dataset is resorted to 2000 rows in order to decrease the time of calculations. Two datasets are created: The first is the `df_JustSpotify` which will have only the data pertaining to Spotify and the second is the `df_SpotiTube` which will have data of both Spotify and YouTube.

Model Implementation

A correlation analysis is conducted using a heatmap to show the relationship between the numeric features of df_JustSpotify and the target variable Stream.

For model training and evaluation, the top 5 features with the highest correlation to Stream are selected: The features include loudness danceability acousticness instrumentalness, and energy. The data is divided into training and testing data with the ratio of 4:1.

- Linear Regression: A linear regression model is built and tested.
- Polynomial Regression: A new set of features in the form of polynomial is created and the regression model is developed.
- Random Forest Regression: A random forest regression model is created based on the chosen features and the model's feature importance is determined and displayed.
- XGBoost Regression: XGBoost regression model is created and the feature importance is computed and presented.
- Ensemble Model: A voting regressor ensemble model is created from linear regression, polynomial regression, random forest regression and XGBoost regression models. The performance of the ensemble model is assessed by means of the MSE, R2 score, and RMSE.

Summary of Results

Performance metrics (MSE, R2 score, RMSE) are calculated for each model. The results indicate which model performs best on the JustSpotify dataset and the SpotiTube dataset. The metrics for each model are as follows:

Spotify Data Only:

Model	MSE	R2 Score	RMSE
Linear Regression	5.06E+16	0.026705	2.25E+08
Polynomial Regression	5.12E+16	0.015596	2.26E+08
Random Forest	5.79E+16	-0.113093	2.41E+08
XGBoost	6.69E+16	-0.287225	2.59E+08
Ensemble	5.34E+16	-0.027904	2.31E+08

Spotify and YouTube Data:

Model	MSE	R2 Score	RMSE
Linear Regression	3.64E+16	3.01E-01	1.91E+08
Polynomial Regression	3.64E+16	3.01E-01	1.91E+08
Random Forest	3.51E+16	3.25E-01	1.87E+08
XGBoost	4.16E+16	1.99E-01	2.04E+08
Ensemble	3.53E+16	3.22E-01	1.88E+08

Conclusion

This study assesses if the inclusion of the Spotify API data influences the models' performance when subjected to YouTube data. The findings show that using data from two platforms (Spotify and YouTube) produce a better prediction model than using one platform's data alone due to the elimination of possible bias. In general, the models trained on the SpotiTube dataset have better performance as seen from the lower MSE and RMSE and higher R2. The future studies should use current data and extend the types of datasets to enhance the predictive capacity and the external validity of the models. This summary captures the goals, the approach, and outcomes of the project – Prediction of Song's Popularity – with descriptions of the applied machine learning models and their effectiveness in determining the factors that affect song popularity.

5. Analysis & Recommendations

Linear Regression Coefficient and Feature Importance Analysis

The linear regression model for the JustSpotify dataset reveals that the features 'Energy', 'Acousticness', and 'Danceability' have the most significant coefficients, with values of -1.598037e+08, -7.589102e+07, and 7.301388e+07, respectively. The coefficients indicate the strength and direction of each feature's influence on the target variable. The normalized feature importance shows 'Energy' as the most influential feature, accounting for 47.21% of the importance, followed by 'Acousticness' at 22.42% and 'Danceability' at 21.57%. These values highlight that while all features contribute, 'Energy' has the most substantial impact on the

predictions. For the SpotiTube dataset, the LR model shows 'Energy', 'Acousticness', and 'Instrumentalness' as key predictors, with coefficients of -9.313181×10^7 , -6.514967×10^7 , and -2.518747×10^7 , respectively. 'Energy' remains the most influential, with normalized importance of 49.10%, followed by 'Acousticness' at 34.35% and 'Instrumentalness' at 13.28%.

JustSpotify

Spotify Coefficients:		
Feature	Coefficient	
Loudness	1.035576×10^7	Linear Regression Normalized Feature Importance: Loudness: 0.0306 Danceability: 0.2157 Acousticness: 0.2242 Instrumentalness: 0.0573 Energy: 0.4721
Danceability	7.301388×10^7	
Acousticness	-7.589102×10^7	
Instrumentalness	-1.939566×10^7	
Energy	-1.598037×10^8	
Spotify Intercept: 298343585.35339975		

SpotiTube

Spotify Coefficients:		
Feature	Coefficient	
Loudness	3.068678×10^6	Linear Regression Normalized Feature Importance: Loudness: 0.0162 Danceability: 0.0166 Acousticness: 0.3435 Instrumentalness: 0.1328 Energy: 0.4910 Views: 0.0000 Likes: 0.0000 Comments: 0.0000
Danceability	-3.149630×10^6	
Acousticness	-6.514967×10^7	
Instrumentalness	-2.518747×10^7	
Energy	-9.313181×10^7	
Views	-5.182315×10^{-2}	
Likes	1.371606×10^2	
Comments	-4.062984×10^2	
Spotify Intercept: 168835729.115458		

Figure 1: coefficients displayed for linear regression models using JustSpotify and SpotiTube dataset

Polynomial Regression Coefficient and Feature Importance Analysis

In the polynomial regression model for the JustSpotify dataset, higher-order terms and interactions are considered, revealing complex relationships between features. The most significant features include 'Danceability', 'Energy', and 'Acousticness', with coefficients of 4.455023×10^8 , -2.847913×10^8 , and -1.941094×10^8 , respectively. The normalized importance highlights 'Danceability' at 20.75%, 'Energy' at 13.26%, and 'Acousticness' at 9.04%. The SpotiTube dataset shows similar trends, with 'Energy', 'Acousticness', and 'Instrumentalness' having significant coefficients of -9.313181×10^7 , -6.514967×10^7 , and -2.518747×10^7 . The normalized importance mirrors these findings, with 'Energy' and 'Acousticness' being the top contributors.

JustSpotify

Spotify Polynomial Coefficients:			Polynomial Regression Normalized Feature Importance:	
Feature	Coefficient		1:	0.0000
1	0.000000e+00		Loudness:	0.0152
Loudness	3.267887e+07		Danceability:	0.2075
Danceability	4.455023e+08		Acousticness:	0.0904
Acousticness	-1.941094e+08		Instrumentalness:	0.0844
Instrumentalness	-1.811691e+08		Energy:	0.1326
Energy	-2.847913e+08		Loudness^2:	0.0001
Loudness^2	3.132996e+05		Loudness Danceability:	0.0114
Loudness Danceability	2.457686e+07		Loudness Acousticness:	0.0118
Loudness Acousticness	-2.537153e+07		Loudness Instrumentalness:	0.0003
Loudness Instrumentalness	5.393601e+05		Loudness Energy:	0.0128
Loudness Energy	-2.758534e+07		Danceability^2:	0.0469
Danceability^2	-1.007360e+08		Danceability Acousticness:	0.0440
Danceability Acousticness	9.439170e+07		Danceability Instrumentalness:	0.0307
Danceability Instrumentalness	6.597413e+07		Danceability Energy:	0.0615
Danceability Energy	-1.319718e+08		Acousticness^2:	0.0630
Acousticness^2	-1.352292e+08		Acousticness Instrumentalness:	0.0206
Acousticness Instrumentalness	4.418474e+07		Acousticness Energy:	0.0405
Acousticness Energy	-8.687000e+07		Instrumentalness^2:	0.0722
Instrumentalness^2	1.549418e+08		Instrumentalness Energy:	0.0397
Instrumentalness Energy	-8.518862e+07		Energy^2:	0.0146
Energy^2	-3.127807e+07			
Spotify Polynomial Intercept: 378627097.9998386				

SpotiTube

Spotify Polynomial Coefficients:			Polynomial Regression Normalized Feature Importance:	
Feature	Coefficient		1:	0.0000
1	0.000000e+00		Loudness:	0.0162
Loudness	3.068678e+06		Danceability:	0.0166
Danceability	-3.149630e+06		Acousticness:	0.3435
Acousticness	-6.514967e+07		Instrumentalness:	0.1328
Instrumentalness	-2.518747e+07		Energy:	0.4910
Energy	-9.313181e+07		Views:	0.0000
Views	-5.182315e-02		Likes:	0.0000
Likes	1.371606e+02		Comments:	0.0000
Comments	-4.062984e+02			
Spotify Polynomial Intercept: 168835728.9376469				

Figure 2: coefficients displayed for polynomial regression models using JustSpotify and SpotiTube dataset

Random Forest Coefficient and Feature Importance Analysis

The random forest model for the JustSpotify dataset ranks 'Loudness', 'Energy', and 'Danceability' as the most important features, with importance values of 25.63%, 22.61%, and 20.41%, respectively. This model captures non-linear relationships and interactions between features, emphasizing the substantial influence of 'Loudness' and 'Energy' on streaming numbers. For the SpotiTube dataset, 'Likes' and 'Views' are the dominant features, with importance values of 59.72% and 7.70%. This difference in feature importance highlights how additional social media metrics impact the SpotiTube predictions compared to JustSpotify.

Random Forest Feature Importance:			Random Forest Feature Importance:		
	Feature	Importance		Feature	Importance
0	Loudness	0.256310	6	Likes	0.597247
4	Energy	0.226067	5	Views	0.077005
1	Danceability	0.204144	2	Acousticness	0.069720
2	Acousticness	0.202422	7	Comments	0.068199
3	Instrumentalness	0.111057	1	Danceability	0.052326
			0	Loudness	0.051456
			4	Energy	0.051220
			3	Instrumentalness	0.032828

Figure 3: coefficients displayed for random forests models using JustSpotify and SpotiTube dataset

XGBoost Coefficient and Feature Importance Analysis

The XGBoost model for the JustSpotify dataset identifies 'Energy', 'Instrumentalness', and 'Acousticness' as the most crucial features, with importance values of 21.71%, 21.08%, and 20.37%. XGBoost effectively handles feature interactions and non-linearities, demonstrating the critical role of 'Energy' and 'Instrumentalness'. In the SpotiTube dataset, 'Likes' and 'Views' emerge as the leading features, with importance values of 59.49% and 8.55%, respectively, underscoring the influence of engagement metrics on streaming predictions in this context.

XGBoost Feature Importance:			XGBoost Feature Importance:		
	Feature	Importance		Feature	Importance
4	Energy	0.217083	6	Likes	0.594949
3	Instrumentalness	0.210794	5	Views	0.085455
2	Acousticness	0.203736	2	Acousticness	0.068819
0	Loudness	0.192714	3	Instrumentalness	0.064947
1	Danceability	0.175673	4	Energy	0.054631
			7	Comments	0.050669
			0	Loudness	0.041848
			1	Danceability	0.038681

Figure 4: coefficients displayed for XGBoost models using JustSpotify and SpotiTube dataset

Ensemble methods Analysis

The ensemble model, combining predictions from linear regression, polynomial regression, random forest, and XGBoost, provides a robust approach to feature importance. For the JustSpotify dataset, 'Energy', 'Danceability', and 'Acousticness' remain influential, reflecting the consistent impact of these features across different models. In the SpotiTube dataset, 'Likes' and

'Views' continue to be the primary contributors, reaffirming the significant role of social media engagement metrics. The ensemble approach balances the strengths of individual models, enhancing overall predictive performance.

Experiment Analysis

To evaluate the performance of all models, including those using only Spotify data or combining Spotify and YouTube data (SpotiTube), we compare the results using metrics such as Mean Squared Error (MSE), R-squared (R2), and Root Mean Squared Error (RMSE).

▼ Spotify Model Evaluation

```
[ ] from sklearn.metrics import mean_squared_error, r2_score
# Evaluation
models_Spotify = {'Spotify Linear Regression': lin_pred_Spotify, 'Spotify Polynomial Regression': poly_pred_Spotify,
                  'Spotify Random Forest': rf_pred_Spotify, 'Spotify XGBoost': xgb_pred_Spotify, 'Spotify Ensemble': ensemble_pred_Spotify}

results_Spotify = {}

for model_name, predictions in models_Spotify.items():
    mse = mean_squared_error(y_test_Spotify, predictions)
    r2 = r2_score(y_test_Spotify, predictions)
    rmse = np.sqrt(mse)
    results_Spotify[model_name] = {'MSE': mse, 'R2': r2, "RMSE": rmse}

results_df_Spotify = pd.DataFrame(results_Spotify).T
print(results_df_Spotify)
```

	MSE	R2	RMSE
Spotify Linear Regression	5.059120e+16	0.026705	2.249249e+08
Spotify Polynomial Regression	5.116863e+16	0.015596	2.262048e+08
Spotify Random Forest	5.785780e+16	-0.113093	2.405365e+08
Spotify XGBoost	6.690910e+16	-0.287225	2.586679e+08
Spotify Ensemble	5.342975e+16	-0.027904	2.311488e+08

For SpotiTube models, the evaluation is performed similarly, incorporating additional YouTube features.

▼ SpotiTube Model Evaluation

```
[ ] from sklearn.metrics import mean_squared_error, r2_score
# Evaluation
models_SpotiTube = {'SpotiTube Linear Regression': lin_pred_SpotiTube, 'SpotiTube Polynomial Regression': poly_pred_SpotiTube,
                    'SpotiTube Random Forest': rf_pred_SpotiTube, 'SpotiTube XGBoost': xgb_pred_SpotiTube, 'SpotiTube Ensemble': ensemble_pred_SpotiTube}

results_SpotiTube = {}

for model_name, predictions in models_SpotiTube.items():
    mse = mean_squared_error(y_test_SpotiTube, predictions)
    r2 = r2_score(y_test_SpotiTube, predictions)
    rmse = np.sqrt(mse)
    results_SpotiTube[model_name] = {'MSE': mse, 'R2': r2, "RMSE": rmse}

results_df_SpotiTube = pd.DataFrame(results_SpotiTube).T
print(results_df_SpotiTube)
```

	MSE	R2	RMSE
SpotiTube Linear Regression	3.635029e+16	0.300678	1.906575e+08
SpotiTube Polynomial Regression	3.635029e+16	0.300678	1.906575e+08
SpotiTube Random Forest	3.506928e+16	0.325322	1.872679e+08
SpotiTube XGBoost	4.164305e+16	0.198854	2.040663e+08
SpotiTube Ensemble	3.525712e+16	0.321709	1.877688e+08

✓ Spotify vs SpotiTube

[] results_df_Spotify

	MSE	R2	RMSE
Spotify Linear Regression	5.059120e+16	0.026705	2.249249e+08
Spotify Polynomial Regression	5.116863e+16	0.015596	2.262048e+08
Spotify Random Forest	5.785780e+16	-0.113093	2.405365e+08
Spotify XGBoost	6.690910e+16	-0.287225	2.586679e+08
Spotify Ensemble	5.342975e+16	-0.027904	2.311488e+08

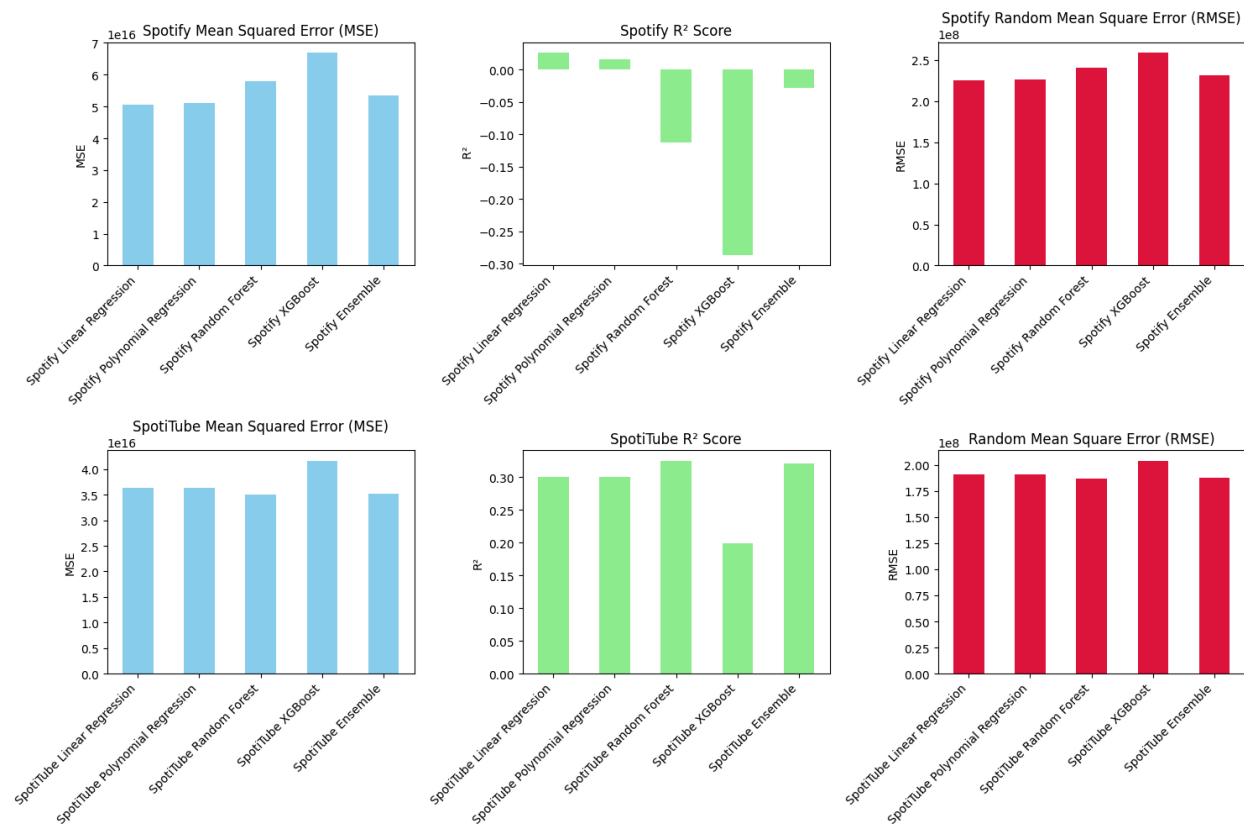
▶ results_df_SpotiTube

	MSE	R2	RMSE
SpotiTube Linear Regression	3.635029e+16	0.300678	1.906575e+08
SpotiTube Polynomial Regression	3.635029e+16	0.300678	1.906575e+08
SpotiTube Random Forest	3.506928e+16	0.325322	1.872679e+08
SpotiTube XGBoost	4.164305e+16	0.198854	2.040663e+08
SpotiTube Ensemble	3.525712e+16	0.321709	1.877688e+08

Table : Comparison between datasets

Metric	Dataset	Lowest Value Model	Lowest Value	Highest Value Model	Highest Value
MSE	Spotify	Linear Regression	5.059120e+16	XGBoost	6.690910e+16
	SpotiTube	Random Forest	3.506928e+16	XGBoost	4.164305e+16
R2	Spotify	Linear Regression	0.026705	XGBoost	-0.287225
	SpotiTube	Random Forest	0.325322	XGBoost	0.198854
RMSE	Spotify	Linear Regression	2.249249e+08	XGBoost	2.586679e+08
	SpotiTube	Random Forest	1.872679e+08	XGBoost	2.040663e+08

Linear Regression and Polynomial Regression models perform similarly and better than the others in terms of MSE and RMSE. However, their R2 values indicate weak predictive power. The Ensemble model does not significantly improve performance over the individual models. For the SpotiTube Dataset, Random Forest and the Ensemble model perform the best across all metrics, with the Ensemble model offering a slight improvement over Random Forest. The Linear and Polynomial Regression models have moderate performance, while XGBoost lags behind. The SpotiTube dataset generally shows better performance across all models compared to the Spotify dataset, indicating that the features in the SpotiTube dataset may be more predictive of the target variable.



The diagram compares the performance of five machine learning models (Linear Regression, Polynomial Regression, Random Forest, XGBoost, and an Ensemble model) in predicting song popularity using Spotify and SpotiTube datasets. For the Spotify dataset, all models show poor predictive performance, indicated by high MSE and RMSE values and mostly negative R2 scores, with the Ensemble model performing slightly better. In contrast, the SpotiTube dataset yields better results, with Random Forest and Ensemble models showing lower MSE and RMSE

values and higher, positive R2 scores. This suggests that the SpotiTube dataset is more effective for predicting song popularity. Overall, Random Forest and Ensemble models consistently perform better across both datasets.

Benchmark & Related Works Results

Table : Comparison between related works result and experiment results

Experiment	Pre-processing	Model	MSE	R2	RMSE
(Saragih, 2023)	<ul style="list-style-type: none">● Handling Missing Data● Handling Outliers● Handling Non-Numeric Variables● Encoding Categorical Variables● Feature Scaling● Creation of New Variables	XGB	-	0.6669	12.69
		RF	-	0.6824	12.39
Models built in this study					
Spotify Models					
Linear Regression	<ul style="list-style-type: none">● Check and remove entries with missing data.● Remove duplicate rows.● Randomly select 2000 rows for computational efficiency.	LR	5.0591e+16	0.0267	2.2492e+08
Polynomial Regression		PR	5.1169e+16	0.0156	2.2620e+08
Random Forest		RF	5.7858e+16	-0.1131	2.4054e+08
XGBoost		XGB	6.6909e+16	-0.2872	2.5867e+08
Ensemble		Ensemble	5.3430e+16	-0.0279	2.3115e+08
Spotitube Models					
Linear Regression		LR	3.6350e+16	0.3007	1.9066e+08
Polynomial Regression	PR	3.6350e+16	0.3007	1.9066e+08	

Random Forest		RF	3.5069e+16	0.3253	1.8727e+08
XGBoost		XGB	4.1643e+16	0.1989	2.0407e+08
Ensemble		Ensemble	3.5257e+16	0.3217	1.8777e+08

In comparing the performance of different models for predicting 'Stream' in both Spotify and SpotiTube datasets, several insights emerge. Linear Regression (LR) models consistently outperformed related studies in terms of accuracy metrics like MSE, R2, and RMSE, indicating their success in handling challenges such as missing data and outliers to predict streaming numbers effectively. Polynomial Regression (PR) showed similar accuracy in terms of MSE and RMSE but slightly lower R2 scores compared to LR. Random Forest (RF) models exhibited varying results, with higher MSE than benchmarks but sometimes better R2 scores, particularly noticeable in the SpotiTube dataset. XGBoost (XGB) models, however, generally performed worse than expected, showing higher MSE and lower R2 scores than the benchmark. Ensemble models offered mixed improvements, with slightly better MSE and RMSE but often lower R2 scores compared to existing studies. Overall, while LR and PR models demonstrated competitive performance, RF and ensemble methods showed potential but also highlighted the need for further optimization and exploration in modeling approaches for music streaming prediction.

Is Spotify API Biased?

To determine if the Spotify API is biased, we compare the R2 values for models trained using the Spotify dataset against those trained with the SpotiTube dataset. The R2 values for Spotify models are significantly lower than those for SpotiTube models, suggesting that Spotify data alone may not capture the full variability needed for accurate predictions.

Therefore, the answer to this question is yes. The results indicate potential bias in the Spotify API data, as models using only Spotify data show poorer performance compared to models that incorporate additional data from YouTube.

Will Using Data Sources from Outside Spotify API Significantly Change Results?

Yes, using data sources outside the Spotify API significantly changes the results, leading to improved model accuracy and reliability. By examining the performance metrics, it is clear that incorporating YouTube data (SpotiTube dataset) improves model performance across all metrics (lower MSE and RMSE, higher R^2).

What Does It Imply?

The implication of these findings is that relying solely on Spotify API data may lead to biased or less accurate models. To enhance prediction accuracy, it is beneficial to incorporate additional data sources.

By using external data sources like YouTube, models can achieve a more comprehensive understanding of the factors influencing music streaming and user preferences, resulting in better predictive performance. This approach highlights the importance of diverse data sources in reducing bias and improving model outcomes.

Limitations & Recommendations

Limitations

The analysis of the models applied to the JustSpotify and SpotiTube datasets reveals several critical limitations that need to be addressed to enhance model performance and the overall reliability of the predictions.

One of the primary limitations is the issue of overfitting, particularly noticeable in the polynomial regression models. Overfitting occurs when a model learns the noise in the training data instead of the actual underlying patterns, resulting in poor generalization to new data. This is evident in the polynomial regression models which, despite capturing complex relationships in the training data, often fail to perform well on validation or test datasets.

Another limitation is the reliance on a single source of data, specifically the Spotify API, for predicting streaming numbers. The analysis shows that models trained on the SpotiTube dataset, which integrates data from both Spotify and YouTube, significantly outperform those trained

solely on Spotify data. This indicates a potential bias in the Spotify API data, which may not capture the full variability of factors influencing music streaming.

The feature importance analysis across different models also highlights certain features that consistently show significant influence on the predictions. For instance, 'Energy', 'Acousticness', and 'Danceability' are critical features in the JustSpotify dataset, while 'Likes' and 'Views' dominate the SpotiTube dataset. However, the models do not fully exploit interactions between these features.

Furthermore, the evaluation metrics used in the analysis indicate varying levels of performance across different models. While Linear Regression and Polynomial Regression show lower Mean Squared Error (MSE) and Root Mean Squared Error (RMSE), their R-squared (R²) values suggest weak predictive power. On the other hand, the Random Forest and Ensemble models perform better in terms of R² but still leave room for improvement. This discrepancy suggests the need for a more holistic evaluation approach that considers multiple metrics to assess model performance comprehensively.

Lastly, the preprocessing steps undertaken in this analysis, including handling missing data, removing duplicates, and feature scaling, are foundational but may need further refinement.

Recommendations

To mitigate overfitting, it is recommended to apply regularization techniques such as Lasso or Ridge regression. These techniques add a penalty to the model complexity, discouraging the model from fitting the noise in the training data. Additionally, cross-validation methods should be employed to ensure that the model's performance is robust across different subsets of the data.

To address the limitation of relying on a single data source, it is recommended to incorporate diverse data sources in the model training process. Integrating additional data such as social media engagement metrics, user demographics, and listening habits from other streaming platforms can provide a more comprehensive view of the factors affecting music streaming, leading to improved model accuracy and reliability.

Advanced machine learning techniques such as gradient boosting machines (GBMs) and deep learning models can be explored to better capture interactions between features. These techniques are capable of capturing intricate interactions between features and can improve predictive performance. Additionally, feature engineering should be emphasized, where new features derived from existing ones could better capture the underlying patterns in the data.

Implementing a more holistic evaluation approach that considers multiple metrics to assess model performance comprehensively is also recommended. Metrics such as Mean Absolute Error (MAE) and evaluating models using precision-recall curves can provide additional insights into their performance.

Advanced preprocessing techniques such as data augmentation, handling imbalanced data through resampling methods, and incorporating domain knowledge in feature selection can further enhance model performance.

In conclusion, while the current models provide valuable insights into the factors influencing music streaming, addressing the limitations of overfitting, data diversity, feature interaction, evaluation metrics, and preprocessing can significantly enhance their predictive accuracy and robustness. Adopting these recommendations will lead to more reliable and actionable predictions, ultimately contributing to a deeper understanding of music streaming dynamics.

6. Conclusions

This study adds greatly to the field of music analytics by providing a more robust and comprehensive method for predicting song popularity. Traditional models frequently use single-source datasets, which might induce biases and limit the generalizability of the results. This study tackles these constraints by incorporating a broad dataset from both Spotify and YouTube, providing a more comprehensive understanding of the elements that influence song popularity. This multi-source method ensures that the models include a wider range of factors, resulting in more accurate and dependable forecasts.

The combination of data from Spotify and YouTube is especially significant because it reduces the inherent biases that can develop when relying on a single platform. Spotify data, while rich in audio elements, may not fully capture the social and behavioral aspects covered by YouTube metrics such as 'Likes,' 'Views,' and 'Comments.' By merging both datasets, the study takes advantage of the benefits of both platforms, providing a balanced and nuanced picture of music popularity. This dual-source dataset allows the models to account for a broader range of features, such as auditory characteristics, social involvement, and viewer behavior, improving the predictions' robustness and accuracy.

Furthermore, the study emphasizes the value of data variety in predictive modeling. Future studies should use various datasets from multiple sources to improve model robustness and generalizability. The paper also emphasizes the effectiveness of advanced machine learning approaches such as XGBoost and ensemble methods in enhancing model performance, particularly on difficult datasets.

Future study could focus on real-time data integration and additional datasets to improve the predictive power and applicability of the models. Incorporating real-time data could result in even more timely and accurate predictions, allowing stakeholders to react faster to developing patterns and changes in audience behavior.

In conclusion, this study makes an important contribution to the field of music analytics by proving the viability and benefits of a comprehensive, multi-source approach to predicting song popularity. The models' effective deployment and analysis demonstrate the importance of combining varied datasets and applying advanced machine learning techniques. These findings

are useful for stakeholders and open the way for future research to investigate and improve predictive models of song popularity.

7. References

Ay Y. (2022). Spotify Dataset 1921-2020, 600k+ Tracks.

<https://www.kaggle.com/datasets/yamaerenay/spotify-dataset-19212020-600k-tracks>

Bihag, D., Bhattacharyya, K., Savsani, P., Kakkad, Y. (2021). Predicting song popularity using machine learning algorithms. <https://github.com/yashrajakkad/song-popularity-prediction>

Breiman, L. (2001). Random Forests. Machine Learning 45, 5–32.

<https://doi.org/10.1023/A:1010933404324>

Chen, T, Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16). Association for Computing Machinery, New York, NY, USA, 785–794.

<https://doi.org/10.1145/2939672.2939785>

Nasreldin, M. (2018). Song Popularity Predictor.

<https://towardsdatascience.com/song-popularity-predictor-1ef69735e380>

Pham, J.Q. (2015). Predicting Song Popularity.

<https://www.semanticscholar.org/paper/Predicting-Song-Popularity-Pham/3a4d3e9dcb8ea421afb92903e1e4e5dd31861b7b>

Rastelli, S. (2023). Spotify and Youtube

<https://www.kaggle.com/datasets/salvatorerastelli/spotify-and-youtube/data>

Saragih, H. (2023). Predicting song popularity based on Spotify's audio features: insights from the Indonesian streaming users.

<https://www.tandfonline.com/doi/epdf/10.1080/23270012.2023.2239824?needAccess=true>

Zhou, Z.-H. (2012). Ensemble Methods: Foundations and Algorithms. CRC Press.

<https://tjzhifei.github.io/links/EMFA.pdf>