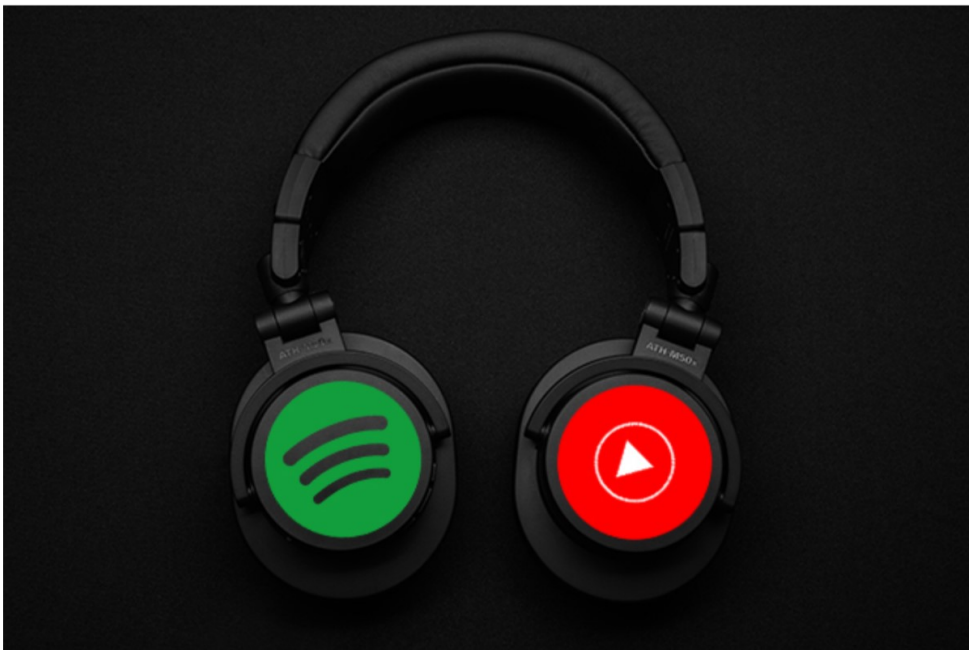
Predicting Track Popularity on Spotify and YouTube: A Machine Learning Approach



Ximena Moure
Ange Xu
ximena.moure@estudiantant.upc.edu
ange.xu@estudiantant.upc.edu
Master of Data Science - Machine Learning

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

June 12, 2023

Contents

1 Introduction

On one hand, Spotify is one of the largest music platforms in the world with more than 400 million registered users and a song catalog of more than 100 million songs [1] . Every day, more than 60000 songs are added to Spotify [2]. Artists from all over the world try to make it and have a hit song. On another hand, YouTube Music is a music streaming service that offers a wide variety of music content, including official music videos, live performances, covers, remixes, and more. It offers a variety of playlists, including curated playlists, mood-based playlists, and genre-specific playlists.

For this project, the dataset Spotify-Youtube. was chosen to gain insights into how people consume music across different platforms and also try to identify potential areas of improvement in the music industry. It contains information about songs of different artists and for each song it contains several statistics of the song in spotify and the number of views of the official music video in youtube. It has 20718 records and 28 explanatory variables.

The objective of this project is to analyze the dataset and uncover patterns within it, with the aim of predicting the number of streams a song will receive. To achieve this, various machine learning algorithms were employed.

2 Related previous work

The majority of the existing prevous work we encountered utilize a separate Spotify dataset that exclusively focuses on Spotify songs and includes a variable called Genre. The primary objective of the studies conducted on this alternate dataset is to employ various classification techniques to predict the genre of a song.
In the link we provided of our dataset there are some notebooks that analyze it, but the majority of them primarly focus on exploratory analysis.

3 Data Exploration process

We first conducted an initial exploration and then we proceeded with various preprocessing tasks. Pre-processing plays a vital role in our work as it can greatly impact our outcomes. Let's examine our strategy for managing the dataset during the pre-processing stage.

3.1 Initial exploration

As an initial step, we performed data exploration to familiarize ourselves with the dataset and to uncover meaningful insights from it.

We started by just looking at the amount of missing values, minimum and maximum values and other basic data inspection. We did not find values that were out of range. For instance, stream has very large values but this is common since this variable refers to the number of times a song is reproduced in Spotify and this can be very big. As for null values, there is not a big amount of them. The variable with the most null values is Description which is not an important one sine it is just test describing the video and we will not use it for our models.

Then, we examined the correlation matrix to identify the variables with the highest correlation. In Figure 1 the matrix is displayed and in 1 the most correlated variables with Stream are

displyed. The ones that exhibited the strongest positive correlation with our target variable (Stream) were Views and Likes. This finding was anticipated, as songs with a large number of streams typically have high view counts and receive numerous likes on their videos Additionally, we observed a significant correlation between Loudness and Energy, which was also expected. Conversely, we noted that songs with a higher degree of acoustic elements tend to posses lower energy levels, which aligns with logical expectations.

Subsequently, we looked at the distribution plots (see Figure 4) . We observed that some variables like Stream, Views, Speechiness, etc. are left skewed. So, we would need to take a closer look of them when evaluating possible transformations. To gain familiarity with the features with did some visualization. Some of them can be found in the Appendix in Figures 5-6-7.

Finally, we took a look at the outliers just to have a general idea since we will deal with them in another step of the project.

Table 1: Variables most correlated with Stream

Column	Correlation
Likes	0.6543
Views	0.6019
Comments	0.2677
Loudness	0.118695
Acousticness	-0.104843

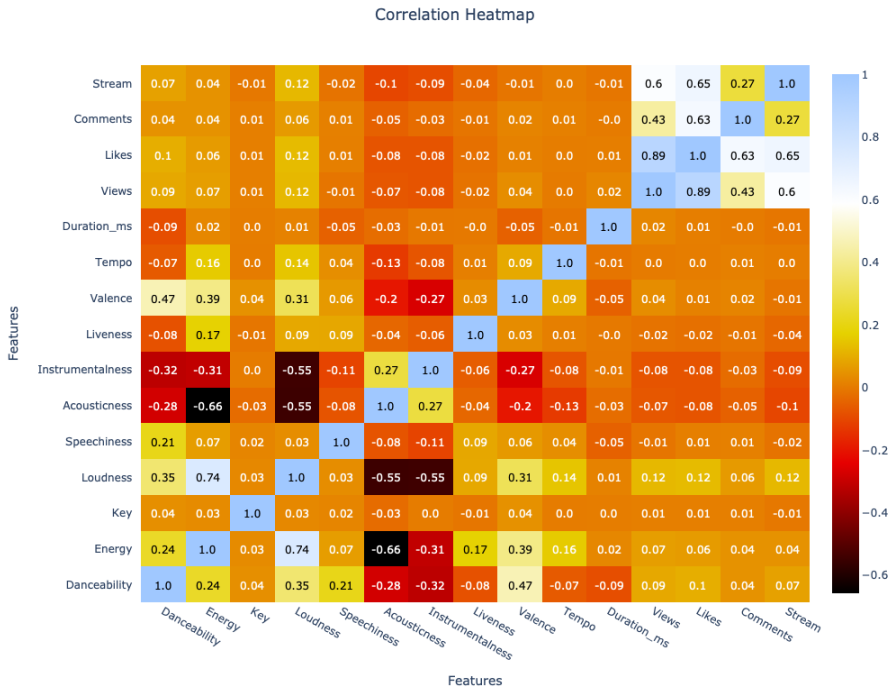


Figure 1: Correlation Matrix

3.2 General data cleaning

We decided to remove duplicates. We considered that a duplicate was those rows whose variable Uri was the same. Subsequently, we removed the rows that had missing values in the target

variable. This decision was made due to the limited number of rows containing null values for the "Stream" variable.

3.3 Missing values

As an initial step, we assessed the presence of missing values in each column of the dataset to understand the distribution of these missing values. The frequency/count of missing values in each variable was calculated, and Table 2 presents the statistics for the variables with the highest number of missing values. Please note that variables with only two missing values across the entire dataset are not included in the table. As depicted in the table, the dataset does not exhibit a substantial percentage of missing values, and no specific column stands out due to an excessive number of it.

Table 2: Amount of missing values per column

Column	Missing Values 2	Percentage of total values
Description	755	4.1
Comments	477	2.6
Likes	450	2.5
`official_video`	384	2.1
Licensed	384	2.1
Views	384	2.1
Channel	384	2.1
`Url_youtube`	384	2.1
Energy	2	0

We explored two different methods for imputing the missing data: K-Nearest Neighbors (KNN) and Multiple Imputation by Chained Equations (MICE).

KNN imputation estimates missing values based on the values of their nearest neighbors. However, it's important to be cautious as the results can be influenced by the choice of K (number of neighbors) and the distance metric used. The quality of imputation may vary depending on the underlying structure of the data.

On the other hand, MICE is a multivariate imputation method that iteratively estimates missing values by modeling the relationships between variables. It performs multiple regressions over a random sample of the data, then combines the average of the multiple regression values to impute the missing value. This approach can capture complex dependencies and preserve the uncertainty associated with imputed values.

By employing both KNN and MICE, we can leverage their respective strengths and assess which method provides more accurate imputations for our dataset.

After analyzing the result we decided to use the results obtained from MICE.

3.4 Outlier detection

Outlier detection is a fundamental step in data analysis aimed at identifying and handling observations that deviate significantly from the expected patterns or norms. Outliers are data

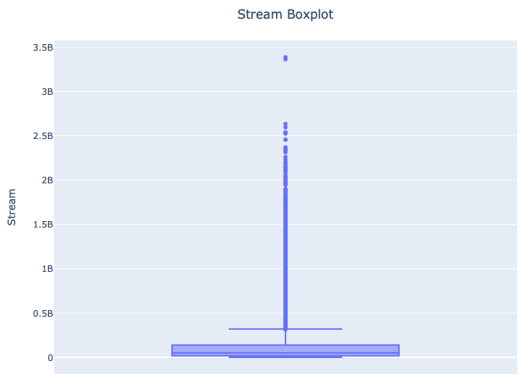
points that exhibit unusual behavior or extreme values compared to the majority of the dataset.

By detecting and addressing outliers, we can gain valuable insights into potential errors, anomalies, or unique occurrences within the data. Outliers can arise due to various factors, such as data entry mistakes, measurement errors, or rare events that are not representative of the general data distribution.

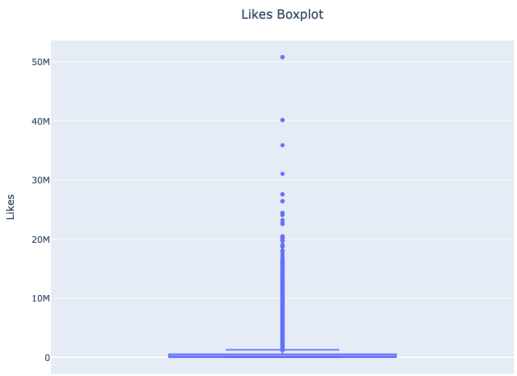
The process of outlier detection is crucial for ensuring data integrity, enhancing the accuracy of statistical analyses, and preventing biased conclusions. By removing or appropriately handling outliers, we can improve the reliability and validity of our models.

To detect outliers, we generated boxplots for each variable to gain a better visualization of their distributions. By examining the boxplots, we aimed to identify any data points that deviate significantly from the central tendencies and understand the extent of their spread across the variables.

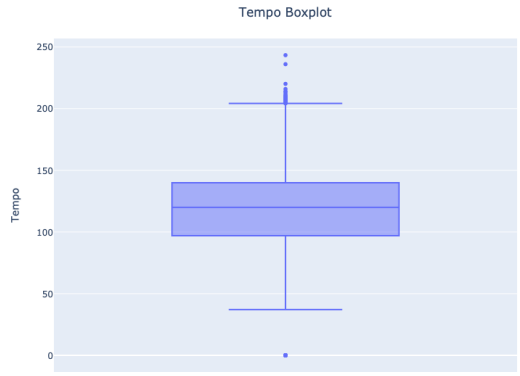
Since we are dealing with the number of streams a song has and the number of views a video for a song has we expect to see high values. Some of the outliers found, were closely examined in order to establish if they were wrong. We investigate a bit and the detected outliers were correct, for example one outlier was a song with more than 140 billion streams but this value was accurate.



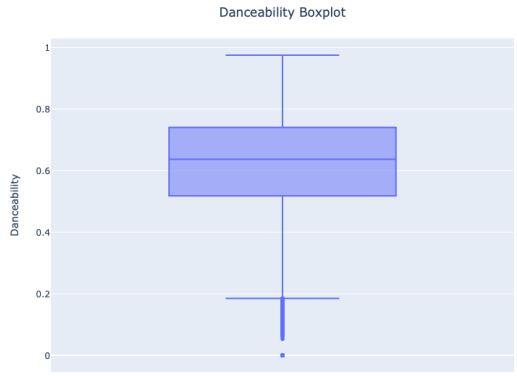
(a) Stream outliers



(b) Likes outliers



(c) Tempo outliers



(d) Danceability outliers

Figure 2: Plots of some outliers

3.5 Data transformation

We opted to perform more targeted feature transformations during the modeling stage rather than incorporating them into the initial data transformation step. This approach allowed us the flexibility to apply specific transformations based on the requirements of each modeling technique or algorithm being employed. By applying transformations at this stage, we could tailor the data to better suit the needs of the modeling process and improve the performance and interpretability of our models.

However, we did do feature encoding using one-hot encoder of the variable `Album_type`. We decided to do it because this categorical variable has no inherent ordinal relationship between their categories. Since, the variable has only three categories it will not increase that much the dimensionality of the dataset.

3.6 New Features Derivation

For each track in the dataset, we can find its corresponding Spotify URI, which is stored in a column. By using this URI we can get lots of information about the tracks by accessing the Spotify API, a great public tool. Thus, after reviewing the features related to a track through

the API, we extracted the following new features:

- album_tracks: the number of tracks in the album.
- available_markets: the number of markets/countries in which the album is available.
- album_release_days: we calculated this feature that represents the antiquity of a track by calculating the difference between the date the album was first released and the date for which the dataset is being published for the first time in Kaggle.

Besides the before-mentioned features extracted from the Spotify API, we also generated some new features that we believe would be useful for the prediction:

- trackLength: the length of the track's name in characters.
- albumLength: the length of the track's album name in characters.
- ArtistEqualsChannel: it indicates if the name of the youtube channel where the track is released coincides with the artist's name.
- ifTop100: it represents if the track's artist is one of the Billboard top 100 artists in the year 2022.

3.7 Dimensionality Reduction

One of the most important aspects of model training is Dimensionality reduction, which is the key to reducing model complexity and overfitting. For this project, we used the two most commonly applied techniques for dimensionality reduction: feature selection and feature extraction.

3.7.1 Feature Selection

The goal of this data pre-processing task is to reduce the dimensionality of the feature space by selecting a subset of relevant features from the original features set. If we don't apply feature selection when training a model, we might encounter two problems. Firstly, the inclusion of too many features can lead the model to be computationally expensive and may impact its generalization on unseen data. Secondly, irrelevant or redundant features can introduce noise and bias to the model, which causes the overfitting problem.

Among the many feature selection methods we chose the Recursive Feature Elimination(RFE), a wrapper method that involves repeatedly training and evaluating a model on different subsets of features, and selecting the subset that achieves the best performance, which means, it selects the most important features by recursively removing the least important features and retrains the model to evaluate the performance. One of the downsides of this method is that it is computationally expensive, especially when dealing with our datasets or complex models. The first step we did is to remove some categorical features that did not contain useful information for our prediction task from the dataset, such as the track's name, the track's URI in Spotify, the album name, etc.

As our validation method chosen is k-fold cross-validation, We decided that the features importance search using the method RFE should be done after the train-test splitting and within each fold of cross-validation. The reason for doing so is to avoid leaking information from the test set into the training pipeline. Having to repeat the feature selection process in each fold is computationally expensive, but we believe that this is the right way to avoid getting a biased feature selection result toward information of (what will be) the test set.

3.7.2 Feature Extraction

In this stage of the data pre-processing pipeline, we aim to transform the original features into a lower-dimensional feature space by only capturing the essential information from the original feature set. The approach we chose is the Principal Component Analysis (PCA) method which consists of finding the principal components that explain the maximum variance in our dataset.

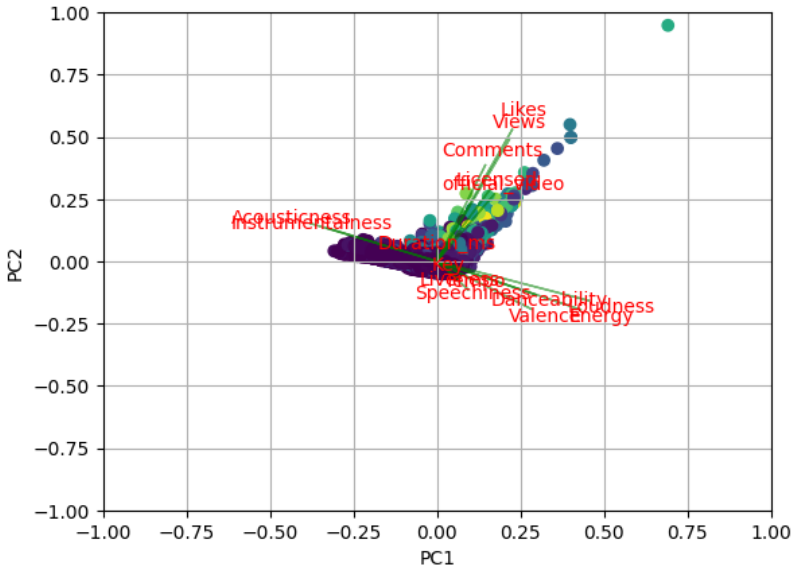


Figure 3: Features projected in the PCA space

From figure 3 we can see that the features can be divided into 3 groups where the features in each group are either positively or negatively correlated with the ones in other groups.
We finally decided not to transform the features into principal components to reduce the features' space dimensionality because by doing so, we will low interpretability problem, and in order to avoid information loss, a characteristic part of PCA, we should choose at least 20 principal components, which we thought that it is not a good trade-off between what we gained by using PCA and the downsides compromised by using it.

4 Modeling methods

Since we are dealing with a continuous target we decide to utilize the following approaches:

- **Linear Regression(LR)**: a simple and widely used algorithm that works well when there is a linear relationship between the predictor variables and the target variable. In the case of our dataset, the linear relationship is not presented. However, we decide to use the algorithm to get the baseline model to compare it with the other more robust algorithms later.
- **Polynomial Regression(PR)**: an extension of Linear Regression that allows the relationship between variables to be non-linear. It uses polynomial functions to model this non-linear relationship.
- **Decision Tree(DT)**: it can handle non-linear relationships and interactions between variables by partitioning the input space into regions based on the input variables' values.

- **Random Forest(RF)**: an ensemble method that creates a collection of decision trees and aggregates their predictions to get the final prediction. It's widely used in regression tasks and performs well in most cases.
- **Support Vector Regression(SVR)**: an extension of Support Vector Machines(SVM) that maps the input variables into a higher-dimensional future space and tries to find the hyperplane that minimizes the prediction error. It handles well with high-dimensional data.
- **Neural Networks**: Deep learning models can capture complex relationships with the use of input, hidden, and output layers. It's suitable to apply when we have a large amount of data.

4.1 Validation protocol

In order to evaluate the performance of the models we did firstly a train-test split, a commonly used technique to evaluate the performance of a model on unseen data. By doing so, the dataset is divided into two subsets: the training set and the test set. A 80%-20% split was used. We performed 10-fold cross-validation on the training where 9 folds are used for training and 1 fold acts as a validation test to assess the model's performance and tune hyperparameters. The test set remains separate and is not used during the cross-validation process, which is repeated 10 times. When this process is done, we select the model that performed the best according to the evaluation metrics chosen. Finally, we evaluate the selected model on the separate test set that was not used during the cross-validation process.

Algorithm 1 Best model selection with K-fold cross-validation

```
for each model do
  for each fold do
    Partition the training set into K-1 training folds and one validation fold.
    Train the model on the K-1 training folds.
    Evaluate the model on the validation fold and compute the performance metric chosen.
  end for
end for
Choose the best-performing model.
Evaluate the best model selected in the previous step on the test set.
```

4.2 Evaluation Metrics

As we are dealing with a continuous target the following metrics evaluation metrics were used:

- Mean Absolute Error (MAE): it calculates the average absolute difference between the predicted and actual values. It provides a measure of the average prediction error.
- R-squared (R2) Score: represents the proportion of the variance in the target variable that can be explained by the model.
- Root Mean Squared Error (RMSE): it is the square root of MSE and provides a measure of the average prediction error in the original units of the target variable.

4.3 Linear and Polynomial Regression

To build these two models for the regression task, we used the functions LinearRegression and PolynomialFeatures of the library scikit-learn. As we already know, the difference between the two algorithms is that Linear regression assumes a linear relationship between the input variables and the target where Polynomial regression allows for modeling nonlinear relationships between them. When modeling the polynomial regressor we tried different values for the degree term, which refers to the highest power to which the input variables are raised in a polynomial equation. As for the linear regression, there is no degree associated and the degree 1 in polynomial regression is actually doing the linear regression.

4.4 Neural Network: MLPR

To implement the model using neural networks, we utilized MLPRegressor from the scikit-learn library. Neural networks have gained significant popularity in a variety of task because of their capability to capture intricate relationships within the data. However, we need to be careful because their performance is heavily influenced by the selection of appropriate hyperparameters.

The first thing we did was to apply a Box-Cox transformation for the following features: 'Duration_ms','Views','Likes','Comments','album_release_days'. During exploratory analysis, we observed that these features exhibited a right-skewed distribution. In such cases, applying a Box-Cox transformation can assist in achieving a more symmetric distribution that approximates a normal distribution.

To conduct hyperparamter tunning, we utilized a randomized search approach. We should mention that we did try doing a grid search, but due to its extensive execution time, we opted for the randomized search instead. The different parameters tried are depicted in Table 3.
The best parameters turned out to be the following:

```
{'solver': 'adam',
 'max_iter': 1000,
 'hidden_layer_sizes': (50,),
 'alpha': 0.01,
 'activation': 'relu'}
```

Parameter	**Value**
hidden_layer_sizes	[(10,), (50,), (64,), (128,), (256,), (64, 64), (100,), (100, 50,), (128, 128), (256, 256), (200, 100, 50)]
activation	relu, tanh
alpha	0.0001, 0.001, 0.01
solver	adam
max_iter	50, 100, 1000

Table 3: Parameter values for the MLRegressor

After training the model with the best parameters found we obtained the following results:

- Mean absolute error: 5.28
- Root mean squared error: 6.89
- R2 score: 0.56

4.5 Support Vector Regression (SVR

To implement the model, we utilized SVR from the scikit-learn library.
Similarly to our approach with neural networks, we applied a Box-Cox transformation to the same variables mentioned earlier.
SVR has several hyperparamters that can be tuned to optimize its performance. We decided to perform a randomized search in order to improve the following hyperparamters:

- Kernel: kernel function to model the nonlinear relationships between the features and the target variable.
- C: it is the regularization parameter which controls the trade-off between achieving a low training error and a low margin violation.
- Gamma: it determines the influence of each training sample.
- Epsilon: defines the margin around the regression line where errors are considered acceptable.

Table 4 depicts the parameters distribution for the randomized search

Parameter	**Value**
kernel	['rbf', 'linear']
C	[0.1, 1, 10]
gamma	['scale', 'auto']
epsilon	[0.1, 0.2, 0.3]

Table 4: Parameter values for the SVR

The best parameters turned out to be the following:
`{'kernel': 'linear', 'gamma': 'auto', 'epsilon': 0.2, 'C': 10}`

It is worth noting that we attempted to use a grid search for hyperparamter optimization, however, the execution time increased significantly, surpassing even the duration observed with neural networks.

This model was the one that took the longest to run surpassing more than 5 hours.

4.6 Decision Tree

To build a decision tree model for the regression task, we used the function DecisionTreeRegressor of the library scikit-learn. In the case of the decision tree, normalization or standardization methods are not required because the splitting criteria of nodes is not affected by the scale or magnitude of the input variables. However, we decided to normalize some variables that have significantly different scales with respect to the rest in order to make all the features in a similar scale and to speed up the convergence process.

4.6.1 Model training

We first trained the model without applying the hyperparameters tuning to find the optimal values for the hyperparameters, and obtained an R2=0.182 and an MSE =7.211 on the validation set, as for the training set, we got an R2=1.0 and MSE=0. It is obvious that the model is overfitted. Thus we proceed to tune the following hyperparameters of decision tree in regression:

- **Maximum Depth(max_depth)**: it defines the maximum depth of decision tree. Tuning this parameter can help control the model complexity and generalization capability.
- **Maximum Features (max_features)**: it determines the maximum number of features to take into account when considering the best-split feature. It can help to reduce the chance of overfitting.
- **Minimum Sample Leaf (min_samples_leaf)**: it determines the minimum number of samples needed to be a leaf node.
- **Minimum Weight Fraction Leaf(min_weight_fraction_leaf)**: it specifies the minimum weighted fraction of the total number of samples needed to be a leaf node. Similar to (min_samples_leaf).
- **Splitting Criterion(Splitter)**: criteria that evaluate the quality of split.
- **Maximum Leaf Nodes(max_leaf_nodes)**: it specifies the maximum number of leaf nodes allowed in the tree.

The table 5 shows all the parameters values we tried when performing the Grid Search:

Parameter	**Value**
splitter	['best', 'random']
`max_depth`	[1,3,5,7,9,11,12]
`min_samples_leaf`	[1,2,3,4,5,6,7,8,9,10]
`min_weight_fraction_leaf`	[0.1,0.2,0.3,0.4,0.5]
`max_features`	["auto","log2","sqrt",None]
`max_leaf_nodes`	[None,10,20,30,40,50,60,70,80,90]

Table 5: Parameter values for the Decision Tree

After running the hyperparameters tuning we got the following best features:

```
{'max_depth': 5,
 'max_features': 'auto',
 'max_leaf_nodes': 10,
 'min_samples_leaf': 1,
 'min_weight_fraction_leaf': 0.1,
 'splitter': 'best'}
```

The final step we did when modeling the decision tree was to train again the model with the best parameters found in the previous step, as results we got MSE=5.96 and R2=0,45 on the training set, as for the validation set, MSE=6.06, and R2=0.44. After tuning the parameters the overfitting problem disappears but the model's performance is quite poor.

4.7 Random Forest

To build a random forest model for the regression task, we used the function RandomForestRegressor of the library scikit-learn. As we already know, random forest is an ensemble learning algorithm that uses the Bootstrap Aggregation (or bagging) technique to create multiple decision trees to make predictions. Once all the trees are built, predictions are made by each tree for new input and the final prediction is usually the average of the predictions from all the trees. Same as the previous section(decision tree), we first created the baseline model of random forest and then applied the hyperparameters tuning process. The results obtained before and after

tuning the parameters were quite similar, the model improvement was small(0.564 vs 0.607 in R2 and 5.21 vs 5.012 in MSE)

5 Results obtained

Upon examining the results in Table 6, it is evident that no model stands out significantly among the other. However, we can observe that the Random Forest model has the lowest RMSE (6.581) value and it also has the highest R2 value (0.607), indicating a better fit to the data compared to other models. This is why the model selected as final is the Random Forest.

Table 6: Metrics for the Different Models

Model	**RMSE**	**MAE**	**R2**
Linear regression	7.125	2.367	0.523
Polynomial regression degree 2	8.185	2.280	0.392
Polynomial regression degree 3	10.530	2.496	-0.041
Neural Network	6.89	5.28	0.56
Decision tree	7.8134	6.0620	0.448
Random forest	6.581	5.012	0.607
SVR	7.33	5.62	0.5

6 Final model selection

We chose the Random Forest as the final model as it provides the best prediction performance on the validation test. After the best model selection, we proceed to evaluate the model on test data to asses its performance on unseen data and generalization ability. Thus we retrain the Random Forest model with all the training set data(used by the k-fold cross-validation process in the modeling section), and then evaluate it with test data. The results are shown below:

Table 7: Best model performance on testing data

Dataset	**MSE**	**RMSE**	**MAE**	**R2**
Training	0.62	1.1	0.62	0.99
Testing	5.88	7.192	5.88	0.24

As we can see from the table 7, our best model performs poorly on the testing set, which means that the model's generalization ability is not satisfactory, The reason could be that the model learns too much from the training data, and captured noise(as we have high-dimensional features) that makes it unable to generalize well to unseen data. Or maybe we have applied some incorrect preprocessing steps that lead to the pool model performance. We believe that we do not have an insufficient training data problem because our preprocessed dataset has almost 18.000 rows, which means that the model has enough information to learn robust patterns.
We also tried to convert the regression task into a classification task by discretizing our continuous target(stream) into 3 intervals: low, medium, and high. The results obtained by applying a random forest classifier shows that the model performs well when handling classification task. There is no overfitting on the model because the accuracy was about 0.820 on the training and 0.789 on the testing data.

7 Scientific and personal conclusions

In this project, we conducted a comprehensive machine learning process using a real dataset from the music industry. The primary goal was to predict the number of streams a song would receive based on various song-related factors such as energy, loudness, etc., as well as information related to the music video. Through this project, we gained valuable insights into the development process involved in such projects.

Our approach encompassed several crucial steps, including data pre-processing, data visualization, defining a validation protocol, selecting an appropriate model, and ultimately creating a final model while estimating its generalization error. This project provided us with a valuable opportunity to understand the intricacies involved in developing and implementing a machine learning project from start to finish.

During the project, we encountered challenges and realized that we allocated more time than anticipated to the preprocessing step. While we acknowledge the significance of this step, we understand that in real-life scenarios, it is acceptable to dedicate ample time to it. However, considering our academic commitments and other time-consuming tasks, we believe we should have prioritized the modeling aspect more efficiently. This would have allowed us to allocate more time and resources to refining the models and further improving the predictive performance.

Based on the results of the project, we can concluded that none of the attempted models yielded satisfactory results. This may be due to a bad choice of hyperparameters or maybe to an improper data preprocessing among other. We should have performed a better feature selection. Due to lack of time, we were not able to try to improve more the models in order to reduce overfitting.
In conclusion, this project provided us with learning experience in the development of a machine learning project using a real-world dataset. It allowed us to gain practical knowledge and hands-on experience with the methods and techniques discussed in our lectures. Overall, we found the project to be an engaging opportunity to apply theoretical concepts to practical scenarios, further enhancing our understanding of machine learning in real-world applications.

8 Possible extensions and known limitations

As part of future research, there is a need to explore additional machine learning techniques that were not considered during the model selection phase of this project. For instance, investigating ensemble methods.

We believe that treating the Stream variable as a categorical target, rather than a continuous one, could yield improved results. Unfortunately, due to time limitations and to the fact that we wanted to first try to use regression methods, we were unable to thoroughly explore this approach. However, we did train one model using this categorical transformation and observed better outcomes compared to when utilizing regression methods.

Another approach could be transforming the streams variable into a binary one, like for instance hit and flop and use some binary classification model.

During the experimentation phase, we encountered limitations in terms of the execution time required for hyperparameter tuning of the various models. In certain instances, the execution

time exceeded what was feasible within our available resources. We believe that with more advanced computing resources, the models could have been further improved and optimized.

In order to improve the performance of the model, we should explore alternative hyperparamters or consider different machine learning algorithms. Models like Gardient Boosting Regression or Ensemble methods, such as Bagging and Stacking could be tested.

References

[1] Support Vector Regression for Machine Learning, https://medium.com/analytics-vidhya/support-vector-regression-for-machine-learning-843978ba6279

[2] An Introduction to Support Vector Regression (SVR) https://towardsdatascience.com/an-introduction-to-support-vector-regression-svr-a3ebc1672c2

[3] Understanding Random Forest https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[4] 3 Best metrics to evaluate Regression Model?, https://towardsdatascience.com/what-are-the-best-metrics-to-evaluate-your-regression-model-418ca481755b

[5] Understanding Polynomial Regression!!!, https://medium.com/analytics-vidhya/understanding-polynomial-regression-5ac25b970e18

9 Appendix

9.1 Figures and Tables

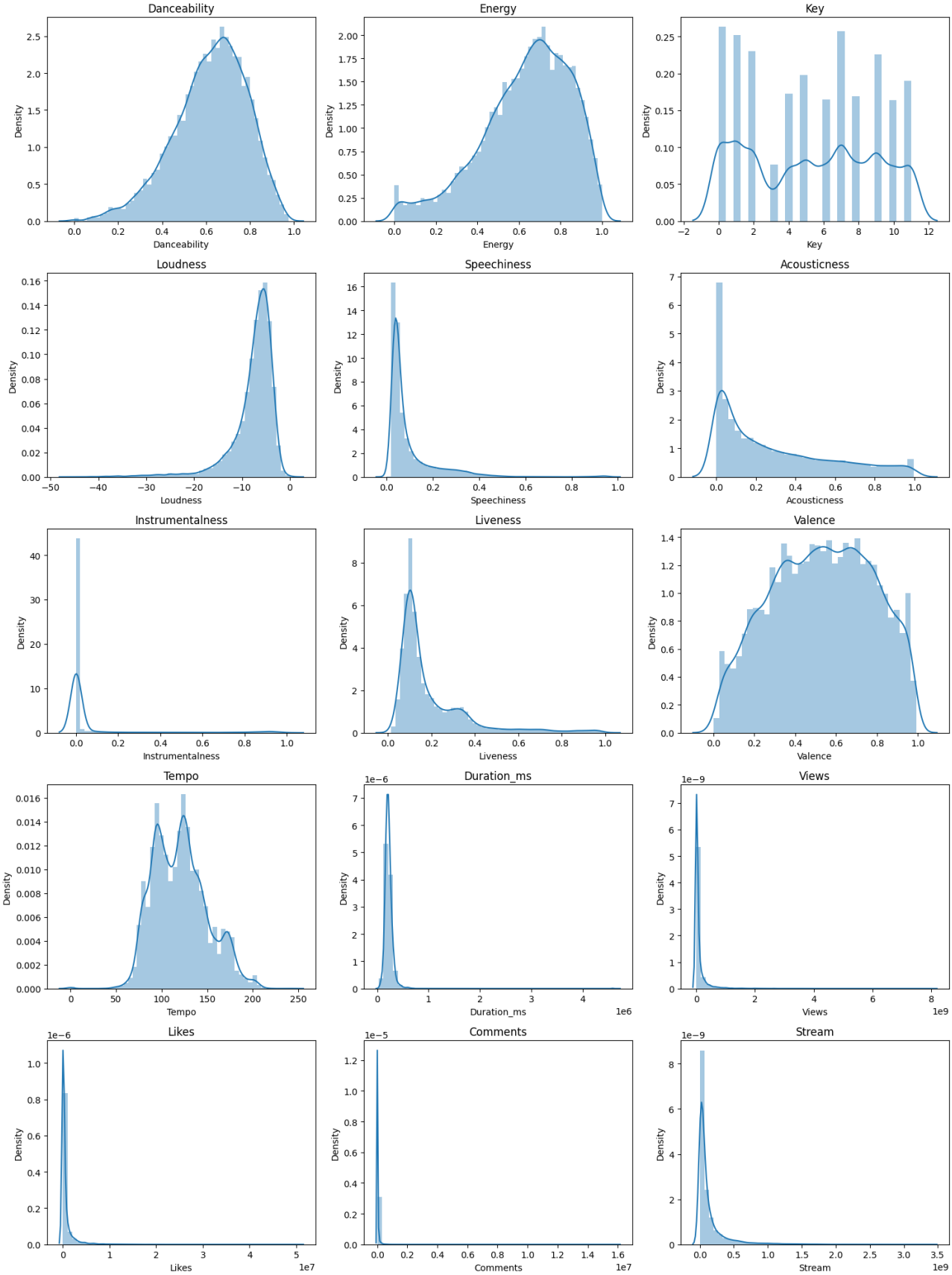


Figure 4: Distribution plots

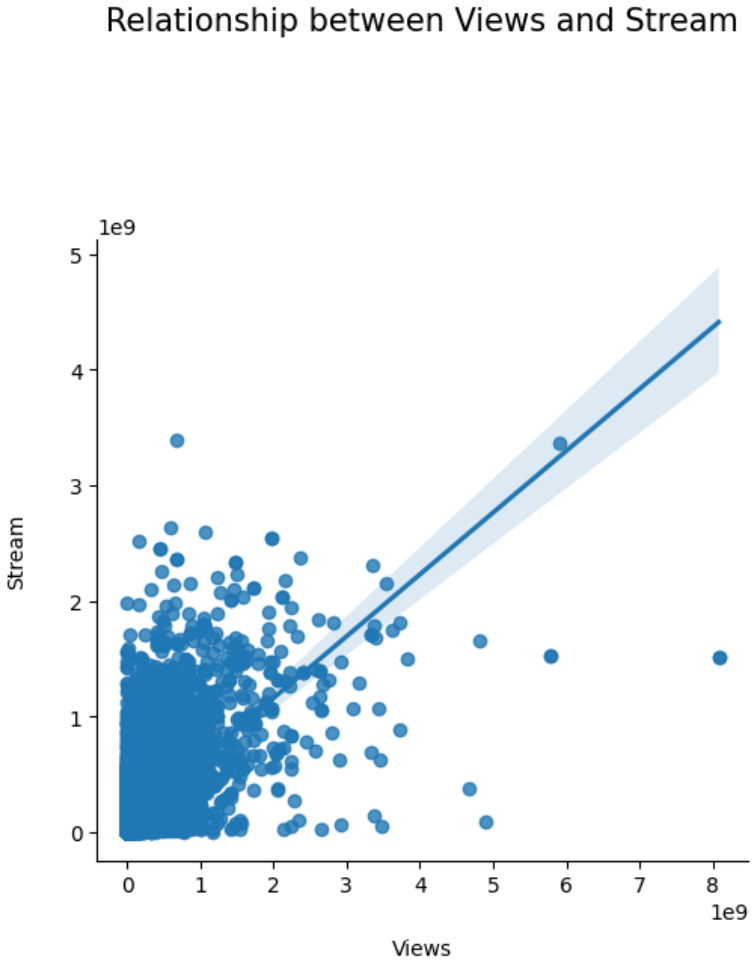


Figure 5: Streams vs Views

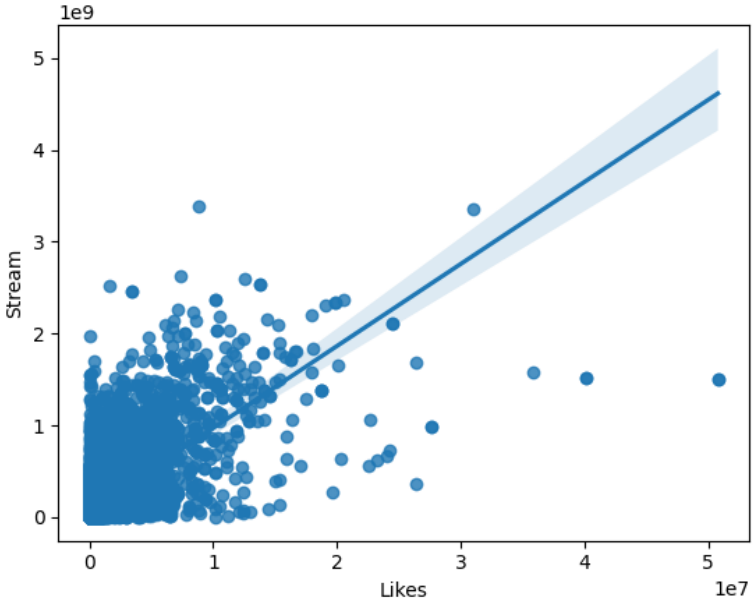


Figure 6: Likes vs Streams

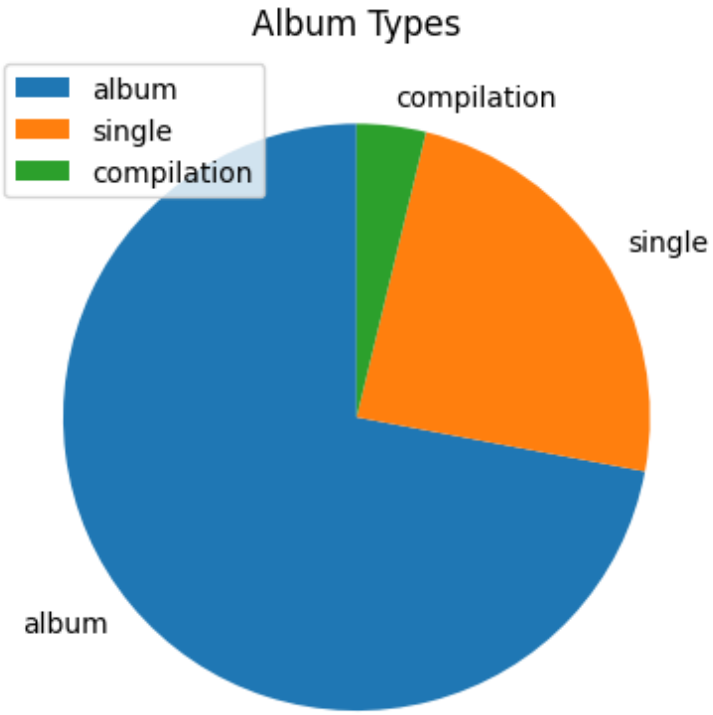


Figure 7: Album types distribution