Libraries

In [1]:
```
import os

import joblib

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

import sklearn

import plotly.express as px
import plotly.graph_objects as go

import statsmodels.api as sm
from sklearn.ensemble import RandomForestClassifier,RandomForestRegressor
from sklearn.preprocessing import MinMaxScaler, StandardScaler,LabelEncoder
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.linear_model import Ridge,ElasticNet,LogisticRegression, LinearRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score,accuracy_score,mea
from sklearn.multiclass import OneVsRestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier,DecisionTreeRegressor
from plotly.subplots import make_subplots
```

Upload Dataset

In [2]:
```
# Get the current working directory
directorio_actual = os.getcwd()

# Specify the relative path from the current directory
ruta_csv_relativa = os.path.join('..', 'data', '03_primary','3.spotify.csv')

# Load the CSV file
spotify = pd.read_csv(ruta_csv_relativa)
```

Model of Regression

The results indicate that Random Forest is the best model for predicting energy_scaled, with an R^2 of 0.778 meaning that around 78% of the variation in energy_scaled is explained by loudness_scaled and intensity.

The MSE and RMSE (both low) also indicate that the prediction error is low, meaning that the Random Forest model makes a good approximation to the actual values of energy_scaled.

This type of model can be very useful in any system that seeks to personalize, categorize or analyze music based on its loudness or energy.

In [3]:
```
# Define function to evaluate models
def train_and_evaluate(X, y):
    # Split the data into training and testing sets
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Initialize models
    models = {
        "Linear Regression": LinearRegression(),
        "Decision Tree": DecisionTreeRegressor(max_depth=5),
        "Random Forest": RandomForestRegressor(n_estimators=100, max_depth=5, random_state=42)
    }

    # Dictionary to store metrics for each model
    metrics = {}

    # Train and evaluate each model
    for model_name, model in models.items():
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)
```

```
        # Calculate metrics
        mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)
        rmse = np.sqrt(mse)

        # Store metrics in dictionary
        metrics[model_name] = {
            "MSE": mse,
            "R²": r2,
            "RMSE": rmse
        }

    return pd.DataFrame(metrics).T

# Model 1: energy_scaled ~ loudness_scaled + intensity
X1 = spotify[['loudness_scaled', 'intensity']].values
y1 = spotify['energy_scaled'].values
results_model_1 = train_and_evaluate(X1, y1)
print("Model 1 - energy_scaled prediction")
print(results_model_1)
```

```
Model 1 - energy_scaled prediction
                        MSE        R²      RMSE
Linear Regression  0.017176  0.725872  0.131056
Decision Tree      0.014304  0.771707  0.119598
Random Forest      0.013879  0.778491  0.117808
```

Model of Regression

The results indicate that Random Forest is the best performing model for predicting danceability_scaled with an R^2 of 0.314, meaning that the model explains only 31.4% of the variation in danceability_scaled. While this is better than the other models, it is still a relatively low value, suggesting that valence_scaled and energy_scaled are not sufficient to predict danceability with high accuracy.

The MSE and RMSE are also relatively low, indicating that the absolute error of the model is not high, but the low R^2 implies that there are other influential variables that are not being considered.

While this model offers some predictive power over danceability_scaled, the results suggest that valence_scaled and energy_scaled do not fully capture what makes a song “danceable.” Still, this model can be useful for preliminary song classification and improving music recommendations for users who prefer danceable songs.

In [4]:

```
# Model 2: danceability_scaled - valence_scaled + energy_scaled
X2 = spotify[['valence_scaled', 'energy_scaled']].values
y2 = spotify['danceability_scaled'].values
results_model_2 = train_and_evaluate(X2, y2)
print("Model 2 - danceability_scaled prediction")
print(results_model_2)
```

```
Model 2 - danceability_scaled prediction
                        MSE        R²      RMSE
Linear Regression  0.026195  0.223513  0.161849
Decision Tree      0.023377  0.307057  0.152894
Random Forest      0.023128  0.314434  0.152078
```

Model of Regression

The results show a very low R^2 for all models (at best, Random Forest has an R^2 of 0.031), meaning that only 3.1% of the variation in popularity can be explained by intensity, valence_scaled, and energy_scaled. This indicates that these musical features are not sufficient to predict popularity, which makes sense, since popularity depends on many additional factors (promotion, release time, collaborations, etc.) that are not reflected in these variables.

The MSE and RMSE are also relatively high relative to the popularity scale, suggesting that the model has a considerable error in its predictions.

This model reveals that the popularity of a song cannot be predicted well by the musical features intensity, valence, and energy alone. Although these features may contribute to some extent, the popularity of a song seems to be influenced by other external factors that are not represented in this dataset.

Still, this model can be useful as a preliminary reference and to understand the complexity behind musical popularity.

In [5]:
```
# Model 3: popularity_scaled - intensity + valence_scaled + energy_scaled
X3 = spotify[['intensity', 'valence_scaled', 'energy_scaled']].values
y3 = spotify['popularity_scaled'].values
results_model_3 = train_and_evaluate(X3, y3)
print("Model 3 - popularity_scaled prediction")
print(results_model_3)
```

```
Model 3 - popularity_scaled prediction
                        MSE        R²      RMSE
Linear Regression  0.029694  0.001832  0.172319
Decision Tree      0.029022  0.024405  0.170359
Random Forest      0.028818  0.031285  0.169757
```

Model Of Classification

In [6]:
```
# Feature selection for classification
X_class_opt = spotify[['danceability_scaled', 'energy_scaled', 'acousticness_scaled', 'valence_scaled']]
y_class_opt = spotify['popularity_class']  # Using popularity_class as the target variable

# Splitting data into training and testing sets
X_train_opt, X_test_opt, y_train_opt, y_test_opt = train_test_split(X_class_opt, y_class_opt, test_size=0.2, ra

# Parameter configuration for random search
param_dist = {
    'n_estimators': [100, 300, 500],
    'learning_rate': [0.01, 0.05, 0.1, 0.2],
    'max_depth': [4, 6, 8, 10],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'gamma': [0, 0.1, 0.3, 0.5],
    'min_child_weight': [1, 3, 5]
}

# Creating the XGBoost model
xgb_opt = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)

# Configuring the random search
random_search = RandomizedSearchCV(
    estimator=xgb_opt,
    param_distributions=param_dist,
    n_iter=50,  # Number of hyperparameter combinations to try
    scoring='accuracy',
    cv=3,  # 3-fold cross-validation
    random_state=42,
    n_jobs=-1
)

# Running the random search to find the best parameters
random_search.fit(X_train_opt, y_train_opt)

# Training the best model found
best_xgb = random_search.best_estimator_
best_xgb.fit(X_train_opt, y_train_opt)

# Making predictions
y_pred_opt = best_xgb.predict(X_test_opt)

# Calculating the confusion matrix, classification report, and accuracy
print("Confusion Matrix:\n", confusion_matrix(y_test_opt, y_pred_opt))
print("\nClassification Report:\n", classification_report(y_test_opt, y_pred_opt))
accuracy_opt = accuracy_score(y_test_opt, y_pred_opt)
print("Optimized Accuracy:", accuracy_opt)
```

```
C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\xgboo
st\core.py:158: UserWarning: [02:18:25] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gro
up-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
C:\Users\diego\OneDrive\Imágenes\Escritorio\Spotify-Recomendation-Machine-Learning\.venv\Lib\site-packages\xgboo
st\core.py:158: UserWarning: [02:18:28] WARNING: C:\buildkite-agent\builds\buildkite-windows-cpu-autoscaling-gro
up-i-0ed59c031377d09b8-1\xgboost\xgboost-ci-windows\src\learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
```

```
Confusion Matrix:
 [[8309 3621]
 [3154 7676]]


Classification Report:
               precision    recall  f1-score   support

           0       0.72      0.70      0.71     11930
           1       0.68      0.71      0.69     10830

    accuracy                           0.70     22760
   macro avg       0.70      0.70      0.70     22760
weighted avg       0.70      0.70      0.70     22760

Optimized Accuracy: 0.7023286467486819
```

Clasificación

Model Of Classification

In [7]:
```python
# Feature selection for classification
X_class = spotify[['danceability_scaled', 'energy_scaled', 'acousticness_scaled', 'valence_scaled']]
y_class = spotify['popularity_class']  # Using popularity_class instead of explicit

# Splitting data into training and testing sets
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X_class, y_class, test_size=0.2, ra

# Creating the KNN model for classification
knn = KNeighborsClassifier(n_neighbors=5)

# Training the model
knn.fit(X_train_class, y_train_class)

# Making predictions
y_pred_knn = knn.predict(X_test_class)

# Calculating the confusion matrix
conf_matrix = confusion_matrix(y_test_class, y_pred_knn)
print("Confusion Matrix:\n", conf_matrix)

# Generating the classification report
report = classification_report(y_test_class, y_pred_knn)
print("Classification Report:\n", report)

# Calculating accuracy
accuracy_knn = accuracy_score(y_test_class, y_pred_knn)
print("Accuracy:", accuracy_knn)
```

```
Confusion Matrix:
 [[7859 4071]
 [3962 6868]]
Classification Report:
               precision    recall  f1-score   support

           0       0.66      0.66      0.66     11930
           1       0.63      0.63      0.63     10830

    accuracy                           0.65     22760
   macro avg       0.65      0.65      0.65     22760
weighted avg       0.65      0.65      0.65     22760

Accuracy: 0.6470562390158172
```

Model Of Classification

In [ ]:
```python
# Feature selection for classification
X_class_rf = spotify[['danceability_scaled', 'energy_scaled', 'acousticness_scaled', 'valence_scaled']]
y_class_rf = spotify['popularity_class']  # Using popularity_class instead of explicit

# Splitting data into training and testing sets
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_class_rf, y_class_rf, test_size=0.2, random_s

# Creating the Random Forest model for classification
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Training the model
rf.fit(X_train_rf, y_train_rf)
```

```python
# Making predictions
y_pred_rf = rf.predict(X_test_rf)

# Calculating the confusion matrix and classification report
print("Confusion Matrix:\n", confusion_matrix(y_test_rf, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test_rf, y_pred_rf))
print("Accuracy:", accuracy_score(y_test_rf, y_pred_rf))
```