Project Documentation – Spotify Track Popularity Prediction

Executive Summary

This project aimed to predict the popularity of Spotify tracks based on their audio features and metadata.

Using a dataset from Kaggle, we implemented a complete machine learning pipeline: data preparation, exploratory data analysis, feature engineering, model training (both regression and classification), evaluation, and visualization.

Multiple models were tested (Random Forest, XGBoost, etc.), but regression performance remained modest ($R^2 \approx 0.26$), highlighting that track popularity is heavily influenced by external factors not present in the dataset—such as marketing campaigns, social trends, or viral exposure.

Reframing the problem as a binary classification task (popular vs. not popular) yielded better results, with improved F1 scores and ROC-AUC metrics.

The project underlines the need to enrich the dataset with external sources, such as streaming statistics, social media data (via APIs), or sentiment analysis of song lyrics.

As it stands, the pipeline is well-documented, modular, and ready for reuse or further development in real-world applications like music recommendation systems or trend monitoring tools.

1. Data Preparation

In this section, we load the initial dataset `**spotify_songs.csv**` and examine its basic structure, including column characteristics, shape, missing values, duplicates, and more. The dataset contains 28,356 songs with various audio and metadata features, such as danceability, energy, loudness, tempo, subgenre, and release date. The target variable is *track_popularity*, an integer score between 0 and 100. Features: audio features (numeric),

metadata (release year, artist), subgenre categories. The purpose of this initial analysis is to identify areas where preprocessing is needed, such as merging tables, handling textual and categorical columns, and other necessary adjustments.

The following cleaning steps have been performed:

- Elimination of duplicate tracks based on *track_id*, with a summary exported to CSV.
- Text normalization: removal of special characters and conversion to lowercase for fields like *track_name*, *track_artist*, *track_album_name* and *playlist_name* columns. A separate *df_text* DataFrame was created for NLP use cases.
-Created a new feature *playlist_count* to count how many playlists a track appeared in and dropped *playlist_id* column due to redundancy.
- Categorical columns like *playlist_genre*, *playlist_subgenre*, and *track_artist* were processed to reduce cardinality. Rare categories (less than 1% of the dataset) were grouped under the label `"Other"` using frequency-based mapping. To reduce the granularity of the *playlist_subgenre* variable and facilitate analysis, a new feature called *subgenre_family* was engineered. These variable groups related subgenres into broader musical families using a manually defined mapping dictionary. For example, various subgenres such as "trap", "southern hip hop", and "gangster rap" were grouped under the broader family "Hip-Hop / Rap". Similarly, "dance pop" and "latin pop" were grouped under "Pop", and so on. Any subgenre not explicitly listed in the dictionary was assigned the label "Other". The new column *subgenre_family* replaced the original *playlist_subgenre* feature in our modeling pipeline.
- The cleaned dataset was saved as **'data_preparation_23_03_25.pkl'**

2. Exploratory Data Analysis (EDA)

A comprehensive EDA was conducted to better understand the structure, distributions, and trends in the Spotify dataset prior to modeling.

The steps below summarize the key parts of the analysis:

-After importing the dataset, redundant columns such as Unnamed: 0 were dropped. Structural information (such as number of rows, data types, missing values, and unique value counts) was extracted and saved to Excel files for external review.

- The distribution of the target variable, *track_popularity*, was analyzed using: a histogram with KDE to examine skewness and modality, a boxplot to identify outliers and a violin plot to visualize distribution density. These plots revealed a moderately skewed distribution with several high-popularity outliers.

-Twelve core audio features (such as danceability, energy, loudness, valence, tempo, etc.) were explored using individual histograms. This highlighted the presence of skewed distributions, which were later quantified using the skewness metric, with values visually flagged for significant deviations from normality.

-A correlation matrix was computed on all numerical features and visualized with a Seaborn heatmap. This helped identify strong relationships between variables such as *energy*, *loudness*, *danceability* and *valence* which could inform feature selection or dimensionality reduction later in the pipeline. No variable shows very strong correlation with *track_popularity*, reinforcing the complexity of the target.

- Scatterplots were used to analyze the relationship between musical features and popularity showing a positive but loose relationship between popularity and both danceability and energy. These features alone are not strong predictors but may still contribute to popularity when combined with others. It suggests that popular songs tend to be more energetic and danceable, but with many exceptions.

- From the release date column, new features were extracted: album_release_year, month, and day. The evolution of music over time was analyzed by plotting song release distribution per year and trends in average acousticness, liveness, and tempo over the years. We saw that most songs in the dataset were released between 2000 and 2020, with a strong increase in releases in the 2010s. Over time, acousticness has declined, suggesting music has become more electronic, the tempo has remained relatively stable and liveness has small fluctuations, possibly due to the rise of live performance recordings in streaming.

To complement the manual EDA, the AutoViz library was used to automatically generate visual summaries, scatterplots, and feature distributions from the CSV file.

3. Outliers and Missing Value Handling

A separate notebook provided a complete pipeline to detect, label, analyze, and cap outliers. We understood how outliers affect both distribution and correlation before imputing extreme values without losing important patterns. We also visualized and handled missing data to ensure model robustness.

-Created a new feature `*popularity_log*` using logarithmic transformation on `*track_popularity*` to normalize the peak at 0. But with the models, we concluded that it didn't help.

-A robust IQR-based method was used to detect outliers for all numeric features. This included:

- Boxplot visualizations to identify extreme values across features.
- IQR computation to detect values outside the [Q1 - 1.5×IQR, Q3 + 1.5×IQR] range.
- A DataFrame *outliers_df* was created, listing outlier counts and percentages for each feature.
- Outliers were explicitly labeled as "Outlier" to facilitate tracking and transformation.

-To assess the impact of outliers on data distribution and model interpretability:

- Kolmogorov-Smirnov tests were used to compare feature distributions with and without outliers.
- Fisher's z-transformation was applied to assess whether outliers significantly changed the correlation between features and the target variable (*track_popularity*).
- A summary table (*out_df*) was generated, showing each feature: outlier count, whether distribution changed, whether correlation changed, recommendation to drop or keep the feature.

Result: Features like loudness, speechiness, acousticness, and tempo were marked for outlier treatment due to significant distribution and correlation impact.

-Instead of removing outliers, they were replaced with *NaNs* (capping strategy), creating missing entries only in affected features. MICE imputation (via IterativeImputer) was used to intelligently fill missing values based on inter-feature relationships.

-The dataset was examined using missingno:

- A matrix view showed where missing values appeared.
- A correlation heatmap helped determine if missingness was related across columns.
- The overall percentage of missing values was computed per feature.

- Remaining missing values (e.g., *track_artist*) were filled with "Unknown Artist" or imputed.

-The cleaned dataset was saved as **df_cleaned24b_03.pkl** for future modeling steps.

4. Feature Engineering and selection

To improve data quality:

- Irrelevant columns like *popularity_log* were removed.
- A new feature *track_age* was computed as *2025 - release_year*.
- A filtered dataset *df_recent* was created, focusing only on songs released in the last 20 years.
- Skewed numeric features were automatically transformed using a custom function `smart_skew_transform`, which tested: `log1p`, `sqrt` and `boxcox`. The best transformation (with the lowest skewness) was applied for each feature. A transformation summary table was generated.
- Several custom features were engineered to enhance the predictive power: *dance_energy* = danceability * energy, *valence_tempo* = valence * tempo, *energy_loudness_ratio, acoustic_non_acoustic_ratio, rhythmic_stability*, etc. These features were added to the dataset and standardized where needed.
-A voting-based system was implemented: Five different models were used to compute feature importance(Lasso, Ridge, GradientBoosting, Random Forest, XGBoost). Features were scored and filtered using a custom rule: keep only those that are selected by ≥ 4 models
-Final dataset with selected features were stored in `**df_transformed _features_track2.pkl'**

5. One hot encoding

- StandardScaler was applied to the selected features.

- Boolean features were encoded properly.

6. Regression Modeling

A 3-way split was done: 70% Training, 15% Validation (dev) and 15% Test

Model Selection & Hyperparameter Tuning

- A set of regressors were evaluated using `Pipeline` and `GridSearchCV` :Linear Regression, Decision Tree, Random Forest, AdaBoost, Gradient Boosting, XGBoost and SVR

- The evaluation metrics included: R^2 (coefficient of determination), MAE (Mean Absolute Error), RMSE (Root Mean Squared Error)

- The best model was selected based on the highest R^2 score on the validation set**.**

Model Evaluation on Test Set

- The best model was retrained on the training + validation set (85%).

- Predictions were made on the held-out test set (15%).

- Final test performance was reported using: **R^2,MAE, and RMSE**

Error Visualization

- A scatter plot of actual vs predicted values was plotted.

- A residual histogram was shown to assess error distribution.

- Model comparison barplots (R^2, MAE, RMSE) were included.

Insight

-Despite trying multiple models, **R^2 scores remained modest (≈ 0.26)** indicating the popularity variable is complex and likely influenced by external, unobserved factors (e.g., marketing, social trends).

- Ensemble models like Random Forest and XGBoost outperformed linear ones, justifying their selection despite longer training times.

7. Classification Models

The regression problem was transformed into a classification task. The steps of the process were divided almost like the regression part. The differences are:

Data Preparation

- Dataset was loaded from a transformed pickle file (`df_transformed_features_track2.pkl`), The file includes the selected features from previous models

- Target variable: `track_popularity` was **binarized** into a classification task (`popular >= 50` → 1, else 0).

- Features (`X`) and target (`y`) were split into:70% Training, 15% Validation and 15%Testing using stratified sampling for balanced classes.

Feature Scaling & Pipelines

Each model was used within a `Pipeline` including: `StandardScaler()` for normalization and the classifier model.

Classification Modeling

- The following classifiers were tested using `GridSearchCV`: Decision Tree, Random Forest (with class_weight and SMOTE), AdaBoost, Gradient Boosting, SV, XGBoost

- Each model was fine-tuned using a dedicated hyperparameter grid.

- Evaluation metrics used: Accuracy, Precision, Recall, F1-Score, ROC AUC

Evaluation and Visualizations

- For each model: Confusion matrix, ROC curve, and Precision-Recall curves were generated

- ROC + PR curves were shown **2 per row** for visual comparison

- Final classification performance was compiled into a comparison DataFrame

Insight

- Binarizing popularity (≥ 50 = popular) allowed reframing the problem into a more stable classification task.
- The class distribution was imbalanced, requiring balancing methods like **SMOTE** and **class_weight='balanced'** to avoid bias toward the majority class.
- XGBoost and Random Forest yielded the highest F1-scores and ROC-AUC values, proving effective in this context.

8. Reporting and Stakeholder Communication

To communicate the project's insights and results effectively to both technical and non-technical stakeholders, a structured reporting and visualization strategy was followed:

Visual Reports

A series of data visualizations were created to illustrate key findings during the project lifecycle:

- Distribution plots: to explore the popularity distribution, identify skewness, and detect outliers.
- Correlation heatmaps: to highlight feature relationships and redundancy.
- Scatter plots: to show the relationship between selected audio features and popularity.
- Boxplots and violin plots: for visual outlier detection.
- Trend analysis: line plots showing the evolution of music characteristics (acousticness, tempo, liveness) over time.
- Model performance charts:
  - ROC and Precision-Recall curves for classification models.
  - Barplots comparing R^2, MAE, RMSE for regression models.
  - Confusion matrices for classification results.

Deliverables

- Executive summary and clear documentation of each modeling phase.
- A Word report with key charts embedded and explained.

- Notebooks and scripts for reproducibility.
- Exportable models and cleaned datasets for deployment or integration.

Stakeholder Focus

- Business stakeholders were presented with simplified visuals (e.g., bar plots, trend curves, binary classification results).
- Data science teams received detailed notebooks with code, metrics, and preprocessing decisions.
- Future audiences (e.g., collaborators or clients) can easily follow the process thanks to modular pipelines and clear comments.

Conclusion

This project followed a modular, reproducible, and explainable machine learning pipeline:

- Clean and enriched datasets
- In-depth EDA
- Powerful feature selection
- Comparison of multiple ML models with proper tuning
- Transparent visual and metric-based evaluation

The project successfully transitioned from regression to classification to improve prediction performance. While exact prediction of popularity was challenging, classifying songs as 'popular' or 'not popular' was much more feasible. Future work should focus on enriching the dataset with external data sources and potentially exploring deep learning for better results.