

Report nr 3

Krzysztof Kurzak

krzysztof.kurzak@student.pk.edu.pl

7 czerwca 2025

Spis treści

1 Abstract	2
2 Wprowadzenie	2
3 Zbiór danych	2
4 Motywacja	3
5 Zasoby	4
6 Zastosowane metody	4
6.1 Przetwarzanie danych kategorycznych	4
6.2 Brakujące dane	5
6.3 Skalowanie cech	5
6.4 Analiza danych odstających	5
7 Eksperyment	6
7.1 Preprocessing	6
7.2 Trening modelu	11
7.3 Optymalizacja	15
8 Podsumowanie	23

1 Abstract

Celem projektu, zrealizowanego w ramach przedmiotu Zaawansowane Metody Uczenia Maszynowego II, była analiza oraz przetwarzanie danych w kontekście uczenia maszynowego, a także budowa modelu zdolnego do przewidywania wyników dla nowych danych. Praca obejmuje wybór oraz charakterystykę zbioru danych, przegląd literatury i uzasadnienie badawcze, a także określenie kryteriów oceny sukcesu oraz użytych technologii. W części eksperymentalnej przeprowadzono preprocessing danych, początkowy trening modelu i jego późniejszą optymalizację. Całość dokumentuje proces projektowy od etapu przygotowania danych po uzyskanie finalnych wyników predykcyjnych.

2 Wprowadzenie

Jednym z kluczowych etapów projektu było dobranie odpowiedniego zbioru danych, spełniającego określone wymagania ilościowe i jakościowe. Zbiór powinien zawierać co najmniej 1000 rekordów oraz około 10 cech, co umożliwia przeprowadzenie wiarygodnej analizy statystycznej oraz modelowania. Duży zbiór danych pozwala lepiej uchwycić zależności między poszczególnymi atrybutami, co przekłada się na jakość budowanego modelu. Istotnym kryterium był również udział brakujących danych – preferowano zbiory zawierające około 10% niekompletnych wartości, co umożliwiało praktyczne przetestowanie różnych metod radzenia sobie z brakami. W przypadku braku takich danych w zbiorze, wprowadzono je ręcznie, aby zwiększyć wartość edukacyjną etapu preprocessingu.

3 Zbiór danych

W projekcie wykorzystano zbiór danych „Song Popularity Dataset”, znajdujący się na serwisie Kaggle. Zawiera on informacje dotyczące cech utworów muzycznych i ich popularności. Celem analizy jest przewidywanie, czy dany utwór osiągnie wysoki poziom popularności, na podstawie dostępnych atrybutów. Zbiór obejmuje 13 cech opisujących utwory. Atrybutem decyzyjnym jest kolumna `song_popularity`, która przyjmuje wartości liczbowe od 0 do 100.

Charakterystyka zbioru:

- Liczba próbek: > 18.000
- Liczba cech: 13
- Atrybut docelowy: `song_popularity`

Cechy:

- `song_name` – nazwa utworu
- `song_duration_ms` – długość trwania utworu w milisekundach
- `acousticness` – akustyczność
- `danceability` – taneczność utworu
- `energy` – poziom energii
- `instrumentalness` – poziom instrumentalności
- `liveness` – obecność publiczności
- `loudness` – głośność
- `audio_mode` – tryb audio
- `speechiness` – ilość słów w utworze
- `tempo` – tempo
- `time_signature` – oznaczenie liczby uderzeń
- `audio_valence` – pozytywność utworu

4 Motywacja

Projekt ma charakter edukacyjny i został zrealizowany w ramach zajęć z przedmiotu *Zaawansowane Metody Ucznia Maszynowego II*. Jego zastosowanie w przemyśle muzycznym może być ograniczone, ale może on stanowić punkt wyjścia do opracowania bardziej zaawansowanych systemów rekomendacyjnych lub narzędzi analitycznych wykorzystywanych w branży muzycznej, np. przez wytwórnie płytowe lub platformy streamingowe.

Z punktu widzenia autorów, projekt ten jest szczególnie interesujący ze względu na dobrze opisany zbiór danych, który umożliwia przeprowadzenie głębszej analizy i kompleksowego preprocessingu.

Dzięki dużej liczbie danych i zróżnicowanym cechom opisującym utwory, możliwe jest zaobserwowanie wpływu poszczególnych atrybutów na wynikową popularność utworów. Tego rodzaju analiza może posłużyć do lepszego zrozumienia, jakie właściwości muzyczne przyczyniają się do osiągnięcia sukcesu przez utwór. Projekt daje także możliwość przetestowania różnych strategii radzenia sobie z brakami danych oraz do eksperymentowania z technikami selekcji cech i oceny skuteczności modeli predykcyjnych.

5 Zasoby

Projekt został zaimplementowany w języku Python, z wykorzystaniem środowiska Jupyter Notebook. Notatnik ten umożliwia podział kodu na odrębne komórki, co zwiększa czytelność i ułatwia analizę wyników cząstkowych na poszczególnych etapach realizacji projektu.

W celu przyspieszenia i ułatwienia pracy zastosowano popularne biblioteki Pythona, zawierające gotowe implementacje funkcji i algorytmów związanych z analizą danych oraz uczeniem maszynowym. W projekcie wykorzystano m.in. następujące biblioteki:

- `pandas` – do wczytywania, przetwarzania i analizy danych tabelarycznych,
- `numpy` – do obliczeń numerycznych i operacji na macierzach,
- `matplotlib` i `seaborn` – do wizualizacji danych i wyników,
- `sklearn` – do budowy, trenowania i oceny modeli uczenia maszynowego.

Taki zestaw narzędzi pozwolił na sprawną realizację wszystkich etapów projektu: od wczytania i przygotowania danych, przez analizę eksploracyjną i pre-processing, po budowę i ocenę modeli predykcyjnych.

6 Zastosowane metody

6.1 Przetwarzanie danych kategorycznych

W analizowanym zbiorze danych nie występowały zmienne kategoryczne w formacie tekstowym (z wyjątkiem kolumny `song_name`). Wszystkie cechy miały postać numeryczną (`int64` lub `float64`) i były gotowe do bezpośredniego użycia w modelach uczenia maszynowego.

Choć niektóre kolumny (np. `audio_mode` czy `time_signature`) mają charakter dyskretny, ich wartości są już zakodowane liczbowo i nie wymagają dodatkowego przekształcenia. W związku z tym nie było konieczności stosowania metod kodowania.

Gdyby jednak w zbiorze danych występowały zmienne kategoryczne w postaci tekstuowej (np. nazwy gatunków muzycznych), konieczne byłoby ich zakodowanie. W takim przypadku można by zastosować jedną z poniższych metod:

- **Label encoding** – do kategorii przypisywana jest liczba całkowita. Metoda szybka, ale wprowadza sztuczną kolejność między kategoriami.
- **One-hot encoding** – dla każdej unikalnej wartości tworzona jest osobna kolumna binarna.

6.2 Brakujące dane

Pierwszym krokiem analizy było sprawdzenie, czy w zbiorze danych występują brakujące wartości. Początkowo zbiór nie zawierał braków, dlatego w celu umożliwienia przetestowania różnych metod uzupełniania danych, zasymulowano ich obecność. W kolumnie `danceability` losowo usunięto około 10% wartości.

W celu uzupełnienia brakujących danych przetestowano następujące metody:

- **Średnia** – brakujące wartości zastąpiono średnią wartością kolumny.
- **Mediana** – użyto mediany, co pozwala lepiej poradzić sobie z danymi odstającymi.
- **Najczęstsza wartość** – stosowana dla zmiennych kategorycznych.

Wybór właściwej metody jest zależna od charakteru danych. Dla danych numerycznych takich jak `danceability`, skuteczne są zarówno średnia, jak i mediana. W projekcie wybrano metodę uzupełniania średnią.

6.3 Skalowanie cech

Różnice w skali pomiędzy cechami mogą znacząco wpływać na skuteczność wybranych algorytmów uczenia maszynowego. W związku z tym zastosowano proces skalowania danych, który umożliwia ujednolicenie zakresów wartości dla wszystkich zmiennych numerycznych.

W projekcie przetestowano dwie najpopularniejsze metody:

- **Standaryzacja (StandardScaler)** – przekształcenie danych tak, aby średnia wynosiła 0, a odchylenie standardowe 1.
- **Normalizacja min-max (MinMaxScaler)** – przeskalowanie wartości do przedziału [0, 1].

6.4 Analiza danych odstających

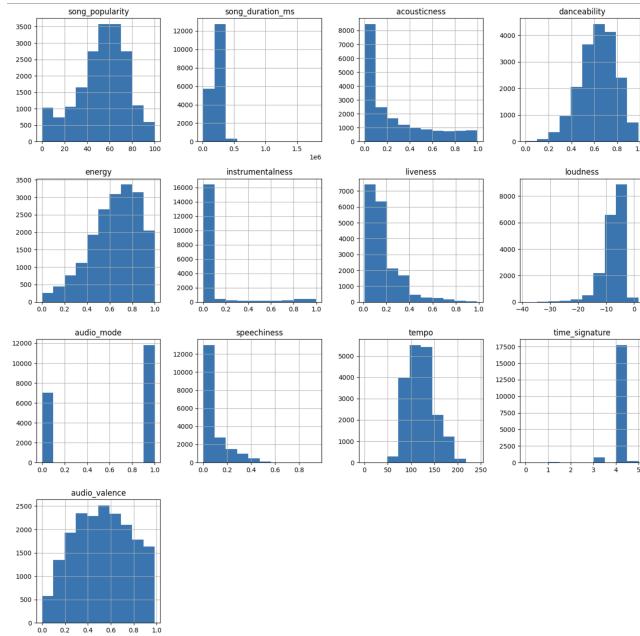
Dane odstające mogą zaburzać rozkłady wartości oraz wpływać negatywnie na skuteczność modeli. W projekcie przeprowadzono analizę występowania takich wartości dla wybranych zmiennych numerycznych (np. `instrumentalness`, `liveness`, `speechiness`) na podstawie wykresów typu boxplot oraz odchyleń standardowych.

Boxploty pozwalają szybko zidentyfikować wartości wykraczające poza typowy zakres danych. Dodatkowo, wyznaczono z-score dla wybranych cech i wskazano wartości oddalone od średniej o więcej niż trzy odchylenia standardowe ($|z| > 3$), co również może świadczyć o ich odstającej naturze.

7 Eksperyment

7.1 Preprocessing

- Zamiana danych tekstowych na wartości liczbowe - badany zbiór danych nie posiadał danych tekstowych więc ten krok został pominięty.
- Histogram – przedstawiono rozkład cech numerycznych. Widać, że wiele z nich ma rozkład niesymetryczny, co może wpływać na działanie modeli. Dodatkowo histogram popularności utworów posłużył do ustalenia progu klasyfikacyjnego.



Rysunek 1: Histogram cech numerycznych

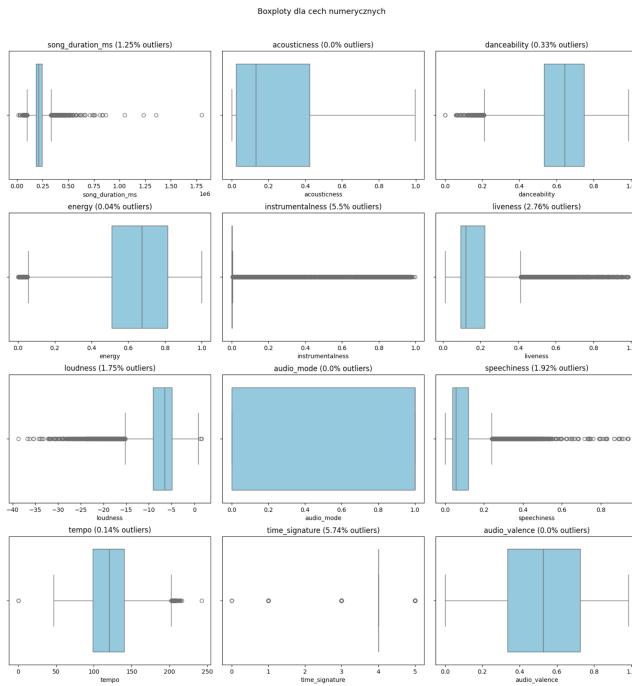
- W celu przekształcenia problemu regresyjnego w klasyfikacyjny, zdecydowano się podzielić utwory na dwie klasy: „popularne” i „niepopularne”. Graniczna wartość popularności została ustalona na poziomie **70 punktów**.

Wybór tej wartości oparto na analizie rozkładu zmiennej `song_popularity`. Histogram pokazał, że większość utworów skupia się w zakresie od około 0 do 70, natomiast wartości powyżej 70 występują rzadziej. Wartość ta stanowi zatem dobrą granicę oddzielającą utwory o ponadprzeciętnej popularności od pozostałych.

Dodatkowo wybór progu 70 zapewnia względną równowagę klas, co jest

istotne dla jakości modeli klasyfikacyjnych i unikania problemu niezbalansowanych danych.

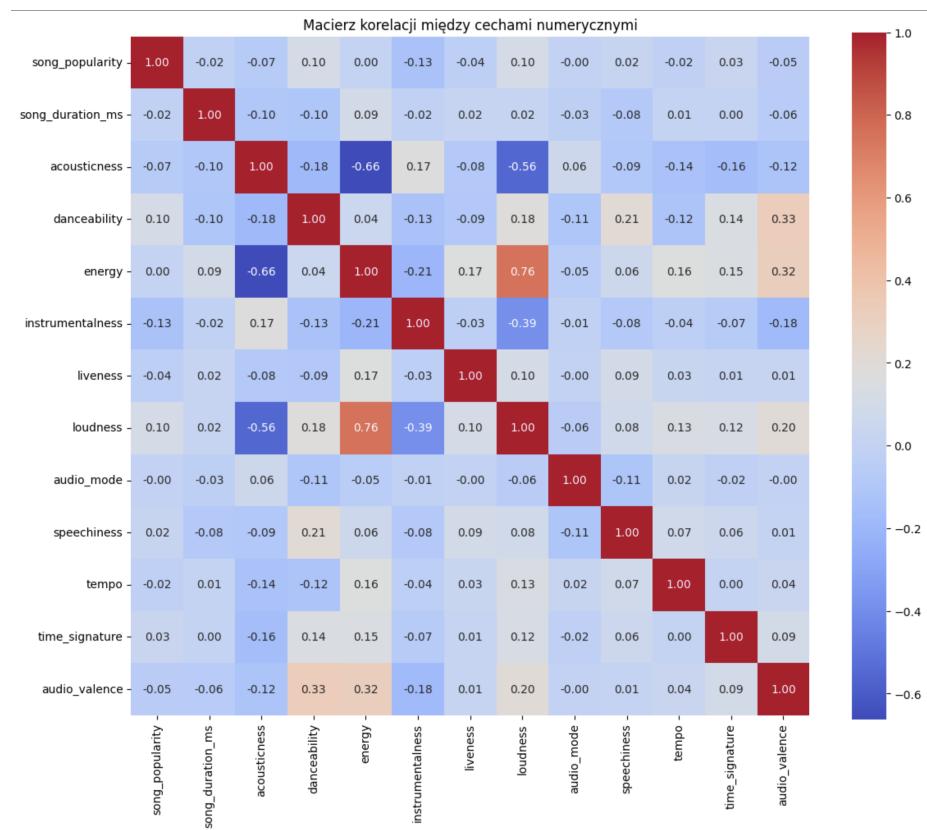
- Boxploty – dla każdej cechy numerycznej wygenerowano wykres typu boxplot. Umożliwia to identyfikację wartości odstających. W tytułach wykresów podano również procentowy udział outlierów (na podstawie z-score).



Rysunek 2: Boxploty dla wszystkich cech numerycznych z zaznaczeniem wartości odstających

- Większość cech (z wyjątkiem `audio_mode` oraz `time_signature`) ma charakter ciągły w przedziale 0–1.
- Rozkłady niektórych cech, na przykład `instrumentalness` i `speechiness`, są wyraźnie skośne. Widać to po skupieniu większości danych w dolnych wartościach oraz obecności wartości odstających w górnej części.
- Zmienna `song_duration_ms` ma szeroki zakres wartości, przy czym jedynie około 1,25% stanowią outlierы, co wskazuje na niewielką liczbę utworów o nietypowo krótkiej lub bardzo długiej długości.
- Zmienne `time_signature` oraz `audio_mode` są dyskretne, co przekłada się na specyficzny wygląd wykresów pudełkowych – dane skoncentrowane są głównie wokół jednej lub dwóch wartości.

- Procent wartości odstających różni się w zależności od cechy: od 0% (np. `acousticness`, `audio_valence`) do aż 5,74% w przypadku `time_signature`, co wskazuje na zróżnicowaną spójność danych dla poszczególnych zmiennych.
- Macierz korelacji – przedstawiono macierz korelacji cech numerycznych z użyciem współczynnika korelacji Pearsona. Dzięki niej możliwa była identyfikacja silnie skorelowanych cech.



Rysunek 3: Macierz korelacji pomiędzy cechami numerycznymi

- Silne zależności:
 - * Wyraźnie dodatnia korelacja między `energy` a `loudness`.
 - * Wyraźnie ujemna korelacja między `acousticness` a `energy` oraz `loudness`.
- Umiarkowane korelacje:
 - * `danceability` oraz `audio_valence` wykazują umiarkowane dodatnie korelacje z parametrami dynamicznymi (energia, głośność),

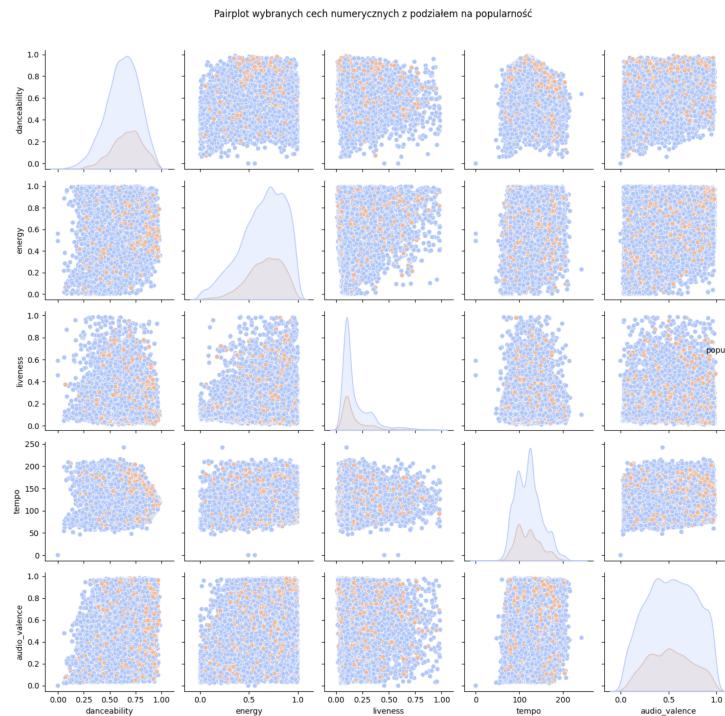
co sugeruje, że utwory bardziej taneczne i „radosne” są zazwyczaj bardziej energetyczne.

- * **song_popularity** ma niewielkie dodatnie zależności z **danceability**, **energy** i **audio_valence**, oraz ujemną korelację z **acousticness**.

– Wnioski ogólne:

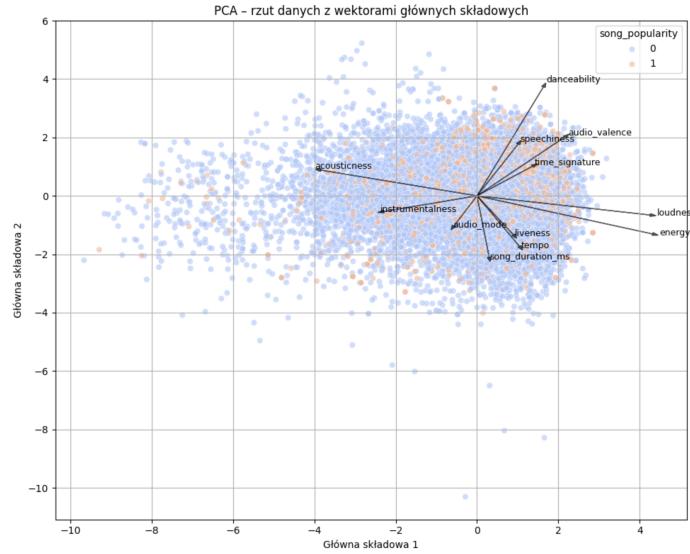
- * Cechy opisujące dynamikę utworu, takie jak **loudness** i **energy**, są ze sobą silnie powiązane.
- * Atrybuty takie jak **acousticness** wpływają istotnie na charakter nagrania – utwory bardziej akustyczne mają tendencję do bycia mniej energetycznymi i głośnymi.
- * Popularność utworu nie jest jednoznacznie determinowana przez pojedyncze cechy audio, lecz wynika z kombinacji różnych parametrów.

- Pairploty – utworzono pairplot dla wybranych cech, z rozróżnieniem klas na podstawie binarnej zmiennej popularności. Pozwoliło to wizualnie ocenić separację klas w przestrzeni cech.



Rysunek 4: Pairplot wybranych cech z rozróżnieniem na klasy „popularne” i „niepopularne”.

- Wykresy punktowe wskazują na duże nachodzenie się danych utworów popularnych i niepopularnych, co sugeruje, że pojedyncze cechy lub ich pary nie pozwalają jednoznacznie rozróżnić obu grup.
 - Dla cech takich jak **danceability**, **energy** oraz **audio_valence** krzywe gęstości dla utworów popularnych są nieco przesunięte w stronę wyższych wartości, co wskazuje na ich statystycznie wyższe średnie.
 - Cechy takie jak **liveness** i **tempo** nie wykazują istotnych różnic między grupami.
 - Utwory popularne częściej występują w obszarach wysokich wartości tych cech, jednak zależności te nie są wystarczająco silne, aby stanowić jednoznaczny czynnik determinujący popularność.
 - Pojedyncze zmienne nie są wystarczające do rozróżnienia utworów popularnych od niepopularnych.
 - Do lepszego zrozumienia popularności, warto rozważyć modele wielowymiarowe, które analizują kombinację cech jednocześnie.
- Redukcja wymiarowości - większa ilość cech w zbiorze może prowadzić do przeuczenia modeli oraz niepotrzebnego zwiększenia złożoności obliczeniowej. W projekcie zastosowano metodę ekstrakcji cech w postaci **analizy głównych składowych (PCA)**.
PCA pozwala przekształcić dane do nowej przestrzeni, w której kolejne składowe główne maksymalizują wariancję danych. Dzięki temu można ograniczyć liczbę cech przy zachowaniu większości informacji zawartej w oryginalnym zbiorze.
Na podstawie wykresu skumulowanej wariancji wyjaśnionej przez kolejne komponenty, zdecydowano się na wybór 8 komponentów, które łącznie tłumaczą około 84% całkowitej wariancji danych.
Redukcja wymiarowości umożliwiła uproszczenie przestrzeni cech oraz poprawę wydajności modeli predykcyjnych bez utraty istotnych informacji.



Rysunek 5: PCA - rzut danych z wektorami cech

- Podobny kierunek strzałek (np. `loudness` i `energy`) wskazuje na wysoką dodatnią korelację między cechami.
- Brak wyraźnej granicy między utworami popularnymi a niepopularnymi (jeśli kolory się przenikają), co sugeruje, że te dwie pierwsze składowe nie gwarantują silnego rozdziału klas.
- Dwie pierwsze składowe tłumaczą jedynie 35% wariancji w badanym zbiorze danych.

7.2 Trening modelu

Celem drugiego etapu projektu było porównanie skuteczności różnych modeli klasyfikacyjnych w zadaniu przewidywania popularności utworu muzycznego na podstawie jego cech. Proces ten został zrealizowany w kilku krokach, obejmujących przygotowanie danych, trenowanie modeli klasycznych, zespołowych oraz ich ocenę z użyciem ręcznych metryk i walidacji krzyżowej.

- Dane zostały podzielone na zbiór treningowy (80%) oraz testowy (20%) z wykorzystaniem ręcznie zaimplementowanego podziału stratyfikowanego. Dane dla klas pozytywnej i negatywnej zostały najpierw rozdzielone, a następnie z każdej z nich wylosowano odpowiedni procent próbek do zbioru testowego, aby zachować proporcje klas. Finalnie zbiory zostały ponownie przemieszane w sposób kontrolowany (`random_state=42`).

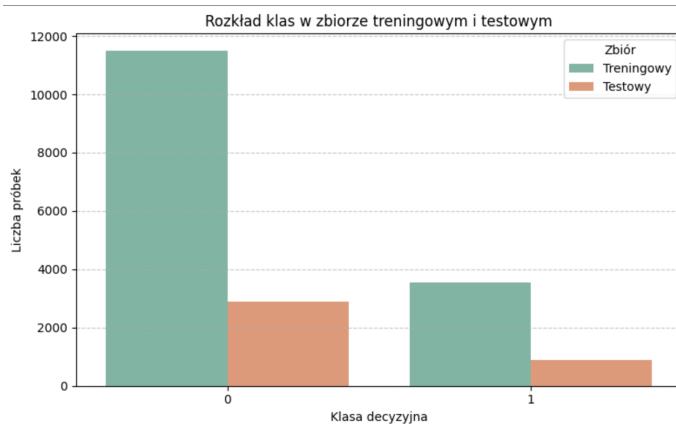
- Implementacja funkcji podziału:

```

1 def manual_stratified_split(X, y, test_size=0.2, random_state=42):
2     data = X.copy()
3     data['label'] = y
4     class_0 = data[data['label'] == 0]
5     class_1 = data[data['label'] == 1]
6     class_0 = shuffle(class_0, random_state=random_state)
7     class_1 = shuffle(class_1, random_state=random_state)
8     n_test_0 = int(len(class_0) * test_size)
9     n_test_1 = int(len(class_1) * test_size)
10    test_data = pd.concat([class_0.iloc[:n_test_0], class_1.iloc[:n_test_1]])
11    train_data = pd.concat([class_0.iloc[n_test_0:], class_1.iloc[n_test_1:]])
12    test_data = shuffle(test_data, random_state=random_state)
13    train_data = shuffle(train_data, random_state=random_state)
14    X_train = train_data.drop(columns='label')
15    y_train = train_data['label']
16    X_test = test_data.drop(columns='label')
17    y_test = test_data['label']
18    return X_train, X_test, y_train, y_test

```

Listing 1: Funkcja manualnego podziału z uwzględnieniem proporcji klas



Rysunek 6: Rozkład klas w zbiorze treningowym i testowym

- Dane zostały przeskalowane za pomocą **StandardScaler**. Proces ten wykonano osobno dla zbioru treningowego i testowego.

- Własna funkcja do liczenia metryk klasyfikacyjnych:

```

1 def manual_metrics(y_true, y_pred):
2     TP = np.sum((y_true == 1) & (y_pred == 1))
3     TN = np.sum((y_true == 0) & (y_pred == 0))
4     FP = np.sum((y_true == 0) & (y_pred == 1))
5     FN = np.sum((y_true == 1) & (y_pred == 0))
6     accuracy = (TP + TN) / (TP + TN + FP + FN)
7     precision = TP / (TP + FP) if (TP + FP) else 0
8     recall = TP / (TP + FN) if (TP + FN) else 0
9     f1 = 2 * precision * recall / (precision + recall) if (
precision + recall) else 0
10    return accuracy, precision, recall, f1

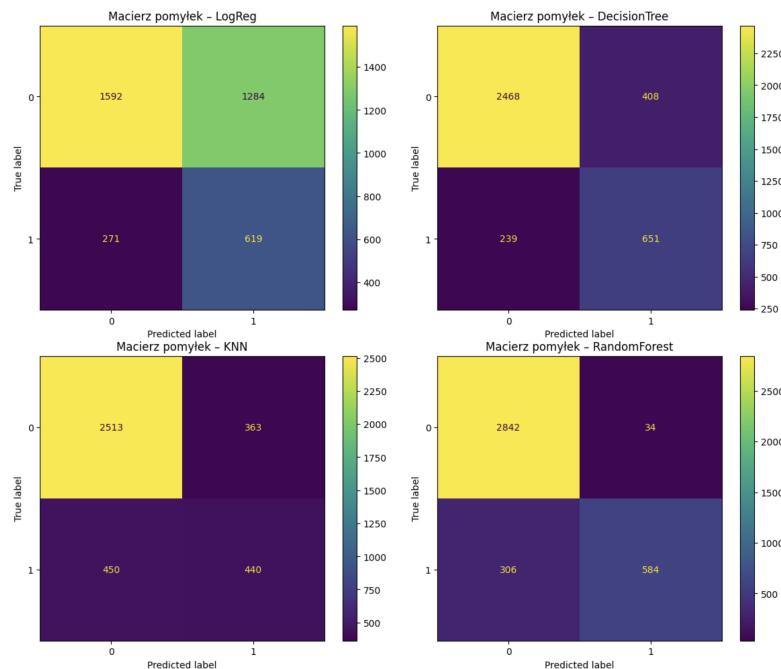
```

Listing 2: Funkcja obliczająca accuracy; precision; recall; f1

- Przetestowano następujące modele klasyczne:

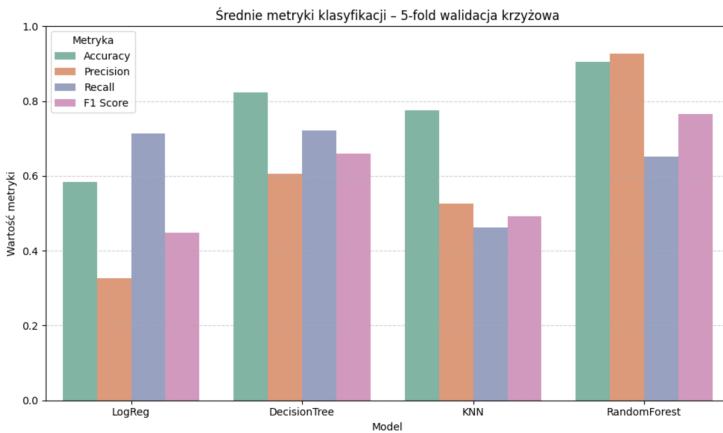
- Logistic Regression (z parametrami `class_weight='balanced'`, `solver='liblinear'`, `C=0.1`)
- Decision Tree
- K-Nearest Neighbors
- Random Forest

Każdy model był trenowany na `X_train_scaled` i oceniany na `X_test_scaled`.



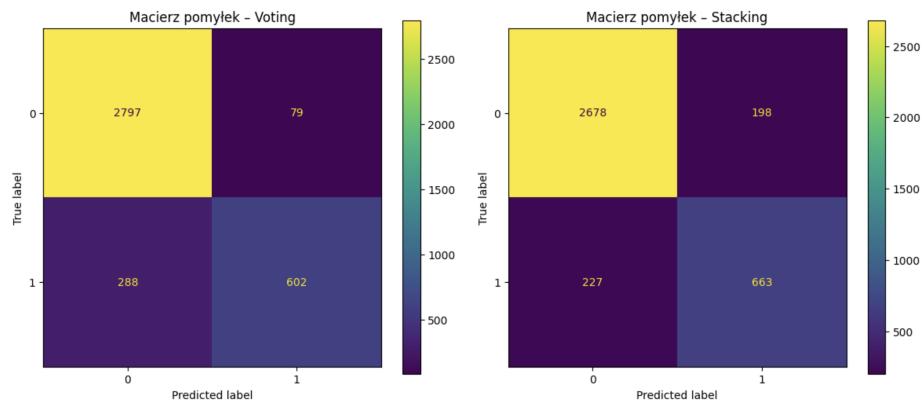
Rysunek 7: Macierz pomyłek dla klasycznych modeli

- Zastosowano ręczną walidację krzyżową (5-fold) przy użyciu `StratifiedKFold`. W każdej iteracji dane treningowe były dzielone na nowe foldy, trenowane model, a następnie oceniano go funkcją `manual_metrics`. Wyniki zostały uśrednione.



Rysunek 8: Wyniki z walidacji krzyżowej

- Zaimplementowano dwa modele zespołowe:
 - `VotingClassifier` z modelami `LogisticRegression`, `RandomForest`, `XGBBoost`
 - `StackingClassifier` z `LogisticRegression` i `RandomForest` (meta-modelem była również regresja logistyczna)



Rysunek 9: Macierze pomyłek modeli zespołowych

Tabela 1: Wyniki modeli klasyfikacyjnych na zbiorze testowym

Model	Accuracy	Precision	Recall	F1 Score
RandomForest	0.910	0.945	0.656	0.775
Voting	0.903	0.884	0.676	0.766
Stacking	0.887	0.770	0.745	0.757
XGBoost	0.877	0.843	0.591	0.695
DecisionTree	0.828	0.615	0.731	0.668
CatBoost	0.839	0.848	0.389	0.533
LightGBM	0.838	0.867	0.373	0.522
KNN	0.784	0.548	0.494	0.520
LogisticRegression	0.587	0.325	0.696	0.443

- Na podstawie wyników zaprezentowanych w tabeli można zauważyć, że najlepsze rezultaty na zbiorze testowym osiągnął model RandomForest, uzyskując najwyższą dokładność i precyzyję oraz najwyższy wynik F1-score. Modele zespołowe VotingClassifier oraz StackingClassifier również wykazały się bardzo dobrą skutecznością, przy czym VotingClassifier osiągnął najwyższą wartość czułości (Recall), natomiast StackingClassifier zbalansował metryki najrówniej.

Model XGBoost uzyskał wysoki wynik precyzyji, ale niższy Recall, co wpłynęło na ogólną wartość F1-score. DecisionTree wykazał się solidnym Recall, lecz słabszą precyzyją.

Modele CatBoost oraz LightGBM miały wysoką precyzyję, ale bardzo niską czułość, co negatywnie wpłynęło na końcowy wynik F1. KNN uzyskał wyniki przeciętne, a LogisticRegression choć miał wysoki Recall, to niska precyzyja sprawiła, że jego F1-score był najsłabszy spośród wszystkich modeli.

- Wnioski: model RandomForest uzyskał najlepszy ogólny wynik na zbiorze testowym. Zastosowanie uczenia zespołowego (ensemble) pozwoliło dodatkowo poprawić jakość predykcji i zbalansować metryki.

7.3 Optymalizacja

Celem trzeciego etapu projektu była optymalizacja cech oraz hiperparametrów modeli w celu zwiększenia skuteczności klasyfikacji popularności utworu. Etap ten obejmował zastosowanie różnych metod selekcji cech, a także trzech niezależnych podejść do strojenia parametrów: GridSearch, Optuna oraz TPOT.

Wybór cech i redukcja wymiarowości

SelectKBest

Zastosowano metodę **SelectKBest** z funkcją `f_classification`, aby wybrać dziesięć najbardziej istotnych cech względem zmiennej docelowej:

```

1 kbest = SelectKBest(score_func=f_classif, k=10)
2 X_kbest = kbest.fit_transform(X, y)
3 print(X.columns[kbest.get_support()])

```

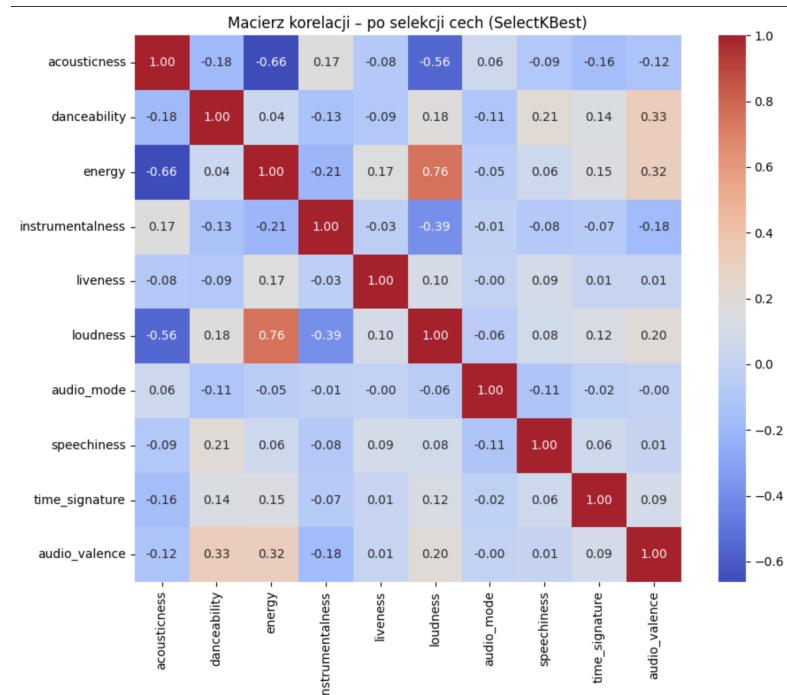
Listing 3: Fragment kodu do wyboru najbardziej istotnych cech

W wyniku selekcji wybrano następujące cechy:

```
acousticness, danceability, energy, instrumentalness, liveness,
loudness, audio_mode, speechiness, time_signature, audio_valence
```

Macierz korelacji po selekcji cech (SelectKBest)

Po zastosowaniu algorytmu `SelectKBest` wybrano dziesięć najistotniejszych cech numerycznych. Dla tych atrybutów obliczono macierz korelacji z użyciem współczynnika Pearsona, co umożliwiło analizę zależności pomiędzy wybranymi cechami.



Rysunek 10: Macierz korelacji między wybranymi cechami (SelectKBest)

Najważniejsze obserwacje:

- **Silna dodatnia korelacja** między cechami:
 - **energy** i **loudness** ($r = 0.76$) – dynamiczne utwory są jednocześnie głośniejsze.

- Silna ujemna korelacja:
 - `acousticness` i `energy` ($r = -0.66$) – spokojne, akustyczne piosenki są mniej energetyczne.
 - `acousticness` i `loudness` ($r = -0.56$) – ciche utwory cechują się większą akustycznością.
- Większość pozostałych korelacji jest znaczco niższa, co oznacza niewielką zależność między zmiennymi takimi jak `liveness`, `speechiness`, `time_signature` czy `audio_mode`.

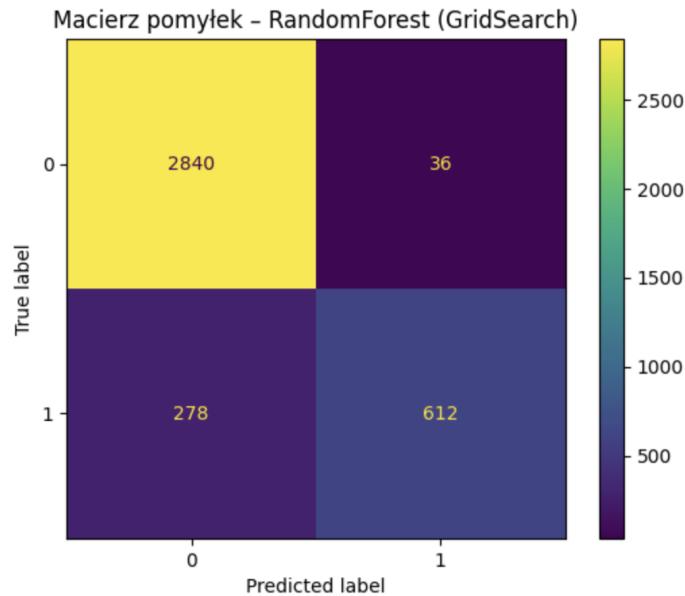
Optymalizacja hiperparametrów

W celu poprawy jakości predykcji przeprowadzono optymalizację hiperparametrów dla najlepszego modelu – Random Forest.

GridSearch

Łącznie przetestowano 36 kombinacji hiperparametrów, a każda z nich była oceniana w pięciokrotnej walidacji krzyżowej. Całkowita liczba wytrenowanych modeli wyniosła 180.

```
1 'max_depth': None, 'min_samples_leaf': 1,
2 'min_samples_split': 2, 'n_estimators': 300
```



Rysunek 11: Macierz pomyłek GridSearch

Wyniki klasyfikacji na zbiorze testowym:

- Accuracy: 91.7%
- Precision: 94.4%
- Recall: 68.8%
- F1 Score: **0.796**

Raport klasyfikacji:

	precision	recall	f1-score	support
0	0.911	0.987	0.948	2876
1	0.944	0.688	0.796	890
accuracy			0.917	3766
macro avg	0.928	0.838	0.872	3766
weighted avg	0.919	0.917	0.912	3766

Wnioski:

- Optymalizacja parametrów modelu RandomForest za pomocą GridSearch poprawiła jego wynik względem domyślnej konfiguracji.
- Poprawie uległ głównie **F1-score** (z 0.775 do 0.796), co świadczy o lepszym kompromisie między czułością a precyzją.
- Utrzymana została bardzo wysoka precyzja (94.4%) – oznacza to, że większość utworów przewidywanych jako „popularne” rzeczywiście nimi jest.
- Czułość (Recall) nadal pozostaje umiarkowana – model nie wykrywa wszystkich utworów popularnych, ale czyni to bardzo precyzyjnie.

Optuna

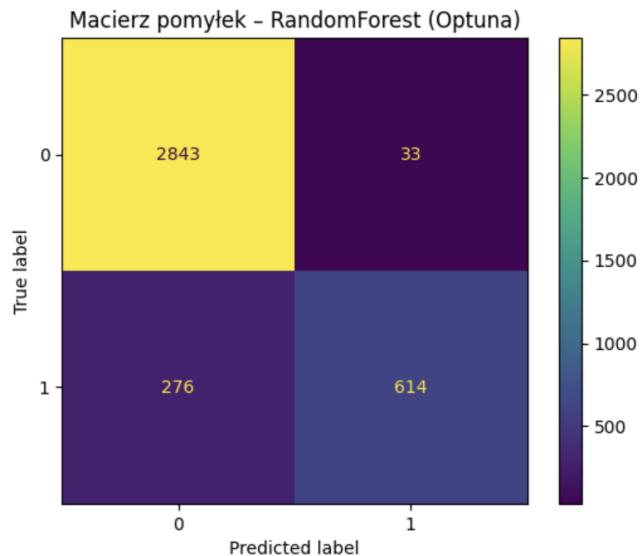
W celu dalszej poprawy skuteczności modelu RandomForest zastosowano bibliotekę Optuna, która umożliwia wydajną optymalizację hiperparametrów przy użyciu strategii opartych na przeszukiwaniu bayesowskim. Proces ten pozwala na inteligentne wybieranie prób na podstawie wcześniejszych wyników, co często prowadzi do lepszych rezultatów niż GridSearch przy mniejszej liczbie iteracji.

Zakres przeszukiwanych parametrów:

- `n_estimators`: od 100 do 300
- `max_depth`: od 5 do 30
- `min_samples_split`: od 2 do 10
- `min_samples_leaf`: od 1 do 5
- `class_weight`: 'balanced'

Do oceny każdego zestawu parametrów zastosowano pięciokrotną validację krzyżową z metryką F1-score.

```
1 {
2   'n_estimators': 161,
3   'max_depth': 29,
4   'min_samples_split': 2,
5   'min_samples_leaf': 1
6 }
```



Rysunek 12: Macierz pomyłek Optuna

Wyniki klasyfikacji na zbiorze testowym:

- **Accuracy:** 91.8%
- **Precision:** 94.9%
- **Recall:** 69.0%
- **F1 Score:** **0.799**

Raport klasyfikacji:

	precision	recall	f1-score	support
0	0.912	0.989	0.948	2876
1	0.949	0.690	0.799	890
accuracy			0.918	3766
macro avg	0.930	0.839	0.874	3766
weighted avg	0.920	0.918	0.913	3766

Wnioski:

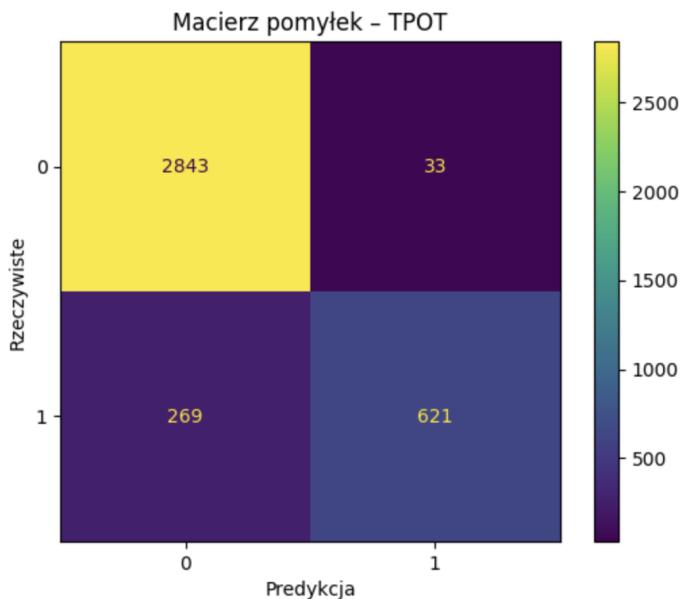
- Optuna osiągnęła nieznacznie lepszy wynik niż GridSearch – poprawa F1-score z **0.796** do **0.799**.
- Czułość (Recall) klasy „popularnych” utworów wzrosła minimalnie, co jest istotne przy problemie z niezbalansowanymi danymi.
- Model pozostał bardzo precyzyjny przy klasyfikacji utworów popularnych – precision na poziomie 94.9%.

TPOT

Aby jeszcze bardziej poprawić skuteczność modelu RandomForest, w ostatnim kroku zastosowano narzędzie TPOT – automatycznego optymalizatora modeli uczenia maszynowego, wykorzystującego algorytmy ewolucyjne do wyszukiwania najlepszych pipeline'ów. TPOT generuje, ocenia i ewoluje setki różnych kombinacji modeli, preprocessingu i parametrów, co czyni go narzędziem klasy AutoML.

Parametry TPOT:

- `generations`: 45
- `population_size`: 75
- `cv`: 5
- `scoring`: `f1_score` (ustawiony ręcznie)
- `max_time_mins`: 500
- `n_jobs`: 6 (przetwarzanie równoległe)



Rysunek 13: Macierz pomyłek TPOT

Wyniki klasyfikacji na zbiorze testowym:

- **Accuracy:** 90.2%
- **Precision:** 95.0%
- **Recall:** 69.8%
- **F1 Score:** 0.804

Raport klasyfikacji – TPOT:

	precision	recall	f1-score	support
0	0.914	0.989	0.950	2876
1	0.950	0.698	0.804	890
accuracy			0.920	3766
macro avg	0.932	0.843	0.877	3766
weighted avg	0.922	0.920	0.915	3766

Uwagi techniczne:

- TPOT korzystał z pełnej przestrzeni operatorów i modeli.
- Czas trwania optymalizacji wyniósł ponad 4 godziny.

Wnioski:

- Zachowana została wysoka precyza (95%) przy bardzo dobrym ogólnym dopasowaniu modelu (accuracy 92%).
- Choć czas obliczeń był znaczny, TPOT potwierdził swoją skuteczność jako narzędzie klasy AutoML w znajdowaniu optymalnych rozwiązań bez ręcznej konfiguracji.

Porównanie metod optymalizacji hiperparametrów

W poniższej tabeli zestawiono metryki uzyskane przez model RandomForest po zastosowaniu trzech różnych strategii optymalizacji hiperparametrów: **GridSearchCV**, **Optuna** oraz **TPOT**.

Tabela 2: Porównanie wyników optymalizacji hiperparametrów

Metoda	Accuracy	Precision	Recall	F1 Score
GridSearchCV	0.917	0.944	0.688	0.796
Optuna	0.918	0.949	0.690	0.799
TPOT	0.920	0.950	0.698	0.804

Wnioski:

- Wszystkie trzy metody doprowadziły do uzyskania bardzo podobnych i wysokich wyników, jednak **TPOT** okazał się najskuteczniejszy, uzyskując najwyższy **F1-score = 0.804** oraz najwyższą **accuracy = 0.920**.
- **Optuna** osiągnęła drugie miejsce w tej klasyfikacji.
- **GridSearchCV** uzyskał nieco niższy wynik F1 i recall w porównaniu do metod automatycznych, ale był najszybszy do wykonania.
- TPOT jako podejście AutoML dostarcza najlepsze wyniki kosztem dłużego czasu treningu – warto rozważyć jego użycie w zadaniach, gdzie czas nie jest krytyczny, a liczy się maksymalizacja skuteczności.

8 Podsumowanie

Celem projektu było zbudowanie modelu klasyfikującego utwory muzyczne na podstawie ich cech audio do kategorii „popularne” i „niepopularne”. W ramach projektu przeprowadzono kompleksowy proces analityczny, obejmujący eksplorację danych, wstępny preprocessing, transformację problemu regresji do klasyfikacji binarnej, redukcję wymiarowości, a także selekcję cech i wieloetapową optymalizację modeli klasyfikacyjnych.

Główne osiągnięcia

- Wdrożono zaawansowane techniki preprocessingowe, w tym symulację braków danych, analizę danych odstających, skalowanie cech oraz wizualizacje rozkładów.
- Przekształcono problem regresyjny w zadanie klasyfikacji binarnej z zachowaniem zrównoważonego rozkładu klas.

- Przeprowadzono analizę korelacji i selekcję cech (SelectKBest) oraz redukcję wymiarowości (PCA), co pomogło zwiększyć skuteczność modeli.
- Przetestowano klasyczne algorytmy klasyfikacyjne i modele zespołowe, w tym: `RandomForest`, `StackingClassifier`, `VotingClassifier`, `XGBoost`, `CatBoost`, `LightGBM`, `Logistic Regression`, `KNN` i `DecisionTree`.
- Przeprowadzono optymalizację hiperparametrów dla najlepszego modelu (`RandomForest`) przy użyciu trzech metod: `GridSearchCV`, `Optuna` oraz `TPOT`.

Wnioski końcowe

- Najlepszym pojedynczym modelem klasyfikacyjnym okazał się **RandomForest**, który uzyskał najwyższy wynik F1-score w każdej wersji optymalizacyjnej.
- Automatyczna optymalizacja z użyciem **TPOT** zapewniła najlepszy ogólny wynik modelu ($F1 = 0.804$), ale była również najbardziej czasochłonna.
- Modele zespołowe typu `Voting` i `Stacking` osiągały wyniki zbliżone do `RandomForest`, ale nie przewyższały go w metryce F1.
- Popularność utworu nie jest bezpośrednio determinowana przez pojedynczą cechę audio – liczy się kombinacja wielu właściwości akustycznych.
- Projekt udowadnia skuteczność klasyfikatorów drzewiastych w analizie danych muzycznych i wskazuje na wysoką przydatność narzędzi typu AutoML w kontekście automatyzacji doboru modeli i parametrów.