Klasyfikacja gatunków i podgatunków muzycznych

Autorzy: Yury Yarmishyn, Ivan Matveichyk

Streszczenie

Celem projektu było opracowanie modelu predykcyjnego do klasyfikacji gatunku i podgatunku muzycznego utworów na podstawie ich cech akustycznych. Do analizy wykorzystano zbiór danych zawierający informacje o utworach muzycznych, w tym zmienne takie jak danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo i duration. Przeprowadzono wstępną analizę danych, obliczono statystyki opisowe, a także zastosowano metody wizualizacji, takie jak boxploty i histogramy. Zastosowano trzy metody klasyfikacji: RandomForest, SVM i KNN, oraz utworzono model hybrydowy wykorzystujący głosowanie ważone tych metod. Modele zostały ocenione za pomocą wskaźników takich jak Accuracy i raporty klasyfikacyjne. Model hybrydowy osiągnął najlepsze wyniki, co sugeruje, że kombinacja różnych podejść do klasyfikacji może prowadzić do bardziej dokładnych predykcji gatunków muzycznych.

Słowa kluczowe

- Klasyfikacja
- Muzyka
- RandomForest
- SVM
- KNN
- Analiza
- Predykcja
- Gatunki
- Podgatunki

Wprowadzenie

Wprowadzenie do projektu obejmuje ogólną charakterystykę problemu klasyfikacji gatunku i podgatunku muzycznego na podstawie cech akustycznych utworów. Opisuje również cel i zakres pracy, który skupia się na porównaniu skuteczności różnych algorytmów klasyfikacji oraz na tworzeniu modelu hybrydowego.

Przedmiot badania

Przedmiotem badania są utwory muzyczne, dla których dostępne są dane opisujące ich cechy akustyczne. Zbiór danych zawiera informacje takie jak danceability, energy, key, loudness, mode, speechiness, acousticness, instrumentalness, liveness, valence, tempo i duration.

Cel

Celem badania jest opracowanie skutecznego modelu predykcyjnego do klasyfikacji gatunku i podgatunku muzycznego, a także porównanie skuteczności różnych algorytmów klasyfikacji.

Wstępna analiza danych

Wstępna analiza danych obejmowała obliczenie statystyk opisowych (średnia, mediana, minimum, maksimum, odchylenie standardowe, skośność) dla każdej zmiennej. Przeprowadzono również wizualizację danych za pomocą boxplotów i histogramów.

Opis zbioru danych

- Zbiór danych zawiera około 20 zmiennych opisujących piosenkę
- Istnieją zmienne, które będą podstawą uczenia maszynowego (gatunek i podgatunek)
- Istnieją zmienne liczbowe, które charakteryzują gatunek i podgatunek
- Istnieją zmienne identyfikacyjne, które pomagają zidentyfikować piosenkę
- Zbiór zawiera łącznie 32833 piosenki

Zmienne w zbiorze danych

- track_id
- artists
- album_name
- track_name
- popularity
- duration_ms
- explicit
- danceability
- energy
- key
- loudness
- mode
- speechiness
- Acousticness
- instrumentalness
- liveness
- valence
- tempo
- time_signature
- track_genre

Optymalizacja zmiennych, pt.1

Nie wszystkie zmienne ze zbioru są potrzebne do wytrenowania modelu, więc stworzyliśmy poniższą listę zmiennych do analizy

- **Name**: nazwa piosenki, zmienna identyfikacyjna, nie ma wplywu na analize
- **Artist**: imie grupy/twórca, zmienna identyfikacyjna, nie ma wplywu na analize
- **Genre**: gatunek, jest zmienną dla trenowania modelu
- **Subgenre**: podgatunek, jest zmienną dla trenowania modelu
- **Danceability**: taneczność, zmienna liczbowa, opisuje, jak odpowiedni jest utwór do tańca od 0 do 1
- **Energy**: energia, zmienna liczbowa, jest miarą od 0,0 do 1,0 i reprezentuje percepcyjną miarę intensywności i aktywności
- **Key**: klucz, zmienna liczbowa, pokazuje klucz, w którym znajduje się piosenka. Liczby całkowite są mapowane na wysokości dźwięku przy użyciu standardowej notacji Pitch Class (C = 0, C♯/D♭ = 1, D = 2, D♯/E♭ = 3, E = 4). Jeśli nie wykryto żadnego klucza, wartość wynosi -1

Optymalizacja zmiennych, pt.2

- **Loudness**: głośność, zmienna liczbowa, ogólna głośność utworu w decybelach (dB)
- **Mode**: tryb, zmienna liczbowa, wskazuje modalność (dur lub moll) utworu, rodzaj skali, z której pochodzi jego treść melodyczna. Major jest reprezentowany przez 1, a minor przez 0
- **Speechiness**: mówność, zmienna liczbowa, wykrywa obecność wypowiadanych słów w utworze
- **Acousticness**: akustyczność, zmienna liczbowa, miara zaufania od 0,0 do 1,0 określająca, czy utwór jest akustyczny. 1.0 oznacza wysokie zaufanie, że utwór jest akustyczny
- **Instrumentalness**: instrumentalność, zmienna liczbowa, przewiduje, czy utwór nie zawiera wokalu. Im wartość instrumentalności jest bliższa 1,0, tym większe prawdopodobieństwo, że utwór nie zawiera treści wokalnych
- **Liveness**: publiczność, zmienna liczbowa, wykrywa obecność publiczności w nagraniu. Wyższe wartości liveness oznaczają zwiększone prawdopodobieństwo, że utwór został wykonany na żywo
- **Valence**: pozytywność, zmienna liczbowa, miara od 0,0 do 1,0 opisująca muzyczną pozytywność przekazywaną przez utwór
- **Tempo**: tempo, zmienna liczbowa, ogólne szacowane tempo utworu w uderzeniach na minutę (BPM)
- **Duration**: dlugość, zmienna liczbowa, pokazuje długość piosenki w milisekundach

Przetwarzanie danych

W kodzie proces przetwarzania danych z pliku tekstowego odbywa się w następujący sposób:

- Funkcja create_track_list(file_path) otwiera wskazany plik tekstowy i czyta go wiersz po wierszu.
- Każdy wiersz pliku zawiera informacje o pojedynczym utworze muzycznym. Dla każdego wiersza następuje podział danych za pomocą przecinków, aby wydobyć odpowiednie wartości dla każdego atrybutu utworu.
- Tworzony jest obiekt klasy Track, do którego przekazywane są wydobyte wartości atrybutów utworu.
- Ten obiekt jest dodawany do listy track_list.
- Na końcu funkcji zwracana jest lista track_list, zawierająca obiekty klasy Track, z których każdy reprezentuje pojedynczy utwór.

Funkcja create_track_list(file_path)

```python
def create_track_list(file_path):
    track_list = []
    with open(file_path, 'r', encoding='utf-8') as file:
        for line in file:
            data = []
            variable = ""
            is_quotes = False

            for char in line:
                if char == '"':
                    is_quotes = not is_quotes
                if char == ',' and not is_quotes:
                    data.append(variable)
                    variable = ''
                else:
                    variable += char

            data.append(variable)

            track = Track(
                name=data[1],
                artist=data[2],
                genre=data[9],
                subgenre=data[10],
                danceability=data[11],
                energy=data[12],
                key=data[13],
                loudness=data[14],
                mode=data[15],
                speechiness=data[16],
                acousticness=data[17],
                instrumentalness=data[18],
                liveness=data[19],
                valence=data[20],
                tempo=data[21],
                duration=data[22]
            )
            track_list.append(track)
    return track_list
```

Class Track

```python
class Track:
    # Track initialization
    Stan-Nip *
    def __init__(self, name, artist, genre, subgenre, danceability,
                 energy, key, loudness, mode, speechiness, acousticness,
                 instrumentalness, liveness, valence, tempo, duration):
        self.name = name
        self.artist = artist
        self.genre = genre
        self.subgenre = subgenre
        self.danceability = float(danceability)
        self.energy = float(energy)
        self.key = int(key)
        self.loudness = float(loudness)
        self.mode = int(mode)
        self.speechiness = float(speechiness)
        self.acousticness = float(acousticness)
        self.instrumentalness = float(instrumentalness)
        self.liveness = float(liveness)
        self.valence = float(valence)
        self.tempo = float(tempo)
        self.duration = int(duration)

    # Show data
    Stan-Nip
    def __repr__(self):
        return (f"Track(name={self.name}, artist={self.artist}, genre={self.genre}, subgenre = {self.subgenre}, danceability={self.danceability}, "
                f"energy={self.energy}, key={self.key}, loudness={self.loudness}, mode={self.mode}, speechiness={self.speechiness}, "
                f"acousticness={self.acousticness}, instrumentalness={self.instrumentalness}, liveness={self.liveness}, "
                f"valence={self.valence}, tempo={self.tempo}, duration={self.duration})")
```

Obliczenia statystyk opisowych

Dane zostały obliczone przy użyciu funkcji calculate_statistics

```python
def calculate_statistics(tracks):
    numeric_parameters = ['danceability', 'energy', 'key', 'loudness', 'mode',
                          'speechiness', 'acousticness', 'instrumentalness',
                          'liveness', 'valence', 'tempo', 'duration']
    all_stats = {}

    for parameter in numeric_parameters:
        values = [getattr(track, parameter) for track in tracks if isinstance(getattr(track, parameter), (int, float))]

        if not values:
            continue

        stats = {}
        stats['mean'] = statistics.mean(values)
        stats['median'] = statistics.median(values)
        stats['min'] = min(values)
        stats['max'] = max(values)
        stats['std_dev'] = statistics.stdev(values) if len(values) > 1 else 0.0
        stats['skewness'] = calc_skew(values) if len(values) > 1 else 0.0
```

calc_skew została pobrana `from scipy.stats import skew as calc_skew`

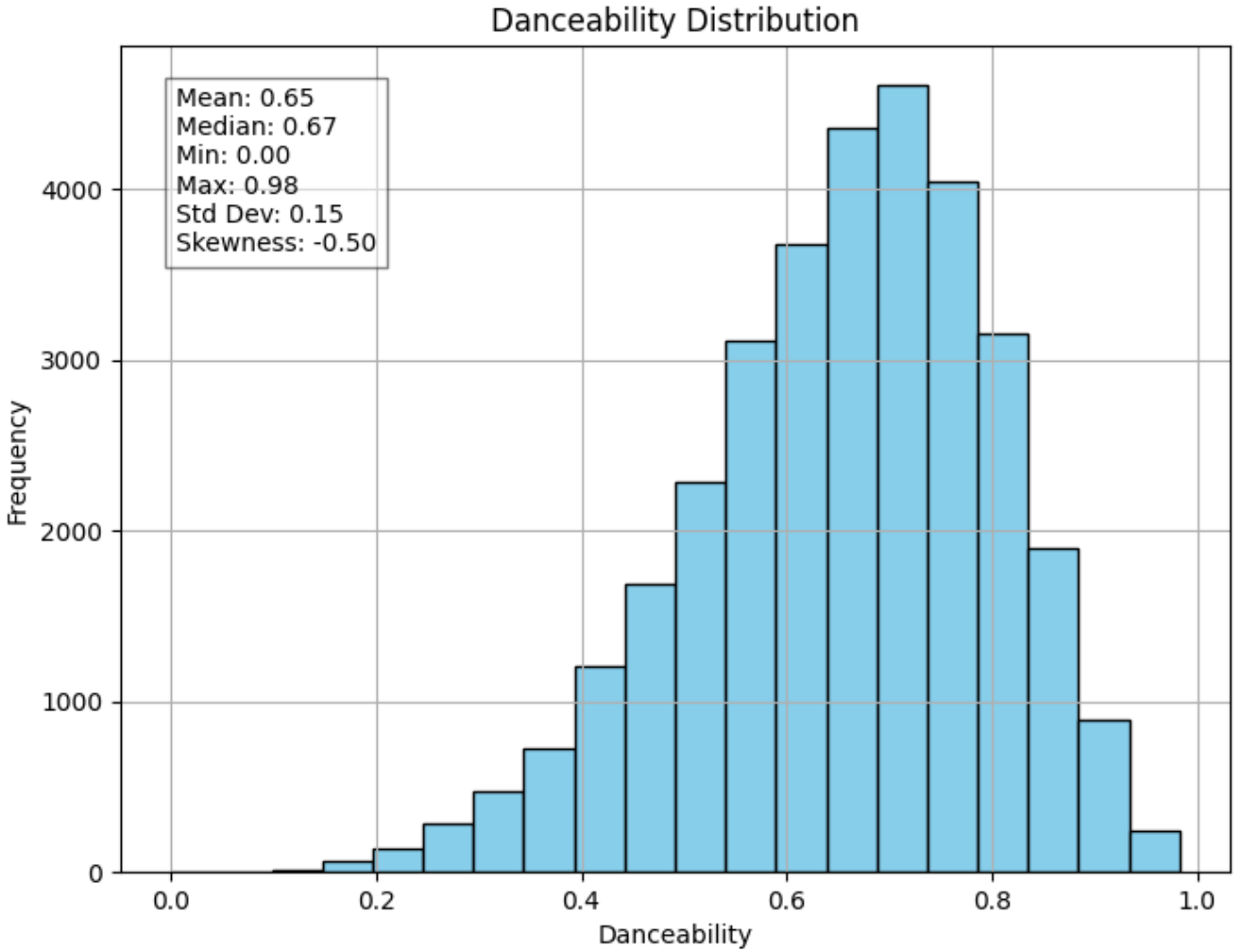
Pozostałe dane zostały obliczone przy użyciu biblioteki `import statistics`

Wizualizacja statystyk opisowych

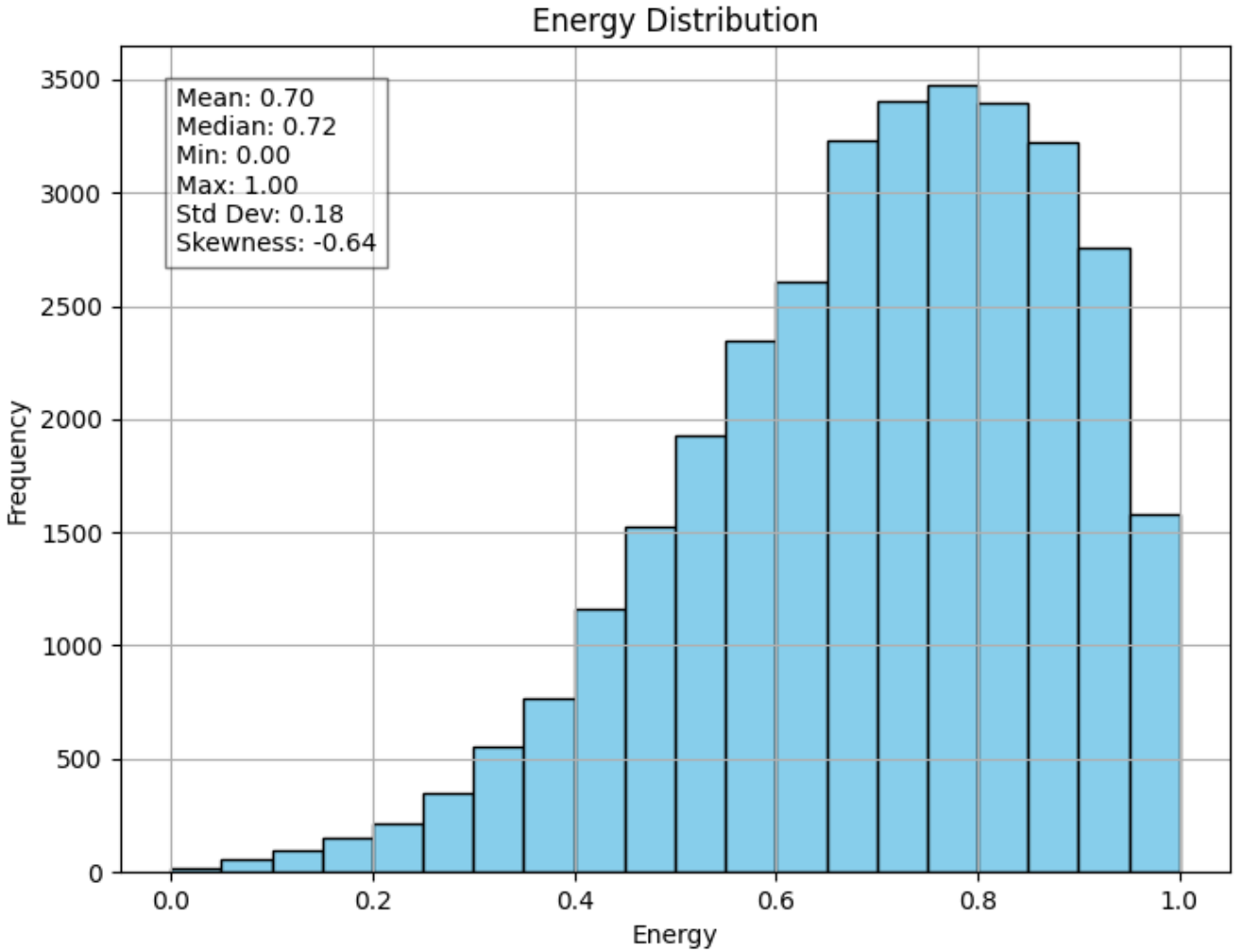
Dane zostały zwizualizowane też za pomocą funkcji calculate_statistics (niżej druga część kodu)

```
    # Visualization - Histogram
    plt.figure(figsize=(8, 6))
    plt.hist(values, bins=20, color='skyblue', edgecolor='black')
    plt.title(f'{parameter.capitalize()} Distribution')
    plt.xlabel(parameter.capitalize())
    plt.ylabel('Frequency')
    plt.grid(True)

    # Annotating statistical values
    plt.text( x: 0.05,  y: 0.95,  s: f"Mean: {stats['mean']:.2f}\nMedian: {stats['median']:.2f}\nMin: {stats['min']:.2f}\nMax:
             verticalalignment='top', horizontalalignment='left',
             transform=plt.gca().transAxes, bbox=dict(facecolor='white', alpha=0.5))

    plt.show()

    all_stats[parameter] = stats

return all_stats
```

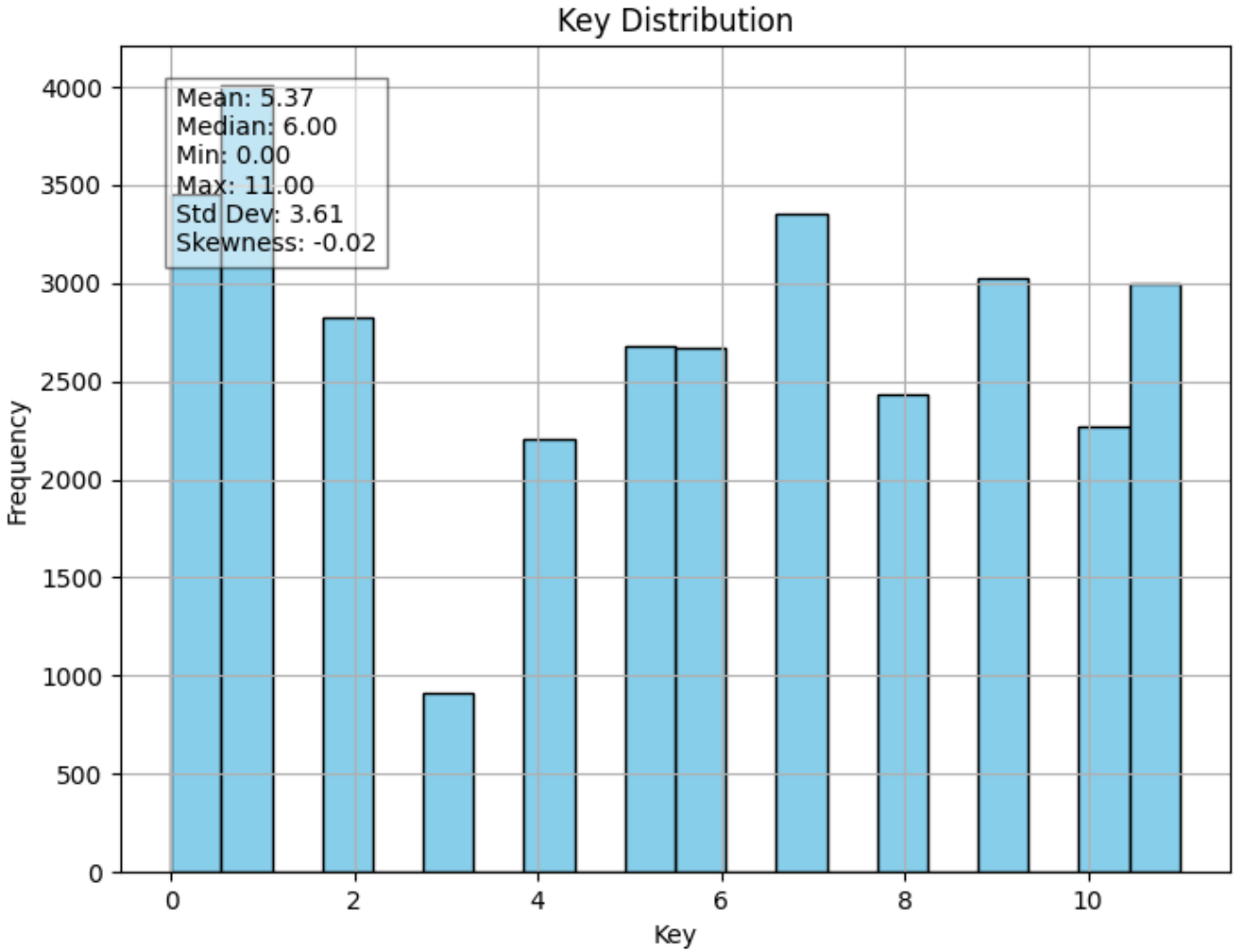
Dancebility



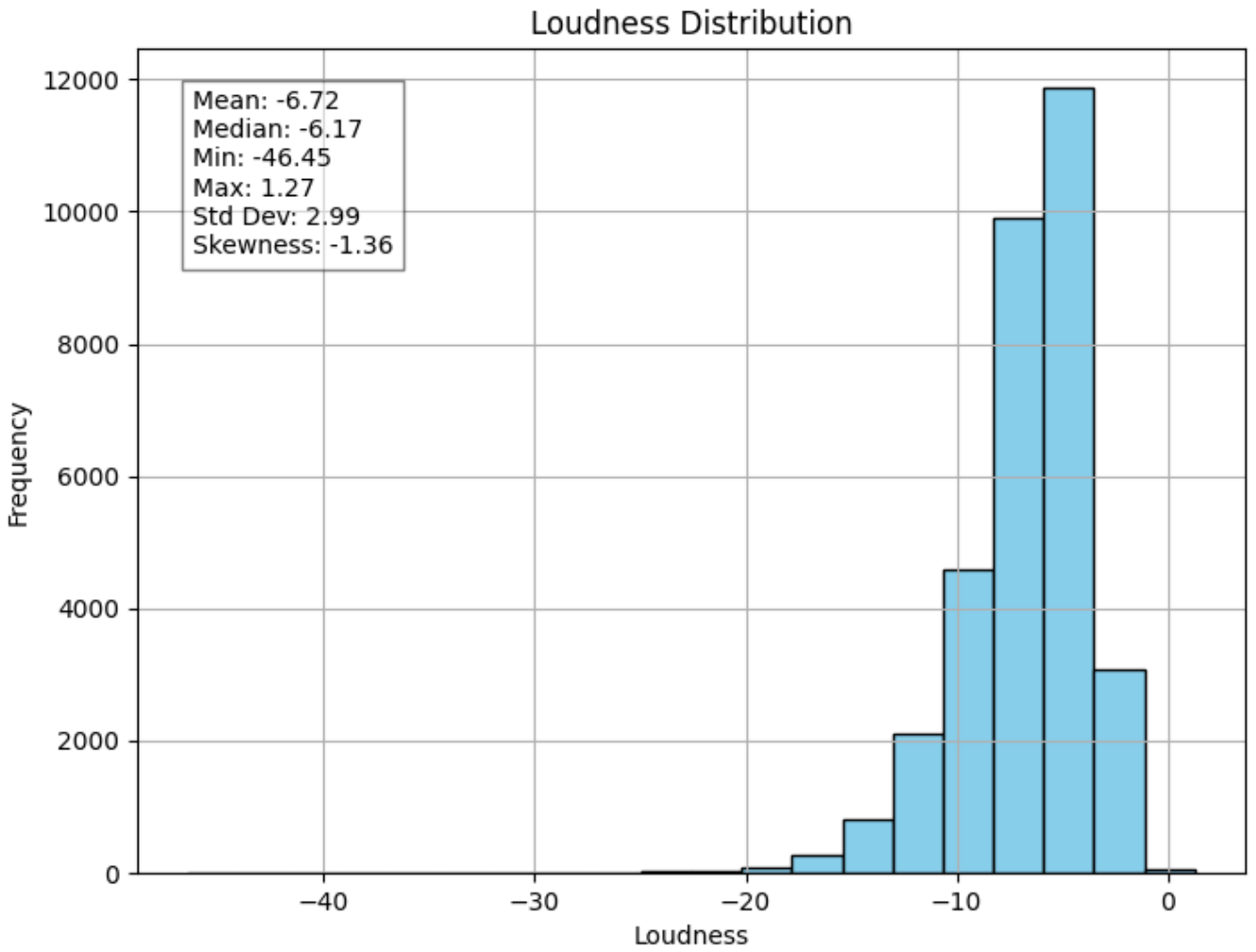
Energy


Energy Distribution
Mean: 0.70
Median: 0.72
Min: 0.00
Max: 1.00
Std Dev: 0.18
Skewness: -0.64
Frequency
Energy
0
500
1000
1500
2000
2500
3000
3500
0.0
0.2
0.4
0.6
0.8
1.0

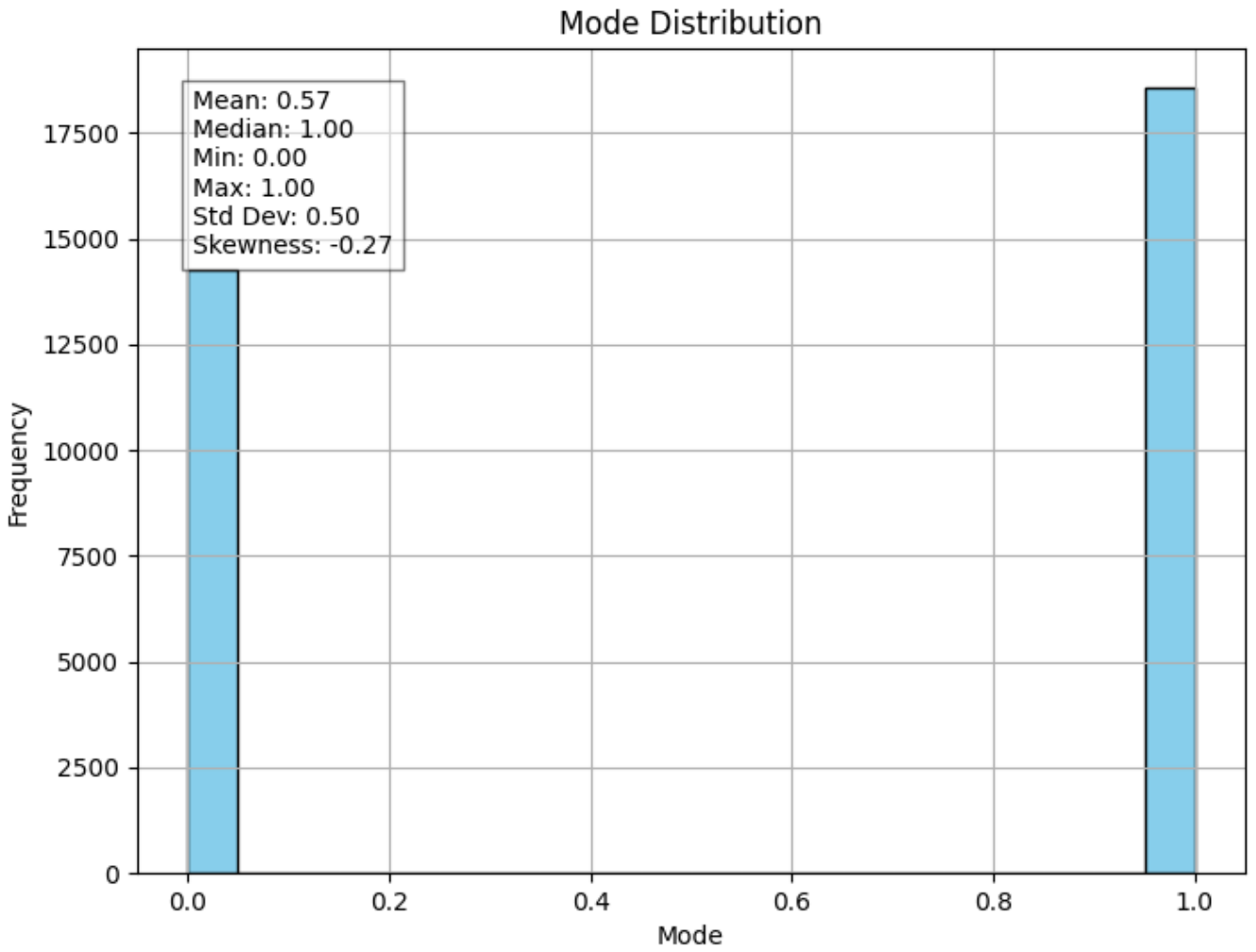
Key


Key Distribution
Mean: 5.37
Median: 6.00
Min: 0.00
Max: 11.00
Std Dev: 3.61
Skewness: -0.02
Frequency
Key
0
500
1000
1500
2000
2500
3000
3500
4000
0
2
4
6
8
10

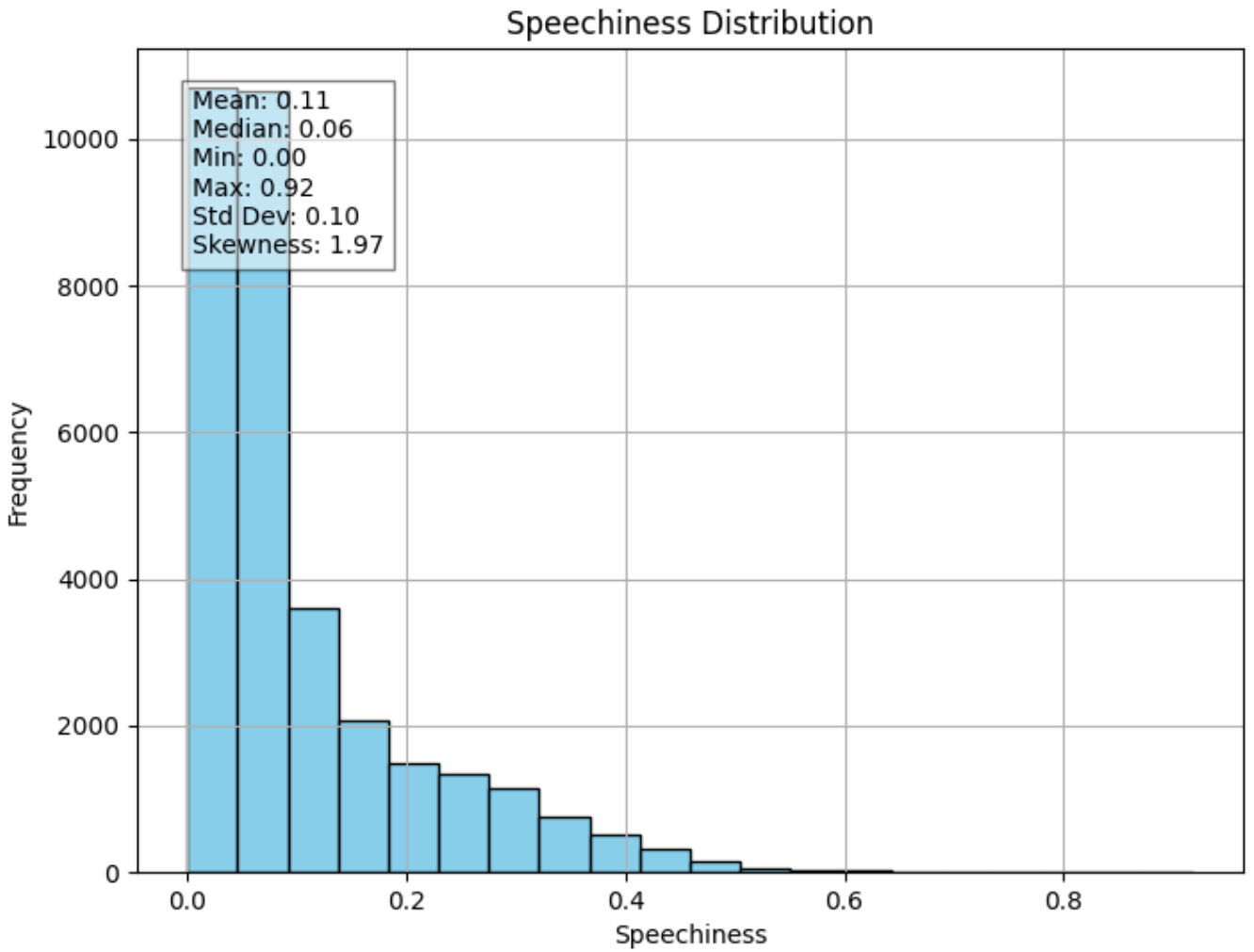
Loudness


Loudness Distribution
Mean: -6.72
Median: -6.17
Min: -46.45
Max: 1.27
Std Dev: 2.99
Skewness: -1.36
Frequency
12000
10000
8000
6000
4000
2000
0
−40
−30
−20
−10
0
Loudness

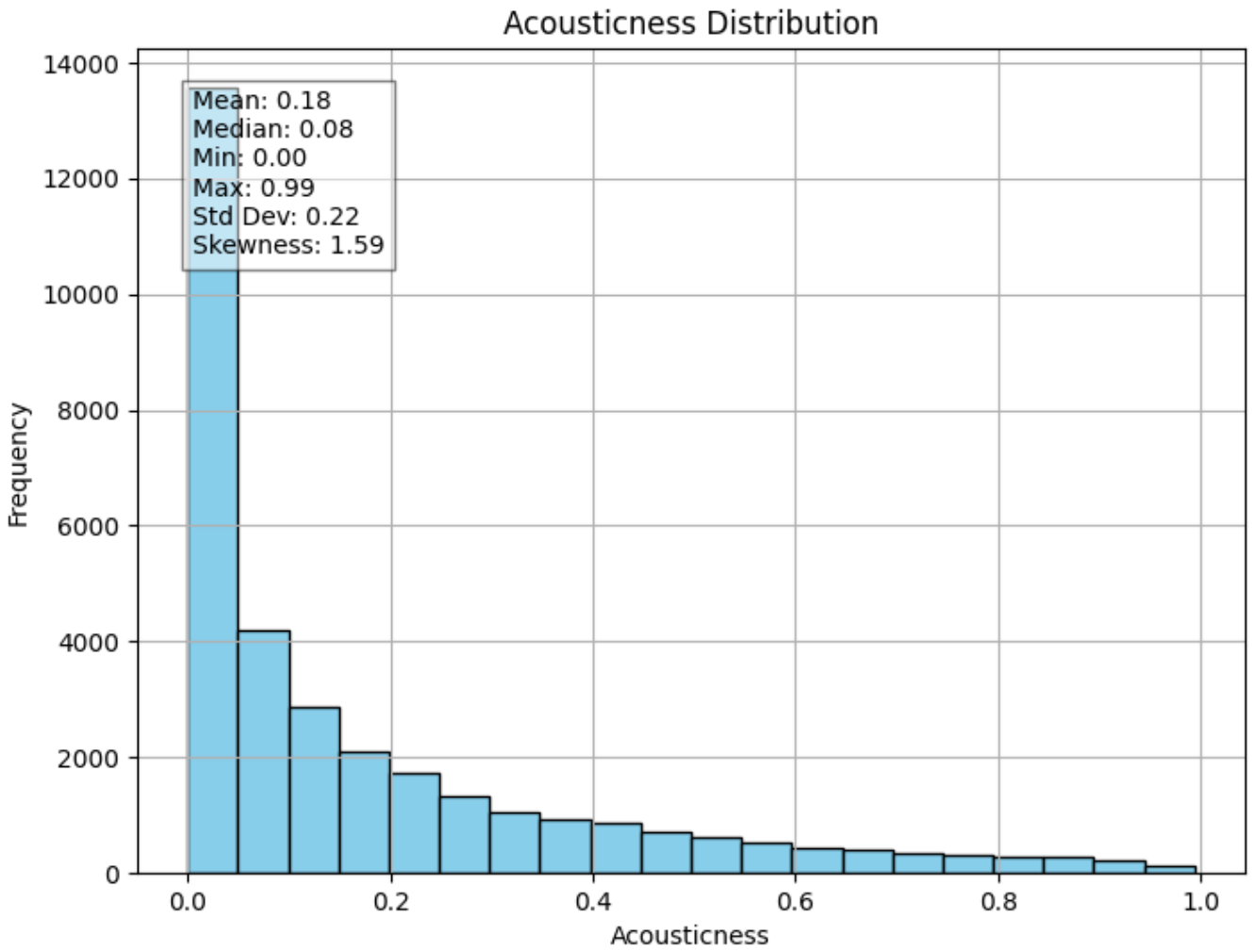
Mode


Mode Distribution
Mean: 0.57
Median: 1.00
Min: 0.00
Max: 1.00
Std Dev: 0.50
Skewness: -0.27
Frequency
17500
15000
12500
10000
7500
5000
2500
0
0.0
0.2
0.4
0.6
0.8
1.0
Mode

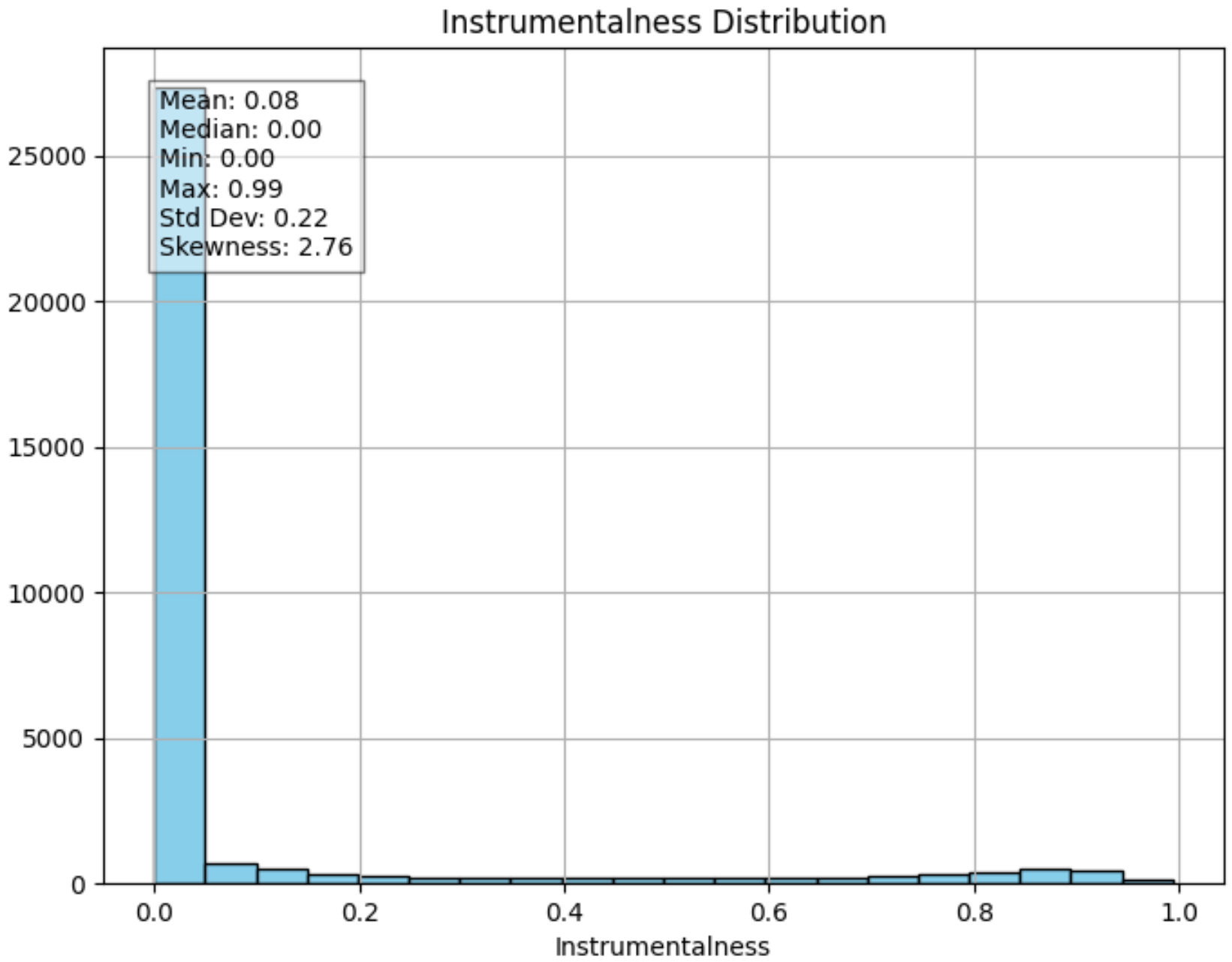
Speechiness



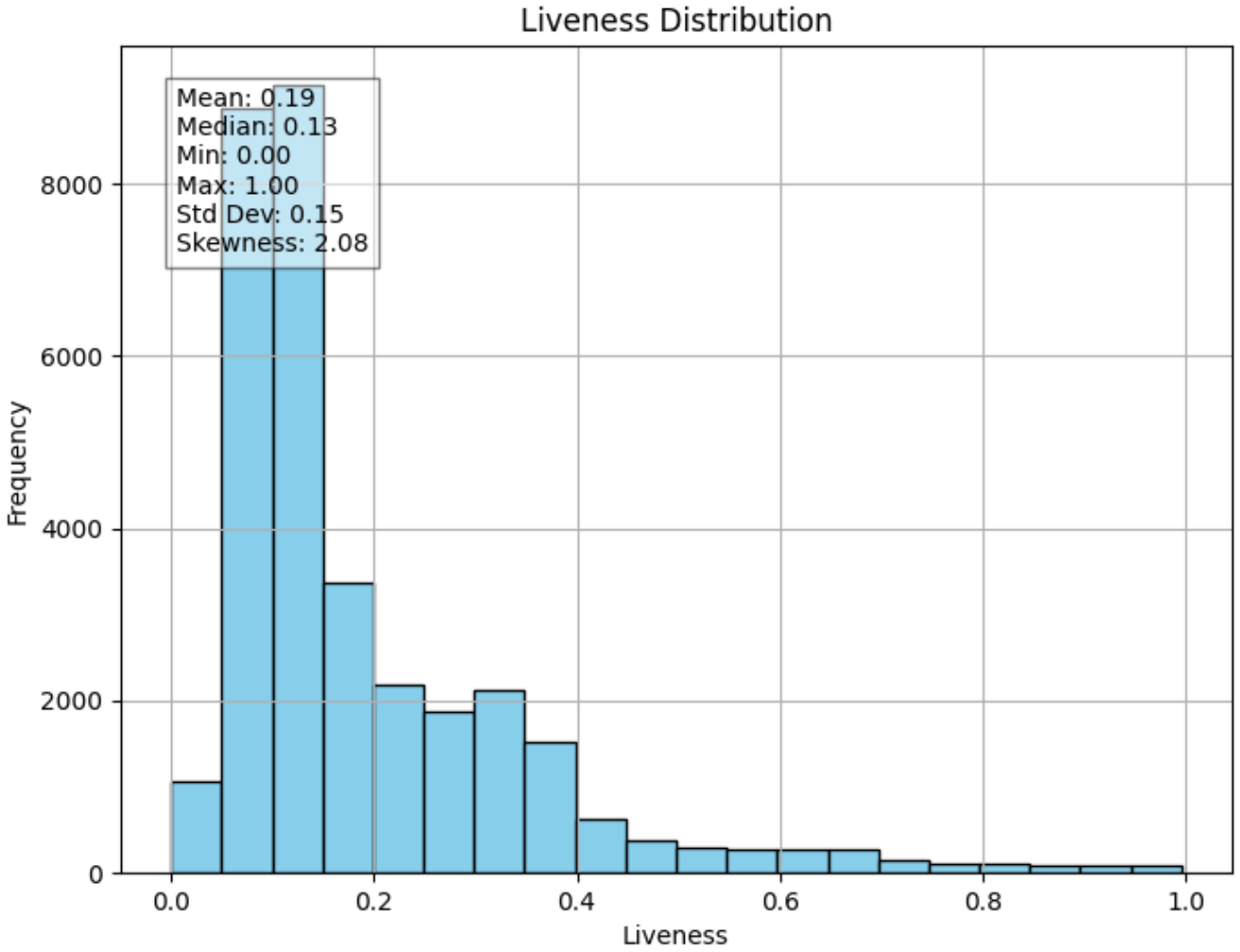
Acousticness



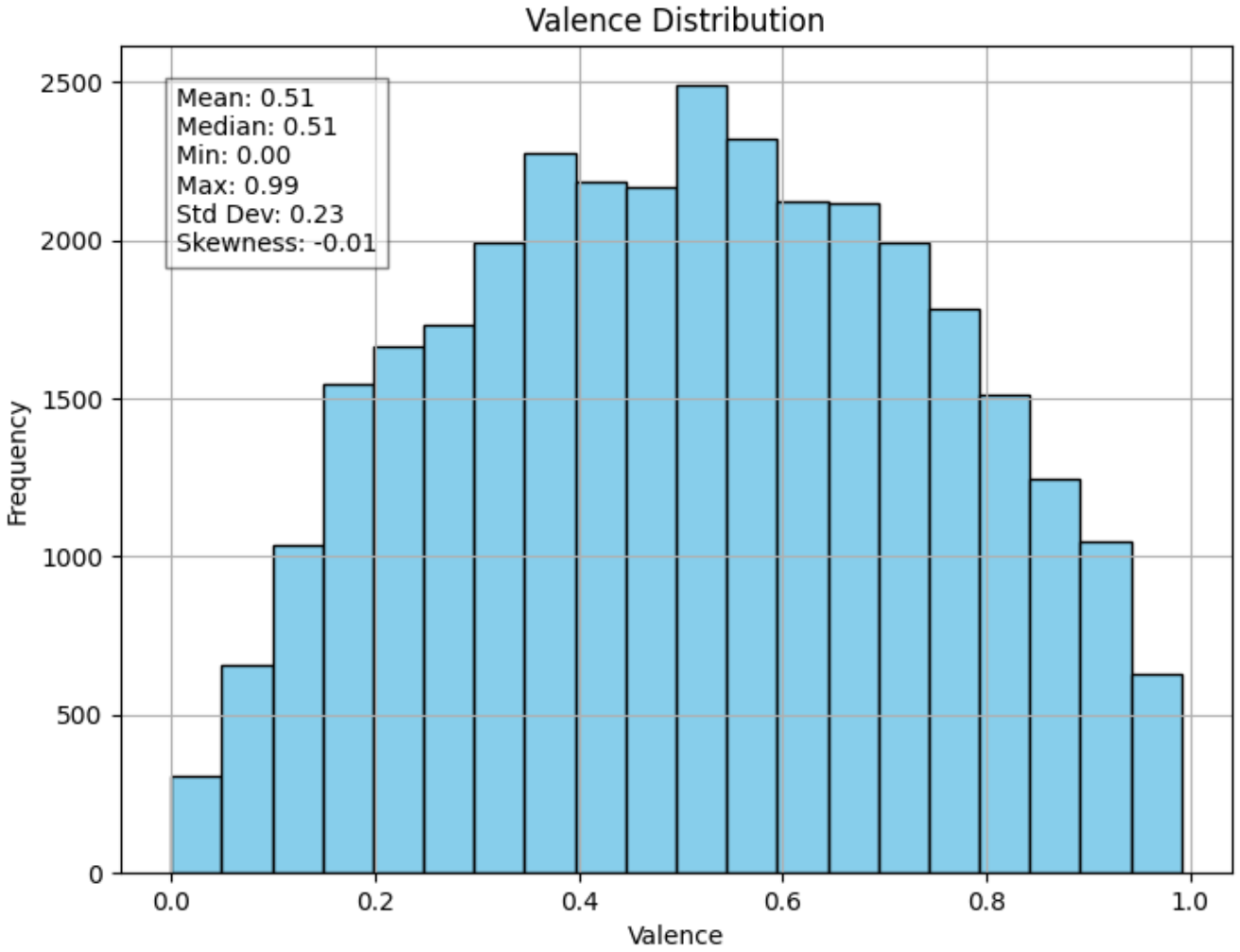
Instrumetalness



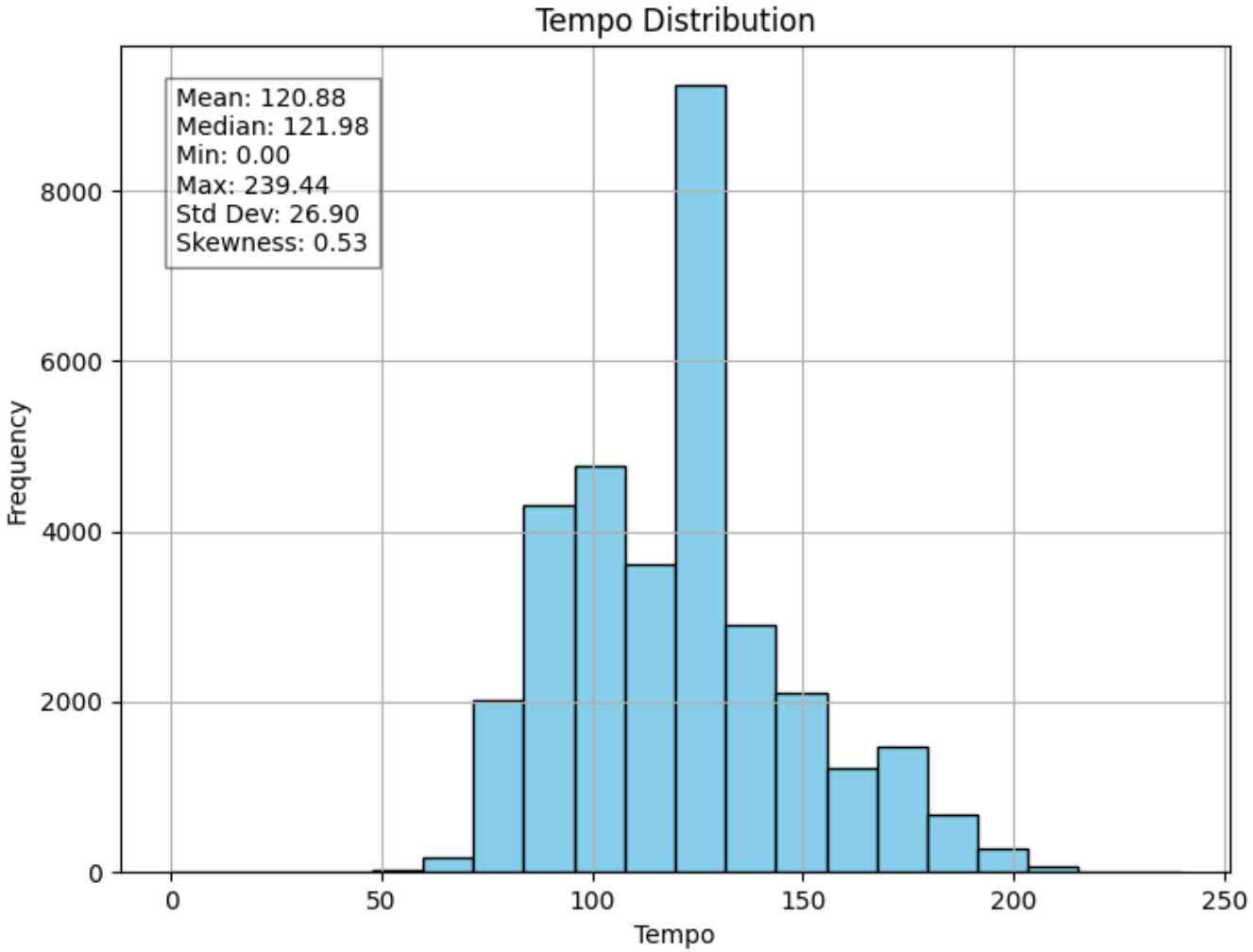
Liveness


Liveness Distribution
Mean: 0.19
Median: 0.13
Min: 0.00
Max: 1.00
Std Dev: 0.15
Skewness: 2.08
Frequency
8000
6000
4000
2000
0
0.0
0.2
0.4
0.6
0.8
1.0
Liveness

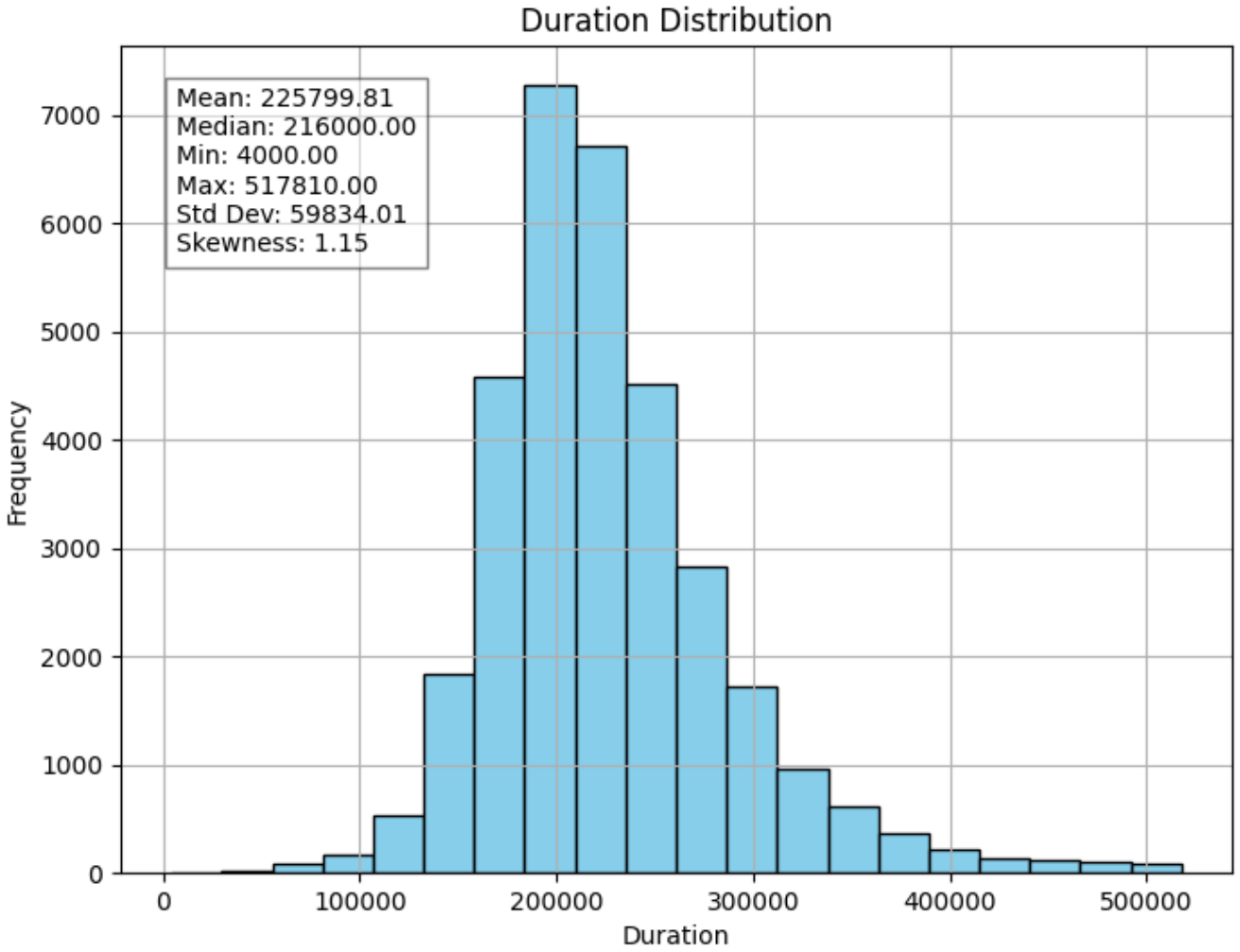
Valence


Valence Distribution
Mean: 0.51
Median: 0.51
Min: 0.00
Max: 0.99
Std Dev: 0.23
Skewness: -0.01
Frequency
Valence
0
500
1000
1500
2000
2500
0.0
0.2
0.4
0.6
0.8
1.0

Tempo


Tempo Distribution
Mean: 120.88
Median: 121.98
Min: 0.00
Max: 239.44
Std Dev: 26.90
Skewness: 0.53
Frequency
0
2000
4000
6000
8000
0
50
100
150
200
250
Tempo

Duration


Duration Distribution
Mean: 225799.81
Median: 216000.00
Min: 4000.00
Max: 517810.00
Std Dev: 59834.01
Skewness: 1.15
Frequency
0
1000
2000
3000
4000
5000
6000
7000
0
100000
200000
300000
400000
500000
Duration

Genre

Używając funkcji get_unique_values, otrzymujemy tablicę unikalnych gatunków i podgatunków z listy utworów.

W zbiorze są zdefiniowane 6 gatunków:

- Pop
- r&b
- Edm
- Rap
- rock
- latin

Subgenres

Oraz 24 podgatunki:

- Urban Contemporary
- Dance Pop
- Hip Pop
- Hard Rock
- Trap
- Post-Teen Pop
- Neo Soul
- Album Rock
- Electropop
- Latin Pop
- Hip Hop
- Southern Hip Hop
- Classic Rock
- Big Room
- Pop EDM
- Reggaeton
- Tropical
- Latin Hip Hop
- Electro House
- Permanent Wave
- New Jack Swing
- Progressive Electro House
- Gangster Rap
- Indie Poptimism

Funkcja get_unique_values

```python
def get_unique_values(tracks, parameter):
    unique_values = set()
    for track in tracks:
        val = getattr(track, parameter, None)
        if val is not None:
            unique_values.add(val)
    return list(unique_values)
```

Transformacje danych

Dane zostały przeskalowane przy użyciu StandardScaler, co pozwoliło na poprawę efektywności algorytmów klasyfikacji.

Standaryzacja elementów przez usunięcie średniej i skalowanie do wariancji jednostkowej.

Standardowy wynik próby oblicza się w następujący sposób:x

z = (x - u) / s

gdzie jest średnią z próbek treningowych lub zerem, jeśli , i jest odchyleniem standardowym próbek treningowych lub jednym z jeżeli.uwith_mean=Falseswith_std=False

Centrowanie i skalowanie odbywa się niezależnie od każdej funkcji przez obliczenia odpowiednie dane statystyczne dotyczące próbek w zbiorze treningowym. Średnia i odchylenia standardowe są następnie zapisywane do wykorzystania w późniejszych danych za pomocą .

Standaryzacja zbioru danych jest powszechnym wymogiem dla wielu estymatory uczenia maszynowego: mogą zachowywać się źle, jeśli Poszczególne funkcje nie wyglądają mniej więcej jak normalnie dane rozproszone (np. gaussowskie ze średnią 0 i wariancją jednostkową).

Braki danych

Zbiór danych nie posiada braków danych. Wszystkie wartości są poprawne.

Obserwacje odstające

Wszystkie dane mieszczą się w granicach operacyjnych, a ich ilość niweluje ewentualne skoki.

Opis metod

Kod wykorzystuje 4 metody do analizy danych (do trenowania modeli używamy bibliotek sklearn w Pythonie):

- RandomForest
- SVM
- KNN
- Model hybrydowy

Wrzucenie danych do datafram'a

```python
def tracks_to_dataframe(tracks):
    data = []
    for track in tracks:
        data.append({
            'name': track.name,
            'artist': track.artist,
            'genre': track.genre,
            'subgenre': track.subgenre,
            'danceability': track.danceability,
            'energy': track.energy,
            'key': track.key,
            'loudness': track.loudness,
            'mode': track.mode,
            'speechiness': track.speechiness,
            'acousticness': track.acousticness,
            'instrumentalness': track.instrumentalness,
            'liveness': track.liveness,
            'valence': track.valence,
            'tempo': track.tempo,
            'duration': track.duration
        })
    return pd.DataFrame(data)
```

RandomForest

Random Forest to metoda zespołowa, która wykorzystuje wiele drzew decyzyjnych do poprawy dokładności i zapobiegania przetrenowaniu.

Główne kroki algorytmu Random Forest:

Bootstrap-wybieranie: Wygeneruj BB nowych zbiorów uczących poprzez losowanie ze zwracaniem z oryginalnego zbioru uczącego. Każdy nowy zbiór będzie miał taki sam rozmiar jak oryginalny.

Budowanie drzew decyzyjnych: Dla każdej bootstrap-wybierki zbuduj drzewo decyzyjne. Przy każdym podziale węzła:

Losowo wybierz podzbiór cech o rozmiarze mm spośród wszystkich MM cech.

Wybierz najlepszą cechę z tego podzbioru do podziału węzła.

Formalnie, dla węzła tt z podzbiorem cech $Mt \subseteq \{1,\ldots,M\}$Mt ⊆{1,…,M}:

θt=argmin$\theta \in Mt \sum j$=12$H(Nj(\theta))$θt =argθ∈Mt min j=1∑2 H(Nj (θ))

gdzie HH to funkcja niepewności (np. entropia lub wskaźnik Giniego), $Nj(\theta)$Nj (θ) to podzbiory danych utworzone po podziale.

Agregacja wyników: Po zbudowaniu wszystkich BB drzew, dla nowej próbki xx:

Uzyskaj przewidywania $y\hat{}b(x)$y^ b (x) od każdego drzewa bb.

Uśrednij przewidywania (dla regresji) lub użyj reguły większości (dla klasyfikacji):

$y\hat{}$=mode({$y\hat{}1(x),y\hat{}2(x),\ldots,y\hat{}B(x)$})y^ =mode({y^ 1 (x),y^ 2 (x),…,y^ B (x)})

RandomForest train

```python
def train_model_rf(X_train, y_train):
    model_path = "random_forest_model.pkl"
    if os.path.exists(model_path):
        pipeline = joblib.load(model_path)
    else:
        pipeline = Pipeline([
            ('scaler', StandardScaler()),
            ('classifier', MultiOutputClassifier(RandomForestClassifier(random_state=42)))
        ])
        pipeline.fit(X_train, y_train)
        joblib.dump(pipeline, model_path)
    return pipeline
```

SVM

Algorytm Support Vector Machine (SVM) jest metodą klasyfikacji, która znajduje optymalną hiperpłaszczyznę do rozdzielenia danych na dwie klasy. Główna idea polega na maksymalizacji marginesu między dwiema klasami, uwzględniając możliwość błędów.
Główne kroki algorytmu SVM:
Optymalizacja hiperpłaszczyzny: Dla zbioru danych (xi,yi)(xi ,yi), gdzie $xi \in Rn$xi ∈Rn to wektor cech, a $yi \in \{-1,1\}$yi ∈{−1,1} to etykieta klasy, SVM szuka hiperpłaszczyzny $wTx+b=0$wTx+b=0, która maksymalizuje margines między klasami. Zadanie optymalizacji formułuje się następująco:
minw,b,ξ12‖w‖2+$C\sum i=1n\xi i$w,b,ξmin 21 ‖w‖2+Ci=1∑n ξi
przy ograniczeniach:
$yi(wTxi+b) \geq 1-\xi i, \xi i \geq 0, i=1,\ldots,n$yi (wTxi +b)≥1−ξi ,ξi ≥0,i=1,…,n
gdzie ‖w‖‖w‖ to norma wektora ww, ξiξi to zmienne elastyczności (slack variables), a CC to parametr regularyzacji kontrolujący równowagę między maksymalizacją marginesu a karą za błędną klasyfikację.
Forma dualna zadania: Przekształcamy zadanie w formę dualną, aby uprościć rozwiązanie i wykorzystać funkcje jądra. Zadanie dualne formułuje się następująco:
max$\alpha \sum i=1n\alpha i-12\sum i,j=1n\alpha i\alpha jyiyjK(xi,xj)$αmax i=1∑n αi −21 i,j=1∑n αi αj yi yj K(xi ,xj)
przy ograniczeniach:
$0 \leq \alpha i \leq C, \sum i=1n\alpha iyi=0$0≤αi ≤C,i=1∑n αi yi =0
gdzie αiαi to zmienne dualne, a $K(xi,xj)$K(xi ,xj) to funkcja jądra, określająca iloczyn skalarny wektorów w przestrzeni cech.
Funkcja jądra: Funkcja jądra $K(xi,xj)$K(xi ,xj) pozwala pracować w przestrzeniach wysokowymiarowych bez jawnego obliczania współrzędnych w tych przestrzeniach. Niektóre popularne jądra:
Jądro liniowe: $K(xi,xj)=xiTxj$K(xi ,xj)=xiT xj
Jądro wielomianowe: $K(xi,xj)=(xiTxj+1)d$K(xi ,xj)=(xiT xj +1)d
Jądro RBF (funkcja radialna): $K(xi,xj)=\exp(-\gamma \|xi-xj\|2)$K(xi ,xj)=exp(−γ‖xi −xj ‖2)
Rozwiązanie zadania optymalizacyjnego: Po rozwiązaniu zadania dualnego za pomocą metod optymalizacji (np. algorytmu SMO), otrzymujemy optymalne wartości αiαi . Wektor wag ww i przesunięcie bb oblicza się następująco:
$w=\sum i=1n\alpha iyixi$w=i=1∑n αi yi xi
$b=yj-\sum i=1n\alpha iyiK(xi,xj)$dla dowolnego $\alpha j>0$b=yj −i=1∑n αi yi K(xi ,xj)dla dowolnego αj >0
Funkcja decyzyjna: Po uzyskaniu optymalnych parametrów, funkcja decyzyjna dla nowej próbki xx formułuje się następująco:
decision_function(x)=$\sum i=1n\alpha iyiK(xi,x)+b$decision_function(x)=i=1∑n αi yi K(xi ,x)+b
Klasa nowej próbki określana jest na podstawie znaku funkcji decyzyjnej:
$y\hat{}$=sign(decision_function(x))y^ =sign(decision_function(x))

SVM train

```python
def train_model_svm(X_train, y_train):
    model_path = "svm_model.pkl"
    if os.path.exists(model_path):
        pipeline = joblib.load(model_path)
    else:
        pipeline = Pipeline([
            ('scaler', StandardScaler()),
            ('classifier', MultiOutputClassifier(SVC(kernel='linear', random_state=42)))
        ])
        pipeline.fit(X_train, y_train)
        joblib.dump(pipeline, model_path)
    return pipeline
```

KNN

K-Nearest Neighbors (KNN)
Algorytm K-Nearest Neighbors (KNN) jest metodą klasyfikacji, która wykorzystuje bliskość między próbkami do podejmowania decyzji.
Główne kroki algorytmu KNN:
Określenie odległości: Dla każdej próbki z zestawu uczącego obliczamy odległość do nowej próbki. Najczęściej używana odległość to odległość euklidesowa:
$d(xi,x)$=∑j=1$m(xij−xj)$2d(xi ,x)=j=1∑m (xij −xj)2
gdzie xixi to i-ta próbka ucząca, xx to nowa próbka, a mm to liczba cech.
Wybór K najbliższych sąsiadów: Znajdź K próbek uczących, które mają najmniejszą odległość do nowej próbki xx.
Klasyfikacja: Sklasyfikuj nową próbkę xx na podstawie etykiet klas yiyi jej K najbliższych sąsiadów. Najczęściej używana jest reguła większości:
y^=mode({yi1,yi2,…,yiK})y^ =mode({yi1 ,yi2 ,…,yiK })
gdzie yi1,yi2,…,yiKyi1 ,yi2 ,…,yiK to etykiety klas K najbliższych sąsiadów.

KNN train

```python
def train_model_knn(X_train, y_train):
    model_path = "knn_model.pkl"
    if os.path.exists(model_path):
        pipeline = joblib.load(model_path)
    else:
        pipeline = Pipeline([
            ('scaler', StandardScaler()),
            ('classifier', MultiOutputClassifier(KNeighborsClassifier()))
        ])
        pipeline.fit(X_train, y_train)
        joblib.dump(pipeline, model_path)
    return pipeline
```

Model hybrydowy

Jest średnią ważoną score’ów z KNN, SVM i RandomForest. Pokazuje najbardziej prawdziwe prognozy ze wszystkich metod.

Model hybrydowy train

```python
def train_model_hybrid(X_train, y_train):
    model_path = "hybrid_model.pkl"
    if os.path.exists(model_path):
        pipeline = joblib.load(model_path)
    else:
        rf_classifier = RandomForestClassifier(random_state=42)
        svm_classifier = SVC(kernel='linear', probability=True, random_state=42)
        knn_classifier = KNeighborsClassifier()

        hybrid_classifier = VotingClassifier(
            estimators=[
                ('rf', rf_classifier),
                ('svm', svm_classifier),
                ('knn', knn_classifier)
            ],
            voting='soft'
        )

        pipeline = Pipeline([
            ('scaler', StandardScaler()),
            ('classifier', MultiOutputClassifier(hybrid_classifier))
        ])
        pipeline.fit(X_train, y_train)
        joblib.dump(pipeline, model_path)
    return pipeline
```

Funkcja oceny modelu

```python
def evaluate_model(model, X_test, y_test):
    y_pred = model.predict(X_test)
    accuracy_genre = accuracy_score(y_test['genre'], y_pred[:, 0])
    accuracy_subgenre = accuracy_score(y_test['subgenre'], y_pred[:, 1])
    report_genre = classification_report(y_test['genre'], y_pred[:, 0])
    report_subgenre = classification_report(y_test['subgenre'], y_pred[:, 1])
    return accuracy_genre, report_genre, accuracy_subgenre, report_subgenre
```

Rezultaty, Merniki

Jako mierniki skuteczności zastosowano Accuracy oraz raporty klasyfikacyjne zawierające precision, recall i F1 Score.

Rezultaty, sposób walidacji

W kodzie użyto metody weryfikacji znanej jako podział danych na zbiory treningowy i testowy (train-test split), która jest jedną z najprostszych i najbardziej powszechnych metod oceny modelu. W tej metodzie dane są dzielone na dwa odrębne zbiory: zbiór treningowy, który jest używany do trenowania modelu, oraz zbiór testowy, który jest używany do oceny wydajności modelu.

Rezultaty, RandomForest

RandomForest Genre Accuracy: 0.5547434140398965
RandomForest Genre Classification Report:

	precision	recall	f1-score	support
edm	0.65	0.69	0.67	1218
latin	0.50	0.41	0.45	1033
pop	0.36	0.33	0.34	1081
r&b	0.48	0.48	0.48	1031
rap	0.59	0.64	0.62	1168
rock	0.68	0.75	0.71	1036
accuracy			0.55	6567
macro avg	0.54	0.55	0.55	6567
weighted avg	0.55	0.55	0.55	6567

RandomForest Subgenre Accuracy: 0.24805847418912746
RandomForest Subgenre Classification Report:

	precision	recall	f1-score	support
album rock	0.19	0.15	0.17	220
big room	0.31	0.32	0.32	248
classic rock	0.24	0.25	0.25	279
dance pop	0.12	0.11	0.11	236
electro house	0.31	0.37	0.34	309
electropop	0.15	0.09	0.11	299
gangster rap	0.30	0.33	0.31	296
hard rock	0.36	0.54	0.44	313
hip hop	0.39	0.40	0.39	261
hip pop	0.03	0.02	0.03	228
indie poptimism	0.17	0.18	0.17	330
latin hip hop	0.15	0.14	0.14	315
latin pop	0.15	0.11	0.13	281
neo soul	0.29	0.34	0.31	323
new jack swing	0.45	0.63	0.52	223
permanent wave	0.15	0.12	0.13	224
pop edm	0.12	0.10	0.11	325
post-teen pop	0.11	0.09	0.10	216
progressive electro house	0.29	0.38	0.33	336
reggaeton	0.29	0.32	0.30	187
southern hip hop	0.31	0.33	0.32	347
trap	0.27	0.24	0.26	264
tropical	0.22	0.23	0.22	250
urban contemporary	0.11	0.08	0.09	257
accuracy			0.25	6567
macro avg	0.23	0.24	0.23	6567
weighted avg	0.23	0.25	0.24	6567

Rezultaty, SVM

SVM Genre Accuracy: 0.46977310796406274
SVM Genre Classification Report:

	precision	recall	f1-score	support
edm	0.53	0.59	0.55	1218
latin	0.38	0.37	0.38	1033
pop	0.35	0.27	0.30	1081
r&b	0.42	0.36	0.39	1031
rap	0.51	0.54	0.52	1168
rock	0.56	0.67	0.61	1036
accuracy			0.47	6567
macro avg	0.46	0.47	0.46	6567
weighted avg	0.46	0.47	0.46	6567

SVM Subgenre Accuracy: 0.2198873153647023
SVM Subgenre Classification Report:

	precision	recall	f1-score	support
album rock	0.26	0.13	0.17	220
big room	0.31	0.28	0.29	248
classic rock	0.24	0.34	0.28	279
dance pop	0.05	0.03	0.04	236
electro house	0.32	0.28	0.30	309
electropop	0.20	0.03	0.05	299
gangster rap	0.22	0.33	0.26	296
hard rock	0.31	0.60	0.41	313
hip hop	0.43	0.29	0.35	261
hip pop	0.14	0.03	0.04	228
indie poptimism	0.14	0.18	0.16	330
latin hip hop	0.11	0.25	0.15	315
latin pop	0.13	0.07	0.09	281
neo soul	0.27	0.36	0.31	323
new jack swing	0.28	0.41	0.33	223
permanent wave	0.12	0.04	0.06	224
pop edm	0.16	0.15	0.16	325
post-teen pop	0.12	0.06	0.08	216
progressive electro house	0.26	0.42	0.32	336
reggaeton	0.00	0.00	0.00	187
southern hip hop	0.27	0.31	0.29	347
trap	0.19	0.21	0.20	264
tropical	0.14	0.11	0.12	250
urban contemporary	0.13	0.04	0.06	257
accuracy			0.22	6567
macro avg	0.20	0.21	0.19	6567
weighted avg	0.20	0.22	0.20	6567

Rezultaty, KNN

KNN Genre Accuracy: 0.4592660271052231

KNN Genre Classification Report:

	precision	recall	f1-score	support
edm	0.52	0.67	0.58	1218
latin	0.37	0.43	0.40	1033
pop	0.30	0.29	0.30	1081
r&b	0.38	0.34	0.36	1031
rap	0.58	0.45	0.51	1168
rock	0.63	0.54	0.58	1036
accuracy			0.46	6567
macro avg	0.46	0.45	0.45	6567
weighted avg	0.46	0.46	0.46	6567

KNN Subgenre Accuracy: 0.18486371250190345

KNN Subgenre Classification Report:

	precision	recall	f1-score	support
album rock	0.12	0.23	0.16	220
big room	0.21	0.44	0.28	248
classic rock	0.18	0.27	0.22	279
dance pop	0.07	0.15	0.09	236
electro house	0.19	0.25	0.22	309
electropop	0.10	0.13	0.11	299
gangster rap	0.22	0.33	0.27	296
hard rock	0.35	0.38	0.36	313
hip hop	0.33	0.34	0.33	261
hip pop	0.07	0.07	0.07	228
indie poptimism	0.13	0.11	0.12	330
latin hip hop	0.11	0.09	0.10	315
latin pop	0.17	0.10	0.13	281
neo soul	0.21	0.15	0.18	323
new jack swing	0.38	0.37	0.37	223
permanent wave	0.13	0.07	0.09	224
pop edm	0.09	0.03	0.05	325
post-teen pop	0.10	0.05	0.07	216
progressive electro house	0.28	0.26	0.27	336
reggaeton	0.26	0.20	0.23	187
southern hip hop	0.28	0.16	0.20	347
trap	0.24	0.11	0.15	264
tropical	0.12	0.08	0.09	250
urban contemporary	0.11	0.05	0.07	257
accuracy			0.18	6567
macro avg	0.19	0.18	0.18	6567
weighted avg	0.19	0.18	0.18	6567

Rezultaty, Hybrid

Hybrid Genre Accuracy: 0.5309882747068677

Hybrid Genre Classification Report:

	precision	recall	f1-score	support
edm	0.61	0.69	0.64	1218
latin	0.46	0.40	0.43	1033
pop	0.36	0.31	0.33	1081
r&b	0.44	0.42	0.43	1031
rap	0.59	0.60	0.60	1168
rock	0.65	0.73	0.69	1036
accuracy			0.53	6567
macro avg	0.52	0.53	0.52	6567
weighted avg	0.52	0.53	0.52	6567

Hybrid Subgenre Accuracy: 0.22460788792447084

Hybrid Subgenre Classification Report:

	precision	recall	f1-score	support
album rock	0.16	0.11	0.13	220
big room	0.31	0.33	0.32	248
classic rock	0.21	0.23	0.22	279
dance pop	0.07	0.07	0.07	236
electro house	0.26	0.28	0.27	309
electropop	0.10	0.07	0.08	299
gangster rap	0.29	0.32	0.31	296
hard rock	0.35	0.54	0.42	313
hip hop	0.41	0.39	0.40	261
hip pop	0.04	0.04	0.04	228
indie poptimism	0.17	0.17	0.17	330
latin hip hop	0.10	0.10	0.10	315
latin pop	0.15	0.11	0.12	281
neo soul	0.26	0.29	0.28	323
new jack swing	0.41	0.58	0.48	223
permanent wave	0.16	0.12	0.14	224
pop edm	0.09	0.07	0.08	325
post-teen pop	0.09	0.07	0.08	216
progressive electro house	0.27	0.38	0.31	336
reggaeton	0.25	0.30	0.27	187
southern hip hop	0.32	0.32	0.32	347
trap	0.22	0.18	0.20	264
tropical	0.17	0.17	0.17	250
urban contemporary	0.08	0.06	0.07	257
accuracy			0.22	6567
macro avg	0.21	0.22	0.21	6567
weighted avg	0.21	0.22	0.21	6567

Bibliografia

Dokumentacja używanych narzędzi i bibliotek:

- RandomForestClassifier — scikit-learn 1.5.0 documentation
- Support Vector Machines — scikit-learn 1.5.0 documentation
- KNeighborsClassifier — scikit-learn 1.5.0 documentation
- pandas documentation — pandas 2.2.2 documentation (pydata.org)
- os — Miscellaneous operating system interfaces — Python 3.12.3 documentation
- Joblib: running Python functions as pipeline jobs — joblib 1.4.2 documentation
- Matplotlib — Visualization with Python

Pzyklad użycia

- Wchodzimy na stronę musicstax.com i znajdujemy interesujący nas utwór
- Wprowadzamy dane utworu w poniższym kodzie
- Uruchamiamy (pierwsze uruchomienie będzie długie, aby uzyskać szybsze wyniki, można skomentować trening modeli RandomForest i Hybrid)

```
# Example of predicting the genre and subgenre for a new track
new_track = {
    'danceability': 0.57,
    'energy': 0.8,
    'key': 4,
    'loudness': -4.55,
    'mode': 0,
    'speechiness': 0.06,
    'acousticness': 0.01,
    'instrumentalness': 0.82,
    'liveness': 0.3,
    'valence': 0.28,
    'tempo': 179.0,
    'duration': 230000
}
```

Przykładowe wyniki

```
RandomForest Predicted Genre: edm
RandomForest Predicted Subgenre: trap
SVM Predicted Genre: edm
SVM Predicted Subgenre: big room
KNN Predicted Genre: edm
KNN Predicted Subgenre: big room
Hybrid Predicted Genre: edm
Hybrid Predicted Subgenre: trap
```

Test 10 losowych piosenek z bazy

RandomForest Predicted Genre for 'I'll Be Around - Remastered Version' by The Spinners: latin
RandomForest Predicted Subgenre for 'I'll Be Around - Remastered Version' by The Spinners: latin pop
SVM Predicted Genre for 'I'll Be Around - Remastered Version' by The Spinners: latin
SVM Predicted Subgenre for 'I'll Be Around - Remastered Version' by The Spinners: latin hip hop
KNN Predicted Genre for 'I'll Be Around - Remastered Version' by The Spinners: latin
KNN Predicted Subgenre for 'I'll Be Around - Remastered Version' by The Spinners: tropical
Hybrid Predicted Genre for 'I'll Be Around - Remastered Version' by The Spinners: latin
Hybrid Predicted Subgenre for 'I'll Be Around - Remastered Version' by The Spinners: tropical

RandomForest Predicted Genre for 'MГesica' by DJ Luian: latin
RandomForest Predicted Subgenre for 'MГesica' by DJ Luian: latin hip hop
SVM Predicted Genre for 'MГesica' by DJ Luian: latin
SVM Predicted Subgenre for 'MГesica' by DJ Luian: latin hip hop
KNN Predicted Genre for 'MГesica' by DJ Luian: latin
KNN Predicted Subgenre for 'MГesica' by DJ Luian: reggaeton
Hybrid Predicted Genre for 'MГesica' by DJ Luian: latin
Hybrid Predicted Subgenre for 'MГesica' by DJ Luian: latin hip hop

RandomForest Predicted Genre for 'Trippie Redd' by CoqeГ©in Montana: rap
RandomForest Predicted Subgenre for 'Trippie Redd' by CoqeГ©in Montana: southern hip hop
SVM Predicted Genre for 'Trippie Redd' by CoqeГ©in Montana: latin
SVM Predicted Subgenre for 'Trippie Redd' by CoqeГ©in Montana: latin hip hop
KNN Predicted Genre for 'Trippie Redd' by CoqeГ©in Montana: latin
KNN Predicted Subgenre for 'Trippie Redd' by CoqeГ©in Montana: latin hip hop
Hybrid Predicted Genre for 'Trippie Redd' by CoqeГ©in Montana: rap
Hybrid Predicted Subgenre for 'Trippie Redd' by CoqeГ©in Montana: southern hip hop

RandomForest Predicted Genre for 'Close Enough to Hurt' by Rod Wave: rap
RandomForest Predicted Subgenre for 'Close Enough to Hurt' by Rod Wave: gangster rap
SVM Predicted Genre for 'Close Enough to Hurt' by Rod Wave: rap
SVM Predicted Subgenre for 'Close Enough to Hurt' by Rod Wave: trap
KNN Predicted Genre for 'Close Enough to Hurt' by Rod Wave: rap
KNN Predicted Subgenre for 'Close Enough to Hurt' by Rod Wave: trap
Hybrid Predicted Genre for 'Close Enough to Hurt' by Rod Wave: rap
Hybrid Predicted Subgenre for 'Close Enough to Hurt' by Rod Wave: trap

RandomForest Predicted Genre for 'I Miss You' by Jeriqo: rock
RandomForest Predicted Subgenre for 'I Miss You' by Jeriqo: permanent wave
SVM Predicted Genre for 'I Miss You' by Jeriqo: rock
SVM Predicted Subgenre for 'I Miss You' by Jeriqo: hard rock
KNN Predicted Genre for 'I Miss You' by Jeriqo: pop
KNN Predicted Subgenre for 'I Miss You' by Jeriqo: album rock
Hybrid Predicted Genre for 'I Miss You' by Jeriqo: rock
Hybrid Predicted Subgenre for 'I Miss You' by Jeriqo: hard rock

RandomForest Predicted Genre for 'Who Are You' by The Who: rock
RandomForest Predicted Subgenre for 'Who Are You' by The Who: hard rock
SVM Predicted Genre for 'Who Are You' by The Who: rock
SVM Predicted Subgenre for 'Who Are You' by The Who: new jack swing
KNN Predicted Genre for 'Who Are You' by The Who: r&b
KNN Predicted Subgenre for 'Who Are You' by The Who: new jack swing
Hybrid Predicted Genre for 'Who Are You' by The Who: rock
Hybrid Predicted Subgenre for 'Who Are You' by The Who: new jack swing

RandomForest Predicted Genre for 'Happy' by The Beef Seeds: latin
RandomForest Predicted Subgenre for 'Happy' by The Beef Seeds: latin hip hop
SVM Predicted Genre for 'Happy' by The Beef Seeds: latin
SVM Predicted Subgenre for 'Happy' by The Beef Seeds: latin pop
KNN Predicted Genre for 'Happy' by The Beef Seeds: latin
KNN Predicted Subgenre for 'Happy' by The Beef Seeds: tropical
Hybrid Predicted Genre for 'Happy' by The Beef Seeds: latin
Hybrid Predicted Subgenre for 'Happy' by The Beef Seeds: tropical

RandomForest Predicted Genre for 'ONE' by Rev Theory: rock
RandomForest Predicted Subgenre for 'ONE' by Rev Theory: hard rock
SVM Predicted Genre for 'ONE' by Rev Theory: rock
SVM Predicted Subgenre for 'ONE' by Rev Theory: hard rock
KNN Predicted Genre for 'ONE' by Rev Theory: rock
KNN Predicted Subgenre for 'ONE' by Rev Theory: hard rock
Hybrid Predicted Genre for 'ONE' by Rev Theory: rock
Hybrid Predicted Subgenre for 'ONE' by Rev Theory: hard rock

RandomForest Predicted Genre for 'Palace/Curse' by The Internet: r&b
RandomForest Predicted Subgenre for 'Palace/Curse' by The Internet: southern hip hop
SVM Predicted Genre for 'Palace/Curse' by The Internet: r&b
SVM Predicted Subgenre for 'Palace/Curse' by The Internet: southern hip hop
KNN Predicted Genre for 'Palace/Curse' by The Internet: rap
KNN Predicted Subgenre for 'Palace/Curse' by The Internet: southern hip hop
Hybrid Predicted Genre for 'Palace/Curse' by The Internet: rap
Hybrid Predicted Subgenre for 'Palace/Curse' by The Internet: southern hip hop

RandomForest Predicted Genre for 'HГ¤nde hoch' by Baba Saad: rap
RandomForest Predicted Subgenre for 'HГ¤nde hoch' by Baba Saad: gangster rap
SVM Predicted Genre for 'HГ¤nde hoch' by Baba Saad: rap
SVM Predicted Subgenre for 'HГ¤nde hoch' by Baba Saad: gangster rap
KNN Predicted Genre for 'HГ¤nde hoch' by Baba Saad: rap
KNN Predicted Subgenre for 'HГ¤nde hoch' by Baba Saad: gangster rap
Hybrid Predicted Genre for 'HГ¤nde hoch' by Baba Saad: rap
Hybrid Predicted Subgenre for 'HГ¤nde hoch' by Baba Saad: gangster rap

Wyniki

Program dobrze wykonał swoje zadanie, poprawnie przewidując 9 na 10 utworów (w losowym zestawie zmiennych, które napotkaliśmy).

Warto pamiętać, że definiowanie gatunków i podgatunków utworów jest zabiegiem bardzo delikatnym i kontrowersyjnym, w tym sensie, że wiele gatunków ma elementy wspólne, przez co nawet pozornie różne utwory mogą być ze sobą powiązane. O czym tu mówić, skoro nawet dziś istnieje wiele zespołów i gatunków, toczą się debaty na temat ich przynależności i odrębności.