Projekt Indywidualny – Raport

Rozpoznawanie gatunków muzycznych za pomocą uczenia maszynowego

Bartosz Dybowski
325461

Spis treści

1. Cel ćwiczenia

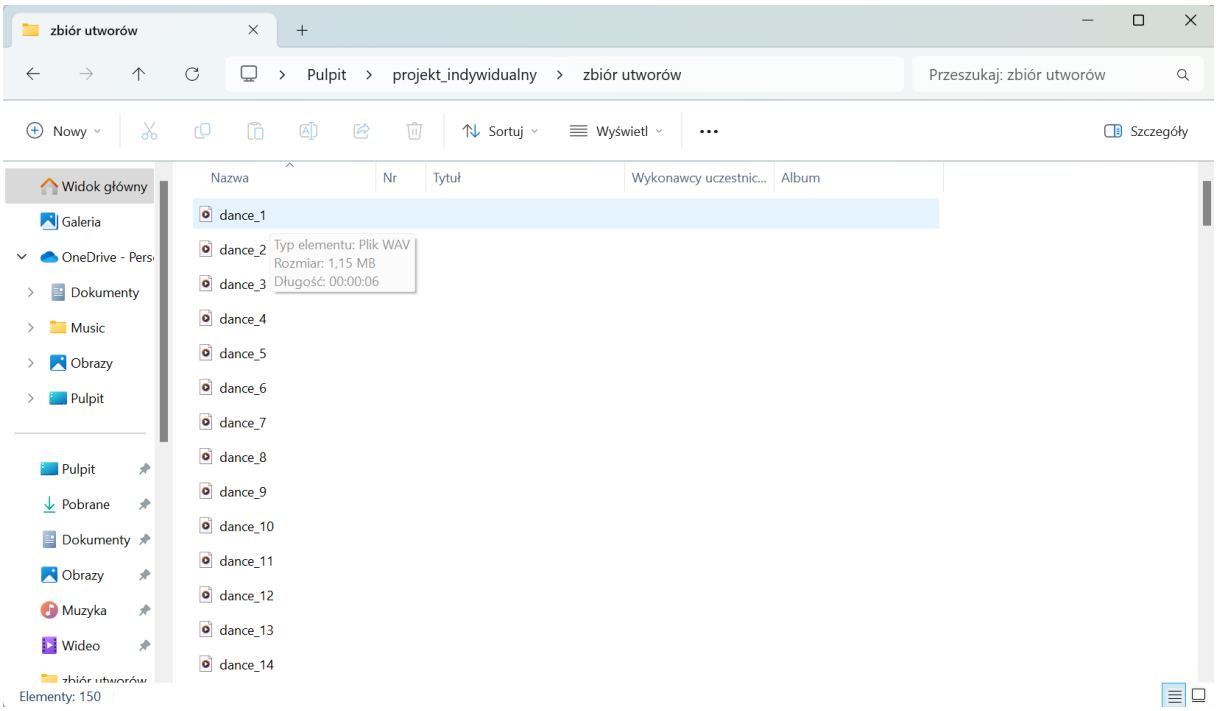
Celem tego projektu było stworzenie modelu uczenia maszynowego zdolnego do rozpoznawania gatunków muzycznych na podstawie wyodrębnionych cech z utworów. Projekt miał na celu zastosowanie technik przetwarzania sygnałów dźwiękowych oraz różnych klasyfikatorów, aby zbudować dokładny model przewidujący gatunek muzyczny. Kolejnym celem było opracowanie praktycznej aplikacji webowej, która wykorzysta stworzony model do rozpoznawania gatunków na podstawie wgrywanych przez użytkowników utworów.

2. Podejście do projektu

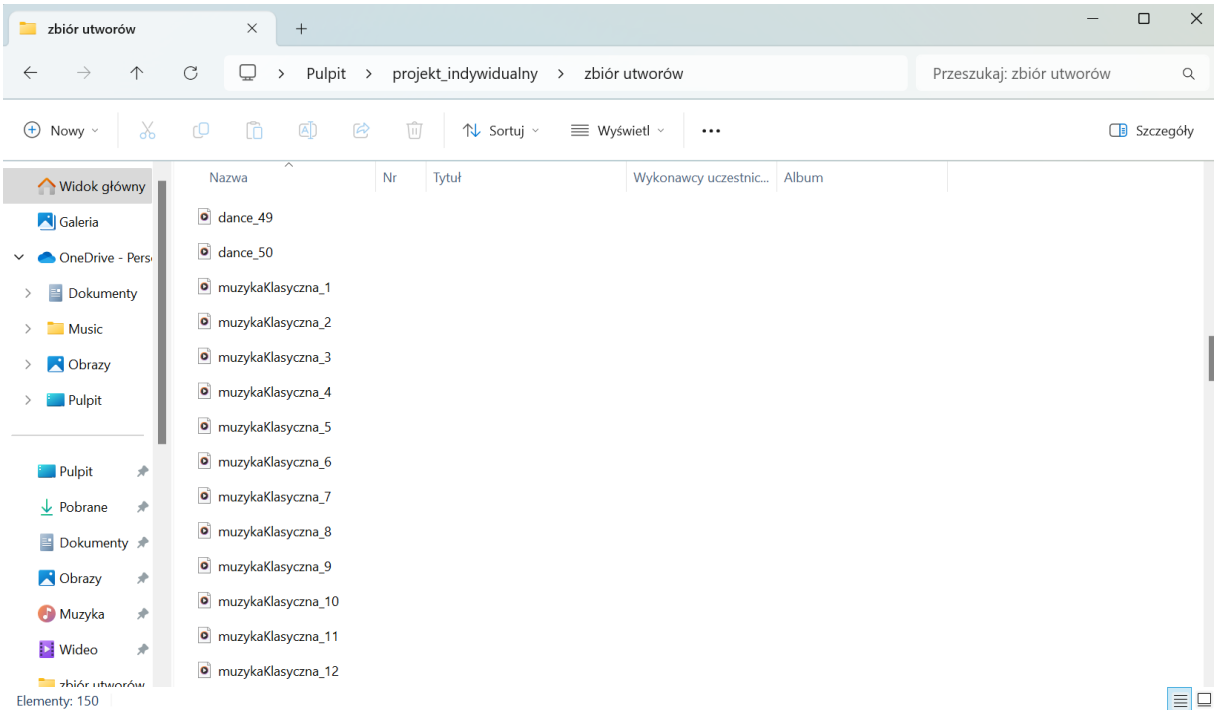
Projekt rozpoczął się od samodzielnego zebrania utworów muzycznych, tworząc tym samym własny zbiór. Następnie wyodrębniono z nich cechy. Kolejnym etapem było przeanalizowanie tych cech, aby zrozumieć zależności między nimi i ich wpływ na klasyfikację gatunków muzycznych. Po analizie usunięto mniej istotne cechy. Proces klasyfikacji przeprowadzono za pomocą trzech różnych algorytmów, a ich wyniki porównano przed i po redukcji cech. Na podstawie wyników wybrano najlepszy model, który następnie zapisano. Ostatnim krokiem było stworzenie aplikacji webowej, która w praktyczny sposób wykorzystywała wcześniej wspomniany model.
Odczytanie zbioru utworów muzycznych, wyodrębnienie z nich cech, przeprowadzenie analizy, proces klasyfikacji oraz stworzenie modelu zostało wykonane w środowisku Jupyter Notebook w języku Python.
Aplikacja webowa została stworzona w środowisku IDE PyCharm, wykorzystując język Python oraz HTML, CSS i JavaScript do stworzenia interfejsu użytkownika.

3. Dane

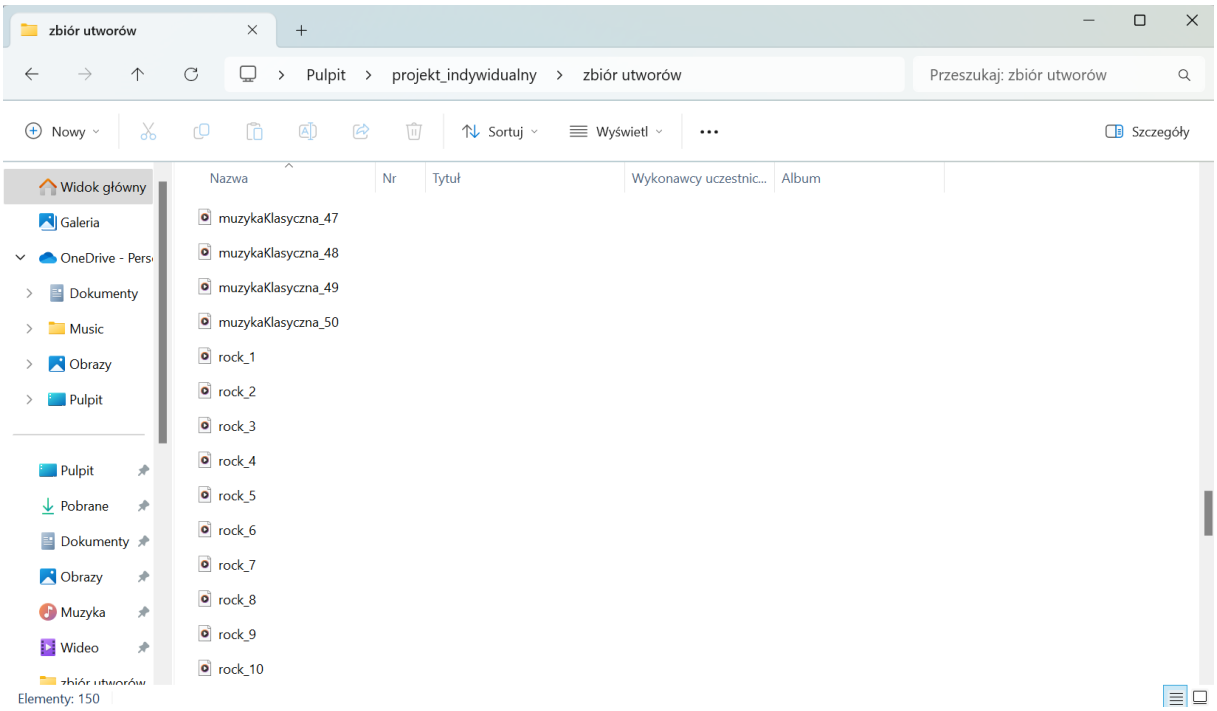
Dane do projektu zostały zebrane samodzielnie, tworząc zbiór utworów, podzielonych na trzy gatunki muzyczne: **Dance**, **Muzyka Klasyczna** i **Rock**. Wstępnie było ich 90, po 30 na każdy gatunek. Następnie dołożyłem kolejne 20 utworów do każdego z gatunków, tworząc tym samym zbiór 150 utworów muzycznych. Była to praca stosunkowo prosta, lecz bardzo czasochłonna, gdyż każdy z utworów pobierałem z serwisu Youtube, a potem konwertowałem do formatu wav. Następnie skracałem je ręcznie do 6 sekund i zapisywałem jako pliki w formacie *nazwaGatunku_numer.wav* do folderu „zbiór utworów”.



Rysunek 1. Folder „zbiór utworów”



Rysunek 2. Folder „zbiór utworów”



Rysunek 3. Folder „zbiór utworów”

Każdy utwór został przetworzony przy użyciu biblioteki *librosa* w celu wyodrębnienia następujących cech:

- Tempo
- Tonacja
- Liczba przejść przez zero
- Środek ciężkości widma
- Szerokość pasm spektralnych
- Kontrast spektralny
- Rolloff spektralny
- Płaskość spektralna
- Skuteczna wartość średnia (RMS)
- Stosunek harmoniczności do szumu
- Chromagram
- Współczynniki cepstralne (MFCCs)
- Częstotliwość fundamentalna (Pitch)
- Rytmika

Na tej podstawie stworzyłem ramkę danych Pandas, w której znajdowały się wszystkie wspomniane wcześniej cechy oraz gatunek każdego z utworów.

In [25]:
```
# Ścieżka do folderu z plikami WAV
folder_path = "zbiór utworów"

# Listy przechowujące cechy i przypisane gatunki
data = []
genres = []

# Iteracja po plikach w folderze
for file_name in os.listdir(folder_path):
    if file_name.endswith('.wav'):
        file_path = os.path.join(folder_path, file_name)
        y, sr = librosa.load(file_path)
        onset_env = librosa.onset.onset_strength(y=y, sr=sr)

        # Wyodrębnienie cech
        tempo = librosa.feature.tempo(onset_envelope=onset_env, sr=sr)  # Tempo
        key, _ = librosa.beat.beat_track(y=y, sr=sr)  # Tonacja
        zero_crossing_rate = librosa.feature.zero_crossing_rate(y).mean()  # Liczba przejść przez zero
        spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr).mean()  # Środek ciężkości widma
        spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr).mean()  # Szerokość pasm spektralnych
        spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr).mean()  # Kontrast spektralny
        spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr).mean()  # Rolloff spektralny
        spectral_flatness = librosa.feature.spectral_flatness(y=y).mean()  # Płaskość spektralna
        rms = librosa.feature.rms(y=y).mean()  # Skuteczna wartość średnia (RMS - Root Mean Square)
        harmonic_to_noise = librosa.effects.harmonic(y).mean()  # Stosunek harmoniczności do szumu(Harmonic-to-Noise Ratio)
        chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr).mean()  # Chromagram
        chroma_cens = librosa.feature.chroma_cens(y=y, sr=sr).mean()
        mfcc = librosa.feature.mfcc(y=y, sr=sr).mean()  # Współczynniki cepstralne (MFCCs)
        pitches, magnitudes = librosa.piptrack(y=y, sr=sr)  # Częstotliwość fundamentalna (Pitch)
        rhythmics = librosa.feature.tempogram(y=y, sr=sr).mean()  # Rytmika

        # Dodanie cech do listy danych
        data.append([tempo, key, zero_crossing_rate, spectral_centroid, spectral_bandwidth,
                     spectral_contrast, spectral_rolloff, spectral_flatness, rms, harmonic_to_noise,
                     chroma_stft, chroma_cens, mfcc, pitches.mean(), rhythmics])

        # Odczyt gatunku z nazwy pliku (nazwa pliku jest w formacie "nazwaGatunku_numer.wav")
        genre = file_name.split('_')[0]
        genres.append(genre)


# Utworzenie ramki danych Pandas
df = pd.DataFrame(data, columns=['tempo', 'key', 'zero_crossing_rate', 'spectral_centroid', 'spectral_bandwidth',
                                 'spectral_contrast', 'spectral_rolloff', 'spectral_flatness', 'rms', 'harmonic_to_noise',
                                 'chroma_stft', 'chroma_cens', 'mfcc', 'pitches', 'rhythmics'])

# Przekształcenie kolumn na typ danych numeryczny
df = df.astype(float)

# Dodanie kolumny z gatunkami
df['genre'] = genres
```

Rysunek 4. Kod służący do wyodrębnienia cech i stworzenia DataFrame

In [26]:
```
#Tabela z danymi
df
```

Out[26]:

	tempo	key	zero_crossing_rate	spectral_centroid	spectral_bandwidth	spectral_contrast	spectral_rolloff	spectral_flatness	rms	harmonic_
0	129.199219	129.199219	0.095657	3042.639670	2949.797337	22.781430	6812.530765	0.026945	0.301268	-
1	129.199219	129.199219	0.136758	3024.923494	2889.150594	22.226640	6544.857046	0.046728	0.262533	-
2	129.199219	129.199219	0.093199	2873.674501	2923.080842	21.355991	6569.918493	0.030516	0.301514	-
3	103.359375	172.265625	0.118420	2957.629037	2984.659321	22.335675	6791.761409	0.035638	0.246190	
4	129.199219	129.199219	0.106564	2651.502555	2755.807988	21.982968	5879.801157	0.032928	0.203660	-
...	...	...	...	...	...	...	...	...	...	
145	129.199219	129.199219	0.134944	2743.729337	2630.222207	22.543786	5525.994626	0.032376	0.172026	-
146	107.666016	78.302557	0.093187	2168.408659	2326.458140	21.586055	4414.779498	0.015682	0.254031	-
147	112.347147	215.332031	0.150833	2670.531735	2356.521242	22.244157	5213.690433	0.031758	0.126142	-
148	83.354335	83.354335	0.112529	2455.482027	2514.236146	22.051774	5232.604733	0.022485	0.105014	-
149	151.999081	151.999081	0.095997	2223.677841	2435.224461	22.647425	4672.377715	0.016273	0.209982	

150 rows × 16 columns

Rysunek 5. Tabela z danymi cz.1

In [26]:
```
#Tabela z danymi
df
```

Out[26]:

l_bandwidth	spectral_contrast	spectral_rolloff	spectral_flatness	rms	harmonic_to_noise	chroma_stft	chroma_cens	mfcc	pitches	rhythmics	genre
2949.797337	22.781430	6812.530765	0.026945	0.301268	-0.000042	0.430720	0.253709	8.713091	10.902593	0.207383	dance
2889.150594	22.226640	6544.857046	0.046728	0.262533	-0.000047	0.475112	0.268532	7.238986	25.007980	0.178050	dance
2923.080842	21.355991	6569.918493	0.030516	0.301514	-0.000125	0.494209	0.253327	9.006914	10.639940	0.216991	dance
2984.659321	22.335675	6791.761409	0.035638	0.246190	0.000023	0.410645	0.263831	5.377316	19.754229	0.166068	dance
2755.807988	21.982968	5879.801157	0.032928	0.203660	-0.000014	0.464160	0.268960	2.052255	30.200880	0.150502	dance
...	...	...	...	...	...	...	...	...	...	...	...
2630.222207	22.543786	5525.994626	0.032376	0.172026	-0.000004	0.417779	0.243927	6.651185	34.489635	0.234211	rock
2326.458140	21.586055	4414.779498	0.015682	0.254031	-0.000033	0.432371	0.257195	7.846785	21.682793	0.192932	rock
2356.521242	22.244157	5213.690433	0.031758	0.126142	-0.000074	0.403091	0.268691	1.445697	64.056343	0.223102	rock
2514.236146	22.051774	5232.604733	0.022485	0.105014	-0.000020	0.352254	0.228826	1.545673	23.297522	0.192881	rock
2435.224461	22.647425	4672.377715	0.016273	0.209982	0.000033	0.344080	0.256167	3.116391	24.719164	0.175359	rock

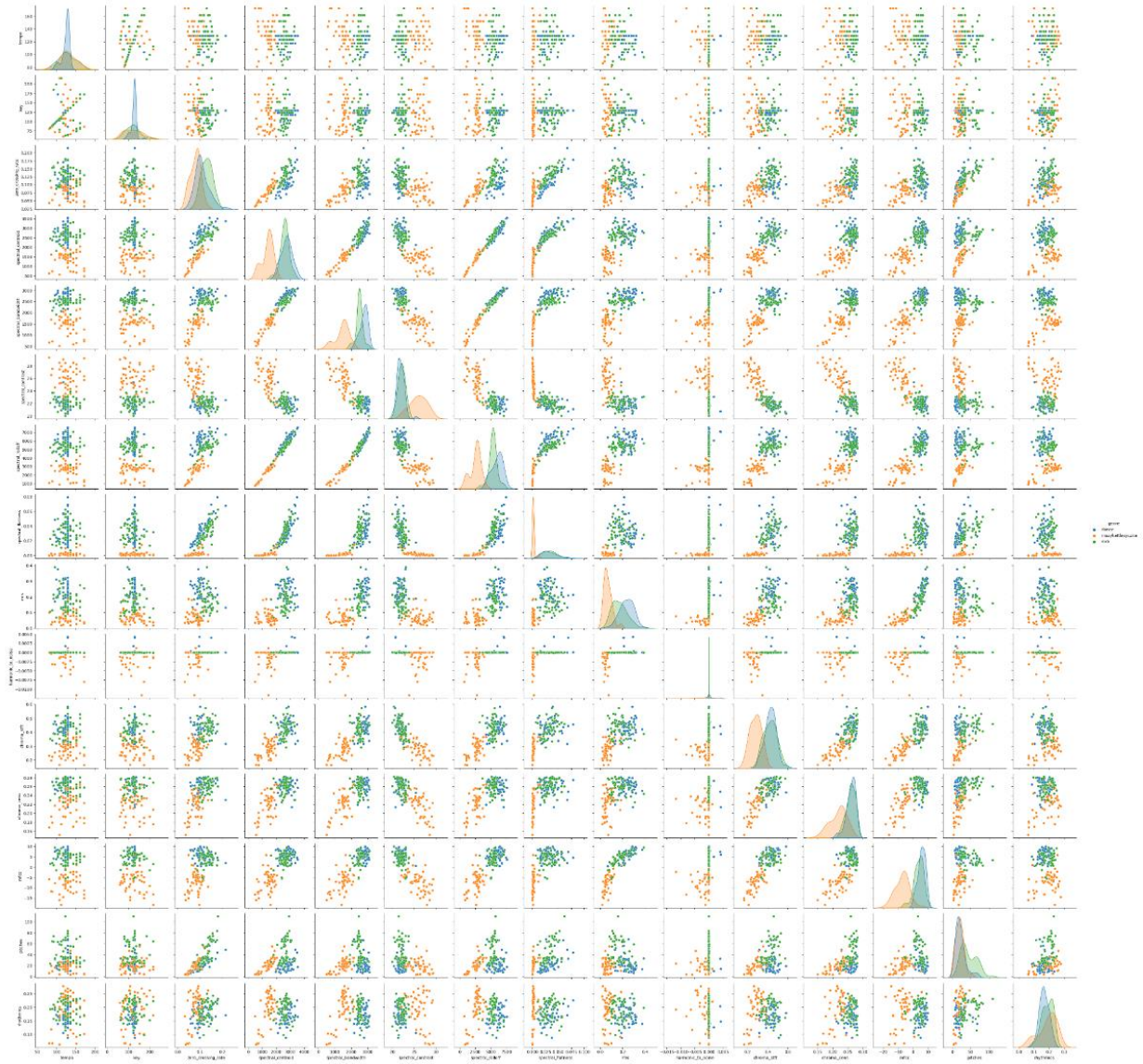
Rysunek 6. Tabela z danymi cz.2

Powyższa tabela danych posłużyła mi do dalszej pracy nad stworzeniem modelu.

4. Opracowanie wyników

Po wyodrębnieniu cech przystąpiłem do analizy atrybutów przy użyciu wykresu pairplot oraz macierzy korelacji. Przy macierzy korelacji zwracałem uwagę na poziom korelacji poszczególnych atrybutów (jeśli korelacja pomiędzy dwoma atrybutami wynosi >=0.9 to jeden z nich jest usuwany). W przypadku wykresu pairplot patrzyłem na to, jak bardzo wyróżniają się klasy (gatunki) dla danego atrybutu. Na tej podstawie decydowałem czy dany atrybut zostaje w zbiorze, czy jest zbędny.

Out[28]: <seaborn.axisgrid.PairGrid at 0x175e60e4510>

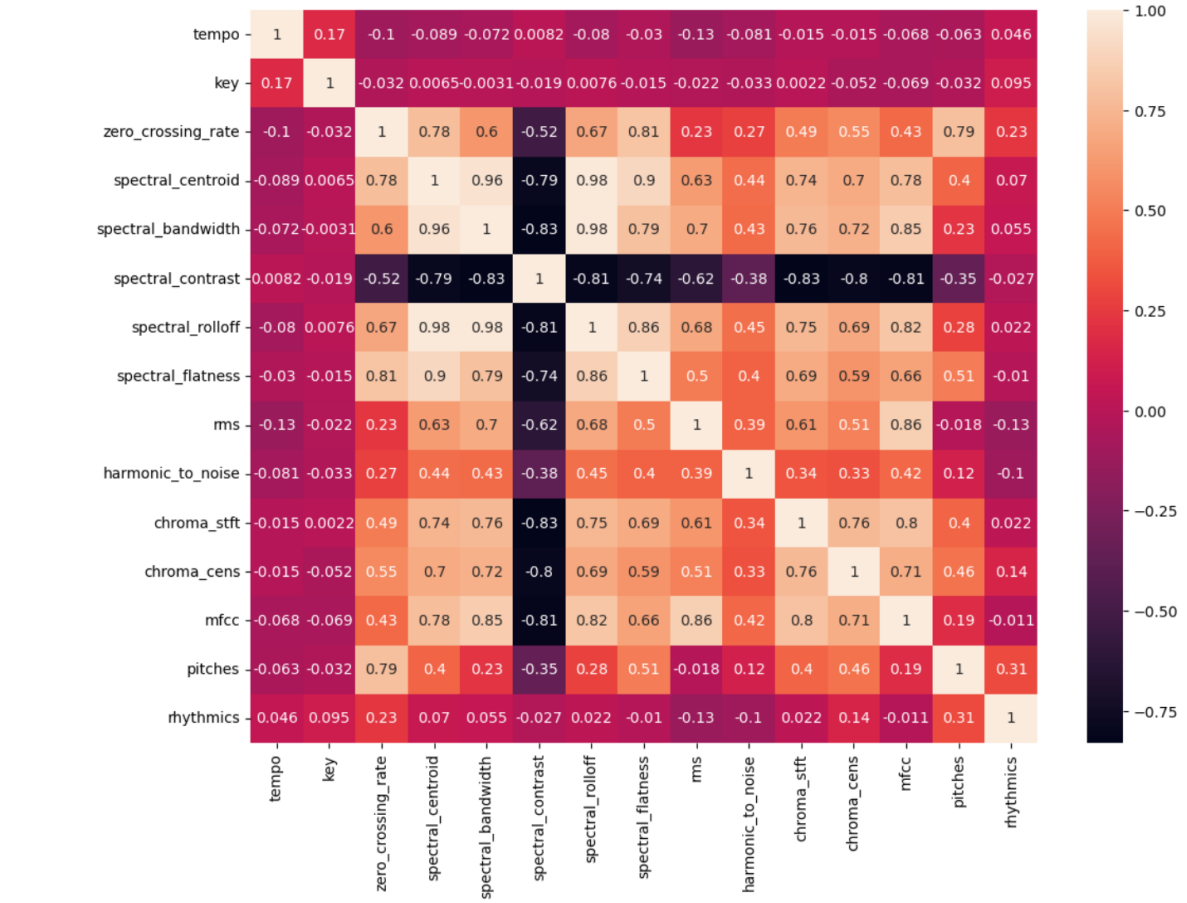


Rysunek 7. Wykres pairplot

Kolor:

- Niebieski - Dance
- Żółty - Muzyka Klasyczna
- Czerwony - Rock

Out[29]: <Axes: >



Rysunek 8. Macierz korelacji

Na podstawie analizy, w ostatecznym rozrachunku usunięto następujące cechy:

- Szerokość pasm spektralnych - 'spectral_bandwidth'
- Kontrast spektralny - 'spectral_contrast'
- Rolloff spektralny - 'spectral_rolloff'
- Płaskość spektralna - 'spectral_flatness'
- Rytmika - 'rhythmics'

W kolejnym etapie przeszedłem do klasyfikacji. Najpierw podzieliłem dane na zestawy uczący i testowy w proporcji 60:40. Zrobiłem to zarówno dla zbioru ze wszystkimi cechami przed redukcją, jak i dla zbioru z cechami pozostawionymi po analizie, aby ocenić jak bardzo wcześniejsza analiza atrybutów przyczyniła się do wyników dokładności modelu.

bez analizy atrybutów

```
In [33]: # Podział zbioru na uczący i testowy w proporcji 60:40
df_lrn_and_test1 = divide(df, 0.4)
df_dscr_lrn1 = df_lrn_and_test1['dscr_lrn']
df_dscr_test1 = df_lrn_and_test1['dscr_test']
df_dec_lrn1 = df_lrn_and_test1['dec_lrn']
df_dec_test1 = df_lrn_and_test1['dec_test']

print('Liczba obiektów zbioru uczącego: ', len(df_dscr_lrn1))
print('Liczba obiektów zbioru testowego: ', len(df_dscr_test1))
```

```
Liczba obiektów zbioru uczącego:  90
Liczba obiektów zbioru testowego:  60
```

```
In [34]: #Lista wszystkich atrybutów zbioru df
atr_list1 = list(range(df.shape[1]-1))
```

z analizą atrybutów

```
In [35]: # Podział zbioru na uczący i testowy w proporcji 60:40
df_lrn_and_test2 = divide(df_reduced, 0.4)
df_dscr_lrn2 = df_lrn_and_test2['dscr_lrn']
df_dscr_test2 = df_lrn_and_test2['dscr_test']
df_dec_lrn2 = df_lrn_and_test2['dec_lrn']
df_dec_test2 = df_lrn_and_test2['dec_test']

print('Liczba obiektów zbioru uczącego: ', len(df_dscr_lrn2))
print('Liczba obiektów zbioru testowego: ', len(df_dscr_test2))
```

```
Liczba obiektów zbioru uczącego:  90
Liczba obiektów zbioru testowego:  60
```

```
In [36]: #Lista wszystkich atrybutów zbioru dr_reduced
atr_list2 = list(range(df_reduced.shape[1]-1))
```

Rysunek 9. Podział obu zbiorów na testowy i uczący

Następnie zastosowałem trzy klasyfikatory:

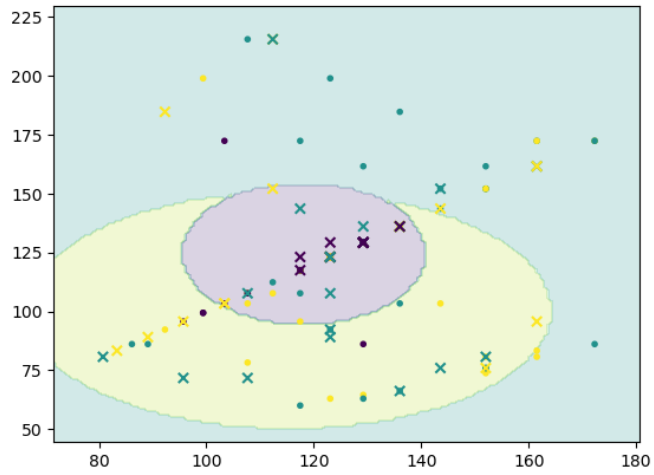
- Naiwny klasyfikator Bayesa
- Las losowy
- XGBoost

Naiwny klasyfikator Bayesa

In [37]:
```
# Tworzenie modelu naiwnego klasyfikatora Bayesa (wszystkie cechy)
bayes1 = GaussianNB()

verify(bayes1, df_dscr_lrn1, df_dscr_test1, df_dec_lrn1, df_dec_test1, atr_list1)
lims(bayes1, df_dscr_lrn1, df_dscr_test1, df_dec_lrn1, df_dec_test1, 0, 1)

# Trenowanie modelu
bayes1.fit(df_dscr_lrn1, df_dec_lrn1)

# Ocena wydajności modelu
y_pred = bayes1.predict(df_dscr_test1)
accuracy = accuracy_score(df_dec_test1, y_pred)
print("Dokładność modelu:", accuracy)
print("Raport klasyfikacji:")
print(classification_report(df_dec_test1, y_pred))
```

```
Macierz pomylek dla zbioru uczącego, dokładność: 0.9
col_0   0   1   2
row_0
0      27   0   3
1       0  30   0
2       5   1  24
Macierz pomylek dla zbioru testowego, dokładność: 0.8
col_0   0   1   2
row_0
0      16   1   3
1       0  19   1
2       7   0  13
Dokładność modelu: 0.8
Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.70      0.80      0.74        20
           1       0.95      0.95      0.95        20
           2       0.76      0.65      0.70        20

    accuracy                           0.80        60
   macro avg       0.80      0.80      0.80        60
weighted avg       0.80      0.80      0.80        60
```

```
C:\Users\barte\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but GaussianNB was fitted with feature names
  warnings.warn(
```

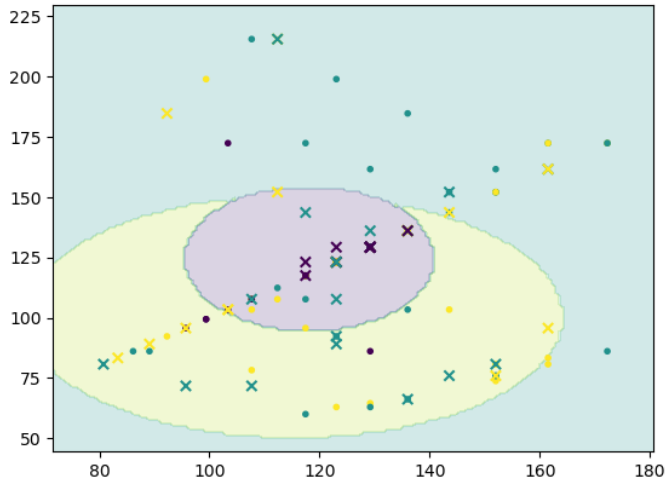


Rysunek 10. Wyniki dla Naiwnego klasyfikatora Bayesa dla zbioru z wszystkimi atrybutami

In [38]:
```
# Tworzenie modelu naiwnego klasyfikatora Bayesa (wybrane cechy)
bayes2 = GaussianNB()

verify(bayes2, df_dscr_lrn2, df_dscr_test2, df_dec_lrn2, df_dec_test2, atr_list2)
lims(bayes2, df_dscr_lrn2, df_dscr_test2, df_dec_lrn2, df_dec_test2, 0, 1)

# Trenowanie modelu
bayes2.fit(df_dscr_lrn2, df_dec_lrn2)

# Ocena wydajności modelu
y_pred = bayes2.predict(df_dscr_test2)
accuracy = accuracy_score(df_dec_test2, y_pred)
print("Dokładność modelu:", accuracy)
print("Raport klasyfikacji:")
print(classification_report(df_dec_test2, y_pred))
```

```
Macierz pomylek dla zbioru uczącego, dokładność: 0.9
col_0   0   1   2
row_0
0      27   0   3
1       0  30   0
2       6   0  24
Macierz pomylek dla zbioru testowego, dokładność: 0.8333333333333334
col_0   0   1   2
row_0
0      18   0   2
1       0  20   0
2       8   0  12
Dokładność modelu: 0.8333333333333334
Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.69      0.90      0.78        20
           1       1.00      1.00      1.00        20
           2       0.86      0.60      0.71        20

    accuracy                           0.83        60
   macro avg       0.85      0.83      0.83        60
weighted avg       0.85      0.83      0.83        60
```

```
C:\Users\barte\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but GaussianN
B was fitted with feature names
  warnings.warn(
```



Rysunek 11. Wyniki dla Naiwnego klasyfikatora Bayesa dla zbioru z wybranymi atrybutami

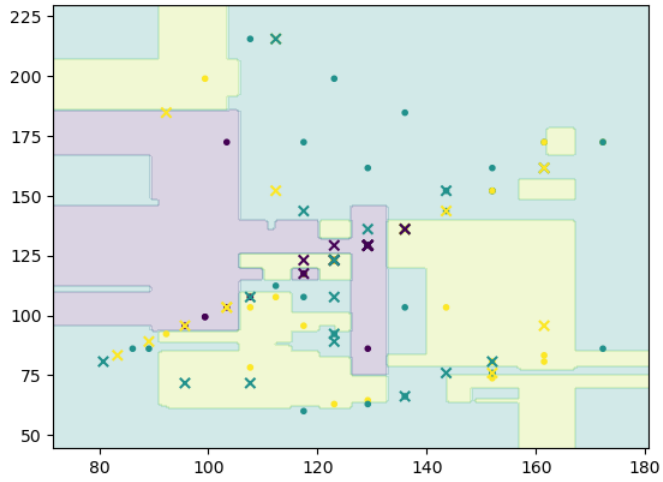
Las losowy

In [39]:
```
# Tworzenie modelu lasu losowego (wszystkie cechy)
randomForest1 = RandomForestClassifier(n_estimators=100, random_state=42)

verify(randomForest1, df_dscr_lrn1, df_dscr_test1, df_dec_lrn1, df_dec_test1, atr_list1)
lims(randomForest1, df_dscr_lrn1, df_dscr_test1, df_dec_lrn1, df_dec_test1, 0, 1)

# Trenowanie modelu
randomForest1.fit(df_dscr_lrn1, df_dec_lrn1)

# Ocena wydajności modelu
y_pred = randomForest1.predict(df_dscr_test1)
accuracy = accuracy_score(df_dec_test1, y_pred)
print("Dokładność modelu:", accuracy)
print("Raport klasyfikacji:")
print(classification_report(df_dec_test1, y_pred))
```

```
Macierz pomylek dla zbioru uczącego, dokładność: 1.0
col_0   0   1   2
row_0
0      30   0   0
1       0  30   0
2       0   0  30
Macierz pomylek dla zbioru testowego, dokładność: 0.8
col_0   0   1   2
row_0
0      17   0   3
1       0  17   3
2       6   0  14
```

```
C:\Users\barte\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but RandomFor
estClassifier was fitted with feature names
  warnings.warn(
```

```
Dokładność modelu: 0.8
Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.74      0.85      0.79        20
           1       1.00      0.85      0.92        20
           2       0.70      0.70      0.70        20

    accuracy                           0.80        60
   macro avg       0.81      0.80      0.80        60
weighted avg       0.81      0.80      0.80        60
```



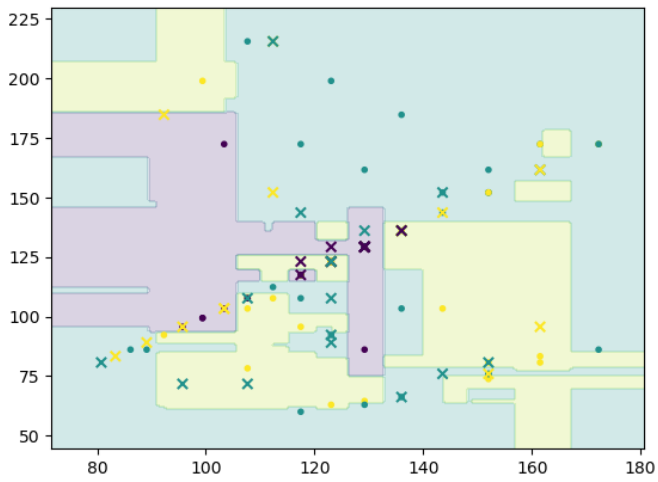
Rysunek 12. Wyniki dla Lasu losowego dla zbioru z wszystkimi atrybutami

In [40]:
```
# Tworzenie modelu lasu losowego (wybrane cechy)
randomForest2 = RandomForestClassifier(n_estimators=100, random_state=42)

verify(randomForest2, df_dscr_lrn2, df_dscr_test2, df_dec_lrn2, df_dec_test2, atr_list2)
lims(randomForest2, df_dscr_lrn2, df_dscr_test2, df_dec_lrn2, df_dec_test2, 0, 1)

# Trenowanie modelu
randomForest2.fit(df_dscr_lrn2, df_dec_lrn2)

# Ocena wydajności modelu
y_pred = randomForest2.predict(df_dscr_test2)
accuracy = accuracy_score(df_dec_test2, y_pred)
print("Dokładność modelu:", accuracy)
print("Raport klasyfikacji:")
print(classification_report(df_dec_test2, y_pred))
```

```
Macierz pomylek dla zbioru uczącego, dokładność: 1.0
col_0   0   1   2
row_0
0      30   0   0
1       0  30   0
2       0   0  30
Macierz pomylek dla zbioru testowego, dokładność: 0.9166666666666666
col_0   0   1   2
row_0
0      17   1   2
1       0  20   0
2       2   0  18
```

```
C:\Users\barte\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X does not have valid feature names, but RandomFor
estClassifier was fitted with feature names
  warnings.warn(
```

```
Dokładność modelu: 0.9166666666666666
Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.89      0.85      0.87        20
           1       0.95      1.00      0.98        20
           2       0.90      0.90      0.90        20

    accuracy                           0.92        60
   macro avg       0.92      0.92      0.92        60
weighted avg       0.92      0.92      0.92        60
```

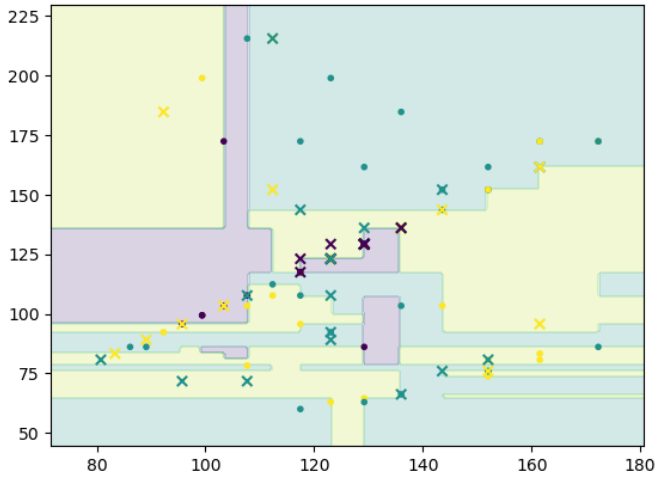


Rysunek 13. Wyniki dla Lasu losowego dla zbioru z wybranymi atrybutami

XGBoost

In [42]:
```
# Tworzenie modelu XGBoost (wszystkie cechy)
xgb_model1 = XGBClassifier()

verify(xgb_model1, df_dscr_lrn1, df_dscr_test1, df_dec_lrn1, df_dec_test1, atr_list1)
lims(xgb_model1, df_dscr_lrn1, df_dscr_test1, df_dec_lrn1, df_dec_test1, 0, 1)

# Trenowanie modelu
xgb_model1.fit(df_dscr_lrn1, df_dec_lrn1)

# Ocena wydajności modelu
y_pred = xgb_model1.predict(df_dscr_test1)
accuracy = accuracy_score(df_dec_test1, y_pred)
print("Dokładność modelu:", accuracy)
print("Raport klasyfikacji:")
print(classification_report(df_dec_test1, y_pred))
```

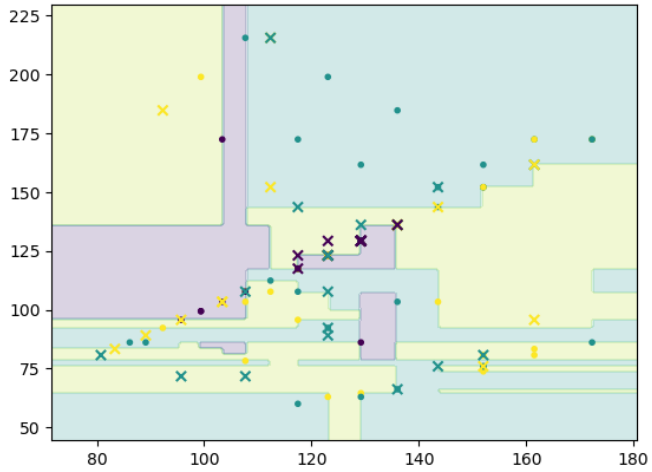
```
Macierz pomylek dla zbioru uczącego, dokładność: 1.0
col_0   0   1   2
row_0
0      30   0   0
1       0  30   0
2       0   0  30
Macierz pomylek dla zbioru testowego, dokładność: 0.8166666666666667
col_0   0   1   2
row_0
0      17   0   3
1       0  17   3
2       5   0  15
Dokładność modelu: 0.8166666666666667
Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.77      0.85      0.81        20
           1       1.00      0.85      0.92        20
           2       0.71      0.75      0.73        20

    accuracy                           0.82        60
   macro avg       0.83      0.82      0.82        60
weighted avg       0.83      0.82      0.82        60
```



Rysunek 14. Wyniki dla XGBoost dla zbioru z wszystkimi atrybutami

In [43]:
```
# Tworzenie modelu XGBoost (wybrane cechy)
xgb_model2 = XGBClassifier()

verify(xgb_model2, df_dscr_lrn2, df_dscr_test2, df_dec_lrn2, df_dec_test2, atr_list2)
lims(xgb_model2, df_dscr_lrn2, df_dscr_test2, df_dec_lrn2, df_dec_test2, 0, 1)

# Trenowanie modelu
xgb_model2.fit(df_dscr_lrn2, df_dec_lrn2)

# Ocena wydajności modelu
y_pred = xgb_model2.predict(df_dscr_test2)
accuracy = accuracy_score(df_dec_test2, y_pred)
print("Dokładność modelu:", accuracy)
print("Raport klasyfikacji:")
print(classification_report(df_dec_test2, y_pred))
```

```
Macierz pomylek dla zbioru uczącego, dokładność: 1.0
col_0   0   1   2
row_0
0      30   0   0
1       0  30   0
2       0   0  30
Macierz pomylek dla zbioru testowego, dokładność: 0.8833333333333333
col_0   0   1   2
row_0
0      16   2   2
1       0  20   0
2       3   0  17
Dokładność modelu: 0.8833333333333333
Raport klasyfikacji:
              precision    recall  f1-score   support

           0       0.84      0.80      0.82        20
           1       0.91      1.00      0.95        20
           2       0.89      0.85      0.87        20

    accuracy                           0.88        60
   macro avg       0.88      0.88      0.88        60
weighted avg       0.88      0.88      0.88        60
```



Rysunek 15. Wyniki dla XGBoost dla zbioru z wybranymi atrybutami

Jak widać, w przypadku każdego z trzech wybranych klasyfikatorów, dokładność modelu jest zdecydowanie większa dla zbioru z wybranymi atrybutami (po analizie). Oznacza to, że analiza została przeprowadzona prawidłowo, z pozytywnym skutkiem, a decyzja o pozostawieniu bądź usunięciu danych atrybutów prawidłowa.

Wyniki klasyfikacji pokazały, że model Random Forest na zredukowanych danych osiągnął najwyższą dokładność na poziomie 92%. Model ten został zapisany do pliku ‘model.pkl’ przy użyciu biblioteki *pickle*.

```
In [44]: # Zapisanie modelu
         import pickle
         with open('model.pkl', 'wb') as file:
             pickle.dump(randomForest2, file)
```

Rysunek 16. Zapisanie modelu do pliku model.pkl

5. Opis aplikacji

Stworzona aplikacja webowa wykorzystuje framework Flask. Aplikacja pozwala użytkownikom na wgrywanie utworów, które są następnie analizowane pod kątem gatunku przy użyciu zapisanego modelu. W aplikacji użyto również HTML, CSS oraz JavaScript do stworzenia interfejsu użytkownika.

Działa ona w następujący sposób:

1. Użytkownik wgrywa utwór do aplikacji.
2. Utwór jest skracany do 6 sekund.
3. Wyodrębniane są cechy z utworu przy użyciu tej samej metody co przy tworzeniu modelu.
4. Model zapisany w model.pkl przewiduje gatunek utworu na podstawie wyodrębnionych cech.
5. Wynik jest zwracany użytkownikowi.

```python
import pickle
from flask import Flask, request, jsonify, render_template
import numpy as np
import os
import pandas as pd
import librosa
from pydub import AudioSegment

app = Flask(__name__)

# Załadowanie modelu
with open('model.pkl', 'rb') as file:
    model = pickle.load(file)

# Mapa wyników predykcji na nazwy gatunków
genre_map = {0: 'Dance', 1: 'Muzyka Klasyczna', 2: 'Rock'}

1 usage
def extract_features(file_name):
    y, sr = librosa.load(file_name)
    onset_env = librosa.onset.onset_strength(y=y, sr=sr)

    # Wyodrębnienie cech
    tempo = librosa.feature.tempo(onset_envelope=onset_env, sr=sr)[0]  # Tempo
    key, _ = librosa.beat.beat_track(y=y, sr=sr)  # Tonacja
    zero_crossing_rate = librosa.feature.zero_crossing_rate(y).mean()  # Liczba przejść przez zero
    spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr).mean()  # Środek ciężkości widma
    #spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr).mean()  # Szerokość pasm spektralnych
    #spectral_contrast = librosa.feature.spectral_contrast(y=y, sr=sr).mean()  # Kontrast spektralny
    #spectral_rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr).mean()  # Rolloff spektralny
    #spectral_flatness = librosa.feature.spectral_flatness(y=y).mean()  # Płaskość spektralna
    rms = librosa.feature.rms(y=y).mean()  # Skuteczna wartość średnia (RMS - Root Mean Square)
    harmonic_to_noise = librosa.effects.harmonic(y).mean()  # Stosunek harmoniczności do szumu (Harmonic-to-Noise Ratio)
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr).mean()  # Chromagram
    chroma_cens = librosa.feature.chroma_cens(y=y, sr=sr).mean()
    mfcc = librosa.feature.mfcc(y=y, sr=sr).mean()  # Współczynniki cepstralne (MFCCs)
    pitches, magnitudes = librosa.piptrack(y=y, sr=sr)  # Częstotliwość fundamentalna (Pitch)
    #rhythmics = librosa.feature.tempogram(y=y, sr=sr).mean()  # Rytmika

    return np.array([tempo, key.mean(), zero_crossing_rate, spectral_centroid, rms, harmonic_to_noise,
                     chroma_stft, chroma_cens, mfcc, pitches.mean()])
```

```python
1 usage
def shorten_audio(file_path, duration=6):
    audio = AudioSegment.from_file(file_path)
    start_time = (len(audio) / 2) - (duration * 1000 / 2)
    end_time = start_time + (duration * 1000)
    shortened_audio = audio[start_time:end_time]
    shortened_file_path = "temp/shortened_" + os.path.basename(file_path)
    shortened_audio.export(shortened_file_path, format="wav")
    return shortened_file_path

@app.route('/')
def home():
    return render_template('index.html')

11 usages (11 dynamic)
@app.route( rule: '/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return "No file part", 400

    file = request.files['file']

    if file.filename == '':
        return "No selected file", 400

    try:
        print("File received:", file.filename)
        file_path = os.path.join("temp", file.filename)
        file.save(file_path)
        print("File saved at:", file_path)

        shortened_file_path = shorten_audio(file_path, duration=6)
        print("Shortened file saved at:", shortened_file_path)

        features = extract_features(shortened_file_path)
        print("Features extracted:", features)

        features_df = pd.DataFrame( data: [features], columns=['tempo', 'key', 'zero_crossing_rate', 'spectral_centroid',
                                   'rms', 'harmonic_to_noise', 'chroma_stft', 'chroma_cens', 'mfcc', 'pitches'])

        prediction = model.predict(features_df)

        genre = genre_map[int(prediction[0])]
        print("Prediction made:", genre)

        # Usunięcie tymczasowego pliku
        os.remove(file_path)
        os.remove(shortened_file_path)

        return jsonify({'prediction': genre})

    except Exception as e:
        print("Error during prediction:", e)
        return str(e), 500


if __name__ == "__main__":
    if not os.path.exists("temp"):
        os.makedirs("temp")
    app.run(debug=False)
```

Rysunek 17. app.py - główny skrypt aplikacji Flask, który definiuje serwer i logikę aplikacji

Opis:

- **Importowanie bibliotek**: Importujemy niezbędne biblioteki, w tym Flask, pickle, numpy, pandas, librosa oraz AudioSegment z pydub.
- **Inicjalizacja aplikacji Flask**: Tworzymy instancję aplikacji Flask.
- **Załadowanie modelu**: Ładujemy wcześniej zapisany model z pliku model.pkl.
- **Mapa gatunków**: Definiujemy mapę, która przekształca wyniki predykcji na nazwy gatunków muzycznych.
- **Funkcja extract_features**: Wyodrębnia cechy z pliku audio. Utwór jest ładowany, a następnie wyodrębniane są różne cechy, takie jak tempo, tonacja, liczba przejść przez zero, środek ciężkości widma, skuteczna wartość średnia, stosunek harmoniczności do szumu, chromagram, współczynniki cepstralne i częstotliwość fundamentalna.
- **Funkcja shorten_audio**: Skraca utwór do 6 sekund ze środka. Utwór jest ładowany, skracany do odpowiedniego fragmentu, a następnie zapisywany do pliku.
- **Strona główna**: Funkcja home renderuje szablon index.html.
- **Przewidywanie gatunku**: Funkcja predict obsługuje przesłany plik, zapisuje go, skraca, wyodrębnia cechy i przewiduje gatunek muzyczny przy użyciu modelu. Wynik jest zwracany jako JSON.
- **Uruchamianie serwera**: Jeśli skrypt jest uruchamiany bezpośrednio, startuje serwer Flask. Tworzony jest również folder temp do przechowywania tymczasowych plików, jeśli nie istnieje.

```html
<!DOCTYPE html>
<html lang="pl">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Rozpoznawanie gatunków muzycznych</title>
    <link rel="stylesheet" href="/static/styles.css">
</head>
<body>
    <div class="container">
        <div class="header">
            <div class="icon">
                <img src="/static/music_note.png" alt="Music Note Icon">
            </div>
            <h1>Rozpoznawanie gatunków muzycznych</h1>
        </div>
        <form action="/predict" method="post" enctype="multipart/form-data">
            <div class="file-input">
                <input type="file" name="file" accept=".wav,.mp3">
            </div>
            <input type="submit" value="Sprawdź" class="btn">
        </form>
        <div id="loading" class="loading-bar" style="display: none;">
            <div class="progress"></div>
        </div>
        <div id="result">Gatunek: [predykcja]</div>
    </div>
    <script src="/static/script.js"></script>
</body>
</html>

```

Rysunek 18. index.html - definiuje strukturę strony internetowej dla aplikacji webowej

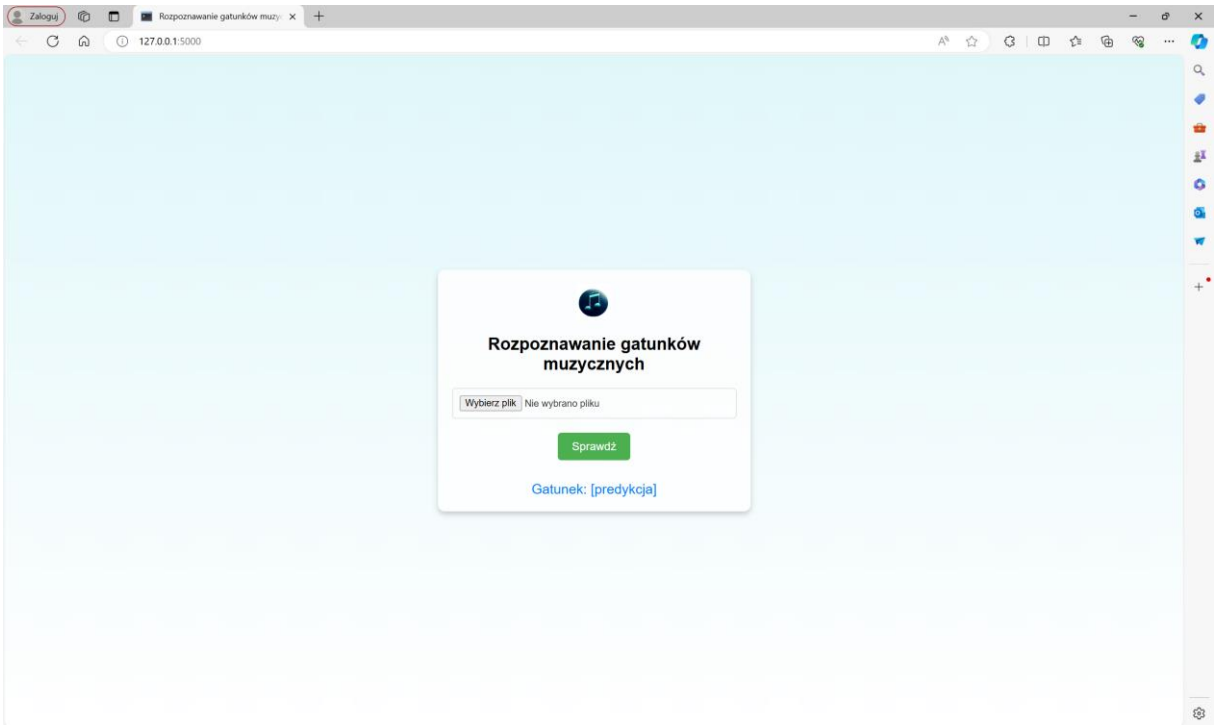
```css
body {
    background: linear-gradient(to bottom, #e0f7fa, #ffffff);
    font-family: 'Arial', sans-serif;
    color: #333;
    text-align: center;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background: rgba(255, 255, 255, 0.8);
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.2);
    text-align: center;
    width: 90%;
    max-width: 400px;
}

.header {
    margin-bottom: 20px;
}

.icon img {
    width: 50px;
    height: 50px;
}

h1 {
    font-size: 1.5em;
    color: #000;
}

form {
    margin: 20px 0;
}

.file-input {
    margin: 10px 0;
}

input[type="file"] {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 100%;
    box-sizing: border-box;
}
```

```css
input[type="submit"] {
    padding: 10px 20px;
    margin: 10px 0;
    border: none;
    border-radius: 5px;
    background-color: #4CAF50;
    color: white;
    font-size: 1em;
    cursor: pointer;
    transition: background-color 0.3s ease;
}

input[type="submit"]:hover {
    background-color: #45a049;
}

#result {
    margin-top: 20px;
    font-size: 1.2em;
    color: #007BFF;
}

.loading-bar {
    width: 100%;
    background-color: #f3f3f3;
    border: 1px solid #ccc;
    border-radius: 5px;
    overflow: hidden;
    margin-top: 20px;
}

.progress {
    width: 0;
    height: 20px;
    background-color: #4CAF50;
}
```

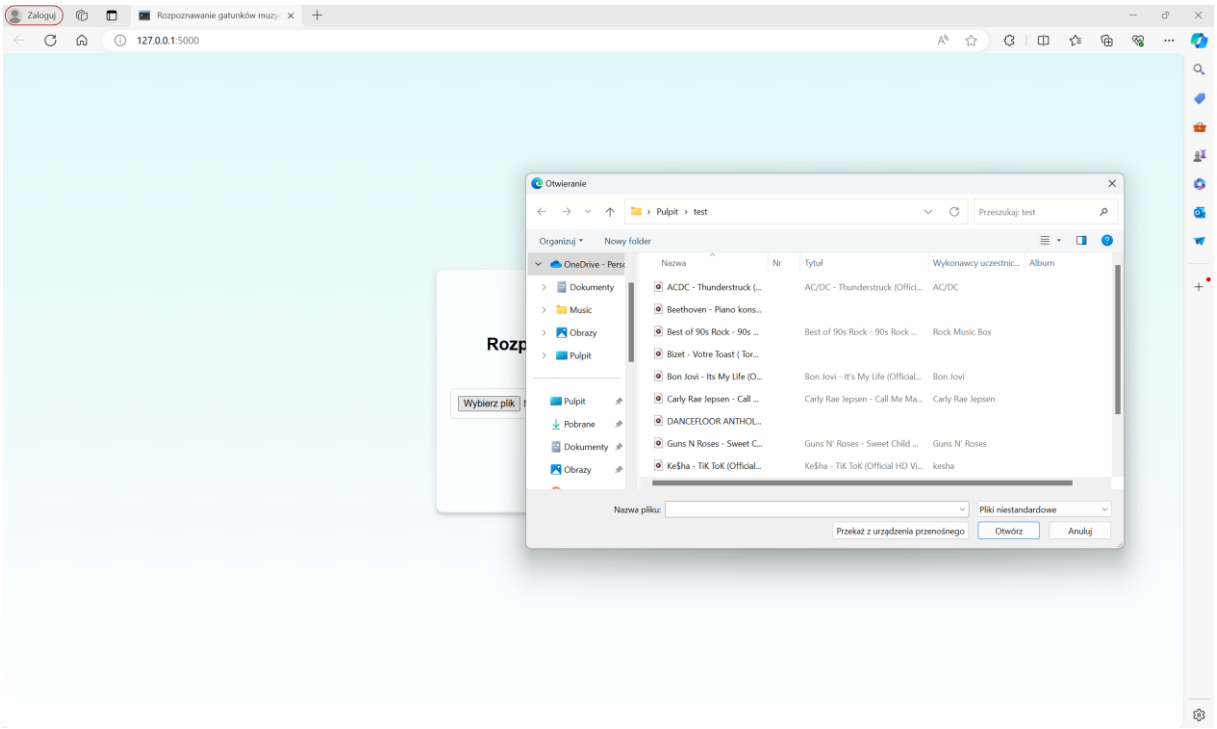
Rysunek 19. styles.css - odpowiada za stylizację strony internetowej

```javascript
document.querySelector( selectors: 'form').addEventListener( type: 'submit', listener: function(event :SubmitEvent ) :void  {
    event.preventDefault();

    // Ukryj poprzedni wynik predykcji
    document.getElementById( elementId: 'result').innerText = '';

    // Pokaż pasek ładowania
    document.getElementById( elementId: 'loading').style.display = 'block';
    let progressBar :Element  = document.querySelector( selectors: '.progress');
    progressBar.style.width = '0%';

    let formData = new FormData(this);

    // Symulacja pierwszej fazy ładowania (do 80%) wolniej i płynniej
    let progress :number  = 0;
    let firstPhaseDuration :number  = 8000; // Czas w ms dla pierwszej fazy (8 sekund)
    let firstPhaseSteps :number  = 80; // Liczba kroków do 80%
    let firstPhaseInterval :number  = firstPhaseDuration / firstPhaseSteps;

    let interval :number  = setInterval( handler: () :void  => {
        if (progress < 80) {
            progress += 1;
            progressBar.style.width = `${progress}%`;
        } else {
            clearInterval(interval);
        }
    }, firstPhaseInterval);

    let startTime :number  = new Date().getTime();

    fetch( input: '/predict', init: {
        method: 'POST',
        body: formData
    }) Promise<Response>
    .then(response :Response  => response.json()) Promise<any>
    .then(data => {
        // Oblicz czas przetwarzania
        let endTime :number  = new Date().getTime();
        let processingTime :number  = endTime - startTime;

        // Upewnij się, że pierwsza faza ładowania jest zakończona
        clearInterval(interval);

        // Symulacja drugiej fazy ładowania (od 80% do 100%) natychmiast po zakończeniu pierwszej
        let remainingTime :number  = 1000; // Czas w ms, w którym pasek zapełni się od 80% do 100%
        let remainingInterval :number  = remainingTime / 20; // 20 kroków do pełnego zapełnienia

        let secondInterval :number  = setInterval( handler: () :void  => {
            if (progress < 100) {
                progress += 1;
                progressBar.style.width = `${progress}%`;
            } else {
                clearInterval(secondInterval);
                // Ukryj pasek ładowania po zakończeniu
                document.getElementById( elementId: 'loading').style.display = 'none';
                document.getElementById( elementId: 'result').innerText = `Gatunek: ${data.prediction}`;
            }
        }, remainingInterval);
    }) Promise<void>
    .catch(error => {
        console.error('Error:', error);
        clearInterval(interval); // Upewnij się, że pierwszy interwał jest zatrzymany w przypadku błędu
        document.getElementById( elementId: 'loading').style.display = 'none';
    });
});

document.querySelector( selectors: 'input[type="file"]').addEventListener( type: 'change', listener: function(event) :void  {
    // Ukryj poprzedni wynik predykcji przy wybraniu nowego pliku
    document.getElementById( elementId: 'result').innerText = '';
});

```

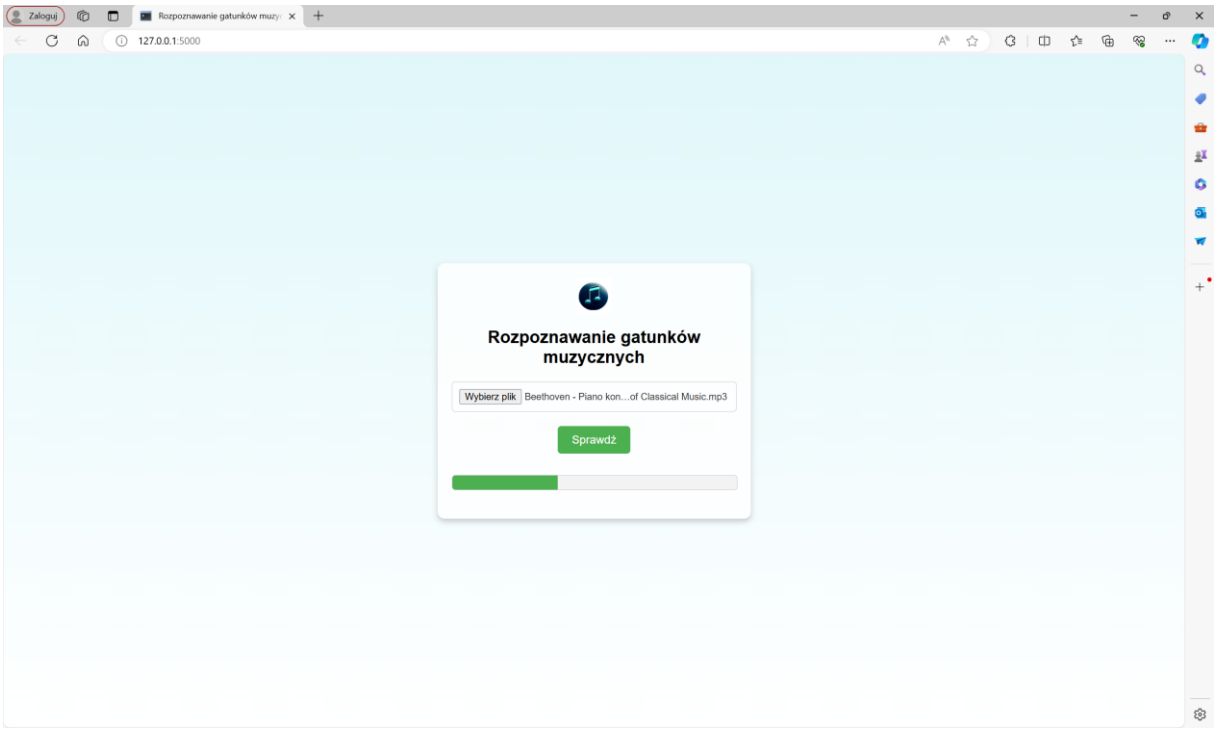
Rysunek 20. script.js - obsługuje interakcję użytkownika na stronie internetowej



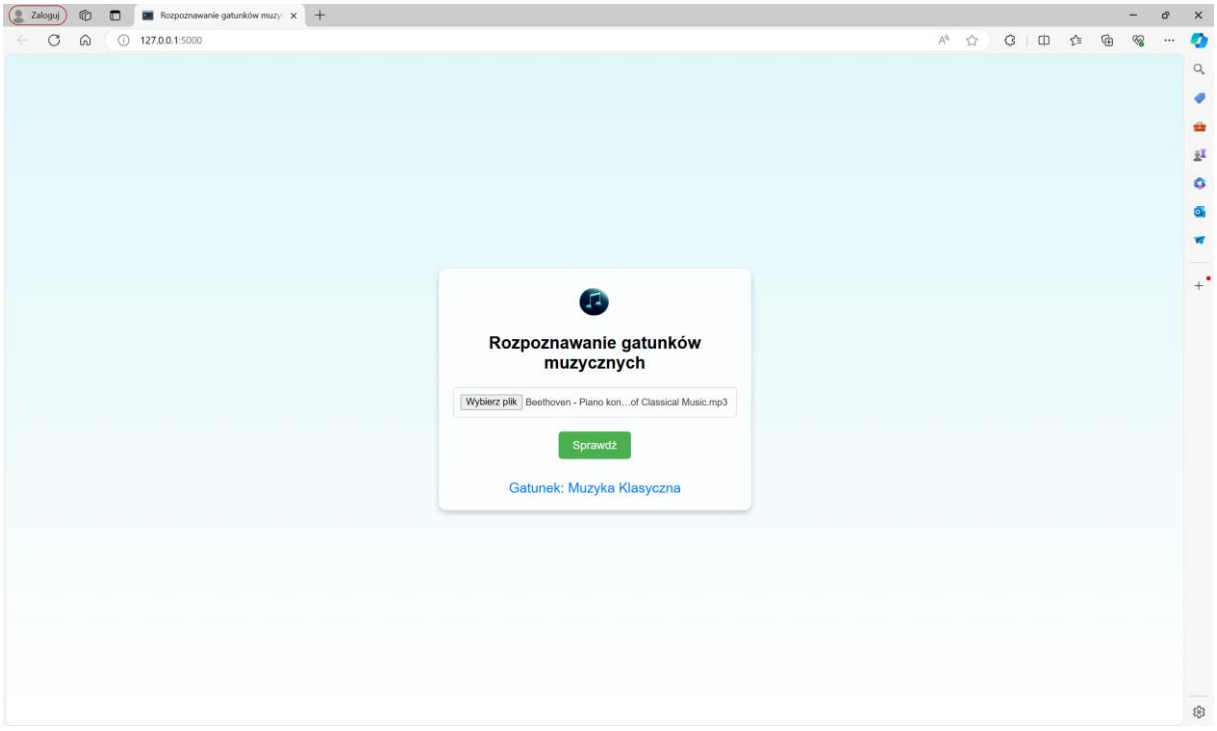
Rysunek 21. Okno aplikacji webowej



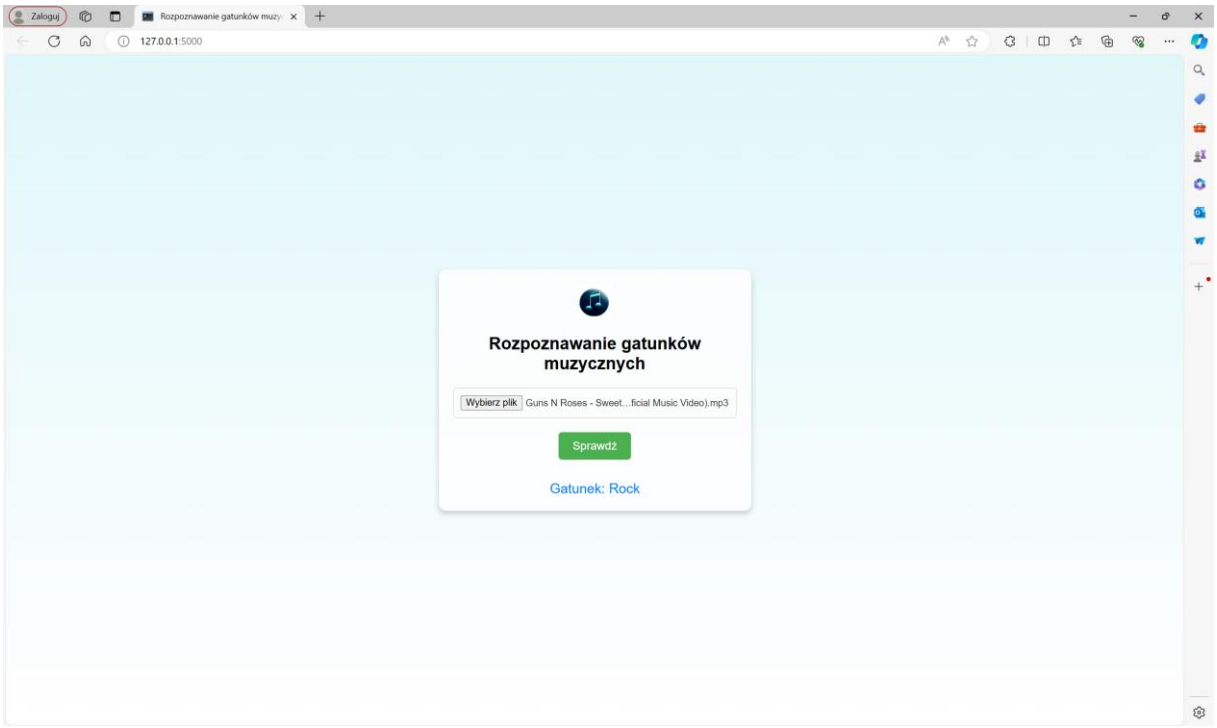
Rysunek 22. Okno aplikacji webowej – wybór utworu do rozpoznania gatunku



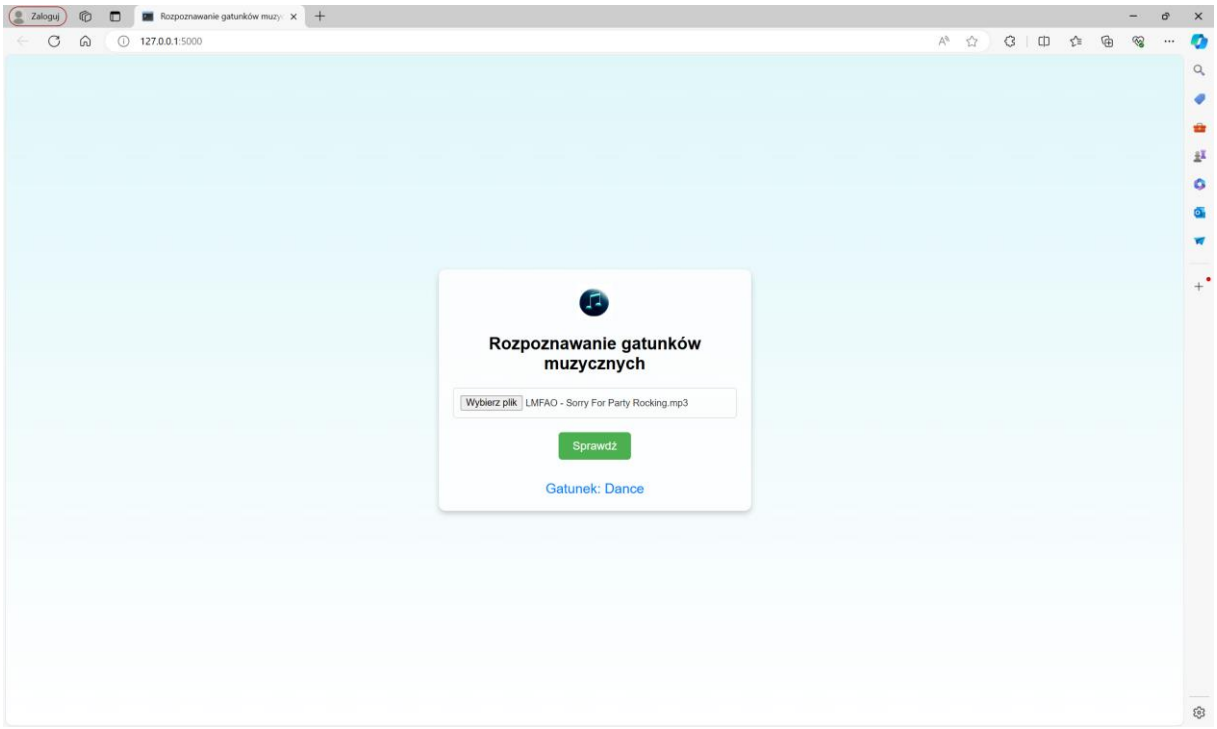
Rysunek 22. Okno aplikacji webowej – w trakcie analizy



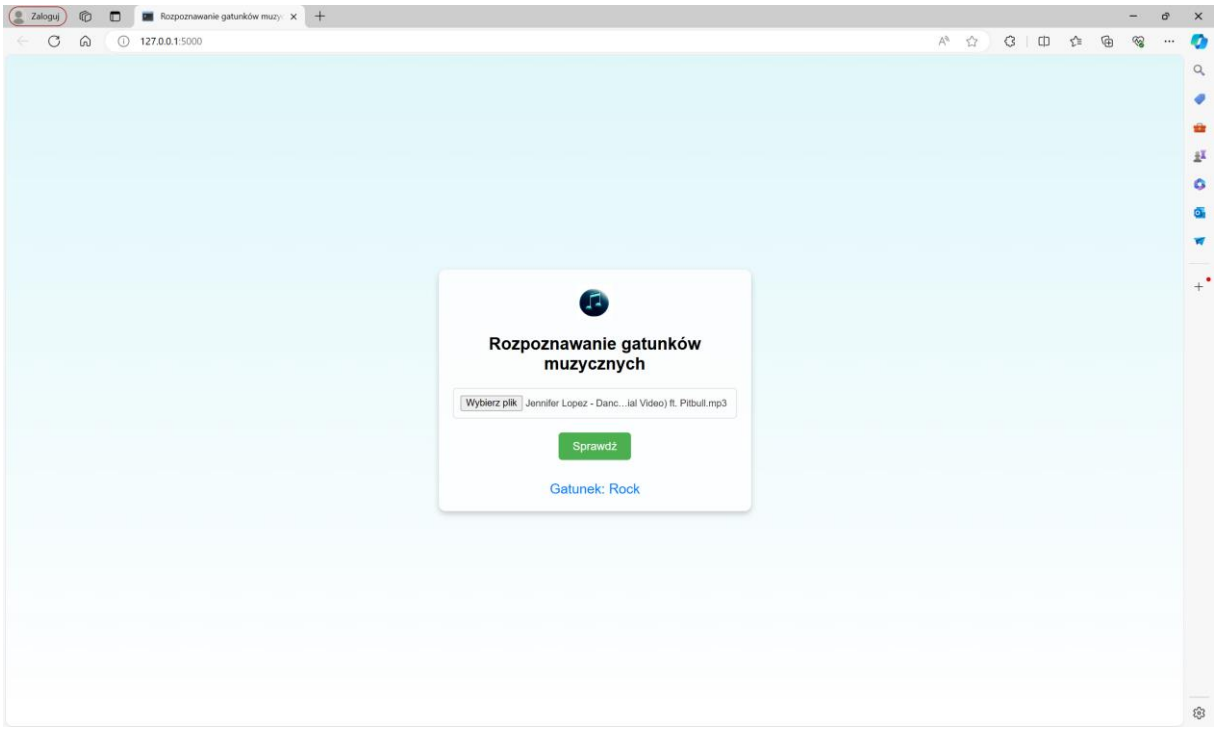
Rysunek 23. Okno aplikacji webowej – wynik (prawidłowy)



Rysunek 23. Okno aplikacji webowej – wynik (prawidłowy)



Rysunek 23. Okno aplikacji webowej – wynik (prawidłowy)



Rysunek 23. Okno aplikacji webowej – wynik (błędny, powinno być Dance)

Większość wyników jest prawidłowa ze względu na stosunkowo wysoką dokładność modelu (92%), natomiast nie zawsze działa on prawidłowo (ostatnie zdjęcie). Najlepsze wyniki są dla Muzyki Klasycznej. Radzi sobie on trochę gorzej dla gatunków Dance i Rock, co można było już zauważyć w wynikach przy tworzeniu modelu.

6. Osiągnięcia i Wyzwania

Podczas projektu udało mi się zrealizować następujące punkty:

- Zebranie utworów do zbioru.
- Wyodrębnienie cech każdego z nich i stworzenie ramki danych.
- Przeprowadzenie analizy i redukcja cech, dzięki czemu dokładności modeli były większe.
- Wytrenowanie i ocena modeli klasyfikacyjnych, a następnie zapisanie jednego z nich.
- Wykonanie aplikacji webowej, wykorzystującej stworzony model, do praktycznego zastosowania projektu.

Niestety, główną rzeczą , której nie udało mi się osiągnąć, było dodanie większej ilości gatunków. Aplikacja stałaby się dzięki temu jeszcze bardziej praktyczna.

Podczas projektu napotkałem kilka problemów. Głównym wyzwaniem była analiza dużej liczby cech i ich korelacji. Było to bardzo czasochłonne. Ciężko było zdecydować jakie atrybuty usunąć, aby dokładność modelu była jak największa. Po wykonaniu kilku kombinacji, udało mi się osiągnąć dokładność na poziomie 92%, co jest dobrym wynikiem, natomiast nie idealnym. Drugim problemem była instalacja i konfiguracja ffmpeg, która składała się z wielu kroków. Był on potrzebny do skracania wstawianych utworów w aplikacji do tej samej długości co utwory ze zbioru.

7. Możliwy rozwój projektu

Projekt można rozwijać w następujący sposób:

- Zwiększenie zbioru danych o więcej utworów i gatunków muzycznych.
- Wyodrębnienie jeszcze większej ilości przydatnych cech z utworów.
- Zastosowanie bardziej zaawansowanych technik wyodrębniania cech.
- Ulepszanie, udoskonalanie algorytmów klasyfikacyjnych w celu stworzenia jeszcze dokładniejszego modelu.
- Implementacja interaktywnej wizualizacji danych i wyników w aplikacji.

8. Praktyczne zastosowania

Projekt ma wiele praktycznych zastosowań, takich jak:

- Automatyczne tagowanie utworów muzycznych w serwisach streamingowych.
- Pomoc w organizacji kolekcji muzycznych.
- Wykorzystanie w aplikacjach rekomendujących muzykę.

9. Podsumowanie

Projekt rozpoznawania gatunków muzycznych za pomocą uczenia maszynowego z powodzeniem zastosował techniki przetwarzania sygnałów dźwiękowych i klasyfikacji, prowadząc do stworzenia funkcjonalnej aplikacji webowej. W ramach projektu zebrano i przygotowano zbiór danych składający się z 150 utworów muzycznych, reprezentujących trzy gatunki: Dance, Muzyka Klasyczna i Rock. Każdy gatunek był reprezentowany przez 50 utworów, które zostały poddane przetwarzaniu w celu wyodrębnienia charakterystycznych cech przy użyciu biblioteki librosa. Wyodrębnione cechy obejmowały tempo, tonację, liczbę przejść przez zero, środek ciężkości widma, RMS, stosunek harmoniczności do szumu, chromagram i MFCC.

Przeprowadzono analizę cech z wykorzystaniem pairplotów i macierzy korelacji, co pozwoliło na identyfikację mniej istotnych cech i ich usunięcie. Następnie zredukowany zbiór cech posłużył do trenowania modeli klasyfikacyjnych. Wykorzystano trzy klasyfikatory: Naiwny klasyfikator Bayesa, Las losowy i XGBoost, które zostały przetestowane na pełnych oraz zredukowanych zbiorach danych. Modele były trenowane i testowane na odpowiednich podziałach zbiorów danych, co pozwoliło na ocenę ich wydajności.

Wyniki pokazały, że model Las losowy na zredukowanych danych osiągnął najwyższą dokładność na poziomie 92%. Ostatecznie wybrano ten model do dalszego wykorzystania. Model został zapisany jako plik model.pkl i zaimplementowany w aplikacji webowej stworzonej przy użyciu Flask, HTML, CSS i JavaScript. Aplikacja umożliwia użytkownikom przesyłanie utworów muzycznych, które są skracane do 6-sekundowych fragmentów, a następnie analizowane przez model, który przewiduje gatunek muzyczny. Aplikacja działa sprawnie, a interfejs użytkownika jest przejrzysty i intuicyjny, oferując płynny pasek ładowania, który informuje o postępie analizy.

Projekt pokazuje potencjał wykorzystania technik uczenia maszynowego w automatycznej klasyfikacji muzyki, co może mieć szerokie zastosowanie w różnych dziedzinach, takich jak streaming muzyki, rekomendacje utworów i analiza treści muzycznych. Aplikacja może być rozwijana dalej, aby obejmować większą liczbę gatunków muzycznych oraz poprawić dokładność klasyfikacji.