

# Temat zadania: analityczny - detekcja anomalii

**Autorzy:** Michał Bartnicki, Michał Piotrak

**Kod zadania:** DAz11

**Wybrany zbiór danych:** [Credit Card Fraud Detection](#)

## Instrukcja uruchomienia

Należy uruchomić cały notatnik poprzez wywołanie funkcji "Run all". Notatnik ustawiony jest w trybie szybkim, czas trwania przetwarzania to około 10 - 15 minut. Po tym czasie, wygenerują się wszystkie wykresy i wyniki. W trybie szybkim kod uruchamiany jest na uszczuplonym zbiorze danych i parametrów, natomiast wykresy i wyniki generowane są z wcześniej zapisanych w plikach modeli i wyników.

## Wprowadzenie

Celem pracy jest porównanie skuteczności modeli uczenia maszynowego z podziałem na dwie grupy:

- uczenie nadzorowane
- uczenie nienadzorowane

w działaniu na wysoce niebilansowanym zbiorze danych, w klasyfikacji binarnej.

Zbadano następujące algorytmy:

- Las losowy,
- Drzewo decyzyjne,
- Jednoklasowy SVM,
- Las izolacyjny,
- Lokalny czynnik odstający.

Niestety, nie udało się dokonać badań dla wszystkich planowanych algorytmów, a dokładnie dla poniższych zadeklarowanych:

- SVM z wagami klas,
- XGBoost,
- DBSCAN.

Przerosła nas złożoność danego projektu, dlatego musielismy ograniczyć zakres naszych badań.

## Przygotowanie oraz konfiguracja środowiska uruchomieniowego

W tej sekcji umieszczamy listę zmiennych konfiguracyjnych, które mają wpływ na wykonanie przetwarzania zawartego w danym notatniku. Poza tym, przygotowaliśmy fragment skryptu, który zadba o instalację niezbędnych bibliotek zewnętrznych.

Rozwiążanie dostarczamy z włączonym trybem *QUICK\_MODE*, który umożliwia szybsze wykonanie przetwarzania zawartego w notatniku w celu demonstracyjnym. Wiąże się to z:

1. Wczytaniem mniejszej ilości danych ze zbioru źródłowego.
2. Wykonaniem mniejszej liczby iteracji w k-krotnej krzyżowej walidacji.
3. Wykonaniem mniejszej liczby iteracji w strojeniu parametrów, ze względu na mniejszą liczbę podanych wartości parametrów.
4. Jeśli w środowisku uruchomieniowym, mamy już wynik k-krotnej krzyżowej walidacji bądź wytrenowany model, to zostanie on wczytany z pliku.

```
In [ ]: # Lista zmiennych konfiguracyjnych

# Flaga, której ustawienie umożliwia szybsze wykonanie przetwarzania zawartego w
# w celu demonstracyjnym.
QUICK_MODE = True
# Procent danych, który bierzemy pod uwagę przy pracy w trybie QUICK_MODE
QUICK_MODE_DATA_PERCENTAGE = 0.05
```

```
# Liczba foldów (warstw, na które został podzielony wejściowy zbiór danych) w k-
CV_FOLDS_COUNT = 5
if QUICK_MODE:
    CV_FOLDS_COUNT = 2

# Flaga, której ustawienie powoduje wczytanie pliku z rezultatami strojenia para-
# który został wygenerowany wcześniej, zazwyczaj na większym zbiorze danych
LOAD_PARAMS_TUNE_RESULTS_FROM_MASTER_FILE = True

# Flaga, która określa, czy chcemy załadować przygotowany wcześniej model
LOAD_PRETRAINED_MODEL = True

# Flaga, która umożliwia ograniczenie rozmiaru zbioru treningowego do określonego
# Ma to zagwarantować lepszą wydajność oraz jakość w przypadku trenowania niektórych
# Należy pamiętać, że dane ograniczenie wpływa jedynie na zbiór treningowy, a nie
LIMIT_TRAINING_DATA_SIZE_MODE = False
LIMIT_TRAINING_DATA_SIZE_PERCENTAGE = 0.2

SEED = 42

# Zainstalowanie niezbędnych bibliotek
try:
    import gdown
except ImportError:
    !pip install -q gdown

try:
    import imblearn
except ImportError:
    !pip install -q imbalanced-learn

try:
    import joblib
except ImportError:
    !pip install -q joblib

try:
    import matplotlib.pyplot
except ImportError:
    !pip install -q matplotlib

try:
    import numpy as np
except ImportError:
    !pip install -q numpy

try:
    import pandas as pd
except ImportError:
    !pip install -q pandas

try:
    import seaborn as sns
except ImportError:
    !pip install -q seaborn

try:
    import sklearn
except ImportError:
```

```

!pip install -q scikit-learn

from google.colab import data_table

data_table.enable_dataframe_formatter()

```

Tutaj następujeinicjalizacja niezbędnych zmiennych dla całego przetwarzania związanego z metodami nienadzorowanymi. Powód takiego postępowania, należy szukać w tym, że dzięki temu jest możliwe częstekowe wykonanie danego przetwarzania, które może dotyczyć wyłącznie budowania ostatecznego modelu i też po prostu takie podejście zapewnia nam agregację wszystkich niezbędnych informacji w jednym miejscu.

```

In [ ]: # Generalnie podajemy ścieżki do pliku tymczasowego z wynikami, który został wyg
# oraz do pliku wcześniejszy wygenerowanego, który zawiera dane bardziej reprezentatywne
# Także inicjalizacja pewnych zmiennych związanych z procesem strojenia parametrów

oc_svm_model_name = "oc_svm"
oc_svm_filtered_train_df_model_name = "oc_svm_filtered_train_df"
oc_svm_tune_results_tmp_file_path = f"results-{oc_svm_model_name}.csv"
oc_svm_tune_results_master_file_path = f"Results/results-{oc_svm_model_name}-33"
oc_svm_runtime_model_file_path=f"{oc_svm_model_name}.joblib"
oc_svm_runtime_model_filtered_train_df_file_path=f"{oc_svm_filtered_train_df_mod
oc_svm_pretrained_model_file_path=f"Results/{oc_svm_model_name}.joblib"
oc_svm_params_runtime_tune_results = None
oc_svm_params_full_tune_results = None
oc_svm_runtime_model_filtered_train_df_evaluation_result = None
oc_svm_runtime_model_evaluation_result = None
oc_svm_full_model_evaluation_result = None

isolation_forest_model_name = "isolation_forest"
isolation_forest_filtered_train_df_model_name = "isolation_forest_filtered_train_
isolation_forest_tune_results_tmp_file_path = f"results-{isolation_forest_model_n
isolation_forest_tune_results_master_file_path = f"Results/results-{isolation_fo
isolation_forest_runtime_model_file_path=f"{isolation_forest_model_name}.joblib"
isolation_forest_runtime_model_filtered_train_df_file_path=f"{isolation_forest_f
isolation_forest_pretrained_model_file_path=f"Results/{isolation_forest_model_na
isolation_forest_params_runtime_tune_results = None
isolation_forest_params_full_tune_results = None
isolation_forest_runtime_model_evaluation_result = None
isolation_forest_runtime_model_filtered_train_df_evaluation_result = None
isolation_forest_full_model_evaluation_result = None

lof_model_name = "lof"
lof_filtered_train_df_model_name = "lof_filtered_train_df"
lof_tune_results_tmp_file_path = f"results-{lof_model_name}.csv"
lof_tune_results_master_file_path = f"Results/results-{lof_model_name}.csv"
lof_runtime_model_file_path=f"{lof_model_name}.joblib"
lof_runtime_model_filtered_train_df_file_path=f"{lof_filtered_train_df_model_name
lof_pretrained_model_file_path=f"Results/{lof_model_name}.joblib"
lof_params_runtime_tune_results = None
lof_params_full_tune_results = None
lof_runtime_model_evaluation_result = None
lof_runtime_model_filtered_train_df_evaluation_result = None
lof_full_model_evaluation_result = None

```

## Opis danych oraz ich wczytanie

Wybrany zbiór zawiera zanonimizowane informacje o transakcjach dokonanych za pomocą kart kredytowych przez Europejczyków we wrześniu 2013 roku przez okres dwóch dni.

Transakcje są opisane przy pomocy następujących cech:

- Wektor cech  $[V1, \dots, V28]$  - cechy transakcji po transformacji PCA,
- $Time$  - upływ czasu w sekundach pomiędzy pierwszą a określona transakcją (bez transformacji PCA),
- $Amount$  - kwota transakcji (bez transformacji PCA),
- $Class$  - klasa transakcji, wartość "1" - transakcja fałszywa (oszustwo), wartość "0" - transakcja uczciwa.

Ze względu na to, że metody nienadzorowane przy predykcji obserwacji odstającej nie zwracają etykiety klasy, która odpowiada temu rodzajowi obiektów, tylko wartość "-1", zainicjalizowano w tym miejscu stałą `UNSUPERVISED_METHODS_FRAUD_CLASS_LABEL`, która jest przydatna do mapowania na docelowe etykiety klas przy procesie predykcji.

```
In [ ]: from enum import Enum

UNSUPERVISED_METHODS_FRAUD_CLASS_LABEL = -1

class Transaction(Enum):
    VALID_TRANSACTION = (0, "Transakcja regularna")
    FRAUD = (1, "Oszustwo")

    def __init__(self, value, label):
        self._value_ = value
        self.label = label

    @classmethod
    def to_string(cls, val):
        """
        Zwraca postać tekstową etykiety.
        """
        for member in cls:
            if member.value == val:
                return member.label
        return str(val)

# Tworzymy mapę: {0: "Transakcja regularna", 1: "Oszustwo"}, która ułatwi nam in
class_labels_map = {t.value: t.label for t in Transaction}
```

Z powodów praktycznych, postanowiliśmy udostępnić dany zbiór danych na własnym dysku Google Drive i stamtąd go pobierać. Korzystając z Kaggle API, musielibyśmy zadbać o kwestie uwierzytelnienia, co uznaliśmy za ewentualne źródło problemów.

Zadbaliśmy także o to, żeby plik pobierać na środowisko uruchomieniowe jeden raz w trakcie jego działania.

Podobnie podeszliśmy do przechowywania wyników naszych badań oraz docelowych modeli. Są one przetrzymywane w katalogu Results i kiedy jest taka potrzeba, to następuje załadowanie tego rodzaju danych z tego katalogu.

```
In [ ]: import os

# ID pliku z Google Drive
file_id = "14HSXm9CDBA0DxLfSxGCgXQxUXyBxvfno"
output_path = "creditcard.csv"

# Pobierz tylko, jeśli pliku jeszcze nie ma
if not os.path.exists(output_path):
    url = f"https://drive.google.com/uc?id={file_id}"
    gdown.download(url, output_path, quiet=False)
else:
    print("Plik ze zbiorem danych już istnieje lokalnie.")

if not os.path.exists('Results'):
    !gdown --id 1aNwXGm176GJgdsZ5s5urxgMgPofyHs9 --output Results --folder
else:
    print("Folder Results już istnieje. Nie pobieram")

# Wczytaj do DataFrame
df = pd.read_csv(output_path)

df.head(3)
```

```
Plik ze zbiorem danych już istnieje lokalnie.
Folder Results już istnieje. Nie pobieram
Warning: Total number of columns (31) exceeds max_columns (20). Falling back to pandas display.
```

```
Out[ ]:   Time      V1      V2      V3      V4      V5      V6      V7
0    0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599  0.098
1    0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803  0.081
2    1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461  0.247
```

3 rows × 31 columns

Poniżej przedstawiamy krótką charakterystykę danych. Warto zauważyć, że zbiór jest wysoce niebalansowany.

```
In [ ]: print("Liczba atrybutów:", df.shape[1] - 1)
print("Całkowita liczba transakcji:", df.shape[0], end='\n\n')
print("Odsetek oszustw i transakcji regularnych w całym zbiorze:\n\n',
```

```
df['Class'].value_counts(normalize=True).rename(index=class_labels_map), e  
print("Całkowita liczba oszustw i transakcji regularnych:\n\n", df['Class'].valu
```

Liczba atrybutów: 30  
Całkowita liczba transakcji: 284807

Odsetek oszustw i transakcji regularnych w całym zbiorze:

```
Class  
Transakcja regularna    0.998273  
Oszustwo                0.001727  
Name: proportion, dtype: float64
```

Całkowita liczba oszustw i transakcji regularnych:

```
Class  
Transakcja regularna    284315  
Oszustwo                  492  
Name: count, dtype: int64
```

## Przygotowanie danych

Zgodnie z tym co planowaliśmy, aby zwiększyć prawdopodobieństwo uzyskania jak najlepszych rezultatów, przeprowadzimy należyty preprocessing zbioru wejściowego.

Będzie on składał się z następujących etapów:

- sprawdzenia, czy mamy w zbiorze jakiekolwiek braki danych czy duplikaty,
- skalowania cech *Time* oraz *Amount* (w wyniku zastosowania transformacji PCA, wektor cech  $[V1, \dots, V28]$  powinien być uprzednio odpowiednio wyskalowany),
- pomimo zastosowania już transformacji PCA na części parametrów, zbadamy jeszcze koreacje pomiędzy nimi, w celu potencjalnej redukcji wymiarowości.

Poza tym, jeśli użytkownik tego notatnika postanowi działać z ustawioną flagą *QUICK\_MODE*, nastąpi ograniczenie ilościowe wejściowego zbioru danych do ustalonego procentu całości, którego wartość możemy podać w sekcji **Przygotowanie oraz konfiguracja środowiska uruchomieniowego**. Przy tej operacji, zadbane o zachowanie proporcji klas.

## Usuwanie duplikatów

Zbiór posiada następującą liczbę duplikatów:

```
In [ ]: duplicates = df.duplicated(keep='first')  
duplicates_count = duplicates.sum()  
print(duplicates_count)
```

1081

Oczywiście, należy je usunąć ze zbioru, co czyni poniższy kod:

```
In [ ]: df.drop_duplicates(inplace=True)  
print("Usunięto duplikaty. Pozostało transakcji w zbiorze:", df.shape[0])
```

Usunięto duplikaty. Pozostało transakcji w zbiorze: 283726

## Sprawdzanie brakujących danych (NaN / null)

Poniżej następujące weryfikacji, czy mamy w zbiorze egzemplarze, których jakieś atrybuty przyjmują wartości puste bądź nieokreślone:

```
In [ ]: missing_data = df.isnull().values.any()
print(f"Czy zbiór zawiera brakujące dane? Wynik walidacji - {missing_data}")
```

Czy zbiór zawiera brakujące dane? Wynik walidacji - False

## Skalowanie atrybutów *Time* oraz *Amount*

Większość atrybutów została już wcześniej przeskalowana w wyniku transformacji PCA. Pozostało wykonać to samo dla kolumn:

- *Time*
- *Amount*

W tym celu wykorzystamy [RobustScaler](#) z biblioteki scikit-learn, który jest odporny na wartości odstające.

```
In [ ]: from sklearn.preprocessing import RobustScaler

if set(['Amount', 'Time']).issubset(df.columns):
    scaler = RobustScaler()
    amount_scaled = scaler.fit_transform(df['Amount'].values.reshape(-1, 1))
    time_scaled = scaler.fit_transform(df['Time'].values.reshape(-1, 1))
    df.drop(['Time', 'Amount'], axis=1, inplace=True)

    df.insert(1, 'time_scaled', time_scaled)
    df.insert(0, 'amount_scaled', amount_scaled)

print("Kolumny Amount i Time zostały przeskalowane:")
df.head(3)
```

Kolumny Amount i Time zostały przeskalowane:

Warning: Total number of columns (31) exceeds max\_columns (20). Falling back to pandas display.

```
Out[ ]:   amount_scaled      V1  time_scaled      V2      V3      V4      V5
0       1.774718 -1.359807 -0.995290 -0.072781  2.536347  1.378155 -0.338321  0.4
1      -0.268530  1.191857 -0.995290  0.266151  0.166480  0.448154  0.060018 -0.0
2       4.959811 -1.358354 -0.995279 -1.340163  1.773209  0.379780 -0.503198  1.8
```

3 rows × 31 columns

## Redukcja wymiarowości

Według opisu zbioru danych, dane przeszły transformację PCA, która jest techniką redukcji wymiarowości, co wskazuje na brak konieczności dalszych redukcji. Sprawdzenie korelacji wszystkich par kolumn potwierdza nasze przypuszczenia.

```
In [ ]: correlation_matrix = df.corr().abs()
upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))
highest_correlation = upper_triangle.max().max()

print(f"Najwyższa korelacja między dwoma dowolnymi parami atrybutów: {highest_co
```

Najwyższa korelacja między dwoma dowolnymi parami atrybutów: 0.5334280139266703

## Ograniczenie zbioru danych w trybie **QUICK\_MODE**

Tryb **QUICK\_MODE** należy uruchomić w przypadku, gdy zależy nam na szybkiej demonstracji całego rozwiązania.

Włączenie tego trybu zmniejsza też liczbę foldów otrzymanych w procesie k-krotnej walidacji krzyżowej. W celu zobaczenia, jak się dokładnie zmienia dana wartość, należy wrócić do rozdziału **Przygotowanie oraz konfiguracja środowiska uruchomieniowego**.

```
In [ ]: # Do losowania wskazanego procentu danych, z zachowaniem proporcji klas
def stratified_sample(df, frac=0.2, seed=SEED):
    sampled_parts = []
    for label in df["Class"].unique():
        part = df[df["Class"] == label].sample(frac=frac, random_state=seed)
        sampled_parts.append(part)
    return pd.concat(sampled_parts).sample(frac=1, random_state=seed).reset_index()

# Mniejszy zbiór danych do szybszego testowania z zachowaniem proporcji klas
if QUICK_MODE:
    full_df = df.copy()
    df = stratified_sample(df, frac=QUICK_MODE_DATA_PERCENTAGE)
```

## Dostępne metody samplingu

Zgodnie z ustalenia w dokumentacji wstępnej, udostępniamy możliwość wykorzystania konkretnych implementacji algorytmów samplingu. Będą to algorytmy dostarczane przez bibliotekę *Imbalanced-learn*, a dokładnie będą to poniższe metody:

- dla undersamplingu - [imblearn.under\\_sampling.RandomUnderSampler](#),
- dla oversamplingu - [imblearn.over\\_sampling.SMOTE](#).

W tym miejscu dostarczamy metodę do komfortowego reużywania przy pobieraniu żądanego samplera.

```
In [ ]: from enum import Enum
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
```

```

# Enum do wyboru metody samplingu
class SamplingMethod(Enum):
    NONE = 0
    UNDER = 1
    OVER = 2

    @staticmethod
    def from_string(string: str):
        return SamplingMethod[string.split(".")[-1]]

    def to_string(self):
        if self == SamplingMethod.NONE:
            return "Brak próbkowania"
        elif self == SamplingMethod.UNDER:
            return "Undersampling"
        elif self == SamplingMethod.OVER:
            return "Oversampling"

    def get_sampler(sampling_method: SamplingMethod, sampling_strategy: float = 1.0,
                   if sampling_method == SamplingMethod.UNDER:
                       sampler = RandomUnderSampler(sampling_strategy=sampling_strategy, random_
                   elif sampling_method == SamplingMethod.OVER:
                       sampler = SMOTE(sampling_strategy=sampling_strategy, random_state=random_
                   else:
                       sampler = None

    return sampler

```

## Metoda przygotowująca podziały zbioru danych

W celu łatwiejszego definiowania kolejnych przypadków dotyczących podziału danych na zbiory treningowe, walidacyjne czy testowe, postanowiliśmy udostępnić poniższą metodę. W jej parametrach możemy przekazać wszelkie szczegóły określające jak dany podział ma zostać zrealizowany. Dotyczy to m.in.:

- użycia k - krotnej walidacji krzyżowej z ustaloną wartością parametru k,
- jaki procent danych wejściowych, ma stanowić zbiór trenujący,
- czy i jakiej metody samplingu należy użyć przy realizacji danego podziału danych, z możliwością zadeklarowania stosunku ilościowego obiektów jednej klasy do drugiej.

```

In [ ]: import pandas as pd
from sklearn.model_selection import StratifiedKFold, train_test_split

# Klasa reprezentująca podział danych
class DataSplit:
    def __init__(self, x_train, x_test, y_train, y_test):
        self.x_train = x_train
        self.x_test = x_test
        self.y_train = y_train
        self.y_test = y_test

    # Funkcja przygotowująca dane
    def prepare_train_test_data_splits(
        data: pd.DataFrame,
        use_cv: bool = True,

```

```

    k: int = 5,
    test_data_size: float = 0.2,
    sampling_method: SamplingMethod = SamplingMethod.NONE,
    sampling_strategy: float = 1.0,
    random_seed: int = SEED
):
    """
    Przygotowuje dane do trenowania modeli z opcją k-krotnej walidacji i samplingu.

    Parametry:
    - data: DataFrame z danymi wejściowymi, gdzie kolumna "Class" jest etykietą
    - use_cv: czy używać k-krotnej walidacji (domyślnie True)
    - k: liczba foldów (warstw, na które został podzielony wejściowy zbiór danych)
    - test_data_size: rozmiar zbioru testowego (domyślnie 0.2, czyli 20% danych)
    - sampling_method: metoda samplingu (domyślnie SamplingMethod.NONE)
    - sampling_strategy: proporcja liczby egzemplarzy klasy mniejszościowej do k
    - random_seed: ziarno dla generatora liczb losowych (domyślnie 42)

    Zwraca:
    - Listę DataSplit: jeden element przy use_cv=False, k elementów przy use_cv=True
    """

    x = data.drop(columns=["Class"])
    y = data["Class"]

    sampler = get_sampler(sampling_method, sampling_strategy, random_seed)

    data_splits = []
    if use_cv:
        skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=random_seed)

        for train_idx, test_idx in skf.split(x, y):
            x_train, x_test = x.iloc[train_idx], x.iloc[test_idx]
            y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

            if sampler:
                x_train, y_train = sampler.fit_resample(x_train, y_train)

            data_splits.append(DataSplit(x_train, x_test, y_train, y_test))
    else:
        x_train, x_test, y_train, y_test = train_test_split(
            x, y, test_size=test_data_size, stratify=y, random_state=random_seed
        )
        if sampler:
            x_train, y_train = sampler.fit_resample(x_train, y_train)

        data_splits.append(DataSplit(x_train, x_test, y_train, y_test))

    return data_splits

```

## Metoda do wizualizacji rozkładu klas w danych zbiorach

Poniższa funkcja służy do wizualizacji rozkładu klas dla przekazanych podziałów danych.

In [ ]: `import matplotlib.pyplot as plt`

```

def plot_class_distribution(data_splits, plot_title="Rozkład klas"):
    """
    Rysuje wykresy słupkowe pokazujące rozkład klas w zbiorach danych.

    Parametry:
    - data_splits: obiekt DataSplit albo lista obiektów DataSplit
    - plot_title: tytuł wykresu (domyślnie "Rozkład klas")
    """

    if not isinstance(data_splits, list):
        data_splits = [data_splits]

    data_splits_count = len(data_splits)
    for i, data_split in enumerate(data_splits, 1):
        train_examples_counts = data_split.y_train.value_counts().sort_index()
        test_examples_counts = data_split.y_test.value_counts().sort_index()
        classes = class_labels_map.keys()

    bar_width = 0.35
    x = np.arange(len(classes))

    fig, ax = plt.subplots(figsize=(6, 4))
    bars1 = ax.bar(x - bar_width/2, [train_examples_counts.get(c, 0) for c in classes], width=bar_width, label='Dane treningowe', color='skyblue')
    bars2 = ax.bar(x + bar_width/2, [test_examples_counts.get(c, 0) for c in classes], width=bar_width, label='Dane testowe', color='salmon')

    # Dodanie wartości nad słupkami
    for bar in bars1 + bars2:
        height = bar.get_height()
        ax.annotate(f'{height}', xy=(bar.get_x() + bar.get_width() / 2, height), xytext=(0, 3), textcoords="offset points", ha='center', va='bottom')

    train_columns_height = [train_examples_counts.get(c, 0) for c in classes]
    test_columns_height = [test_examples_counts.get(c, 0) for c in classes]

    max_height = max(train_columns_height + test_columns_height)
    # Ustawienie marginesu na osi Y
    y_margin = max_height * 0.35

    ax.set_title(f"{plot_title} - podział danych numer {i}" if data_splits_count > 1 else plot_title)
    ax.set_xlabel("Klasa")
    ax.set_ylabel("Liczba przykładów")
    ax.set_xticks(x)
    ax.set_xticklabels([class_labels_map.get(c, str(c)) for c in classes])
    ax.set_ylim(0, max_height + y_margin)
    ax.legend()
    ax.grid(axis='y', linestyle='--', alpha=0.7)

    plt.tight_layout()
    plt.show()

```

## Przygotowanie konkretnych podziałów zbioru danych

W tym miejscu następuje podział na dane treningowe oraz testowe. W zależności od trybu, w jakim pracujemy mogą nastąpić jeszcze dwie operacje:

- zadbaliśmy o to, że jak jest uruchomiony tryb *QUICK\_MODE*, to żeby ewaluacja danego modelu odbywała się na pełnym zbiorze testowym, a nie ograniczonym.
- ze względu na to, że dla metod nienadzorowanych czasem lepiej jest budować na mniejszym zbiorze danych, żeby nie wprowadzać niepotrzebnego szумu, mamy możliwość ograniczenia zbioru treningowego do ustalonego procenta danych, zachowując oczywiście proporcje klas. Jest to tryb bardziej deweloperski, który posłużył nam do budowania lepszych docelowych modeli.

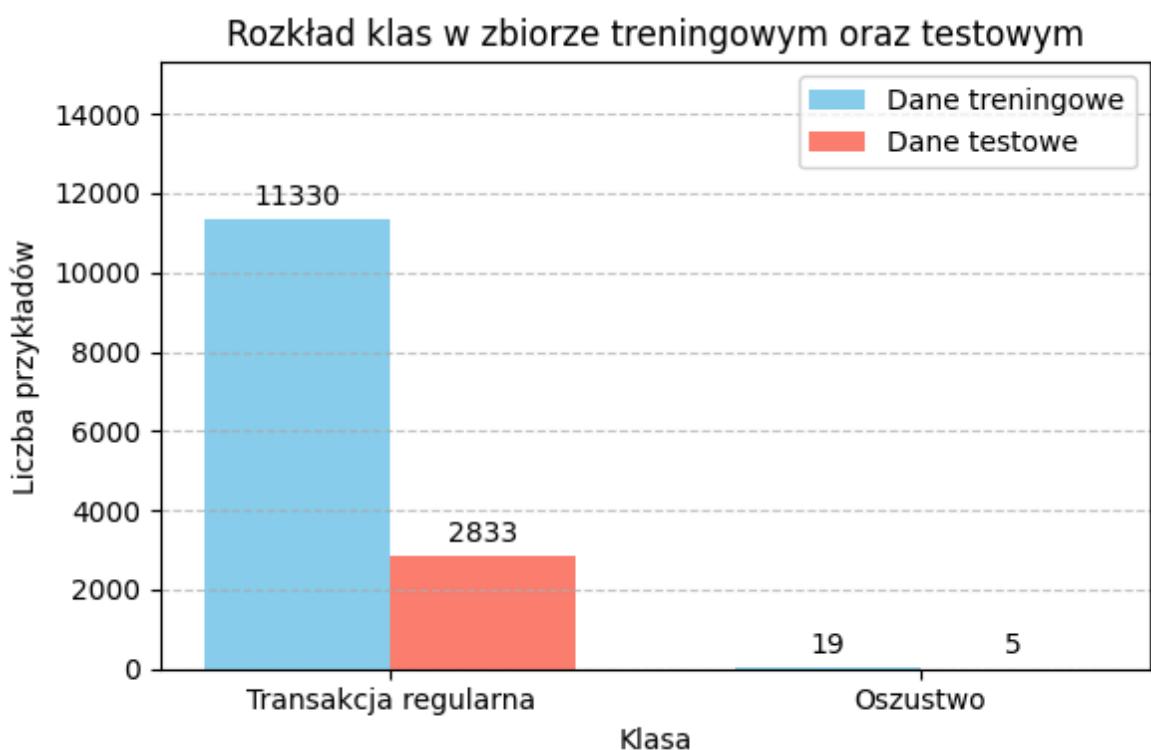
Poniżej, wizualizacja w postaci wykresu rozkładu klas, który pokazuje zarówno z jak niezbalansowanym zbiorem danych pracujemy oraz stosunek podziału na dane treningowe i testowe.

```
In [ ]: # Podział danych na zbiór treningowy oraz zbiór testowy
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df["Class"], random_state=42)

if QUICK_MODE:
    _, full_test_df = train_test_split(full_df, test_size=0.2, stratify=full_df["Class"])

# Ograniczenie rozmiaru zbioru treningowego jeśli włączona jest flaga LIMIT_TRAINING_DATA_SIZE_MODE
if LIMIT_TRAINING_DATA_SIZE_MODE:
    train_df = stratified_sample(df, frac=LIMIT_TRAINING_DATA_SIZE_PERCENTAGE)

train_test_data_split = DataSplit(train_df.drop(columns=["Class"]), test_df.drop(columns=["Class"]),
                                   train_df["Class"], test_df["Class"])
plot_class_distribution(train_test_data_split, "Rozkład klas w zbiorze treningowym")
```



Przy badaniach dotyczących metod nienadzorowanych, zauważliśmy, że czasem bardziej jakościowy model możemy dostać przy treningu na danych zawierających tylko obiekty

klasy większościowej. Z tego powodu, w tym miejscu dokonujemy przygotowania zbioru danych pod właśnie taki przypadek.

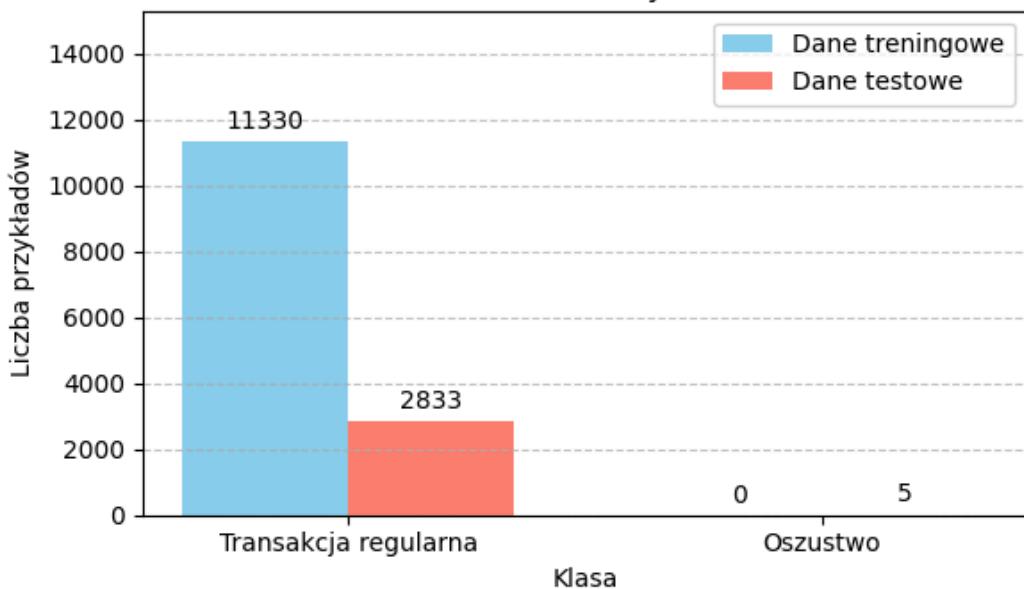
```
In [ ]: def filter_df(df, class_label=Transaction.VALID_TRANSACTION.value):
    """
    Filtruje ramkę danych, pozostawiając tylko przykłady ustalonej klasy.

    Parametry:
        df: ramka danych zawierająca cechy i etykiety
        class_label: etykieta oznaczająca klasę, której egzemplarze mają pozostać

    Zwraca:
        Przefiltrowana ramka z samymi przykładami żądanej klasy
    """
    filtered_df = df[df["Class"] == class_label].copy()
    return filtered_df

# Odfiltrowane dane treningowe dla niektórych metod nienadzorowanych:
# chcemy trenować tylko na egzemplarzach klasy większościowej
filtered_train_df = filter_df(train_df)
filtered_train_test_data_split = DataSplit(filtered_train_df.drop(columns=["Class"]),
                                             filtered_train_df["Class"], test_df["Class"])
plot_class_distribution(filtered_train_test_data_split, "Rozkład klas w zbiorze
```

Rozkład klas w zbiorze treningowym (bez przykładów klasy mniejszościowej) oraz testowym



W procesie strojenia parametrów modeli, gdzie używamy k - krotnej walidacji krzyżowej, chcemy zbadać również wpływ metod samplingu. Z tego powodu, już w tym miejscu zadaliśmy o odpowiedni podział danych dla różnych przypadków testowych odnoszących się właśnie do dobranej metody samplingu. Owe podziały danych mogą być reużywane pomiędzy różnymi modelami.

```
In [ ]: # Przygotowanie podziałów danych na różne przypadki z uwzględnieniem metod samplingu
standard_data_splits = prepare_train_test_data_splits(data=train_df, use_cv=True,
                                                       sampling_method=SamplingMethod.STANDARD)
undersampling_50_data_splits = prepare_train_test_data_splits(data=train_df, use_cv=True,
                                                               sampling_method=SamplingMethod.UNDERAMPLING_50)
undersampling_data_splits = prepare_train_test_data_splits(data=train_df, use_cv=True)
```

```
sampling_method=Sam  
oversampling_50_data_splits = prepare_train_test_data_splits(data=train_df, use_  
sampling_method=Sam  
oversampling_data_splits = prepare_train_test_data_splits(data=train_df, use_cv=  
sampling_method=Sam
```

## Miary jakości

Do oceny jakości modelów wybrano następujące miary jakości:

- **odzysk (recall)** - współczynnik pokazujący, jaki odsetek oszustw zostało wykryty przez model wśród wszystkich transakcji-oszustw,
- **precyzja (precision)** - jaki odsetek wykrytych transakcji-oszustw było faktycznie oszustwem - miara pomocnicza,
- **wskaźnik F1** - średnia harmoniczna precyzji i odzysku - pokazuje balans pomiędzy tymi dwoma wskaźnikami - miara pomocnicza,
- **average precision** - średnia precyzja przy różnych progach klasyfikacji, również oparta na obszarze pod krzywą Precision - Recall - główna miara jakości,
- **AUC** - pole pod krzywą Precision - Recall - główna miara jakości, lecz z tego powodu, że jest wyznaczana za pomocą metody trapezów, która potrafi zachowywać się w sposób zbyt optymistyczny, jeśli chodzi o przybliżaną wartość, to należy bardziej swój wzrok kierować na miarę **average precision**.

```
In [ ]: import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
from sklearn.metrics import (  
    precision_score,  
    recall_score,  
    f1_score,  
    precision_recall_curve,  
    average_precision_score,  
    auc  
)  
  
def map_y_pred_for_unsupervised_methods(y_pred):  
    return np.where(y_pred == UNSUPERVISED_METHODS_FRAUD_CLASS_LABEL, Transaction.  
  
def evaluate_model(model, x_test, y_test, predict_positive_class_label=Transacti  
    """  
        Ocena jakości modelu: precision, recall, F1 score, krzywa PR, average precis  
        Wyświetla tabelę z wynikami i wykres krzywej PR.  
  
        Parametry:  
        - model: wytrenowany model  
        - x_test, y_test: dane testowe  
        - predict_positive_class_label - określa etykietę klasy pozytywnej, która bę  
  
        Zwraca:  
        - Słownik metrykami modelu  
    """  
  
    # Predykcja  
    y_pred = model.predict(x_test)
```

```

if hasattr(model, "predict_proba"):
    fraud_idx = list(model.classes_).index(predict_positive_class_label)
    y_scores = model.predict_proba(x_test)[:, fraud_idx]
else:
    y_scores = model.decision_function(x_test)

if predict_positive_class_label == UNSUPERVISED_METHODS_FRAUD_CLASS_LABEL:
    y_pred = map_y_pred_for_unsupervised_methods(y_pred)

# Podstawowe metryki
recall = recall_score(y_test, y_pred, pos_label=Transaction.FRAUD.value)
precision = precision_score(y_test, y_pred, pos_label=Transaction.FRAUD.value)
f1 = f1_score(y_test, y_pred, pos_label=Transaction.FRAUD.value)

# Wyznaczenie krzywej PR i metryk z nią związanych
precision_vals, recall_vals, _ = precision_recall_curve(y_test, y_scores, pos_label=Transaction.FRAUD.value)
avg_precision = average_precision_score(y_test, y_scores, pos_label=Transaction.FRAUD.value)
interpolated_auc = auc(recall_vals, precision_vals)

# Tabela metryk
results_df = pd.DataFrame({
    "Precision": [precision],
    "Recall": [recall],
    "F1-Score": [f1],
    "Avg. Precision": [avg_precision],
    "AUC PR (interpolowane)": [interpolated_auc]
})

print("\nOcena jakości modelu:")
print(results_df.head(1))

return dict(precision=precision,
            recall=recall,
            f1=f1,
            avg_precision=avg_precision,
            interpolated_auc=interpolated_auc,
            precision_vals=precision_vals,
            recall_vals=recall_vals)

```

## Badania wybranych metod do detekcji anomalii

Dla wybranych algorytmów ustanowiono jednolity proces badawczy, który opisano poniżej:

1. Wstępnie przetworzone dane losowo podzielono na zbiór testowy (20% całości) i treningowy (80% całości).
2. Zbiór treningowy podzielono na 5 podzbiorów - do wykorzystania dla walidacji krzyżowej.
3. Dla każdego testowanego algorytmu wyszukano zestaw najlepszych parametrów.
4. Wybrano zestaw parametrów wraz z listą wartości do przetestowania.
5. Wybrano metody i strategie samplingu do przetestowania (strategia - docelowy stosunek rekordów klasy mniejszościowej do większościowej).

6. Dla każdej wartości parametru, wypróbowano każdą z ustalonych wcześniej metod próbkowania.
7. Dla pojedynczego testu, wytrenowano model 5 razy, za każdym razem wykorzystując 4 części podzbioru walidacji krzyżowej do treningu, a pozostała 1 część jako zbiór testowy. Zbiory treningowe przekształcono wcześniej z użyciem undersamplingu lub oversamplingu (lub brak przekształcenia).
8. Jako ostateczną wartość poszczególnych miar jakości dla pojedynczego testu obliczano średnią arytmetyczną uzyskanych wartości z 5 modeli wytrenowanych w pojedynczym procesie walidacji krzyżowej.
9. Na podstawie wyników, wybrano najlepszy zestaw parametrów.
10. Wytrenowano ostateczny model i wyliczono jego miary jakości, jako zbiór testowy wykorzystano oddzielony wcześniej zestaw danych w rozmiarze 20% zbioru początkowego. Zbiór testowy nie był wykorzystywany w procesie poszukiwania parametrów.

```
In [ ]: from dataclasses import dataclass, field
from typing import List
from datetime import datetime

@dataclass
class ParameterToTest:
    name: str
    values: List
    required_params: dict = field(default_factory = lambda: {})

@dataclass
class SamplingMethodToTest:
    method: SamplingMethod
    sampling_strategy: float
    data_splits: List[DataSplit]

def params_tune(params_to_test: List[ParameterToTest], sampling_methods_to_test: List[SamplingMethodToTest], build_model_function, model_name, seed=SEED, predict_positive_class_label=POSITIVE_CLASS_LABEL, special_stop_case=None, model_specific_params={}):
    all_results = []

    for param in params_to_test:
        for sampling_method in sampling_methods_to_test:
            for param_value in param.values:
                # TODO: Add more parameters if needed
                if special_stop_case and special_stop_case(param.name, sampling_method):
                    continue

                precision_list, recall_list, f1_list, avg_precision_list, interp_list = [], [], [], [], []
                for fold_id, split in enumerate(sampling_method.data_splits, 1):
                    params = model_specific_params.copy()
                    if param.required_params:
                        params.update(param.required_params)
                    params[param.name] = param_value
                    model = build_model_function(**params)
                    print(f"({datetime.now().strftime('%H:%M:%S')}) Fold {fold_id} {model_name} {param.name} {param_value} {sampling_method.method} {split}")
                    if predict_positive_class_label == UNSUPERVISED_METHODS_FRAUD:
                        # ...
                    else:
                        # ...
```
```

```

        model.fit(split.x_train)
    else:
        model.fit(split.x_train, split.y_train)

    metrics_df = evaluate_model(
        model,
        split.x_test,
        split.y_test,
        predict_positive_class_label=predict_positive_class_label
    )
    precision_list.append(metrics_df["precision"])
    recall_list.append(metrics_df["recall"])
    f1_list.append(metrics_df["f1"])
    avg_precision_list.append(metrics_df["avg_precision"])
    interpolated_auc_list.append(metrics_df["interpolated_auc"])

    # Uśrednianie
    all_results.append(
        dict(sampling=sampling_method.method.name,
             sampling_strategy=sampling_method.sampling_strategy,
             param=param.name,
             value=param_value,
             mean_precision=np.mean(precision_list),
             mean_recall=np.mean(recall_list),
             mean_f1=np.mean(f1_list),
             mean_avg_precision=np.mean(avg_precision_list),
             mean_interpolated_auc=np.mean(interpolated_auc_list),
             # TODO: Opisac dlaczego ddof=1
             # std_avg_precision=np.std(avg_precision_list, ddof=1),
             )
    )

results_df = pd.DataFrame(all_results)
results_df.to_csv(f"results-{model_name}.csv", sep='\t')

return results_df

```

```

In [ ]: def train_model(train_df: pd.DataFrame, test_df: pd.DataFrame, best_params, build_model_function):
            sampling_method: SamplingMethod, sampling_strategy: float = 1.0,
            unsupervised_method=False, seed=SEED):
    x_train = train_df.drop(columns=["Class"])
    y_train = train_df["Class"]

    x_test = test_df.drop(columns=["Class"])
    y_test = test_df["Class"]

    sampler = get_sampler(sampling_method, sampling_strategy, seed)
    if sampler:
        x_train, y_train = sampler.fit_resample(x_train, y_train)

    model = build_model_function(**best_params)

    if unsupervised_method:
        model.fit(x_train)
    else:
        model.fit(x_train, y_train)

    joblib.dump(model, f"{model_name}.joblib")

```

```
    return model
```

```
In [ ]: import joblib
import os

def train_and_evaluate_best_model(train_df: pd.DataFrame, test_df: pd.DataFrame,
                                  model_name, runtime_model_file_path, sampling_
                                  sampling_strategy: float = 1.0, unsupervised_m
if (QUICK_MODE and os.path.exists(runtime_model_file_path)):
    model = joblib.load(runtime_model_file_path)
else:
    model = train_model(train_df, test_df, best_params, build_model_function, mo
                          sampling_method, sampling_strategy, unsupervised_method)

x_test = test_df.drop(columns=["Class"])
y_test = test_df["Class"]

predict_positive_class_label = UNSUPERVISED_METHODS_FRAUD_CLASS_LABEL if unsup

return evaluate_model(model, x_test, y_test, predict_positive_class_label=pred
```

Poniżej metody pomocnicze do generowania wykresów, odzwierciedlających nasze wyniki.

```
In [ ]: import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

def add_values_labels(plt):
    ax = plt.gca()
    for container in ax.containers:
        for bar in container:
            height = bar.get_height()
            x = bar.get_x()
            width = bar.get_width()
            ax.text(
                x + width * 0.05, # Lekko na lewo/prawo
                height + 0.002, # delikatnie nad stupkiem
                f"{height:.3f}",
                ha='left',         # wyrównanie do lewej
                va='bottom',
                fontsize=8,
                rotation=0
            )

def plot_metric_by_param(results_df, param_name: str, metric_name: str, hue: str
    if results_df is None:
        print(f"Brak danych z wynikami")
        return

    # Filtrowanie tylko dla danego parametru
    results_df_filtered = results_df[results_df["param"] == param_name].copy()

    if results_df_filtered.empty:
        print(f"Brak danych dla parametru: {param_name}")
        return

    # Tworzymy wykres
```

```

plt.figure(figsize=(10, 6))
sns.barplot(
    data=results_df_filtered,
    x="value",
    y=metric_name,
    hue=hue if hue else "value",
    palette="Set2",
    legend=True if hue else False
)

# Na razie wyłączone, bo gorzej jednak to wygląda
# Dodaj wartości nad słupkami
# add_values_labels(plt)

plt.title(title or f"{metric_name} vs {param_name}", fontsize=14)
plt.xlabel(f"Wartości parametru: {param_name}")
plt.ylabel(metric_name)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()
plt.show()

def plot_metric_groups_for_param(results_df, param_name: str, metric_names: list):
    if results_df is None:
        print(f"Brak danych z wynikami")
        return

    # Filtrowanie tylko dla danego parametru
    results_df_filtered = results_df[results_df["param"] == param_name].copy()

    if results_df_filtered.empty:
        print(f"Brak danych dla parametru: {param_name}")
        return

    # Tworzenie "długiego" DataFrame do rysowania
    results_df_melted = results_df_filtered.melt(
        id_vars=["value"],
        value_vars=metric_names,
        var_name="metric",
        value_name="score"
    )

    plt.figure(figsize=(10, 6))
    sns.barplot(
        data=results_df_melted,
        x="value",
        y="score",
        hue="metric",
        palette="Set2",
    )

    # Na razie wyłączone, bo gorzej jednak to wygląda
    # Dodaj wartości nad słupkami
    # add_values_labels(plt)

    plt.title(title or f"Porównanie metryk dla parametru: {param_name}", fontsize=14)
    plt.xlabel(f"Wartości parametru: {param_name}")
    plt.ylabel("Wartość metryki")
    plt.grid(axis='y', linestyle='--', alpha=0.5)
    plt.legend(
        title="Metryka",

```

```

        bbox_to_anchor=(1.02, 1), # x=102% szerokości, y=100% wysokości
        loc='upper left',
        borderaxespad=0
    )
    plt.tight_layout(rect=[0, 0, 0.9, 1]) # Zostaw miejsce po prawej stronie
    plt.show()

def plot_metric_groups_for_sampling(results_df, metric_names: list[str], title:
    if results_df is None:
        print("Brak danych z wynikami")
        return

    # Upewnij się, że sampling i strategy to stringi
    results_df_copy = results_df.copy()
    results_df_copy["sampling"] = results_df_copy["sampling"].astype(str)
    results_df_copy["sampling_strategy"] = results_df_copy["sampling_strategy"].

    # Stwórz kolumnę grupującą
    results_df_copy["sampling_group"] = results_df_copy.apply(
        lambda row: f"{row['sampling']}\nstrategy={row['sampling_strategy']}", a
    )

    # Przekształć dane do formatu długiego
    results_df_melted = results_df_copy.melt(
        id_vars=["sampling_group"],
        value_vars=metric_names,
        var_name="metric",
        value_name="score"
    )

    # Tworzymy wykres
    plt.figure(figsize=(12, 6))
    sns.barplot(
        data=results_df_melted,
        x="sampling_group",
        y="score",
        hue="metric",
        palette="Set2",
    )

    # Na razie wyłączone, bo gorzej jednak to wygląda
    # Dodaj wartości nad słupkami
    # add_values_labels(plt)

    plt.title(title or "Porównanie metryk dla różnych konfiguracji samplingu", f
    plt.xlabel("Konfiguracja: rodzaj metody samplingu + zastosowana strategia")
    plt.ylabel("Wartość metryki")
    plt.grid(axis='y', linestyle='--', alpha=0.4)
    plt.legend(
        title="Metryka",
        bbox_to_anchor=(1.02, 1), # x=102% szerokości, y=100% wysokości
        loc='upper left',
        borderaxespad=0
    )
    plt.tight_layout(rect=[0, 0, 0.9, 1]) # Zostaw miejsce po prawej stronie
    plt.show()

def plot_metric_summary(
    metric_values: dict,
    metric_labels: list[str] = None,

```

```

        title: str = "Podsumowanie metryk jakości modelu"
    ):
    if metric_labels is None:
        metric_labels = ["precision", "recall", "f1", "avg_precision", "interpolat

    # Mapowanie nazw metryk do czytelnych etykiety
    label_names_map = {
        "precision": "Precision",
        "recall": "Recall",
        "f1": "F1-score",
        "avg_precision": "Avg Precision",
        "interpolated_auc": "AUC PR (interpolowany)"
    }

    # Filtrowanie i przygotowanie danych
    selected_metrics = [metric_values[label] for label in metric_labels]
    selected_labels = [label_names_map.get(label, label) for label in metric_labels]

    df_plot = pd.DataFrame({
        "Metryka": selected_labels,
        "Wartość": selected_metrics
    })

    plt.figure(figsize=(10, 6))
    ax = sns.barplot(
        data=df_plot,
        x="Metryka",
        y="Wartość",
        hue="Metryka",
        palette="Set2"
    )

    plt.title(title, fontsize=14)
    plt.ylim(0, 1.05)
    plt.ylabel("Wartość metryki")
    plt.xlabel("Metryka")
    plt.grid(axis='y', linestyle='--', alpha=0.4)
    plt.tight_layout()
    plt.show()

def plot_pr_curve(recall_vals, precision_vals, avg_precision,
                  plot_title="Krzywa Precision-Recall"):
    # Wykres krzywej PR
    plt.figure(figsize=(6, 5))
    plt.plot(recall_vals, precision_vals, label=f"AP={avg_precision:.4f}", color='red')
    plt.fill_between(recall_vals, precision_vals, alpha=0.1, color='teal')
    plt.xlabel("Recall")
    plt.ylabel("Precision")
    plt.title(plot_title)
    plt.grid(True, linestyle='--', alpha=0.7)
    plt.legend()
    plt.tight_layout()
    plt.show()

```

## Metody nadzorowane

### Las losowy

```
In [ ]: from sklearn.ensemble import RandomForestClassifier

if QUICK_MODE:
    random_forest_params_to_test = [
        ParameterToTest(name="n_estimators", values=[50, 300]),
        ParameterToTest(name="max_features", values=["sqrt", None]),
        ParameterToTest(name="class_weight", values=["balanced", "balanced_subsample"])
]

random_forest_sampling_method_to_test = [
    SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data_s
    SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data_
    SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, data_s
]
else:
    random_forest_params_to_test = [
        ParameterToTest(name="n_estimators", values=[50, 100, 200, 300]),
        ParameterToTest(name="max_features", values=["sqrt", "log2", 0.3, 0.5, N
        ParameterToTest(name="class_weight", values=["balanced", "balanced_subsa
    ]

random_forest_sampling_method_to_test = [
    SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, da
    SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=0.5,
        data_splits=undersampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, d
    SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=0.5,
        data_splits=oversampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, da
]
]

def build_random_forest_model(**kwargs):
    return RandomForestClassifier(**kwargs)

def random_forest_special_stop_case(param_name: str, sampling_method: SamplingMe
is_class_weight_test = param_name == "class_weight"
is_data_sampled = sampling_method.method != SamplingMethod.NONE

# Jeżeli dane sa samplowane, nie testujemy zadnych wartosci parametru "class
if is_class_weight_test and is_data_sampled:
    print(f"Pomiń, ponieważ sampling method: {sampling_method.method} i para
    return True
else:
    return False

model_specific_params = {
    "random_state": SEED,
    "n_jobs": -1}

results = params_tune(random_forest_params_to_test, sampling_methods_to_test=ran
    special_stop_case=random_forest_special_stop_case, build_m
    model_specific_params=model_specific_params, model_name='r
results
print("Pełne wyniki (dla quick_mode=False)")

random_forest_results = pd.read_csv("Results/results-random_forest.csv", sep="\t"
random_forest_results.replace({np.nan: None}, inplace=True)
random_forest_results
```

(17:27:04) Fold 1. Main param: n\_estimators=50. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 50}.  
Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 1.0       | 0.7    | 0.823529 | 0.9625         | 0.96875                |

(17:27:05) Fold 2. Main param: n\_estimators=50. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 50}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.75      | 0.333333 | 0.461538 | 0.385714       | 0.411162               |

(17:27:05) Fold 1. Main param: n\_estimators=300. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 300}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 1.0       | 0.5    | 0.666667 | 1.0            | 1.0                    |

(17:27:09) Fold 2. Main param: n\_estimators=300. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 300}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.75      | 0.333333 | 0.461538 | 0.397619       | 0.405517               |

(17:27:11) Fold 1. Main param: n\_estimators=50. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 50}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.053476  | 1.0    | 0.101523 | 1.0            | 1.0                    |

(17:27:11) Fold 2. Main param: n\_estimators=50. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 50}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.714286  | 0.555556 | 0.625    | 0.484652       | 0.398616               |

(17:27:12) Fold 1. Main param: n\_estimators=300. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 300}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.050761  | 1.0    | 0.096618 | 1.0            | 1.0                    |

(17:27:12) Fold 2. Main param: n\_estimators=300. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 300}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.714286  | 0.555556 | 0.625    | 0.466348       | 0.417203               |

(17:27:13) Fold 1. Main param: n\_estimators=50. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 50}.

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|--------|----------|------|-----------|-----|--------------------|
| 0 | 1.0       | 0.4    | 0.571429 |      | 0.876243  |     | 0.869709           |

(17:27:15) Fold 2. Main param: n\_estimators=50. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 50}.

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|----------|----------|------|-----------|-----|--------------------|
| 0 | 0.8       | 0.444444 | 0.571429 |      | 0.483598  |     | 0.434932           |

(17:27:16) Fold 1. Main param: n\_estimators=300. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 300}.

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|--------|----------|------|-----------|-----|--------------------|
| 0 | 1.0       | 0.3    | 0.461538 |      | 0.9525    |     | 0.950694           |

(17:27:26) Fold 2. Main param: n\_estimators=300. All params: {'random\_state': 42, 'n\_jobs': -1, 'n\_estimators': 300}.

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|----------|----------|------|-----------|-----|--------------------|
| 0 | 0.75      | 0.333333 | 0.461538 |      | 0.465079  |     | 0.427505           |

(17:27:30) Fold 1. Main param: max\_features=sqrt. All params: {'random\_state': 42, 'n\_jobs': -1, 'max\_features': 'sqrt'}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|--------|----------|------|-----------|-----|--------------------|
| 0 | 1.0       | 0.6    | 0.75     |      | 0.983333  |     | 0.982576           |

(17:27:32) Fold 2. Main param: max\_features=sqrt. All params: {'random\_state': 42, 'n\_jobs': -1, 'max\_features': 'sqrt'}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|----------|----------|------|-----------|-----|--------------------|
| 0 | 0.8       | 0.444444 | 0.571429 |      | 0.38373   |     | 0.347046           |

(17:27:33) Fold 1. Main param: max\_features=None. All params: {'random\_state': 42, 'n\_jobs': -1, 'max\_features': None}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|--------|----------|------|-----------|-----|--------------------|
| 0 | 1.0       | 0.8    | 0.888889 |      | 1.0       |     | 1.0                |

(17:27:37) Fold 2. Main param: max\_features=None. All params: {'random\_state': 42, 'n\_jobs': -1, 'max\_features': None}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|----------|----------|------|-----------|-----|--------------------|
| 0 | 0.6       | 0.333333 | 0.428571 |      | 0.412742  |     | 0.422685           |

(17:27:39) Fold 1. Main param: max\_features=sqrt. All params: {'random\_state': 42, 'n\_jobs': -1, 'max\_features': 'sqrt'}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. | Precision | AUC | PR (interpolowane) |
|---|-----------|--------|----------|------|-----------|-----|--------------------|
| 0 | 0.049751  | 1.0    | 0.094787 |      | 1.0       |     | 1.0                |

(17:27:39) Fold 2. Main param: max\_features=sqrt. All params: {'random\_state': 42, 'n\_jobs': -1, 'max\_features': 'sqrt'}.

```
2, 'n_jobs': -1, 'max_features': 'sqrt'}].  
Sampling method SamplingMethod.UNDER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                             | Precision | Recall   | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|-----------------------------------------------------------------------------------------------------------------------------|-----------|----------|----------|----------------|-----|--------------------|
| 0                                                                                                                           | 0.714286  | 0.555556 | 0.625    | 0.466093       |     | 0.417034           |
| (17:27:40) Fold 1. Main param: max_features=None. All params: {'random_state': 4<br>2, 'n_jobs': -1, 'max_features': None}. |           |          |          |                |     |                    |

```
Sampling method SamplingMethod.UNDER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                             | Precision | Recall | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|-----------------------------------------------------------------------------------------------------------------------------|-----------|--------|----------|----------------|-----|--------------------|
| 0                                                                                                                           | 0.049505  | 1.0    | 0.09434  | 0.953846       |     | 0.976923           |
| (17:27:40) Fold 2. Main param: max_features=None. All params: {'random_state': 4<br>2, 'n_jobs': -1, 'max_features': None}. |           |        |          |                |     |                    |

```
Sampling method SamplingMethod.UNDER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                               | Precision | Recall   | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|-------------------------------------------------------------------------------------------------------------------------------|-----------|----------|----------|----------------|-----|--------------------|
| 0                                                                                                                             | 0.714286  | 0.555556 | 0.625    | 0.484326       |     | 0.509329           |
| (17:27:40) Fold 1. Main param: max_features=sqrt. All params: {'random_state': 4<br>2, 'n_jobs': -1, 'max_features': 'sqrt'}. |           |          |          |                |     |                    |

```
Sampling method SamplingMethod.OVER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                               | Precision | Recall | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|-------------------------------------------------------------------------------------------------------------------------------|-----------|--------|----------|----------------|-----|--------------------|
| 0                                                                                                                             | 1.0       | 0.3    | 0.461538 | 0.872937       |     | 0.877561           |
| (17:27:45) Fold 2. Main param: max_features=sqrt. All params: {'random_state': 4<br>2, 'n_jobs': -1, 'max_features': 'sqrt'}. |           |        |          |                |     |                    |

```
Sampling method SamplingMethod.OVER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                             | Precision | Recall   | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|-----------------------------------------------------------------------------------------------------------------------------|-----------|----------|----------|----------------|-----|--------------------|
| 0                                                                                                                           | 0.8       | 0.444444 | 0.571429 | 0.483598       |     | 0.418961           |
| (17:27:46) Fold 1. Main param: max_features=None. All params: {'random_state': 4<br>2, 'n_jobs': -1, 'max_features': None}. |           |          |          |                |     |                    |

```
Sampling method SamplingMethod.OVER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                             | Precision | Recall | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|-----------------------------------------------------------------------------------------------------------------------------|-----------|--------|----------|----------------|-----|--------------------|
| 0                                                                                                                           | 0.5       | 0.5    | 0.5      | 0.397856       |     | 0.394087           |
| (17:28:02) Fold 2. Main param: max_features=None. All params: {'random_state': 4<br>2, 'n_jobs': -1, 'max_features': None}. |           |        |          |                |     |                    |

```
Sampling method SamplingMethod.OVER, sampling strategy 1.
```

Ocena jakości modelu:

|                                                                                                                                       | Precision | Recall   | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|---------------------------------------------------------------------------------------------------------------------------------------|-----------|----------|----------|----------------|-----|--------------------|
| 0                                                                                                                                     | 0.6       | 0.333333 | 0.428571 | 0.385626       |     | 0.512341           |
| (17:28:06) Fold 1. Main param: class_weight=balanced. All params: {'random_stat<br>e': 42, 'n_jobs': -1, 'class_weight': 'balanced'}. |           |          |          |                |     |                    |

```
Sampling method SamplingMethod.NONE, sampling strategy 0.
```

Ocena jakości modelu:

|                                                                                                                                       | Precision | Recall | F1-Score | Avg. Precision | AUC | PR (interpolowane) |
|---------------------------------------------------------------------------------------------------------------------------------------|-----------|--------|----------|----------------|-----|--------------------|
| 0                                                                                                                                     | 1.0       | 0.2    | 0.333333 | 1.0            |     | 1.0                |
| (17:28:07) Fold 2. Main param: class_weight=balanced. All params: {'random_stat<br>e': 42, 'n_jobs': -1, 'class_weight': 'balanced'}. |           |        |          |                |     |                    |

```
Sampling method SamplingMethod.NONE, sampling strategy 0.
```

Ocena jakości modelu:

```
Precision    Recall   F1-Score   Avg. Precision   AUC PR (interpolowane)
0  0.666667  0.222222  0.333333      0.45582          0.425019
(17:28:07) Fold 1. Main param: class_weight=balanced_subsample. All params: {'random_state': 42, 'n_jobs': -1, 'class_weight': 'balanced_subsample'}.
Sampling method SamplingMethod.NONE, sampling strategy 0.
```

Ocena jakości modelu:

```
Precision    Recall   F1-Score   Avg. Precision   AUC PR (interpolowane)
0  1.0        0.1     0.181818      1.0            1.0
(17:28:08) Fold 2. Main param: class_weight=balanced_subsample. All params: {'random_state': 42, 'n_jobs': -1, 'class_weight': 'balanced_subsample'}.
Sampling method SamplingMethod.NONE, sampling strategy 0.
```

Ocena jakości modelu:

```
Precision    Recall   F1-Score   Avg. Precision   AUC PR (interpolowane)
0  0.666667  0.222222  0.333333      0.45582          0.429979
(17:28:10) Fold 1. Main param: class_weight=None. All params: {'random_state': 42, 'n_jobs': -1, 'class_weight': None}.
Sampling method SamplingMethod.NONE, sampling strategy 0.
```

Ocena jakości modelu:

```
Precision    Recall   F1-Score   Avg. Precision   AUC PR (interpolowane)
0  1.0        0.6     0.75        0.983333          0.982576
(17:28:12) Fold 2. Main param: class_weight=None. All params: {'random_state': 42, 'n_jobs': -1, 'class_weight': None}.
Sampling method SamplingMethod.NONE, sampling strategy 0.
```

Ocena jakości modelu:

```
Precision    Recall   F1-Score   Avg. Precision   AUC PR (interpolowane)
0  0.8        0.444444  0.571429      0.38373          0.347046
Pomiń, ponieważ sampling method: SamplingMethod.UNDER i param_name: class_weight
Pomiń, ponieważ sampling method: SamplingMethod.UNDER i param_name: class_weight
Pomiń, ponieważ sampling method: SamplingMethod.UNDER i param_name: class_weight
Pomiń, ponieważ sampling method: SamplingMethod.OVER i param_name: class_weight
Pomiń, ponieważ sampling method: SamplingMethod.OVER i param_name: class_weight
Pomiń, ponieważ sampling method: SamplingMethod.OVER i param_name: class_weight
Pełne wyniki (dla quick_mode=False)
```

Out[ ]:

|    | sampling | sampling_strategy | param        | value | mean_precision | me |
|----|----------|-------------------|--------------|-------|----------------|----|
| 0  | NONE     | 0.0               | n_estimators | 50    | 0.949544       |    |
| 1  | NONE     | 0.0               | n_estimators | 100   | 0.942264       |    |
| 2  | NONE     | 0.0               | n_estimators | 200   | 0.939346       |    |
| 3  | NONE     | 0.0               | n_estimators | 300   | 0.934282       |    |
| 4  | UNDER    | 0.5               | n_estimators | 50    | 0.156230       |    |
| 5  | UNDER    | 0.5               | n_estimators | 100   | 0.155396       |    |
| 6  | UNDER    | 0.5               | n_estimators | 200   | 0.156405       |    |
| 7  | UNDER    | 0.5               | n_estimators | 300   | 0.155190       |    |
| 8  | UNDER    | 1.0               | n_estimators | 50    | 0.066562       |    |
| 9  | UNDER    | 1.0               | n_estimators | 100   | 0.066395       |    |
| 10 | UNDER    | 1.0               | n_estimators | 200   | 0.068381       |    |
| 11 | UNDER    | 1.0               | n_estimators | 300   | 0.069500       |    |
| 12 | OVER     | 0.5               | n_estimators | 50    | 0.890194       |    |
| 13 | OVER     | 0.5               | n_estimators | 100   | 0.893616       |    |
| 14 | OVER     | 0.5               | n_estimators | 200   | 0.901111       |    |
| 15 | OVER     | 0.5               | n_estimators | 300   | 0.906055       |    |
| 16 | OVER     | 1.0               | n_estimators | 50    | 0.896048       |    |
| 17 | OVER     | 1.0               | n_estimators | 100   | 0.901342       |    |
| 18 | OVER     | 1.0               | n_estimators | 200   | 0.903465       |    |
| 19 | OVER     | 1.0               | n_estimators | 300   | 0.906116       |    |
| 20 | NONE     | 0.0               | max_features | sqrt  | 0.942264       |    |
| 21 | NONE     | 0.0               | max_features | log2  | 0.941712       |    |
| 22 | NONE     | 0.0               | max_features | 0.3   | 0.937374       |    |
| 23 | NONE     | 0.0               | max_features | 0.5   | 0.932209       |    |
| 24 | NONE     | 0.0               | max_features | None  | 0.937764       |    |
| 25 | UNDER    | 0.5               | max_features | sqrt  | 0.155396       |    |
| 26 | UNDER    | 0.5               | max_features | log2  | 0.160653       |    |
| 27 | UNDER    | 0.5               | max_features | 0.3   | 0.137465       |    |
| 28 | UNDER    | 0.5               | max_features | 0.5   | 0.112842       |    |
| 29 | UNDER    | 0.5               | max_features | None  | 0.097577       |    |
| 30 | UNDER    | 1.0               | max_features | sqrt  | 0.066395       |    |
| 31 | UNDER    | 1.0               | max_features | log2  | 0.067675       |    |
| 32 | UNDER    | 1.0               | max_features | 0.3   | 0.057353       |    |

|    | sampling | sampling_strategy | param        | value              | mean_precision | me |
|----|----------|-------------------|--------------|--------------------|----------------|----|
| 33 | UNDER    | 1.0               | max_features | 0.5                | 0.048984       |    |
| 34 | UNDER    | 1.0               | max_features | None               | 0.044551       |    |
| 35 | OVER     | 0.5               | max_features | sqrt               | 0.893616       |    |
| 36 | OVER     | 0.5               | max_features | log2               | 0.905355       |    |
| 37 | OVER     | 0.5               | max_features | 0.3                | 0.889520       |    |
| 38 | OVER     | 0.5               | max_features | 0.5                | 0.889069       |    |
| 39 | OVER     | 0.5               | max_features | None               | 0.739697       |    |
| 40 | OVER     | 1.0               | max_features | sqrt               | 0.901342       |    |
| 41 | OVER     | 1.0               | max_features | log2               | 0.902270       |    |
| 42 | OVER     | 1.0               | max_features | 0.3                | 0.884775       |    |
| 43 | OVER     | 1.0               | max_features | 0.5                | 0.855581       |    |
| 44 | OVER     | 1.0               | max_features | None               | 0.712222       |    |
| 45 | NONE     | 0.0               | class_weight | balanced           | 0.951455       |    |
| 46 | NONE     | 0.0               | class_weight | balanced_subsample | 0.948298       |    |
| 47 | NONE     | 0.0               | class_weight | None               | 0.942264       |    |
| 48 | NONE     | 0.0               | n_estimators | 500                | 0.939281       |    |
| 49 | UNDER    | 0.5               | n_estimators | 500                | 0.157271       |    |
| 50 | UNDER    | 1.0               | n_estimators | 500                | 0.068811       |    |
| 51 | OVER     | 0.5               | n_estimators | 500                | 0.900635       |    |
| 52 | OVER     | 1.0               | n_estimators | 500                | 0.906116       |    |

```
In [ ]: def get_cmap_iterator(n, color_name):
    cmap = matplotlib.colormaps[color_name].resampled(n)
    list = [matplotlib.colors.to_hex(cmap(i)) for i in range(n)]
    return iter(reversed(list))

def draw_n_estimators_plot(results: pd.DataFrame):
    overs = results[(results["sampling"] == "OVER") & (results["param"] == "n_es")]

    plt.figure()

    for val, group in overs.groupby("sampling_strategy"):
        plt.plot(group["value"], group["mean_avg_precision"], label=f"Sampling s

        plt.ylim(bottom=0.76, top=0.86)
        plt.xlabel("n_estimators - liczba drzew")
        plt.ylabel("Avg Precision Score - średnia z 5 walidacji krzyżowych\n")
        plt.title("Avg Precision Score vs. n_estimators dla SMOTE oversampling")
        plt.tight_layout()
        plt.legend()
```

```

plt.grid(True, linestyle='--', alpha=0.6)
plt.show()

def draw_recall_precision_plot(results: pd.DataFrame):
    filtered_results = results[(results["value"] == "100") | (results["param"] == "class_weight")]

    cmaps = [get_cmap_iterator(4, 'Purples'),
             get_cmap_iterator(5, 'Oranges'), get_cmap_iterator(4, 'Greens')]

    for sampling_method, group_sm in filtered_results.groupby("sampling"):
        sampling_strategy = SamplingMethod.from_string(sampling_method)
        if sampling_strategy == SamplingMethod.NONE:
            for value, group_val in group_sm[group_sm["param"] == "class_weight"]:
                cmap = cmaps[sampling_strategy.value]
                color = next(cmap)
                plt.scatter(group_val["mean_recall"], group_val["mean_precision"],
                            label=f"{sampling_strategy.to_string()}" (class_weight))
        else:
            for strategy, group_strat in group_sm.groupby("sampling_strategy"):
                cmap = cmaps[sampling_strategy.value]
                color = next(cmap)
                plt.scatter(group_strat["mean_recall"], group_strat["mean_precision"],
                            label=f"{sampling_strategy.to_string()}" (strategy=sampling_strategy))

    ax = plt.gca()
    plt.xlabel("Średni odzysk")
    plt.ylabel("Średnia precyzja")
    plt.title("Porównanie precyzji i odzysku na podstawie metody próbkowania\n(podanych dla 100 sampli)")
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.legend(loc="lower left", prop={'size': 8})
    plt.figure(figsize=(8, 6))
    plt.show()

```

```

def draw_best_f1_plot(results: pd.DataFrame):
    names = []
    values = []
    best_f1_ids = results.groupby("sampling")["mean_f1"].idxmax()
    for id in best_f1_ids:
        entry = results.iloc[id]
        names.append(
            f"{SamplingMethod.from_string(entry['sampling']).to_string()}\n{sampling_strategy}"
        )
        values.append(entry["mean_f1"])

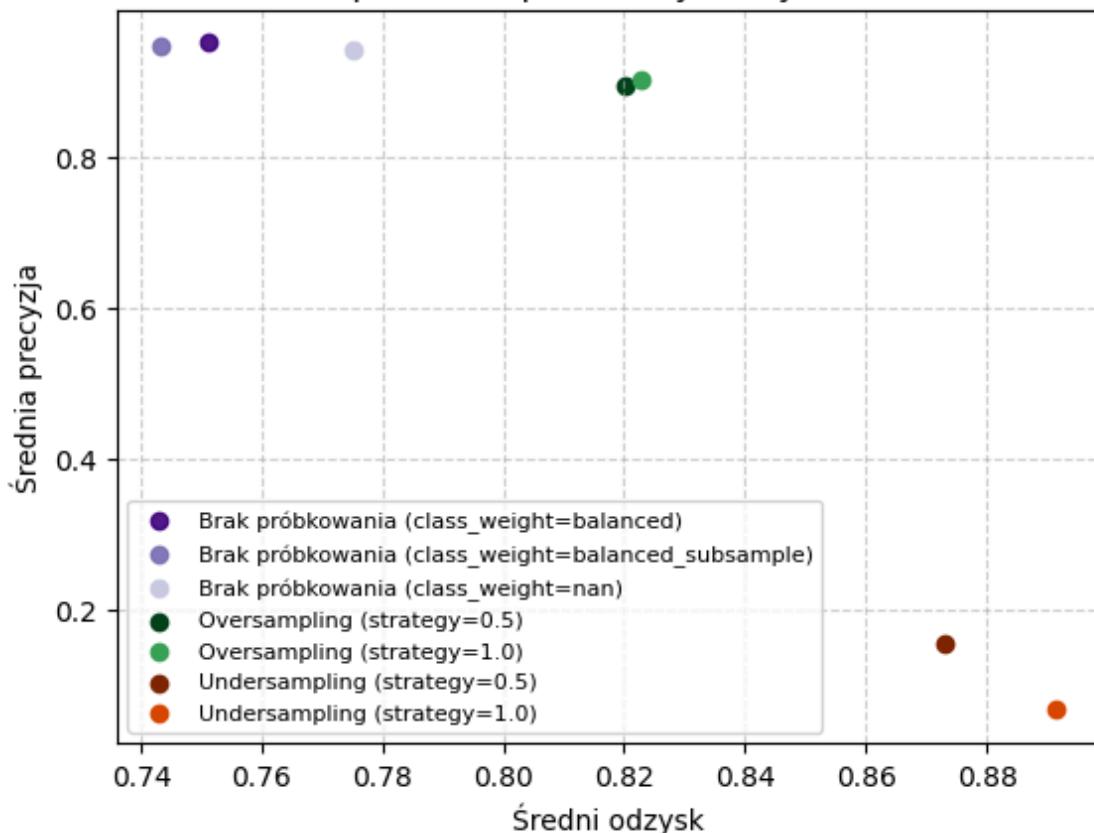
    plt.bar(names, values)
    plt.title('Najwyższy współczynnik F1 uzyskany dla\n poszczególnych strategii')
    plt.xlabel("Strategia")
    plt.ylabel('F1 - średnia z 5 walidacji krzyżowych')
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.tight_layout()
    plt.figure(figsize=(8, 6))
    plt.show()

```

## Porównanie precyzji i odzysku

In [ ]: draw\_recall\_precision\_plot(random\_forest\_results)

## Porównanie precyzji i odzysku na podstawie metody próbkowania (pozostałe parametry domyślne)



<Figure size 800x600 with 0 Axes>

Na wykresie pokazano 3 metody samplingu:

- brak próbkowania
- undersampling
- oversampling

dla domyślnych parametrów modelu:

- `n_estimators = 100`
- `max_features = 'sqrt'`
- `class_weight = None`

z różnymi wartościami *strategy*, gdzie:

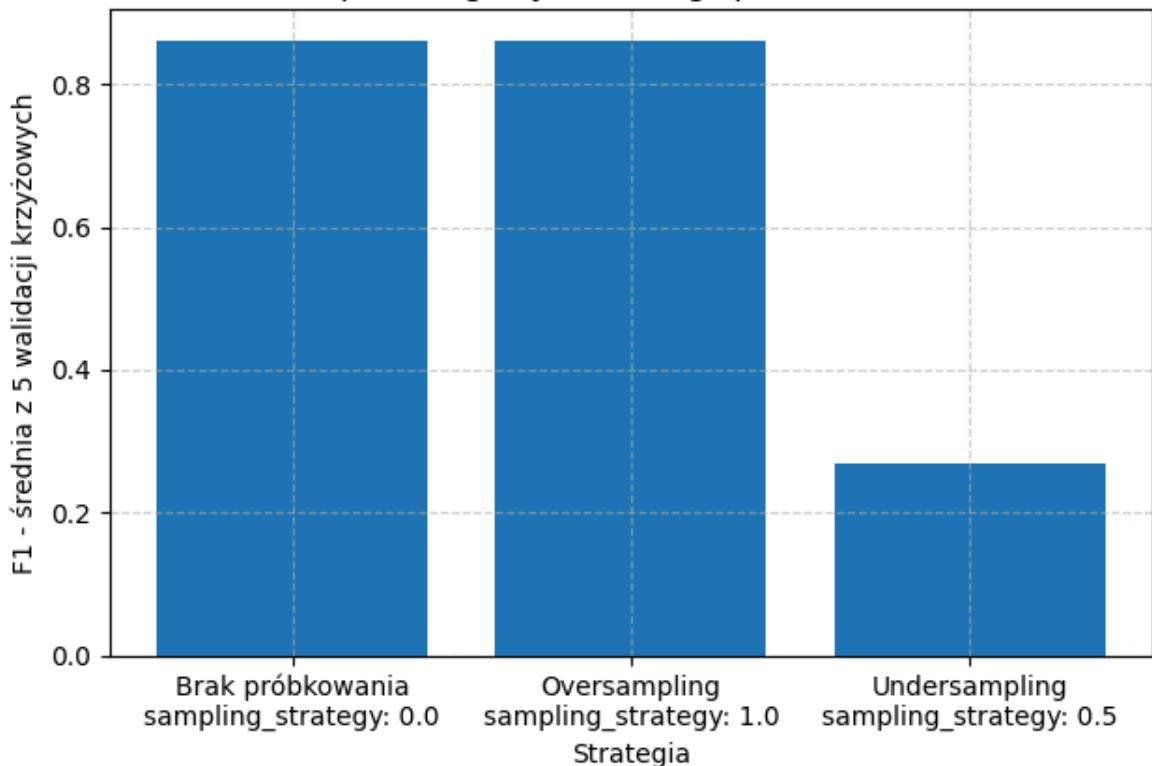
- *1* = klasy całkowicie zbilansowane
- *0.5* = klasa większościowa pozostaje w większości, liczba rekordów klasy mniejszościowej jest o 50% mniejsza.

Na wykresie widać, że najlepszy odzysk (~0.89) uzyskał undersampling. Jest to jednak obarczone bardzo niską precyzją (poniżej 0.2). Najlepszą równowagę między precyzją a odzyskiem uzyskaliśmy wykorzystując **oversampling** z wartością **strategy=0.1**.

## F1 dla poszczególnych strategii próbkowania

In [ ]: `draw_best_f1_plot(random_forest_results)`

Najwyższy współczynnik F1 uzyskany dla poszczególnych strategii próbkowania

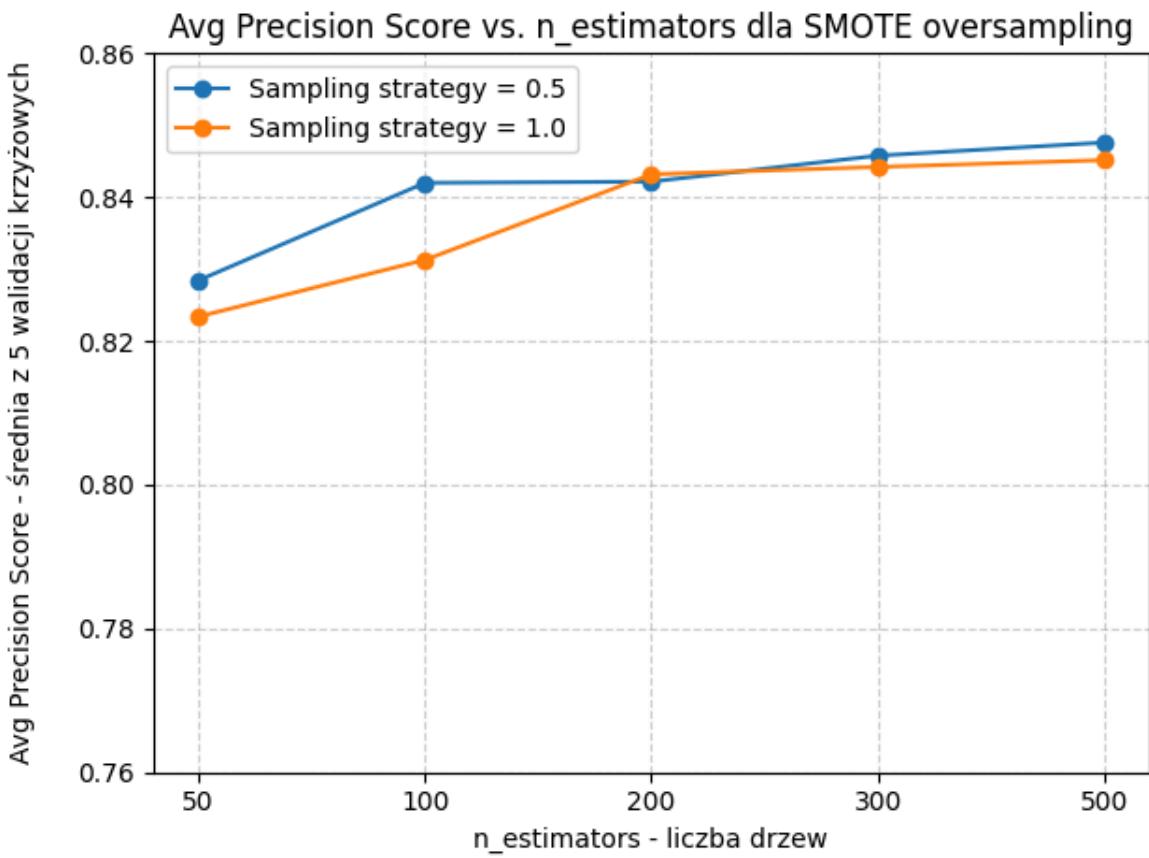


<Figure size 800x600 with 0 Axes>

Dla poszczególnych strategii wyszczególniono najwyższą wartość współczynnika F1 dla dowolnych wartości pozostałych parametrów. Tu również widać, że undersampling uzyskał najsłabsze wyniki.

### Wpływ liczby drzew

```
In [ ]: draw_n_estimators_plot(random_forest_results)
```



Dla nadpróbkowania, które wykazuje najlepsze wyniki w testach, sprawdzono wpływ parametru `n_estimators` - czyli liczby drzew, na *Average Precision Score*. Okazuje się, że krzywa zaczyna "wypłaszczać się" w okolicach wartości parametru `n_estimators=200`. Dla wartości `500` można było dalej uzyskać nieznacznie lepszy model przy wydłużonym, jednak nadal akceptowalnym czasie trwania treningu.

## Wybór najlepszych parametrów

| In [ ]: | random_forest_results.sort_values(by=[ "mean_interpolated_auc" ], ascending=False) |                   |              |       |                |             |      |  |
|---------|------------------------------------------------------------------------------------|-------------------|--------------|-------|----------------|-------------|------|--|
| Out[ ]: | sampling                                                                           | sampling_strategy | param        | value | mean_precision | mean_recall | me   |  |
| 51      | OVER                                                                               | 0.5               | n_estimators | 500   | 0.900635       | 0.822702    | 0.85 |  |
| 15      | OVER                                                                               | 0.5               | n_estimators | 300   | 0.906055       | 0.820070    | 0.85 |  |
| 36      | OVER                                                                               | 0.5               | max_features | log2  | 0.905355       | 0.820105    | 0.85 |  |
| 13      | OVER                                                                               | 0.5               | n_estimators | 100   | 0.893616       | 0.820070    | 0.85 |  |
| 35      | OVER                                                                               | 0.5               | max_features | sqrt  | 0.893616       | 0.820070    | 0.85 |  |

Najwyższą średnią `AUC=0.848472` dla wykresu precision-recall uzyskał zestaw parametrów (Oversampling, `strategy=0.5` , `n_estimators=500` , `max_features=sqrt` , `class_weight=None` ). Oprócz tego wysoki współczynnik `F1` (~0.859) i `precision` (~0.90). Dla uzyskanych parametrów przechodzimy do treningu modelu na pełnym zbiorze danych i testów na zbiorze testowym.

```
In [ ]: import joblib

best_params = {
    "random_state": SEED,
    "n_jobs": -1,
    "n_estimators": 500,
    "max_features": "sqrt",
    "class_weight": None,
}

model = train_model(train_df, test_df, best_params, build_random_forest_model, "ra

x_test = test_df.drop(columns=["Class"])
y_test = test_df["Class"]

if QUICK_MODE:
    x_test_full = full_test_df.drop(columns=["Class"])
    y_test_full = full_test_df["Class"]
    results_quick = evaluate_model(model, x_test, y_test)
    plot_pr_curve(recall_vals=results_quick["recall_vals"], precision_vals=results
                  avg_precision=results_quick["avg_precision"], plot_title="Krzywa

# Pełen model wytrenowany wcześniej z tymi samymi parametram
full_model = joblib.load("Results/random_forest.joblib")
results = evaluate_model(full_model, x_test_full, y_test_full)
plot_pr_curve(recall_vals=results["recall_vals"], precision_vals=results["precision"]
              avg_precision=results["avg_precision"], plot_title="Krzywa Preci

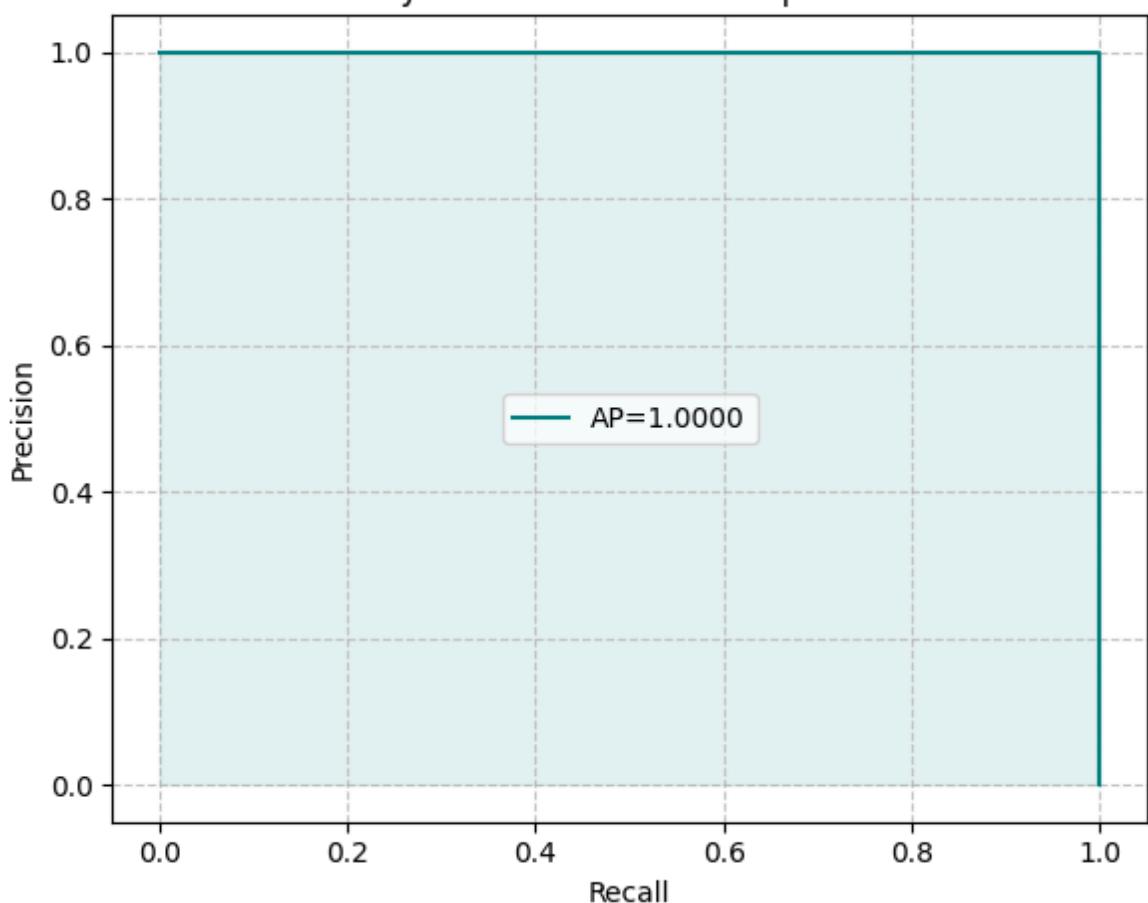
results_df = pd.DataFrame({
    "Tryb": ["quickrun", "full"],
    "Precision": [results_quick["precision"], results["precision"]],
    "Recall": [results_quick['recall'], results['recall']],
    "F1-Score": [results_quick['f1'], results['f1']],
    "Avg. Precision": [results_quick['avg_precision'], results['avg_precision']],
    "AUC PR (interpolowane)": [results_quick['interpolated_auc'], results['interpolated_auc']]
})
display(results_df)
else:
    results = evaluate_model(model, x_test, y_test)
    plot_pr_curve(recall_vals=results["recall_vals"], precision_vals=results["precision"]
                  avg_precision=results["avg_precision"], plot_title="Krzywa Preci
    results

best_results_df = pd.DataFrame({
    "Model": ["Random Forest"],
    "Precision": [results["precision"]],
    "Recall": [results['recall']],
    "F1-Score": [results['f1']],
    "Avg. Precision": [results['avg_precision']],
    "AUC PR (interpolowane)": [results['interpolated_auc']]
})
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.833333  | 1.0    | 0.909091 |                | 1.0                    |

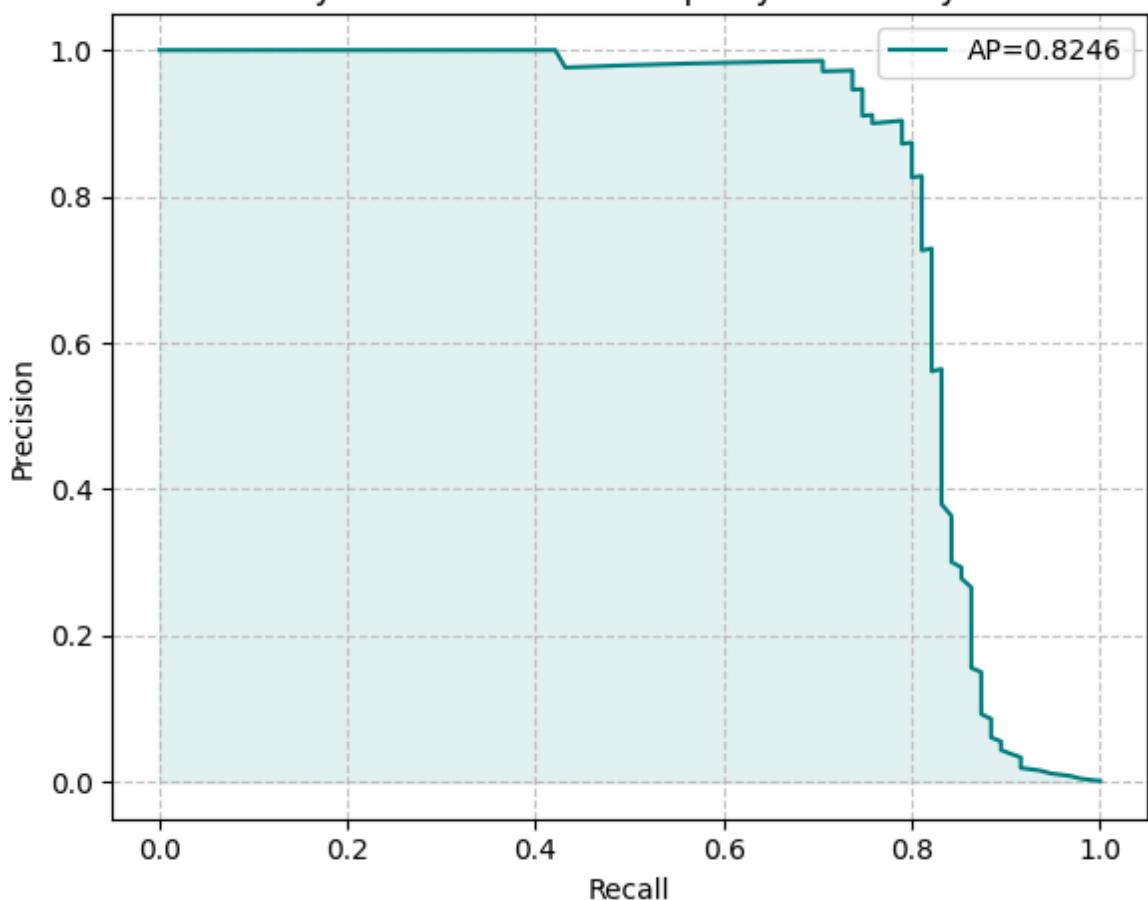
### Krzywa Precision-Recall - quickrun



Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.911392  | 0.757895 | 0.827586 | 0.824574       | 0.825025               |

## Krzywa Precision-Recall - pełny zbiór danych



|   | Tryb     | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|----------|-----------|----------|----------|----------------|------------------------|
| 0 | quickrun | 0.833333  | 1.000000 | 0.909091 | 1.000000       | 1.000000               |
| 1 | full     | 0.911392  | 0.757895 | 0.827586 | 0.824574       | 0.825025               |

Powyżej przedstawiono ostateczne wyniki modelu dla najlepszych parametrów - do testów uzyto 20% zbioru ktory nie był wykorzystywany w trakcie procesu trenowania i tuningu parametrów.

## Drzewo decyzyjne

```
In [ ]: from sklearn.tree import DecisionTreeClassifier

if QUICK_MODE:
    decision_tree_params_to_test = [
        ParameterToTest(name="max_depth", values=[None, 10]),
        ParameterToTest(name="class_weight", values=["balanced"]),
    ]

    decision_tree_sampling_method_to_test = [
        SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data),
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data),
        SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, data),
    ]

else:
    decision_tree_params_to_test = [
```

```

ParameterToTest(name="max_depth", values=[None, 3, 5, 10, 15, 20, 30]),
ParameterToTest(name="class_weight", values=[None, "balanced"]),
]

decision_tree_sampling_method_to_test = [
    SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data_splits=None),
    SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=0.5, data_splits=undersampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data_splits=oversampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=0.5, data_splits=oversampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, data_splits=None)
]

def build_decision_tree_classifier(**kwargs):
    return DecisionTreeClassifier(**kwargs)

def decision_tree_classifier(param_name: str, sampling_method: SamplingMethodToTest):
    is_class_weight_test = param_name == "class_weight"
    is_data_sampled = sampling_method.method != SamplingMethod.NONE

    # Jeżeli dane sa samplowane, nie testujemy zadnych wartosci parametru "class_weight"
    if is_class_weight_test and is_data_sampled:
        print(f"Pomiń, ponieważ sampling method: {sampling_method.method} i parametr {param_name} nie ma sensu")
        return True
    else:
        return False

model_specific_params = {
    "random_state": SEED
}

results = params_tune(decision_tree_params_to_test, sampling_methods_to_test=decision_tree_sampling_method_to_test,
                      special_stop_case=decision_tree_classifier,
                      build_model_function=build_decision_tree_classifier,
                      model_specific_params=model_specific_params, model_name='decision_tree')
print(results)
print("\nPełne wyniki (dla quick_mode=False)\n")
decision_tree_results = pd.read_csv("Results/results-decision_tree.csv", sep="\t")
decision_tree_results

```

(17:28:44) Fold 1. Main param: max\_depth=None. All params: {'random\_state': 42, 'max\_depth': None}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.6       | 0.9    | 0.72     | 0.540176       | 0.750088               |

(17:28:44) Fold 2. Main param: max\_depth=None. All params: {'random\_state': 42, 'max\_depth': None}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.75      | 0.333333 | 0.461538 | 0.251057       | 0.542195               |

(17:28:44) Fold 1. Main param: max\_depth=10. All params: {'random\_state': 42, 'max\_depth': 10}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.6       | 0.9    | 0.72     | 0.540176       | 0.750088               |

(17:28:44) Fold 2. Main param: max\_depth=10. All params: {'random\_state': 42, 'max\_depth': 10}.

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.75      | 0.333333 | 0.461538 | 0.251057       | 0.542195               |

(17:28:44) Fold 1. Main param: max\_depth=None. All params: {'random\_state': 42, 'max\_depth': None}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.009653  | 1.0    | 0.01912  | 0.009653       | 0.504826               |

(17:28:44) Fold 2. Main param: max\_depth=None. All params: {'random\_state': 42, 'max\_depth': None}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.028409  | 0.555556 | 0.054054 | 0.016488       | 0.292335               |

(17:28:44) Fold 1. Main param: max\_depth=10. All params: {'random\_state': 42, 'max\_depth': 10}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.009653  | 1.0    | 0.01912  | 0.009653       | 0.504826               |

(17:28:44) Fold 2. Main param: max\_depth=10. All params: {'random\_state': 42, 'max\_depth': 10}.

Sampling method SamplingMethod.UNDER, sampling strategy 1.

Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.028409  | 0.555556 | 0.054054 | 0.016488       | 0.292335               |

(17:28:44) Fold 1. Main param: max\_depth=None. All params: {'random\_state': 42, 'max\_depth': None}.

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|                    | Precision                   | Recall                                               | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|--------------------|-----------------------------|------------------------------------------------------|----------|----------------|------------------------|
| 0                  | 0.416667                    | 0.5                                                  | 0.454545 | 0.209214       | 0.458774               |
| (17:28:45) Fold 2. | Main param: max_depth=None. | All params: {'random_state': 42, 'max_depth': None}. |          |                |                        |

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|                    | Precision                 | Recall                                             | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|--------------------|---------------------------|----------------------------------------------------|----------|----------------|------------------------|
| 0                  | 0.75                      | 0.333333                                           | 0.461538 | 0.251057       | 0.542195               |
| (17:28:45) Fold 1. | Main param: max_depth=10. | All params: {'random_state': 42, 'max_depth': 10}. |          |                |                        |

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|                    | Precision                 | Recall                                             | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|--------------------|---------------------------|----------------------------------------------------|----------|----------------|------------------------|
| 0                  | 0.416667                  | 0.5                                                | 0.454545 | 0.209214       | 0.458774               |
| (17:28:45) Fold 2. | Main param: max_depth=10. | All params: {'random_state': 42, 'max_depth': 10}. |          |                |                        |

Sampling method SamplingMethod.OVER, sampling strategy 1.

Ocena jakości modelu:

|                    | Precision                          | Recall                                                        | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|--------------------|------------------------------------|---------------------------------------------------------------|----------|----------------|------------------------|
| 0                  | 0.75                               | 0.333333                                                      | 0.461538 | 0.251057       | 0.542195               |
| (17:28:45) Fold 1. | Main param: class_weight=balanced. | All params: {'random_state': 42, 'class_weight': 'balanced'}. |          |                |                        |

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|                    | Precision                          | Recall                                                        | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|--------------------|------------------------------------|---------------------------------------------------------------|----------|----------------|------------------------|
| 0                  | 0.25                               | 0.2                                                           | 0.222222 | 0.05141        | 0.225705               |
| (17:28:45) Fold 2. | Main param: class_weight=balanced. | All params: {'random_state': 42, 'class_weight': 'balanced'}. |          |                |                        |

Sampling method SamplingMethod.NONE, sampling strategy 0.

Ocena jakości modelu:

|        | Precision                                                                 | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|--------|---------------------------------------------------------------------------|----------|----------|----------------|------------------------|
| 0      | 0.6                                                                       | 0.333333 | 0.428571 | 0.201057       | 0.467195               |
| Pomiń, | ponieważ sampling method: SamplingMethod.UNDER i param_name: class_weight |          |          |                |                        |
| Pomiń, | ponieważ sampling method: SamplingMethod.OVER i param_name: class_weight  |          |          |                |                        |

|   | sampling | sampling_strategy | param | value        | mean_precision |          |
|---|----------|-------------------|-------|--------------|----------------|----------|
| 0 | NONE     |                   | 0     | max_depth    | None           | 0.675000 |
| 1 | NONE     |                   | 0     | max_depth    | 10             | 0.675000 |
| 2 | UNDER    |                   | 1     | max_depth    | None           | 0.019031 |
| 3 | UNDER    |                   | 1     | max_depth    | 10             | 0.019031 |
| 4 | OVER     |                   | 1     | max_depth    | None           | 0.583333 |
| 5 | OVER     |                   | 1     | max_depth    | 10             | 0.583333 |
| 6 | NONE     |                   | 0     | class_weight | balanced       | 0.425000 |

|   | mean_recall | mean_f1  | mean_avg_precision | mean_interpolated_auc |
|---|-------------|----------|--------------------|-----------------------|
| 0 | 0.616667    | 0.590769 | 0.395617           | 0.646142              |
| 1 | 0.616667    | 0.590769 | 0.395617           | 0.646142              |
| 2 | 0.777778    | 0.036587 | 0.013070           | 0.398581              |
| 3 | 0.777778    | 0.036587 | 0.013070           | 0.398581              |
| 4 | 0.416667    | 0.458042 | 0.230136           | 0.500485              |
| 5 | 0.416667    | 0.458042 | 0.230136           | 0.500485              |
| 6 | 0.266667    | 0.325397 | 0.126234           | 0.346450              |

Pełne wyniki (dla quick\_mode=False)

Out[ ]:

|    | sampling | sampling_strategy | param     | value | mean_precision | mean_recall | n |
|----|----------|-------------------|-----------|-------|----------------|-------------|---|
| 0  | NONE     | 0.0               | max_depth | NaN   | 0.757831       | 0.753684    | 0 |
| 1  | NONE     | 0.0               | max_depth | 3     | 0.846747       | 0.732667    | 0 |
| 2  | NONE     | 0.0               | max_depth | 5     | 0.906584       | 0.764596    | 0 |
| 3  | NONE     | 0.0               | max_depth | 10    | 0.873200       | 0.753684    | 0 |
| 4  | NONE     | 0.0               | max_depth | 15    | 0.788965       | 0.753684    | 0 |
| 5  | NONE     | 0.0               | max_depth | 20    | 0.763640       | 0.753684    | 0 |
| 6  | NONE     | 0.0               | max_depth | 30    | 0.757831       | 0.753684    | 0 |
| 7  | UNDER    | 0.5               | max_depth | NaN   | 0.025423       | 0.878351    | 0 |
| 8  | UNDER    | 0.5               | max_depth | 3     | 0.130750       | 0.849158    | 0 |
| 9  | UNDER    | 0.5               | max_depth | 5     | 0.066925       | 0.854526    | 0 |
| 10 | UNDER    | 0.5               | max_depth | 10    | 0.030566       | 0.888912    | 0 |
| 11 | UNDER    | 0.5               | max_depth | 15    | 0.025423       | 0.878351    | 0 |
| 12 | UNDER    | 0.5               | max_depth | 20    | 0.025423       | 0.878351    | 0 |
| 13 | UNDER    | 0.5               | max_depth | 30    | 0.025423       | 0.878351    | 0 |
| 14 | UNDER    | 1.0               | max_depth | NaN   | 0.015496       | 0.902035    | 0 |
| 15 | UNDER    | 1.0               | max_depth | 3     | 0.056393       | 0.870281    | 0 |
| 16 | UNDER    | 1.0               | max_depth | 5     | 0.023275       | 0.896737    | 0 |
| 17 | UNDER    | 1.0               | max_depth | 10    | 0.016226       | 0.902035    | 0 |
| 18 | UNDER    | 1.0               | max_depth | 15    | 0.015496       | 0.902035    | 0 |
| 19 | UNDER    | 1.0               | max_depth | 20    | 0.015496       | 0.902035    | 0 |
| 20 | UNDER    | 1.0               | max_depth | 30    | 0.015496       | 0.902035    | 0 |
| 21 | OVER     | 0.5               | max_depth | NaN   | 0.394253       | 0.777789    | 0 |
| 22 | OVER     | 0.5               | max_depth | 3     | 0.240568       | 0.833158    | 0 |
| 23 | OVER     | 0.5               | max_depth | 5     | 0.098593       | 0.846526    | 0 |
| 24 | OVER     | 0.5               | max_depth | 10    | 0.143328       | 0.838526    | 0 |
| 25 | OVER     | 0.5               | max_depth | 15    | 0.259565       | 0.809544    | 0 |
| 26 | OVER     | 0.5               | max_depth | 20    | 0.318214       | 0.788386    | 0 |
| 27 | OVER     | 0.5               | max_depth | 30    | 0.379935       | 0.777789    | 0 |
| 28 | OVER     | 1.0               | max_depth | NaN   | 0.388434       | 0.756596    | 0 |
| 29 | OVER     | 1.0               | max_depth | 3     | 0.042453       | 0.894140    | 0 |
| 30 | OVER     | 1.0               | max_depth | 5     | 0.050419       | 0.872947    | 0 |
| 31 | OVER     | 1.0               | max_depth | 10    | 0.112384       | 0.843860    | 0 |
| 32 | OVER     | 1.0               | max_depth | 15    | 0.226973       | 0.804281    | 0 |

|    | sampling | sampling_strategy | param        | value    | mean_precision | mean_recall | n |
|----|----------|-------------------|--------------|----------|----------------|-------------|---|
| 33 | OVER     |                   | max_depth    | 20       | 0.288464       | 0.791123    | 0 |
| 34 | OVER     |                   | max_depth    | 30       | 0.343641       | 0.756561    | 0 |
| 35 | NONE     |                   | class_weight | NaN      | 0.757831       | 0.753684    | 0 |
| 36 | NONE     |                   | class_weight | balanced | 0.755800       | 0.709123    | 0 |

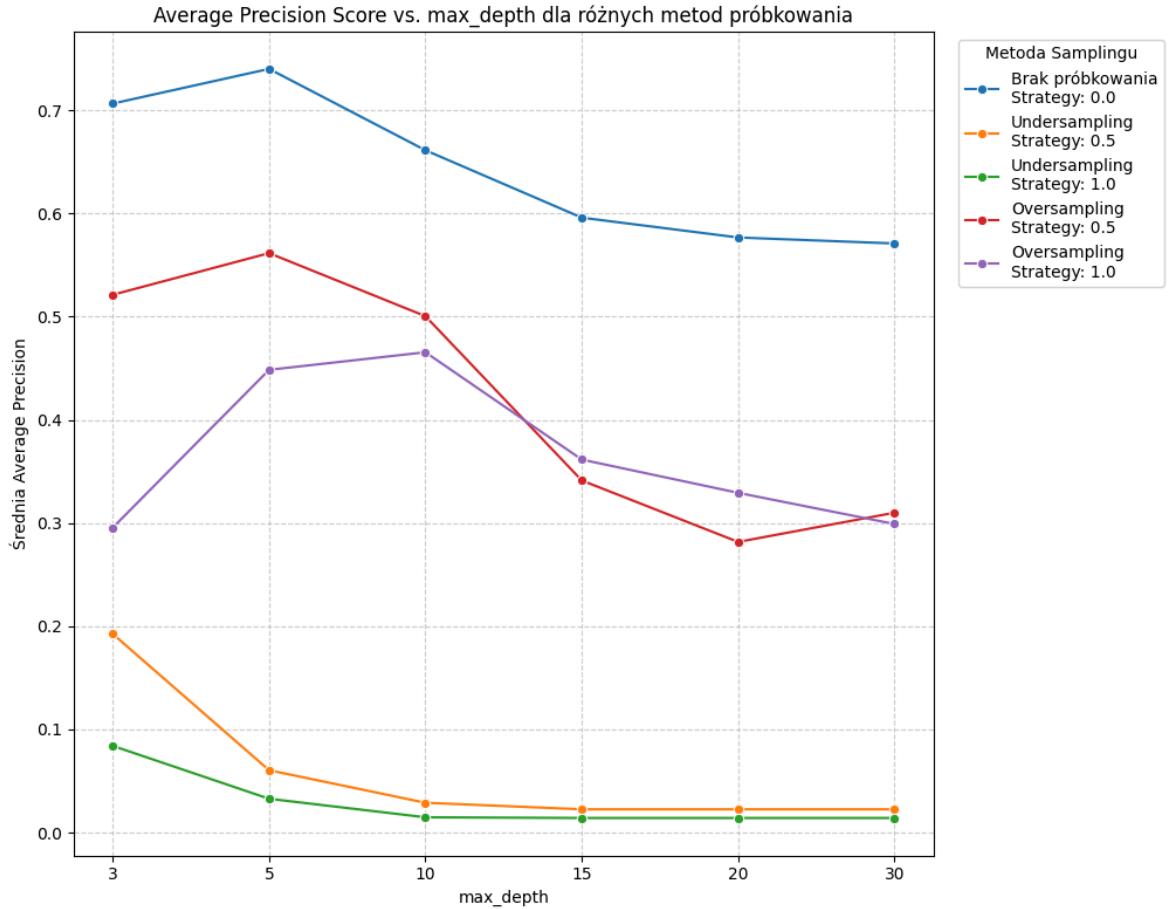
```
In [ ]: def plot_max_depth_vs_ap(results_df):
    max_depth_results = results_df[results_df['param'] == 'max_depth'].copy()

    max_depth_results['sampling_label'] = max_depth_results.apply(
        lambda row: f"{SamplingMethod.from_string(row['sampling'])}.to_string()"}\n
        axis=1
    )
    plt.figure(figsize=(12, 8))
    sns.lineplot(
        data=max_depth_results,
        x='value',
        y='mean_avg_precision',
        hue='sampling_label',
        marker='o'
    )

    plt.title('Average Precision Score vs. max_depth dla różnych metod próbkowań')
    plt.xlabel('max_depth')
    plt.ylabel('Średnia Average Precision')
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.legend(title='Metoda Samplingu', bbox_to_anchor=(1.02, 1), loc='upper left')
    plt.tight_layout(rect=[0, 0, 0.85, 1]) # Zostaw miejsce na Legende
    plt.show()
```

## Wpływ parametru `max_depth`

```
In [ ]: plot_max_depth_vs_ap(decision_tree_results)
```



Na wykresie przedstawiono wpływ parametru `max_depth` - czyli maksymalna głębokość drzewa - na uzyskany wynik parametru Average Precision Score dla różnych metod próbkowania.

Widać tutaj, że dla drzewa decyzyjnego, zastosowanie samplingu nie poprawia jakości modelu. Najlepszy wynik uzyskano dla braku próbkowania i maksymalnej głębokości wynoszącej 5.

## Porównanie precyzji i odzysku

```
In [ ]: def draw_recall_precision_plot(results: pd.DataFrame):

    cmaps = [get_cmap_iterator(4, 'Purples'),
             get_cmap_iterator(5, 'Oranges'), get_cmap_iterator(4, 'Greens')]

    for sampling_method, group_sm in results.groupby("sampling"):
        sampling_strategy = SamplingMethod.from_string(sampling_method)
        if sampling_strategy == SamplingMethod.NONE:
            for value, group_val in group_sm[group_sm["param"] == "class_weight"]:
                cmap = cmaps[sampling_strategy.value]
                color = next(cmap)
                plt.scatter(group_val["mean_recall"], group_val["mean_precision"],
                            label=f"{sampling_strategy.to_string()} {class_weight}")
        else:
            for strategy, group_strat in group_sm.groupby("sampling_strategy"):
                cmap = cmaps[sampling_strategy.value]
                color = next(cmap)
                plt.scatter(group_strat["mean_recall"], group_strat["mean_precision"],
                            label=f"{sampling_strategy.to_string()} {strategy} {strategy}")
```

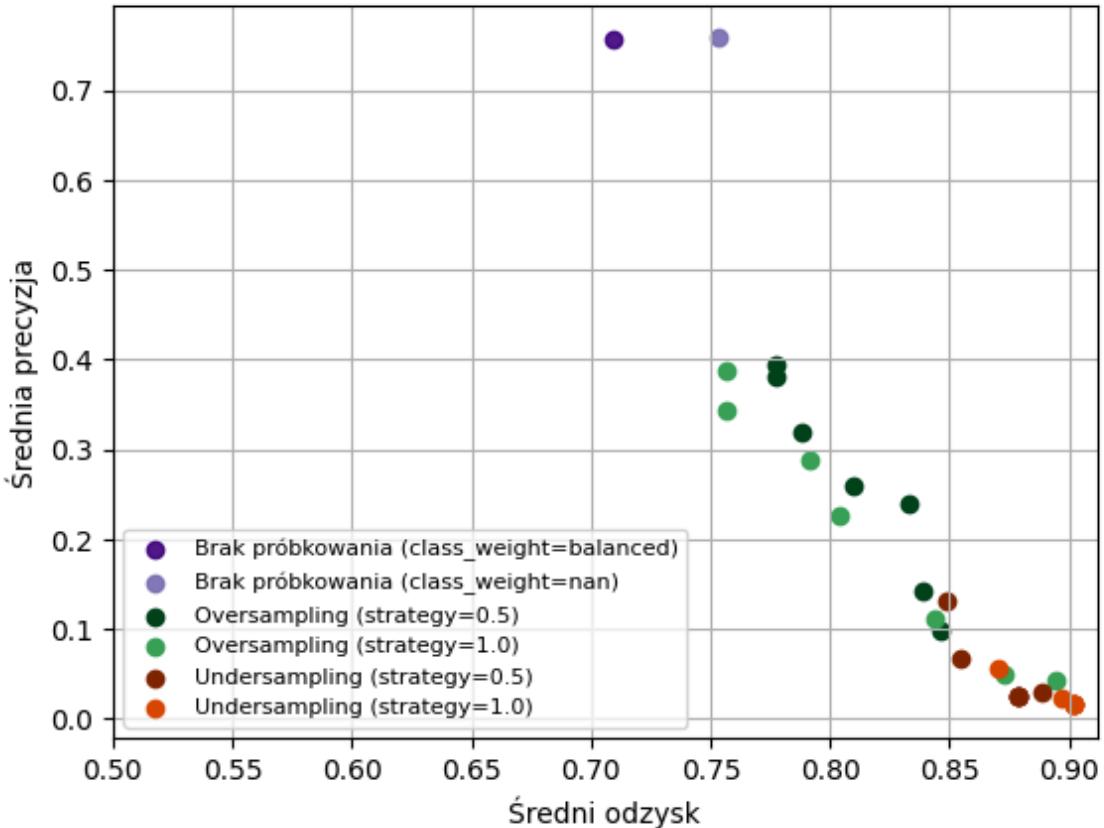
```

ax = plt.gca()
plt.xlabel("Średni odzysk")
plt.ylabel("Średnia precyza")
plt.title("Porównanie precyzji i odzysku na podstawie metody próbkowania")
plt.grid(True)
plt.xlim(left = 0.5)
plt.legend(loc="lower left", prop={'size': 8})
plt.figure(figsize=(8, 6))
plt.show()

draw_recall_precision_plot(decision_tree_results)

```

Porównanie precyzji i odzysku na podstawie metody próbkowania



<Figure size 800x600 with 0 Axes>

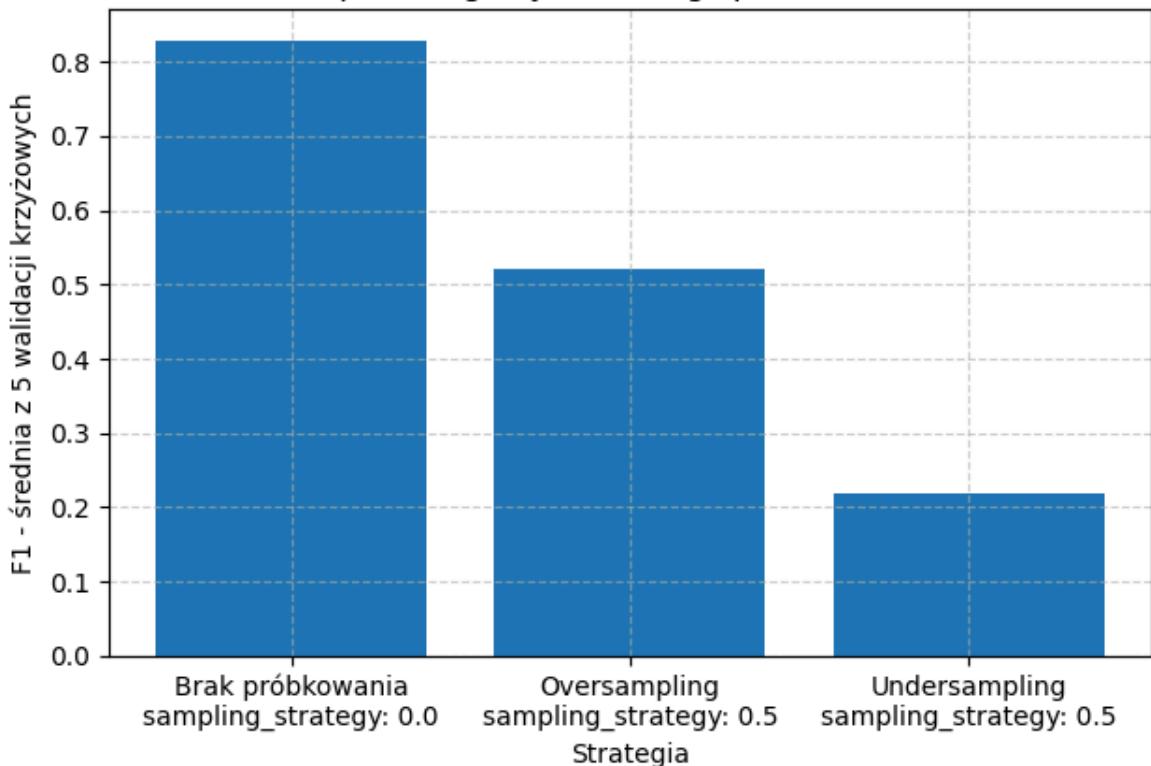
Na wykresie porównano precyzję i odzysk dla różnych wartości `max_depth` z podziałem na metody próbkowania.

Najwyższy odzysk można uzyskać stosując podpróbkowanie, jednak kosztem bardzo niskiej precyzji, poniżej (< 0.15). W zależności od kosztów obsługi fałszywego alarmu oraz nie wykrycia transakcji-oszustwa, w modelu produkcyjnym można rozważyć nadpróbkowanie albo całkowitą rezygnację z próbkowania.

## F1 dla poszczególnych strategii próbkowania

```
In [ ]: draw_best_f1_plot(decision_tree_results)
```

## Najwyższy współczynnik F1 uzyskany dla poszczególnych strategii próbkowania



<Figure size 800x600 with 0 Axes>

Dla poszczególnych strategii wyszczególniony najwyższą wartość współczynnika F1 dla dowolnych wartości pozostałych parametrów. Brak próbkowania wybija się z wynikiem >  
0.8

## Wybór najlepszych parametrów

In [ ]: `decision_tree_results.sort_values(by=["mean_interpolated_auc"], ascending=False)`

|    | sampling | sampling_strategy | param        | value | mean_precision | mean_recall | me    |
|----|----------|-------------------|--------------|-------|----------------|-------------|-------|
| 1  | NONE     | 0.0               | max_depth    | 3     | 0.846747       | 0.732667    | 0.78: |
| 2  | NONE     | 0.0               | max_depth    | 5     | 0.906584       | 0.764596    | 0.82: |
| 0  | NONE     | 0.0               | max_depth    | NaN   | 0.757831       | 0.753684    | 0.75: |
| 35 | NONE     | 0.0               | class_weight | NaN   | 0.757831       | 0.753684    | 0.75: |
| 6  | NONE     | 0.0               | max_depth    | 30    | 0.757831       | 0.753684    | 0.75: |

Najwyższą średnią AUC=0.803736 dla wykresu precision-recall uzyskano bez próbkowania, z parametrem `max_depth=3`. Na drugim miejscu, z wartością `max_depth=5`, wartość AUC jest nieznacznie niższa ( AUC=0.784761 ), natomiast z uwagi na lepsze wartości precyzji, odzysku i współczynnika F1, uznano że jest to bardziej optymalne ustawienie.

In [ ]: `import joblib  
best_params = {`

```

        "max_depth": 5,
        "class_weight": None
    }

model = train_model(train_df, test_df, best_params, build_decision_tree_classifier)
x_test = test_df.drop(columns=["Class"])
y_test = test_df["Class"]

if QUICK_MODE:
    x_test_full = full_test_df.drop(columns=["Class"])
    y_test_full = full_test_df["Class"]
    results_quick = evaluate_model(model, x_test, y_test)
    plot_pr_curve(recall_vals=results_quick["recall_vals"], precision_vals=results_quick["precision_vals"], avg_precision=results_quick["avg_precision"], plot_title="Krzywa Precyzyjno-Pozytywnego Modelu")

    # Pełen model wytrenowany wcześniej z tymi samymi parametrami
    full_model = joblib.load("Results/decision_tree.joblib")
    results = evaluate_model(full_model, x_test_full, y_test_full)
    plot_pr_curve(recall_vals=results["recall_vals"], precision_vals=results["precision_vals"], avg_precision=results["avg_precision"], plot_title="Krzywa Precyzyjno-Pozytywnego Modelu")

    results_df = pd.DataFrame({
        "Tryb": ["quickrun", "full"],
        "Precision": [results_quick["precision"], results["precision"]],
        "Recall": [results_quick['recall'], results['recall']],
        "F1-Score": [results_quick['f1'], results['f1']],
        "Avg. Precision": [results_quick['avg_precision'], results['avg_precision']],
        "AUC PR (interpolowane)": [results_quick['interpolated_auc'], results['interpolated_auc']]
    })
    display(results_df)

else:
    results = evaluate_model(model, x_test, y_test)
    plot_pr_curve(recall_vals=results["recall_vals"], precision_vals=results["precision_vals"], avg_precision=results["avg_precision"], plot_title="Krzywa Precyzyjno-Pozytywnego Modelu")
    display(results)

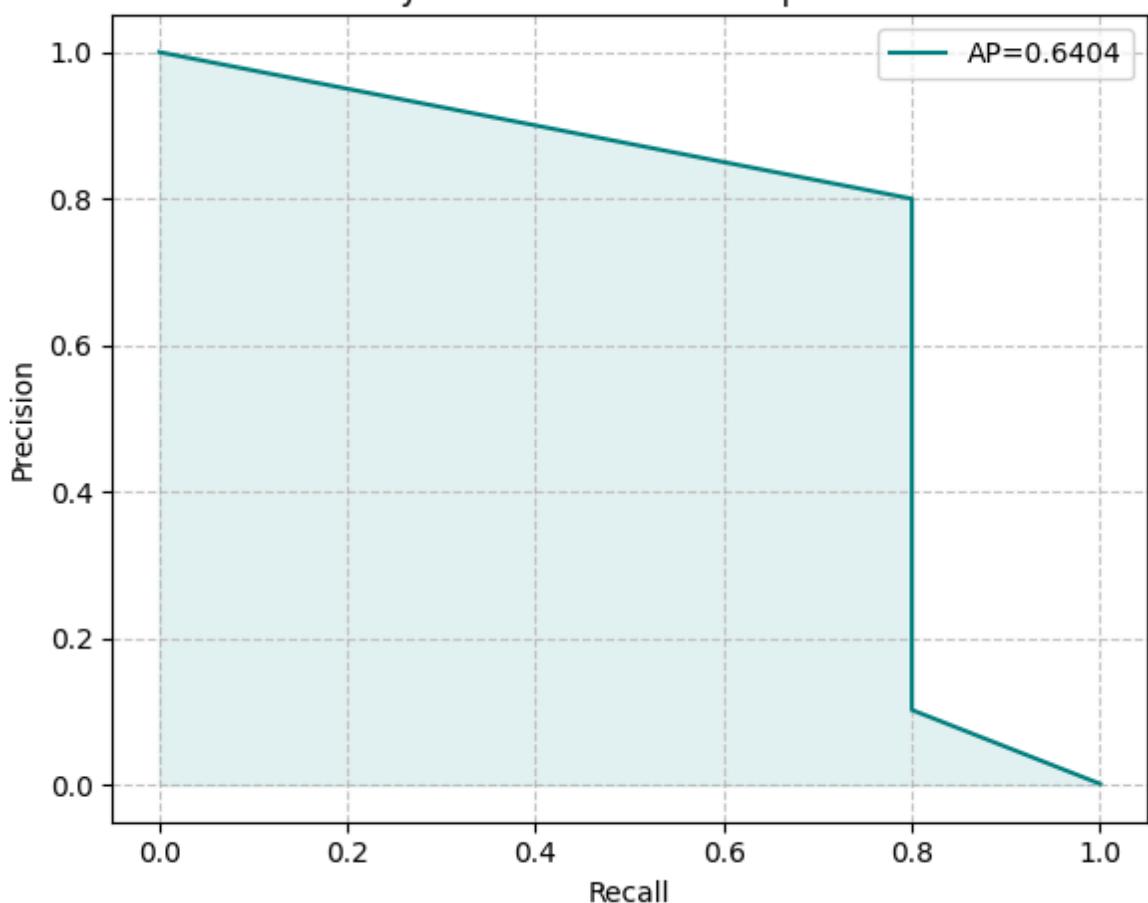
best_results_df = pd.concat([best_results_df,
                             pd.DataFrame([[["Decision Tree", results["precision"], results['f1'], results['avg_precision']]]], columns=best_results_df.columns)],
                            ignore_index=True)

```

Ocena jakości modelu:

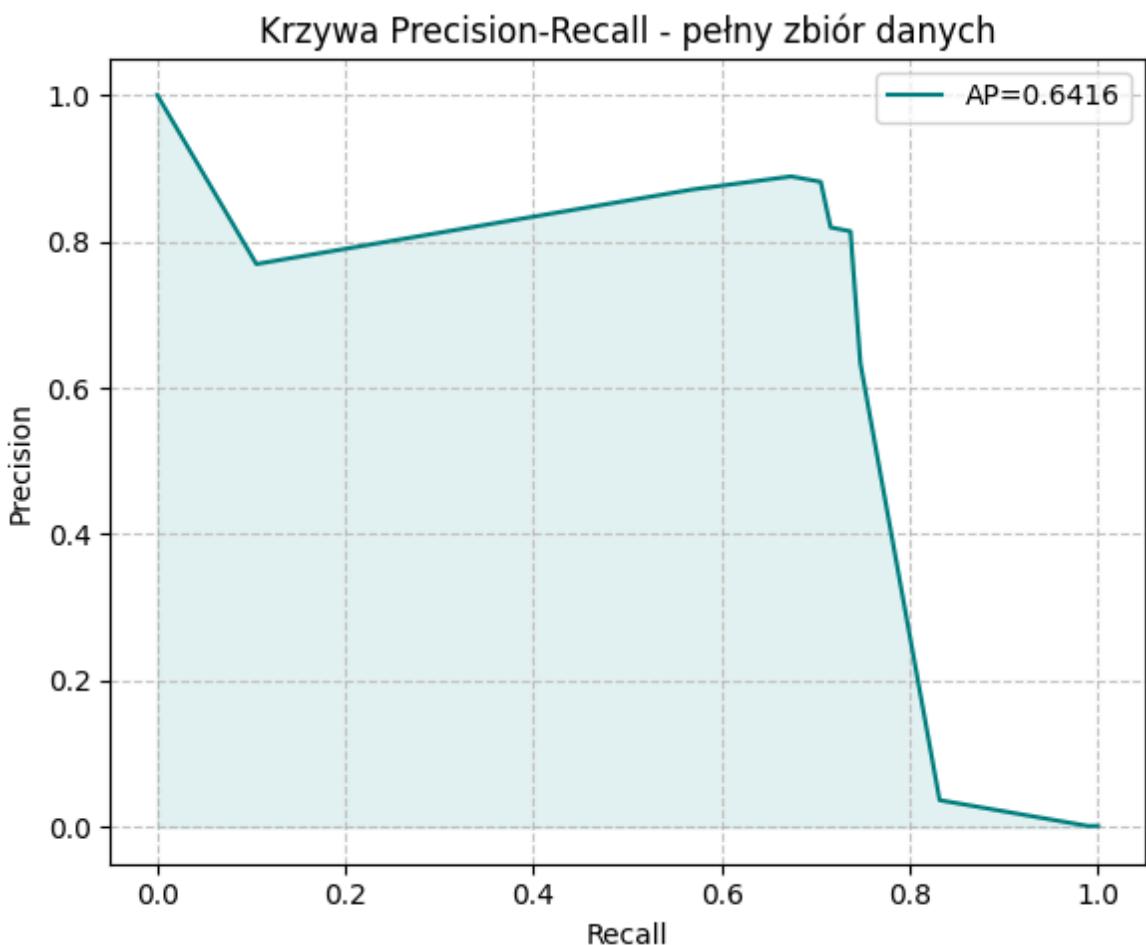
|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.8       | 0.8    | 0.8      | 0.640352       | 0.730433               |

### Krzywa Precision-Recall - quickrun



Ocena jakości modelu:

|   | Precision | Recall   | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|----------|----------|----------------|------------------------|
| 0 | 0.881579  | 0.705263 | 0.783626 | 0.641593       | 0.658603               |



## Metody nienadzorowane

Analiza każdej metody nienadzorowanej będzie składało się z dwóch etapów:

1. Proces strojenia parametrów algorytmu, gdzie została wykorzystana technika k - krotnej walidacji krzyżowej - całe przeprowadzone tutaj działanie umożliwiło nam dokonanie badania, jak konkretne wartości danych parametrów wpływają na jakość uzyskanego modelu.
2. Trening i ocena modelu - posiadawszy już wiedzę o najlepszych parametrach, chcemy uzyskać ostateczny najlepszy model możliwy dla danej metody i dostarczonych danych, będziemy tutaj porównywać modele wytrenowane w ramach danego przetwarzania oraz modele, które zostały przygotowane wcześniej.

Przy budowaniu już konkretnych modeli zbadamy również kwestie treningu na danych reprezentujących wyłącznie klasę większościową. Będziemy starali się uzyskać jak najbardziej zbalansowane modele między wartościami dla Recall a Precision, gdzie chcielibyśmy uzyskać dla Recall wynik wyższy niż 0.8, a dla Precision, żeby oscylował w okolicach min. 0.1.

# One-Class SVM

Przy budowaniu modelu opartego na algorytmie One-Class SVM, skupimy się na następujących parametrach:

- *kernel* - rodzaj jądra, definiuje sposób obliczania podobieństwa między punktami, rozważamy dwie wartości - *linear* oraz *rbf*,
- *nu* - stanowi górne ograniczenie na odsetek osobników odstających i dolne na liczbę wektorów wspierających,
- *gamma* - parametr jądra *rbf* - kontroluje wpływ pojedynczych punktów.

Warto jeszcze zaznaczyć, że proces budowania takiego modelu jest mocno uzależniony od rozmiaru danych i potrafi się bardzo wydłużyć w przypadku dużych zbiorów. Dlatego przy procesie treningu postanowiłem zwiększyć domyślny *cache\_size* do 8192 MB oraz jeśli chodzi o strojenie parametrów, to tam jeszcze zwiększyłem wartość parametru *tol*, aby kryterium stopu było mniej twardze.

```
In [ ]: from sklearn.svm import OneClassSVM
def build_oc_svm_model_function(**kwargs):
    return OneClassSVM(**kwargs)
```

## Proces strojenia parametrów i badań ich wpływu na jakość modelu

Poniżej następuje uruchomienie całego procesu strojenia parametrów, wraz z prezentacją wyników w formie tabularycznej.

```
In [ ]: import os

if QUICK_MODE:
    oc_svm_params_to_test = [
        ParameterToTest(name="kernel", values=['linear', 'rbf']),
        ParameterToTest(name="nu", values=[0.01, 0.1, 0.2]),
        ParameterToTest(name="gamma", values=['scale', 'auto'], required_params=[])
]

oc_svm_sampling_method_to_test = [
    SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data_splits=undersampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data_splits=undersampling_50_data_splits),
    SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, data_splits=undersampling_50_data_splits)
]
else:
    oc_svm_params_to_test = [
        ParameterToTest(name="kernel", values=['linear', 'rbf']),
        ParameterToTest(name="nu", values=[0.01, 0.05, 0.1, 0.2]),
        ParameterToTest(name="gamma", values=['scale', 'auto', 0.01, 0.1, 1.0],
    ]

    oc_svm_sampling_method_to_test = [
        SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data_splits=undersampling_50_data_splits),
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=0.5, data_splits=undersampling_50_data_splits),
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data_splits=undersampling_50_data_splits),
        SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=0.5, data_splits=undersampling_50_data_splits)
    ]
```

```
        data_splits=oversampling_50_data_splits),
SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, da
]

# Domyślnie używamy rbf, nu=0.2 - kwestie wydajności
# 8192 MB rozmiar cache, domyślnie 200 MB, żeby było szybciej
# Także zwiększymy tolerancję, będą gorsze wyniki, ale szybciej
ocv_svm_model_specific_params = {
    "kernel": "rbf",
    "nu": 0.5,
    "cache_size": 8192,
    "tol": 0.1,
}

if (QUICK_MODE and os.path.exists(oc_svm_tune_results_tmp_file_path)):
    oc_svm_params_runtime_tune_results = pd.read_csv(oc_svm_tune_results_tmp_file_
else:
    oc_svm_params_runtime_tune_results = params_tune(oc_svm_params_to_test, sampli
  build_model_function=build_oc_svm_model
  model_specific_params=ocv_svm_model spe
display(oc_svm_params_runtime_tune_results)
```

| sampling | sampling_strategy | param | value  | mean_precision | mean_recall | mean_f1  |
|----------|-------------------|-------|--------|----------------|-------------|----------|
| 0        | NONE              | 0     | kernel | linear         | 0.001403    | 0.400000 |
| 1        | NONE              | 0     | kernel | rbf            | 0.003341    | 1.000000 |
| 2        | UNDER             | 1     | kernel | linear         | 0.000943    | 0.527778 |
| 3        | UNDER             | 1     | kernel | rbf            | 0.003063    | 0.844444 |
| 4        | OVER              | 1     | kernel | linear         | 0.001018    | 0.583333 |
| 5        | OVER              | 1     | kernel | rbf            | 0.003409    | 0.888889 |
| 6        | NONE              | 0     | nu     | 0.01           | 0.033168    | 0.833333 |
| 7        | NONE              | 0     | nu     | 0.1            | 0.014079    | 0.833333 |
| 8        | NONE              | 0     | nu     | 0.2            | 0.007549    | 0.888889 |
| 9        | UNDER             | 1     | nu     | 0.01           | 0.001674    | 1.000000 |
| 10       | UNDER             | 1     | nu     | 0.1            | 0.002989    | 0.788889 |
| 11       | UNDER             | 1     | nu     | 0.2            | 0.002655    | 0.738889 |
| 12       | OVER              | 1     | nu     | 0.01           | 0.042132    | 0.455556 |
| 13       | OVER              | 1     | nu     | 0.1            | 0.010974    | 0.566667 |
| 14       | OVER              | 1     | nu     | 0.2            | 0.005852    | 0.566667 |
| 15       | NONE              | 0     | gamma  | scale          | 0.003341    | 1.000000 |
| 16       | NONE              | 0     | gamma  | auto           | 0.003344    | 1.000000 |
| 17       | UNDER             | 1     | gamma  | scale          | 0.003063    | 0.844444 |
| 18       | UNDER             | 1     | gamma  | auto           | 0.002341    | 1.000000 |
| 19       | OVER              | 1     | gamma  | scale          | 0.003409    | 0.888889 |
| 20       | OVER              | 1     | gamma  | auto           | 0.004184    | 1.000000 |

Tutaj dla porównania zostają wyświetlane wcześniej uzyskane wyniki, które zostały zapisane do odpowiedniego pliku. Proces strojenia parametrów odbywał się na zbiorze danych, stanowiącym 33% całości. Przyczyny należy szukać w bardzo długim procesie budowania i ewaluacji danego modelu przy dużej ilości danych.

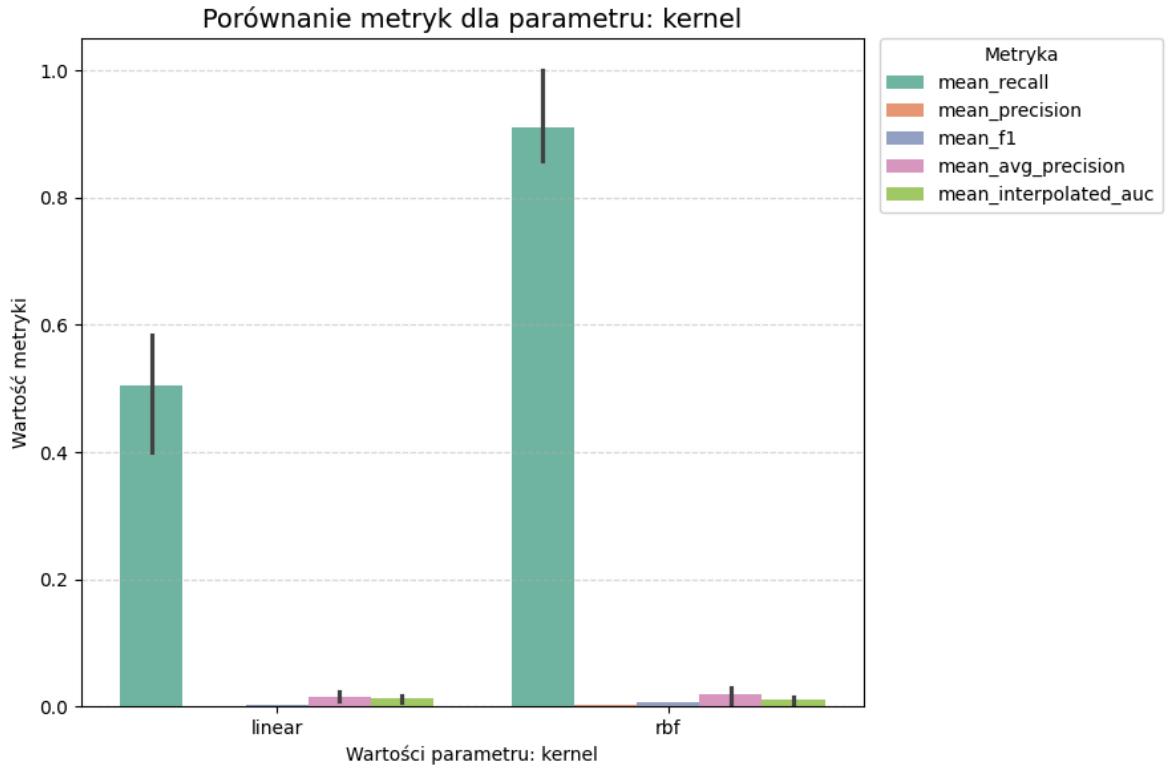
```
In [ ]: if (LOAD_PARAMS_TUNE_RESULTS_FROM_MASTER_FILE and os.path.exists(oc_svm_tune_res
oc_svm_params_full_tune_results = pd.read_csv(oc_svm_tune_results_master_file_
display(oc_svm_params_full_tune_results)
```

|    | sampling | sampling_strategy | param | value  | mean_precision | mean_recall | mean_f1  | l        |
|----|----------|-------------------|-------|--------|----------------|-------------|----------|----------|
| 0  | NONE     |                   | 0.0   | kernel | linear         | 0.001440    | 0.436333 | 0.002871 |
| 1  | NONE     |                   | 0.0   | kernel | rbf            | 0.003198    | 0.968000 | 0.006374 |
| 2  | UNDER    |                   | 0.5   | kernel | linear         | 0.000214    | 0.096667 | 0.000426 |
| 3  | UNDER    |                   | 0.5   | kernel | rbf            | 0.004317    | 0.871000 | 0.008592 |
| 4  | UNDER    |                   | 1.0   | kernel | linear         | 0.000251    | 0.136667 | 0.000502 |
| 5  | UNDER    |                   | 1.0   | kernel | rbf            | 0.003242    | 0.685667 | 0.006453 |
| 6  | OVER     |                   | 0.5   | kernel | linear         | 0.000185    | 0.080667 | 0.000370 |
| 7  | OVER     |                   | 0.5   | kernel | rbf            | 0.004347    | 0.895667 | 0.008653 |
| 8  | OVER     |                   | 1.0   | kernel | linear         | 0.000190    | 0.104667 | 0.000379 |
| 9  | OVER     |                   | 1.0   | kernel | rbf            | 0.003399    | 0.750000 | 0.006768 |
| 10 | NONE     |                   | 0.0   | nu     | 0.01           | 0.079457    | 0.694667 | 0.142557 |
| 11 | NONE     |                   | 0.0   | nu     | 0.05           | 0.028606    | 0.871667 | 0.055394 |
| 12 | NONE     |                   | 0.0   | nu     | 0.1            | 0.014734    | 0.895667 | 0.028991 |
| 13 | NONE     |                   | 0.0   | nu     | 0.2            | 0.007659    | 0.927667 | 0.015193 |
| 14 | UNDER    |                   | 0.5   | nu     | 0.01           | 0.004034    | 0.474667 | 0.007979 |
| 15 | UNDER    |                   | 0.5   | nu     | 0.05           | 0.010188    | 0.291667 | 0.019579 |
| 16 | UNDER    |                   | 0.5   | nu     | 0.1            | 0.009850    | 0.323667 | 0.019038 |
| 17 | UNDER    |                   | 0.5   | nu     | 0.2            | 0.007836    | 0.452333 | 0.015389 |
| 18 | UNDER    |                   | 1.0   | nu     | 0.01           | 0.004844    | 0.445000 | 0.009461 |
| 19 | UNDER    |                   | 1.0   | nu     | 0.05           | 0.004887    | 0.227333 | 0.009507 |
| 20 | UNDER    |                   | 1.0   | nu     | 0.1            | 0.004910    | 0.235333 | 0.009606 |
| 21 | UNDER    |                   | 1.0   | nu     | 0.2            | 0.005765    | 0.372667 | 0.011338 |
| 22 | OVER     |                   | 0.5   | nu     | 0.01           | 0.044435    | 0.234667 | 0.074672 |
| 23 | OVER     |                   | 0.5   | nu     | 0.05           | 0.020595    | 0.315667 | 0.038657 |
| 24 | OVER     |                   | 0.5   | nu     | 0.1            | 0.016463    | 0.428000 | 0.031704 |
| 25 | OVER     |                   | 0.5   | nu     | 0.2            | 0.009851    | 0.541333 | 0.019348 |
| 26 | OVER     |                   | 1.0   | nu     | 0.01           | 0.028680    | 0.170000 | 0.049042 |
| 27 | OVER     |                   | 1.0   | nu     | 0.05           | 0.016466    | 0.267000 | 0.031010 |
| 28 | OVER     |                   | 1.0   | nu     | 0.1            | 0.011352    | 0.323667 | 0.021932 |
| 29 | OVER     |                   | 1.0   | nu     | 0.2            | 0.008170    | 0.468667 | 0.016059 |
| 30 | NONE     |                   | 0.0   | gamma  | scale          | 0.003198    | 0.968000 | 0.006374 |
| 31 | NONE     |                   | 0.0   | gamma  | auto           | 0.003196    | 0.968000 | 0.006370 |
| 32 | NONE     |                   | 0.0   | gamma  | 0.01           | 0.003198    | 0.968000 | 0.006375 |

|    | sampling | sampling_strategy | param | value | mean_precision | mean_recall | mean_f1  |
|----|----------|-------------------|-------|-------|----------------|-------------|----------|
| 33 | NONE     | 0.0               | gamma | 0.1   | 0.003222       | 0.976000    | 0.006424 |
| 34 | NONE     | 0.0               | gamma | 1.0   | 0.002187       | 0.992000    | 0.004365 |
| 35 | UNDER    | 0.5               | gamma | scale | 0.004317       | 0.871000    | 0.008592 |
| 36 | UNDER    | 0.5               | gamma | auto  | 0.004510       | 0.919667    | 0.008975 |
| 37 | UNDER    | 0.5               | gamma | 0.01  | 0.004727       | 0.911667    | 0.009405 |
| 38 | UNDER    | 0.5               | gamma | 0.1   | 0.002653       | 0.959667    | 0.005292 |
| 39 | UNDER    | 0.5               | gamma | 1.0   | 0.001656       | 1.000000    | 0.003306 |
| 40 | UNDER    | 1.0               | gamma | scale | 0.003242       | 0.685667    | 0.006453 |
| 41 | UNDER    | 1.0               | gamma | auto  | 0.004104       | 0.887000    | 0.008169 |
| 42 | UNDER    | 1.0               | gamma | 0.01  | 0.004544       | 0.790333    | 0.009035 |
| 43 | UNDER    | 1.0               | gamma | 0.1   | 0.002215       | 0.959667    | 0.004420 |
| 44 | UNDER    | 1.0               | gamma | 1.0   | 0.001653       | 1.000000    | 0.003301 |
| 45 | OVER     | 0.5               | gamma | scale | 0.004347       | 0.895667    | 0.008653 |
| 46 | OVER     | 0.5               | gamma | auto  | 0.004941       | 0.943667    | 0.009830 |
| 47 | OVER     | 0.5               | gamma | 0.01  | 0.004875       | 0.919667    | 0.009699 |
| 48 | OVER     | 0.5               | gamma | 0.1   | 0.003893       | 0.952000    | 0.007755 |
| 49 | OVER     | 0.5               | gamma | 1.0   | 0.002091       | 0.975667    | 0.004173 |
| 50 | OVER     | 1.0               | gamma | scale | 0.003399       | 0.750000    | 0.006768 |
| 51 | OVER     | 1.0               | gamma | auto  | 0.005089       | 0.911667    | 0.010122 |
| 52 | OVER     | 1.0               | gamma | 0.01  | 0.005076       | 0.871000    | 0.010092 |
| 53 | OVER     | 1.0               | gamma | 0.1   | 0.003537       | 0.935667    | 0.007047 |
| 54 | OVER     | 1.0               | gamma | 1.0   | 0.001904       | 0.983667    | 0.003801 |

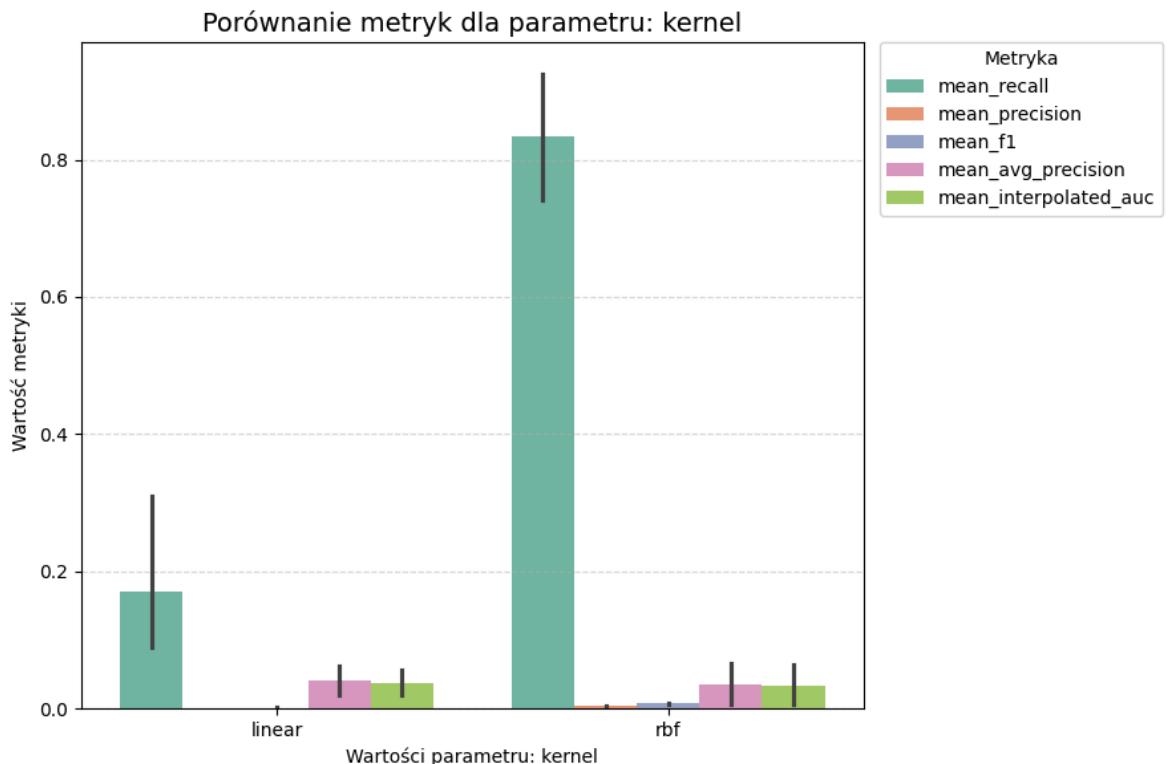
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(oc_svm_params_runtime_tune_results, param_name="ker
```



Wykres dla wyników wcześniejszych otrzymanych:

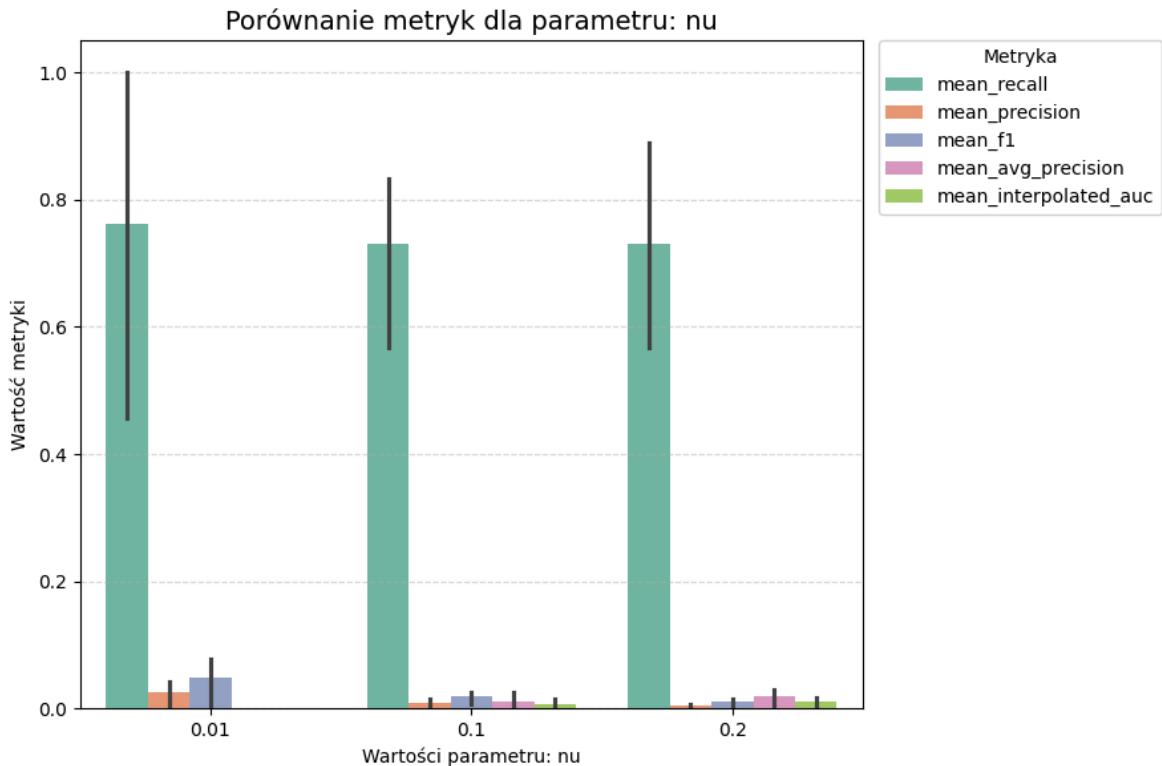
```
In [ ]: plot_metric_groups_for_param(oc_svm_params_full_tune_results, param_name="kernel")
```



Bezapelacyjnie lepszą opcją dla danego modelu jest ustawienie jądra *rbf*. Niestety, martwią dość słabe rezultaty, jeśli chodzi o miarę Precision, co sugeruje, że model będzie miał tendencje do zgłaszania wielu fałszywych alarmów.

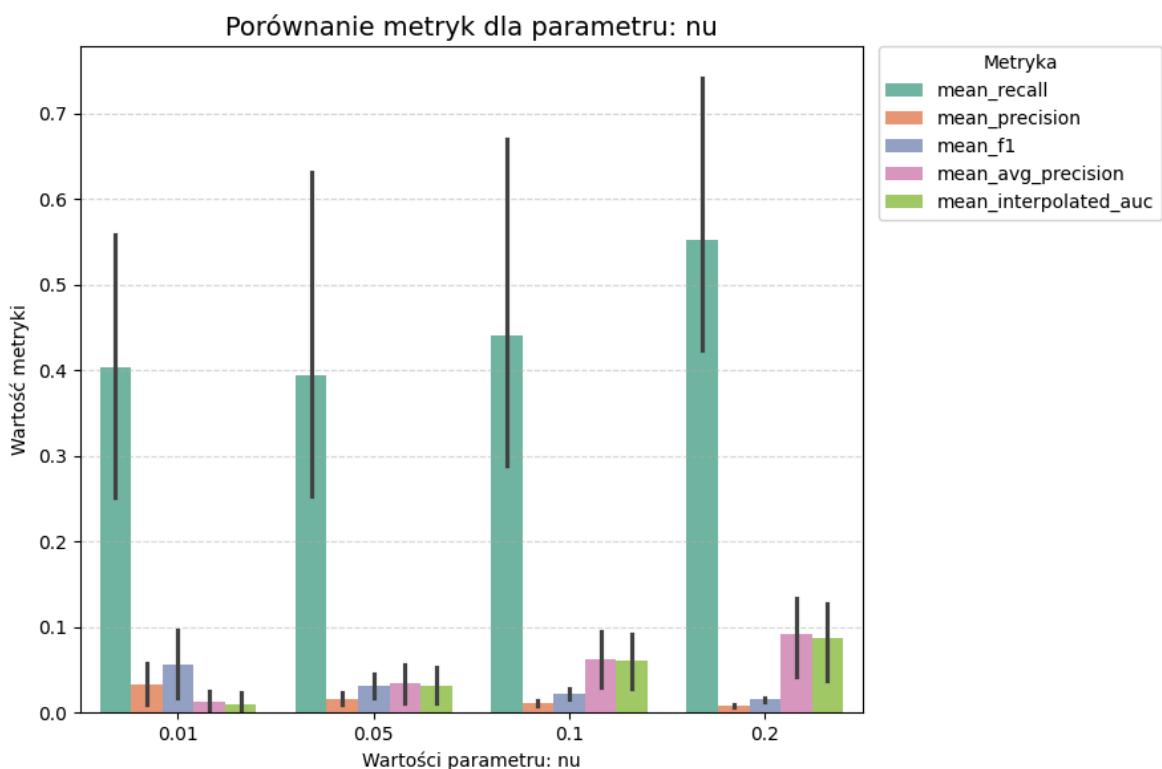
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(oc_svm_params_runtime_tune_results, param_name="nu")
```



Wykres dla wyników wcześniejszy otrzymanych:

```
In [ ]: plot_metric_groups_for_param(oc_svm_params_full_tune_results, param_name="nu", m
```



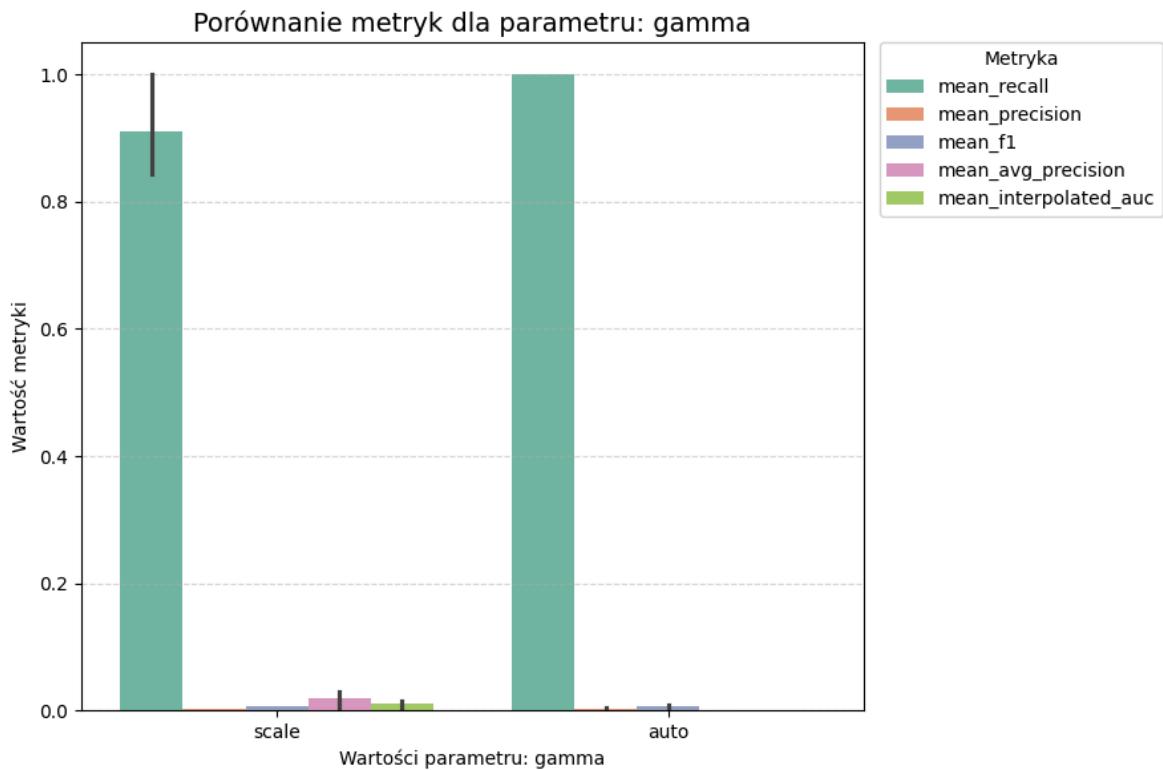
Dla mniejszych zbiorów danych jest preferowana mniejsza wartość  $nu$ , a dla większych już większa wartość. Może mieć to uzasadnienie takie, że w przypadku małych zbiorów osobniki odstające są już w tak małej liczbie, że ich rozpoznanie wymaga postawienia naprawdę cienkiej granicy. Postawienie takiej granicy dla większego zbioru może

przyczynić się do uznawania wielu prawidłowych transakcji za fałszywe, dlatego należy wtedy zwiększyć wartość parametru *nu*.

Niestety, mamy tutaj dość niezadowalające wyniki, zarówno jeśli chodzi o Recall, jak i Precision.

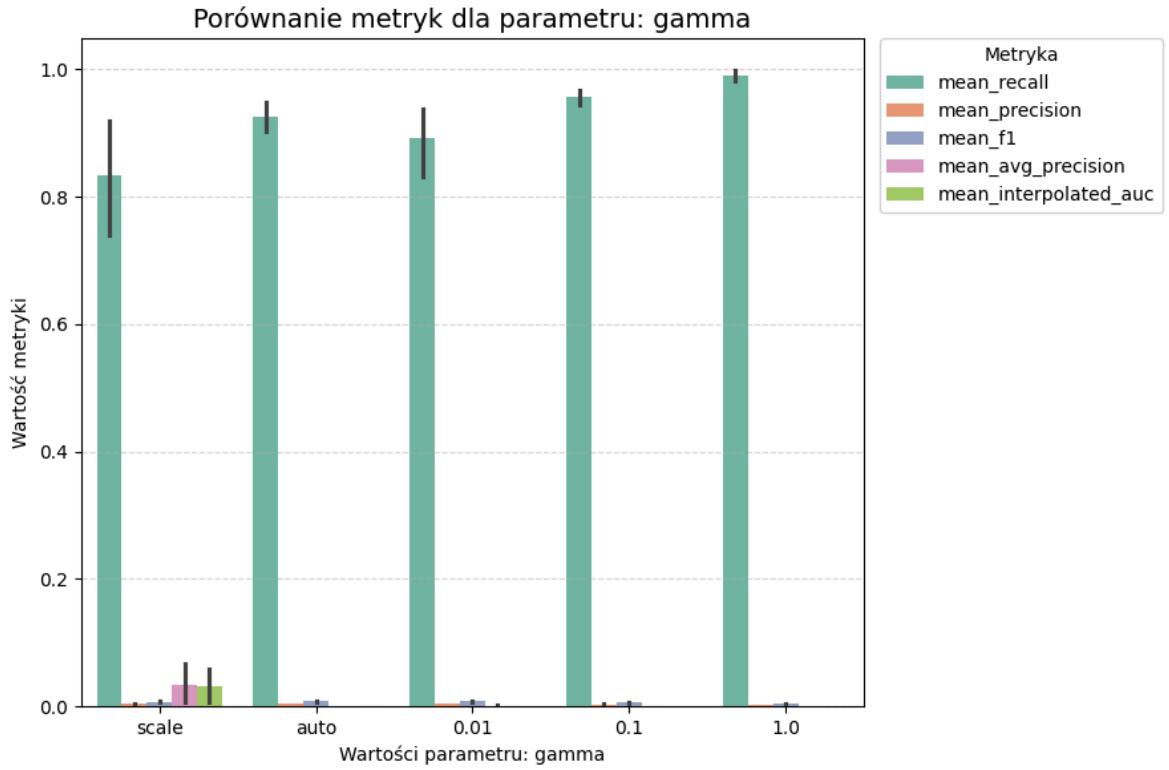
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(oc_svm_params_runtime_tune_results, param_name="gamma")
```



Wykres dla wyników wcześniejszych otrzymanych:

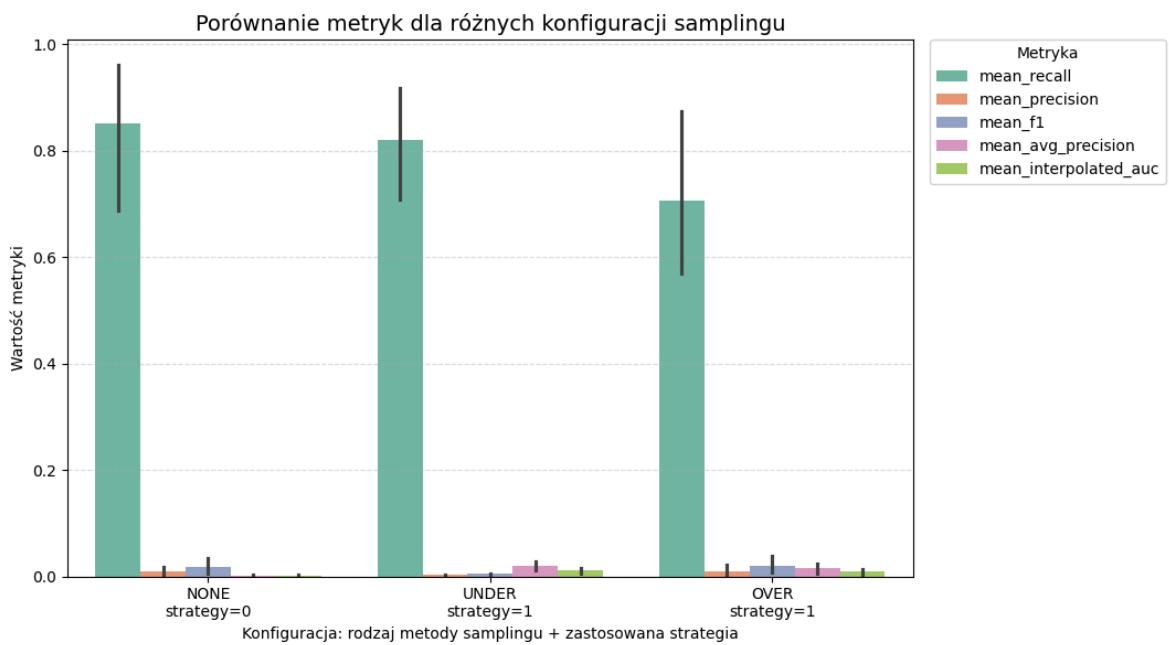
```
In [ ]: plot_metric_groups_for_param(oc_svm_params_full_tune_results, param_name="gamma")
```



W przypadku parametru *gamma*, widać, że jego duża wartość może przyczynić się do prawie idealnego wyłapywania wszelkich anomalii. Niestety, skutkuje to drastycznym pogorszeniem innych miar jakości, dlatego ewentualnie trzeba rozważyć wartość *scale* jako potencjalną opcję ratowniczą dla Precision, choć nie wygląda to obiecująco.

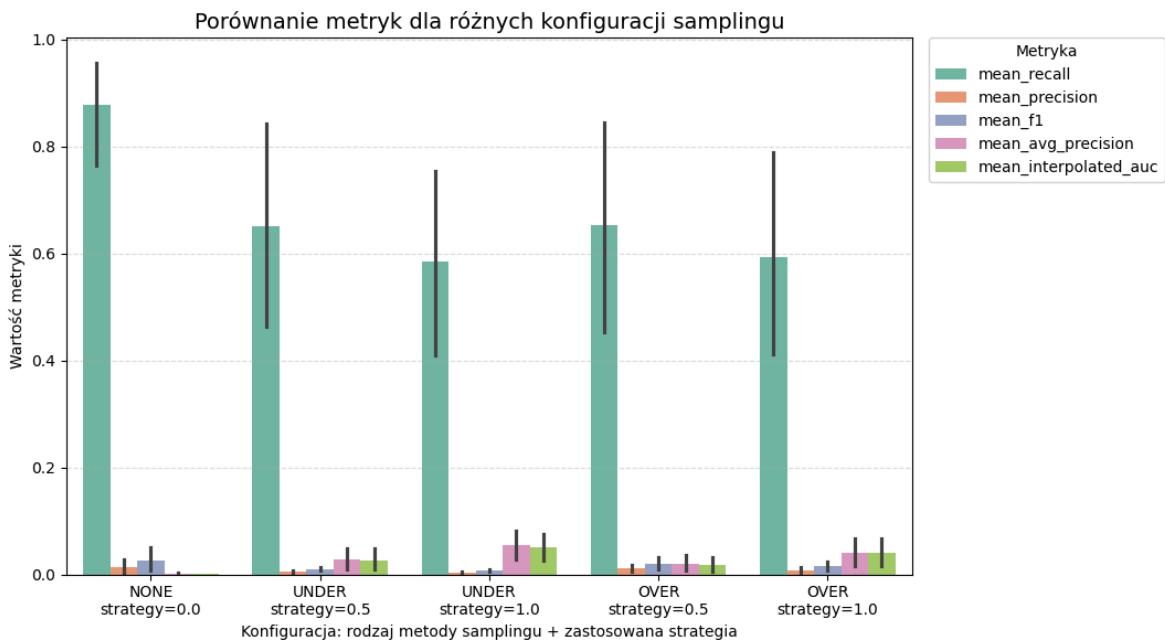
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_sampling(oc_svm_params_runtime_tune_results, metric_names=
```



Wykres dla wyników wcześniejszo otrzymanych:

```
In [ ]: plot_metric_groups_for_sampling(oc_svm_params_full_tune_results, metric_names=[
```



Wygląda, że najwięcej osobników odstających wykryjemy w przypadku zaniechania samplingu. Właściwości algorytmu OC - SVM mogą być bardziej uwidocznione w przypadku, gdy przedstawiciele naszej klasy mniejszościowej są faktycznie osobnikami odstającymi, których jest naprawdę nie wiele, więc nie ma co ingerować w stosunek liczebnościowy naszych dwóch klas.

## Trening i ocena modelu

Budowanie oraz ocena modelu podstawowego dla dobranych parametrów, z uwzględnieniem aktualnie załadowanego zbioru treningowego oraz testowego

```
In [ ]: oc_svm_best_params = {
    "kernel": "rbf",
    "gamma": 1.0,
    "nu": 0.2,
    "cache_size": 8192,
}
oc_svm_sampling_method = SamplingMethod.NONE
oc_svm_sampling_strategy = 0.0

oc_svm_runtime_model_evaluation_result = train_and_evaluate_best_model(train_df,
build_oc_
oc_svm_mo
oc_svm_sa
unsupervi
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.001989  | 1.0    | 0.00397  | 0.001275       | 0.000888               |

Wariant modelu z treningiem bez osobników odstających

```
In [ ]: oc_svm_filtered_train_df_best_params = {
    "kernel": "rbf",
    "gamma": 1.0,
    "nu": 0.2,
    "cache_size": 8192,
```

```

}

oc_svm_filtered_train_df_sampling_method = SamplingMethod.NONE
oc_svm_filtered_train_df_sampling_strategy = 0.0

oc_svm_runtime_model_filtered_train_df_evaluation_result = train_and_evaluate_be
test_df,
oc_svm_fil
build_oc_s
oc_svm_fil
oc_svm_run
oc_svm_fil
oc_svm_fil
unsupervis

```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.001988  | 1.0    | 0.003968 | 0.001482       | 0.000884               |

Załadowanie wcześniej przygotowanego modelu, który został wytrenowany na zbiorze treningowym stanowiącym 20% zbioru treningowego pierwotnego.

```
In [ ]: if (LOAD_PRETRAINED_MODEL and os.path.exists(oc_svm_pretrained_model_file_path))
    oc_svm_full_model = joblib.load(oc_svm_pretrained_model_file_path)

    x_test = test_df.drop(columns=["Class"])
    y_test = test_df["Class"]

    oc_svm_full_model_evaluation_result = evaluate_model(oc_svm_full_model, x_test
  predict_positive_class_la
  predict_proba=True)

    best_results_df = pd.concat([best_results_df, pd.DataFrame([["OC-SVM", oc_svm_
   full_model
   full_model
   full_model
   full_model
   full_model
   columns=best_result
   ]])])

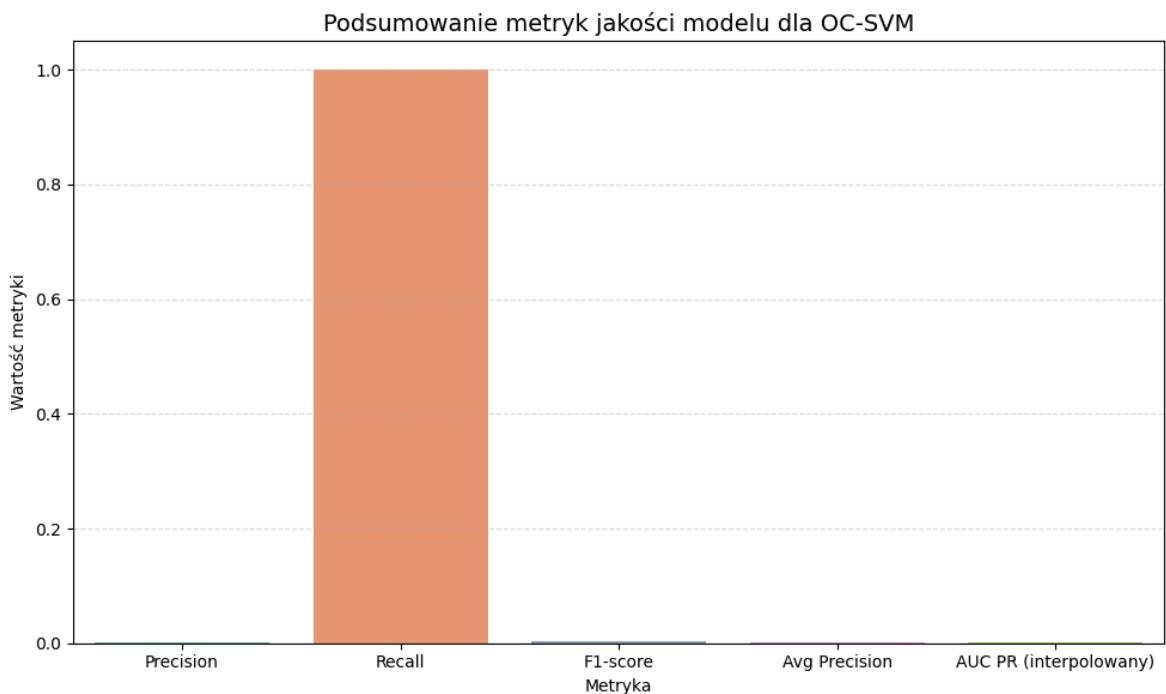
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.004931  | 1.0    | 0.009814 | 0.00148        | 0.000881               |

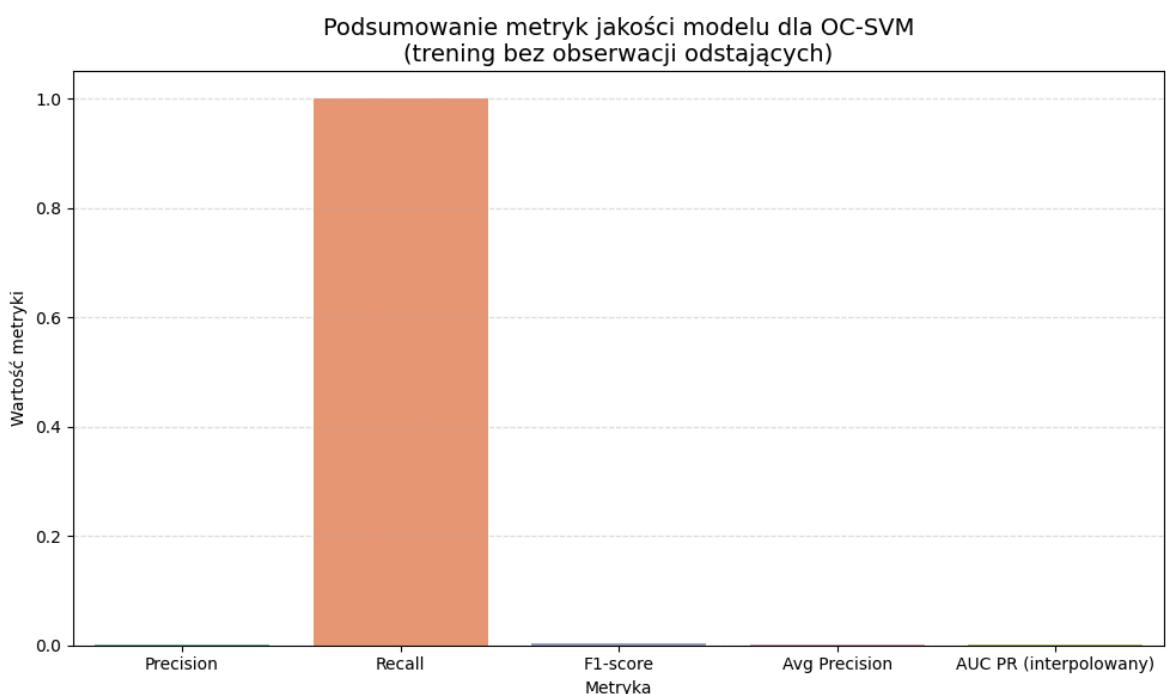
Podstawowe miary jakości dla modelu podstawowego

```
In [ ]: if oc_svm_runtime_model_evaluation_result is not None:
    plot_metric_summary(
        metric_values=oc_svm_runtime_model_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla OC-SVM"
    )
else:
    print("Brak wyniku ewaluacji dla modelu w runtime")
```



Podstawowe miary jakości dla modelu z treningiem bez osobników odstających

```
In [ ]: if oc_svm_runtime_model_filtered_train_df_evaluation_result is not None:
    plot_metric_summary(
        metric_values=oc_svm_runtime_model_filtered_train_df_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla OC-SVM\n(trening bez obserwacji odstających)")
else:
    print("Brak wyniku ewaluacji dla modelu w runtime (przypadek treningu bez obserwacji odstających)")
```



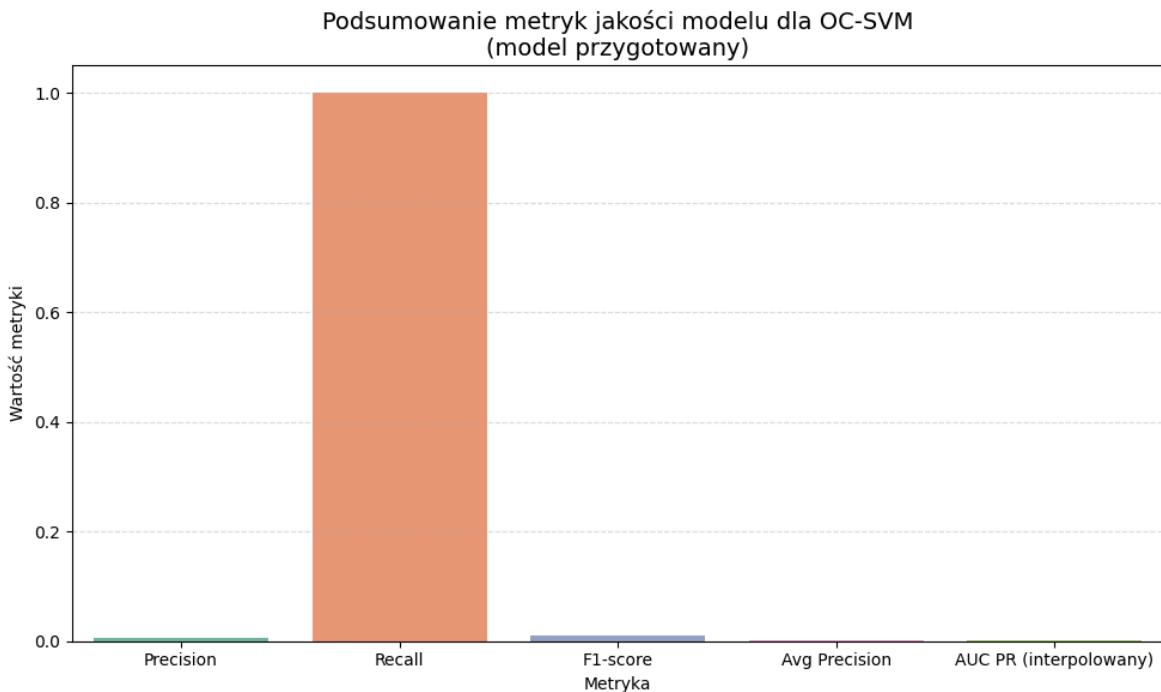
Podstawowe miary jakości dla modelu wcześniejszej przygotowanego

```
In [ ]: if oc_svm_full_model_evaluation_result is not None:
    plot_metric_summary(
        metric_values=oc_svm_full_model_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla OC-SVM\n(model przygotowany)")
```

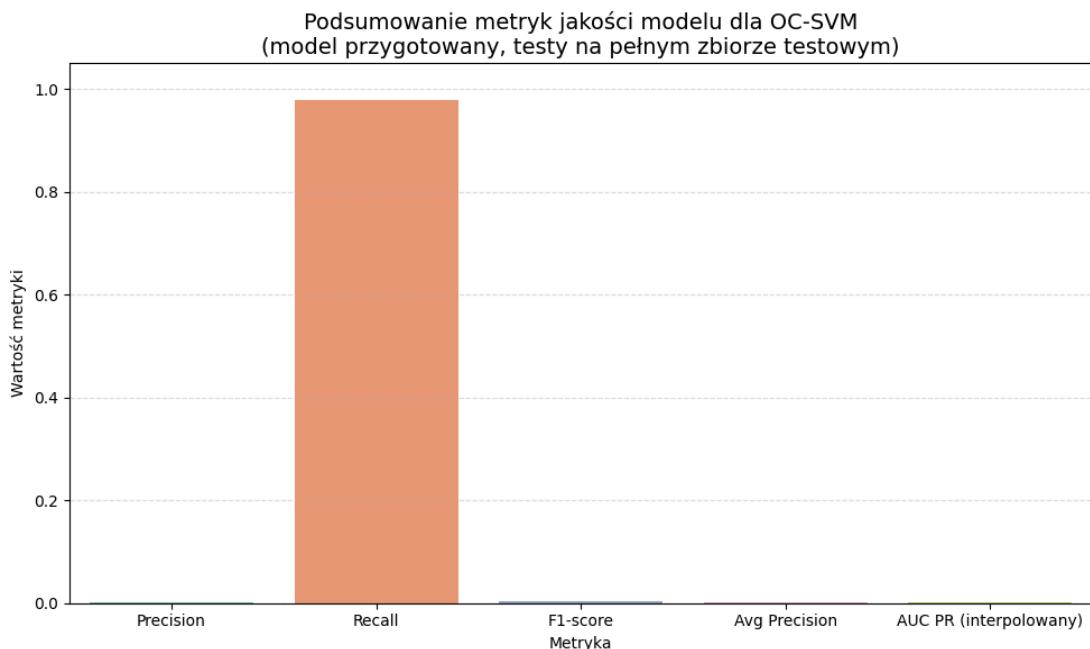
```

    )
else:
    print("Brak wyniku ewaluacji dla modelu wcześniej przygotowanego")

```



Podstawowe miary jakości dla modelu wcześniejszej przygotowanego, określające wynik testów na pełnym zbiorze testowym (zapisany wykres po wcześniejszej wykonanej ewaluacji)



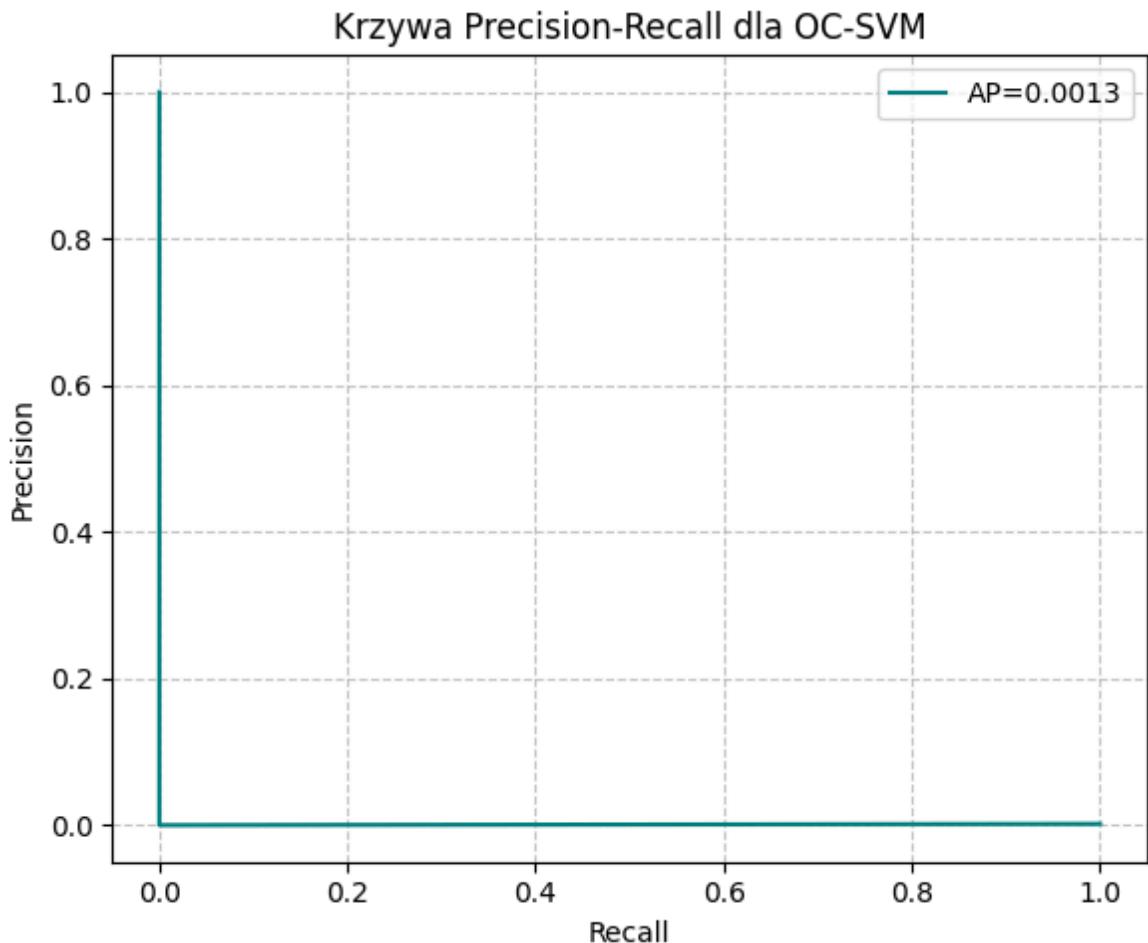
Krzywa PR dla modelu podstawowego

```

In [ ]: if oc_svm_runtime_model_evaluation_result is not None:
    plot_pr_curve(
        recall_vals=oc_svm_runtime_model_evaluation_result["recall_vals"],
        precision_vals=oc_svm_runtime_model_evaluation_result["precision_vals"],
        avg_precision=oc_svm_runtime_model_evaluation_result["avg_precision"],
        plot_title="Krzywa Precision-Recall dla OC-SVM"
    )

```

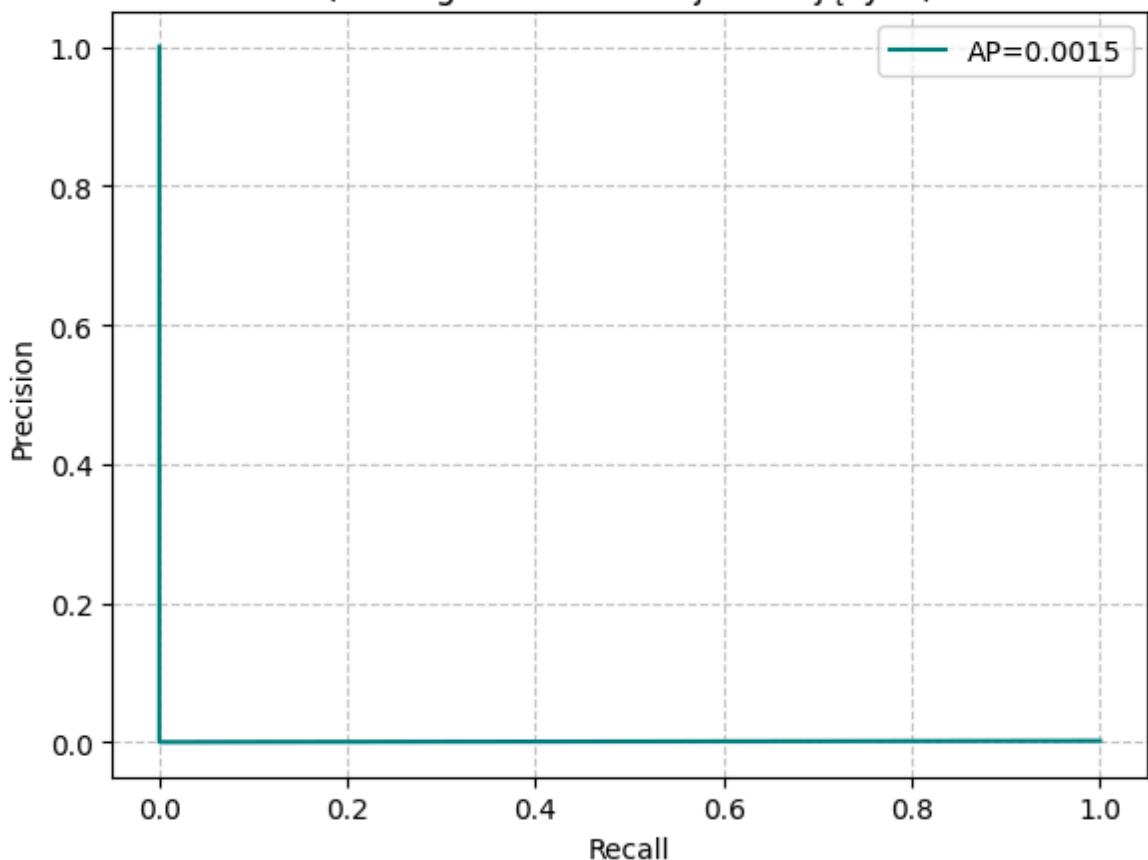
```
else:  
    print("Brak wyniku ewaluacji dla modelu w runtime")
```



Krzywa PR dla modelu z treningiem bez osobników odstających

```
In [ ]: if oc_svm_runtime_model_filtered_train_df_evaluation_result is not None:  
    plot_pr_curve()  
    recall_vals=oc_svm_runtime_model_filtered_train_df_evaluation_result["recall"]  
    precision_vals=oc_svm_runtime_model_filtered_train_df_evaluation_result["pre  
    avg_precision=oc_svm_runtime_model_filtered_train_df_evaluation_result["avg_  
    plot_title="Krzywa Precision-Recall dla OC-SVM\n(trening bez obserwacji odst  
    )  
else:  
    print("Brak wyniku ewaluacji dla modelu w runtime (przypadek treningu bez obse
```

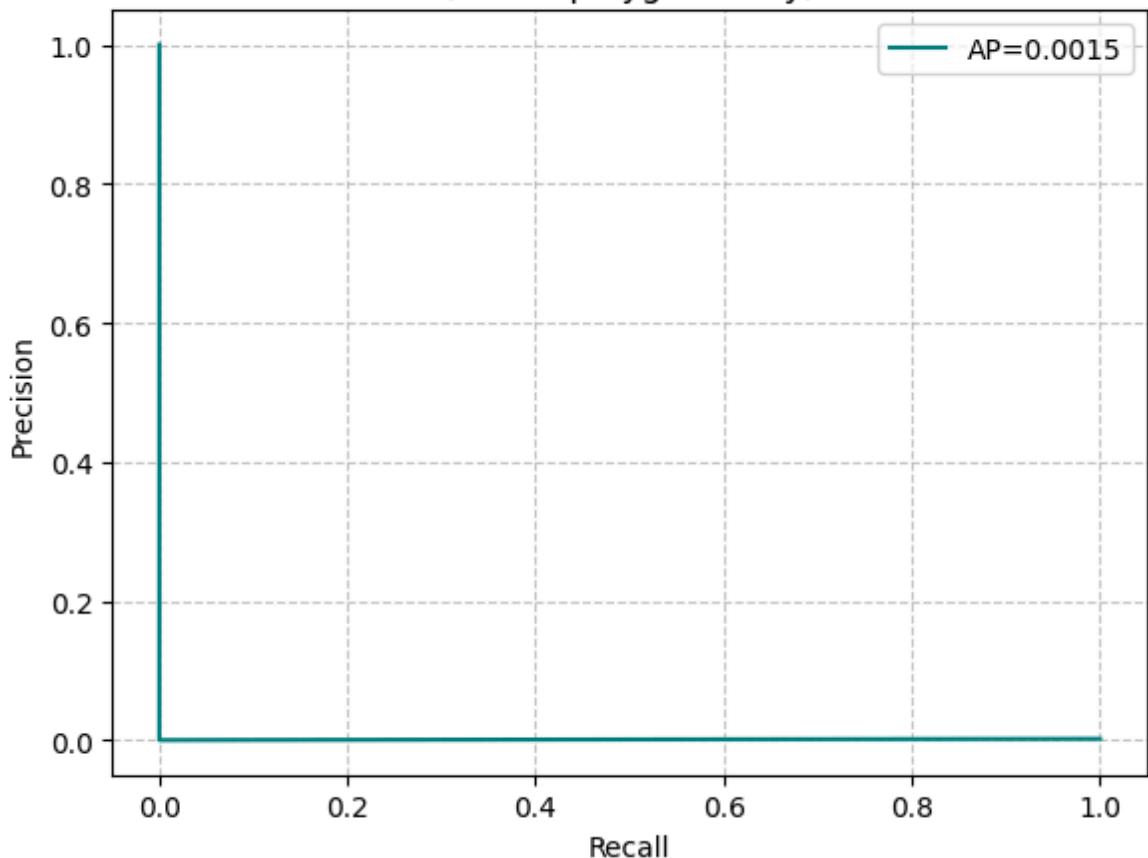
### Krzywa Precision-Recall dla OC-SVM (trening bez obserwacji odstających)



Krzywa PR dla modelu wcześniejszy przygotowanego

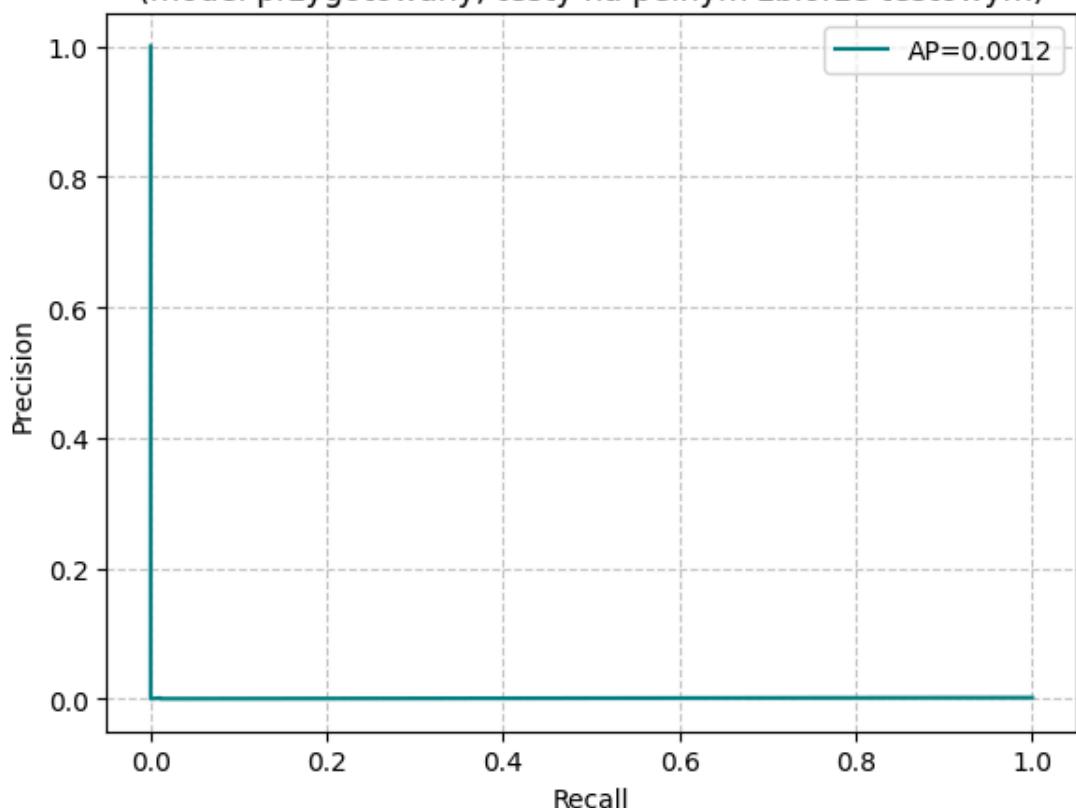
```
In [ ]: if oc_svm_full_model_evaluation_result is not None:  
    plot_pr_curve(  
        recall_vals=oc_svm_full_model_evaluation_result["recall_vals"],  
        precision_vals=oc_svm_full_model_evaluation_result["precision_vals"],  
        avg_precision=oc_svm_full_model_evaluation_result["avg_precision"],  
        plot_title="Krzywa Precision-Recall dla OC-SVM\n(model przygotowany)"  
    )  
else:  
    print("Brak wyniku ewaluacji dla modelu wcześniejszy przygotowanego")
```

### Krzywa Precision-Recall dla OC-SVM (model przygotowany)



Krzywa PR dla modelu wcześniej przygotowanego, określająca wynik testów na pełnym zbiorze testowym (zapisany wykres po wcześniej wykonanej ewaluacji)

### Krzywa Precision-Recall dla OC-SVM (model przygotowany, testy na pełnym zbiorze testowym)



Niestety, nie udało nam się tutaj zbalansować odpowiednio wartości miar Recall oraz Precision. Model zbudowany na wyłącznie przykładach z klasy większościowej jest porównywalny z modelem, którego trening odbył się z zachowaniem klasy mniejszościowej z rozkładem zgodnym ze zbiorem pierwotnym. Jednak, w ostateczności udało nam się najlepszy model zbudować w oparciu o pozbycie się tych przykładów odstających i to jest nasz model wcześniejszy przygotowany.

Warto także dodać, że pomijając fakt, że trening modelu OC-SVM na większych zbiorach traci sens, ze względów czasowych, to także ostatecznie dostawaliśmy gorsze rezultaty, jak próbowaliśmy to robić. Możliwe, że był wprowadzany w ten sposób niepotrzebny szum.

## Las izolacyjny

Przy budowaniu modelu opartego na algorytmie lasu izolacyjnego, skupimy się na następujących parametrach:

- *n\_estimators* - liczba drzew (izolujących),
- *max\_samples* - liczba próbek do stworzenia każdego drzewa (kontrola losowości),
- *contamination* - procent danych, który traktowany jest jako anomalia.

```
In [ ]: from sklearn.ensemble import IsolationForest
def build_isolation_forest_model_function(**kwargs):
    return IsolationForest(**kwargs)
```

## Proces strojenia parametrów i badań ich wpływu na jakość modelu

Poniżej następuje uruchomienie całego procesu strojenia parametrów, wraz z prezentacją wyników w formie tabularycznej.

```
In [ ]: import os

if QUICK_MODE:
    isolation_forest_params_to_test = [
        ParameterToTest(name="n_estimators", values=[50, 100, 300]),
        ParameterToTest(name="max_samples", values=['auto', 128, 512]),
        ParameterToTest(name="contamination", values=[0.0001, 0.001, 0.01]),
    ]

    isolation_forest_sampling_method_to_test = [
        SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data_s
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data_
        SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, data_s
    ]
else:
    isolation_forest_params_to_test = [
        ParameterToTest(name="n_estimators", values=[50, 100, 200, 300]),
        ParameterToTest(name="max_samples", values=['auto', 128, 256, 512]),
        ParameterToTest(name="contamination", values=[0.0001, 0.001, 0.005, 0.01
    ]

    isolation_forest_sampling_method_to_test = [
```

```
SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, da
SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=0.5,
                     data_splits=undersampling_50_data_splits),
SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, d
SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=0.5,
                     data_splits=oversampling_50_data_splits),
SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, da
]

isolation_forest_model_specific_params = {
    "random_state": SEED,
    "n_jobs": -1}

if (QUICK_MODE and os.path.exists(isolation_forest_tune_results_tmp_file_path)):
    isolation_forest_params_runtime_tune_results = pd.read_csv(isolation_forest_tu
else:
    isolation_forest_params_runtime_tune_results = params_tune(isolation_forest_pa
   build_model_function=build_isolation_fo
   model_specific_params=isolation_forest_
display(isolation_forest_params_runtime_tune_results)
```

|    | sampling | sampling_strategy | param         | value  | mean_precision | mean_recall | mean_f1 |
|----|----------|-------------------|---------------|--------|----------------|-------------|---------|
| 0  | NONE     | 0                 | n_estimators  | 50     | 0.041450       | 0.833333    | 0.07    |
| 1  | NONE     | 0                 | n_estimators  | 100    | 0.040948       | 0.833333    | 0.07    |
| 2  | NONE     | 0                 | n_estimators  | 300    | 0.039598       | 0.833333    | 0.07    |
| 3  | UNDER    | 1                 | n_estimators  | 50     | 0.057306       | 0.516667    | 0.10    |
| 4  | UNDER    | 1                 | n_estimators  | 100    | 0.048346       | 0.405556    | 0.08    |
| 5  | UNDER    | 1                 | n_estimators  | 300    | 0.038165       | 0.405556    | 0.06    |
| 6  | OVER     | 1                 | n_estimators  | 50     | 0.028048       | 0.516667    | 0.05    |
| 7  | OVER     | 1                 | n_estimators  | 100    | 0.022052       | 0.461111    | 0.04    |
| 8  | OVER     | 1                 | n_estimators  | 300    | 0.023402       | 0.461111    | 0.04    |
| 9  | NONE     | 0                 | max_samples   | auto   | 0.040948       | 0.833333    | 0.07    |
| 10 | NONE     | 0                 | max_samples   | 128    | 0.029536       | 0.833333    | 0.05    |
| 11 | NONE     | 0                 | max_samples   | 512    | 0.048856       | 0.777778    | 0.09    |
| 12 | UNDER    | 1                 | max_samples   | auto   | 0.048346       | 0.405556    | 0.08    |
| 13 | UNDER    | 1                 | max_samples   | 128    | 0.048346       | 0.405556    | 0.08    |
| 14 | UNDER    | 1                 | max_samples   | 512    | 0.048346       | 0.405556    | 0.08    |
| 15 | OVER     | 1                 | max_samples   | auto   | 0.022052       | 0.461111    | 0.04    |
| 16 | OVER     | 1                 | max_samples   | 128    | 0.020972       | 0.461111    | 0.04    |
| 17 | OVER     | 1                 | max_samples   | 512    | 0.030525       | 0.461111    | 0.05    |
| 18 | NONE     | 0                 | contamination | 0.0001 | 0.500000       | 0.100000    | 0.16    |
| 19 | NONE     | 0                 | contamination | 0.001  | 0.275641       | 0.305556    | 0.28    |
| 20 | NONE     | 0                 | contamination | 0.01   | 0.127350       | 0.677778    | 0.21    |
| 21 | UNDER    | 1                 | contamination | 0.0001 | 0.000000       | 0.000000    | 0.00    |
| 22 | UNDER    | 1                 | contamination | 0.001  | 0.000000       | 0.000000    | 0.00    |
| 23 | UNDER    | 1                 | contamination | 0.01   | 0.000000       | 0.000000    | 0.00    |
| 24 | OVER     | 1                 | contamination | 0.0001 | 0.000000       | 0.000000    | 0.00    |
| 25 | OVER     | 1                 | contamination | 0.001  | 0.000000       | 0.000000    | 0.00    |
| 26 | OVER     | 1                 | contamination | 0.01   | 0.176282       | 0.205556    | 0.17    |

Tutaj dla porównania zostają wyświetlane wcześniejsze uzyskane wyniki, które zostały zapisane do odpowiedniego pliku. Proces strojenia parametrów odbywał się na całym zbiorze danych.

```
In [ ]: if (LOAD_PARAMS_TUNE_RESULTS_FROM_MASTER_FILE and os.path.exists(isolation_forest_params_full_tune_results = pd.read_csv(isolation_forest_tune_
```

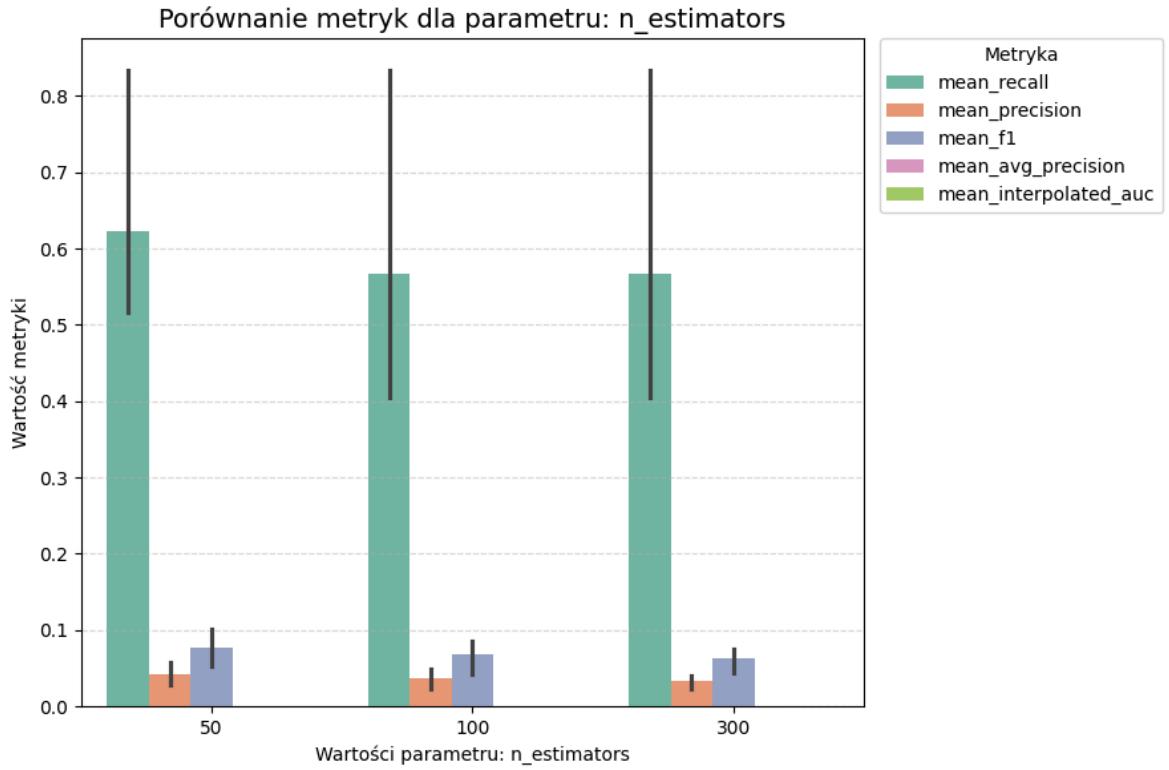
```
display(isolation_forest_params_full_tune_results)
```

|    | sampling | sampling_strategy | param | value        | mean_precision | mean_recall | mean_f1  |
|----|----------|-------------------|-------|--------------|----------------|-------------|----------|
| 0  | NONE     |                   | 0.0   | n_estimators | 50             | 0.033195    | 0.838561 |
| 1  | NONE     |                   | 0.0   | n_estimators | 100            | 0.035770    | 0.835930 |
| 2  | NONE     |                   | 0.0   | n_estimators | 200            | 0.035363    | 0.822667 |
| 3  | NONE     |                   | 0.0   | n_estimators | 300            | 0.036282    | 0.822737 |
| 4  | UNDER    |                   | 0.5   | n_estimators | 50             | 0.038243    | 0.256772 |
| 5  | UNDER    |                   | 0.5   | n_estimators | 100            | 0.040503    | 0.240877 |
| 6  | UNDER    |                   | 0.5   | n_estimators | 200            | 0.037944    | 0.243509 |
| 7  | UNDER    |                   | 0.5   | n_estimators | 300            | 0.038848    | 0.240842 |
| 8  | UNDER    |                   | 1.0   | n_estimators | 50             | 0.036885    | 0.232947 |
| 9  | UNDER    |                   | 1.0   | n_estimators | 100            | 0.039849    | 0.225053 |
| 10 | UNDER    |                   | 1.0   | n_estimators | 200            | 0.036999    | 0.222386 |
| 11 | UNDER    |                   | 1.0   | n_estimators | 300            | 0.038147    | 0.219754 |
| 12 | OVER     |                   | 0.5   | n_estimators | 50             | 0.035682    | 0.309439 |
| 13 | OVER     |                   | 0.5   | n_estimators | 100            | 0.036630    | 0.277825 |
| 14 | OVER     |                   | 0.5   | n_estimators | 200            | 0.033429    | 0.267228 |
| 15 | OVER     |                   | 0.5   | n_estimators | 300            | 0.033530    | 0.272561 |
| 16 | OVER     |                   | 1.0   | n_estimators | 50             | 0.028203    | 0.248772 |
| 17 | OVER     |                   | 1.0   | n_estimators | 100            | 0.031568    | 0.246140 |
| 18 | OVER     |                   | 1.0   | n_estimators | 200            | 0.028937    | 0.243474 |
| 19 | OVER     |                   | 1.0   | n_estimators | 300            | 0.029674    | 0.246105 |
| 20 | NONE     |                   | 0.0   | max_samples  | auto           | 0.035770    | 0.835930 |
| 21 | NONE     |                   | 0.0   | max_samples  | 128            | 0.027119    | 0.849193 |
| 22 | NONE     |                   | 0.0   | max_samples  | 256            | 0.035770    | 0.835930 |
| 23 | NONE     |                   | 0.0   | max_samples  | 512            | 0.048546    | 0.796140 |
| 24 | UNDER    |                   | 0.5   | max_samples  | auto           | 0.040503    | 0.240877 |
| 25 | UNDER    |                   | 0.5   | max_samples  | 128            | 0.035276    | 0.280526 |
| 26 | UNDER    |                   | 0.5   | max_samples  | 256            | 0.040503    | 0.240877 |
| 27 | UNDER    |                   | 0.5   | max_samples  | 512            | 0.046729    | 0.227684 |
| 28 | UNDER    |                   | 1.0   | max_samples  | auto           | 0.039849    | 0.225053 |
| 29 | UNDER    |                   | 1.0   | max_samples  | 128            | 0.036427    | 0.246175 |
| 30 | UNDER    |                   | 1.0   | max_samples  | 256            | 0.039849    | 0.225053 |
| 31 | UNDER    |                   | 1.0   | max_samples  | 512            | 0.040715    | 0.188035 |
| 32 | OVER     |                   | 0.5   | max_samples  | auto           | 0.036630    | 0.277825 |

|    | sampling | sampling_strategy | param         | value  | mean_precision | mean_recall | mean_f1 |
|----|----------|-------------------|---------------|--------|----------------|-------------|---------|
| 33 | OVER     | 0.5               | max_samples   | 128    | 0.037344       | 0.367825    | 0.06    |
| 34 | OVER     | 0.5               | max_samples   | 256    | 0.036630       | 0.277825    | 0.06    |
| 35 | OVER     | 0.5               | max_samples   | 512    | 0.038088       | 0.251439    | 0.06    |
| 36 | OVER     | 1.0               | max_samples   | auto   | 0.031568       | 0.246140    | 0.05    |
| 37 | OVER     | 1.0               | max_samples   | 128    | 0.030223       | 0.259368    | 0.05    |
| 38 | OVER     | 1.0               | max_samples   | 256    | 0.031568       | 0.246140    | 0.05    |
| 39 | OVER     | 1.0               | max_samples   | 512    | 0.032201       | 0.235544    | 0.05    |
| 40 | NONE     | 0.0               | contamination | 0.0001 | 0.123016       | 0.010596    | 0.01    |
| 41 | NONE     | 0.0               | contamination | 0.001  | 0.346202       | 0.209228    | 0.26    |
| 42 | NONE     | 0.0               | contamination | 0.005  | 0.156122       | 0.463018    | 0.23    |
| 43 | NONE     | 0.0               | contamination | 0.01   | 0.104677       | 0.616456    | 0.17    |
| 44 | UNDER    | 0.5               | contamination | 0.0001 | 0.200000       | 0.002667    | 0.00    |
| 45 | UNDER    | 0.5               | contamination | 0.001  | 0.200000       | 0.002667    | 0.00    |
| 46 | UNDER    | 0.5               | contamination | 0.005  | 0.115238       | 0.013228    | 0.02    |
| 47 | UNDER    | 0.5               | contamination | 0.01   | 0.102525       | 0.026456    | 0.04    |
| 48 | UNDER    | 1.0               | contamination | 0.0001 | 0.200000       | 0.002667    | 0.00    |
| 49 | UNDER    | 1.0               | contamination | 0.001  | 0.400000       | 0.005298    | 0.01    |
| 50 | UNDER    | 1.0               | contamination | 0.005  | 0.103896       | 0.013228    | 0.02    |
| 51 | UNDER    | 1.0               | contamination | 0.01   | 0.071439       | 0.018561    | 0.02    |
| 52 | OVER     | 0.5               | contamination | 0.0001 | 0.000000       | 0.000000    | 0.00    |
| 53 | OVER     | 0.5               | contamination | 0.001  | 0.400000       | 0.005298    | 0.01    |
| 54 | OVER     | 0.5               | contamination | 0.005  | 0.131307       | 0.018561    | 0.03    |
| 55 | OVER     | 0.5               | contamination | 0.01   | 0.092312       | 0.039719    | 0.05    |
| 56 | OVER     | 1.0               | contamination | 0.0001 | 0.000000       | 0.000000    | 0.00    |
| 57 | OVER     | 1.0               | contamination | 0.001  | 0.400000       | 0.005298    | 0.01    |
| 58 | OVER     | 1.0               | contamination | 0.005  | 0.101991       | 0.013228    | 0.02    |
| 59 | OVER     | 1.0               | contamination | 0.01   | 0.064458       | 0.023825    | 0.03    |

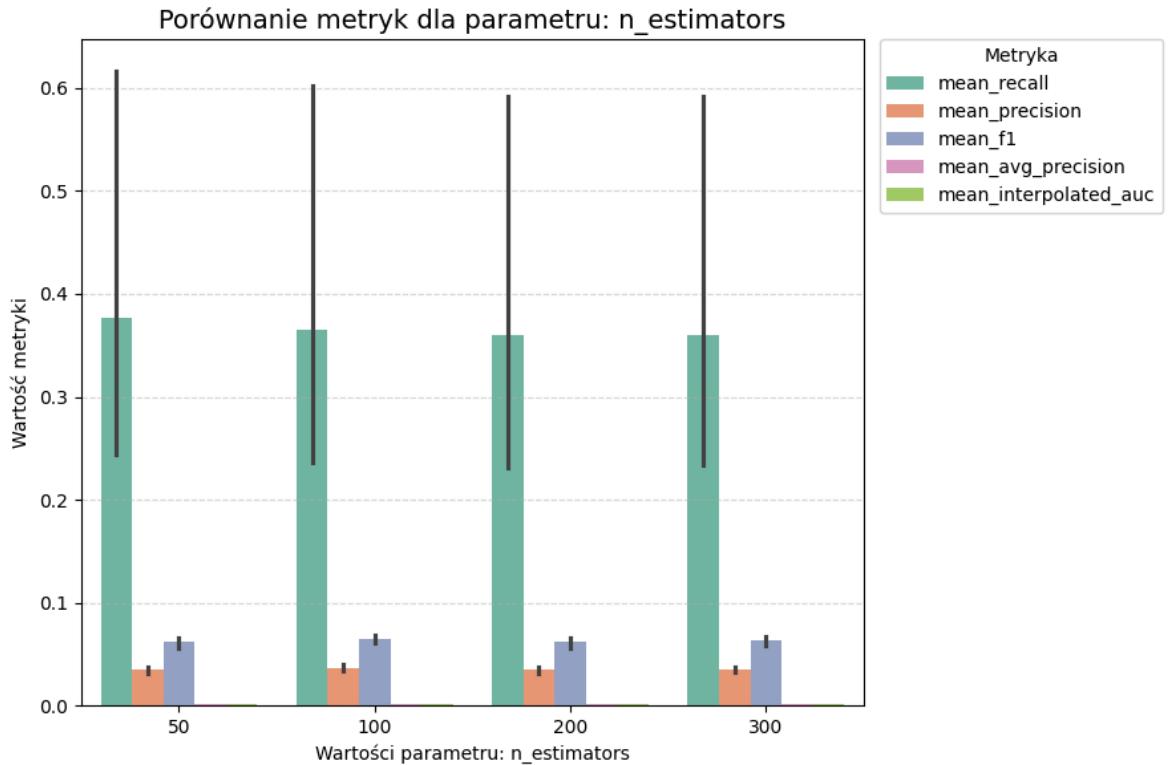
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(isolation_forest_params_runtime_tune_results, param='contamination')
```



Wykres dla wyników wcześniejszych otrzymanych:

```
In [ ]: plot_metric_groups_for_param(isolation_forest_params_full_tune_results, param_na
```

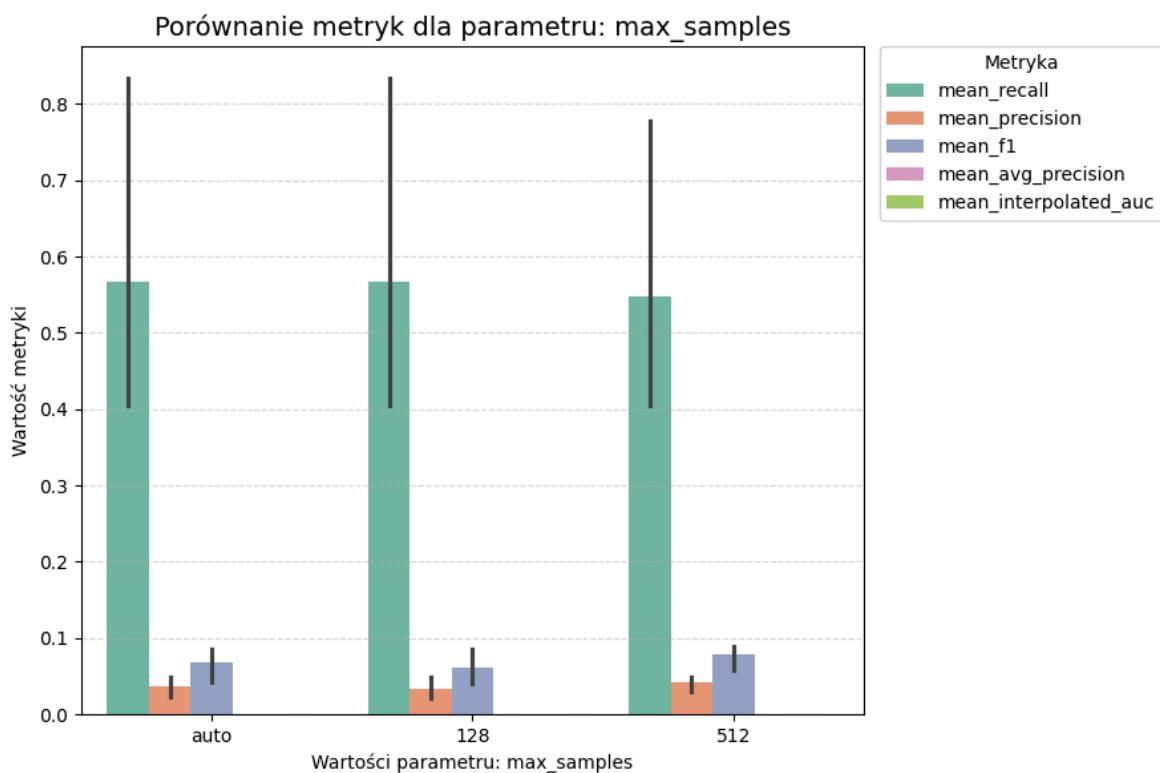


Uzyskaliśmy bardzo podobne wyniki dla różnej liczby drzew, ze wskazaniem na mniejsze wartości. Dzięki ustaleniu mniejszych wartości dla parametru *n\_estimators*, uzyskamy także mniej złożone rozwiązanie pod kątem obliczeniowym.

Niestety, wartości Recall i Precision są niezadawalające, choć mamy ciut lepszy balans niż w przypadku OC-SVM.

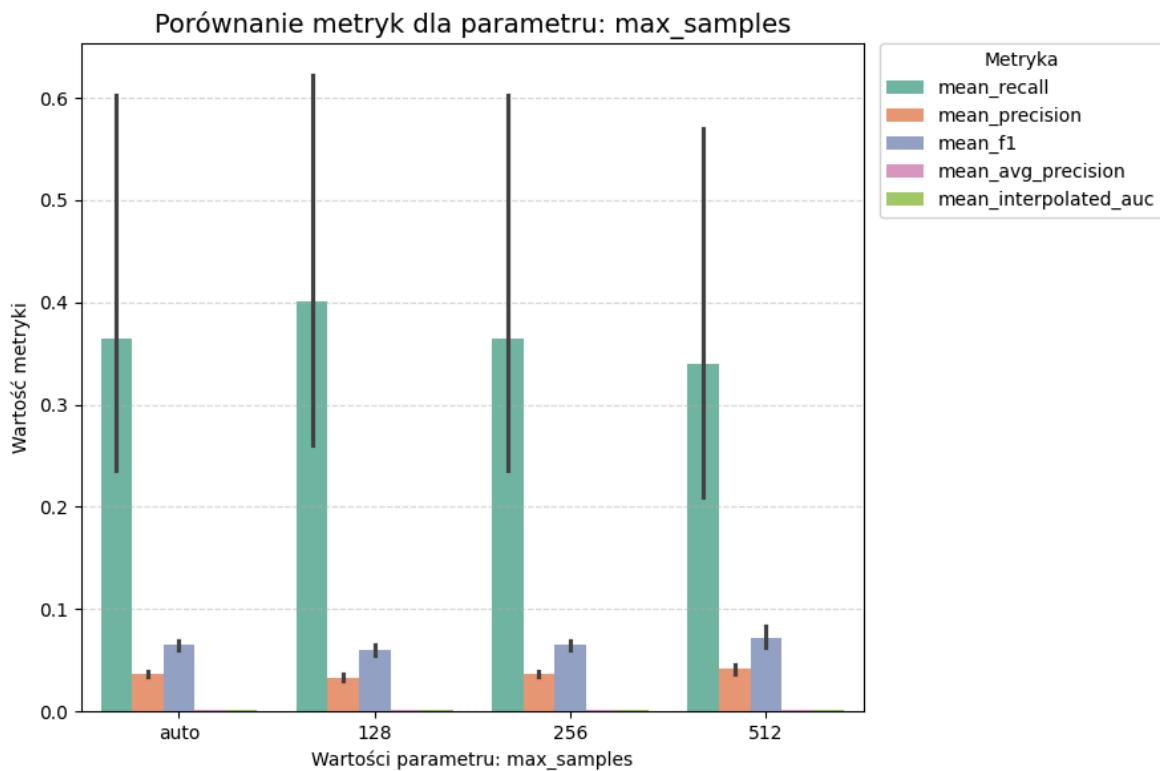
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(isolation_forest_params_runtime_tune_results, param
```



Wykres dla wyników wcześniejszych otrzymanych:

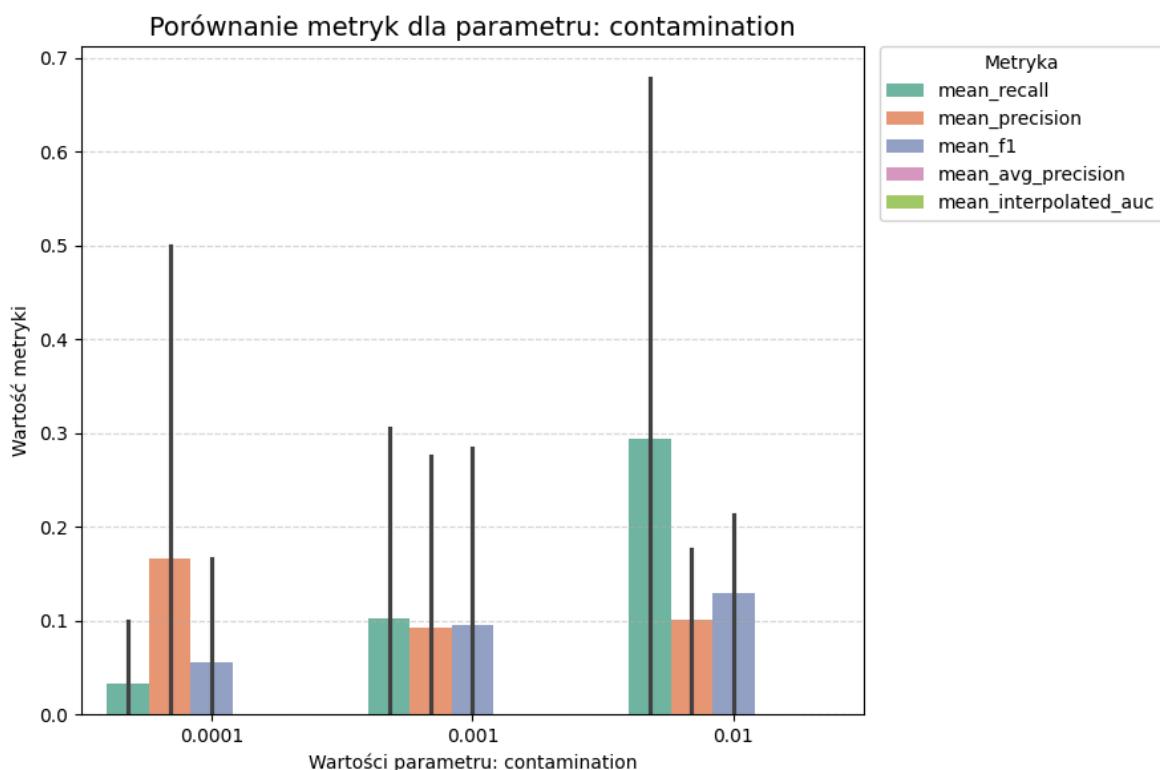
```
In [ ]: plot_metric_groups_for_param(isolation_forest_params_full_tune_results, param_na
```



Wartość 128 wydaje się najodpowiedniejsza oraz daje nadzieję na uzyskanie ciut większego balansu między Recall a Precision.

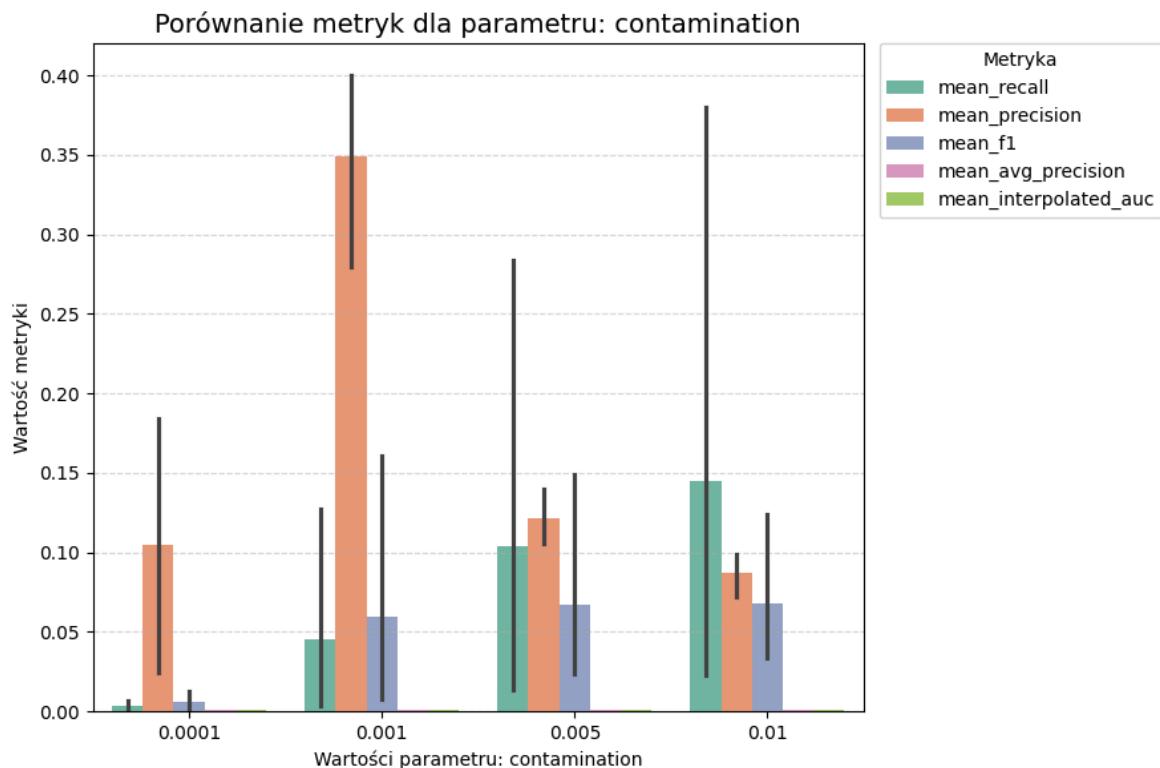
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(isolation_forest_params_runtime_tune_results, param
```



Wykres dla wyników wcześniejszych otrzymanych:

```
In [ ]: plot_metric_groups_for_param(isolation_forest_params_full_tune_results, param_na
```

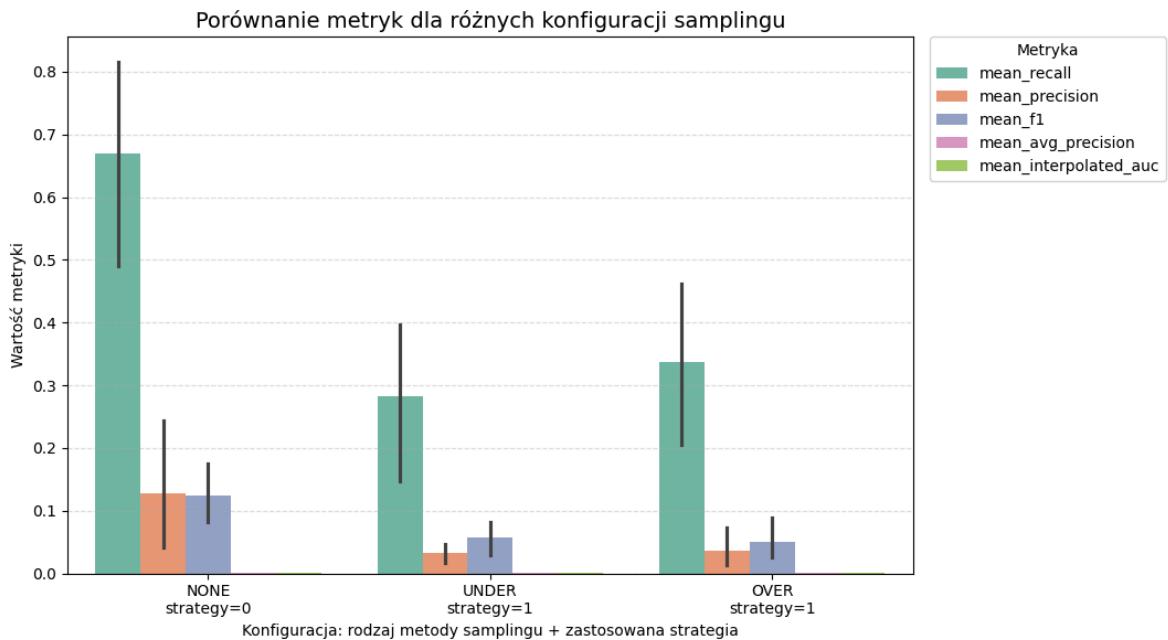


W tym miejscu warto przypomnieć, że oszustwa stanowią w danym zbiorze 0.172% wszystkich transakcji i właśnie tą wartość powinniśmy brać pod uwagę przy ustalaniu

wartości dla parametru *contamination*. Wygląda, że jeśli będziemy oscylować w zakresie 0.001 - 0.005, to możemy pomóc algorytmowi w uzyskaniu balansu między Precision a Recall.

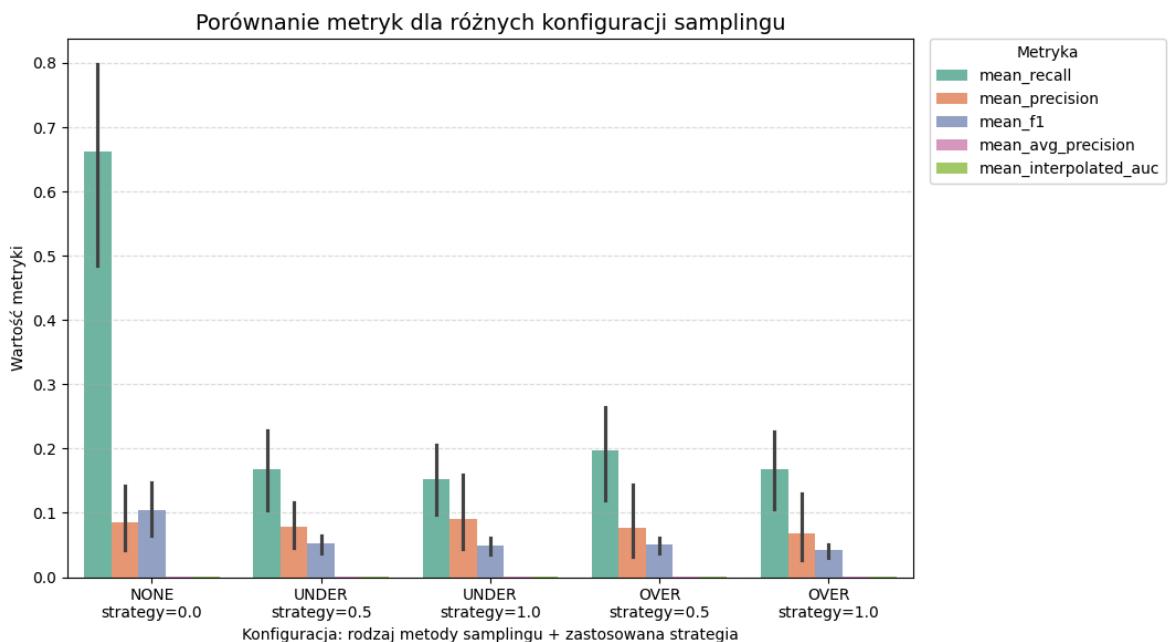
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_sampling(isolation_forest_params_runtime_tune_results, metrics)
```



Wykres dla wyników wcześniejszych otrzymanych:

```
In [ ]: plot_metric_groups_for_sampling(isolation_forest_params_full_tune_results, metrics)
```



W szczególności ewaluacja modelu na pełnym zbiorze testowym udowodniła, że w przypadku lasu izolacyjnego nie warto stosować metod samplingu. Las izolacyjny jest specjalnie tak skonstruowany, żeby wykrywać egzemplarze klasy mniejszościowej, a jeśli zaburzymy mu proporcje między klasą większościową, a mniejszościową to możemy ostatecznie pogorszyć jakość modelu.

## Trening i ocena modelu

Budowanie oraz ocena modelu podstawowego dla dobranych parametrów, z uwzględnieniem aktualnie załadowanego zbioru treningowego oraz testowego

```
In [ ]: isolation_forest_best_params = {
    "n_estimators": 50,
    "max_samples": 128,
    "contamination": 0.03,
    "random_state": SEED,
    "n_jobs": -1
}
isolation_forest_sampling_method = SamplingMethod.NONE
isolation_forest_sampling_strategy = 0.0

isolation_forest_runtime_model_evaluation_result = train_and_evaluate_best_model
  test_df,
  isolation_
  build_isol
  isolation_
  isolation_
  isolation_
  isolation_
  unsupervis
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.043011  | 0.8    | 0.081633 | 0.001063       | 0.000885               |

Wariant modelu z treningiem bez osobników odstających

```
In [ ]: isolation_forest_filtered_train_df_best_params = {
    "n_estimators": 50,
    "max_samples": 128,
    "contamination": 0.03,
    "random_state": SEED,
    "n_jobs": -1
}
isolation_forest_filtered_train_df_sampling_method = SamplingMethod.NONE
isolation_forest_filtered_train_df_sampling_strategy = 0.0

isolation_forest_runtime_model_filtered_train_df_evaluation_result = train_and_e
  test_df,
  isolation_
  build_isol
  isolation_
  isolation_
  isolation_
  isolation_
  unsupervis
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.057471  | 1.0    | 0.108696 | 0.001061       | 0.000883               |

Załadowanie wcześniej przygotowanego modelu, który został wytrenowany na zbiorze treningowym stanowiącym 20% zbioru treningowego pierwotnego.

```
In [ ]: if (LOAD_PRETRAINED_MODEL and os.path.exists(isolation_forest_pretrained_model_file)):
    isolation_forest_full_model = joblib.load(isolation_forest_pretrained_model_file)

    x_test = test_df.drop(columns=["Class"])
    y_test = test_df["Class"]

    isolation_forest_full_model_evaluation_result = evaluate_model(isolation_forest_full_model,
  predict_positive_class_label)

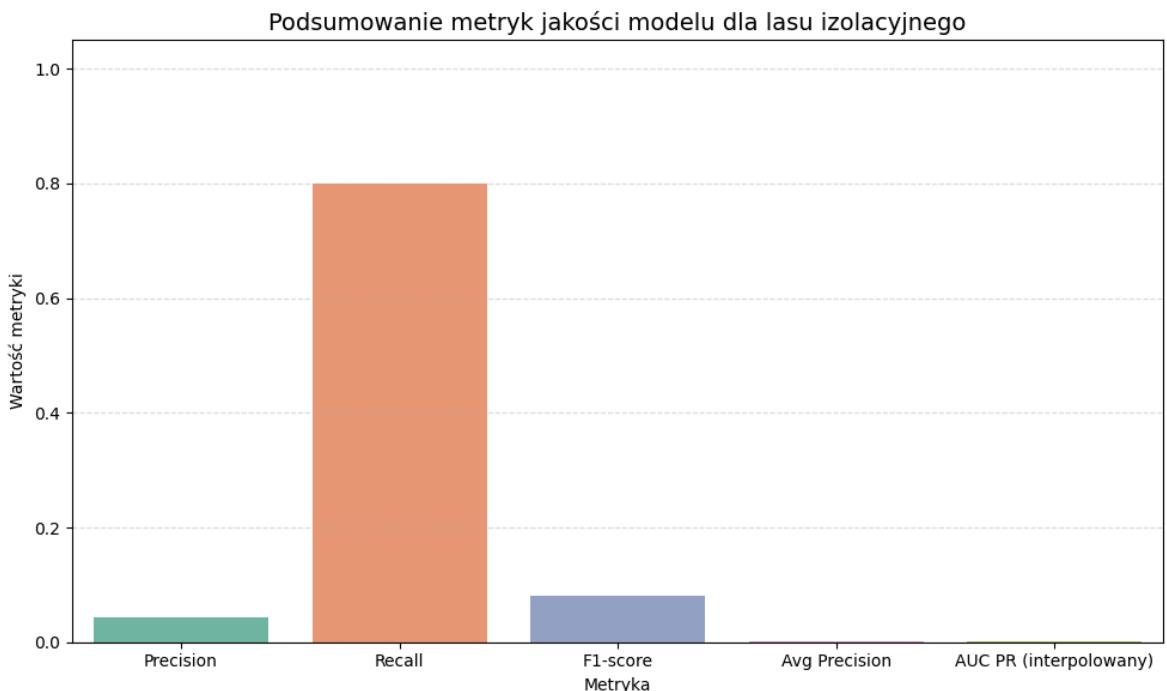
    best_results_df = pd.concat([best_results_df, pd.DataFrame([[{"Isolation Forest": isolation_forest_full_model_evaluation_result}]]),
                                isolation_forest_full_model_evaluation_result,
                                isolation_forest_full_model_evaluation_result,
                                isolation_forest_full_model_evaluation_result,
                                isolation_forest_full_model_evaluation_result,
                                columns=best_results_df.columns]]))
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.043478  | 0.8    | 0.082474 | 0.001063       | 0.000884               |

Podstawowe miary jakości dla modelu podstawowego

```
In [ ]: if isolation_forest_runtime_model_evaluation_result is not None:
    plot_metric_summary(
        metric_values=isolation_forest_runtime_model_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla lasu izolacyjnego")
else:
    print("Brak wyniku ewaluacji dla modelu w runtime")
```

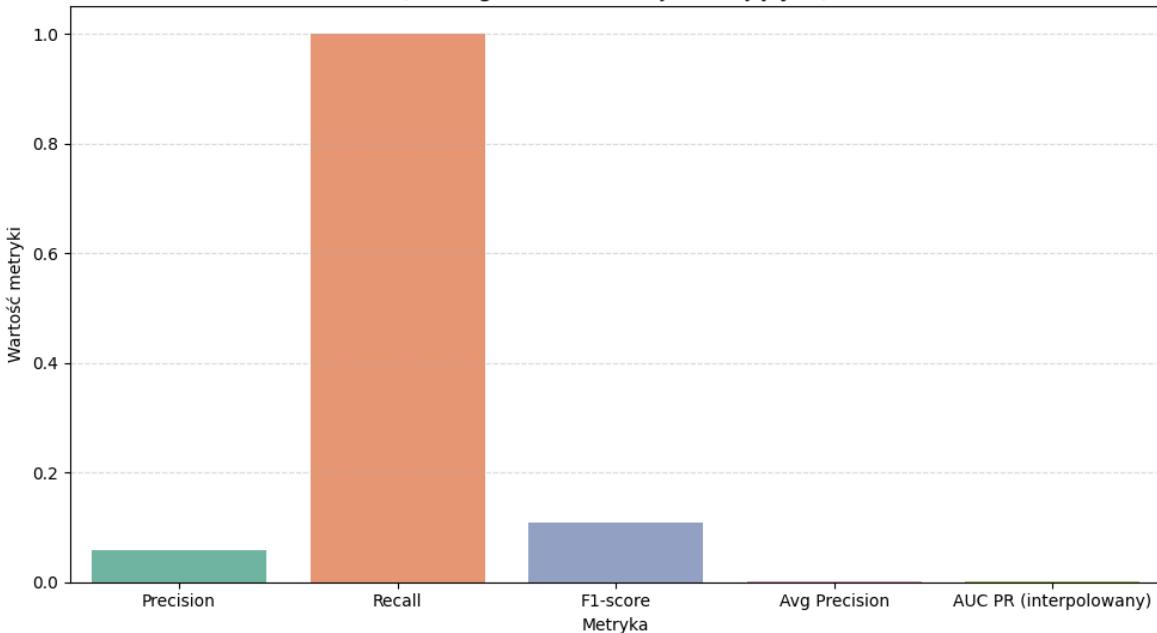


Podstawowe miary jakości dla modelu z treningiem bez osobników odstających

```
In [ ]: if isolation_forest_runtime_model_filtered_train_df_evaluation_result is not None:
    plot_metric_summary(
        metric_values=isolation_forest_runtime_model_filtered_train_df_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla lasu izolacyjnego\n(trening bez osobników odstających)")
```

```
else:  
    print("Brak wyniku ewaluacji dla modelu w runtime (przypadek treningu bez obse
```

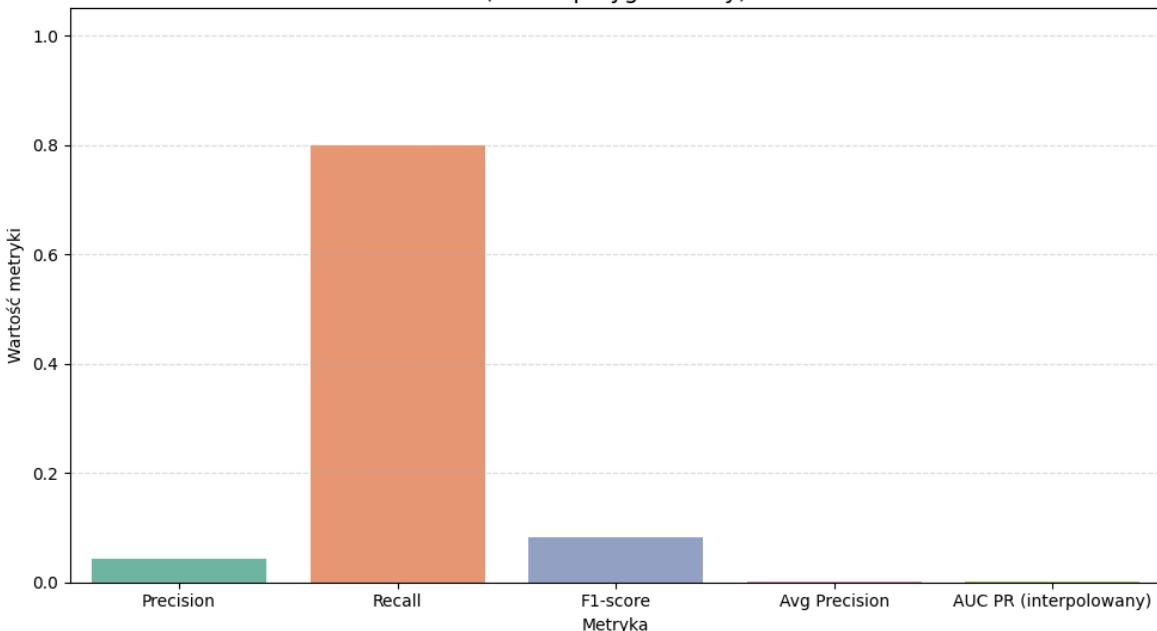
Podsumowanie metryk jakości modelu dla lasu izolacyjnego  
(trening bez obserwacji odstających)



Podstawowe miary jakości dla modelu wcześniejszej przygotowanego

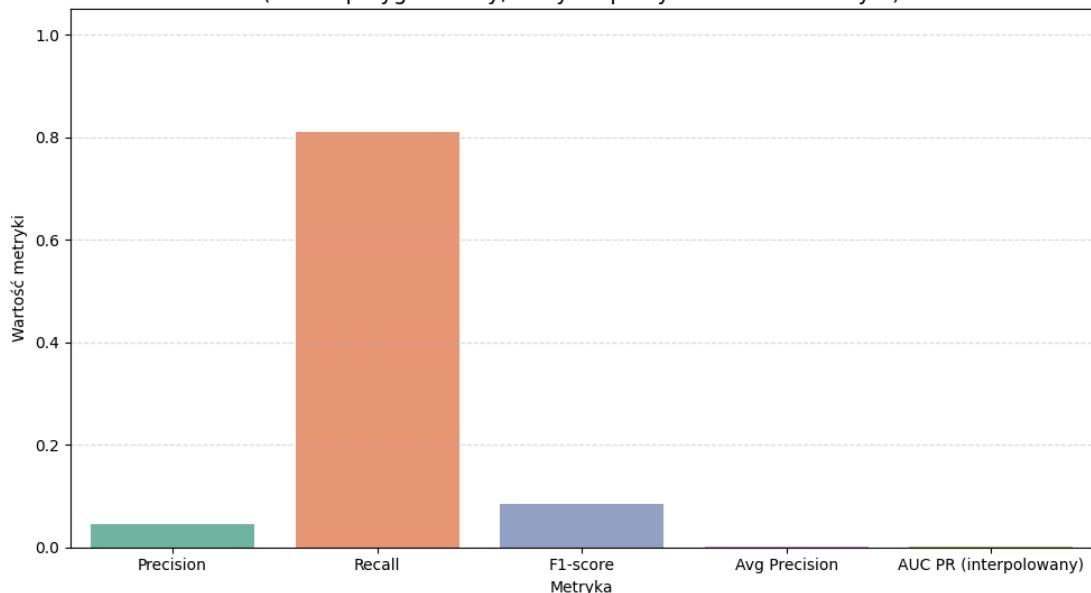
```
In [ ]: if isolation_forest_full_model_evaluation_result is not None:  
    plot_metric_summary(  
        metric_values=isolation_forest_full_model_evaluation_result,  
        title="Podsumowanie metryk jakości modelu dla lasu izolacyjnego\n(model przygotowany)"  
    )  
else:  
    print("Brak wyniku ewaluacji dla modelu wcześniejszej przygotowanego")
```

Podsumowanie metryk jakości modelu dla lasu izolacyjnego  
(model przygotowany)



Podstawowe miary jakości dla modelu wcześniejszej przygotowanego, określające wynik testów na pełnym zbiorze testowym (zapisany wykres po wcześniejszej wykonanej ewaluacji)

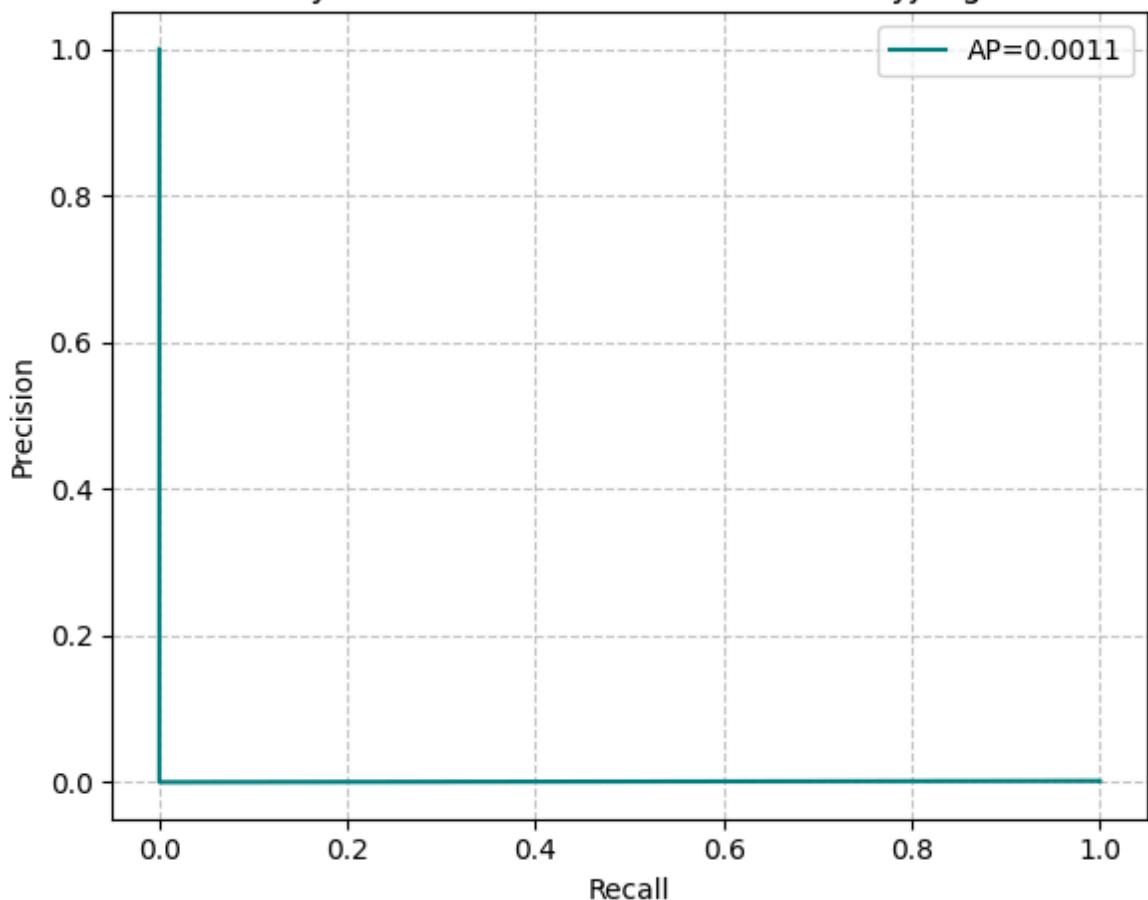
Podsumowanie metryk jakości modelu dla lasu izolacyjnego  
(model przygotowany, testy na pełnym zbiorze testowym)



Krzywa PR dla modelu podstawowego

```
In [ ]: if isolation_forest_runtime_model_evaluation_result is not None:  
    plot_pr_curve(  
        recall_vals=isolation_forest_runtime_model_evaluation_result["recall_vals"],  
        precision_vals=isolation_forest_runtime_model_evaluation_result["precision_v  
        avg_precision=isolation_forest_runtime_model_evaluation_result["avg_precisio  
        plot_title="Krzywa Precision-Recall dla lasu izolacyjnego"  
    )  
else:  
    print("Brak wyniku ewaluacji dla modelu w runtime")
```

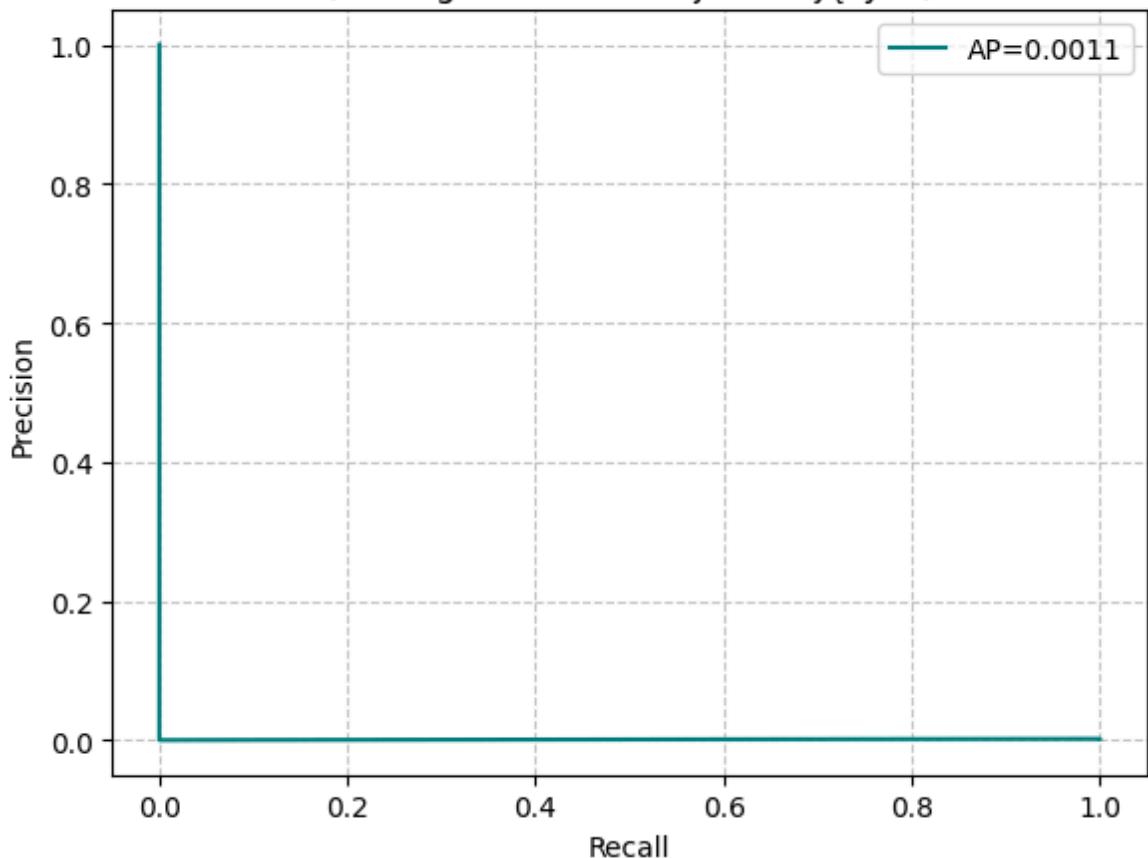
## Krzywa Precision-Recall dla lasu izolacyjnego



Krzywa PR dla modelu z treningiem bez osobników odstających

```
In [ ]: if isolation_forest_runtime_model_filtered_train_df_evaluation_result is not None:
    plot_pr_curve(
        recall_vals=isolation_forest_runtime_model_filtered_train_df_evaluation_resu
        precision_vals=isolation_forest_runtime_model_filtered_train_df_evaluation_r
        avg_precision=isolation_forest_runtime_model_filtered_train_df_evaluation_re
        plot_title="Krzywa Precision-Recall dla lasu izolacyjnego\n(trening bez obse
    )
else:
    print("Brak wyniku ewaluacji dla modelu w runtime (przypadek treningu bez obse
```

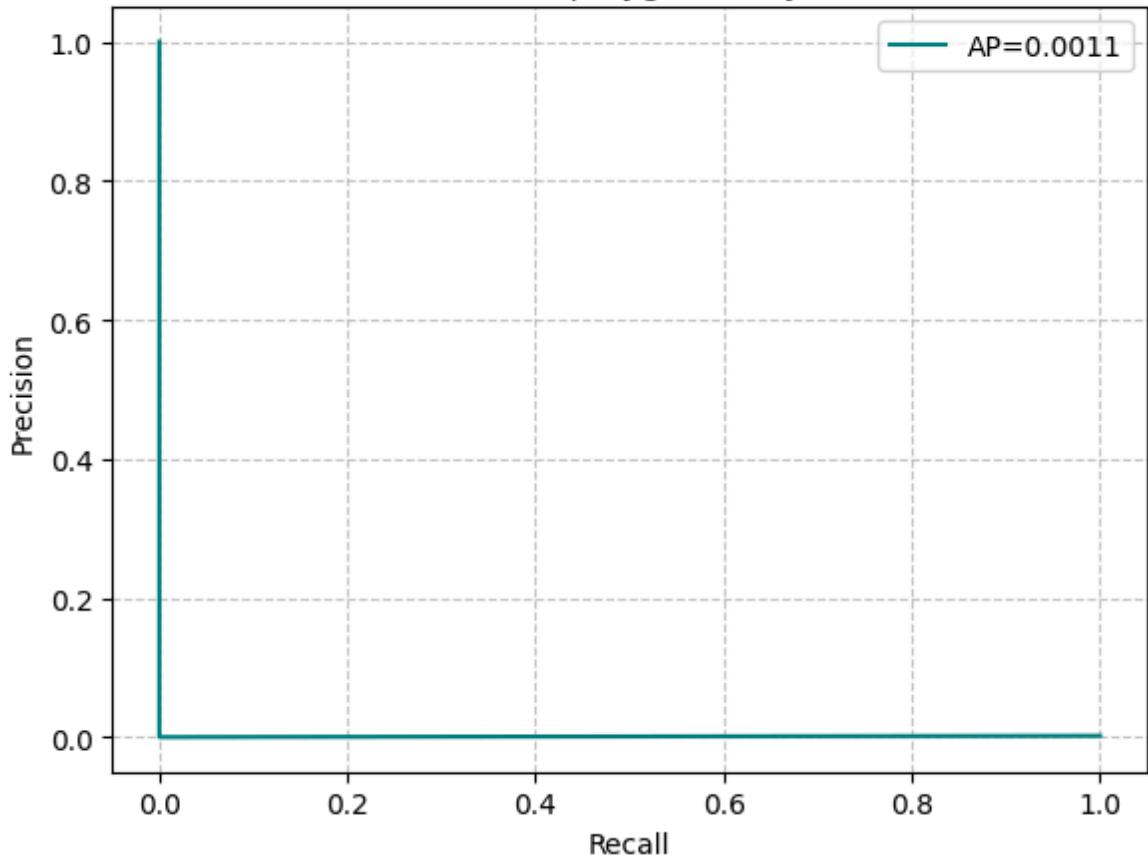
### Krzywa Precision-Recall dla lasu izolacyjnego (trening bez obserwacji odstających)



Krzywa PR dla modelu wcześniejszy przygotowanego

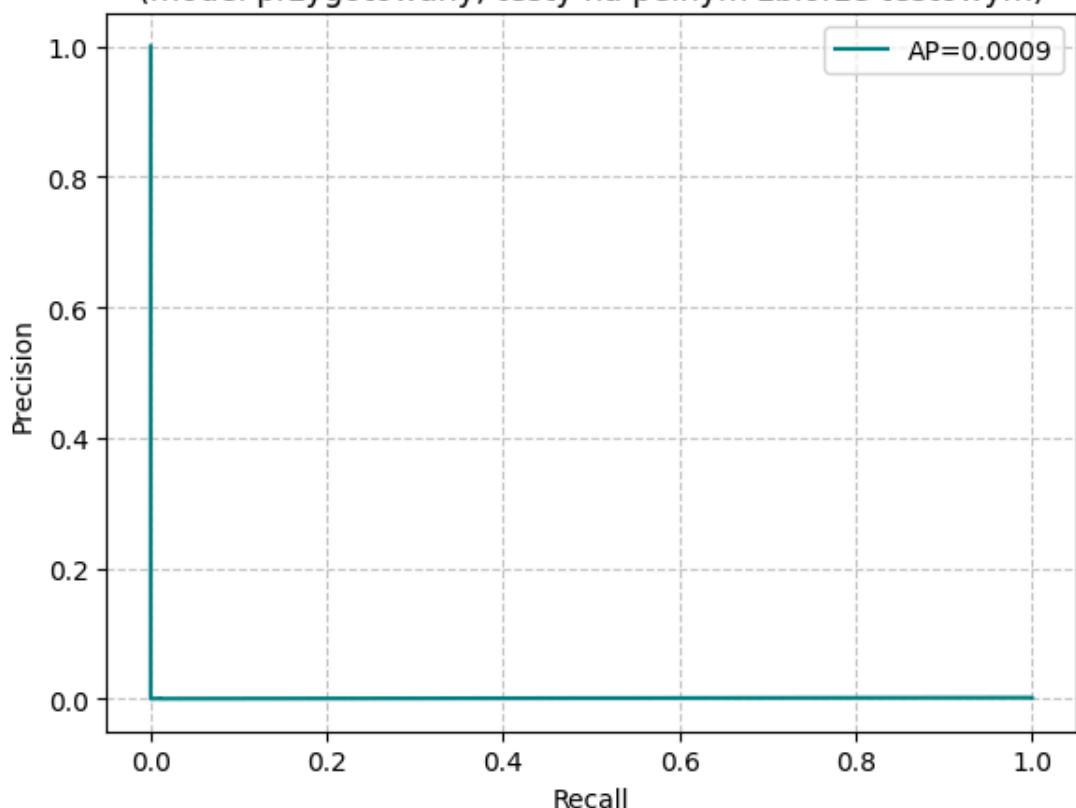
```
In [ ]: if isolation_forest_full_model_evaluation_result is not None:  
    plot_pr_curve(  
        recall_vals=isolation_forest_full_model_evaluation_result["recall_vals"],  
        precision_vals=isolation_forest_full_model_evaluation_result["precision_vals"],  
        avg_precision=isolation_forest_full_model_evaluation_result["avg_precision"],  
        plot_title="Krzywa Precision-Recall dla lasu izolacyjnego\n(model przygotowany)")  
else:  
    print("Brak wyniku ewaluacji dla modelu wcześniejszy przygotowanego")
```

### Krzywa Precision-Recall dla lasu izolacyjnego (model przygotowany)



Krzywa PR dla modelu wcześniej przygotowanego, określająca wynik testów na pełnym zbiorze testowym (zapisany wykres po wcześniej wykonanej ewaluacji)

### Krzywa Precision-Recall dla lasu izolacyjnego (model przygotowany, testy na pełnym zbiorze testowym)



W porównaniu do OC-SVM, udało się uzyskać trochę lepszy balans Precision a Recall, ale dalej odbiega sporo od ideału. Pomimo, że dla mniejszego zbioru danych pokazało, że lepiej w tym przypadku trenować model w oparciu o dane bez osobników odstających, to dla zbioru treningowego, który stanowił 20% pierwotnego zbioru ciut lepsze wyniki uzyskiwaliśmy z reprezentantami klasy mniejszościowej.

## Lokalny czynnik odstający

Przy budowaniu modelu opartego na algorytmie lokalnego czynnika odstającego, skupimy się na następujących parametrach:

- *n\_neighbors* - liczba sąsiadów do porównania gęstości lokalnej,
- *contamination* - procent danych, który traktowany jest jako anomalia.

```
In [ ]: from sklearn.neighbors import LocalOutlierFactor
def build_lof_model_function(**kwargs):
    return LocalOutlierFactor(**kwargs)
```

## Proces strojenia parametrów i badań ich wpływu na jakość modelu

Poniżej następuje uruchomienie całego procesu strojenia parametrów, wraz z prezentacją wyników w formie tabularycznej.

```
In [ ]: import os

if QUICK_MODE:
    lof_params_to_test = [
        ParameterToTest(name="n_neighbors", values=[5, 20, 50]),
        ParameterToTest(name="contamination", values=[0.0001, 0.001, 0.01]),
    ]

    lof_sampling_method_to_test = [
        SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, data_s
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, data_
        SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, data_s
    ]
else:
    lof_params_to_test = [
        ParameterToTest(name="n_neighbors", values=[5, 10, 20, 35, 50]),
        ParameterToTest(name="contamination", values=[0.0001, 0.001, 0.005, 0.01
    ]

    lof_sampling_method_to_test = [
        SamplingMethodToTest(method=SamplingMethod.NONE, sampling_strategy=0, da
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=0.5,
            data_splits=undersampling_50_data_splits),
        SamplingMethodToTest(method=SamplingMethod.UNDER, sampling_strategy=1, d
        SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=0.5,
            data_splits=oversampling_50_data_splits),
        SamplingMethodToTest(method=SamplingMethod.OVER, sampling_strategy=1, da
    ]

def build_model_function(**kwargs):
```

```

    return LocalOutlierFactor(**kwargs)

lof_model_specific_params = {
    "novelty": True,
    "n_jobs": -1}

if (QUICK_MODE and os.path.exists(lof_tune_results_tmp_file_path)):
    lof_params_runtime_tune_results = pd.read_csv(lof_tune_results_tmp_file_path,
else:
    lof_params_runtime_tune_results = params_tune(lof_params_to_test, sampling_met
  build_model_function=build_lof_model_fun
  model_specific_params=lof_model_specific
display(lof_params_runtime_tune_results)

```

|    | sampling | sampling_strategy | param           | value   | mean_precision | mean_recall | mean_f1 |
|----|----------|-------------------|-----------------|---------|----------------|-------------|---------|
| 0  | NONE     |                   | 0 n_neighbors   | 5.0000  | 0.011754       | 0.516667    | 0.0     |
| 1  | NONE     |                   | 0 n_neighbors   | 20.0000 | 0.034778       | 0.833333    | 0.0     |
| 2  | NONE     |                   | 0 n_neighbors   | 50.0000 | 0.034582       | 0.888889    | 0.0     |
| 3  | UNDER    |                   | 1 n_neighbors   | 5.0000  | 0.013916       | 0.305556    | 0.0     |
| 4  | UNDER    |                   | 1 n_neighbors   | 20.0000 | 0.000000       | 0.000000    | 0.0     |
| 5  | UNDER    |                   | 1 n_neighbors   | 50.0000 | 0.000000       | 0.000000    | 0.0     |
| 6  | OVER     |                   | 1 n_neighbors   | 5.0000  | 0.017863       | 0.833333    | 0.0     |
| 7  | OVER     |                   | 1 n_neighbors   | 20.0000 | 0.028291       | 0.833333    | 0.0     |
| 8  | OVER     |                   | 1 n_neighbors   | 50.0000 | 0.026612       | 0.888889    | 0.0     |
| 9  | NONE     |                   | 0 contamination | 0.0001  | 0.500000       | 0.055556    | 0.1     |
| 10 | NONE     |                   | 0 contamination | 0.0010  | 0.857143       | 0.555556    | 0.5     |
| 11 | NONE     |                   | 0 contamination | 0.0100  | 0.127305       | 0.833333    | 0.2     |
| 12 | UNDER    |                   | 1 contamination | 0.0001  | 0.108696       | 0.250000    | 0.1     |
| 13 | UNDER    |                   | 1 contamination | 0.0010  | 0.108696       | 0.250000    | 0.1     |
| 14 | UNDER    |                   | 1 contamination | 0.0100  | 0.108696       | 0.250000    | 0.1     |
| 15 | OVER     |                   | 1 contamination | 0.0001  | 0.650000       | 0.272222    | 0.3     |
| 16 | OVER     |                   | 1 contamination | 0.0010  | 0.483333       | 0.372222    | 0.4     |
| 17 | OVER     |                   | 1 contamination | 0.0100  | 0.069610       | 0.683333    | 0.1     |



Tutaj dla porównania zostają wyświetlane wcześniejsze uzyskane wyniki, które zostały zapisane do odpowiedniego pliku. Proces strojenia parametrów odbywał się na całym zbiorze danych.

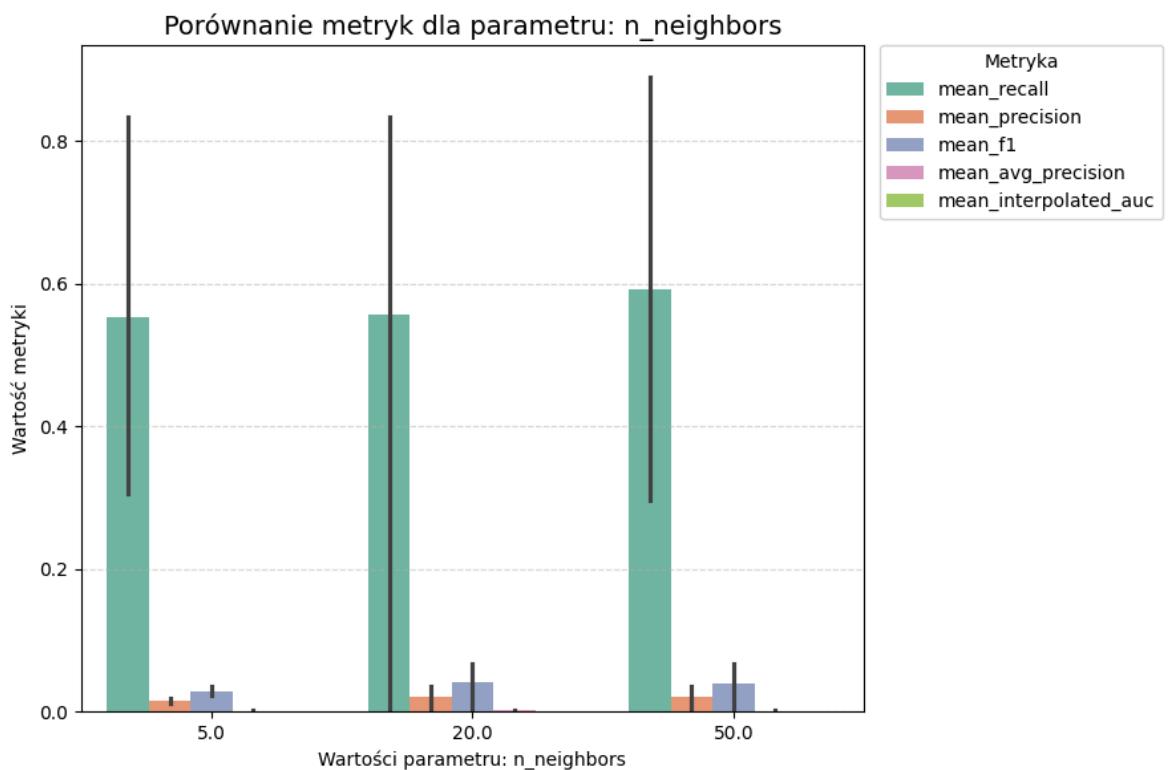
```
In [ ]: if (LOAD_PARAMS_TUNE_RESULTS_FROM_MASTER_FILE and os.path.exists(lof_tune_results_master_file_path,
    lof_params_full_tune_results = pd.read_csv(lof_tune_results_master_file_path,
    display(lof_params_full_tune_results)
```

|    | sampling | sampling_strategy | param         | value   | mean_precision | mean_recall | mean_f1  |
|----|----------|-------------------|---------------|---------|----------------|-------------|----------|
| 0  | NONE     |                   | n_neighbors   | 5.0000  | 0.002615       | 0.105825    | 0.053965 |
| 1  | NONE     |                   | n_neighbors   | 10.0000 | 0.002914       | 0.095228    | 0.047635 |
| 2  | NONE     |                   | n_neighbors   | 20.0000 | 0.003614       | 0.105825    | 0.053965 |
| 3  | NONE     |                   | n_neighbors   | 35.0000 | 0.004617       | 0.126912    | 0.066829 |
| 4  | NONE     |                   | n_neighbors   | 50.0000 | 0.004375       | 0.113684    | 0.056929 |
| 5  | UNDER    |                   | n_neighbors   | 5.0000  | 0.001750       | 0.063474    | 0.031235 |
| 6  | UNDER    |                   | n_neighbors   | 10.0000 | 0.002133       | 0.071474    | 0.038535 |
| 7  | UNDER    |                   | n_neighbors   | 20.0000 | 0.002655       | 0.095263    | 0.047635 |
| 8  | UNDER    |                   | n_neighbors   | 35.0000 | 0.002145       | 0.082000    | 0.040965 |
| 9  | UNDER    |                   | n_neighbors   | 50.0000 | 0.002519       | 0.105719    | 0.049795 |
| 10 | UNDER    |                   | n_neighbors   | 5.0000  | 0.001804       | 0.066105    | 0.033965 |
| 11 | UNDER    |                   | n_neighbors   | 10.0000 | 0.002274       | 0.079368    | 0.040965 |
| 12 | UNDER    |                   | n_neighbors   | 20.0000 | 0.002482       | 0.092667    | 0.047635 |
| 13 | UNDER    |                   | n_neighbors   | 35.0000 | 0.001934       | 0.082035    | 0.040965 |
| 14 | UNDER    |                   | n_neighbors   | 50.0000 | 0.002238       | 0.100526    | 0.049795 |
| 15 | OVER     |                   | n_neighbors   | 5.0000  | 0.020047       | 0.851789    | 0.650000 |
| 16 | OVER     |                   | n_neighbors   | 10.0000 | 0.023532       | 0.849263    | 0.640000 |
| 17 | OVER     |                   | n_neighbors   | 20.0000 | 0.024501       | 0.849263    | 0.640000 |
| 18 | OVER     |                   | n_neighbors   | 35.0000 | 0.024281       | 0.841333    | 0.630000 |
| 19 | OVER     |                   | n_neighbors   | 50.0000 | 0.023920       | 0.828175    | 0.620000 |
| 20 | OVER     |                   | n_neighbors   | 5.0000  | 0.020223       | 0.859719    | 0.630000 |
| 21 | OVER     |                   | n_neighbors   | 10.0000 | 0.023522       | 0.851895    | 0.630000 |
| 22 | OVER     |                   | n_neighbors   | 20.0000 | 0.024446       | 0.862491    | 0.640000 |
| 23 | OVER     |                   | n_neighbors   | 35.0000 | 0.023921       | 0.865123    | 0.640000 |
| 24 | OVER     |                   | n_neighbors   | 50.0000 | 0.023468       | 0.859860    | 0.630000 |
| 25 | NONE     |                   | contamination | 0.0001  | 0.000000       | 0.000000    | 0.000000 |
| 26 | NONE     |                   | contamination | 0.0010  | 0.000000       | 0.000000    | 0.000000 |
| 27 | NONE     |                   | contamination | 0.0050  | 0.005233       | 0.015895    | 0.008000 |
| 28 | NONE     |                   | contamination | 0.0100  | 0.004752       | 0.029053    | 0.012000 |
| 29 | UNDER    |                   | contamination | 0.0001  | 0.004580       | 0.008000    | 0.004000 |
| 30 | UNDER    |                   | contamination | 0.0010  | 0.003046       | 0.008000    | 0.004000 |
| 31 | UNDER    |                   | contamination | 0.0050  | 0.001290       | 0.008000    | 0.004000 |
| 32 | UNDER    |                   | contamination | 0.0100  | 0.000917       | 0.008000    | 0.004000 |

|    | sampling | sampling_strategy | param | value         | mean_precision | mean_recall | mean_f1  | mean_avg_precision | mean_interpolated_auc |
|----|----------|-------------------|-------|---------------|----------------|-------------|----------|--------------------|-----------------------|
| 33 | UNDER    |                   | 1.0   | contamination | 0.0001         | 0.003774    | 0.008000 | 0.000000           | 0.000000              |
| 34 | UNDER    |                   | 1.0   | contamination | 0.0010         | 0.003061    | 0.008000 | 0.000000           | 0.000000              |
| 35 | UNDER    |                   | 1.0   | contamination | 0.0050         | 0.001496    | 0.008000 | 0.000000           | 0.000000              |
| 36 | UNDER    |                   | 1.0   | contamination | 0.0100         | 0.001905    | 0.013263 | 0.000000           | 0.000000              |
| 37 | OVER     |                   | 0.5   | contamination | 0.0001         | 0.000000    | 0.000000 | 0.000000           | 0.000000              |
| 38 | OVER     |                   | 0.5   | contamination | 0.0010         | 0.206508    | 0.230316 | 0.200000           | 0.200000              |
| 39 | OVER     |                   | 0.5   | contamination | 0.0050         | 0.157436    | 0.648140 | 0.200000           | 0.200000              |
| 40 | OVER     |                   | 0.5   | contamination | 0.0100         | 0.115166    | 0.769895 | 0.200000           | 0.200000              |
| 41 | OVER     |                   | 1.0   | contamination | 0.0001         | 0.018182    | 0.002632 | 0.000000           | 0.000000              |
| 42 | OVER     |                   | 1.0   | contamination | 0.0010         | 0.241482    | 0.383789 | 0.200000           | 0.200000              |
| 43 | OVER     |                   | 1.0   | contamination | 0.0050         | 0.139875    | 0.793684 | 0.200000           | 0.200000              |
| 44 | OVER     |                   | 1.0   | contamination | 0.0100         | 0.093487    | 0.825439 | 0.100000           | 0.100000              |

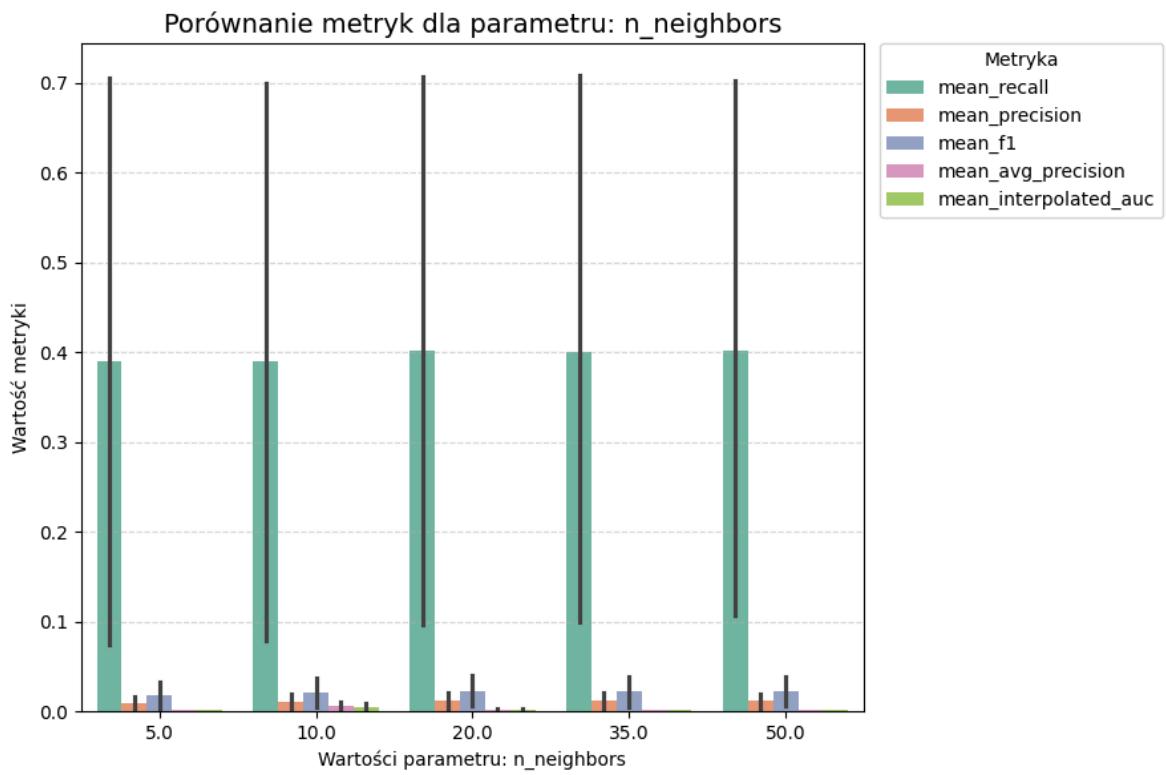
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(lof_params_runtime_tune_results, param_name="n_neig
```



Wykres dla wyników wcześniejszych otrzymanych:

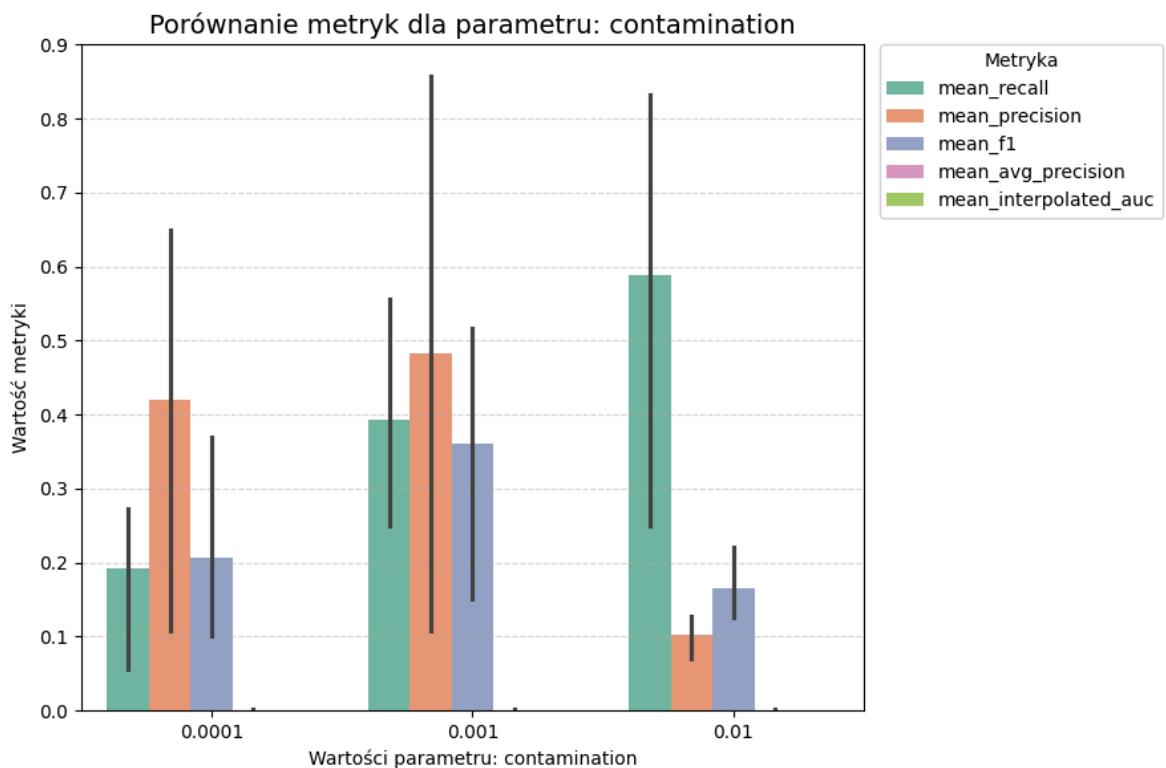
```
In [ ]: plot_metric_groups_for_param(lof_params_full_tune_results, param_name="n_neighbo
```



Bardzo zbliżone wyniki z delikatną tendencją wzrostową dla miary jakości Recall wraz ze wzrostem liczby sąsiadów.

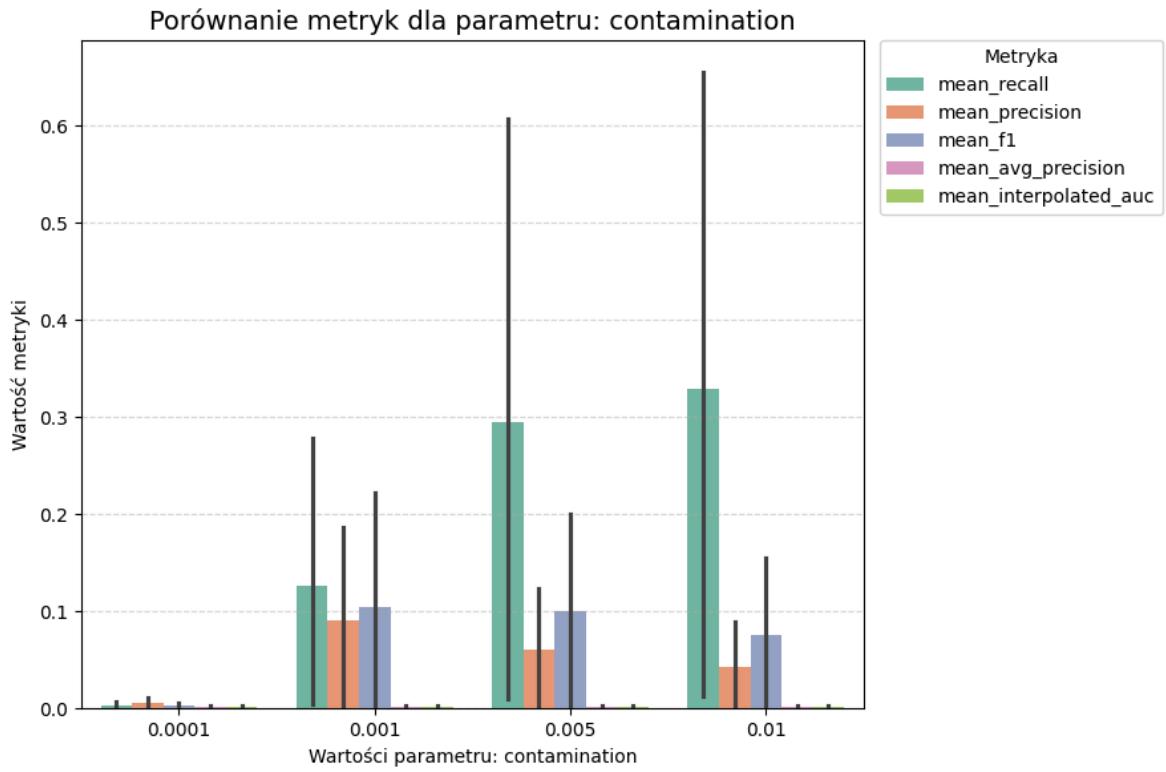
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_param(lof_params_runtime_tune_results, param_name="contamination")
```



Wykres dla wyników wcześniej otrzymanych:

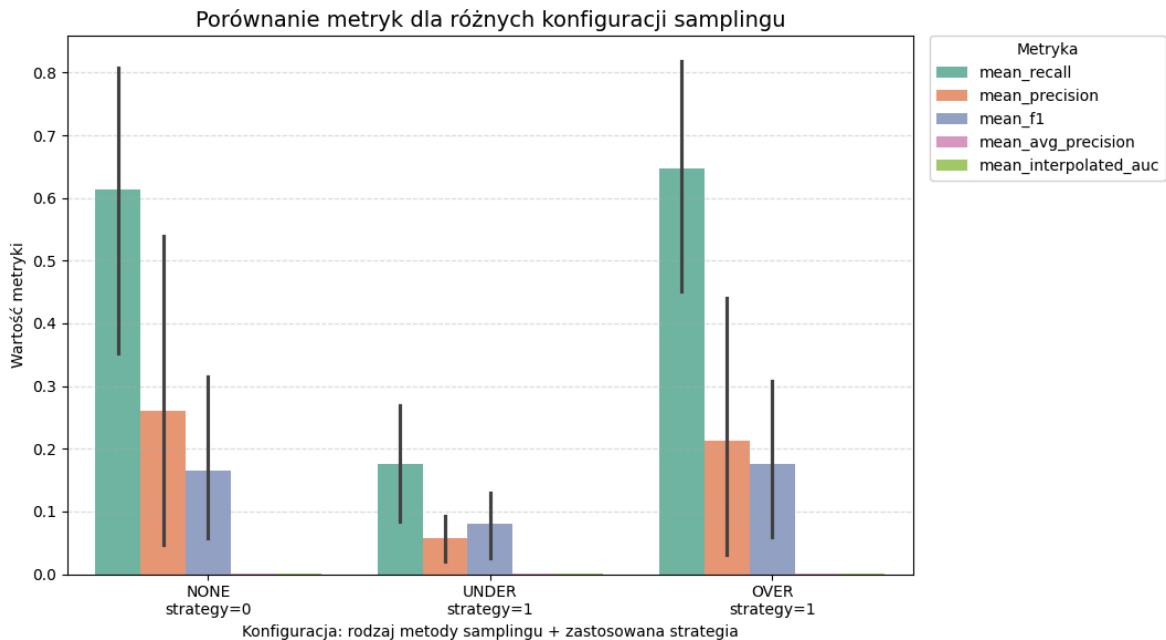
```
In [ ]: plot_metric_groups_for_param(lof_params_full_tune_results, param_name="contamination")
```



Tutaj sytuacja bardzo podobna, jak w przypadku analogicznego parametru dla modelu lasu izolacyjnego - w celu zachowania balansu między Recall a Precision powinniśmy szukać wartości dla parametru *contamination* w zakresie 0.001 - 0.005.

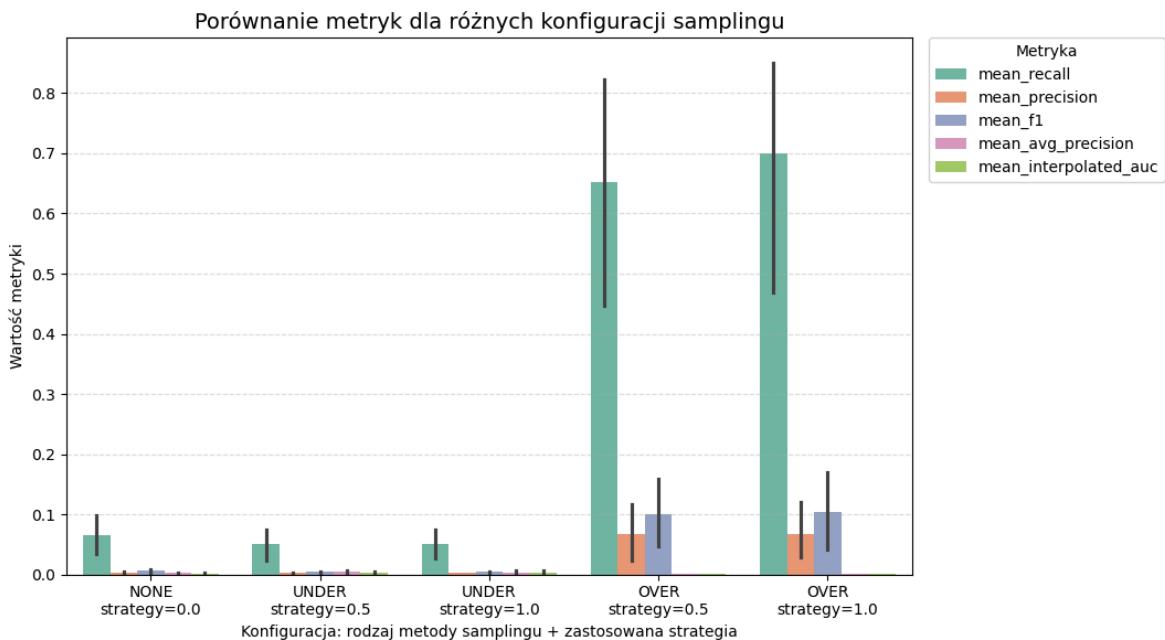
Wykres dla aktualnych wyników:

```
In [ ]: plot_metric_groups_for_sampling(lof_params_runtime_tune_results, metric_names=[ "me
```



Wykres dla wyników wcześniejszy otrzymanych:

```
In [ ]: plot_metric_groups_for_sampling(lof_params_full_tune_results, metric_names=[ "me
```



Tutaj mamy ciekawą sytuację, ponieważ wydawałoby się, że i tym razem najlepszym rozwiązaniem będzie porzucenie samplingu, ponieważ zaburzy ono gęstość występowania konkretnych rodzajów obiektów. Dla treningu i ewaluacji modelu na pełnym zbiorze okazało się, że oversampling może być ciekawą opcją do rozważenia. Możliwe, że generacja sztucznych osobników klasy mniejszościowej, pozwoliła uwidocznić "anomalia lokalne", które były trudne do uchwycenia z tego powodu, że było ich bardzo mało.

## Trening i ocena modelu

Budowanie oraz ocena modelu podstawowego dla dobranych parametrów, z uwzględnieniem aktualnie załadowanego zbioru treningowego oraz testowego

```
In [ ]: lof_best_params = {
    "n_neighbors": 50,
    "contamination": 0.03,
    "novelty": True,
    "n_jobs": -1
}
lof_sampling_method = SamplingMethod.OVER
lof_sampling_strategy = 1.0

lof_runtime_model_evaluation_result = train_and_evaluate_best_model(train_df, te
build_lof_mo
lof_model_na
lof_sampling
unsupervised
```

```
/usr/local/lib/python3.11/dist-packages/scikit-learn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with
feature names
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/scikit-learn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with
feature names
    warnings.warn(
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.037383  | 0.8    | 0.071429 | 0.00112        | 0.000917               |

Wariant modelu z treningiem bez osobników odstających

```
In [ ]: lof_filtered_train_df_best_params = {
    "n_neighbors": 50,
    "contamination": 0.03,
    "novelty": True,
    "n_jobs": -1
}
lof_filtered_train_df_sampling_method = SamplingMethod.NONE
lof_filtered_train_df_sampling_strategy = 0.0

lof_runtime_model_filtered_train_df_evaluation_result = train_and_evaluate_best_
    test_df,
    lof_filter
    build_lof_
    lof_filter
    lof_runtim
    lof_filter
    lof_filter
    unsupervis
```

```
/usr/local/lib/python3.11/dist-packages/scikit-learn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with
feature names
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/scikit-learn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with
feature names
    warnings.warn(
```

Ocena jakości modelu:

|   | Precision | Recall | F1-Score | Avg. Precision | AUC PR (interpolowane) |
|---|-----------|--------|----------|----------------|------------------------|
| 0 | 0.049505  | 1.0    | 0.09434  | 0.001058       | 0.000882               |

Załadowanie wcześniejszej przygotowanego modelu, który został wytrenowany na zbiorze treningowym stanowiącym 20% zbioru treningowego pierwotnego.

```
In [ ]: if (LOAD_PRETRAINED_MODEL and os.path.exists(lof_pretrained_model_file_path)):
    lof_full_model = joblib.load(lof_pretrained_model_file_path)

    x_test = test_df.drop(columns=["Class"])
    y_test = test_df["Class"]

    lof_full_model_evaluation_result = evaluate_model(lof_full_model, x_test, y_te
        predict_positive_class_label

    best_results_df = pd.concat([best_results_df, pd.DataFrame([[{"Local Outlier Fa
        lof_full_model_evalu
        lof_full_model_evalu
        lof_full_model_evalu
        lof_full_model_evalu
        columns=best_results_d
```

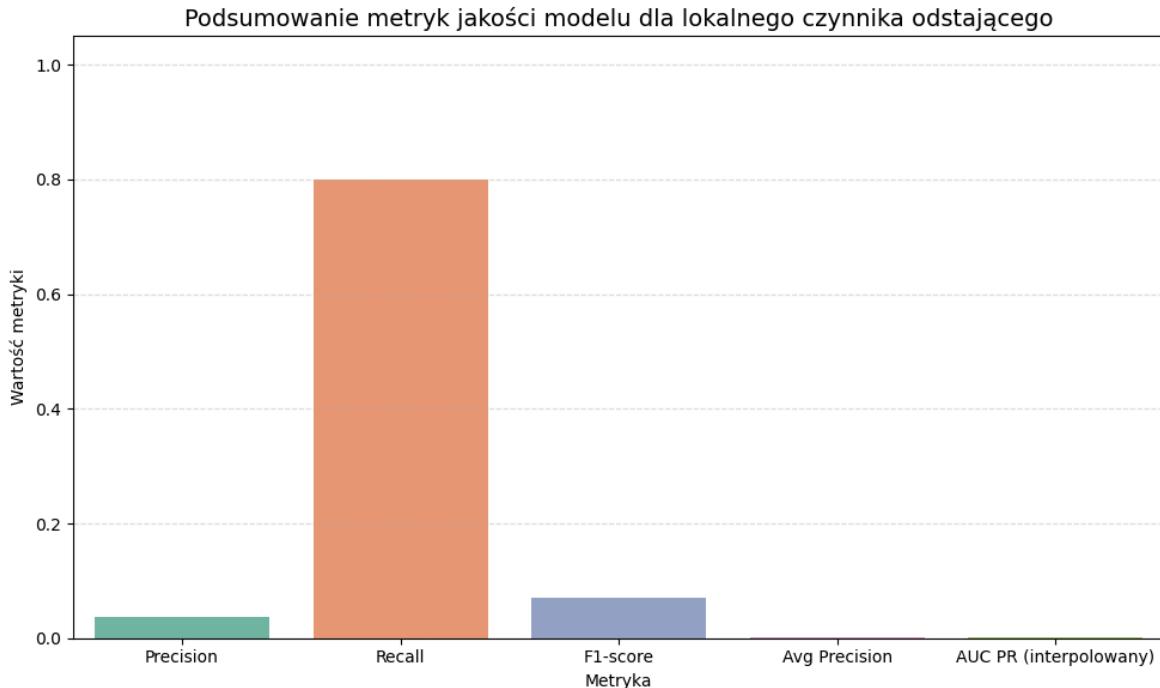
```

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with
feature names
    warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but LocalOutlierFactor was fitted with
feature names
    warnings.warn(
Ocena jakości modelu:
   Precision  Recall  F1-Score  Avg. Precision  AUC PR (interpolowane)
0      0.060241     1.0    0.113636          0.001058          0.000882

```

Podstawowe miary jakości dla modelu podstawowego

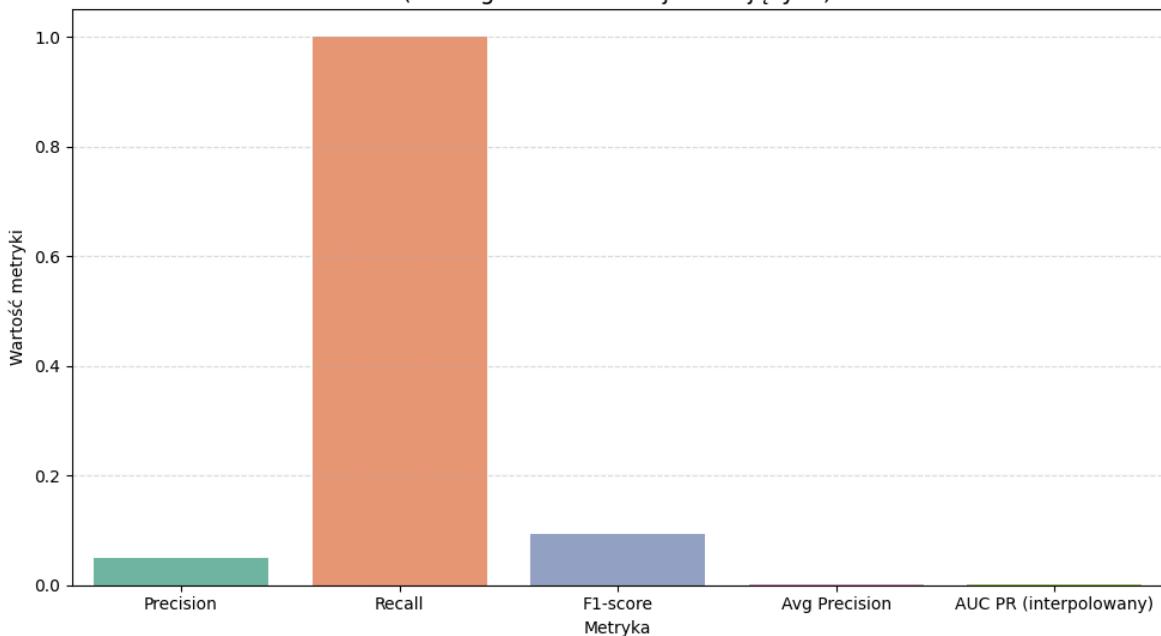
```
In [ ]: if lof_runtime_model_evaluation_result is not None:
    plot_metric_summary(
        metric_values=lof_runtime_model_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla lokalnego czynnika odstającego"
    )
else:
    print("Brak wyniku ewaluacji dla modelu w runtime")
```



Podstawowe miary jakości dla modelu z treningiem bez osobników odstających

```
In [ ]: if lof_runtime_model_filtered_train_df_evaluation_result is not None:
    plot_metric_summary(
        metric_values=lof_runtime_model_filtered_train_df_evaluation_result,
        title="Podsumowanie metryk jakości modelu dla lokalnego czynnika odstającego"
    )
else:
    print("Brak wyniku ewaluacji dla modelu w runtime (przypadek treningu bez obse")
```

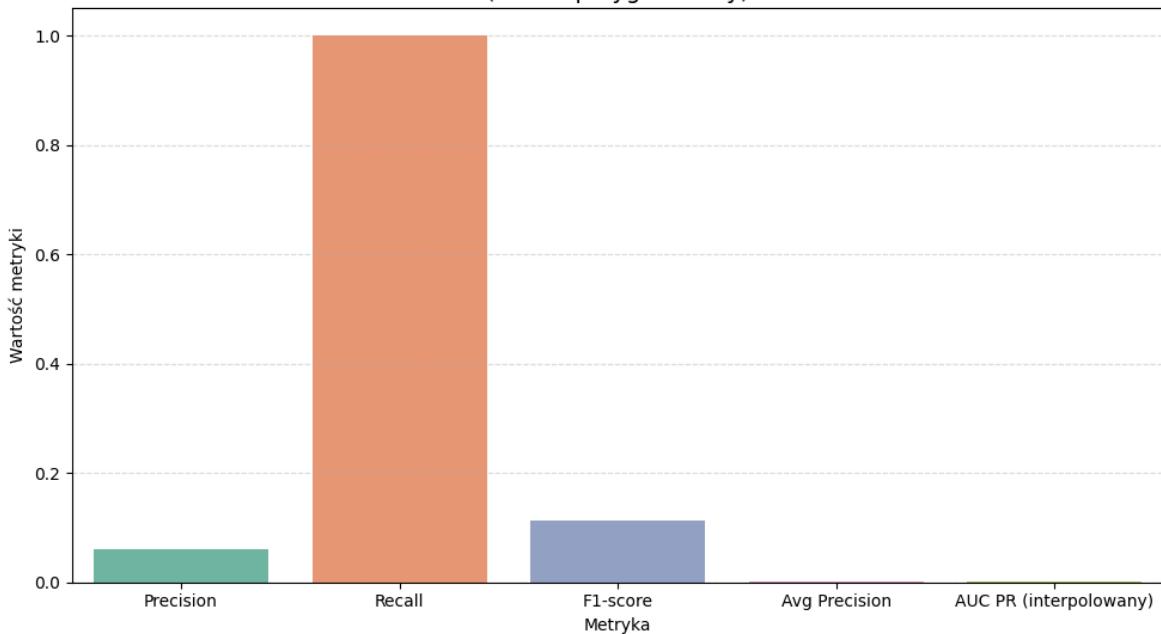
Podsumowanie metryk jakości modelu dla lokalnego czynnika odstającego  
(trening bez obserwacji odstających)



Podstawowe miary jakości dla modelu wcześniejszej przygotowanego

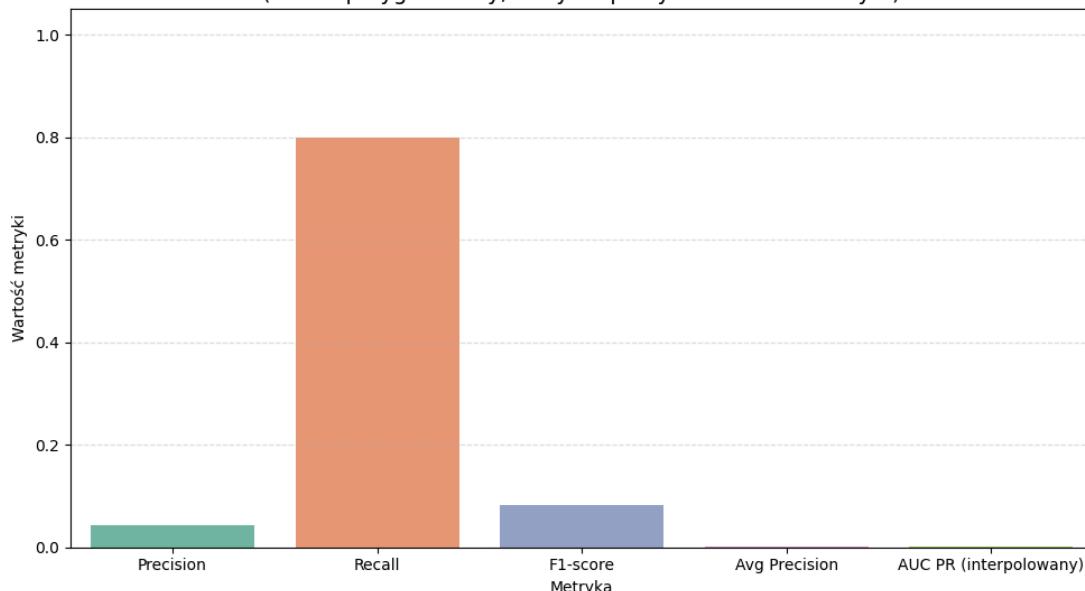
```
In [ ]: if lof_full_model_evaluation_result is not None:  
    plot_metric_summary(  
        metric_values=lof_full_model_evaluation_result,  
        title="Podsumowanie metryk jakości modelu dla lokalnego czynnika odstającego"  
    )  
else:  
    print("Brak wyniku ewaluacji dla modelu wcześniejszej przygotowanego")
```

Podsumowanie metryk jakości modelu dla lokalnego czynnika odstającego  
(model przygotowany)



Podstawowe miary jakości dla modelu wcześniejszej przygotowanego, określające wynik testów na pełnym zbiorze testowym (zapisany wykres po wcześniejszej wykonanej ewaluacji)

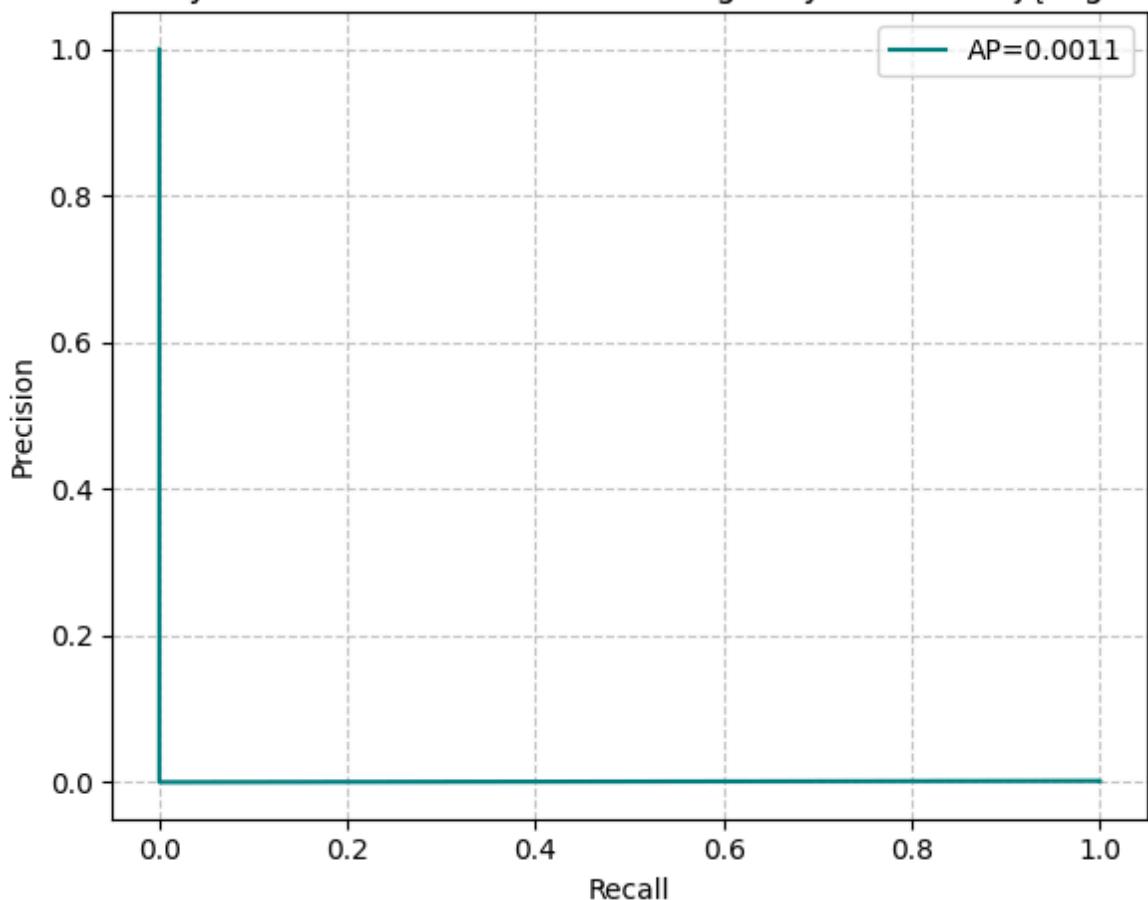
Podsumowanie metryk jakości modelu dla lokalnego czynnika odstającego  
(model przygotowany, testy na pełnym zbiorze testowym)



Krzywa PR dla modelu podstawowego

```
In [ ]: if lof_runtime_model_evaluation_result is not None:  
    plot_pr_curve(  
        recall_vals=lof_runtime_model_evaluation_result["recall_vals"],  
        precision_vals=lof_runtime_model_evaluation_result["precision_vals"],  
        avg_precision=lof_runtime_model_evaluation_result["avg_precision"],  
        plot_title="Krzywa Precision-Recall dla lokalnego czynnika odstającego"  
    )  
else:  
    print("Brak wyniku ewaluacji dla modelu w runtime")
```

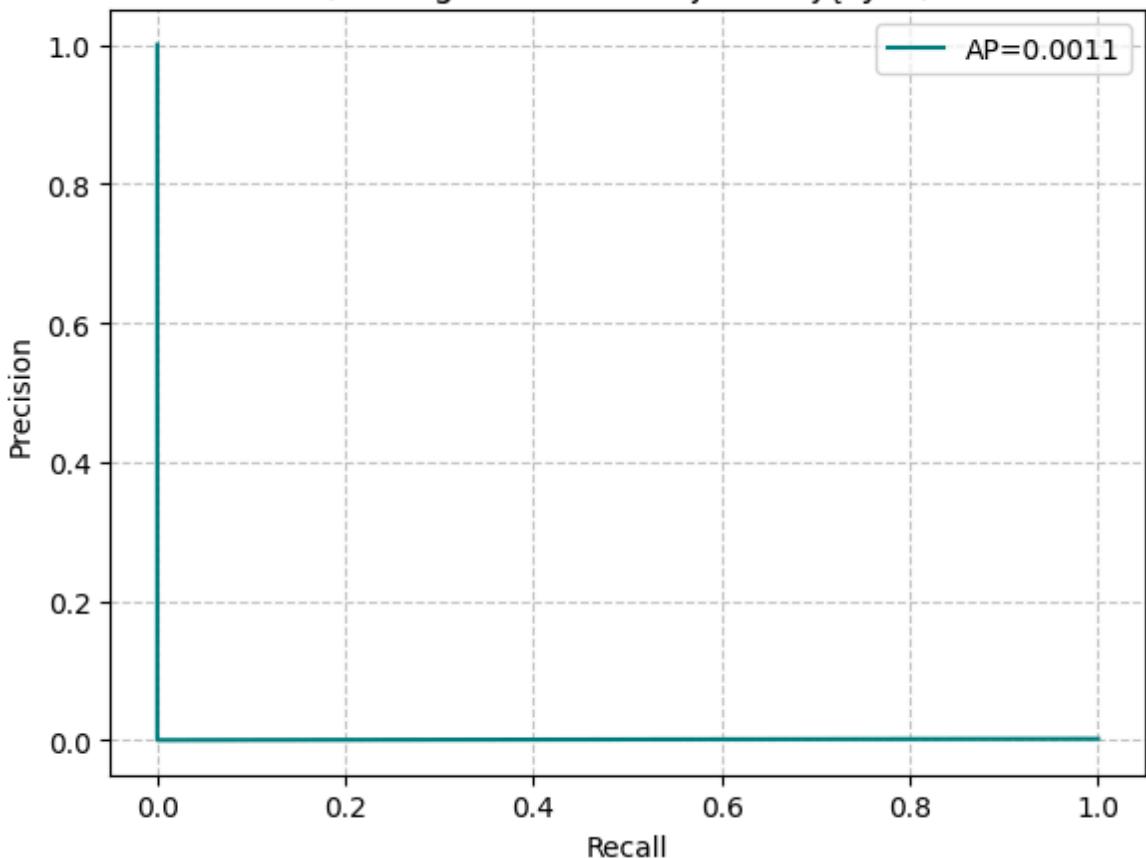
## Krzywa Precision-Recall dla lokalnego czynnika odstającego



Krzywa PR dla modelu z treningiem bez osobników odstających

```
In [ ]: if lof_runtime_model_filtered_train_df_evaluation_result is not None:
    plot_pr_curve(
        recall_vals=lof_runtime_model_filtered_train_df_evaluation_result["recall_va
        precision_vals=lof_runtime_model_filtered_train_df_evaluation_result["precision"
        avg_precision=lof_runtime_model_filtered_train_df_evaluation_result["avg_precision"
        plot_title="Krzywa Precision-Recall dla lokalnego czynnika odstającego\n(tre
    )
else:
    print("Brak wyniku ewaluacji dla modelu w runtime (przypadek treningu bez obserwatorów)")
```

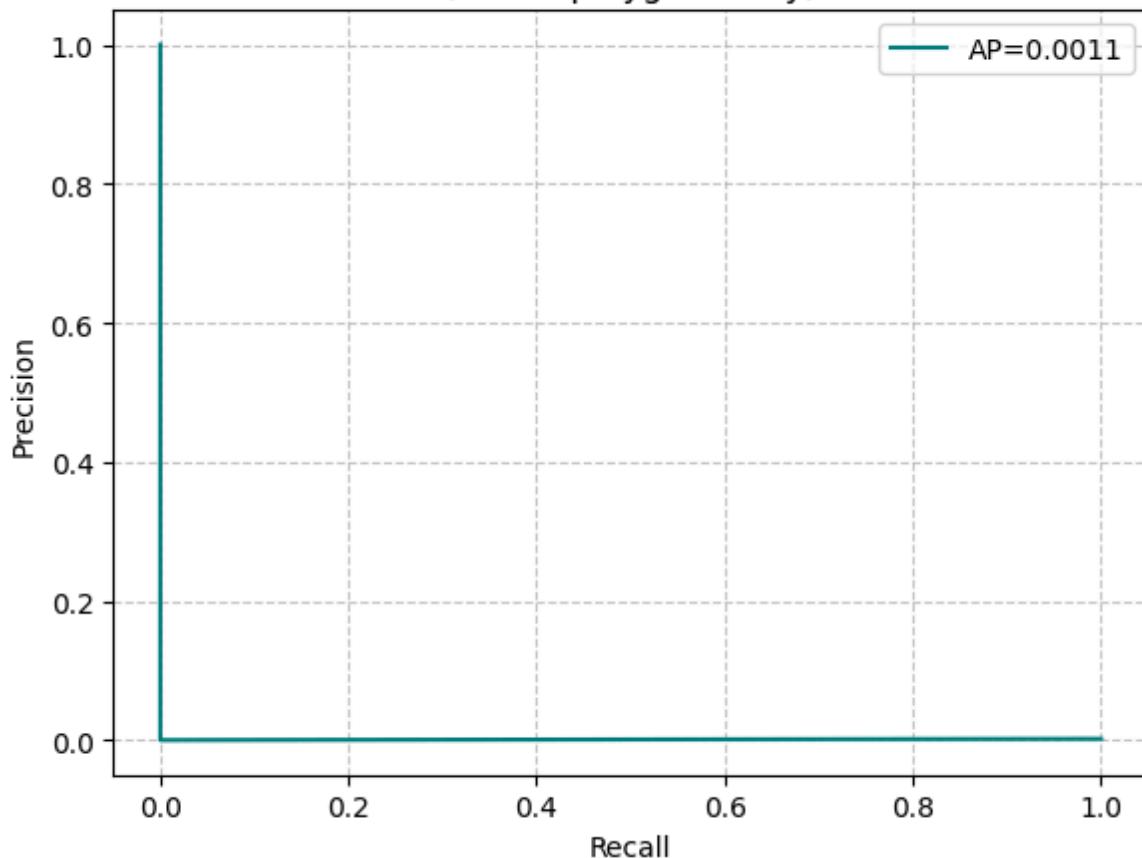
### Krzywa Precision-Recall dla lokalnego czynnika odstającego (trening bez obserwacji odstających)



Krzywa PR dla modelu wcześniej przygotowanego

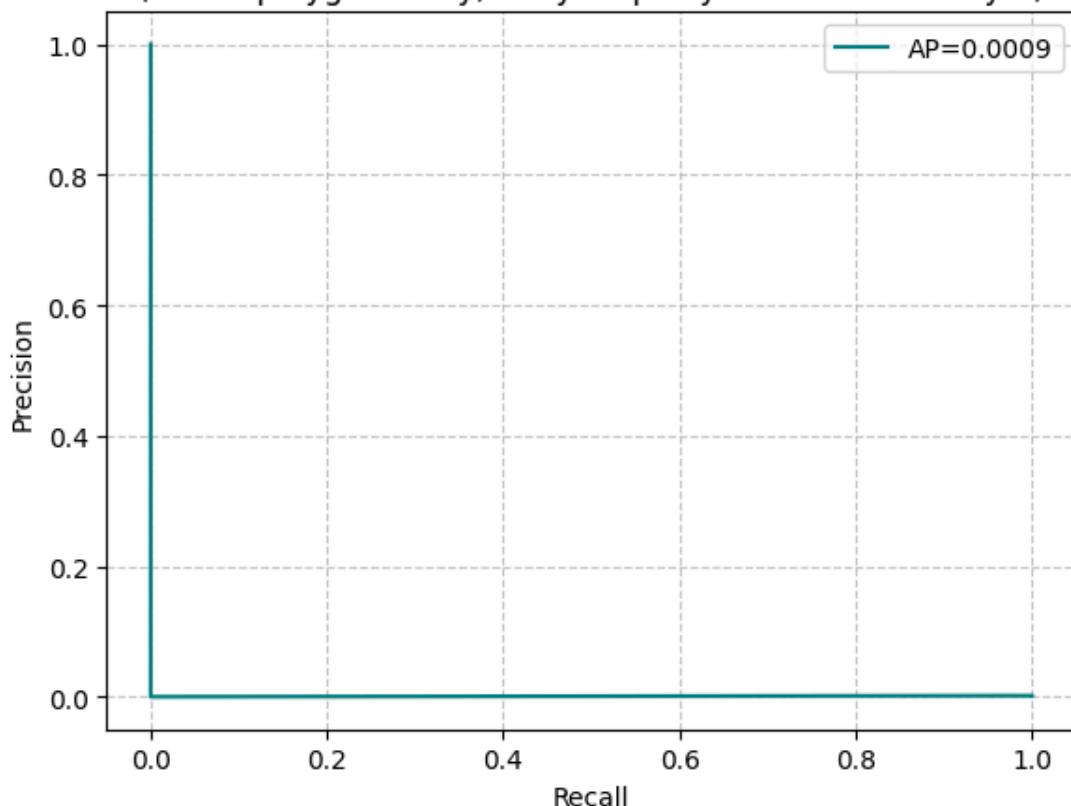
```
In [ ]: if lof_full_model_evaluation_result is not None:  
    plot_pr_curve(  
        recall_vals=lof_full_model_evaluation_result["recall_vals"],  
        precision_vals=lof_full_model_evaluation_result["precision_vals"],  
        avg_precision=lof_full_model_evaluation_result["avg_precision"],  
        plot_title="Krzywa Precision-Recall dla lokalnego czynnika odstającego\n(moduł LOF)")  
else:  
    print("Brak wyniku ewaluacji dla modelu wcześniej przygotowanego")
```

### Krzywa Precision-Recall dla lokalnego czynnika odstającego (model przygotowany)



Krzywa PR dla modelu wcześniej przygotowanego, określająca wynik testów na pełnym zbiorze testowym (zapisany wykres po wcześniej wykonanej ewaluacji)

### Krzywa Precision-Recall dla lokalnego czynnika odstającego (model przygotowany, testy na pełnym zbiorze testowym)



Z balansem Precision i Recall mamy podobnie jak u innych metod nienadzorowanych, choć w sumie tutaj się udało uzyskać najwyższą wartość dla Precision i F1-Score. W ostateczności, finalny model przez nas przygotowany był wytrenowany bez osobników odstających i tym samym bez samplingu - dla takiej konfiguracji uzyskaliśmy najlepsze rezultaty, trenując na zbiorze treningowym stanowiącym 20% pierwotnego zbioru oraz testując na pełnym zbiorze testowym.

## Podsumowanie

Dla poszczególnych algorytmów uzyskano następujące wyniki:

```
In [ ]: display(best_results_df)
```

|   | Model                | Precision | Recall   | F1-Score | Avg. Precision | AUC PR<br>(interpolowane) |
|---|----------------------|-----------|----------|----------|----------------|---------------------------|
| 0 | Random Forest        | 0.911392  | 0.757895 | 0.827586 | 0.824574       | 0.825025                  |
| 1 | Decision Tree        | 0.881579  | 0.705263 | 0.783626 | 0.641593       | 0.658603                  |
| 2 | OC-SVM               | 0.004931  | 1.000000 | 0.009814 | 0.001480       | 0.000881                  |
| 3 | Isolation Forest     | 0.043478  | 0.800000 | 0.082474 | 0.001063       | 0.000884                  |
| 4 | Local Outlier Factor | 0.060241  | 1.000000 | 0.113636 | 0.001058       | 0.000882                  |

Z powyższych wyników oraz całego procesu badawczego można wyciągnąć następujące wnioski:

1. Metody nadzorowane zapewniają o wiele lepszy balans między wartościami parametrów Precision i Recall, sprawiając, że wykrywamy dalej dużo przypadków prawdziwych anomalii i przy okazji mamy mało fałszywych alarmów.
2. Las losowy wykazał lepsze wyniki niż drzewo decyzyjne dla tego zbioru danych, wymagał jednak dłuższego czasu treningu.
3. Dla metod nadzorowanych nastąpił spadek precyzji i recall o ~0,1 na odseparowanym teście w porównaniu do najlepszego modelu wyłonionego w procesie dobory parametrów. Prawdopodobnie nastąpił lekki overfitting do danych walidacyjnych, przez co pojedynczy finalny model osiągnął niższą skuteczność niż sugerowałby uśredniony wynik 5-krotnej walidacji krzyżowej.
4. Ogólnie rzecz biorąc, nie warto używać metod samplingu przy metodach nienadzorowanych.
5. Warto czasem rozważyć w przypadku metod nienadzorowanych trening na zbiorze bez osobników odstających.
6. Trening na jakimś procencie danych może być całkiem pozytywny, nie tylko ze względu na ograniczenia narzutu obliczeniowego, ale także z możliwej redukcji niepotrzebnego szumu.