Politechnika Wrocławska

Autor: Filip Strózik 260377 Prowadzący: Mgr Inż. Michał Karol

Sztuczna Inteligencja i Inżynieria Wiedzy

Sprawozdanie z zadania 4

Cel ćwiczenia

Celem ćwiczenia jest praktyczne zapoznanie z prostymi algorytmami uczenia maszynowego oraz podstawowymi krokami realizacji projektu opartego o uczenie maszynowe: eksploracja danych i zdefiniowanie problemu, przygotowanie danych, dobór algorytmów i hiperparametrów, ocena algorytmów, poprawa wyników, przedstawienie wyników.

Na podstawie podanych źródeł w liście laboratoryjnej zostały przeprowadzone zadania na temat uczenia maszynowego w rozpoznawaniu rodzaju szkła.

Technologia

Python 3.11, Jupyter Notebook. Całość rozwiązania zadania 4 znajduje się w pliku z4.ipynb.

Realizacja zadań

eksploracja danych – przedstaw podstawowe dane statystyczne i uwagi dotyczące cech i etykiet zbioru danych. (10 punktów)

Dataset pod zasobem https://archive.ics.uci.edu/ml/machine-learning-databases/glass/glass.data. Jest plikiem .csv, który zawiera wartości cech próbek badanych. Dataset będzie używany do uczenia nadzorowaneg, ponieważ znamy wartości klas/rodzajów szkła dla każdej próbki. Pierwsze 20 wierszy wygląda następująco:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.00	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.00	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.00	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.00	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.00	1
5	6	1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0.0	0.26	1
6	7	1.51743	13.30	3.60	1.14	73.09	0.58	8.17	0.0	0.00	1
7	8	1.51756	13.15	3.61	1.05	73.24	0.57	8.24	0.0	0.00	1
8	9	1.51918	14.04	3.58	1.37	72.08	0.56	8.30	0.0	0.00	1
9	10	1.51755	13.00	3.60	1.36	72.99	0.57	8.40	0.0	0.11	1
10	11	1.51571	12.72	3.46	1.56	73.20	0.67	8.09	0.0	0.24	1
11	12	1.51763	12.80	3.66	1.27	73.01	0.60	8.56	0.0	0.00	1
12	13	1.51589	12.88	3.43	1.40	73.28	0.69	8.05	0.0	0.24	1
13	14	1.51748	12.86	3.56	1.27	73.21	0.54	8.38	0.0	0.17	1
14	15	1.51763	12.61	3.59	1.31	73.29	0.58	8.50	0.0	0.00	1
15	16	1.51761	12.81	3.54	1.23	73.24	0.58	8.39	0.0	0.00	1
16	17	1.51784	12.68	3.67	1.16	73.11	0.61	8.70	0.0	0.00	1
17	18	1.52196	14.36	3.85	0.89	71.36	0.15	9.15	0.0	0.00	1
18	19	1.51911	13.90	3.73	1.18	72.12	0.06	8.89	0.0	0.00	1
19	20	1.51735	13.02	3.54	1.69	72.73	0.54	8.44	0.0	0.07	1

Kolumny określające cechy trzeba było odpowiednio nazwać. Odwiedzając źródło możemy się dowiedzieć czym są odpowiednie cechy:

RI: współczynnik załamania światła

Na: Sód

Mg: Magnez

Al: Aluminium

K: Potas

Ca: Wapń

Ba: Bar

Fe: Żelazo

Rodzaj szkła: (atrybut klasy) [1-7]

1. Building_windows_float_processed
2. Building_windows_non_float_processed
3. Vehicle_windows_float_processed
4. Vehicle_windows_non_float_processed
5. Pojemniki
6. Zastawa stołowa
7. Reflektory

```
glass_data.shape
(214, 11)
```

Dataset ma 214 wierszy i 11 kolumn.

```
glass_data.isnull().sum()
Id      0
RI      0
Na      0
Mg      0
Al      0
Si      0
K       0
Ca      0
Ba      0
Fe      0
Type    0
dtype: int64
```

Dataset nie ma brakujących danych.

```
glass_data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 214 entries, 0 to 213
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Id      214 non-null    int64
```

```
 1   RI      214 non-null    float64
 2   Na      214 non-null    float64
 3   Mg      214 non-null    float64
 4   Al      214 non-null    float64
 5   Si      214 non-null    float64
 6   K       214 non-null    float64
 7   Ca      214 non-null    float64
 8   Ba      214 non-null    float64
 9   Fe      214 non-null    float64
 10  Type    214 non-null    int64
dtypes: float64(9), int64(2)
memory usage: 18.5 KB
```

Id oraz Type są cechami, których typ danych to liczba całkowita. Reszta zaś jest reprezentowana przez liczbę zmiennoprzecinkową.

Przedstawmy podstawowe dane statystyczne dotyczące zbioru danych.

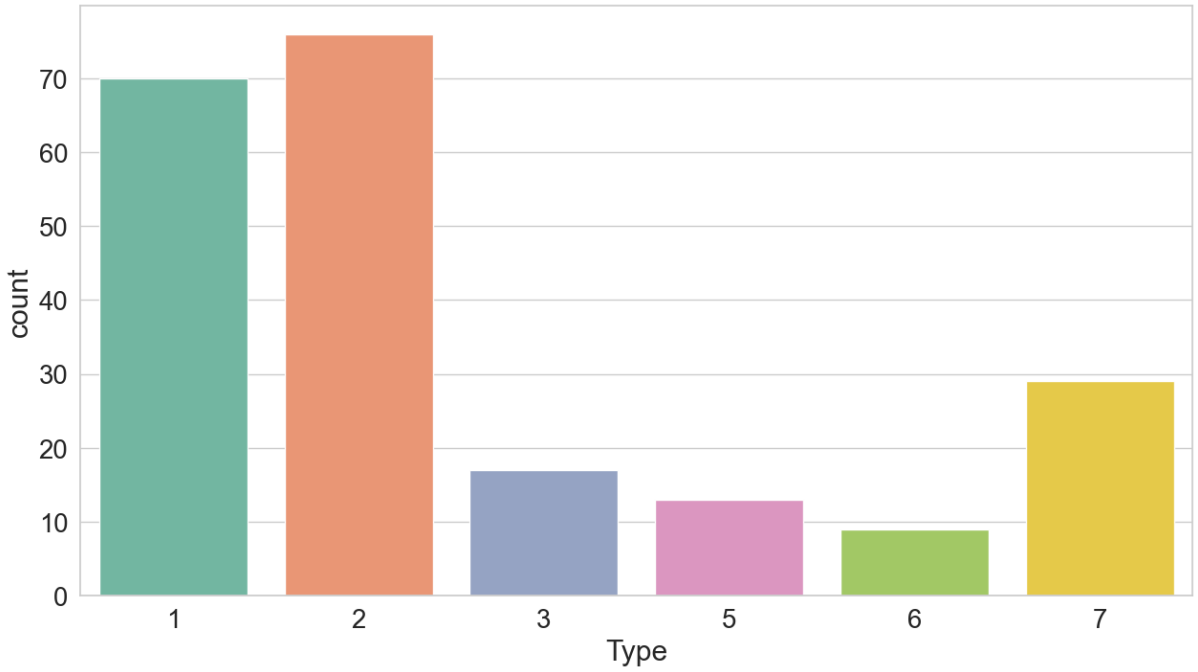
```
glass_data.describe()
```

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	107.500000	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047	0.057009	2.780374
std	61.920648	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219	0.097439	2.103739
min	1.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000	0.000000	1.000000
25%	54.250000	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000	0.000000	1.000000
50%	107.500000	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000	0.000000	2.000000
75%	160.750000	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000	0.100000	3.000000
max	214.000000	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000	0.510000	7.000000

Dane statystyczne kolumny Id zostaną pominięte, ponieważ są one indeksami wierszy. Odchylenie standardowe (std) dla większości cech jest niskie, co sugeruje, że dane są skoncentrowane wokół wartości średnich. Jednakże dla niektórych cech, takich jak Mg i Fe, odchylenie standardowe jest wyższe, co może wskazywać na większą zmienność w tych cechach. Wartości kwartylowe (25%, 50%, 75%) dla cech dostarczają informacji o rozkładzie danych w zbiorze: Mediana dla Ba wynosi 0.000000, co sugeruje, że większość próbek nie zawiera barowego dodatku.

Przeanalizujmy dataset w kontekście celu zadania. Chcemy rozpoznawać rodzaj szkła za pomocą wartości cech szkła. Jeżeli przyjrzymy się przez ile próbek szkła jest reprezentowany dany rodzaj zauważymy, że nie jest to równomiernie podzielone.

```
import seaborn as sns
sns.set(style="whitegrid", font_scale=1.8)
plt.subplots(figsize = (15,8))
sns.countplot(x="Type", data=glass_data, palette="Set2")
```



przygotowanie danych – podziel dane na zestaw uczący i walidacyjny (alternatywnie użyj walidacji krzyżowej), zbadaj wpływ różnego typu przetworzenia danych na wyniki klasyfikacji (proponowane: normalizacja, standaryzacja, dyskretyzacja, selekcja cech, PCA) - czyli wykonaj porównanie wyników bez przetworzenia danych z rezultatami po ich przetworzeniu, wykorzystując co najmniej 2 metody różnego typu (osobno). (30 punktów)

Usuwanie niepotrzebnej kolumny Id

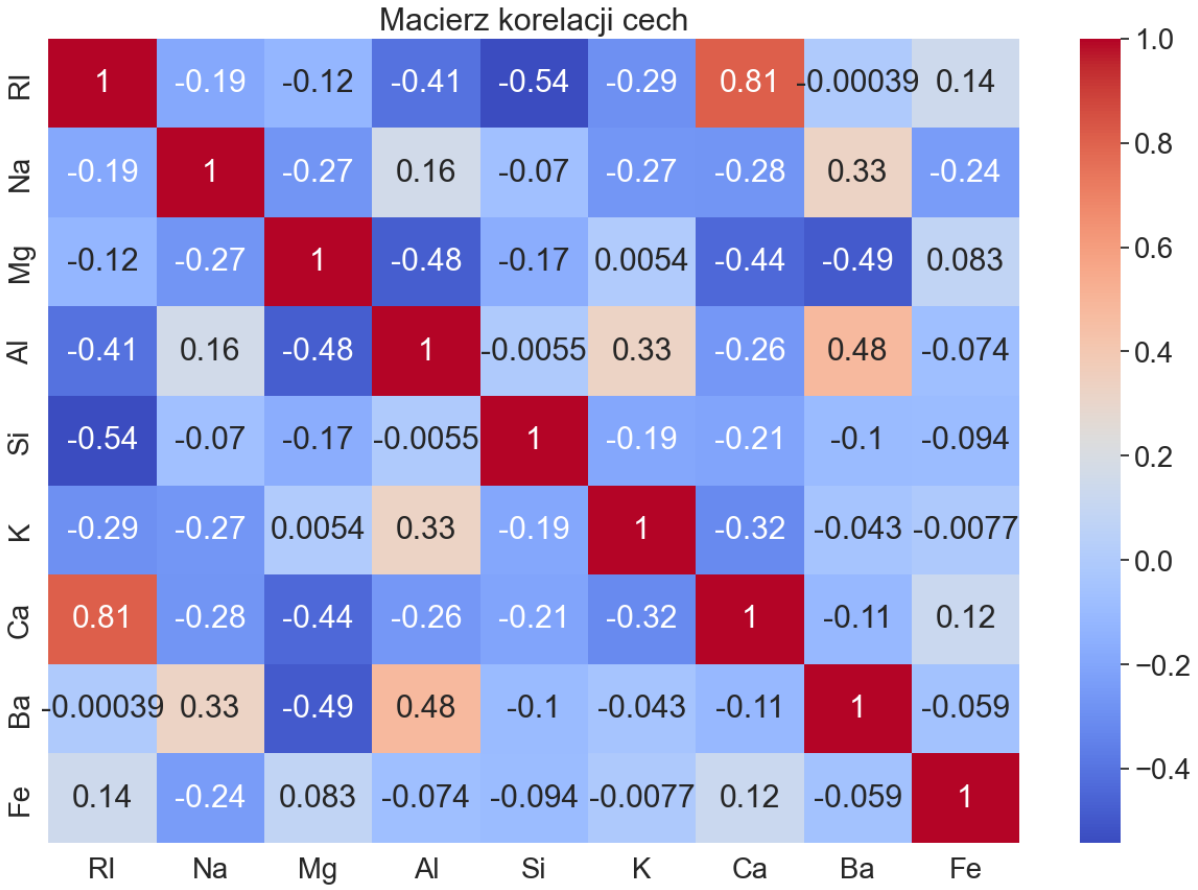
```
glass_data.drop(['Id'], axis=1, inplace=True)
glass_data.head(3)
```

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1

Odseparowanie kolumny klasy od reszty danych

```
y = glass_data['Type']

X = glass_data.drop('Type', axis=1)
```

Teraz możemy spojrzeć na macierz korelacji cech.



Jak widać RI oraz Ca mają wysoką dodatnią korelację, a RI oraz Si mają ujemną korelację.

Wysoka dodatnia korelacja między RI a Ca wskazuje, że wraz ze wzrostem wartości RI, można oczekiwać wzrostu wartości Ca. Oznacza to, że istnieje tendencja, że próbki szkła o wyższym współczynniku załamania światła będą miały również wyższą zawartość wapnia.

Ujemna korelacja między RI a Si sugeruje, że wzrost wartości RI wiąże się ze spadkiem wartości Si. Może to oznaczać, że szkła o wyższym współczynniku załamania światła są mniej skłonne do zawierania krzemionki.

Przetwarzanie danych:

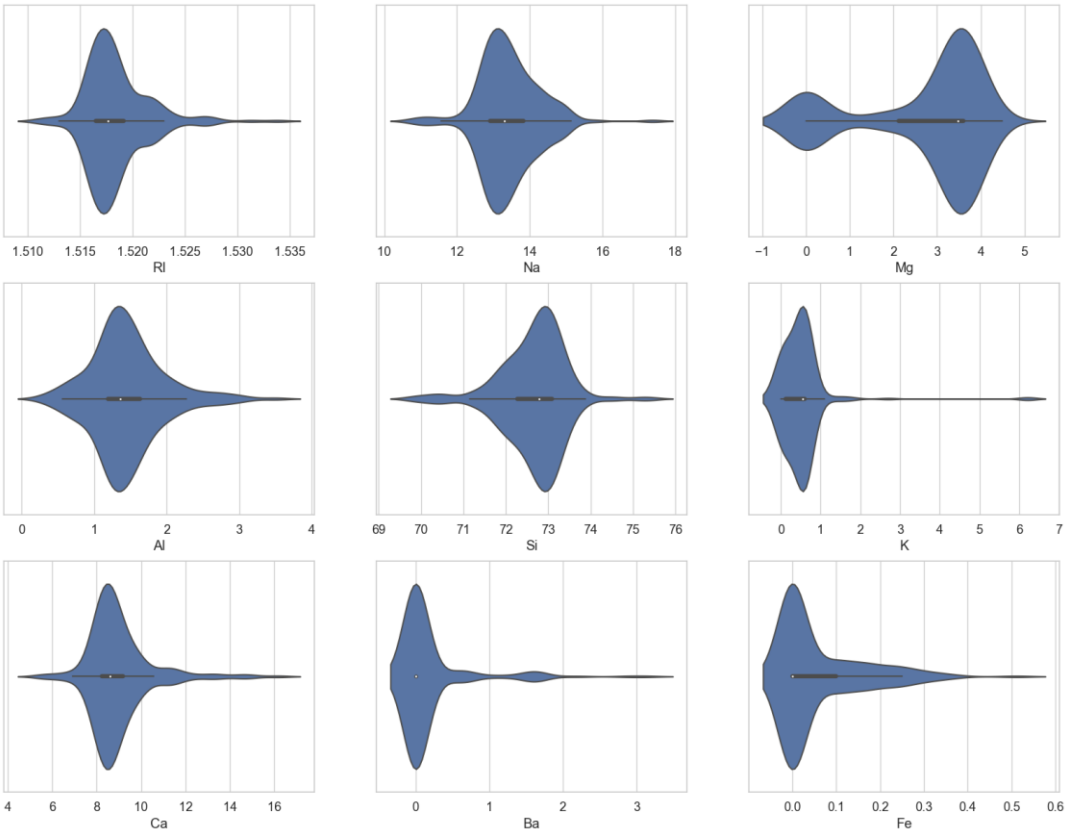
```
normalized_X = Normalizer().fit_transform(X)
normalized_X_DF = pd.DataFrame(normalized_X, columns=X.columns)
```

```
standarized_X = StandardScaler().fit_transform(X)standarized_X_DF =
pd.DataFrame(standarized_X, columns=X.columns)
```

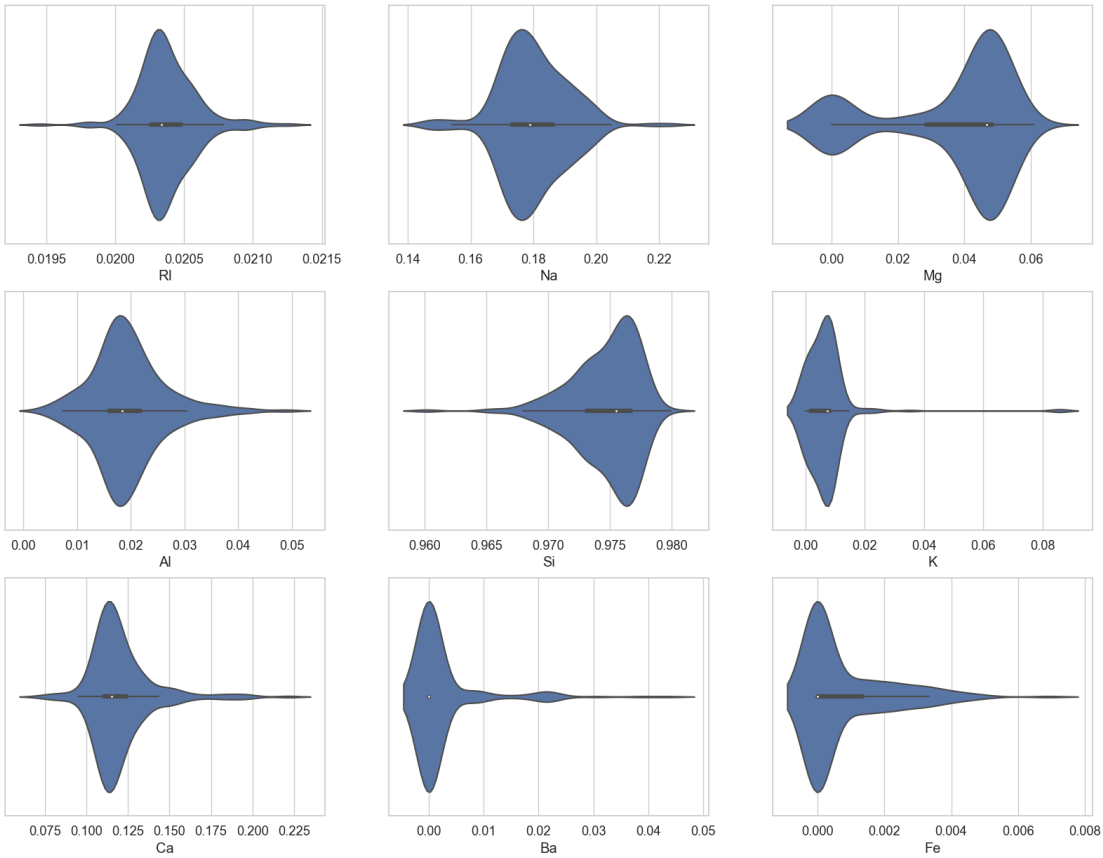
```
transformed_X = PowerTransformer(method='yeo-johnson').fit_transform(X)
transformed_X_DF = pd.DataFrame(transformed_X, columns=X.columns)
```

```
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X)
principal_DF = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
```

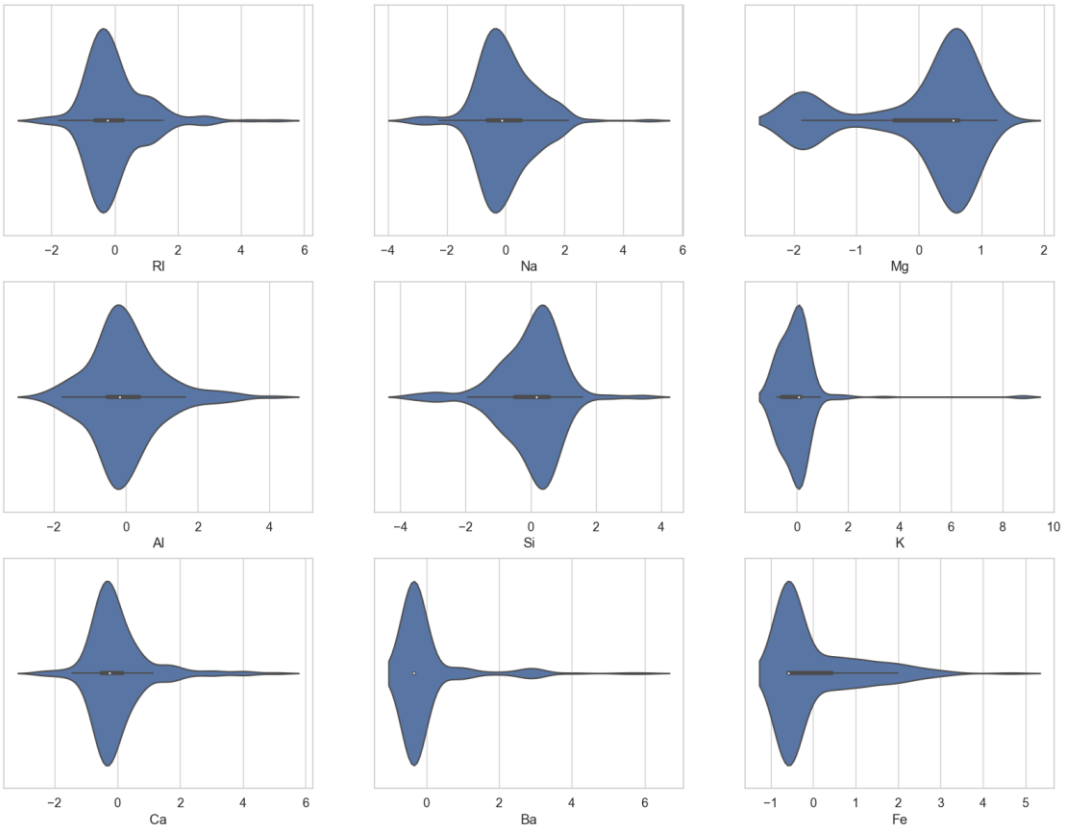
Zbiór bez przetworzenia:



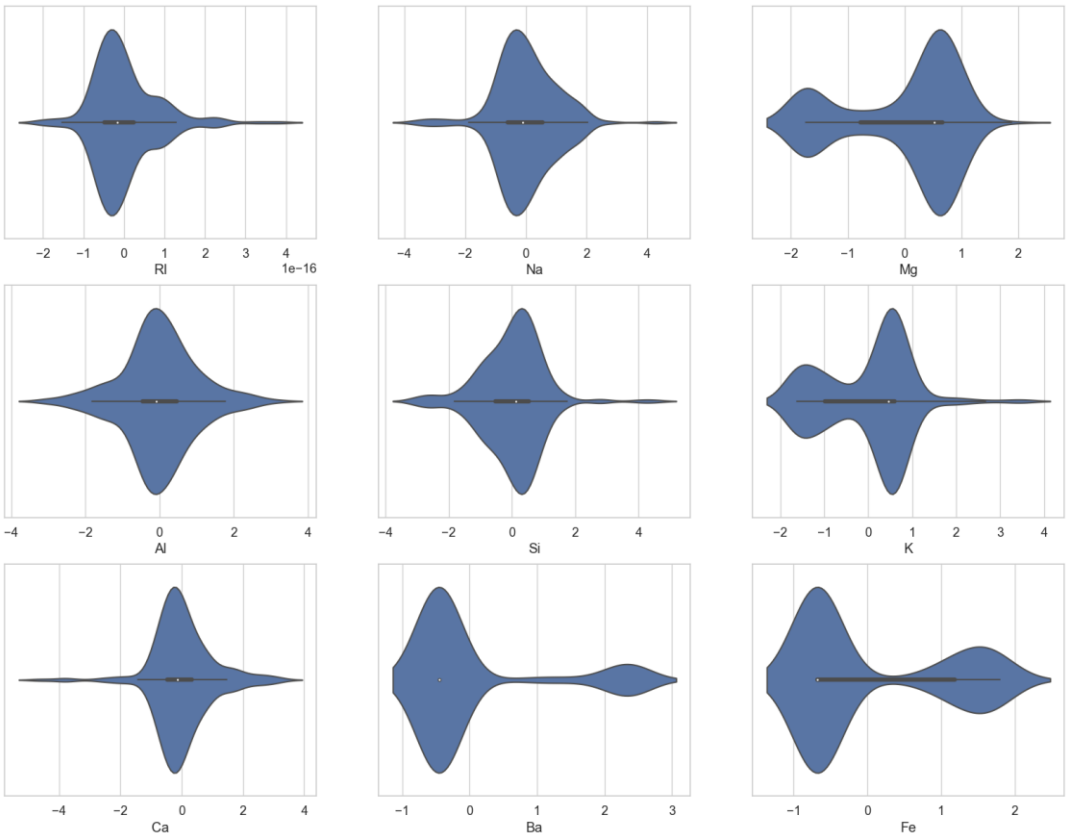
Normalizacja:



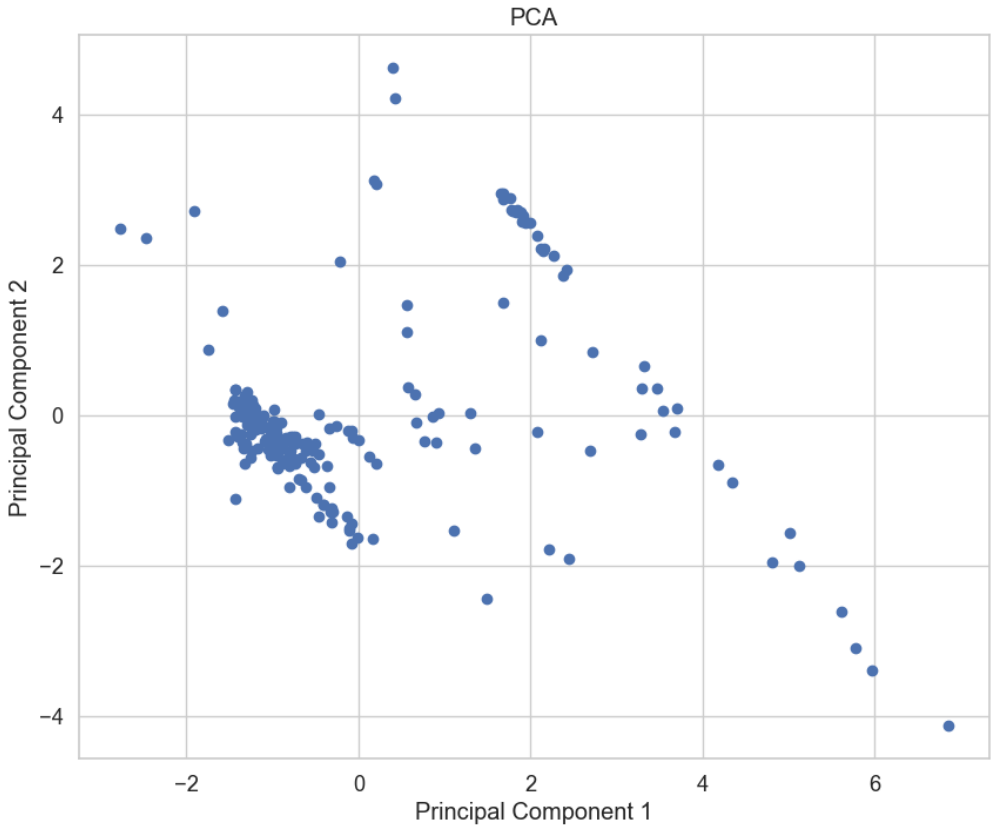
Standaryzacja:



Power Transform („yeo-johnson”)



Principal Component Analysis (2 komponenty)



Power transformation w trybie Yeo-Johnson to technika transformacji danych używana do przekształcenia zmiennych numerycznych w celu spełnienia założeń normalności lub innych wymagań modelu statystycznego.

Metoda Yeo-Johnson jest rozszerzeniem transformacji Boxa-Coxa, która umożliwia przekształcanie zarówno zmiennych o wartościach dodatnich, jak i ujemnych, a nie tylko dodatnich, jak w przypadku transformacji Boxa-Coxa.

Podzielenie danych przetworzonych na zbiór trenujący i walidacyjny:

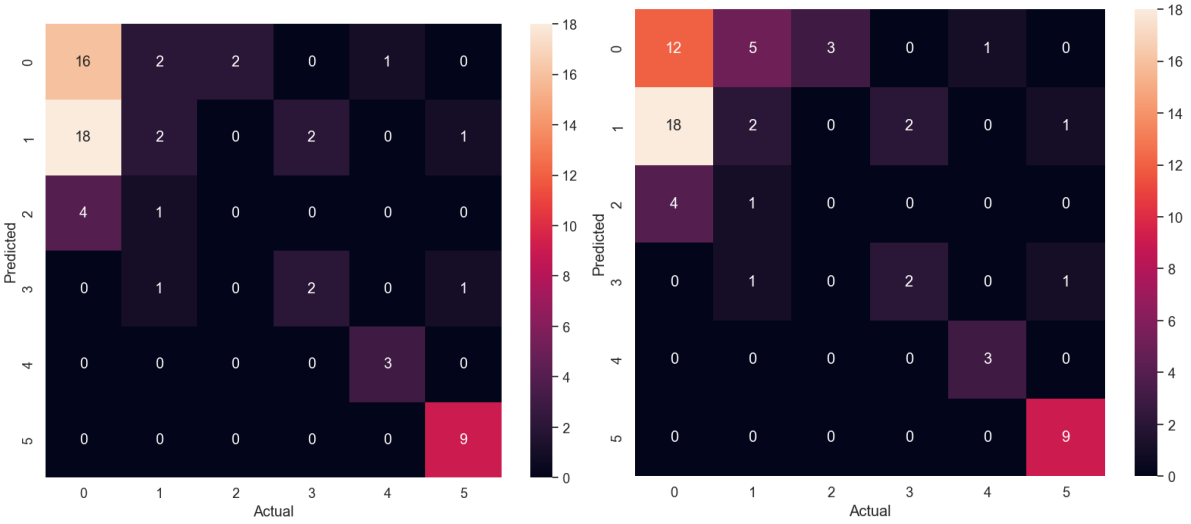
```
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, random_state=0, stratify=y)

X_train_norm, X_val_norm, y_train_norm, y_val_norm = train_test_split(normalized_X, y, test_size=0.3,
random_state=0, stratify=y)

X_train_stand, X_val_stand, y_train_stand, y_val_stand = train_test_split(standarized_X, y,
test_size=0.3, random_state=0, stratify=y)

X_train_transformed , X_val_transformed, y_train_transformed, y_val_transformed =
train_test_split(transformed_X, y, test_size=0.3, random_state=0, stratify=y)

X_train_pca, X_val_pca, y_train_pca, y_val_pca = train_test_split(principal_DF, y, test_size=0.3,
random_state=0, stratify=y)
```

Metoda testowania przyjętego modelu na przyjętych przetworzonych danych:

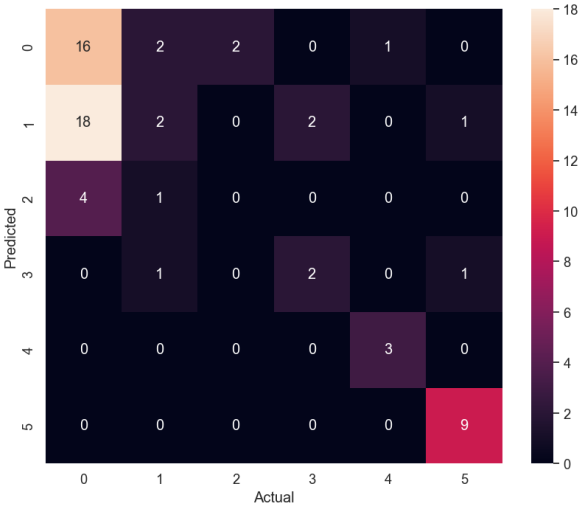
```
def train_and_test_model(X_train, X_val, y_train, y_val, model):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)

    accuracy = accuracy_score(y_val, y_pred)
    precision = precision_score(y_val, y_pred, average='weighted')
    recall = recall_score(y_val, y_pred, average='weighted')
    f1 = f1_score(y_val, y_pred, average='weighted')

    return accuracy, precision, recall, f1
```

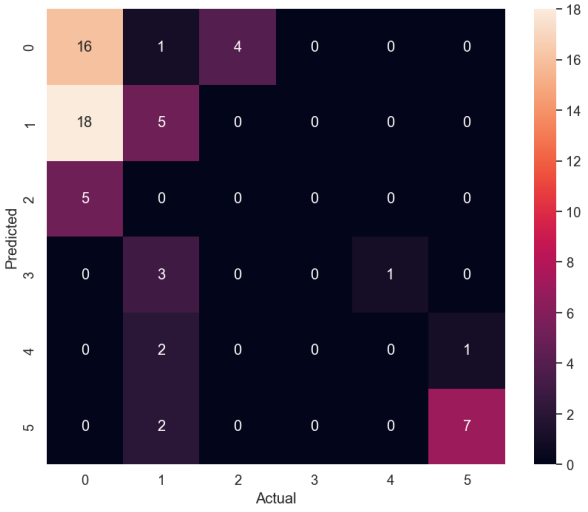
Porównanie wyników domyślnego klasyfikatora GaussianNB() dla danych nieprzetworzonych oraz przetworzonych:

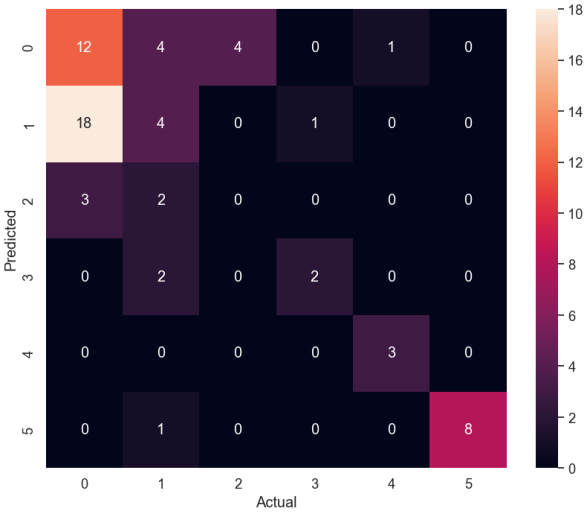
Macierze pomyłek dla Not processed, Normalized, Standarized, PCA, Transformed


Predicted
Actual
0 1 2 3 4 5
16 2 2 0 1 0
18 2 0 2 0 1
4 1 0 0 0 0
0 1 0 2 0 1
0 0 0 0 3 0
0 0 0 0 0 9
0 2 4 6 8 10 12 14 16 18

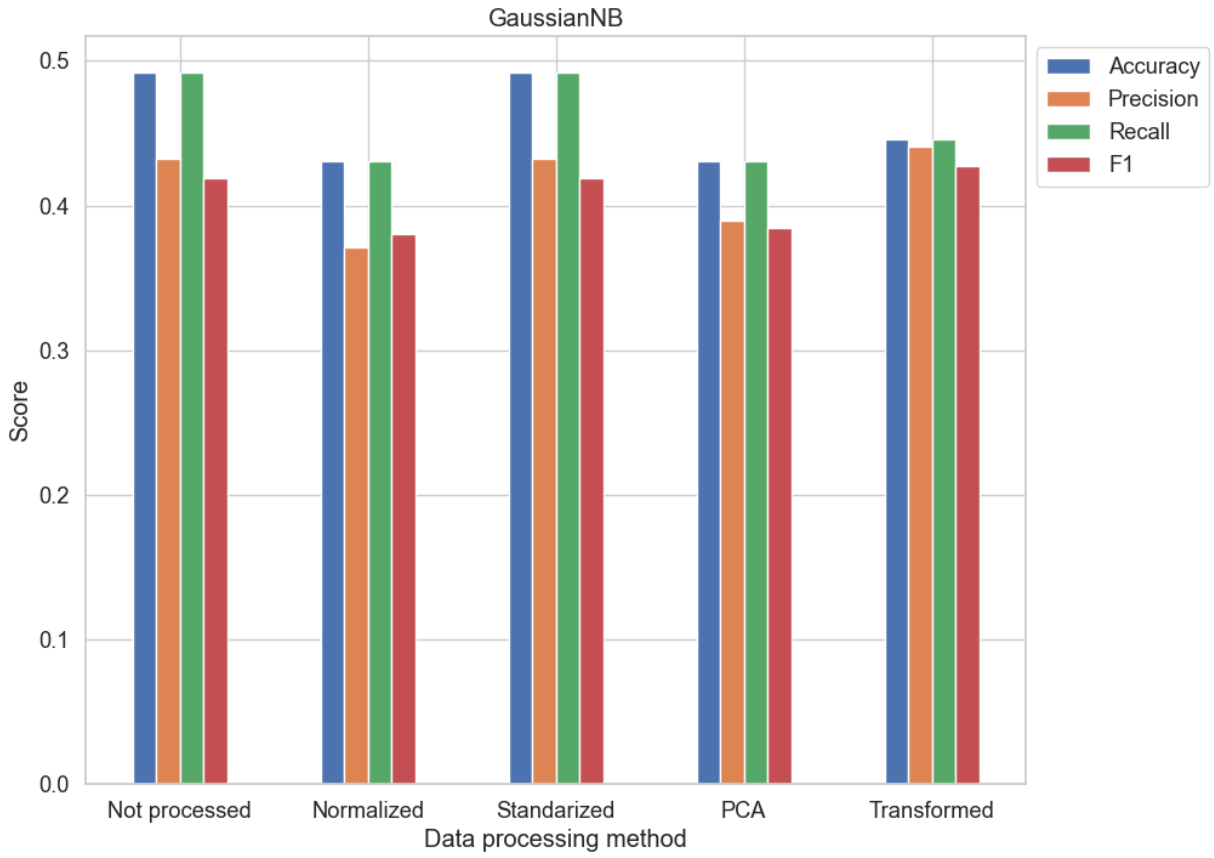
Predicted
Actual
0 1 2 3 4 5
12 5 3 0 1 0
18 2 0 2 0 1
4 1 0 0 0 0
0 1 0 2 0 1
0 0 0 0 3 0
0 0 0 0 0 9
0 2 4 6 8 10 12 14 16 18


Predicted
Actual
0 1 2 3 4 5
16 2 2 0 1 0
18 2 0 2 0 1
4 1 0 0 0 0
0 1 0 2 0 1
0 0 0 0 3 0
0 0 0 0 0 9
0 2 4 6 8 10 12 14 16 18


Predicted
Actual
0 1 2 3 4 5
16 1 4 0 0 0
18 5 0 0 0 0
5 0 0 0 0 0
0 3 0 0 1 0
0 2 0 0 0 1
0 2 0 0 0 7
0 2 4 6 8 10 12 14 16 18


Predicted
Actual
0 1 2 3 4 5
12 4 4 0 1 0
18 4 0 1 0 0
3 2 0 0 0 0
0 2 0 2 0 0
0 0 0 0 3 0
0 1 0 0 0 8
0 2 4 6 8 10 12 14 16 18

	Accuracy	Precision	Recall	F1
Not processed	0.492308	0.432652	0.492308	0.418980
Normalized	0.430769	0.371331	0.430769	0.380155
Standarized	0.492308	0.432652	0.492308	0.418980
PCA	0.430769	0.389793	0.430769	0.384625
Transformed	0.446154	0.440461	0.446154	0.427264



Wnioski:

Dla metody "Not processed" i "Standarized" osiągnięto takie same wyniki dla wszystkich miar oceny klasyfikacji. Oznacza to, że standaryzacja danych nie miała wpływu na wyniki klasyfikacji naiwnego klasyfikatora Bayesa.

Metoda "Normalized" wydaje się mieć najniższą dokładność (accuracy) i średnią wartość F1, co sugeruje, że normalizacja danych mogła wpływać negatywnie na działanie klasyfikatora.

Metoda "PCA" również osiągnęła podobne wyniki jak normalizacja, co wskazuje na to, że przetwarzanie danych za pomocą analizy składowych głównych nie poprawiło znacząco wyników klasyfikacji, pewnie dlatego, że mamy dość małą liczbę wymiarów, zatem ograniczanie jeszcze bardziej do 2 komponentów jest szkodliwe.

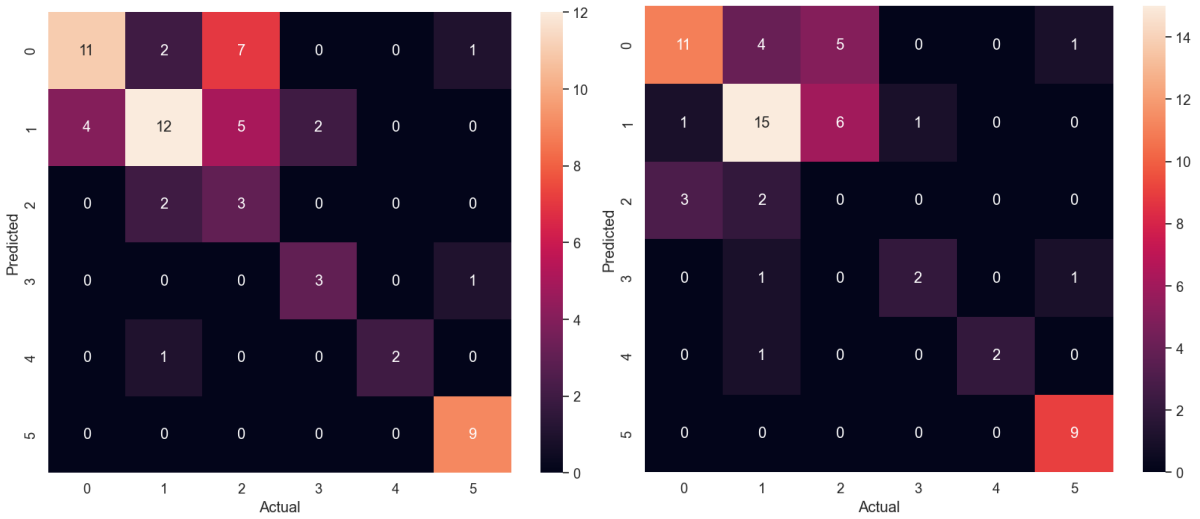
Metoda "Transformed" osiągnęła lepsze wyniki niż normalizacja i PCA, szczególnie w precyzji (precision) i wartości F1. Może to sugerować, że zastosowane przekształcenie danych w trybie „yeo-johnson” było bardziej odpowiednie dla klasyfikacji naiwnego klasyfikatora Bayesa.

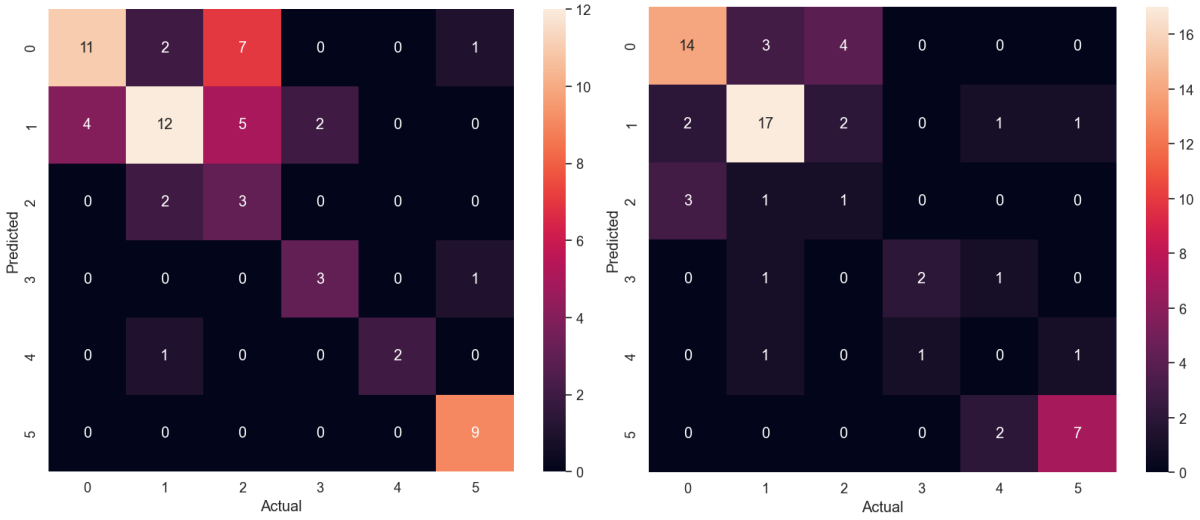
Politechnika Wrocławska

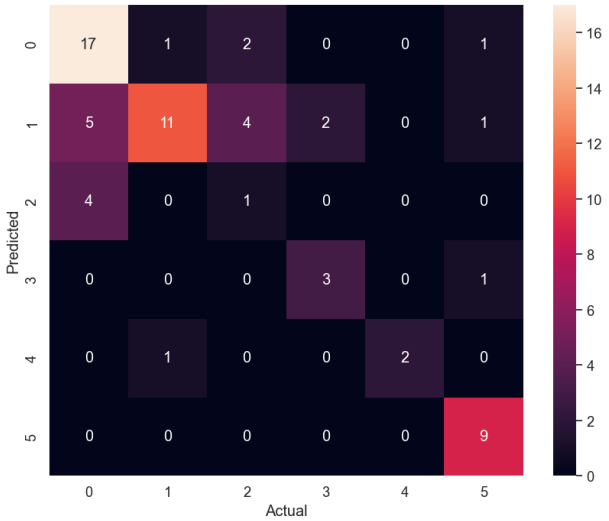
Autor: Filip Strózik 260377 Prowadzący: Mgr Inż. Michał Karol

Porównanie wyników domyślnego klasyfikatora DecisionTreeClassifier(random_state=0) dla danych nieprzetworzonych oraz przetworzonych:

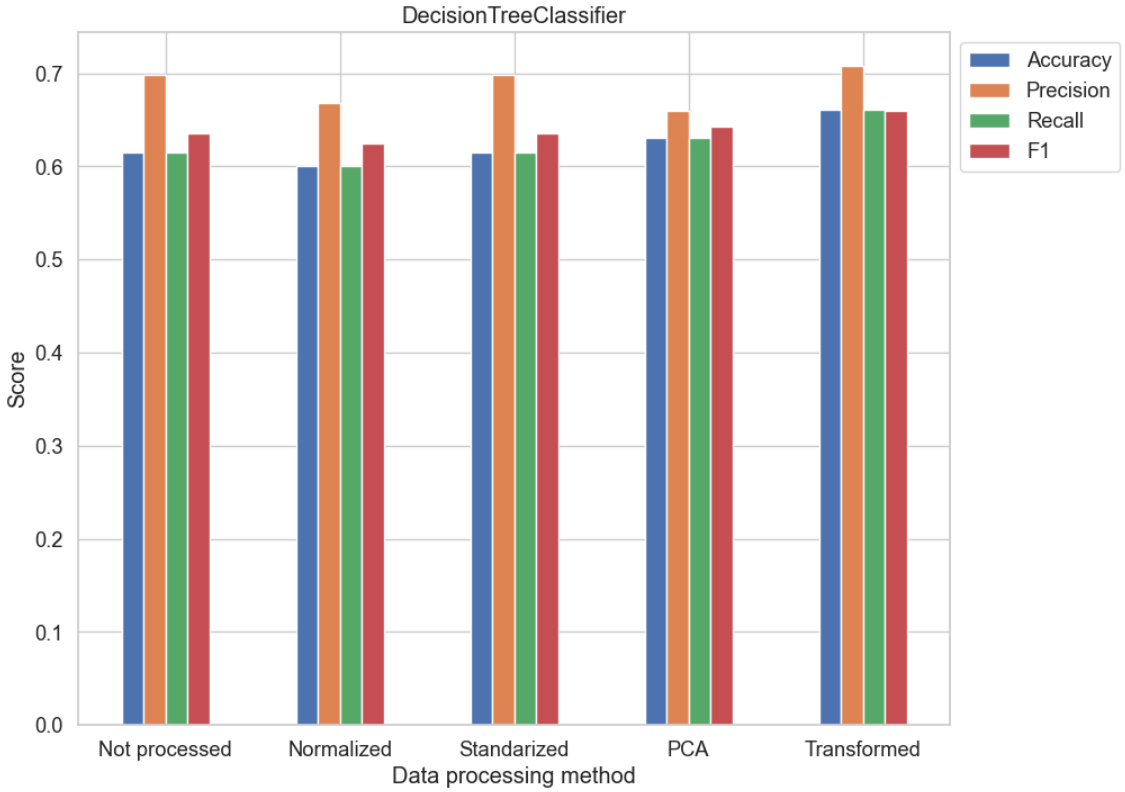
Macierze pomyłek dla: Not processed, Normalized, Standarized, PCA, Transformed







	Accuracy	Precision	Recall	F1
Not processed	0.615385	0.698445	0.615385	0.635385
Normalized	0.600000	0.668159	0.600000	0.624908
Standarized	0.615385	0.698445	0.615385	0.635385
PCA	0.630769	0.659302	0.630769	0.643370
Transformed	0.661538	0.708563	0.661538	0.659405



Wnioski:

W przypadku drzewa decyzyjnego, przetwarzanie danych miało mniejszy wpływ na wyniki klasyfikacji w porównaniu do naiwnego klasyfikatora Bayesa.

Metody "Not processed" i "Standarized" osiągnęły takie same wyniki dla wszystkich miar oceny klasyfikacji. Oznacza to, że standaryzacja danych nie miała wpływu na wyniki klasyfikacji drzewa decyzyjnego.

Metoda "Normalized" miała nieznacznie niższą dokładność (accuracy) i wartość F1 oraz znacznie mniejszą precyzję (precision) w porównaniu do innych metod. Może to sugerować, że normalizacja danych mogła wpływać negatywnie na działanie drzewa decyzyjnego.

Metoda "PCA" osiągnęła nieco lepsze wyniki niż inne metody w większości miar oceny klasyfikacji oprócz prezycji, której wynik był niższy. Oznacza to, że przetwarzanie danych za pomocą analizy składowych głównych mogło poprawić wyniki klasyfikacji drzewa decyzyjnego.

Metoda "Transformed" osiągnęła najwyższą dokładność, precyzję i wartość F1 spośród wszystkich metod. Wydaje się, że przekształcenie danych miało pozytywny wpływ na działanie drzewa decyzyjnego.

klasyfikacja – przetestuj klasyfikatory i zbadaj wpływ na wyniki: naiwny klasyfikator Bayesa oraz drzewo decyzjne używając przynajmniej 3 różnych zestawów hiperparametrów. (40 punktów)

Dobrym wyborem danych, na których będziemy porównywać zestawy hiperparametrów dla badanych klasyfikatorów byłoby użycie danych pochodzących z transformacji Power Transform („yeo-johnson"). Aczkolwiek trzeba zauważyć, że dla różnego przetworzenia w raz ze zmianami hiper parametrów inaczej zachowują się wyniki klasyfikatorów.

GaussianNB:

```
def nb_classifiers_create(params):
    nb_params = params
    nb_classifiers = []

    for param in nb_params:
        nb = GaussianNB(var_smoothing=param)
        nb_classifiers.append(nb)

    return nb_classifiers
```

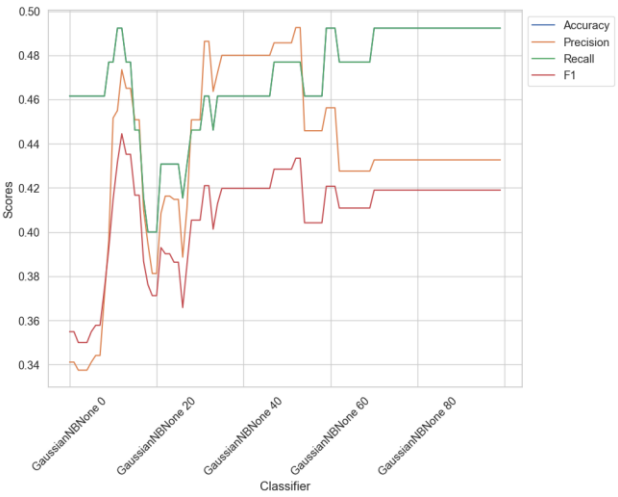
Hiperparametr var_smoothing jest używany w przypadku klasyfikatora GaussianNB do regularyzacji. Pomaga w uniknięciu problemów z zerowymi lub bliskimi zeru wariancjami, które mogą wystąpić w danych treningowych. Parametr var_smoothing kontroluje siłę regularyzacji poprzez dodanie stałej wartości do estymowanej wariancji każdej cechy. Im większa wartość var_smoothing, tym większa regularyzacja i większe ujednolicenie estymowanych wariancji.

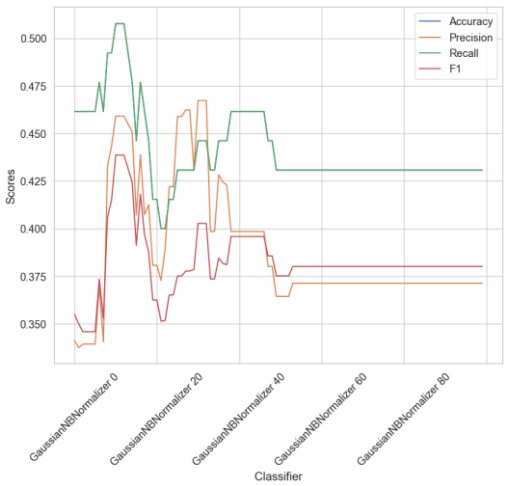
Tworzenie listy klasyfikatorów z różnymi wartościami var_smoothing ma na celu porównanie wyników klasyfikacji dla różnych poziomów regularyzacji. Dzięki temu można zidentyfikować optymalną wartość var_smoothing, która prowadzi do najlepszych wyników klasyfikacji dla konkretnego problemu i zestawu danych. Przez iterację po różnych wartościach var_smoothing można sprawdzić, jak różne poziomy regularyzacji wpływają na wyniki klasyfikacji i wybrać ten, który daje najlepsze wyniki.

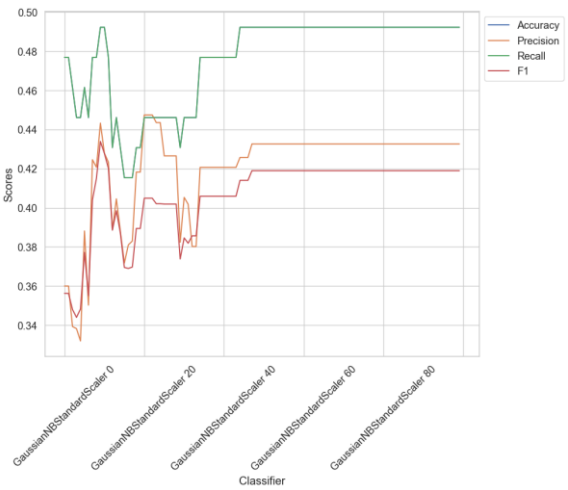
```
def iterate_classifiers(classifiers, X_train, X_val, y_train, y_val):

    for name, X_processed_train, X_processed_val, y_processed_train, y_processed_val in processing():
        results = pd.DataFrame(columns=['Accuracy', 'Precision', 'Recall', 'F1'])
        index = 0
        best_iter = 0
        best_accuracy = 0
        for classifier in classifiers:
            accuracy, precision, recall, f1 = train_and_test_model(X_processed_train, X_processed_val,
y_processed_train, y_processed_val, classifier)
            results.loc[classifier.__class__.__name__ + str(name) + ' ' +  str(index)] = [accuracy,
precision, recall, f1]
            if accuracy > best_accuracy:
                best_accuracy = accuracy
                best_iter = index
            index += 1
        print('Best iteration: ', best_iter)
```

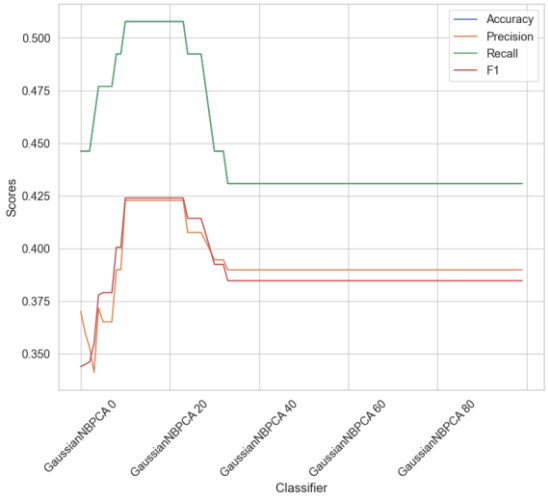
```
        results.plot(kind='line', figsize=(10, 8))
        plt.xlabel('Classifier')
        plt.ylabel('Scores')
        plt.xticks(rotation=45)
        plt.legend(bbox_to_anchor=(1.0, 1.0))
```

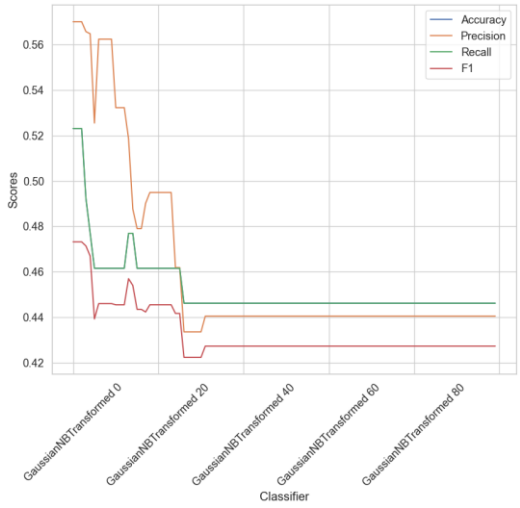
```
import numpy as np
nb_params = np.logspace(0, -9, num=100)
nb_classifiers = nb_classifiers_create(nb_params)
print('Naive Bayes')
iterate_classifiers(nb_classifiers, X_train_transformed,
X_val_transformed, y_train_transformed, y_val_transformed)
```







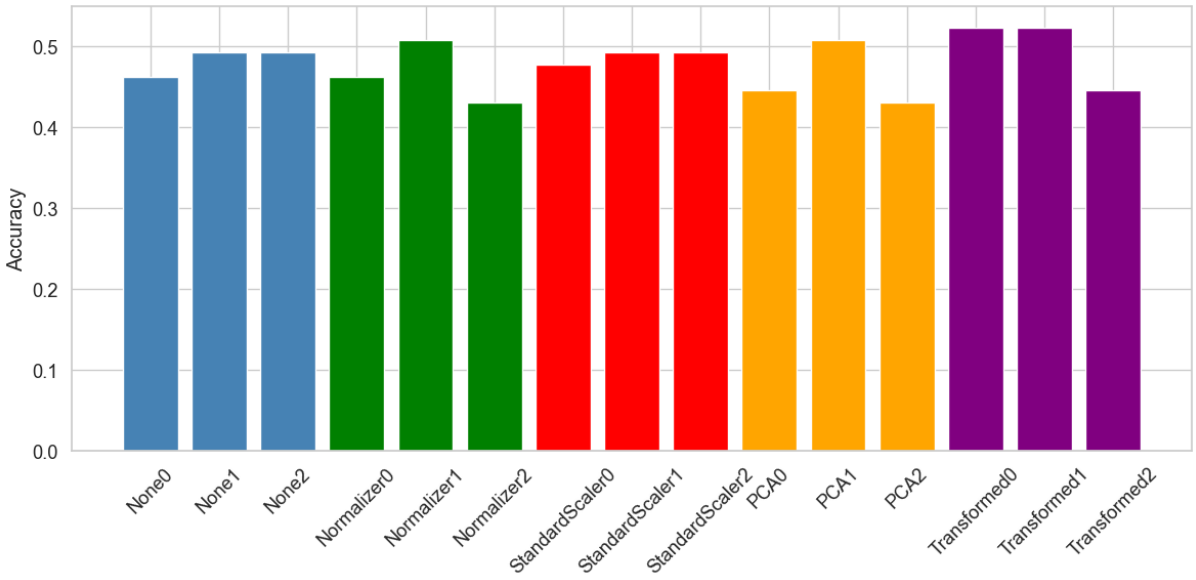


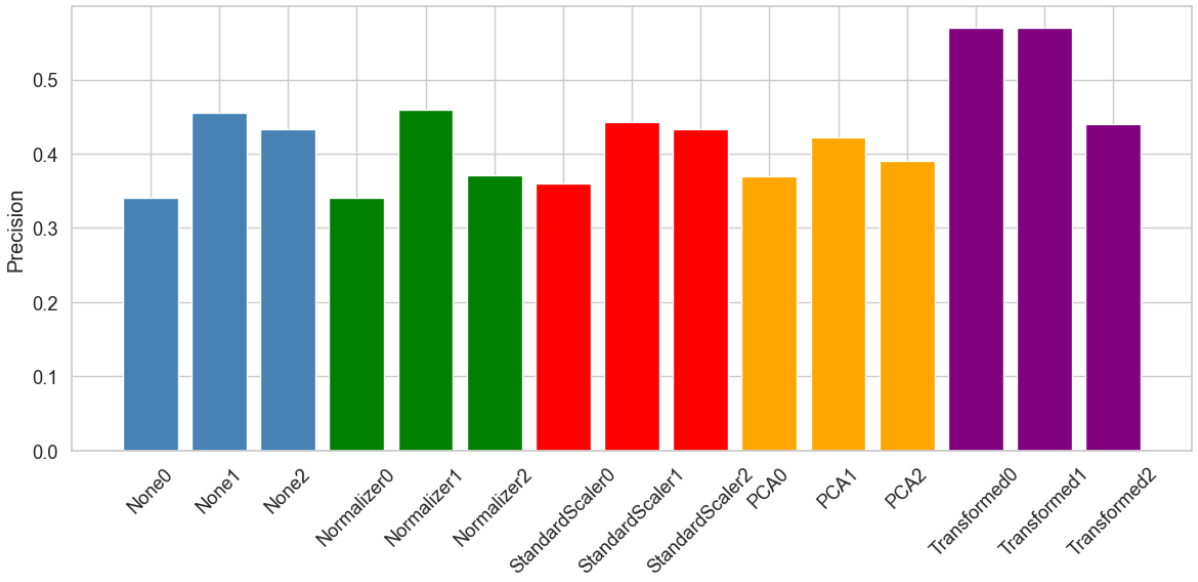


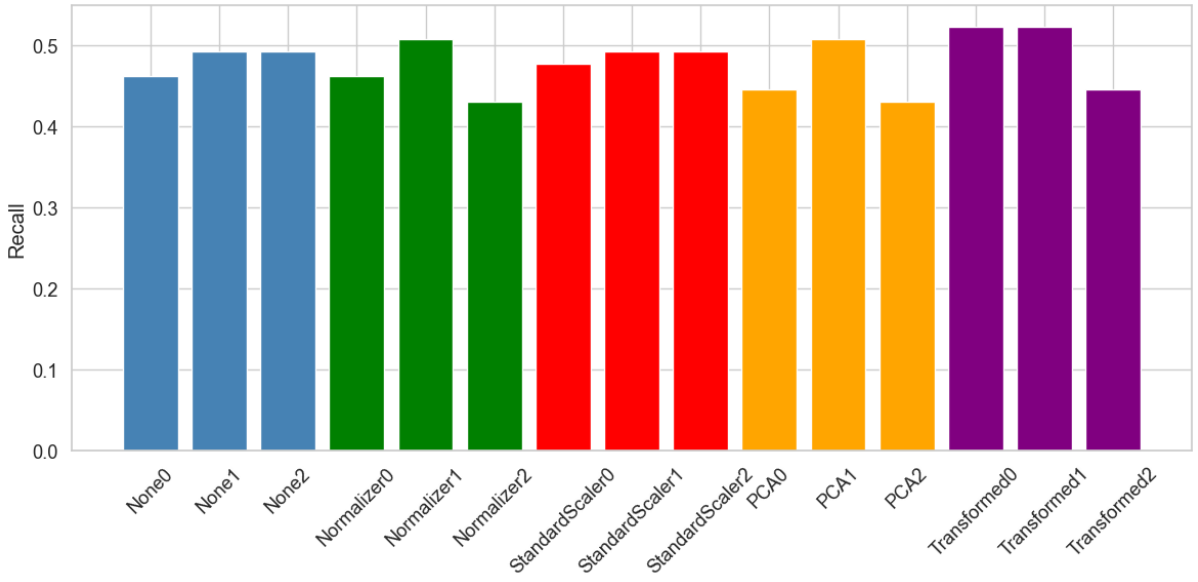
Widać, że nie zawsze bardzo mała wartość regularyzacji wpływa pozytywnie na wyniki. Domyślnie jest wynosi 1e-9 zatem wyniki z poprzedniego etapu możnabyłoby polepszyc jak i pogorszyć dobierając odpowienie wartosci tego parametru.

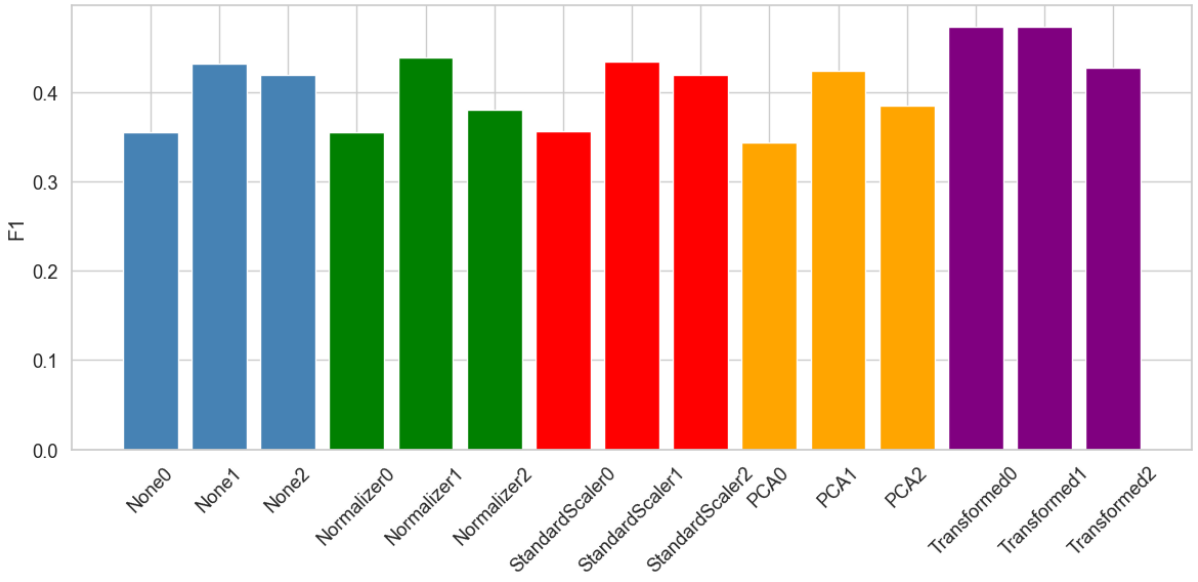
Porównajmy wyniki każdego przetworzenia z zestawem trzech wartości hiperparametru: pirewszy - 0, najlepszy dla przetworzenia - 1 oraz ostatnim(domyślnym) - 2:

	Classifier	Accuracy	Precision	Recall	F1
0	None0	0.461538	0.341132	0.461538	0.354913
1	None1	0.492308	0.455000	0.492308	0.431603
2	None2	0.492308	0.432652	0.492308	0.418980
3	Normalizer0	0.461538	0.341132	0.461538	0.354913
4	Normalizer1	0.507692	0.459114	0.507692	0.438654
5	Normalizer2	0.430769	0.371331	0.430769	0.380155
6	StandardScaler0	0.476923	0.359994	0.476923	0.356231
7	StandardScaler1	0.476923	0.359994	0.476923	0.356231
8	StandardScaler2	0.492308	0.432652	0.492308	0.418980
9	PCA0	0.446154	0.370034	0.446154	0.343829
10	PCA1	0.507692	0.422786	0.507692	0.423883
11	PCA2	0.430769	0.389793	0.430769	0.384625
12	Transformed0	0.523077	0.570105	0.523077	0.473173
13	Transformed1	0.523077	0.570105	0.523077	0.473173
14	Transformed2	0.446154	0.440461	0.446154	0.427264




Precision
0.0
0.1
0.2
0.3
0.4
0.5
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2


Recall
0.0
0.1
0.2
0.3
0.4
0.5
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2


F1
0.0
0.1
0.2
0.3
0.4
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2

DecisionTreeClassifier:

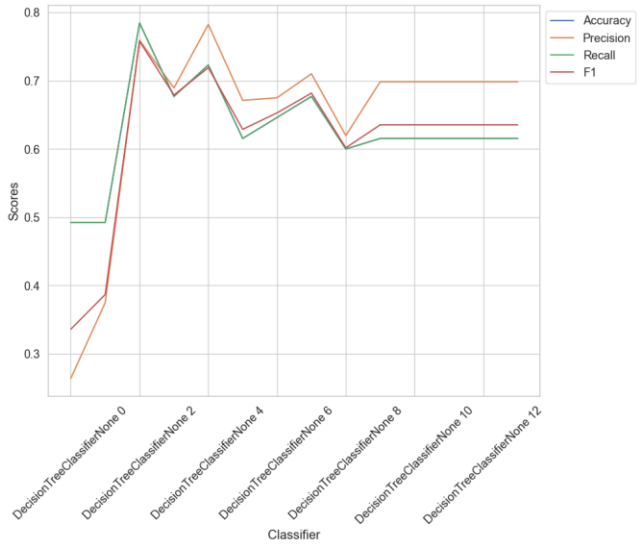
```
def dt_classifiers_create(dt_params):
    dt_classifiers = []
    for params in dt_params:
        dt =
DecisionTreeClassifier(criterion=params.get('criterion'),
max_depth=params.get('max_depth'),
                                    min_samples_split=params.get(
'min_samples_split'), random_state=0)
        dt_classifiers.append(dt)

    return dt_classifiers
```

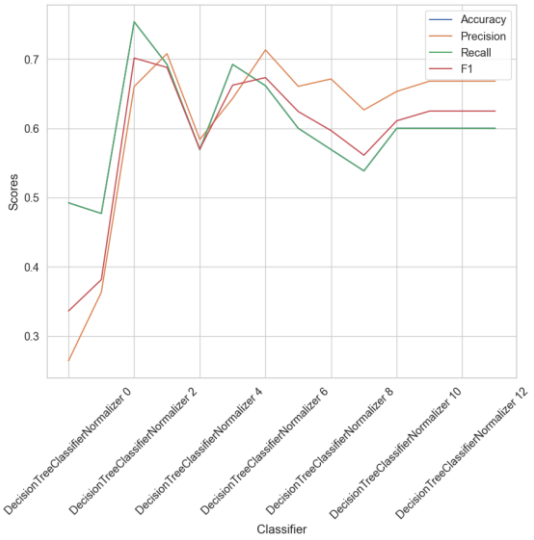
W wyołaniu funkcji będziemy odpowiednio wstawiać parametry, których będziemy zmieniać wartości:

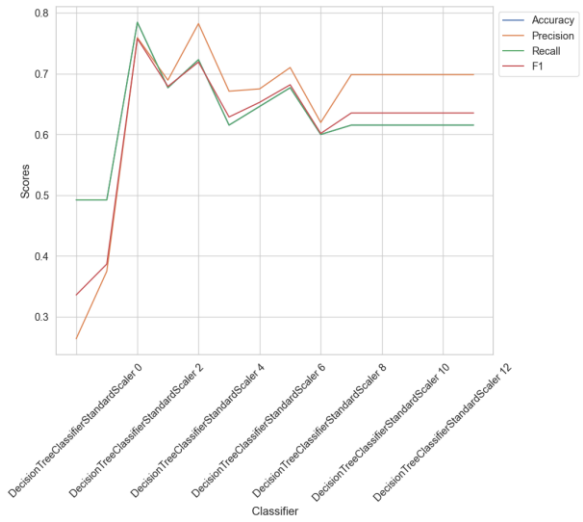
Parametr max_depth określa maksymalną głębokość drzewa decyzyjnego. Głębokość drzewa oznacza liczbę poziomów decyzji, które drzewo może podjąć, zaczynając od korzenia. Domyślnie max_depth=None, co oznacza, że drzewo będzie rosło, aż wszystkie liście będą czyste (zawierają tylko jedną klasę) lub nie zostaną spełnione inne kryteria stopu. Ustalenie wartości max_depth może pomóc w uniknięciu przeuczenia (overfittingu) i kontrolować złożoność drzewa. Zbyt duża wartość max_depth może prowadzić do zbyt skomplikowanego modelu, który dopasowuje się do szumów w danych, podczas gdy zbyt mała wartość max_depth może prowadzić do niedouczenia (underfittingu), gdy model nie może uchwycić wystarczająco złożonych zależności w danych.

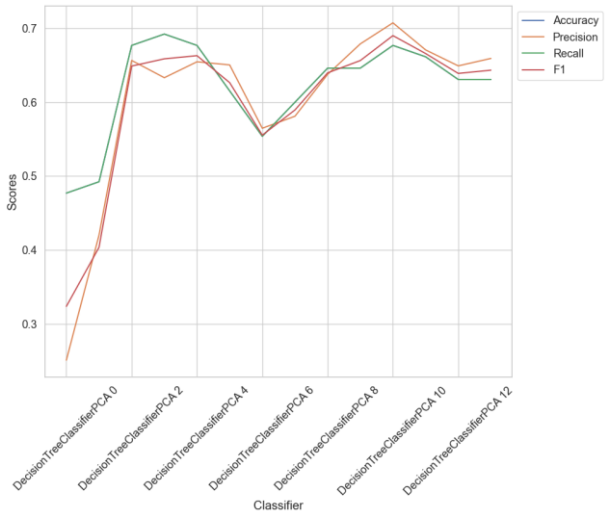
min_samples_split: Parametr min_samples_split określa minimalną liczbę próbek wymaganą do podziału wewnętrznego w węźle. Domyślnie min_samples_split=2, co oznacza, że węzeł może być dzielony tylko wtedy, gdy ma co najmniej 2 próbki. Ustawienie większej wartości min_samples_split może prowadzić do bardziej ogólnych podziałów, co pomaga w redukcji przeuczenia. Zbyt mała wartość min_samples_split może prowadzić do podziałów, które są dopasowane do pojedynczych próbek lub szumów w danych.

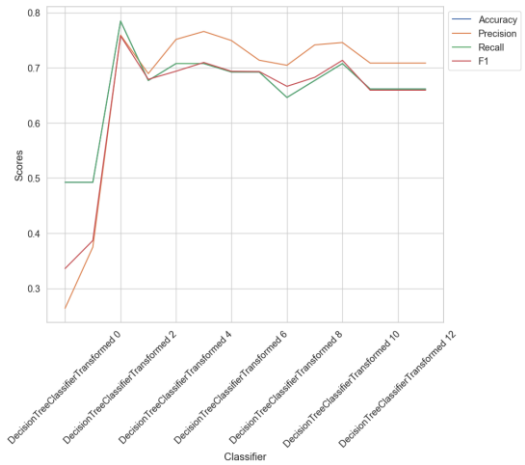
```
dt_params = []
for i in range(1, 15):
    dt_params.append({'max_depth': i, 'min_samples_split': 2,
'criterion': 'gini'})

dt_classifiers = dt_classifiers_create(dt_params)
iterate_classifiers(dt_classifiers)
```





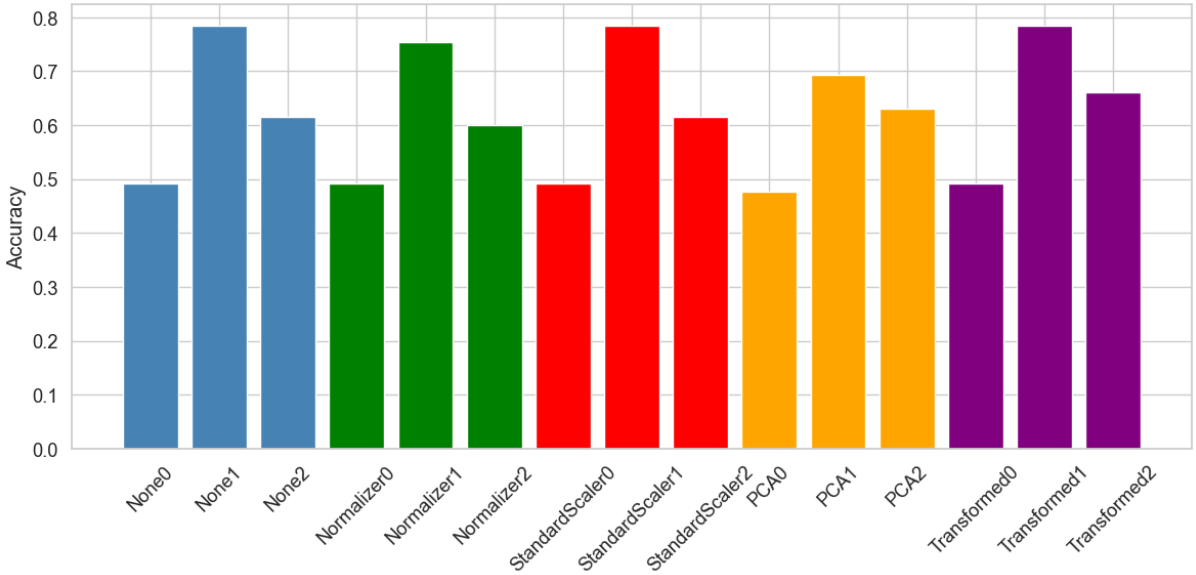


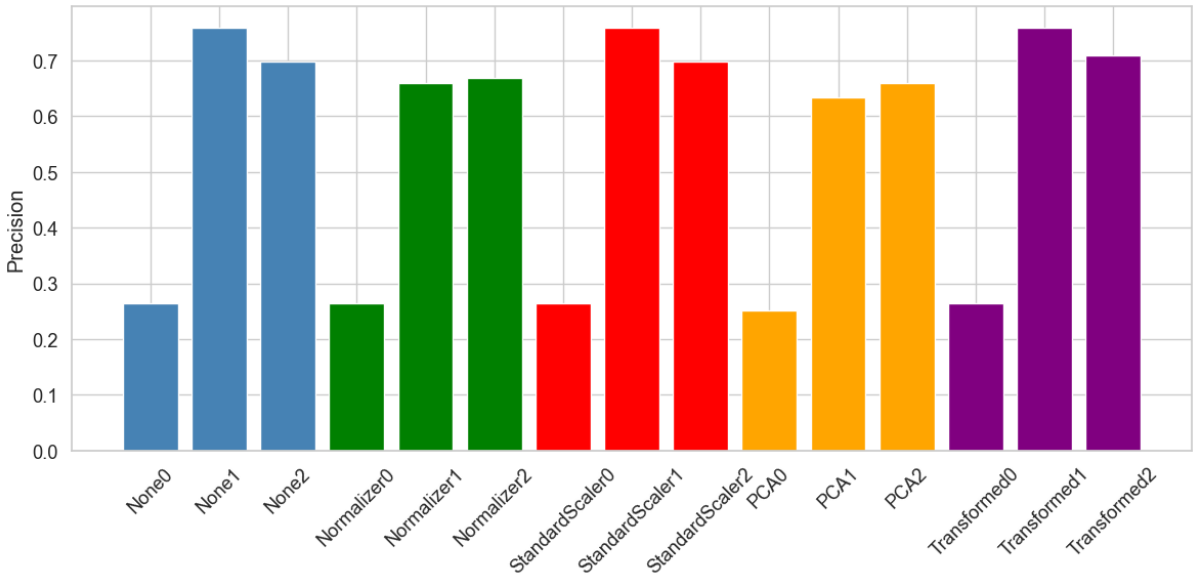


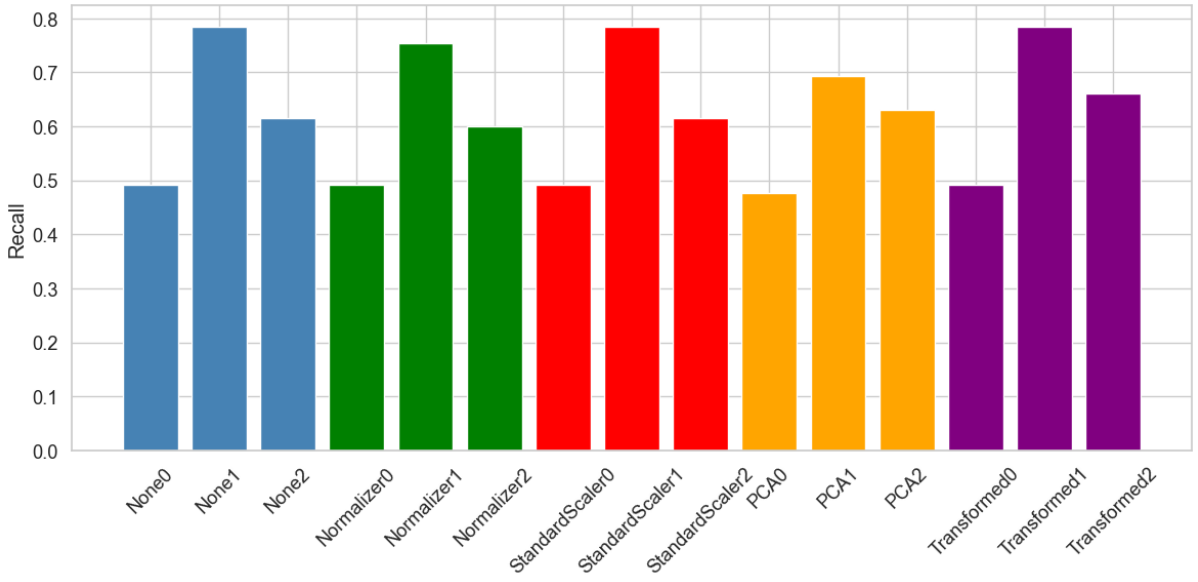


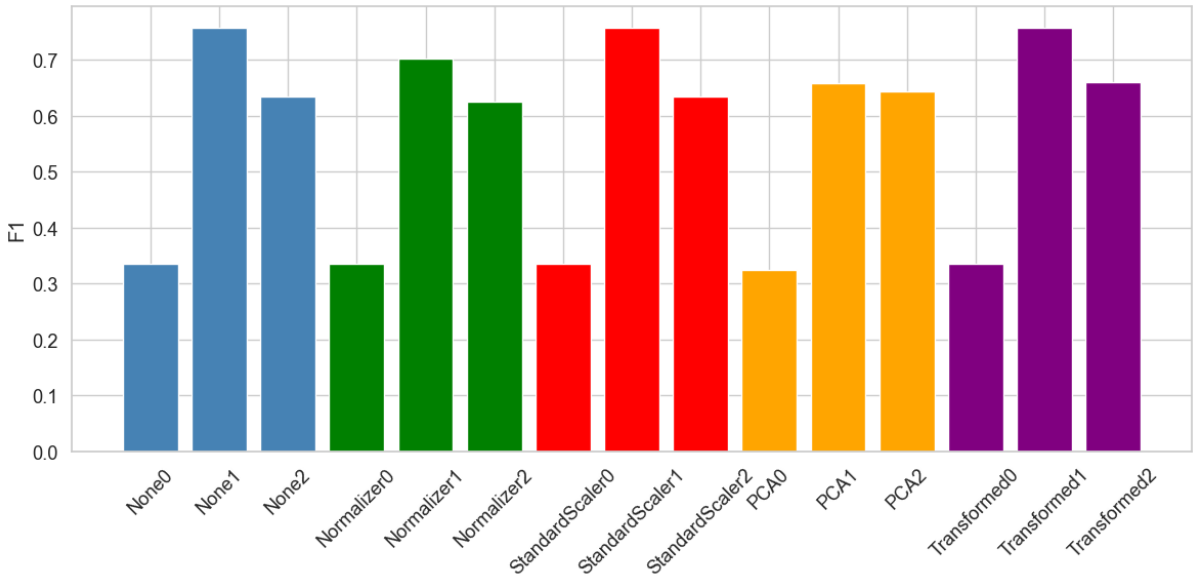
Widać, że dla pierwszych wartości występuje underfitting, zbyt ogólne dopasowanie. Z drugiej strony dla większych głębokości występuje overfitting i model będzie mógł zapamiętać dane treningowe przez co wyniki mogą być gorsze niż dla mniejszych głębokości.

	Classifier	Accuracy	Precision	Recall	F1
0	None0	0.492308	0.263999	0.492308	0.336004
1	None1	0.784615	0.759318	0.784615	0.757769
2	None2	0.615385	0.698445	0.615385	0.635385
3	Normalizer0	0.492308	0.263999	0.492308	0.336004
4	Normalizer1	0.753846	0.660295	0.753846	0.701538
5	Normalizer2	0.600000	0.668159	0.600000	0.624908
6	StandardScaler0	0.492308	0.263999	0.492308	0.336004
7	StandardScaler1	0.784615	0.759318	0.784615	0.757769
8	StandardScaler2	0.615385	0.698445	0.615385	0.635385
9	PCA0	0.476923	0.250726	0.476923	0.323540
10	PCA1	0.692308	0.633210	0.692308	0.658786
11	PCA2	0.630769	0.659302	0.630769	0.643370
12	Transformed0	0.492308	0.263999	0.492308	0.336004
13	Transformed1	0.784615	0.759318	0.784615	0.757769
14	Transformed2	0.661538	0.708563	0.661538	0.659405



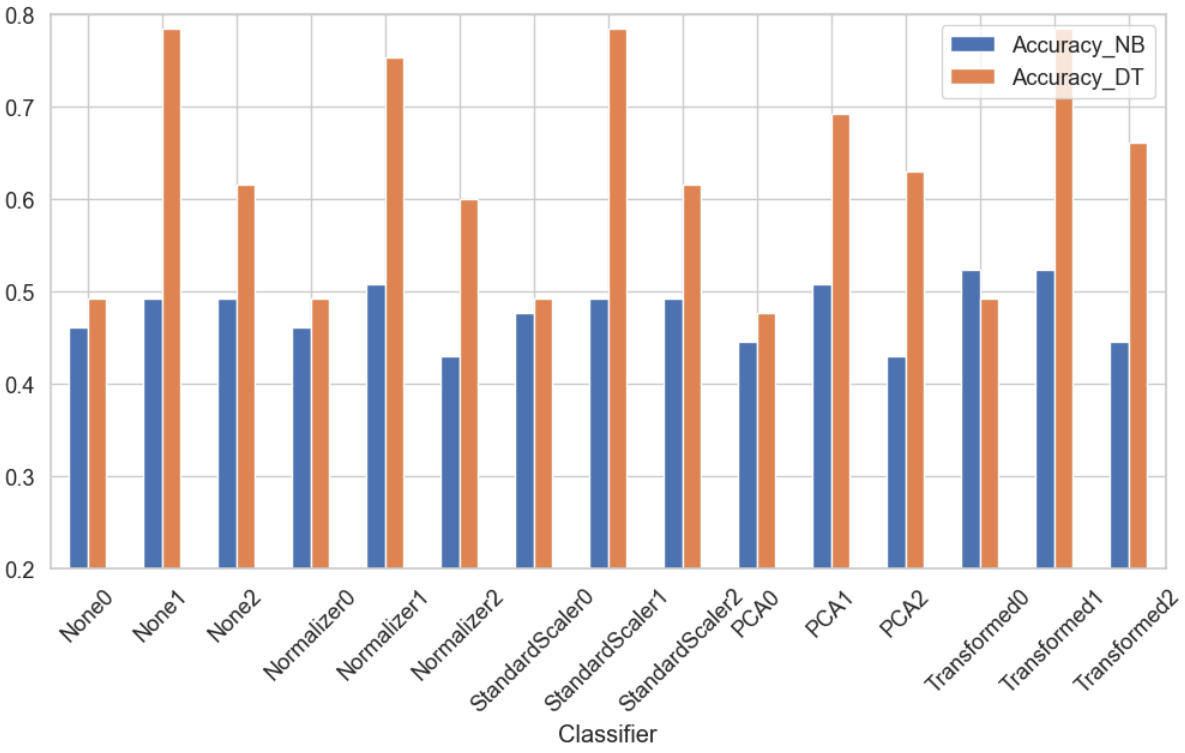

Precision
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2

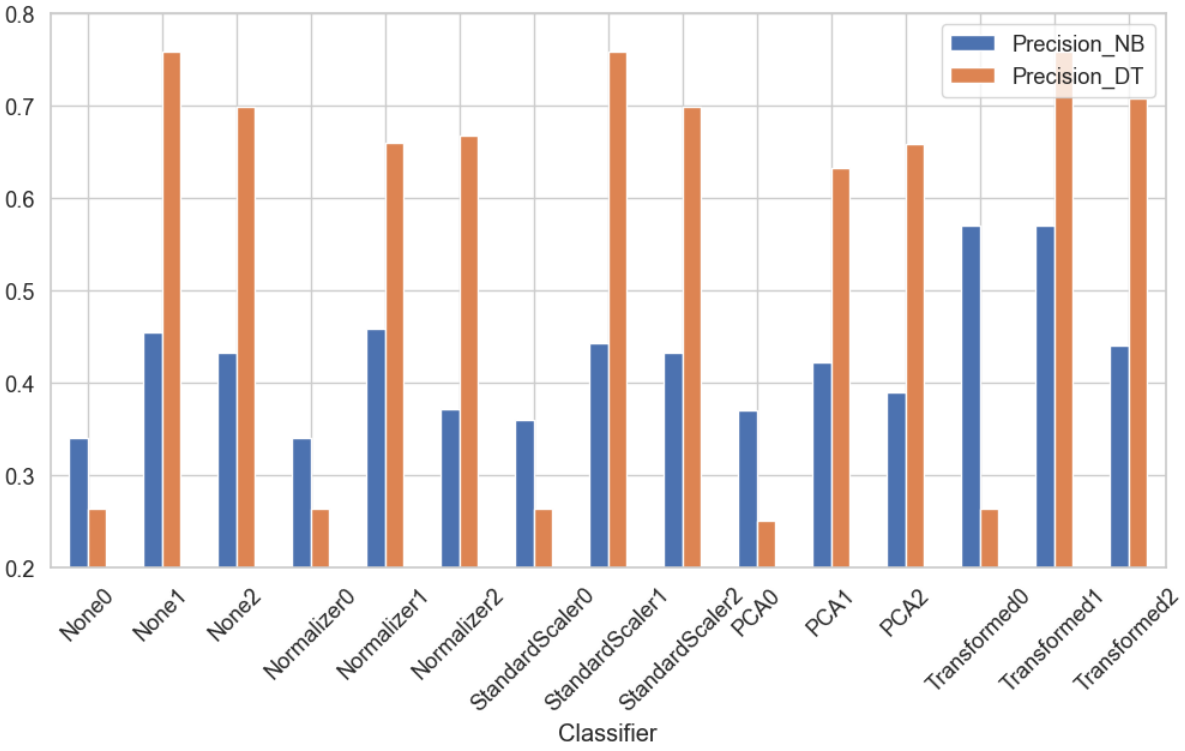

Recall
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2

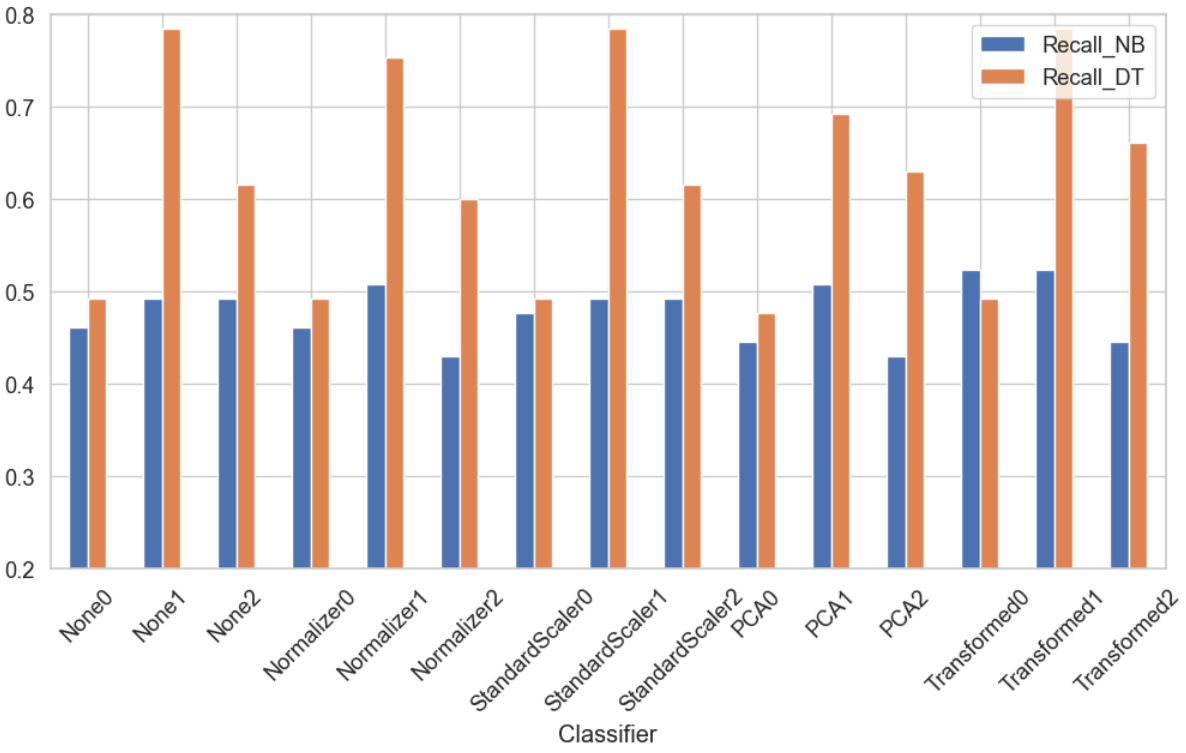

F1
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2

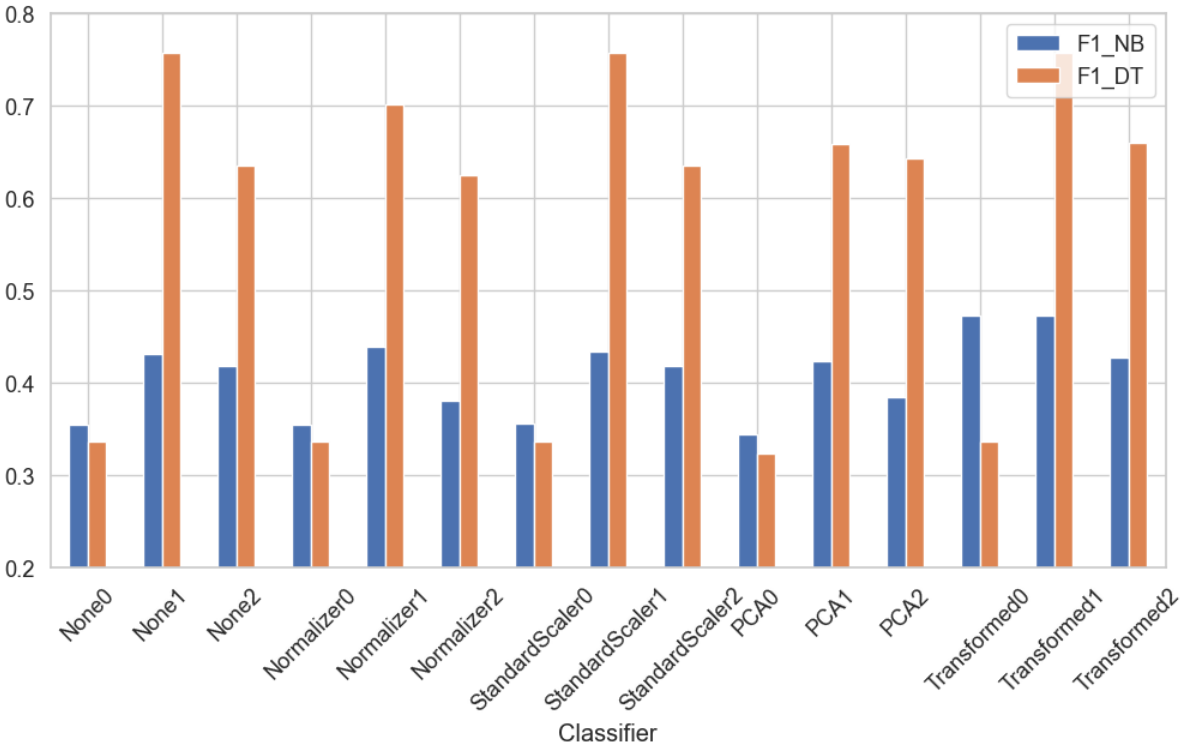
ocena klasyfikacji – do porównania wyników różnego typu przygotowania danych oraz wykorzystanego klasyfikatora użyj poznanych metryk oceny klasyfikacji i zinterpretuj wyniki. (20 punktów)

	Classifier	Accuracy_NB	Precision_NB	Recall_NB	F1_NB	Accuracy_DT	Precision_DT	Recall_DT	F1_DT
0	None0	0.461538	0.341132	0.461538	0.354913	0.492308	0.263999	0.492308	0.336004
1	None1	0.492308	0.455000	0.492308	0.431603	0.784615	0.759318	0.784615	0.757769
2	None2	0.492308	0.432652	0.492308	0.418980	0.615385	0.698445	0.615385	0.635385
3	Normalizer0	0.461538	0.341132	0.461538	0.354913	0.492308	0.263999	0.492308	0.336004
4	Normalizer1	0.507692	0.459114	0.507692	0.438654	0.753846	0.660295	0.753846	0.701538
5	Normalizer2	0.430769	0.371331	0.430769	0.380155	0.600000	0.668159	0.600000	0.624908
6	StandardScaler0	0.476923	0.359994	0.476923	0.356231	0.492308	0.263999	0.492308	0.336004
7	StandardScaler1	0.492308	0.443287	0.492308	0.433926	0.784615	0.759318	0.784615	0.757769
8	StandardScaler2	0.492308	0.432652	0.492308	0.418980	0.615385	0.698445	0.615385	0.635385
9	PCA0	0.446154	0.370034	0.446154	0.343829	0.476923	0.250726	0.476923	0.323540
10	PCA1	0.507692	0.422786	0.507692	0.423883	0.692308	0.633210	0.692308	0.658786
11	PCA2	0.430769	0.389793	0.430769	0.384625	0.630769	0.659302	0.630769	0.643370
12	Transformed0	0.523077	0.570105	0.523077	0.473173	0.492308	0.263999	0.492308	0.336004
13	Transformed1	0.523077	0.570105	0.523077	0.473173	0.784615	0.759318	0.784615	0.757769
14	Transformed2	0.446154	0.440461	0.446154	0.427264	0.661538	0.708563	0.661538	0.659405


Accuracy_NB
Accuracy_DT
0.8
0.7
0.6
0.5
0.4
0.3
0.2
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2
Classifier


Precision_NB
Precision_DT
0.8
0.7
0.6
0.5
0.4
0.3
0.2
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2
Classifier

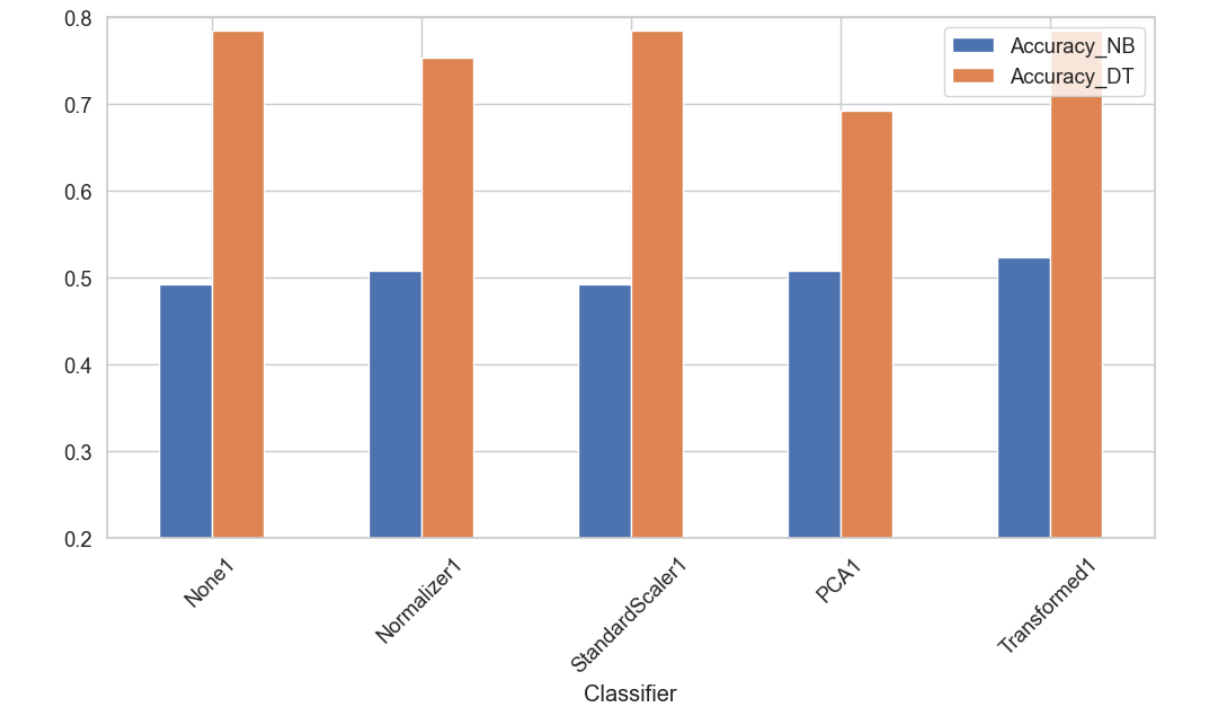

Recall_NB
Recall_DT
0.8
0.7
0.6
0.5
0.4
0.3
0.2
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2
Classifier

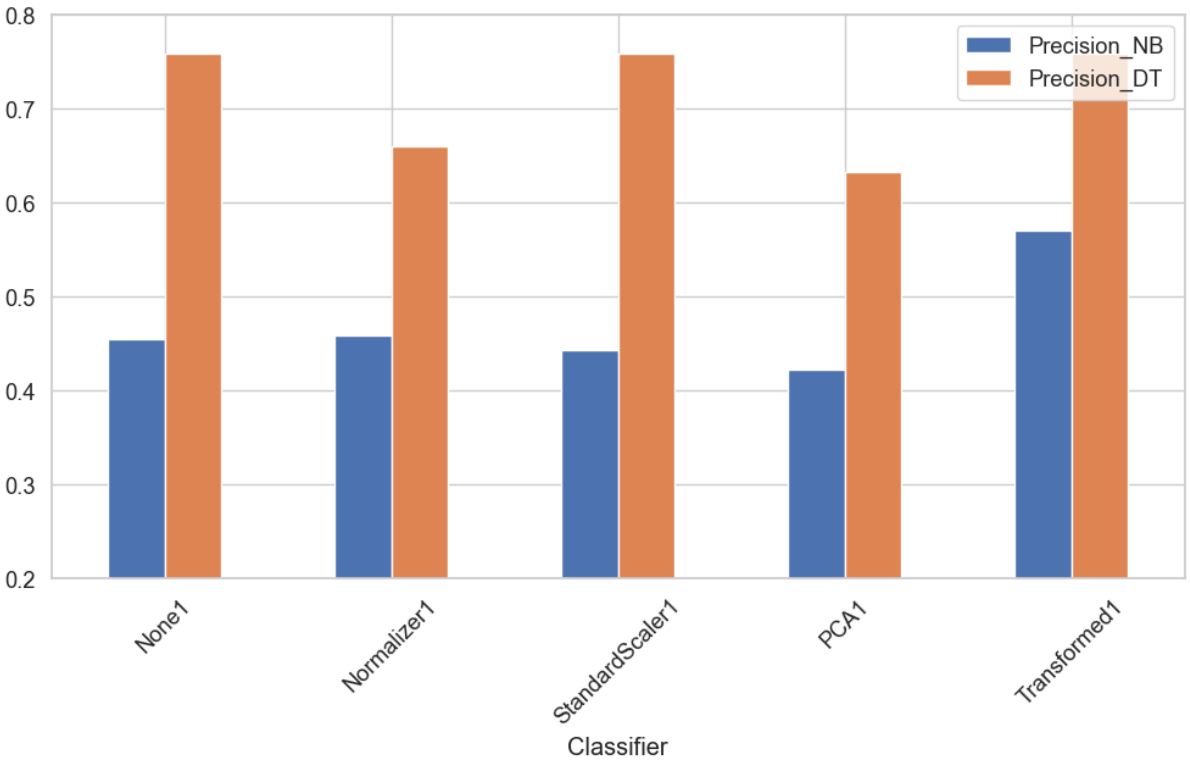

F1_NB
F1_DT
0.8
0.7
0.6
0.5
0.4
0.3
0.2
None0
None1
None2
Normalizer0
Normalizer1
Normalizer2
StandardScaler0
StandardScaler1
StandardScaler2
PCA0
PCA1
PCA2
Transformed0
Transformed1
Transformed2
Classifier

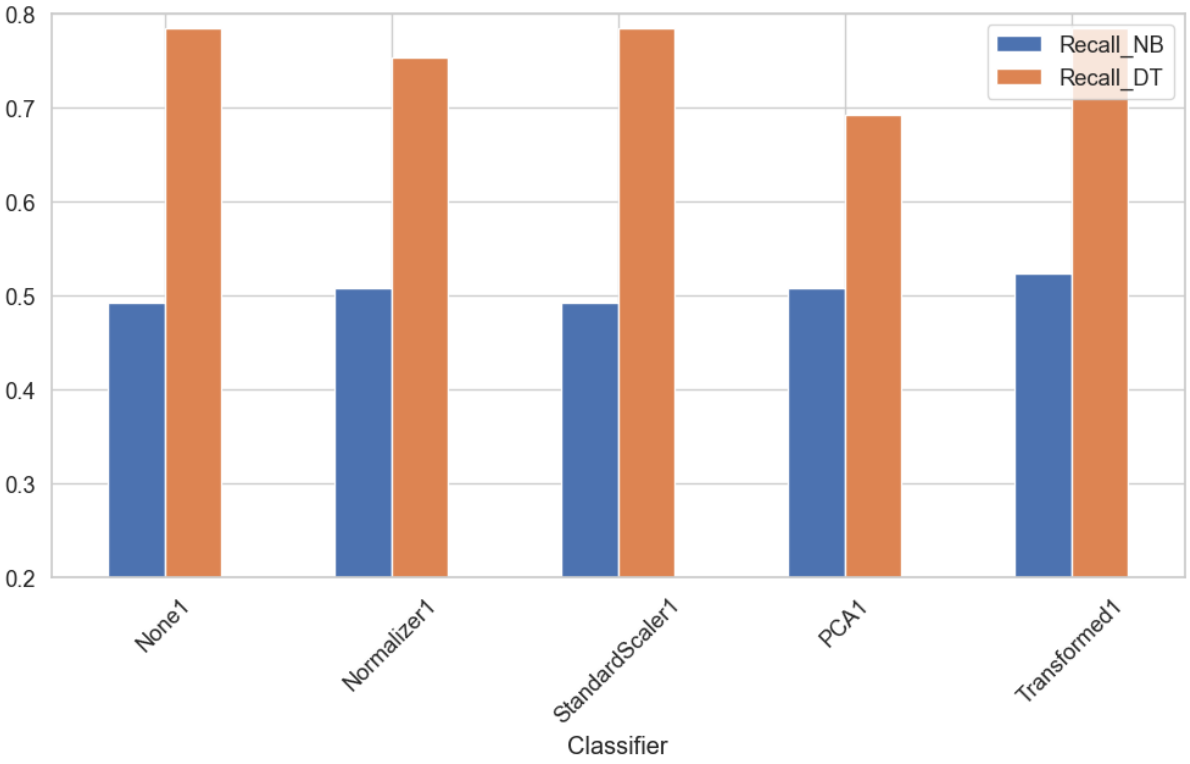
Wybierzmy najlepsze zestawy parametrów:

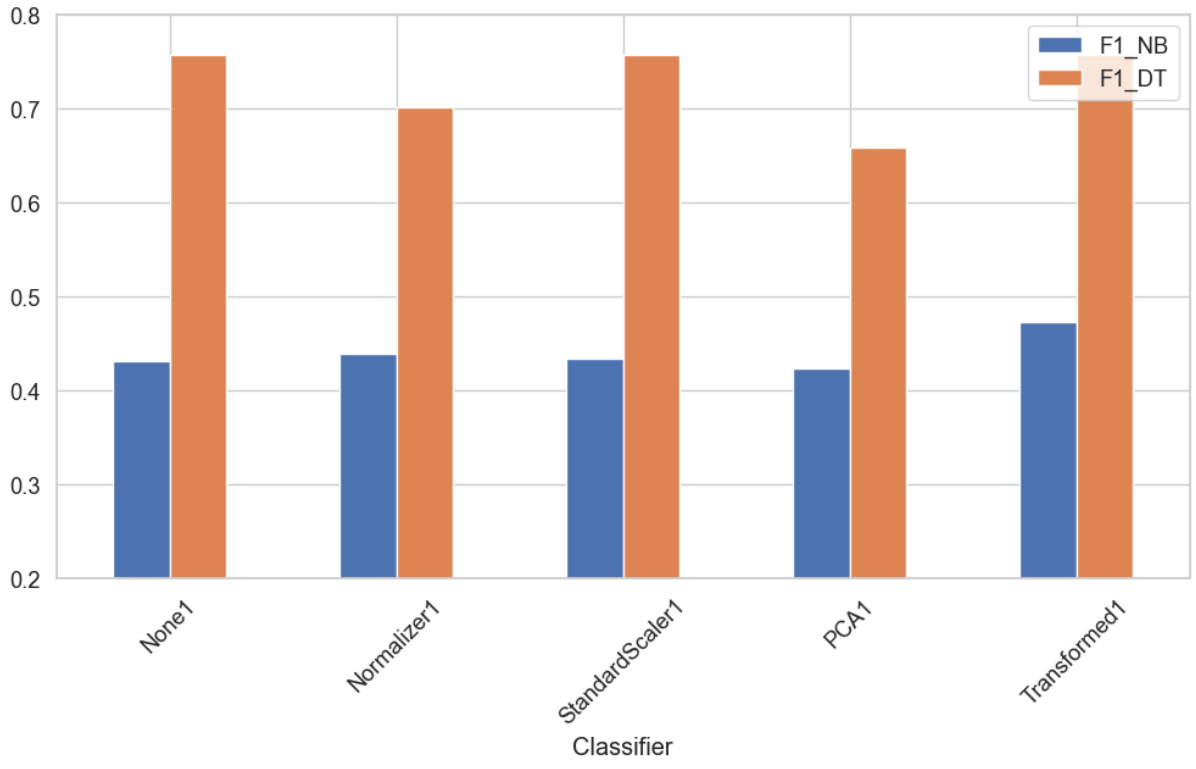
	Classifier	Accuracy_NB	Precision_NB	Recall_NB	F1_NB	Accuracy_DT	Precision_DT	Recall_DT	F1_DT
0	None1	0.492308	0.455000	0.492308	0.431603	0.784615	0.759318	0.784615	0.757769
1	Normalizer1	0.507692	0.459114	0.507692	0.438654	0.753846	0.660295	0.753846	0.701538
2	StandardScaler1	0.492308	0.443287	0.492308	0.433926	0.784615	0.759318	0.784615	0.757769
3	PCA1	0.507692	0.422786	0.507692	0.423883	0.692308	0.633210	0.692308	0.658786
4	Transformed1	0.523077	0.570105	0.523077	0.473173	0.784615	0.759318	0.784615	0.757769

```
df.plot.bar(x='Classifier', y=['Accuracy_NB', 'Accuracy_DT'], ylim=(0.2, 0.8), figsize=(12, 6), rot=45)
df.plot.bar(x='Classifier', y=['Precision_NB', 'Precision_DT'], ylim=(0.2, 0.8), figsize=(12, 6), rot=45)
df.plot.bar(x='Classifier', y=['Recall_NB', 'Recall_DT'], ylim=(0.2, 0.8), figsize=(12, 6), rot=45)
df.plot.bar(x='Classifier', y=['F1_NB', 'F1_DT'], ylim=(0.2, 0.8), figsize=(12, 6), rot=45)
```




0.8
0.7
0.6
0.5
0.4
0.3
0.2
Precision_NB
Precision_DT
None1
Normalizer1
StandardScaler1
PCA1
Transformed1
Classifier


0.8
0.7
0.6
0.5
0.4
0.3
0.2
Recall_NB
Recall_DT
None1
Normalizer1
StandardScaler1
PCA1
Transformed1
Classifier


0.8
0.7
0.6
0.5
0.4
0.3
0.2
F1_NB
F1_DT
None1
Normalizer1
StandardScaler1
PCA1
Transformed1
Classifier

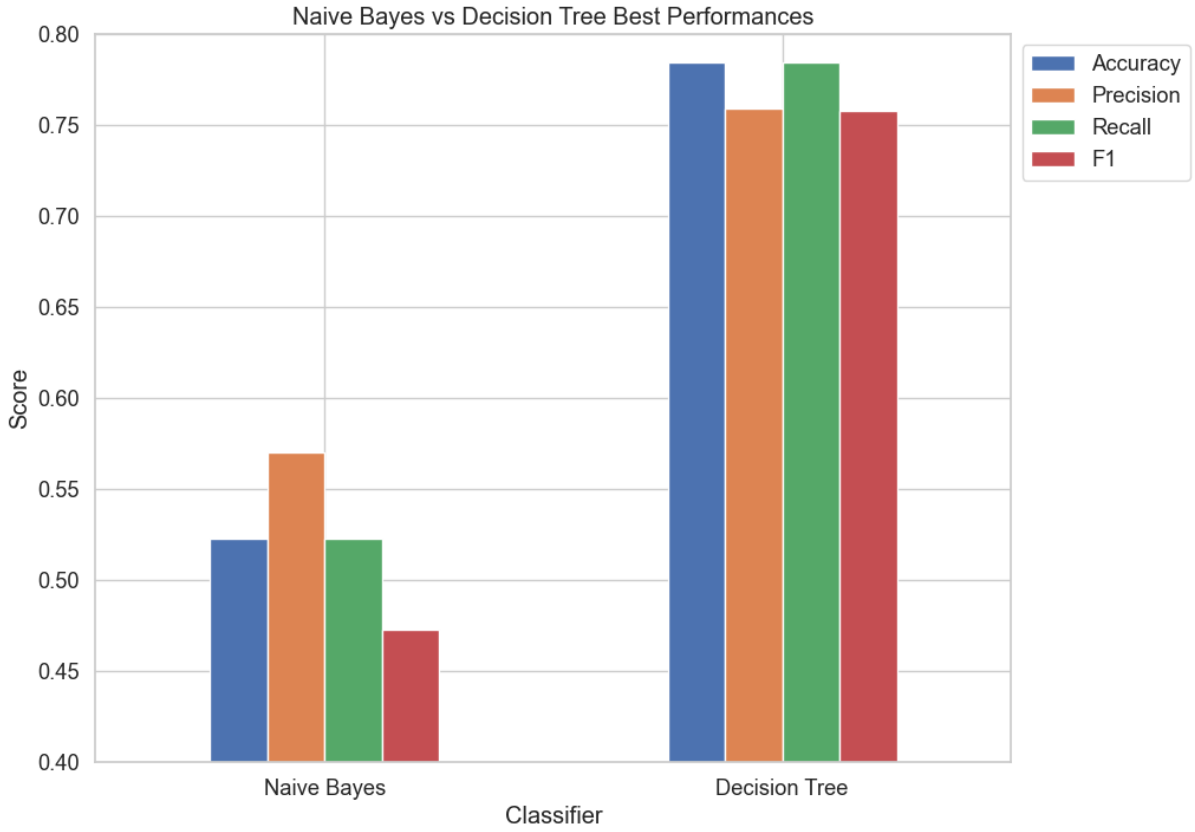
Najlepsze przetworzenie:

```
accuracy_nb = df['Accuracy_NB'][4]
accuracy_dt = df['Accuracy_DT'][4]

precision_nb = df['Precision_NB'][4]
precision_dt = df['Precision_DT'][4]

recall_nb = df['Recall_NB'][4]
recall_dt = df['Recall_DT'][4]

f1_nb = df['F1_NB'][4]
f1_dt = df['F1_DT'][4]
```

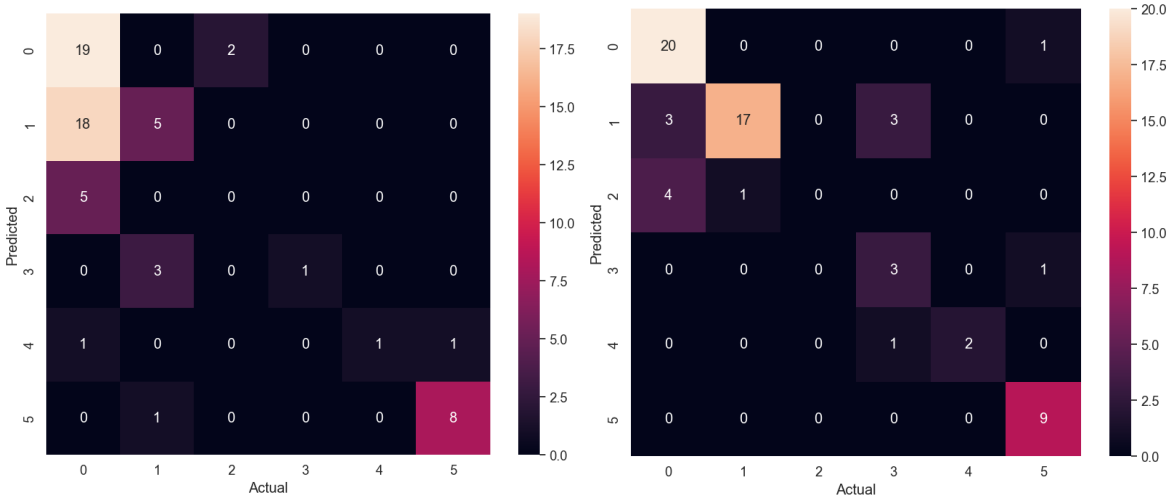
```
results = pd.DataFrame({'Accuracy': [accuracy_nb, accuracy_dt],
                        'Precision': [precision_nb, precision_dt],
                        'Recall': [recall_nb, recall_dt],
                        'F1': [f1_nb, f1_dt]},
                       index=['Naive Bayes', 'Decision Tree'])
results
```

	Accuracy	Precision	Recall	F1
Naive Bayes	0.523077	0.570105	0.523077	0.473173
Decision Tree	0.784615	0.759318	0.784615	0.757769

```
results.plot(kind='bar', figsize=(10, 8))
plt.title('Naive Bayes vs Decision Tree Best Performances')
plt.xlabel('Classifier')
plt.ylabel('Score')
plt.ylim(0.4, 0.8)
plt.xticks(rotation=0)
plt.legend(bbox_to_anchor=(1.0, 1.0))
```



Confussion Martrix dla NB Trans 1 oraz DT Trans 1



Wnioski

W ramach listy zostały wykonane wszystkie punkty. Lista ta stanowiła dobre wprowadzenie do podstaw uczenia maszynowego. Została pokazana różnica wyników dwóch klasyfikatorów naiwnego klasyfikatora Bayesa oraz drzewa decyzyjnego.

Pokazano wpływ przetworzenia danych na wyniki domyślnych klasyfikatorów. Poprawnie dobrana metoda przetworzenia ma duży wpływ na późniejsze wyniki oraz powinna być wybierana na podstawie wartości w zbiorze danych.

Każdy klasyfikator może zwracać różne wyniki swojego działania na podstawie różnie przyjętych parametrów. Okazuje się, że nie zawsze najbardziej „dokładne” ustawienia dają najlepszy wynik. Często domyślne wartości pogarszały ogólne wyniki.

Ogólnie można zauważyć, że dla akurat tego zbioru danych lepiej sobie radzi klasyfikator drzewa decyzyjnego. Może być to spowodowane tym, że zbiór danych nie był ogromny oraz faktem, że zakłada niezależność cech. Widać było, że niektóre cechy można zaznaczyć jako bardziej ważne a z racji tego, że drzewo decyzyjne zawiera w sobie odpowiednie warunki logiczne idealnie się wpasuje w „ważenie” decyzji.