Korelacja

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
import seaborn as sns

# Wczytanie zbioru danych  
df = pd.read\_csv("StudentPerformanceFactors.csv", sep = ",");

# Wyświetlenie pierwszych pięciu wierszy ramki danych  
df.head()

# Zamiana wartości kategorycznych na wartości liczbowe  
# (dla pojedyńczej kolumny)  
df["Parental\_Involvement"] = df["Parental\_Involvement"].replace(  
 {"Low":1, "Medium":2, "High": 3})

df["Internet\_Access"] = df["Internet\_Access"].replace({"Yes":1, "No":0})

# Kodowanie wartości kategorycznych na liczby całkowite  
# dla wszystkich kolumn  
for col in df.select\_dtypes(include=['object']).columns:  
 df[col], \_ = pd.factorize(df[col])

#---------------------------------------------------

# Wyjaśnienie działania funkcji pd.factorize  
# Przykładowa kolumna z wartościami kategorycznymi  
data = pd.Series(['apple', 'banana', 'apple', 'orange', 'banana', 'apple'])

# Użycie pd.factorize do kodowania  
codes, uniques = pd.factorize(data)  
print("Zakodowane wartości:", codes)  
print("Unikalne wartości:", uniques)

#---------------------------------------------------

# Obliczenie korelacji  
correlation\_matrix = df.corr()

# Wybór tylko kolumn zawierających wartości liczbowe  
correlation\_matrix = df.select\_dtypes(include=[np.number]).corr()

# Ustawienie rozmiaru wykresu  
plt.figure(figsize=(16, 16))

# Wizualizacji macierzy w formie graficznej   
sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', square=True)

# Wyświetlenie wykresu  
plt.show()

# Kodowane One-Hot Encoding ( 1 z n)  
# Kodowanie one-hot to technika używana do reprezentacji danych  
# kategorycznych w formie numerycznej. Dla każdej unikalnej  
# wartości kategorii tworzona jest nowa kolumna (tzw. cecha binarna),  
# w której wartość 1 oznacza wystąpienie danej kategorii, a 0 oznacza  
# jej brak. Technika ta jest bardzo przydatna w przygotowywaniu danych  
# do modeli uczenia maszynowego, które wymagają danych  
# numerycznycznych  
# Jak działa one-hot encoding?  
# Dla każdej unikalnej wartości kategorii w danej kolumnie powstaje  
# osobna kolumna. Jeśli w oryginalnej kolumnie znajduje się 5  
# różnych wartości (np. 'czerwony', 'zielony', 'niebieski'),  
# w wyniku kodowania one-hot powstają 3 nowe kolumny, np.:  
 #'kolor\_czerwony'  
 #'kolor\_zielony'  
 #'kolor\_niebieski'  
# Każda z tych kolumn zawiera wartość 1, jeśli dana obserwacja  
# należy do tej kategorii, i 0 w przeciwnym wypadku.

# Przykładowa ramka danych z kolumną kategoryczną  
df2 = pd.DataFrame({  
 'kolor': ['czerwony', 'zielony', 'niebieski', 'czerwony']  
})  
print(df2)

# Zastosowanie kodowania one-hot  
df2\_encoded = pd.get\_dummies(df2, columns=['kolor'])  
print(df2\_encoded)

Regresja Liniowa

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd  
# z pakietu scikit-learn  
from sklearn.linear\_model import LinearRegression

# Ustawienie ziarna dla generatora liczb losowych w celu powtarzalności   
# danych  
np.random.seed(2001)

# Wygenerowanie przykładowych danych  
x = np.arange(-3,3,0.1)

# Wartości funkcji tanh(x) są lekko zakłócane/zaburzane  
y = np.tanh(x) + np.random.randn(x.shape[0])\*0.2

# Funkcja np.random.randn z pakietu NumPy generuje losowe liczby z   
# rozkładu normalnego (o średniej 0 i odchyleniu standardowym 1)

# Generowanie 5 liczb z rozkładu normalnego o średniej 10 i odchyleniu   
# standardowym 2  
losowe\_liczby = np.random.normal(loc=10, scale=2, size=5)

# Wykres funkcji tanh(x) oraz wygenerowanych danych  
plt.plot(x, np.tanh(x), color = "green")  
plt.scatter(x,y, marker="s")  
plt.legend(['tanh(x)', 'wygenerowane dane'])  
plt.show()

# Model wymaga, aby dane były w postaci wektora kolumnowego,  
# domyślnie x i y to wektory wierszowe.

# (-1) - automatycznie dostosowując wymiar tablicy na podstawie pozostałych  
# podanych wymiarów. Oznacza to, że NumPy sam obliczy ten wymiar tak, aby   
# liczba elementów pozostała niezmieniona.  
x = x.reshape(-1,1)   
y = y.reshape(-1,1)

# kształt musi być taki sam  
x.shape  
y.shape

# LinearRegression to nazwa klasy  
# w tym miejscu jest tworzony obiekt  
model = LinearRegression()  
# Dopasowanie/uczenie modelu  
model.fit(x,y)  
# Predykcja wartości  
ypred = model.predict(x)

# Powyższy przypadek dotyczył tylko jednej zmiennej niezależnej, czyli   
# model regresji liniowej ma postać y=ax+b (równanie prostej)  
# coef\_ zawiera informacje o przyjętych współczynnikach/wagach przez model, # ale bez wyrazu wolnego   
model.coef\_ # (wartość a)  
# wyraz wolny  
model.intercept\_ # (wartość b)  
# Aby pobrać samą liczbę z model.coef\_ należy napisać model.coef\_[0][0]

plt.scatter(x,y)  
plt.xlabel('x')  
plt.ylabel('y')  
plt.plot(x, x\*model.coef\_[0][0] + model.intercept\_[0], color = "green")  
plt.legend(['Dane', 'Model'])  
plt.show()

plt.scatter(x,y)  
plt.xlabel('x')  
plt.ylabel('y')  
plt.plot(x, ypred)  
plt.legend([ 'F(x) - aproksymująca', 'f(x) - aproksymowana zaszumiona'])  
plt.show()

from sklearn.model\_selection import train\_test\_split  
# potrzeby jest moduł openpyxl  
bh\_data = pd.read\_excel('housing.xlsx')  
bh\_data  
# bh\_data.values zwraca wartości przechowywane przez ramkę danych jako   
# tablicę dwu-wymiarową z pakietu numpy  
bh\_arr = bh\_data.values

# X - wszystkie kolumny oprócz ostatniej  
# y - cała ostatnia kolumna  
X, y = bh\_arr[:,:-1], bh\_arr[:,-1]  
# bh\_arr[:,:-1] - wszystkie wiersze, kolumny od pierwszej do przedostatniej  
# bh\_arr[:,-1] - wszystkie wiersze, ostatnia kolumna

# znak - w kontekście indeksu oznacza numerację od końca   
# Podział zbioru danych na część treningową oraz część testową  
# Znak \ pozwala umieścić instrukcję w więcej niż jednej linijce  
X\_train, X\_test, y\_train, y\_test = \  
train\_test\_split(X,y, test\_size=0.2, random\_state= 221, shuffle=False)  
linReg = LinearRegression()  
linReg.fit(X\_train, y\_train)  
y\_pred = linReg.predict(X\_test)  
minval = min(y\_test.min(), y\_pred.min())  
maxval = max(y\_test.max(), y\_pred.max())  
plt.scatter(y\_test, y\_pred)  
# [minval, maxval] = zbiór x-ów, [minval, maxval] = zbiór y-ów  
plt.plot([minval, maxval], [minval, maxval], color="black")  
plt.xlabel('Cena medianowa - rzeczywista (y\_test)')  
plt.ylabel('Cena medianowa - przewidziana przez model (y\_pred)')  
plt.show()  
# Punkty, znajdujące się na prostej oznaczają poprawną predykcję

from sklearn.metrics import mean\_absolute\_error, \  
mean\_squared\_error, mean\_absolute\_percentage\_error

# MSE (Mean Squared Error) to średnia kwadratów błędów, czyli różnic między  
# wartościami rzeczywistymi ytest​ a przewidywaniami ypred​.   
# MSE mocniej karze duże błędy, ponieważ są one podnoszone do kwadratu  
# MSE = 1/n\*(sum\_{i}^{n} (ypred\_i-ytest\_i)^2)  
# gdzie n - liczba obserwacji  
MSE = mean\_squared\_error(y\_test, y\_pred, squared=True)  
# MAE (Mean Absolute Error) to średnia wartość bezwzględna różnic  
# między wartościami rzeczywistymi a przewidywanymi.  
# W przeciwieństwie do MSE, MAE nie podnosi błędów do kwadratu,  
# dzięki czemu każdy błąd wpływa na wynik w sposób liniowy.  
# MAE = 1/n\*(sum\_{i}^{n} |ypred\_i-ytest\_i|  
# gdzie n - liczba obserwacji, | | - wartość bezwzględna  
MAE = mean\_absolute\_error(y\_test, y\_pred)  
# MAPE (Mean Absolute Percentage Error) jest miarą względną,  
# która wyraża średni błąd bezwzględny jako procent wartości  
# rzeczywistej. Jest szczególnie przydatna, gdy interesuje  
# nas wielkość błędu w stosunku do rzeczywistej wartości  
# (np. Gdy chcemy kupić 300g sera w sklepie, a otrzymamy  
# 310g błąd jest względnie mały. Natomiast jeżeli pewna osoba przyjmuje  
# codziennie 10mg leku i omyłkowo przyjmie podwójną dawkę,  
# czyli 20mg, błąd względem referencyjnej wartości jest olbrzymi).  
# Wadą MAPE jest to, że nie działa dobrze, gdy wartości rzeczywiste  
# są bliskie zera, ponieważ wtedy miara może być niestabilna.  
# MAPE = 100%/n\*(sum\_{i}^{n} |(ypred\_i-ytest\_i)/ytest\_i|  
# gdzie n - liczba obserwacji, | | - wartość bezwzględna  
MAPE = mean\_absolute\_percentage\_error (y\_test, y\_pred)

# Mediana (Q2): Linia wewnątrz pudełka, która dzieli dane na pół   
# (50% danych jest mniejszych).

# Interkwartylowy rozstęp (IQR): Różnica między Q3 a Q1 (IQR=Q3−Q1).   
# IQR pokazuje zakres wartości, w którym znajduje się środkowe 50% danych.

# Wartości odstające (outliers): To punkty danych, które znajdują się   
# powyżej górnego wąsa lub poniżej dolnego wąsa.

# Wysokość pudełka wskazuje na rozrzut danych.  
plt.boxplot(y\_train)  
plt.title("Medianowa wartosc mieszkania")  
plt.show()

# y\_train - y\_train.mean(): Oblicza różnicę pomiędzy każdą wartością w   
# y\_train a średnią (mean) tych wartości. To daje nam odchylenie każdej   
# wartości od średniej.

# y\_train.std(): Dzieli odchylenia przez odchylenie standardowe (std)   
# y\_train, co normalizuje różnice, tworząc tzw. z-score dla każdej   
# wartości. Z-score wskazuje, o ile odchyleń standardowych dana wartość   
# znajduje się od średniej.

# Z - score: z =(x-m)/(std), gdzie:  
# z to z-score,  
# x to wartość, dla której obliczamy z-score,  
# m to średnia (mean) wszystkich wartości w zbiorze danych,  
# std to odchylenie standardowe (standard deviation) tych wartości.

# Funkcja np.abs oblicza wartość bezwzględną z z-score. To sprawia, że   
# niezależnie od tego, czy wartość jest powyżej, czy poniżej średniej,   
# skupiamy się na jej odległości od niej. W praktyce, wartości odstające to  
# te, które znajdują się więcej niż 3 odchylenia standardowe od średniej.  
outliers = np.abs((y\_train - y\_train.mean())/ y\_train.std())>3

# Znak ~ to operator bitowej negacji w Pythonie. Dla macierzy wartości   
# logicznych zamienia on True na False i False na True. Oznacza to, że   
# odwraca wszystkie wartości logiczne.  
X\_train\_no\_outliers = X\_train[~outliers,:]  
y\_train\_no\_outliers = y\_train[~outliers]  
# Kopiowanie danych  
y\_train\_mean = y\_train.copy()  
# Zastąpienie wartości odstających średnią   
y\_train\_mean[outliers] = y\_train.mean()  
linReg = LinearRegression()  
linReg.fit(X\_train, y\_train\_mean)

# Czy predykcja się poprawiła?  
ypred = linReg.predict(X\_test)  
MSE2 = mean\_squared\_error(y\_test, y\_pred, squared=True)  
MAE2 = mean\_absolute\_error(y\_test, y\_pred)  
MAPE2 = mean\_absolute\_percentage\_error (y\_test, y\_pred)  
wagi = linReg.coef\_

# bh\_data.columns - zwraca nazwy kolumny  
niezleżne\_cechy = bh\_data.columns[:-1]

# funkcja len zwraca długość listy  
# wygenerowanie liczb 0,1,2,3... aż do len(niezleżne\_cechy)  
x = np.arange(len(niezleżne\_cechy))  
fig, ax = plt.subplots(1,1)  
ax.bar(x, wagi)  
ax.set\_xticks(x)  
ax.set\_xticklabels(niezleżne\_cechy, rotation = 90)  
fig.show()  
plt.show()

# Funkcja np.stack łączy ciąg tablic wzdłuż określonego kierunku axis=-1   
# oznacza, że nowe cechy będą dodawane jako dodatkowe kolumny (w ostatniej  
# osi) w wyniku nowe\_dane. Wynikowy obiekt będzie macierzą, w której każda  
# kolumna odpowiada jednej z nowo utworzonych cech.  
nowe\_dane = np.stack([X[:, 4]/X[:, 7],   
 X[:, 4]/X[:, 5],   
 X[:, 4]\*X[:, 3],  
 X[:, 4]/X[:, -1]],   
 axis=-1)  
# Funkcja np. concatenate służy do łączenia dwóch lub więcej tablic wzdług # określonej osi  
X\_additional = np.concatenate([X, nowe\_dane], axis=-1)

Algorytm najbliższego sąsiada i wektory nośne

import numpy as np  
import matplotlib.pyplot as plt  
import pandas as pd

####################  
### Informacje dotyczące pewnej zmiany w pakiecie pandas  
### Można pominąć  
####################  
pd.options.mode.copy\_on\_write = False  
# Uwaga  
# Aż do wersji pandas 3.0, odwoływanie się do poszczególnych  
# elementów ramki danych poprzez nawiasy [][] zwracało  
# referencję do oryginalnych wartości   
df = pd.DataFrame({"foo": [10, 20, 30], "bar": [4, 5, 6]})  
print(df)  
df['foo'][2] = 3 # oryginalny obiekt df jest modyfikowany  
print(df)  
df['foo'] = [-8,-8,-3] # oryginalny obiekt df jest modyfikowany  
print(df)  
# Od wersji 3.0 notacja wykorzystująca [][] zwraca kopię danych  
# Wprowadzona zmiana nazywa się ,,Copy on write'' (CoW) i  
# w starszych wersjach pandas można ją włączyć za pomocą   
# instrukcji:  
pd.options.mode.copy\_on\_write = True  
df = pd.DataFrame({"foo": [10, 20, 30], "bar": [4, 5, 6]})  
print(df)  
df['foo'][2] = 3 # oryginalny obiekt df NIE jest modyfikowany  
print(df)  
df['foo'] = [-8,-8,-3] # oryginalny obiekt df jest modyfikowany  
print(df)  
# Od wersji pandas 3.0 modyfikacja konkretnego elementu może   
# odbyć się przy pomocy funkcji df.iloc  
df.iloc[0,0] = 3  
print(df)

####################  
### Laboratorium – Kod  
####################

# Wczytanie danych - potrzeby jest moduł openpyxl  
data\_const = pd.read\_excel('loan\_data.xlsx')  
data = pd.read\_excel('loan\_data.xlsx')  
# Wyświetlenie 5 początkowych wierszy z ramki danych  
data.head()  
# Wyświetlenie podstawowych informacji o ramce danych  
data.info()

# Uwaga! Loan\_Status to zmienna zależna  
# Pobranie nazw kolumn  
columns = list(data.columns)  
# Mask to wektor zawierający True/False w zależności czy obserwacja jest   
# kobietą, czy mężczyzną.  
mask = (data['Gender'].values == 'Female')  
# data['Gender'][mask] zwraca wiersze (referencje, od pandas 3.0 zwraca   
# kopie) kolumny  
  
# Gender, które mają wartość Female  
data['Gender'][mask] = 1  
# zamiast ~ można użyć funkcji np.logical\_not(mask)  
data['Gender'][~mask] = 0

# Aby zmodyfikować dane bezpośrednio w oryginalnej ramce, należy użyć   
# metody .loc[], która zapewnia bezpośredni dostęp do komórek, unikając   
# problemu kopiowania danych.  
data.loc[mask, 'Gender'] = 1  
data.loc[~mask, 'Gender'] = 0

# Kodowane One-Hot Encoding ( 1 z n )  
# Kodowanie one-hot to technika używana do reprezentacji danych   
# kategorycznych w formie numerycznej. Dla każdej unikalnej wartości  
# kategorii tworzona jest nowa kolumna (tzw. cecha binarna), w której  
# wartość 1 oznacza wystąpienie danej kategorii, a 0 oznacza jej brak.  
# Technika ta jest bardzo przydatna w przygotowywaniu danych do modeli  
# uczenia maszynowego, które wymagają danych numerycznycznych

# Jak działa one-hot encoding?  
# Dla każdej unikalnej wartości kategorii w danej kolumnie powstaje osobna  
# kolumna. Jeśli w oryginalnej kolumnie znajduje się 5 różnych wartości  
# (np. 'czerwony', 'zielony', 'niebieski'), w wyniku kodowania one-hot  
# powstają 3 nowe kolumny, np.:  
 #'kolor\_czerwony'  
 #'kolor\_zielony'  
 #'kolor\_niebieski'  
# Każda z tych kolumn zawiera wartość 1, jeśli dana obserwacja należy do  
# tej kategorii, i 0 w przeciwnym wypadku.

# Wyświetlenie występujących wartości zmiennej kategorycznej  
data['Property\_Area'].unique()  
# Zliczenie wystąpień unikatowych wartości  
data['Property\_Area'].value\_counts()

# Do kolumn można odwoływać się stosując notację data.Nazwa\_Kolumny   
# cat\_feature = pd.Categorical(data.Property\_Area)  
# one\_hot = pd.get\_dummies(cat\_feature)  
one\_hot = pd.get\_dummies(data.Property\_Area)

# Jak to wygląda?  
print(one\_hot)

# Wartości są True/False, a chcemy aby były 1/0  
one\_hot = one\_hot.astype(int)  
print(one\_hot)

# Dołączenie nowych kolumn do oryginalnej ramki danych  
data = pd.concat([data, one\_hot], axis = 1)

# Usunięcie oryginalnej kolumny Property\_Area  
data = data.drop(columns = ['Property\_Area'])

# Czy wszystko jest ok?  
data.head()

# Przekształcenie pozostałych kolumn o typie kategorycznym factorize zwraca  
# dwie wartości dlatego jest , \_  
data['Married'], \_ = pd.factorize(data["Married"])  
data['Self\_Employed'], \_ = pd.factorize(data["Self\_Employed"])  
data['Education'].unique()  
# Education zawiera tylko dwie wartości  
data['Education'] = data['Education'].replace(  
 {"Graduate":1, "Not Graduate":0})

# Czy wszystko jest ok?  
data.head()  
data\_const.head()

#---------------------------------------------------

# Wyjaśnienie działania funkcji pd.factorize  
# Przykładowa kolumna z wartościami kategorycznymi  
data2 = pd.Series(['apple', 'banana', 'apple', 'orange', 'banana', 'apple'])  
# Użycie pd.factorize do kodowania  
codes, uniques = pd.factorize(data2)  
print("Zakodowane wartości:", codes)  
print("Unikalne wartości:", uniques)

#---------------------------------------------------

# Obecnie data zawiera dodatkowe kolumny, więc należy ponownie  
# pobrać informacje o nazwach cech  
features = data.columns

# Uwaga  
# Loan\_Status to zmienna zależna  
# Należy przenieść kolumnę Loan\_Status na koniec tabeli!  
ko = data['Loan\_Status']  
data = data.drop(columns=['Loan\_Status'])

# Dodanie kolumny na końcu  
data['Loan\_Status'] = ko

# Czy wszystko jest ok?  
data.head()

data['Loan\_Status'].unique()  
# factorize działa poprawnie dla Yes/No  
data['Loan\_Status'] = data['Loan\_Status'].replace(  
 {"Y":1, "N":0})

# Czy wszystko jest ok?  
data.head()   
vals = data.values.astype(np.float)

# Konwertowanie danych na typ float64  
vals = data.values.astype(np.float64)  
X = vals[:, :-1]  
y = vals[:,-1]

# Podział danych na zbiór treningowy/uczący oraz zbiór testowy  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = \  
train\_test\_split(X,y, test\_size=0.2, random\_state= 221, shuffle=False)

# Działanie train\_test\_split:  
# Funkcja dzieli dane wejściowe (X i y) na cztery części:  
# X\_train: dane wejściowe do uczenia modelu.   
# X\_test: dane wejściowe do testowania modelu.  
# y\_train: ewyniki dla zbioru treningowego.  
# y\_test: wyniki dla zbioru testowego.

# Parametry funkcji train\_test\_split:  
# X, y: Dane wejściowe i odpowiadające im etykiety.  
# test\_size=0.2: Ustalony rozmiar zbioru testowego.   
# W tym przypadku 20% danych zostanie przeznaczone  
# na testy, a pozostałe 80% na trenowanie.  
# random\_state=221: Ustawienie tego parametru pozwala   
# na uzyskanie powtarzalnych wyników — podział danych  
# na zbiory treningowy i testowy będzie taki sam  
# za każdym uruchomieniem kodu.  
# shuffle=False: Bez przetasowania danych przed  
# podziałem, co oznacza, że dane są dzielone  
# w kolejności, w jakiej występują w zestawie.  
# Może to być istotne, gdy dane są uporządkowane  
# w sposób, który chcemy zachować.

# z pakietu scikit-learn  
from sklearn.neighbors import KNeighborsClassifier as kNN  
from sklearn.svm import SVC as SVM  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import precision\_score  
from sklearn.metrics import recall\_score  
from sklearn.metrics import f1\_score  
from sklearn.metrics import roc\_auc\_score  
import seaborn as sns  
import matplotlib.pyplot as plt

models = [kNN(), SVM()]  
model\_names = ["k-Nearest Neighbors", "Support Vector Machine"]  
# Ustalanie etykiet klas  
class\_labels = ['Loan\_Status: No', 'Loan\_Status: Yes']

# Metryki  
accuracies = []  
precisions = []  
recalls = []  
f1s = []  
roc\_aucs = []

# funkcja zip łączy elementy list w pary (x,y)   
# (X-element z listy X)  
# (Y-element z listy Y)  
for model, name in zip(models, model\_names):  
 model.fit(X\_train,y\_train)  
 y\_pred = model.predict(X\_test)  
 cm = confusion\_matrix(y\_test, y\_pred)  
 # print(cm)  
 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",   
 xticklabels=class\_labels, yticklabels=class\_labels)  
 plt.xlabel("Predykcje")  
 plt.ylabel("Wartości rzeczywiste")  
 plt.title(f"Macierz pomyłek dla {name}")  
 plt.show()  
 # append dodaje element do końca listy  
 accuracies.append(accuracy\_score(y\_test, y\_pred))  
 precisions.append(precision\_score(y\_test, y\_pred))  
 recalls.append(recall\_score(y\_test, y\_pred))  
 f1s.append(f1\_score(y\_test, y\_pred))  
 roc\_aucs.append(roc\_auc\_score(y\_test, y\_pred))

# Przedstawienie graficzne metryk  
metrics = ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']  
wartosci = np.array([accuracies, precisions, recalls, f1s, roc\_aucs])  
fig, ax = plt.subplots(figsize=(10, 6))

# Ustawienia słupków  
width = 0.35 # Szerokość słupków  
x = np.arange(len(metrics)) # Pozycje na osi x

# accuracies, precisions .... każda z tych list zawiera dwie wartości zatem  
# tablica „wartości” jest tablicą dwu-wymiarową  
# Dodawanie słupków dla obu modeli  
ax.bar(x - width/2, wartosci[:, 0], width, label='kNN', color='lightblue')  
ax.bar(x + width/2, wartosci[:, 1], width, label='SVM', color='salmon')  
ax.set\_title('Porównanie metryk dla kNN i SVM')  
ax.set\_xticks(x)  
ax.set\_xticklabels(metrics)  
ax.legend()

# Wyświetlenie wykresu  
plt.tight\_layout()  
plt.show()

# Tworzenie wykresów pudełkowych dla każdej kolumny  
# Rozmiar wykresu  
plt.figure(figsize=(16, 12))   
# Funkcja enumerate() generuje automatycznie indeksy. Wynikiem funkcji  
# jest indeks elementu oraz wartość. Przydatne gdy  
# potrzebujemy zarówno elementu, jak i jego indeksu.  
for i, column in enumerate(data.columns):  
 # plt.subplot(liczba\_wierszy, liczba\_kolumn, indeks\_aktualnego\_wykresu)   
 plt.subplot(3, 5, i + 1)   
 plt.boxplot(data[column])  
 plt.title(column)  
plt.show()

# Kolumny o wartościach 0 lub 1 wyłączone są ze skalowania  
# (ale można też standaryzować wszystkie dane)  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
# Metoda fit oblicza średnią (mean) i odchylenie standardowe (standard   
# deviation) dla każdej cechy w zbiorze danych treningowych X\_train  
scaler.fit(X\_train[:,[2,5,6,7,8]])  
# Metoda transform przekształca dane treningowe X\_train zgodnie  
# z wcześniej obliczonymi średnimi i odchyleniami standardowymi.  
# z = (x-mean)/(std)

X\_train[:,[2,5,6,7,8]] = scaler.transform(X\_train[:,[2,5,6,7,8]])  
X\_test[:,[2,5,6,7,8]] = scaler.transform(X\_test[:,[2,5,6,7,8]])  
accuracies\_2 = []  
precisions\_2 = []  
recalls\_2 = []  
f1s\_2 = []  
roc\_aucs\_2 = []  
  
for model, name in zip(models, model\_names):  
 model.fit(X\_train,y\_train)  
 y\_pred = model.predict(X\_test)  
 cm = confusion\_matrix(y\_test, y\_pred)  
 # print(cm)  
 sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",   
 xticklabels=class\_labels, yticklabels=class\_labels)  
 plt.xlabel("Predykcje")  
 plt.ylabel("Wartości rzeczywiste")  
 plt.title(f"Macierz pomyłek dla {name}")  
 plt.show()  
 # append dodaje element do końca listy  
 accuracies\_2.append(accuracy\_score(y\_test, y\_pred))  
 precisions\_2.append(precision\_score(y\_test, y\_pred))  
 recalls\_2.append(recall\_score(y\_test, y\_pred))  
 f1s\_2.append(f1\_score(y\_test, y\_pred))  
 roc\_aucs\_2.append(roc\_auc\_score(y\_test, y\_pred))

# Porównanie metryk  
# Szerokość słupków  
width = 0.9  
# Ustalamy pozycje słupków na osi X  
x = np.arange(2)  
metryki = np.array([(accuracies, accuracies\_2),  
 (precisions, precisions\_2),  
 (recalls, recalls\_2),  
 (f1s, f1s\_2),  
 (roc\_aucs, roc\_aucs\_2)])

# Tworzenie wykresów  
fig, axs = plt.subplots(ncols=3,nrows=2,figsize=(16, 9))  
# Spłaszczanie tablicy do 1D, aby można było liniowa iterować  
axs = axs.flatten()  
# Ukrycie ostatniego nieużywanego wykresu  
axs[5].axis('off')  
# Funkcja enumerate() generuje automatycznie indeksy. Wynikiem funkcji  
# jest indeks elementu oraz wartość. Przydatne gdy  
# potrzebujemy zarówno elementu, jak i jego indeksu.  
# start - parametr określający początkową wartość indeksu (domyślnie 0).  
for i, m in enumerate(metryki):  
 # Dodawanie słupków dla obu modeli  
 # [m[0][1], m[1][1]] - dla SVM  
 axs[i].bar(x - width/2, [m[0][0], m[1][0]], width, color=['#4CAF50', '#FF9800'])  
 # Etykietowanie  
 axs[i].set\_ylabel('Dokładność')  
 axs[i].set\_title(f'Porównanie {["Dokładności", "Precyzji", "Czułości", "F1 Score", "ROC AUC"][i]}')  
 # Ustawianie oznaczeń na osi X  
 axs[i].set\_xticks(x - width/2)  
 axs[i].set\_xticklabels(['Model podstawowy', 'Model zmodyfikowany'])

# Dodanie tytułu całego obrazka  
fig.suptitle('Porównanie metryk kNN', fontsize=16)  
# Wyświetlenie wykresu  
plt.tight\_layout(rect=[0, 0.03, 1, 0.95])   
# Dostosowanie, aby tytuł się nie nakładał  
plt.show()  
# Dotychczasowe dane X\_train, X\_test zostały przeskalowane  
X\_train, X\_test, y\_train, y\_test = \  
train\_test\_split(X,y, test\_size=0.2, random\_state= 221, shuffle=False)  
from sklearn.tree import DecisionTreeClassifier as DT  
from sklearn.tree import plot\_tree  
model = DT(max\_depth=3)  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_pred)  
#print(cm)  
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",   
 xticklabels=class\_labels, yticklabels=class\_labels)  
plt.xlabel("Predykcje")  
plt.ylabel("Wartości rzeczywiste")  
plt.title(f"Macierz pomyłek dla Decision Tree Classifier")  
plt.show()

plt.figure(figsize=(25,15))  
tree\_vis = plot\_tree(model,feature\_names= data.columns[:-1],  
 class\_names=['N', 'Y'], fontsize = 20)

Analiza Fouriera, Ekstrakcja i Generalizacja Cech

# potrzebny moduł scipy  
from scipy.io import wavfile  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
# os jest modułem podstawowym  
import os  
# z modułu librosa  
import librosa

#############  
### UWAGA ###  
#############  
# Poprawiłem zawartość pliku gender\_voices.csv  
# Teraz kolejność danych w gender\_voices.csv odpowiada  
# kolejności plików w folderze voices\_pl

# Informacje o klasyfikacji głosu (kobieta mężczyzna)  
metadata = pd.read\_csv('gender\_voices.csv', sep=',')  
# Folder z nagraniami  
path = 'voices\_pl'  
# Pobierz listę wszystkich plików w danym folderze  
files = os.listdir(path)  
fs = 16000  
# Sprawdzenie częstotliwości próbkowania 16kHz  
for file in files:  
 y, sr = librosa.load(f"{path}\\{file}", sr=None)   
 #print(f"Częstotliwość próbkowania: {sr} Hz")  
# Surowe dane z nagrań w postaci zwykłej listy  
X\_raw\_list = []  
# Zmienna zależna   
y = metadata['gender'].values  
# Maksymalna długość nagrania  
max\_length = 0  
# Domyślnie enumerate zaczyna od zera, ale można to zaznaczyć jawnie  
for idx, file in enumerate(files, start=0):  
 # f"{path}\\{file}" <-- formatowany string, czyli do string  
 # wrzucamy zawartość zmiennej path oraz file  
 sample\_rate, data = wavfile.read(f"{path}\\{file}")  
 X\_raw\_list.append(data)  
 max\_length = max(max\_length, len(data))  
print(f"Załadowano {len(X\_raw\_list)} nagrań.")

# Do obliczeń potrzeba danych zapisanych w obiekcie np.array  
# Tworzenie macierzy 2D o ustalonej maksymalnej długości wypełnionej zerami  
X\_raw = np.zeros((len(X\_raw\_list), max\_length))  
for i, nagranie in enumerate(X\_raw\_list):  
 length = len(nagranie)  
 # pozostałe miejsce wypełnione jest 0  
 X\_raw[i, :length] = nagranie   
# Kształt (liczba wierszy, liczba kolumn) obiektu X\_raw  
X\_raw.shape

from scipy.fft import fft  
# Długość nagrania w sekundach  
seconds = max\_length/ fs  
# axis=-1: Określa, że transformata Fouriera zostanie   
# obliczona wzdłuż ostatniej osi tablicy X\_raw.  
# czyli dane będzie pobierać wzdłuż osi kolumn  
X\_fft = np.abs(fft(X\_raw, axis=-1))/X\_raw.shape[1]  
# Sprawdź co się tanie bez dzielenia przez X\_raw.shape[1]  
low\_cut = int(50\*seconds)  
hight\_cut = int(280\*seconds)  
X\_fft\_cut = X\_fft[:, low\_cut:hight\_cut]  
fig, ax = plt.subplots(2,1)  
ax[0].plot(np.arange(X\_raw.shape[1]), X\_raw[0,:])  
ax[1].scatter(np.arange(X\_raw.shape[1]), X\_fft[0,:], s = 0.5)  
plt.title("Pierwsze nagranie (indeks 0)\n common\_voice\_pl\_20603074.wav",  
 fontsize=14)  
fig.tight\_layout()

# Wynikiem fft jest tablica 2D  
# Wiersze odpowiadają różnym próbką sygnałów (czyli różnym nagraniom).  
# Kolumny odpowiadają częstotliwościom, które są wynikiem transformaty   
# Fouriera dla każdego nagrania.

# Próbki w tablicy wynikowej, takie jak 7, -1, -7, reprezentują wartości   
# amplitudy sygnału dźwiękowego w poszczególnych momentach czasowych. Są to  
# próbki nagrania audio w postaci dyskretnych wartości, które są wynikiem  
# procesu próbkowania sygnału ciągłego.

X\_raw[0,67000:70000]

# Jeśli mamy sygnał dźwiękowy o częstotliwości próbkowania 16 kHz, to kolejne próbki w tablicy X\_raw reprezentują:  
# X\_raw[0] — wartość amplitudy po 0 sekund (moment rozpoczęcia   
# nagrania),  
# X\_raw[1] — wartość amplitudy po 1/16000 = 0.0000625 sekundy,  
# X\_raw[2] — wartość amplitudy po 2/16000​ sekundy,  
# i tak dalej.

# Ustawienie wyświetlania liczb w pełnej precyzji wyłącza notację naukową  
np.set\_printoptions(suppress=True)   
X\_raw[0,20\_000:21\_000]

# Operator // w Pythonie oznacza dzielenie całkowitoliczbowe (ang. floor   
# division). Oznacza to, że wynik dzielenia jest zaokrąglany w dół do   
# najbliższej liczby całkowitej  
a = 10  
b = 3  
a // b

# Rozdzielczość widma  
# fs / (seconds\* fs) = 1/(seconds) [Hz]  
mean\_num = int(np.round(seconds))

# Obcięcie dodatkowych kolumn, jeśli jest to konieczne  
num\_columns = (X\_fft.shape[1] // mean\_num) \* mean\_num  
X\_fft\_trimmed = X\_fft[:, :num\_columns]

# Teraz reshape będzie działać poprawnie  
# Wyjaśnienie poniższego kodu znajduje się niżej  
X\_fft2 = np.reshape(X\_fft\_trimmed, (X\_fft.shape[0],   
 X\_fft\_trimmed.shape[1] // mean\_num,   
 mean\_num))

X\_fft2 = X\_fft2.mean(axis=-1)  
low\_cut = 50 # 50Hz  
hight\_cut = 280 # 280 Hz  
# od 50 do 280 kolumny, czyli 230 czestotliwosci pobieramy  
X\_fft\_cut = X\_fft2[:, low\_cut:hight\_cut]

fig, ax = plt.subplots()  
ax.plot(np.arange(X\_fft\_cut.shape[1]), X\_fft\_cut[0,:])  
ax.set\_xlabel("Częstotliwość [Hz]")  
ax.set\_ylabel("Amplituda")  
ax.set\_title('Pierwsze nagranie (indeks 0) przed skalowaniem')  
fig.tight\_layout()

# Skalowanie  
X\_fft\_cut = X\_fft\_cut/ np.expand\_dims(X\_fft\_cut.max(axis=1), axis=-1)  
fig, ax = plt.subplots()  
ax.plot(np.arange(X\_fft\_cut.shape[1]), X\_fft\_cut[0,:])  
ax.set\_xlabel("Częstotliwość [Hz]")  
ax.set\_ylabel("Amplituda")  
ax.set\_title('Pierwsze nagranie (indeks 0) po skalowaniu')  
fig.tight\_layout()

# X\_fft\_cut.max(axis=1) zwraca maksymalne wartości dla każdej próbki w   
# X\_fft\_cut, tworząc wektor o kształcie (n,), gdzie n to liczba próbek.   
# Funkcja np.expand\_dims(..., axis=-1) zmienia kształt tego wektora z (n,)  
# na (n, 1), co umożliwia poprawne dzielenie (broadcasting) przez  
# oryginalną tablicę X\_fft\_cut. Po tej operacji, każda próbka w X\_fft\_cut  
# będzie znormalizowana do zakresu od 0 do 1.

# Wyjaśnienie  
# X\_fft2 = np.reshape(X\_fft\_trimmed, (X\_fft.shape[0],   
# X\_fft\_trimmed.shape[1] // mean\_num,   
# mean\_num))

# Załóżmy, że mamy tablicę 2D G o kształcie (3, 8):  
# Przykładowe dane  
G = np.array([[1, 2, 3, 4, 5, 6, 7, 8], # Próbka 1  
 [9, 10, 11, 12, 13, 14, 15, 16], # Próbka 2  
 [17, 18, 19, 20, 21, 22, 23, 24]]) # Próbka 3

# Powinno być (3, 8)  
G.shape  
# Chcemy uśredniać co 2 wartości, więc ustalamy mean\_num = 2  
mean\_num = 2  
G\_reshaped = np.reshape(G, (G.shape[0], G.shape[1] // mean\_num, mean\_num))

# Powinno być (3, 4, 2)   
# (3 próbki, w każdej próbce 4 grupy po 2 elementy)  
G\_reshaped.shape  
G  
G\_reshaped

# Oryginalna tablica  
# [[ 1, 2, 3, 4, 5, 6, 7, 8], # Próbka 1  
# [ 9, 10, 11, 12, 13, 14, 15, 16], # Próbka 2  
# [17, 18, 19, 20, 21, 22, 23, 24]] # Próbka

# Po reshaping:  
# [  
# [[ 1, 2], # Próbka 1, Grupa 1  
# [ 3, 4], # Próbka 1, Grupa 2  
# [ 5, 6], # Próbka 1, Grupa 3  
# [ 7, 8]], # Próbka 1, Grupa 4  
# [[ 9, 10], # Próbka 2, Grupa 1  
# [11, 12], # Próbka 2, Grupa 2  
# [13, 14], # Próbka 2, Grupa 3  
# [15, 16]], # Próbka 2, Grupa 4  
# [[17, 18], # Próbka 3, Grupa 1  
# [19, 20], # Próbka 3, Grupa 2  
# [21, 22], # Próbka 3, Grupa 3  
# [23, 24]] # Próbka 3, Grupa 4  
# ]

from matplotlib import rcParams  
from sklearn.decomposition import PCA  
example = np.random.randn(500,2)  
example[:,1] \*= 0.4  
# \*\* oznacza potęgę  
2\*\*3 # powinno zwrócić 8  
# 1/2\*\*0.5 = 1/sqrt(2)  
rot\_matrix = np.array([[1/2\*\*0.5, 1/2\*\*0.5], [1/2\*\*0.5, -1/2\*\*0.5]])

# operator @ oznacza mnożenie macierzowe  
example = example @ rot\_matrix  
example\_PCAed = PCA(2).fit\_transform(example)  
fig, axs = plt.subplots(1,2, figsize=(20, 10))

axs[0].scatter(example[:,0], example[:,1])  
axs[1].scatter(example\_PCAed[:,0], example\_PCAed[:,1])  
axs[0].set\_xlim([-3, 3])  
axs[0].set\_ylim([-3, 3])  
axs[1].set\_xlim([-3, 3])  
axs[1].set\_ylim([-3, 3])  
axs[0].set\_xlabel('x')  
axs[0].set\_ylabel('y')  
axs[1].set\_xlabel('PC 1')  
axs[1].set\_ylabel('PC 2')  
axs[0].set\_title('Dane pierwotne')  
axs[1].set\_title('Dane po PCA')

# Podział danych na zbiór treningowy/uczący oraz zbiór testowy  
from sklearn.model\_selection import train\_test\_split  
# Zmienna zależna  
y  
# Zamiana na liczby: 'male\_masculine' -> 1, 'female\_feminine' -> 0  
dv = [1 if x == 'male\_masculine' else 0 for x in y]

# W pythonie, aby utworzyć szybko tablicę o ustalonych wartościach można   
# posłużyć się składnią  
tablica = [x\*\*2 for x in range(5)] #[0,1,4,9,25]  
tablica

# [Wyrażenie pętla\_for]  
# wyrażeniem może być nawet instrukcja if

# Konwersja do tablicy numpy  
dependent\_variable = np.array(dv)  
dependent\_variable

# Podział na zbiór treningowy oraz testowy  
X\_train, X\_test, y\_train, y\_test = \  
train\_test\_split(X\_fft\_cut,dependent\_variable,  
 test\_size=0.2,   
 random\_state= 221,   
 shuffle=False)

pca\_transform = PCA()  
pca\_transform.fit(X\_train)  
variances = pca\_transform.explained\_variance\_ratio\_  
# cumulated\_variances[3] = variances[0]+variances[1]+variances[2]  
# i tak dalej  
cumulated\_variances = variances.cumsum()  
fig, ax = plt.subplots()  
ax.scatter(np.arange(variances.shape[0]), cumulated\_variances)  
# cumulated\_variances<0.95 zwraca tablicę o wartościach True/False,  
# w zależności czy wartość w danym wierszu jest <0.95.  
PC\_num = (cumulated\_variances<0.95).sum()  
# PC\_num -- liczba wierszy spełniająca warunek  
# (potrzebna liczba składowych głównych)

X\_pcaed = PCA(2).fit\_transform(X\_train)  
fig, ax = plt.subplots(1,1)  
# wynikiem polecenia (y\_train==0) jest tablica wartości True/False  
# w zależności czy wartość w danym wierszu jest = 0.  
females = (y\_train==0)  
ax.scatter(X\_pcaed[females,0], X\_pcaed[females,1], label = 'female')  
ax.scatter(X\_pcaed[~females,0], X\_pcaed[~females,1], label = 'males')  
ax.legend()  
# ~ to operator negacji wszystkich elementów w tablicy

# potrzebny moduł scikit-learn  
from sklearn.datasets import load\_digits  
from sklearn.decomposition import FastICA

# Importujemy zestaw danych load\_digits, który zawiera cyfry zapisane jako  
# obrazy 8x8 pikseli, spłaszczone do wektorów o 64 wymiarach.  
X, y = load\_digits(return\_X\_y=True)  
transformer = FastICA(n\_components=7,   
 random\_state=0)  
# n\_components=7 oznacza, że wybieramy 7 składowych niezależnych.   
# ICA znajdzie 7 kierunków, w których składowe są jak najbardziej   
# niezależne.  
X\_transformed = transformer.fit\_transform(X)  
# X\_transformed ma ten sam liczba wierszy co X, ale tylko 7 kolumn,   
# ponieważ wybraliśmy 7 składowych niezależnych.  
X\_transformed.shape

# Wyświetlenie kilku oryginalnych obrazów  
fig, axes = plt.subplots(1, 5, figsize=(10, 5))  
for i, ax in enumerate(axes):  
 ax.imshow(X[i].reshape(8, 8), cmap='gray')  
 ax.set\_title(f'Cyfra {y[i]}')  
 ax.axis('off')  
plt.suptitle("Oryginalne obrazy cyfr")  
plt.show()

# Wyświetlenie pierwszych 7 niezależnych komponentów  
fig, axes = plt.subplots(1, 7, figsize=(15, 5))  
for i, ax in enumerate(axes):  
 # Przekształcenie na obrazek 8x8  
 component\_image = transformer.components\_[i].reshape(8, 8)   
 ax.imshow(component\_image, cmap='gray')  
 ax.set\_title(f'Składowa {i+1}')  
 ax.axis('off')  
plt.suptitle("Pierwsze 7 składowych niezależnych ICA")  
plt.show()

# Wyświetlenie przekształconych danych w nowych wymiarach  
plt.figure(figsize=(10, 7))  
plt.scatter(X\_transformed[:, 3], X\_transformed[:, 2], c=y, cmap='tab10',  
 edgecolor='k')  
plt.colorbar(label='Cyfra')  
plt.xlabel('Składowa 4 (przypomina 5)')  
plt.ylabel('Składowa 3 (przypomina 2)')  
plt.title("Dane w przestrzeni dwóch składowych ICA")  
plt.show()

# Jeśli punkty odpowiadające tej samej cyfrze tworzą wyraźne skupiska   
# (klastry), oznacza to, że ICA dobrze oddzieliła charakterystyczne cechy   
# różnych cyfr.  
# Klasteryzacja wskazuje, że dane są dobrze rozdzielone w przestrzeni ICA,  
# co sugeruje, że składowe uchwyciły kluczowe, niezależne cechy. To dobry  
# sygnał, że te składowe mogą być użyteczne do rozpoznawania cyfr.  
# Na wykresie widać, że cyfry 2 i 5 tworzą dwa klastery (grupy)

# Osie składowych ICA (tutaj Składowa 3 i Składowa 4) nie mają  
# bezpośredniego fizycznego znaczenia. Są kombinacjami cech (pikseli),  
# które ICA uznała za statystycznie niezależne i najbardziej  
# reprezentatywne dla struktury danych. Można próbować zrozumieć, które  
# aspekty obrazu cyfr najprawdopodobniej wpływają na te składowe, ale ICA  
# nie dostarcza informacji o interpretacji kierunków, w przeciwieństwie do  
# PCA, które maksymalizuje zmienność w kierunku głównych osi.

from sklearn.preprocessing import StandardScaler  
from sklearn.neighbors import KNeighborsClassifier as kNN  
from sklearn.svm import SVC as SVM  
from sklearn.tree import DecisionTreeClassifier as DT

# \ - oznacza przełamanie kodu na dwie linijki  
X\_train, X\_test, y\_train, y\_test = \  
 train\_test\_split(X\_fft\_cut,   
 dependent\_variable,   
 test\_size=0.2,   
 random\_state=6,   
 stratify=dependent\_variable)

# Jeśli masz problem klasyfikacji z dwoma klasami, np.:  
# 0 (np. klasa "kobieta") i  
# 1 (np. klasa "mężczyzna"),  
# gdzie 0 stanowi 70% danych, a 1 – 30%,  
# to z stratify=dependent\_variable funkcja train\_test\_split zadba o to, aby  
# zarówno w zbiorze treningowym, jak i testowym, te proporcje zostały  
# zachowane — 70% przypadków z klasą 0 i 30% z klasą 1.

# W przypadku voices\_pl mamy   
# 50% głosów męskich oraz 50% głosów kobiecych.  
pca\_transform = PCA(PC\_num)  
X\_train\_pcaed = pca\_transform.fit\_transform(X\_train)  
X\_test\_pcaed = pca\_transform.transform(X\_test)

scaler = StandardScaler()  
X\_train\_pcaed\_scaled = scaler.fit\_transform(X\_train\_pcaed)  
X\_test\_pcaed\_scaled = scaler.transform(X\_test\_pcaed)

model = kNN(5, weights = 'distance')  
model.fit(X\_train\_pcaed\_scaled, y\_train)  
y\_predict = model.predict(X\_test\_pcaed\_scaled)

from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import accuracy\_score  
from sklearn.metrics import f1\_score  
# moduł seaborn  
import seaborn as sns  
cm = confusion\_matrix(y\_test, y\_predict)  
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")  
# 0 - Female (kobieta), 1 - Male (mężczyzna)

def DrawCM(confusion\_matrix, title=None):  
 sns.heatmap(confusion\_matrix, annot=True,   
 fmt="d", cmap="Blues",   
 xticklabels=['Female',' Male'],   
 yticklabels=['Female', 'Male'])  
 plt.xlabel("Predykcje")  
 plt.ylabel("Wartości rzeczywiste")  
 plt.title(f"Macierz pomyłek rozpoznawania głosu {title}")  
 plt.show()

DrawCM(cm, 'kNN(5)')

accuracy\_score(y\_test, y\_predict)  
# Accuracy liczone ręcznie  
(cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][1]+cm[0][1]+cm[1][0])

from sklearn.pipeline import Pipeline

def RunAndEvaluateModel(pipe\_object, xtrain, xtest, ytrain, ytest, title = None):  
 pipe\_object.fit(xtrain, ytrain)  
 ypredict = pipe\_object.predict(xtest)  
 cm = confusion\_matrix(ytest, ypredict)  
 if title:  
 DrawCM(cm, title)  
 else:  
 DrawCM(cm, type(pipe\_object.named\_steps['classifier']).\_\_name\_\_)  
 print(f"Accuracy: {accuracy\_score(ytest, ypredict)}")  
 print(f"f1\_score: {f1\_score(ytest, ypredict)}")

# PCA(62) daje precyzyjniejszą klasyfikacje niż PCA(PC\_num)   
pipe\_kNN = Pipeline([['transformer', PCA(62)],   
 ['scaler', StandardScaler()],  
 ['classifier', kNN(weights='distance')]])

pipe\_SVM = Pipeline([['transformer', PCA(62)],   
 ['scaler', StandardScaler()],  
 ['classifier', SVM()]])

pipe\_SVM\_SIG = Pipeline([['transformer', PCA(62)],   
 ['scaler', StandardScaler()],  
 ['classifier', SVM(kernel='sigmoid')]])

pipe\_SVM\_RBF = Pipeline([['transformer', PCA(62)],   
 ['scaler', StandardScaler()],  
 ['classifier', SVM(kernel='rbf')]])

pipe\_DT = Pipeline([['transformer', PCA(62)],   
 ['scaler', StandardScaler()],  
 ['classifier', DT()]])

RunAndEvaluateModel(pipe\_kNN, X\_train, X\_test, y\_train, y\_test)  
RunAndEvaluateModel(pipe\_SVM, X\_train, X\_test, y\_train, y\_test, 'SVM')  
RunAndEvaluateModel(pipe\_SVM\_RBF, X\_train, X\_test, y\_train, y\_test, 'SVM\_RBF')  
RunAndEvaluateModel(pipe\_SVM\_SIG, X\_train, X\_test, y\_train, y\_test, 'SVM\_SIG')  
RunAndEvaluateModel(pipe\_DT, X\_train, X\_test, y\_train, y\_test)

# Random Forest (las losowy )to algorytm oparty na połączeniu wielu drzew   
# decyzyjnych, który jest odporny na przeuczenie i dobrze sprawdza się w  
# przypadkach z wieloma cechami.

from sklearn.ensemble import RandomForestClassifier  
pipe\_RF = Pipeline([['transformer', PCA(62)],   
 ['scaler', StandardScaler()],  
 ['classifier', RandomForestClassifier(  
 n\_estimators=100, random\_state=42)]])  
RunAndEvaluateModel(pipe\_RF, X\_train, X\_test, y\_train, y\_test)