**KORELACJA**

arr = np.array([[2, 3, 5, 1], [5, 1, 2, 8], [5, 1, 6, -1]])  
kolumna = arr[:, 0]  
wiersz = arr[2, :]  
obszar = arr[0:2, 1:4]  
obszar\_pomijajac\_koncowe\_indeksy = arr[:2, 1:]  
parzyste\_kolumny = arr[:, ::2]  
nieparzyste\_kolumny = arr[:, 1::2]  
odwrocone\_kolumny = arr[:, ::-1]

arr = np.array([[2, 3, 5, 1], [5, 1, 2, 8], [5, 1, 6, -1]])  
sumy\_kolumn = arr.sum(axis=0)  *# wynik: [12, 5, 13, 8]*sumy\_wierszy = arr.sum(axis=1)  *# wynik: [11, 16, 11]*

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

import numpy as np  
from matplotlib import pyplot as plt  
x = np.arange(0, 10, 0.1)  
y = np.sin(x\*\*2 - 5 \* x + 3)  
plt.scatter(x, y) *# wykres punktowy - punkty na wykresie*  
plt.plot(x, y) *# wykres ciagly - linia ciagla pomiedzy punktami*  
plt.xlabel("X")  
plt.ylabel("Y")  
plt.show()  
*# wykres słupkowy – bar  
# mapa cieplna – imshow  
#wykresy pudełkowe – boxplot  
# histogram - hist*

# 2 wykresy obok siebie  
import numpy as np  
from matplotlib import pyplot as plt  
x = np.arange(0, 10, 0.1)  
y = np.sin(x\*\*2 - 5 \* x + 3)  
fig, ax = plt.subplots(1, 2, figsize=(10, 5)) *# dla dwuwymiarowej plt.subplots(2, 2, figsize = (…))*ax[0].plot(x, y) # dla dwuwymiarowej ax[0,0]  
ax[0].set\_xlabel("X")  
ax[0].set\_ylabel("Y")  
ax[1].scatter(x, y)  
ax[1].set\_xlabel("X")  
ax[1].set\_ylabel("Y")  
fig.tight\_layout()  
plt.show()

df = pd.read\_csv("arkusze/StudentPerformanceFactors.csv", sep=",")  
df.head()  *# wyświetlenie pierwszych 5. wierszy ramki danych (wyświetlenie jeśli print(df.head()))*  
*# zmiana wartości kategorycznych na wartości logiczne (zmiana z "Low" na 1) w ramce danych*  
df["Parental\_Involvement"] = df["Parental\_Involvement"].replace({

        "Low": 1, "Medium": 2, "High": 3,

    })  
df["Internet\_Access"] = df["Internet\_Access"].replace({"Yes": 1, "No": 0})  
*# Faktoryzacja (zmiana wartości kategorycznych na liczby całkowite)*for col in df.select\_dtypes(include=["object"]).columns:  
 df[col], \_ = pd.factorize(df[col])  
data = pd.Series(["apple", "banana", "apple", "orange", "banana", "apple"])  
codes, uniques = pd.factorize(data)  
print("Zakodowane wartosci: ", codes)  *# Zakodowane wartosci:  [0 1 0 2 1 0]*print(  
    "Unikalne wartosci: ", uniques  
)  *# Unikalne wartosci:  Index(['apple', 'banana', 'orange'], dtype='object')  
# obliczanie macierzy korelacji ma na celu zbadanie związków między różnymi zmiennymi w danych*correlation\_matrix = df.corr()  
*# correlation\_matrix = df.select\_dtypes(include=[np.number]).corr()*plt.figure(figsize=(16, 16))  
sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", square=True)  
plt.show()

Kodowanie One-Hot Encoding

Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie

df2 = pd.DataFrame({"kolor": ["czerwony", "zielony", "niebieski", "czerwony"]})  
print(df2)  
df2\_encoded = pd.get\_dummies(df2, columns=["kolor"])  
print(df2\_encoded)

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

**REGRESJA LINIOWA**

Regresji używa się, aby przewidzieć wartość zmiennej zależnej (cecha ilościowa).

import numpy as np  
import pandas as pd  
from matplotlib import pyplot as plt  
from sklearn.linear\_model import LinearRegression

np.random.seed(2001)  *# ustawienie ziarna w celu powtarzalnosci danych*  
x = np.arange(-3, 3, 0.1)  
y = (  
    np.tanh(x) + np.random.randn(x.shape[0]) \* 0.2  
)  *# wartosci tanh lekko zaklocane/zaburzane*  
*# funkcja randn generuje losowe liczby z rozkladu normalnego (o sredniej 0 i odchyleniu standardowym 1)*

plt.plot(x, np.tanh(x), color="green")  
plt.scatter(  
    x, y, marker="o"  
)  *# parametr marker pozwala na wybor wizualnej reprezentacji punktow na wykresie*  
plt.legend(["tanh(x)", "wygenerowane dane"])  
plt.show()

*# model wymaga aby dane byly tablicami dwuwymiarowymi (macierzami) a nie tak jak wczesniej tablicami jednowymiarowymi*  
x = x.reshape(-1, 1)  
y = y.reshape(-1, 1)  
x.shape  
y.shape  
model = LinearRegression()  
model.fit(x, y)  *# uczenie modelu*  
ypred = model.predict(x)  *# uzywa nauczonego wczesniej modelu do przewidzenia wartosci y*  
model.coef\_  *# zmienna zawiera wartość współczynnika regresji, który pokazuje, jak bardzo zmienia się y, gdy x zmienia się o jednostkę*  
model.intercept\_  *# zmienna zawiera wartość wyrazu wolnego w równaniu regresji, który jest stałą wartością dodawaną do wyniku regresji, niezależną od x*

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) - aproksymujaca", "f(x) - aproksymowana zaszumiona"])  
plt.show()

Obraz zawierający zrzut ekranu, diagram, linia, Wykres

Opis wygenerowany automatycznieObraz zawierający zrzut ekranu, linia, Wykres, diagram

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznie

import pandas as pd  
import numpy as np  
from sklearn.linear\_model import LinearRegression  
from sklearn.model\_selection import train\_test\_split  
from matplotlib import pyplot as plt

bh\_data = pd.read\_excel("arkusze/housing.xlsx")  
bh\_arr = bh\_data.values  
X, y = bh\_arr[:, :-1], bh\_arr[:, -1]  
*# X - wszystkie wiersze, kolumny od pierwszej do przedostatniej*  
*# y - cala ostatnia kolumna*  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
    X, y, test\_size=0.2, random\_state=221, shuffle=False  
)  
*# test\_size=0.2 - Określa, jaką część danych ma stanowić zestaw testowy. Wartość 0.2 oznacza, że 20% danych zostanie użyte do testowania, a reszta (80%) do trenowania modelu*  
linReg = LinearRegression()  
linReg.fit(X\_train, y\_train)  
y\_pred = linReg.predict(X\_test)  
min\_val = min(y\_test.min(), y\_pred.min())  
max\_val = max(y\_test.max(), y\_pred.max())  
plt.scatter(y\_test, y\_pred)  
plt.plot([min\_val, max\_val], [min\_val, max\_val], color="black")  
plt.xlabel("Cena medianowa - rzeczywista (y\_test)")  
plt.ylabel("Cena medianowa - przewidziana przez model (y\_pred)")  
plt.show()

from sklearn.metrics import (  
    mean\_absolute\_error,  
    mean\_squared\_error,  
    mean\_absolute\_percentage\_error,  
)  
MSE = mean\_squared\_error(y\_test, y\_pred, squared=True)  
*# srednia kwadratow bledow czyli roznic miedzy y\_test-y\_pred*

MAE = mean\_absolute\_error(y\_test, y\_pred)  
*# srednia wartosc bezwzgledna roznic miedzy wartosciami rzeczywistymi a przewidywanymi*

MAPE = mean\_absolute\_percentage\_error(y\_test, y\_pred)  
*# miara wzgledna, ktora wyraza sredni blad bezwzgledny jako procent wartosci rzeczywistej*

print("Średni błąd kwadratowy (MSE):", MSE)  
print("Średni błąd bezwzględny (MAE):", MAE)  
print("Średni błąd procentowy (MAPE):", MAPE)

plt.boxplot(y\_train)  
plt.title("Medianowa wartosc mieszkania")  
plt.show()

outliers = np.abs((y\_train - y\_train.mean()) / y\_train.std()) > 3  
*# wartosci odstajace to te ktore znajduja sie o wiecej niz 3 odchylenia standardowe od sredniej*  
X\_train\_no\_outliers = X\_train[~outliers, :]  
y\_train\_no\_outliers = y\_train[~outliers]  
y\_train\_mean = y\_train.copy()  
y\_train\_mean[outliers] = y\_train.mean()  
linReg = LinearRegression()  
linReg.fit(X\_train, y\_train\_mean)  
y\_pred = 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)  
print("Średni błąd kwadratowy (MSE):", MSE2)  
print("Średni błąd bezwzględny (MAE):", MAE2)  
print("Średni błąd procentowy (MAPE):", MAPE2)  
wagi = linReg.coef\_  
kreślonej\_cechy = bh\_data.columns[:-1]  *# bh\_data.columns zwraca nazwy kolumn*  
x = np.arange(len(kreślonej\_cechy))  
fig, ax = plt.subplots(1, 1)  
ax.bar(x, wagi)  *# rysuje wykres słupkowy*  
ax.set\_xticks(x)  *# ustawia pozycje ticków na osi X*  
ax.set\_xticklabels(kreślonej\_cechy, rotation=90)  *# ustawia etykiety ticków na osi X*  
fig.show()  
plt.show()

nowe\_dane = np.stack([X[:, 4] / X[:, 7],   
                      X[:, 4] / X[:, 5],   
                      X[:, 4] \* X[:, 3],   
                      X[:, 4] / X[:, -1]],  
                        axis=1,  
)  *# stack laczy ciag tablic kreśl kreślonej kierunku axis=-1 oznacza ze nowe cechy np. dodawane jako dodatkowe kolumny*  
X\_additional = np.concatenate(  
    [X, nowe\_dane], axis=-1  
)  *# concatenate sluzy do laczenia kreś lub kreśl tablic kreśl kreślonej osi*

Obraz zawierający tekst, zrzut ekranu, linia, diagram

Opis wygenerowany automatycznieObraz zawierający tekst, diagram, zrzut ekranu, linia

Opis wygenerowany automatycznie

Obraz zawierający tekst, zrzut ekranu, diagram, linia

Opis wygenerowany automatycznie

**ALGORYTM k NAJBLIŻSZYCH SĄSIADÓW I MASZYNY WEKTORÓW NOŚNYCH**

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

pd.options.mode.copy\_on\_write = False  
df = pd.DataFrame({"foo": [10, 20, 30], "bar": [4, 5, 6]})  
print(df)  
df["foo"][2] = 3  
print(df)  
df["foo"] = [-8, -8, -3]  
print(df)

pd.options.mode.copy\_on\_write = True  
df = pd.DataFrame({"foo": [10, 20, 30], "bar": [4, 5, 6]})  
print(df)  
df["foo"][2] = 3  
print(df)  
df["foo"] = [-8, -8, -3]  
print(df)

*# od wersji pandas 3.0 modyfikacja konkretnego elementu moze odbyc sie przy pomocy funkcji df.iloc*  
df.iloc[0, 0] = 3  
print(df)

import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
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  
from matplotlib import pyplot as plt  
from sklearn.preprocessing import StandardScaler  
from sklearn.tree import DecisionTreeClassifier as DT  
from sklearn.tree import plot\_tree

data\_const = pd.read\_excel("arkusze/loan\_data.xlsx")  
data = pd.read\_excel("arkusze/loan\_data.xlsx")  
columns = list(data.columns)  *# pobranie nazw kolumn*  
mask = data["Gender"].values == "Female"  
data.loc[mask, "Gender"] = 1  
data.loc[~mask, "Gender"] = 0  
print(  
    data["Property\_Area"].unique()  
)  *# wyswietlenie wystepujacych wartosci zmiennej kategorycznej*  
print(  
    data["Property\_Area"].value\_counts()  
)  *# zliczenie wystapien unikatowych wartosci zmiennej kategorycznej*  
one\_hot = pd.get\_dummies(data.Property\_Area)  
print(one\_hot)  
one\_hot = one\_hot.astype(int)  *# chcemy zeby wartosci byly 1/0 a nie True/False*  
print(one\_hot)  
data = pd.concat(  
    [data, one\_hot], axis=1  
)  *# dolaczenie nowych kolumn do oryginalnej ramki danych (1/0)*  
data = data.drop(columns=["Property\_Area"])  
data["Married"], \_ = pd.factorize(data["Married"])  
data["Self\_Employed"], \_ = pd.factorize(data["Self\_Employed"])  
data["Education"] = data["Education"].replace({"Graduate": 1, "Not Graduate": 0})

features = data.columns  
ko = data["Loan\_Status"]  
data = data.drop(columns=["Loan\_Status"])  
data["Loan\_Status"] = ko  *# dodanie kolumny na koncu*  
data["Loan\_Status"] = data["Loan\_Status"].replace({"Y": 1, "N": 0})  
vals = data.values.astype(np.float64)  
X = vals[:, :-1]  *# wszystkie wiersze i wszystkie kolumny oprocz ostatniej*  
y = vals[:, -1]  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
    X, y, test\_size=0.2, random\_state=221, shuffle=False  
)  
models = [kNN(), SVM()]  
model\_names = ["k-Nearest Neighbors", "Support Vector Machine"]  
class\_labels = ["Loan\_Status: No", "Loan\_Status: Yes"]

*# metryki*  
accuracies = []  
precisions = []  
recalls = []  
f1s = []  
roc\_aucs = []

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,  
    )  *# parametr fmt oznacza format liczb*  
    plt.xlabel("Predykcje")  
    plt.ylabel("Wartosci rzeczywiste")  
    plt.title(f"Macierz pomylek dla {name}")  
    plt.show()

    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))

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 slupkow*  
width = 0.35  *# szerokosc*  
x = np.arange(len(metrics))  *# pozycje na osi x*

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("Porownanie metryk dla kNN i SVM")  
ax.set\_xticks(x)  
ax.set\_xticklabels(metrics)  
ax.legend()

plt.tight\_layout()  
plt.show()

plt.figure(figsize=(16, 12))  
for i, column in enumerate(data.columns):  
    plt.subplot(3, 5, i + 1)  
    plt.boxplot(data[column])  
    plt.title(column)  
plt.show()

scaler = StandardScaler()  
scaler.fit(  
    X\_train[:, [2, 5, 6, 7, 8]]  
)  *# oblicza srednia i odchylenie dla kazdej cechy w zbiorze danych treningowych X\_train*  
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)  
    sns.heatmap(  
        cm,  
        annot=True,  
        fmt="d",  
        cmap="Blues",  
        xticklabels=class\_labels,  
        yticklabels=class\_labels,  
    )  
    plt.xlabel("Predykcje")  
    plt.ylabel("Wartosci rzeczywiste")  
    plt.title(f"Macierz pomyłek dla {name}")  
    plt.show()  
    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))

width = 0.9  
x = np.arange(2)  
metryki = np.array(  
    [  
        (accuracies, accuracies\_2),  
        (precisions, precisions\_2),  
        (recalls, recalls\_2),  
        (f1s, f1s\_2),  
        (roc\_aucs, roc\_aucs\_2),  
    ]  
)  
fig, axs = plt.subplots(ncols=3, nrows=2, figsize=(16, 9))  
axs = axs.flatten()  
axs[5].axis("off")  
for i, m in enumerate(metryki):  
    axs[i].bar(x - width / 2, [m[0][0], m[1][0]], width, color=["#4CAF50", "#FF9800"])  
    axs[i].set\_ylabel("Dokladność")  
    axs[i].set\_title(  
        f'Porównanie {["Dokładności", "Precyzji", "Czułości", "F1 Score", "ROC AUC"]}'  
    )

axs[i].set\_xticks(x - width / 2)  
    axs[i].set\_xticklabels(["Model podstawowy", "Model zmodyfikowany"])

fig.suptitle("Porównanie metryk kNN", fontsize=16)  
plt.tight\_layout(rect=[0, 0.03, 1, 0.95])  
plt.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
    X, y, test\_size=0.2, random\_state=221, shuffle=False  
)  
model = DT(max\_depth=3)  
model.fit(X\_train, y\_train)  
y\_pred = model.predict(X\_test)  
cm = confusion\_matrix(y\_test, y\_pred)  
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  
)

**Obraz zawierający tekst, zrzut ekranu, Czcionka, diagram

Opis wygenerowany automatycznie** **Obraz zawierający tekst, zrzut ekranu

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, Wykres, diagram

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, diagram, Plan, Równolegle

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, diagram, Czcionka

Opis wygenerowany automatycznie** **Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, diagram, Wielobarwność

Opis wygenerowany automatycznie**

**Obraz zawierający tekst, zrzut ekranu, diagram, numer

Opis wygenerowany automatycznie**

**LABORATORIUM ZADANIA**

* *Laboratorium 2*

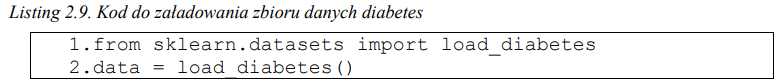
Zadanie 2.1. Pierwsze spojrzenie Pobierz plik „housing.xlsx” ze strony kursu. Podobnie do zadań z poprzednich zajęć (Zadanie 1.4) wygeneruj macierz korelacji dla wczytanego zbioru. Przeanalizuj macierz korelacji. Jakie zależności mogą mieć związek, a jakie są przypadkowe? Wygeneruj wykresy korelacji pomiędzy cechami niezależnymi a cecha zależną (medianową ceną mieszkania).

Zadanie 2.2.Wielokrotne testowanie modelu Zmodyfikuj skrypt, który pokazuje Listing 2.4, tak, żeby wyszedł skrypt pozwalający na wielokrotne przetestowanie modelu regresji liniowej. Skrypt umieść w oddzielnej funkcji, która jako argument przyjmie liczbę powtórzeń, które trzeba wykonać. Podpowiedź: w funkcji zastosuj pętlę for. Za każdym razem zbiór danych ma zostać podzielony na podzbiory: uczący oraz testowy w sposób losowy, w tym celu nie podawaj argumentu random\_state, aby wyniki za każdym razem się różniły. Jako wynik eksperymentu ma być zwrócona średnia wartość miary mean\_absolute\_percentage\_error – czyli średni procent błędu regresji.

Zadanie 2.3. Uwzględnienie wartości odstających Wykonaj Zadanie 2.2 dodając do niego procedurę usuwania/zastępowania wartości odstających. Porównaj wyniki uzyskane w poprzednim zadaniu z nowymi wynikami.

Zadanie 2.4. Generacja nowych cech Spróbuj zaproponować cechy/kombinacje cech, które mogły by ulepszyć jakość predykcji regresji liniowej. Przetestuj zaproponowane rozwiązania.

Zadanie 2.5. Samodzielne badanie danych. Za pomocą kodu, który przedstawia Listing 2.9, załaduj zbiór danych Diabetes. Przeanalizuj go podobnie do tego, jak zrobiliśmy ze zbiorem danych „Boston Housing”.



import pandas as pd  
import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_absolute\_percentage\_error  
from sklearn.preprocessing import PolynomialFeatures, StandardScaler  
from sklearn.datasets import load\_diabetes

def zadanie2\_1(data):  
    corr\_of\_data = data.corr()  
    *#print(corr\_of\_data)*  
    dependent\_var = 'MedianowaCena'  
    independent\_var = [col for col in data.columns if col != dependent\_var]  
    num\_plots = len(independent\_var)  
    num\_cols = 3  
    num\_rows = (num\_plots + num\_cols - 1) // num\_cols  
fig, ax = plt.subplots(num\_rows, num\_cols, figsize=(10,  
5\*num\_rows),squeeze=False,sharex='col',sharey="row",gridspec\_kw={'hspace':0.5,'wspace':0.3})  
  
for i, var in enumerate(independent\_var):  
        row = i // num\_cols  
        col = i % num\_cols  
        ax[row, col].scatter(data[var], data[dependent\_var], alpha=0.5)  
        ax[row, col].set\_xlabel(var)  
        ax[row, col].set\_ylabel(dependent\_var)  
        ax[row, col].grid(True)      
if num\_plots % num\_cols != 0:  
        for j in range(num\_cols - num\_plots % num\_cols):  
            fig.delaxes(ax[num\_rows - 1, num\_plots % num\_cols + j])  
    plt.autoscale()  
    plt.show()

def zadanie2\_2(data, repeat):  
    bh\_arr = data.values  
    x,y = bh\_arr[:,:-1], bh\_arr[:,-1]  
    avg\_error = 0  
    for i in range(repeat):  
        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.2)  
        linReg = LinearRegression()    
        linReg.fit(x\_train,y\_train)  
        y\_pred = linReg.predict(x\_test)  
        mse = mean\_squared\_error(y\_test,y\_pred)  
        mae = mean\_absolute\_error(y\_test,y\_pred)  
        mape = mean\_absolute\_percentage\_error(y\_test,y\_pred)  
        avg\_error += mape

return avg\_error/repeat

def zadanie2\_3(data,repeat):  
    bh\_arr = data.values  
    x,y = bh\_arr[:,:-1], bh\_arr[:,-1]  
    avg\_error = 0  
    for i in range(repeat):  
        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.2)  
        outliers = np.abs((y\_train - y\_train.mean())/y\_train.std()) > 3  
        x\_train\_clean = x\_train[~outliers,:]  
        y\_train\_clean = y\_train[~outliers]  
        linReg = LinearRegression()    
        linReg.fit(x\_train\_clean,y\_train\_clean)  
        y\_pred = linReg.predict(x\_test)  
        mse = mean\_squared\_error(y\_test,y\_pred)  
        mae = mean\_absolute\_error(y\_test,y\_pred)  
        mape = mean\_absolute\_percentage\_error(y\_test,y\_pred)  
        avg\_error += mape

        return avg\_error/repeat

def zadanie2\_4(data,repeat):  
    bh\_arr = data.values  
    x, y = bh\_arr[:,:-1], bh\_arr[:,-1]  
    avg\_error = 0  
    poly = PolynomialFeatures(degree=2, include\_bias=False)  
    scaler = StandardScaler()  
    for i in range(repeat):  
        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2)  
        *#STANDARYZOWANIE CECH*  
        x\_train = scaler.fit\_transform(x\_train)  
        x\_test = scaler.transform(x\_test)  
        *#POLYNOMINOWANIE CECH*  
        x\_train\_poly = poly.fit\_transform(x\_train)  
        x\_test\_poly = poly.transform(x\_test)  
        outliers = np.abs((y\_train - y\_train.mean())/y\_train.std()) > 3  
        x\_train\_clean = x\_train\_poly[~outliers,:]  
        y\_train\_clean = y\_train[~outliers]  
        linReg = LinearRegression()  
        linReg.fit(x\_train\_clean,y\_train\_clean)  
        y\_pred = linReg.predict(x\_test\_poly)  
        mse = mean\_squared\_error(y\_test,y\_pred)  
        mae = mean\_absolute\_error(y\_test,y\_pred)  
        mape = mean\_absolute\_percentage\_error(y\_test,y\_pred)  
        avg\_error += mape

    return avg\_error/repeat

if \_\_name\_\_ == "\_\_main\_\_":

    bh\_data = pd.read\_excel('files/lab 2/practice\_lab\_2.xlsx')  
 print(zadanie2\_4(bh\_data,3))

* *Laboratorium 3*

Zadanie 3.2. Przekształcenie binarnych cech jakościowych   
Utwórz funkcję na podstawie skryptu, który przedstawia Listing 3.1 do funkcji o następującym nagłówku:   
def qualitative\_to\_0\_1(data, column, value\_to\_be\_1):   
Funkcja ma za zadanie zwrócenia zmiennej data z kolumną o nazwie ‘column’ zawierającą wartości 0 albo 1. Wartości 1 przyporządkuj do wartości ‘value\_to\_be\_1‘ kolumny ‘column’. Za pomocą utworzonej funkcji przekształć resztę binarnych cech jakościowych.

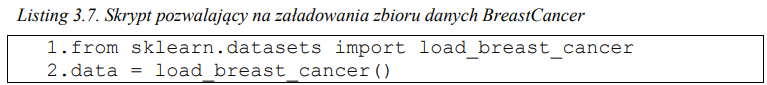
Zadanie 3.3. Kodowanie 1-z-n   
Korzystając z kodu, który przedstawia Listing 3.2, zamień cechę ‘Purpose’ na zbiór cech o nazwach poszczególnych kategorii i wartościach 0/1.

Zadanie 3.4. Wyznaczenie wartości metryk klasyfikacji   
Dla danych, które przedstawia Tabela 3.3Tabela 3.4 wyznacz wartości wszystkich metryk klasyfikacji.

Zadanie 3.5. Wpływ parametrów na jakość pracy modeli klasyfikacji   
Przetestuj algorytmy kNN oraz SVM z różnymi parametrami: kNN – liczba sąsiadów (n\_neighbors), sposób wyznaczania wag (‘uniform’, ‘distance’), SVM – jądro (kernel)

Zadanie 3.6. Wpływ metody skalowania   
Zbadaj wpływ różnych metod skalowania na jakość klasyfikacji za pomocą metod SMV oraz kNN. Przetestuj klasy StandardScaler, MinMaxScaler, RobustScaler.

Zadanie 3.7. Samodzielna klasyfikacja   
Za pomocą kodu, który przedstawia Listing 3.7 załaduj dane, pozwalające na wytrenowanie modelu do odróżnienia nowotworów łagodnych od złośliwych. Przeprowadź analizę za pomocą omawianych metod klasyfikacji. Zbuduj drzewo decyzyjne o wysokości 5, wygeneruj jego ilustrację, przedyskutuj, jakie cechy mają wpływ na wynik.



import pandas as pd  
import numpy as np  
from matplotlib import pyplot as plt  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import confusion\_matrix, mean\_absolute\_error, mean\_squared\_error, mean\_absolute\_percentage\_error  
from sklearn.preprocessing import MinMaxScaler, PolynomialFeatures, RobustScaler, StandardScaler  
from sklearn.svm import SVC  
from sklearn.neighbors import KNeighborsClassifier as kNN  
from sklearn.datasets import load\_breast\_cancer  
from sklearn.tree import DecisionTreeClassifier as DT  
from sklearn.tree import plot\_tree

def zadanie3\_2(data,column,value\_to\_be\_1):

mask = data[column].values == value\_to\_be\_1  
data.loc[mask,column] = 1  
data.loc[~mask,column] = 0

return data

def zadanie3\_3(data,column):

one\_hot = pd.get\_dummies(data[column])  
data = pd.concat([data,one\_hot],axis=1)  
data = data.drop(columns=[column])

    return data

def zadanie3\_4(TP,FN,FP,TN):

sensitivity = TP/(TP+FN)  
precision = TP/(TP+FP)  
specificity = TN/(FP+TN)  
accuracy = (TP+TN)/(TP+FN+FP+TN)

    return sensitivity, precision, specificity, accuracy

def zadanie3\_5(data):

bh\_arr = data.values  
x,y = bh\_arr[:,:-1], bh\_arr[:,-1]  
y = y.astype(int)  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=41)  
scaler = RobustScaler() *#tu jest zadanie 3.6 MinMaxScaler najlepszy*scaler.fit(x\_train)  
x\_train = scaler.transform(x\_train)  
x\_test = scaler.transform(x\_test)  
models = [kNN(), SVC()]  
for model in models:  
    model.fit(x\_train,y\_train)  
    y\_pred = model.predict(x\_test)  
    cm = confusion\_matrix(y\_test,y\_pred)  
    print(f"Confusion Matrix for {model.\_\_class\_\_.\_\_name\_\_}:\n{cm}\n

def zadanie3\_7():

data = load\_breast\_cancer()  
x = data.data  
y = data.target  
x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.2,random\_state=42)  
dtree = DT(max\_depth=5,random\_state=42)  
dtree.fit(x\_train,y\_train)  
plt.figure(figsize=(20,10))  
plot\_tree(dtree,feature\_names=data.feature\_names, class\_names=data.target\_names, filled=True)  
plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

pd.set\_option('display.max\_columns',None)  
data = pd.read\_excel('files/lab 3/practice\_lab\_3.xlsx')  
data = zadanie3\_2(data,'Gender','Female')  
data = zadanie3\_2(data,'Married','Yes')  
data = zadanie3\_2(data,'Education','Graduate')  
data = zadanie3\_2(data,'Self\_Employed','Yes')  
data = zadanie3\_2(data,'Loan\_Status','Y')  
data = zadanie3\_3(data,'Dependents')  
data = zadanie3\_3(data,'Property\_Area')  
zadanie3\_5(data)

* *Laboratorium 4*

Zadanie 4.1.Samodzielna ekstrakcja cech.   
Powtórz operacje które pokazują Listing 4.1-Listing 4.3.

Zadanie 4.2. Wielokrotne przeprowadzenie eksperymentów klasyfikacji   
Przeprowadź 30 cykli uczenia-testowania dla danych uzyskanych za pomocą FFT. Przetestuj algorytmy kNN, SVM oraz Decision Tree. Dla każdego cyklu testowania wygeneruj macierz pomyłek. Uśrednij macierze, pomyśl, jaką płeć metody wykrywają łatwiej? Uzasadnij odpowiedź. Podpowiedź: spróbuj znaleźć uzasadnienie w macierzy pomyłek.

Zadanie 4.3. Samodzielne zastosowanie PCA   
Pobierz ze strony kursu zbiór danych ‘voice\_extracted\_features.csv’. Przeprowadź na nim analizę z zastosowaniem PCA: 1. podziel zbiór na podzbiory, 2. wygeneruj wykres punktowy pozostawiając 2 składowe główne, etykiety klas zastosuj do oznaczenia kolorów punktów odpowiadających poszczególnym obserwacjom, 3. zbuduj wykres procentu wyjaśnionej wariancji, dobierz optymalną liczbę cech dla progu 95%, 4. przeprowadź klasyfikację tworząc własny obiekt typu Pipeline.

Zadanie 4.4. Definicja własnej klasy wyznaczającej optymalną liczbę składowych głównych dla PCA   
Napisz klasę, której obiekt może być dołączony do obiektu typu Pipeline. Niech ta klasa pozwala na dobór liczby cech na podstawie procentu wyjaśnionej wariancji. Podpowiedź: aby obiekt klasy można było dołączyć do Pipeline’u, musi on mieć takie metody, jak: ‘fit(x)’, ‘transform(x)’, ‘fit\_transform(x)’.

Zadanie 4.5. Definicja własnej klasy odpowiedzialnej za usuwanie wartości odstających   
Napisz klasę, której obiekt może być dołączony do obiektu typu Pipeline. Ta klasa mazapewniać znalezienie wartości odstających oraz ich eliminację poprzez zastąpienie średnią.

import pandas as pd  
import os  
import numpy as np  
from scipy.io import wavfile  
from scipy.fft import fft  
from matplotlib import pyplot as plt  
from sklearn.base import BaseEstimator, TransformerMixin  
from sklearn.decomposition import PCA  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.metrics import confusion\_matrix  
from sklearn.preprocessing import StandardScaler

def zadanie4\_1(path):

    fs, audio\_data = wavfile.read(path,mmap=True)  
    if audio\_data.ndim > 1:  
        audio\_data = np.mean(audio\_data, axis=1)

    x\_raw = np.zeros((1, len(audio\_data)))  
    x\_raw[0, :len(audio\_data)] = audio\_data  
    x\_fft = np.abs(fft(x\_raw, axis=1)) / x\_raw.shape[1]  
    resolution = fs / len(audio\_data)  
    mean\_num = max(1,int(round(resolution)))  
    x\_fft = np.reshape(x\_fft, (x\_fft.shape[0], x\_fft.shape[1] // mean\_num, mean\_num))  
    x\_fft = x\_fft.mean(axis=-1)  
    low\_cut = 50  
    high\_cut = 280  
    x\_fft\_cut = x\_fft[:, low\_cut:high\_cut]  
    x\_fft\_cut /= np.expand\_dims(x\_fft\_cut.max(axis=1), axis=-1)  
    *# freq\_axis = np.linspace(low\_cut, high\_cut, x\_fft\_cut.shape[1])*  
    *# plt.figure(figsize=(8, 6))*  
    *# plt.plot(freq\_axis, x\_fft\_cut[0, :])*  
    *# plt.xlim(low\_cut, high\_cut)*  
    *# plt.grid(True)*  
    *# plt.show()*

    return x\_fft\_cut[0,:]

def zadanie4\_2():

    paths = ['files/voices/man\_sample\_1.wav','files/voices/man\_sample\_2.wav','files/voices/man\_sample\_3.wav','files/voices/man\_sample\_4.wav',\

             'files/voices/woman\_sample\_1.wav','files/voices/woman\_sample\_2.wav','files/voices/woman\_sample\_3.wav','files/voices/woman\_sample\_4.wav',]

    labels = [0,0,0,0,1,1,1,1]  
    features = []  
    for path in paths:  
        features.append(zadanie4\_1(path))

    x, y = np.array(features), np.array(labels)  
    conf\_matrix\_knn = np.zeros((2,2))  
    conf\_matrix\_svm = np.zeros((2,2))  
    conf\_matrix\_dt = np.zeros((2,2))

    for \_ in range(30):  
        x\_train, x\_test, y\_train, y\_test = train\_test\_split(x,y,test\_size=0.2)  
        scaler = StandardScaler() *#sluzy do przeksztalcania danych tak aby mialy srednia 0 i  
odchylenie standardowe 1*  
        x\_train = scaler.fit\_transform(x\_train)   
        x\_test = scaler.transform(x\_test)

        knn = KNeighborsClassifier(n\_neighbors=1)  
        knn.fit(x\_train,y\_train)  
        y\_pred\_knn = knn.predict(x\_test)  
        conf\_matrix\_knn += confusion\_matrix(y\_test,y\_pred\_knn, labels=[0,1])

        svm = SVC()  
        svm.fit(x\_train,y\_train)  
        y\_pred\_svm = svm.predict(x\_test)  
        conf\_matrix\_svm += confusion\_matrix(y\_test,y\_pred\_svm, labels=[0,1])

dt = DecisionTreeClassifier()  
dt.fit(x\_train,y\_train)  
y\_pred\_dt = dt.predict(x\_test)  
conf\_matrix\_dt += confusion\_matrix(y\_test,y\_pred\_dt, labels=[0,1])

    conf\_matrix\_knn /= 30  
    conf\_matrix\_svm /= 30  
    conf\_matrix\_dt /= 30  
    print("Wyniki kNN:")  
    print(conf\_matrix\_knn)  
    print("Wyniki SVM:")  
    print(conf\_matrix\_svm)  
    print("Wyniki Decision Tree:")  
    print(conf\_matrix\_dt)

def zadanie4\_3():

*#TODO nie ma tego pliku nigdzie takze nie zrobie tego*

    pass

*#zadanie 4.4*

class featuresCountBasedOnVariation(BaseEstimator, TransformerMixin):

    def \_\_init\_\_ (self,explained\_variance\_ratio=0.95):

*self*.explained\_variance\_ratio = explained\_variance\_ratio  
*self*.pca = None  
*self*.n\_components = None

    def fit(self, x, y=None):

*self*.pca = PCA()  
*self*.pca.fit(x)  
cumulative\_variance = np.cumsum(*self*.pca.explained\_variance\_ratio\_)  
*self*.n\_components = np.argmax(cumulative\_variance >= *self*.explained\_variance\_ratio) + 1  
*self*.pca = PCA(n\_components=*self*.n\_components)  
*self*.pca.fit(x)

        return *self*

    def transform(self,x):

        return *self*.pca.transform(x)

    def fit\_transform(self,x,y=None):

*self*.fit(x,y)

return *self*.transform(x)

*#zadanie 4.5*

class removeStandOuts(BaseEstimator,TransformerMixin):

    def \_\_init\_\_ (self,explained\_variance\_ratio=0.95):

*self*.explained\_variance\_ratio = explained\_variance\_ratio

*self*.pca = None

*self*.n\_components = None

    def fit(self,x,y=None):

*self*.feature\_means\_ = np.mean(x, axis=0)

        return *self*

    def transform(self, x):

x\_transformed = np.copy(x)  
for i in range(x.shape[1]):  
feature\_mean = *self*.feature\_means\_[i]  
feature\_std = np.std(x[:,i])  
threshold = 3 \* feature\_std  
outliers = np.abs(x[:,i]-feature\_mean) > threshold  
x\_transformed[outliers,i] = feature\_mean

        return x\_transformed

    def fit\_transform(self,x,y=None):

*self*.fit(x,y)

return *self*.transform(x)

if \_\_name\_\_ == "\_\_main\_\_":

    zadanie4\_2()

import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
import seaborn as sns  
from sklearn.preprocessing import StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsClassifier as kNN  
from sklearn.tree import DecisionTreeClassifier as DT  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC as SVM  
from sklearn.metrics import confusion\_matrix, accuracy\_score, f1\_score

# Etap 1:  
df = pd.read\_csv("data.csv")

# wyswietlenie 5 pierwszych wierszy  
print(df.head())

# podstawowy opis zbioru danych  
print(df.info())

# rozklad cech alcohol i color\_intensity  
fig, ax = plt.subplots(1, 2, figsize=(12, 6))  
ax[0].hist(df["Alcohol"], bins=20, edgecolor="black", color="skyblue")  
ax[0].set\_title("Rozkład cechy Alcohol")  
ax[0].set\_xlabel("Alcohol")  
ax[0].set\_ylabel("Liczba obserwacji")  
ax[1].hist(df["Color Intensity"], bins=20, edgecolor="black", color="skyblue")  
ax[1].set\_title("Rozkład cechy Color Intensity")  
ax[1].set\_xlabel("Color Intensity")  
plt.tight\_layout()  
plt.show()

# boxplot dla cechy Magnesium  
plt.figure(figsize=(6, 6))  
plt.boxplot(df["Magnesium"])  
plt.title("Boxplot dla cechy Magnesium")  
plt.ylabel("Magnesium")  
plt.show()

# uzupelnij ponizszy kod  
print(f"\nWystepuja {len(df[np.abs((df['Magnesium'] - df['Magnesium'].mean()) / df['Magnesium'].std()) > 3])} wartosci odstajace")  
print(f"Mediana cechy magnesium wynosi: {df['Magnesium'].median()}")  
print(f"Srednia wartosc cechy magnesium wynosi: {df['Magnesium'].mean()}")

# Etap 2:  
correlation\_matrix = df.corr()  
plt.figure(figsize=(16, 16))  
sns.heatmap(correlation\_matrix, annot=True, cmap="coolwarm", square=True)  
plt.title("Macierz korelacji")  
plt.show()

# uzupelnij kod  
print("Cecha najmocniej skorelowana dodatnio z\n cecha zalezna: [...]")  # nie wiem jak tutaj uzupełnić  
print("Cecha najmocniej skorelowana ujemnie z\n cecha zaleznia: [...]")  # nie wiem jak tutaj uzupełnić  
print("Najmocniej skorelowane dodatnio cechy: [...]")  # nie wiem jak tutaj uzupełnić

# Etap 3:  
y = np.array(df.iloc[:, -1])  
X = np.array(df.iloc[:, :-1])  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
    X\_scaled, y, test\_size=0.3, random\_state=29, shuffle=True  
)

# Etap 4:  
models = [  
    kNN(n\_neighbors=5),  
    SVM(kernel="rbf"),  
    DT(random\_state=29),  
    DT(random\_state=29, criterion="entropy"),  
    RandomForestClassifier(random\_state=29),  
]  
model\_names = [  
    "k-Nearest Neighbors",  
    "Support Vector Machine",  
    "Decision Tree",  
    "Decision Tree (entropy)",  
    "Random Forest",  
]  
accuracies = []  
f1s = []  
for model, name in zip(models, model\_names):  
    model.fit(X\_train, y\_train)  
    y\_pred = model.predict(X\_test)  
    accuracies.append(accuracy\_score(y\_test, y\_pred))  
    f1s.append(f1\_score(y\_test, y\_pred, average="weighted"))  
    cm = confusion\_matrix(y\_test, y\_pred)  
    print(f"Model: {name}")  
    print(f"Accuracy: {(accuracy\_score(y\_test, y\_pred))}")  
    print(f'F1-Score: {(f1\_score(y\_test, y\_pred, average="weighted"))}')  
    plt.figure(figsize=(10, 10))  
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")  
    plt.title(f"Macierz pomyłek dla {name}")  
    plt.xlabel("Predykcje")  
    plt.ylabel("Wartości rzeczywiste")  
    plt.show()