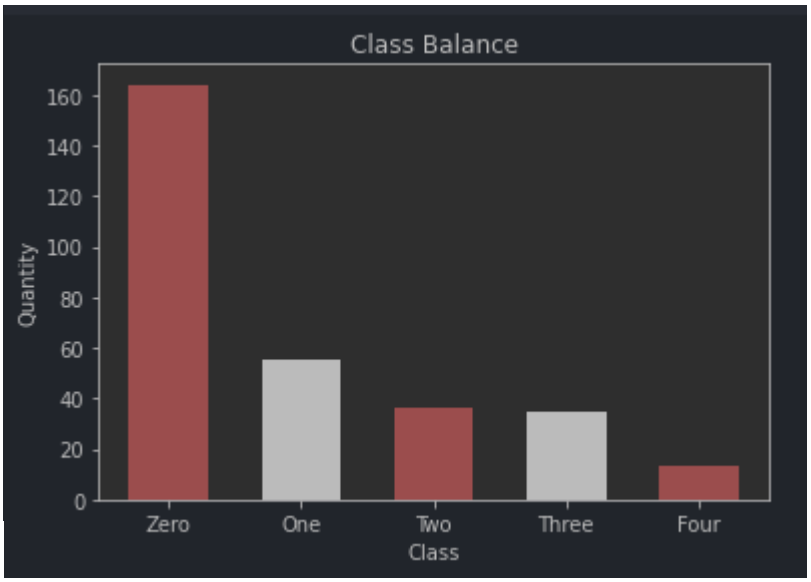
Raport 1

Krzysztof Maciejewski 260449

Analiza eksploracyjna

Czy zbiór jest zbalansowany pod względem liczby próbek na klasy?



1. Wykres zależności liczby próbek od klasy

Po przeprowadzeniu analizy liczby próbek należących do danej klasy, zauważyłem że najwięcej próbek przynależy do klasy 0, która mówi o jej braku. Pozostałe cztery klasy mają podobnie zbalansowane wartości.

Jakie są średnie i odchylenia cech liczbowych?

age	54.438944
trestbps	131.689769
chol	246.693069
thalach	149.607261
oldpeak	1.039604
ca	0.672241

Tabela przedstawia średnie wartości cech liczbowych

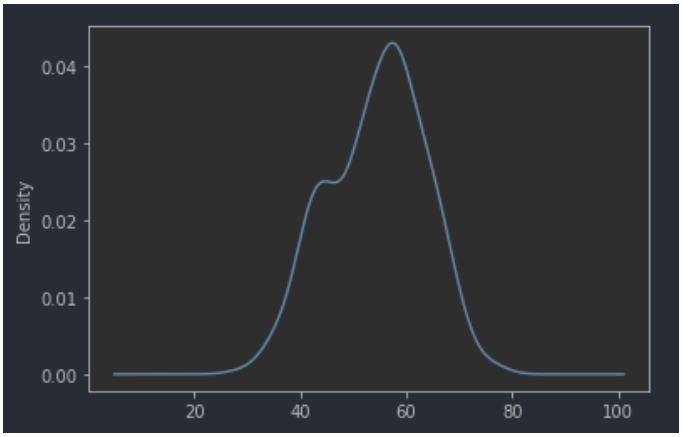
Średnie wartości cech liczbowych znacznie się od siebie różnią co pozwala stwierdzić, że w przyszłości trzeba będzie je znormalizować aby umożliwić wzajemne porównywanie i dalszą analizę.

age	9.038662
trestbps	17.599748
chol	51.776918
thalach	22.875003
oldpeak	1.161075
ca	0.937438

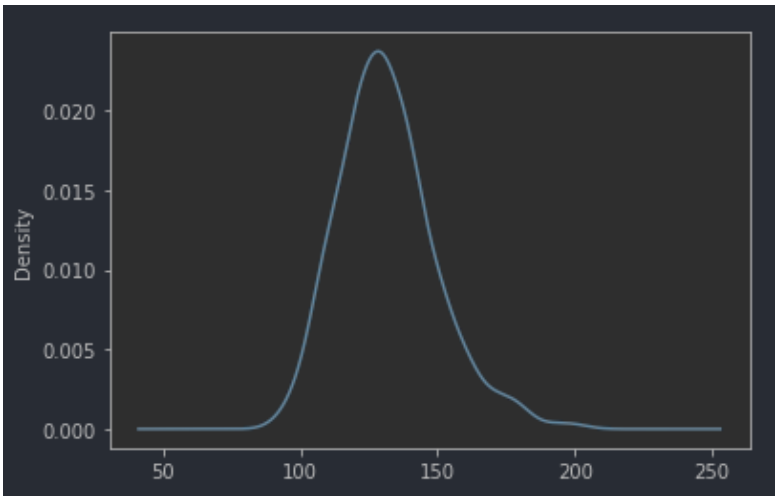
Tabela przedstawia odchylenia cech liczbowych

Wartości pozwalają stwierdzić, że takie dane jak np. wskaźnik cholesterolu mają bardzo zróżnicowane wartości i są mocno rozproszone.

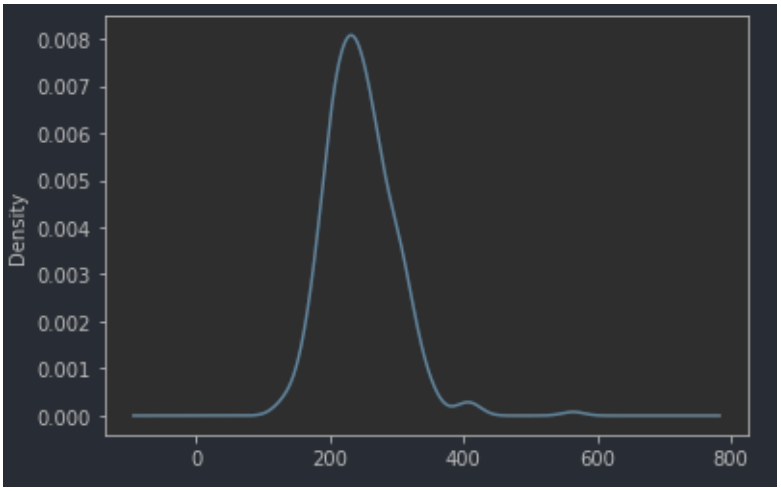
Dla cech liczbowych: czy ich rozkład jest w przybliżeniu normalny?



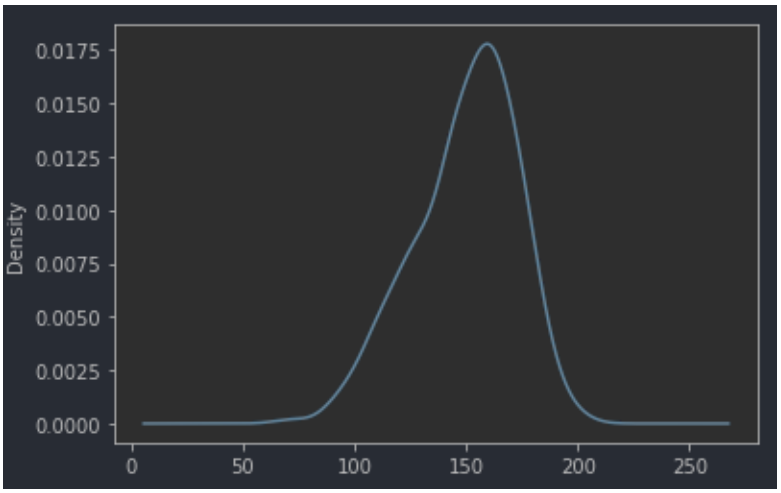
Rozkład wieku



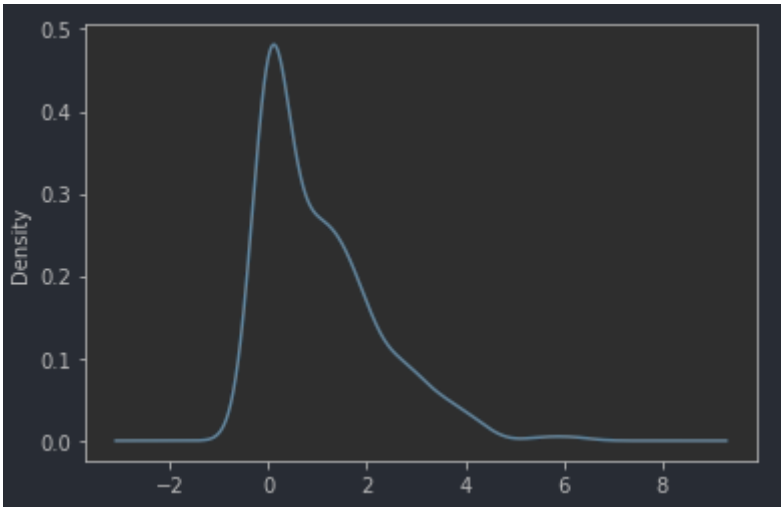
Rozkład resting blood pressure



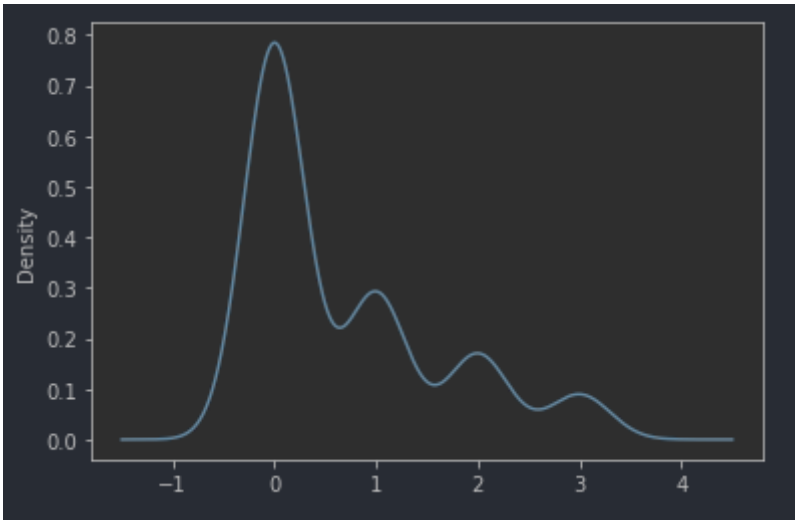
Rozkład cholesterolu



Rozkład maximum heart rate



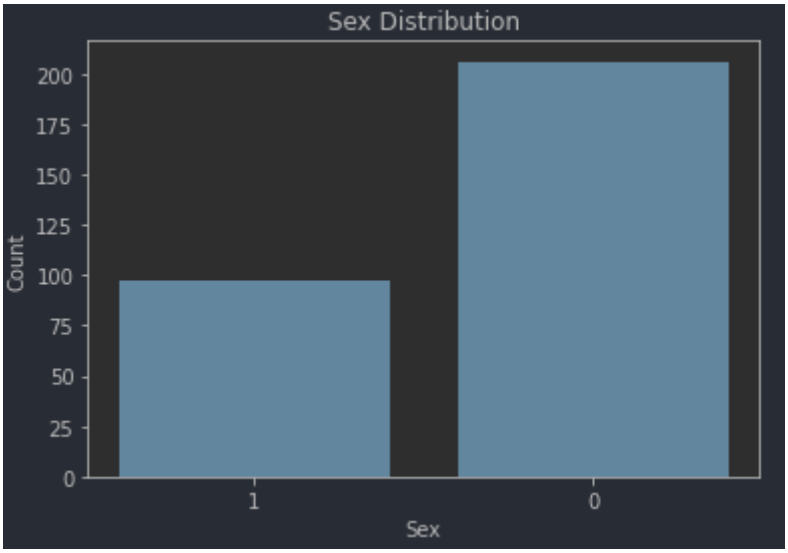
Rozkład depression induced by exercise



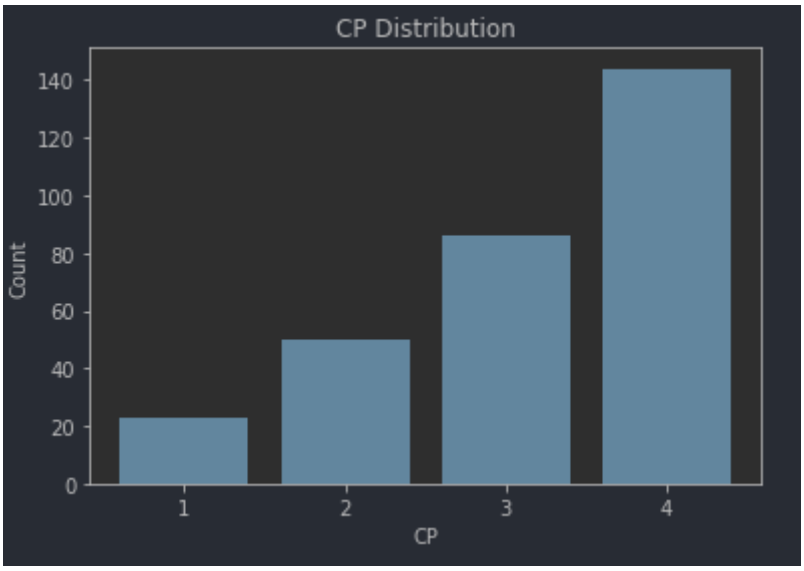
Rozkład cechy ca

Większość rozkładów dla cech liczbowych jest w przybliżeniu normalnych. Najbardziej od kształtu „dzwona” odbiegają dwa ostatnie wykresy. Wartości koncentrują się w nich wokół 0.

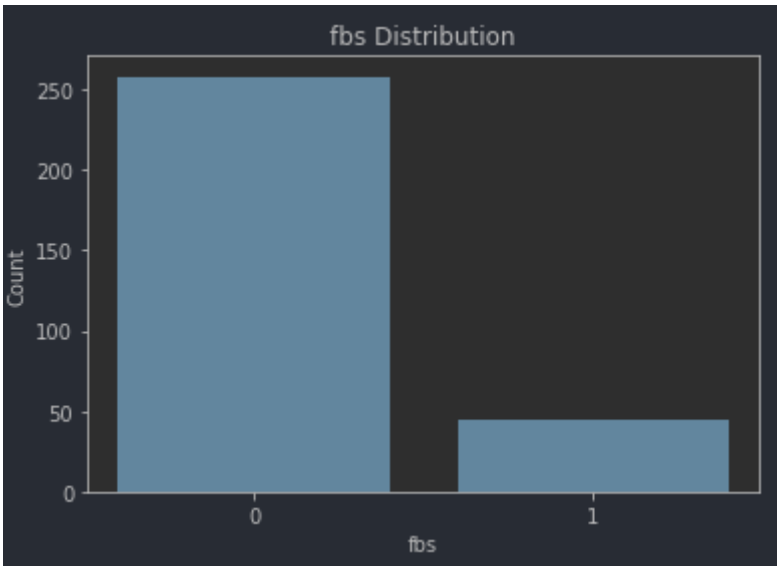
Dla cech kategorycznych: czy rozkład jest w przybliżeniu równomierny?



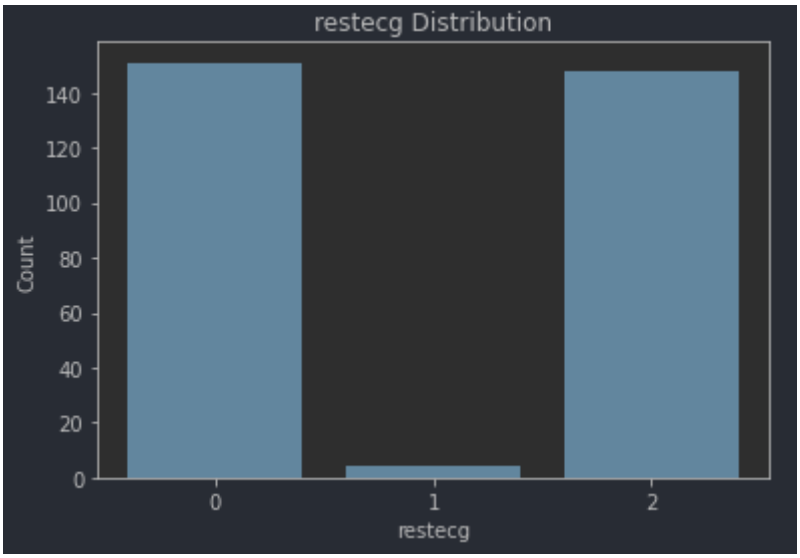
Rozkład cechy kategorycznej płeć



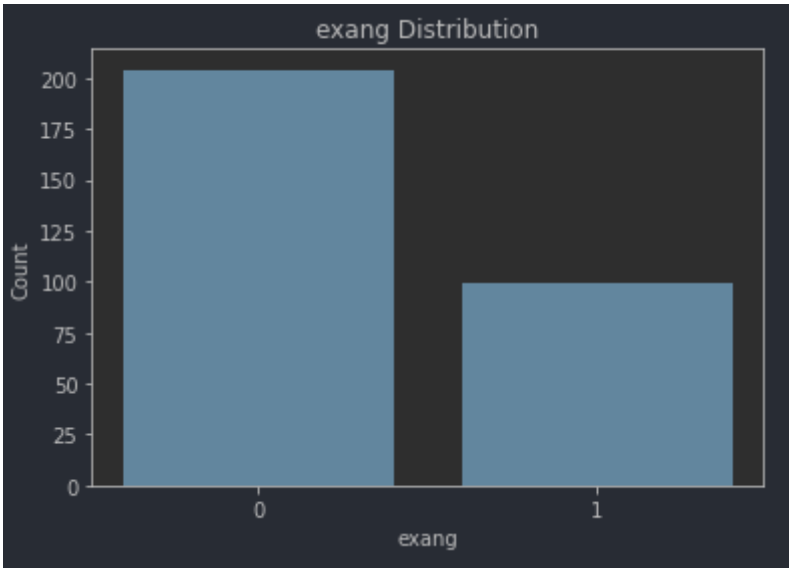
Rozkład cechy kategorycznej CP



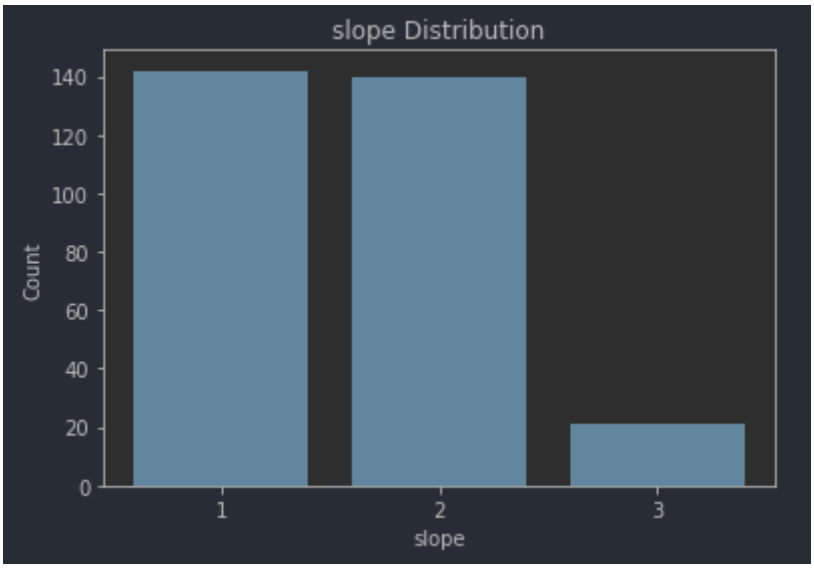
Rozkład cechy kategorycznej fbs



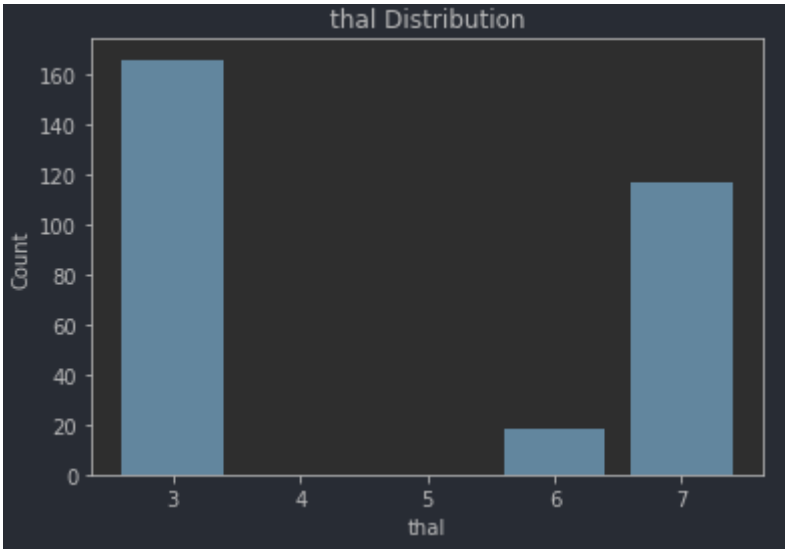
Rozkład cechy kategorycznej restecg



Rozkład cechy kategorycznej exang



Rozkład cechy slope



Rozkład cechy thal

Jak pokazują przedstawione rozkłady cech kategorycznych, żaden z nich nie jest rozkładem jednostajnym. Wartości tych cech występują nierównomiernie.

Czy występują cechy brakujące i jaką strategię możemy zastosować żeby je zastąpić?

W danych następujące cechy mają brakujące wartości: ca (cecha liczbowa) oraz thal (cecha kategoryczna). Cecha ca to liczba głównych naczyń zabarwionych za pomocą fluoroskopii. Po sprawdzeniu ile wartości jest brakujących, okazało się że dla ca są to 4 wartości, a dla thal 2 wartości. Stosunkowo jest to ilość, która nie jest duża więc nie warto byłoby w tych przypadkach odrzucać całą kolumnę. Ponieważ liczba wierszy z brakującymi wartościami jest mała, można by usunąć te wiersze.

Dla cechy liczbowej w przypadku rozkładu normalnego skuteczną strategią pozwalającą na nieodrzucanie wierszy byłoby wstawienie w miejsce brakującej wartości średniej z danej kolumny. Jednakże jak wynika z wykresu rozkładu cechy ca jej rozkład nie przypomina rozkładu normalnego,

dlatego w tym przypadku lepsze okazałoby się wstawienie mediany która jest mniej wrażliwa na wartości odstające.

Inną strategią było by wypełnienie brakujących kolumn w wierszach losowo wybraną wartością spośród wierszy posiadających wszystkie wartości. Technika ta jest odpowiednia zarówno dla cech liczbowych jak i kategorycznych.

Ostatnią strategią jest technika zwana Multiple Imputation, która pozwala na wyznaczenie brakujących wartości biorąc pod uwagę pozostałe wartości kolumn. Na początku zastępujemy brakujące wartości prostą strategią jak np. średnią danej kolumny. Brakujące wartości są wyznaczane na podstawie modelu regresji, w którym brakująca zmienna jest zmienną zależną, a pozostałe zmienne są zmiennymi niezależnymi. Następnie kolejna brakująca wartość jest używana jako zmienna zależna, a pozostałe jako niezależne. Proces trwa dopóki wszystkie brakujące wartości nie zostaną uwzględnione jako zmienne zależne. Po wyznaczeniu ich wartości początkowe tymczasowe wartości wyznaczone za pomocą prostej strategii są zastępowane przewidywaniami z modelu regresji. Proces zastępowania jest wykonywany kilka razy i wartości są aktualizowane po każdym z nich, aż do momentu gdy najlepiej odzwierciedlają relacje zidentyfikowane w danych.

Raport 2

Krzysztof Maciejewski 260449

Wstęp

Przy pomocy modelu regresji logistycznej wykorzystującej entropię krzyżową jako funkcję straty będę klasyfikował dane binarnie. Na początku pobieram dane i wypełniam brakujące wartości kolumn modą i medianą w zależności od tego czy jest to cecha kategoryczna czy liczbowa. Ponieważ będę klasyfikował dane binarnie zamieniłem klasy 1-4 na jedną wspólną klasę 1 (chory).

```
for index, row in Y.iterrows():
    if row[0] != 0:
        row[0] = 1 #jeżeli nie ma klasy 0 to ma klasę 1

y = Y.to_numpy()
X = (X-X.min())/(X.max()-X.min())
x = X.to_numpy()
```

Potem znormalizowałem dane i zamieniłem na np array.

Kolejno podzieliłem dane na zbiory treningowe i testowe oraz losowo zainicjowałem tablicę wag i bias.

```
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size= 0.2)

number_of_features = 13
#losowa inicjalizacja wag
weights = np.random.uniform(0.0, 100.0, size=number_of_features)
bias= np.random.rand(1)
```

Implementacja matematycznych funkcji

Następnie stworzyłem funkcję służącą jako sigmoid do obliczania wartości wyjściowych neuronów. Aby lepiej dostosować funkcję sigmoidalną (rozciągnąć ją) dodałem parametr, sprawiło to, że wartości neuronów stały się bardziej zróżnicowane.

```
def sigmoid(z):
    return 1 / (1 + np.exp(-z/200))
    # return 1 / (1 + np.exp(-(z-200)))
```

Następnie zdefiniowałem funkcje odpowiedzialne za wyliczanie funkcji straty i aktualizacje wag. Do wzoru entropii krzyżowej dodałem małą wartość która, zapobiega zwracaniu przez funkcję logarytmiczną -inf. W funkcji update_weights wykorzystuje wzór pochodnej wyprowadzony na wykładzie.

```
def loss_fun(y, y_pred):
    epsilon = 1e-15  # zapobieganie log(0)
    loss = - (y * np.log(y_pred + epsilon) + (1 - y) * np.log(1 - y_pred +
epsilon))
    return loss


def update_weights(X, y, y_pred, weights, bias, learning_rate):
    arX = np.squeeze(np.asarray(X))
    ary = np.squeeze(np.asarray((y_pred - y)))
    gradient_weights = np.dot(arX, ary)
    gradient_bias = np.sum(y_pred - y)
    #aktualizacja wag
    weights -= learning_rate * gradient_weights
    bias -= learning_rate * gradient_bias

    return weights, bias
```

Uczenie modelu

Model uczy się po jednym przykładzie. Zbieżność modelu zdefiniowałem jako wystarczająco małą zmianę funkcji kosztu (procentowo) i maksymalną liczbę epok.

```
learning_rate = 0.3
epochs = 1200
prev_loss = 1
avg_loss = 1
for epoch in range(epochs):
    loss_arr = []
    for i in range(len(x_train)):
        X = x_train[i]
        y = y_train[i]
        y_pred = sigmoid(np.dot(X, weights) + bias)
        loss = loss_fun(y, y_pred)
        loss_arr.append(loss)
        weights, bias = update_weights(X, y, y_pred, weights, bias,
learning_rate)
    if epoch % 100 == 0:
        prev_loss = avg_loss
        avg_loss = sum(loss_arr) / len(loss_arr)
        print(f"Epoch {epoch}: Average loss = {avg_loss}")
        #if (1-(avg_loss/prev_loss))*100 < 2: break #zbyt mała zmiana
f.kosztu procentowo

print("Trained Weights:", weights)
print("Trained Bias:", bias)
```

Następnie stworzyłem funkcję, która na podstawie przewidzianej wartości przyporządkowuje do niej klasę 0 lub 1.

```
def predict(x_data):
    y_preds = []
    for i in range(len(x_data)):
        X = x_data[i]
        y_pred = sigmoid(np.dot(X, weights) + bias)
```

```
            binary_prediction = 1 if y_pred >= 0.5 else 0
            y_preds.append(binary_prediction)
        return np.array(y_preds)
```

Ocena działania modelu

Dane treningowe

Wyniki:

Accuracy: 0.8388429752066116

Confusion: [[117 16]

[23 86]]

klasa	precision	recall	f1-score	support
0	0,84	0, 88	0,86	133
1	0,84	0,79	0,82	109

Dane testowe

```
y_pred = predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
confusion = confusion_matrix(y_test, y_pred)
report = classification_report(y_test, y_pred)
print(f'Accuracy: {accuracy}')
print(f'Confusion: {confusion}')
print(f'Report: {report}')
```

Wyniki:

Accuracy: 0.8524590163934426

Confusion: [[28 3]

[6 24]]

klasa	precision	recall	f1-score	support
0	0,82	0,90	0,86	31
1	0,89	0,80	0,84	30

Wnioski

Jak wynika z przedstawionych powyżej wyników accuracy jest wyższe w danych testowych, co jest niespodziewane. Może to prawdopodobnie wynikać z niereprezentatywnego podzielenia danych na treningowe i testowe. Właściwie wszystkie metryki oceny dla danych testowych wypadły lepiej albo porównywalnie do danych treningowych. Confusion matrix w obydwóch zbiorach wygląda podobnie i znacznie przeważają w niej wartości TP i TN co jest pożądanym rezultatem.