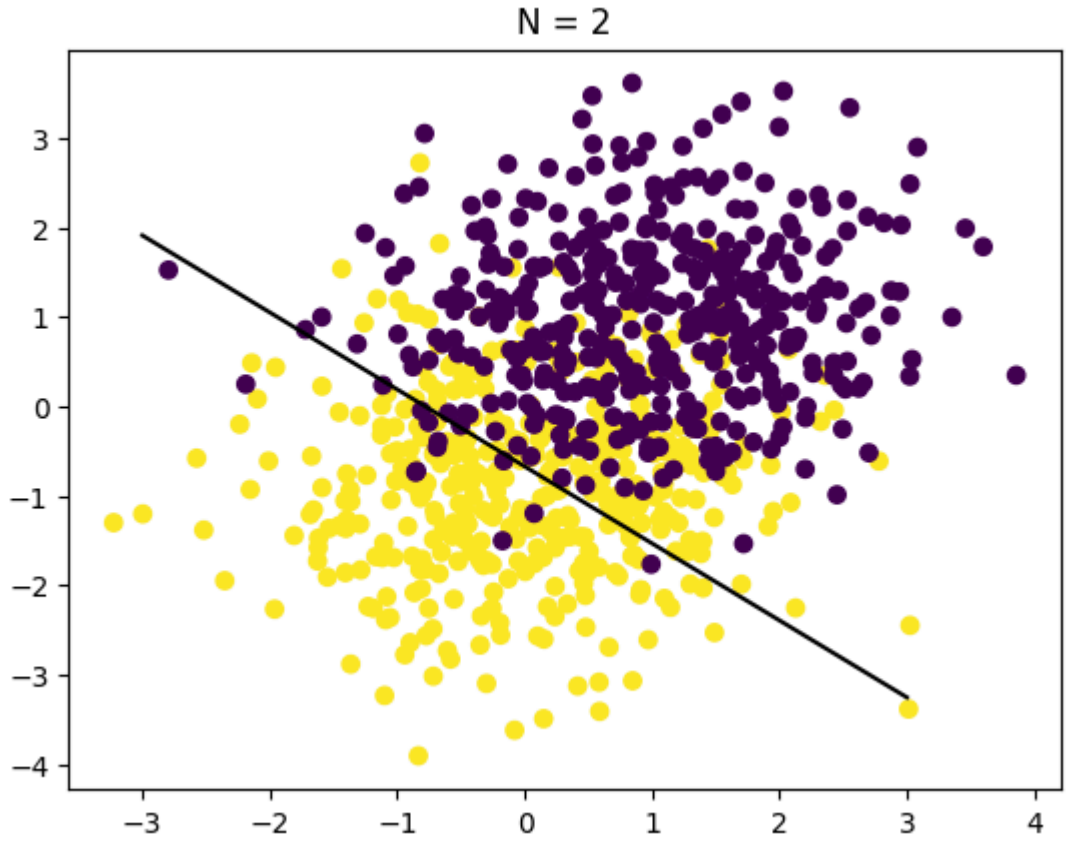
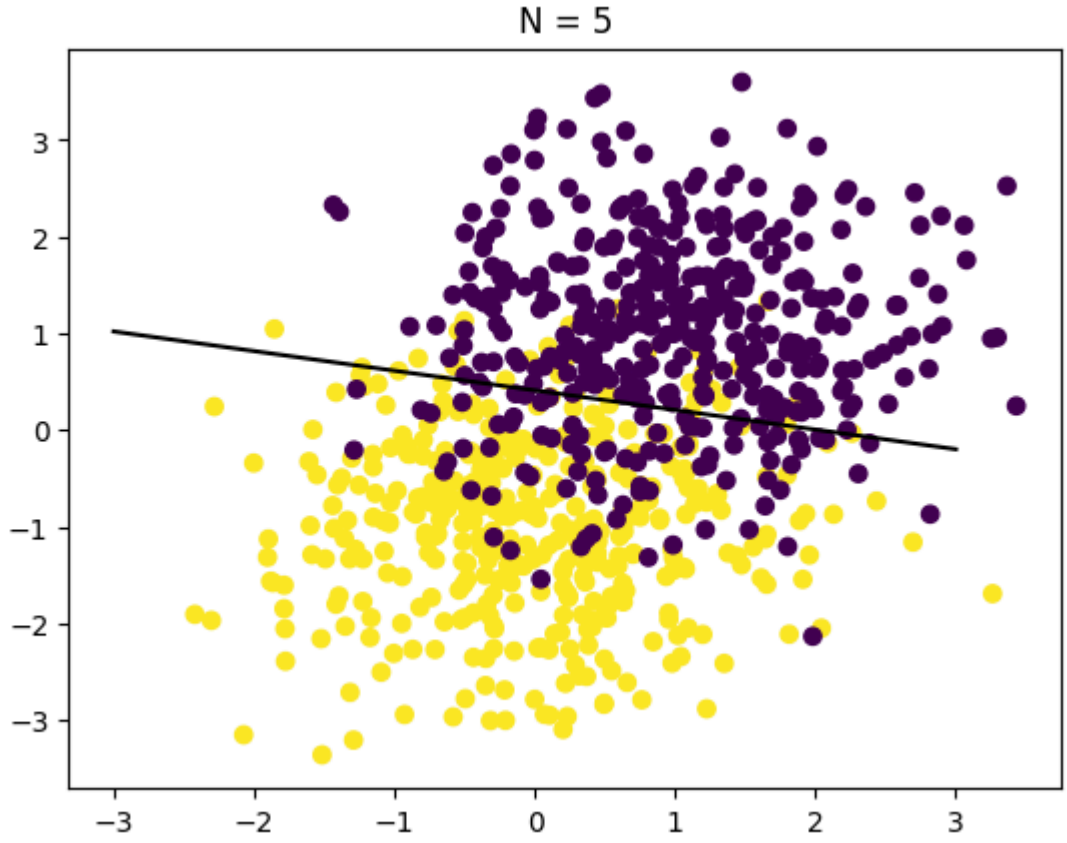
Proszę wygenerować po 400 2-wymiarowych punktów przypisanych do dwóch klas K1 i K2, pochodzących z rozkładów normalnych N([0,-1],1) i N([1,1],1) i podzielić je losowo na zbiory uczące i testujące w proporcji N do 400-N. **Proszę sprawdzić średnią dokładność klasyfikacji i podać odchylenie standardowe dla N = 2, 5, 10, 20, 100, powtarzając eksperyment 10 razy dla każdego N**. Dla każdego N dla jednej z powtórek proszę ustalić wzór hiperpłaszczyzny (w naszym wypadku - prostej) oddzielającej klasy, a następnie pokazać ją na wykresie razem z danymi (w sumie 5 wykresów).

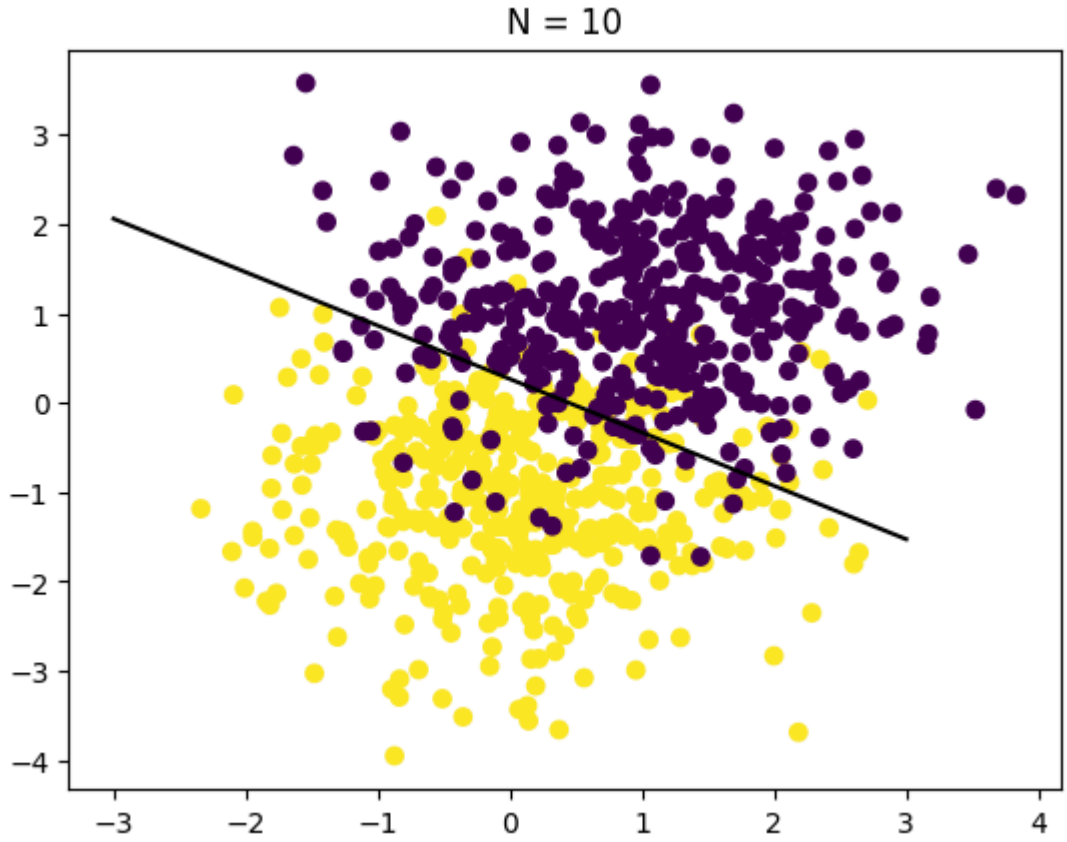
In [ ]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
mean1 = [0, -1]
mean2 = [1, 1]
cov = [[1, 0], [0, 1]]
N_values = [2, 5, 10, 20, 100]
exp_size = 10
for N in N_values:
    accur_list = []
    for i in range(exp_size):
        X1 = np.random.multivariate_normal(mean1, cov, 400)
        X2 = np.random.multivariate_normal(mean2, cov, 400)
        y1 = np.ones(400)
        y2 = np.zeros(400)
        X = np.concatenate((X1, X2))
        y = np.concatenate((y1, y2))
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=400-
        sgd_clf = SGDClassifier(loss="hinge", penalty="l2")
        sgd_clf.fit(X_train, y_train)
        y_pred = sgd_clf.predict(X_test)
        accur_list.append(accuracy_score(y_test, y_pred))
        if i == 0:
            plt.figure()
            plt.scatter(X[:, 0], X[:, 1], c=y)
            xx = np.linspace(-3, 3)
            yy = -sgd_clf.coef_[0][0]/sgd_clf.coef_[0][1] * xx - sgd_clf.interce
            plt.plot(xx, yy, 'k-')
            plt.title(f'N = {N}')
            plt.show()
    print(f'Wzór prostej: y = {sgd_clf.coef_[0][0]/sgd_clf.coef_[0][1]}x + {sgd_
    print(f'N = {N}, średnia dokładność: {np.mean(accur_list)}, odchylenie stand
```



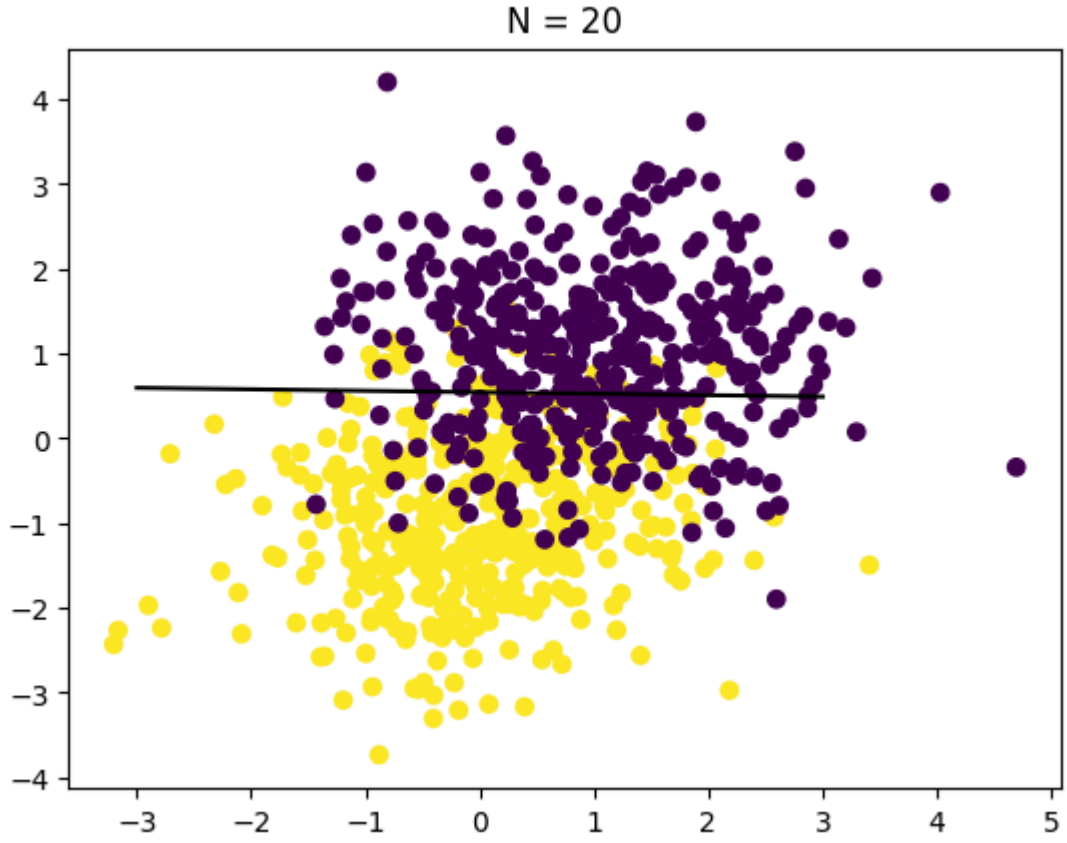

```
Wzór prostej: y = 0.13892932541111253x + [0.45067083]
N = 2, średnia dokładność: 0.8412060301507538, odchylenie standardowe: 0.03452008
943838311
```



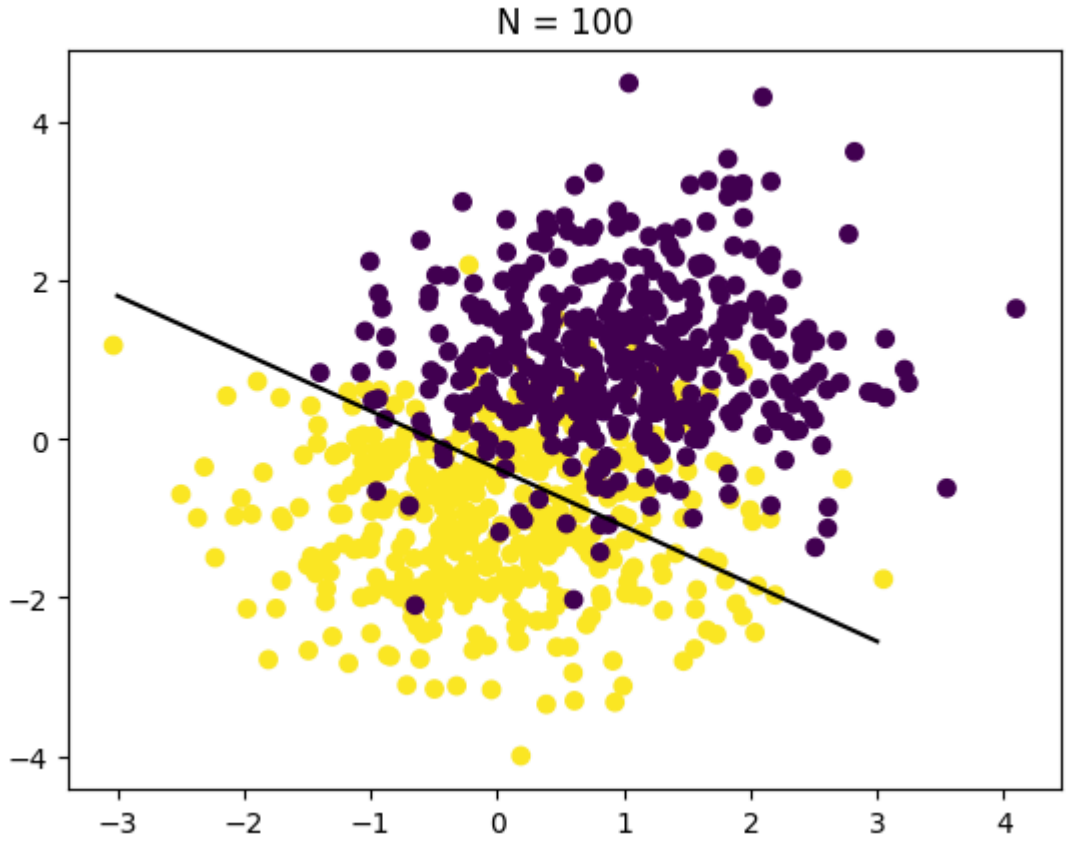

```
Wzór prostej: y = 0.026557454363429052x + [-0.05742353]
N = 5, średnia dokładność: 0.8437974683544305, odchylenie standardowe: 0.04022130
735236975
```




```
Wzór prostej: y = 0.3629463747870723x + [-0.32236761]
N = 10, średnia dokładność: 0.8405128205128204, odchylenie standardowe: 0.0335260
70007551076
```




```
Wzór prostej: y = 0.32689703406219733x + [-0.292954]
N = 20, średnia dokładność: 0.8344736842105263, odchylenie standardowe: 0.0329199
9030639061
```




```
Wzór prostej: y = 0.5741911357368953x + [-0.34575553]
N = 100, średnia dokładność: 0.8649999999999999, odchylenie standardowe: 0.017400
51084818423
```

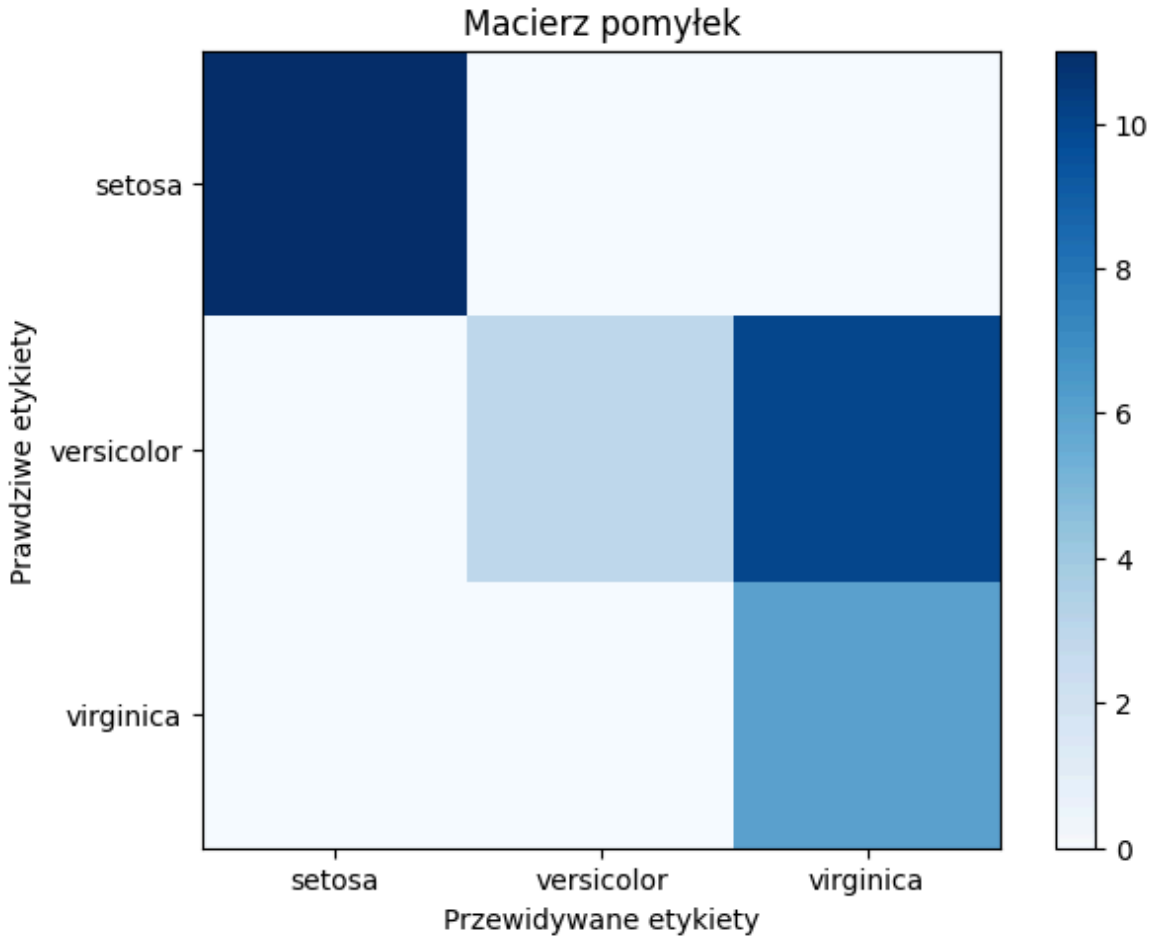
Wnioski: Dokładność jest najwyższa dla najwyższej ilości danych testujących, ale odchylenie standardowe pozostaje na podobnym poziomie. Wzór prostej dla N równego 100 jest wyraźnie inny niż dla reszty testów, co sugeruje, że potrzebna jest dużo większa wartość niż 20, aby osiągnąć większą dokładność. W ogólności jednak średnia dokładność oscyluje w granicach 85%, co jest wynikiem satysfakcjonującym.

Proszę pobrać zbiór https://archive.ics.uci.edu/ml/datasets/iris. Można to też zrobić w pythonie używając funkcji **sklearn.datasets.load_iris( )**. Następnie proszę dokonać samodzielnego podziału na dane uczące i testujące w proporcji 80%/20%. Proszę zbudować sieć złożoną z pojedynczej warstwy perceptronów (np. używając omawianej już tutaj funkcji sklearn.linear_model.Perceptron), której zadaniem będzie jak najdokładniejsza klasyfikacja gatunków irysów na podstawie ich pomiarów. **Proszę dokonać analizy macierzy pomyłek dla kilku uruchomień algorytmu. Jaką największą trafność jest w stanie uzyskać pojedyncza warstwa perceptronów w tym zadaniu?** Dlaczego? (Podpowiedź: polecamy przyjrzeć się pojęciu liniowej separowalności). **Proszę spróbować podzielić zbiór irysów na zbiór uczący i testujący na co najmniej 3 różne sposoby**. Jak duży jest wpływ podziału na wynik?

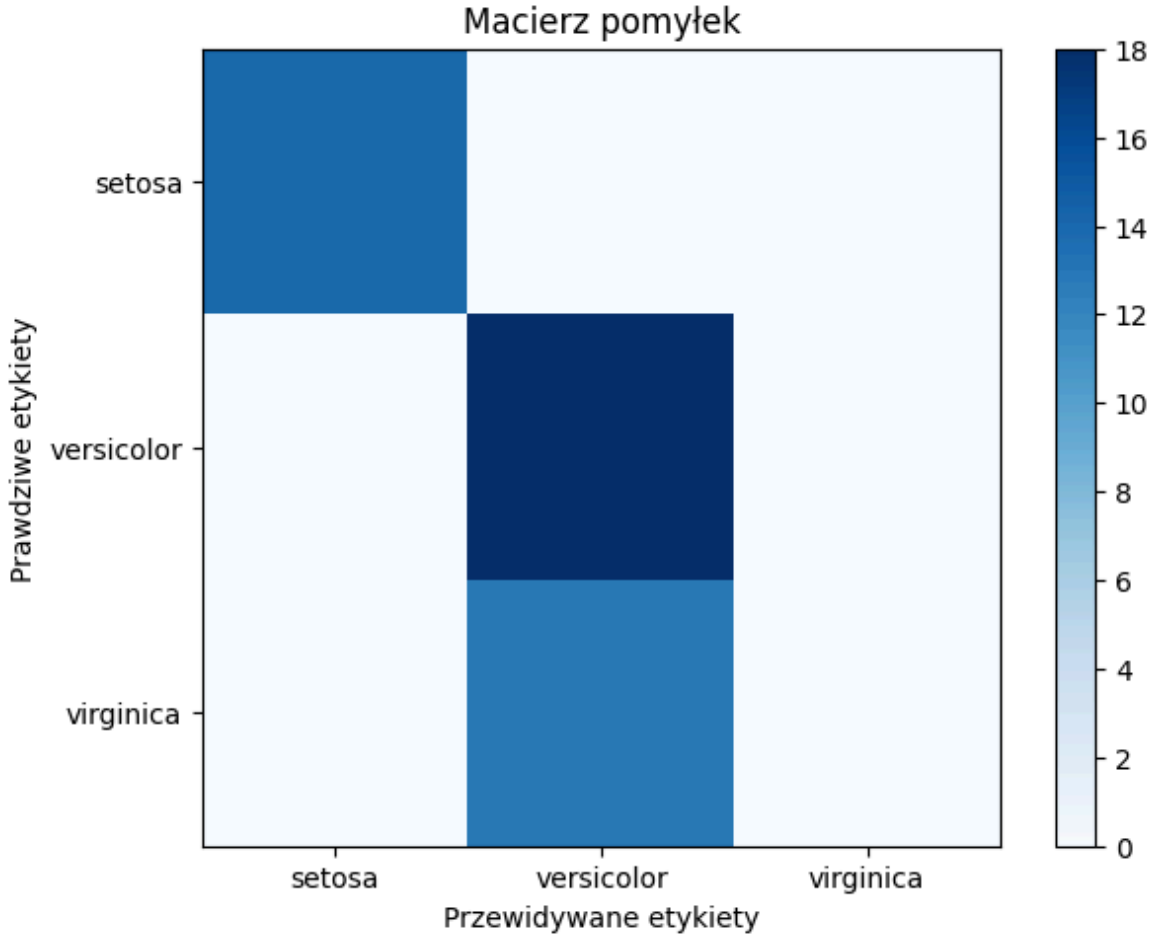
In [ ]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import confusion_matrix, accuracy_score
import matplotlib.pyplot as plt
def fun(X_train, X_test, y_train, y_test, split):
```

```
    clf = Perceptron(tol=1e-3, random_state=1)
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Dokładność: {accuracy} dla podziału {split}')
    cm = confusion_matrix(y_test, y_pred)
    plt.imshow(cm, interpolation='nearest',cmap=plt.cm.Blues)
    plt.title('Macierz pomyłek')
    plt.colorbar()
    tick_marks = np.arange(len(iris.target_names))
    plt.xticks(tick_marks, iris.target_names)
    plt.yticks(tick_marks, iris.target_names)
    plt.tight_layout()
    plt.ylabel('Prawdziwe etykiety')
    plt.xlabel('Przewidywane etykiety')
    plt.show()
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
fun(X_train, X_test, y_train, y_test, 0.2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_
fun(X_train, X_test, y_train, y_test, 0.3)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_
fun(X_train, X_test, y_train, y_test, 0.4)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_
fun(X_train, X_test, y_train, y_test, 0.5)
```

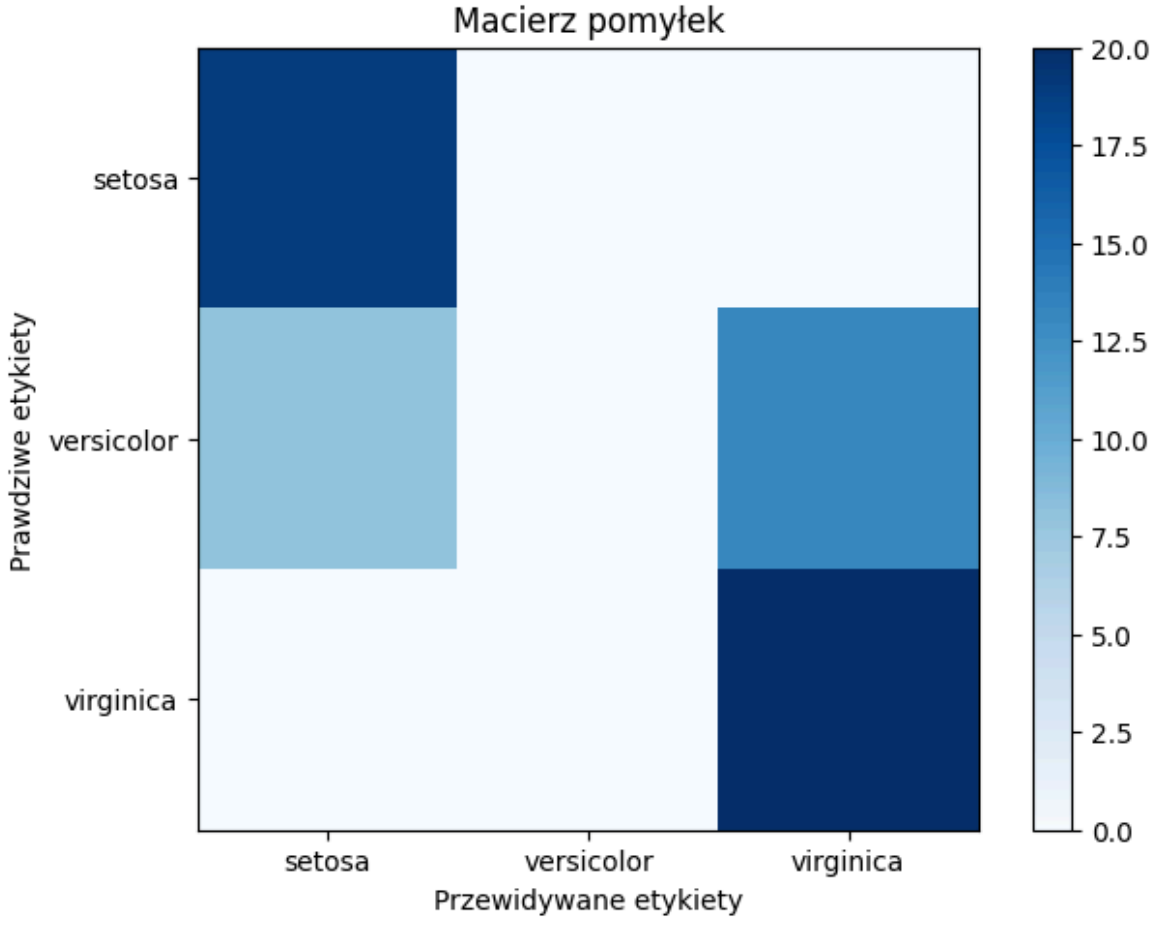
```
Dokładność: 0.6666666666666666 dla podziału 0.2
```



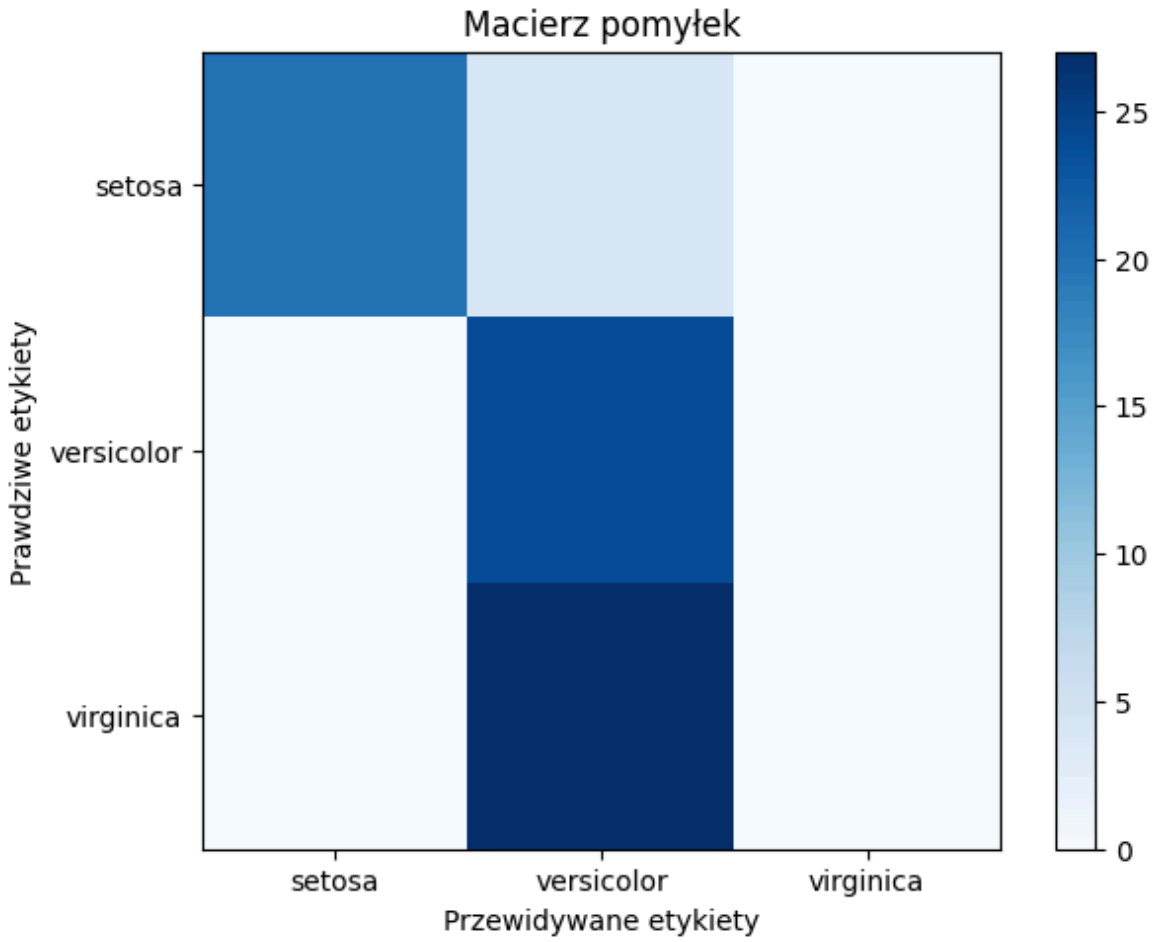

```
Dokładność: 0.7111111111111111 dla podziału 0.3
```




```
Dokładność: 0.65 dla podziału 0.4
```




```
Dokładność: 0.5866666666666667 dla podziału 0.5
```



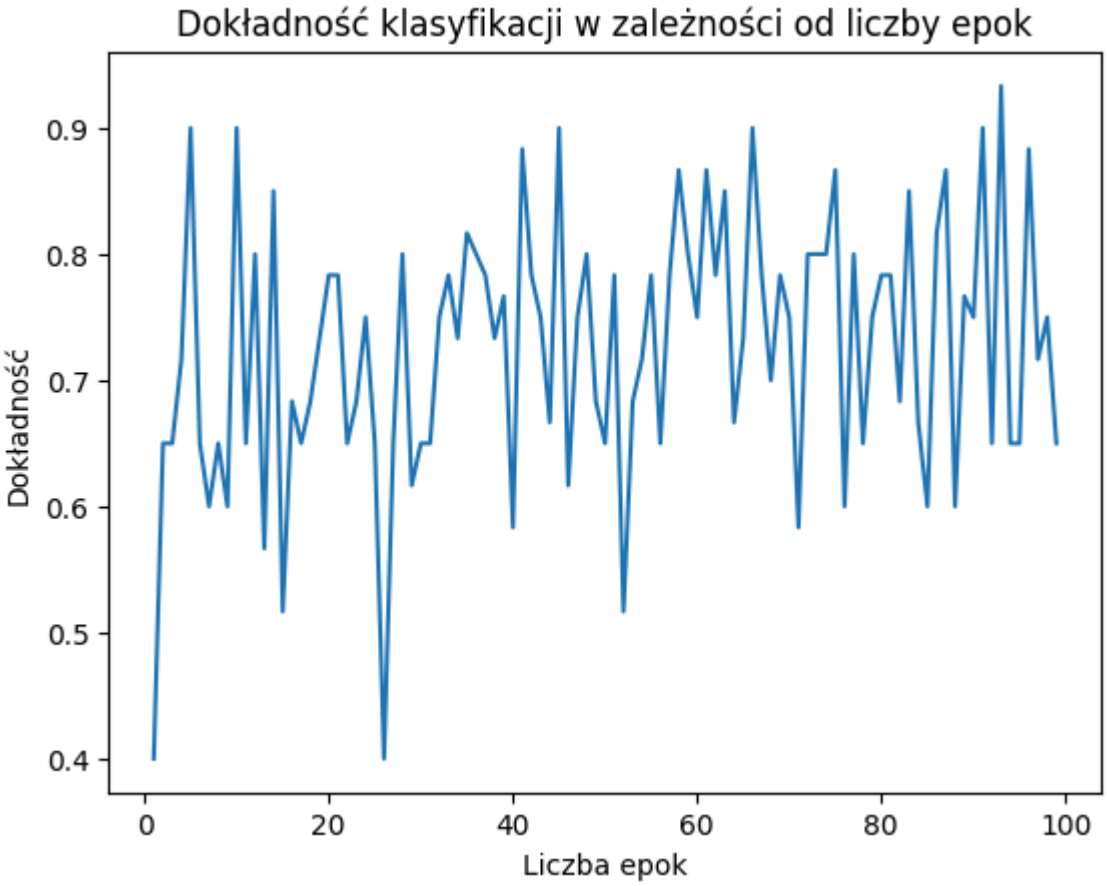
Wnioski: Dokładność klasyfikacji perceptronu zależy od podziału danych na zbiory treningowe i testowe. Najwyższą dokładność osiągnięto dla podziału 0.3, a najniższą dokładność dla podziału 0.5. Wyniki te sugerują, że podział danych ma znaczący wpływ na wydajność modelu. Jeśli chodzi o pojedyńcze warstwy perceptronów, największą dokładność ogranicza liniowa separowalność danych. Jeżeli danych nie da się liniowo separować, pojedyncza warstwa perceptronów nie będzie w stanie osiągnąć 100% dokładności, ponieważ nie jest w stanie modelować złożonych granic decyzyjnych.

Proszę sprawdzić, jak ilość epok wpływa na dokładność klasyfikacji zbioru irysów. Proszę przedstawić wnioski oraz wykres trafności klasyfikacji na zbiorze testującym w zależności od liczby epok.

Żeby zapobiec wcześniejszemu przerywaniu uczenia, w pakiecie Sklearn można ustalić argument *tol* na odpowiednio małą liczbę lub ustawiając argument *early_stopping* na False.

In [ ]:
```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
iris = load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_
```

```
accur_list = []
epochs = range(1, 100)
for epoch in epochs:
    clf = Perceptron(max_iter=epoch, tol=None, early_stopping=False, random_stat
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accur_list.append(accuracy)
plt.plot(epochs, accur_list)
plt.xlabel('Liczba epok')
plt.ylabel('Dokładność')
plt.title('Dokładność klasyfikacji w zależności od liczby epok')
plt.show()
```



Wnioski: Zgodnie z wykresem, dokładność klasyfikacji generalnie rośnie wraz z liczbą epok do pewnego punktu, po czym stabilizuje się. To sugeruje, że model potrzebuje pewnej minimalnej liczby epok do "nauczenia się" z danych, ale po przekroczeniu tego punktu, dodatkowe epoki nie przynoszą znaczącej poprawy dokładności. Na wykresie wartości mocno różnią się od sąsiednich, ale normalizując je widać wzrost oraz następującą po nim stabilizację.