MHE - sprawozdanie 3 - s26766@pjwstk.edu.pl - Kamil Koniak

Zadanie 2

Zaczynam od importów:

In [212…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, r2_score, classification_report, accuracy_score
```

Wczytuje dane i dla pewności usuwam NaN-y:

In [213…
```
df = pd.read_csv("C:/Users/kamil/Downloads/product.csv")

print("Braki danych:")
print(df.isnull().sum())

df = df.dropna()
```

```
Braki danych:
Unnamed: 0    0
x             0
y             0
dtype: int64
```

Przy pomocy metody IQR (rozstęp międzykwartylowy, różnica między trzecim a pierwszym kwartylem) usuwam próbki odstające:

In [214…
```
def remove_outliers_iqr(df):
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    IQR = Q3 - Q1
    return df[~((df < (Q1 - 1.5 * IQR)) | (df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
df_clean = remove_outliers_iqr(df)
```

Dzielę dane na cechy i etykiety:

In [215…
```
X = df_clean[['x']]
y = df_clean['y']
```

Dobrą praktyką jest standaryzacja danych tuż przed wykonaniem analizy danych, tak więc wykorzystuję do tego StandardScaler, skaluję cechy X:

In [216…
```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Ustawiam random_state=0 dzięki temu otrzymujemy te same zestawy treningowe i testowe po każdym uruchomieniu. Dzielę dane na zbiór treningowy (70%) i testowy (30%):

In [217…
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=0)
```

Przeprowadzam regresję liniową na zbiorze treningowym:

In [218…
```
model = LinearRegression()
model.fit(X_train, y_train)
```

Out[218…
```
LinearRegression
LinearRegression()
```

Liczę MSE i R^2 dla zbioru treningowego i testowego:

In [219…
```
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

print("Dla zbioru treningowego:")
print(f"MSE: {mean_squared_error(y_train, y_train_pred):.4f}")
print(f"R²: {r2_score(y_train, y_train_pred):.4f}\n")
```

```
print("Dla zbioru testowego:")
print(f"MSE: {mean_squared_error(y_test, y_test_pred):.4f}")
print(f"R²: {r2_score(y_test, y_test_pred):.4f}")
```

```
Dla zbioru treningowego:
MSE: 0.0321
R²: 0.4616

Dla zbioru testowego:
MSE: 0.0223
R²: 0.5900
```

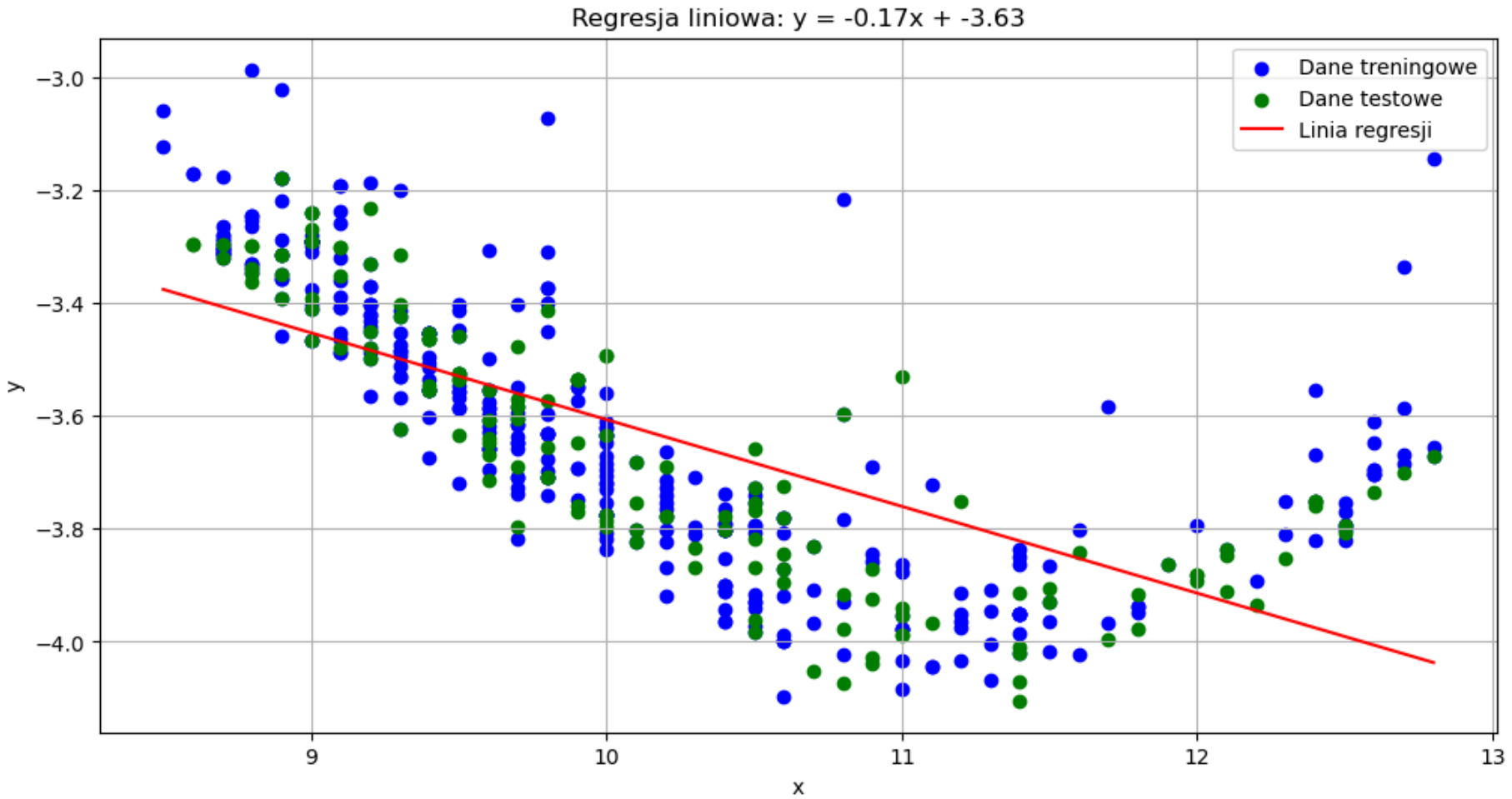
Cofam skalowanie linii regresji przed wizualizacją ponieważ chcę rysować na oryginalnej skali danych, używam do tego "scaler.inverse_transform", aby przekształcić dane regresji z powrotem na oryginalną skalę. Wizualizuję regresje i dane wejściowe na wykresie w tej samej przestrzeni:

In [220…

```
x_line_scaled = np.linspace(X_train.min(), X_train.max(), 100).reshape(-1, 1)
y_line = model.predict(x_line_scaled)
x_line_original = scaler.inverse_transform(x_line_scaled)

plt.figure(figsize=(12, 6))
plt.scatter(scaler.inverse_transform(X_train), y_train, color='blue', label='Dane treningowe')
plt.scatter(scaler.inverse_transform(X_test), y_test, color='green', label='Dane testowe')
plt.plot(x_line_original, y_line, color='red', label='Linia regresji')

coef = model.coef_[0]
intercept = model.intercept_
plt.title(f"Regresja liniowa: y = {coef:.2f}x + {intercept:.2f}")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.show()
```



Zadanie 3

Definiuję regresję wielomianową stopnia drugiego:

In [221…

```
poly2 = PolynomialFeatures(degree=2)
X_train_poly2 = poly2.fit_transform(X_train)
X_test_poly2 = poly2.transform(X_test)
poly2_model = LinearRegression()
poly2_model.fit(X_train_poly2, y_train)
```

```
y_train_pred_poly2 = poly2_model.predict(X_train_poly2)
y_test_pred_poly2 = poly2_model.predict(X_test_poly2)
```

Definiuję regresję wielomianową stopnia trzeciego:

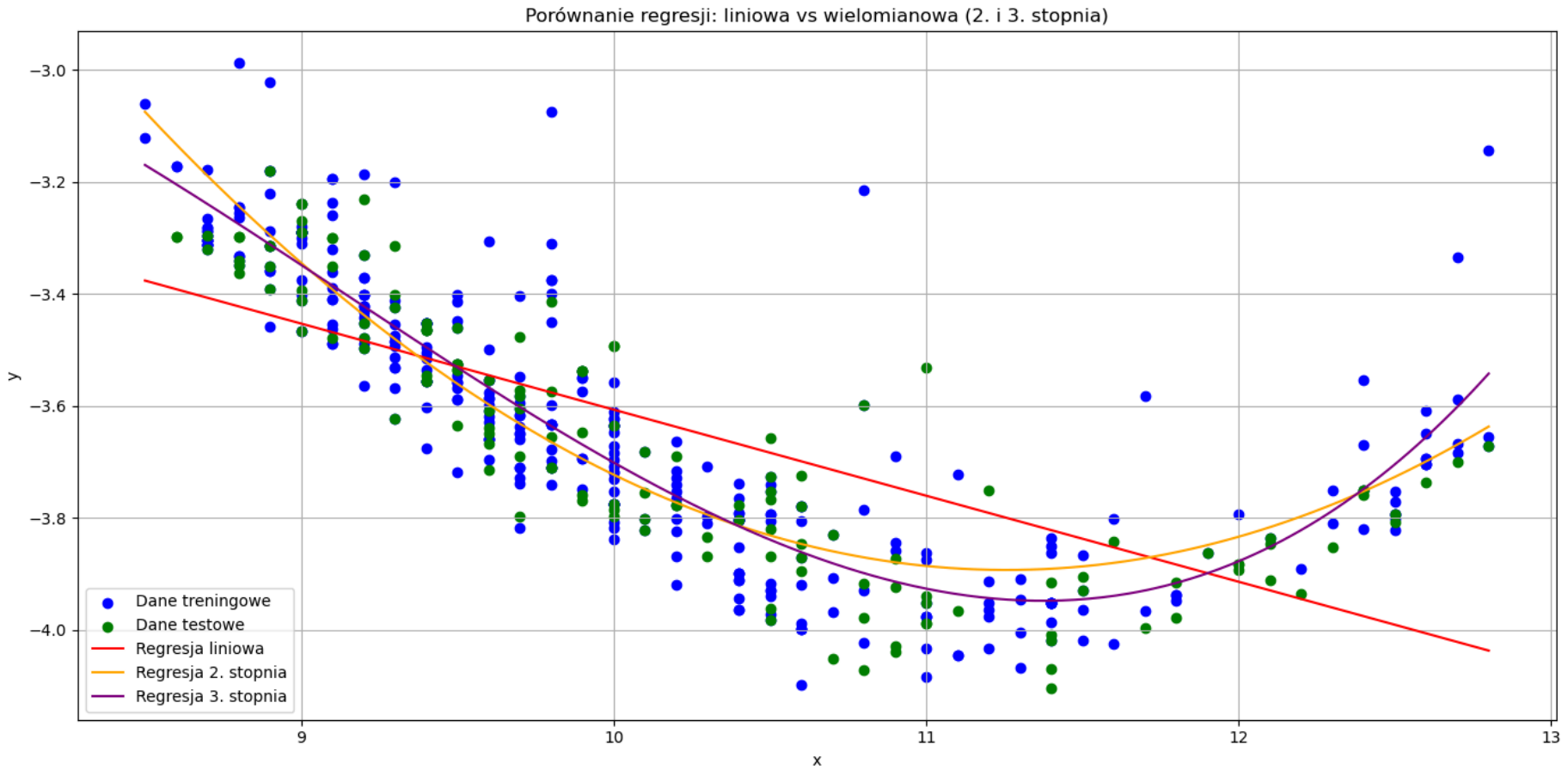
In [222…
```
poly3 = PolynomialFeatures(degree=3)
X_train_poly3 = poly3.fit_transform(X_train)
X_test_poly3 = poly3.transform(X_test)
poly3_model = LinearRegression()
poly3_model.fit(X_train_poly3, y_train)
y_train_pred_poly3 = poly3_model.predict(X_train_poly3)
y_test_pred_poly3 = poly3_model.predict(X_test_poly3)
```

Generuję punkty do wykresu:

In [223…
```
x_line_scaled = np.linspace(X_train.min(), X_train.max(), 300).reshape(-1, 1)
x_line_original = scaler.inverse_transform(x_line_scaled)
y_line_linear = model.predict(x_line_scaled)
y_line_poly2 = poly2_model.predict(poly2.transform(x_line_scaled))
y_line_poly3 = poly3_model.predict(poly3.transform(x_line_scaled))
```

Tworzę wykres porównawczy zawierający dane treningowe, testowe, regresję liniową oraz regresję wielomianową drugiego i trzeciego stopnia:

In [224…
```
plt.figure(figsize=(14, 7))
plt.scatter(scaler.inverse_transform(X_train), y_train, color='blue', label='Dane treningowe')
plt.scatter(scaler.inverse_transform(X_test), y_test, color='green', label='Dane testowe')
plt.plot(x_line_original, y_line_linear, color='red', label='Regresja liniowa')
plt.plot(x_line_original, y_line_poly2, color='orange', label='Regresja 2. stopnia')
plt.plot(x_line_original, y_line_poly3, color='purple', label='Regresja 3. stopnia')
plt.title("Porównanie regresji: liniowa vs wielomianowa (2. i 3. stopnia)")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Tworzę tabele wyników dzięki której można porównać MSE i R^2 dla trzech rodzajów regresji:

In [225…

```
results = {
    "Model": ["Liniowy", "Wielomian 2 st.", "Wielomian 3 st."],
    "Train MSE": [
        mean_squared_error(y_train, y_train_pred),
        mean_squared_error(y_train, y_train_pred_poly2),
        mean_squared_error(y_train, y_train_pred_poly3)
    ],
    "Test MSE": [
        mean_squared_error(y_test, y_test_pred),
        mean_squared_error(y_test, y_test_pred_poly2),
```

```
        mean_squared_error(y_test, y_test_pred_poly3)
    ],
    "Train R²": [
        r2_score(y_train, y_train_pred),
        r2_score(y_train, y_train_pred_poly2),
        r2_score(y_train, y_train_pred_poly3)
    ],
    "Test R²": [
        r2_score(y_test, y_test_pred),
        r2_score(y_test, y_test_pred_poly2),
        r2_score(y_test, y_test_pred_poly3)
    ]
}

results_df = pd.DataFrame(results)
results_df
```

Out[225...

	Model	Train MSE	Test MSE	Train R^2	Test R^2
0	Liniowy	0.032150	0.022344	0.461624	0.590024
1	Wielomian 2 st.	0.012652	0.010235	0.788129	0.812209
2	Wielomian 3 st.	0.011578	0.009091	0.806117	0.833184

WNIOSKI:

Regresja liniowa ma wyraźnie gorsze dopasowanie (niższe R^2, wyższe MSE), co widać też na wykresie – linia nie oddaje nieliniowej zależności danych.

Regresja wielomianowa znacząco poprawia dopasowanie zarówno na danych treningowych, jak i testowych.

Regresja wielomianowa stopnia trzeciego jest minimalnie lepszy niż stopnia drugiego, ale różnice są niewielkie.

Zadanie 4

Pytanie:

Jaka jest zasadnicza różnica pomiędzy regresją liniową / wielomianową, a logistyczną?

Odpowiedź:

Zasadnicza różnica między regresją liniową / wielomianową służy do regresji, czyli przewidywania wartości liczbowych (np. ceny, temperatury). Wyjście modelu jest liczbą rzeczywistą. Model próbuje dopasować linię do punktów pochodzących z wejściowych danych.

Regresja logistyczna służy do klasyfikacji binarnej. Wyjście modelu to prawdopodobieństwo (między 0 a 1), które jest przekształcane do klasy (np. 0 lub 1) za pomocą progu (np. 0.5). Model uczy się funkcji logistycznej (sigmoid), która „zgina" przestrzeń liniową w zakres pomiędzy 0 a 1.

Wczytanie danych:

In [226…
```
df = pd.read_csv("C:/Users/kamil/Downloads/product2.csv")
```

Model z X i Y:

In [227…
```
X = df[["x", "y"]]
y = df["good"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

print("== Wyniki z użyciem x i y ==")
print("Dokładność:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
== Wyniki z użyciem x i y ==
Dokładność: 0.64
              precision    recall  f1-score   support

           0       0.50      0.37      0.43        54
           1       0.69      0.79      0.74        96

    accuracy                           0.64       150
   macro avg       0.60      0.58      0.58       150
weighted avg       0.62      0.64      0.63       150
```

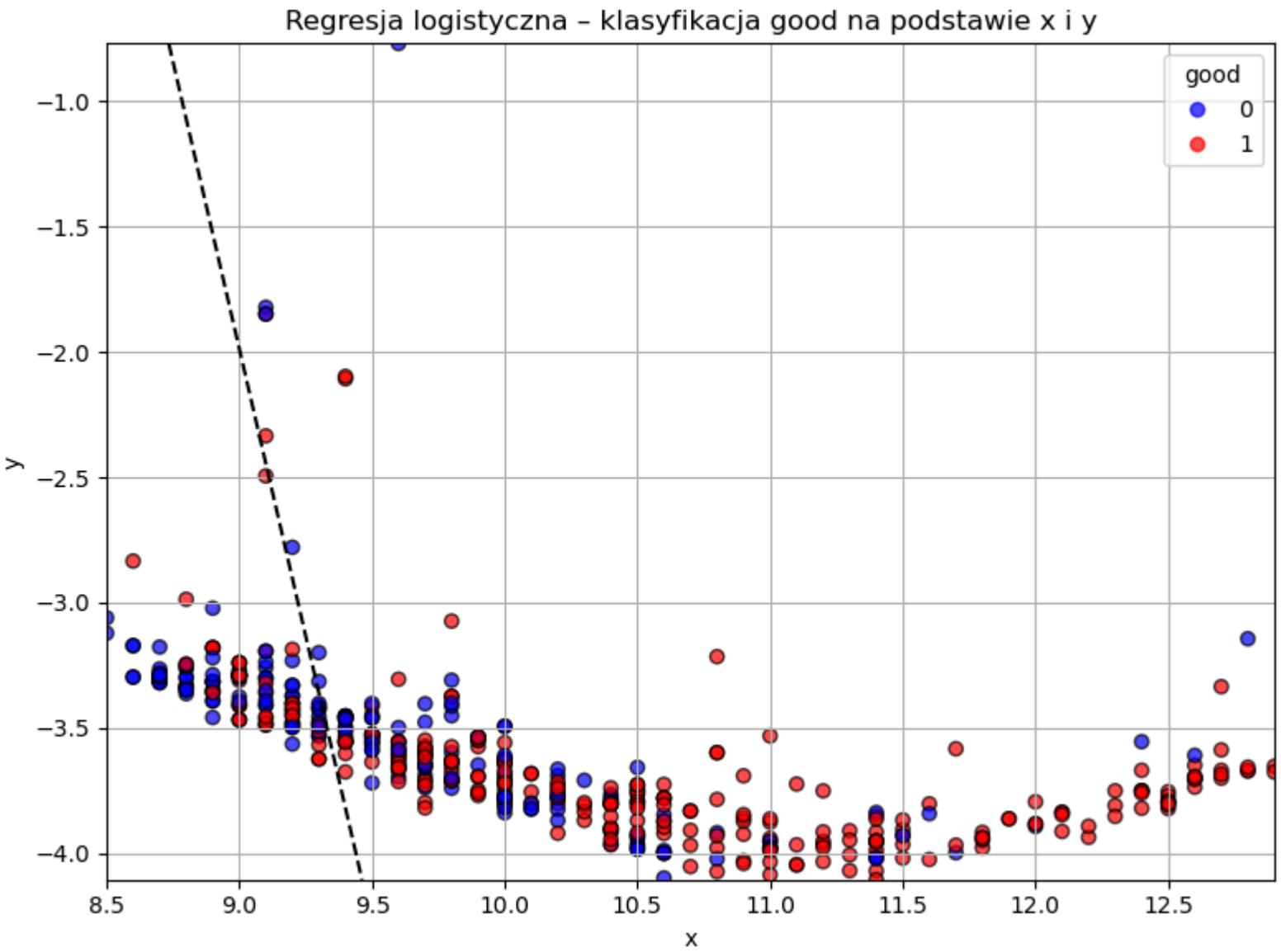
Wizualizacja X, Y, GOOD:

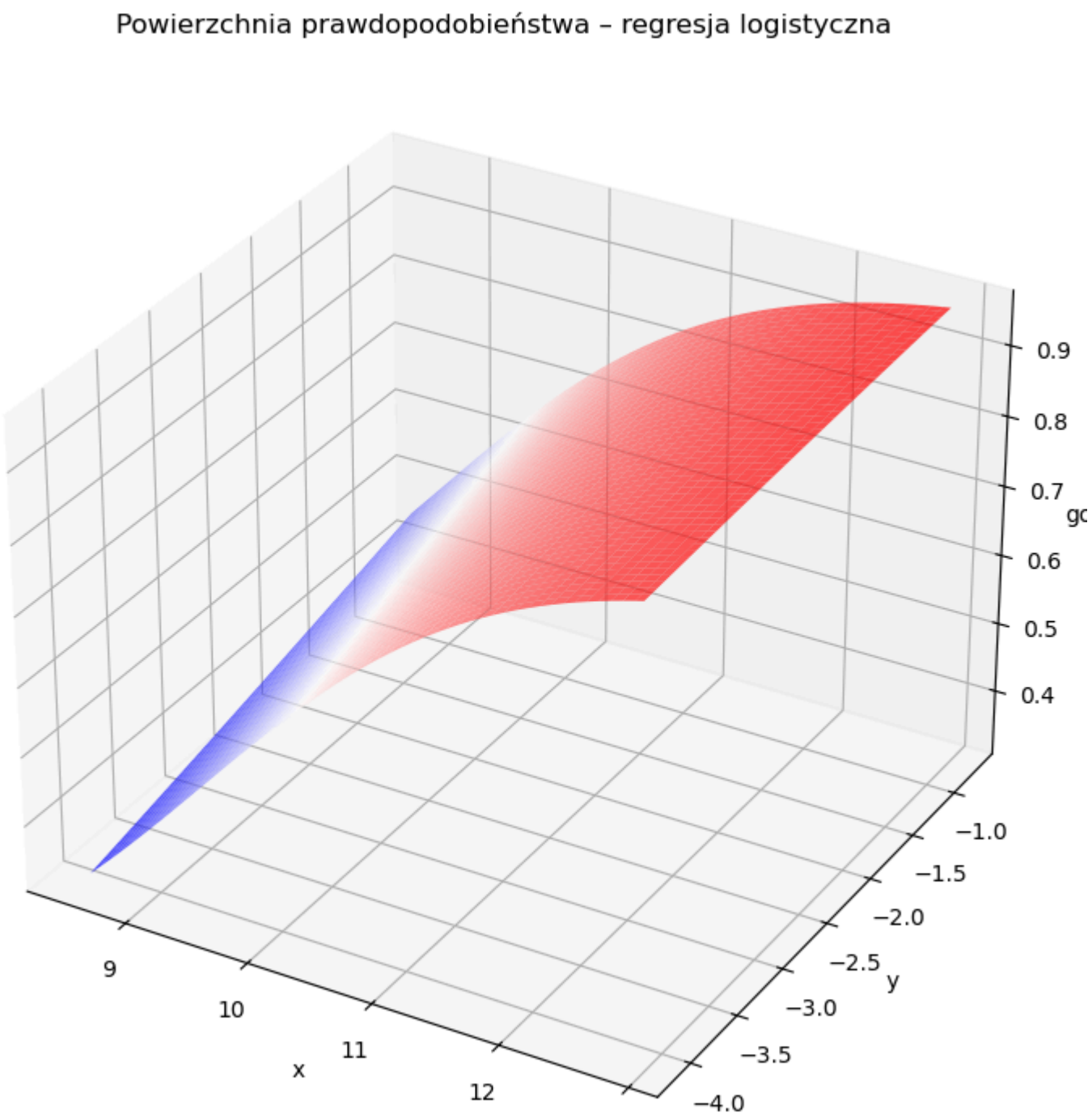
In [228…

```
xx, yy = np.meshgrid(np.linspace(df["x"].min(), df["x"].max(), 200),
                     np.linspace(df["y"].min(), df["y"].max(), 200))
grid = pd.DataFrame(np.c_[xx.ravel(), yy.ravel()], columns=["x", "y"])
probs = model.predict_proba(grid)[:, 1].reshape(xx.shape)

plt.figure(figsize=(8, 6))
scatter = plt.scatter(X["x"], X["y"], c=y, cmap="bwr", edgecolor="k", alpha=0.7)
plt.contour(xx, yy, probs, levels=[0.5], colors="black", linestyles="--")
plt.title("Regresja logistyczna – klasyfikacja good na podstawie x i y")
plt.xlabel("x")
plt.ylabel("y")
plt.legend(*scatter.legend_elements(), title="good")
plt.grid(True)
plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(11, 9))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(xx, yy, probs, cmap='bwr', alpha=0.7)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('good')
plt.title("Powierzchnia prawdopodobieństwa – regresja logistyczna")
plt.show()
```


Regresja logistyczna – klasyfikacja good na podstawie x i y
good
0
1
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
y
8.5
9.0
9.5
10.0
10.5
11.0
11.5
12.0
12.5
x


Powierzchnia prawdopodobieństwa – regresja logistyczna
0.9
0.8
0.7
0.6
0.5
0.4
9
10
11
12
x
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
y

13

Model tylko z X:

In [229…
```
X_single = df[["x"]]
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_single, y, test_size=0.3, random_state=0)

model_s = LogisticRegression()
model_s.fit(X_train_s, y_train_s)
y_pred_s = model_s.predict(X_test_s)

print("== Wyniki z użyciem tylko x ==")
print("Dokładność:", accuracy_score(y_test_s, y_pred_s))
print(classification_report(y_test_s, y_pred_s))
```

```
== Wyniki z użyciem tylko x ==
Dokładność: 0.6533333333333333
              precision    recall  f1-score   support

           0       0.52      0.43      0.47        54
           1       0.71      0.78      0.74        96

    accuracy                           0.65       150
   macro avg       0.62      0.60      0.61       150
weighted avg       0.64      0.65      0.64       150
```

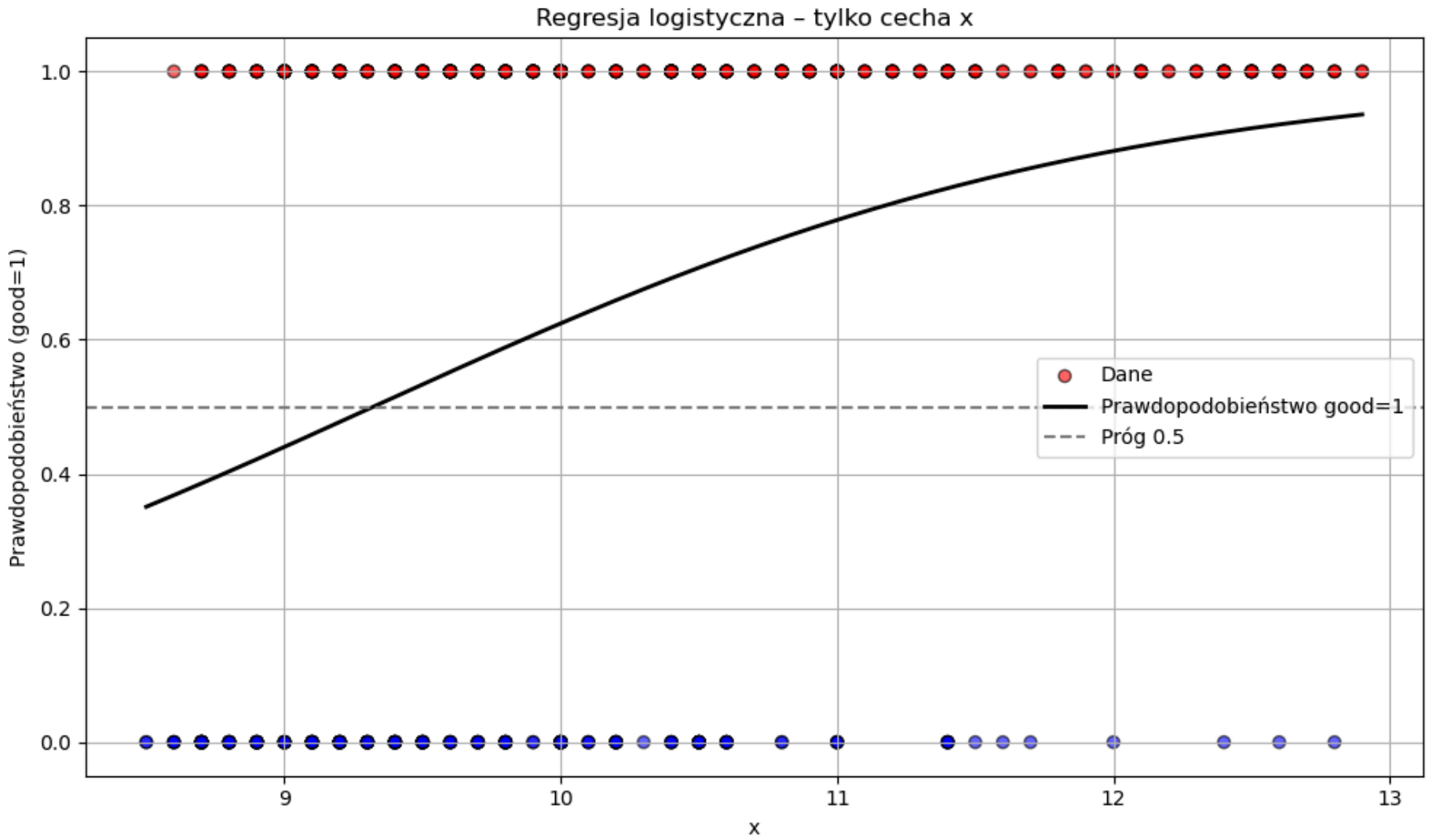
Wizualizacja na wykresie:

In [230…
```
x_range = np.linspace(X_single["x"].min(), X_single["x"].max(), 300).reshape(-1, 1)
x_range_df = pd.DataFrame(x_range, columns=["x"])

probs = model_s.predict_proba(x_range_df)[:, 1]

plt.figure(figsize=(10, 6))
plt.scatter(X_single, y, c=y, cmap='bwr', edgecolor='k', alpha=0.6, label="Dane")
plt.plot(x_range, probs, color='black', linewidth=2, label="Prawdopodobieństwo good=1")
plt.axhline(0.5, color='gray', linestyle='--', label='Próg 0.5')
plt.xlabel("x")
```

```
plt.ylabel("Prawdopodobieństwo (good=1)")
plt.title("Regresja logistyczna – tylko cecha x")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```



Pytanie:

Czy model zadziała gorzej jeśli pod uwagę weźmiemy tylko kolumnę x i klasę good? Być może kolumna y jest zbędna?

Porównanie dokładności (accuracy):

x + y: 0.64

x tylko: 0.6533

Porównanie F1-score dla klasy 1 (czyli good = 1):

x + y: 0.74

x tylko: 0.74

Makrośrednia F1-score (macro avg) – ogólna jakość klasyfikacji:

x + y: 0.58

x tylko: 0.61

Odpowiedź:

Kolumna y nie wnosi wartościowej informacji do klasyfikacji wręcz może nawet wprowadzać szum lub korelację, która pogarsza działanie modelu.

Model zadziałał lepiej, gdy użyłem tylko kolumny x. Oznacza to, że kolumna y może być zbędna, przynajmniej w kontekście regresji logistycznej w tym przypadku.