

## SPRAWOZDANIE

Zajęcia: Nauka o danych I

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 6

Data 01.03.2025

Temat: Analiza danych z  
wykorzystaniem narzędzi do  
modelowania regresji

Krzysztof Świerczek

Informatyka

II stopień, stacjonarne,  
1semestr, gr. A

1. Polecenie: Analiza danych z wykorzystaniem narzędzi do modelowania regresji

2. Github:

<https://github.com/Krzycho165/STUDIA>

In [1]: *# 0. Krzysztof Świerczek Przygotowanie danych*

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import probplot
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error

df = pd.read_csv('final_solution.csv')
numeryczne_kolumny = ['age', 'income', 'outcome']
numeryczny_df = df[numeryczne_kolumny]

if numeryczny_df.isnull().sum().any():
    numeryczny_df.fillna(numeryczny_df.mean(), inplace=True)
```

In [6]: *# 1. Krzysztof Świerczek W Pythonie, R oraz KNIME porównaj wyniki regresji Lin*

```
# Features and target variable
X = df[['age', 'income', 'savings', 'children', 'credit_score', 'spending_score']]
y = df['outcome']

# Split data into training and testing sets
X_treningowe, X_testowe, y_treningowe, y_testowe = train_test_split(X, y, test_s

# Liniowa regresja
lr = LinearRegression()
lr.fit(X_treningowe, y_treningowe)
przewidywany_y_lr = lr.predict(X_testowe)
mse_lr = mean_squared_error(y_testowe, przewidywany_y_lr)

# Regresja Ridge
ridge = Ridge(alpha=1.0)
ridge.fit(X_treningowe, y_treningowe)
y_przewidywane_ridge = ridge.predict(X_testowe)
mse_ridge = mean_squared_error(y_testowe, y_przewidywane_ridge)

# Sieć neuronowa
siec_neuronowa = MLPRegressor(hidden_layer_sizes=(10,), max_iter=1500, random_st
siec_neuronowa.fit(X_treningowe, y_treningowe)
y_przewidywane_siec_neuronowa = siec_neuronowa.predict(X_testowe)
mse_siec_neuronowa = mean_squared_error(y_testowe, y_przewidywane_siec_neuronowa)

print(f"Regresja liniowa: {mse_lr}")
print(f"Regresja Ridge: {mse_ridge}")
print(f"Sieć Neuronowa: {mse_siec_neuronowa}")
```

Regresja liniowa: 770579.8520962859

Regresja Ridge: 770588.3407806738

Sieć Neuronowa: 993068.6666852413

```
C:\Users\krzys\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (1500) reached and the optimization hasn't converged yet.
  warnings.warn(
```

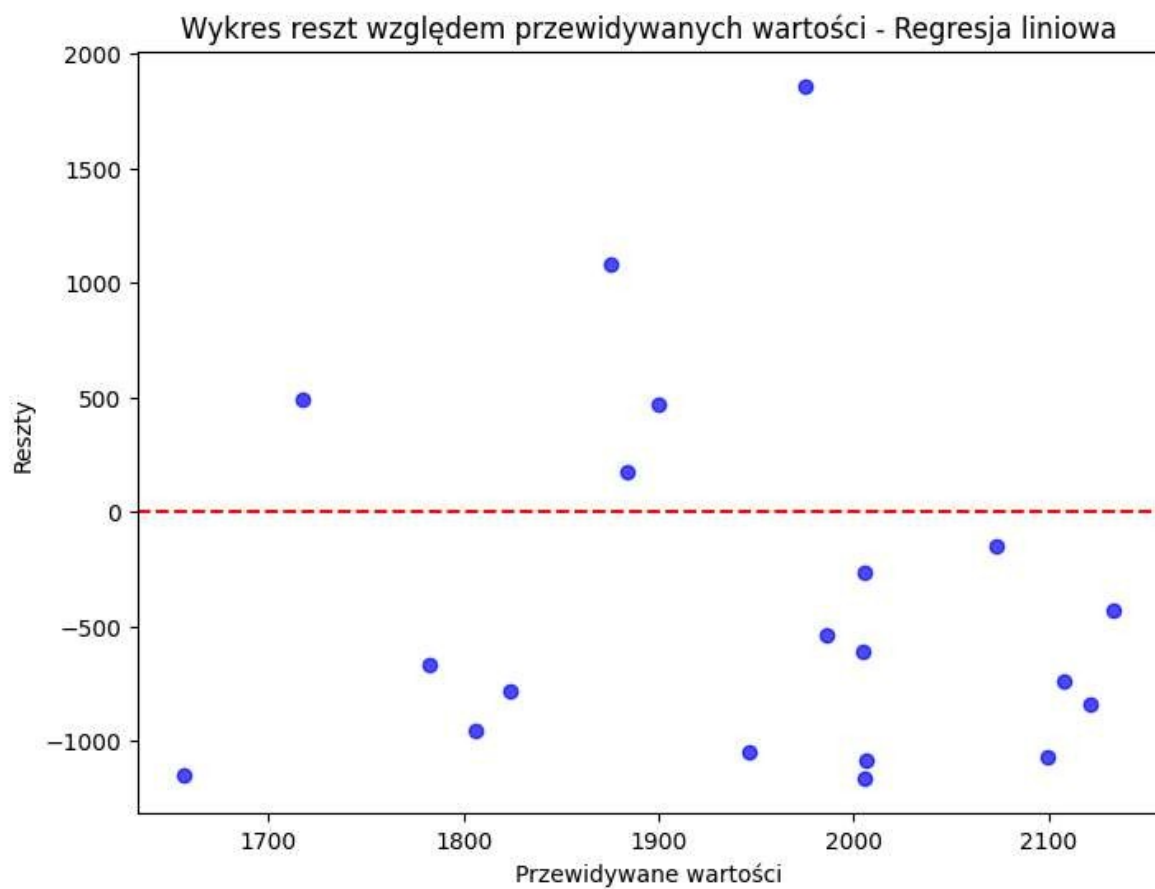
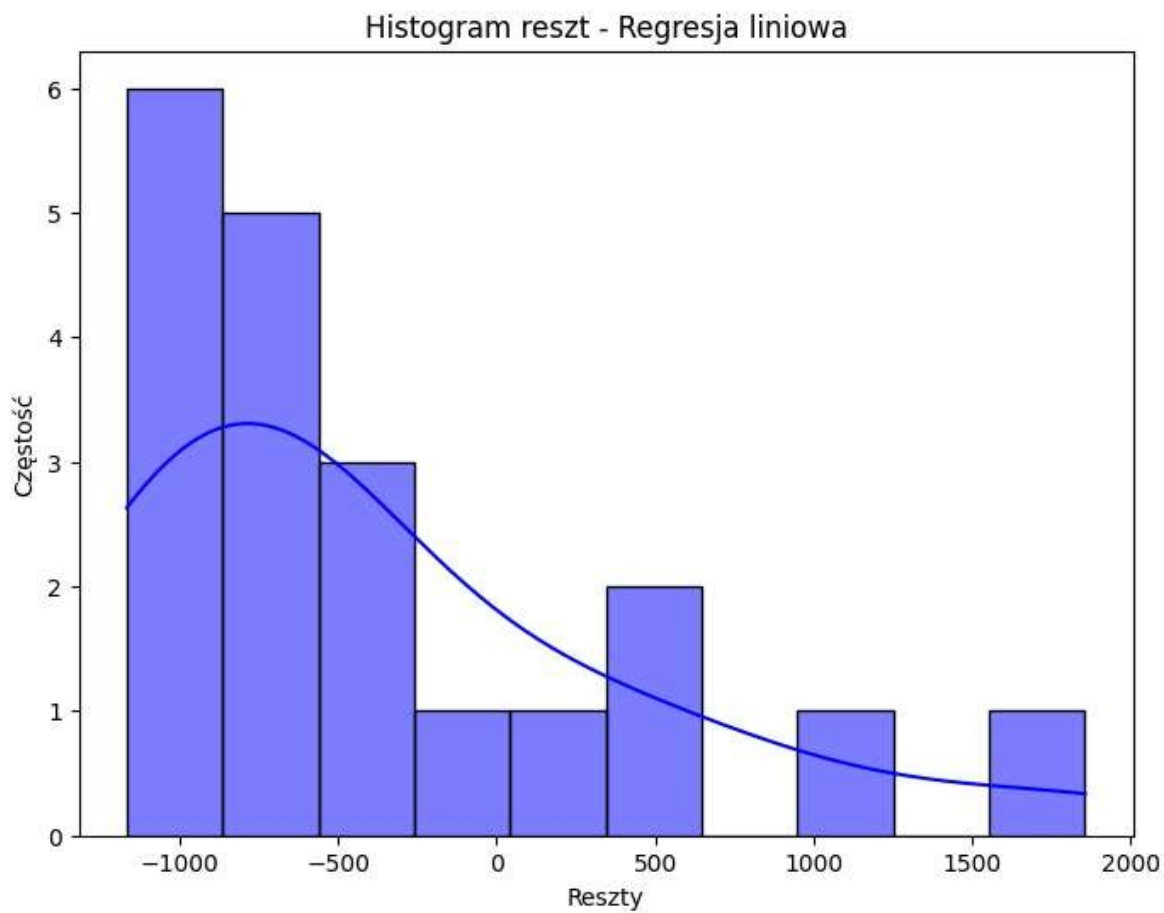
```
In [8]: reszty = y_testowe - przewidywany_y_lr # Obliczenie reszt

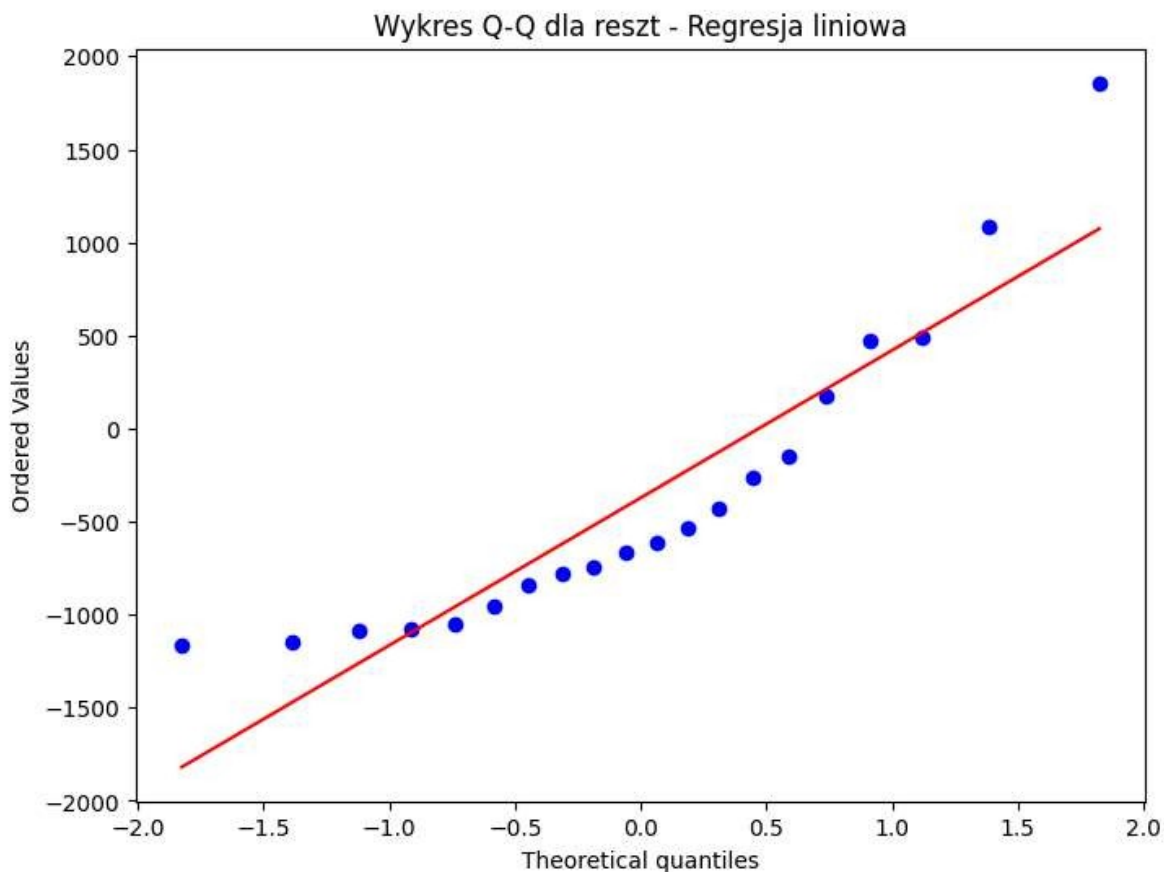
# Histogram reszt
plt.figure(figsize=(8, 6))
sns.histplot(reszty, kde=True, bins=10, color='blue')
plt.title('Histogram reszt - Regresja liniowa')
plt.xlabel('Reszty')
plt.ylabel('Częstość')
plt.show()

# Wykres reszt względem przewidywanych wartości
plt.figure(figsize=(8, 6))
plt.scatter(przewidywany_y_lr, reszty, alpha=0.7, color='blue')
plt.axhline(y=0, color='red', linestyle='--')
plt.title('Wykres reszt względem przewidywanych wartości - Regresja liniowa')
plt.xlabel('Przewidywane wartości')
plt.ylabel('Reszty')
plt.show()

# Normalność reszt - wykres Q-Q
plt.figure(figsize=(8, 6))
probplot(reszty, dist="norm", plot=plt)
plt.title('Wykres Q-Q dla reszt - Regresja liniowa')
plt.show()

# Średnia kwadratowa błędu (MSE) dla modelu regresji liniowej
print(f"Mean Squared Error (MSE) dla regresji liniowej: {mse_lr:.2f}")
```





Mean Squared Error (MSE) dla regresji liniowej: 770579.85

```
In [9]: from statsmodels.stats.stattools import durbin_watson
from scipy.stats import shapiro

# 3.1. Sprawdzenie normalności reszt (Shapiro-Wilk)
shapiro_test_stat, shapiro_p_value = shapiro(reszty)
print("Test Shapiro-Wilka dla normalności reszt:")
print(f"Statystyka testowa: {shapiro_test_stat:.4f}, p-wartość: {shapiro_p_value:.4f}")

if shapiro_p_value > 0.05:
    print("Brak podstaw do odrzucenia hipotezy zerowej: reszty są normalnie rozłożone.")
else:
    print("Odrzucenie hipotezy zerowej: reszty nie są normalnie rozłożone.")

# 3.2. Test autokorelacji reszt (Durbin-Watson)
durbin_watson_stat = durbin_watson(reszty)
print("\nTest Durbin-Watson:")
print(f"Statystyka Durbin-Watson: {durbin_watson_stat:.4f}")

# Interpretacja wyników testu Durbin-Watson
if durbin_watson_stat < 1.5:
    print("Wskazanie na autokorelację dodatnią reszt.")
elif durbin_watson_stat > 2.5:
    print("Wskazanie na autokorelację ujemną reszt.")
else:
    print("Brak istotnej autokorelacji reszt.")
```

Test Shapiro-Wilka dla normalności reszt:  
Statystyka testowa: 0.8518, p-wartość: 5.7237e-03  
Odrzucenie hipotezy zerowej: reszty nie są normalnie rozłożone.

Test Durbin-Watson:  
Statystyka Durbin-Watson: 1.3496  
Wskazanie na autokorelację dodatnią reszt.

```
In [10]: from sklearn.preprocessing import StandardScaler
# 4. Porównaj jakość modeli przy użyciu danych o różnych skalach (np. znormalizowane)

# Przygotowanie danych
X = df[['age', 'income', 'savings', 'children', 'credit_score', 'spending_score']]
y = df['outcome']

# Podział na zbiór treningowy i testowy
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# 1. Oryginalne dane
lr_orig = LinearRegression()
lr_orig.fit(X_train, y_train)
y_pred_lr_orig = lr_orig.predict(X_test)
mse_lr_orig = mean_squared_error(y_test, y_pred_lr_orig)

# 2. Znormalizowane dane
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Regresja Liniowa (na znormalizowanych danych)
lr_scaled = LinearRegression()
lr_scaled.fit(X_train_scaled, y_train)
y_pred_lr_scaled = lr_scaled.predict(X_test_scaled)
mse_lr_scaled = mean_squared_error(y_test, y_pred_lr_scaled)

# Regresja Ridge (na obu zbiorach)
ridge_orig = Ridge(alpha=1.0)
ridge_orig.fit(X_train, y_train)
y_pred_ridge_orig = ridge_orig.predict(X_test)
mse_ridge_orig = mean_squared_error(y_test, y_pred_ridge_orig)

ridge_scaled = Ridge(alpha=1.0)
ridge_scaled.fit(X_train_scaled, y_train)
y_pred_ridge_scaled = ridge_scaled.predict(X_test_scaled)
mse_ridge_scaled = mean_squared_error(y_test, y_pred_ridge_scaled)

# Sieć neuronowa (na obu zbiorach)
nn_orig = MLPRegressor(hidden_layer_sizes=(10,), max_iter=500, random_state=42)
nn_orig.fit(X_train, y_train)
y_pred_nn_orig = nn_orig.predict(X_test)
mse_nn_orig = mean_squared_error(y_test, y_pred_nn_orig)

nn_scaled = MLPRegressor(hidden_layer_sizes=(10,), max_iter=500, random_state=42)
nn_scaled.fit(X_train_scaled, y_train)
y_pred_nn_scaled = nn_scaled.predict(X_test_scaled)
mse_nn_scaled = mean_squared_error(y_test, y_pred_nn_scaled)

# Porównanie wyników
print("Regresja Liniowa:")
print(f"- Oryginalne dane: MSE = {mse_lr_orig:.2f}")
```

```
print(f"- Znormalizowane dane: MSE = {mse_lr_scaled:.2f}")

print("\nRegresja Ridge:")
print(f"- Oryginalne dane: MSE = {mse_ridge_orig:.2f}")
print(f"- Znormalizowane dane: MSE = {mse_ridge_scaled:.2f}")

print("\nSieć Neuronowa:")
print(f"- Oryginalne dane: MSE = {mse_nn_orig:.2f}")
print(f"- Znormalizowane dane: MSE = {mse_nn_scaled:.2f}")
```

## Regresja Liniowa:

- Oryginalne dane: MSE = 770579.85
- Znormalizowane dane: MSE = 770579.85

### Regressja Ridge:

- Oryginalne dane: MSE = 770588.34
- Znormalizowane dane: MSE = 770417.22

## Sieć Neuronowa:

- Oryginalne dane: MSE = 1928394.26
- Znormalizowane dane: MSE = 3079022.97

```
C:\Users\krzys\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\nural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged yet.
```

```
warnings.warn(
```

```
C:\Users\krzys\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\nural_network\_multilayer_perceptron.py:691: ConvergenceWarning: Stochastic Optimizer: Maximum iterations (500) reached and the optimization hasn't converged yet.
```

```
warnings.warn(
```

In [ ]:

```
# Tworzymy podział danych na zbiór treningowy i testowy (80% - 20%)
trainIndex <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[trainIndex,] # Treningowe cechy
X_test <- X[-trainIndex,] # Testowe cechy
y_train <- y[trainIndex] # Treningowe wyniki
y_test <- y[-trainIndex] # Testowe wyniki

# Model regresji liniowej
lm_model <- lm(outcome ~ age + income + savings + children + credit_score + spending_score, data = df)
# Przewidywanie wyników na zbiorze testowym
y_pred_lm <- predict(lm_model, X_test)
# Obliczenie MSE dla regresji liniowej
mse_lm <- mean((y_test - y_pred_lm)^2)

# Model Ridge Regression
ridge_model <- cv.glmnet(as.matrix(X_train), y_train, alpha = 0)
# Przewidywanie wyników na zbiorze testowym przy użyciu najlepszego lambda
y_pred_ridge <- predict(ridge_model, as.matrix(X_test), s = "lambda.min")
# Obliczenie MSE dla regresji Ridge
mse_ridge <- mean((y_test - y_pred_ridge)^2)

# Model sieci neuronowej
nn_model <- neuralnet(outcome ~ age + income + savings + children + credit_score + spending_score, data = df,
hidden = 10, linear.output = TRUE)
# Przewidywanie wyników na zbiorze testowym przy użyciu sieci neuronowej
```

## Documentation

[caret](#): Classification and Regression Training

[caret](#): Classification and Regression Training

[glmnet](#): Lasso and Elastic-Net Regularized Generalized Linear Models

[neuralnet](#): Training of Neural Networks

Run (Ctrl-Enter)

Any scripts or data that you put into this service are public.

```
Loading required package: lattice
Loading required package: ggplot2
Loading required package: Matrix
Loaded glmnet 4.0-2
Warning message:
Option grouped=FALSE enforced in cv.glmnet, since < 3 observations per fold
[1] "Regresja Liniowa: 691179.2465348984"
[1] "Regresja Ridge: 691210.1828397188"
[1] "Sieć 304/207 neuronowa: 1928776.2844036984"
```



Podczas realizacji ćwiczenia wykorzystano wiele zaawansowanych bibliotek Pythona, co pozwoliło na zdobycie praktycznych umiejętności w zakresie przetwarzania danych, budowy modeli regresyjnych i ich ewaluacji.

### **1. Obsługa pandas i numpy – operacje na danych**

Na początku ćwiczenia wczytano dane z pliku CSV przy użyciu `pandas`, co umożliwiło poznanie metod manipulacji danymi, takich jak wybór kolumn numerycznych i uzupełnianie brakujących wartości (`fillna()`).

Dodatkowo zastosowanie `numpy` pozwoliło na lepsze zrozumienie operacji na tablicach numerycznych.

### **2. Wizualizacja danych – matplotlib, seaborn i scipy.stats**

Do analizy danych wykorzystano różne techniki wizualizacji:

`matplotlib` – do tworzenia podstawowych wykresów,

`seaborn` – do estetycznych wizualizacji zależności między zmiennymi,

`scipy.stats.probplot()` – do analizy normalności rozkładu danych, co jest istotnym krokiem w analizie regresji.

Dzięki temu ćwiczenie pomogło zrozumieć, jak odpowiednio interpretować dane przed ich dalszym przetwarzaniem.

### **3. Podział danych na zbiory treningowe i testowe – sklearn.model\_selection**

W kodzie zastosowano metodę `train_test_split()` z `sklearn.model_selection`, co pozwoliło na naukę poprawnego podziału danych na zbiór uczący i testowy. To kluczowy krok w budowie modeli predykcyjnych, zapobiegający nadmiernemu dopasowaniu modelu do danych.

### **4. Budowa modeli regresyjnych – sklearn.linear\_model oraz sklearn.neural\_network**

W ćwiczeniu zastosowano kilka metod regresyjnych:

`LinearRegression()` – klasyczną regresję liniową,

`ridge()` – regresję grzbietową, która pozwala na regularyzację modelu,

`LPR regressor()` – sieć neuronową do rozwiązywania problemów regresyjnych.

Dzięki temu możliwe było porównanie różnych metod przewidywania wartości docelowej oraz zrozumienie zalet i wad każdego podejścia.

### **5. Ewaluacja modeli – sklearn.metrics**

W celu oceny skuteczności modeli zastosowano metrykę `mean_squared_error()` (MSE). Dzięki temu można było zobaczyć, jak mierzyć błędy modeli i interpretować wyniki w kontekście dopasowania do rzeczywistych danych. Ćwiczenie pozwoliło na praktyczne zapoznanie się z wieloma popularnymi bibliotekami stosowanymi w analizie danych i uczeniu maszynowym. Nauczyłem się poprawnego przygotowania danych, ich wizualizacji, podziału na zbiory, budowy modeli predykcyjnych oraz ich oceny. Takie umiejętności są kluczowe przy pracy nad rzeczywistymi problemami analizy danych.