

SPRAWOZDANIE

Zajęcia: Nauka o danych I

Prowadzący: prof. dr hab. Vasyl Martsenyuk

Laboratorium Nr 6 Data 07.12.2024 Temat: Analiza danych z wykorzystaniem narzędzi do modelowania regresji	Bartosz Bieniek Informatyka II stopień, stacjonarne, 1 semestr, gr.A
---	---

1. Polecenie

A) W Pythonie, R oraz KNIME porównaj wyniki regresji liniowej, Ridge, sieci neuronowych na tym samym zbiorze danych.

B) Zbadaj wpływ zmiennych objaśniających na predykcję (np. analiza ważności cech w Ridge).

C) Wykonaj analizę reszt dla modelu regresji liniowej:

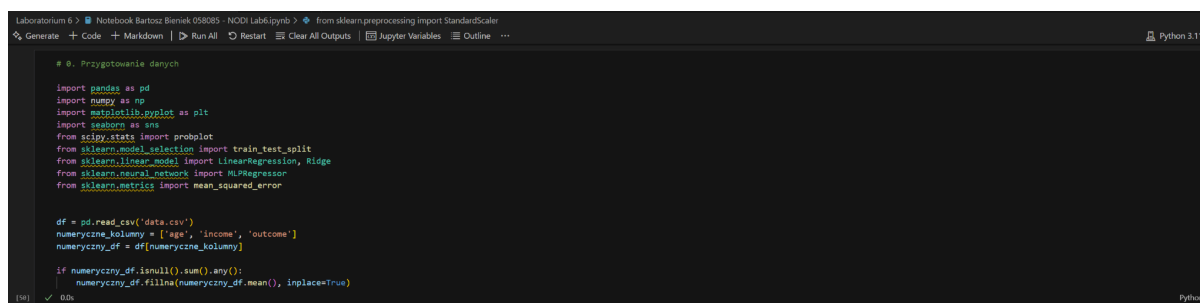
D) Sprawdź założenie normalności błędów,

E) Zbadaj autokorelację reszt (np. test Durbin-Watson w Pythonie lub R).

F) Porównaj jakość modeli przy użyciu danych o różnych skalach (np. znormalizowanych i oryginalnych).

2. Opis programu opracowanego

<https://github.com/mindgoner/Studia/tree/master/Nauka%20o%20Danych/Laboratorium%206>

A screenshot of a Jupyter Notebook interface. The top bar shows 'Laboratorium 6' and 'Notebook Bartosz Bieniek (9808) - NCI3 lab6.ipynb'. Below the bar, there are tabs for 'Generate', 'Code', 'Markdown', 'Run All', 'Restart', 'Clear All Outputs', 'Jupyter Variables', and 'Outline'. The main area contains Python code for data preparation. The code starts with a comment '# 0. Przygotowanie danych' and imports libraries: pandas as pd, numpy as np, matplotlib.pyplot as plt, seaborn as sns, scipy.stats as stats, and sklearn.model_selection.train_test_split, sklearn.linear_model.LinearRegression, sklearn.neural_network.MLPRegressor, and sklearn.metrics.mean_squared_error. It then reads a CSV file 'data.csv' into a DataFrame 'df', selects columns 'age', 'income', and 'outcome', and creates a new DataFrame 'numeryczny_df' with only numerical columns. Finally, it checks for null values and fills them with the mean of each column using 'numeryczny_df.fillna(numeryczny_df.mean(), inplace=True)'. The bottom status bar shows '(10) ✓ 00s' and 'Python 3.11.9'.

Rys 1. Fragment programu

Standardowo, przed przystąpieniem do analizy danych zaimportowano biblioteki oraz przygotowano dane poprzez nadanie numerycznych własności kolumn i wypełnienie pustych danych wartościami średnimi.

```

# 1. W Pythonie, R oraz KNIME porównaj wyniki regresji liniowej, Ridge, sieci neuronowych na tym samym zbiorze danych.

# 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_trainowe, X_testowe, y_trainowe, y_testowe = train_test_split(X, y, test_size=0.2, random_state=42)

# Liniowa regresja
lr = LinearRegression()
lr.fit(X_trainowe, y_trainowe)
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_trainowe, y_trainowe)
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=500, random_state=42)
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}")

```

Python

Regresja liniowa: 691179.2465348984
 Regresja Ridge: 691210.1828397188
 Sieć Neuronowa: 1920776.2844836984
 warnings.warn(

Rys 2. Fragment programu oraz wynik jego wykonania

W kodzie przedstawionym na rysunku pierwszym porównano trzy modele regresji: liniową, Ridge oraz sieci neuronowe, używając tych samych danych. Dane zostały podzielone na zbiory treningowe i testowe, a następnie każdy model został wytrenowany i oceniony za pomocą średniego błędu kwadratowego (MSE). Wyniki pokazują, że regresja liniowa i Ridge osiągają bardzo podobne wartości średniego błędu kwadratowego (MSE), co wskazuje, że Ridge nie wnosi istotnej poprawy w regularizacji dla tego zbioru danych. Sieć neuronowa uzyskała znacznie wyższy MSE, co może sugerować, że model jest mniej dopasowany do danych, prawdopodobnie z powodu niewystarczającej liczby epok, złożoności modelu lub problemu przeuczenia. Dla tego konkretnego przypadku prostsze modele działają lepiej.

https://rdrr.io/snippets/

Find an R package R language docs Run R in your browser

```

# Tworzymy podział danych na zbiór treningowy i testowy (80% - 20%)
scatIndex <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[scatIndex,] # Treningowe cechy
X_test <- X[-scatIndex,] # Testowe cechy
y_train <- y[scatIndex] # Treningowe wyniki
y_test <- y[-scatIndex] # 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: 1920776.2844836984"

```

Rys 3. Fragment programu oraz wynik jego wykonania

Kod w R wykonuje analogiczne kroki, co kod w Pythonie: buduje trzy modele regresji (liniową, Ridge i sieć neuronową), oblicza ich przewidywania na zbiorze testowym i ocenia jakość za pomocą średniego błędu kwadratowego (MSE). Wyniki MSE są niemal identyczne z tymi uzyskanymi w Pythonie, co potwierdza zgodność implementacji w obu językach. Takie zestawienie podkreśla, że regresja liniowa i Ridge działają porównywalnie, podczas gdy sieć neuronowa ma trudności z dopasowaniem do danych.

```
# 2. Zbadaj wpływ zmiennych objaśniających na predykcję (np. analiza ważności cech w Ridge).

model_ride_cv = RidgeCV(alphas=np.logspace(-6, 6, 13), store_cv_values=True) # Model Ridge Regression (z cross-validation)
model_ride_cv.fit(X_treningowe, y_treningowe) # Dopasowanie modelu do danych treningowych
waznoscie = model_ride_cv.coef_ # Współczynniki ważności cech
nazwy_cech = X.columns

for feature, waznosc in zip(nazwy_cech, waznoscie):
    print(f'Cecha: {feature}, ważność: {waznosc}') # Wyniki ważności cech

print(f'Optymalna lambda: {model_ride_cv.alpha_}')

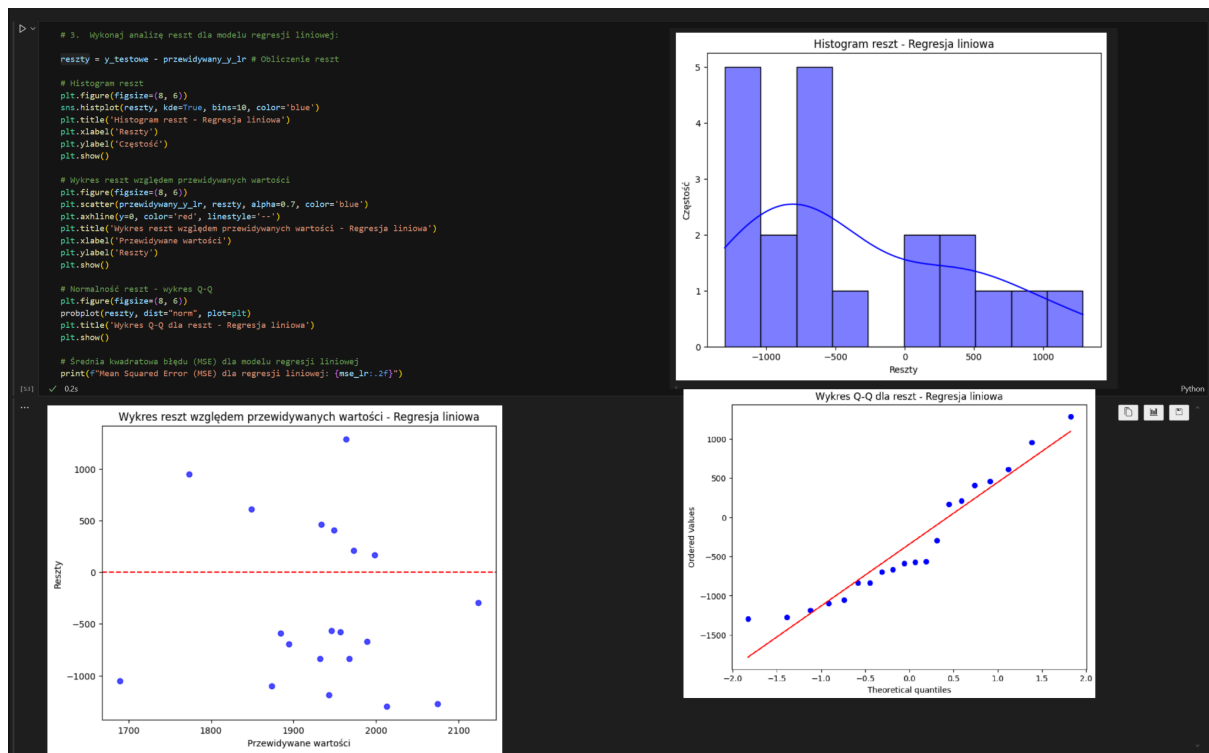
y_przewidywane_ride = model_ride_cv.predict(X_testowe) # Predykcja na zbiorze testowym
mse_ride = np.mean((y_testowe - y_przewidywane_ride)**2) # Obliczenie MSE
print(f'Regresja Ridge MSE: {mse_ride}')

✓ 0.0s Python

Cecha: age, ważność: -0.057758087287262775
Cecha: income, ważność: 0.06234187711480921
Cecha: savings, ważność: 0.00023841181034349776
Cecha: children, ważność: -0.001157720855211704
Cecha: credit_score, ważność: -0.12035665599223672
Cecha: spending_score, ważność: 0.09089194333596211
Optymalna lambda: 1000000.0
Regresja Ridge MSE: 697888.813788701
C:\Programy\Python\3.11\Lib\site-packages\sklearn\linear_model\ridge.py:3375: FutureWarning: 'store_cv_values' is deprecated in version 1.5 and will be removed in 1.7. Use 'store_cv_results' instead.
  warnings.warn(
```

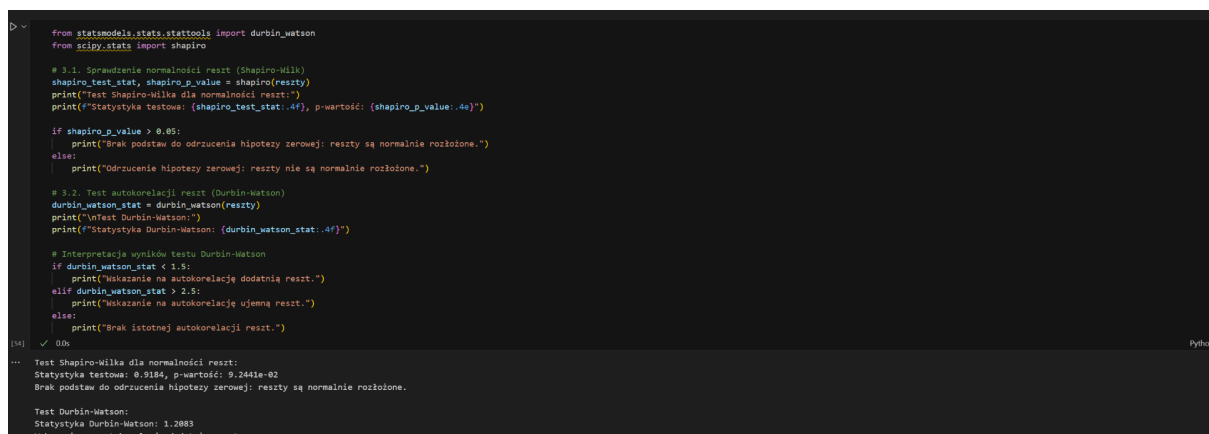
Rys 4. Fragment programu oraz wynik jego wykonania

Kod przedstawiony na rysunku czwartym przeprowadza analizę ważności cech w modelu Ridge Regression, korzystając z ich współczynników wag po wytrenowaniu modelu. Każda cecha otrzymuje wartość, która pokazuje jej wpływ na wynik predykcji. Optymalna wartość regularyzacji $\lambda = 10^6$, co wskazuje na silną regularizację, czyli minimalizuje wpływ mniej istotnych cech. Wynik MSE dla Ridge jest porównywalny z wcześniejszymi wynikami, co potwierdza stabilność modelu, a najbardziej znaczące cechy to `spending_score` i `credit_score`.



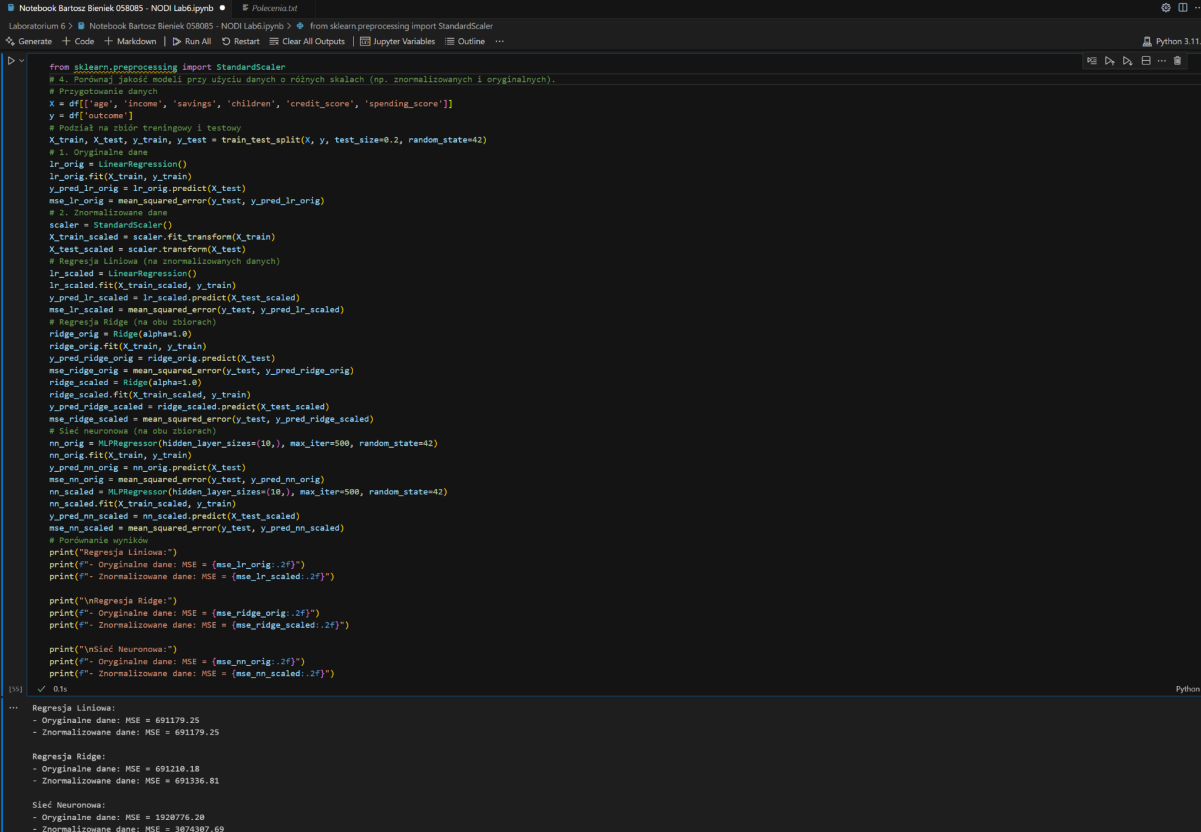
Rys 5. Fragment programu oraz wynik jego wykonania

Kod Źródłowy z powyższego rysunku przeprowadza szczegółową analizę reszt dla modelu regresji liniowej, aby sprawdzić zgodność z założeniami modelu. Histogram reszt pozwala ocenić ich rozkład, wykres reszt względem przewidywań sprawdza losowość rozkładu błędów, a wykres Q-Q weryfikuje normalność reszt. Te wizualizacje oraz podany MSE pomagają w ocenie jakości dopasowania modelu i identyfikacji potencjalnych problemów, takich jak heteroskedastyczność czy nieliniowość. Mean Squared Error (MSE) dla regresji liniowej wynosi: 691179.25, co jest wartością zbliżoną do regresji liniowej z rysunku drugiego.



Rys 6. Fragment programu oraz wynik jego wykonania

W tym kroku weryfikowane są dwa kluczowe założenia regresji liniowej: normalność reszt i brak ich autokorelacji. Test Shapiro-Wilka wskazuje, że reszty mają rozkład zgodny z normalnym (p-wartość > 0.05), co jest zgodne z założeniami modelu. Z kolei statystyka Durbin-Watson wynosząca 1.2083 sugeruje występowanie autokorelacji dodatniej, co może oznaczać, że błędy modelu są skorelowane w czasie lub przestrzeni i wskazuje na potrzebę dalszych analiz lub ulepszenia modelu.



```
from sklearn.preprocessing import StandardScaler
# 4. Porównaj jakość modeli przy użyciu danych o różnych skalach (np. znormalizowanych i oryginalnych).
# 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_state=42)
# 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("\nRegresja 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}")
```

Python

```
Regresja liniowa:
- Oryginalne dane: MSE = 691179.25
- Znormalizowane dane: MSE = 691179.25

Regresja Ridge:
- Oryginalne dane: MSE = 691210.18
- Znormalizowane dane: MSE = 691336.81

Sieć Neuronowa:
- Oryginalne dane: MSE = 1920776.20
- Znormalizowane dane: MSE = 3874307.69
```

Rys 7. Fragment programu oraz wynik jego wykonania

Ten kod porównuje wydajność trzech modeli regresji na danych w oryginalnej skali oraz po ich znormalizowaniu. Regresja liniowa zachowuje tę samą jakość na obu zbiorach (MSE nie zmienia się), co wynika z jej niezmienności na skalowanie danych. W przypadku regresji Ridge i sieci neuronowych znormalizowane dane prowadzą do pogorszenia wyników (większe MSE), co może wskazywać na konieczność dalszego dostosowania hiperparametrów lub problem wrażliwości tych modeli na skalowanie w danym przypadku.

3. Wnioski

Modele regresji liniowej i Ridge wykazały porównywalną jakość predykcji, co można zaobserwować po zbliżonych wartościach MSE. Oznacza to, że regularizacja Ridge nie miała istotnego wpływu na poprawę wyników, co sugeruje, że dane mogą nie zawierać silnie skorelowanych cech lub model regresji liniowej wystarczająco dobrze dopasował się do danych.

Analiza współczynników Ridge pokazała, że takie cechy jak "spending_score" i "income" mają największy wpływ na predykcje modelu, podczas gdy inne, jak "savings" czy "children," mają mniejsze znaczenie. To może pomóc w selekcji cech w przyszłych modelach i zrozumieniu, które zmienne są najbardziej istotne w przewidywaniu zmiennej zależnej.

Reszty modelu regresji liniowej okazały się zgodne z założeniem normalności, co potwierdził test Shapiro-Wilka. Jednak analiza testu Durbin-Watson ujawniła dodatnią autokorelację reszt, co sugeruje, że model mógł nie w pełni uchwycić struktury w danych. Może to wymagać dalszych analiz

Regresja liniowa wykazała odporność na skalowanie danych, podczas gdy Ridge i sieć neuronowa uzyskały różne wyniki w zależności od skali danych. To pokazuje, że wybór i przetwarzanie danych mogą mieć kluczowe znaczenie dla jakości predykcji w bardziej złożonych modelach.